Georgios N. Yannakakis and Julian Togelius

# Artificial Intelligence and Games

January 26, 2018

Springer

# Chapter 2
# AI Methods

This chapter presents a number of basic AI methods that are commonly used in games, and which will be discussed and referred to in the remainder of this book. These are methods that are frequently covered in introductory AI courses—if you have taken such a course, it should have exposed you to at least half of the methods in this chapter. It should also have prepared you for easily understanding the other methods covered in this chapter.

As noted previously, this book assumes that the reader is already familiar with core AI methods at the level of an introductory university course in AI. Therefore, we recommend you to make sure that you are at least cursorily familiar with the methods presented in this chapter before proceeding to read the rest of the book. The algorithm descriptions in this chapter are **high-level descriptions** meant to refresh your memory if you have learned about the particular algorithm at some previous point, or to explain the general idea of the algorithm if you have never seen it before. Each section comes with pointers to the literature, either research papers or other textbooks, where you can find more details about each method.

In this chapter we divide the relevant parts of AI (for the purposes of the book) into six categories: ad-hoc authoring, tree search, evolutionary computation, supervised learning, reinforcement learning and unsupervised learning. In each section we discuss some of the main algorithms in general terms, and give suggestions for further reading. Throughout the chapter we use the game of *Ms Pac-Man* (Namco, 1982) (or Ms Pac-Man for simplicity) as an overarching testbed for all the algorithms we cover. For the sake of consistency, all the methods we cover are employed to **control** Ms Pac-Man's behavior even though they can find a multitude of other uses in this game (e.g., generating content or analyzing player behavior). While a number of other games could have been used as our testbed in this chapter, we picked Ms Pac-Man for its popularity and its game design simplicity as well as for its high complexity when it comes to playing the game. It is important to remember that Ms Pac-Man is a **non-deterministic** variant of its ancestor *Pac-Man* (Namco, 1980) which implies that the movements of ghosts involve a degree of randomness.

In Section 2.1, we go through a quick overview of two key overarching components of all methods in this book: representation and utility. **Behavior authoring**,

covered in Section 2.2, refers to methods employing static ad-hoc representations without any form of search or learning such as finite state machines, behavior trees and utility-based AI. **Tree search**, covered in Section 2.3, refers to methods that search the space of future actions and build trees of possible action sequences, often in an adversarial setting; this includes the Minimax algorithm, and Monte Carlo tree search. Covered in Section 2.4, **evolutionary computation** refers to population-based global stochastic optimization algorithms such as genetic algorithms, or evolution strategies. **Supervised learning** (see Section 2.5) refers to learning a model that maps instances of datasets to target values such as classes; target values are necessary for supervised learning. Common algorithms used here are backpropagation (artificial neural networks), support vector machines, and decision tree learning. **Reinforcement learning** is covered in Section 2.6 and refers to methods that solve reinforcement learning problems, where a sequence of actions is associated with positive or negative rewards, but not with a "target value" (the correct action). The paradigmatic algorithm here is temporal difference (TD) learning and its popular instantiation Q-learning. Section 5.6.3 outlines **unsupervised learning** which refers to algorithms that find patterns (e.g., clusters) in datasets that do *not* have target values. This includes clustering methods such as k-means, hierarchical clustering and self-organizing maps as well as frequent pattern mining methods such as Apriori and generalized sequential patterns. The chapter concludes with a number of notable algorithms that combine elements of the algorithms above to yield **hybrid** methods. In particular we cover neuroevolution and TD learning with ANN function approximation as the most popular hybrid algorithms used in the field of game AI.

## 2.1 General Notes

Before detailing each of the algorithm types we outline two overarching elements that bind together all the AI methods covered in this book. The former is the algorithm's **representation**; the second is its **utility**. On the one hand, any AI algorithm somehow stores and maintains knowledge obtained about a particular task at hand. On the other hand, most AI algorithms seek to find better representations of knowledge. This seeking process is driven by a utility function of some form. We should note that the utility is of no use solely in methods that employ static knowledge representations such as finite state machines or behavior trees.

### *2.1.1 Representation*

Appropriately representing knowledge is a key challenge for artificial intelligence at large and it is motivated by the capacity of the human brain to store and retrieve obtained knowledge about the world. The key questions that drive the design of representations for AI are as follows. How do people represent knowledge and how

can AI potentially mimic that capacity? What is the nature of knowledge? How generic can a representation scheme be? General answers to the above questions, however, are far from trivial at this point.

As a response to the open *general* questions regarding knowledge and its representation, AI has identified numerous and very *specific* ways to store and retrieve information which is authored, obtained, or learned. The representation of knowledge about a task or a problem can be viewed as the computational mapping of the task under investigation. On that basis, the representation needs to store knowledge about the task in a format that a machine is able to process, such as a data structure.

To enable any form of artificial intelligence knowledge needs to be represented computationally and the ways this can happen are many. Representation types include **grammars** such as grammatical evolution, **graphs** such as finite state machines or probabilistic models, **trees** such as decision trees, behavior trees and genetic programming, **connectionism** such as artificial neural networks, **genetic** such as genetic algorithms and evolutionary strategies and **tabular** such as temporal difference learning and Q-learning. As we will see in the remainder of this book, all above representation types find dissimilar uses in games and can be associated with various game AI tasks.

One thing is certain for any AI algorithm that is tried on a particular task: the chosen representation has a major impact on the performance of the algorithm. Unfortunately, the type of representation to be chosen for a task follows the *no free lunch theorem* [756], suggesting that there is no single representation type which is ideal for the task at hand. As a general set of guidelines, however, the representation chosen should be as *simple* as possible. Simplicity usually comes as a delicate balance between computational effort and algorithm performance as either being over-detailed or over-simplistic will affect the performance of the algorithm. Furthermore, the representation chosen should be as *small* as possible given the complexity of the task at hand. Neither simplicity nor size are trivial decisions to make with respect to the representation. Good representations come with sufficient practical wisdom and empirical knowledge about the complexity and the qualitative features of the problem the AI is trying to solve.

### 2.1.2 Utility

Utility in game theory (and economics at large) is a measure of rational choice when playing a game. In general, it can be viewed as a function that is able to assist a search algorithm to decide which path to take. For that purpose, the utility function samples aspects of the search space and gathers information about the "goodness" of areas in the space. In a sense, a utility function is an approximation of the solution we try to find. In other words, it is a **measure of goodness** of the existing representation we search through.

Similar concepts to the utility include the **heuristic** used by computer science and AI as an *approximate* way to solve a problem faster when *exact* methods are too

slow to afford, in particular associated with the tree search paradigm. The concept of **fitness** is used similarly as a utility function that measures the degree to which a solution is good, primarily, in the area of evolutionary computation. In mathematical optimization, the **objective**, **loss**, **cost**, or **error** function is the utility function to be minimized (or maximized if that is the objective). In particular, in supervised learning the error function represents how well an approach maps training examples to target (desired) outputs. In the area of reinforcement learning and Markov decision processes instead, the utility is named **reward**, which is a function an agent attempts to maximize by learning to take the right action in a particular state. Finally, in the area of **unsupervised learning** utility is often provided internally and within the representation via e.g., competitive learning or self-organization.

Similarly to selecting an appropriate representation, the selection of a utility function follows the no free lunch theorem. A utility is generally difficult to design and sometimes the design task is basically impossible. The simplicity of its design pays off, but the completeness as well. The quality of a utility function largely depends on thorough empirical research and practical experience, which is gained within the domain under investigation.

### *2.1.3 Learning = Maximize Utility (Representation)*

The utility function is the drive for search and essential for learning. On that basis, the utility function is the *training signal* of any machine learning algorithm as it offers a measure of goodness of the representation we have. Thereby it implicitly provides indications on what to do to further increase the current goodness of the presentation. Systems that do not require learning (such as AI methods that are based on ad-hoc designed representations; or expert-knowledge systems) do not require a utility. In supervised learning the utility is sampled from data—i.e., good input-output patterns. In reinforcement learning and evolutionary computation, instead, the training signal is provided by the environment—i.e., rewards for doing something well and punishments for doing something wrong. Finally, in unsupervised learning the training signal derives from the internal structure of the representation.

## 2.2 Ad-Hoc Behavior Authoring

In this section we discuss the first, and arguably the most popular, class of AI methods for game development. **Finite state machines**, **behavior trees** and **utility-based AI** are ad-hoc behavior authoring methods that have traditionally dominated the control of non-player characters in games. Their dominance is evident by the fact that the term *game AI* in the game development scene is still nowadays synonymous with the use of these methods.

### 2.2.1 Finite State Machines

A **Finite State Machine** (FSM) [230]—and FSM variants such as hierarchical FSMs—is the game AI method that dominated the control and decision making processes of non-player characters in games up until the mid-2000s.

FSMs belong to the expert-knowledge systems area and are represented as graphs. An FSM graph is an abstract representation of an interconnected set of objects, symbols, events, actions or properties of the phenomenon that needs to be ad-hoc designed (represented). In particular, the graph contains nodes (states) which embed some mathematical abstraction and edges (transitions) which represent a conditional relationship between the nodes. The FSM can only be in one state at a time; the current state can change to another if the condition in the corresponding transition is fulfilled. In a nutshell, an FSM is defined by three main components:

- A number of **states** which store information about a task—e.g., you are currently on the *explore* state.
- A number of **transitions** between states which indicate a state change and are described by a condition that needs to be fulfilled—e.g., if you hear a fire shot, move to the *alerted* state.
- A set of **actions** that need to be followed within each state—e.g., while in the *explore* state *move randomly* and *seek opponents*.

FSMs are incredibly simple to design, implement, visualize, and debug. Further they have proven they work well with games over the years of their co-existence. However, they can be extremely complex to design on a large scale and are, thereby, computationally limited to certain tasks within game AI. An additional critical limitation of FSMs (and all ad-hoc authoring methods) is that they are not flexible and dynamic (unless purposely designed). After their design is completed, tested and debugged there is limited room for adaptivity and evolution. As a result, FSMs end up depicting very predictable behaviors in games. We can, in part, overcome such a drawback by representing transitions as fuzzy rules [532] or probabilities [109].

#### 2.2.1.1 An FSM for Ms Pac-Man

In this section we showcase FSMs as employed to control the Ms Pac-Man agent. A hypothetical and simplified FSM controller for Ms Pac-Man is illustrated in Fig. 2.1. In this example our FSM has three states (seek pellets, chase ghosts and evade ghosts) and four transitions (ghosts flashing, no visible ghost, ghost in sight, and power pill eaten). While in the *seek pellets* state, Ms Pac-Man moves randomly up until it detects a pellet and then follows a pathfinding algorithm to eat as many pellets as possible and as soon as possible. If a power pill is eaten, then Ms Pac-Man moves to the *chase ghosts* state in which it can use any tree-search algorithm to chase the blue ghosts. When the ghosts start flashing, Ms Pac-Man moves to the *evade ghosts* state in which it uses tree search to evade ghosts so that none is visible

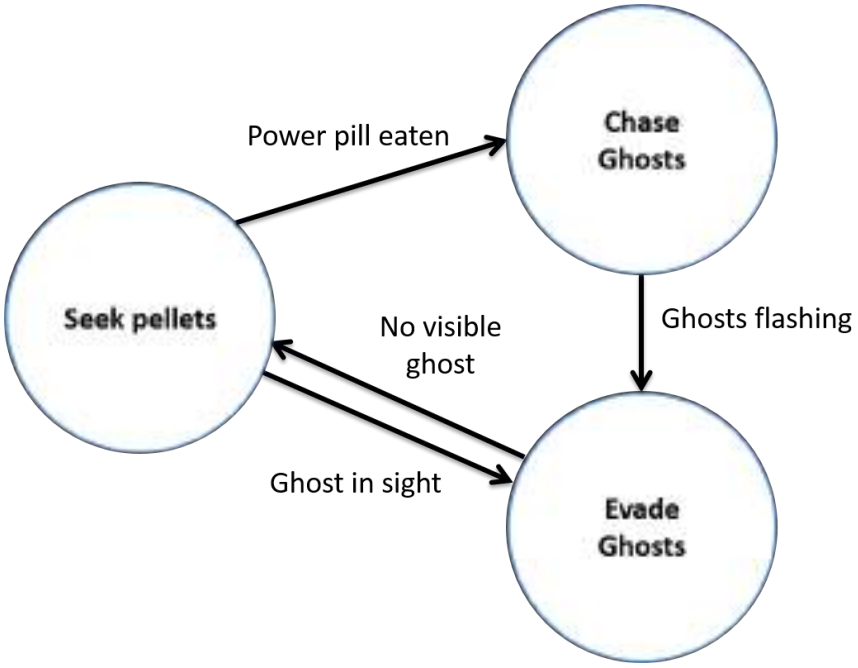


**Fig. 2.1** A high-level and simplified FSM example for controlling Ms Pac-Man.

within a distance; when that happens Ms Pac-Man moves back to the *seek pellets* state.

### *2.2.2 Behavior Trees*

A **Behavior Tree** (BT) [110, 112, 111] is an expert-knowledge system which, similarly to an FSM, models transitions between a finite set of tasks (or behaviors). The strength of BTs compared to FSMs is their modularity: if designed well, they can yield complex behaviors composed of simple tasks. The main difference between BT and FSMs (or even hierarchical FSMs) is that they are composed of *behaviors* rather than states. As with finite state machines, BTs are easy to design, test and debug, which made them dominant in the game development scene after their successful application in games such as *Halo 2* (Microsoft Game Studios, 2004) [291] and *Bioshock* (2K Games, 2007).

BT employs a tree structure with a root node and a number of parent and corresponding child nodes representing behaviors—see Fig. 2.2 for an example. We traverse a BT starting from the root. We then activate the execution of parent-child pairs as denoted in the tree. A child may return the following values to the parent in predetermined time steps (ticks): *run* if the behavior is still active, *success* if the

behavior is completed, *failure* if the behavior failed. BTs are composed of three node types: the **sequence**, the **selector**, and the **decorator** the basic functionality of which is described below:

- **Sequence** (see blue rectangle in Fig. 2.2): if the child behavior succeeds, the sequence continues and eventually the parent node succeeds if all child behaviors succeed; otherwise the sequence fails.
- **Selector** (see red rounded rectangle in Fig. 2.2): there are two main types of selector nodes: the *probability* and the *priority* selectors. When a probability selector is used child behaviors are selected based on parent-child probabilities set by the BT designer. On the other hand if priority selectors are used, child behaviors are ordered in a list and tried one after the other. Regardless of the selector type used, if the child behavior succeeds the selector succeeds. If the child behavior fails, the next child in the order is selected (in priority selectors) or the selector fails (in probability selectors).
- **Decorator** (see purple hexagon in Fig. 2.2): the decorator node adds complexity to and enhances the capacity of a single child behavior. Decorator examples include the number of times a child behavior runs or the time given to a child behavior to complete the task.

Compared to FSM, BTs are more flexible to design and easier to test; they still however suffer from similar drawbacks. In particular, their dynamicity is rather low given that they are static knowledge representations. The probability selector nodes may add to their unpredictability and methods to adapt their tree structures have already shown some promise [385]. There is also a certain degree of similarity between BTs and ABL (A Behavior Language) [440] introduced by Mateas and Stern for story-based believable characters; their dissimilarities have also been reported [749]. Note however that this section barely scratches the surface of what is possible with BT design as there are several extensions to their basic structure that help BTs improve on their modularity and their capacity to deal with more complex behavior designs [170, 627].

#### 2.2.2.1 A BT for Ms Pac-Man

Similarly to the FSM example above we use Ms Pac-Man to demonstrate the use of BTs in a popular game. In Fig. 2.3 we illustrate a simple BT for the *seek pellets* behavior of Ms Pac-Man. While in the *seek pellets* sequence behavior Ms Pac-Man will first *move* (selector), it will then find a pellet and finally it will keep eating pellets until a ghost is found in sight (decorator). While in the *move* behavior—which is a priority selector—Ms Pac-Man will prioritize ghost-free corridors over corridors with pellets and over corridors without pellets.

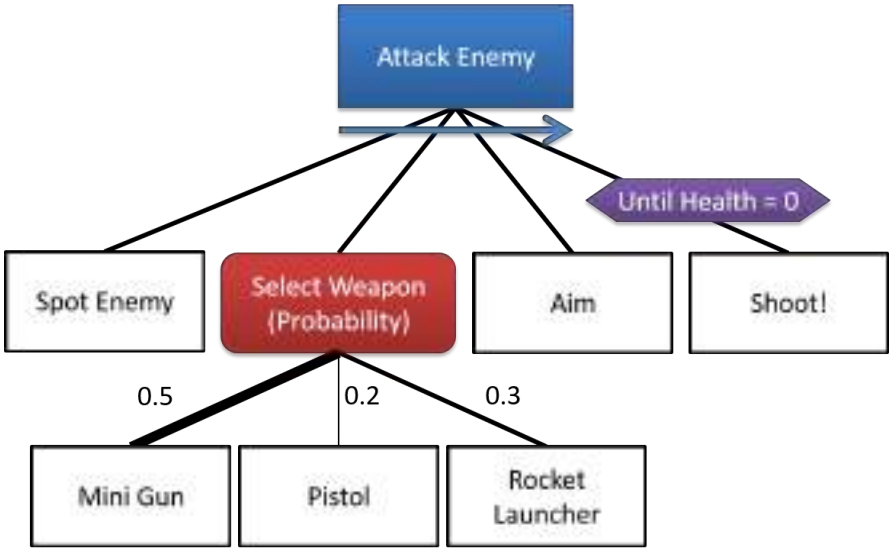


**Fig. 2.2** A behavior tree example. The root of the BT is a sequence behavior (attack enemy) which executes the child behaviors *spot enemy*, *select weapon*, *aim* and *shoot* in sequence from left to right. The *select weapon* behavior is a probability selector giving higher probability—denoted by the thickness of the parent-child connecting lines—to the mini gun (0.5) compared to the rocket launcher (0.3) or the pistol (0.2). Once in the *shoot* behavior the decorator *until health = 0* requests the behavior to run until the enemy dies.

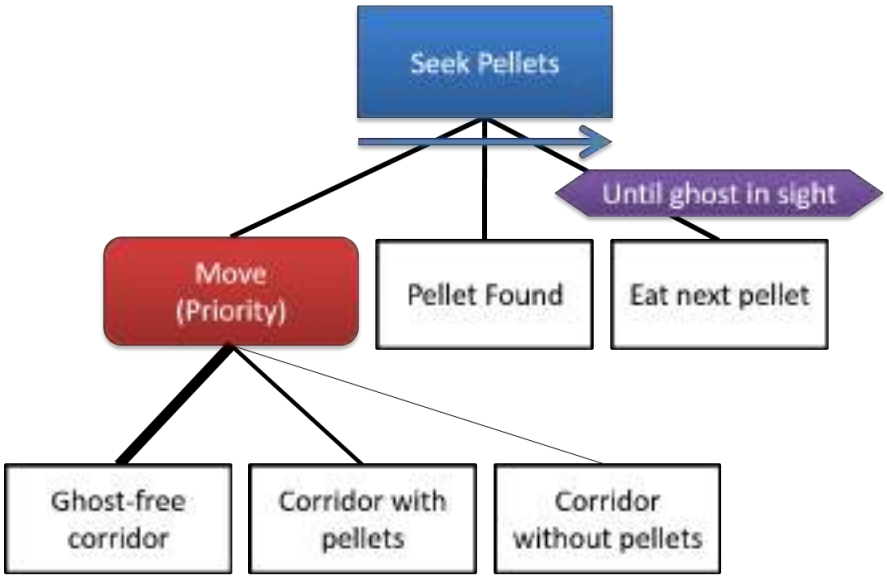


**Fig. 2.3** A BT example for the *seek pellets* behavior of Ms Pac-Man.

## 2.3 Tree Search

It has been largely claimed that most, if not all, of artificial intelligence is really just search. Almost every AI problem can be cast as a search problem, which can be solved by finding the best (according to some measure) plan, path, model, function, etc. Search algorithms are therefore often seen as being at the core of AI, to the point that many textbooks (such as Russell and Norvig's famous textbook [582]) start with a treatment of search algorithms.

The algorithms presented below can all be characterized as **tree search algorithms** as they can be seen as building a **search tree** where the root is the node representing the state where the search starts. Edges in this tree represent actions the agent takes to get from one state to another, and nodes represent states. Because there are typically several different actions that can be taken in a given state, the tree

[1] https://docs.unrealengine.com/latest/INT/Engine/

[2] For instance, see http://nodecanvas.paradoxnotion.com/ or http://www.opsive.com/.

[3] http://eej.dk/community/documentation/behave/0-Introduction.html

branches. Tree search algorithms mainly differ in which branches are explored and in what order.

### 2.3.1 Uninformed Search

Uninformed search algorithms are algorithms which search a state space without any further information about the goal. The basic uninformed search algorithms are commonly seen as fundamental computer science algorithms, and are sometimes not even seen as AI.

**Depth-first search** is a search algorithm which explores each branch as far as possible before backtracking and trying another branch. At every iteration of its main loop, depth-first search selects a branch and then moves on to explore the resulting node in the next iteration. When a terminal node is reached—one from which it is not possible to advance further—depth-first search advances up the list of visited nodes until it finds one which has unexplored actions. When used for playing a game, depth-first search explores the consequences of a single move until the game is won or lost, and then goes on to explore the consequences of taking a different move close to the end states.

**Breadth-first search** does the opposite of depth-first search. Instead of exploring all the consequences of a single action, breadth-first search explores all the actions from a single node before exploring any of the nodes resulting from taking those actions. So, all nodes at depth one are explored before all nodes at depth two, then all nodes at depth three, etc.

While the aforementioned are fundamental uninformed search algorithms, there are many variations and combinations of these algorithms, and new uninformed search algorithms are being developed. More information about uninformed search algorithms can be found in Chapter 4 of [582].

It is rare to see uninformed search algorithms used effectively in games, but there are exceptions such as iterative width search [58], which does surprisingly well in general video game playing, and the use of breadth-first search to evaluate aspects of strategy game maps in *Sentient Sketchbook* [379]. Also, it is often illuminating to compare the performance of state-of-the-art algorithms with a simple uninformed search algorithm.

#### 2.3.1.1 Uninformed Search for Ms Pac-Man

A depth-first approach in Ms Pac-Man would normally consider the branches of the game tree until Ms Pac-Man either completes the level or loses. The outcome of this search for each possible action would determine which action to take at a given moment. Breadth-first instead would first explore all possible actions of Ms Pac-Man at the current state of the game (e.g., going left, up, down or right) and

would then explore all their resulting nodes (children) and so on. The game tree of either method is too big and complex to visualize within a Ms Pac-Man example.

### 2.3.2 Best-First Search

In **best-first search**, the expansion of nodes in the search tree is informed by some knowledge about the goal state. In general, the node that is closest to the goal state by some criterion is expanded first. The most well-known best-first search algorithm is A* (pronounced A star). The A* algorithm keeps a list of "open" nodes, which are next to an explored node but which have not themselves been explored. For each open node, an estimate of its distance from the goal is made. New nodes are chosen to explore based on a lowest cost basis, where the cost is the distance from the origin node plus the estimate of the distance to the goal.

A* can easily be understood as navigation in two- or three-dimensional space. Variants of this algorithm are therefore commonly used for **pathfinding** in games. In many games, the "AI" essentially amounts to non-player characters using A* pathfinding to traverse scripted points. In order to cope with large, deceptive spaces numerous modifications of this basic algorithm have been proposed, including hierarchical versions of A* [61, 661], real-time heuristic search [82], **jump point search** for uniform-cost grids [246], 3D pathfinding algorithms [68], planning algorithms for dynamic game worlds [495] that enable the animation of crowds in collision-free paths [631] and approaches for pathfinding in navigation meshes [68, 722]. The work of Steve Rabin and Nathan Sturtevant on grid-based pathfinding [551, 662] and pathfinding architectures [550] are notable examples. Sturtevant and colleagues have also been running a dedicated competition to grid-based path-planning [665] since 2012.[4] For the interested reader Sturtevant [663] has released a list of benchmarks for grid-based pathfinding in games[5] including *Dragon Age: Origins* (Electronic Arts, 2009), *StarCraft* (Blizzard Entertainment, 1998) and *Warcraft III: Reign of Chaos* (Blizzard Entertainment, 2002).

However, A* can also be used to search in the space of game states, as opposed to simply searching physical locations. This way, best-first search can be used for **planning** rather than just navigation. The difference is in taking the changing state of the world (rather than just the changing state of a single agent) into account. Planning with A* can be surprisingly effective, as evidenced by the winner of the 2009 Mario AI Competition—where competitors submitted agents playing *Super Mario Bros* (Nintendo, 1985)—being based on a simple A* planner that simply tried to get to the right end of the screen at all times [717, 705] (see also Fig. 2.5).

[4] http://movingai.com/GPPC/

[5] http://movingai.com/benchmarks/


**Fig. 2.5** The A* controller of the 2009 Mario AI Competition champion by R. Baumgarten [705]. The red lines illustrate possible future trajectories considered by the A* controller of Mario, taking the dynamic nature of the game into account.

#### 2.3.2.1 Best-First Search for Ms Pac-Man

Best-first search can be applicable in Pac-Man in the form of A*. Following the paradigm of the 2009 Mario AI competition champion, Ms Pac-Man can be controlled by an A* algorithm that searches through possible game states within a short time frame and takes a decision on where to move next (up, down, left or right). The game state can be represented in various ways: from a very direct, yet costly, representation that takes ghost and pellet coordinates into account to an indirect representation that considers the distance to the closest ghost or pellet. Regardless of the representation chosen, A* requires the design of a cost function that will drive the search. Relevant cost functions for Ms Pac-Man would normally reward moves to areas containing pellets and penalizing areas containing ghosts.

### *2.3.3 Minimax*

For single-player games, simple uninformed or informed search algorithms can be used to find a path to the optimal game state. However, for two-player adversarial

games, there is another player that tries to win as well, and the actions of each player depend very much on the actions of the other player. For such games we need adversarial search, which includes the actions of two (or more) adversarial players. The basic adversarial search algorithm is called **Minimax**. This algorithm has been used very successfully for playing classic perfect-information two-player board games such as Checkers and Chess, and was in fact (re)invented specifically for the purpose of building a Chess-playing program [725].

The core loop of the Minimax algorithm alternates between player 1 and player 2—such as the white and black player in Chess—named the *min* and the *max* player. For each player, all possible moves are explored. For each of the resulting states, all possible moves by the other player are also explored, and so on until all the possible combinations of moves have been explored to the point where the game ends (e.g., with a win, a loss or a draw). The result of this process is the generation of the whole game tree from the root node down to the leaves. The outcome of the game informs the utility function which is applied onto the leaf nodes. The utility function estimates how good the current game configuration is for a player. Then, the algorithm traverses up the search tree to determine what action each player would have taken at any given state by backing-up values from leaves through the branch nodes. In doing so, it assumes that each player tries to play optimally. Thus, from the standpoint of the *max* player, it tries to maximize its score, whereas *min* tries to minimize the score of *max*; hence, the name *Minimax*. In other words, a *max* node of the tree computes the max of its child values whereas a *min* node computes the min of its child values. The optimal winning strategy is then obtained for *max* if, on *min*'s turn, a win is obtainable for *max* for *all moves* that *min* can make. The corresponding optimal strategy for *min* is when a win is possible independently of what move *max* will take. To obtain a winning strategy for max, for instance, we start at the root of the tree and we iteratively choose the moves leading to child nodes of highest value (on min's turn the child nodes with the lowest value are selected instead). Figure 2.6 illustrates the basic steps of Minimax through a simple example.

Of course, exploring all possible moves and countermoves is infeasible for any game of interesting complexity, as the size of the search tree increases exponentially with the depth of the game or the number of moves that are simulated. Indicatively, tic-tac-toe has a game tree size of $9! = 362,880$ states which is feasible to traverse through; however, the Chess game tree has approximately $10^{154}$ nodes which is infeasible to search through with modern computers. Therefore, almost all actual applications of the Minimax algorithm cut off search at a given depth, and use a **state evaluation** function to evaluate the desirability of each game state at that depth. For example, in Chess a simple state evaluation function would be to merely sum the number of white pieces on the board and subtract the number of black pieces; the higher this number is, the better the situation is for the white player. (Of course, much more sophisticated board evaluation functions are commonly used.) Together with improvements to the basic Minimax algorithm such as $\alpha$-$\beta$ **pruning** and the use of non-deterministic state evaluation functions, some very competent programs emerged for many classic games (e.g., IBM's Deep Blue). More information about Minimax and other adversarial search algorithms can be found in Chapter 6 of [582].

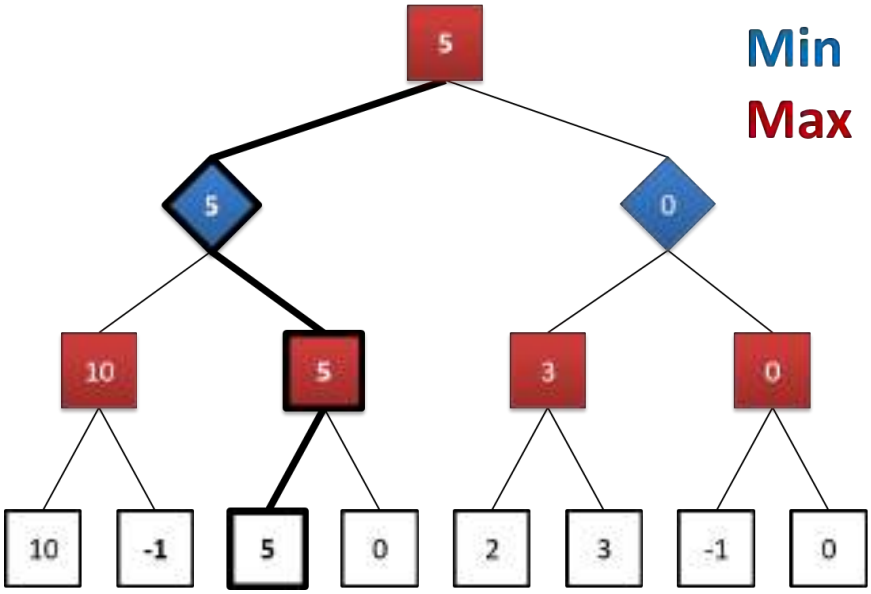


**Fig. 2.6** An abstract game tree illustrating the Minimax algorithm. In this hypothetical game of two options for each player max (represented as red squares) plays first, min (represented as blue diamonds) plays second and then max plays one last time. White squares denote terminal nodes containing a winning (positive), a losing (negative) or a draw (zero) score for the max player. Following the Minimax strategy, the scores (utility) are traversed up to the root of the game tree. The optimal play for max and min is illustrated in bold. In this simple example if both players play optimally, max wins a score of 5.

#### 2.3.3.1 Minimax for Ms Pac-Man

Strictly speaking, Minimax is not applicable to Ms Pac-Man as the game is non-deterministic and, thus, the Minimax tree is formally unknown. (Of course Minimax variants with heuristic evaluation functions can be eventually applicable.) Minimax is however applicable to Ms Pac-Man's deterministic ancestor, *Pac-Man* (Namco, 1980). Again strictly speaking, Pac-Man is a single-player adversarial game. As such Minimax is applicable only if we assume that Pac-Man plays against adversaries (ghosts) who make optimal decisions. It is important to note that ghosts' movements are not represented by tree nodes; instead, they are simulated based on their assumed optimal play. Game tree nodes in Pac-Man may represent the game state including the position of Pac-Man, the ghosts, and the current pellets and power pills available. The branches of the Minimax tree are the available moves of the Pac-Man in each game state. The terminal nodes can, for instance, feature either a binary utility (1 if Pac-Man completes the level; 0 if Pac-Man was killed by a ghost) or the final score of the game.

### 2.3.4 Monte Carlo Tree Search

There are many games which Minimax will not play well. In particular, games with a high **branching factor** (where there are many potential actions to take at any given point in time) lead to Minimax that will only ever search a very shallow tree. Another aspect of games which frequently throws spanners in the works of Minimax is when it is hard to construct a good state evaluation function. The board game Go is a deterministic, perfect information game that is a good example of both of these phenomena. Go has a branching factor of approximately 300, whereas Chess typically has around 30 actions to choose from. The positional nature of the Go game, which is all about surrounding the adversary, makes it very hard to correctly estimate the value of a given board state. For a long time, the best Go-playing programs in the world, most of which were based on Minimax, could barely exceed the playing strength of a human beginner. In 2007, **Monte Carlo Tree Search** (MCTS) was invented and the playing strength of the best Go programs increased drastically.

Beyond complex perfect information, deterministic games such as Go, Chess and Checkers, **imperfect information** games such a Battleship, Poker, Bridge and/or **non-deterministic** games such as backgammon and monopoly cannot be solved via Minimax due to the very nature of the algorithm. In such games, MCTS not only overcomes the tree size limitation of Minimax but, given sufficient computation, it approximates the Minimax tree of the game.

So how does MCTS handle high branching factors, lack of good state evaluation functions, and lack of perfect information and determinism? To begin with, it does not search all branches of the search tree to an even depth, instead it concentrates on the more promising branches. This makes it possible to search certain branches to a considerable depth even though the branching factor is high. Further, to get around the lack of good evaluation functions, determinism and imperfect information, the standard formulation of MCTS uses **rollouts** to estimate the quality of the game state, randomly playing from a game state until the end of the game to see the expected win (or loss) outcome. The utility values obtained via the random simulations may be used efficiently to adjust the policy towards a best-first strategy (a Minimax tree approximation).

At the start of a run of the MCTS algorithm, the tree consists of a single node representing the current state of the game. The algorithm then iteratively builds a search tree by adding and evaluating new nodes representing game states. This process can be interrupted at any time, rendering MCTS an **anytime** algorithm. MCTS requires only two pieces of information to operate: the *game rules* that would, in turn, yield the available moves in the game and the *terminal state evaluation*—whether that is win, a loss, a draw, or a game score. The vanilla version of MCTS does not require a heuristic function, which is, in turn, a key advantage over Minimax.

The core loop of the MCTS algorithm can be divided into four steps: **Selection**, **Expansion** (the first two steps are also known as *tree policy*), **Simulation** and **Backpropagation**. The steps are also depicted in Fig. 2.7.

**Selection**: In this phase, it is decided which node should be expanded. The process starts at the root of the tree, and continues until a node is selected which has unexpanded children. Every time a node (action) is to be selected within the existing tree a child node $j$ is selected to maximise the UCB1 formula:

$$\text{UCB1} = \overline{X}_j + 2C_p\sqrt{\frac{2\ln n}{n_j}} \tag{2.1}$$

where $\overline{X}_j$ is the average reward of all nodes beneath this node, $C_p$ is an exploration constant (often set to $1/\sqrt{2}$), $n$ is the number of times the parent node has been visited, and $n_j$ is the number of times the child node $j$ has been visited. It is important to note that while UCB1 is the most popular formula used for action selection it is certainly not the only one available. Beyond equation (2.1) other options include epsilon-greedy, Thompson sampling, and Bayesian bandits. For instance, Thompson sampling selects actions stochastically based on their posterior probabilities of being optimal [692].

**Expansion**: When a node is selected that has unexpanded children—i.e., that represents a state from which actions can be taken that have not been attempted yet—one of these children is chosen for *expansion*, meaning that a simulation is done starting in that state. Selecting which child to expand is often done at random.

**Simulation (Default Policy)**: After a node is expanded, a simulation (or *roll-out*) is done starting from the non-terminal node that was just expanded until the end of game to produce a value estimate. Usually, this is performed by taking random actions until a termination state is reached, i.e., until the game is either won or lost. The state at the end of the game (e.g., $-1$ if losing, $+1$ if winning, but could be more nuanced) is used as the reward ($\Delta$) for this simulation, and propagated up the search tree.

**Backpropagation**: The reward (the outcome of the simulation) is added to the total reward $X$ of the new node. It is also "backed up": added to the total reward of its parent node, its parent's parent and so on until the root of the tree.

The simulation step might appear counter-intuitive—taking random actions seems like no good way to play a game—but it provides a relatively unbiased estimate of the quality of a game state. Essentially, the better a game state is, the more simulations are likely to end up winning the game. At least, this is true for games like Go where a game will always reach a terminal state within a certain relatively small number of moves (400 for Go). For other games like Chess, it is theoretically possible to play an arbitrary number of moves without winning or losing the game. For many video games, it is *probable* that any random sequence of actions will not end the game unless some timer runs out, meaning that most simulations will be

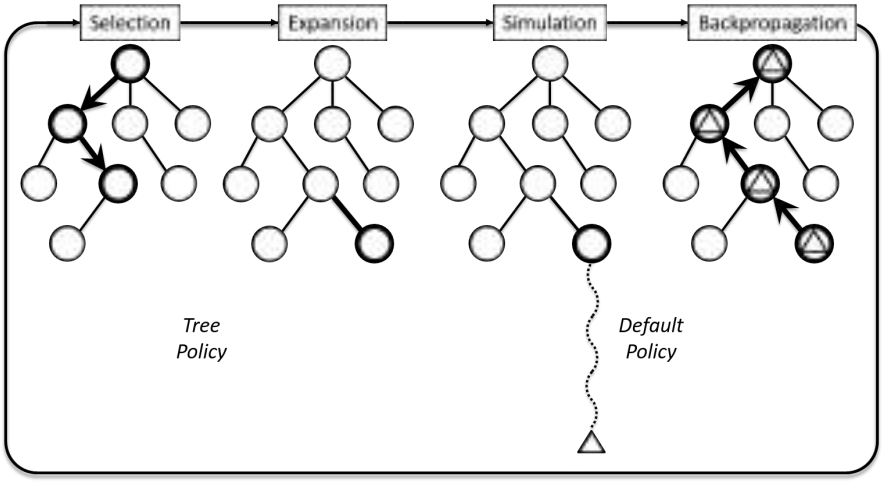


**Fig. 2.7** The four basic steps of MCTS exemplified through one iteration of the algorithm. The figure is a recreation of the corresponding MCTS outline figure by Chaslot et al. [118].

very long (tens or hundreds of thousands of steps) and not yield useful information. For example, in *Super Mario Bros* (Nintendo,1985), the application of random actions would most likely make Mario dance around his starting point until his time is up [294]. In many cases it is therefore useful to complement the simulation step with a state evaluation function (as commonly used in Minimax), so that a simulation is performed for a set number of steps and if a terminal state is not reached a state evaluation is performed in lieu of a win-lose evaluation. In some cases it might even be beneficial to replace the simulation step entirely with a state evaluation function.

It is worth noting that there are many variations of the basic MCTS algorithm—it may in fact be more useful to see MCTS as an algorithm family or framework rather than a single algorithm.

#### 2.3.4.1 MCTS for Ms Pac-Man

MCTS can be applicable to the real-time control of the Ms Pac-Man agent. There are obviously numerous ways to represent a game state (and thereby a game tree node) and design a reward function for the game, which we will not discuss in detail here. In this section, instead, we will outline the approach followed by Pepels et al. [524] given its success in obtaining high scores for Ms Pac-Man. Their agent, named Maastricht, managed to obtain over $87,000$ points and was ranked first (among 36 agents) in the Ms Pac-Man competition of the IEEE Computational Intelligence and Games conference in 2012.

When MCTS is used for real-time decision making a number of challenges become critical. First, the algorithm has limited rollout computational budget which increases the importance of heuristic knowledge. Second, the action space can be

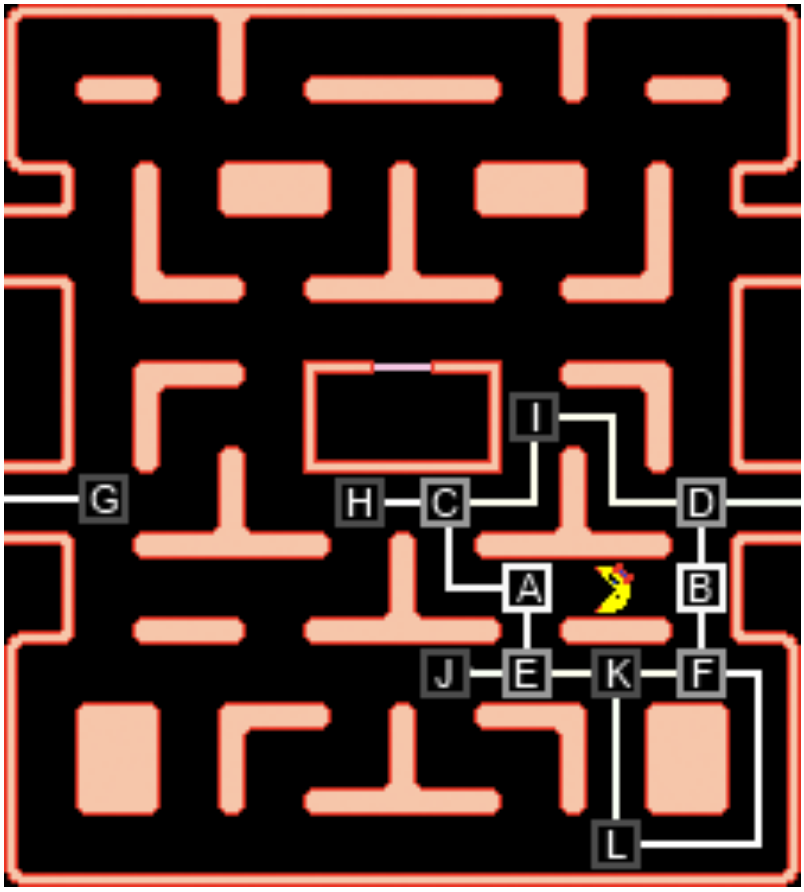


**Fig. 2.8** The junction-based representation of a game state for the Maastricht MCTS controller [524]. All letter nodes refer to game tree nodes (decisions) for Ms Pac-Man. Imaged adapted from [524] with permission from authors.

particularly fine-grained which suggests that macro-actions are a more powerful way to model the game tree; otherwise the agent's planning will be very short-term. Third, there might be no terminal node in sight which calls for good heuristics and possibly restricting the simulation depth. The MCTS agent of Pepels et al. [524] managed to cope with all the above challenges of using MCTS for real-time control by using a restricted game tree and a junction-based game state representation (see Fig. 2.8).

### 2.3.5 Further Reading

The basic search algorithms are well covered in Russell and Norvig's classic AI textbook [582]. The A* algorithm was invented in 1972 for robot navigation [247]; a good description of the algorithm can be found in Chapter 4 of [582]. There is plenty of more advanced material on tailoring and optimizing this algorithm for specific game problems in dedicated game AI books such as [546]. The different components of Monte Carlo tree search [141] were invented in 2006 and 2007 in the context of playing Go [142]; a good overview of and introduction to MCTS and some of its variants is given in a survey paper by Browne et al. [77].

## 2.4 Evolutionary Computation

While tree search algorithms start from the root node representing an origin state, and build a search tree based on the available actions, *optimization* algorithms do not build a search tree; they only consider complete solutions, and not the path taken to get there. As mentioned earlier in Section 2.1, all optimization algorithms assume that there is something to optimize solutions for; there must be an **objective**, alternatively called **utility function**, **evaluation function** or **fitness function**, which can assign a numerical value (the **fitness**) to a solution, which can be maximized (or minimized). Given a utility function, an optimization algorithm can be seen as an algorithm that seeks in a search space solutions that have the highest (or lowest) value of that utility.

A broad family of optimization algorithms is based on randomized variation of solutions, where one or multiple solutions are kept at any given time, and new solutions (or candidates, or search points; different terminology is used by different authors) are created through randomly changing some of the existing solutions, or maybe combining some of them. Randomized optimization algorithms which keep multiple solutions are called **evolutionary algorithms**, by analogy with natural evolution.

Another important concept when talking about optimization algorithms (and AI at large as covered in Section 2.1) is their **representation**. All solutions are represented in some way, for example, as fixed-size vectors of real numbers, or variable-length strings of characters. Generally, the same artifact can be represented in many different ways; for example, when searching for a sequence of actions that solves a maze, the action sequence can be represented in several different ways. In the most direct representation, the character at step $t$ determines what action to take at time step $t+1$. A somewhat more indirect representation for a sequence of actions would be a sequence of tuples, where the character at time step $t$ decides what action to take and the number $t+n$ determines for how many time steps $n$ to take that action. The choice of representation has a big impact on the efficiency and efficacy of the search algorithm, and there are several tradeoffs at play when making these choices.

**Optimization** is an extremely general concept, and optimization algorithms are useful for a wide variety of tasks in AI as well as in computing more generally. Within AI and games, optimization algorithms such as evolutionary algorithms have been used in many roles as well. In Chapter 3 we explain how optimization algorithms can be used for searching for game-playing agents, and also for searching for action sequences (these are two very different uses of optimization that are both in the context of game-playing); in Chapter 4 we explain how we can use optimization to create game content such as levels; and in Chapter 5 we discuss how to use optimization to find player models.

### *2.4.1 Local Search*

The simplest optimization algorithms are the local optimization algorithms. These are so called because they only search "locally", in a small part of the search space, at any given time. A local optimization algorithm generally just keeps a single solution candidate at any given time, and explores variations of that solution.

The arguably simplest possible optimization algorithm is the **hill climber**. In its most common formulation, which we can call the deterministic formulation, it works as follows:

1. *Initialization:* Create a solution $s$ by choosing a random point in search space. Evaluate its fitness.
2. Generate all possible neighbors of $s$. A neighbor is any solution that differs from $s$ by at most a certain given distance (for example, a change in a single position).
3. Evaluate all the neighbors with the fitness function.
4. If none of the neighbors has a better fitness score than $s$, exit the algorithm and return $s$.
5. Otherwise, replace $s$ with the neighbor that has the highest fitness value and go to step 2.

The deterministic hill climber is only practicable when the representation is such that each solution has a small number of neighbors. In many representations there are an astronomically high number of neighbors. It is therefore preferable to use variants of hill climbers that may guide the search effectively. One approach is the **gradient-based hill climber** that follows the gradient towards minimizing a cost function. That algorithmic approach trains artificial neural networks for instance (see Section 2.5). Another approach that we cover here is the **randomized hill climber**. This instead relies on the concept of **mutation**: a small, random change to a solution. For example, a string of letters can be mutated by randomly flipping one or two characters to some other character (see Fig. 2.9), and a vector of real

(a) **Mutation**: A number of genes is selected to be mutated with a small probability e.g., less than 1%. The selected genes are highlighted with a red outline at the top chromosome and are mutated by flipping their binary value (red genes) at the bottom chromosome.

(b) **Inversion**: Two positions in the offspring are randomly chosen and the positions between them—the gene sequence highlighted by a red outline at the top chromosome—are inversed (red genes) at the bottom chromosome.

**Fig. 2.9** Two ways of mutating a binary chromosome. In this example we use a chromosome of eleven genes. A chromosome is selected (top bit-string) and mutated (bottom bit-string).

numbers can be mutated by adding another vector to it drawn from a random distribution around zero, and with a very small standard deviation. Macro-mutations such as gene **inversion** can also be applied as visualized in Fig. 2.9. Given a representation, fitness function and mutation operator, the randomized hill climber works as follows:

1. *Initialization*: Create a solution $s$ by choosing a random point in the search space. Evaluate its fitness.
2. *Mutation*: Generate an offspring $s'$ by mutating $s$.
3. *Evaluation*: Evaluate the fitness of $s'$.
4. *Replacement*: If $s'$ has higher fitness than $s$, replace $s$ with $s'$.
5. Go to step 2.

While very simple, the randomized hill climber can be surprisingly effective. Its main limitation is that it is liable to get stuck in local optima. A **local optimum** is sort of a "dead end" in search space from which there is "no way out"; a point from which there are no better (higher-fit) points within the immediate vicinity. There are many ways of dealing with this problem. One is to simply restart the hill climber at a new randomly chosen point in the search space whenever it gets stuck. Another is **simulated annealing**, to accept moving to solutions with *lower* fitness with a given probability; this probability gradually diminishes during the search. A far more popular response to the problem of local optima is to keep not just a single solution at any time, but a **population** of solutions.

#### 2.4.1.1 Local Search for Ms Pac-Man

While we can think of a few ways one can apply local search in Ms Pac-Man we outline an example of its use for controlling path-plans. Local search could, for

instance, evolve short local plans (action sequences) of Ms Pac-Man. A solution could be represented as a set of actions that need to be taken and its fitness could be determined by the score obtained after following this sequence of actions.

### 2.4.2 Evolutionary Algorithms

Evolutionary algorithms are randomized **global** optimization algorithms; they are called global rather than local because they search many points in the search space simultaneously, and these points can be far apart. They accomplish this by keeping a population of solutions in memory at any given time. The general idea of evolutionary computation is to optimize by "breeding" solutions: generate many solutions, throw away the bad ones and keep the good (or at least less bad) ones, and create new solutions from the good ones.

The idea of keeping a population is taken from Darwinian evolution by natural selection, from which evolutionary algorithms also get their name. The size of the population is one of the key parameters of an evolutionary algorithm; a population size of 1 yields something like a randomized hill climber, whereas populations of several thousand solutions are not unheard of.

Another idea which is taken from evolution in nature is **crossover**, also called **recombination**. This is the equivalent of sexual reproduction in the natural world; two or more solutions (called **parents**) produce an offspring by combining elements of themselves. The idea is that if we take two good solutions, a solution that is a combination of these two—or intermediate between them—ought to be good as well, maybe even better than the parents. The offspring operator is highly dependent on the solution representation. When the solution is represented as a string or a vector, operators such as uniform crossover (which flips a fair coin and randomly picks values from each parent for each position in the offspring) or one-point crossover (where a position $p$ in the offspring is randomly chosen, and values of positions before $p$ are taken from parent 1 and values of positions after $p$ are taken from parent 2) can be used. Crossover can be applied to any chromosome representation varying from a bit-string to a real-valued vector. Figure 2.10 illustrates these two crossover operators. It is in no way guaranteed, however, that the crossover operator generates an offspring that is anything as highly fit as the parents. In many cases, crossover can be highly destructive. If crossover is used, it is therefore important that the offspring operator is chosen with care for each problem. Figure 2.11 illustrates this possibility through a simple two-dimensional example.

The basic template for an evolutionary algorithm is as follows:

1. *Initialization:* The population is filled with $N$ solutions created randomly, i.e., random points in search space. Known highly-fit solutions can also be added to this initial population.


(a) **1-point crossover**: The vertical line across the two parents denotes the crossover point at position $p$.

(b) **Uniform crossover**: To select genes from each parent to form offspring the operator flips a fair coin at each position of the chromosome.

**Fig. 2.10** Two popular types of crossover used in evolutionary algorithms. In this example we use a **binary** representation and a chromosome size of eleven genes. The two bit-strings used in both crossover operators represent the two parents selected for recombination. Red and blue genes represent the two different offspring emerged from each crossover operator. Note that the operators are directly applicable to **real-valued** (floating point) representations too.

2. *Evaluation:* The fitness function is used to evaluate all solutions in the population and assign fitness values to them.
3. *Parent selection:* Based on fitness and possibly other criteria, such as distance between solutions, those population members that will be used for reproduction are selected. Selection strategies include methods directly or indirectly dependent on the fitness of the solutions, including roulette-wheel (proportionally to fitness), ranking (proportionally to rank in population) and tournament.
4. *Reproduction:* Offspring are generated through crossover from parents, or through simply copying parent solutions, or some combination of these.
5. *Variation:* Mutation is applied to some or all of the parents and/or offspring.
6. *Replacement:* In this step, we select which of the parents and/or offspring will make it to the next generation. Popular replacement strategies of the current population include the **generational** (parents die; offspring replace them), **steady state** (offspring replaces worst parent if and only if offspring is better) and **elitism** (generational, but best $x\%$ of parents survive) approaches.
7. *Termination:* Are we done yet? Decide based on how many generations or evaluations have elapsed (**exhaustion**), the highest fitness attained by any solution (**success**), and/or some other termination condition.
8. Go to step 2.

Every iteration of the main loop (i.e., every time we reach step 2) is called a **generation**, keeping with the nature-inspired terminology. The total number of fitness evaluations performed is typically proportional to the size of the population times the number of generations.

This high-level template can be implemented and expanded in a myriad different ways; there are thousands of evolutionary or evolution-like algorithms out there, and

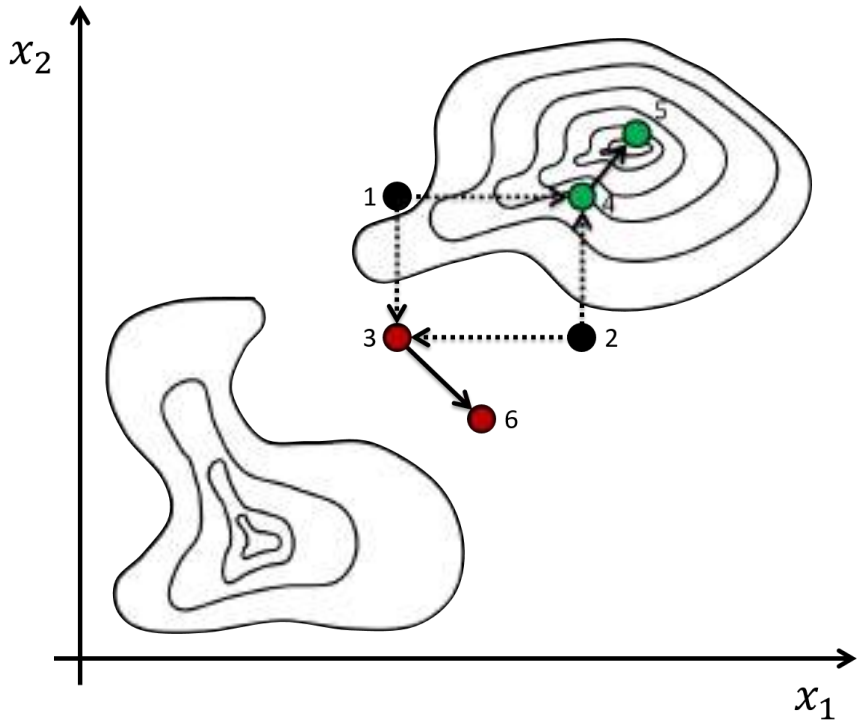


**Fig. 2.11** An illustration of the mutation and crossover operators in a simplified two-dimensional fitness landscape. The problem is represented by two real-valued variables ($x_1$ and $x_2$) that define the two genes of the vector chromosome. The fitness landscape is represented by the contour lines on the 2D plane. Chromosomes 1 and 2 are selected to be parents. They are recombined via 1-point crossover (dotted arrows) which yields offspring 3 and 4. Both offspring are mutated (solid arrows) to yield solutions 5 and 6. Operators that lead to poorer-fit or higher-fit solutions are, respectively, depicted with green and red color.

many of them rearrange the overall flow, add new steps and remove existing steps. In order to make this template a bit more concrete, we will give a simple example of a working evolutionary algorithm below. This is a form of **evolution strategy**, one of the main families of evolutionary algorithms. While the $\mu + \lambda$ evolution strategy is a simple algorithm that can be implemented in 10 to 20 lines of code, it is a fully functional global optimizer and quite useful. The two main parameters are $\mu$, which signifies the "elite" or the size of the part of the population that is kept every generation, and $\lambda$, the size of the part of the population that is re-generated every generation.

1. Fill the population with $\mu + \lambda$ randomly generated solutions.
2. Evaluate the fitness of all solutions.

3. Sort the population by decreasing fitness, so that the lowest-numbered solutions have highest fitness.
4. Remove the least fit $\lambda$ individuals.
5. Replace the removed individuals with copies of the $\mu$ best individuals.
6. Mutate the offspring.
7. Stop if success or exhaustion. Otherwise go to step 2.

Evolution strategies, the type of algorithms which the $\mu + \lambda$ evolution strategy above is a simple example of, are characterized by a reliance on mutation rather than crossover to create variation, and by the use of self-adaptation to adjust mutation parameters (though that is not part of the simple algorithm above). They are also generally well suited to optimize artifacts represented as vectors of real numbers, so-called continuous optimization. Some of the very best algorithms for continuous optimization, such as the covariance matrix adaptation evolution strategy (CMA-ES) [245] and the natural evolution strategy (NES) [753], are conceptual descendants of this family of algorithms.

Another prominent family of evolutionary algorithms is **genetic algorithms** (GAs). These are characterized by a reliance on crossover rather than mutation for variation (some genetic algorithms have no mutation at all), fitness-proportional selection and solutions being often represented as bit-strings or other discrete strings. It should be noted, however, that the distinctions between different types of evolutionary algorithms are mainly based on their historical origins. These days, there are so many variations and such extensive hybridization that it often makes little sense to categorize a particular algorithm as belonging to one or the other family.

A variant of evolutionary algorithms emerges from the need of satisfying particular constraints within which a solution is not only fit but also **feasible**. When evolutionary algorithms are used for constrained optimization we are faced with a number of challenges such as that mutation and crossover cannot preserve or guarantee the feasibility of a solution. It may very well be that a mutation or a recombination between two parents may yield an infeasible offspring. One approach to deal with constraint handling is **repair**, which could be any process that turns infeasible individuals into feasible ones. A second approach is to modify the genetic operators so that the probability of an infeasible individual to appear becomes smaller. A popular approach is to merely penalize the existence of infeasible solutions by assigning them low fitness values or, alternatively, in proportion to the number of constraint violations. This strategy however may over-penalize the actual fitness of a solution which in turn will result in its rapid elimination from the population. Such a property might be undesirable and is often accused for the weak performance of evolutionary algorithms on handling constraints [456]. As a response to this limitation the **feasible-infeasible 2-population** (FI-2pop) algorithm [341] evolves two populations, one with feasible and one with infeasible solutions. The infeasible population optimizes its members towards minimizing the distance from feasibility. As the infeasible population converges to the border of feasibility, the likelihood of dis-

covering new feasible individuals increases. Feasible offspring of infeasible parents are transferred to the feasible population, boosting its diversity (and vice versa for infeasible offspring). FI-2pop has been used in games on instances where we require fit and feasible solutions such as well-designed and playable game levels [649, 379].

Finally, another blend of evolutionary algorithms considers more than one objective when attempting to find a solution to a problem. For many problems it is hard to combine all requirements and specifications into a single objective measure. It is also often true that these objectives are conflicting; for instance, if our objectives are to buy the fastest and cheapest possible laptop we will soon realize the two objectives are partially conflicting. The intuitive solution is to merely add the different objective values—as a weighted sum—and use this as your fitness under optimization. Doing so, however, has several drawbacks such as the non-trivial ad-hoc design of the weighting among the objectives, the lack of insight on the interactions between the objectives (e.g., what is the price threshold above which faster laptops are not more expensive?) and the fact that a weighted-sum single-objective approach cannot reach solutions that achieve an optimal compromise among their weighted objectives. The response to these limitations is the family of algorithms known as **multiobjective evolutionary algorithms**. A multiobjective evolutionary algorithm considers at least two objective functions—that are partially conflicting—and searches for a **Pareto front** of these objectives. The Pareto front contains solutions that cannot be improved in one objective without worsening in another. Further details about multiobjective optimization by means of evolutionary algorithms can be found in [126]. The approach is applicable in game AI on instances where more than one objective is relevant for the problem we attempt to solve: for instance, we might wish to optimize both the balance and the asymmetry of a strategy game map [712, 713], or design non-player characters that are interestingly diverse in their behavioral space [5].

#### 2.4.2.1 Evolutionary Algorithms for Ms Pac-Man

A simple way to employ evolutionary algorithms (EAs) in Ms Pac-Man is as follows. You could design a utility function based on a number of important parameters Ms Pac-Man must consider for taking the right decision on where to move next. These parameters, for instance, could be the current placement of ghosts, the presence of power pills, the number of pellets available on the level and so on. The next step would be to design a utility function as the weighted sum of these parameters. At each junction, Ms Pac-Man would need to consult its utility function for all its possible moves and pick the move with the highest utility. The weights of the utility function are unknown of course and this is where an EA can be of help by evolving the weights of the utility so that they optimize the score for Ms Pac-Man. In other words, the fitness of each chromosome (weight vector of utility) is determined by the score obtained from Ms Pac-Man within a number of simulation steps, or game levels played.

### 2.4.3 Further Reading

We recommend three books for further reading on evolutionary computation: Eiben and Smith's *Introduction to Evolutionary Computing* [184], Ashlock's *Evolutionary Computation for Modeling and Optimization* [21] and finally, the genetic programming field guide by Poli et al. [536].

## 2.5 Supervised Learning

Supervised learning is the algorithmic process of approximating the underlying function between labeled data and their corresponding attributes or features [49]. A popular example of supervised learning is that of a machine that is asked to distinguish between apples and pears (**labeled data**) given a set of **features** or **data attributes** such as the fruits' color and size. Initially, the machine learns to classify between apples and pears by *seeing* a number of available fruit examples—which contain the color and size of each fruit, on one hand, and their corresponding label (apple or pear) on the other. After learning is complete, the machine should ideally be able to tell whether a new and *unseen* fruit is a pear or an apple based solely on its color and size. Beyond distinguishing between apples and pears supervised learning nowadays is used in a plethora of applications including financial services, medical diagnosis, fraud detection, web page categorization, image and speech recognition, and user modeling (among many).

Evidently, supervised learning requires a set of labeled training examples; hence supervised. More specifically, the training signal comes as a set of supervised labels on the data (e.g., this is an apple whereas that one is a pear) which acts upon a set of characterizations of these labels (e.g., this apple has red color and medium size). Consequently, each data example comes as a pair of a set of labels (or outputs) and features that correspond to these labels (or inputs). The ultimate goal of supervised learning is not to merely learn from the input-output pairs but to derive a function that approximates (better, imitates) their relationship. The derived function should be able to map well to new and *unseen* instances of input and output pairs (e.g., unseen apples and pears in our example), a property that is called **generalization**. Here are some examples of input-output pairs one can meet in games and make supervised learning relevant: {player health, own health, distance to player} $\rightarrow$ {action (shoot, flee, idle)}; {player's previous position, player's current position} $\rightarrow$ {player's next position}; {number of kills and headshots, ammo spent} $\rightarrow$ {skill rating}; {score, map explored, average heart rate} $\rightarrow$ {level of player frustration}; {Ms Pac-Man and ghosts position, pellets available} $\rightarrow$ {Ms Pac-Man direction}.

Formally, supervised learning attempts to derive a function $f : X \rightarrow Y$, given a set of $N$ training examples $\{(\mathbf{x}_1, \mathbf{y}_1), \ldots, (\mathbf{x}_N, \mathbf{y}_N)\}$; where $X$ and $Y$ is the input and output space, respectively; $\mathbf{x}_i$ is the feature (input) vector of the $i$-th example and $\mathbf{y}_i$ is its corresponding set of labels. A supervised learning task has two core steps. In the first **training** step, the training samples—attributes and corresponding labels—

are presented and the function $f$ between attributes and labels is derived. As we will see in the list of algorithms below $f$ can be represented as a number of classification rules, decision trees, or mathematical formulae. In the second **testing** step $f$ can be used to predict the labels of unknown data given their attributes. To validate the generalizability of $f$ and to avoid overfitting to the data [49], it is common practice that $f$ is evaluated on a new independent (test) dataset using a performance measure such as accuracy, which is the percentage of test samples that are correctly predicted by our trained function. If the accuracy is acceptable, we can use $f$ to predict new data samples.

But how do we derive this $f$ function? In general, an algorithmic process modifies the parameters of this function so that we achieve a good match between the given labels of our training samples and the function we attempt to approximate. There are numerous ways to find and represent that function, each one corresponding to a different supervised learning algorithm. These include artificial neural networks, case-based reasoning, decision tree learning, random forests, Gaussian regression, naive Bayes classifiers, k-nearest neighbors, and support vector machines [49]. The variety of supervised learning algorithms available is, in part, explained by the fact that there is no single learning algorithm that works best on all supervised learning problems out there. This is widely known as the *no free lunch theorem* [756].

Before covering the details of particular algorithms we should stress that the data type of the label determines the output type and, in turn, the type of the supervised learning approach that can be applied. We can identify three main types of supervised learning algorithms depending on the data type of the labels (outputs). First, we meet **classification** [49] algorithms which attempt to predict categorical class labels (discrete or nominal) such as the apples and pears of the previous example or the level in which a player will achieve her maximum score. Second, if the output data comes as an interval—such as the completion time of a game level or retention time—the supervised learning task is metric **regression** [49]. Finally, **preference learning** [215] predicts ordinal outputs such as ranks and preferences and attempts to derive the underlying global order that characterizes those ordinal labels. Examples of ordinal outputs include the ranked preferences of variant camera viewpoints, or a preference of a particular sound effect over others. The training signal in the preference learning paradigm provides information about the *relative* relation between instances of the phenomenon we attempt to approximate, whereas regression and classification provide information, respectively, about the *intensity* and the *classes* of the phenomenon.

In this book, we focus on a subset of the most promising and popular supervised learning algorithms for game AI tasks such as game playing (see Chapter 3), player behavior imitation or player preference prediction (see Chapter 5). The three algorithms outlined in the remainder of this section are artificial neural networks, support vector machines and decision tree learning. All three supervised learning algorithms covered can be used for either classification, prediction or preference learning tasks.

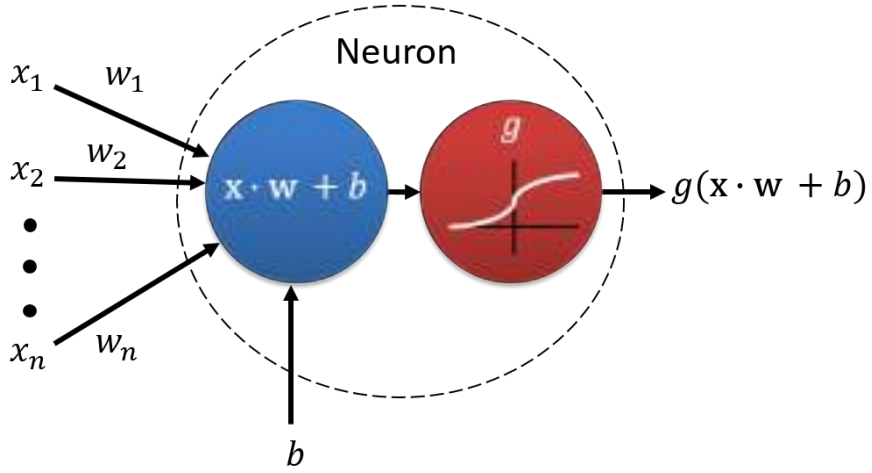


**Fig. 2.12** An illustration of an artificial neuron. The neuron is fed with the input vector $\mathbf{x}$ through $n$ connections with corresponding weight values $\mathbf{w}$. The neuron processes the input by calculating the weighted sum of inputs and corresponding connection weights and adding a bias weight ($b$): $\mathbf{x} \cdot \mathbf{w} + b$. The resulting formula feeds an activation function ($g$), the value of which defines the output of the neuron.

### 2.5.1 Artificial Neural Networks

**Artificial Neural Networks** (ANNs) are a bio-inspired approach for computational intelligence and machine learning. An ANN is a set of interconnected processing units (named **neurons**) which was originally designed to model the way a biological brain—containing over $10^{11}$ neurons—processes information, operates, learns and performs in several tasks. Biological neurons have a cell body, a number of dendrites which bring information into the neuron and an axon which transmits electrochemical information outside the neuron. The artificial neuron (see Fig. 2.12) resembles the biological neuron as it has a number of **inputs** $\mathbf{x}$ (corresponding to the neuron dendrites) each with an associated **weight** parameter $\mathbf{w}$ (corresponding to the synaptic strength). It also has a processing unit that combines inputs with their corresponding weights via an inner product (weighted sum) and adds a **bias** (or threshold) weight $b$ to the weighted sum as follows: $\mathbf{x} \cdot \mathbf{w} + b$. This value is then fed to an **activation function** $g$ (cell body) that yields the **output** of the neuron (corresponding to an axon terminal). ANNs are essentially simple mathematical models defining a function $f : \mathbf{x} \rightarrow \mathbf{y}$.

Various forms of ANNs are applicable for regression analysis, classification, and preference learning, and even unsupervised learning (via e.g., Hebbian learning [256] and self-organizing maps [347]). Core application areas include pattern recognition, robot and agent control, game-playing, decision making, gesture, speech and text recognition, medical and financial applications, affective modeling, and image recognition. The benefits of ANNs compared to other supervised learning ap-

proaches is their capacity to approximate any continuous real-valued function given sufficiently large ANN architectures and computational resources [348, 152]. This capacity characterizes ANNs as *universal approximators* [279].

#### 2.5.1.1 Activation Functions

Which activation function should one use in an ANN? The original model of a neuron by McCulloch and Pitts [450] in 1943 featured a Heaviside step activation function which either allows the neuron to *fire* or not. When such neurons are employed and connected to a multi-layered ANN the resulting network can merely solve linearly separable problems. The algorithm that trains such ANNs was invented in 1958 [576] and is known as the Rosenblatt's perceptron algorithm. Non-linearly separable problems such as the exclusive-or gate could only be solved after the invention of the **backpropagation** algorithm in 1975 [752]. Nowadays, there are several activation functions used in conjunction with ANNs and their training. The use of the activation function, in turn, yields different types of ANNs. Examples include Gaussian activation function that is used in radial basis function (RBF) networks [71] and the numerous types of activation functions that can be used in the compositional pattern producing networks (CPPNs) [653]. The most common function used for ANN training is the sigmoid-shaped logistic function ($g(x) = 1/(1+e^{-x})$) for the following properties: 1) it is bounded, monotonic and non-linear; 2) it is continuous and smooth and 3) its derivative is calculated trivially as $g'(x) = g(x)(1-g(x))$. Given the properties above the logistic function can be used in conjunction with gradient-based optimization algorithms such as backpropagation which is described below. Other popular activation functions for training **deep** architectures of neural networks include the **rectifier**—named **rectified linear unit** (ReLU) when employed to a neuron—and its smooth approximation, the **softplus** function [231]. Compared to sigmoid-shaped activation functions, ReLUs allow for faster and (empirically) more effective training of deep ANNs, which are generally trained on large datasets (see more in Section 2.5.1.6).

#### 2.5.1.2 From a Neuron to a Network

To form an ANN a number of neurons need to be structured and connected. While numerous ways have been proposed in the literature the most common of them all is to structure neurons in layers. In its simplest form, known as the **multi-layer perceptron** (MLP), neurons in an ANN are layered across one or more layers but not connected to other neurons in the same layer (see Fig. 2.13 for a typical MLP structure). The output of each neuron in each layer is connected to all the neurons in the next layer. Note that a neuron's output value feeds merely the neurons of the next layer and, thereby, becomes their input. Consequently, the outputs of the neurons in the last layer are the outputs of the ANN. The last layer of the ANN is also known as the **output layer** whereas all intermediate layers between the output

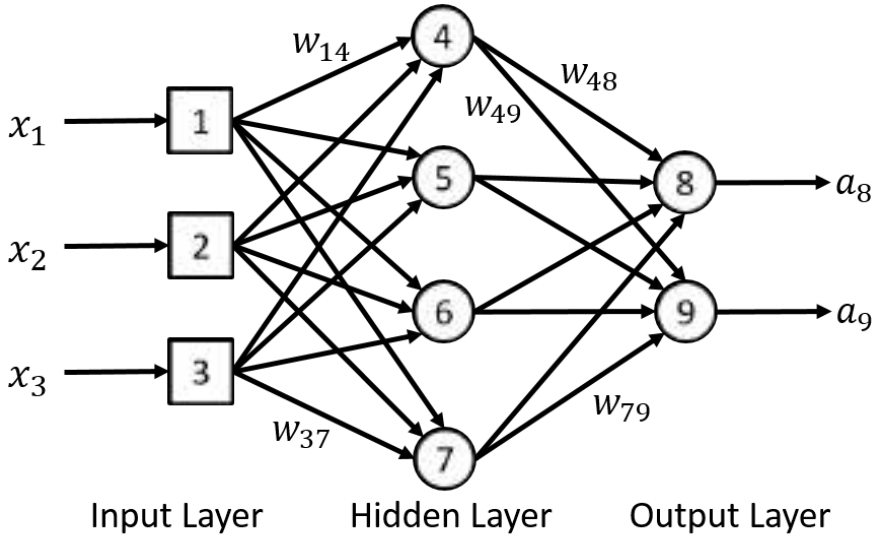


**Fig. 2.13** An MLP example with three inputs, one hidden layer containing four hidden neurons and two outputs. The ANN has labeled and ordered neurons and example connection weight labels. Bias weights $b_j$ are not illustrated in this example but are connected to each neuron $j$ of the ANN.

and the input are the **hidden layers**. It is important to note that the inputs of the ANN, **x**, are connected to all the neurons of the first hidden layer. We illustrate this with an additional layer we call the **input layer**. The input layer does not contain neurons as it only distributes the inputs to the first layer of neurons. In summary, MLPs are 1) *layered* because they are grouped in layers; 2) *feed-forward* because their connections are unidirectional and always forward (from a previous layer to the next); and 3) *fully connected* because every neuron is connected to all neurons of the next layer.

#### 2.5.1.3 Forward Operation

In the previous section we defined the core components of an ANN whereas in this section we will see how we compute the output of the ANN when an input pattern is presented. The process is called **forward operation** and propagates the inputs of the ANN throughout its consecutive layers to yield the outputs. The basic steps of the forward operation are as follows:

1. Label and order neurons. We typically start numbering at the input layer and increment the numbers towards the output layer (see Fig. 2.13). Note that

the input layer does not contain neurons, nevertheless is treated as such for numbering purposes only.
2. Label connection weights assuming that $w_{ij}$ is the connection weight from neuron $i$ (pre-synaptic neuron) to neuron $j$ (post-synaptic neuron). Label bias weights that connect to neuron $j$ as $b_j$.
3. Present an input pattern $\mathbf{x}$.
4. For each neuron $j$ compute its output as follows: $\alpha_j = g(\sum_i \{w_{ij}\alpha_i\} + b_j)$, where $\alpha_j$ and $\alpha_i$ are, respectively, the output of and the inputs to neuron $j$ (n.b. $\alpha_i = x_i$ in the input layer); $g$ is the activation function (usually the logistic sigmoid function).
5. The outputs of the neurons of the output layer are the outputs of the ANN.

#### 2.5.1.4 How Does an ANN Learn?

How do we approximate $f(\mathbf{x};\mathbf{w},\mathbf{b})$ so that the outputs of the ANN match the desired outputs (labels) of our dataset, $\mathbf{y}$? We will need a training algorithm that adjusts the weights ($\mathbf{w}$ and $\mathbf{b}$) so that $f : \mathbf{x} \rightarrow \mathbf{y}$. A training algorithm as such requires two components. First, it requires a cost function to evaluate the quality of any set of weights. Second, it requires a search strategy within the space of possible solutions (i.e., the weight space). We outline these aspects in the following two subsections.

##### Cost (Error) Function

Before we attempt to adjust the weights to approximate $f$, we need some measure of MLP performance. The most common performance measure for training ANNs in a supervised manner is the squared Euclidean distance (error) between the vectors of the actual output of the ANN ($\alpha$) and the desired labeled output $y$ (see equation 2.2).

$$E = \frac{1}{2}\sum_j (y_j - \alpha_j)^2 \tag{2.2}$$

where the sum is taken over all the output neurons (the neurons in the final layer). Note that the $y_j$ labels are constant values and more importantly, also note that $E$ is a function of all the weights of the ANN since the actual outputs depend on them. As we will see below, ANN training algorithms build strongly upon this relationship between error and weights.

### Backpropagation

The **backpropagation** (or backprop) [579] algorithm is based on gradient descent optimization and is arguably the most common algorithm for training ANNs. Backpropagation stands for *backward propagation of errors* as it calculates weight updates that minimize the error function—that we defined earlier (2.2)—from the output to the input layer. In a nutshell, backpropagation computes the partial derivative (gradient) of the error function $E$ with respect to each weight of the ANN and adjusts the weights of the ANN following the (opposite direction of the) gradient that minimizes $E$.

As mentioned earlier, the squared Euclidean error of (2.2) depends on the weights as the ANN output which is essentially the $f(\mathbf{x};\mathbf{w},\mathbf{b})$ function. As such we can calculate the gradient of $E$ with respect to any weight ($\frac{\theta E}{\theta w_{ij}}$) and any bias weight ($\frac{\theta E}{\theta b_j}$) in the ANN, which in turn will determine the degree to which the error will change if we change the weight values. We can then determine how much of such change we desire through a parameter $\eta \in [0,1]$ called **learning rate**. In the absence of any information about the general shape of the function between the error and the weights but the existence of information about its gradient it appears that a **gradient descent** approach would seem to be a good fit for attempting to find the global minimum of the $E$ function. Given the lack of information about the $E$ function, the search can start from some random point in the weight space (i.e., random initial weight values) and follow the gradient towards lower $E$ values. This process is repeated iteratively until we reach $E$ values we are happy with or we run out of computational resources.

More formally, the basic steps of the **backpropagation** algorithm are as follows:

1. Initialize **w** and **b** to random (commonly small) values.
2. For each training pattern (input-output pair):
   (a) Present input pattern **x**, ideally normalized to a range (e.g., $[0,1]$).
   (b) Compute ANN actual outputs $\alpha_j$ using the forward operation.
   (c) Compute $E$ according to (2.2).
   (d) Compute error derivatives with respect to each weight $\frac{\theta E}{\theta w_{ij}}$ and bias weight $\frac{\theta E}{\theta b_j}$ of the ANN from the output all the way to the input layer.
   (e) Update weights and bias weights as $\Delta w_{ij} = -\eta \frac{\theta E}{\theta w_{ij}}$ and $\Delta b_j - \eta \frac{\theta E}{\theta b_j}$, respectively.
3. If $E$ is *small* or you are out of *computational budget*, stop! Otherwise go to step 2.

Note that we do not wish to detail the derivate calculations of step 2(d) as doing so would be out of scope for this book. We instead refer the interested reader to the original backpropagation paper [579] for the exact formulas and to the reading list at the end of this section.

### Limitations and Solutions

It is worth noting that backpropagation is not guaranteed to find the global minimum of $E$ given its local search (hill-climbing) property. Further, given its gradient-based (local) search nature, the algorithm fails to overcome potential plateaux areas in the error function landscape. As these are areas with near-zero gradient, crossing them results in near-zero weight updates and further in **premature convergence** of the algorithm. Typical solutions and enhancements of the algorithm to overcome convergence to local minima include:

- **Random restarts**: One can rerun the algorithm with new random connection weight values in the hope that the ANN is not too dependent on luck. No ANN model is good if it depends too much on luck—for instance, if it performs well only in one or two out of ten runs.
- **Dynamic learning rate**: One can either modify the learning rate parameter and observe changes in the performance of the ANN or introduce a dynamic learning rate parameter that increases when convergence is slow whereas it decreases when convergence to lower $E$ values is fast.
- **Momentum**: Alternatively, one may add a momentum amount to the weight update rule as follows:

$$\Delta w_{ij}^{(t)} = m\Delta w_{ij}^{(t-1)} - \eta \frac{\theta E}{\theta w_{ij}} \tag{2.3}$$

  where $m \in [0,1]$ is the momentum parameter and $t$ is the iteration step of the weight update. The addition of a momentum value of the previous weight update ($a\Delta w_{ij}^{(t-1)}$) attempts to help backpropagation to overcome a potential local minimum.

While the above solutions are directly applicable to ANNs of small size, practical wisdom and empirical evidence with modern (deep) ANN architectures, however, suggests that the above drawbacks are largely eliminated [366].

### Batch vs. Non-batch Training

Backpropagation can be employed following a batch or a non-batch learning mode. In **non-batch** mode, weights are updated every time a training sample is presented to the ANN. In **batch mode**, weights are updated after all training samples are presented to the ANN. In that case, errors are accumulated over the samples of the batch prior to the weight update. The non-batch mode is more unstable as it iteratively relies on a single data point; however, this might be beneficial for avoiding a convergence to a local minimum. The batch mode, on the other hand, is naturally a more stable gradient descent approach as weight updates are driven by the average error of all training samples in the batch. To best utilize the advantages of both approaches it is common to apply batch learning of randomly selected samples in small batch sizes.

#### 2.5.1.5 Types of ANNs

Beyond the standard feedforward MLP there are numerous other types of ANN used for classification, regression, preference learning, data processing and filtering, and clustering tasks. Notably, **recurrent** neural networks (such as Hopfield networks [278], Boltzmann machines [4] and Long Short-Term Memory [266]) allow connections between neurons to form directed cycles, thus enabling an ANN to capture dynamic and temporal phenomena (e.g., time-series processing and prediction). Further, there are ANN types mostly used for clustering and data dimensionality reduction such as Kohonen self-organizing maps [347] and Autoencoders [41].

#### 2.5.1.6 From Shallow to Deep

A critical parameter for ANN training is the size of the ANN. So, how *wide* and *deep* should my ANN architecture be to perform well on this particular task? While there is no formal and definite answer to this question, there is a generally accepted rule-of-thumb suggesting that the size of the network should match the complexity of the problem. According to Goodfellow et al. in their deep learning book [231] an MLP is essentially a deep (feedforward) neural network. Its *depth* is determined by the number of hidden layers it contains. Goodfellow et al. state that "It is from this terminology that the name **deep learning** arises". On that basis, training of ANN architectures containing (at least) a hidden layer can be viewed as a deep learning task whereas single output-layered architectures can be viewed as *shallow*. Various methods have been introduced in recent years to enable training of deep architectures containing several layers. The methods largely rely on gradient search and are covered in detail in [231] for the interested reader.

#### 2.5.1.7 ANNs for Ms Pac-Man

As with every other method in this chapter we will attempt to employ ANNs in the Ms Pac-Man game. One straightforward way to use ANNs in Ms Pac-Man is to attempt to imitate expert players of the game. Thus, one can ask experts to play the game and record their playthroughs, through which a number of features can be extracted and used as the input of the ANN. The resolution of the ANN input may vary from simple statistics of the game—such as the average distance between ghosts and Ms Pac-Man—to detailed pixel-to-pixel RGB values of the game level image. The output data, on the other hand, may contain the actions selected by Ms Pac-Man in each frame of the game. Given the input and desired output pairs, the ANN is trained via backpropagation to predict the action performed by expert players (ANN output) given the current game state (ANN input). The size (width and depth) of the ANN depends on both the amount of data available from the expert Ms Pac-Man players and the size of the input vector considered.

### 2.5.2 Support Vector Machines

**Support vector machines** (SVMs) [139] are an alternative and very popular set of supervised learning algorithms that can be used for classification, regression [179] and preference learning [302] tasks. A support vector machine is a binary linear classifier that is trained so as to maximize the margin between the training examples of the separate classes in the data (e.g., apples and pears). As with every other supervised learning algorithm, the attributes of new and unseen examples are seeding the SVM which predicts the class they belong to. SVMs have been used widely for text categorization, speech recognition, image classification, and hand-written character recognition among many other areas.

Similarly to ANNs, SVMs construct a hyperplane that divides the input space and represents the function $f$ that maps between the input and the target outputs. Instead of implicitly attempting to minimize the difference between the model's actual output and the target output following the gradient of the error (as backpropagation does), SVMs construct a hyperplane that maintains the largest distance to the nearest training-data point of any other class. That distance is called a **maximum-margin** and its corresponding hyperplane divides the points ($\mathbf{x_i}$) of class with label ($y_i$) 1 from those with label $-1$ in a dataset of $n$ samples in total. In other words, the distance between the derived hyperplane and the nearest point $\mathbf{x}_i$ from either class is maximized. Given the input attributes of a training dataset, $\mathbf{x}$, the general form of a hyperplane can be defined as: $\mathbf{w} \cdot \mathbf{x} - b = 0$ where, as in backpropagation training, $\mathbf{w}$ is the weight (normal) vector of the hyperplane and $\frac{b}{\|\mathbf{w}\|}$ determines the offset (or weight threshold/bias) of the hyperplane from the origin (see Fig. 2.14). Thus, formally put, an SVM is a function $f(\mathbf{x}; \mathbf{w}, b)$ that predicts target outputs ($\mathbf{y}$) and attempts to

$$\textbf{minimize } \|\mathbf{w}\|, \tag{2.4}$$

$$\textbf{subject to } y_i(\mathbf{w} \cdot \mathbf{x}_i - b) \geq 1, \text{for } i = 1, \ldots, n \tag{2.5}$$

The weights $\mathbf{w}$ and $b$ determine the SVM classifier. The $\mathbf{x}_i$ vectors that lie nearest to the derived hyperplane are called **support vectors**. The above problem is solvable if the training data is linearly separable (also known as a **hard-margin** classification task; see Fig. 2.14). If the data is not linearly separable (**soft-margin**) the SVM instead attempts to

$$\textbf{minimize } \left[ \frac{1}{n} \sum_{i=1}^{n} \max(0, 1 - y_i(\mathbf{w} \cdot \mathbf{x}_i - b)) \right] + \lambda ||\mathbf{w}||^2 \tag{2.6}$$

which equals $\lambda ||\mathbf{w}||^2$ if the hard constraints of equation 2.5 are satisfied—i.e., if all data points are correctly classified on the right side of the margin. The value of equation (2.6) is proportional to the distance from the margin for misclassified data and $\lambda$ is designed so as to qualitatively determine the degree to which the margin-size should be increased versus ensuring that the $\mathbf{x}_i$ will lie on the correct side of the

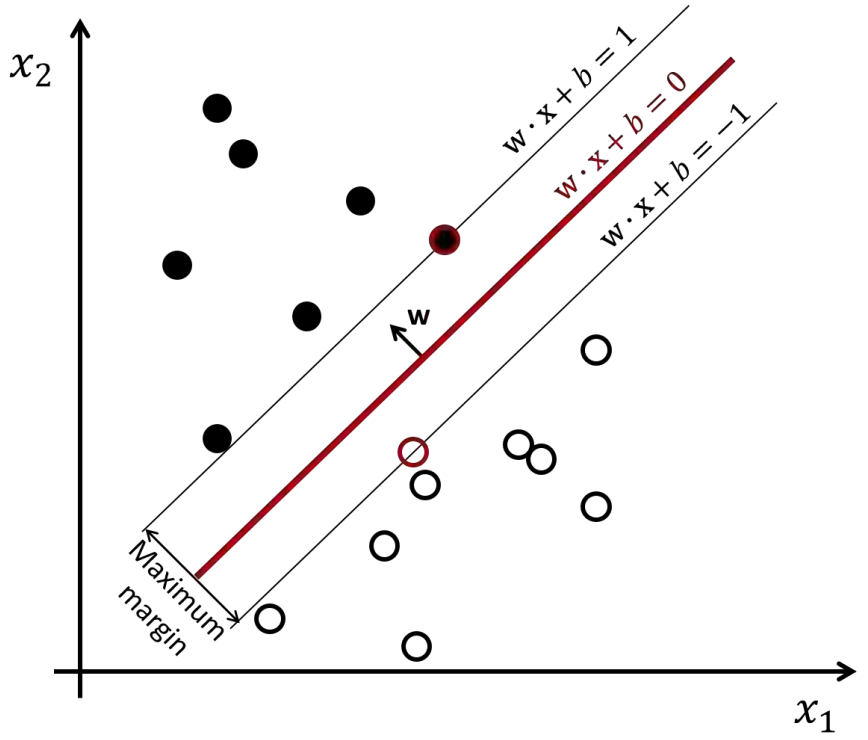


**Fig. 2.14** An example of a maximum-margin hyperplane (red thick line) and margins (black lines) for an SVM which is trained on data samples from two classes. Solid and empty circles correspond to data with labels 1 and $-1$, respectively. The classification is mapped onto a two-dimensional input vector $(x_1, x_2)$ in this example. The two data samples on the margin—the circles depicted with red outline—are the support vectors.

margin. Evidently, if we choose a small value for $\lambda$ we approximate the hard-margin classifier for linearly separable data.

The standard approach for training soft-margin classifiers is to treat the learning task as a quadratic programming problem and search the space of $\mathbf{w}$ and $b$ to find the widest possible margin that matches all data points. Other approaches include sub-gradient descent and coordinate descent.

In addition to linear classification tasks, SVMs can support non-linear classification by employing a number of different non-linear **kernels** which map the input space onto higher-dimensional feature spaces. The SVM task remains similar, except that every dot product is replaced by a nonlinear kernel function. This allows the algorithm to fit the maximum-margin hyperplane in a transformed feature space. Popular kernels used in conjunction with SVMs include polynomial functions, Gaussian radial basis functions or hyperbolic tangent functions.

While SVMs were originally designed to tackle **binary** classification problems there exist several SVM variants that can tackle multi-class classification [284], regression [179] and preference learning [302] that the interested reader can refer to.

SVMs have a number of advantages compared to other supervised learning approaches. They are efficient in finding solutions when dealing with large, yet sparse, datasets as they only depend on support vectors to construct hyperplanes. They also handle well large feature spaces as the learning task complexity does not depend on the dimensionality of the feature space. SVMs feature a simple convex optimization problem which can be guaranteed to converge to a single global solution. Finally, overfitting can be controlled easily through the soft margin classification approach.

#### 2.5.2.1 SVMs for Ms Pac-Man

Similarly to ANNs, SVMs can be used for imitating the behavior of Ms Pac-Man expert players. The considerations about the feature (input) space and the action (output) space remain the same. In addition to the design of the input and output vectors, the size and quality of the data obtained from expert players will determine the performance of the SVM controlling Ms Pac-Man towards maximizing its score.

### *2.5.3 Decision Tree Learning*

In **decision tree learning** [67], the function $f$ we attempt to derive uses a decision tree representation which maps attributes of data observations to their target values. The former (inputs) are represented as the nodes and the latter (outputs) are represented as the leaves of the tree. The possible values of each node (input) are represented by the various branches of that node. As with the other supervised learning algorithms, decision trees can be classified depending on the output data type they attempt to learn. In particular, decision trees can be distinguished into classification, regression and rank trees if, respectively, the target output is a finite set of values, a set of continuous (interval) values, or a set of ordinal relations among observations.

An example of a decision tree is illustrated in Fig. 2.15. Tree nodes correspond to input attributes; there are branches to children for each of the possible values of each input attribute. Further leaves represent values of the output—car type in this example—given the values of the input attributes as determined by the path from the root to the leaf.

The goal of decision tree learning is to construct a mapping (a tree model) that predicts the value of target outputs based on a number of input attributes. The basic and most common approach for learning decision trees from data follows a top-down **recursive** tree induction strategy which has the characteristics of a greedy process. The algorithm assumes that both the input attributes and the target outputs have finite discrete domains and are of categorical nature. If inputs or outputs are continuous values, they can be discretized prior to constructing the tree. A tree is

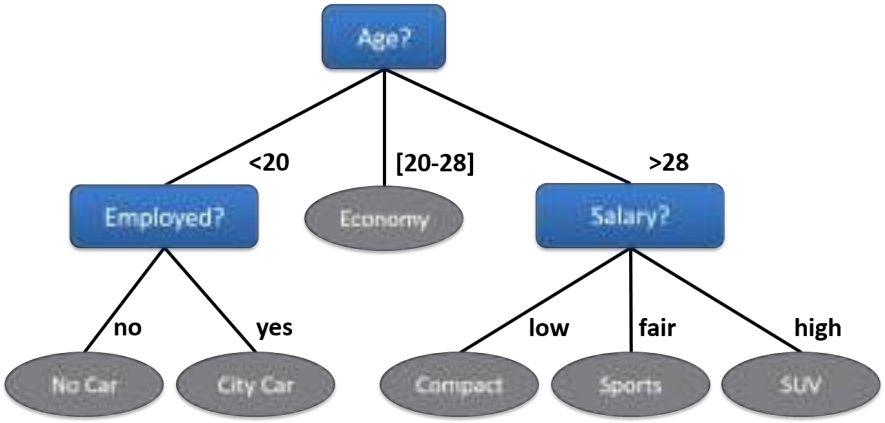


**Fig. 2.15** A decision tree example: Given *age*, *employment* status and *salary* (data attributes) the tree predicts the *type of car* (target value) a person owns. Tree nodes (blue rounded rectangles) represent data attributes, or inputs, whereas leaves (gray ovals) represent target values, or outputs. Tree branches represent possible values of the corresponding parent node of the tree.

.

gradually constructed by splitting the available training dataset into subsets based on selections made for the attributes of the dataset. This process is repeated on a attribute-per-attribute basis in a recursive manner.

There are several variants of the above process that lead to dissimilar decision-tree algorithms. The two most notable variants of decision tree learning, however, are the **Iterative Dichotomiser 3 (ID3)** [544] and its successor **C4.5** [545]. The basic tree learning algorithm has the following general steps:

1. At start, all the training examples are at the root of the tree.
2. Select an attribute on the basis of a heuristic and pick the attribute with the maximum heuristic value. The two most popular heuristics are as follows:
   - **Information gain**: This heuristic is used by both the ID3 and the C4.5 tree-generation algorithms. Information gain $G(A)$ is based on the concept of entropy from information theory and measures the difference in entropy $H$ from before to after the dataset $D$ is split on an attribute $A$.

     $$G(A) = H(D) - H_A(D) \tag{2.7}$$

     where $H(D)$ is the entropy of $D$ ($H(D) = -\sum_i^m p_i \log_2(p_i)$); $p_i$ is the probability that an arbitrary sample in $D$ belongs to class $i$; $m$ is the total number of classes; $H_A(D)$ is the information needed (after using attribute $A$ to split $D$ into $v$ partitions) to classify D and is calculated as $H_A(D) = -\sum_j^v (|D_j|/|D|)H(D_j)$ with $|x|$ being the size of $x$.

- **Gain ratio**: The C4.5 algorithm uses the gain ratio heuristic to reduce the bias of information gain towards attributes with a large number of values. The gain ratio normalizes information gain by taking into account the number and size of branches when choosing an attribute. The information gain ratio is the ratio between the information gain and the intrinsic value $IV_A$ of attribute $A$:

$$GR(A) = G(A)/IV_A(D) \tag{2.8}$$

where

$$IV_A(D) = -\sum_{j}^{v} \frac{|D_j|}{|D|} \log_2(\frac{|D_j|}{|D|}) \tag{2.9}$$

3. Based on the selected attribute from step 2, construct a new node of the tree and split the dataset into subsets according to the possible values of the selected attribute. The possible values of the attribute become the branches of the node.
4. Repeat steps 2 and 3 until one of the following occurs:
   - All samples for a given node belong to the same class.
   - There are no remaining attributes for further partitioning.
   - There are no data samples left.

#### 2.5.3.1 Decision Trees for Ms Pac-Man

As with ANNs and SVMs, decision tree learning requires data to be trained on. Presuming that data from expert Ms Pac-Man players would be of good quality and quantity, decision trees can be constructed to predict the strategy of Ms Pac-Man based on a number of ad-hoc designed attributes of the game state. Figure 2.16 illustrates a simplified hypothetical decision tree for controlling Ms Pac-Man. According to that example if a ghost is nearby then Ms Pac-Man checks if power pills are available in a close distance and aims for those; otherwise it takes actions so that it evades the ghost. If alternatively, ghosts are not visible Ms Pac-Man checks for pellets. If those are nearby or in a fair distance then it aims for them; otherwise it aims for the fruit, if that is available on the level. It is important to note that the leaves of the tree in our example represent control strategies (macro-actions) rather than actual actions (up, down, left, right) for Ms Pac-Man.

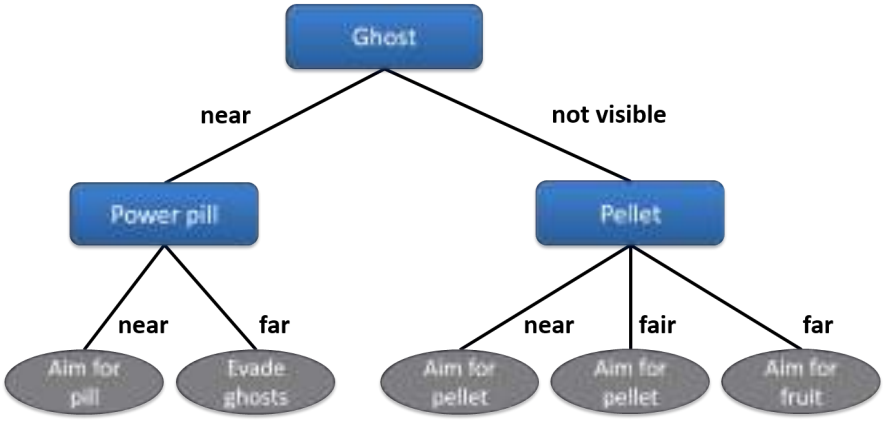


**Fig. 2.16** A decision tree example for controlling Ms Pac-Man. The tree is trained on data from expert Ms Pac-Man players. Given the distance from the nearest *ghost*, *power pill* and *pellet* (data attributes) the tree predicts the *strategy* Ms Pac-Man needs to follow.

### *2.5.4 Further Reading*

The core supervised learning algorithms are covered in detail in the Russell and Norvig classic AI textbook [582] including decision tree learning (Chapter 18) and artificial neural networks (Chapter 19). Detailed descriptions of artificial neural networks and backpropagation can also be found in the book of Haykin [253]. Deep architectures of ANNs are covered in great detail in the deep learning book by Goodfellow et al. [231]. Finally, support vector machines are covered in the tutorial paper of Burges [86].

The preference learning version of backpropagation in shallow and deep architectures can be found in [430, 436] whereas RankSVM is covered in the original paper of Joachims [303].

## 2.6 Reinforcement Learning

**Reinforcement Learning** (RL) [672] is a machine learning approach inspired by behaviorist psychology and, in particular, the way humans and animals learn to take decisions via (positive or negative) rewards received by their environment. In reinforcement learning, samples of good behavior are usually not available (as in supervised learning); instead, similarly to evolutionary (reinforcement) learning, the training signal of the algorithm is provided by the environment based on how an agent is interacting with it. At a particular point in time $t$, the agent is on a particular **state** $s$ and decides to take an **action** $a$ from all the available actions in its current state. As a response the environment delivers an immediate **reward**, $r$. Through

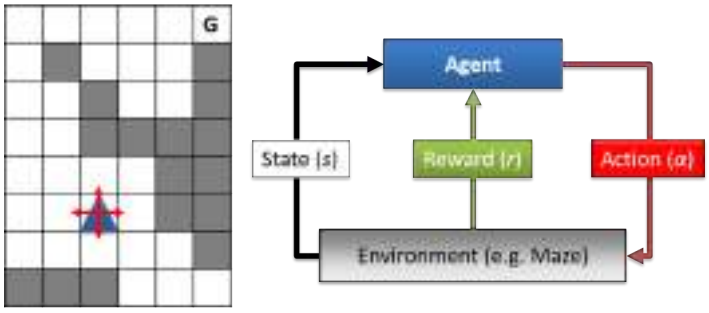


**Fig. 2.17** A reinforcement learning example. The agent (triangle) attempts to reach the goal (G) by taking an action ($a$) among all available actions in its current state ($s$). The agent receives an immediate reward ($r$) and the environment notifies the agent about its new state after taking the action.

.

the continuous interaction between the agent and its environment, the agent gradually learns to select actions that maximize its sum of rewards. RL has been studied from a variety of disciplinary perspectives including operations research, game theory, information theory, and genetic algorithms and has been successfully applied in problems which involve a balance between long-term and short-term rewards such as robot control and games [464, 629]. An example of the reinforcement problem is illustrated through a maze navigation task in Fig. 2.17.

More formally, the aim of the agent is to discover a **policy** ($\pi$) for selecting actions that maximize a measure of a long-term reward such as the expected cumulative reward. A policy is a strategy that the agent follows in selecting actions, given the state it is in. If the function that characterizes the value of each action either exists or is learned, the optimal policy ($\pi^*$) can be derived by selecting the action with the highest value. The interactions with the environment occur in discrete time steps ($t = \{0, 1, 2, \ldots\}$) and are modeled as a **Markov decision process** (MDP). The MDP is defined by

- $S$: A set of states $\{s_1, ..., s_n\} \in S$. The environment states are a function of the agent's information about the environment (i.e., the agent's inputs).
- $A$: A set of actions $\{a_1, ..., a_m\} \in A$ possible in each state $s$. The actions represent the different ways the agent can act in the environment.
- $P(s, s', a)$: The probability of transition from $s$ to $s'$ given $a$. $P$ gives the probability of ending in state $s'$ after picking action $a$ in state $s$ and it follows the **Markov property** implying that future states of the process depend only upon the present state, not on the sequence of events that preceded it. As a result, the Markov property of $P$ makes predictions of 1-step dynamics possible.
- $R(s, s', a)$: The reward function on transition from $s$ to $s'$ given $a$. When the agent in state $s$ picks an action $a$ and moves to state $s'$, it receives an immediate reward $r$ from the environment.

$P$ and $R$ define the **world model** and represent, respectively, the environment's dynamics ($P$) and the long-term reward ($R$) for each policy. If the world model is *known* there is no need to learn to estimate the transition probability and reward function and we thus directly calculate the optimal strategy (policy) using **model-based** approaches such as dynamic programming [44]. If, instead, the world model is *unknown* we approximate the transition and the reward functions by learning estimates of future rewards given by picking action $a$ in state $s$. We then calculate our policy based on these estimates. Learning occurs via **model-free** methods such as Monte Carlo search and **temporal difference learning** [672]. In this section we put an emphasis on the latter set of algorithms and in particular, we focus on the most popular algorithm of TD learning: Q-learning. Before delving into the details of the Q-learning algorithm, we first discuss a few core RL concepts and provide a high-level taxonomy of RL algorithms according to RL problems and tools used for tackling them. We will use this taxonomy to place Q-learning with respect to RL as a whole.

### *2.6.1 Core Concepts and a High-Level Taxonomy*

A central question in RL problems is the right balance between the **exploitation** of current learned knowledge versus the **exploration** of new unseen territories in the search space. Both randomly selecting actions (no exploitation) and always greedily selecting the best action according to a measure of performance or reward (no exploration) are strategies that generally yield poor results in stochastic environments. While several approaches have been proposed in the literature to address the exploration-exploitation balance issue, a popular and rather efficient mechanism for RL action selection is called $\varepsilon$-greedy, determined by the $\varepsilon \in [0,1]$ parameter. According to $\varepsilon$-greedy the RL agent chooses the action it believes will return the highest future reward with probability $1-\varepsilon$; otherwise, it chooses an action uniformly at random.

RL problems can be classified into **episodic** versus **incremental**. In the former class, algorithm training occurs offline and within a finite horizon of multiple training instances. The finite sequence of states, actions and reward signals received within that horizon is called an **episode**. Monte Carlo methods that rely on repeated random sampling, for instance, are a typical example of episodic RL. In the latter class of algorithms, instead, learning occurs online and it is not bounded by an horizon. We meet TD learning under incremental RL algorithms.

Another distinction is between **off-policy** and **on-policy** RL algorithms. An off-policy learner approximates the optimal policy independently of the agent's actions. As we will see below, Q-learning is an off-policy learner since it estimates the return for state-action pairs assuming that a greedy policy is followed. An on-policy RL algorithm instead approximates the policy as a process being tied to the agent's actions including the exploration steps.

**Bootstrapping** is a central notion within RL that classifies algorithms based on the way they optimize state values. Bootstrapping estimates how good a state is based on how good we think the next state is. In other words, with bootstrapping we update an estimate based on another estimate. Both TD learning and dynamic programming use bootstrapping to learn from the experience of visiting states and updating their values. Monte Carlo search methods instead do not use bootstrapping and thus learn each state value separately.

Finally, the notion of **backup** is central in RL and acts as a distinctive feature among RL algorithms. With backup we go backwards from a state in the future, $s_{t+h}$, to the (current) state we want to evaluate, $s_t$, and consider the in-between state values in our estimates. The backup operation has two main properties: its **depth**—which varies from one step backwards to a full backup—and its **breadth**—which varies from a (randomly) selected number of sample states within each time step to a full-breadth backup.

Based on the above criteria we can identify three major RL algorithm types:

1. **Dynamic programming**. In dynamic programming knowledge of the world model ($P$ and $R$) is required and the optimal policy is calculated via bootstrapping.
2. **Monte Carlo methods**. Knowledge of the world model is not required for Monte Carlo methods. Algorithms of this class (e.g., MCTS) are ideal for offline (episodic) training and they learn via sample-breadth and full-depth backup. Monte Carlo methods do not use bootstrapping, however.
3. **TD learning**. As with Monte Carlo methods knowledge of the world model is not required and it is thus estimated. Algorithms of this type (e.g., Q-learning) learn from experience via bootstrapping and variants of backup.

In the following section we cover the most popular TD learning algorithm in the RL literature with the widest use in game AI research.

### 2.6.2 Q-Learning

**Q-learning** [748] is a model-free, off-policy, TD learning algorithm that relies on a tabular representation of $Q(s,a)$ values (hence its name). Informally, $Q(s,a)$ represents how good it is to pick action $a$ in state $s$. Formally, $Q(s,a)$ is the expected discounted reinforcement of taking action $a$ in state $s$. The Q-learning agent learns from experience by picking actions and receiving rewards via bootstrapping.

The goal of the Q-learning agent is to maximize its expected reward by picking the right action at each state. The reward, in particular, is a weighted sum of the expected values of the discounted future rewards. The Q-learning algorithm is a simple update on the Q values in an iterative fashion. Initially, the Q table has arbitrary values as set by the designer. Then each time the agent selects an action

$a$ from state $s$, it visits state $s'$, it receives an immediate reward $r$, and updates its $Q(s,a)$ value as follows:

$$Q(s,a) \leftarrow Q(s,a) + \alpha\{r + \gamma \max_{a'} Q(s',a') - Q(s,a)\} \tag{2.10}$$

where $\alpha \in [0,1]$ is the **learning rate** and $\gamma \in [0,1]$ is the **discount factor**. The learning rate determines the extent to which the new estimate for Q will override the old estimate. The discount factor weights the importance of earlier versus later rewards; the closer $\gamma$ is to 1, the greater the weight is given to future reinforcements. As seen from equation (2.10), the algorithm uses bootstrapping since it maintains estimates of how good a state-action pair is (i.e., $Q(s,a)$) based on how good it thinks the next state is (i.e., $Q(s',a')$). It also uses a one-step-depth, full-breadth backup to estimate Q by taking into consideration all Q values of all possible actions $a'$ of the newly visited state $s'$. It is proven that by using the learning rule of equation (2.10) the $Q(s,a)$ values converge to the expected future discounted reward [748]. The optimal policy can then be calculated based on the Q-values; the agent in state $s$ selects the action $a$ with the highest $Q(s,a)$ value. In summary, the basic steps of the algorithm are as follows:

Given an immediate reward function $r$ and a table of $Q(s,a)$ values for all possible actions in each state:

1. Initialize the table with arbitrary $Q$ values; e.g., $Q(s,a) = 0$.
2. $s \leftarrow$ Start state.
3. While not finished* do:
   (a) Choose an action $a$ based on policy derived from Q (e.g., $\varepsilon$-greedy).
   (b) Apply the action, transit to state $s'$, and receive an immediate reward $r$.
   (c) Update the value of $Q(s,a)$ as per (2.10).
   (d) $s \leftarrow s'$.

*The most commonly used termination conditions are the algorithm's *speed*—i.e., stop within a number of iterations—or the *quality* of convergence—i.e., stop if you are satisfied with the obtained policy.

#### 2.6.2.1 Limitations of Q-Learning

Q-learning has a number of limitations associated primarily with its tabular representation. First of all, depending on the chosen state-action representation the size of the state-action space might be computationally very expensive to handle. As the Q table size grows our computational needs for memory allocation and information retrieval increase. Further, we may experience very long convergence since learning time is exponential to the size of the state-action space. To overcome these

obstacles and get decent performance from RL learners we need to devise a way of reducing the state-action space. Section 2.8 outlines the approach of using artificial neural networks as Q-value function approximators, directly bypassing the Q-table limitation and yielding compressed representations for our RL learner.

#### 2.6.2.2 Q-Learning for Ms Pac-Man

Q-learning is applicable for controlling Ms Pac-Man as long as we define a suitable state-action space and we design an appropriate reward function. A state in Ms Pac-Man could be represented directly as the current snapsnot of the game—i.e., where Ms Pac-Man and ghosts are and which pellets and power pills are still available. That representation, however, yields a prohibitive number of game states for a Q-table to be constructed and processed. Instead, it might be preferred to choose a more indirect representation such as whether ghosts and pellets are nearby or not. Possible actions for Ms Pac-Man could be that it either keeps its current direction, it turns backward, it turns left, or it turns right. Finally, the reward function can be designed to reward Ms Pac-Man positively when it eats a pellet, a ghost or a power pill, whereas it could penalize Ms Pac-Man when it dies.

It is important to note that both Pac-Man and Ms Pac-Man follow the Markov property in the sense that any future game states may depend only upon the present game state. There is one core difference however: while the transition probability in Pac-Man is known given its deterministic nature, it is largely unknown in Ms Pac-Man given the stochastic behavior of the ghosts in that game. Thereby, Pac-Man can theoretically be solved via model-based approaches (e.g., dynamic programming) whereas the world model of Ms Pac-Man can only be approximated via model-free methods such as temporal difference learning.

### *2.6.3 Further Reading*

The RL book of Sutton and Barto [672] is highly recommended for a thorough presentation of RL including Q-learning (Chapter 6). The book is freely available online.[6] A draft version of the latest (2017) version of the book is also available.[7] The survey paper of Kaelbling et al. [316] is another recommended reading of the approaches covered. Finally, for an in-depth analysis of model-based RL approaches you are referred to the dynamic programming book of Bertsekas [44].

[6] http://incompleteideas.net/sutton/book/ebook/the-book.html

[7] http://incompleteideas.net/sutton/book/the-book-2nd.html

## 2.7 Unsupervised Learning

As stated earlier, the utility type (or training signal) determines the class of the AI algorithm. In supervised learning the training signal is provided as data labels (target outputs) and in reinforcement learning it is derived as a reward from the environment. Unsupervised learning instead attempts to discover associations of the input by searching for patterns among all input data attributes and without having access to a target output—a machine learning process that is usually inspired by Hebbian learning [256] and the principles of self-organization [20]. With unsupervised learning we focus on the intrinsic structure of and associations in the data instead of attempting to imitate or predict target values. We cover two unsupervised learning tasks with corresponding algorithms: **clustering** and **frequent pattern mining**.

### *2.7.1 Clustering*

**Clustering** is the unsupervised learning task of finding unknown groups of a number of data points so that data within a group (or else, **cluster**) is similar to each other and dissimilar to data from other clusters. Clustering has found applications in detecting groups of data across multiple attributes and in data reduction tasks such as data compression, noise smoothing, outlier detection and dataset partition. Clustering is of key importance for games with applications in player modeling, game playing and content generation.

As with classification, clustering places data into classes; the labels of the classes, however, are unknown a priori and clustering algorithms aim to discover them by assessing their quality iteratively. Since the correct clusters are unknown, similarity (and dissimilarity) depends only on the data attributes used. Good clusters are characterized by two core properties: 1) high *intra*-cluster similarity, or else, high compactness and 2) low *inter*-cluster similarity, or else, good separation. A popular measure of compactness is the average distance between every sample in the cluster and the closest representative point—e.g., centroid—as used in the k-means algorithm. Examples of separation measures include the **single link** and the **complete link**: the former is the *smallest* distance between any sample in one cluster and any sample in the other cluster; the latter is the *largest* distance between any sample in one cluster and any sample in the other cluster. While compactness and separation are objective measures of cluster validity, it is important to note that they are not indicators of cluster meaningfulness.

Beyond the validity metrics described above, clustering algorithms are defined by a **membership function** and a **search procedure**. The membership function defines the structure of the clusters in relation to the data samples. The search procedure is a strategy we follow to cluster our data given a membership function and a validity metric. Examples of such strategies include splitting all data points into clusters at once (as in k-means), or recursively merging (or splitting) clusters (as in hierarchical clustering).

Clustering can be realized via a plethora of algorithms including hierarchical clustering, k-means [411], k-medoids [329], DBSCAN [196] and self-organizing maps [347]. The algorithms are dissimilar in the way they define what a cluster is and how they form it. Selecting an appropriate clustering algorithm and its corresponding parameters, such as which distance function to use or the number of clusters to expect, depends on the aims of the study and the data available. In the remainder of the section we outline the clustering algorithms we find to be the most useful for the study of AI in games.

#### 2.7.1.1 K-Means Clustering

**K-means** [411] is a vector quantization method that is considered the most popular clustering algorithm as it offers a good balance between simplicity and effectiveness. It follows a simple data partitioning approach according to which it partitions a database of objects into a set of $k$ clusters, such that the sum of squared Euclidean distances between data points and their corresponding cluster center (centroid) is minimized—this distance is also known as the **quantization error**.

In k-means each cluster is defined by one point, that is the centroid of the cluster, and each data sample is assigned to the closest centroid. The centroid is the mean of the data samples in the cluster. The intra-cluster validity metric used by k-means is the average distance to the centroid. Initially, the data samples are randomly assigned to a cluster and then the algorithm proceeds by alternating between the re-assignment of data into clusters and the update of the resulting centroids. The basic steps of the algorithm are as follows:

Given $k$

1. Randomly partition the data points into $k$ nonempty clusters.
2. Compute the position of the centroids of the clusters of the current partitioning. Centroids are the centers (mean points) of the clusters.
3. Assign each data point to the cluster with the nearest centroid.
4. Stop when the assignment does not change; otherwise go to step 2.

While k-means is very popular due to its simplicity it has a number of considerable weaknesses. First, it is applicable only to data objects in a continuous space. Second, one needs to specify the number of clusters, $k$, in advance. Third, it is not suitable to discover clusters with non-convex shapes as it can only find hyper-spherical clusters. Finally, k-means is sensitive to outliers as data points with extremely large (or small) values may substantially distort the distribution of the data and affect the performance of the algorithm. As we will see below, hierarchical clustering manages to overcome some of the above drawbacks, suggesting a useful alternative approach to data clustering.

#### 2.7.1.2 Hierarchical Clustering

Clustering methods that attempt to build a hierarchy of clusters fall under the **hierarchical clustering** approach. Generally speaking there are two main strategies available: the *agglomerative* and the *divisive*. The former constructs hierarchies in a bottom-up fashion by gradually merging data points together, whereas the latter constructs hierarchies of clusters by gradually splitting the dataset in a top-down fashion. Both clustering strategies are greedy. Hierarchical clustering uses a distance matrix as the clustering strategy (whether agglomerative or divisive). This method does not require the number of clusters $k$ as an input, but needs a termination condition.

Indicatively, we present the basic steps of the agglomerative clustering algorithm which are as follows:

Given $k$

1. Create one cluster per data sample.
2. Find the two closest data samples—i.e., find the shortest Euclidean distance between two points (single link)—which are not in the same cluster.
3. Merge the clusters containing these two samples.
4. Stop if there are $k$ clusters; otherwise go to step 2.

In divisive hierarchical clustering instead, all data are initially in the same cluster which is split until every data point is on its own cluster following a split strategy—e.g., DIvisive ANAlysis Clustering (DIANA) [330]—or employing another clustering algorithm to split the data in two clusters—e.g., 2-means.

Once clusters of data are iteratively merged (or split), one can visualize the clusters by decomposing the data into several levels of nested partitioning. In other words, one can observe a tree representation of clusters which is also known as a **dendrogram**. The clustering of data is obtained by cutting the dendrogram at the desired level of squared Euclidean distance. For the interested reader, a dendrogram example is illustrated in Chapter 5.

Hierarchical clustering represents clusters as the set of data samples contained in them and, as a result, a data sample belongs to the same cluster as its closest sample. In k-means instead, each cluster is represented by a centroid and thus a data sample belongs to the cluster represented by the closest centroid. Further, when it comes to cluster validity metrics, agglomerative clustering uses the shortest distance between any sample in one cluster and a sample in another whereas k-means uses the average distance to the centroid. Due to these different algorithmic properties hierarchical clustering has the capacity to cluster data that come in any form of a connected shape; k-means, on the other hand, is only limited to hyper-spherical clusters.

#### 2.7.1.3 Clustering for Ms Pac-Man

One potential application of clustering for controlling Ms Pac-Man would be to model ghost behaviors and use that information as an input to the controller of Ms Pac-Man. Whether it is k-means or hierarchical clustering, the algorithm would consider different attributes of ghost behavior—such as level exploration, behavior divergence, distance between ghosts, etc.—and cluster the ghosts into behavioral patterns or profiles. The controller of Ms Pac-Man would then consider the ghost profile met in a particular level as an additional input for guiding the agent better.

Arguably, beyond agent control, we can think of better uses of clustering for this game such as profiling Ms Pac-Man players and generating appropriate levels or challenges for them so that the game is balanced. As mentioned earlier, however, the focus of the Ms Pac-Man examples is on the control of the playing agent for the purpose of maintaining a consistent paradigm throughout this chapter.

### *2.7.2 Frequent Pattern Mining*

**Frequent pattern mining** is a set of techniques that attempt to derive frequent patterns and structures in data. Patterns include sequences and itemsets. Frequent pattern mining was first proposed for mining association rules [6], which aims to identify a number of data attributes that frequently associate to each other, thereby forming conditional rules among them. There are two types of frequent pattern mining that are of particular interest for game AI: **frequent itemset mining** and **frequent sequence mining**. The former aims to find structure among data attributes that have no particular internal order whereas the latter aims to find structure among data attributes based on an inherent temporal order. While associated with the unsupervised learning paradigm, frequent pattern mining is dissimilar in both the aims and the algorithmic procedures it follows.

Popular and scalable frequent pattern mining methods include the **Apriori** algorithm [6] for itemset mining, and SPADE [793] and **GSP** [652, 434, 621] for sequence mining. In the remainder of this section we outline Apriori and GSP as representative algorithms for frequent itemset and frequent sequence mining, respectively.

#### 2.7.2.1 Apriori

Apriori [7] is an algorithm for frequent itemset mining. The algorithm is appropriate for mining datasets that contain sets of instances (also named transactions) that each feature a set of items, or an **itemset**. Examples of transactions include books bought by an Amazon customer or apps bought by a smartphone user. The algorithm is very simple and can be described as follows: given a predetermined threshold named **support** ($T$), Apriori detects the itemsets which are subsets of at least $T$ transactions

in the database. In other words, Apriori will attempt to identify all itemsets that have at least a minimum support which is the minimum number of times an itemset exists in the dataset.

To demonstrate Apriori in a game example, below we indicatively list events from four players of an online role playing game:

- <Completed more than 10 levels; Most achievements unlocked; Bought the shield of the magi>
- <Completed more than 10 levels; Bought the shield of the magi>
- <Most achievements unlocked; Bought the shield of the magi; Found the Wizard's purple hat>
- <Most achievements unlocked; Found the Wizard's purple hat; Completed more than 10 levels; Bought the shield of the magi>

If in the example dataset above we assume that the support is 3, the following 1-itemsets (sets of only one item) can be found: <Completed more than 10 levels>, <Most achievements unlocked> and <Bought the shield of the magi>. If instead, we seek 2-itemsets with a support threshold of 3 we can find <Completed more than 10 levels, Bought the shield of the magi>, as three of the transactions above contain both of these items. Longer itemsets are not available (not frequent) for support count 3. The process can be repeated for any support threshold we wish to detect frequent itemsets for.

#### 2.7.2.2 Generalized Sequential Patterns

Frequent itemset mining algorithms are not adequate if the **sequence** of events is the critical information we wish to mine from a dataset. The dataset may contain events in an ordered set of sequences such as temporal sequence data or time series. Instead, we need to opt for a frequent sequence mining approach. The sequence mining problem can be simply described as the process of finding frequently occurring subsequences given a sequence or a set of sequences.

More formally, given a dataset in which each sample is a sequence of events, namely a **data sequence**, a sequential pattern defined as a subsequence of events is a **frequent sequence** if it occurs in the samples of the dataset regularly. A frequent sequence can be defined as a sequential pattern that is supported by, at least, a minimum amount of data-sequences. This amount is determined by a threshold named minimum support value. A data sequence supports a sequential pattern if and only if it contains all the events present in the pattern in the same order. For example, the data-sequence $< x_0, x_1, x_2, x_3, x_4, x_5 >$ supports the pattern $< x_0, x_5 >$. As with frequent itemset mining, the amount of data sequences that support a sequential pattern is referred as the **support count**.

The Generalized Sequential Patterns (GSP) algorithm [652] is a popular method for mining frequent sequences in data. GSP starts by extracting the frequent sequences with a single event, namely 1-sequences. That set of sequences is self-joined to generate all 2-sequence candidates for which we calculate their support

count. Those sequences that are frequent (i.e., their support count is greater than a threshold value) are then self-joined to generate the set of 3-sequence candidates. The algorithm is gradually increasing the length of the sequences in each algorithmic step until the next set of candidates is empty. The basic principle of the algorithm is that if a sequential pattern is frequent, then its *contiguous* subsequences are also frequent.

#### 2.7.2.3 Frequent Pattern Mining for Ms Pac-Man

Patterns of events of sequences can be extracted to assist the control of Ms Pac-Man. Itemsets may be identified across successful events of expert Ms Pac-Man players given a particular support count. For instance, an Apriori algorithm running on events across several different expert players might reveal that a frequent 2-itemset is the following: <player went for the upper left corner first, player ate the bottom right power pill first>. Such information can be useful explicitly for designing rules for controlling Ms Pac-Man.

Beyond itemsets, frequencies of ghost events can be considered for playing Ms Pac-Man. For example, by running GSP on extracted attributes of ghosts it might turn out that when Ms Pac-Man eats a power pill it is very likely that the Blinky ghost moves left (<power pill, Blinky left>). Such frequent sequences can form additional inputs of any Ms Pac-Man controller—e.g., an ANN. Chapter 5 details an example on this frequent sequence mining approach in a 3D prey-predator game.

### *2.7.3 Further Reading*

A general introduction to frequent pattern mining is offered in [6]. The Apriori algorithm is detailed in the original article of Agrawal and Srikant [7] whereas GSP is covered throughly in [652].

## 2.8 Notable Hybrid Algorithms

AI methods can be interwoven in numerous ways to yield new sophisticated algorithms that aggregate the strengths of their combined parts, often with an occurring *gestalt* effect. You can, for instance, let GAs evolve your behavior trees or FSMs; you can instead empower MCTS with ANN estimators for tree pruning; or you can add a component of local search in every search algorithm covered earlier. We name the resulting combinations of AI methods as **hybrid** algorithms and in this section we cover the two most influential, in our opinion, hybrid game AI algorithms: neuroevolution and temporal difference learning with ANN function approximators.

### 2.8.1 Neuroevolution

The evolution of artificial neural networks, or else **neuroevolution**, refers to the design of artificial neural networks—their connection weights, their topology, or both—using evolutionary algorithms [786]. Neuroevolution has been successfully applied in the domains of artificial life, robot control, generative systems and computer games. The algorithm's wide applicability is primarily due to two main reasons. First, many AI problems can be viewed as function optimization problems whose underlying general function can be approximated via an ANN. Second, neuroevolution is a method grounded in biological metaphors and evolutionary theory and inspired by the way brains evolve [567].

This evolutionary (reinforcement) learning approach is applicable either when the error function available is not differentiable or when target outputs are not available. The former may occur, for instance, when the activation functions employed in the ANN are not continuous and, thus, not differentiable. (This is a prominent phenomenon, for instance, in the compositional pattern producing networks [653].) The latter may occur in a domain for which we have no samples of good (or bad) behavior or it is impossible to define objectively what a good behavior might be. Instead of backpropagating the error and adjusting the ANN based on gradient search, neuroevolution designs ANNs via metaheuristic (evolutionary) search. In contrast to supervised learning, neuroevolution does not require a dataset of input-output pairs to train ANNs. Rather, it requires only a measure of a ANN's performance on the problem under investigation, for instance, the score of a game playing agent that is controlled by an ANN.

The core algorithmic steps of neuroevolution are as follows:

1. A population of chromosomes that represent ANNs is evolved to optimize a fitness function that characterizes the utility (quality) of the ANN representation. The population of chromosomes (ANNs) is typically initialized randomly.
2. Each chromosome is encoded into an ANN which is, in turn, tested on the task under optimization.
3. The testing procedure assigns a fitness value for each ANN of the population. The fitness of an ANN defines its measure of performance on the task.
4. Once the fitness values for all genotypes in the current population are determined, a selection strategy (e.g., roulette-wheel, tournament) is applied to pick the parents for the next generation.
5. A new population of offspring is generated by applying genetic operators on the selected ANN-encoded chromosomes. Mutation and/or crossover are applied on the chromosomes in the same way as in any evolutionary algorithm.
6. A replacement strategy (e.g., steady-state, elitism, generational) is applied to determine the final members of the new population.

7. Similarly to a typical evolutionary algorithm, the generational loop (steps 2 to 6) is repeated until we exhaust our computational budget or we are happy with the obtained fitness of the current population.

Typically there are two types of neuroevolution approaches: those that consider the evolution of a network's connection weights only and those that evolve both the connection weights and the topology of the network (including connection types and activation functions). In the former type of neuroevolution, the weight vector is encoded and represented genetically as a chromosome; in the latter type, the genetic representation includes an encoding of the ANN topology. Beyond simple MLPs, the ANN types that have been considered for evolution include the NeuroEvolution of Augmenting Topologies (NEAT) [655] and the compositional pattern producing networks [653].

Neuroevolution has found extensive use in the games domain in roles such as those of evaluating the state-action space of a game, selecting an appropriate action, selecting among possible strategies, modeling opponent strategies, generating content, and modeling player experience [567]. The algorithm's efficiency, scalability, broad applicability, and open-ended learning are a few of the reasons that make neuroevolution a good general method for many game AI tasks [567].

#### 2.8.1.1 Neuroevolution for Ms Pac-Man

One simple way to implement neuroevolution in Ms Pac-Man is to first design an ANN that considers the game state as input and output actions for Ms Pac-Man. The weights of the ANN can be evolved using a typical evolutionary algorithm and following the steps of neuroevolution as described above. The fitness of each ANN in the population is obtained by equipping Ms Pac-Man with each ANN in the population and letting her play the game for a while. The performance of the agent within that simulation time (e.g., the score) can determine the fitness value of the ANN. Figure 2.18 illustrates the steps of ANN encoding and fitness assignment in this hypothetical implementation of neuroevolution in Ms Pac-Man.

### *2.8.2 TD Learning with ANN Function Approximators*

Reinforcement learning typically uses tabular representations to store knowledge. As mentioned earlier in the RL section, representing knowledge this way may drain our available computational resources since the size of the look-up table increases exponentially with respect to the action-state space. The most popular way of addressing this challenge is to use an ANN as a value (or Q value) approximator, thereby replacing the table. Doing so makes it possible to apply the algorithm to

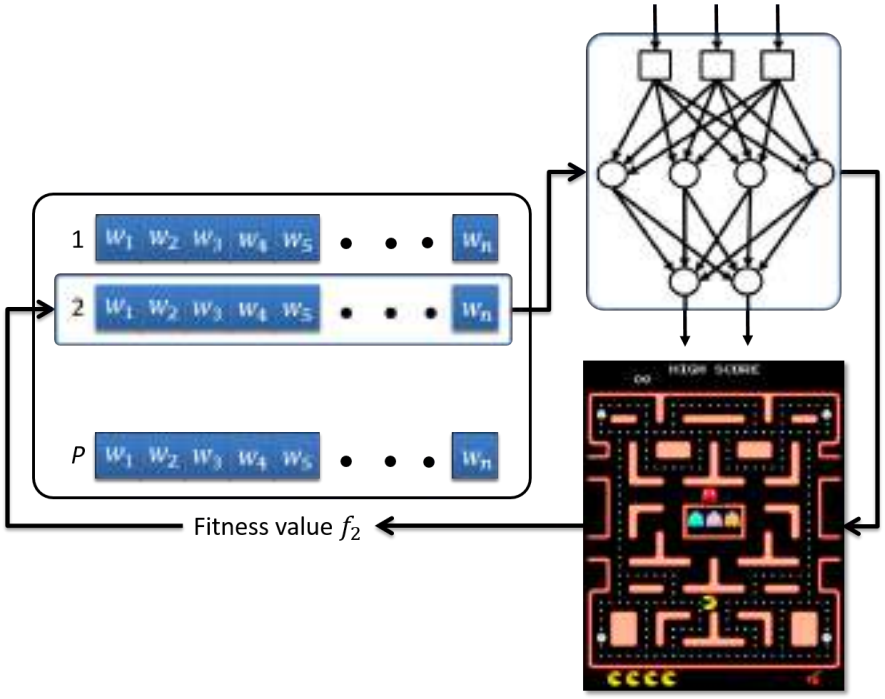


**Fig. 2.18** Neuroevolution in Ms Pac-Man. The figure visualizes step 2 (ANN encoding) and step 3 (fitness assignment) of the algorithm for assigning a fitness value to chromosome 2 in the population (of size $P$). In this example, only the weights of the ANN are evolved. The $n$ weights of the chromosome are first encoded in the ANN and then the ANN is tested in Ms Pac-Man for a number of simulation steps (or game levels). The result of the game simulation determines the fitness value ($f_2$) of the ANN.

larger spaces of action-state representations. Further, an ANN as a function approximator of Q, for instance, can handle problems with continuous state spaces which are infinitely large.

In this section, we outline two milestone examples of algorithms that utilize the ANN universal approximation capacity for temporal difference learning. The algorithms of TD-Gammon and deep Q network have been applied, respectively, to master the game of backgammon and play Atari 2600 arcade games at super-human level. Both algorithms are applicable to any RL task beyond these particular games, but the games that made them popular are used to describe the algorithms below.

#### 2.8.2.1 TD-Gammon

Arguably one of the most popular success stories of AI in games is that of Tesauro's TD-Gammon software that plays backgammon on the grandmaster-level [689]. The learning algorithm was a hybrid combination of an MLP and a temporal difference

variant named TD($\lambda$); see Chapter 7 of [672] for further details on the TD($\lambda$) algorithm.

TD-Gammon used a standard multilayer neural network to approximate the value function. The input of the MLP was a representation of the current state of the board (Tesauro used 192 inputs) whereas the output of the MLP was the predicted probability of winning given the current state. Rewards were defined as zero for all board states except those on which the game was won. The MLP was then trained iteratively by playing the game against itself and selecting actions based on the estimated probability of winning. Each game was treated as a training episode containing a sequence of positions which were used to train the weights of the MLP by backpropagating temporal difference errors of its output.

TD-Gammon 0.0 played about $300,000$ games against itself and managed to play as well as the best backgammon computer of its time. While TD-Gammon 0.0 did not win the performance horse race, it gave us a first indication of what is achievable with RL even without any backgammon expert knowledge integrated in the AI algorithm. The next iteration of the algorithm (TD-Gammon 1.0) naturally incorporated expert knowledge through specialized backgammon features that altered the input of the MLP and achieved substantially higher performance. From that point onwards the number of hidden neurons and the number of self-payed games determined greatly the version of the algorithm and its resulting capacity. From TD-Gammon 2.0 (40 hidden neurons) to TD-Gammon 2.1 (80 hidden neurons) the performance of TD-Gammon gradually increased and, with TD Gammon 3.0 (160 hidden neurons), it reached the playing strength of the best human player in backgammon [689].

#### 2.8.2.2 Deep Q Network

While the combination of RL and ANNs results in very powerful hybrid algorithms, the performance of the algorithm traditionally depended on the design of the input space for the ANN. As we saw earlier, even the most successful applications of RL such as the TD-Gammon agent managed to reach human-level playing performance by integrating game specific features in the input space, thereby adding expert knowledge about the game. It was up until very recently that the combination of RL and ANNs managed to reach human-level performance in a game without considering ad-hoc designed features but rather discovering them merely through learning. A team from Google's DeepMind [464] developed a reinforcement learning agent called *deep Q network* (DQN) that trains a deep convolutional ANN via Q-learning. DQN managed to reach or exceed human-level playing performance in 29 out of 46 arcade (Atari 2600) games of the Arcade Learning Environment [40] it was trained on [464].

DQN is inspired by and based upon TD-Gammon since it uses an ANN as the function approximator for TD learning via gradient descent. As in TD-Gammon, the gradient is calculated by backpropagating the temporal difference errors. However, instead of using TD($\lambda$) as the underlying RL algorithm, DQN uses Q-learning. Fur-

ther, the ANN is not a simple MLP but rather a deep convolutional neural network. DQN played each game of ALE for a large amount of frames (50 million frames). This amounts to about 38 days of playing time for each game [464].

The DQN analyses a sequence of four game screens simultaneously and approximates the future game score per each possible action given its current state. In particular, the DQN uses the pixels from the four most recent game screens as its inputs, resulting in ANN input size of $84 \times 84$ (screen size in pixels) $\times 4$. No other game-specific knowledge was given to the DQN beyond the screen pixel information. The architecture used for the convolutional ANN has three hidden layers that yield 32 $20 \times 20$, 64 $9 \times 9$ and 64 $7 \times 7$ feature maps, respectively. The first (low-level) layers of the DQN process the pixels of the game screen and extract specialized visual features. The convolutional layers are followed by a fully connected hidden layer and an output layer. Each hidden layer is followed by a rectifier nolinearity. Given a game state represented by the network's input, the outputs of the DQN are the estimated optimal action values (optimal Q-values) of the corresponding state-action pairs. The DQN is trained to approximate the Q-values (the actual score of the game) by receiving immediate rewards from the game environment. In particular, the reward is $+1$ if the score increases in between two successive time steps (frames), it is $-1$ if the score decreases, and 0 otherwise. DQN uses an $\varepsilon$-greedy policy for its action-selection strategy. It is worth mentioning that, at the time of writing, there are newer and more efficient implementations of the deep reinforcement learning concept such as the Asynchronous Advantage Actor-Critic (A3C) algorithm [463].

#### 2.8.2.3 TD Learning with ANN Function Appoximator for Ms Pac-Man

We can envisage a DQN approach for controlling Ms Pac-Man in a similar fashion to that with which ALE agents were trained [464]. A deep convolutional neural network scans the level image on a pixel-to-pixel basis (see Fig. 2.19). The image goes through a number of convolution and fully connected layers which eventually feed the input of an MLP that outputs the four possible actions for Ms Pac-Man (keep direction, move backwards, turn left, turn right). Once an action is applied, the score of the game is used as the immediate reward for updating the weights of the deep network (the convolutional ANN and the MLP). By playing for a sufficient time period the controller gathers experience (image snapshots, actions, and corresponding rewards) which trains the deep ANN to approximate a policy that maximizes the score for Ms Pac-Man.

### *2.8.3 Further Reading*

For a recent thorough survey on the application of neuroevolution in games the reader may refer to [567]. For a complete review of neuroevolution please refer to Floreano et al. [205]. CPPNs and NEAT are covered in detail in [653] and [655]

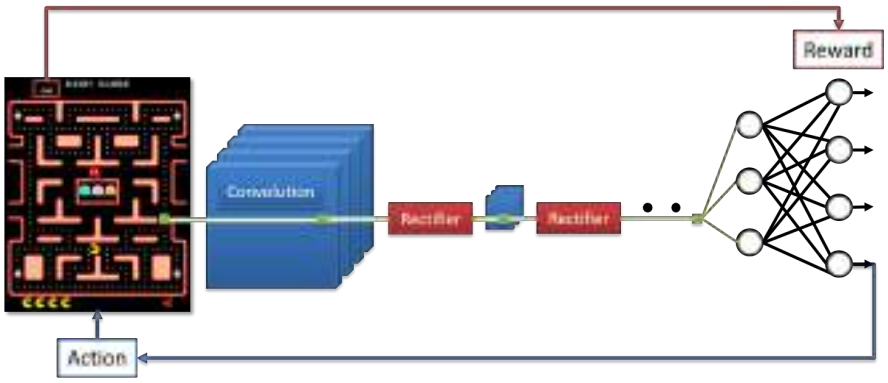


**Fig. 2.19** A deep Q-learning approach for Ms Pac-Man. Following [464], the network's first part contains a set of convolution layers which are followed by rectifier nonlinearities. The final layers of the DQN we present in this example are fully connected employing ReLUs, as in [464].

respectively. TD-Gammon and DQN are covered in detail in [689] and [464], respectively. Both are also placed within the greater RL field in the upcoming second edition of [672]. Details about the A3C algorithm can be found in [463] and implementations of the algorithm can be found directly as part of Tensorflow.

## 2.9 Summary

This chapter covered the AI methods we feel the reader of this book needs to be familiar with. We expect, however, that our readers have a basic background in AI or have completed a course in fundamentals of AI prior to reading this book. Hence, the algorithms were not covered in detail since the emphasis of this book is on the application of AI within the domain of games and not on AI per se. On that basis, we used the game of Ms Pac-Man as the overarching application testbed of all algorithms throughout this chapter.

The families of algorithms we discussed include traditional ad-hoc behavior authoring methods (such as finite state machines and behavior trees), tree search (such as best-first, Minimax and Monte Carlo tree search), evolutionary computation (such as local search and evolutionary algorithms), supervised learning (e.g., neural networks, support vector machines and decision trees), reinforcement learning (e.g., Q-learning), unsupervised learning (such as clustering and frequent pattern mining), and hybrid algorithms such as evolving artificial neural networks and artificial neural networks as approximators of expected rewards.

With this chapter we reached the end of the first, introductory, part of the book. The next part begins with a chapter on the most traditional and widely explored task of AI in games: playing!

# References


1. Espen Aarseth. Genre trouble. *Electronic Book Review*, 3, 2004.
2. Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, et al. TensorFlow: Large-scale machine learning on heterogeneous distributed systems. *arXiv preprint arXiv:1603.04467*, 2016.
3. Ryan Abela, Antonios Liapis, and Georgios N. Yannakakis. A constructive approach for the generation of underwater environments. In *Proceedings of the FDG workshop on Procedural Content Generation in Games*, 2015.
4. David H. Ackley, Geoffrey E. Hinton, and Terrence J. Sejnowski. A learning algorithm for Boltzmann machines. *Cognitive Science*, 9(1):147–169, 1985.
5. Alexandros Agapitos, Julian Togelius, Simon M. Lucas, Jürgen Schmidhuber, and Andreas Konstantinidis. Generating diverse opponents with multiobjective evolution. In *Computational Intelligence and Games, 2008. CIG'08. IEEE Symposium On*, pages 135–142. IEEE, 2008.
6. Rakesh Agrawal, Tomasz Imieliński, and Arun Swami. Mining association rules between sets of items in large databases. In *ACM SIGMOD Record*, pages 207–216. ACM, 1993.
7. Rakesh Agrawal and Ramakrishnan Srikant. Fast algorithms for mining association rules. In *Proceedings of the 20th International Conference on Very Large Data Bases, VLDB*, pages 487–499, 1994.
8. John B. Ahlquist and Jeannie Novak. *Game development essentials: Game artificial intelligence*. Delmar Pub, 2008.
9. Zach Aikman. Galak-Z: Forever: Building Space-Dungeons Organically. In *Game Developers Conference*, 2015.
10. Bob Alexander. The beauty of response curves. *AI Game Programming Wisdom*, page 78, 2002.
11. Krishna Aluru, Stefanie Tellex, John Oberlin, and James MacGlashan. Minecraft as an experimental world for AI in robotics. In *AAAI Fall Symposium*, 2015.
12. Samuel Alvernaz and Julian Togelius. Autoencoder-augmented neuroevolution for visual doom playing. In *IEEE Conference on Computational Intelligence and Games*. IEEE, 2017.
13. Omar Alzoubi, Rafael A. Calvo, and Ronald H. Stevens. Classification of EEG for Affect Recognition: An Adaptive Approach. In *AI 2009: Advances in Artificial Intelligence*, pages 52–61. Springer, 2009.
14. Mike Ambinder. Biofeedback in gameplay: How Valve measures physiology to enhance gaming experience. In *Game Developers Conference*, San Francisco, California, US, 2011.
15. Dan Amerson, Shaun Kime, and R. Michael Young. Real-time cinematic camera control for interactive narratives. In *Proceedings of the 2005 ACM SIGCHI International Conference on Advances in Computer Entertainment Technology*, pages 369–369. ACM, 2005.

16. Elisabeth André, Martin Klesen, Patrick Gebhard, Steve Allen, and Thomas Rist. Integrating models of personality and emotions into lifelike characters. In *Affective interactions*, pages 150–165. Springer, 2000.
17. John L. Andreassi. *Psychophysiology: Human Behavior and Physiological Response*. Psychology Press, 2000.
18. Rudolf Arnheim. *Art and visual perception: A psychology of the creative eye*. University of California Press, 1956.
19. Ivon Arroyo, David G. Cooper, Winslow Burleson, Beverly Park Woolf, Kasia Muldner, and Robert Christopherson. Emotion sensors go to school. In *Proceedings of Conference on Artificial Intelligence in Education (AIED)*, pages 17–24. IOS Press, 2009.
20. W. Ross Ashby. Principles of the self-organizing system. In *Facets of Systems Science*, pages 521–536. Springer, 1991.
21. Daniel Ashlock. *Evolutionary computation for modeling and optimization*. Springer, 2006.
22. Stylianos Asteriadis, Kostas Karpouzis, Noor Shaker, and Georgios N. Yannakakis. Does your profile say it all? Using demographics to predict expressive head movement during gameplay. In *Proceedings of UMAP Workshops*, 2012.
23. Stylianos Asteriadis, Paraskevi Tzouveli, Kostas Karpouzis, and Stefanos Kollias. Estimation of behavioral user state based on eye gaze and head pose—application in an e-learning environment. *Multimedia Tools and Applications*, 41(3):469–493, 2009.
24. Phillipa Avery, Sushil Louis, and Benjamin Avery. Evolving coordinated spatial tactics for autonomous entities using influence maps. In *Computational Intelligence and Games, 2009. CIG 2009. IEEE Symposium on*, pages 341–348. IEEE, 2009.
25. Phillipa Avery, Julian Togelius, Elvis Alistar, and Robert Pieter van Leeuwen. Computational intelligence and tower defence games. In *Evolutionary Computation (CEC), 2011 IEEE Congress on*, pages 1084–1091. IEEE, 2011.
26. Ruth Aylett, Sandy Louchart, Joao Dias, Ana Paiva, and Marco Vala. FearNot!–an experiment in emergent narrative. In *Intelligent Virtual Agents*, pages 305–316. Springer, 2005.
27. Simon E. Ortiz B., Koichi Moriyama, Ken-ichi Fukui, Satoshi Kurihara, and Masayuki Numao. Three-subagent adapting architecture for fighting videogames. In *Pacific Rim International Conference on Artificial Intelligence*, pages 649–654. Springer, 2010.
28. Sander Bakkes, Pieter Spronck, and Jaap van den Herik. Rapid and reliable adaptation of video game AI. *IEEE Transactions on Computational Intelligence and AI in Games*, 1(2):93–104, 2009.
29. Sander Bakkes, Shimon Whiteson, Guangliang Li, George Viorel Vişniuc, Efstathios Charitos, Norbert Heijne, and Arjen Swellengrebel. Challenge balancing for personalised game spaces. In *Games Media Entertainment (GEM), 2014 IEEE*, pages 1–8. IEEE, 2014.
30. Rainer Banse and Klaus R. Scherer. Acoustic profiles in vocal emotion expression. *Journal of Personality and Social Psychology*, 70(3):614, 1996.
31. Ray Barrera, Aung Sithu Kyaw, Clifford Peters, and Thet Naing Swe. *Unity AI Game Programming*. Packt Publishing Ltd, 2015.
32. Gabriella A. B. Barros, Antonios Liapis, and Julian Togelius. Data adventures. In *Proceedings of the FDG workshop on Procedural Content Generation in Games*, 2015.
33. Richard A. Bartle. *Designing virtual worlds*. New Riders, 2004.
34. Chris Bateman and Richard Boon. *21st Century Game Design (Game Development Series)*. Charles River Media, Inc., 2005.
35. Chris Bateman and Lennart E. Nacke. The neurobiology of play. In *Proceedings of the International Academic Conference on the Future of Game Design and Technology*, pages 1–8. ACM, 2010.
36. Christian Bauckhage, Anders Drachen, and Rafet Sifa. Clustering game behavior data. *IEEE Transactions on Computational Intelligence and AI in Games*, 7(3):266–278, 2015.
37. Yoann Baveye, Jean-Noël Bettinelli, Emmanuel Dellandrea, Liming Chen, and Christel Chamaret. A large video database for computational models of induced emotion. In *Proceedings of Affective Computing and Intelligent Interaction*, pages 13–18, 2013.
38. Jessica D. Bayliss. Teaching game AI through Minecraft mods. In *2012 IEEE International Games Innovation Conference (IGIC)*, pages 1–4. IEEE, 2012.

39. Farès Belhadj. Terrain modeling: a constrained fractal model. In *Proceedings of the 5th international conference on Computer graphics, virtual reality, visualisation and interaction in Africa*, pages 197–204. ACM, 2007.
40. Marc G. Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. The arcade learning environment: An evaluation platform for general agents. *arXiv preprint arXiv:1207.4708*, 2012.
41. Yoshua Bengio. Learning deep architectures for AI. *Foundations and Trends in Machine Learning*, 2(1):1–127, 2009.
42. José Luis Bernier, C. Ilia Herráiz, J. J. Merelo, S. Olmeda, and Alberto Prieto. Solving Mastermind using GAs and simulated annealing: a case of dynamic constraint optimization. In *Parallel Problem Solving from Nature (PPSN) IV*, pages 553–563. Springer, 1996.
43. Kent C. Berridge. Pleasures of the brain. *Brain and Cognition*, 52(1):106–128, 2003.
44. Dimitri P. Bertsekas. *Dynamic programming and optimal control*. Athena Scientific Belmont, MA, 1995.
45. Nadav Bhonker, Shai Rozenberg, and Itay Hubara. Playing SNES in the Retro Learning Environment. *arXiv preprint arXiv:1611.02205*, 2016.
46. Mateusz Bialas, Shoshannah Tekofsky, and Pieter Spronck. Cultural influences on play style. In *Computational Intelligence and Games (CIG), 2014 IEEE Conference on*, pages 1–7. IEEE, 2014.
47. Nadia Bianchi-Berthouze and Christine L. Lisetti. Modeling multimodal expression of user's affective subjective experience. *User Modeling and User-Adapted Interaction*, 12(1):49–84, 2002.
48. Darse Billings, Denis Papp, Jonathan Schaeffer, and Duane Szafron. Opponent modeling in poker. In *AAAI/IAAI*, pages 493–499, 1998.
49. Christopher M. Bishop. *Pattern Recognition and Machine Learning*. 2006.
50. Staffan Björk and Jesper Juul. Zero-player games. In *Philosophy of Computer Games Conference, Madrid*, 2012.
51. Vikki Blake. Minecraft Has 55 Million Monthly Players, 122 Million Sales. *Imagine Games Network*, February 2017.
52. Paris Mavromoustakos Blom, Sander Bakkes, Chek Tien Tan, Shimon Whiteson, Diederik M. Roijers, Roberto Valenti, and Theo Gevers. Towards Personalised Gaming via Facial Expression Recognition. In *Proceedings of AIIDE*, 2014.
53. Margaret A. Boden. What is creativity. *Dimensions of creativity*, pages 75–117, 1994.
54. Margaret A. Boden. Creativity and artificial intelligence. *Artificial Intelligence*, 103(1):347–356, 1998.
55. Margaret A. Boden. *The creative mind: Myths and mechanisms*. Psychology Press, 2004.
56. Slawomir Bojarski and Clare Bates Congdon. REALM: A rule-based evolutionary computation agent that learns to play Mario. In *Computational Intelligence and Games (CIG), 2010 IEEE Symposium on*, pages 83–90. IEEE, 2010.
57. Luuk Bom, Ruud Henken, and Marco Wiering. Reinforcement learning to train Ms. Pac-Man using higher-order action-relative inputs. In *Adaptive Dynamic Programming and Reinforcement Learning (ADPRL), 2013 IEEE Symposium on*, pages 156–163. IEEE, 2013.
58. Blai Bonet and Héctor Geffner. Planning as heuristic search. *Artificial Intelligence*, 129(1-2):5–33, 2001.
59. Philip Bontrager, Ahmed Khalifa, Andre Mendes, and Julian Togelius. Matching games and algorithms for general video game playing. In *Twelfth Artificial Intelligence and Interactive Digital Entertainment Conference*, 2016.
60. Michael Booth. The AI systems of Left 4 Dead. In *Fifth Artificial Intelligence and Interactive Digital Entertainment Conference (Keynote)*, 2009.
61. Adi Botea, Martin Müller, and Jonathan Schaeffer. Near optimal hierarchical path-finding. *Journal of Game Development*, 1(1):7–28, 2004.
62. David M. Bourg and Glenn Seemann. *AI for game developers*. O'Reilly Media, Inc., 2004.
63. Michael Bowling, Neil Burch, Michael Johanson, and Oskari Tammelin. Heads-up limit holdem poker is solved. *Science*, 347(6218):145–149, 2015.

64. Danah Boyd and Kate Crawford. Six provocations for big data. In *A decade in internet time: Symposium on the dynamics of the internet and society*. Oxford Internet Institute, Oxford, 2011.
65. S. R. K. Branavan, David Silver, and Regina Barzilay. Learning to win by reading manuals in a Monte-Carlo framework. *Journal of Artificial Intelligence Research*, 43:661–704, 2012.
66. Michael E. Bratman, David J. Israel, and Martha E. Pollack. Plans and resource-bounded practical reasoning. *Computational Intelligence*, 4(3):349–355, 1988.
67. Leo Breiman, Jerome Friedman, Charles J. Stone, and Richard A. Olshen. *Classification and regression trees*. CRC Press, 1984.
68. Daniel Brewer. Tactical pathfinding on a navmesh. *Game AI Pro: Collected Wisdom of Game AI Professionals*, page 361, 2013.
69. Gerhard Brewka, Thomas Eiter, and Mirosław Truszczyński. Answer set programming at a glance. *Communications of the ACM*, 54(12):92–103, 2011.
70. Rodney Brooks. A robust layered control system for a mobile robot. *IEEE Journal on Robotics and Automation*, 2(1):14–23, 1986.
71. David S. Broomhead and David Lowe. Radial basis functions, multi-variable functional interpolation and adaptive networks. *Royals Signals & Radar Establishment*, 1988.
72. Anna Brown and Alberto Maydeu-Olivares. How IRT can solve problems of ipsative data in forced-choice questionnaires. *Psychological Methods*, 18(1):36, 2013.
73. Daniel Lankford Brown. Mezzo: An adaptive, real-time composition program for game soundtracks. In *Eighth Artificial Intelligence and Interactive Digital Entertainment Conference*, 2012.
74. Cameron Browne. *Automatic generation and evaluation of recombination games*. PhD thesis, Queensland University of Technology, 2008.
75. Cameron Browne. Yavalath. In *Evolutionary Game Design*, pages 75–85. Springer, 2011.
76. Cameron Browne and Frederic Maire. Evolutionary game design. *IEEE Transactions on Computational Intelligence and AI in Games*, 2(1):1–16, 2010.
77. Cameron B. Browne, Edward Powley, Daniel Whitehouse, Simon M. Lucas, Peter I. Cowling, Philipp Rohlfshagen, Stephen Tavener, Diego Perez, Spyridon Samothrakis, and Simon Colton. A survey of Monte Carlo tree search methods. *Computational Intelligence and AI in Games, IEEE Transactions on*, 4(1):1–43, 2012.
78. Nicholas J. Bryan, Gautham J. Mysore, and Ge Wang. ISSE: An Interactive Source Separation Editor. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 257–266, 2014.
79. Bobby D. Bryant and Risto Miikkulainen. Evolving stochastic controller networks for intelligent game agents. In *Evolutionary Computation, 2006. CEC 2006. IEEE Congress on*, pages 1007–1014. IEEE, 2006.
80. Mat Buckland. *Programming game AI by example*. Jones & Bartlett Learning, 2005.
81. Mat Buckland and Mark Collins. *AI techniques for game programming*. Premier Press, 2002.
82. Vadim Bulitko, Yngvi Björnsson, Nathan R. Sturtevant, and Ramon Lawrence. Real-time heuristic search for pathfinding in video games. In *Artificial Intelligence for Computer Games*, pages 1–30. Springer, 2011.
83. Vadim Bulitko, Greg Lee, Sergio Poo Hernandez, Alejandro Ramirez, and David Thue. Techniques for AI-Driven Experience Management in Interactive Narratives. In *Game AI Pro 2: Collected Wisdom of Game AI Professionals*, pages 523–534. AK Peters/CRC Press, 2015.
84. Paolo Burelli. Virtual cinematography in games: investigating the impact on player experience. *Foundations of Digital Games*, 2013.
85. Paolo Burelli and Georgios N. Yannakakis. Combining Local and Global Optimisation for Virtual Camera Control. In *Proceedings of the 2010 IEEE Conference on Computational Intelligence and Games*, Copenhagen, Denmark, August 2010. IEEE.
86. Christopher J. C. Burges. A tutorial on support vector machines for pattern recognition. *Data mining and Knowledge Discovery*, 2(2):121–167, 1998.
87. Michael Buro and David Churchill. Real-time strategy game competitions. *AI Magazine*, 33(3):106, 2012.

88. Carlos Busso, Zhigang Deng, Serdar Yildirim, Murtaza Bulut, Chul Min Lee, Abe Kazemzadeh, Sungbok Lee, Ulrich Neumann, and Shrikanth Narayanan. Analysis of emotion recognition using facial expressions, speech and multimodal information. In *Proceedings of the International Conference on Multimodal Interfaces (ICMI)*, pages 205–211. ACM, 2004.
89. Eric Butler, Adam M. Smith, Yun-En Liu, and Zoran Popovic. A mixed-initiative tool for designing level progressions in games. In *Proceedings of the 26th Annual ACM Symposium on User Interface Software and Technology*, pages 377–386. ACM, 2013.
90. Martin V. Butz and Thies D. Lonneker. Optimized sensory-motor couplings plus strategy extensions for the TORCS car racing challenge. In *IEEE Symposium on Computational Intelligence and Games*, pages 317–324. IEEE, 2009.
91. John T. Cacioppo, Gary G. Berntson, Jeff T. Larsen, Kirsten M. Poehlmann, and Tiffany A. Ito. The psychophysiology of emotion. *Handbook of emotions*, 2:173–191, 2000.
92. Francesco Calimeri, Michael Fink, Stefano Germano, Andreas Humenberger, Giovambattista Ianni, Christoph Redl, Daria Stepanova, Andrea Tucci, and Anton Wimmer. Angry-HEX: an artificial player for Angry Birds based on declarative knowledge bases. *IEEE Transactions on Computational Intelligence and AI in Games*, 8(2):128–139, 2016.
93. Aylin Caliskan, Joanna J. Bryson, and Arvind Narayanan. Semantics derived automatically from language corpora contain human-like biases. *Science*, 356(6334):183–186, 2017.
94. Gordon Calleja. *In-game: from immersion to incorporation*. MIT Press, 2011.
95. Rafael Calvo, Iain Brown, and Steve Scheding. Effect of experimental factors on the recognition of affective mental states through physiological measures. In *AI 2009: Advances in Artificial Intelligence*, pages 62–70. Springer, 2009.
96. Elizabeth Camilleri, Georgios N. Yannakakis, and Alexiei Dingli. Platformer Level Design for Player Believability. In *IEEE Computational Intelligence and Games Conference*. IEEE, 2016.
97. Elizabeth Camilleri, Georgios N. Yannakakis, and Antonios Liapis. Towards General Models of Player Affect. In *Affective Computing and Intelligent Interaction (ACII), 2017 International Conference on*, 2017.
98. Murray Campbell, A. Joseph Hoane, and Feng-hsiung Hsu. Deep blue. *Artificial intelligence*, 134(1-2):57–83, 2002.
99. Henrique Campos, Joana Campos, João Cabral, Carlos Martinho, Jeppe Herlev Nielsen, and Ana Paiva. My Dream Theatre. In *Proceedings of the 2013 International Conference on Autonomous Agents and Multi-Agent Systems*, pages 1357–1358. International Foundation for Autonomous Agents and Multiagent Systems, 2013.
100. Joana Campos, Carlos Martinho, Gordon Ingram, Asimina Vasalou, and Ana Paiva. My dream theatre: Putting conflict on center stage. In *FDG*, pages 283–290, 2013.
101. Alessandro Canossa, Josep B. Martinez, and Julian Togelius. Give me a reason to dig Minecraft and psychology of motivation. In *Computational Intelligence in Games (CIG), 2013 IEEE Conference on*. IEEE, 2013.
102. Luigi Cardamone, Daniele Loiacono, and Pier Luca Lanzi. Interactive evolution for the procedural generation of tracks in a high-end racing game. In *Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation*, pages 395–402. ACM, 2011.
103. Luigi Cardamone, Georgios N. Yannakakis, Julian Togelius, and Pier Luca Lanzi. Evolving interesting maps for a first person shooter. In *Applications of Evolutionary Computation*, pages 63–72. Springer, 2011.
104. Justine Cassell. *Embodied conversational agents*. MIT Press, 2000.
105. Justine Cassell, Timothy Bickmore, Mark Billinghurst, Lee Campbell, Kenny Chang, Hannes Vilhjálmsson, and Hao Yan. Embodiment in conversational interfaces: Rea. In *Proceedings of the SIGCHI conference on Human Factors in Computing Systems*, pages 520–527. ACM, 1999.
106. Marc Cavazza, Fred Charles, and Steven J. Mead. Character-based interactive storytelling. *IEEE Intelligent Systems*, 17(4):17–24, 2002.

107. Marc Cavazza, Fred Charles, and Steven J. Mead. Interacting with virtual characters in interactive storytelling. In *Proceedings of the First International Joint Conference on Autonomous Agents and Multiagent Systems: part 1*, pages 318–325. ACM, 2002.
108. Georgios Chalkiadakis, Edith Elkind, and Michael Wooldridge. Computational aspects of cooperative game theory. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, 5(6):1–168, 2011.
109. Alex J. Champandard. *AI game development: Synthetic creatures with learning and reactive behaviors*. New Riders, 2003.
110. Alex J. Champandard. Behavior trees for next-gen game AI. In *Game Developers Conference, Audio Lecture*, 2007.
111. Alex J. Champandard. Understanding Behavior Trees. *AiGameDev. com*, 2007.
112. Alex J. Champandard. Getting started with decision making and control systems. *AI Game Programming Wisdom*, 4:257–264, 2008.
113. Jason C. Chan. Response-order effects in Likert-type scales. *Educational and Psychological Measurement*, 51(3):531–540, 1991.
114. Senthilkumar Chandramohan, Matthieu Geist, Fabrice Lefevre, and Olivier Pietquin. User simulation in dialogue systems using inverse reinforcement learning. In *Interspeech 2011*, pages 1025–1028, 2011.
115. Devendra Singh Chaplot and Guillaume Lample. Arnold: An autonomous agent to play FPS games. In *Thirty-First AAAI Conference on Artificial Intelligence*, 2017.
116. Darryl Charles and Michaela Black. Dynamic player modelling: A framework for player-centric digital games. In *Proceedings of the International Conference on Computer Games: Artificial Intelligence, Design and Education*, pages 29–35, 2004.
117. Fred Charles, Miguel Lozano, Steven J. Mead, Alicia Fornes Bisquerra, and Marc Cavazza. Planning formalisms and authoring in interactive storytelling. In *Proceedings of TIDSE*, 2003.
118. Guillaume M. J. B. Chaslot, Mark H. M. Winands, H. Jaap van Den Herik, Jos W. H. M. Uiterwijk, and Bruno Bouzy. Progressive strategies for Monte-Carlo tree search. *New Mathematics and Natural Computation*, 4(03):343–357, 2008.
119. Xiang 'Anthony' Chen, Tovi Grossman, Daniel J. Wigdor, and George Fitzmaurice. Duet: Exploring joint interactions on a smart phone and a smart watch. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 159–168, 2014.
120. Zhengxing Chen, Magy Seif El-Nasr, Alessandro Canossa, Jeremy Badler, Stefanie Tignor, and Randy Colvin. Modeling individual differences through frequent pattern mining on role-playing game actions. In *Eleventh Artificial Intelligence and Interactive Digital Entertainment Conference, AIIDE*, 2015.
121. Sonia Chernova, Jeff Orkin, and Cynthia Breazeal. Crowdsourcing HRI through online multiplayer games. In *AAAI Fall Symposium: Dialog with Robots*, pages 14–19, 2010.
122. Wei Chu and Zoubin Ghahramani. Preference learning with Gaussian processes. In *Proceedings of the International Conference on Machine learning (ICML)*, pages 137–144, 2005.
123. David Churchill and Michael Buro. Portfolio greedy search and simulation for large-scale combat in StarCraft. In *Computational Intelligence in Games (CIG), 2013 IEEE Conference on*. IEEE, 2013.
124. David Churchill, Mike Preuss, Florian Richoux, Gabriel Synnaeve, Alberto Uriarte, Santiago Ontañón, and Michal Certickỳ. StarCraft Bots and Competitions. In *Encyclopedia of Computer Graphics and Games*. Springer, 2016.
125. Andrea Clerico, Cindy Chamberland, Mark Parent, Pierre-Emmanuel Michon, Sebastien Tremblay, Tiago H. Falk, Jean-Christophe Gagnon, and Philip Jackson. Biometrics and classifier fusion to predict the fun-factor in video gaming. In *IEEE Computational Intelligence and Games Conference*. IEEE, 2016.
126. Carlos A. Coello Coello, Gary B. Lamont, and David A. van Veldhuizen. *Evolutionary algorithms for solving multi-objective problems*. Springer, 2007.
127. Nicholas Cole, Sushil J. Louis, and Chris Miles. Using a genetic algorithm to tune first-person shooter bots. In *Congress on Evolutionary Computation (CEC)*, pages 139–145. IEEE, 2004.

128. Karen Collins. An introduction to procedural music in video games. *Contemporary Music Review*, 28(1):5–15, 2009.
129. Karen Collins. *Playing with sound: a theory of interacting with sound and music in video games*. MIT Press, 2013.
130. Simon Colton. Creativity versus the perception of creativity in computational systems. In *AAAI Spring Symposium: Creative Intelligent Systems*, 2008.
131. Cristina Conati. Intelligent tutoring systems: New challenges and directions. In *IJCAI*, pages 2–7, 2009.
132. Cristina Conati, Abigail Gertner, and Kurt VanLehn. Using Bayesian networks to manage uncertainty in student modeling. *User Modeling and User-Adapted Interaction*, 12(4):371–417, 2002.
133. Cristina Conati and Heather Maclaren. Modeling user affect from causes and effects. *User Modeling, Adaptation, and Personalization*, pages 4–15, 2009.
134. John Conway. The game of life. *Scientific American*, 223(4):4, 1970.
135. Michael Cook and Simon Colton. Multi-faceted evolution of simple arcade games. In *IEEE Computational Intelligence and Games*, pages 289–296, 2011.
136. Michael Cook and Simon Colton. Ludus ex machina: Building a 3D game designer that competes alongside humans. In *Proceedings of the 5th International Conference on Computational Creativity*, 2014.
137. Michael Cook, Simon Colton, and Alison Pease. Aesthetic Considerations for Automated Platformer Design. In *AIIDE*, 2012.
138. Seth Cooper, Firas Khatib, Adrien Treuille, Janos Barbero, Jeehyung Lee, Michael Beenen, Andrew Leaver-Fay, David Baker, Zoran Popović, et al. Predicting protein structures with a multiplayer online game. *Nature*, 466(7307):756–760, 2010.
139. Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine Learning*, 20(3):273–297, 1995.
140. Paul T. Costa and Robert R. MacCrae. *Revised NEO personality inventory (NEO PI-R) and NEO five-factor inventory (NEO-FFI): Professional manual*. Psychological Assessment Resources, Incorporated, 1992.
141. Rémi Coulom. Efficient selectivity and backup operators in Monte-Carlo tree search. In *International Conference on Computers and Games*, pages 72–83. Springer, 2006.
142. Rémi Coulom. Computing Elo ratings of move patterns in the game of Go. In *Computer Games Workshop*, 2007.
143. Roddy Cowie and Randolph R. Cornelius. Describing the emotional states that are expressed in speech. *Speech Communication*, 40(1):5–32, 2003.
144. Roddy Cowie, Ellen Douglas-Cowie, Susie Savvidou, Edelle McMahon, Martin Sawey, and Marc Schröder. 'FEELTRACE': An instrument for recording perceived emotion in real time. In *ISCA Tutorial and Research Workshop (ITRW) on Speech and Emotion*, 2000.
145. Roddy Cowie and Martin Sawey. GTrace-General trace program from Queen's University, Belfast, 2011.
146. Peter I. Cowling, Edward J. Powley, and Daniel Whitehouse. Information set Monte Carlo tree search. *IEEE Transactions on Computational Intelligence and AI in Games*, 4(2):120–143, 2012.
147. Koby Crammer and Yoram Singer. Pranking with ranking. *Advances in Neural Information Processing Systems*, 14:641–647, 2002.
148. Chris Crawford. *Chris Crawford on interactive storytelling*. New Riders, 2012.
149. Mihaly Csikszentmihalyi. *Creativity: Flow and the psychology of discovery and invention*. New York: Harper Collins, 1996.
150. Mihaly Csikszentmihalyi. *Beyond boredom and anxiety.* Jossey-Bass, 2000.
151. Mihaly Csikszentmihalyi. *Toward a psychology of optimal experience*. Springer, 2014.
152. George Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals and Systems*, 2(4):303–314, 1989.
153. Ryan S. J. d. Baker, Gregory R. Moore, Angela Z. Wagner, Jessica Kalka, Aatish Salvi, Michael Karabinos, Colin A. Ashe, and David Yaron. The Dynamics between Student Affect and Behavior Occurring Outside of Educational Software. In *Affective Computing and Intelligent Interaction*, pages 14–24. Springer, 2011.

154. Anders Dahlbom and Lars Niklasson. Goal-Directed Hierarchical Dynamic Scripting for RTS Games. In *AIIDE*, pages 21–28, 2006.
155. Steve Dahlskog and Julian Togelius. Patterns as objectives for level generation. In *Proceedings of the International Conference on the Foundations of Digital Games*. ACM, 2013.
156. Steve Dahlskog, Julian Togelius, and Mark J. Nelson. Linear levels through n-grams. In *Proceedings of the 18th International Academic MindTrek Conference: Media Business, Management, Content & Services*, pages 200–206. ACM, 2014.
157. Antonio R. Damasio, Barry J. Everitt, and Dorothy Bishop. The somatic marker hypothesis and the possible functions of the prefrontal cortex [and discussion]. *Philosophical Transactions of the Royal Society B: Biological Sciences*, 351(1346):1413–1420, 1996.
158. Gustavo Danzi, Andrade Hugo Pimentel Santana, André Wilson Brotto Furtado, André Roberto Gouveia, Amaral Leitao, and Geber Lisboa Ramalho. Online adaptation of computer games agents: A reinforcement learning approach. In *II Workshop de Jogos e Entretenimento Digital*, pages 105–112, 2003.
159. Isaac M. Dart, Gabriele De Rossi, and Julian Togelius. SpeedRock: procedural rocks through grammars and evolution. In *Proceedings of the 2nd International Workshop on Procedural Content Generation in Games*. ACM, 2011.
160. Fernando de Mesentier Silva, Scott Lee, Julian Togelius, and Andy Nealen. AI-based Playtesting of Contemporary Board Games. In *Proceedings of Foundations of Digital Games (FDG)*, 2017.
161. Maarten de Waard, Diederik M. Roijers, and Sander Bakkes. Monte Carlo tree search with options for general video game playing. In *Computational Intelligence and Games (CIG), 2016 IEEE Conference on*. IEEE, 2016.
162. Edward L. Deci and Richard M. Ryan. *Intrinsic motivation*. Wiley Online Library, 1975.
163. Erik D. Demaine, Giovanni Viglietta, and Aaron Williams. Super Mario Bros. is Harder/Easier than We Thought. In *Proceedings of the 8th International Conference on Fun with Algorithms (FUN 2016)*, pages 13:1–13:14, La Maddalena, Italy, June 8–10 2016.
164. Jack Dennerlein, Theodore Becker, Peter Johnson, Carson Reynolds, and Rosalind W. Picard. Frustrating computer users increases exposure to physical factors. In *Proceedings of the International Ergonomics Association (IEA)*, 2003.
165. Jörg Denzinger, Kevin Loose, Darryl Gates, and John W. Buchanan. Dealing with Parameterized Actions in Behavior Testing of Commercial Computer Games. In *IEEE Symposium on Computational Intelligence and Games*, 2005.
166. L. Devillers, R. Cowie, J. C. Martin, E. Douglas-Cowie, S. Abrilian, and M. McRorie. Real life emotions in French and English TV video clips: an integrated annotation protocol combining continuous and discrete approaches. In *Proceedings of the 5th International Conference on Language Resources and Evaluation (LREC 2006), Genoa, Italy*, page 22, 2006.
167. Ravi Dhar and Itamar Simonson. The effect of forced choice on choice. *Journal of Marketing Research*, 40(2), 2003.
168. Joao Dias, Samuel Mascarenhas, and Ana Paiva. Fatima modular: Towards an agent architecture with a generic appraisal framework. In *Emotion Modeling*, pages 44–56. Springer, 2014.
169. Kevin Dill. A pattern-based approach to modular AI for Games. *Game Programming Gems*, 8:232–243, 2010.
170. Kevin Dill. Introducing GAIA: A Reusable, Extensible architecture for AI behavior. In *Proceedings of the 2012 Spring Simulation Interoperability Workshop*, 2012.
171. Kevin Dill and L. Martin. A game AI approach to autonomous control of virtual characters. In *Interservice/Industry Training, Simulation, and Education Conference (I/ITSEC)*, 2011.
172. Sidney D'Mello and Art Graesser. Automatic detection of learner's affect from gross body language. *Applied Artificial Intelligence*, 23(2):123–150, 2009.
173. Joris Dormans. Adventures in level design: generating missions and spaces for action adventure games. In *Proceedings of the 2010 Workshop on Procedural Content Generation in Games*. ACM, 2010.

174. Joris Dormans and Sander Bakkes. Generating missions and spaces for adaptable play experiences. *IEEE Transactions on Computational Intelligence and AI in Games*, 3(3):216–228, 2011.
175. Aanders Drachen, Lennart Nacke, Georgios N. Yannakakis, and Anja Lee Pedersen. Correlation between heart rate, electrodermal activity and player experience in first-person shooter games. In *Proceedings of the SIGGRAPH Symposium on Video Games*. ACM-SIGGRAPH Publishers, 2010.
176. Anders Drachen, Alessandro Canossa, and Georgios N. Yannakakis. Player modeling using self-organization in Tomb Raider: Underworld. In *Proceedings of the 2009 IEEE Symposium on Computational Intelligence and Games*, pages 1–8. IEEE, 2009.
177. Anders Drachen and Matthias Schubert. Spatial game analytics. In *Game Analytics*, pages 365–402. Springer, 2013.
178. Anders Drachen, Christian Thurau, Julian Togelius, Georgios N. Yannakakis, and Christian Bauckhage. Game Data Mining. In *Game Analytics*, pages 205–253. Springer, 2013.
179. H. Drucker, C.J. C. Burges, L. Kaufman, A. Smola, and V. Vapnik. Support vector regression machines. In *Advances in Neural Information Processing Systems (NIPS)*, pages 155–161. Morgan Kaufmann Publishers, 1997.
180. David S. Ebert. *Texturing & modeling: a procedural approach*. Morgan Kaufmann, 2003.
181. Marc Ebner, John Levine, Simon M. Lucas, Tom Schaul, Tommy Thompson, and Julian Togelius. Towards a video game description language. *Dagstuhl Follow-Ups*, 6, 2013.
182. Arthur S. Eddington. The Constants of Nature. In *The World of Mathematics 2*, pages 1074–1093. Simon & Schuster, 1956.
183. Arjan Egges, Sumedha Kshirsagar, and Nadia Magnenat-Thalmann. Generic personality and emotion simulation for conversational agents. *Computer animation and virtual worlds*, 15(1):1–13, 2004.
184. Agoston E. Eiben and James E. Smith. *Introduction to Evolutionary Computing*. Springer, 2003.
185. Magy Seif El-Nasr. Intelligent lighting for game environments. *Journal of Game Development*, 2005.
186. Magy Seif El-Nasr, Anders Drachen, and Alessandro Canossa. *Game analytics: Maximizing the value of player data*. Springer, 2013.
187. Magy Seif El-Nasr, Shree Durga, Mariya Shiyko, and Carmen Sceppa. Data-driven retrospective interviewing (DDRI): a proposed methodology for formative evaluation of pervasive games. *Entertainment Computing*, 11:1–19, 2015.
188. Magy Seif El-Nasr, Athanasios Vasilakos, Chinmay Rao, and Joseph Zupko. Dynamic intelligent lighting for directing visual attention in interactive 3-D scenes. *Computational Intelligence and AI in Games, IEEE Transactions on*, 1(2):145–153, 2009.
189. Magy Seif El-Nasr, John Yen, and Thomas R. Ioerger. Flame—fuzzy logic adaptive model of emotions. *Autonomous Agents and Multi-Agent Systems*, 3(3):219–257, 2000.
190. Mirjam Palosaari Eladhari and Michael Mateas. Semi-autonomous avatars in World of Minds: A case study of AI-based game design. In *Proceedings of the 2008 International Conference on Advances in Computer Entertainment Technology*, pages 201–208. ACM, 2008.
191. Mirjam Palosaari Eladhari and Michael Sellers. Good moods: outlook, affect and mood in dynemotion and the mind module. In *Proceedings of the 2008 Conference on Future Play: Research, Play, Share*, pages 1–8. ACM, 2008.
192. George Skaff Elias, Richard Garfield, K. Robert Gutschera, and Peter Whitley. *Characteristics of games*. MIT Press, 2012.
193. David K. Elson and Mark O. Riedl. A lightweight intelligent virtual cinematography system for machinima production. In *AIIDE*, pages 8–13, 2007.
194. Nathan Ensmenger. Is Chess the Drosophila of AI? A Social History of an Algorithm. *Social Studies of Science*, 42(1):5–30, 2012.
195. Ido Erev and Alvin E. Roth. Predicting how people play games: Reinforcement learning in experimental games with unique, mixed strategy equilibria. *American Economic Review*, pages 848–881, 1998.

196. Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proceedings of the International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 226–231, 1996.
197. Richard Evans and Emily Short. Versu—a simulationist storytelling system. *IEEE Transactions on Computational Intelligence and AI in Games*, 6(2):113–130, 2014.
198. Vincent E. Farrugia, Héctor P. Martínez, and Georgios N. Yannakakis. The preference learning toolbox. *arXiv preprint arXiv:1506.01709*, 2015.
199. Bjarke Felbo, Alan Mislove, Anders Søgaard, Iyad Rahwan, and Sune Lehmann. Using millions of emoji occurrences to learn any-domain representations for detecting sentiment, emotion and sarcasm. *arXiv preprint arXiv:1708.00524*, 2017.
200. Lisa A. Feldman. Valence focus and arousal focus: Individual differences in the structure of affective experience. *Journal of personality and social psychology*, 69(1):153, 1995.
201. David Ferrucci, Eric Brown, Jennifer Chu-Carroll, James Fan, David Gondek, Aditya A. Kalyanpur, Adam Lally, J. William Murdock, Eric Nyberg, John Prager, Nico Schlaefer, and Chris Welty. Building Watson: An overview of the DeepQA project. *AI Magazine*, 31(3):59–79, 2010.
202. Hilmar Finnsson and Yngvi Björnsson. Learning simulation control in general game-playing agents. In *AAAI*, pages 954–959, 2010.
203. Jacob Fischer, Nikolaj Falsted, Mathias Vielwerth, Julian Togelius, and Sebastian Risi. Monte-Carlo Tree Search for Simulated Car Racing. In *Proceedings of FDG*, 2015.
204. John H. Flavell. *The developmental psychology of Jean Piaget*. Ardent Media, 1963.
205. Dario Floreano, Peter Dürr, and Claudio Mattiussi. Neuroevolution: from architectures to learning. *Evolutionary Intelligence*, 1(1):47–62, 2008.
206. Dario Floreano, Toshifumi Kato, Davide Marocco, and Eric Sauser. Coevolution of active vision and feature selection. *Biological Cybernetics*, 90(3):218–228, 2004.
207. David B. Fogel. *Blondie24: Playing at the Edge of AI*. Morgan Kaufmann, 2001.
208. David B. Fogel, Timothy J. Hays, Sarah L. Hahn, and James Quon. The Blondie25 chess program competes against Fritz 8.0 and a human chess master. In *Computational Intelligence and Games, 2006 IEEE Symposium on*, pages 230–235. IEEE, 2006.
209. Tom Forsyth. Cellular automata for physical modelling. *Game Programming Gems*, 3:200–214, 2002.
210. Alain Fournier, Don Fussell, and Loren Carpenter. Computer rendering of stochastic models. *Communications of the ACM*, 25(6):371–384, 1982.
211. Michael Freed, Travis Bear, Herrick Goldman, Geoffrey Hyatt, Paul Reber, A. Sylvan, and Joshua Tauber. Towards more human-like computer opponents. In *Working Notes of the AAAI Spring Symposium on Artificial Intelligence and Interactive Entertainment*, pages 22–26, 2000.
212. Nico Frijda. *The Emotions*. Cambridge University Press, Englewood Cliffs, NJ, 1986.
213. Frederik Frydenberg, Kasper R. Andersen, Sebastian Risi, and Julian Togelius. Investigating MCTS modifications in general video game playing. In *Computational Intelligence and Games (CIG), 2015 IEEE Conference on*, pages 107–113. IEEE, 2015.
214. Drew Fudenberg and David K. Levine. *The theory of learning in games*. MIT Press, 1998.
215. J. Fürnkranz and E. Hüllermeier. *Preference learning*. Springer, 2010.
216. Raluca D. Gaina, Jialin Liu, Simon M. Lucas, and Diego Pérez-Liébana. Analysis of Vanilla Rolling Horizon Evolution Parameters in General Video Game Playing. In *European Conference on the Applications of Evolutionary Computation*, pages 418–434. Springer, 2017.
217. Maurizio Garbarino, Simone Tognetti, Matteo Matteucci, and Andrea Bonarini. Learning general preference models from physiological responses in video games: How complex is it? In *Affective Computing and Intelligent Interaction*, pages 517–526. Springer, 2011.
218. Pablo García-Sánchez, Alberto Tonda, Giovanni Squillero, Antonio Mora, and Juan J. Merelo. Evolutionary deckbuilding in Hearthstone. In *Computational Intelligence and Games (CIG), 2016 IEEE Conference on*. IEEE, 2016.
219. Tom A. Garner. From Sinewaves to Physiologically-Adaptive Soundscapes: The Evolving Relationship Between Sound and Emotion in Video Games. In *Emotion in Games: Theory and Praxis*, pages 197–214. Springer, 2016.

220. Tom A. Garner and Mark Grimshaw. Sonic virtuality: Understanding audio in a virtual world. *The Oxford Handbook of Virtuality*, 2014.
221. H. P. Gasselseder. Re-scoring the games score: Dynamic music and immersion in the ludonarrative. In *Proceedings of the Intelligent Human Computer Interaction conference*, 2014.
222. Jakub Gemrot, Rudolf Kadlec, Michal Bída, Ondřej Burkert, Radek Píbil, Jan Havlíček, Lukáš Zemčák, Juraj Šimlovič, Radim Vansa, Michal Štolba, Tomáš Plch, and Cyril Brom. Pogamut 3 can assist developers in building AI (not only) for their videogame agents. In *Agents for games and simulations*, pages 1–15. Springer, 2009.
223. Michael Genesereth, Nathaniel Love, and Barney Pell. General game playing: Overview of the AAAI competition. *AI Magazine*, 26(2):62, 2005.
224. Michael Georgeff, Barney Pell, Martha Pollack, Milind Tambe, and Michael Wooldridge. The belief-desire-intention model of agency. In *International Workshop on Agent Theories, Architectures, and Languages*, pages 1–10. Springer, 1998.
225. Kallirroi Georgila, James Henderson, and Oliver Lemon. Learning user simulations for information state update dialogue systems. In *Interspeech*, pages 893–896, 2005.
226. Panayiotis G. Georgiou, Matthew P. Black, Adam C. Lammert, Brian R. Baucom, and Shrikanth S. Narayanan. "That's Aggravating, Very Aggravating": Is It Possible to Classify Behaviors in Couple Interactions Using Automatically Derived Lexical Features? In *Affective Computing and Intelligent Interaction*, pages 87–96. Springer, 2011.
227. Maryrose Gerardi, Barbara Olasov Rothbaum, Kerry Ressler, Mary Heekin, and Albert Rizzo. Virtual reality exposure therapy using a virtual Iraq: case report. *Journal of Traumatic Stress*, 21(2):209–213, 2008.
228. Malik Ghallab, Dana Nau, and Paolo Traverso. *Automated Planning: theory and practice*. Elsevier, 2004.
229. Spyridon Giannatos, Yun-Gyung Cheong, Mark J. Nelson, and Georgios N. Yannakakis. Generating narrative action schemas for suspense. In *Eighth Artificial Intelligence and Interactive Digital Entertainment Conference*, 2012.
230. Arthur Gill. *Introduction to the theory of Finite-State Machines*. McGraw-Hill, 1962.
231. Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016.
232. Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in Neural Information Processing Systems*, pages 2672–2680, 2014.
233. Nitesh Goyal, Gilly Leshed, Dan Cosley, and Susan R. Fussell. Effects of implicit sharing in collaborative analysis. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 129–138, 2014.
234. Katja Grace, John Salvatier, Allan Dafoe, Baobao Zhang, and Owain Evans. When Will AI Exceed Human Performance? Evidence from AI Experts. *arXiv preprint arXiv:1705.08807*, 2017.
235. Thore Graepel, Ralf Herbrich, and Julian Gold. Learning to fight. In *Proceedings of the International Conference on Computer Games: Artificial Intelligence, Design and Education*, pages 193–200, 2004.
236. Joseph F. Grafsgaard, Kristy Elizabeth Boyer, and James C. Lester. Predicting facial indicators of confusion with hidden Markov models. In *Proceedings of International Conference on Affective Computing and Intelligent Interaction (ACII)*, pages 97–106. Springer, 2011.
237. Jonathan Gratch. Emile: Marshalling passions in training and education. In *Proceedings of the Fourth International Conference on Autonomous Agents*, pages 325–332. ACM, 2000.
238. Jonathan Gratch and Stacy Marsella. A domain-independent framework for modeling emotion. *Cognitive Systems Research*, 5(4):269–306, 2004.
239. Jonathan Gratch and Stacy Marsella. Evaluating a computational model of emotion. *Autonomous Agents and Multi-Agent Systems*, 11(1):23–43, 2005.
240. Daniele Gravina, Antonios Liapis, and Georgios N. Yannakakis. Constrained surprise search for content generation. In *Computational Intelligence and Games (CIG), 2016 IEEE Conference on*. IEEE, 2016.

241. Elin Rut Gudnadottir, Alaina K. Jensen, Yun-Gyung Cheong, Julian Togelius, Byung Chull Bae, and Christoffer Holmgård Pedersen. Detecting predatory behaviour in online game chats. In *The 2nd Workshop on Games and NLP*, 2014.
242. Johan Hagelbck. Potential-field based navigation in StarCraft. In *IEEE Conference on Computational Intelligence and Games (CIG)*. IEEE, 2012.
243. Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H. Witten. The WEKA data mining software: an update. *ACM SIGKDD explorations newsletter*, 11(1):10–18, 2009.
244. Jiawei Han and Micheline Kamber. *Data mining: concepts and techniques*. Morgan Kaufmann, 2006.
245. Nikolaus Hansen and Andreas Ostermeier. Completely derandomized self-adaptation in evolution strategies. *Evolutionary Computation*, 9(2):159–195, 2001.
246. Daniel Damir Harabor and Alban Grastien. Online Graph Pruning for Pathfinding on Grid Maps. In *AAAI*, 2011.
247. Peter E. Hart, Nils J. Nilsson, and Bertram Raphael. Correction to a formal basis for the heuristic determination of minimum cost paths. *ACM SIGART Bulletin*, (37):28–29, 1972.
248. Ken Hartsook, Alexander Zook, Sauvik Das, and Mark O. Riedl. Toward supporting stories with procedurally generated game worlds. In *Computational Intelligence and Games (CIG), 2011 IEEE Conference on*, pages 297–304. IEEE, 2011.
249. Erin J. Hastings, Ratan K. Guha, and Kenneth O. Stanley. Automatic content generation in the Galactic Arms Race video game. *IEEE Transactions on Computational Intelligence and AI in Games*, 1(4):245–263, 2009.
250. Erin J. Hastings, Ratan K. Guha, and Kenneth O. Stanley. Evolving content in the Galactic Arms Race video game. In *IEEE Symposium on Computational Intelligence and Games*, pages 241–248. IEEE, 2009.
251. Matthew Hausknecht, Joel Lehman, Risto Miikkulainen, and Peter Stone. A neuroevolution approach to general Atari game playing. *IEEE Transactions on Computational Intelligence and AI in Games*, 6(4):355–366, 2014.
252. Brian Hawkins. *Real-Time Cinematography for Games (Game Development Series)*. Charles River Media, Inc., 2004.
253. Simon Haykin. *Neural Networks: A Comprehensive Foundation*. Macmillian College Publishing Company Inc., Upper Saddle River, NJ, USA, 1998.
254. Richard L. Hazlett. Measuring emotional valence during interactive experiences: boys at video game play. In *Proceedings of SIGCHI Conference on Human Factors in Computing Systems (CHI)*, pages 1023–1026. ACM, 2006.
255. Jennifer Healey. Recording affect in the field: Towards methods and metrics for improving ground truth labels. In *Affective Computing and Intelligent Interaction*, pages 107–116. Springer, 2011.
256. D. O. Hebb. *The Organization of Behavior*. Wiley, New York, 1949.
257. Norbert Heijne and Sander Bakkes. Procedural Zelda: A PCG Environment for Player Experience Research. In *Proceedings of the International Conference on the Foundations of Digital Games*. ACM, 2017.
258. Harry Helson. *Adaptation-level theory*. Harper & Row, 1964.
259. Ralf Herbrich, Michael E. Tipping, and Mark Hatton. Personalized behavior of computer controlled avatars in a virtual reality environment, August 15 2006. US Patent 7,090,576.
260. Javier Hernandez, Rob R. Morris, and Rosalind W. Picard. Call center stress recognition with person-specific models. In *Affective Computing and Intelligent Interaction*, pages 125–134. Springer, 2011.
261. David Hilbert. Über die stetige Abbildung einer Linie auf ein Flächenstück. *Mathematische Annalen*, 38(3):459–460, 1891.
262. Philip Hingston. A Turing test for computer game bots. *IEEE Transactions on Computational Intelligence and AI in Games*, 1(3):169–186, 2009.
263. Philip Hingston. A new design for a Turing test for bots. In *Computational Intelligence and Games (CIG), 2010 IEEE Symposium on*, pages 345–350. IEEE, 2010.

264. Philip Hingston. *Believable Bots: Can Computers Play Like People?* Springer, 2012.
265. Philip Hingston, Clare Bates Congdon, and Graham Kendall. Mobile games with intelligence: A killer application? In *Computational Intelligence in Games (CIG), 2013 IEEE Conference on*, pages 1–7. IEEE, 2013.
266. Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.
267. Christoffer Holmgård, Antonios Liapis, Julian Togelius, and Georgios N. Yannakakis. Evolving personas for player decision modeling. In *Computational Intelligence and Games (CIG), 2014 IEEE Conference on*. IEEE, 2014.
268. Christoffer Holmgård, Antonios Liapis, Julian Togelius, and Georgios N. Yannakakis. Generative agents for player decision modeling in games. In *FDG*, 2014.
269. Christoffer Holmgård, Antonios Liapis, Julian Togelius, and Georgios N. Yannakakis. Personas versus clones for player decision modeling. In *International Conference on Entertainment Computing*, pages 159–166. Springer, 2014.
270. Christoffer Holmgård, Georgios N. Yannakakis, Karen-Inge Karstoft, and Henrik Steen Andersen. Stress detection for PTSD via the Startlemart game. In *Affective Computing and Intelligent Interaction (ACII), 2013 Humaine Association Conference on*, pages 523–528. IEEE, 2013.
271. Christoffer Holmgård, Georgios N. Yannakakis, Héctor P. Martínez, and Karen-Inge Karstoft. To rank or to classify? Annotating stress for reliable PTSD profiling. In *Affective Computing and Intelligent Interaction (ACII), 2015 International Conference on*, pages 719–725. IEEE, 2015.
272. Christoffer Holmgård, Georgios N. Yannakakis, Héctor P. Martínez, Karen-Inge Karstoft, and Henrik Steen Andersen. Multimodal PTSD characterization via the Startlemart game. *Journal on Multimodal User Interfaces*, 9(1):3–15, 2015.
273. Nils Iver Holtar, Mark J. Nelson, and Julian Togelius. Audioverdrive: Exploring bidirectional communication between music and gameplay. In *Proceedings of the 2013 International Computer Music Conference*, pages 124–131, 2013.
274. Vincent Hom and Joe Marks. Automatic design of balanced board games. In *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE)*, pages 25–30, 2007.
275. Amy K. Hoover, William Cachia, Antonios Liapis, and Georgios N. Yannakakis. AudioInSpace: Exploring the Creative Fusion of Generative Audio, Visuals and Gameplay. In *Evolutionary and Biologically Inspired Music, Sound, Art and Design*, pages 101–112. Springer, 2015.
276. Amy K. Hoover, Paul A. Szerlip, and Kenneth O. Stanley. Functional scaffolding for composing additional musical voices. *Computer Music Journal*, 2014.
277. Amy K. Hoover, Julian Togelius, and Georgios N. Yannakakis. Composing video game levels with music metaphors through functional scaffolding. In *First Computational Creativity and Games Workshop, ICCC*, 2015.
278. John J. Hopfield. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the National Academy of Sciences*, 79(8):2554–2558, 1982.
279. Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359–366, 1989.
280. Ben Houge. Cell-based music organization in Tom Clancy's EndWar. In *Demo at the AIIDE 2012 Workshop on Musical Metacreation*, 2012.
281. Ryan Houlette. *Player Modeling for Adaptive Games. AI Game Programming Wisdom II*, pages 557–566. Charles River Media, Inc., 2004.
282. Andrew Howlett, Simon Colton, and Cameron Browne. Evolving pixel shaders for the prototype video game Subversion. In *The Thirty Sixth Annual Convention of the Society for the Study of Artificial Intelligence and Simulation of Behaviour (AISB10), De Montfort University, Leicester, UK, 30th March*, 2010.
283. Johanna Höysniemi, Perttu Hämäläinen, Laura Turkki, and Teppo Rouvi. Children's intuitive gestures in vision-based action games. *Communications of the ACM*, 48(1):44–50, 2005.

284. Chih-Wei Hsu and Chih-Jen Lin. A comparison of methods for multiclass support vector machines. *IEEE Transactions on Neural Networks*, 13(2):415–425, 2002.
285. Feng-Hsiung Hsu. *Behind Deep Blue: Building the computer that defeated the world chess champion*. Princeton University Press, 2002.
286. Wijnand IJsselsteijn, Karolien Poels, and Y. A. W. De Kort. The game experience questionnaire: Development of a self-report measure to assess player experiences of digital games. *TU Eindhoven, Eindhoven, The Netherlands*, 2008.
287. Interactive Data Visualization. SpeedTree, 2010. http://www.speedtree.com/.
288. Aaron Isaksen, Dan Gopstein, Julian Togelius, and Andy Nealen. Discovering unique game variants. In *Computational Creativity and Games Workshop at the 2015 International Conference on Computational Creativity*, 2015.
289. Aaron Isaksen, Daniel Gopstein, and Andrew Nealen. Exploring Game Space Using Survival Analysis. In *Proceedings of Foundations of Digital Games (FDG)*, 2015.
290. Katherine Isbister and Noah Schaffer. *Game usability: Advancing the player experience*. CRC Press, 2015.
291. Damian Isla. Handling complexity in the Halo 2 AI. In *Game Developers Conference*, 2005.
292. Damian Isla and Bruce Blumberg. New challenges for character-based AI for games. In *Proceedings of the AAAI Spring Symposium on AI and Interactive Entertainment*, pages 41–45. AAAI Press, 2002.
293. Susan A. Jackson and Robert C. Eklund. Assessing flow in physical activity: the flow state scale-2 and dispositional flow scale-2. *Journal of Sport & Exercise Psychology*, 24(2), 2002.
294. Emil Juul Jacobsen, Rasmus Greve, and Julian Togelius. Monte Mario: platforming with MCTS. In *Proceedings of the 2014 Annual Conference on Genetic and Evolutionary Computation*, pages 293–300. ACM, 2014.
295. Alexander Jaffe, Alex Miller, Erik Andersen, Yun-En Liu, Anna Karlin, and Zoran Popovic. Evaluating competitive game balance with restricted play. In *AIIDE*, 2012.
296. Rishabh Jain, Aaron Isaksen, Christoffer Holmgård, and Julian Togelius. Autoencoders for level generation, repair, and recognition. In *ICCC Workshop on Computational Creativity and Games*, 2016.
297. Daniel Jallov, Sebastian Risi, and Julian Togelius. EvoCommander: A Novel Game Based on Evolving and Switching Between Artificial Brains. *IEEE Transactions on Computational Intelligence and AI in Games*, 9(2):181–191, 2017.
298. Susan Jamieson. Likert scales: how to (ab) use them. *Medical Education*, 38(12):1217–1218, 2004.
299. Aki Järvinen. Gran stylissimo: The audiovisual elements and styles in computer and video games. In *Proceedings of Computer Games and Digital Cultures Conference*, 2002.
300. Arnav Jhala and R. Michael Young. Cinematic visual discourse: Representation, generation, and evaluation. *Computational Intelligence and AI in Games, IEEE Transactions on*, 2(2):69–81, 2010.
301. Yuu Jinnai and Alex S. Fukunaga. Learning to prune dominated action sequences in online black-box planning. In *AAAI*, pages 839–845, 2017.
302. Thorsten Joachims. Text categorization with support vector machines: Learning with many relevant features. *Machine Learning: ECML-98*, pages 137–142, 1998.
303. Thorsten Joachims. Optimizing search engines using clickthrough data. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery in Data Mining (KDD)*, pages 133–142. ACM, 2002.
304. Lawrence Johnson, Georgios N. Yannakakis, and Julian Togelius. Cellular automata for real-time generation of infinite cave levels. In *Proceedings of the 2010 Workshop on Procedural Content Generation in Games*. ACM, 2010.
305. Matthew Johnson, Katja Hofmann, Tim Hutton, and David Bignell. The Malmo Platform for Artificial Intelligence Experimentation. In *IJCAI*, pages 4246–4247, 2016.
306. Tom Johnstone and Klaus R. Scherer. Vocal communication of emotion. In *Handbook of emotions*, pages 220–235. Guilford Press, New York, 2000.

307. German Gutierrez Jorge Munoz and Araceli Sanchis. Towards imitation of human driving style in car racing games. In Philip Hingston, editor, *Believable Bots: Can Computers Play Like People?* Springer, 2012.
308. Patrik N. Juslin and Klaus R. Scherer. *Vocal expression of affect*. Oxford University Press, Oxford, UK, 2005.
309. Niels Justesen, Tobias Mahlmann, and Julian Togelius. Online evolution for multi-action adversarial games. In *European Conference on the Applications of Evolutionary Computation*, pages 590–603. Springer, 2016.
310. Niels Justesen and Sebastian Risi. Continual Online Evolutionary Planning for In-Game Build Order Adaptation in StarCraft. In *Proceedings of the Conference on Genetic and Evolutionary Computation (GECCO)*, 2017.
311. Niels Justesen, Bálint Tillman, Julian Togelius, and Sebastian Risi. Script-and cluster-based UCT for StarCraft. In *Computational Intelligence and Games (CIG), 2014 IEEE Conference on*. IEEE, 2014.
312. Tróndur Justinussen, Peter Hald Rasmussen, Alessandro Canossa, and Julian Togelius. Resource systems in games: An analytical approach. In *Computational Intelligence and Games (CIG), 2012 IEEE Conference on*, pages 171–178. IEEE, 2012.
313. Jesper Juul. Games telling stories. *Game Studies*, 1(1):45, 2001.
314. Jesper Juul. *A casual revolution: Reinventing video games and their players*. MIT Press, 2010.
315. Souhila Kaci. *Working with preferences: Less is more*. Springer, 2011.
316. Leslie Pack Kaelbling, Michael L. Littman, and Andrew W. Moore. Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, 4:237–285, 1996.
317. Daniel Kahneman. A perspective on judgment and choice: mapping bounded rationality. *American psychologist*, 58(9):697, 2003.
318. Daniel Kahneman and Jason Riis. Living, and thinking about it: Two perspectives on life. *The science of well-being*, pages 285–304, 2005.
319. Theofanis Kannetis and Alexandros Potamianos. Towards adapting fantasy, curiosity and challenge in multimodal dialogue systems for preschoolers. In *Proceedings of International Conference on Multimodal Interfaces (ICMI)*, pages 39–46. ACM, 2009.
320. Theofanis Kannetis, Alexandros Potamianos, and Georgios N. Yannakakis. Fantasy, curiosity and challenge as adaptation indicators in multimodal dialogue systems for preschoolers. In *Proceedings of the 2nd Workshop on Child, Computer and Interaction*. ACM, 2009.
321. Ashish Kapoor, Winslow Burleson, and Rosalind W. Picard. Automatic prediction of frustration. *International Journal of Human-Computer Studies*, 65(8):724–736, 2007.
322. Sergey Karakovskiy and Julian Togelius. The Mario AI benchmark and competitions. *IEEE Transactions on Computational Intelligence and AI in Games*, 4(1):55–67, 2012.
323. Daniël Karavolos, Anders Bouwer, and Rafael Bidarra. Mixed-initiative design of game levels: Integrating mission and space into level generation. In *Proceedings of the 10th International Conference on the Foundations of Digital Games*, 2015.
324. Daniel Karavolos, Antonios Liapis, and Georgios N. Yannakakis. Learning the patterns of balance in a multi-player shooter game. In *Proceedings of the FDG workshop on Procedural Content Generation in Games*, 2017.
325. Kostas Karpouzis and Georgios N. Yannakakis. *Emotion in Games: Theory and Praxis*. Springer, 2016.
326. Kostas Karpouzis, Georgios N. Yannakakis, Noor Shaker, and Stylianos Asteriadis. The Platformer Experience Dataset. In *Affective Computing and Intelligent Interaction (ACII), 2015 International Conference on*, pages 712–718. IEEE, 2015.
327. Igor V. Karpov, Leif Johnson, and Risto Miikkulainen. Evaluation methods for active human-guided neuroevolution in games. In *2012 AAAI Fall Symposium on Robots Learning Interactively from Human Teachers (RLIHT)*, 2012.
328. Igor V. Karpov, Jacob Schrum, and Risto Miikkulainen. Believable bot navigation via playback of human traces. In Philip Hingston, editor, *Believable Bots: Can Computers Play Like People?* Springer, 2012.

329. Leonard Kaufman and Peter J. Rousseeuw. *Clustering by means of medoids*. North-Holland, 1987.
330. Leonard Kaufman and Peter J. Rousseeuw. *Finding groups in data: an introduction to cluster analysis*. John Wiley & Sons, 2009.
331. Richard Kaye. Minesweeper is NP-complete. *The Mathematical Intelligencer*, 22(2):9–15, 2000.
332. Markus Kemmerling and Mike Preuss. Automatic adaptation to generated content via car setup optimization in TORCS. In *Computational Intelligence and Games (CIG), 2010 IEEE Symposium on*, pages 131–138. IEEE, 2010.
333. Michał Kempka, Marek Wydmuch, Grzegorz Runc, Jakub Toczek, and Wojciech Jaśkowski. Vizdoom: A doom-based AI research platform for visual reinforcement learning. *arXiv preprint arXiv:1605.02097*, 2016.
334. Graham Kendall, Andrew J. Parkes, and Kristian Spoerer. A Survey of NP-Complete Puzzles. *ICGA Journal*, 31(1):13–34, 2008.
335. Manuel Kerssemakers, Jeppe Tuxen, Julian Togelius, and Georgios N. Yannakakis. A procedural procedural level generator generator. In *Computational Intelligence and Games (CIG), 2012 IEEE Conference on*, pages 335–341. IEEE, 2012.
336. Rilla Khaled and Georgios N. Yannakakis. Village voices: An adaptive game for conflict resolution. In *Proceedings of FDG*, pages 425–426, 2013.
337. Ahmed Khalifa, Aaron Isaksen, Julian Togelius, and Andy Nealen. Modifying MCTS for Human-like General Video Game Playing. In *Proceedings of IJCAI*, 2016.
338. Ahmed Khalifa, Diego Perez-Liebana, Simon M. Lucas, and Julian Togelius. General video game level generation. In *Proceedings of IJCAI*, 2016.
339. K-J Kim, Heejin Choi, and Sung-Bae Cho. Hybrid of evolution and reinforcement learning for Othello players. In *Computational Intelligence and Games, 2007. CIG 2007. IEEE Symposium on*, pages 203–209. IEEE, 2007.
340. Kyung-Min Kim, Chang-Jun Nan, Jung-Woo Ha, Yu-Jung Heo, and Byoung-Tak Zhang. Pororobot: A deep learning robot that plays video Q&A games. In *AAAI 2015 Fall Symposium on AI for Human-Robot Interaction (AI-HRI 2015)*, 2015.
341. Steven Orla Kimbrough, Gary J. Koehler, Ming Lu, and David Harlan Wood. On a Feasible–Infeasible Two-Population (FI-2Pop) genetic algorithm for constrained optimization: Distance tracing and no free lunch. *European Journal of Operational Research*, 190(2):310–327, 2008.
342. Diederik P. Kingma and Max Welling. Auto-encoding variational Bayes. *arXiv preprint arXiv:1312.6114*, 2013.
343. A. Kleinsmith and N. Bianchi-Berthouze. Affective body expression perception and recognition: A survey. *IEEE Transactions on Affective Computing*, 2012.
344. Andrea Kleinsmith and Nadia Bianchi-Berthouze. Form as a cue in the automatic recognition of non-acted affective body expressions. In *Affective Computing and Intelligent Interaction*, pages 155–164. Springer, 2011.
345. Yana Knight, Héctor Perez Martínez, and Georgios N. Yannakakis. Space maze: Experience-driven game camera control. In *FDG*, pages 427–428, 2013.
346. Matthias J. Koepp, Roger N. Gunn, Andrew D. Lawrence, Vincent J. Cunningham, Alain Dagher, Tasmin Jones, David J. Brooks, C. J. Bench, and P. M. Grasby. Evidence for striatal dopamine release during a video game. *Nature*, 393(6682):266–268, 1998.
347. Teuvo Kohonen. *Self-Organizing Maps*. Springer, Secaucus, NJ, USA, 3rd edition, 2001.
348. Andrey N. Kolmogorov. On the representation of continuous functions of several variables by superposition of continuous functions of one variable and addition. *Russian, American Mathematical Society Translation 28 (1963) 55-59. Doklady Akademiia Nauk SSR*, 14(5):953–956, 1957.
349. Richard Konečnỳ. Modeling of fighting game players. Master's thesis, Institute of Digital Games, University of Malta, 2016.
350. Michael Kosfeld, Markus Heinrichs, Paul J. Zak, Urs Fischbacher, and Ernst Fehr. Oxytocin increases trust in humans. *Nature*, 435(7042):673–676, 2005.

351. Raph Koster. *Theory of fun for game design*. O'Reilly Media, Inc., 2013.
352. Bartosz Kostka, Jaroslaw Kwiecien, Jakub Kowalski, and Pawel Rychlikowski. Text-based Adventures of the Golovin AI Agent. *arXiv preprint arXiv:1705.05637*, 2017.
353. Jan Koutník, Giuseppe Cuccu, Jürgen Schmidhuber, and Faustino Gomez. Evolving large-scale neural networks for vision-based reinforcement learning. In *Proceedings of the 15th Annual Conference on Genetic and Evolutionary Computation*, pages 1061–1068. ACM, 2013.
354. Jakub Kowalski and Andrzej Kisielewicz. Towards a Real-time Game Description Language. In *ICAART (2)*, pages 494–499, 2016.
355. Jakub Kowalski and Marek Szykuła. Evolving chess-like games using relative algorithm performance profiles. In *European Conference on the Applications of Evolutionary Computation*, pages 574–589. Springer, 2016.
356. John R. Koza. *Genetic programming: on the programming of computers by means of natural selection*. MIT Press, 1992.
357. Teofebano Kristo and Nur Ulfa Maulidevi. Deduction of fighting game countermeasures using Neuroevolution of Augmenting Topologies. In *Data and Software Engineering (ICoDSE), 2016 International Conference on*. IEEE, 2016.
358. Ben Kybartas and Rafael Bidarra. A semantic foundation for mixed-initiative computational storytelling. In *Interactive Storytelling*, pages 162–169. Springer, 2015.
359. Alexandros Labrinidis and Hosagrahar V. Jagadish. Challenges and opportunities with big data. *Proceedings of the VLDB Endowment*, 5(12):2032–2033, 2012.
360. John Laird and Michael van Lent. Human-level AI's killer application: Interactive computer games. *AI Magazine*, 22(2):15, 2001.
361. G. B. Langley and H. Sheppeard. The visual analogue scale: its use in pain measurement. *Rheumatology International*, 5(4):145–148, 1985.
362. Frank Lantz, Aaron Isaksen, Alexander Jaffe, Andy Nealen, and Julian Togelius. Depth in strategic games. In *Proceedings of the AAAI WNAIG Workshop*, 2017.
363. Pier Luca Lanzi, Wolfgang Stolzmann, and Stewart W. Wilson. *Learning classifier systems: from foundations to applications*. Springer, 2003.
364. Richard S. Lazarus. *Emotion and adaptation*. Oxford University Press, 1991.
365. Nicole Lazzaro. Why we play games: Four keys to more emotion without story. Technical report, XEO Design Inc., 2004.
366. Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015.
367. David Lee and Mihalis Yannakakis. Principles and methods of testing finite state machines—a survey. *Proceedings of the IEEE*, 84(8):1090–1123, 1996.
368. Alan Levinovitz. The mystery of Go, the ancient game that computers still can't win. *Wired Magazine*, 2014.
369. Mike Lewis and Kevin Dill. Game AI appreciation, revisited. In *Game AI Pro 2: Collected Wisdom of Game AI Professionals*, pages 3–18. AK Peters/CRC Press, 2015.
370. Boyang Li, Stephen Lee-Urban, Darren Scott Appling, and Mark O. Riedl. Crowdsourcing narrative intelligence. *Advances in Cognitive Systems*, 2(1), 2012.
371. Antonios Liapis. Creativity facet orchestration: the whys and the hows. *Artificial and Computational Intelligence in Games: Integration; Dagstuhl Follow-Ups*, 2015.
372. Antonios Liapis. Mixed-initiative Creative Drawing with webIconoscope. In *Proceedings of the 6th International Conference on Computational Intelligence in Music, Sound, Art and Design. (EvoMusArt)*. Springer, 2017.
373. Antonios Liapis, Héctor P. Martınez, Julian Togelius, and Georgios N. Yannakakis. Transforming exploratory creativity with DeLeNoX. In *Proceedings of the Fourth International Conference on Computational Creativity*, pages 56–63, 2013.
374. Antonios Liapis, Gillian Smith, and Noor Shaker. Mixed-initiative content creation. In *Procedural Content Generation in Games*, pages 195–214. Springer, 2016.
375. Antonios Liapis and Georgios N. Yannakakis. Boosting computational creativity with human interaction in mixed-initiative co-creation tasks. In *Proceedings of the ICCC Workshop on Computational Creativity and Games*, 2016.

376. Antonios Liapis, Georgios N. Yannakakis, and Julian Togelius. Neuroevolutionary constrained optimization for content creation. In *Computational Intelligence and Games (CIG), 2011 IEEE Conference on*, pages 71–78. IEEE, 2011.
377. Antonios Liapis, Georgios N. Yannakakis, and Julian Togelius. Adapting models of visual aesthetics for personalized content creation. *IEEE Transactions on Computational Intelligence and AI in Games*, 4(3):213–228, 2012.
378. Antonios Liapis, Georgios N. Yannakakis, and Julian Togelius. Designer modeling for personalized game content creation tools. In *Proceedings of the AIIDE Workshop on Artificial Intelligence & Game Aesthetics*, 2013.
379. Antonios Liapis, Georgios N. Yannakakis, and Julian Togelius. Sentient Sketchbook: Computer-aided game level authoring. In *Proceedings of ACM Conference on Foundations of Digital Games*, pages 213–220, 2013.
380. Antonios Liapis, Georgios N. Yannakakis, and Julian Togelius. Sentient World: Human-Based Procedural Cartography. In *Evolutionary and Biologically Inspired Music, Sound, Art and Design*, pages 180–191. Springer, 2013.
381. Antonios Liapis, Georgios N. Yannakakis, and Julian Togelius. Computational Game Creativity. In *Proceedings of the Fifth International Conference on Computational Creativity*, pages 285–292, 2014.
382. Antonios Liapis, Georgios N. Yannakakis, and Julian Togelius. Constrained novelty search: A study on game content generation. *Evolutionary Computation*, 23(1):101–129, 2015.
383. Vladimir Lifschitz. Answer set programming and plan generation. *Artificial Intelligence*, 138(1-2):39–54, 2002.
384. Rensis Likert. A technique for the measurement of attitudes. *Archives of Psychology*, 140:1–55, 1932.
385. Chong-U Lim, Robin Baumgarten, and Simon Colton. Evolving behaviour trees for the commercial game DEFCON. In *European Conference on the Applications of Evolutionary Computation*, pages 100–110. Springer, 2010.
386. Chong-U Lim and D. Fox Harrell. Revealing social identity phenomena in videogames with archetypal analysis. In *Proceedings of the 6th International AISB Symposium on AI and Games*, 2015.
387. Aristid Lindenmayer. Mathematical models for cellular interactions in development I. Filaments with one-sided inputs. *Journal of Theoretical Biology*, 18(3):280–299, 1968.
388. R. L. Linn and N. E. Gronlund. *Measurement and assessment in teaching*. Prentice-Hall, 2000.
389. Nir Lipovetzky and Hector Geffner. Width-based algorithms for classical planning: New results. In *Proceedings of the Twenty-first European Conference on Artificial Intelligence*, pages 1059–1060. IOS Press, 2014.
390. Nir Lipovetzky, Miquel Ramirez, and Hector Geffner. Classical Planning with Simulators: Results on the Atari Video Games. In *Proceedings of IJCAI*, pages 1610–1616, 2015.
391. Maria Teresa Llano, Michael Cook, Christian Guckelsberger, Simon Colton, and Rose Hepworth. Towards the automatic generation of fictional ideas for games. In *Experimental AI in Games (EXAG14), a Workshop collocated with the Tenth Annual AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE14). AAAI Publications*, 2014.
392. Daniele Loiacono, Pier Luca Lanzi, Julian Togelius, Enrique Onieva, David A. Pelta, Martin V. Butz, Thies D. Lönneker, Luigi Cardamone, Diego Perez, Yago Sáez, Mike Preuss, and Jan Quadflieg. The 2009 simulated car racing championship. *Computational Intelligence and AI in Games, IEEE Transactions on*, 2(2):131–147, 2010.
393. Daniele Loiacono, Julian Togelius, Pier Luca Lanzi, Leonard Kinnaird-Heether, Simon M. Lucas, Matt Simmerson, Diego Perez, Robert G. Reynolds, and Yago Saez. The WCCI 2008 simulated car racing competition. In *IEEE Symposium on Computational Intelligence and Games*, pages 119–126. IEEE, 2008.
394. Phil Lopes, Antonios Liapis, and Georgios N. Yannakakis. Sonancia: Sonification of procedurally generated game levels. In *Proceedings of the ICCC workshop on Computational Creativity & Games*, 2015.

395. Phil Lopes, Antonios Liapis, and Georgios N. Yannakakis. Framing tension for game generation. In *Proceedings of the Seventh International Conference on Computational Creativity*, 2016.
396. Phil Lopes, Antonios Liapis, and Georgios N. Yannakakis. Modelling affect for horror soundscapes. *IEEE Transactions on Affective Computing*, 2017.
397. Phil Lopes, Georgios N. Yannakakis, and Antonios Liapis. RankTrace: Relative and Unbounded Affect Annotation. In *Affective Computing and Intelligent Interaction (ACII), 2017 International Conference on*, 2017.
398. Ricardo Lopes and Rafael Bidarra. Adaptivity challenges in games and simulations: a survey. *Computational Intelligence and AI in Games, IEEE Transactions on*, 3(2):85–99, 2011.
399. Sandy Louchart, Ruth Aylett, Joao Dias, and Ana Paiva. Unscripted narrative for affectively driven characters. In *AIIDE*, pages 81–86, 2005.
400. Nathaniel Love, Timothy Hinrichs, David Haley, Eric Schkufza, and Michael Genesereth. General game playing: Game description language specification. Technical Report LG-2006-01, Stanford Logic Group, Computer Science Department, Stanford University, 2008.
401. A. Bryan Loyall and Joseph Bates. Personality-rich believable agents that use language. In *Proceedings of the First International Conference on Autonomous Agents*, pages 106–113. ACM, 1997.
402. Feiyu Lu, Kaito Yamamoto, Luis H. Nomura, Syunsuke Mizuno, YoungMin Lee, and Ruck Thawonmas. Fighting game artificial intelligence competition platform. In *Consumer Electronics (GCCE), 2013 IEEE 2nd Global Conference on*, pages 320–323. IEEE, 2013.
403. Simon M. Lucas. Evolving a Neural Network Location Evaluator to Play Ms. Pac-Man. In *Proceedings of the IEEE Symposium on Computational Intelligence and Games*, pages 203–210, 2005.
404. Simon M. Lucas. Ms Pac-Man competition. *ACM SIGEVOlution*, 2(4):37–38, 2007.
405. Simon M. Lucas. Computational intelligence and games: Challenges and opportunities. *International Journal of Automation and Computing*, 5(1):45–57, 2008.
406. Simon M. Lucas and Graham Kendall. Evolutionary computation and games. *Computational Intelligence Magazine, IEEE*, 1(1):10–18, 2006.
407. Simon M. Lucas, Michael Mateas, Mike Preuss, Pieter Spronck, and Julian Togelius. Artificial and Computational Intelligence in Games (Dagstuhl Seminar 12191). *Dagstuhl Reports*, 2(5):43–70, 2012.
408. Simon M. Lucas and T. Jeff Reynolds. Learning finite-state transducers: Evolution versus heuristic state merging. *IEEE Transactions on Evolutionary Computation*, 11(3):308–325, 2007.
409. Jeremy Ludwig and Art Farley. A learning infrastructure for improving agent performance and game balance. In Georgios N. Yannakakis and John Hallam, editors, *Proceedings of the AIIDE'07 Workshop on Optimizing Player Satisfaction, Technical Report WS-07-01*, pages 7–12. AAAI Press, 2007.
410. Kevin Lynch. *The Image of the City*. MIT Press, 1960.
411. James MacQueen. Some methods for classification and analysis of multivariate observations. In *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability*, number 14, pages 281–297. Oakland, CA, USA, 1967.
412. Brian Magerko. Story representation and interactive drama. In *AIIDE*, pages 87–92, 2005.
413. Brendan Maher. Can a video game company tame toxic behaviour? *Nature*, 531(7596):568–571, 2016.
414. Tobias Mahlmann, Anders Drachen, Julian Togelius, Alessandro Canossa, and Georgios N. Yannakakis. Predicting player behavior in Tomb Raider: Underworld. In *Proceedings of the 2010 IEEE Conference on Computational Intelligence and Games*, pages 178–185. IEEE, 2010.
415. Tobias Mahlmann, Julian Togelius, and Georgios N. Yannakakis. Modelling and evaluation of complex scenarios with the strategy game description language. In *Computational Intelligence and Games (CIG), 2011 IEEE Conference on*, pages 174–181. IEEE, 2011.

416. Tobias Mahlmann, Julian Togelius, and Georgios N. Yannakakis. Evolving card sets towards balancing Dominion. In *Proceedings of the IEEE Congress on Evolutionary Computation (CEC)*. IEEE, 2012.
417. Kevin Majchrzak, Jan Quadflieg, and Günter Rudolph. Advanced dynamic scripting for fighting game AI. In *International Conference on Entertainment Computing*, pages 86–99. Springer, 2015.
418. Nikos Malandrakis, Alexandros Potamianos, Georgios Evangelopoulos, and Athanasia Zlatintsi. A supervised approach to movie emotion tracking. In *Acoustics, Speech and Signal Processing (ICASSP), 2011 IEEE International Conference on*, pages 2376–2379. IEEE, 2011.
419. Thomas W. Malone. What makes computer games fun? *Byte*, 6:258–277, 1981.
420. Regan L. Mandryk and M. Stella Atkins. A fuzzy physiological approach for continuously modeling emotion during interaction with play technologies. *International Journal of Human-Computer Studies*, 65(4):329–347, 2007.
421. Regan L. Mandryk, Kori M. Inkpen, and Thomas W. Calvert. Using psychophysiological techniques to measure user experience with entertainment technologies. *Behaviour & Information Technology*, 25(2):141–158, 2006.
422. Jacek Mandziuk. Computational intelligence in mind games. *Challenges for Computational Intelligence*, 63:407–442, 2007.
423. Jacek Mandziuk. *Knowledge-free and learning-based methods in intelligent game playing*. Springer, 2010.
424. Benjamin Mark, Tudor Berechet, Tobias Mahlmann, and Julian Togelius. Procedural Generation of 3D Caves for Games on the GPU. In *Proceedings of the Conference on the Foundations of Digital Games (FDG)*, 2015.
425. Dave Mark. *Behavioral Mathematics for game AI*. Charles River Media, 2009.
426. Dave Mark and Kevin Dill. Improving AI decision modeling through utility theory. In *Game Developers Conference*, 2010.
427. Gloria Mark, Yiran Wang, and Melissa Niiya. Stress and multitasking in everyday college life: An empirical study of online activity. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 41–50, 2014.
428. Stacy Marsella, Jonathan Gratch, and Paolo Petta. Computational models of emotion. *A Blueprint for Affective Computing—A sourcebook and manual*, 11(1):21–46, 2010.
429. Chris Martens. Ceptre: A language for modeling generative interactive systems. In *Eleventh Artificial Intelligence and Interactive Digital Entertainment Conference*, 2015.
430. Héctor P. Martínez, Yoshua Bengio, and Georgios N. Yannakakis. Learning deep physiological models of affect. *Computational Intelligence Magazine, IEEE*, 9(1):20–33, 2013.
431. Héctor P. Martínez, Maurizio Garbarino, and Georgios N. Yannakakis. Generic physiological features as predictors of player experience. In *Affective Computing and Intelligent Interaction*, pages 267–276. Springer, 2011.
432. Héctor P. Martínez, Kenneth Hullett, and Georgios N. Yannakakis. Extending neuroevolutionary preference learning through player modeling. In *Proceedings of the 2010 IEEE Conference on Computational Intelligence and Games*, pages 313–320. IEEE, 2010.
433. Héctor P. Martínez and Georgios N. Yannakakis. Genetic search feature selection for affective modeling: a case study on reported preferences. In *Proceedings of the 3rd International Workshop on Affective Interaction in Natural Environments*, pages 15–20. ACM, 2010.
434. Héctor P. Martínez and Georgios N. Yannakakis. Mining multimodal sequential patterns: a case study on affect detection. In *Proceedings of International Conference on Multimodal Interfaces (ICMI)*, pages 3–10. ACM, 2011.
435. Héctor P. Martínez and Georgios N. Yannakakis. Deep multimodal fusion: Combining discrete events and continuous signals. In *Proceedings of the 16th International Conference on Multimodal Interaction*, pages 34–41. ACM, 2014.
436. Héctor P. Martínez, Georgios N. Yannakakis, and John Hallam. Don't Classify Ratings of Affect; Rank them! *IEEE Transactions on Affective Computing*, 5(3):314–326, 2014.

437. Giovanna Martinez-Arellano, Richard Cant, and David Woods. Creating AI Characters for Fighting Games using Genetic Programming. *IEEE Transactions on Computational Intelligence and AI in Games*, 2016.
438. Michael Mateas. *Interactive Drama, Art and Artificial Intelligence*. PhD thesis, Carnegie Mellon University, Pittsburgh, PA, USA, 2002.
439. Michael Mateas. Expressive AI: Games and Artificial Intelligence. In *DIGRA Conference*, 2003.
440. Michael Mateas and Andrew Stern. A behavior language for story-based believable agents. *IEEE Intelligent Systems*, 17(4):39–47, 2002.
441. Michael Mateas and Andrew Stern. Façade: An experiment in building a fully-realized interactive drama. In *Game Developers Conference*, 2003.
442. Michael Mauderer, Simone Conte, Miguel A. Nacenta, and Dhanraj Vishwanath. Depth perception with gaze-contingent depth of field. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 217–226, 2014.
443. John D. Mayer and Peter Salovey. The intelligence of emotional intelligence. *Intelligence*, 17(4):433–442, 1993.
444. Allan Mazur, Elizabeth J. Susman, and Sandy Edelbrock. Sex difference in testosterone response to a video game contest. *Evolution and Human Behavior*, 18(5):317–326, 1997.
445. Andrew McAfee, Erik Brynjolfsson, Thomas H. Davenport, D. J. Patil, and Dominic Barton. Big data. *The management revolution. Harvard Bus Rev*, 90(10):61–67, 2012.
446. John McCarthy. Partial formalizations and the Lemmings game. Technical report, Stanford University, 1998.
447. Josh McCoy, Mike Treanor, Ben Samuel, Michael Mateas, and Noah Wardrip-Fruin. Prom week: social physics as gameplay. In *Proceedings of the 6th International Conference on Foundations of Digital Games*, pages 319–321. ACM, 2011.
448. Josh McCoy, Mike Treanor, Ben Samuel, Aaron A. Reed, Noah Wardrip-Fruin, and Michael Mateas. Prom week. In *Proceedings of the International Conference on the Foundations of Digital Games*, pages 235–237. ACM, 2012.
449. Robert R. McCrae and Paul T. Costa Jr. A five-factor theory of personality. *Handbook of personality: Theory and research*, 2:139–153, 1999.
450. Warren S. McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, 5(4):115–133, 1943.
451. Scott W. McQuiggan, Sunyoung Lee, and James C. Lester. Early prediction of student frustration. In *Proceedings of International Conference on Affective Computing and Intelligent Interaction*, pages 698–709. Springer, 2007.
452. Scott W. McQuiggan, Bradford W. Mott, and James C. Lester. Modeling self-efficacy in intelligent tutoring systems: An inductive approach. *User Modeling and User-Adapted Interaction*, 18(1):81–123, 2008.
453. Andre Mendes, Julian Togelius, and Andy Nealen. Hyper-heuristic general video game playing. In *Computational Intelligence and Games (CIG), 2016 IEEE Conference on*. IEEE, 2016.
454. Daniel S. Messinger, Tricia D. Cassel, Susan I. Acosta, Zara Ambadar, and Jeffrey F. Cohn. Infant smiling dynamics and perceived positive emotion. *Journal of Nonverbal Behavior*, 32(3):133–155, 2008.
455. Angeliki Metallinou and Shrikanth Narayanan. Annotation and processing of continuous emotional attributes: Challenges and opportunities. In *10th IEEE International Conference and Workshops on Automatic Face and Gesture Recognition (FG)*. IEEE, 2013.
456. Zbigniew Michalewicz. Do not kill unfeasible individuals. In *Proceedings of the Fourth Intelligent Information Systems Workshop*, pages 110–123, 1995.
457. Risto Miikkulainen, Bobby D. Bryant, Ryan Cornelius, Igor V. Karpov, Kenneth O. Stanley, and Chern Han Yong. Computational intelligence in games. *Computational Intelligence: Principles and Practice*, pages 155–191, 2006.
458. Benedikte Mikkelsen, Christoffer Holmgård, and Julian Togelius. Ethical Considerations for Player Modeling. In *Proceedings of the AAAI WNAIG workshop*, 2017.

459. Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.
460. George A. Miller. The magical number seven, plus or minus two: some limits on our capacity for processing information. *Psychological Review*, 63(2):81, 1956.
461. Ian Millington and John Funge. *Artificial intelligence for games*. CRC Press, 2009.
462. Talya Miron-Shatz, Arthur Stone, and Daniel Kahneman. Memories of yesterday's emotions: Does the valence of experience affect the memory-experience gap? *Emotion*, 9(6):885, 2009.
463. Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International Conference on Machine Learning*, pages 1928–1937, 2016.
464. Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.
465. Mathew Monfort, Matthew Johnson, Aude Oliva, and Katja Hofmann. Asynchronous data aggregation for training end to end visual control networks. In *Proceedings of the 16th Conference on Autonomous Agents and Multi-Agent Systems*, pages 530–537. International Foundation for Autonomous Agents and Multiagent Systems, May 2017.
466. Nick Montfort and Ian Bogost. *Racing the beam: The Atari video computer system*. MIT Press, 2009.
467. Matej Moravčík, Martin Schmid, Neil Burch, Viliam Lisỳ, Dustin Morrill, Nolan Bard, Trevor Davis, Kevin Waugh, Michael Johanson, and Michael Bowling. Deepstack: Expert-level artificial intelligence in no-limit poker. *arXiv preprint arXiv:1701.01724*, 2017.
468. Jon D. Morris. Observations: SAM: The self-assessment manikin—An efficient cross-cultural measurement of emotional response. *Journal of Advertising Research*, 35(6):63–68, 1995.
469. Jorge Munoz, Georgios N. Yannakakis, Fiona Mulvey, Dan Witzner Hansen, German Gutierrez, and Araceli Sanchis. Towards gaze-controlled platform games. In *Computational Intelligence and Games (CIG), 2011 IEEE Conference on*, pages 47–54. IEEE, 2011.
470. Hector Munoz-Avila, Christian Bauckhage, Michal Bida, Clare Bates Congdon, and Graham Kendall. Learning and Game AI. *Dagstuhl Follow-Ups*, 6, 2013.
471. Roger B. Myerson. Game theory: analysis of conflict. 1991. *Cambridge: Mass, Harvard University*.
472. Roger B. Myerson. *Game theory*. Harvard University Press, 2013.
473. Lennart Nacke and Craig A. Lindley. Flow and immersion in first-person shooters: measuring the player's gameplay experience. In *Proceedings of the 2008 Conference on Future Play: Research, Play, Share*, pages 81–88. ACM, 2008.
474. Frederik Nagel, Reinhard Kopiez, Oliver Grewe, and Eckart Altenmüller. Emujoy: Software for continuous measurement of perceived emotions in music. *Behavior Research Methods*, 39(2):283–290, 2007.
475. Karthik Narasimhan, Tejas Kulkarni, and Regina Barzilay. Language understanding for text-based games using deep reinforcement learning. *arXiv preprint arXiv:1506.08941*, 2015.
476. Alexander Nareyek. Intelligent agents for computer games. In T.A. Marsland and I. Frank, editors, *Computers and Games, Second International Conference, CG 2002*, pages 414–422, 2002.
477. Alexander Nareyek. Game AI is dead. Long live game AI! *IEEE Intelligent Systems*, (1):9–11, 2007.
478. John F. Nash. Equilibrium points in n-person games. In *Proceedings of the National Academy of Sciences*, number 1, pages 48–49, 1950.
479. Steve Nebel, Sascha Schneider, and Günter Daniel Rey. Mining learning and crafting scientific experiments: a literature review on the use of Minecraft in education and research. *Journal of Educational Technology & Society*, 19(2):355, 2016.

480. Graham Nelson. Natural language, semantic analysis, and interactive fiction. *IF Theory Reader*, 141, 2006.
481. Mark J. Nelson. Game Metrics Without Players: Strategies for Understanding Game Artifacts. In *AIIDE Workshop on Artificial Intelligence in the Game Design Process*, 2011.
482. Mark J. Nelson, Simon Colton, Edward J. Powley, Swen E. Gaudl, Peter Ivey, Rob Saunders, Blanca Pérez Ferrer, and Michael Cook. Mixed-initiative approaches to on-device mobile game design. In *Proceedings of the CHI Workshop on Mixed-Initiative Creative Interfaces*, 2017.
483. Mark J. Nelson and Michael Mateas. Search-Based Drama Management in the Interactive Fiction Anchorhead. In *Proceedings of the First Artificial Intelligence and Interactive Digital Entertainment Conference*, pages 99–104, 2005.
484. Mark J. Nelson and Michael Mateas. An interactive game-design assistant. In *Proceedings of the 13th International Conference on Intelligent User Interfaces*, pages 90–98, 2008.
485. Mark J. Nelson and Adam M. Smith. ASP with applications to mazes and levels. In *Procedural Content Generation in Games*, pages 143–157. Springer, 2016.
486. Mark J. Nelson, Julian Togelius, Cameron Browne, and Michael Cook. Rules and mechanics. In *Procedural Content Generation in Games*, pages 99–121. Springer, 2016.
487. John Von Neumann. *Theory of Self-Reproducing Automata*. University of Illinois Press, Champaign, IL, USA, 1966.
488. Truong-Huy D. Nguyen, Shree Subramanian, Magy Seif El-Nasr, and Alessandro Canossa. Strategy Detection in Wuzzit: A Decision Theoretic Approach. In *International Conference on Learning Science—Workshop on Learning Analytics for Learning and Becoming a Practice*, 2014.
489. Jakob Nielsen. Usability 101: Introduction to usability, 2003. Available at http://www.useit.com/alertbox/20030825.html.
490. Jon Lau Nielsen, Benjamin Fedder Jensen, Tobias Mahlmann, Julian Togelius, and Georgios N. Yannakakis. AI for General Strategy Game Playing. *Handbook of Digital Games*, pages 274–304, 2014.
491. Thorbjørn S. Nielsen, Gabriella A. B. Barros, Julian Togelius, and Mark J. Nelson. General Video Game Evaluation Using Relative Algorithm Performance Profiles. In *Applications of Evolutionary Computation*, pages 369–380. Springer, 2015.
492. Thorbjørn S. Nielsen, Gabriella A. B. Barros, Julian Togelius, and Mark J. Nelson. Towards generating arcade game rules with VGDL. In *Proceedings of the 2015 IEEE Conference on Computational Intelligence and Games*, 2015.
493. Anton Nijholt. BCI for games: A state of the art survey. In *Entertainment Computing-ICEC 2008*, pages 225–228. Springer, 2009.
494. Nils J. Nilsson. Shakey the robot. Technical report, DTIC Document, 1984.
495. Kai Ninomiya, Mubbasir Kapadia, Alexander Shoulson, Francisco Garcia, and Norman Badler. Planning approaches to constraint-aware navigation in dynamic environments. *Computer Animation and Virtual Worlds*, 26(2):119–139, 2015.
496. Stefano Nolfi and Dario Floreano. *Evolutionary robotics: The biology, intelligence, and technology of self-organizing machines*. MIT Press, 2000.
497. David G. Novick and Stephen Sutton. What is mixed-initiative interaction. In *Proceedings of the AAAI Spring Symposium on Computational Models for Mixed Initiative Interaction*, pages 114–116, 1997.
498. Gabriela Ochoa. On genetic algorithms and Lindenmayer systems. In *Parallel Problem Solving from Nature—PPSN V*, pages 335–344. Springer, 1998.
499. Jacob Kaae Olesen, Georgios N. Yannakakis, and John Hallam. Real-time challenge balance in an RTS game using rtNEAT. In *Computational Intelligence and Games, 2008. CIG'08. IEEE Symposium On*, pages 87–94. IEEE, 2008.
500. Jacob Olsen. Realtime procedural terrain generation. 2004.
501. Peter Thorup Ølsted, Benjamin Ma, and Sebastian Risi. Interactive evolution of levels for a competitive multiplayer FPS. In *Evolutionary Computation (CEC), 2015 IEEE Congress on*, pages 1527–1534. IEEE, 2015.

502. Cathy O'Neil. *Weapons of math destruction: How big data increases inequality and threatens democracy*. Crown Publishing Group (NY), 2016.
503. Santiago Ontañón. The combinatorial multi-armed bandit problem and its application to real-time strategy games. In *Ninth Artificial Intelligence and Interactive Digital Entertainment Conference*, 2013.
504. Santiago Ontañón, Gabriel Synnaeve, Alberto Uriarte, Florian Richoux, David Churchill, and Mike Preuss. A survey of real-time strategy game AI research and competition in StarCraft. *IEEE Transactions on Computational Intelligence and AI in Games*, 5(4):293–311, 2013.
505. Santiago Ontañón, Gabriel Synnaeve, Alberto Uriarte, Florian Richoux, David Churchill, and Mike Preuss. RTS AI: Problems and Techniques. In *Encyclopedia of Computer Graphics and Games*. Springer, 2015.
506. Jeff Orkin. Applying goal-oriented action planning to games. *AI game programming wisdom*, 2:217–228, 2003.
507. Jeff Orkin. Three states and a plan: the AI of F.E.A.R. In *Game Developers Conference*, 2006.
508. Jeff Orkin and Deb Roy. The restaurant game: Learning social behavior and language from thousands of players online. *Journal of Game Development*, 3(1):39–60, 2007.
509. Mauricio Orozco, Juan Silva, Abdulmotaleb El Saddik, and Emil Petriu. The role of haptics in games. In *Haptics Rendering and Applications*. InTech, 2012.
510. Brian O'Neill and Mark Riedl. Emotion-driven narrative generation. In *Emotion in Games: Theory and Praxis*, pages 167–180. Springer, 2016.
511. Juan Ortega, Noor Shaker, Julian Togelius, and Georgios N. Yannakakis. Imitating human playing styles in Super Mario Bros. *Entertainment Computing*, 4(2):93–104, 2013.
512. Andrew Ortony, Gerald L. Clore, and Allan Collins. *The cognitive structure of emotions*. Cambridge University Press, 1990.
513. Martin J. Osborne. *An introduction to game theory*. Oxford University Press, 2004.
514. Alexander Osherenko. *Opinion Mining and Lexical Affect Sensing. Computer-aided analysis of opinions and emotions in texts*. PhD thesis, University of Augsburg, 2010.
515. Seth Ovadia. Ratings and rankings: Reconsidering the structure of values and their measurement. *International Journal of Social Research Methodology*, 7(5):403–414, 2004.
516. Ana Paiva, Joao Dias, Daniel Sobral, Ruth Aylett, Polly Sobreperez, Sarah Woods, Carsten Zoll, and Lynne Hall. Caring for agents and agents that care: Building empathic relations with synthetic agents. In *Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 194–201. IEEE Computer Society, 2004.
517. Bo Pang and Lillian Lee. Opinion mining and sentiment analysis. *Foundations and Trends in Information Retrieval*, 2(1–2):1–135, 2008.
518. Matt Parker and Bobby D. Bryant. Visual control in Quake II with a cyclic controller. In *Computational Intelligence and Games, 2008. CIG'08. IEEE Symposium On*, pages 151–158. IEEE, 2008.
519. Matt Parker and Bobby D. Bryant. Neurovisual control in the Quake II environment. *IEEE Transactions on Computational Intelligence and AI in Games*, 4(1):44–54, 2012.
520. Chris Pedersen, Julian Togelius, and Georgios N. Yannakakis. Modeling Player Experience in Super Mario Bros. In *Proceedings of the IEEE Symposium on Computational Intelligence and Games*, pages 132–139. IEEE, 2009.
521. Chris Pedersen, Julian Togelius, and Georgios N. Yannakakis. Modeling Player Experience for Content Creation. *IEEE Transactions on Computational Intelligence and AI in Games*, 2(1):54–67, 2010.
522. Barney Pell. *Strategy generation and evaluation for meta-game playing*. PhD thesis, University of Cambridge, 1993.
523. Peng Peng, Quan Yuan, Ying Wen, Yaodong Yang, Zhenkun Tang, Haitao Long, and Jun Wang. Multiagent Bidirectionally-Coordinated Nets for Learning to Play StarCraft Combat Games. *arXiv preprint arXiv:1703.10069*, 2017.
524. Tom Pepels, Mark H. M. Winands, and Marc Lanctot. Real-time Monte Carlo tree search in Ms Pac-Man. *IEEE Transactions on Computational Intelligence and AI in Games*, 6(3):245–257, 2014.

525. Diego Perez, Edward J. Powley, Daniel Whitehouse, Philipp Rohlfshagen, Spyridon Samothrakis, Peter I. Cowling, and Simon M. Lucas. Solving the physical traveling salesman problem: Tree search and macro actions. *IEEE Transactions on Computational Intelligence and AI in Games*, 6(1):31–45, 2014.
526. Diego Perez, Spyridon Samothrakis, Simon Lucas, and Philipp Rohlfshagen. Rolling horizon evolution versus tree search for navigation in single-player real-time games. In *Proceedings of the 15th Annual Conference on Genetic and Evolutionary Computation*, pages 351–358. ACM, 2013.
527. Diego Perez-Liebana, Spyridon Samothrakis, Julian Togelius, Tom Schaul, and Simon M. Lucas. General video game AI: Competition, challenges and opportunities. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, 2016.
528. Diego Perez-Liebana, Spyridon Samothrakis, Julian Togelius, Tom Schaul, Simon M. Lucas, Adrien Couëtoux, Jerry Lee, Chong-U Lim, and Tommy Thompson. The 2014 general video game playing competition. *IEEE Transactions on Computational Intelligence and AI in Games*, 8(3):229–243, 2016.
529. Ken Perlin. An image synthesizer. *ACM SIGGRAPH Computer Graphics*, 19(3):287–296, 1985.
530. Rosalind W. Picard. *Affective Computing*. MIT Press, Cambridge, MA, 1997.
531. Grant Pickett, Foaad Khosmood, and Allan Fowler. Automated generation of conversational non player characters. In *Eleventh Artificial Intelligence and Interactive Digital Entertainment Conference*, 2015.
532. Michele Pirovano. The use of Fuzzy Logic for Artificial Intelligence in Games. Technical report, University of Milano, Milano, 2012.
533. Jacques Pitrat. Realization of a general game-playing program. In *IFIP congress (2)*, pages 1570–1574, 1968.
534. Isabella Poggi, Catherine Pelachaud, Fiorella de Rosis, Valeria Carofiglio, and Berardina De Carolis. GRETA. A believable embodied conversational agent. In *Multimodal intelligent information presentation*, pages 3–25. Springer, 2005.
535. Mihai Polceanu. Mirrorbot: Using human-inspired mirroring behavior to pass a Turing test. In *Computational Intelligence in Games (CIG), 2013 IEEE Conference on*. IEEE, 2013.
536. Riccardo Poli, William B. Langdon, and Nicholas F. McPhee. *A field guide to genetic programming*. 2008. Published via http://lulu.com and freely available at http://www.gp-field-guide.org.uk (With contributions by J. R. Koza).
537. Jordan B. Pollack and Alan D. Blair. Co-evolution in the successful learning of backgammon strategy. *Machine learning*, 32(3):225–240, 1998.
538. Jordan B. Pollack, Alan D. Blair, and Mark Land. Coevolution of a backgammon player. In *Artificial Life V: Proceedings of the Fifth International Workshop on the Synthesis and Simulation of Living Systems*, pages 92–98. Cambridge, MA: The MIT Press, 1997.
539. Jonathan Posner, James A. Russell, and Bradley S. Peterson. The circumplex model of affect: An integrative approach to affective neuroscience, cognitive development, and psychopathology. *Development and psychopathology*, 17(03):715–734, 2005.
540. David Premack and Guy Woodruff. Does the chimpanzee have a theory of mind? *Behavioral and brain sciences*, 1(04):515–526, 1978.
541. Mike Preuss, Daniel Kozakowski, Johan Hagelbäck, and Heike Trautmann. Reactive strategy choice in StarCraft by means of Fuzzy Control. In *Computational Intelligence in Games (CIG), 2013 IEEE Conference on*. IEEE, 2013.
542. Przemyslaw Prusinkiewicz and Aristid Lindenmayer. *The algorithmic beauty of plants*. Springer, 1990.
543. Jan Quadflieg, Mike Preuss, and Günter Rudolph. Driving as a human: a track learning based adaptable architecture for a car racing controller. *Genetic Programming and Evolvable Machines*, 15(4):433–476, 2014.
544. J. Ross Quinlan. Induction of decision trees. *Machine Learning*, 1(1):81–106, 1986.
545. J. Ross Quinlan. *C4. 5: programs for machine learning*. Elsevier, 2014.
546. Steve Rabin. *AI Game Programming Wisdom*. Charles River Media, Inc., 2002.

547. Steve Rabin. *AI Game Programming Wisdom 2*. Charles River Media, Inc., 2003.
548. Steve Rabin. *AI Game Programming Wisdom 3*. Charles River Media, Inc., 2006.
549. Steve Rabin. *AI Game Programming Wisdom 4*. Nelson Education, 2014.
550. Steve Rabin and Nathan Sturtevant. Pathfinding Architecture Optimizations. In *Game AI Pro: Collected Wisdom of Game AI Professionals*. CRC Press, 2013.
551. Steve Rabin and Nathan Sturtevant. Combining Bounding Boxes and JPS to Prune Grid Pathfinding. In *AAAI Conference on Artificial Intelligence*, 2016.
552. Steven Rabin. *Game AI Pro: Collected Wisdom of Game AI Professionals*. CRC Press, 2013.
553. Steven Rabin. *Game AI Pro 2: Collected Wisdom of Game AI Professionals*. CRC Press, 2015.
554. William L. Raffe, Fabio Zambetta, and Xiaodong Li. A survey of procedural terrain generation techniques using evolutionary algorithms. In *IEEE Congress on Evolutionary Computation (CEC)*. IEEE, 2012.
555. Judith Ramey, Ted Boren, Elisabeth Cuddihy, Joe Dumas, Zhiwei Guan, Maaike J. van den Haak, and Menno D. T. De Jong. Does think aloud work? How do we know? In *CHI'06 Extended Abstracts on Human Factors in Computing Systems*, pages 45–48. ACM, 2006.
556. Pramila Rani, Nilanjan Sarkar, and Changchun Liu. Maintaining optimal challenge in computer games through real-time physiological feedback. In *Proceedings of the 11th International Conference on Human Computer Interaction*, pages 184–192, 2005.
557. Jakob Rasmussen. *Are Behavior Trees a Thing of the Past?* Gamasutra, 2016.
558. Niklas Ravaja, Timo Saari, Mikko Salminen, Jari Laarni, and Kari Kallinen. Phasic emotional reactions to video game events: A psychophysiological investigation. *Media Psychology*, 8(4):343–367, 2006.
559. Genaro Rebolledo-Mendez, Ian Dunwell, Erika Martínez-Mirón, Maria Dolores Vargas-Cerdán, Sara De Freitas, Fotis Liarokapis, and Alma R. García-Gaona. Assessing Neurosky's usability to detect attention levels in an assessment exercise. *Human-Computer Interaction. New Trends*, pages 149–158, 2009.
560. Jochen Renz, Xiaoyu Ge, Stephen Gould, and Peng Zhang. The Angry Birds AI Competition. *AI Magazine*, 36(2):85–87, 2015.
561. Antonio Ricciardi and Patrick Thill. Adaptive AI for Fighting Games. Technical report, Stanford University, 2008.
562. Mark O. Riedl and Vadim Bulitko. Interactive narrative: An intelligent systems approach. *AI Magazine*, 34(1):67, 2012.
563. Mark O. Riedl and Andrew Stern. Believable agents and intelligent story adaptation for interactive storytelling. *Technologies for Interactive Digital Storytelling and Entertainment*, pages 1–12, 2006.
564. Mark O. Riedl and Alexander Zook. AI for game production. In *IEEE Conference on Computational Intelligence in Games (CIG)*. IEEE, 2013.
565. Sebastian Risi, Joel Lehman, David B. D'Ambrosio, Ryan Hall, and Kenneth O. Stanley. Combining Search-Based Procedural Content Generation and Social Gaming in the Petalz Video Game. In *Proceedings of AIIDE*, 2012.
566. Sebastian Risi, Joel Lehman, David B. D'Ambrosio, Ryan Hall, and Kenneth O. Stanley. Petalz: Search-based procedural content generation for the casual gamer. *IEEE Transactions on Computational Intelligence and AI in Games*, 8(3):244–255, 2016.
567. Sebastian Risi and Julian Togelius. Neuroevolution in games: State of the art and open challenges. *IEEE Transactions on Computational Intelligence and AI in Games*, 9(1):25–41, 2017.
568. David L. Roberts, Harikrishna Narayanan, and Charles L. Isbell. Learning to influence emotional responses for interactive storytelling. In *Proceedings of the 2009 AAAI Symposium on Intelligent Narrative Technologies II*, 2009.
569. Glen Robertson and Ian D. Watson. A review of real-time strategy game AI. *AI Magazine*, 35(4):75–104, 2014.
570. Glen Robertson and Ian D. Watson. An Improved Dataset and Extraction Process for StarCraft AI. In *FLAIRS Conference*, 2014.

571. Michael D. Robinson and Gerald L. Clore. Belief and feeling: evidence for an accessibility model of emotional self-report. *Psychological Bulletin*, 128(6):934, 2002.
572. Jennifer Robison, Scott McQuiggan, and James Lester. Evaluating the consequences of affective feedback in intelligent tutoring systems. In *Proceedings of International Conference on Affective Computing and Intelligent Interaction (ACII)*. IEEE, 2009.
573. Philipp Rohlfshagen, Jialin Liu, Diego Perez-Liebana, and Simon M. Lucas. Pac-Man Conquers Academia: Two Decades of Research Using a Classic Arcade Game. *IEEE Transactions on Computational Intelligence and AI in Games*, 2017.
574. Philipp Rohlfshagen and Simon M. Lucas. Ms Pac-Man versus Ghost team CEC 2011 competition. In *IEEE Congress on Evolutionary Computation (CEC)*, pages 70–77. IEEE, 2011.
575. Edmund T. Rolls. The orbitofrontal cortex and reward. *Cerebral Cortex*, 10(3):284–294, 2000.
576. Frank Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological Review*, 65(6):386, 1958.
577. Jonathan Rowe, Bradford Mott, Scott McQuiggan, Jennifer Robison, Sunyoung Lee, and James Lester. Crystal Island: A narrative-centered learning environment for eighth grade microbiology. In *Workshop on Intelligent Educational Games at the 14th International Conference on Artificial Intelligence in Education, Brighton, UK*, pages 11–20, 2009.
578. Jonathan P. Rowe, Lucy R. Shores, Bradford W. Mott, and James C. Lester. Integrating learning, problem solving, and engagement in narrative-centered learning environments. *International Journal of Artificial Intelligence in Education*, 21(1-2):115–133, 2011.
579. David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536, 1986.
580. Thomas Philip Runarsson and Simon M. Lucas. Coevolution versus self-play temporal difference learning for acquiring position evaluation in small-board Go. *IEEE Transactions on Evolutionary Computation*, 9(6):628–640, 2005.
581. James A. Russell. A circumplex model of affect. *Journal of Personality and Social Psychology*, 39(6):1161, 1980.
582. Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice-Hall, Englewood Cliffs, 1995.
583. Richard M. Ryan, C. Scott Rigby, and Andrew Przybylski. The motivational pull of video games: A self-determination theory approach. *Motivation and emotion*, 30(4):344–360, 2006.
584. Jennifer L. Sabourin and James C. Lester. Affect and engagement in Game-Based Learning environments. *IEEE Transactions on Affective Computing*, 5(1):45–56, 2014.
585. Owen Sacco, Antonios Liapis, and Georgios N. Yannakakis. A holistic approach for semantic-based game generation. In *Computational Intelligence and Games (CIG), 2016 IEEE Conference on*. IEEE, 2016.
586. Frantisek Sailer, Michael Buro, and Marc Lanctot. Adversarial planning through strategy simulation. In *Computational Intelligence and Games, 2007. CIG 2007. IEEE Symposium on*, pages 80–87. IEEE, 2007.
587. Katie Salen and Eric Zimmerman. *Rules of play: Game design fundamentals*. MIT Press, 2004.
588. Christoph Salge, Christian Lipski, Tobias Mahlmann, and Brigitte Mathiak. Using genetically optimized artificial intelligence to improve gameplaying fun for strategical games. In *Sandbox '08: Proceedings of the 2008 ACM SIGGRAPH symposium on Video games*, pages 7–14, New York, NY, USA, 2008. ACM.
589. Spyridon Samothrakis, Simon M. Lucas, Thomas Philip Runarsson, and David Robles. Coevolving game-playing agents: Measuring performance and intransitivities. *Evolutionary Computation, IEEE Transactions on*, 17(2):213–226, 2013.
590. Spyridon Samothrakis, David Robles, and Simon M. Lucas. Fast approximate max-n Monte Carlo tree search for Ms Pac-Man. *IEEE Transactions on Computational Intelligence and AI in Games*, 3(2):142–154, 2011.
591. Arthur L. Samuel. Some studies in machine learning using the game of Checkers. *IBM Journal of research and development*, 3(3):210–229, 1959.

592. Frederik Schadd, Sander Bakkes, and Pieter Spronck. Opponent modeling in real-time strategy games. In *GAMEON*, pages 61–70, 2007.
593. Jonathan Schaeffer, Neil Burch, Yngvi Björnsson, Akihiro Kishimoto, Martin Müller, Robert Lake, Paul Lu, and Steve Sutphen. Checkers is solved. *Science*, 317(5844):1518–1522, 2007.
594. Jonathan Schaeffer, Robert Lake, Paul Lu, and Martin Bryant. Chinook: the world man-machine Checkers champion. *AI Magazine*, 17(1):21, 1996.
595. Jost Schatzmann, Karl Weilhammer, Matt Stuttle, and Steve Young. A survey of statistical user simulation techniques for reinforcement-learning of dialogue management strategies. *Knowledge Engineering Review*, 21(2):97–126, 2006.
596. Tom Schaul. A video game description language for model-based or interactive learning. In *Computational Intelligence in Games (CIG), 2013 IEEE Conference on*. IEEE, 2013.
597. Tom Schaul. An extensible description language for video games. *IEEE Transactions on Computational Intelligence and AI in Games*, 6(4):325–331, 2014.
598. Tom Schaul, Julian Togelius, and Jürgen Schmidhuber. Measuring intelligence through games. *arXiv preprint arXiv:1109.1314*, 2011.
599. Jesse Schell. *The Art of Game Design: A book of lenses*. CRC Press, 2014.
600. Klaus R. Scherer. What are emotions? and how can they be measured? *Social Science Information*, 44(4):695–729, 2005.
601. Klaus R. Scherer, Angela Schorr, and Tom Johnstone. *Appraisal processes in emotion: Theory, methods, research*. Oxford University Press, 2001.
602. Jürgen Schmidhuber. Developmental robotics, optimal artificial curiosity, creativity, music, and the fine arts. *Connection Science*, 18(2):173–187, 2006.
603. Jacob Schrum, Igor V. Karpov, and Risto Miikkulainen. UT^2: Human-like behavior via neuroevolution of combat behavior and replay of human traces. In *Computational Intelligence and Games (CIG), 2011 IEEE Conference on*, pages 329–336. IEEE, 2011.
604. Brian Schwab. *AI game engine programming*. Nelson Education, 2009.
605. Brian Schwab, Dave Mark, Kevin Dill, Mike Lewis, and Richard Evans. GDC: Turing tantrums: AI developers rant, 2011.
606. Marco Scirea, Yun-Gyung Cheong, Mark J. Nelson, and Byung-Chull Bae. Evaluating musical foreshadowing of videogame narrative experiences. In *Proceedings of the 9th Audio Mostly: A Conference on Interaction With Sound*. ACM, 2014.
607. Ben Seymour and Samuel M. McClure. Anchors, scales and the relative coding of value in the brain. *Current Opinion in Neurobiology*, 18(2):173–178, 2008.
608. Mohammad Shaker, Mhd Hasan Sarhan, Ola Al Naameh, Noor Shaker, and Julian Togelius. Automatic generation and analysis of physics-based puzzle games. In *Computational Intelligence in Games (CIG), 2013 IEEE Conference on*. IEEE, 2013.
609. Noor Shaker, Stylianos Asteriadis, Georgios N. Yannakakis, and Kostas Karpouzis. A game-based corpus for analysing the interplay between game context and player experience. In *Affective Computing and Intelligent Interaction*, pages 547–556. Springer, 2011.
610. Noor Shaker, Stylianos Asteriadis, Georgios N. Yannakakis, and Kostas Karpouzis. Fusing visual and behavioral cues for modeling user experience in games. *Cybernetics, IEEE Transactions on*, 43(6):1519–1531, 2013.
611. Noor Shaker, Miguel Nicolau, Georgios N. Yannakakis, Julian Togelius, and Michael O'Neil. Evolving levels for Super Mario Bros using grammatical evolution. In *IEEE Conference on Computational Intelligence and Games*, pages 304–311. IEEE, 2012.
612. Noor Shaker, Mohammad Shaker, and Mohamed Abou-Zleikha. Towards generic models of player experience. In *Proceedings, the Eleventh AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*. AAAI Press, 2015.
613. Noor Shaker, Mohammad Shaker, and Julian Togelius. Evolving Playable Content for Cut the Rope through a Simulation-Based Approach. In *AIIDE*, 2013.
614. Noor Shaker, Mohammad Shaker, and Julian Togelius. Ropossum: An Authoring Tool for Designing, Optimizing and Solving Cut the Rope Levels. In *AIIDE*, 2013.
615. Noor Shaker, Gillian Smith, and Georgios N. Yannakakis. Evaluating content generators. In *Procedural Content Generation in Games*, pages 215–224. Springer, 2016.

616. Noor Shaker, Julian Togelius, and Mark J. Nelson, editors. *Procedural Content Generation in Games*. Springer, 2016.
617. Noor Shaker, Julian Togelius, and Georgios N. Yannakakis. Towards Automatic Personalized Content Generation for Platform Games. In *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE)*. AAAI Press, October 2010.
618. Noor Shaker, Julian Togelius, and Georgios N. Yannakakis. The experience-driven perspective. In *Procedural Content Generation in Games*, pages 181–194. Springer, 2016.
619. Noor Shaker, Julian Togelius, Georgios N. Yannakakis, Likith Poovanna, Vinay S. Ethiraj, Stefan J. Johansson, Robert G. Reynolds, Leonard K. Heether, Tom Schumann, and Marcus Gallagher. The Turing test track of the 2012 Mario AI championship: entries and evaluation. In *Computational Intelligence in Games (CIG), 2013 IEEE Conference on*. IEEE, 2013.
620. Noor Shaker, Julian Togelius, Georgios N. Yannakakis, Ben Weber, Tomoyuki Shimizu, Tomonori Hashiyama, Nathan Sorenson, Philippe Pasquier, Peter Mawhorter, Glen Takahashi, Gillian Smith, and Robin Baumgarten. The 2010 Mario AI championship: Level generation track. *Computational Intelligence and AI in Games, IEEE Transactions on*, 3(4):332–347, 2011.
621. Noor Shaker, Georgios N. Yannakakis, and Julian Togelius. Crowdsourcing the aesthetics of platform games. *Computational Intelligence and AI in Games, IEEE Transactions on*, 5(3):276–290, 2013.
622. Amirhosein Shantia, Eric Begue, and Marco Wiering. Connectionist reinforcement learning for intelligent unit micro management in StarCraft. In *Neural Networks (IJCNN), The 2011 International Joint Conference on*, pages 1794–1801. IEEE, 2011.
623. Manu Sharma, Manish Mehta, Santiago Ontañón, and Ashwin Ram. Player modeling evaluation for interactive fiction. In *Proceedings of the AIIDE 2007 Workshop on Optimizing Player Satisfaction*, pages 19–24, 2007.
624. Nandita Sharma and Tom Gedeon. Objective measures, sensors and computational techniques for stress recognition and classification: A survey. *Computer methods and programs in biomedicine*, 108(3):1287–1301, 2012.
625. Peter Shizgal and Andreas Arvanitogiannis. Gambling on dopamine. *Science*, 299(5614):1856–1858, 2003.
626. Yoav Shoham and Kevin Leyton-Brown. *Multiagent systems: Algorithmic, game-theoretic, and logical foundations*. Cambridge University Press, 2008.
627. Alexander Shoulson, Francisco M. Garcia, Matthew Jones, Robert Mead, and Norman I. Badler. Parameterizing behavior trees. In *International Conference on Motion in Games*, pages 144–155. Springer, 2011.
628. Nikolaos Sidorakis, George Alex Koulieris, and Katerina Mania. Binocular eye-tracking for the control of a 3D immersive multimedia user interface. In *Everyday Virtual Reality (WEVR), 2015 IEEE 1st Workshop on*, pages 15–18. IEEE, 2015.
629. David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.
630. Herbert A. Simon. A behavioral model of rational choice. *The quarterly journal of economics*, 69(1):99–118, 1955.
631. Shawn Singh, Mubbasir Kapadia, Glenn Reinman, and Petros Faloutsos. Footstep navigation for dynamic crowds. *Computer Animation and Virtual Worlds*, 22(2-3):151–158, 2011.
632. Moshe Sipper. *Evolved to Win*. Lulu.com, 2011.
633. Burrhus Frederic Skinner. *The behavior of organisms: An experimental analysis*. BF Skinner Foundation, 1990.
634. Ruben M. Smelik, Tim Tutenel, Klaas Jan de Kraker, and Rafael Bidarra. Interactive creation of virtual worlds using procedural sketching. In *Proceedings of Eurographics*, 2010.
635. Adam M. Smith, Erik Andersen, Michael Mateas, and Zoran Popović. A case study of expressively constrainable level design automation tools for a puzzle game. In *Proceedings of the International Conference on the Foundations of Digital Games*, pages 156–163. ACM, 2012.

636. Adam M. Smith, Chris Lewis, Kenneth Hullett, Gillian Smith, and Anne Sullivan. An inclusive taxonomy of player modeling. Technical Report UCSC-SOE-11-13, University of California, Santa Cruz, 2011.
637. Adam M. Smith and Michael Mateas. Variations forever: Flexibly generating rulesets from a sculptable design space of mini-games. In *Computational Intelligence and Games (CIG), 2010 IEEE Symposium on*, pages 273–280. IEEE, 2010.
638. Adam M. Smith and Michael Mateas. Answer set programming for procedural content generation: A design space approach. *Computational Intelligence and AI in Games, IEEE Transactions on*, 3(3):187–200, 2011.
639. Adam M. Smith, Mark J. Nelson, and Michael Mateas. Ludocore: A logical game engine for modeling videogames. In *Computational Intelligence and Games (CIG), 2010 IEEE Symposium on*, pages 91–98. IEEE, 2010.
640. Gillian Smith and Jim Whitehead. Analyzing the expressive range of a level generator. In *Proceedings of the 2010 Workshop on Procedural Content Generation in Games*. ACM, 2010.
641. Gillian Smith, Jim Whitehead, and Michael Mateas. Tanagra: A mixed-initiative level design tool. In *Proceedings of the Fifth International Conference on the Foundations of Digital Games*, pages 209–216. ACM, 2010.
642. Gillian Smith, Jim Whitehead, and Michael Mateas. Tanagra: Reactive planning and constraint solving for mixed-initiative level design. *Computational Intelligence and AI in Games, IEEE Transactions on*, 3(3):201–215, 2011.
643. Ian Sneddon, Gary McKeown, Margaret McRorie, and Tijana Vukicevic. Cross-cultural patterns in dynamic ratings of positive and negative natural emotional behaviour. *PloS ONE*, 6(2), 2011.
644. Sam Snodgrass and Santiago Ontañón. A Hierarchical MdMC Approach to 2D Video Game Map Generation. In *Eleventh Artificial Intelligence and Interactive Digital Entertainment Conference*, 2015.
645. Dennis Soemers. Tactical planning using MCTS in the game of StarCraft, 2014. Bachelor Thesis, Department of Knowledge Engineering, Maastricht University.
646. Andreas Sonderegger, Andreas Uebelbacher, Manuela Pugliese, and Juergen Sauer. The influence of aesthetics in usability testing: the case of dual-domain products. In *Proceedings of the Conference on Human Factors in Computing Systems*, pages 21–30, 2014.
647. Bhuman Soni and Philip Hingston. Bots trained to play like a human are more fun. In *IEEE International Joint Conference on Neural Networks (IJCNN); IEEE World Congress on Computational Intelligence*, pages 363–369. IEEE, 2008.
648. Patrikk D. Sørensen, Jeppeh M. Olsen, and Sebastian Risi. Interactive Super Mario Bros Evolution. In *Proceedings of the 2016 Genetic and Evolutionary Computation Conference*, pages 41–42. ACM, 2016.
649. Nathan Sorenson and Philippe Pasquier. Towards a generic framework for automated video game level creation. *Applications of Evolutionary Computation*, pages 131–140, 2010.
650. Pieter Spronck, Marc Ponsen, Ida Sprinkhuizen-Kuyper, and Eric Postma. Adaptive game AI with dynamic scripting. *Machine Learning*, 63(3):217–248, 2006.
651. Pieter Spronck, Ida Sprinkhuizen-Kuyper, and Eric Postma. Difficulty scaling of game AI. In *Proceedings of the 5th International Conference on Intelligent Games and Simulation (GAME-ON 2004)*, pages 33–37, 2004.
652. Ramakrishnan Srikant and Rakesh Agrawal. Mining sequential patterns: Generalizations and performance improvements. In *International Conference on Extending Database Technology*, pages 1–17. Springer, 1996.
653. Kenneth O. Stanley. Compositional Pattern Producing Networks: A novel abstraction of development. *Genetic Programming and Evolvable Machines*, 8(2):131–162, 2007.
654. Kenneth O. Stanley, Bobby D. Bryant, and Risto Miikkulainen. Real-time neuroevolution in the NERO video game. *Evolutionary Computation, IEEE Transactions on*, 9(6):653–668, 2005.
655. Kenneth O. Stanley and Risto Miikkulainen. Evolving neural networks through augmenting topologies. *Evolutionary Computation*, 10(2):99–127, 2002.

656. Kenneth O. Stanley and Risto Miikkulainen. Evolving a roving eye for Go. In *Genetic and Evolutionary Computation Conference*, pages 1226–1238. Springer, 2004.
657. Stanley Smith Stevens. On the Theory of Scales of Measurement. *Science*, 103(2684):677–680, 1946.
658. Neil Stewart, Gordon D. A. Brown, and Nick Chater. Absolute identification by relative judgment. *Psychological Review*, 112(4):881, 2005.
659. Andreas Stiegler, Keshav Dahal, Johannes Maucher, and Daniel Livingstone. Symbolic Reasoning for Hearthstone. *IEEE Transactions on Computational Intelligence and AI in Games*, 2017.
660. Jeff Stuckman and Guo-Qiang Zhang. Mastermind is NP-complete. *arXiv preprint cs/0512049*, 2005.
661. Nathan Sturtevant. Memory-Efficient Pathfinding Abstractions. In *AI Programming Wisdom 4*. Charles River Media, 2008.
662. Nathan Sturtevant and Steve Rabin. Canonical orderings on grids. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 683–689, 2016.
663. Nathan R. Sturtevant. Benchmarks for grid-based pathfinding. *IEEE Transactions on Computational Intelligence and AI in Games*, 4(2):144–148, 2012.
664. Nathan R. Sturtevant and Richard E. Korf. On pruning techniques for multi-player games. *Proceedings of The National Conference on Artificial Intelligence (AAAI)*, pages 201–208, 2000.
665. Nathan R. Sturtevant, Jason Traish, James Tulip, Tansel Uras, Sven Koenig, Ben Strasser, Adi Botea, Daniel Harabor, and Steve Rabin. The Grid-Based Path Planning Competition: 2014 Entries and Results. In *Eighth Annual Symposium on Combinatorial Search*, pages 241–251, 2015.
666. Adam James Summerville and Michael Mateas. Mystical Tutor: A Magic: The Gathering Design Assistant via Denoising Sequence-to-Sequence Learning. In *Twelfth Artificial Intelligence and Interactive Digital Entertainment Conference*, 2016.
667. Adam James Summerville, Shweta Philip, and Michael Mateas. MCMCTS PCG 4 SMB: Monte Carlo Tree Search to Guide Platformer Level Generation. In *Eleventh Artificial Intelligence and Interactive Digital Entertainment Conference*, 2015.
668. Adam James Summerville, Sam Snodgrass, Matthew Guzdial, Christoffer Holmgård, Amy K. Hoover, Aaron Isaksen, Andy Nealen, and Julian Togelius. Procedural Content Generation via Machine Learning (PCGML). *arXiv preprint arXiv:1702.00539*, 2017.
669. Adam James Summerville, Sam Snodgrass, Michael Mateas, and Santiago Ontañón Villar. The VGLC: The Video Game Level Corpus. *arXiv preprint arXiv:1606.07487*, 2016.
670. Petra Sundström. *Exploring the affective loop*. PhD thesis, Stockholm University, 2005.
671. Ben Sunshine-Hill, Michael Robbins, and Chris Jurney. Off the Beaten Path: Non-Traditional Uses of AI. In *Game Developers Conference, AI Summit*, 2012.
672. Richard S. Sutton and Andrew G. Barto. *Reinforcement learning: An introduction*. MIT Press, 1998.
673. Reid Swanson and Andrew S. Gordon. Say anything: Using textual case-based reasoning to enable open-domain interactive storytelling. *ACM Transactions on Interactive Intelligent Systems (TiiS)*, 2(3):16, 2012.
674. William R. Swartout, Jonathan Gratch, Randall W. Hill Jr, Eduard Hovy, Stacy Marsella, Jeff Rickel, and David Traum. Toward virtual humans. *AI Magazine*, 27(2):96, 2006.
675. Penelope Sweetser, Daniel M. Johnson, and Peta Wyeth. Revisiting the GameFlow model with detailed heuristics. *Journal: Creative Technologies*, 2012(3), 2012.
676. Penelope Sweetser and Janet Wiles. Scripting versus emergence: issues for game developers and players in game environment design. *International Journal of Intelligent Games and Simulations*, 4(1):1–9, 2005.
677. Penelope Sweetser and Janet Wiles. Using cellular automata to facilitate emergence in game environments. In *Proceedings of the 4th International Conference on Entertainment Computing (ICEC05)*, 2005.
678. Penelope Sweetser and Peta Wyeth. GameFlow: a model for evaluating player enjoyment in games. *Computers in Entertainment (CIE)*, 3(3):3–3, 2005.

679. Maciej Świechowski and Jacek Mańdziuk. Self-adaptation of playing strategies in general game playing. *IEEE Transactions on Computational Intelligence and AI in Games*, 6(4):367–381, 2014.
680. Gabriel Synnaeve and Pierre Bessière. Multiscale Bayesian Modeling for RTS Games: An Application to StarCraft AI. *IEEE Transactions on Computational intelligence and AI in Games*, 8(4):338–350, 2016.
681. Gabriel Synnaeve, Nantas Nardelli, Alex Auvolat, Soumith Chintala, Timothée Lacroix, Zeming Lin, Florian Richoux, and Nicolas Usunier. TorchCraft: a Library for Machine Learning Research on Real-Time Strategy Games. *arXiv preprint arXiv:1611.00625*, 2016.
682. Nicolas Szilas. IDtension: a narrative engine for Interactive Drama. In *Proceedings of the Technologies for Interactive Digital Storytelling and Entertainment (TIDSE) Conference*, pages 1–11, 2003.
683. Niels A. Taatgen, Marcia van Oploo, Jos Braaksma, and Jelle Niemantsverdriet. How to construct a believable opponent using cognitive modeling in the game of set. In *Proceedings of the Fifth International Conference on Cognitive Modeling*, pages 201–206, 2003.
684. Nima Taghipour, Ahmad Kardan, and Saeed Shiry Ghidary. Usage-based web recommendations: a reinforcement learning approach. In *Proceedings of the 2007 ACM Conference on Recommender Systems*, pages 113–120. ACM, 2007.
685. Bulent Tastan and Gita Reese Sukthankar. Learning policies for first person shooter games using inverse reinforcement learning. In *Seventh Artificial Intelligence and Interactive Digital Entertainment Conference*, 2011.
686. Shoshannah Tekofsky, Pieter Spronck, Aske Plaat, Jaap van Den Herik, and Jan Broersen. Play style: Showing your age. In *Computational Intelligence in Games (CIG), 2013 IEEE Conference on*. IEEE, 2013.
687. Shoshannah Tekofsky, Pieter Spronck, Aske Plaat, Jaap van den Herik, and Jan Broersen. Psyops: Personality assessment through gaming behavior. In *BNAIC 2013: Proceedings of the 25th Benelux Conference on Artificial Intelligence, Delft, The Netherlands, November 7-8, 2013*, 2013.
688. Gerald Tesauro. Practical issues in temporal difference learning. *Machine learning*, 8(3-4):257–277, 1992.
689. Gerald Tesauro. Temporal difference learning and TD-Gammon. *Communications of the ACM*, 38(3):58–68, 1995.
690. Ruck Thawonmas, Yoshitaka Kashifuji, and Kuan-Ta Chen. Detection of MMORPG bots based on behavior analysis. In *Proceedings of the 2008 International Conference on Advances in Computer Entertainment Technology*, pages 91–94. ACM, 2008.
691. Michael Thielscher. A General Game Description Language for Incomplete Information Games. In *AAAI*, pages 994–999, 2010.
692. William R. Thompson. On the likelihood that one unknown probability exceeds another in view of the evidence of two samples. *Biometrika*, 25(3/4):285–294, 1933.
693. David Thue, Vadim Bulitko, Marcia Spetch, and Eric Wasylishen. Interactive Storytelling: A Player Modelling Approach. In *AIIDE*, pages 43–48, 2007.
694. Christian Thurau, Christian Bauckhage, and Gerhard Sagerer. Learning human-like opponent behavior for interactive computer games. *Pattern Recognition, Lecture Notes in Computer Science 2781*, pages 148–155, 2003.
695. Christian Thurau, Christian Bauckhage, and Gerhard Sagerer. Imitation learning at all levels of game AI. In *Proceedings of the International Conference on Computer Games, Artificial Intelligence, Design and Education*, 2004.
696. Christian Thurau, Christian Bauckhage, and Gerhard Sagerer. Learning human-like Movement Behavior for Computer Games. In S. Schaal, A. Ijspeert, A. Billard, S. Vijayakumar, J. Hallam, and J.-A. Meyer, editors, *From Animals to Animats 8: Proceedings of the Eighth International Conference on Simulation of Adaptive Behavior (SAB-04)*, pages 315–323, Santa Monica, CA, July 2004. The MIT Press.
697. Tim J. W. Tijs, Dirk Brokken, and Wijnand A. Ijsselsteijn. Dynamic game balancing by recognizing affect. In *Proceedings of International Conference on Fun and Games*, pages 88–93. Springer, 2008.

698. Julian Togelius. Evolution of a subsumption architecture neurocontroller. *Journal of Intelligent & Fuzzy Systems*, 15(1):15–20, 2004.
699. Julian Togelius. A procedural critique of deontological reasoning. In *Proceedings of DiGRA*, 2011.
700. Julian Togelius. AI researchers, Video Games are your friends! In *Computational Intelligence*, pages 3–18. Springer, 2015.
701. Julian Togelius. How to run a successful game-based AI competition. *IEEE Transactions on Computational Intelligence and AI in Games*, 8(1):95–100, 2016.
702. Julian Togelius, Alex J. Champandard, Pier Luca Lanzi, Michael Mateas, Ana Paiva, Mike Preuss, and Kenneth O. Stanley. Procedural Content Generation in Games: Goals, Challenges and Actionable Steps. *Dagstuhl Follow-Ups*, 6, 2013.
703. Julian Togelius, Renzo De Nardi, and Simon M. Lucas. Making racing fun through player modeling and track evolution. In *Proceedings of the SAB'06 Workshop on Adaptive Approaches for Optimizing Player Satisfaction in Computer and Physical Games*, 2006.
704. Julian Togelius, Renzo De Nardi, and Simon M. Lucas. Towards automatic personalised content creation for racing games. In *Computational Intelligence and Games, 2007. CIG 2007. IEEE Symposium on*, pages 252–259. IEEE, 2007.
705. Julian Togelius, Sergey Karakovskiy, and Robin Baumgarten. The 2009 Mario AI competition. In *Evolutionary Computation (CEC), 2010 IEEE Congress on*. IEEE, 2010.
706. Julian Togelius, Sergey Karakovskiy, Jan Koutník, and Jürgen Schmidhuber. Super Mario evolution. In *Computational Intelligence and Games, 2009. CIG 2009. IEEE Symposium on*, pages 156–161. IEEE, 2009.
707. Julian Togelius and Simon M. Lucas. Evolving controllers for simulated car racing. In *IEEE Congress on Evolutionary Computation*, pages 1906–1913. IEEE, 2005.
708. Julian Togelius and Simon M. Lucas. Arms races and car races. In *Parallel Problem Solving from Nature-PPSN IX*, pages 613–622. Springer, 2006.
709. Julian Togelius and Simon M. Lucas. Evolving robust and specialized car racing skills. In *IEEE Congress on Evolutionary Computation (CEC)*, pages 1187–1194. IEEE, 2006.
710. Julian Togelius, Simon M. Lucas, Ho Duc Thang, Jonathan M. Garibaldi, Tomoharu Nakashima, Chin Hiong Tan, Itamar Elhanany, Shay Berant, Philip Hingston, Robert M. MacCallum, Thomas Haferlach, Aravind Gowrisankar, and Pete Burrow. The 2007 IEEE CEC Simulated Car Racing Competition. *Genetic Programming and Evolvable Machines*, 9(4):295–329, 2008.
711. Julian Togelius, Mark J. Nelson, and Antonios Liapis. Characteristics of generatable games. In *Proceedings of the Fifth Workshop on Procedural Content Generation in Games*, 2014.
712. Julian Togelius, Mike Preuss, Nicola Beume, Simon Wessing, Johan Hagelbäck, and Georgios N. Yannakakis. Multiobjective exploration of the StarCraft map space. In *Computational Intelligence and Games (CIG), 2010 IEEE Symposium on*, pages 265–272. IEEE, 2010.
713. Julian Togelius, Mike Preuss, and Georgios N. Yannakakis. Towards multiobjective procedural map generation. In *Proceedings of the 2010 Workshop on Procedural Content Generation in Games*. ACM, 2010.
714. Julian Togelius, Tom Schaul, Jürgen Schmidhuber, and Faustino Gomez. Countering poisonous inputs with memetic neuroevolution. In *International Conference on Parallel Problem Solving from Nature*, pages 610–619. Springer, 2008.
715. Julian Togelius, Tom Schaul, Daan Wierstra, Christian Igel, Faustino Gomez, and Jürgen Schmidhuber. Ontogenetic and phylogenetic reinforcement learning. *Künstliche Intelligenz*, 23(3):30–33, 2009.
716. Julian Togelius and Jürgen Schmidhuber. An experiment in automatic game design. In *Computational Intelligence and Games, 2008. CIG'08. IEEE Symposium On*, pages 111–118. IEEE, 2008.
717. Julian Togelius, Noor Shaker, Sergey Karakovskiy, and Georgios N. Yannakakis. The Mario AI championship 2009-2012. *AI Magazine*, 34(3):89–92, 2013.
718. Julian Togelius and Georgios N. Yannakakis. General General Game AI. In *2016 IEEE Conference on Computational Intelligence and Games (CIG)*. IEEE, 2016.

719. Julian Togelius, Georgios N. Yannakakis, Sergey Karakovskiy, and Noor Shaker. Assessing believability. In Philip Hingston, editor, *Believable bots*, pages 215–230. Springer, 2012.
720. Julian Togelius, Georgios N. Yannakakis, Kenneth O. Stanley, and Cameron Browne. Search-based procedural content generation: A taxonomy and survey. *Computational Intelligence and AI in Games, IEEE Transactions on*, 3(3):172–186, 2011.
721. Simone Tognetti, Maurizio Garbarino, Andrea Bonarini, and Matteo Matteucci. Modeling enjoyment preference from physiological responses in a car racing game. In *Computational Intelligence and Games (CIG), 2010 IEEE Symposium on*, pages 321–328. IEEE, 2010.
722. Paul Tozour and I. S. Austin. Building a near-optimal navigation mesh. *AI Game Programming Wisdom*, 1:298–304, 2002.
723. Mike Treanor, Bryan Blackford, Michael Mateas, and Ian Bogost. Game-O-Matic: Generating Videogames that Represent Ideas. In *Procedural Content Generation Workshop at the Foundations of Digital Games Conference*. ACM, 2012.
724. Mike Treanor, Alexander Zook, Mirjam P. Eladhari, Julian Togelius, Gillian Smith, Michael Cook, Tommy Thompson, Brian Magerko, John Levine, and Adam Smith. AI-based game design patterns. 2015.
725. Alan M. Turing. Digital computers applied to games. *Faster than thought*, 101, 1953.
726. Hiroto Udagawa, Tarun Narasimhan, and Shim-Young Lee. Fighting Zombies in Minecraft With Deep Reinforcement Learning. Technical report, Stanford University, 2016.
727. Alfred Ultsch. Data mining and knowledge discovery with emergent self-organizing feature maps for multivariate time series. *Kohonen Maps*, 46:33–46, 1999.
728. Alberto Uriarte and Santiago Ontañón. Automatic learning of combat models for RTS games. In *Eleventh Artificial Intelligence and Interactive Digital Entertainment Conference*, 2015.
729. Nicolas Usunier, Gabriel Synnaeve, Zeming Lin, and Soumith Chintala. Episodic Exploration for Deep Deterministic Policies: An Application to StarCraft Micromanagement Tasks. *arXiv preprint arXiv:1609.02993*, 2016.
730. Josep Valls-Vargas, Santiago Ontañón, and Jichen Zhu. Towards story-based content generation: From plot-points to maps. In *Computational Intelligence in Games (CIG), 2013 IEEE Conference on*. IEEE, 2013.
731. Wouter van den Hoogen, Wijnand A. IJsselsteijn, and Yvonne de Kort. Exploring behavioral expressions of player experience in digital games. In *Proceedings of the Workshop on Facial and Bodily Expression for Control and Adaptation of Games (ECAG)*, pages 11–19, 2008.
732. Roland van der Linden, Ricardo Lopes, and Rafael Bidarra. Procedural generation of dungeons. *Computational Intelligence and AI in Games, IEEE Transactions on*, 6(1):78–89, 2014.
733. Pascal van Hentenryck. *Constraint satisfaction in logic programming*. MIT Press, Cambridge, 1989.
734. Niels van Hoorn, Julian Togelius, and Jürgen Schmidhuber. Hierarchical controller learning in a first-person shooter. In *Computational Intelligence and Games, 2009. CIG 2009. IEEE Symposium on*, pages 294–301. IEEE, 2009.
735. Niels van Hoorn, Julian Togelius, Daan Wierstra, and Jürgen Schmidhuber. Robust player imitation using multiobjective evolution. In *IEEE Congress on Evolutionary Computation (CEC)*, pages 652–659. IEEE, 2009.
736. Giel van Lankveld, Sonny Schreurs, Pieter Spronck, and Jaap van Den Herik. Extraversion in games. In *International Conference on Computers and Games*, pages 263–275. Springer, 2010.
737. Giel van Lankveld, Pieter Spronck, Jaap van den Herik, and Arnoud Arntz. Games as personality profiling tools. In *Computational Intelligence and Games (CIG), 2011 IEEE Conference on*, pages 197–202. IEEE, 2011.
738. Harm van Seijen, Mehdi Fatemi, Joshua Romoff, Romain Laroche, Tavian Barnes, and Jeffrey Tsang. Hybrid Reward Architecture for Reinforcement Learning. *arXiv preprint arXiv:1706.04208*, 2017.
739. Pascal Vincent, Hugo Larochelle, Yoshua Bengio, and Pierre-Antoine Manzagol. Extracting and composing robust features with denoising autoencoders. In *Proceedings of the 25th International Conference on Machine Learning (ICML)*, pages 1096–1103. ACM, 2008.

740. Madhubalan Viswanathan. Measurement of individual differences in preference for numerical information. *Journal of Applied Psychology*, 78(5):741–752, 1993.
741. Thurid Vogt and Elisabeth André. Comparing feature sets for acted and spontaneous speech in view of automatic emotion recognition. In *Proceedings of IEEE International Conference on Multimedia and Expo (ICME)*, pages 474–477. IEEE, 2005.
742. John Von Neumann. The general and logical theory of automata. *Cerebral Mechanisms in Behavior*, 1(41):1–2, 1951.
743. John Von Neumann and Oskar Morgenstern. *Theory of games and economic behavior*. Princeton University Press, 1944.
744. Karol Walędzik and Jacek Mańdziuk. An automatically generated evaluation function in general game playing. *IEEE Transactions on Computational Intelligence and AI in Games*, 6(3):258–270, 2014.
745. Che Wang, Pan Chen, Yuanda Li, Christoffer Holmgård, and Julian Togelius. Portfolio Online Evolution in StarCraft. In *Twelfth Artificial Intelligence and Interactive Digital Entertainment Conference*, 2016.
746. Colin D. Ward and Peter I. Cowling. Monte Carlo search applied to card selection in Magic: The Gathering. In *IEEE Symposium on Computational Intelligence and Games (CIG)*, pages 9–16. IEEE, 2009.
747. Joe H. Ward Jr. Hierarchical grouping to optimize an objective function. *Journal of the American Statistical Association*, 58(301):236–244, 1963.
748. Christopher J. C. H. Watkins and Peter Dayan. Q-learning. *Machine Learning*, 8(3-4):279–292, 1992.
749. Ben G. Weber. ABL versus Behavior Trees. *Gamasutra*, 2012.
750. Ben G. Weber and Michael Mateas. A data mining approach to strategy prediction. In *2009 IEEE Symposium on Computational Intelligence and Games*, pages 140–147. IEEE, 2009.
751. Joseph Weizenbaum. ELIZA—a computer program for the study of natural language communication between man and machine. *Communications of the ACM*, 9(1):36–45, 1966.
752. Paul John Werbos. *Beyond regression: new tools for prediction and analysis in the behavioral sciences*. PhD thesis, Harvard University, 1974.
753. Daan Wierstra, Tom Schaul, Jan Peters, and Juergen Schmidhuber. Natural evolution strategies. In *IEEE Congress on Evolutionary Computation (CEC) 2008. (IEEE World Congress on Computational Intelligence)*., pages 3381–3387. IEEE, 2008.
754. Geraint A. Wiggins. A preliminary framework for description, analysis and comparison of creative systems. *Knowledge-Based Systems*, 19(7):449–458, 2006.
755. Minecraft Wiki. Minecraft. *Mojang AB, Stockholm, Sweden*, 2013.
756. David H. Wolpert and William G. Macready. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 1(1):67–82, 1997.
757. Robert F. Woodbury. Searching for designs: Paradigm and practice. *Building and Environment*, 26(1):61–73, 1991.
758. Steven Woodcock. Game AI: The State of the Industry 2000-2001: It's not Just Art, It's Engineering. *Game Developer Magazine*, 2001.
759. Xindong Wu, Vipin Kumar, J. Ross Quinlan, Joydeep Ghosh, Qiang Yang, Hiroshi Motoda, Geoffrey J. McLachlan, Angus Ng, Bing Liu, S. Yu Philip, Zhi-Hua Zhou, Michael Steinbach, David J. Hand, and Dan Steinberg. Top 10 algorithms in data mining. *Knowledge and Information Systems*, 14(1):1–37, 2008.
760. Kaito Yamamoto, Syunsuke Mizuno, Chun Yin Chu, and Ruck Thawonmas. Deduction of fighting-game countermeasures using the k-nearest neighbor algorithm and a game simulator. In *Computational Intelligence and Games (CIG), 2014 IEEE Conference on*. IEEE, 2014.
761. Yi-Hsuan Yang and Homer H. Chen. Ranking-based emotion recognition for music organization and retrieval. *Audio, Speech, and Language Processing, IEEE Transactions on*, 19(4):762–774, 2011.
762. Georgios N. Yannakakis. *AI in Computer Games: Generating Interesting Interactive Opponents by the use of Evolutionary Computation*. PhD thesis, University of Edinburgh, November 2005.

763. Georgios N. Yannakakis. Preference learning for affective modeling. In *Affective Computing and Intelligent Interaction and Workshops, 2009. ACII 2009. 3rd International Conference on*, pages 1–6. IEEE, 2009.
764. Georgios N. Yannakakis. Game AI revisited. In *Proceedings of the 9th conference on Computing Frontiers*, pages 285–292. ACM, 2012.
765. Georgios N. Yannakakis, Roddy Cowie, and Carlos Busso. The Ordinal Nature of Emotions. In *Affective Computing and Intelligent Interaction (ACII), 2017 International Conference on*, 2017.
766. Georgios N. Yannakakis and John Hallam. Evolving Opponents for Interesting Interactive Computer Games. In S. Schaal, A. Ijspeert, A. Billard, S. Vijayakumar, J. Hallam, and J.-A. Meyer, editors, *From Animals to Animats 8: Proceedings of the* $8^{th}$ *International Conference on Simulation of Adaptive Behavior (SAB-04)*, pages 499–508, Santa Monica, CA, July 2004. The MIT Press.
767. Georgios N. Yannakakis and John Hallam. A Generic Approach for Generating Interesting Interactive Pac-Man Opponents. In *Proceedings of the IEEE Symposium on Computational Intelligence and Games*, 2005.
768. Georgios N. Yannakakis and John Hallam. A generic approach for obtaining higher entertainment in predator/prey computer games. *Journal of Game Development*, 1(3):23–50, December 2005.
769. Georgios N. Yannakakis and John Hallam. Modeling and augmenting game entertainment through challenge and curiosity. *International Journal on Artificial Intelligence Tools*, 16(06):981–999, 2007.
770. Georgios N. Yannakakis and John Hallam. Towards optimizing entertainment in computer games. *Applied Artificial Intelligence*, 21(10):933–971, 2007.
771. Georgios N. Yannakakis and John Hallam. Entertainment modeling through physiology in physical play. *International Journal of Human-Computer Studies*, 66(10):741–755, 2008.
772. Georgios N. Yannakakis and John Hallam. Real-time game adaptation for optimizing player satisfaction. *IEEE Transactions on Computational Intelligence and AI in Games*, 1(2):121–133, 2009.
773. Georgios N. Yannakakis and John Hallam. Rating vs. preference: A comparative study of self-reporting. In *Affective Computing and Intelligent Interaction*, pages 437–446. Springer, 2011.
774. Georgios N. Yannakakis, Antonios Liapis, and Constantine Alexopoulos. Mixed-initiative co-creativity. In *Proceedings of the 9th Conference on the Foundations of Digital Games*, 2014.
775. Georgios N. Yannakakis, Henrik Hautop Lund, and John Hallam. Modeling children's entertainment in the playware playground. In *2006 IEEE Symposium on Computational Intelligence and Games*, pages 134–141. IEEE, 2006.
776. Georgios N. Yannakakis and Manolis Maragoudakis. Player modeling impact on player's entertainment in computer games. In *Proceedings of International Conference on User Modeling (UM)*. Springer, 2005.
777. Georgios N. Yannakakis and Héctor P. Martínez. Grounding truth via ordinal annotation. In *Affective Computing and Intelligent Interaction (ACII), 2015 International Conference on*, pages 574–580. IEEE, 2015.
778. Georgios N. Yannakakis and Héctor P. Martínez. Ratings are Overrated! *Frontiers in ICT*, 2:13, 2015.
779. Georgios N. Yannakakis, Héctor P. Martínez, and Maurizio Garbarino. Psychophysiology in games. In *Emotion in Games: Theory and Praxis*, pages 119–137. Springer, 2016.
780. Georgios N. Yannakakis, Héctor P. Martínez, and Arnav Jhala. Towards affective camera control in games. *User Modeling and User-Adapted Interaction*, 20(4):313–340, 2010.
781. Georgios N. Yannakakis and Ana Paiva. Emotion in games. *Handbook on Affective Computing*, pages 459–471, 2014.
782. Georgios N. Yannakakis, Pieter Spronck, Daniele Loiacono, and Elisabeth André. Player modeling. *Dagstuhl Follow-Ups*, 6, 2013.

783. Georgios N. Yannakakis and Julian Togelius. Experience-driven procedural content generation. *Affective Computing, IEEE Transactions on*, 2(3):147–161, 2011.
784. Georgios N. Yannakakis and Julian Togelius. Experience-driven procedural content generation. In *Affective Computing and Intelligent Interaction (ACII), 2015 International Conference on*, pages 519–525. IEEE, 2015.
785. Georgios N. Yannakakis and Julian Togelius. A panorama of artificial and computational intelligence in games. *IEEE Transactions on Computational Intelligence and AI in Games*, 7(4):317–335, 2015.
786. Xin Yao. Evolving artificial neural networks. *Proceedings of the IEEE*, 87(9):1423–1447, 1999.
787. Nick Yee. The demographics, motivations, and derived experiences of users of massively multi-user online graphical environments. *Presence: Teleoperators and virtual environments*, 15(3):309–329, 2006.
788. Nick Yee, Nicolas Ducheneaut, Les Nelson, and Peter Likarish. Introverted elves & conscientious gnomes: the expression of personality in World of WarCraft. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 753–762. ACM, 2011.
789. Serdar Yildirim, Shrikanth Narayanan, and Alexandros Potamianos. Detecting emotional state of a child in a conversational computer game. *Computer Speech & Language*, 25(1):29–44, 2011.
790. Shubu Yoshida, Makoto Ishihara, Taichi Miyazaki, Yuto Nakagawa, Tomohiro Harada, and Ruck Thawonmas. Application of Monte-Carlo tree search in a fighting game AI. In *Consumer Electronics, 2016 IEEE 5th Global Conference on*. IEEE, 2016.
791. David Young. *Learning game AI programming with Lua*. Packt Publishing Ltd, 2014.
792. R. Michael Young, Mark O. Riedl, Mark Branly, Arnav Jhala, R. J. Martin, and C. J. Saretto. An architecture for integrating plan-based behavior generation with interactive game environments. *Journal of Game Development*, 1(1):51–70, 2004.
793. Mohammed J. Zaki. SPADE: An efficient algorithm for mining frequent sequences. *Machine Learning*, 42(1-2):31–60, 2001.
794. Zhihong Zeng, Maja Pantic, Glenn I. Roisman, and Thomas S. Huang. A survey of affect recognition methods: Audio, visual, and spontaneous expressions. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 31(1):39–58, 2009.
795. Jiakai Zhang and Kyunghyun Cho. Query-efficient imitation learning for end-to-end autonomous driving. *arXiv preprint arXiv:1605.06450*, 2016.
796. Peng Zhang and Jochen Renz. Qualitative Spatial Representation and Reasoning in Angry Birds: The Extended Rectangle Algebra. In *Proceedings of the Fourteenth International Conference on Principles of Knowledge Representation and Reasoning*, 2014.
797. Martin Zinkevich, Michael Johanson, Michael Bowling, and Carmelo Piccione. Regret minimization in games with incomplete information. In *Advances in Neural Information Processing Systems*, pages 1729–1736, 2008.
798. Albert L. Zobrist. *Feature extraction and representation for pattern recognition and the game of Go*. PhD thesis, The University of Wisconsin, Madison, 1970.
799. Alexander Zook. Game AGI beyond Characters. In *Integrating Cognitive Architectures into Virtual Character Design*, pages 266–293. IGI Global, 2016.
800. Alexander Zook and Mark O. Riedl. A Temporal Data-Driven Player Model for Dynamic Difficulty Adjustment. In *8th AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*. AAAI, 2012.
801. Robert Zubek and Ian Horswill. Hierarchical Parallel Markov Models of Interaction. In *AIIDE*, pages 141–146, 2005.