Esto es una prueba
sdjfnsjdfndddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddd
dddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddd
dddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddd
dddddddddddddddddddddddddddddddddddddddddddddddddddd
dddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddd
ddddddddddddddddddddddddgfsgfdfffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff
ffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff
ffffffffffffffffffff
ggggggggggggggggggggggggggggggggggggggggggggggggggggggggggggggggggggggggggggggggggg
ggggggggggggggggggggggggggggggggggggggggggggggggggggggggggggggggggggggggggggggggggg
ggggggggggggggggggggggggggggggggggggggggggggggggggggggggggggggggggggggggggggggggggg
gggggggggggggggggggggggggggggggggggggggggggggggggggggggggggggggggggggggggggg
daaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaa
dddddddddddddddddddddddddddddddddddddddddddddddddddddddccccccccccccccccccccccccccccccccc
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
ccccccccccccccccccccccccccccccccccccccccccccccccc
zzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzz
zzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzxxxxxxxxxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxxxx

cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccas
dhbddhfsdf

4.2.1.1 Type: Necessary Versus Optional
PCG can be used to generate necessary game content that is required for the com pletion of a level, or it can be used to generate optional content that can be discarded
or exchanged for other content. The main distinguishing feature between necessary
and optional content is that necessary content should always be correct while this
condition does not hold for optional content. Necessary needs to be consumed or
passed as the player makes their way through the game, whereas optional content
can be avoided or bypassed. An example of optional content is the generation of dif ferent types of weapons in first-person shooter games [240] or the auxiliary reward
items in Super Mario Bros (Nintendo, 1985). Necessary content can be the main
structure of the levels in Super Mario Bros (Nintendo, 1985), or the collection of
certain items required to pass to the next level.
4.2.2 Taxonomy for Methods
PCG algorithms can be classified according to a number of properties such as their
level of controllability, determinism and so on. This section outlines the three dimen sions across which PCG methods can be classified. Figure 4.2 offers an illustration
of the taxonomy that we discuss in this section.
4.2.2.1 Determinism: Stochastic Versus Deterministic
Our first distinction with regards to PCG methods concerns the amount of random ness in content generation. The right amount of variation in outcomes between dif ferent runs of an algorithm with identical parameters is a design decision. Stochas ticity2
allows an algorithm (such as an evolutionary algorithm) to offer great vari ation, necessary for many PCG tasks, at the cost of controllability. While content

diversity and expressivity are desired properties of generators, the effect of random ness on the final outcomes can only be observed and controlled after the fact. Com pletely deterministic PCG algorithms, on the other hand, can be seen as a form of
data compression. A good example of deterministic PCG is the first-person shooter
.kkrieger (.theprodukkt 2004), which manages to compress all of its textures, ob jects, music and levels together with its game engine in just 96 KB of storage space.
2 Strictly speaking there is a distinction between a stochastic and a non-deterministic process in that the former has a defined random distribution whereas the latter does not. For the purposes of this book, however, we use the two terms interchangeably.

4.2.2.2 Controllability: Controllable Versus Non-controllable
The generation of content by PCG can be controlled in different ways. The use of a random seed is one way to gain control over the generation space; another way is to use a set of parameters that control the content generation along a number of dimensions. Random seeds were used when generating the world in Minecraft (Mojang, 2011), which means the same world can be regenerated if the same seed is used [755]. A vector of content features was used in [617] to generate levels for Infinite Mario Bros (Persson, 2008) that satisfy a set of feature specifications.
4.2.2.3 Iterativity: Constructive Versus Generate-and-Test
A final distinction may be made between algorithms that can be called construc tive and those that can be described as generate-and-test. A constructive algorithm
generates the content once, and is done with it; however, it needs to make sure that the content is correct or at least “good enough” as it is being constructed. An ex ample of this approach is using fractals or cellular automata to generate terrains
or grammars to generate levels (also refer to the corresponding PCG method sec tions below). A generate-and-test algorithm, instead, incorporates both a generate
and a test mechanism. After a candidate content instance is generated, it is tested according to some criteria (e.g., is there a path between the entrance and exit of the dungeon, or does the tree have proportions within a certain range?). If the test fails, all or some of the candidate content is discarded and regenerated, and this process continues until the content is good enough. A popular PCG framework that builds upon the generate-and-test paradigm is the search-based [720] approach discussed in Section 4.3.
4.2.3 Taxonomy of Roles
In this section we identify and briefly outline the four possible roles a PCG algorithm can take in the game design process classified across the dimensions of autonomy and player-based adaptivity. The various PCG roles are illustrated in Fig. 4.2 and are extensively discussed in Section 4.4.
4.2.3.1 Autonomy: Autonomous Versus Mixed-Initiative
The generative process that does not consider any input from the human designer is defined as autonomous PCG whereas mixed-initiative PCG refers to the process that involves the human designer in the creative task. Both roles are discussed in further detail in Section 4.4.

4.2.3.2 Adaptivity: Experience-Agnostic Versus Experience-Driven
Experience-agnostic content generation refers to the paradigm of PCG where con tent is generated without taking player behavior or player experience into account,
as opposed to experience-driven [783], adaptive, personalized or player-centered content generation where player interaction with the game is analyzed and content is created based on a player’s previous behavior. Most commercial games tackle PCG in a generic, experience-agnostic way, while experience-driven PCG has been receiving increasing attention in academia. Recent extensive reviews of PCG for player-adaptive games can be found in [783, 784].
4.3 How Could We Generate Content?
There are many different algorithmic approaches to generating content for games. While many of these methods are commonly thought of as AI methods, others are drawn from graphics, theoretical computer science or even biology. The various

methods differ also in what types of content they are suitable to generate. In this sec tion, we discuss a number of PCG methods that we consider important and include search-based, solver-based, and grammar-based methods but also cellular automata, noise and fractals.

4.3.1 Search-Based Methods

The search-based approach to PCG [720] has been intensively investigated in aca demic PCG research in recent years. In search-based procedural content generation, an evolutionary algorithm or some other stochastic search or optimization algorithm is used to search for content with the desired qualities. The basic metaphor is that of design as a search process: a good enough solution to the design problem exists within some space of solutions, and if we keep iterating and tweaking one or many possible solutions, keeping those changes which make the solution(s) better and discarding those that are harmful, we will eventually arrive at the desired solution. This metaphor has been used to describe the design process in many different disci plines: for example, Will Wright—designer of SimCity (Electronic Arts, 1989) and The Sims (Electronic Arts, 2000)—described the game design process as search in his talk at the 2005 Game Developers Conference. Others have previously described the design process in general, and in other specialized domains such as architecture, the design process can be conceptualized as search and implemented as a computer program [757, 55].

The core components of the search-based approach to solving a content genera tion problem are the following:

- A search algorithm. This is the "engine" of a search-based method. Often rel atively simple evolutionary algorithms work well enough, however sometimes there are substantial benefits to using more sophisticated algorithms that take e.g., constraints into account, or that are specialized for a particular content rep resentation.
- A content representation. This is the representation of the artifacts you want to generate, e.g., levels, quests or winged kittens. The content representation could be anything from an array of real numbers to a graph to a string. The content representation defines (and thus also limits) what content can be generated, and determines whether effective search is possible; see also the discussion about representation in Chapter 2.
- One or more evaluation functions. An evaluation function is a function from an artifact (an individual piece of content) to a number indicating the quality of the artifact. The output of an evaluation function could indicate e.g., the playabil ity of a level, the intricacy of a quest or the aesthetic appeal of a winged kitten.

Crafting an evaluation function that reliably measures the aspect of game qual ity that it is meant to measure is often among the hardest tasks in developing a search-based PCG method. Refer also to the discussion about utility in Chapter 2.

Let us look at some of the choices for content representations. To take a very well-known example, a level in Super Mario Bros (Nintendo, 1985) might be repre sented in any of the following ways:

1. Directly, as a level map, where each variable in the genotype corresponds to one "block" in the phenotype (e.g., bricks, question mark blocks, etc.).
2. More indirectly, as a list of the positions and properties of the different game entities such as enemies, platforms, gaps and hills (an example of this can be found in [611]).
3. Even more indirectly, as a repository of different reusable patterns (such as col lections of coins or hills), and a list of how they are distributed (with various transforms such as rotation and scaling) across the level map (an example of this can be found in [649]).
4. Very indirectly, as a list of desirable properties such as the number of gaps, ene mies, or coins, the width of gaps, etc. (an example of this can be found in [617]).
5. Most indirectly, as a random number seed.

While it clearly makes no sense to evolve random number seeds (it is a represen tation with no locality whatsoever) the other levels of abstraction can all make sense under certain circumstances. The fundamental tradeoff is between more direct, more fine-grained representations with potentially higher locality (higher correlation of fitness between neighboring points in search space) and less direct, more coarse grained representations with probably lower locality but smaller search spaces.
Smaller search spaces are in general easier to search. However, larger search spaces would (all other things being equal) allow for more different types of content to be expressed, or in other words increase the expressive range of the generator.

example study that implements a human-like agent for assessing content quality is presented in [704] where neural-network-based controllers are trained to drive like human players in a car racing game and then used to evaluate the generated tracks. Each track generated is given a fitness value according to statistics calcu lated while the AI controller is playing. Another example of a simulation-based evaluation function is measuring the average fighting time of bots in a first-person shooter game [103]. In that study, levels were simply selected to maximize the amount of time bots spent on the level before killing each other.

Fig. 4.4 A screenshot from the Refraction educational game. A solver-based PCG method (ASP) is used to generate the levels and puzzles of the game. Further details about the application of ASP to Refraction can be found in [635, 89].
out any evaluation functions or re-generation. However, grammar methods can also be used together with search-based methods, so that the grammar expansion is used a genotype-to-phenotype mapping.
A (formal) grammar is a set of production rules for rewriting strings, i.e., turn ing one string into another. Each rule is of the form (symbol(s)) → (other symbol(s)).
Here are some example production rules:
1. A → AB
2. B → b
Expanding a grammar is as simple as going through a string, and each time a symbol or sequence of symbols that occurs in the left-hand side of a rule is found, those symbols are replaced by the right-hand side of that rule. For example, if the initial string is A, in the first rewriting step the A would be replaced by AB by rule 1, and the resulting string will be AB. In the second rewriting step, the A would again be transformed to AB and the B would be transformed to b using rule 2, resulting in the string ABb. The third step yields the string ABbb and so on. A convention in grammars is that upper-case characters are nonterminal symbols, which are on the left-hand side of rules and therefore rewritten further, whereas lower-case characters are terminal symbols which are not rewritten further.

Starting with the axiom A (in L-systems the seed strings are called axioms) the first few expansions look as follows:
A
AB
ABA
ABAAB
ABAABABA
ABAABABAABAAB
ABAABABAABAABABAABABA
ABAABABAABAABABAABABAABAABABAABAB
This particular grammar is an example of an L-system. L-systems are a class of grammars whose defining feature is parallel rewriting, and was introduced by the biologist Aristid Lindenmayer in 1968 explicitly to model the growth of organic systems such as plants and algae [387]. With time, they have turned out to be very

useful for generating plants in games as well as in theoretical biology.
One way of using the power of L-systems to generate 2D (and 3D) artifacts is to interpret the generated strings as instructions for a turtle in turtle graphics. Think of the turtle as moving across a plane holding a pencil, and simply drawing a line that traces its path. We can give commands to the turtle to move forwards, or to turn left or right. For example, we can define the L-system alphabet {F, +, -, [,]} and then use the following key to interpret the generated strings:
• F: move forward a certain distance (e.g., 10 pixels).
• +: turn left 30 degrees.
• −: turn right 30 degrees.
• [: push the current position and orientation onto the stack.
•]: pop the position and orientation off the stack.
Bracketed L-systems can be used to generate surprisingly plant-like structures.
Consider the L-system defined by the single rule F → F[−F]F[+F][F]. Figure 4.5 shows the graphical interpretation of the L-system after 1, 2, 3 and 4 rewrites start ing from the single symbol F. Minor variations of the rule in this system generate different but still plant-like structures, and the general principle can easily be ex tended to three dimensions by introducing symbols that represent rotation along the axis of drawing. For this reason, many standard packages for generating plants in game worlds are based on L-systems or similar grammars. For a multitude of beau tiful examples of plants generated by L-systems refer to the book The Algorithmic Beauty of Plants by Prusinkiewicz and Lindenmayer [542].
There are many extensions of the basic L-system formalism, including non deterministic L-systems, which can help with increasing diversity of the generated content, and context-sensitive L-systems, which can produce more complicated pat terns. Formally specifying L-systems can be a daunting task, in particular as the mapping between the axiom and rules on the one hand and the results after expan sion on the other are so complex. However, search-based methods can be used to
4.3. How Could We Generate Content? 165
geometric interpretations of L-systems were proposed in order to turn them into a versatile tool for fractal and plant modeling. An interpretation based on turtle geome- try, was proposed by Prusinkiewics et al. (1990). The basic idea of turtle interpreta tion is given below.
A state of the turtle is defined as a triplet (x, y, α), where the Cartesian coordinates (x, y) represent the turtle's position, and the angle α, called the heading, is interpreted as the direction in which the turtle is facing. Given the step size d and the angle in- crement δ, the turtle can respond to the commands represented by the following symbols:
F Move forward a step of length d. The state of the turtle changes to (x', y', α), where x' = x + d cos α and y' = y + d sin α. . A line segment between points (x, y) and (x, y') is drawn.
f Move forwards a step of length d without drawing a line. The state of the tur tle changes as above.
+ Turn left by angle δ. The next state of the turtle is (x, y,α +δ).
- Turn left by angle δ. The next state of the turtle is (x, y,α -δ).
To represent branching structures, the L-system alphabet is extended with two new symbols, '[' and ']', to delimit a branch. They are interpreted by the turtle as follows:
[Push the current state of the turtle onto a pushdown stack.
] Pop a state from the stack and make it the current state of the turtle.
Given a string v, the initial state of the turtle (x0 , y0 , α0), and fixed parameters d and δ, the turtle interpretation of v is the figure (set of lines) drawn by the turtle in re- sponse to the string v. This description gives us a rigorous method for mapping strings to pictures, which may be applied to interpret strings generated by L-systems.
An example of a bracketed L-system and its turtle interpretation, obtained in deri- vations of length n = 1 - 4, is shown in Fig. 2. These figures were obtained by inter preting strings generated by the following L-system:

{w: F, p: F → F[-F]F[+F][F]}.
n = 1 n = 2 n = 3 n = 4
Fig. 4.5
Fig. 2.
Four rewrites of the bracketed L-system
Generating a plant-like structure.
F → F[−F]F[+F][F].
find good axioms or rules, using for example desired height or complexity of the
plant in the evaluation function [498].

4.3.4 Cellular Automata

A cellular automaton (plural: cellular automata) is a discrete model of computa tion which is widely studied in computer science, physics, complexity science and
even some branches of biology, and can be used to computationally model biologi cal and physical phenomena such as growth, development, patterns, forms, or even
emergence. While cellular automata (CA) have been the subject of extensive study,
the basic concept is actually very simple and can be explained in a sentence or two:
cellular automata are a set of cells placed on a grid that change through a number of
discrete time steps according to a set of rules; these rules rely on the current state of
each cell and the state of its neighboring cells. The rules can be applied iteratively
for as many time steps as we desire. The conceptual idea behind cellular automata
was introduced by Stanislaw Ulam and John von Neumann [742, 487] back in the
1940s; it took about 30 more years, however, for us to see an application of cel lular automata that showed their potential beyond basic research. That application
was the two-dimensional cellular automaton designed in Conway's Game of Life
[134]. The Game of Life is a zero-player game; its outcome is not influenced by
the player's input throughout the game and it is solely dependent on its initial state
(which is determined by the player).

A cellular automaton contains a number of cells represented in any number of
dimensions; most cellular automata, however, are either one-dimensional (vectors)
or two-dimensional (matrices). Each cell can have a finite number of states; for in stance, the cell can be on or off. A set of cells surrounding each cell define its neigh borhood. The neighborhood defines which cells around a particular cell affect the
cell's future state and its size can be represented by any integer number greater than
1. For one-dimensional cellular automata, for instance, the neighborhood is defined
by the number of cells to the left or the right of the cell. For two-dimensional cellu lar automata, the two most common neighborhood types are the Moore and the von
Neumann neighborhood. The former neighborhood type is a square consisting of
the cells surrounding a cell, including those surrounding it diagonally; for example,
a Moore neighborhood of size 1 contains the eight cells surrounding each cell. The
latter neighborhood type, instead, forms a cross of cells which are centered on the
cell considered. For example, a von Neumann neighborhood of size 1 consists of
the four cells surrounding the cell, above, below, to the left and to the right.

At the beginning of an experiment (at time t = 0) we initialize the cells by as signing a state for each one of them. At each time step t we create a new generation
of cells according to a rule or a mathematical function which specifies the new state
of each cell given the current state of the cell and the states of the cells in its neigh borhood at time t −1. Normally, the rule for updating the state of the cells remains
the same across all cells and time steps (i.e., it is static) and is applied to the whole
grid.

Cellular automata have been used extensively in games for modeling environ mental systems like heat and fire, rain and fluid flow, pressure and explosions
[209, 676] and in combination with influence maps for agent decision making
[678, 677]. Another use for cellular automata has been for thermal and hydraulic
erosion in procedural terrain generation [500]. Of particular interest for the pur poses of this section is the work of Johnson et al. [304] on the generation of infinite

cave-like dungeons using cellular automata. The motivation in that study was to cre ate an infinite cave-crawling game, with environments stretching out endlessly and seamlessly in every direction. An additional design constraint was that the caves are supposed to look organic or eroded, rather than having straight edges and angles. No storage medium is large enough to store a truly endless cave, so the content must be generated at runtime, as players choose to explore new areas. The game does not scroll but instead presents the environment one screen at a time, which offers a time window of a few hundred milliseconds in which to create a new room every time the player exits a room.

The method introduced by Johnson et al. [304] used the following four parame ters to control the map generation process:

• a percentage of rock cells (inaccessible areas) at the beginning of the process;
• the number of CA generations (iterations);
• a neighborhood threshold value that defines a rock;
• the Moore neighborhood size.

(a) A random map (b) A map generated with cellular automata

Fig. 4.6 Cave generation: Comparison between a CA and a randomly generated map. The CA pa rameters used are as follows: the CA runs for four generations; the size of the Moore neighborhood considered is 1; the threshold value for the CA rule is 5 (T = 5); and the percentage of rock cells at the beginning of the process is 50% (for both maps). Rock and wall cells are represented by white and red color respectively. Colored areas represent different tunnels (floor clusters). Images adapted from [304].

In the dungeon generation implementation presented in [304], each room is a 50 × 50 grid, where each cell can be in one of two states: empty or rock. Initially, the grid is empty. The generation of a single room is as follows.

1. The grid is "sprinkled" with rocks: for each cell, there is probability (e.g., 0.5) that it is turned into rock. This results in a relatively uniform distribution of rock cells.
2. A number of CA generations (iterations) are applied to this initial grid.
3. For each generation the following simple rule is applied to all cells: a cell turns into rock in the next iteration if at least T (e.g., 5) of its neighbors are rock, otherwise it will turn into free space.
4. For aesthetic reasons the rock cells that border on empty space are designated as "wall" cells, which are functionally rock cells but look different.

The aforementioned simple procedure generates a surprisingly lifelike cave room. Figure 4.6 shows a comparison between a random map (sprinkled with rocks) and the results of a few iterations of the cellular automaton. But while this process generates a single room, a game would normally require a number of connected rooms. A generated room might not have any openings in the confining rocks, and there is no guarantee that any exits align with entrances to the adjacent rooms. Therefore, whenever a room is generated, its immediate neighbors are also gen erated. If there is no connection between the largest empty spaces in the two rooms, a tunnel is drilled between those areas at the point where they are least separated. A few more iterations of the CA algorithm are then run on all nine neighboring rooms

Fig. 4.7 Cave generation: a 3 × 3 base grid map generated with CA. Rock and wall cells are represented by white and red color respectively; gray areas represent floor. Moore neighborhood size is 2, T is 13, number of CA iterations is 4, and the percentage of rock cells at the initialization phase is 50%. Image adapted from [304].

together, to smooth out any sharp edges. Figure 4.7 shows the result of this process, in the form of nine rooms that seamlessly connect. This generation process is ex tremely fast, and can generate all nine rooms in less than a millisecond on a modern computer. A similar approach to that of Johnson et al. is featured in the Galak-Z (17-bit, 2016) game for dungeon generation [9]. In that game cellular automata gen erate the individual rooms of levels and the rooms are tied together via a variation of a Hilbert curve, which is a continuous fractal space-filling curve [261]. Galak-Z

(17-bit, 2016) shows an alternative way of combining CA with other methods for achieving the desired map generation result.
In summary, CA are very fast constructive methods that can be used effectively to generate certain kinds of content such as terrains and levels (as e.g., in [304]), but they can also be potentially used to generate other types of content. The greatest benefits a CA algorithm can offer to a game content generator is that it depends on a small number of parameters and that it is intuitive and relatively simple to grasp and implement. However, the algorithm's constructive nature is the main cause for

.3.5 Noise and Fractals
One class of algorithms that are very frequently used to generate heightmaps and textures are noise algorithms, many of which are fractal algorithms, meaning that they exhibit scale-invariant properties. Noise algorithms are usually fast and easy to use but lack in controllability.
Both textures and many aspects of terrains can fruitfully be represented as two dimensional matrices of real numbers. The width and height of the matrix map to the x and y dimensions of a rectangular surface. In the case of a texture, this is called an intensity map, and the values of cells correspond directly to the brightness of the associated pixels. In the case of terrains, the value of each cell corresponds to the height of the terrain (over some baseline) at that point. This is called a heightmap. If the resolution with which the terrain is rendered is greater than the resolution of the heightmap, intermediate points on the ground can simply be interpolated between points that do have specified height values. Thus, using this common representation, any technique used to generate noise could also be used to generate terrains, and vice versa—though they might not be equally suitable.
It should be noted that in the case of terrains, other representations are possible and occasionally suitable or even necessary. For example, one could represent the terrain in three dimensions, by dividing the space up into voxels (cubes) and comput ing the three-dimensional voxel grid. An example is the popular open-world game Minecraft (Mojang, 2011), which uses unusually large voxels. Voxel grids allow structures that cannot be represented with heightmaps, such as caves and overhang ing cliffs, but they require a much larger amount of storage.
Fractals [180, 500] such as midpoint displacement algorithms [39] are in com mon use for real-time map generation. Midpoint displacement is a simple algorithm for generating two-dimensional landscapes (seen from the side) by repeatedly sub dividing a line. The procedure is as follows: start with a horizontal line. Find the midpoint of that line, and move the line up or down by a random amount, thus breaking the line in two. Now do the same thing for both of the resulting lines, and so on for as many steps as you need in order to reach sufficient resolution. Ev-

Fig. 4.8 The Midpoint Displacement algorithm visualized.
ery time you call the algorithm recursively, lower the range of the random number generator somewhat (see Fig. 4.8 for an example).
A useful and simple way of extending the midpoint displacement idea to two di mensions (and thus creating two-dimensional heightmaps which can be interpreted as three-dimensional landscapes) is the Diamond-Square algorithm (also known as "the cloud fractal" or "the plasma fractal" because of its frequent use for creating such effects) [210]. This algorithm uses a square 2D matrix with width and height 2n +1. To run the algorithm you normally initialize the matrix by setting the values of all cells to 0, except the four corner values which are set to random values in some chosen range (e.g., [−1,1]). Then you perform the following steps:
1. Diamond step: Find the midpoint of the four corners, i.e., the most central cell in the matrix. Set the value of that cell to the average value of the cor ners. Add a random value to the middle cell.

2. Square step: Find the four cells in between the corners. Set each of those to the average value of the two corners surrounding it. Add a random value to

each of these cells.
Call this method recursively for each of the four subsquares of the matrix, until you reach the resolution limit of the matrix (3 ∗ 3 sub-squares). Every time you call the method, reduce the range of the random values somewhat. The process is illustrated in Fig. 4.9.
There are many more advanced methods for generating fractal noise, with differ ent properties. One of the most important is Perlin noise, which has some benefits over Diamond Square [529]. These algorithms are covered thoroughly in books that focus on texturing and modeling from a graphics perspective [180].

4.3.6 Machine Learning

An emerging direction in PCG research is to train generators on existing content, to be able to produce more content of the same type and style. This is inspired by the recent results in deep neural networks, where network architectures such as generative adversarial networks [232] and variational autoencoders [342] have attained good results in learning to produce images of e.g., bedrooms, cats or faces, and also by earlier results where both simpler learning mechanisms such as Markov chains and more complex architectures such as recurrent neural networks have learned to produce text and music after training on some corpus.
While these kinds of generative methods based on machine learning work well for some types of content—most notably music and images—many types of game content pose additional challenges. In particular, a key difference between game content generation and procedural generation in many other domains is that most game content has strict structural constraints to ensure playability. These constraints differ from the structural constraints of text or music because of the need to play games in order to experience them. A level that structurally prevents players from finishing it is not a good level, even if it is visually attractive; a strategy game map with a strategy-breaking shortcut will not be played even if it has interesting fea tures; a game-breaking card in a collectible card game is merely a curiosity; and so on. Thus, the domain of game content generation poses different challenges from that of other generative domains. The same methods that can produce “mostly cor rect” images of bedrooms and horses, that might still have a few impossible angles or vestigial legs, are less suitable for generating mazes which must have an exit.
This is one of the reasons why machine learning-based approaches have so far only attained limited success in PCG for games. The other main reason is that for many types of game content, there simply isn’t enough existing content to train on. This

(a) Initiate corner values
(b) Perform diamond step (c) Perform square step
(d) Perform diamond step (e) Perform square step
Fig. 4.9 The Diamond-Square algorithm visualized in five steps. Adapted from a figure by Christo pher Ewin, licensed under CC BY-SA 4.0.

is, however, an active research direction where much progress might be achieved in the next few years.

(a) n = 1
(b) n = 2
(c) n = 3
Fig. 4.10 Mario levels reconstructed by n-grams with n set to 1, 2, and 3, respectively.

The core difference between PCG via machine learning and approaches such as search-based PCG is that the content is created directly (e.g., via sampling) from models which have been trained on game content. While some search-based PCG approaches use evaluation functions that have been trained on game content—for instance, the work of Shaker et al. [621] or Liapis et al. [373]—the actual content generation is still based on search. Below, we present some examples of PCG via machine learning; these particular PCG studies built on the use of n-grams, Markov models and artificial neural networks. For more examples of early work along these lines, see the recent survey paper [668].

4.3.6.1 n-grams and Markov Models

For content that can be expressed as one- or two-dimensional discrete structures, such as many game levels, methods based on Markov models can be used. One par ticularly straightforward Markov model is the n-gram model, which is commonly used for text prediction. The n-gram method is very simple—essentially, you build conditional probability tables from strings and sample from these tables when con structing new strings—and also very fast.

Dahlskog et al. trained n-gram models on the levels of the original Super Mario Bros (Nintendo, 1985) game, and used these models to generate new levels [156]. As n-gram models are fundamentally one-dimensional, these levels needed to be converted to strings in order for n-grams to be applicable. This was done through di viding the levels into vertical "slices," where most slices recur many times through out the level [155]. This representational trick is dependent on there being a large

amount of redundancy in the level design, something that is true in many games. Models were trained using various levels of n, and it was observed that while n = 0 creates essentially random structures and n = 1 creates barely playable levels, n = 2 and n = 3 create rather well-shaped levels. See Fig. 4.10 for examples of this.

Summerville et al. [667] extended these models with the use of Monte Carlo tree search to guide the generation. Instead of solely relying on the learned condi tional probabilities, they used the learned probabilities during rollouts (generation of whole levels) that were then scored based on an objective function specified by a designer (e.g., allowing them to bias the generation towards more or less diffi- cult levels). The generated levels could still only come from observed configura tions, but the utilization of MCTS meant that playability guarantees could be made and allowed for more designer control than just editing of the input corpus. This can be seen as a hybrid between a search-based method and a machine learning based method. In parallel, Snodgrass and Onta˜n´on trained two-dimensional Markov Chains—a more complex relative of the n-gram—to generate levels for both Su per Mario Bros (Nintendo, 1985) and other similar platform games, such as Lode Runner (Brøderbund, 1983) [644].

4.3.6.2 Neural Networks

Given the many uses of neural networks in machine learning, and the many dif ferent neural network architectures, it is little wonder that neural networks are also highly useful for machine learning-based PCG. Following on from the Super Mario Bros (Nintendo, 1985) examples in the previous section, Hoover et al. [277] gen erated levels for that same game by extending a representation called functional scaffolding for musical composition (FSMC) that was originally developed to com pose music. The original FSMC representation posits 1) music can be represented as a function of time and 2) musical voices in a given piece are functionally re lated [276]. Through a method for evolving neural networks called NeuroEvolu tion of Augmenting Topologies [655], additional musical voices are evolved to be played simultaneously with an original human-composed voice. To extend this mu sical metaphor and represent Super Mario Bros (Nintendo, 1985) levels as functions of time, each level is broken down into a sequence of tile-width columns. The height of each column extends the height of the screen. While FSMC represents a unit of time by the length of an eighth-note, a unit of time in this approach is the width of each column. At each unit of time, the system queries the ANN to decide a height to place a tile. FSMC then inputs a musical note's pitch and duration to the ANNs. This approach translates pitch to the height at which a tile is placed and duration to the number of times a tile-type is repeated at that height. For a given tile-type or musical voice, this information is then fed to a neural network that is trained on two-thirds of the existing human-authored levels to predict the value of a tile-type at each col umn. The idea is that the neural network will learn hidden relationships between the tile-types in the human-authored levels that can then help humans construct entire levels from as little starting information as the layout of a single tile-type.

Of course, machine learning can also be used to generate other types of game content that are not levels. A fascinating example of this is Mystical Tutor, a design assistant for Magic: The Gathering cards [666]. In contrast to some of the other generators that aim to produce complete, playable levels, Mystical Tutor acknowl edges that its output is likely to be flawed in some ways and instead aims to provide inspirational raw material for card designers.

4.4 Roles of PCG in Games

The generation of content algorithmically may take different roles within the do main of games. We can identify two axes across which PCG roles can be placed:
players and designers. We envision PCG systems that consider designers while they generate content or they operate interdependently of designers; the same applies for players. Figure 4.11 visualizes the key roles of PCG in games across the dimensions of designer initiative and player experience.
Regardless of the generation method used, game genre or content type PCG can act either autonomously or as a collaborator in the design process. We refer to the former role as autonomous generation (Section 4.4.2) and the latter role as mixed initiative (Section 4.4.1) generation. Further, we cover the experience-driven PCG role by which PCG algorithms consider the player experience in whatever they try to generate (Section 4.4.3). As a result, the generated content is associated to the player and her experience. Finally, if PCG does not consider the player as part of the generation process it becomes experience-agnostic (Section 4.4.4).
PCG techniques can be used to generate content in runtime, as the player is playing the game, allowing the generation of endless variations, making the game infinitely replayable and opening the possibility of generating player-adapted con tent, or offline during the development of the game or before the start of a game session. The use of PCG for offline content generation is particularly useful when generating complex content such as environments and maps; several examples of that was discussed at the beginning of the chapter. An example of the use of run time content generation can be found in the game Left 4 Dead (Valve, 2008), a first-person shooter game that provides dynamic experience for each player by an alyzing player behavior on the fly and altering the game state accordingly using PCG techniques [14, 60]. A trend related to runtime content generation is the cre ation and sharing of player-generated content. Some games such as LittleBigPlanet (Sony Computer Entertainment, 2008) and Spore (Electronic Arts, 2008) provide a content editor (level editor in the case of LittleBigPlanet and the Spore Creature Creator) that allows the players to edit and upload complete creatures or levels to a central online server where they can be downloaded and used by other players.
With respect to the four different roles of PCG in games, runtime generation is pos sible in the autonomous and experience-agnostic roles, it is always the case in the experience-driven role whereas it is impossible in the mixed-initiative role. On the other hand, the offline generation of content can occur both autonomously and in

Fig. 4.11 The four key PCG roles in games across the dimensions of designer initiative and player experience. For each combination of roles the figure lists a number of indicative examples of tools or studies covered in this chapter.

an experience-agnostic manner. Further it is exclusively the only way to generate content in a mixed-initiative fashion whereas it is not relevant for experience-driven generation as this PCG role occurs in runtime by definition.
The following four subsections outline the characteristics of each of the four PCG roles with a particular emphasis on the mixed-initiative and the experience-driven roles that have not yet covered in length in this chapter.

4.4.1 Mixed-Initiative

AI-assisted game design refers to the use of AI-powered tools to support the game design and development process. This is perhaps the AI research area which is most promising for the development of better games [764]. In particular, AI can assist in the creation of game content varying from levels and maps to game mechanics and narratives.

Fig. 4.12 The mixed-initiative spectrum between human and computer initiative (or creativity) across a number of mixed-initiative design tools discussed in this section. Iconoscope is a mixed initiative drawing game [372], Sentient Sketchbook is a level editor for strategy games [379], Sen tient World is mixed-initiative map editor [380] and Spaceship Design is a mixed-initiative (mostly computer initiative) tool powered by interactive evolution [377]. Adapted from [375].

We identify AI-assisted game design as the task of creating artifacts via the in teraction of a human initiative and a computational initiative. The computational initiative is a PCG process and thus, we discuss this co-design approach under the heading of procedural content generation. Although the term mixed-initiative lacks a concrete definition [497], in this book we define it as the process that considers both the human and the computer proactively making content contributions to the game design task although the two initiatives do not need to contribute to the same degree [774]. Mixed-initiative PCG thus differs from other forms of co-creation, such as the collaboration of multiple human creators or the interaction between a human and non-proactive computer support tools (e.g., spell-checkers or image ed itors) or non-computer support tools (e.g., artboards or idea cards). The initiative of the computer can be seen as a continuum between the no initiative state, leaving full control of the design to the designer and having the computer program simply carry out the commands of the human designer, to the full initiative state which yields an autonomously creative system. Any state between the two is also possible as we will see in the examples below and as depicted in Fig. 4.12.

4.4.1.1 Game Domains

While the process of AI-assisted game design is applicable to any creative facets within game design [381] it is level design that has benefited the most from it. Within commercial-standard game development, we can find AI-based tools that allow varying degrees of computer initiative. On one end of the scale, level editors such as the Garden of Eden Creation Kit (Bethesda, 2009) or game engines such

as the Unreal Development Kit (Epic Games 2009) leave most of creative process to the designer but they, nevertheless, boost game development through automat ing interpolations, pathfinding and rendering [774]. On the other end of the com puter initiative scale, PCG tools specialized on e.g., vegetation—SpeedTree (IDV, 2002)—or FPS levels—Oblige (Apted, 2007)—only require the designer to set a small amount of generation parameters and, thus, the generation process is almost entirely autonomous.

Within academia, the area of AI-assisted game design tools has seen significant research interest in recent years [785] with contributions mainly to the level de sign task across several game genres including platformers [641], strategy games [380, 379, 774, 378] (see Fig. 4.13(a)), open world games [634], racing games [102], casual puzzle games [614] (see Fig. 4.13(b)), horror games [394], first-person shooters [501], educational games [89, 372], mobile games [482], and adventure games [323]. The range of mixed-initiative game design tools expands to tools that are designed to assist with the generation of complete game rulesets such as the MetaGame [522], the RuLearn [699] and the Ludocore [639] tools to tools that are purposed to generate narratives [480, 673] and stories within games [358].

4.4.1.2 Methods

Any PCG approach could potentially be utilized for mixed-initiative game design. The dominant methods that have so far being used, however, rely on evolutionary computation following the search-based PCG paradigm. Even though evolution, at fist sight, does not appear to be the most attractive approach for real-time processing and generation of content, it offers great advantages associated, in particular, with the stochasticity of artificial evolution, diversity maintenance and potential for bal ancing multiple design objectives. Evolution can be constrained to the generation of playable, usable, or, in general, content of particular qualities within desired design specifications. At the same time, it can incorporate metrics such as novelty [382] or surprise [240], for maximizing the diversity of generated content and thus enabling a

change in the creative path of the designer [774]. Evolution can be computationally costly, however, and thus, interactive evolution is a viable and popular alternative for mixed-initiative evolutionary-based generation (e.g., see [102, 380, 377, 501]).
Beyond artificial evolution, another class of algorithms that is relevant for mixed initiative content generation is constraint solvers and constraint optimization. Meth ods such as answer set programming [383, 69] have been used in several AI-assisted level design tools including Tanagra [641] for platformers and Refraction [89] for educational puzzle games. Artificial neural networks can also perform certain tasks in a mixed-initiative manner, such as performing "autocomplete" or level repair [296] through the use of deep learning approachers such as stacked autoencoders.
The goal here is to provide a tool that "fills in" parts of a level that the human de signer does not want or have time to create, and correcting other parts to achieve further consistency.
4.4. Roles of PCG in Games 179
(a) Sentient Sketchbook
(b) Ropossum
Fig. 4.13 Examples of mixed-initiative level design tools. Sentient Sketchbook (a) offers map sketch suggestions to the designer via artificial evolution (see rightmost part of the image); the suggestions are evolved to either maximize particular objectives of the map (e.g., balance) or are evolved to maximize the novelty score of the map. In Ropossum (b) the designer may select to design elements of Cut the Rope (Chillingo, 2010) game levels; the generation of the remaining elements are left to evolutionary algorithms to design.
180 Chapter 4. Generating Content
4.4.2 Autonomous
The role of autonomous generation is arguably the most dominant PCG role in games. The earlier parts of this chapter are already dedicated to extensive discus sions and studies of PCG systems that do not consider the designer in their creative process. As a result we will not cover this PCG role in further detail here. What is important to be discussed, however, is the fuzzy borderline between mixed-initiative and autonomous PCG systems. It might be helpful, for instance, to consider au tonomous PCG as the process by which the role of the designer starts and ends with an offline setup of the algorithm. For instance, the designer is only involved in the parameterization of the algorithm as in the case of SpeedTree (IDV, 2002).
One might wish, however, to further push the borderline between autonomous and mixed-initiative generation and claim that generation is genuinely autonomous only if the creative process reconsiders and adapts the utility function that drives the con tent generation—thereby becoming creative in its own right. A static utility function that drives the generation is often referred to as mere generation within the compu tational creativity field [381].
While the line between autonomy and collaboration with designers is still an open research question, for the purposes of this book, we can safely claim that the PCG process is autonomous when the initiative of the designer is limited to algo rithmic parameterizations before the generation starts.
4.4.3 Experience-Driven
As games offer one of the most representative examples of rich and diverse content creation applications and are elicitors of unique user experiences experience-driven PCG (EDPCG) [783, 784] views game content as the building block of games and the generated games as the potentiators of player experience. Based on the above, EDPCG is defined as a generic and effective approach for the optimization of user (player) experience via the adaptation of the experienced content. According to the experience-driven role of PCG in games player experience is the collection of af fective patterns elicited, cognitive processes emerged and behavioral traits observed during gameplay [781].
By coupling player experience with procedural content generation, the experience driven perspective offers a new, player-centered role to PCG. Since games are com posed by game content that, when played by a particular player, elicits experience

patterns, one needs to assess the quality of the content generated (linked to the ex perience of the player), search through the available content, and generate content that optimizes the experience for the player (see Fig. 4.14). In particular, the key components of EDPCG are:

• Player experience modeling: player experience is modeled as a function of game content and player.

Fig. 4.14 The four key components of the experience-driven PCG framework.

• Content quality: the quality of the generated content is assessed and linked to the modeled experience.

• Content representation: content is represented accordingly to maximize search efficacy and robustness.

• Content generator: the generator searches through the generative space for con tent that optimizes the experience for the player according to the acquired model.

Each component of EDPCG has its own dedicated literature and the extensive review of each is covered in other parts of the book. In particular, player experi ence modeling is covered in Chapter 5 whereas the remaining three components of the framework have already been covered in this chapter. A detailed survey and discussion about EDPCG is available in [783].

4.4.3.1 Experience-Driven PCG in Practice

Left 4 Dead (Valve, 2008) is an example of the use of experience-driven PCG in a commercial game where an algorithm is used to adjust the pacing of the game on the fly based on the player's emotional intensity. In this case, adaptive PCG is used to adjust the difficulty of the game in order to keep the player engaged [60]. Adap tive content generation can also be used with another motive such as the generation of more content of the kind the player seems to like. This approach was followed, for instance, in the Galactic Arms Race [250] game where the weapons presented to the player are evolved based on her previous weapon use and preferences. In an other EDPCG study, El-Nasr et al. implemented a direct fitness function—derived from visual attention theory—for the procedural generation of lighting [188, 185].

The procedural Zelda game engine [257], a game engine designed to emulate the

(a) Human

(b) World-Champion AI

Fig. 4.15 Example levels generated for two different Mario players. The generated levels maxi mize the modeled fun value for each player. The level on top is generated for one of the experiment subjects that participated in [521] while the level below is generated for the world champion agent of the Mario AI competition in 2009.

popular The Legend of Zelda (Nintendo, 1986–2017) action-RPG game series, is built mainly to support experience-driven PCG research. Another example is the work by Pedersen et al. [521], who modified an open-source clone of the classic platform game Super Mario Bros (Nintendo, 1985) to allow for personalized level generation. The realization of EDPCG in this example is illustrated in Fig. 4.16.

The first step was to represent the levels in a format that would yield an easily searchable space. A level was represented as a short parameter vector describing the number, size and placement of gaps which the player can fall through, and the presence or absence of a switching mechanic. The next step was to create a model of player experience based on the level played and the player's playing style. Data was collected from hundreds of players, who played pairs of levels with different parameters and were asked to rank which of two levels best induced each of the following user states: fun, challenge, frustration, predictability, anxiety, boredom. While playing, the game also recorded a number of metrics of the players' playing styles, such as the frequency of jumping, running and shooting. This data was then used to train neural networks to predict the examined user states using evolutionary preference learning. Finally, these player experience models were utilized to opti mize game levels for particular players [617]. Two examples of such levels can be seen in Fig. 4.15. It is worth noting—as discussed in Chapter 5—that one may wish

to further improve the models of experience of Mario players by including infor mation about the player beyond gameplay [29] such as her head pose [610] or her facial expressions [52].

4.4.4 Experience-Agnostic

With experience-agnostic PCG we refer to any PCG approach that does not con sider the role of the player in the generation of content. But where should we set the boundary of involvement? When do we consider a player as part of the gener ation process and when don't we? While the borderline between experience-driven and experience-agnostic is not trivial to draw we define any PCG approach whose content quality function does not include a player (experience) model or it does not

Fig. 4.16 The EDPCG framework in detail. The gradient grayscale-colored boxes represent a con tinuum of possibilities between the two ends of the box while white boxes represent discrete, exclusive options within the box. The blue arrows illustrate the EDPCG approach followed for the Super Mario Bros (Nintendo, 1985) example study [521, 617]: Content quality is assessed via a direct, data-driven evaluation function which is based on a combination of a gameplay-based (model-free) and a subjective (pairwise preference) player experience modeling approach; content is represented indirectly and exhaustive search is applied to generate better content.

interact with the player in any way during generation as experience-agnostic. As with the role of autonomous PCG, this chapter has already gone through several examples of content generation that do not involve a player or a player experience model. To avoid being repetitive we will refer the reader to the PCG studies covered already that are outside the definition of experience-driven PCG.

4.5 What Could Be Generated?

In this section we briefly outline the possible content types that a PCG algorithm can generate in a game. Generally speaking Liapis et al. [381] identified six creative domains (or else facets) within games that we will follow for our discussion in this section. These include level architecture (design), audio, visuals, rules (game design), narrative, and gameplay. In this chapter we will cover the first five facets and we purposely exclude the gameplay facet. Creative gameplay is directly associated with play and as such is covered in the previous chapter. We conclude this section with a discussion on complete game generation.

4.5.1 Levels and Maps

The generation of levels is by far the most popular use of PCG in games. Levels can be viewed as necessary content since every game has some form of spatial representation or virtual world within which the player can perform a set of actions. The properties of the game level, in conjunction with the game rules, frame the ways a player can interact with the world and determine how the player can progress from one point in the game to another. The game's level design contributes to the challenges a player faces during the game. While games would often have a fixed set of mechanics throughout, the way a level is designed can influence the gameplay and the degree of game challenge. For that reason, a number of researchers have argued that levels coupled with game rules define the absolutely necessary building blocks of any game; in that regard the remaining facets covered below are optional [371].

The variations of possible level designs are endless: a level representation can vary from simple two-dimensional illustrations of platforms and coins—as in the Super Mario Bros (Nintendo, 1985) series—to the constrained 2D space of Candy Crush Saga (King, 2012), to the three-dimensional and large urban spaces of Assassin's Creed (Ubisoft, 2007) and Call of Duty (Infinity Ward, 2003), to the 2D elaborated structures of Angry Birds (Rovio, 2009), to the voxel-based open gameworld of Minecraft (Mojang 2011).

Due to their several similarities we can view the procedural generation of game levels from the lens of procedural architecture. Similarly to architecture, level design needs to consider both the aesthetic properties and the functional requirements of whatever is designed within the game world. Depending on the game genre, func tional requirements may vary from a reachable end-goal for platform games, to a

challenging gameplay in driving games such as Forza Motorsport (Turn 10 Stu dios 2005), to waves of gameplay intensity as in Pac-Man (Namco, 1980), Left 4
Dead (Valve, 2008), Resident Evil 4 (Capcom, 2005) and several other games. A
procedural level generator also needs to consider the aesthetics of the content as
the level's aesthetic appearance may have a significant impact not only on the vi sual stimuli it offers to the player but also on navigation. For example, a sequence
of identical rooms can easily make the player disoriented—as was intended in the
4.5. What Could Be Generated? 185
Fig. 4.17 The procedurally generated levels of Diablo (Blizzard Entertainment, 1996): one of the most characteristic examples of level generation in commercial games. Diablo is a relatively recent descendant of Rogue (Toy and Wichmann, 1980) (i.e., rogue-like) role-playing game characterized by dungeon-based procedurally generated game levels. Image obtained from Wikipedia (fair use).
dream sequences of Max Payne (Remedy, 2001)—while dark areas can add to the
challenge of the game due to low visibility or augment the player's arousal as in
the case of Amnesia: The Dark Descent (Frictional Games, 2010), Nevermind (Fly ing Mollusk, 2015) and Sonancia [394]. When the level generator considers larger,
open levels or gameworlds then it draws inspiration from urban and city planning
[410], with edges to constrain player freedom—landmarks to orient the player and
motivate exploration [381]—as in the Grand Theft Auto (Rockstar Games, 1997) se ries and districts and areas to break the world's monotony—as in World of Warcraft
(Blizzard Entertainment, 2004) which uses highly contrasting colors, vegetation and
architectural styles to split the world into districts that are suitable for characters of
different level ranges.
As already seen broadly in this chapter the generation of levels in a procedural
manner is clearly the most popular and possibly the oldest form of PCG in the game
industry. We already mentioned the early commercial use of PCG for automatic level
design in games such as Rogue (Toy and Wichman, 1980) and the Rogue-inspired
Diablo (see Fig. 4.17) series (Blizzard Entertainment, 1996), and the more recent
world generation examples of Civilization IV (Firaxis, 2005) and Minecraft (Mo jang, 2011). The level generation algorithms used in commercial games are usually
186 Chapter 4. Generating Content
Fig. 4.18 The caricaturized and highly-emotive visuals of the game Limbo (Playdead, 2010). Im age obtained from Wikipedia (fair use).
constructive, in particular, in games where players can interact with and change the
game world via play. Players of Spelunky (Yu and Hull, 2009), for instance, are al lowed to modify a level which is not playable (i.e., the exit cannot be reached) by
blowing up the blocking tiles with a bomb provided by the game level.
The academic interest in procedural level generation is only recent [720, 783,
616] but it has produced a substantial volume of studies already. Most of the aca demic studies described in this chapter are focused on levels. The interested reader
may refer to those for examples of level generation across various methodologies,
PCG roles and game genres.
4.5.2 Visuals
Games are, by definition, visual media unless the game is designed explicitly to
not have visuals—e.g., the Real Sound: Kaze no Regret (Sega, 1997) adventure au dio game. The visual information displayed on the screen conveys messages to the
player which are dependent on the graphical style, color palette and visual texture.
Visuals in games can vary from simple abstract and pixelized representations as the
8-bit art of early arcade games, to caricaturized visuals as in Limbo (Playdead, 2010)
(see Fig. 4.18), to photorealistic graphics as in the FIFA series (EA Sports, 1993)
[299].
Within the game industry PCG has been used broadly for the generation of any
of the above visual types. Arguably, the complexity of the visual generation task
increases the more the resolution and the photorealism of the desired output in creases. There are examples of popular generative tools such as SpeedTree (IDV,
2002) for vegetation and FaceGen (Singular Inversions, 2001) for faces, however,

that can successfully output photorealistic 3D visuals. Within academia notable ex amples of visual generation are the games Petalz [565, 566] (flower generation),
Galactic Arms Race [250] (weapon generation; see also Fig. 4.3) and AudioInSpace

[275] (weapon generation). In all three games visuals are represented by neural networks that are evolved via interactive evolution. Within the domain of weapon particle generation another notable example is the generation of surprising yet bal anced weapons for the game Unreal Tournament III (Midway Games, 2007) using
constrained surprise search [240]; an algorithm that maximizes the surprise score of a weapon but at the same time imposes designer constraints to it so that it is bal anced. Other researchers have been inspired by theories about "universal" properties of beauty [18] to generate visuals of high appeal and appreciation [377]. The PCG algorithm in that study generates spaceships based on their size, simplicity, balance and symmetry, and adapts its visual output to the taste of the visual's designer via interactive evolution. The PCG-assisted design process referred to as iterative refinement [380] is another way of gradually increasing the resolution of the visuals a designer creates by being engaged in an iterative and creative dialog with the visuals generator. Beyond the generation of in-game entities a visuals PCG algorithm may focus on general properties of the visual output such as pixel shaders [282], lighting [188, 185], brightness and saturation, which can all influence the overall appearance of any game scene.

4.5.3 Audio

Even though audio can be seen as optional content it can affect the player directly and its impact on player experience is apparent in most games [219, 221, 129]. Au dio in games has reached a great level of maturity as demonstrated by two BAFTA
Game Awards and an MTV video music award for best video game soundtrack [381]. The audio types one meets in games may vary from fully orchestrated sound track (background) music, as in Skyrim (Bethesda, 2011), to sound effects, as the dy ing or pellet-eating sounds of Pac-Man (Namco, 1980), to the voice-acted sounds of
Fallout 3 (Bethesda, 2008). Most notably within indie game development, Proteus (Key and Kanaga, 2013) features a mapping between spatial positioning, visuals and player interaction, which collectively affect the sounds that are played. Profes sional tools such as the sound middleware of UDK (Epic Games, 2004) and the
popular sfxr and bfxr sound generation tools provide procedural sound components to audio designers, demonstrating a commercial interest in and need of procedurally generated audio.

At a first glance, the generation of game audio, music and sounds might not seem to be particularly different from any other type of audio generation outside games. Games are interactive, however, and that particular feature makes the generation of audio a rather challenging task. When it comes to the definition of procedural audio in games, a progressive stance has been that its procedurality is caused by the very interaction with the game. (For instance, game actions that cause sound effects of music can be considered as procedural audio creation [220, 128].) Ideally, the game audio must be able to adapt to the current game state and the player behavior. As a result adaptive audio is a grand challenge for composers since the combinations of

all possible game and player states could be largely unknown. Another difficulty for the autonomous generation of adaptive music, in particular, is that it requires real time composition and production; both of which need to be embedded in a game
engine. Aside from a few efforts in that direction3
the current music generation
models are not particularly designed to perform well in games. In the last decade, however, academic work on procedural game music and sound has seen substantial advancements in venues such as the Procedural Content Generation and the Musical Metacreation workshop series.

Generally speaking, sound can be either diegetic or non-diegetic. A sound is diegetic if its source is within the game's world. The source of the diegetic sound

can be visible on the screen (on-screen) or can be implied to be present at the current game state (off-screen). Diegetic sounds include characters, voices, sounds made by items on-screen or interactions of the player, or even music represented as coming from instruments available in the game. A non-diegetic sound, on the other hand, is any sound whose source is outside the game's world. As such non-diegetic sounds cannot be visible on-screen or even implied to be off-screen. Examples include com mentaries of a narrator, sound effects which are not linked to game actions, and background music.

A PCG algorithm can generate both diegetic and non-diegetic sounds including music, sound effects and commentaries. Examples of non-diegetic sound generation in games include the Sonancia horror sound generator that tailors the tension of the game to the desires of the designer based on the properties of the game level [394]. The mapping between tension and sounds in Sonancia has been derived through crowdsourcing [396]. Similarly to Sonancia—and towards exploring the creative space between audio and level design—Audioverdrive generates levels from audio and audio from levels [273]. Notably within diegetic audio examples, Scirea et al. [606] explores the relationship between procedurally generated music and narrative. Studies have also considered the presence of game characters on-display for the composition of game soundtracks [73] or the combination of short musical phrases that are driven by in-game events and, in turn, create responsive background audio for strategy games [280].

Finally, it is worth mentioning that there are games featuring PCG that use music as the input for the generation of other creative domains rather than music per se. For example, games such as Audio Surf (Fitterer, 2008) and Vib Ribbon (Sony En tertainment, 2000) do not procedurally generate music but they instead use music to drive the generation of levels. AudioInSpace [275] is another example of a side scrolling space shooter game that does not generate music but uses the background music as the basis for weapon particle generation via artificial evolution.

3 For instance, see the upcoming melodrive app at: http://melodrive.com.

4.5.4 Narrative

Many successful games are relying heavily on their narratives; the clear distinction however, between such narratives and traditional stories is the interactivity element that is offered by games. Now, whether games can tell stories [313] or games are instead a form of narrative [1] is still an open research question within game stud ies, and beyond the scope of this book. The study of computational (or procedural) narrative focuses on the representational and generational aspects of stories as those can be told via a game. Stories can play an essential part in creating the aesthetics of a game which, in turn, can impact affective and cognitive aspects of the playing experience [510].

By breaking the game narrative into subareas of game content we can find core game content elements such as the game's plotline [562, 229], but also the ways a story is represented in the game environment [730, 83]. The coupling of a game's representation and the story of the game is of vital importance for player experi ence. Stories and plots are taking place in an environment and are usually told via a virtual camera lens. The behavior of the virtual camera—viewed as a parame terized element of computational narrative—can drastically influence the player's experience. That can be achieved via an affect-based cinematographic representa tion of multiple cameras as those used in Heavy Rain (Quantic Dream, 2010) or through an affect-based automatic camera controller as that used in the Maze-Ball game [780]. Choosing the best camera angle to highlight an aspect of a story can be seen as a multi-level optimization problem, and approached with combinations of optimization algorithms [85]. Games such as World of Warcraft (Blizzard Entertain ment, 2004) use cut scenes to raise the story's climax and lead the player to partic ular player experience states. The creation or semi-automatic generation of stories and narratives belongs to the area of interactive storytelling, which can be viewed as a form of story-based PCG. The story can adjust according to the actions of the

player targeting personalized story generation (e.g., see [568, 106] among others).
Ultimately, game worlds and plot point story representations can be co-generated as demonstrated in a few recent studies (e.g., see [248]).
Computational narrative methods for generating or adapting stories of exposi tions are typically build on planning algorithms, and planning is therefore essential for narrative [792]. The space of stories can be represented in various ways, and the representations in turn make use of dissimilar search/planning algorithms, includ ing traditional optimization and reinforcement learning approaches [483, 117, 106].
Cavazza et al. [106], for instance, introduced an interactive storytelling system built with the Unreal game engine that uses Hierarchical Task Network planning to sup port story generation and anytime user intervention. Young et al. [792] introduced an architecture called Mimesis, primarily designed to generate intelligent, plan-based character and system behavior at runtime with direct uses in narrative generation.
Finally the IDtension engine [682] dynamically generates story paths based on the

player's choices; the engine was featured in Nothing for Dinner,4 a 3D interactive story aiming to help teenagers living challenging daily life situations at home.
Similarly to dominant approaches of narrative and text generation, interactive storytelling in games relies heavily on stored knowledge about the (game) world. Games that rely on narratives—such as Heavy Rain (Quantic Dream, 2010)—may include thousands of lines of dialog which are manually authored by several writers. To enable interactive storytelling the game should be able to select responses (or paths in the narrative) based on what the player will do or say, as in Fac¸ade [441] (see Fig. 4.19). To alleviate, in part, the burden of manually representing world knowledge, data-driven approaches can be used. For instance, one may crowdsource actions and utterance data from thousand of players that interact with virtual agents of a game and then train virtual agents to respond in similar ways using n-grams [508]. Or instead, one might design a system in which designers collaborate with a computer by taking turns on adding sentences in a story; the computer is able to provide meaningful sentences by matching the current story with similar stories available on the cloud [673]. Alternatively, a designer could use the news of the day from sites, blogs or Wikipedia and generate games that tell the news implicitly via play [137].
Research on interactive narrative and story-based PCG benefits from and influences the use of believable agents that interact with the player and are inter woven in the story plot. The narrative can yield more (or less) believability to agents and thus the relationship between agents and the story they tell is impor tant [801, 401, 531, 106]. In that sense, the computational narrative of a game may define the arena for believable agent design. Research on story-based PCG has also influenced the design and development of games. Starting from popular indepen dent attempts like Fac¸ade [441] (see Fig. 4.19), Prom Week [448] and Nothing for Dinner to the commercial success of The Elder Scrolls V: Skyrim (Bethesda Soft works, 2011), Heavy Rain (Quantic Dream, 2010) and Mass Effect (Bioware, 2007) narrative has traditionally been amongst the key factors of player experience and immersion; particularly in narrative-heavy games as the ones aforementioned.
Examples of sophisticated computational narrative techniques crossing over from academia to commercial-standard products include the storytelling system Versu [197] which was used to produce the game Blood & Laurels (Emily Short, 2014). For the interested reader the interactive fiction database5 contains a detailed list of games built on the principles of interactive narratives and fiction, and the stoy gen.org6 repository, by Chris Martens and Rogelio E. Cardona-Rivera, maintains existing openly-available computational story generation systems. Finally note that the various ways AI can be used to play text-based adventure games and interactive fiction are covered in Chapter 3.
4 http://nothingfordinner.org

5 http://ifdb.tads.org/
6 http://storygen.org/
4.5. What Could Be Generated? 191
Fig. 4.19 A screenshot from the Fac¸ade [441] game featuring its main characters: Grace and Trip. The seminal design and AI technologies of Fac¸ade popularized the vision of interactive narrative and story-based PCG within games.
4.5.5 Rules and Mechanics
The game rules frame the playing experience by providing the conditions of play—for instance, winning and losing conditions—and the actions available to the player (game mechanics). Rules constitute necessary content as they are in a sense the core of any game, and a game's rules pervade it.
For most games, the design of their ruleset largely defines them and contributes to their success. It is common that the rule set follows some standard design patterns within its genre. For example, the genre of platform games is partly defined by run ning and jumping mechanics, whereas these are rare in puzzle games. Evidently, the genre constrains the possibility (design) space of the game rules. While this prac tice has been beneficial—as rule sets are built on earlier successful paradigms—it can also be detrimental to the creativity of the designer. It is often the case that the players themselves can create new successful game variants (or even sub-genres) by merely altering some rules of an existing game. A popular example is the modification of Warcraft III (Blizzard, 2002) which allowed the player to control a single "hero" unit and, in turn, gave rise to a new, popular game genre named Multiplayer Online Battle Arenas (MOBA).
192 Chapter 4. Generating Content
Most existing approaches to rule generation take a search-based approach, and are thus dependent on some way of evaluating a set of rules [711, 355]. However, accurately estimating the quality of a set of game rules is very hard. Game rules differ from most other types of game content in that they are almost impossible to evaluate in isolation from the rest of the game. While levels, characters, textures, and many other types of content can to some extent be evaluated outside of the game, looking at a set of rules generally gives very little information of how they play. For a human, the only way to truly experience the rules of a game is to play the game. For a computer, this would translate to simulating gameplay in some way in order to evaluate the rules. (In this sense, rules can be said to be more similar to program code than they are to e.g., pictures or music.)
So how can simulated playthroughs be used to judge the quality of the rulesets? Several ideas about how to judge a game depending on how agents play it have been introduced. The first is balance; for symmetric two-player games in particular, bal ance between the winning chances of the two players is generally positive [274].
Another idea is outcome uncertainty, meaning that any particular game should be "decided" as late as possible [76]. Yet another idea is learnability: a good game, including its ruleset, is easy to learn and hard to master. In other words, it should have a long, smooth learning curve for the player, as learning to play the game is part of what makes it fun. This idea can be found expressed most clearly in Koster's "Theory of Fun" [351], but can also be said to be implicit in Schmidhuber's the ory of artificial curiosity [602] and in theories in developmental psychology [204].
Within work in game rule generation, attempts have been made to capture this idea in different ways. One way is to use a reinforcement learning agent to try to learn to play the game; games where the agent improves the most over a long time score the best [716]. Another way of capturing this idea is to use several agents of different skill levels to try to play the game. Games score better when they maximize the performance difference between these agents [491]. This idea is also related to the idea of depth in games, which can be seen as the length of the chain of heuristics that can be acquired in a game [362].
Perhaps the most successful example of game rule generation within academic research is Ludi [76]. Ludi follows a search-based PCG approach and evolves gram mars that represent the rules of board games (see Fig. 4.20). The fitness function

that drives the rule generation is composed by several metrics that estimate good design patterns in board games, such as game depth and rule simplicity. A success fully designed game that came out of the Ludi generator is named Yavalath [75] (see Fig. 4.20). The game is played on a 5-by-5 hexagonal board by two or three players. Yavalath's winning and losing conditions are very simple: you win if you make a line of four tokens, whereas you lose if you make a line of three tokens; the game is a draw if the board fills up.

One of the earliest examples of video game rule generation is Togelius and Schmidhuber's experiment with generating simple Pac-Man-like games [716]. In that study rules are evolved to maximize the learnability of player agents as mea sured via simulated playthroughs. Another example is the work by Nielsen et al. in which game rules are represented using the video game description language [492].

Using Answer Set Programming (a solver-based method) [69] rather than search based methods, rules have been generated for simple 2D games that, however, re spect constraints such as playability (i.e., the victory condition is attainable) [637].

It is fair to say that none of these attempts to generate video games have been able to produce good games, i.e., games that anyone (except the creator of the system that creates the games) would want to play. This points to the immense challenge in accurately estimating the quality of a video game rule set. One of the reasons this seems to be a more challenging problem than estimating the quality of a board game rule set is the time dimension, as human reaction time and ability to estimate and predict unfolding processes play an important role in the challenge of many video games.

For more examples and in-depth analysis on rule and mechanics generation the interested reader is referred to the "rules and mechanics" chapter of the PCG book [486].

4.5.6 Games

Game generation refers to the use of PCG algorithms for computationally design ing new complete games. The vast majority of PCG studies so far, however, have been very specific to a particular game facet or domain. It is, for instance, either a level that is generated or the audio for some level but rarely both. Meanwhile it is surprising to think that the relationship between the different facets is naturally interwoven. A characteristic example of the interwoven nature among game facets is given in [381]: player actions—viewed as a manifestation of game rules—are usually accompanied by corresponding sound effects such as the sound of Mario jumping in Super Mario Bros (Nintendo, 1985). Now let us think of a PCG al gorithm that introduces a new rule to the game—hence a new player action. The algorithm automatically constrains the sound effects that can be associated to this new action based on a number of factors such as the action's duration, purpose and overall contribution to the game plot. Actions and sounds appear to have a cause and effect (or hierarchical) relationship and a PCG algorithm would naturally prioritize the creation of the action before it generates its sound. Most relationships between facets, however, are not strictly hierarchical or unidirectional. For example, a game level can be successful because of a memorable landmark as much as the gameplay it affords [381]. Similarly, the narrative of a game relies on a multitude of factors including the camera placement as well as the visuals and the sounds.

The game generation topic has attracted a growing interest in recent years even though the relationship between the different games facets is not considered largely. Most game generation projects focus on a single facet of a game and do not inves tigate the interaction between different facets. The rule generator for Pac-Man-like games [716], for instance, evolves different rules for different colored agents but it does not evolve the agents' color to indicate different playing strategies. Similarly

(a) The Ludi game rule generator

(b) A deluxe version of the Yavalath game

Fig. 4.20 The Ludi game rule generator (a) and its game Yavalath (b). Image (a) is adapted from

[720]. Image (b) is published with permission by Cameron Browne and N´estor Romeral Andr´es.

to the ghosts of Pac-Man (Namco, 1981) we could imagine that red represents an aggressive behavior whereas orange represents a passive behavior.

Among the few notable attempts of game generation, Game-O-Matic [723] is a game generation system that creates games representing ideas. More specifically, Game-O-Matic includes an authoring tool in which a user can enter entities and their interactions via concept maps. The entities and interactions are translated, re spectively, into game visuals and game mechanics; the mechanics, however, do not take into account the visuals or semantics of the game objects they are applied on.

One of the first preliminary discussions on how multi-facet integration might hap pen is offered by Nelson and Mateas [484]. In their paper they present a system for matching sprites (visuals) to very simple WarioWare-style mechanics. Their system is somewhat similar to Game-O-Matic, but works a bit differently: instead of the designer specifying verbs and nouns she wants a game to be about, she gives the system constraints on how verbs and nouns relate in the game (for example, a chas ing game needs a "prey" sprite that is something that can do things like "flee" or "be hunted" and so on). The system then uses ConceptNet7 and WordNet8 to generate games that fit these constraints.

Arguably one of the most elaborate examples of game generation is ANGELINA [135, 137, 136]. ANGELINA9 is a game generator that has seen several develop ments over the years and is currently capable of evolving the rules and levels of the game, collecting and placing pieces of visuals and music (that are relevant to the theme and the emotive mood of the game), giving names for the games it creates and even creating simple commentaries that characterize them. ANGELINA is able to generate games of different genres—including platformer games (see Fig. 4.21) and 3D adventure games—some of which have even participated in game design competitions [136].

The systems above make some initial, yet important, steps towards game gener ation and they attempt to interweave the different domains within games in a mean ingful way, mostly in a hierarchical fashion. However, PCG eventually needs to rise to the challenge of tackling the compound generation of multiple facets in an orchestrated manner [711, 371]. An early study on the fusion of more than one generative facet (domain) in games is the one performed recently by Karavolos et al. [324]. The study employs machine learning-based PCG to derive the common generative space—or the common patterns—of game levels and weapons in first person shooters. The aim of this orchestration process between level design and game design is the generation of level-weapon couplings that are balanced. The un known mapping between level representations and weapon parameters is learned by a deep convolutional neural network which predicts if a given level with a particular set of weapons will be balanced or not. Balance is measured in terms of the win-lose ratio obtained by AI bots playing in a deathmatch scenario. Figure 4.22 illustrates the architecture used to fuse the two domains. For the interested reader in the or-

7 http://conceptnet.io/

8 https://wordnet.princeton.edu/

9 http://www.gamesbyangelina.org/

Fig. 4.21 ANGELINA's Puzzling Present game. The game features an invert gravity mechanic that allows a player to overcome the high obstacle on the left and complete the level. Image obtained with permission from http://www.gamesbyangelina.org/.

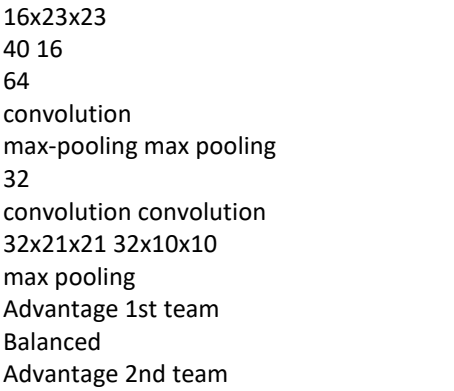


Fig. 4.22 A convolutional neural network (CNN) architecture used for fusing levels and weapons in first-person shooters. The network is trained to predict whether a combination of a level and a weapon would yield a balanced game or not. The CNN can be used to orchestrate the generation of a balanced level given a particular weapon and vice versa. Image adapted from [324].

chestration process we further elaborate on the topic in the last chapter of this book; some early discussions on this vision can also be found in [371].

4.6 Evaluating Content Generators

Creating a generator is one thing; evaluating it is another. Regardless of the method followed all generators shall be evaluated on their ability to achieve the desired goals of the designer. Arguably, the generation of any content is trivial; the generation of valuable content for the task at hand, on the other hand, is a rather challenging procedure. (One may claim, however, that the very process of generating valuable content is also, by itself, trivial as one can design a generator that returns a random sample of hand-crafted masterpieces.) Further, it is more challenging to generate content that is not only valuable but is also novel or even inspiring.

4.6.1 Why Is It Difficult?

But what makes the evaluation of content so difficult? First, it is the diverse, stochas tic and subjective nature of the users that experience the content. Whether players or designers, content users have dissimilar personalities, gameplay aims and behavior, emotive responses, intents, styles and goals [378]. When designing a PCG system it is critical to remember that we can potentially generate massive amounts of content for designers to interact with and players to experience. It is thus of utmost impor tance to be able to evaluate how successful the outcomes of the generator might be across dissimilar users: players and designers. While content generation is a cheap process relying on algorithms, design and game-play are expensive tasks relying on humans who cannot afford the experience of bad content. Second, content quality might be affected by algorithms and their underlying stochasticity, for instance, in evolutionary search. Content generators often exhibit non-deterministic behavior, making it very hard to predict a priori what the outcomes of a particular generative system might be.

4.6.2 Function vs. Aesthetics

Particular properties of content can be objectively defined and tested whereas other properties of it can only be assessed subjectively. It is only natural to expect that functional properties of content quality can be objectively defined whereas a large part of its aesthetics can only be defined subjectively. For instance, playability of a level is a functional characteristic that can be objectively measured—e.g., an AI agent manages to complete the level; hence it is playable. Balance and symmetry can also be objectively defined to a degree through estimates of deviation from a norm—it may be a score (balance) or a distance from a central choke point in the map (symmetry). There are games, however, for which content balance, symmetry and other functional properties are not trivially measurable. And of course there are several aspects of content such as the comprehensibility of a narrative, the pleas-

antnesses of a color scheme, the preference for a room's architectural style or the graphics style, and the experience of sound effects and music that are not objectively measured.

Functional, objectively defined, content properties can be expressed either as metrics or as constraints that a generator needs to satisfy. Constraints can be spec ified by the content designers or imposed by other game content already available.

For instance, let us assume that a well-designed generated strategy game level needs to be both balanced and playable. Playability can form a simple binary constraint: the level is playable when an AI agent manages to complete it; it is non-playable oth erwise. Balance can form another constraint by which all items, bases and resources are accessible to similar degrees by all players; if equal accessibility is below a threshold value then the constraint is not satisfied. Next, let us suppose we wish to generate a new puzzle for the map we just generated. Naturally, the puzzle needs to be compatible with our level. A PCG algorithm needs to be able to satisfy these constraints as part of its quality evaluation. Constrained satisfaction algorithms such as the feasible-infeasible two-population genetic algorithm [379, 382], constrained divergent search rewarding content value but also content novelty [382] or surprise [240], and constraint solvers such as answer set programming [638] are able to handle this. The generated results are within constraints, thereby valuable for the designer. Value, however, may have varying degrees of success and this is where alternative methods or heuristics can help, as we cover in the section below.

4.6.3 How Can We Evaluate a Generator?

Generally speaking, a content generator can be evaluated in three ways: directly by the designer or indirectly by either human players or AI agents. Designers can directly observe properties of the content generator and take decisions based on data visualization methods. Human players can play and test the content and/or provide feedback about the content via subjective reporting. AI agents can do the same: play the content or measure something about the content and report it to us in the form of a quality metric, or metrics. Clearly, machines cannot experience the content but they can, instead, simulate it and provide us estimates of content experience.

The overall evaluation process can very well combine and benefit from any of the above approaches. In the remainder of this section we cover the approaches of data visualization, AI automated playtesting and human playtesting in further detail.

4.6.3.1 Visualization

The visualization approach to content quality assessment is associated with a) the computation of meaningful metrics that can assign measurable characteristics to content and b) ways to visualize these metrics. The task of metric design can be viewed as the equivalent of fitness function design. As such, designing good con-

Fig. 4.23 The expressive range of the Ropossum level generator for the metrics of linearity and density. Adapted from [608].

tent generation quality metrics in an ad-hoc manner involves a degree of practical wisdom. Metrics, for instance, can be based on the expressive range of the gener ator under assessment, so-called expressivity metrics [640, 608]. The analysis of a generator's expressivity gives us access to the potential overall quality of the gen erator across its full range of generative space. The generative space can then be visualized as heatmaps or alternative graphical representations such as 2D or 3D scatter plots (see Fig. 4.23 for an example). It is one thing if a level generator is able to create only a few meaningful or playable levels and another if the genera tor is robust and consistent with respect to the playability of its generated levels.

It is also one thing if our generator is only able to generate levels with very spe cific characteristics within a narrow space of its expressive range and another if our level generator is able to express a broad spectrum of level properties, yielding uni formly covered expressive ranges. Such information can be directly visible on the illustrated heatmaps or scatter plots. Alternatively, data compression methods can be used directly on the generated content and offer us 2D or 3D representations of the generative space, thereby, bypassing the limitations of ad-hoc metric design. An

example of this approach is the use of autoencoders for compressing the images produced by the DeLeNoX autonomous content generator [373].

4.6.3.2 AI
Using AI to playtest our generator is a safe and relatively cheap way to rapidly retrieve quality metrics for it without relying on human playtesting. In the same way that a search-based PCG method would use AI to simulate the content before generating it, AI agents can test the potential of a generator across a number of metrics and return us values about its quality. The metrics can be in the form of classes—for instance, test checks performed for completing an area of a level—scalar values—e.g., a level's balance—or even ordinal values—e.g., the rank of the level in terms of asymmetry. The relevance of the metrics to the generator's quality is obviously dependent on the designer's ad-hoc decisions. Once again, designing appropriate metrics for our AI agent to compute is comparable to the challenge of designing any utility function. An interesting approach to AI-based testing is the use of procedural personas [267, 269]. These are data-driven inferred models of dissimilar play styles that potentially imitate the different styles of human play. In a sense, procedural personas provide a more human-realistic approach to AI-based testing of a generator. Finally, by visualizing particular game artifacts or simulating them through the use of AI agents we can have access to the information we might be able to extract from a game, we can understand what is possible within our content space, we can infer how rules and functions operate in whatever we generate, and we can possibly understand how the information we are able to extract relates to data we can extract from human playtesting [481].
4.6.3.3 Human Players
In addition to data visualization and AI-based simulation for the evaluation of a con tent generator a designer might wish to use complementary approaches that rely on quantitative user studies and playtesting. Playtesting is regarded to be an expensive way to test a generator but it can be of immense benefit for content quality assurance, in particular for those aspects of content that cannot be measured objectively—e.g., aesthetics and playing experience. The most obvious approach to evaluate the con tent experienced by players is to explicitly ask them about it. A game user study can involve a small number of dedicated players that will play through various amounts of content or, alternatively, a crowdsourcing approach can provide sufficient data to machine learn content evaluation functions (see [621, 370, 121] among others). Data obtained can be in any format including classes (e.g., a binary answer about the quality of a level), scores (e.g., the likeliness of a sound) or ranks (e.g., a preference about a particular level). It is important to note that the playtesting of content can be complemented by annotations coming from the designers of the game or other experts involved in content creation. In other words, our content generator may be labeled with both first-person (player) and third-person (designer) annotations. Fur ther guidelines about which questionnaire type to use and advice about the design of user study protocols can be found in Chapter 5.

4.7 Further Reading
Extensive versions of most of the material covered in this chapter can be found in dedicated chapters of the PCG in Games Textbook [616]. In particular, all the methods and types of PCG in games are covered in further detail (see Chapters 1 to 9 of [616]). Considering the roles of PCG, the mixed-initiative and the experience driven role of PCG are, respectively, detailed in Chapters 11 [374] and 10 [618] of that textbook [616]. In addition to Chapter 11 of [616] the framework named mixed-initiative co-creativity [774] provides a theoretical grounding for the impact of mixed-initiative interaction on the creativity of both designers and computational processes. Further, the original articles about the experience-driven PCG framework can be found in [783, 784]. Finally, the topic of PCG evaluation is covered also in the final chapter [615] of the PCG in Games textbook [616].
4.8 Exercises

PCG offers endless opportunities for generation and evaluation across the different creativity domains in games and across combinations of those. As an initial step we would recommend the reader to start experimenting with maze generation and platformer level generation (as outlined below). The website of the book contains details regarding both frameworks and potential exercises.

4.8.1 Maze Generation

Maze generation is a very popular type of level generation and relevant for several game genres. In the first exercise we recommend that you develop a maze genera tor using both a constructive and a search-based PCG approach and compare their performance according to a number of meaningful criteria that you will define. The reader may use the Unity 3D open-access maze generation framework which is available at: http://catlikecoding.com/unity/tutorials/maze/. Further guidelines and exercises for maze generation can be found at the book's website.

4.8.2 Platformer Level Generation

The platformer level generation framework is based on the Infinite Mario Bros(Pers son, 2008) framework which has been used as the main framework of the Mario AI (and later Platformer AI) Competition since 2010. The competition featured several different tracks including gameplay, learning, Turing test and level generation. For the exercises of this chapter the reader is requested to download the level generation

framework (https://sites.google.com/site/platformersai/) and apply constructive and generate-and-test methods for the generation of platformer levels. The levels need to be evaluated using one or more of the methods covered in this book. Further details and exercises with the platformer level generation framework can be found at the book's website.

4.9 Summary

This chapter viewed AI as a means for generating content in games. We defined procedural content generation as the algorithmic process of creating content in and for games and we explored the various benefits of this process. We then provided a general taxonomy about content and its generation and explored the various ways one can generate content including search-based, solver-based, grammar-based, ma chine learning-based, and constructive generation methods. The use of the PCG method is naturally dependent on the task at hand and the type of content one wishes to generate. It further depends on the potential role the generator might take within games. We outlined the four possible roles a generator can take in games which are determined by the degree to which they involve the designer (autonomous vs. mixed-initiative) and/or the player (experience-agnostic vs. experience-driven) in the process. The chapter ends with a discussion on the important and rather un explored topic of evaluation, the challenges it brings, and a number of evaluation approaches one might consider.

We have so far covered the most traditional use of AI in games (Chapter 3) and the use of AI for generating parts of (or complete) games (this chapter). The next and final chapter of this part is dedicated to the player and the ways we can use AI to model aspects of her behavior and her experience.

Chapter 5
Modeling Players

This chapter is dedicated to players and the use of AI for modeling them. This area of research is often called player modeling [782, 636]. We take player modeling to mean the detection, prediction and expression of human player characteristics that are manifested through cognitive, affective and behavioral patterns while playing games. In the context of this book, player modeling studies primarily the use of AI methods for the construction of computational models of players. By model we re fer to a mathematical representation—it may be a rule set, a vector of parameters, or a set of probabilities—that captures the underlying function between the charac teristics of the player and her interaction with the game, and the player's response to that interaction. Given that every game features at least one player (with some no table exceptions [50]), and that player modeling affects work on game-playing and

content generation, we consider the modeling of player behavior and experience as a very important use of AI in games [764, 785].

Psychology has studied human behavior, cognition and emotions for a long time. Branches of computer science and human-computer interaction that attempt to model and simulate human behavior, cognition, emotion or the feeling of emo tion (affect) include the fields of affective computing and user modeling. Player modeling is related to these fields but focuses on the domain of games. Notably, games can yield dynamic and complex emotions in the player, the manifestations of which cannot be captured trivially by standard methods in empirical psychol ogy, affective computing or cognitive modeling research. The high potential that games have in affecting players is mainly due to their ability to place the player in a continuous mode of interaction, which, in turn, elicits complex cognitive, affective and behavioral responses. Thus, the study of the player may not only contribute to the design of improved forms of human-computer interaction, but also advance our knowledge of human experiences.

As mentioned earlier, every game features at least one user—the player—who controls some aspect of the game environment. The player character could be visible in the game as an avatar or a group of entities [94], or could be invisible as in many puzzle games and casual games. Control may vary from the relatively simple (e.g., limited to movement in an orthogonal grid) to the highly complex (e.g., having

to decide several times per second between hundreds of different possibilities in a highly complex 3D world). Given these intricacies, understanding and modeling the interaction between the player and the game can be seen as a holy grail of game design and development. Designing the interaction and the emergent experience right results in a successful game that manages to elicit unique experiences.

The interaction between the player(s) and the game is dynamic, real-time and in many cases highly complex. The interaction is also rich in the sense that many modalities of interaction may be involved and that the information exchange be tween the game and the player may both be fast and entail large amounts of data for us to process. If the game is well-designed, the interaction is also highly engaging for the player. Given the great amount of information that can be extracted through this interaction and used for creating models of the player, the game should be able to learn much about the person playing it, as a player and perhaps as a human in general. In fact, there is no reason why the model should not know more about how you play than you do.

In the remainder of this chapter we first attempt to define the core ingredients of player modeling (Section 5.1) and then we discuss reasons why AI should be used to model players (Section 5.2). In Section 5.3 we provide a high-level taxonomy of player modeling focusing on two core approaches for constructing a player model: top-down and bottom-up. We then detail the available types of data for the model's input (Section 5.4), a classification for the model's output (Section 5.5) and the various AI methods that are appropriate for the player modeling task (Section 5.6). The key components of player modeling as discussed in this chapter (input, output and model) are depicted in Fig. 5.1. We conclude, in Section 5.7, with a number of concrete examples of AI being used for modeling players.

5.1 What Player Modeling Is and What It Is Not

One could arguably detect behavioral, emotional or cognitive aspects of both hu man players and non-human players, or non-player characters (notwithstanding the actual existence of emotions in the latter). However, in this book we focus on as pects that can be detected from, modeled from, and expressed in games with human players [782]. We explicitly exclude the modeling of NPCs from our discussion in this chapter, as in our definition, player modeling is modeling of a human player. Modeling the experience of an NPC would seem to be a futile exercise, as one can hardly say that an NPC possesses actual emotions or cognition. Modeling the be havior of an NPC is also of little interest, at least if one has access to the game's

code: a perfect model for the NPC already exists. NPC modeling, however, can be a useful testbed for player modeling techniques, for instance, by comparing the model derived from human players with the hand-crafted one. More interestingly, it can be an integral component of AI that adapts its behavior in response to the dynamics of the NPCs—as in [28]. Nevertheless, while the challenges faced in modeling NPCs

Fig. 5.1 The key components of player modeling as discussed in this chapter. The distinction be tween model-based and model-free approaches is outlined in Section 5.3. The various options for the input of the model are discussed in Section 5.4. The taxonomy for the model's output is dis cussed in Section 5.5—each box represents a dedicated subsection. Finally, the various AI methods (supervised learning, reinforcement learning and unsupervised learning) used for modeling corre sponding output data types are discussed thoroughly in Section 5.6.
are substantial, the issues raised from the modeling of human players define a far more complex and important problem for the understanding of player experience.
Sometimes the terms player modeling and opponent modeling [214, 592, 48] are used interchangeably when a human player is modeled. However, opponent mod eling is a more narrow concept referring to predicting behavior of an adversarial player when playing to win in an imperfect information game like Poker [48] or StarCraft (Blizzard Entertainment, 1988) [504]. Some aspects of modeling NPCs or simulated playthroughs for winning in a game are discussed in Chapter 3.
We also make a distinction between player modeling [116, 281] and player pro-filing [782]. The former refers to modeling complex dynamic phenomena during gameplay interaction, whereas the latter refers to the categorization of players based on static information that does not alter during gameplay. Information of static na ture includes personality, cultural background, gender and age. We put an emphasis on the former, but will not ignore the latter, as the availability of a good player profile may contribute to the construction of reliable player models.

In summary, player modeling—as we define it in this book—is the study of com putational means for the modeling of a player's experience or behavior which is based on theoretical frameworks about player experience and/or data derived from the interaction of the player with a game [782, 764]. Player models are built on dy namic information obtained during game-player interaction, but they could also rely on static player profiling information. Unlike studies focusing on taxonomies of be havioral player modeling—e.g., via a number of dimensions [636] or direct/indirect measurements [623]—we view player modeling in a holistic manner including cog nitive, affective, personality and demographic aspects of the player. Moreover, we exclude approaches that are not directly based on human-generated data or not based on empirically-evaluated theories of player experience, human cognition, affect or behavior. The chapter does not intend to provide an exhaustive review of player modeling studies under the above definition, but rather an introduction and a high level taxonomy that explores the possibilities with respect to the modeling approach, the model's input and the model's output.

5.2 Why Model Players?

The primary goal of player modeling is to understand how the interaction with a game is experienced by individual players. Thus, while games can be utilized as an arena for eliciting, evaluating, expressing and even synthesizing experience, we argue that the main aim of the study of players in games is the understanding of players' cognitive, affective and behavioral patterns. Indeed, by the very nature of games, one cannot dissociate games from player experience.
There are two core reasons that drive the use of AI for modeling game play ers and their play, thereby serving the primary goal of player modeling as stated above. The first is for understanding something about their players' experience dur ing play. Models of player experience are often built using machine learning meth ods, typically supervised learning methods like support vector machines or neural networks. The training data here consists of some aspect of the game or player-game

interaction, and the targets are labels derived from some assessment of player ex perience, gathered for example from physiological measurements or questionnaires [781]. Once predictors of player experience are derived they can be taken into ac count for designing the in-game experience. That can be achieved by adjusting the behavior of non-player characters (see Chapter 3) or by adjusting the game environ ment (see Chapter 4).

The second reason why one would want to use AI to model players is for under standing players' behavior in the game. This area of player modeling is concerned with structuring observed player behavior even when no measures of experience are available—for instance, by identifying player types or predicting player behavior via game and player analytics [178, 186]. A popular distinction in data derived from games [186] is the one between player metrics and game metrics. The latter is a superset of the former as it also includes metrics about the game software (system

metrics) and the game development process as a whole (process metrics). System metrics and process metrics are important aspects of modern game development that influence decision making with respect to procedures, business models, and market ing. In this book, however, we focus on player metrics. The interested reader may refer to [186] for alternative uses of metrics in games and the application of analytics to game development and research—i.e., game analytics.

Once aspects of player behavior are identified a number of actions can be taken to improve the game such as the personalization of content, the adjustment of NPCs or, ultimately, the redesign of (parts of) the game. Derived knowledge about the in-game behavior of the player can lead to improved game testing and game design procedures, and better monetization and marketing strategies [186]. Within behavior modeling we identify four main player modeling subtasks that are particularly rel evant for game AI: imitation and prediction—achieved via supervised learning or reinforcement learning—and clustering and association mining—achieved via un supervised learning.

The two main purposes of player imitation is the development of non-player characters with believable, human-like behavioral characteristics, and the understanding of human play per se through creating generative models of it.

The prediction of aspects of player behavior, instead, may provide answers to ques tions such as "when will this player stop playing?" or "how often will that player get stuck in that area of the level?" or "which item type will this player pick in the next room?". The aim of clustering is the classification of player behaviors within a number of clusters depending of their behavioral attributes. Clustering is important for both the personalization of the game and the understanding of playing behavior in association with the game design [178]. Finally, association mining is useful in instances where frequent patterns or sequences of actions (or in-game events) are important for determining how a player behaves in a game.

While player behavior and player experience are interwoven notions there is a subtle difference between them. Player behavior points to what a player does in a game whereas player experience refers to how a player feels during play. The feeling of one's gameplay experience is clearly associated with what one does in the game; player experience, however, is primarily concerned with affective and cognitive aspects of play as opposed to mere reactions of gameplay which refer to player behavior.

Given the above aims, core tasks and sub-tasks of player modeling in the next section we discuss the various available options for constructing a player model.

5.3 A High-Level Taxonomy of Approaches

Irrespective of the application domain, computational models are characterized by three core components: the input the model will consider, the computational model per se, and the output of the model (see Fig. 5.1). The model itself is a mapping between the input and the output. The mapping is either hand-crafted or derived from data, or a mix of the two. In this section we will first go through the most

common approaches for constructing a computational model of players, then we

will go through a taxonomy of possible inputs for a player model (Section 5.4) and finally we will examine aspects of player experience and behavior that a player model can represent as its output (Section 5.5).

A high-level classification of the available approaches for player modeling can be made between model-based (or top-down) and model-free (or bottom-up) ap proaches [782, 783]. The above definitions are inspired by the analogous classification in RL by which a world model is available (i.e., model-based) or not (i.e., model-free). Given the two ends of this continuum hybrid approaches between them can naturally exist. The gradient red color of the player model box in Fig. 5.1 illus trates the continuum between top-down and bottom-up approaches. The remainder of this section presents the key elements of and core differences among the various approaches for modeling of players.

5.3.1 Model-Based (Top-Down) Approaches

In a model-based or top-down [782] approach a player model is built on a theoret ical framework. As such, researchers follow the modus operandi of the humanities and social sciences, which hypothesize models to explain phenomena. Such hy potheses are usually followed by an empirical phase in which it is experimentally determined to what extent the hypothesized models fit observations; however, such a practice is not the norm within player experience research. While user experience has been studied extensively across several disciplines, in this book we identify three main disciplines we can borrow theoretical frameworks from and build models of player experience: psychology and affective sciences, neuroscience, and finally, game studies and game research.

5.3.1.1 Psychology and Affective Sciences

Top-down approaches to player modeling may refer to models derived from popu lar theories about emotion [364] such as the cognitive appraisal theory [212, 601].

Further, the player model may rely on well established affect representations such as the emotional dimensions of arousal and valence [200] that define the circumplex model of affect of Russell [539] (see Fig. 5.2(a)). Valence refers to how pleasur able (positive) or unpleasurable (negative) the emotion is whereas arousal refers to how intense (active) or lethargic (inactive) that emotion is. Following a theoretical model, emotional manifestations of players are often mapped directly to specific player states. For instance, by viewing player experience as a psychophysiological phenomenon [779] a player's increased heart rate may correspond to high arousal and, in turn, to high levels of excitement or frustration.

Beyond established theories of emotion, model-based approaches can also be in spired by a general cognitive-behavioral theoretical framework such as the theory

of mind [540] for modeling aspects of social interactions in games. Popular ex ample frameworks for deriving user models in games include the usability theory [489, 290], the belief-desire-intention (BDI) model [66, 224], the cognitive model by Ortony, Clore, and Collins [512] and Skinner's behavioristic approach [633] with its links to reward systems in games. Further we can draw inspiration from social sciences and linguistics in order to model lexical aspects of gameplay interaction (e.g., chatting). Natural language processing, opinion mining and sentiment analysis are normally relying on theoretical models that build on affective and sociological aspects of textual communication [517, 514].

Of particular importance is the concept of flow by Csikszentmihalyi [151, 149, 150] which has been a popular psychological construct for modeling player experi ence in a top-down fashion. When in a state of flow (or else, state of "happiness") during an activity we tend to concentrate on the present moment, we lose our ability of reflective self-consciousness, we feel a sense of personal control over the sit uation or activity, our perception of time is altered, and we experience the activity as intrinsically rewarding. Analogously the optimal experience during play has been associated with a fine balance between boredom and anxiety, also known as the flow channel (see Fig. 5.2(b)). Given its direct relevance to player experience, flow has been adapted and incorporated for use in game design and for the understanding of

player experience [678, 675, 473].

5.3.1.2 Neuroscience

A number of studies have relied on the working hypothesis of an underlying map ping between the brain, its neural activity and player experience. However, this re lationship is not well explored and the presumptive mapping is largely unknown.

For example, interest has been associated with activity in the visual cortex and the release of endomorphin whereas the sense of achievement has been linked to dopamine levels [35]. According to [35], neuroscientific evidence suggests that the reward systems of games are directly associated with the dopamine-based reward structures in the brain and that dopamine is released during gameplay [346]. Fur ther, pleasure has been associated with areas in the brain responsible for decision making, thereby revealing the direct links between gameplay experience and deci sion making [575]. Pleasure has also been associated with uncertain outcomes or uncertain rewards [625] as well as with interest and curiosity [43], which are all key elements of successful game design. Stress is also tightly coupled with player expe rience given its clear association with anxiety and fear; stress can be both monitored via physiology and regulated via game design. The testosterone levels of players have also been measured in association to digital game activities [444] and findings reveal particular patters of competition in games as testosterone factors. Finally, it appears that trust between players in a social gaming setup could be measured indi rectly via oxytocin levels [350].

The degree to which current findings from neuroscience are applicable to player experience research is largely unknown since access to neural activity and brain hor-

(a) Russell's two-dimensional circumplex model of affect. The figure contains a small number of representative affective states (black circles).

(b) An illustration of the flow channel.

Fig. 5.2 Two popular frameworks used for modeling users and their experience in games: (a) the arousal-valence circumplex model of affect and (b) the flow channel concept.

mone levels remains a rather intrusive process at the time of writing. Manifestations of brain activity such as the brain's electrical waves—measured through electroen cephalography on our scalp—or more indirect manifestations such as stress and anxiety—measured through skin conductance—can give us access to approximates of brain activity. These approximates can be used for modeling the experience of play as discussed later in this chapter.

5.3.1.3 Game Studies and Game Research

Theoretical models of user experience in games are often driven by work in game studies and game research. Examples of models that have been used extensively in the literature include Malone's core design dimensions that collectively contribute to 'fun' games [419] defined as challenge, curiosity and fantasy. In particular, challenge refers to the uncertainty of achieving a goal due to e.g., variable difficulty level, multiple level goals, hidden information, and randomness. Curiosity refers to the player's feeling of uncertainty with respect to what will happen next. Finally, fantasy is the ability of the game to show (or evoke) situations or contexts that are not actually present. These three dimensions have been quantified, operationalized and successfully evaluated in prey-predator games [766], physical games [769, 775], preschooler games [320] and racing games [703].

Bartle's [33] classification of player types within games as a form of general player profiles can be used indirectly for modeling players. Bartle identifies four archetypes of players he names killers (i.e., players that focus on winning and are engaged by ranks and leaderboards), achievers (i.e., players that focus on achieving goals quickly and are engaged by achievements), socializers (i.e., players that focus on social aspects of games such as developing a network of friends) and explorers (i.e., players who focus on the exploration of the unknown). Various other method ologies have also been followed to derive specific player experience archetypes for particular classes of games [34, 787].

Other popular and interconnected views of player experience from a game design perspective include the theory of 'fun' by Koster [351], the notion of the 'magic circle' in games [587] and the four "fun" factor model of Lazzaro [365]. Indicatively, Koster's theory relates the concept of fun with learning in games: the more you learn the more you tend to play a game. According to his theory you stop playing a game that is way too easy (no learning of new skills) or way too hard (no learning either). Lazzaro's four fun factors are named hard fun (e.g., playing to win and see how good I am at it), easy fun (e.g., playing to explore new worlds and game spaces), serious fun (e.g., playing to feel better about myself or get better at something that matters to me) and people fun (e.g., playing as an excuse to invite friends over, or having fun merely by watching them play). Within game studies, the theoretical model of incorporation [94] is a notable multifaceted approach for capturing player immersion. The model is composed of six types of player involvement: affective, kinaesthetic, spatial, shared, ludic, and narrative.

With a careful analysis of the models proposed and their subcomponents one could coherently argue that there is one underlying theoretical model of player ex perience after all. While it is not the intention of this book to thoroughly discuss the interconnections between the aforementioned models it is worth pointing out a num ber of indicative examples of our envisaged overarching player experience model.

An explorer (Bartle), for instance, can be associated with the easy fun factor of Laz zaro and the curiosity dimension of Malone. Further, the achiever archetype (Bartle) can be linked to the serious fun factor (Lazzaro). Accordingly, a killer archetype (Bartle) maps to the hard fun factor (Lazzaro), the challenge dimension of Malone's model, and a number of flow aspects. Finally, a socializer player profile (Bartle) could be associated to people fun (Lazzaro) and, in turn, to the shared involvement facet of Calleja [94].

Even though the literature on theoretical models of experience is rather rich, one needs to be cautious with the application of such theories to games (and game play ers) as the majority of the models have not been derived from or tested on interactive media such as games. Calleja [94], for instance, reflects on the inappropriateness of the concepts of 'fun' and 'magic circle' (among others) for games. At this point it is worth noting that while ad-hoc designed models can be an extremely powerful and expressive they need to be cross-validated empirically to be of practical use for computational player modeling; however, such practices are not as common within the broader area of game studies and game design.

5.3.2 Model-Free (Bottom-Up) Approaches

Model-free approaches refer to the data-driven construction of an unknown map ping (model) between a player input and a player state. Any manifestation of player affect or behavioral pattern could define the input of the model (see more in Section 5.4 below). A player state, on the other hand, is any representation of the player's experience or current emotional, cognitive, or behavioral state; this is es sentially the output of the computational model (see more in Section 5.5). Evidently, model-free approaches follow the modus operandi of the exact sciences, in which observations are collected and analyzed to generate models without a strong initial assumption on what the model looks like or even what it captures. Player data and labels of player states are collected and used to derive the model.

Classification, regression and preference learning techniques adopted from ma chine learning—see Chapter 2—or statistical approaches are commonly used for the construction of the mapping between the input and the output. Examples include studies in which player actions, goals and intentions are modeled and predicted for the purpose of believable gameplay, adaptive gameplay, interactive storytelling or even the improvement of a game's monetization strategy [511, 800, 414, 693, 592].

In contrast to supervised learning, reinforcement learning can be applied when a re ward function, instead, can characterize aspects of playing behavior or experience.

Unsupervised learning is applicable when target outputs are not available for pre-

dictive purposes but, alternatively, data is used for the analysis of playing behavior (see Fig. 5.1).

We meet bottom-up player modeling attempts since the early years of the game AI field in first-person shooters [695, 696], racing games [703] and variants of Pac Man (Namco, 1980) [776]. Recently, the availability of large sets of game and player data has opened up the horizons of behavioral data mining in games—i.e., game data mining [178]. Studies that attempt to identify different behavioral, playing and action patterns within a game are well summarized in [186] and include [36, 176, 687, 690, 750], among many others.

5.3.3 Hybrids

The space between a completely model-based and a completely model-free ap proach can be viewed as a continuum along which any player modeling approach might be placed. While a completely model-based approach relies solely on a the oretical framework that maps a player's responses to game stimuli, a completely model-free approach assumes there is an unknown function between modalities of user input and player states that a machine learner (or a statistical model) may dis cover, but does not assume anything about the structure of this function. Relative to these extremes, the vast majority of studies in player modeling may be viewed as hybrids that synergistically combine elements of the two approaches. The con tinuum between top-down and bottom-up player modeling approaches is illustrated with a gradient color in Fig. 5.1.

5.4 What Is the Model's Input Like?

By now we have covered the various approaches available for modeling players and we will, in this section, focus on what the input of such a model might be like. The model's input can be of three main types: (1) anything that a player is doing in a game environment gathered from gameplay data—i.e., behavioral data of any type such as user interface selections, preferences, or in-game actions; (2) objective data collected as responses to game stimuli such as physiology, speech and body movements; and (3) the game context which comprises of any player agent interactions but also any type of game content viewed, played, and/or created.

The three input types are detailed in the remainder of this section. At the end of the section we also discuss static profile information on the player (such as personality) as well as web data beyond games that could feed and enhance the capacity of a player model.

5.4.1 Gameplay

Given that games may affect the player's cognitive processing patterns and cogni tive focus we assume that a player's actions and preferences are linked directly to her experience. Consequently, one may infer the player's current experience by an alyzing patterns of her interaction with the game, and by associating her experience with game context variables [132, 239]. Any element derived from the direct inter action between the player and the game can be classified as gameplay input. These interpretable measures of gameplay have also been defined as player metrics [186].

Player metrics include detailed attributes of the player's behavior derived from re sponses to game elements such as NPCs, game levels, user menus, or embodied conversational agents. Popular examples of data attributes include detailed spatial locations of players viewed as heatmaps [177], statistics on the use of in-game menus, as well as descriptive statistics about gameplay, and communication with other players. Figure 5.3 shows examples of heatmaps in the MiniDungeons1 puzzle game. Both general measures (such as performance and time spent on a task) and game-specific measures (such as the weapons selected in a shooter game [250]) are relevant and appropriate player metrics.

A major limitation with the gameplay input is that the actual player experience is only indirectly observed. For instance, a player who has little interaction with a game might be thoughtful and captivated, or just bored and busy doing something else. Gameplay metrics can only be used to approach the likelihood of the presence of certain player experiences. Such statistics may hold for player populations, but

may provide little information for individual players. Therefore, when one attempts to use pure player metrics to make estimates of player experiences and make the game respond in an appropriate manner to these perceived experiences, it is advis able to keep track of the feedback of the player to the game responses, and adapt when the feedback indicates that the player experience was gauged incorrectly.

5.4.2 Objective

Computer game players are presented with a wide palette of affective stimuli dur ing game play. Those stimuli vary from simple auditory and visual events (such as sound effects and textures) to complex narrative structures, virtual cinemato graphic views of the game world and emotively expressive game agents. Player emotional responses may, in turn, cause changes in the player's physiology, reflect on the player's facial expression, posture and speech, and alter the player's attention and focus level. Monitoring such bodily alterations may assist in recognizing and constructing the player's model. As such, the objective approach to player model ing incorporates access to multiple modalities of player input.

1 http://minidungeons.com/

(a) In this example the player acts as a com pletionist: succeeding in killing all monsters, drinking all potions and collecting all treasure.
(b) In this example the player prioritizes reach ing the exit, avoiding any monsters and only collecting potions and treasures that are near the path to the exit.

Fig. 5.3 Two example heatmaps (human playtraces) in the MiniDungeons game. MiniDungeons is a simple turn-based rogue-like puzzle game, implemented as a benchmark problem for modeling the decision-making styles of human players [267].

The relationship between psychology and its physiological manifestations has been studied extensively ([17, 95, 779, 558] among many others). What is widely evidenced is that the sympathetic and the parasympathetic components of the au tonomic nervous system are involuntary affected by affective stimuli. In general, arousal-intense events cause dynamic changes in both nervous systems: an increase and a decrease of activity, respectively, in the sympathetic and the parasympathetic nervous system. Alternatively, activity at the parasympathetic nervous system is high during relaxing or resting states. As mentioned above, such nervous system activities cause alterations in one's facial expression, head pose, electrodermal ac tivity, heart rate variability, blood pressure, pupil dilation [91, 624] and so on.

Recent years have seen a significant volume of studies that explore the interplay between physiology and gameplay by investigating the impact of different game play stimuli on dissimilar physiological signals ([697, 473, 421, 420, 556, 721, 175, 451] among others). Such signals are usually obtained through electrocardiogra phy (ECG) [780], photoplethysmography [780, 721], galvanic skin response (GSR) [421, 271, 270, 272], respiration [721], electroencephalography (EEG) [493] and electromyography (EMG).

In addition to physiology one may track the player's bodily expressions (mo tion tracking) at different levels of detail and infer the real-time affective responses from the gameplay stimuli. The core assumption of such input modalities is that particular bodily expressions are linked to expressed emotions and cognitive pro cesses. Objective input modalities, beyond physiology, that have been explored ex-

tensively include facial expressions [321, 19, 236, 88, 794], muscle activation (typ ically face) [133, 164], body movement and posture [23, 731, 321, 172, 47], speech [741, 319, 308, 306, 30], text [517, 137, 391], haptics [509], gestures [283], brain waves [559, 13], and eye movement [23, 469].

While objective measurements can be a very informative way of assessing the player's state during the game a major limitation with most of them is that they can be invasive, thus affecting the player's experience with the game. In fact, some types of objective measures appear to be implausible within commercial-standard game development. Pupillometry and gaze tracking, for instance, are very sensitive to

distance from screen, and variations in light and screen luminance, which collec tively make them rather impractical for use in a game application. The recent rebirth of virtual reality (VR), however, gives eye gaze sensing technologies entirely new opportunities and use within games [628]; a notable example of a VR headset that features eye-tracking is FOVE.2 Other visual cues obtained through a camera (facial expressions, body posture and eye movement) require a well-lit environment which is often not present in home settings (e.g., when playing video-games) and they can be seen by some players as privacy hazards (as the user is continuously recorded). Even though highly unobtrusive, the majority of the vision-based affect-detection systems currently available have additional limitations when asked to operate in real-time [794]. We argue that an exception to this rule is body posture, which can both be effectively detected nowadays and provide us with meaningful estimates of player experience [343]. Aside from the potential they might have, however, the appropriateness of camera-based input modalities for games is questionable since experienced players tend to stay still while playing games [22].

As a response to the limitations of camera-based measurements, speech and text (e.g., chat) offer two highly accessible, real-time efficient and unobtrusive modali ties with great potential for gaming applications; however, they are only applicable to games where speech (or text) forms a control modality (as e.g., in conversational games for children [320, 789]), collaborative games that naturally rely on speech or text for communication across players (e.g., in collaborative first-person shoot ers), or games that rely on natural language processing such as text-based adventure games or interactive fiction (see discussion of Chapter 4).

Within players' physiology, existing hardware for EEG, respiration and EMG require the placement of body parts such as the head, the chest or parts of the face on the sensors, making those physiological signals rather impractical and highly in trusive for most games. On the contrary, recent sensor technology advancements for the measurement of electrodermal activity (skin conductivity), photoplethysmogra phy (blood volume pulse), heart rate variability and skin temperature have made those physiological signals more attractive for the study of affect in games. Real time recordings of these can nowadays be obtained via comfortable wristbands and stored in a personal computer or a mobile device via a wireless connection [779].

At the moment of writing there are a few examples of commercial games that utilize physiological input from players. One particularly interesting example is

2 https://www.getfove.com/

Fig. 5.4 A screenshot from Nevermind (Flying Mollusk, 2015). The game supports several off the-shelf sensors that allow the audiovisual content of the game to adapt to the stress levels of the player. Image obtained from Erin Reynolds with permission.

Nevermind (Flying Mollusk, 2015), a biofeedback-enhanced adventure horror game that adapts to the player's stress levels by increasing the level of challenge it pro vides: the higher the stress the more the challenge for the player (see Fig. 5.4). A number of sensors which detect heart activity are available for affective interac tion with Nevermind. The Journey of Wild Divine (Wild Divine, 2001) is another biofeedback-based game designed to teach relaxation exercises via the player's blood volume pulse and skin conductance. It is also worth noting that AAA game developers such as Valve have experimented with the player's physiological input for the personalization of games such as Left 4 Dead (Valve, 2008) [14].

5.4.3 Game Context

In addition to gameplay and objective input, the game's context is a necessary input for player modeling. Game context refers to the momentanous state of the game during play and excludes any gameplay elements; those are already discussed in the gameplay input section. Clearly, our gameplay affects some aspects of the game context and vice versa but the two can be viewed as separate entities. Viewing this relationship from an analytics lens, the game context can be seen as a form of game metrics, opposed to gameplay which is a form of player metrics.

The importance of the game context for modeling players is obvious. In fact, we

could argue that the context of the game during the interaction is a necessary input for detecting reliably any cognitive and affective responses of players. It could also

be argued that the game context is necessary as a guide during the annotation of the player experience; but more of that we will discuss in Section 5.5. The same way that we require the current social and cultural context to better detect the under lying emotional state of a particular facial expression of our discussant any player reactions cannot be dissociated from the stimulus (or the game context) that elicited them. Naturally, player states are always linked to game context. As a result, player models that do not take context into account run a risk of inferring erroneous states for the player. For example, an increase in galvanic skin response can be linked to different high-arousal affective states such as frustration and excitement. It is very hard to tell however, what the heightened galvanic skin response "means" without knowing what is happening in the game at the moment. In another example, a partic ular facial expression of the player, recorded though a camera, could be associated with either an achievement in the game or a challenging moment, and needs to be triangulated with the current game state to be understood. Evidently, such dualities of the underlying player state may be detrimental for the design of the player model.
While a few studies have investigated physiological reactions of players in isola tion, good practice in player modeling commands that any reactions of the play ers is triangulated with information about the current game state. For instance, the model needs to know if the GSR increases because the player died or com pleted the level. The game context—naturally combined (or fused) with other in put modalities from the player—has been used extensively in the literature for the prediction of different affective and cognitive states relevant to playing experience [451, 434, 521, 617, 572, 133, 254, 558, 452, 433].

5.4.4 Player Profile

A player profile includes all the information about the player which is static and it is not directly (nor necessarily) linked to gameplay. This may include information on player personality (such as expressed by the Five Factor Model of personality [140, 449]), culture dependent factors, and general demographics such as gender and age. A player's profile may be used as input to the player model to comple ment the captured in-game behavior with general attributes about the player. Such information may lead to more precise predictive models about players.
While gender, age [787, 686], nationality [46] and player expertise level [96] have already proven important factors for profiling players the role of personality remains somewhat contentious. On the one hand, the findings of van Lankveld et al. [736, 737], for instance, reveal that gameplay behavior does not necessarily corre spond to a player's behavior beyond the game. On the other hand, Yee et al. have identified strong correlations between player choices in World of Warcraft (Blizzard Entertainment, 2004) and the personalities of its players [788]. Strong correlations have also been found between the playing style and personality in the first-person shooter Battlefield 3 (Electronic Arts, 2011) [687]. In general, we need to acknowl edge that there is no guaranteed one-to-one mapping between a player's in-game

behavior and personality, and that a player's personality profile does not necessarily indicate what the player would prefer or like in a game [782].

5.4.5 Linked Data

Somewhere between the highly dynamic in-game behavior and the static profile in formation about the player we may also consider linked data retrieved from web services that are not associated with gameplay per se. This data, for instance, may include our social media posts, emoticons, emojis [199], tags used, places visited, game reviews written, or any relevant semantic information extracted from diverse Web content. The benefit of adding such information to player models is many fold but it has so far seen limited use in games [32]. In contrast to current player modeling approaches the use of massive amounts and dissimilar types of content across linked online sources would enable the design of player models which are

based on user information stored across various online datasets, thereby realizing semantically-enriched game experiences. For example, both scores and sentiment analyzed textual reviews [517, 514] from game review sites such as Metacritic3 or GameRankings4 can be used as input to a model. This model can then be used to create game content which is expected to appeal to the specific parts of the com munity, based, for instance, on demographics, skill or interests collected from the user's in-game achievements or favored games [585].

5.5 What Is the Model's Output Like?

The model's output, i.e., that which we wish to model, is usually a representation of the player's state. In this section we explore three options for the output of the model that serve different purposes in player modeling. If we wish to model the experi ence of the player the output is provided predominately through manual annotation. If instead we wish to model aspects of player behavior the output is predominately based on in-game actions (see Fig. 5.1). Finally, it may very well be that the model has no output. Section 5.5.1 and Section 5.5.2 discuss the particularities of the out put, respectively, for the purpose of behavioral modeling and experience modeling whereas Section 5.5.3 explores the condition where the model has no outputs.

3 http://www.metacritic.com

4 http://www.gamerankings.com/

5.5.1 Modeling Behavior

The task of modeling player behavior refers to the prediction or imitation of a partic ular behavioral state or a set of states. Note that if no target outputs are available then we are faced with either an unsupervised learning problem or a reinforcement learn ing problem which we discuss in Section 5.5.3. The output we must learn to predict (or imitate) in a supervised learning manner can be of two major types of game play data: either micro-actions or macro-actions (see Fig. 5.1). The first machine learning problem considers the moment-to-moment game state and player action space that are available at a frequency of frame rates. For example, we can learn to imitate the moves of a player on a frame-to-frame basis by comparing the play traces of an AI agent and a human as e.g., done for Super Mario Bros (Nintendo, 1985) [511, 469]. When macro-actions are considered instead, the target output is normally an aggregated feature of player behavior over time, or a behavioral pat tern. Examples of such outputs include game completion times, win rates, churn, trajectories, and game balance.

5.5.2 Modeling Experience

To model the experience of the player one needs to have access to labels of that experience. Those labels ideally need to be as close to the ground truth of expe rience as possible. The ground truth (or gold standard) in affective sciences refers to a hypothesized and unknown label, value, or function, that best characterizes and represents an affective construct or an experience. Labels are normally provided through manual annotation which is a rather laborious process. Manual annotation is however necessary given that we require some estimate of the ground truth for subjective notions such as the emotional states of the player. The accuracy of that estimation is regularly questioned as there are numerous factors contributing to a deviation between a label and the actual underlying player experience.

Manually annotating players and their gameplay is a challenge in its own right with respect to both the human annotators involved and the annotation protocol cho sen [455, 777]. On one hand, the annotators need to be skilled enough to be able to approximate the actual experience well. On the other hand, there are still many open questions left for us to address when it comes to the annotation tools and protocols used. Such questions include: Who will do the labeling: the person experiencing the gameplay or others? Will the labeling of player experience involve states (dis crete representation) or instead involve the use of intensity or experience dimensions (continuous representation)? When it comes to time, should it be done in real-time or offline, in discrete time periods or continuously? Should the annotators be asked

to rate the affect in an absolute fashion or, instead, rank it in a relative fashion?
Answers to the above questions yield different data annotation protocols and, in evitably, varying degrees of data quality, validity and reliability. In the following

sections we attempt to address a number of such critical questions that are usually raised in subjective annotations of player states.

5.5.2.1 Free Response or Forced Response?

Subjective player state annotations can be based either on a player'sfree response—retrieved via e.g., a think-aloud protocol [555]—or on forced responses retrieved through questionnaires or annotation tools. Free response naturally contains richer information about the player's state, but it is often unstructured, even chaotic, and thus hard to analyze appropriately. On the other hand, forcing players to self-report their experiences using directed questions or tasks constrains them to specific ques tionnaire items which could vary from simple tick boxes to multiple choice items.

Both the questions and the answers we provide to annotators may vary from single words to sentences. Questionnaires can contain elements of player experience (e.g., the Game Experience Questionnaire [286]), demographic data and/or personality traits (e.g., a validated psychological profiling questionnaire such as the NEO-PI-R [140]). In the remainder of this section we will focus on forced responses as these are easier to analyze and are far more appropriate for data analysis and player mod eling (as defined in this book).

5.5.2.2 Who Annotates?

Given the subjective nature of player experience the first natural question that comes in mind is who annotates players? In other words, who has the right authority and the best capacity to provide us with reliable tags of player experience? We dis tinguish two main categories: annotations can either be self-reports or reports ex pressed indirectly by experts or external observers [783].

In the first category the player states are provided by the players themselves and we call that first-person annotation. For example, a player is asked to rate the level of engagement while watching her playthrough video. First-person is clearly the most direct way to annotate a player state and build a model based on the solicited annotations. We can only assume there is disparity between the true (inner) expe rience of each player and the experience as felt by herself or perceived by others.

Based on this assumption the player's annotations should normally be closer to her inner experience (ground truth) compared to third-person annotation. First-person annotation, however, may suffer from self-deception and memory limitations [778]. These limitations have been attributed mainly to the discrepancies between "the ex periencing self" and "the remembering self" of a person [318] which is also known as the memory-experience gap [462].

Expert annotators—as a response to the above limitations—may instead be able to surpass the perception of experience and reach out to the inner experience of the player. In this second annotation category, named third-person annotation, an expert—such as a game designer—or an external observer provides the player state

in a more objective manner, thereby reducing the subjective biases of first-person perceptions. For instance, a user experience analyst may provide particular player state tags while observing a first-person shooter deathmatch game. The benefit of third-person annotation is that multiple annotators can be used for a particular game play experience. In fact, the availability of several such subjective perceptions of experience may allow us to approximate the ground truth better as the agreement between many annotators enhances the validity of our data directly. A potential disagreement, on the other hand, might suggest that the gameplay experience we examine is non-trivial or may indicate that some of our annotators are untrained or inexperienced.

5.5.2.3 How Is Player Experience Represented?

Another key question is how player experience is best represented: as a number of different states (discrete) or, alternatively, as a set of dimensions (continuous)? On

one hand, discrete labeling is practical as a means of representing player experience since the labels can easily form individual items (e.g., “excited”, “annoyed” etc.) in an experience questionnaire, making it easy to ask the annotator/player to pick one (e.g., in [621]). Continuous labeling, on the other hand, appears to be advantageous for two key reasons. First, experiential states such as immersion are hard to capture with words or linguistic expressions that have fuzzy boundaries. Second, states do not allow for variations in experience intensity over time since they are binary: ei ther the state is present or not. For example, the complex notions of fun, or even engagement, cannot be easily captured by their corresponding linguistic represen tation in a questionnaire or define well a particular state of a playing experience.
Instead it seems natural to represent them as a continuum of experience intensity that may vary over time. For these reasons we often observe low agreement among the annotators [143] when we represent playing experience via discrete states.
As discussed earlier, the dominant approach in continuous annotation is the use of Russell’s two-dimensional (arousal-valence) circumplex model of affect [581] (see Fig. 5.2(a)). Figure 5.5 illustrates two different annotation tools (FeelTrace and AffectRank) that are based on the arousal-valence circumplex model of affect. Figure 5.6 depicts the RankTrace continuous annotation tool which can be used for the annota

Annotation can happen either within particular time intervals or continuously. Time continuous annotation has been popularized due to the existence of freely available tools such as FeelTrace [144] (see Fig. 5.5(c)) and GTrace [145], which allows for continuous annotation of content (mostly videos and speech) across the dimensions of arousal and valence. In addition to FeelTrace there are annotation tools like the
5.5. What Is the Model’s Output Like? 223
continuous measurement system [454] and EmuJoy [474], where the latter is de signed for the annotation of music content. User interfaces such as wheels and knobs linked to the above annotation tools show further promise for the continu ous annotation of experience in games [125, 397, 97] (see Fig. 5.6). The continuous annotation process, however, appears to require a higher amount of cognitive load compared to a time-discrete annotation protocol. Higher cognitive loads often result in lower levels of agreement between different annotators and yield unreliable data for modeling player experience [166, 418].
As a response to the above limitations, time-discrete annotation provides data at particular intervals when the annotator feels there is a change in the player’s state. And changes are best indicated relatively rather than absolutely. AffectRank, for instance (see Fig. 5.5(b)), is a discrete, rank-based annotation tool that can be used for the annotation of any type of content including images, video, text or speech and it provides annotations that are significantly more reliable (with respect to inter-rater agreement) than the annotations obtained from continuous annotation tools such as FeelTrace [777]. The rank-based design of AffectRank is motivated by observations of recent studies in third-person video annotation indicating that “. . . humans are better at rating emotions in relative rather than absolute terms.” [455, 777]. Further, AffectRank is grounded in numerous findings showcasing the supremacy of ranks over ratings for obtaining annotations of lower inconsistency and order effects [777, 773, 778, 436, 455, 761].
A recent tool that builds on the relative-based annotation of AffectRank and al lows for the annotation of affect in a continuous yet unbounded fashion is RankTrace (see Fig. 5.6). The core idea behind RankTrace is introduced in [125]: the tool asks participants to watch the recorded playthrough of a play session and annotate in real-time the perceived intensity of a single emotional dimension. The annotation process in RankTrace is controlled through a “wheel-like” hardware, allowing par ticipants to meticulously increase or decrease emotional intensity by turning the wheel, similarly to how volume is controlled in a stereo system. Further, the general interfacing design of RankTrace builds on the one-dimensional GTrace annotation

tool [145]. Unlike other continuous annotation tools, however, annotation in Rank Trace is unbounded: participants can continuously increase or decrease the intensity as desired without constraining themselves to an absolute scale. This design deci sion is built on the anchor [607] and adaptation level [258] psychology theories by which affect is a temporal notion based on earlier experience that is best expressed in relative terms [765, 777, 397]. The use of RankTrace has revealed the benefits of relative and unbounded annotation for modeling affect more reliably [397] and has also showed promise for the construction of general models of player emotion across games [97].

5.5.2.5 When to Annotate?

When is it best to annotate experience: before, during or after play (see Fig. 5.1)? In a pre-experience questionnaire we usually ask annotators to set the baseline of

(a) Annotating facial expressions of players during play. The annotation is context-dependent as the video includes the gameplay of the player (see top left corner).
(b) AffectRank: A time-discrete annotation tool for arousal and valence.
(c) FeelTrace. A time-continuous annotation tool for arousal and valence.

Fig. 5.5 An example of third-person annotation based on videos of players and their gameplay using either (a) the AffectRank (b) or the FeelTrace (c) annotation tool. AffectRank is freely avail able at: https://github.com/TAPeri/AffectRank. FeelTrace is freely available at: http://emotion research.net/download/Feeltrace%20Package.zip.

a player's state prior to playing a game. This state can be influenced by a number of factors such as the mood of the day, the social network activity, the caffeine consumption, earlier playing activity and so on. This is a wealth of information that can be used to enrich our models. Again, what is worth detecting is the relative change [765] from the baseline state of the user prior playing to the game.

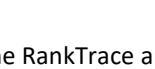


Fig. 5.6 The RankTrace annotation tool. In this example the tool is used for the annotation of tension in horror games. Participants play a game and then they annotate the level of tension by watching a video-recorded playthrough of their game session (top of image). The annotation trace is controlled via a wheel-like user interface. The entire annotation trace is shown for the partici pant's own reference (bottom of image). Image adapted from [397]. The RankTrace tool is available at: http://www.autogamedesign.eu/software.

A during-experience protocol, on the other hand, may involve the player in a first-person think-aloud setup [555] or a third-person annotation design. For the latter protocol you may think of user experience experts that observe and annotate player experience during the beta release of a game, for example. As mentioned earlier, first-person annotation during play is a rather intrusive process that disrupts the gameplay and risks adding experimental noise to annotation data. In contrast, third-person annotation is not intrusive; however, there are expected deviations from the actual first-person experience, which is inaccessible to the observer.

The most popular approach for the annotation of player experience is after a game (or a series of games) has been played, in a post-experience fashion. Post-experience annotation is unobtrusive for the player and it is usually performed by the players themselves. Self-reports, however, are memory-dependent by nature and memory is, in turn, time-dependent. Thus, one needs to consider carefully the time window be tween the experience and its corresponding report. For the reported post-experience to be a good approximation of the actual experience the playing time window needs to be small in order to minimize memory biases, yet sufficiently large to elicit par ticular experiences to the player. The higher the cognitive load required to retrieve the gameplay context is, the more the reports are memory-biased and not relevant to

the actual experience. Further, the longer the time window between the real experi ence and the self-report the more the annotator activates aspects of episodic memory
associated with the gameplay [571]. Episodic memory traces that form the basis of

self-reports fade over time, but the precise rate at which this memory decay occurs
is unknown and most likely individual [571]. Ideally, memory decay is so slow that
the annotator will have a clear feeling of the gameplay session when annotating it.
Now, if the time window becomes substantial—on the scale of hours and days—the
annotator has to activate aspects of semantic memory such as general beliefs about
a game. In summary, the more the episodic memory, and even more so the seman tic memory, are activated during annotation, the more systematic errors are induced
within the annotation data.
As a general rule of thumb the longer it takes for us to evaluate an experience
of ours the larger the discrepancy between the true experience and the evaluation of
the experience, which is usually more intense than the true experience. It also seems
that this gap between our memory of experience and our real experience is more
prominent when we report unpleasant emotions such as anger, sadness and tension
rather than positive emotions [462]. Another bias that affects how we report our
experience is the experience felt near the end of a session, a game level or a game;
this effect has been named peak-end rule [462].
An effective way to assist episodic memory and minimize post-experience cogni tive load is to show annotators replay videos of their gameplay (or the gameplay of
others) and ask them to annotate those. This can be achieved via crowdsourcing [96]
in a third-person manner or on a first-person annotation basis [125, 271, 397, 97].
Another notable approach in this direction is the data-driven retrospective inter viewing method [187]. According to that method player behavioral data is collected
and is analyzed to drive the construction of interview questions. These questions are
then used in retrospect (post-experience) to reflect on the annotator's behavior.
5.5.2.6 Which Annotation Type?
We often are uncertain about the type of labels we wish to assign to a player state or
a player experience. In particular, we can select from three data types for our annota tion: ratings, classes, and ranks (see Fig. 5.1). The rating-based format represents a
player's state with a scalar value or a vector of values. Ratings are arguably the dom inant practice for quantitatively assessing aspects of a user's behavior, experience,
opinion or emotion. In fact, the vast majority of user and psychometric studies have
adopted rating questionnaires to capture the opinions, preferences and perceived ex periences of experiment participants—see [78, 442, 119] among many. The most
popular rating-based questionnaire follows the principles of a Likert scale [384] in
which users are asked to specify their level of agreement with (or disagreement
against) a given statement—see Fig. 5.7(a) for an example. Other popular rating based questionnaires for user and player experience annotation include the Geneva
Wheel model [600], the Self-Assessment Manikin [468], the Positive and Negative
Affect Schedule [646], the Game Experience Questionnaire [286], the Flow State
Scale [293] and the Player Experience of Need Satisfaction (PENS) survey [583],
which was developed based on self-determination theory [162].

Rating-based reporting has notable inherent limitations that are often over looked, resulting in fundamentally flawed analyses [778, 298]. First, ratings are
analyzed traditionally by comparing their values across participants; see [233, 427]
among many. While this is a generally accepted and dominant practice it neglects
the existence of inter-personal differences as the meaning of each level on a rat ing scale may differ across experiment participants. For example, two participants
assessing the difficulty of a level may assess it as exactly the same difficulty, but
then one rates it as "very easy to play" and the other as "extremely easy to play".
It turns out that there are numerous factors that contribute to the different internal
rating scales existent across participants [455] such as differences in personality,
culture [643], temperament and interests [740]. Further, a large volume of studies

has also identified the presence of primacy and recency order effects in rating-based questionnaires (e.g., [113, 773]), systematic biases towards parts of the scale [388] (e.g., right-handed participants may tend to use the right side of the scale) or a fixed tendency over time (e.g., on a series of experimental conditions, the last ones are rated higher). Indicatively, the comparative study of [773] between ratings and ranks showcases higher inconsistency effects and significant order (recency) effects existent in ratings.

In addition to inter-personal differences, a critical limitation arises when ratings are treated as interval values since ratings are by nature ordinal values [657, 298]. Strictly speaking, any approach or method that treats ratings as numbers by, for instance, averaging their ordinal labels is fundamentally flawed. In most question naires Likert items are represented as pictures (e.g., different representations of arousal in the Self-Assessment Manikin [468]) or as adjectives (e.g., “moderately”, “fairly” and “extremely”). These labels (images or adjectives) are often erroneously converted to integer numbers, violating basic axioms of statistics which suggest that ordinal values cannot be treated as interval values [657] since the underlying nu merical scale is unknown. Note that even when a questionnaire features ratings as numbers (e.g., see Fig. 5.7(a)), the scale is still ordinal as the numbers in the in strument represent labels. Thus, the underlying numerical scale is still unknown and dependent on the participant [657, 515, 361]. Treating ratings as interval values is grounded in the assumption that the difference between consecutive ratings is fixed and equal. However, there is no valid assumption suggesting that a subjective rating scale is linear [298]. For instance, the difference between “fairly (4) ” and “ex tremely (5)” may be larger than the distance between “moderately (3)” and “fairly (4)” as some experiment participants rarely use the extremes of the scale or tend to use one extreme more than the other [361]. If, instead, ratings are treated naturally as ordinal data no assumptions are made about the distance between rating labels, which eliminates introducing data noise to the analysis.

The second data type for the annotation of players is the class-based format. Classes allow annotators to select from a finite and non-structured set of options and, thus, a class-based questionnaire provides nominal data among two (binary) or more options. The questionnaire asks subjects to pick a player state from a particu lar representation which could vary from a simple boolean question (was that game level frustrating or not? is this a sad facial expression? which level was the most

stressful?) to a player state selection from, for instance, the circumplex model of affect (is this a high- or a low-arousal game state for the player?). The limitations of ratings are mitigated, in part, via the use of class-based questionnaires. By not providing information about the intensity of each player state, however, classes do not have the level of granularity ratings naturally have. A class-based questionnaire might also yield annotations with an unbalanced number of samples per class. A common practice in psychometrics consists of transforming sets of consecutive rat ings into separate classes (e.g., see [226, 260] among many). In an example study [255], arousal ratings on a 7-point scale are transformed into high, neutral and low arousal classes using 7-5, 4 and 3-1 ratings, respectively. While doing so might seem appropriate, the ordinal relation among classes is not being taken into account. More importantly, the transformation process adds a new set of bias to the subjectivity bias of ratings, namely class splitting criteria [436].

Finally, rank-based questionnaires ask the annotator to rank a preference among options such as two or more sessions of the game [763]. In its simplest form, the annotator compares two options and specifies which one is preferred under a given statement (pairwise preference). With more than two options, the participants are asked to provide a ranking of some or all the options. Examples of rank-based ques tions include: was that level more engaging than this level? which facial expres sion looks happier?). Another example of a rank-based questionnaire (4-alternative forced choice) is illustrated in Fig. 5.7(b). Being a form of subjective reporting, rank-based questionnaires (as much as rating-based and class-based questionnaires)

are associated with the well known limitations of memory biases and self-deception. Reporting about subjective constructs such as experience, preference or emotion via rank-based questionnaires, however, has recently attracted the interest of researchers in marketing [167], psychology [72], user modeling [761, 37] and affective com puting [765, 721, 436, 455, 773] among other fields. This gradual paradigm shift is driven by both the reported benefits of ranks minimizing the effects of self-reporting subjectivity biases and recent findings demonstrating the advantages of ordinal an notation [765, 773, 455].

5.5.2.7 What Is the Value of Player Experience?

Describing, labeling and assigning values to subjective notions, such as player ex perience, is a non-trivial task as evidenced by a number of disciplines including neuroscience [607], psychology [258], economics [630], and artificial intelligence [315]. Annotators can attempt to assign numbers to such notions in an absolute manner, using for instance a rating scale. Annotators can alternatively assign val ues in a relative fashion, using for instance a ranking. There are, however, a mul titude of theoretical and practical reasons to doubt that subjective notions can be encoded as numbers in the first place [765]. For instance, according to Kahneman [317], co-founder of behavioral economics, "...it is safe to assume that changes are more accessible than absolute values"; his theory about judgment heuristics is built on Herbert Simon's psychology of bounded rationality [630]. Further, an important

(a) Rating: A 5-point Likert item example (b) Rank: A 4-alternative forced choice exam ple

Fig. 5.7 Examples of rating-based (a) vs. rank-based (b) questionnaires.

thesis in psychology, named adaptation level theory [258], suggests that humans lack the ability to maintain a constant value about subjective notions and their pref erences about options are, instead, made on a pairwise comparison basis using an internal ordinal scale [460]. The thesis claims that while we are efficient at discrimi nating among options, we are not good at assigning accurate absolute values for the intensity of what we perceive. For example, we are particularly bad at assigning ab solute values to tension, frequency and loudness of sounds, the brightness of an im age, or the arousal level of a video. The above theories have also been supported by neuroscientific evidence suggesting that experience with stimuli gradually creates our own internal context, or anchor [607], against which we rank any forthcoming stimulus or perceived experience. Thus, our choice about an option is driven by our internal ordinal representation of that particular option within a sample of options; not by any absolute value of that option [658].

As a remote observation, one may argue that the relative assessment provides less information than the absolute assessment since it does not express a quantity explicitly and only provides ordinal relations. As argued earlier, however, any addi tional information obtained in an absolute fashion (e.g., when ratings are treated as numbers) violates basic axioms of applied statistics. Thus the value of the additional information obtained (if any) is questioned directly [765].

In summary, results across different domains investigating subjective assessment suggest that relative (rank-based) annotations minimize the assumptions made about experiment participants' notions of highly subjective constructs such as player ex perience. Further, annotating experience in a relative fashion, instead of an absolute fashion, leads to the construction of more generalizable and accurate computational models of experience [765, 436].

5.5.3 No Output

Very often we are faced with datasets where target outputs about player behavioral or experience states are not available. In such instances modeling of players must rely on unsupervised learning [176, 244, 178] (see Fig. 5.1). Unsupervised learning,

as discussed in Chapter 2, focuses on fitting a model to observations by discovering associations of the input and without having access to a target output. The input is generally treated as a set of random variables and a model is built through the obser vations of associations among the input vectors. Unsupervised learning as applied to

modeling players involves tasks such as clustering and association mining which are described in Section 5.6.3.

It may also be the case that we do not have target outputs available but, neverthe less, we can design a reward function that characterizes behavioral or experiential patterns of play. In such instances we can use reinforcement learning approaches to discover policies about player behavior or player experience based on in-game play traces or other state-action representations (see Section 5.6.2). In the following section we detail the approaches used for modeling players in a supervised learning, reinforcement learning and unsupervised learning fashion.

5.6 How Can We Model Players?

In this section we build upon the data-driven approach of player modeling and dis cuss the application of supervised, reinforcement and unsupervised learning to model players, their behavior and their experience. To showcase the difference be tween the three learning approaches let us suppose we wish to classify player be havior. We can only use unsupervised learning if no behavioral classes have been defined a priori [176]. We can instead use supervised learning if, for example, we have already obtained an initial classification of players (either manually or via clus tering) and we wish to fit new players into these predefined classes [178]. Finally, we can use reinforcement learning to derive policies that imitate different types of playing behavior or style. In Section 5.6.1 we focus on the supervised learning paradigm whereas in Section 5.6.2 and Section 5.6.3 we outline, respectively, the re inforcement learning and the unsupervised learning approach for modeling players.

All three machine learning approaches are discussed in Chapter 2.

5.6.1 Supervised Learning

Player modeling consists of finding a function that maps a set of measurable at tributes of the player to a particular player state. Following the supervised learning approach this is achieved by machine learning, or automatically adjusting, the pa rameters of a model to fit a dataset that contains a set of input samples, each one paired with target outputs. The input samples correspond to the list of measurable attributes (or features) while the target outputs correspond to the annotations of the player's states for each of the input samples that we are interested to learn to pre dict. As mentioned already, the annotations may vary from behavioral characteris-

tics, such as completion times of a level or player archetypes, to estimates of player experience, such as player frustration.

As we saw in Chapter 2 popular supervised learning techniques, including artificial neural networks (shallow or deep architectures), decision trees, and support vector machines, can be used in games for the analysis, the imitation and the pre diction of player behavior, and the modeling of playing experience. The data type of the annotation determines the output of the model and, in turn, the type of the machine learning approach that can be applied. The three supervised learning alter natives for learning from numerical (or interval), nominal and ordinal annotations—respectively, regression, classification and preference learning—are discussed in this section.

5.6.1.1 Regression

When the outputs that a player model needs to approximate are interval values, the modeling problem is known as metric or standard regression. Any regression algo rithm is applicable to the task, including linear or polynomial regression, artificial neural networks and support vector machines. We refer the reader to Chapter 2 for details on a number of popular regression algorithms.

Regression algorithms are appropriate for imitation and prediction tasks of player behavior. When the task, however, is modeling of player experience cau tion needs to be put on the data analysis. While it is possible, for instance, to use regression algorithms to learn the exact numeric ratings of experience, in general it should be avoided because regression methods assume that the target values follow an interval (numerical) scale. Ratings naturally define an ordinal scale [765, 773] in stead. As mentioned already, ordinal scales such as ratings should not be converted

to numerical values due to the subjectivity inherent to reports, which imposes a non-uniform and varying distance among questionnaire items [778, 657]. Prediction models trained to approximate a real-value representation of a rating—even though they may achieve high prediction accuracies—do not necessarily capture the true reported playing experience because the ground truth used for training and valida tion of the model has been undermined by the numerous effects discussed above.

We argue that the known fundamental pitfalls of self-reporting outlined above pro vide sufficient evidence against the use of regression for player experience modeling [765, 515, 455, 361]. Thus, we leave the evaluation of regression methods on expe rience annotations outside the scope of this book.

5.6.1.2 Classification

Classification is the appropriate form of supervised learning for player modeling when the annotation values represent a finite and non-structured set of classes. Classification methods can infer a mapping between those classes and player at tributes. Available algorithms include artificial neural networks, decision trees, ran-

dom forests, support vector machines, K-nearest neighbors, and ensemble learning among many others. Further details about some of these algorithms can be found in Chapter 2.

Classes can represent playing behavior which needs to be imitated or predicted, such as completion times (e.g., expressed as low, average or high completion time) or user retention in a free-to-play game (e.g., expressed as weak, mild or strong retention). Classes can alternatively represent player experience such as an excited versus a frustrated player as manifested from facial expressions or low, neutral and high arousal states for a player.

Classification is perfectly suited for the task of modeling player experience if discrete annotations of experience are selected from a list of possibilities and pro vided as target outputs [153, 344]. In other words, annotations of player experience need to be nominal for classification to be applied. A common practice, however, as already mentioned in Section 5.5.2.6, is to treat ratings of experience as classes and transform the ordinal scale—that defines ratings—into a nominal scale of separate classes. For example, ratings of arousal that lie between -1 and 1 are transformed into low, neutral and high arousal classes. By classifying ratings not only the ordinal relation among the introduced classes is ignored but, most importantly, the transfor mation process induces several biases to the data (see Section 5.5.2.6). These biases appear to be detrimental and mislead the search towards the ground truth of player experience [765, 436].

5.6.1.3 Preference Learning

As an alternative to regression and classification methods, preference learning [215] methods are designed to learn from ordinal data such as ranks or preferences. It is important to note that the training signal in the preference learning paradigm merely provides information for the relative relation between instances of the phe nomenon we attempt to approximate. Target outputs that follow an ordinal scale do not provide information about the intensity (regression) or the clusters (classifica tion) of the phenomenon.

Generally we could construct a player model based on in-game behavioral pref erences. The information that this player, for example, prefers the mini-gun over a number of other weapons could form a set of pairwise preferences we could learn from. Alternatively we can build a model based on experience preferences. A player, for instance, reported that area X of the level is more challenging than area Y of the same level. Based on a set of such pairwise preferences we can derive a global func tion of challenge for that player.

As outlined in Chapter 2 a large palette of algorithms is available for the task of preference learning. Many popular classification and regression techniques have been adapted to tackle preference learning tasks, including linear statistical models such as linear discriminant analysis and large margins, and non-linear approaches such as Gaussian processes [122], deep and shallow artificial neural networks [430],

and support vector machines [302].
5.6. How Can We Model Players? 233
Preference learning has already been extensively applied to modeling aspects of players. For example, Mart´ınez et al. [430, 431] and Yannakakis et al. [780, 771] have explored several artificial neural network approaches to learn to predict af fective and cognitive states of players reported as pairwise preferences. Similarly, Garbarino et al. [217] have used linear discriminant analysis to learn pairwise en joyment predictors in racing games. To facilitate the use of proper machine learning methods on preference learning problems, a number of such preference learning methods as well as data preprocessing and feature selection algorithms have been made available as part of the Preference Learning Toolbox (PLT) [198]. PLT is an open-access, user-friendly and accessible toolkit5 built and constantly updated for the purpose of easing the processing of (and promoting the use of) ranks.
As ratings, by definition, express ordinal scales they can directly be transposed to any ordinal representation (e.g., pairwise preferences). For instance, given an anno tator's rating indicating that a condition A felt 'slightly frustrating' and a condition B felt 'very frustrating', a preference learning method can train a model that predicts a higher level of frustration for B than for A. In this way the modeling approach avoids introducing artifacts of what is the actual difference between 'very' and 'slightly' or the usage of the scale for this particular annotator. Further, the limitation of different subjective scales across users can be safely bypassed by transforming rating reports into ordinal relations on a per-annotator basis. Finally, the problem of the scale varying across time due to episodic memory still persists but can be minimized by transforming only consecutive reports, i.e., given a report for three conditions A, B and C, the player model can be trained using only the relation between A and B, and B and C (dropping the comparison between A and C).
5.6.1.4 Summary: The Good, the Bad and the Ugly
The last section on supervised learning is dedicated to the comparison among the three methods—regression, classification and preference learning—for modeling players. Arguably the discussion is limited when the in-game behavior of players is imitated or predicted. If behavioral data about players follows interval, nomi nal or ordinal scales then naturally, regression, classification and preference learn ing should be applied, respectively. Behavioral data have an objective nature which makes the task of learning less challenging. Given the subjective notion of player ex perience, however, there are a number of caveats and limitations of each algorithm that need to be taken into account. Below we discuss the comparative advantages of each and we summarize the key outcomes of supervised learning as applied to modeling the experience of players.
Regression vs. Preference Learning: Motivated by psychological studies sug gesting that interval ratings misrepresent experience [515, 455, 361], we will not dedicate ourselves an extensive comparison between preference learning and regres sion methods. The performance comparison between a regression and preference-
5 http://sourceforge.net/projects/pl-toolbox/

learned model is also irrelevant as the former is arguably a priori incapable of cap turing the underlying experience phenomenon as precisely as the latter. Such de viations from the ground truth, however, are not trivial to illustrate through a data modeling approach and thus the comparison is not straightforward. The main rea son is that the objective ground truth is fundamentally ill-defined when numbers are used to characterize subjective notions such as player experience.
Regression vs. Classification: Classes are easy to analyze and create player models from. Further, their use eliminates part of the inter-personal biases intro duced with ratings. For these reasons classification should be preferred to regres sion for player experience modeling. We already saw that classification, instead of regression, is applied when ratings are available to overcome part of the limitations

inherent in rating-based annotation. For instance, this can be achieved by transform ing arousal ratings to high, neutral and low arousal classes [255]. While this com mon practice in psychometrics eliminates part of the rating subjectivity it adds new forms of data biases inherent in the ad-hoc decisions to split the classes. Further, the analysis of player models across several case studies in the literature has already shown that transforming ratings into classes creates a more complicated machine learning problem [765, 436].

Classification vs. Preference Learning: Preference learning is the supreme method for modeling experience when ranks or pairwise preferences are available. Even when ratings or classes are available comparisons between classification and preference learning player models in the literature suggest that preference learn ing methods lead to more efficient, generic and robust models which capture more information about the ground truth [765]. Indicatively, Crammer and Signer [147] compare classification, regression and preference learning training algorithms in a task to learn ratings. They report the supremacy of preference learning over the other methods based on several synthetic datasets and a movie-ratings dataset. In addition, extensive evidence already shows that preference learning better approxi mates the underlying function between input (e.g., experience manifestations such as gameplay) and output (e.g., annotations) [436]. Figure 5.8 showcases how much closer a preference learned model can reach a hypothesized (artificial) ground truth, compared to a classification model trained on an artificial dataset. In summary, pref erence learning via rank-based annotation controls for reporting memory effects, eliminates subjectivity biases and builds models that are closer to the ground truth of player experience [778, 777].

Grounded in extensive evidence our final note for the selection of a supervised learning approach for modeling player experience is clear: Independently of how experience is annotated we argue that preference learning (the good) is a superior supervised learning method for the task at hand, classification (the ugly) provides a good balance between simplicity and approximation of the ground truth of player experience whereas regression (the bad) is based on rating annotations which are of questionable quality with respect to their relevance to the true experience.

(a) Ground Truth

(b) Classification (c) Preference learning

Fig. 5.8 A hypothesized (artificial) ground truth function (z-axis) which is dependent on two player attributes, x1 and x2 (Fig. 5.8(a)), the best classification model (Fig. 5.8(b)) and the best preference learned model (Fig. 5.8(c)). All images are retrieved from [436].

5.6.2 Reinforcement Learning

While it is possible to use reinforcement learning to model aspects of users dur ing their interaction the RL approach for modeling players has been tried mostly in comparatively simplistic and abstract games [195] and has not seen much applica-

tion in computer games. The key motivation for the use of RL for modeling players is that it can capture the relative valuation of game states as encoded internally by humans during play [685]. At first glance, player modeling with RL may seem to be an application of RL for game playing, and we discuss this in Chapter 3 as part of the play for experience aim. In this section, we instead discuss this approach from the perspective that a policy learned via RL can capture internal player states with no corresponding absolute target values such as decision making, learnability, cog nitive skills or emotive patterns. Further, those policies can be trained on player data such as play traces. The derived player model depicts psychometrically-valid, ab stract simulations of a human player's internal cognitive or affective processes. The model can be used directly to interpret human play, or indirectly, it can be featured in AI agents which can be used as playtesting bots during the game design process, as baselines for adapting agents to mimic classes of human players, or as believable, human-like opponents [268].

Using the RL paradigm, we can construct player models via RL if a reward sig nal can adjust a set of parameters that characterize the player. The reward signal can be based either directly on the in-game behavior of the player—for instance, the decision taken at a particular game state—or indirectly on player annotations (e.g., annotated excitement throughout the level) or objective data (e.g., physiological in dexes showing player stress). In other words, the immediate reward function can be based on gameplay data if the model wishes to predict the behavior of the player or, instead, be based on any objective measure or subjective report if the model attempts to predict the experience of the player. The representation of the RL approach can be anything from a standard Q table that, for instance, models the decision making behavior of a player (e.g., as in [268, 685]) to an ANN that e.g., models in-game behavior (as in [267]), to a set of behavior scripts [650] that are adjusted to imitate gameplay behavior via RL, to a deep Q network.

We can view two ways of constructing models of players via RL: models can be built offline (i.e., before gameplay starts) or at runtime (i.e., during play). We can also envision hybrid approaches by which models are first built offline and then are polished at runtime. Offline RL-based modeling adds value to our playtesting capacity via, for instance, procedural personas (see Section 5.7.1.3) whereas run time RL-based player modeling offers dynamicity to the model with respect to time.

Runtime player modeling further adds on the capacity of the model to adapt to the particular characteristics of the player, thereby increasing the degree of personaliza tion. We can think of models of players, for instance, that are continuously tailored to the current player by using the player's in-game annotations, behavioral deci sions, or even physiological responses during the game.

This way of modeling players is still in its infancy with only a few studies existent on player behavioral modeling in educational games [488], in roguelike adventure games [268, 267] via TD learning or evolutionary reinforcement learning, and in first-person shooter games [685] via inverse RL. However, the application of RL for modeling users beyond games has been quite active for the purposes of model ing web-usage data and interactions on the web [684] or modeling user simulations in dialog systems [114, 225, 595]. Normally in such systems a statistical model is

first trained on a corpus of human-computer interaction data for simulating (imitat ing) user behavior. Then reinforcement learning is used to tailor the model towards an optimal dialog strategy which can be found through trial and error interactions between the user and the simulated user.

5.6.3 Unsupervised Learning

The aim of unsupervised learning (see Chapter 2) is to derive a model given a number of observations. Unlike in supervised learning, there is no specified target output. Unlike in reinforcement learning there is no reward signal. (In short, there is no training signal of any kind.) In unsupervised learning, the signal is hidden in ternally in the interconnections among data attributes. So far, unsupervised learning has mainly been applied to two core player modeling tasks: clustering behaviors and mining associations between player attributes. While Chapter 2 provides the general description of these unsupervised learning algorithms, in this section we focus on their specific application for modeling players.

5.6.3.1 Clustering

As discussed in Chapter 2, clustering is a form unsupervised learning aiming to find clusters in datasets so that data within a cluster is similar to each other and dissimilar to data in other clusters. When it comes to the analysis of user behavior in games, clustering offers a means for reducing the dimensionality of a dataset, thereby yielding a manageable number of critical features that represent user behav ior. Relevant data for clustering in games include player behavior, navigation pat terns, assets bought, items used, game genres played and so on. Clustering can be used to group players into archetypical playing patterns in an effort to evaluate how people play a particular game and as part of a user-oriented testing process [176].

Further, one of the key questions of user testing in games is whether people play the

game as intended. Clustering can be utilized to derive a number of different playing or behavioral styles directly addressing this question. Arguably, the key challenge in successfully applying clustering in games is that the derived clusters should have an intelligible meaning with respect to the game in question. Thus, clusters should be clearly interpretable and labeled in a language that is meaningful to the involved stakeholders (such as designers, artists and managers) [176, 178]. In the case studies of Section 5.7.1 we meet the above challenges and demonstrate the use of clustering in the popular Tomb Raider: Underworld (EIDOS interactive, 2008) game.

5.6.3.2 Frequent Pattern Mining

In Chapter 2 we defined frequent pattern mining as the set of problems and tech niques related to finding patterns and structures in data. Patterns include sequences and itemsets. Both frequent itemset mining (e.g., Apriori [6]) and frequent se quence mining (e.g., GSP [652]) are relevant and useful for player modeling. The key motivation for applying frequent pattern mining on game data is to find inherent and hidden regularities in the data. In that regard, key player modeling problems, such as player type identification and detection of player behavior patterns, can be viewed as frequent pattern mining problems. Frequent pattern mining can for ex ample be used to to discover what game content is often purchased together—e.g., players that buy X tend to buy Y too—or what are the subsequent actions after dying in a level—e.g., players that die often in the tutorial level pick up more health packs in level 1 [120, 621].

5.7 What Can We Model?

As already outlined at the beginning of this chapter, modeling of users in games can be classified into two main tasks: modeling of the behavior of players and modeling of their experience. It is important to remember that modeling of player behavior is (mostly) a task of objective nature whereas the modeling of player experience is subjective given the idiosyncratic nature of playing experience. The examples we present in the remainder of this chapter highlight the various uses of AI for modeling players.

5.7.1 Player Behavior

In this section, we exemplify player behavior modeling via three representative use cases. The two first examples are based on a series of studies on player modeling by Drachen et al. in 2009 [176] and later on by Mahlmann et al. in 2010 [414] in the Tomb Raider: Underworld (EIDOS interactive, 2008) game. The analysis includes both the clustering of players [176] and the prediction [414] of their behavior, which make it an ideal case study for the purposes of this book. The third study presented in this section focuses on the use of play traces for the procedural creation of player models. That case study explores the creation of procedural personas in the MiniDungeons puzzle roguelike game.

Fig. 5.9 A screenshot from the Tomb Raider: Underworld (EIDOS interactive, 2008) game fea turing the player character, Lara Croft. The game is used as one of the player behavior modeling case studies presented in this book. Image obtained from Wikipedia (fair use).

5.7.1.1 Clustering Players in Tomb Raider: Underworld

Tomb Raider: Underworld (TRU) is a third-person perspective, advanced platform puzzle game, where the player has to combine strategic thinking in planning the 3D-movements of Lara Croft (the game's player character) and problem solving in order to go through a series of puzzles and navigate through a number of levels (see Fig. 5.9).

The dataset used for this study includes entries from 25,240 players. The 1,365 of those that completed the game were selected and used for the analysis presented below. Note that TRU consists of seven main levels plus a tutorial level. Six features of gameplay behavior were extracted from the data and are as follows: number of deaths by opponents, number of deaths by the environment (e.g., fire, traps, etc.), number of deaths by falling (e.g., from ledges), total number of deaths, game com pletion time, and the times help was requested. All six features were calculated on

the basis of completed TRU games. The selection of these particular features was based on the core game design of the TRU game and their potential impact on the process of distinguishing among dissimilar patterns of play.

Three different clustering techniques were applied to the task of identifying the number of meaningful and interpretable clusters of players in the data: k-means, hierarchical clustering and self-organizing maps. While the first two have been cov ered in Chapter 2 we will briefly outline the third method here.

A self-organizing map (SOM) [347] creates and iteratively adjusts a low dimensional projection of the input space via vector quantization. In particular, a type of large SOM called an emergent self-organizing map [727] was used in con junction with reliable visualization techniques to help us identify clusters. A SOM consists of neurons organized in a low-dimensional grid. Each neuron in the grid

(map) is connected to the input vector through a connection weight vector. In ad dition to the input vector, the neurons are connected to neighbor neurons of the map through neighborhood interconnections which generate the structure of the map: rectangular and hexagonal lattices organized in a two-dimensional sheet or a three-dimensional toroid shape are some of the most popular topologies used. SOM training can be viewed as a vector quantization algorithm which resembles the k-means algorithm. What differentiates SOM, however, is the update of the topo logical neighbors of the best-matching neuron—a best-matching neuron is a neuron for which there exists at least one input vector for which the Euclidean distance to the weight vector of this neuron is minimal. As a result, the whole neuron neighbor hood is stretched towards the presented input vector. The outcome of SOM training is that neighboring neurons have similar weight vectors which can be used for pro jecting the input data to the two-dimensional space and thereby clustering a set of data through observation on a 2D plane. For a more detailed description of SOMs, the reader is referred to [347].

To get some insight into the possible number of clusters existent in the data, k means was applied for all k values less than or equal to 20. The sum of the Euclidean distances between each player instance and its corresponding cluster centroid (i.e., quantization error) is calculated for all 20 trials of k-means. The analysis reveals that the percent decrease of the mean quantization error due to the increase of k is notably high when k = 3 and k = 4. For k = 3 and k = 4 this value equals 19% and 13% respectively while it lies between 7% and 2% for k > 4. Thus, the k-means clustering analysis provides the first indication of the existence of three or four main player behavioral clusters within the data.

As an alternative approach to k-means, hierarchical clustering is also applied to the dataset. This approach seeks to build a hierarchy of clusters existent in the data. The Ward's clustering method [747] is used to specify the clusters in the data by which the squared Euclidean distance is used as a measure of dissimilarity between data vector pairs. The resulting dendrogram is depicted in Fig. 5.10(a). As noted in Chapter 2 a dendrogram is a treelike diagram that illustrates the merging of data into clusters as a result of hierarchical clustering. It consists of many U-shaped lines connecting the clusters. The height of each U represents the squared Euclidean distance between the two clusters being connected. Depending on where the data analyst sets the squared Euclidean distance threshold a dissimilar number of clusters can be observed.

Both k-means and hierarchical clustering already demonstrate that the 1,365 players can be clustered in a low number of different player types. K-means in dicates there are for three or four clusters while the Ward's dendrogram reveals the existence of two populated and two smaller clusters, respectively, in the middle and at the edges of the tree.

Applying SOM, as the third alternative approach, allows us to cluster the TRU data by observation on a two-dimensional plane. The U-matrix depicted in Fig. 5.10(b) is a visualization of the local distance structure in the data placed onto a two-dimensional map. The average distance value between each neuron's weight

vector and the weight vectors of its immediate neighbors corresponds to the height
5.7. What Can We Model? 241
(a) Dendrogram of TRU data using Ward hierarchical clustering. A squared Euclidean distance of 4.5 (illustrated with a horizontal black line) reveals four clusters. Image adapted from [176].
(b) U-matrix visualization of a self-organizing map depicting the four player clusters identified in a population of 1,365 TRU players (shown as small colored squares). Different square colors depict different player clusters. Valleys represent clusters whereas mountains represent cluster borders. Image adopted from [176].
Fig. 5.10 Detecting player types from TRU data using hierarchical (a) and SOM (b) clustering methods.
of that neuron in the U-matrix (positioned at the map coordinates of the neuron).
242 Chapter 5. Modeling Players
Thus, U-matrix values are large in areas where no or few data points reside, creating mountain ranges for cluster boundaries. On the other hand, visualized valleys indi cate clusters of data as small U-matrix values are observed in areas where the data space distances of neurons are small.
The SOM analysis reveals four main classes of behavior (player types) as de picted in Fig. 5.10(b). The different colors of the U-matrix correspond to the four different clusters of players. In particular, cluster 1 (8.68% of the TRU players) corresponds to players that die very few times, their death is caused mainly by the environment, they do not request help from the game frequently and they complete the game very quickly. Given such game skills these players were labeled as Vet erans. Cluster 2 (22.12%) corresponds to players that die quite often (mainly due to falling), they take a long time to complete the game—indicating a slow-moving, careful style of play—and prefer to solve most puzzles in the game by themselves. Players of this cluster were labeled as Solvers, because they excel particularly at solving the puzzles of the game. Players of cluster 3 form the largest group of TRU players (46.18%) and are labeled as Pacifists as they die primarily from active op ponents. Finally, the group of players corresponding to cluster 4 (16.56% of TRU players), namely the Runners, is characterized by low completion times and frequent deaths by opponents and the environment.
The results showcase how clustering of player behavior can be useful to evalu ate game designs. Specifically, TRU players seem to not merely follow a specific strategy to complete the game but rather they fully explore the affordances provided by the game in dissimilar ways. The findings are directly applicable to TRU game design as clustering provides answers to the critical question of whether people play the game as intended. The main limitation of clustering, however, is that the derived clusters are not intuitively interpretable and that clusters need to be represented into meaningful behavioral patterns to be useful for game design. Collaborations be tween the data analyst and the designers of the game—as was performed in this study—is essential for meaningful interpretations of the derived clusters. The ben efit of such a collaboration is both the enhancement of game design features and the effective phenomenological debugging of the game [176]. In other words, we make sure both that no feature of the game is underused or misused and that the playing experience and the game balance are debugged.

5.7.1.2 Predicting Player Behavior in Tomb Raider: Underworld

Building on the same set of TRU player data, a second study examined the possi bilities of predicting particular aspects of playing behavior via supervised learning [414]. An aspect of player behavior that is particularly important for game design is to predict when a player will stop playing. As one of the perennial challenges of game design is to ensure that as many different types of players are facilitated in the design, being able to predict when players will stop playing a game is of interest because it assists with locating potentially problematic aspects of game design. Fur-
5.7. What Can We Model? 243
ther, such information helps toward the redesign of a game's monetization strategy for maximizing user retention.
Data was drawn from the Square Enix Europe Metrics Suite and was collected

during a two month period (1st Dec 2008 – 31st Jan 2009), providing records from approximately 203,000 players. For the player behavior prediction task it was de cided to extract a subsample of 10,000 players which provides a large enough and representative dataset for the aims of the study, while at the same time is manage able in terms of computational effort. A careful data-preprocessing approach yielded 6,430 players that were considered for the prediction task—these players had com pleted the first level of the game.

As in the TRU cluster analysis the features extracted from the data relate to the core mechanics of the game. In addition to the six features investigated in the cluster ing study of TRU the extracted features for this study include the number of times the adrenalin feature of the game was used, the number of rewards collected, the number of treasures found, and the number of times the player changes settings in the game (including player ammunition, enemy hit points, player hit points, and re covery time when performing platform jumps). Further details about these features can be found in [414].

To test the possibility of predicting the TRU level the player completed last a number of classification algorithms are tested on the data using the Weka machine learning software [243]. The approach followed was to experiment with at least one algorithm from each of the algorithm families existent in Weka and to put additional effort on those classification algorithms that were included in a recent list of the most important algorithms in data mining: decision tree induction, backpropagation and simple regression [759]. The resulting set of algorithms chosen for classification are as follows: logistic regression, multi-layer perceptron backpropagation, variants of decision trees, Bayesian networks, and support vector machines. In the following section, we only outline the most interesting results from those reported in [414]. For all tested algorithms, the reported classification prediction accuracy was achieved through 10-fold cross validation.

Most of the tested algorithms had similar levels of performance, and were able to predict when a player will stop playing substantially better than the baseline. In particular, considering only gameplay of level 1 classification algorithms reach an accuracy between 45% and 48%, which is substantially higher than the baseline per formance (39.8%). When using additional features from level 2, the predictions are much more accurate—between 50% and 78%—compared to the baseline (45.3%). In particular, decision trees and logistic regression manage to reach accuracies of almost 78% in predicting on what level a player will stop playing. The difference in the predictive strength of using level 1 and 2 data as compared to only level 1 data is partly due to increased amount of features used in the latter case.

Beyond accuracy an important feature of machine learning algorithms is their transparency and their expressiveness. The models are more useful to a data analyst and a game designer if they can be expressed in a form which is easy to visualize and comprehend. Decision trees—of the form constructed by the ID3 algorithm [544] and its many derivatives—are excellent from this perspective, especially if pruned to

Fig. 5.11 A decision tree trained by the ID3 algorithm [544] to predict when TRU players will stop playing the game. The leaves of the tree (ovals) indicate the number of the level (2, 3 or 7) the player is expected to complete. Note that the TRU game has seven levels in total. The tree is constrained to tree depth 2 and achieves a classification accuracy of 76.7%.

a small size. For instance, the extremely small decision tree depicted in Fig. 5.11 is constrained to tree depth 2 and was derived on the set of players who completed both levels 1 and 2 with a classification accuracy of 76.7%. The predictive capacity of the decision tree illustrated in Fig. 5.11 is impressive given how extremely simple it is. The fact that we can predict the final played level—with a high accuracy—based only on the amount of time spent in the room named Flush Tunnel of level 2 and the total rewards collected in level 2 is very appealing for game design. What this decision tree indicates is that the amount of time players spend within a given area early in the game and how well they perform are important for determining if they continue playing the game. Time spent on a task or in an area of the game can indeed

be indicative of challenges with progressing through the game, which can result in a frustrating experience.

5.7.1.3 Procedural Personas in MiniDungeons

Procedural personas are generative models of player behavior, meaning that they can replicate in-game behavior and be used for playing games in the same role as players; additionally, procedural personas are meant to represent archetypical players rather than individual players [268, 267, 269]. A procedural persona can be defined as the parameters of a utility vector that describe the preferences of a

player. For example, a player might allocate different weight to finishing a game fast, exploring dialog options, getting a high score, etc.; these preferences can be numerically encoded in a utility vector where each parameter corresponds to the persona's interest in a particular activity or outcome. Once these utilities are defined, reinforcement learning via TD learning or neuroevolution can be used to find a policy that reflects these utilities, or a tree search algorithm such as MCTS can be used with these utilities as evaluation functions. Approaches similar to the proce dural persona concept have also been used for modeling the learning process of the player in educational games via reinforcement learning [488].

As outlined in Section 5.4.1, MiniDungeons is a simple rogue-like game which features turn-based discrete movement, deterministic mechanics and full informa tion. The player avatar must reach the exit of each level to win it. Monsters block the way and can be destroyed, at the cost of decreasing the player character's health; health can be restored by collecting potions. Additionally, treasures are distributed throughout levels. In many levels, potions and treasures are placed behind monsters, and monsters block the shortest path from the entrance to the exit. Like many games, it is therefore possible to play MiniDungeons with different goals, such as reaching the exit in the shortest possible time, collecting as much treasure as possible or killing all the monsters (see Figs. 5.3 and 5.12).

These different playing styles can be formalized as procedural personas by at taching differing utilities to measures such as the number of treasures collected, the number of monsters killed or the number of turns taken to reach the exit. Q-learning can be used to learn policies that implement the appropriate persona in single lev els [268], and evolutionary algorithms can be used to train neural networks that implement a procedural persona across multiple levels [267]. These personas can be compared with the play traces of human players by placing the persona in ev ery situation that the human encountered in the play trace and comparing the action chosen by the procedural persona with the action chosen by the human (as you are comparing human actions with those of a Q function, you might say that you are asking "what would Q do?"). It is also possible to learn the utility values for a procedural persona clone of a particular human player by evolutionary search for those values that make the persona best match a particular play trace (see Fig. 5.12). However, it appears that these "clones" of human players generalize less well than designer-specified personas [269].

5.7.2 Player Experience

The modeling of player experience involves learning a set of target outputs that ap proximate the experience (as opposed of the behavior) of the player. By definition, that which is being modeled (experience) is of subjective nature and the model ing therefore requires target outputs that somehow approximate the ground truth of experience. A model of player experience predicts some aspect of the experience a player would have in some game situation, and learning such models is naturally a

(a) An artificial neural network mapping between a game state (input) and plans (output). The input contains blue circles rep resenting distances to various important elements of the game and a red circle representing hit points of the player character.

(b) A level of MiniDungeons 2 depicting the current state of the

game.
Fig. 5.12 A example of a procedural persona: In this example we evolve the weights of artificial neural networks—an ANN per persona. The ANN takes observations of the player character and its environment and uses these to choose from a selection of possible plans. During evolution the utility function of each persona is used as the fitness function for adjusting its network weights. Each individual of each generation is evaluated by simulating a full game. A utility function allows us to evolve a network to pursue multiple goals across a range of different situations. The method depends on the designer providing carefully chosen observations, appropriate planning algorithms, and well-constructed utility functions. In this example the player opts to move towards a safe treasure. This is illustrated with a green output neuron and highlighted corresponding weights (a) and a green path on the game level (b).
supervised learning problems. As mentioned, there are many ways this can be done, with approaches to player experience modeling varying regarding the inputs (from what the experience is predicted, e.g., physiology, level design parameters, playing style or game speed), the outputs (what sort of experience is predicted, e.g., fun, frustration, attention or immersion) and the modeling methodology.
In this section we will outline a few examples of supervised learning for model ing the experience of players. To best cover the material (methods, algorithms, uses of models) we rely on studies which have been thoroughly examined in the litera ture. In particular, in the remainder of this section we outline the various approaches and extensive methodology for modeling experience in two games: a variant of the popular Super Mario Bros (Nintendo, 1985) game and a 3D prey-predator game named Maze-Ball.
5.7.2.1 Modeling Player Experience in Super Mario Bros
Our first example builds upon the work of Pedersen et al. [521, 520] who modified an open-source clone of the classic platform game Super Mario Bros (Nintendo, 1985) to allow for personalized level generation. That work is important in that it set the foundations for the development of the experience-driven procedural content

generation framework [783] which constitutes a core research trend within proce dural content generation (see also Chapter 4).
The game used in this example is a modified version of Markus Persson's Infinite Mario Bros which is a public domain clone of Nintendo's classic platform game Super Mario Bros (Nintendo, 1985). All experiments reported in this example rely on a model-free approach for the modeling of player experience. Models of player experience are based on both the level played (game context) and the player's play ing style. While playing, the game recorded a number of behavioral metrics of the players, such as the frequency of jumping, running and shooting, that are taken into consideration for modeling player experience. Further, in a follow-up experiment [610], the videos of players playing the game were also recorded and used to extract a number of useful visual cues such as the average head movement during play. The output (ground truth of experience) for all experiments is provided as first-person, rank-based reports obtained via crowdsourcing. Data was crowdsourced from hun dreds of players, who played pairs of levels with different level parameters (e.g., dissimilar numbers, sizes and placements of gaps) and were asked to rank which of two levels best induced a number of player states. Across the several studies of player experience modeling for this variant of Super Mario Bros (Nintendo, 1985) collectively players have been asked to annotate fun, engagement, challenge, frus tration, predictability, anxiety, and boredom. These are the target outputs the player model needs to predict based on the input parameters discussed above.
Given the rank-based nature of the annotations the use of preference learning is necessary for the construction of the player model. The collected data is used to train artificial neural networks that predict the players' experiential states, given a player's behavior (and/or affective manifestations) and a particular game context, using evo lutionary preference learning. In neuroevolutionary preference learning [763], a genetic algorithm evolves an artificial neural network so that its output matches the pairwise preferences in the data set. The input of the artificial neural network is

a set of features that have been extracted from the data set—as mentioned earlier, the input may include gameplay and/or objective data in this example. It is worth noting that automatic feature selection is applied to pick the set of features (model input) that are relevant for predicting variant aspects of player experience. The ge netic algorithm implemented uses a fitness function that measures the difference between the reported preferences and the relative magnitude of the model output. Neuroevolutionary preference learning has been used broadly in the player model ing literature and the interested reader may refer to the following studies (among many): [432, 610, 763, 521, 520, 772].

The crowdsourcing experiment of Pedersen et al. [521, 520] resulted in data (gameplay and subjective reports of experience) from 181 players. The best pre dictor of reported fun reached an accuracy of around 70% on unseen subjective reports of fun. The input of the neural network fun-model is obtained through au tomatic feature selection consists of the time Mario spent moving left, the number of opponents Mario killed from stomping, and the percentage of level played in the left direction. All three playing features appear to contribute positively for the prediction of reported fun in the game. The best-performing model for challenge

Fig. 5.13 Facial feature tracking for head movement. Image adapted from [610].

prediction had an accuracy of approximately 78%. It is more complex than the best fun predictor, using five features: time Mario spends standing still, jump difficulty, coin blocks Mario pressed, number of cannonballs Mario killed, and Mario kills by stomping. Finally, the best predictor for frustration reaches an accuracy of 89%. It is indeed an impressive finding that a player experience model can predict (with near-certainty) whether the player is frustrated by the current game by merely cal culating the time Mario spent standing still, the time Mario spent on its last life, the jump difficulty, and the deaths Mario had from falling in gaps. The general findings of Pedersen et al. [520] suggest that good predictors for experience can be found if a preference learning approach is applied on crowdsourced reports of experience and gameplay data. The prediction accuracies, however, depend on the complexity of the reported state—arguably fun is a much more complicated and fuzzier notion to report compared to challenge or frustration. In a follow up study by Pedersen et al. [521] the additional player states of predictability, anxiety and boredom were predicted with accuracies of approximately 77%, 70% and 61%, respectively. The same player experience methodology was tested on an even larger scale, soliciting data from a total number of 780 players of the game [621]. Frequent pattern mining algorithms were applied to the data to derive frequent sequences of player actions. Using sequential features of gameplay the models reach accuracies of up to 84% for engagement, 83% for frustration and 79% for challenge.

In addition to the behavioral characteristics the visual cues of the player can be taken into account as objective input to the player model. In [610] visual features were extracted from videos of 58 players, both throughout whole game sessions and during small periods of critical events such as when a player completes a level or when the player loses a life (see Figs. 5.13 and 5.14). The visual cues enhance the quality of the information we have about a player's affective state which, in turn, allows us to better approximate player experience. Specifically, fusing the game play and the visual reaction features as inputs to the artificial neural network we achieve average accuracies of up to 84%, 86% and 84% for predicting reported en gagement, frustration and challenge, respectively. The key findings of [610] suggest

(a) Winning (b) Losing

(c) Experiencing challenge (d) Experiencing challenge

Fig. 5.14 Examples of facial expressions of Super Mario Bros (Nintendo, 1985) players for differ ent game states. All images are retrieved from the Platformer Experience Dataset [326].

that players' visual reactions can provide a rich source of information for modeling experience preferences and lead to more accurate models of player experience.

5.7.2.2 Modeling Player Experience in Maze-Ball

Our second example for modeling player experience builds largely upon the exten sive studies of Mart´ınez et al. [434, 430, 435] who analyzed player experience us ing a simple 3D prey-predator game named Maze-Ball towards achieving affective driven camera control in games. While the game is rather simple, the work on Maze Ball offers a thorough analysis of player experience via a set of sophisticated tech niques for capturing the psychophysiological patterns of players including prefer ence learning, frequent pattern mining and deep convolutional neural networks. In addition the dataset that resulted from these studies is publicly available for further experimentation and forms a number of suggested exercises for this book.

Maze-Ball is a three-dimensional prey-predator game (see Fig. 5.15); similar to a 3D version of Pac-Man (Namco,1981). The player (prey) controls a ball which

(a) Maze-Ball (b) Space-Maze

Fig. 5.15 Early Maze-Ball prototype (a) and a polished variant of the game (b) that features real-time camera adaptation [345]. The games can be found and played at http://www.hectorpmartinez.com/.

moves inside a maze hunted by 10 opponents (predators) moving around the maze. The goal of the player is to maximize her score by gathering as many tokens, scat tered in the maze, as possible while avoiding being touched by the opponents in a predefined time window of 90 seconds. A detailed description of Maze-Ball game play can be found in [780].

Gameplay attributes and physiological signals (skin conductance and heart rate variability) were acquired from 36 players of Maze-Ball. Each subject played a pre defined set of eight games for 90 seconds, thus the total number of game sessions available is 288. Gameplay and physiological signals define the input of the player experience model. To obtain the ground truth of experience players self-reported their preference about game pairs they played using a rank-based questionnaire, in particular a 4-alternative forced choice (4-AFC) protocol [780]. They were asked to rank the two games of each game pair with respect to fun, challenge, boredom, frus tration, excitement, anxiety and relaxation. These annotations are the target outputs the player model will attempt to predict based on the input parameters discussed above.

Several features were extracted from the gameplay and physiological signals ob tained. These included features related to kills and deaths in the game as well as features associated with the coverage of the level. For extracting features from the physiological signals the study considered their average, standard deviation, and maximum and minimum values. The complete feature list and the experimental pro tocol followed can be found in [780].

As in the example with the game variant of Super Mario Bros (Nintendo, 1985) the rank-based nature of the annotations requires the use of preference learning for the construction of the player experience model. Thus, the collected data is used to train neural networks that predict the player states, given a player's gameplay be havior and its physiological manifestations, using evolutionary preference learning.

The architecture of the neural network can be either shallow using a simple multi layered perceptron or deep (using convolutional neural networks [430, 435]). Figure 5.16 shows the different ways information from gameplay and physiology can been

Fig. 5.16 Three dissimilar approaches to deep multimodal fusion via convolutional neural net works. Gameplay events are fused with skin conductance in this example. The networks illustrated present two layers with one neuron each. The first convolutional layer receives as input a contin uous signal at a high time resolution, which is further reduced by a pooling layer. The resulting signal (feature map) presents a lower time resolution. The second convolutional layer can combine this feature map with additional modalities at the same low resolution. In the convolution fusion network (left figure), the two events are introduced at this level as a pulse signal. In the pooling fusion network (middle figure), the events are introduced as part of the first pooling layer, resulting in a filtered feature map. Finally, in the training fusion network (right figure), the events affect the training process of the first convolutional layer, leading to an alternative feature map. Image adapted from [435].

fused on a deep convolutional neural network which is trained via preference learning to predict player experience in any game experience dataset that contains discrete in-game events and continuous signals (e.g., the player's skin conductance). Predictors of the player experience states can reach accuracies that vary from 72% for challenge up to 86% for frustration using a shallow multi-layer percep tron player model [780]. Significant improvements are observed in those accuracies when the input space of the model is augmented with frequent sequences of in-game and physiological events (i.e., fusion on the input space). As in [621], Mart´ınez et al. used GSP to extract those frequent patterns that were subsequently used as in puts of the player model [434]. Further accuracy improvements can be observed when physiology is fused with physiological signals on deep architectures of con volutional neural networks [435]. Using deep fusion (see Fig. 5.16) accuracies of predicting player experience may surpass 82% for all player experience states con sidered. Further information about the results obtained in the Maze-Ball game can be found in the following studies: [780, 434, 430, 435, 436].

5.8 Further Reading

For an extensive reading on game and player analytics (including visualization, data preprocessing, data modeling and game domain-dependent tasks) we refer the reader to the edited book by El-Nasr et al. [186]. When it comes to player modeling two papers offer complementary perspectives and taxonomies of player modeling and a thorough discussion on what aspects of players can be modeled and the ways players can be modeled: the survey papers of Smith et al. [636] and Yannakakis et al. [782].

5.9 Exercises

In this section we propose a set of exercises for modeling both the behavior and the experience of game players. For that purpose, we outline a number of datasets that can be used directly for analysis. Please note, however, that the book's website will remain up to date with more datasets and corresponding exercises beyond the ones covered below.

5.9.1 Player Behavior

A proposed semester-long game data mining project is as follows. You have to choose a dataset containing player behavioral attributes to apply the necessary pre processing on the data such as extracting features and selecting features. Then you must apply a relevant unsupervised learning technique for compressing, analyz ing, or reducing the dimensionality of your dataset. Based on the outcome of unsu pervised learning you will need to implement a number of appropriate supervised learning techniques that learn to predict a data attribute (or a set of attributes). We leave the selection of algorithms to the reader or the course instructor. Below we dis cuss a number of example datasets one might wish to start from; the reader, however, may refer to the book's website for more options on game data mining projects.

5.9.1.1 SteamSpy Dataset

SteamSpy (http://steamspy.com/) is a rich dataset of thousands of games released on Steam[6] containing several attributes each. While strictly not a dataset focused on player modeling, SteamSpy offers an accessible and large dataset for game analyt ics. The data attributes of each game include the game's name, the developer, the publisher, the score rank of the game based on user reviews, the number of owners

[6] http://store.steampowered.com/

of the game on Steam, the people that have played this game since 2009, the people that have played this game in the last two weeks, the average and median playtime, the game's price and the game's tags. The reader may use an API[7] to download all data attributes from all games contained in the dataset. Then one might wish to apply supervised learning to be able to predict an attribute (e.g., the game's price) based

on other game features such as the game's score, release date and tags. Or alterna tively, one might wish to construct a score predictor of a new game. The selection of the modeling task and the AI methods is left to the reader.

5.9.1.2 StarCraft: Brood War Repository

The StarCraft: Brood War repository contains a number of datasets that include thousands of professional StarCraft replays. The various data mining papers, datasets as well as replay websites, crawlers, packages and analyzers have been compiled by Alberto Uriarte at Drexel University.[8]

In this exercise you are faced with the chal lenge of mining game replays with the aim to predict a player's strategy. Some results on the StarCraft: Brood War datasets can be found in [750, 728, 570] among others.

5.9.2 Player Experience

As a semester project on player experience modeling it is suggested you choose a game, one or more affective or cognitive states to model (model's output) and one or more input modalities. You are expected to collect empirical data using your selected game and build models for the selected psychological state of the players that rely on the chosen input modalities.

As a smaller project that does not involve data collection you may opt to choose one of the following datasets and implement a number of AI methods that will de rive accurate player experience models. The models should be compared in terms of a performance measure. The two datasets accompanying this book and outlined be low are the platformer experience dataset and the Maze-Ball dataset. The book's website will be up to date with more datasets and exercises beyond the ones covered below.

5.9.2.1 Platformer Experience Dataset

The extensive analysis of player experience in Super Mario Bros (Nintendo, 1985) and our wish to further advance our knowledge and understanding on player expe-

7 http://steamspy.com/api.php

8 Available at: http://nova.wolfwork.com/dataMining.html

rience had led to the construction of the Platformer Experience Dataset [326]. This is the first open-access game experience corpus that contains multiple modalities of data from players of Infinite Mario Bros, a variant of Super Mario Bros (Nintendo, 1985). The open-access database can be used to capture aspects of player experi ence based on behavioral and visual recordings of platform game players. In addi tion, the database contains aspects of the game context—such as level attributes—demographic data of the players and self-reported annotations of experience in two forms: ratings and ranks.

Here are a number of questions you might wish to consider when attempting to build player experience models that are as accurate as possible: Which AI methods should I use? How should I treat my output values? Which feature extraction and selection mechanism should I consider? The detailed description of the dataset can be found here: http://www.game.edu.mt/PED/. The book website contains further details and a set of exercises based on this dataset.

5.9.2.2 Maze-Ball Dataset

As in the case of the Platformer Experience Dataset the Maze-Ball dataset is also publicly available for further experimentation. This open-access game experience corpus contains two modalities of data obtained from Maze-Ball players: their gameplay attributes and three physiological signals: blood volume pulse, heart rate and skin conductance. In addition, the database contains aspects of the game such as features of the virtual camera placement. Finally the dataset contains demographic data of the players and self-reported annotations of experience in two forms: ratings and ranks.

The aim, once again, is to construct the most accurate models of experience for the players of Maze-Ball. So, which modalities of input will you consider? Which

annotations are more reliable for predicting player experience? How will your sig nals be processed? These are only a few of the possible questions you will en counter during your efforts. The detailed description of the dataset can be found
here: http://www.hectorpmartinez.com/. The book website contains further details
about this dataset.

5.10 Summary

This chapter focused on the use of AI for modeling players. The core reasons why
AI should be used for that purpose is either to derive something about the players'
experience (how they feel in a game) or for us to understand something about their
behavior (what they do in a game). In general we can model player behavior and
player experience by following a top-down or a bottom-up approach (or a mix of
the two). Top-down (or model-based) approaches have the advantage of solid the oretical
frameworks usually derived from other disciplines or other domains than
5.10. Summary 255
games. Bottom-up (or model-free) instead rely on data from players and have the
advantage of not assuming anything about players other than that player experience
and behavior are associated with data traces left by the player and that these data
traces are representative of the phenomenon we wish to explain. While a hybrid
between model-based and model-free approaches is in many ways a desirable ap proach to player
modeling, we focus on bottom-up approaches, where we provide
a detailed taxonomy for the options available regarding the input and the output of
the model, and the modeling mechanism per se. The chapter ends with a number
of player modeling examples, for modeling both the behavior of players and their
experience.

The player modeling chapter is the last chapter of the second part of this book,
which covered the core uses of AI in games. The next chapter introduces the third
and last part of the book, which focuses on the holistic synthesis of the various AI
areas, the various methods and the various users of games under a common game
AI framework.

Part III
The Road Ahead

Chapter 6
Game AI Panorama

This chapter attempts to give a high-level overview of the field of game AI, with
particular reference to how the different core research areas within this field inform
and interact with each other, both actually and potentially. For that purpose we first
identify the main research areas and their sub-areas within the game AI field. We
then view and analyze the areas from three key perspectives: (1) the dominant AI
method(s) used under each area; (2) the relation of each area with respect to the end
(human) user; and (3) the placement of each area within a human-computer (player game)
interaction perspective. In addition, for each of these areas we consider how
it could inform or interact with each of the other areas; in those cases where we find
that meaningful interaction either exists or is possible, we describe the character of
that interaction and provide references to published studies, if any.

The main motivations for us writing this chapter is to help the reader understand
how a particular area relates to other areas within this increasingly growing field,
how the reader can benefit from knowledge created in other areas and how the reader
can make her own research more relevant to other areas. To facilitate and foster
synergies across active research areas we place all key studies into a taxonomy with
the hope of developing a common understanding and vocabulary within the field of
AI and games. The structure of this chapter is based on the first holistic overview of
the game AI field presented in [785]. The book takes a new perspective on the key
game AI areas given its educational and research focus.

The main game AI areas and core subareas already identified in this book and
covered in this chapter are as follows:

- Play Games (see Chapter 3) which includes the subareas of Playing to Win and

Playing for the Experience. Independently of the purpose (winning or experi ence) AI can control either the player character or the non-player character.
• Generate Content (see Chapter 4) which includes the subareas of autonomous (procedural) content generation and assisted content generation. Please note that the terms assisted (procedural) content generation and mixed-initiative (proce dural) content generation (as defined in Chapter 4) are used interchangeably in this chapter.

• Model Players (see Chapter 5) which includes the subareas of player experience modeling and player behavior modeling, or else, game data mining [178].
The scope of this chapter is not to provide an inclusive survey of all game AI areas—the details of each area have been covered in preceding chapters of the book—but rather a roadmap of interconnections between them via representative examples. As research progresses in this field, new research questions will pop up and new methods be invented, and other questions and methods recede in impor tance. We believe that all taxonomies of research fields are by necessity tentative.
Consequently, the list of areas defined in this chapter should not be regarded as fixed and final.
The structure of the chapter is as follows: In Section 6.1, we start by holistically analyzing the game AI areas within the game AI field and we provide three alter native views over game AI: one with respect to the methods used, one with respect to the end users within game research and development and one where we outline how each of the research areas fits within the player-game interaction loop of digital games. Then, Section 6.2, digs deeper into the research areas and describes each one of them in detail. With the subsection describing each area, there is a short de scription of the area and a paragraph on the possible interactions with each of the other areas for which we have been able to identify strong or weak influences . The chapter ends with a section containing our key conclusions and vision for the future of the field.
6.1 Panoramic Views of Game AI
Analyzing any research field as a composition of various subareas with intercon nections and interdependencies can be achieved in several different ways. In this section we view game AI research from three high-level perspectives that focus on the computer (i.e., the AI methods), the human (i.e., the potential end user of game AI) and the interaction between the key end user (i.e., player) and the game. In stead in Section 6.2 we outline each game AI area and present the interconnections between the areas.
Game AI is composed of (a set of) methods, processes and algorithms in artificial intelligence as those are applied to, or inform the development of, games. Naturally, game AI can be analyzed through the method used by identifying the dominant AI approaches under each game AI area (see Section 6.1.1). Alternatively, game AI can be viewed from the game domain perspective with a focus on the end users of each game AI area (see Section 6.1.2). Finally game AI is, by nature, realized through systems that entail rich human-computer interaction (i.e., games) and, thus, the different areas can be mapped to the interaction framework between the player and the game (see Section 6.1.3).

Table 6.1 Dominant (•) and secondary (◦) AI methods for each of the core AI areas we cover in this book. The total number of methods used for each area appears at the bottom row of the table.

	Play Games		Generate Content		Model Players
	Winning	Experience	Autonomously	Assisted	Experience	Behavior
Behavior Authoring	•	•				
Tree Search	•	◦	◦	◦		
Evolutionary Computation	•	◦	•	•	•	
Supervised Learning	◦	•	•	•		
Reinforcement Learning	•	◦				

Unsupervised Learning ◦ ◦ •
Total (Dominant) 5 (4) 5 (2) 2 (1) 3 (1) 3 (2) 2 (2)

6.1.1 Methods (Computer) Perspective

The first panoramic view of game AI we present is centered around the AI methods used in the field. As the basis of this analysis we first list the core AI methods mostly used in the game AI field. The key methodology areas identified in Chapter 2 include ad-hoc behavior authoring, tree search, evolutionary computation, reinforcement learning, supervised learning, and unsupervised learning. For each of the game AI areas investigated we have identified the AI methods that are dominant or secondary in the area. While the dominant methods represent the most popular techniques used in the literature, secondary methods represent techniques that have been considered from a substantial volume of studies but are not dominant.

We have chosen to group methods according to what we perceive as a received taxonomy and following the structure of Chapter 2. While it would certainly be possible to classify the various methods differently, we argue that the proposed classification is compact (containing solely key methodology areas) and it follows standard method classifications in AI. While this taxonomy is commonly accepted, the lines can be blurred. In particular evolutionary computation, being a very general optimization method, can be used to perform supervised, unsupervised or reinforcement learning (more or less proficiently). The model-building aspect of reinforcement learning can be seen as a supervised learning problem (mapping from action sequences to rewards), and the commonly used tree search method Monte Carlo tree search can be seen as a form of TD learning. The result of any tree search algorithm can be seen as a plan, though it is often not guaranteed to lead to the desired end state. That the various methods have important commonalities and some overlap does not detract from the fact that each of them is clearly defined.

Table 6.1 illustrates the relationship between game AI areas and corresponding methods. It is evident that evolutionary computation and supervised learning appear to be of dominant or secondary use in most game AI areas. Evolutionary computation is a dominant method for playing to win, for generating content (in an assisted/mixed-initiative fashion or autonomously), and for modeling players; it has also been considered for the design of believable play (play for experience) research. Supervised learning is of substantial use across the game AI areas and appears to be

dominant in player experience and behavioral modeling, as well as in the area of AI that plays for experience. Behavior authoring, on the other hand, is useful solely for game-playing. Reinforcement learning and unsupervised learning find limited use across the game AI areas, respectively, being dominant only on AI that plays to win and player behavior modeling. Finally, tree search finds use primarily in playing to win and it is also considered—as a form of planning—for controlling play for experience and in computational narrative (as part of autonomous or assisted PCG).

Viewing Table 6.1 from the game AI areas' perspective (columns) it seems that AI that plays games (either for wining or for the experience) defines the game AI area with the most diverse and richest palette of AI methods. On the contrary, procedural content generation is solely dominated by evolutionary computation and tree search to a secondary degree. It is important to state that the popularity of any AI method within a particular area is closely tied to the task performed or the goal in mind. For example, evolutionary computation is largely regarded as a computationally heavy process which is mostly used in tasks associated with offline training. As PCG so far mainly relies on content that is generated offline, evolutionary computation offers a good candidate method and the core approach behind search-based PCG [720]. If online learning is a requirement for the task at hand, however, other methods (such as reinforcement learning or pruned tree-search) tend to be preferred.

Clearly the possibility space for future implementations of AI methods under particular game AI areas seems rather large. While particular methods have been traditionally dominant in specific areas for good reasons (e.g., planning in computational narrative) there are equally good reasons to believe that the research in a

game AI area itself has been heavily influenced by (and limited to) its correspond ing dominant AI methods. The empty cells of Table 6.1 indicate potential areas for exploration and offer us an alternative view of promising new intersections between game AI areas and methods.

6.1.2 End User (Human) Perspective

The second panoramic view of the game AI field puts an emphasis on the end user of the AI technology or general outcome (product or solution). Towards that aim we investigate three core dimensions of the game AI field and classify all game AI areas with respect to the process AI follows, the game context under which algorithms operate and, finally, the end user that benefits most from the resulting outcome. The classes identified under the above dimensions are used as the basis of the taxonomy we propose.

The first dimension (phrased as a question) refers to the AI process: In general, what can AI do within games? We identify two potential classes in this dimension: AI can model or generate. For instance, an artificial neural network can model a playing pattern, or a genetic algorithm can generate game assets. Given that AI can model or generate the second dimension refers to the context: What can AI methods model or generate in a game? The two possible classes here are content

Fig. 6.1 The end user perspective of the identified game AI areas. Each AI area follows a process (model or generate) under a context (content or behavior) for a particular end user (designer, player, AI researcher or game producer/publisher). Blue and red arrows represent the processes of modeling and generation, respectively. Modified graph from [785].

and behavior. For example, AI can model a players' affective state, or generate a level. Finally, the third dimension is the end user: AI can model, or generate, either content or behavior; but, for whom? The classes under the third dimension are the designer, the player, the AI researcher, and the producer/publisher.

Note that the above taxonomy serves as a framework for classifying the game AI areas according to the end user and is, by no means, inclusive of all potential processes, contexts, and end users. For instance, one could claim that the producer's role should be distinct from the publisher's role and that a developer should also be included in that class. Moreover, game content could be further split into smaller sub-classes such as narrative, levels, etc. Nevertheless, the proposed taxonomy pro vides distinct roles for the AI process (model vs. generate vs. evaluate), clear-cut classification for the context (content vs. behavior) and a high-level classification of the available stakeholders in game research and development (designer vs. player vs. AI researcher vs. producer/publisher). The taxonomy presented here is a modified version of the one introduced in [785] and it does not consider evaluation as a process for AI since it is out of the primary scope of this book.

Figure 6.1 depicts the relationship between the game AI core areas, the subareas and the end users in game research and development. Assisted, or mixed-initiative, content generation is useful for the designer and entails all possible combinations of processes and context as both content and behavior can be either modeled or gener ated for the designer. Compared to the other stakeholders the player benefits directly from more game AI research areas. In particular the player and her experience are affected by research on player modeling, which results from the modeling of ex perience and behavior; research on autonomous procedural content generation, as

a result of generation of content; and studies on NPC playing (for wining or expe rience) resulting from the generation of behavior. The player character (PC)-based game playing (for winning or experience) areas provide input to the AI researcher primarily. Finally, the game producer/publisher is primarily affected by results on behavioral player modeling, game analytics and game data mining as a result of behavior modeling.

6.1.3 Player-Game Interaction Perspective

The third and final panoramic perspective of game AI presented in this section couples the computational processes with the end user within a game and views

all game AI areas through a human-computer interaction—or, more accurately, a player-game interaction—lens. The analysis builds on the findings of Section 6.1.2 and places the five game AI areas that concern the player as an end user on a player game interaction framework as depicted in Fig. 6.2. Putting an emphasis on player experience and behavior, player modeling directly focuses on the interaction be tween a player and the game context. Game content is influenced primarily by re search on autonomous procedural content generation. In addition to other types of content, most games feature NPCs, the behavior of which is controlled by some form of AI. NPC behavior is informed by research in NPCs that play the game to win or any other playing-experience purpose such as believability.

Looking at the player-game interaction perspective of game AI it is obvious that the player modeling area has the most immediate and direct impact on the player experience as it is the only area linked to the player-game interaction directly. From the remaining areas, PCG influences player experience the most as all games have some form of environment representation and mechanics. Finally, AI that plays as an NPC (either to win or for the experience of play) is constrained to games that include agents or non-player characters.

The areas not considered directly in this game AI perspective affect the player rather remotely. Research on AI tools that assist the generative process of content improves the game's quality as a whole and in retrospect the player experience since designers tend to maintain a second-order player model [378] while designing. Fi nally, AI that plays the game as a player character can be offered for testing both the content and the NPC behaviors of a game, but also the interaction between the player and the game (via e.g., player experience competitions), but is mainly directed to AI researchers (see Fig. 6.1).

6.2 How Game AI Areas Inform Each Other

In this section, we outline the core game AI areas and discuss how they inform or influence (the terms are used interchangeably) each other. All research areas could

Fig. 6.2 Game AI areas and sub-areas viewed from a player-game interaction perspective.

be seen as potentially influencing each other to some degree; however, making a list of all such influences would be impractical and the result would be uninteresting. Therefore we only describe direct influences. Direct influences can be either strong (represented by a • as the bullet point style next to the corresponding influence in the following lists) or weak (represented by a ◦). We do not list influences we do not consider potentially important for the informed research area, or which only go through a third research area.

The sections below list outgoing influence. Therefore, to know how area A influences area B you should look in the section describing area A. Some influences are mutual, some not. The notation A → B in the headings of this section denotes that "A influences B". In addition to the text description each section provides a fig ure representing all outgoing influences of the area as arrows. Dark and light gray colored areas represent, respectively, strong and weak influence. Areas with white background are not influenced by the area under consideration. The figures also de pict the incoming influences from other areas. Incoming strong and weak influences are represented, respectively, with a solid line and a dashed line around the game AI areas that influence the area under consideration. Note that the description of the

incoming influence from an area is presented in the corresponding section of that area.

6.2.1 Play Games

The key area in which AI plays a game (as covered in Chapter 3) involves the sub areas of Playing to Win and Playing for Experience. As mentioned earlier in the chapter the AI can control either player or non-player characters of the game. We cover the influences to (and from) these subareas of game AI in this section.

6.2.1.1 Playing to Win (as a Player or as a Non-Player)

As already seen in Chapter 3 research in AI that learns to play (and win) a game

focuses on using reinforcement learning techniques such as temporal difference learning or evolutionary algorithms to learn policies/behaviors that play games well—whether it is a PC or an NPC playing the game. From the very beginning of AI research, reinforcement learning techniques have been applied to learn how to play board games (see for example Samuel's Checkers player [591]). Basically, playing the game is seen as a reinforcement learning problem, with the reinforcement tied to some measure of success in the game (e.g., the score, or length of time survived). As with all reinforcement learning problems, different methods can be used to solve the problem (find a good policy) [715] including TD learning [689], evolutionary computation [406], competitive co-evolution [24, 538, 589, 580], simulated anneal ing [42], other optimization algorithms and a large number of combinations between such algorithms [339]. In recent years a large number of papers that describe the application of various learning methods to different types of video games have ap peared in the literature (including several overviews [470, 406, 632, 457]). Finally, using games to develop artificial general intelligence builds on the idea that games can be useful environments for algorithms to learn complex and useful behaviors; thus research in algorithms that learn to win is essential.

It is also worth noting that most existing game-based benchmarks measure how well an agent plays a game—see for example [322, 404, 504]. Methods for learning to play a game are vital for such benchmarks, as the benchmarks are only meaningful in the context of the algorithms. When algorithms are developed that "beat" existing benchmarks, new benchmarks need to be developed. For example, the success of an early planning agent in the first Mario AI competition necessitated that the software be augmented with a better level generator for the next competition [322], and for the Simulated Car Racing competition, the performance of the best agents on the original competition game spurred the change to a new more sophisticated racing game [710, 392].

Fig. 6.3 Playing to Win: influence on (and from) other game AI research areas. Outgoing influence (represented by arrows): black and dark gray colored areas reached by arrows represent, respectively, strong and weak influence. Incoming influence is represented by red lines around the areas that influence the area under investigation (i.e., AI that plays to win in this figure): strong and weak influences are represented, respectively, by a solid and a dashed line.

Research in this area impacts game AI at large as three game AI subareas are directly affected; in turn, one subarea is directly affecting AI that plays to win (see Fig. 6.3).

• Playing to Win → Playing for the Experience: An agent cannot be believable or existent to augment the game's experience if it is not proficient. Being able to play a game well is in several ways a precondition for playing games in a be lievable manner though well playing agents can be developed without learning (e.g., via top-down approaches). In recent years, successful entries to competi tions focused on believable agents, such as the 2K BotPrize and the Mario AI Championship Turing test track, have included a healthy dose of learning algo rithms [719, 603].

• Playing to Win → Generate Content (Autonomously): Having an agent that is capable of playing a game proficiently is useful for simulation-based testing in procedural content generation, i.e., the testing of newly generated game con tent by playing through that content with an agent. For example, in a program generating levels for the platform game Super Mario Bros (Nintendo, 1985), the levels can be tested by allowing a trained agent to play them; those that the agent cannot complete can be discarded [335]. Browne's Ludi system, which generates complete board games, evaluates these games through simulated playthrough and uses learning algorithms to adapt the strategy to each game [74].

• Playing to Win → Generate Content (Assisted): Just as with autonomous pro cedural content generation, many tools for AI-assisted game design rely on being able to simulate playthroughs of some aspect of the game. For instance, the Sen tient Sketchbook tool for level design uses simple simulations of game-playing agents to evaluate aspects of levels as they are being edited by a human de-

signer [379]. Another example is the automated playtesting framework named Restricted Play [295] which aims mostly at assisting designers on aspects of game balance during game design. A form of Restricted Play is featured in the Ludocore game engine [639].

6.2.1.2 Playing for the Experience (as a Player or as a Non-Player)

Research on AI that plays a game for a purpose other than winning is central to studies where playing the game well is not the primary research aim. AI can play the game as a player character attempting to maximize the believability value of play as, for instance, in [719, 619, 96]. It can alternatively play the game in a role of an NPC for the same purpose [268]. Work under this research subarea involves the study of believability, interestingness or playing experience in games and the inves tigations of mechanisms for the construction of agent architectures that appear to have e.g., believable or human-like characteristics. The approaches for developing such architectures can be either top-down behavior authoring (such as the FAtiMA model used in My Dream Theatre [100] and the Mind Module model [191] used in The Pataphysic Institute) or bottom-up attempting to imitate believable gameplay from human players such as the early work of Thurau et al. in Quake II (Activision, 1997) bots [696], the human imitation attempts in Super Mario Bros (Nintendo, 1985) [511], the Unreal Tournament 2004 (Epic Games, 2004) believable bots of Schrum et al. [603] and the crowdsourcing studies of the Restaurant game [508].

Evidently, commercial games have for long benefited from agent believability re search. Examples of this include popular games such as the Sims (Electronic Arts, 2000) series. The industry puts a strong emphasis on the design of believability in games as this contributes to more immersive game environments. The funding of believability research through game AI competitions such as the 2K BotPrize is one of the many clear indicators of the commercial value of agent believability.

Over the last few years there has been a growing academic (and commercial) interest in the establishment of competitions that can be used as assessment tools for agent believability [719]. Agent believability research has provided input and given substance to those game benchmarks. A number of game Turing competitions have been introduced to the benefit of agent believability research, including the 2K BotPrize on the Unreal Tournament 2004 (Epic Games, 2004) [647, 264] game and the Mario AI Championship: Turing test track [619] on the Super Mario Bros (Nintendo, 1985) game. Recently, the community saw AI agents passing the Turing test in the 2K BotPrize [603].

The study of AI that plays games not for winning, but for other purposes, affects research on three other game AI areas as illustrated in Fig. 6.4, whereas it is affected by four other game AI areas.

• Playing for the Experience → Model Players (Experience and Behavior): There is a direct link between player modeling and believable agents as research carried out for the modeling of human, human-like, and supposedly believable playing behavior can inform the construction of more appropriate models for

Fig. 6.4 Playing for the Experience: influence on (and from) other game AI research areas.

players. Examples include the imitation of human play styles in Super Mario Bros (Nintendo, 1985) [511] and Quake II (Activision, 1997) [696]. Though computational player modeling uses learning algorithms, it is only in some cases that it is the behavior of an NPC that is modeled. In particular, this is true when the in-game behavior of one or several players is modeled. This can be done using either reinforcement learning techniques, or supervised learning techniques such as backpropagation or decision trees. In either case, the intended outcome for the learning algorithm is not necessarily an NPC that plays as well as possible, but one that plays in the style of the modeled player [735, 511].

• Playing for the Experience → Generate Content (Autonomously): Believ able characters may contribute to better levels [96], more believable stories

[801, 401, 531] and, generally, better game representations [563]. A typical ex ample of the integration of characters in the narrative and the drive of the lat ter based on the former includes the FAtiMa agents in FearNot! [516] and My
Dream Theater [100]. Another example is the generation of Super Mario Bros (Nintendo, 1985) levels that maximize the believability of any Mario player [96].

6.2.2 Generate Content

As covered in detail in Chapter 4 AI can be used to design whole (or parts of) games in an autonomous or in an assisted fashion. This core game AI area includes the subareas of autonomous (procedural) content generation and assisted or mixed initiative (procedural) content generation. The interactions of these subareas with the remaining areas of game AI are covered in this section.

Fig. 6.5 Generate Content (Autonomously): influence on (and from) other game AI research areas.

6.2.2.1 Generate Content (Autonomously)

As stated in Chapter 4 procedural content generation has been included in limited roles in some commercial games since the early 1980s; however, recent years have seen an expansion of research on more controllable PCG for multiple types of game content [764], using techniques such as evolutionary search [720] and constraint solving [638]. The influence of PCG research beyond games is already evident in areas such as computational creativity [381] and interaction design (among others). There are several surveys of PCG available, including a recent book [616] and vision paper [702], as well as surveys of frameworks [783], sub-areas of PCG [554, 732] and methods [720, 638].

Autonomous content generation is one of the areas of recent academic research on AI in games which bears most promise for incorporation into commercial games. A number of recent games have been based heavily on PCG, including independent ("indie") game production successes such as Spelunky (Mossmouth, 2009) and Minecraft (Mojang, 2011), and mainstream AAA games such as Diablo III (Blizzard Entertainment, 2012) and Civilization V (2K Games, 2010). A notable example, as mentioned in Chapter 4 is No Man's Sky (Hello Games, 2016) with its quintillion different procedurally generated planets. Some games heavily based on PCG and developed by researchers have been released as commercial games on plat forms such as Steam and Facebook; two good examples of this are Petalz [565, 566] and Galactic Arms Race [249].

Figure 6.5 depicts the three (and five) areas that are influenced by (and influence) autonomous PCG.

• Generate Content (Autonomously) → Play to Win: If an agent is trained to perform well in only a single game environment, it is easy to overspecialize the

training and arrive at a policy/behavior that will not generalize to other levels. Therefore, it is important to have a large number of environments available for training. PCG can help with this, potentially providing an infinite supply of test environments. For example, when training players for the Mario AI Champi onship it is common practice to test each agent on a large set of freshly generated levels, to avoid overtraining [322]. There has also been research on adapting NPC behavior specifically to generated content [332]. Finally, one approach to artificial general intelligence is to train agents to be good at playing games in general, and test them on a large variety of games drawn from some genre or dis tribution. To avoid overfitting, this requires games to be generated automatically, a form of PCG [598]. The generation of new environments is very important for NPC behavior learning, and this extends to benchmarks that measure some aspect of NPC behavior. Apart from the Mario AI Championship, competitions such as the Simulated Car Racing Championship use freshly generated tracks, unseen by the participants, to test submitted controllers [102]. But there is also scope for benchmarks and competitions focused on measuring the capabilities of PCG systems themselves, such as the Level Generation track of the Mario AI Championship [620].

◦ Generate Content (Autonomously) → Play for the Experience: Research on autonomous PCG naturally influences research on agent (PC or NPC) control for believability, interestingness or other aims aside from winning given that these agents are performing in a particular environment and under a specific game context. This influence is still in its infancy and the only study we can point the reader to is the one by Camilleri et al. [96] where the impact of level design on player character believability is examined in Super Mario Bros (Nintendo, 1985). Further, research on interactive narrative benefits from and influences the use of believable agents that interact with the player and are interwoven in the story plot. The narrative can yield more (or less) believability to agents and thus the relationship between the behavior of the agents and the emergent story is strong [801, 401, 531]. In that sense, the computational narrative of a game may define the arena for believable agent design.

• Generate Content (Autonomously) → Generate Content (Assisted): As con tent design is a central part of game design, many AI-assisted design tools incor porate some form of assisted content design. Examples include Tanagra, which helps designers create complete platform game levels which ensure playabil ity through the use of constraint solvers [641], and SketchaWorld [634]. An other example is Sentient Sketchbook, which assists humans in designing strategy game levels through giving immediate feedback on properties of levels and au tonomously suggesting modifications [379].

6.2.2.2 Generate Content (Assisted)

Assisted content generation refers to the development of AI-powered tools that sup port the game design and development process. This is perhaps the AI research area

Fig. 6.6 Generate Content (Assisted): influence on (and from) other game AI research areas.

which is most promising for the development of better games [764]. In particu lar, AI can assist in the creation of game content varying from levels and maps to game mechanics and narratives. The impact of AI-enabled authoring tools on design and development influences the study of AI that plays games for believability, in terestingness or player experience, and research in autonomous procedural content generation (see Fig. 6.6). AI-assisted game design tools range from those designed to assist with generation of complete game rulesets such as MetaGame [522] or RuLearn [699] to those focused on more specific domains such as strategy game levels [379], platform game levels [642], horror games [394] or physics-based puz zles [613]. It is worth noting that AI tools have been used extensively for supporting design and commercial game development. Examples such as the SpeedTree (Interactive Data Visualization Inc., 2013) generator for trees and other plants [287] have seen uses in several game productions. The mixed-initiative PCG tools mentioned above have a great potential in the near future as most of these are already tested on com mercial games or developed with game industrial partners. Furthermore, there are tools designed for interactive modeling and analysis of game rules and mechanics, which are not focused on generating complete games but on prototyping and un derstanding aspects of complex games; such systems could be applied to existing commercial games [639].

◦ Generate Content (Assisted) → Play for the Experience: Authoring tools in forms of open-world sandboxes could potentially be used for the creation of more believable behaviors. While this is largely still an unexplored area of research and development, notable attempts include the NERO game AI platform where players can train game agents for efficient and believable first-person shooter bot behaviors [654]. An open version of this platform focused on crowdsourcing behaviors has been released recently [327]. A similar line of research is the gen-

eration of Super Mario Bros (Nintendo, 1985) players by means of interactive evolution [648].

• Generate Content (Assisted) → Generate Content (Autonomously): Re search on methods of mixed-initiative co-creation [774] and design can feed

input to and spur discussion on central topics in procedural content genera tion. Given the importance of content design in the development process as a whole, any form of mixed-initiative AI assistance in the generation process can support and augment procedural content generation. Notable examples of mixed-initiative PCG include the Tanagra platform game level design AI assis tant [641], and the SketchaWorld [634], the Sentient World [380], the Sentient Sketchbook [379, 774] and the Sonancia [394] systems which generate game maps and worlds in a mixed-initiative design fashion following different ap proaches and levels of human computation. Further, tools can assist the author ing of narrative in games. In particular, drama management tools have long been investigated within the game AI community. An academic example is ABL which has allowed the authoring of narrative in Fac¸ade [441]. Among the few available and well-functional authoring tools the most notable is the Versu [197] storytelling system which was used in the game Blood & Laurels (Emily Short, 2014) and the Inform 7 [480] software package that led to the design of Mystery House Possessed (Emily Short, 2005). More story generation tools as such can be found at the http://storygen.org/ repository, by Chris Martens and Rogelio E. Cardona-Rivera.

6.2.3 Model Players

As already explored in Chapter 5, modeling players involves the subtasks of mod eling their behavior or their experience. Given the interwoven nature of these two tasks we present their influences to (and from) other game AI areas under one com mon section. In player modeling [782, 636], computational models are created for detecting how the player perceives and reacts to gameplay. As stated in Chapter 5 such models are often built using machine learning methods where data con sisting of some aspect of the game or player-game interaction is associated with labels derived from some assessment of player experience, gathered for example from questionnaires [781]. However, the area of player modeling is also concerned with structuring observed player behavior even when no correlates to experience are available—e.g., for identifying player types or predicting player behavior.

Research and development in player modeling can inform attempts for player ex perience in commercial-standard games. Player experience detection methods and algorithms can advance the study of user experience in commercial games. In addi tion, the appropriateness of sensor technology, the technical plausibility of biofeed back sensors, and the suitability of various modalities of human input can inform in dustrial developments. Quantitative testing via game metrics—varying from behav ioral data mining to in-depth low scale studies—is also improved [764, 178, 186].

Fig. 6.7 Model Players: influence on (and from) other game AI research areas.

By now, a considerable number of academic studies use directly datasets from com mercial games to induce models of players that could inform further development of the game. For example, we refer the reader to the experiments in clustering players of Tomb Raider: Underworld (Square Enix, 2008) into archetypes [176] and pre dicting their late-game performance based on early-game behavior [414]. Examples of player modeling components within high-profile commercial games include the arousal-driven appearance of NPCs in Left 4 Dead 2 (Valve Corporation, 2009), the fearful combat skills of the opponent NPCs in F.E.A.R. (Monolith, 2005), and the avatars' emotion expression in the Sims series (Maxis, 2000) and Black and White (Lionhead Studios, 2001). A notable example of a game that is based on player expe rience modeling is Nevermind (Flying Mollusk, 2016); the game adapts its content based on the stress of the player, which is manifested via a number of physiological sensors.

Player modeling is considered to be one of the core non-traditional uses of AI in games [764] and affects research in AI-assisted game design, believable agents, computational narrative and procedural content generation (see Fig. 6.7).

• Model Players → Play for the Experience: Player models can inform and up date believable agent architectures. Models of behavioral, affective and cognitive

aspects of gameplay can improve the human-likeness and believability of any agent controller—whether it is ad-hoc designed or built on data derived from gameplay. While the link between player modeling and believable agent design is obvious and direct, research efforts towards this integration within games are still sparse. However, the few efforts made on the imitation of human game play ing for the construction of believable architectures have resulted in successful outcomes. For example, human behavior imitation in platform [511] and racing games [735, 307] has provided human-like and believable agents while similar approaches for developing Unreal Tournament 2004 (Epic Games, 2004) bots (e.g., in [328]) recently managed to pass the Turing test in the 2K BotPrize com-
6.3. The Road Ahead 275
petition. Notably, one of the two agents that passed the Turing test in 2K BotPrize managed to do so by imitating (mirroring) aspects of human play [535]. A line of work that stands in between player modeling and playing games for the ex perience is the study on procedural personas [268, 267, 269]. As introduced in Chapter 5 procedural personas are NPCs that are trained to imitate realistically the decision making process of humans during play. Their study both influences our understanding about the internal (cognitive) processes of playing behavior and advances our knowledge on how to build believable characters in games.

• Model Players → Generate Content (Autonomously): There is an obvious link between computational models of players and PCG as player models can drive the generation of new personalized content for the player. The experience driven role of PCG [783], as covered in Chapter 4, views game content as an indirect building block of a player's affective, cognitive and behavioral state and proposes adaptive mechanisms for synthesizing personalized game experiences. The "core loop" of an experience-driven PCG solution involves learning a model that can predict player experience, and then using this model as part of an evalua tion function for evolving (or otherwise optimizing) game content; game content is evaluated based on how well it elicits a particular player experience, accord ing to the model. Examples of PCG that are driven by player models include the generation of game rules [716], camera profiles [780, 85] and platform game levels [617]. Most work that goes under the label "game adaptation" can be said to implement the experience-driven architecture; this includes work on adapting the game content to the player using reinforcement learning [28] or semantic constraint solving [398] rather than evolution. Player models may also inform the generation of computational narrative. Predictive models of playing experi ence can drive the generation of individualized scenarios in a game. Examples of the coupling between player modeling and computational narrative include the affect-driven narrative systems met in Fac¸ade [441] and FearNot! [26], and the affect-centered game narratives such as the one of Final Fantasy VII (Square, 1997).

◦ Model Players → Generate Content (Assisted): User models can enhance au thoring tools that, in turn, can assist the design process. The research area that bridges user modeling and AI-assisted design is in its infancy and only a few ex ample studies can be identified. Indicatively, designer models [378] have been employed to personalize mixed-initiative design processes [774, 377, 379]. Such models may drive the procedural generation of designer-tailored content.

6.3 The Road Ahead

This chapter has initially identified the currently most active areas and subareas within game AI and placed them on three holistic frameworks: an AI method map ping, a game stakeholder (end user) taxonomy and the player-game interaction loop. This analysis revealed dominant AI algorithms within particular areas as well as
276 Chapter 6. Game AI Panorama
room for exploration of new methods within areas. In addition, it revealed the dis similar impact of different areas on different end users such as the AI researcher and the designer and, finally, outlined the influence of the different game AI areas on the player, the game and their interaction. From the high-level analysis of the game AI

field we moved on to the detailed analysis of the game AI areas that compose it and thoroughly surveyed the meaningful interconnections between the different areas. The total number of strong and weak influences is rather small compared to all possible interconnections between the areas, which clearly signals the research ca pacity of the game AI field for further explorations. We can distinguish a number of connections which are currently very active, meaning that much work currently goes on in one area that draws on work in another area. Here we see, for example, the connection between AI that plays to win in a general fashion in conjunction with the use of tree search algorithms: the MCTS algorithm was invented in the context of board game-playing, proved to be really useful in the general game playing compe tition, and is being investigated for use in games as different as StarCraft (Blizzard Entertainment, 1998) and Super Mario Bros (Nintendo, 1985). Improvements and modifications to the algorithm have been flowing back and forth between the vari ous areas. Another indicative connection that is alive and strong is between player modeling and procedural content generation, where it is now common for newly devised PCG algorithms and experimental studies to include player behavioral or player experience models.

One can also study the currently strong areas by trying to cluster the trending topics in recent iterations of the IEEE CIG and AIIDE conferences. Such studies always include some form of selection bias, as papers can usually be counted into more than one area (e.g., depending on if you group by method or domain), but if you start from the session groupings made by the program chairs of each conference you achieve at least some inter-subjective validity. According to such a clustering, the most active topics over the last few years have been player (or emotion) model ing, game analytics, general game AI, real-time strategy game playing—especially StarCraft (Blizzard Entertainment, 1998)—and PCG (in general). Another perspec tive of the trend in game AI research is the varying percentage of studies on NPC (or game agent) behavior learning over other uses of AI in games at the two key conferences in the field (IEEE CIG and AIIDE). Our preliminary calculations sug gest that while, initially, AI was mainly applied for NPC control and for playing board/card games well—more than 75% of CIG and AIIDE papers were linked to NPC behavior and agent game playing in 2005—that trend has drastically changed as entirely new (non-traditional) uses of AI became more common over the years—e.g., roughly 52% of the papers in CIG and AIIDE in 2011 did not involve game agents and NPC behavior. These facts indicate a shift in the use of AI in and for games towards multiple non-traditional applications—which tend to be traditional by now—for the development of better games [764].

But it is maybe even more interesting to look at all those connections that are unexploited or underexploited or potentially strong. For example, player modeling is potentially very important in the development of AI that controls believable, in teresting or curious agents, but this has not been explored in enough depth yet; the

same holds for the application of user (or else, designer) modeling principles to wards the personalization of AI-assisted game design. Believable agents have, in turn, not been used enough in content generation (either autonomous or assisted).

A grand vision of game AI for the years to come is to let it identify its own role within game design and development as it sees fit. In the last chapter of this book we discuss frontier research topics as such and identify unexplored roles of AI in games.

6.4 Summary

We hope that with this chapter of this book, we have been able to give our readers a sense of how this—by now rather large and multifaceted—research field hangs together, and what could be done to integrate it further. We realize that this is only our view of its dynamics and interconnections, and that there are (or could be) many competing views. We look forward to seeing those in upcoming studies in the field.

Finally, it is important to note that it would have been impossible to provide a complete survey of all the areas as, first, the game AI field is growing rapidly and,

second, it is not the core objective of the book. This means that the bibliography is indicative rather than exhaustive and serves as a general guideline for the reader. The website of the book, instead of the book per se, will be kept up to date regarding important new readings for each area.

The next and final chapter of the book is dedicated to a few long-standing, yet rather unexplored, research frontiers of game AI. We believe that any advances made in these directions will lead to scientific breakthroughs not merely within game AI but largely in both games (their design, technology and analysis) and AI per se.

Chapter 7
Frontiers of Game AI Research

In this final chapter of the book we discuss a number of long-term visionary goals of game AI, putting an emphasis on the generality of AI and the extensibility of its roles within games. In particular, in Section 7.1 we discuss our vision for general behavior for each one of the three main uses of AI in games. Play needs to be come general; generators are required to have general generative capacities across games, content types, designers and players; models of players also need to show case general modeling abilities. In Section 7.2 we also discuss roles of AI that are still unexplored and certainly worth investigating in the future. The book ends with a discussion dedicated to general ethical considerations of game AI (Section 7.3).

7.1 General General Game AI

As evidenced from the large volume of studies the game AI research area has been supported by an active and healthy research community for more than a decade—at least since the start of the IEEE CIG and the AIIDE conference series in 2005. Before then, research had been conducted on AI in board games since the dawn of automatic computing. Initially, most of the work published at IEEE CIG or AI IDE was concerned with learning to play a particular game as well as possible, or using search/planning algorithms to play a game as well as possible without learn ing. Gradually, a number of new applications for AI in games and for games in AI have come to complement the original focus on AI for playing games [764]. Papers on procedural content generation, player modeling, game data mining, human-like playing behavior, automatic game testing and so on have become commonplace within the community. As we saw in the previous chapter there is also a recogni tion that all these research endeavors depend on each other [785]. However, almost all research projects in the game AI field are very specific. Most published papers describe a particular method—or a comparison of two or more methods—for per forming a single task (playing, modeling, generating, etc.) in a single game. This is problematic in several ways, both for the scientific value and for the practical appli-

cability of the methods developed and studies made in the field. If an AI approach is only tested on a single task for a single game, how can we argue that is an ad vance in the scientific study of artificial intelligence? And how can we argue that it is a useful method for a game designer or developer, who is likely working on a completely different game than the one the method was tested on?

As discussed in several parts of this book general game playing is an area that has already been studied extensively and constitutes one of the key areas of game AI [785]. The focus of generality solely on play, however, is very narrow as the possible roles of AI and general intelligence in games are many, including game design, content design and player experience design. The richness of the cognitive skills and affective processes required to successfully complete these tasks has so far been largely ignored by game AI research. We thus argue, that while the focus on general AI needs to be retained, research on general game AI needs to expand beyond mere game playing. The new scope for general general game AI beyond game-playing broadens the applicability and capacity of AI algorithms and our understanding of intelligence as tested in a creative domain that interweaves problem solving, art, and engineering.

For general game AI to eventually be truly general, we argue that we need to

extend the generality of general game playing to all other ways in which AI is (or can be) applied to games. More specifically we argue that the field should move towards methods, systems and studies that incorporate three different types of generality:
1. Game generality. We should develop AI methods that work with not just one game, but with any game (within a given range) that the method is applied to.
2. Task generality. We should develop methods that can do not only one task (play ing, modeling, testing, etc) but a number of different, related tasks.
3. User/designer/player generality. We should develop methods that can model, respond to and/or reproduce the very large variability among humans in design style, playing style, preferences and abilities.
We further argue that all of this generality can be embodied into the concept of general game design, which can be thought of as a final frontier of AI research within games. Further details about the notion of general general game AI can be found in the vision paper we co-authored about this frontier research area [718]. It is important to note that we are not arguing that more focused investigations into methods for single tasks in single games are useless; these are often important as proofs-of-concept or industrial applications and they will continue to be important in the future, but there will be an increasing need to validate such case studies in a more general context. We are also not envisioning that everyone will suddenly start working on general methods. Rather, we are positing generalizations as a long-term goal for our entire research community. Finally, the general systems of game AI that we envision ought to have a real-world use. There is a risk that by making systems too general we might end up not finding applications of these general systems to any specific real-world problem. Thus, the system's applicability (or usefulness) sets our core constraint towards this vision of general game AI. More specifically, we envi-

sion general general game AI systems that are nevertheless integrated successfully within specific game platforms or game engines.

7.1.1 General Play

The problem of playing games is the one that has been most generalized so far. There already exist at least three serious benchmarks or competitions attempting to pose the problem of playing games in general, each in its own imperfect way. The Gen eral Game Playing Competition, often abbreviated GGP [223], the Arcade Learning Environment [40] and the General Video Game AI competition [528]; all three have been discussed in various places in this book. The results from these competitions so far indicate that general purpose search and learning algorithms by far outper form more domain-specific solutions and "clever hacks". Somewhat simplified, we can say that variations of Monte Carlo tree search perform best on GVGAI and GGP [202], and for ALE (where no forward model is available so learning a policy for each game is necessary) reinforcement learning with deep networks [464] and search-based iterative width [389, 301, 390] perform best. This is a very marked difference from the results of the game-specific competitions, indicating the lack of domain-independent solutions.
While these are each laudable initiatives and currently the focus of much re search, in the future we will need to expand the scope of these competitions and benchmarks considerably, including expanding the range of games available to play and the conditions under which gameplay happens. We need game playing bench marks and competitions capable of expressing any kind of game, including puzzle games, 2D arcade games, text adventures, 3D action-adventures and so on; this is the best way to test general AI capacities and reasoning skills. We also need a number of different ways of interfacing with these games—there is room for both benchmarks that give agents no information beyond the raw screen data but give them hours to learn how to play the game, and those that give agents access to a forward model and perhaps the game code itself, but expect them to play any game presented to them with no time to learn. These different modes test different AI capabilities and tend to privilege different types of algorithms. It is worth noting that the GVGAI competition is currently expanding to different types of playing modes, and has a

long-term goal to include many more types of games [527].
We also need to differentiate away from just measuring how to play games op timally. In the past, several competitions have focused on agents that play games
in a human-like manner; these competitions have been organized similarly to the classic Turing test [263, 619]. Playing games in a human-like manner is important for a number of reasons, such as being able to test levels and other game content as part of search-based generation, and to demonstrate new content to players. So far, the question of how to play games in a human-like manner in general is mostly unexplored; some preliminary work is reported in [337]. Making progress here will likely involve modeling how humans play games in general, including characteris-

tics such as short-term memory, reaction time and perceptual capabilities, and then translating these characteristics to playing style in individual games.

7.1.2 General Game Generation and Orchestration

The study of PCG [616] for the design of game levels has reached a certain matu rity and is, by far, the most popular domain for the application of PCG algorithms
and approaches (e.g., see [720, 785, 783] among many). What is common in most of the content generation studies covered in this book, however, is their specificity and strong dependency of the representation chosen on the game genre examined. For the Mario AI Framework, for instance, the focus on a single level generation problem has been very much a mixed blessing: it has allowed for the proliferation and simple comparison of multiple approaches to solving the same problem, but has also led to a clear overfitting of methods. Even though some limited generalization is expected within game levels of the same genre, the level generators that have been explored so far clearly do not have the capacity of general level design. We argue that there needs to be a shift in how level generation is viewed. The obvious change of perspective is to create general level generators—level generators with general intelligence that can generate levels for any game (within a specified range). That would mean that levels are generated successfully across game genres and players and that the output of the generation process is meaningful and playable as well as entertaining for the player. Further, a general level generator should be able to coor dinate the generative process with the other computational game designers who are responsible for the other parts of the game design.
To achieve general level design intelligence algorithms are required to capture as much of the level design space as possible at different representation resolutions. We can think of representation learning approaches such as deep autoencoders [739] capturing core elements of the level design space and fusing various game genres within a sole representation—as already showcased by a few methods, such as the Deep Learning Novelty Explorer [373]. The first attempt to create a benchmark for general level generation has recently been launched in the form of the Level Genera tion Track of the GVGAI competition. In this competition track, competitors submit
level generators capable of generating levels for unseen games. The generators are then supplied with the description of several games, and produce levels which are judged by human judges [338]. Initial results suggest that constructing competent level generators that can produce levels for any game is much more challenging than constructing competent level generators for a single game. A related effort is the Video Game Level Corpus [669] which aims to provide a set of game levels across multiple games and genres which can be used for training level generators for data-driven procedural content generation.
While level generation, as discussed above, is one of the main examples of proce dural content generation, there are many other aspects (or facets) of games that can
be generated. These include visuals, such as textures and images; narrative, such

as quests and backstories; audio, such as sound effects and music; and of course all kinds of things that go into game levels, such as items, weapons, enemies and personalities [381, 616]. However, an even greater challenge is the generation of complete games, including some or all of these facets together with the rules of the

game. While, as covered in Chapter 4, there have been several attempts to gener ate games (including their rules) we are not aware of any approach to generating games that tries to generate more than two of the facets of games listed above. We are also not aware of any game generation system that even tries to generate games of more than one genre. Multi-faceted generation systems like Sonancia [394, 395] co-generate horror game levels with corresponding soundscapes but do not cater to the generation of rules. It is clear that the very domain-limited and facet-limited as pects of current game generation systems result from intentionally limiting design choices in order to make the very difficult problem of generating complete games tractable. Yet, in order to move beyond what could be argued to be toy domains and start to fulfill the promise of game generation, we need systems that can generate multiple facets of games at the same time, and that can generate games of different kinds.

This process has been defined as facet (domain) orchestration in games [371, 324]. Orchestration refers to the process of harmonizing game generation. Evi dently, orchestration is a necessary process when we consider the output of two or more content type generators—such as visuals and audio—for the generation of a complete game. Drawing inspiration from music, orchestration may vary from a top-down, conductor-driven process to a bottom-up, free-from generation process [371]. A few years ago, something very much like general game generation and or chestration was outlined as the challenges of "multi-content, multi-domain PCG" and "generating complete games" [702]. It is interesting to note that there has not seemingly been any attempt to create more general game generators since then, per haps due to the complexity of the task. A recent study by Karavolos et al. [324] moves towards the orchestration direction as it fuses level and game design param eters in first-person shooters via deep convolutional neural networks. The trained networks can be used to generate balanced games. Currently the only genre for which generators have been built that can generate high-quality (complete) games is abstract board games. Once more genres have been "conquered", we hope that the task of building more general level generators can begin.

Linked to the tasks of orchestration and general game generation there are im portant questions with respect to the creative capacity of the generation process that remain largely unanswered. For example, how creative can a generator be and how can we assess it? Is it, for instance, deemed to have appreciation, skill, and imagina tion [130]? When it comes to the evaluation of the creative capacity of current PCG algorithms a case can be made that most of them possess only skill. Does the creator manage to explore novel combinations within a constrained space, thereby resulting in exploratory game design creativity [53]; or, is on the other hand trying to break existing boundaries and constraints within game design to come up with entirely new designs, demonstrating transformational creativity [53]? If used in a mixed initiative fashion, does it enhance the designer's creativity by boosting the possi-

bility space for her? Arguably, the appropriateness of various evaluation methods for autonomous PCG creation or mixed-initiative co-creation [774] remains largely unexplored within both human and computational creativity research.

7.1.3 General Game Affective Loop

It stands to reason that general intelligence implies (and is tightly coupled with) general emotional intelligence [443]. The ability to recognize human behavior and emotion is a complex yet critical task for human communication that acts as a fa cilitator of general intelligence [157]. Throughout evolution, we have developed particular forms of advanced cognitive, emotive and social skills to address this challenge. Beyond these skills, we also have the capacity to detect affective patterns across people with different moods, cultural backgrounds and personalities. This generalization ability also extends, to a degree, across contexts and social settings. Despite their importance, the characteristics of social intelligence have not yet been transferred to AI in the form of general emotive, cognitive or behavioral models. While research in affective computing [530] has reached important milestones such

as the capacity for real-time emotion recognition [794]—which can be faster than humans under particular conditions—all key findings suggest that any success of affective computing is heavily dependent on the domain, the task at hand, and the context in general. This specificity limitation is particularly evident in the domain of games [781] as most work in modeling player experience focuses on particu lar games, under well-controlled conditions with particular, small sets of players (see [783, 609, 610, 435] among many). In this section we identify and discuss two core unexplored and interwoven aspects of modeling players that are both important and necessary steps towards the long-term aim of game AI to realize truly adaptive games. The first aspect is the closure of the affective loop in games; the second as pect is the construction of general models capable of capturing experience across players and games.

As stated at the start of this book, affective computing is best realized within games in what we name the game affective loop. While the phases of emotion elic itation, affect modeling and affect expression have offered some robust solutions by now, the very loop of affective-based interaction has not been closed yet. Aside from a few studies demonstrating some affect-enabled adaptation of the game [772, 617] the area remains largely unexplored. It is not only the complexity of modeling play ers and their experience that is the main hurdle against any advancement. What is also far from trivial is the appropriate and meaningful integration of any of these models in a game. The questions of how often the system should adapt, what it should alter and by what degree are not easy to answer. As most of the questions are still open to the research community the only way to move forward is to do more research in adaptive games involving affective aspects of the experience. Existing commercial-standard games that already realize the affective loop such as Never mind (Flying Mollusk, 2016) are the ambassadors for further work in this area.

Once the game affective loop is successfully realized within particular games the next goal for game AI is the generality of affect-based interaction across games. The game affective loop should not only be operational; it should ideally be general too. For AI in games to be general beyond game-playing it needs to be able to recog nize general emotional and cognitive-behavioral patterns. This is essentially AI that can detect context-free emotive and cognitive reactions and expressions across con texts and builds general computational models of human behavior and experience which are grounded in a general gold standard of human behavior. So far we have only seen a few proof-of-concept studies in this direction. Early work within the game AI field focused on the ad-hoc design of general metrics of player interest that were tested across different prey-predator games [768, 767]. In other, more recent, studies predictors of player experience were tested for their ability to capture player experience across dissimilar games [431, 612, 97]. Another study on deep multi modal fusion can be seen as an embryo for further research in this direction [435], in which various modalities of player input such as player metrics, skin conductance and heart activity have been fused using stacked autoencoders. Discovering entirely new representations of player behavior and emotive manifestations across games, modalities of data, and player types is a first step towards achieving general player modeling. Such representations can, in turn, be used as the basis for approximating the ground truth of user experience in games.

7.2 AI in Other Roles in Games

The structure of this book reflects our belief that playing games, generating content and modeling players are the central applications of AI methods in games. How ever, there are many variants and use cases of game playing, player modeling or content generation that we have not had time to explore properly in the book, and which in some cases not have been explored in the literature at all. Further, there are some applications of AI in games that cannot be really classified as special cases of our “big three” AI applications in games, despite our best efforts. This section briefly sketches some of these applications, some of which may be important future research directions.

Playtesting: One of the many use cases for AI for playing games is to test the games. Testing games for bugs, balancing player experience and behavior, and other issues is important in game development, and one of the areas where game devel opers are looking for AI assistance. While playtesting is one of the AI capabilities within many of the mixed-initiative tools discussed in Chapter 4, there has also been work on AI-based playtesting outside of that context. For example, Denzinger et al. evolved action sequences to find exploits in sports games, with discouragingly good results [165]. For the particular case of finding bugs and exploits in games, one of the research challenges is to find a good and representative coverage of problems, so as to deliver an accurate picture to the development team of how many problems

there are and how easy they are to run into, and allow prioritization of which prob lems to fix.
Critiquing Games: Can AI methods meaningfully judge and critique games? Game criticism is hard and generally depends on deep understanding of not only games but also the surrounding cultural context. Still, there might be automated metrics that are useful for game criticism, and can provide information to help reviewers, game cu rators and others in selecting which games to consider for reviewing for inclusion in app stores. The ANGELINA game generation system is one of the few examples towards this direction [136] in which AI generates the overview of the game to be played.
Hyper-formalist Game Studies: AI methods can be applied to corpora of games in order to understand distributions of game characteristics. For example, decision trees can be used to visualize patterns of resource systems in games [312]. There are likely many other ways of using game AI for game studies that are still to be discovered.
Game Directing: The outstanding feature of Left 4 Dead (Valve Corporation, 2008) was its AI director, which adjusted the onslaught of zombies to provide a dramatic curve of challenge for players. While simple and literally one-dimensional (only a single dimension of player experience was tracked), the AI director proved highly effective. There is much room for creating more sophisticated AI directors; the experience-driven PCG framework [783] is one potential way within which to work towards this.
Creative Inspiration: While designing a complete game that actually works likely requires a very complex generator, it can be simpler to generate an idea for new games, that are then designed by humans. Creative ideation tools range from sim ple word-recombination-based tools implemented as card games or Twitter bots, to elaborate computational creativity systems such as the What If-Machine [391].
Chat Monitoring: In-game chats are important in many online multi-player games, as they allow people to collaborate within games and socialize through them. Un fortunately, such chats can also be used to threaten or abuse other players. Given the very large volume of chat messages sent through a successful online game, it becomes impossible for the game developers to curate chats manually. In the efforts to combat toxic behavior, some game developers have therefore turned to machine learning. Notably, Riot Games have trained algorithms to recognize and remove toxic behavior in the MOBA League of Legends (Riot Games, 2009) [413]. Even worse, sexual predation can be seen in some games, where pedophiles use game chats to reach children; there have been attempts to use machine learning to detect sexual predators in game chats too [241].
AI-Based Game Design: Throughout most of the book, we have assumed the ex-

istence of a game or at least a game design, and discussed how AI can be used to play that game, generate content for it or model its players. However, one could also start from some AI method or capability and try to design a game that builds on that method or capability. This could be seen as an opportunity to showcase AI methods in the context of games, but it could also be seen as a way of advancing game de sign. Most classic game designs originate in an era where there were few effective AI algorithms, there was little knowledge among game designers about those AI

algorithms that existed, and CPU and memory capacity of home computers was too limited to allow anything beyond simple heuristic AI and some best-first search to be used. One could even say that many classic video game designs are an attempt to design around the lack of AI—for example, the lack of good dialog AI for NPCs led to the use of dialog trees, the lack of AIs that could play FPS games believably and competently led to FPS game designs where most enemies are only on-screen for a few seconds so that you do not notice their lack of smarts, and the lack of level generation methods that guaranteed balance and playability led to game de signs where levels did not need to be completable. The persistence of such design patterns may be responsible for the relatively low utilization of interesting AI meth ods within commercial game development. By starting with the AI and designing a game around it, new design patterns that actually exploit some of the recent AI advances can be found.

Several games have been developed within the game AI research community specifically to showcase AI capabilities, some of which have been discussed in this book. Three of the more prominent examples are based on Stanley et al.'s work on neuroevolution and the NEAT algorithm: NERO, which is an RTS-like game where the player trains an army through building a training environment rather than controlling it directly [654]; Galactic Arms Race, in which weapons con trolled through neural networks are indirectly collectively evolved by thousands of players [250, 249]; and Petalz, which is a Facebook game about collecting flow ers based on a similar idea of selection-based collective neuroevolution [565, 566].

Other games have been built to demonstrate various adaptation mechanisms, such as Infinite Tower Defense [25] and Maze-Ball [780]. Within interactive narrative it is relatively common to build games that showcase specific theories and methods; a fa mous example is Fac¸ade [441] and another prominent example is Prom Week [447].

Treanor et al. have attempted to identify AI-based game design patterns, and found a diverse array of roles in which AI can be or has been used in games, and a number of avenues for future AI-based game design [724].

7.3 Ethical Considerations

Like all technologies, artificial intelligence, including game AI, can be used for many purposes, some of them nefarious. Perhaps even more importantly, technol ogy can have ethically negative or at least questionable effects even when there is no malicious intent. The ethical effects of using AI with and in games are not always

obvious, and the topic is not receiving the attention it should. This short section looks at some of the ways in which game AI intersects with ethical questions. For general AI research issues, ethics and values we refer the interested reader to the Asilomar AI Principles1 developed in conjunction with the 2017 Asilomar confer ence.

Player modeling is perhaps the part of game AI where the ethical questions are most direct, and perhaps most urgent. There is now a vigorous debate about the mass collection of data about us both by government entities (such as the US Na tional Security Agency or the United Kingdom's GCHQ) and private entities (such as Google, Amazon, Facebook and Microsoft) [64, 502]. With methodological ad vances in data mining, it is becoming possible to learn more and more about in dividual people from their digital traces, including inferring sensitive information and predicting behavior. Given that player modeling involves large-scale data col lection and mining, many of the same ethical challenges exist in player modeling as in the mining of data about humans in general. Mikkelsen et al. present an overview of ethical challenges for player modeling [458]. Below we give some examples of such challenges.

Privacy: It is becoming increasingly possible and even practicable to infer vari ous real-life traits and properties of people from their in-game behavior. This can be done without the consent or even knowledge of the subject, and some of the in formation can be of a private and sensitive nature. For example, Yee and colleagues investigated how player choices in World of Warcraft (Blizzard Entertainment, 2004) correlated with the personalities of players. They used data about players' characters

from the Armory database of World of Warcraft (Blizzard Entertainment, 2004) and correlated this information with personality tests administered to players; multiple strong correlations were found [788]. In a similar vein, a study investigated how players' life motives correlated with their Minecraft (Mojang, 2011) log files [101]. That research used the life motivation questionnaires of Steven Reiss, and found that players' self-reported life motives (independence, family, etc.) were expressed in a multitude of ways inside constructed Minecraft (Mojang, 2011) worlds. Using a very different type of game strong correlations have been found between play ing style in the first-person shooter Battlefield 3 (Electronic Arts, 2011) and player characteristics such as personality [687], age [686] and nationality [46]. It is en tirely plausible that similar methods could be used to infer sexual preferences, po litical views, health status and religious beliefs. Such information could be used by advertising networks to serve targeted ads, by criminals looking to blackmail the player, by insurance companies looking to differentiate premiums, or by malevolent political regimes for various forms of suppression. We do not know yet what can be predicted and with what accuracy, but it is imperative that more research be done on this within the publicly available literature; it is clear that this kind of research will also be carried out behind locked doors.

1 https://futureoflife.org/ai-principles/

Ownership of Data: Some player data can be used to recreate aspects of the player's behavior; this is the case for e.g., the Drivatars in Microsoft's Forza Motorsport series, and more generally for agents created according to the procedural persona concept [267]. It is currently not clear who owns this data, and if the game devel oper/publisher owns the data, what they can do with it. Will the game allow other people to play against a model of you, i.e., how you would have played the game? If so, can it identify you to other players as the origin of this data? Does it have to be faithful to the behavioral model of you, or can it add or distort aspects of your playing behavior?

Adaptation: Much of the research within game AI is concerned with adaptation of games, with the experience-driven PCG framework being the perhaps most com plete account on how to combine player modeling with procedural content gener ation to create personalized game experiences [783]. However, it is not clear that it is always a good thing to adapt games to players. The "filter bubble" is a con cept within discussion of social networks which refers to the phenomenon where collaborative filtering ensures that users are only provided with content that is al ready in line with their political, ethical, or aesthetic preferences, leading to a lack of healthy engagement with other perspectives. Excessive adaptation and personal ization might have a similar effect, where players are funneled into a narrow set of game experiences.

Stereotypes: Anytime we train a model using some dataset, we run the risk of reproducing stereotypes within that dataset. For example, it has been shown that word embeddings trained on standard datasets of the English language reproduce gender-based stereotypes [93]. The same effects could be present when modeling player preferences and behavior, and the model might learn to reproduce prejudiced conceptions regarding gender, race, etc. Such problems can be exacerbated or ame liorated by the tools made available to players for expressing themselves in-game. For example, Lim and Harrell have developed quantitative methods for measuring and addressing bias in character creation tools [386].

Censorship: Of course, it is entirely possible, and advisable, to use AI methods to promote ethical behavior and uphold ethical values. Earlier, Section 7.2 discussed the examples of AI for filtering player chats in online multi-player games, and for detecting sexual predators. While such technologies are generally welcome, there are important ethical considerations in how they should be deployed. For example, a model that has been trained to recognize hate speech might also react to normal in-game jargon; setting the right decision threshold might involve a delicate tradeoff between ensuring a welcoming game environment and not restricting communica tions unduly.

AI Beyond Games: Finally, a somewhat more far-fetched concern, but one we be lieve still merits discussion is the following. Games are frequently used to train and
t

(which is determined by the player).
166 Chapter 4. Generating Content
A cellular automaton contains a number of cells represented in any number of dimensions; most cellular automata, however, are either one-dimensional (vectors) or two-dimensional (matrices). Each cell can have a finite number of states; for in stance, the cell can be on or off. A set of cells surrounding each cell define its neigh borhood. The neighborhood defines which cells around a particular cell affect the cell's future state and its size can be represented by any integer number greater than 1. For one-dimensional cellular automata, for instance, the neighborhood is defined by the number of cells to the left or the right of the cell. For two-dimensional cellu lar automata, the two most common neighborhood types are the Moore and the von Neumann neighborhood. The former neighborhood type is a square consisting of the cells surrounding a cell, including those surrounding it diagonally; for example, a Moore neighborhood of size 1 contains the eight cells surrounding each cell. The latter neighborhood type, instead, forms a cross of cells which are centered on the cell considered. For example, a von Neumann neighborhood of size 1 consists of the four cells surrounding the cell, above, below, to the left and to the right.
At the beginning of an experiment (at time t = 0) we initialize the cells by as signing a state for each one of them. At each time step t we create a new generation of cells according to a rule or a mathematical function which specifies the new state of each cell given the current state of the cell and the states of the cells in its neigh borhood at time t −1. Normally, the rule for updating the state of the cells remains the same across all cells and time steps (i.e., it is static) and is applied to the whole grid.
Cellular automata have been used extensively in games for modeling environ mental systems like heat and fire, rain and fluid flow, pressure and explosions [209, 676] and in combination with influence maps for agent decision making [678, 677]. Another use for cellular automata has been for thermal and hydraulic erosion in procedural terrain generation [500]. Of particular interest for the pur poses of this section is the work of Johnson et al. [304] on the generation of infinite cave-like dungeons using cellular automata. The motivation in that study was to cre ate an infinite cave-crawling game, with environments stretching out endlessly and seamlessly in every direction. An additional design constraint was that the caves are supposed to look organic or eroded, rather than having straight edges and angles. No storage medium is large enough to store a truly endless cave, so the content must be generated at runtime, as players choose to explore new areas. The game does not scroll but instead presents the environment one screen at a time, which offers a time window of a few hundred milliseconds in which to create a new room every time the player exits a room.
The method introduced by Johnson et al. [304] used the following four parame ters to control the map generation process:
• a percentage of rock cells (inaccessible areas) at the beginning of the process;
• the number of CA generations (iterations);
• a neighborhood threshold value that defines a rock;
• the Moore neighborhood size.
4.3. How Could We Generate Content? 167
(a) A random map (b) A map generated with cellular automata

Fig. 4.6 Cave generation: Comparison between a CA and a randomly generated map. The CA pa rameters used are as follows: the CA runs for four generations; the size of the Moore neighborhood considered is 1; the threshold value for the CA rule is 5 (T = 5); and the percentage of rock cells at the beginning of the process is 50% (for both maps). Rock and wall cells are represented by white and red color respectively. Colored areas represent different tunnels (floor clusters). Images adapted from [304].

In the dungeon generation implementation presented in [304], each room is a 50 × 50 grid, where each cell can be in one of two states: empty or rock. Initially, the grid is empty. The generation of a single room is as follows.

1. The grid is "sprinkled" with rocks: for each cell, there is probability (e.g., 0.5) that it is turned into rock. This results in a relatively uniform distribution of rock cells.
2. A number of CA generations (iterations) are applied to this initial grid.
3. For each generation the following simple rule is applied to all cells: a cell turns into rock in the next iteration if at least T (e.g., 5) of its neighbors are rock, otherwise it will turn into free space.
4. For aesthetic reasons the rock cells that border on empty space are designated as "wall" cells, which are functionally rock cells but look different.

The aforementioned simple procedure generates a surprisingly lifelike cave room. Figure 4.6 shows a comparison between a random map (sprinkled with rocks) and the results of a few iterations of the cellular automaton. But while this process generates a single room, a game would normally require a number of connected rooms. A generated room might not have any openings in the confining rocks, and there is no guarantee that any exits align with entrances to the adjacent rooms. Therefore, whenever a room is generated, its immediate neighbors are also gen erated. If there is no connection between the largest empty spaces in the two rooms, a tunnel is drilled between those areas at the point where they are least separated. A few more iterations of the CA algorithm are then run on all nine neighboring rooms

Fig. 4.7 Cave generation: a 3 × 3 base grid map generated with CA. Rock and wall cells are represented by white and red color respectively; gray areas represent floor. Moore neighborhood size is 2, T is 13, number of CA iterations is 4, and the percentage of rock cells at the initialization phase is 50%. Image adapted from [304].

together, to smooth out any sharp edges. Figure 4.7 shows the result of this process, in the form of nine rooms that seamlessly connect. This generation process is ex tremely fast, and can generate all nine rooms in less than a millisecond on a modern computer. A similar approach to that of Johnson et al. is featured in the Galak-Z (17-bit, 2016) game for dungeon generation [9]. In that game cellular automata gen erate the individual rooms of levels and the rooms are tied together via a variation of a Hilbert curve, which is a continuous fractal space-filling curve [261]. Galak-Z (17-bit, 2016) shows an alternative way of combining CA with other methods for achieving the desired map generation result.

In summary, CA are very fast constructive methods that can be used effectively to generate certain kinds of content such as terrains and levels (as e.g., in [304]), but they can also be potentially used to generate other types of content. The greatest benefits a CA algorithm can offer to a game content generator is that it depends on a small number of parameters and that it is intuitive and relatively simple to grasp and implement. However, the algorithm's constructive nature is the main cause for

.3.5 Noise and Fractals

One class of algorithms that are very frequently used to generate heightmaps and textures are noise algorithms, many of which are fractal algorithms, meaning that they exhibit scale-invariant properties. Noise algorithms are usually fast and easy to use but lack in controllability.

Both textures and many aspects of terrains can fruitfully be represented as two dimensional matrices of real numbers. The width and height of the matrix map to the x and y dimensions of a rectangular surface. In the case of a texture, this is called

an intensity map, and the values of cells correspond directly to the brightness of the associated pixels. In the case of terrains, the value of each cell corresponds to the height of the terrain (over some baseline) at that point. This is called a heightmap. If the resolution with which the terrain is rendered is greater than the resolution of the heightmap, intermediate points on the ground can simply be interpolated between points that do have specified height values. Thus, using this common representation, any technique used to generate noise could also be used to generate terrains, and vice versa—though they might not be equally suitable.

It should be noted that in the case of terrains, other representations are possible and occasionally suitable or even necessary. For example, one could represent the terrain in three dimensions, by dividing the space up into voxels (cubes) and comput ing the three-dimensional voxel grid. An example is the popular open-world game Minecraft (Mojang, 2011), which uses unusually large voxels. Voxel grids allow structures that cannot be represented with heightmaps, such as caves and overhang ing cliffs, but they require a much larger amount of storage.

Fractals [180, 500] such as midpoint displacement algorithms [39] are in com mon use for real-time map generation. Midpoint displacement is a simple algorithm for generating two-dimensional landscapes (seen from the side) by repeatedly sub dividing a line. The procedure is as follows: start with a horizontal line. Find the midpoint of that line, and move the line up or down by a random amount, thus breaking the line in two. Now do the same thing for both of the resulting lines, and so on for as many steps as you need in order to reach sufficient resolution. Ev-

Fig. 4.8 The Midpoint Displacement algorithm visualized.

ery time you call the algorithm recursively, lower the range of the random number generator somewhat (see Fig. 4.8 for an example).

A useful and simple way of extending the midpoint displacement idea to two di mensions (and thus creating two-dimensional heightmaps which can be interpreted as three-dimensional landscapes) is the Diamond-Square algorithm (also known as “the cloud fractal” or “the plasma fractal” because of its frequent use for creating such effects) [210]. This algorithm uses a square 2D matrix with width and height $2n+1$. To run the algorithm you normally initialize the matrix by setting the values of all cells to 0, except the four corner values which are set to random values in some chosen range (e.g., $[-1,1]$). Then you perform the following steps:

1. Diamond step: Find the midpoint of the four corners, i.e., the most central cell in the matrix. Set the value of that cell to the average value of the cor ners. Add a random value to the middle cell.

2. Square step: Find the four cells in between the corners. Set each of those to the average value of the two corners surrounding it. Add a random value to each of these cells.

Call this method recursively for each of the four subsquares of the matrix, until you reach the resolution limit of the matrix ($3 * 3$ sub-squares). Every time you call the method, reduce the range of the random values somewhat. The process is illustrated in Fig. 4.9.

There are many more advanced methods for generating fractal noise, with differ ent properties. One of the most important is Perlin noise, which has some benefits over Diamond Square [529]. These algorithms are covered thoroughly in books that focus on texturing and modeling from a graphics perspective [180].

4.3.6 Machine Learning

An emerging direction in PCG research is to train generators on existing content, to be able to produce more content of the same type and style. This is inspired by the recent results in deep neural networks, where network architectures such as generative adversarial networks [232] and variational autoencoders [342] have attained good results in learning to produce images of e.g., bedrooms, cats or faces, and also by earlier results where both simpler learning mechanisms such as Markov chains and more complex architectures such as recurrent neural networks

have learned to produce text and music after training on some corpus.
While these kinds of generative methods based on machine learning work well for some types of content—most notably music and images—many types of game content pose additional challenges. In particular, a key difference between game content generation and procedural generation in many other domains is that most game content has strict structural constraints to ensure playability. These constraints differ from the structural constraints of text or music because of the need to play games in order to experience them. A level that structurally prevents players from finishing it is not a good level, even if it is visually attractive; a strategy game map with a strategy-breaking shortcut will not be played even if it has interesting fea tures; a game-breaking card in a collectible card game is merely a curiosity; and so on. Thus, the domain of game content generation poses different challenges from that of other generative domains. The same methods that can produce "mostly cor rect" images of bedrooms and horses, that might still have a few impossible angles or vestigial legs, are less suitable for generating mazes which must have an exit.
This is one of the reasons why machine learning-based approaches have so far only attained limited success in PCG for games. The other main reason is that for many types of game content, there simply isn't enough existing content to train on. This

(a) Initiate corner values
(b) Perform diamond step (c) Perform square step
(d) Perform diamond step (e) Perform square step
Fig. 4.9 The Diamond-Square algorithm visualized in five steps. Adapted from a figure by Christo pher Ewin, licensed under CC BY-SA 4.0.
is, however, an active research direction where much progress might be achieved in the next few years.

(a) n = 1
(b) n = 2
(c) n = 3
Fig. 4.10 Mario levels reconstructed by n-grams with n set to 1, 2, and 3, respectively.
The core difference between PCG via machine learning and approaches such as search-based PCG is that the content is created directly (e.g., via sampling) from models which have been trained on game content. While some search-based PCG approaches use evaluation functions that have been trained on game content—for instance, the work of Shaker et al. [621] or Liapis et al. [373]—the actual content generation is still based on search. Below, we present some examples of PCG via machine learning; these particular PCG studies built on the use of n-grams, Markov models and artificial neural networks. For more examples of early work along these lines, see the recent survey paper [668].

4.3.6.1 n-grams and Markov Models

For content that can be expressed as one- or two-dimensional discrete structures, such as many game levels, methods based on Markov models can be used. One par ticularly straightforward Markov model is the n-gram model, which is commonly used for text prediction. The n-gram method is very simple—essentially, you build conditional probability tables from strings and sample from these tables when con structing new strings—and also very fast.
Dahlskog et al. trained n-gram models on the levels of the original Super Mario Bros (Nintendo, 1985) game, and used these models to generate new levels [156]. As n-gram models are fundamentally one-dimensional, these levels needed to be converted to strings in order for n-grams to be applicable. This was done through di viding the levels into vertical "slices," where most slices recur many times through out the level [155]. This representational trick is dependent on there being a large

amount of redundancy in the level design, something that is true in many games.
Models were trained using various levels of n, and it was observed that while n = 0 creates essentially random structures and n = 1 creates barely playable levels, n = 2

and n = 3 create rather well-shaped levels. See Fig. 4.10 for examples of this.
Summerville et al. [667] extended these models with the use of Monte Carlo tree search to guide the generation. Instead of solely relying on the learned condi tional probabilities, they used the learned probabilities during rollouts (generation of whole levels) that were then scored based on an objective function specified by a designer (e.g., allowing them to bias the generation towards more or less diffi- cult levels). The generated levels could still only come from observed configura tions, but the utilization of MCTS meant that playability guarantees could be made and allowed for more designer control than just editing of the input corpus. This can be seen as a hybrid between a search-based method and a machine learning based method. In parallel, Snodgrass and Onta˜n´on trained two-dimensional Markov Chains—a more complex relative of the n-gram—to generate levels for both Su per Mario Bros (Nintendo, 1985) and other similar platform games, such as Lode Runner (Brøderbund, 1983) [644].

4.3.6.2 Neural Networks

Given the many uses of neural networks in machine learning, and the many dif ferent neural network architectures, it is little wonder that neural networks are also highly useful for machine learning-based PCG. Following on from the Super Mario Bros (Nintendo, 1985) examples in the previous section, Hoover et al. [277] gen erated levels for that same game by extending a representation called functional scaffolding for musical composition (FSMC) that was originally developed to com pose music. The original FSMC representation posits 1) music can be represented as a function of time and 2) musical voices in a given piece are functionally re lated [276]. Through a method for evolving neural networks called NeuroEvolu tion of Augmenting Topologies [655], additional musical voices are evolved to be played simultaneously with an original human-composed voice. To extend this mu sical metaphor and represent Super Mario Bros (Nintendo, 1985) levels as functions of time, each level is broken down into a sequence of tile-width columns. The height of each column extends the height of the screen. While FSMC represents a unit of time by the length of an eighth-note, a unit of time in this approach is the width of each column. At each unit of time, the system queries the ANN to decide a height to place a tile. FSMC then inputs a musical note's pitch and duration to the ANNs. This approach translates pitch to the height at which a tile is placed and duration to the number of times a tile-type is repeated at that height. For a given tile-type or musical voice, this information is then fed to a neural network that is trained on two-thirds of the existing human-authored levels to predict the value of a tile-type at each col umn. The idea is that the neural network will learn hidden relationships between the tile-types in the human-authored levels that can then help humans construct entire levels from as little starting information as the layout of a single tile-type.

Of course, machine learning can also be used to generate other types of game content that are not levels. A fascinating example of this is Mystical Tutor, a design assistant for Magic: The Gathering cards [666]. In contrast to some of the other generators that aim to produce complete, playable levels, Mystical Tutor acknowl edges that its output is likely to be flawed in some ways and instead aims to provide inspirational raw material for card designers.

4.4 Roles of PCG in Games

The generation of content algorithmically may take different roles within the do main of games. We can identify two axes across which PCG roles can be placed: players and designers. We envision PCG systems that consider designers while they generate content or they operate interdependently of designers; the same applies for players. Figure 4.11 visualizes the key roles of PCG in games across the dimensions of designer initiative and player experience.

Regardless of the generation method used, game genre or content type PCG can act either autonomously or as a collaborator in the design process. We refer to the former role as autonomous generation (Section 4.4.2) and the latter role as mixed initiative (Section 4.4.1) generation. Further, we cover the experience-driven PCG

role by which PCG algorithms consider the player experience in whatever they try to generate (Section 4.4.3). As a result, the generated content is associated to the player and her experience. Finally, if PCG does not consider the player as part of the generation process it becomes experience-agnostic (Section 4.4.4).

PCG techniques can be used to generate content in runtime, as the player is playing the game, allowing the generation of endless variations, making the game infinitely replayable and opening the possibility of generating player-adapted con tent, or offline during the development of the game or before the start of a game session. The use of PCG for offline content generation is particularly useful when generating complex content such as environments and maps; several examples of that was discussed at the beginning of the chapter. An example of the use of run time content generation can be found in the game Left 4 Dead (Valve, 2008), a first-person shooter game that provides dynamic experience for each player by an alyzing player behavior on the fly and altering the game state accordingly using PCG techniques [14, 60]. A trend related to runtime content generation is the cre ation and sharing of player-generated content. Some games such as LittleBigPlanet (Sony Computer Entertainment, 2008) and Spore (Electronic Arts, 2008) provide a content editor (level editor in the case of LittleBigPlanet and the Spore Creature Creator) that allows the players to edit and upload complete creatures or levels to a central online server where they can be downloaded and used by other players.

With respect to the four different roles of PCG in games, runtime generation is pos sible in the autonomous and experience-agnostic roles, it is always the case in the experience-driven role whereas it is impossible in the mixed-initiative role. On the other hand, the offline generation of content can occur both autonomously and in

Fig. 4.11 The four key PCG roles in games across the dimensions of designer initiative and player experience. For each combination of roles the figure lists a number of indicative examples of tools or studies covered in this chapter.

an experience-agnostic manner. Further it is exclusively the only way to generate content in a mixed-initiative fashion whereas it is not relevant for experience-driven generation as this PCG role occurs in runtime by definition.

The following four subsections outline the characteristics of each of the four PCG roles with a particular emphasis on the mixed-initiative and the experience-driven roles that have not yet covered in length in this chapter.

4.4.1 Mixed-Initiative

AI-assisted game design refers to the use of AI-powered tools to support the game design and development process. This is perhaps the AI research area which is most promising for the development of better games [764]. In particular, AI can assist in the creation of game content varying from levels and maps to game mechanics and narratives.

Fig. 4.12 The mixed-initiative spectrum between human and computer initiative (or creativity) across a number of mixed-initiative design tools discussed in this section. Iconoscope is a mixed initiative drawing game [372], Sentient Sketchbook is a level editor for strategy games [379], Sen tient World is mixed-initiative map editor [380] and Spaceship Design is a mixed-initiative (mostly computer initiative) tool powered by interactive evolution [377]. Adapted from [375].

We identify AI-assisted game design as the task of creating artifacts via the in teraction of a human initiative and a computational initiative. The computational initiative is a PCG process and thus, we discuss this co-design approach under the heading of procedural content generation. Although the term mixed-initiative lacks a concrete definition [497], in this book we define it as the process that considers both the human and the computer proactively making content contributions to the game design task although the two initiatives do not need to contribute to the same degree [774]. Mixed-initiative PCG thus differs from other forms of co-creation, such as the collaboration of multiple human creators or the interaction between a human and non-proactive computer support tools (e.g., spell-checkers or image ed itors) or non-computer support tools (e.g., artboards or idea cards). The initiative of

the computer can be seen as a continuum between the no initiative state, leaving full control of the design to the designer and having the computer program simply carry out the commands of the human designer, to the full initiative state which yields an autonomously creative system. Any state between the two is also possible as we will see in the examples below and as depicted in Fig. 4.12.

4.4.1.1 Game Domains

While the process of AI-assisted game design is applicable to any creative facets within game design [381] it is level design that has benefited the most from it. Within commercial-standard game development, we can find AI-based tools that allow varying degrees of computer initiative. On one end of the scale, level editors such as the Garden of Eden Creation Kit (Bethesda, 2009) or game engines such

as the Unreal Development Kit (Epic Games 2009) leave most of creative process to the designer but they, nevertheless, boost game development through automat ing interpolations, pathfinding and rendering [774]. On the other end of the com puter initiative scale, PCG tools specialized on e.g., vegetation—SpeedTree (IDV, 2002)—or FPS levels—Oblige (Apted, 2007)—only require the designer to set a small amount of generation parameters and, thus, the generation process is almost entirely autonomous.

Within academia, the area of AI-assisted game design tools has seen significant research interest in recent years [785] with contributions mainly to the level de sign task across several game genres including platformers [641], strategy games [380, 379, 774, 378] (see Fig. 4.13(a)), open world games [634], racing games [102], casual puzzle games [614] (see Fig. 4.13(b)), horror games [394], first-person shooters [501], educational games [89, 372], mobile games [482], and adventure games [323]. The range of mixed-initiative game design tools expands to tools that are designed to assist with the generation of complete game rulesets such as the MetaGame [522], the RuLearn [699] and the Ludocore [639] tools to tools that are purposed to generate narratives [480, 673] and stories within games [358].

4.4.1.2 Methods

Any PCG approach could potentially be utilized for mixed-initiative game design. The dominant methods that have so far being used, however, rely on evolutionary computation following the search-based PCG paradigm. Even though evolution, at fist sight, does not appear to be the most attractive approach for real-time processing and generation of content, it offers great advantages associated, in particular, with the stochasticity of artificial evolution, diversity maintenance and potential for bal ancing multiple design objectives. Evolution can be constrained to the generation of playable, usable, or, in general, content of particular qualities within desired design specifications. At the same time, it can incorporate metrics such as novelty [382] or surprise [240], for maximizing the diversity of generated content and thus enabling a change in the creative path of the designer [774]. Evolution can be computationally costly, however, and thus, interactive evolution is a viable and popular alternative for mixed-initiative evolutionary-based generation (e.g., see [102, 380, 377, 501]).

Beyond artificial evolution, another class of algorithms that is relevant for mixed initiative content generation is constraint solvers and constraint optimization. Meth ods such as answer set programming [383, 69] have been used in several AI-assisted level design tools including Tanagra [641] for platformers and Refraction [89] for educational puzzle games. Artificial neural networks can also perform certain tasks in a mixed-initiative manner, such as performing “autocomplete” or level repair [296] through the use of deep learning approachers such as stacked autoencoders.

The goal here is to provide a tool that “fills in” parts of a level that the human de signer does not want or have time to create, and correcting other parts to achieve further consistency.

(a) Sentient Sketchbook

(b) Ropossum

Fig. 4.13 Examples of mixed-initiative level design tools. Sentient Sketchbook (a) offers map

sketch suggestions to the designer via artificial evolution (see rightmost part of the image); the suggestions are evolved to either maximize particular objectives of the map (e.g., balance) or are evolved to maximize the novelty score of the map. In Ropossum (b) the designer may select to design elements of Cut the Rope (Chillingo, 2010) game levels; the generation of the remaining elements are left to evolutionary algorithms to design.

4.4.2 Autonomous
The role of autonomous generation is arguably the most dominant PCG role in games. The earlier parts of this chapter are already dedicated to extensive discus sions and studies of PCG systems that do not consider the designer in their creative process. As a result we will not cover this PCG role in further detail here. What is important to be discussed, however, is the fuzzy borderline between mixed-initiative and autonomous PCG systems. It might be helpful, for instance, to consider au tonomous PCG as the process by which the role of the designer starts and ends with an offline setup of the algorithm. For instance, the designer is only involved in the parameterization of the algorithm as in the case of SpeedTree (IDV, 2002).
One might wish, however, to further push the borderline between autonomous and mixed-initiative generation and claim that generation is genuinely autonomous only if the creative process reconsiders and adapts the utility function that drives the con tent generation—thereby becoming creative in its own right. A static utility function that drives the generation is often referred to as mere generation within the compu tational creativity field [381].
While the line between autonomy and collaboration with designers is still an open research question, for the purposes of this book, we can safely claim that the PCG process is autonomous when the initiative of the designer is limited to algo rithmic parameterizations before the generation starts.
4.4.3 Experience-Driven
As games offer one of the most representative examples of rich and diverse content creation applications and are elicitors of unique user experiences experience-driven PCG (EDPCG) [783, 784] views game content as the building block of games and the generated games as the potentiators of player experience. Based on the above, EDPCG is defined as a generic and effective approach for the optimization of user (player) experience via the adaptation of the experienced content. According to the experience-driven role of PCG in games player experience is the collection of af fective patterns elicited, cognitive processes emerged and behavioral traits observed during gameplay [781].
By coupling player experience with procedural content generation, the experience driven perspective offers a new, player-centered role to PCG. Since games are com posed by game content that, when played by a particular player, elicits experience patterns, one needs to assess the quality of the content generated (linked to the ex perience of the player), search through the available content, and generate content that optimizes the experience for the player (see Fig. 4.14). In particular, the key components of EDPCG are:
• Player experience modeling: player experience is modeled as a function of game content and player.

Fig. 4.14 The four key components of the experience-driven PCG framework.
• Content quality: the quality of the generated content is assessed and linked to the modeled experience.
• Content representation: content is represented accordingly to maximize search efficacy and robustness.
• Content generator: the generator searches through the generative space for con tent that optimizes the experience for the player according to the acquired model.
Each component of EDPCG has its own dedicated literature and the extensive review of each is covered in other parts of the book. In particular, player experi ence modeling is covered in Chapter 5 whereas the remaining three components of the framework have already been covered in this chapter. A detailed survey and

discussion about EDPCG is available in [783].

4.4.3.1 Experience-Driven PCG in Practice

Left 4 Dead (Valve, 2008) is an example of the use of experience-driven PCG in a commercial game where an algorithm is used to adjust the pacing of the game on the fly based on the player's emotional intensity. In this case, adaptive PCG is used to adjust the difficulty of the game in order to keep the player engaged [60]. Adap tive content generation can also be used with another motive such as the generation of more content of the kind the player seems to like. This approach was followed, for instance, in the Galactic Arms Race [250] game where the weapons presented to the player are evolved based on her previous weapon use and preferences. In an other EDPCG study, El-Nasr et al. implemented a direct fitness function—derived from visual attention theory—for the procedural generation of lighting [188, 185].

The procedural Zelda game engine [257], a game engine designed to emulate the

(a) Human

(b) World-Champion AI

Fig. 4.15 Example levels generated for two different Mario players. The generated levels maxi mize the modeled fun value for each player. The level on top is generated for one of the experiment subjects that participated in [521] while the level below is generated for the world champion agent of the Mario AI competition in 2009.

popular The Legend of Zelda (Nintendo, 1986–2017) action-RPG game series, is built mainly to support experience-driven PCG research. Another example is the work by Pedersen et al. [521], who modified an open-source clone of the classic platform game Super Mario Bros (Nintendo, 1985) to allow for personalized level generation. The realization of EDPCG in this example is illustrated in Fig. 4.16.

The first step was to represent the levels in a format that would yield an easily searchable space. A level was represented as a short parameter vector describing the number, size and placement of gaps which the player can fall through, and the presence or absence of a switching mechanic. The next step was to create a model of player experience based on the level played and the player's playing style. Data was collected from hundreds of players, who played pairs of levels with different parameters and were asked to rank which of two levels best induced each of the following user states: fun, challenge, frustration, predictability, anxiety, boredom. While playing, the game also recorded a number of metrics of the players' playing styles, such as the frequency of jumping, running and shooting. This data was then used to train neural networks to predict the examined user states using evolutionary preference learning. Finally, these player experience models were utilized to opti mize game levels for particular players [617]. Two examples of such levels can be seen in Fig. 4.15. It is worth noting—as discussed in Chapter 5—that one may wish to further improve the models of experience of Mario players by including infor mation about the player beyond gameplay [29] such as her head pose [610] or her facial expressions [52].

4.4.4 Experience-Agnostic

With experience-agnostic PCG we refer to any PCG approach that does not con sider the role of the player in the generation of content. But where should we set the boundary of involvement? When do we consider a player as part of the gener ation process and when don't we? While the borderline between experience-driven and experience-agnostic is not trivial to draw we define any PCG approach whose content quality function does not include a player (experience) model or it does not

Fig. 4.16 The EDPCG framework in detail. The gradient grayscale-colored boxes represent a con tinuum of possibilities between the two ends of the box while white boxes represent discrete, exclusive options within the box. The blue arrows illustrate the EDPCG approach followed for the Super Mario Bros (Nintendo, 1985) example study [521, 617]: Content quality is assessed via a direct, data-driven evaluation function which is based on a combination of a gameplay-based (model-free) and a subjective (pairwise preference) player experience modeling approach; content is represented indirectly and exhaustive search is applied to generate better content.

interact with the player in any way during generation as experience-agnostic. As with the role of autonomous PCG, this chapter has already gone through several examples of content generation that do not involve a player or a player experience model. To avoid being repetitive we will refer the reader to the PCG studies covered already that are outside the definition of experience-driven PCG.

4.5 What Could Be Generated?

In this section we briefly outline the possible content types that a PCG algorithm can generate in a game. Generally speaking Liapis et al. [381] identified six creative domains (or else facets) within games that we will follow for our discussion in this section. These include level architecture (design), audio, visuals, rules (game design), narrative, and gameplay. In this chapter we will cover the first five facets and we purposely exclude the gameplay facet. Creative gameplay is directly associated with play and as such is covered in the previous chapter. We conclude this section with a discussion on complete game generation.

4.5.1 Levels and Maps

The generation of levels is by far the most popular use of PCG in games. Levels can be viewed as necessary content since every game has some form of spatial representation or virtual world within which the player can perform a set of actions. The properties of the game level, in conjunction with the game rules, frame the ways a player can interact with the world and determine how the player can progress from one point in the game to another. The game's level design contributes to the challenges a player faces during the game. While games would often have a fixed set of mechanics throughout, the way a level is designed can influence the gameplay and the degree of game challenge. For that reason, a number of researchers have argued that levels coupled with game rules define the absolutely necessary building blocks of any game; in that regard the remaining facets covered below are optional [371].

The variations of possible level designs are endless: a level representation can vary from simple two-dimensional illustrations of platforms and coins—as in the Super Mario Bros (Nintendo, 1985) series—to the constrained 2D space of Candy Crush Saga (King, 2012), to the three-dimensional and large urban spaces of Assassin's Creed (Ubisoft, 2007) and Call of Duty (Infinity Ward, 2003), to the 2D elaborated structures of Angry Birds (Rovio, 2009), to the voxel-based open gameworld of Minecraft (Mojang 2011).

Due to their several similarities we can view the procedural generation of game levels from the lens of procedural architecture. Similarly to architecture, level design needs to consider both the aesthetic properties and the functional requirements of whatever is designed within the game world. Depending on the game genre, func tional requirements may vary from a reachable end-goal for platform games, to a challenging gameplay in driving games such as Forza Motorsport (Turn 10 Stu dios 2005), to waves of gameplay intensity as in Pac-Man (Namco, 1980), Left 4 Dead (Valve, 2008), Resident Evil 4 (Capcom, 2005) and several other games. A procedural level generator also needs to consider the aesthetics of the content as the level's aesthetic appearance may have a significant impact not only on the vi sual stimuli it offers to the player but also on navigation. For example, a sequence of identical rooms can easily make the player disoriented—as was intended in the

Fig. 4.17 The procedurally generated levels of Diablo (Blizzard Entertainment, 1996): one of the most characteristic examples of level generation in commercial games. Diablo is a relatively recent descendant of Rogue (Toy and Wichmann, 1980) (i.e., rogue-like) role-playing game characterized by dungeon-based procedurally generated game levels. Image obtained from Wikipedia (fair use).

dream sequences of Max Payne (Remedy, 2001)—while dark areas can add to the challenge of the game due to low visibility or augment the player's arousal as in the case of Amnesia: The Dark Descent (Frictional Games, 2010), Nevermind (Fly ing Mollusk, 2015) and Sonancia [394]. When the level generator considers larger, open levels or gameworlds then it draws inspiration from urban and city planning [410], with edges to constrain player freedom—landmarks to orient the player and

motivate exploration [381]—as in the Grand Theft Auto (Rockstar Games, 1997) se ries and districts and areas to break the world's monotony—as in World of Warcraft (Blizzard Entertainment, 2004) which uses highly contrasting colors, vegetation and architectural styles to split the world into districts that are suitable for characters of different level ranges.

As already seen broadly in this chapter the generation of levels in a procedural manner is clearly the most popular and possibly the oldest form of PCG in the game industry. We already mentioned the early commercial use of PCG for automatic level design in games such as Rogue (Toy and Wichman, 1980) and the Rogue-inspired Diablo (see Fig. 4.17) series (Blizzard Entertainment, 1996), and the more recent world generation examples of Civilization IV (Firaxis, 2005) and Minecraft (Mo jang, 2011). The level generation algorithms used in commercial games are usually

Fig. 4.18 The caricaturized and highly-emotive visuals of the game Limbo (Playdead, 2010). Im age obtained from Wikipedia (fair use).

constructive, in particular, in games where players can interact with and change the game world via play. Players of Spelunky (Yu and Hull, 2009), for instance, are al lowed to modify a level which is not playable (i.e., the exit cannot be reached) by blowing up the blocking tiles with a bomb provided by the game level.

The academic interest in procedural level generation is only recent [720, 783, 616] but it has produced a substantial volume of studies already. Most of the aca demic studies described in this chapter are focused on levels. The interested reader may refer to those for examples of level generation across various methodologies, PCG roles and game genres.

4.5.2 Visuals

Games are, by definition, visual media unless the game is designed explicitly to not have visuals—e.g., the Real Sound: Kaze no Regret (Sega, 1997) adventure au dio game. The visual information displayed on the screen conveys messages to the player which are dependent on the graphical style, color palette and visual texture.

Visuals in games can vary from simple abstract and pixelized representations as the 8-bit art of early arcade games, to caricaturized visuals as in Limbo (Playdead, 2010) (see Fig. 4.18), to photorealistic graphics as in the FIFA series (EA Sports, 1993) [299].

Within the game industry PCG has been used broadly for the generation of any of the above visual types. Arguably, the complexity of the visual generation task increases the more the resolution and the photorealism of the desired output in creases. There are examples of popular generative tools such as SpeedTree (IDV, 2002) for vegetation and FaceGen (Singular Inversions, 2001) for faces, however, that can successfully output photorealistic 3D visuals. Within academia notable ex amples of visual generation are the games Petalz [565, 566] (flower generation), Galactic Arms Race [250] (weapon generation; see also Fig. 4.3) and AudioInSpace

[275] (weapon generation). In all three games visuals are represented by neural networks that are evolved via interactive evolution. Within the domain of weapon particle generation another notable example is the generation of surprising yet bal anced weapons for the game Unreal Tournament III (Midway Games, 2007) using constrained surprise search [240]; an algorithm that maximizes the surprise score of a weapon but at the same time imposes designer constraints to it so that it is bal anced. Other researchers have been inspired by theories about "universal" properties of beauty [18] to generate visuals of high appeal and appreciation [377]. The PCG algorithm in that study generates spaceships based on their size, simplicity, balance and symmetry, and adapts its visual output to the taste of the visual's designer via interactive evolution. The PCG-assisted design process referred to as iterative refinement [380] is another way of gradually increasing the resolution of the visuals a designer creates by being engaged in an iterative and creative dialog with the visuals generator. Beyond the generation of in-game entities a visuals PCG algorithm may focus on general properties of the visual output such as pixel shaders [282], lighting

[188, 185], brightness and saturation, which can all influence the overall appearance of any game scene.

4.5.3 Audio

Even though audio can be seen as optional content it can affect the player directly and its impact on player experience is apparent in most games [219, 221, 129]. Au dio in games has reached a great level of maturity as demonstrated by two BAFTA Game Awards and an MTV video music award for best video game soundtrack [381]. The audio types one meets in games may vary from fully orchestrated sound track (background) music, as in Skyrim (Bethesda, 2011), to sound effects, as the dy ing or pellet-eating sounds of Pac-Man (Namco, 1980), to the voice-acted sounds of Fallout 3 (Bethesda, 2008). Most notably within indie game development, Proteus (Key and Kanaga, 2013) features a mapping between spatial positioning, visuals and player interaction, which collectively affect the sounds that are played. Profes sional tools such as the sound middleware of UDK (Epic Games, 2004) and the popular sfxr and bfxr sound generation tools provide procedural sound components to audio designers, demonstrating a commercial interest in and need of procedurally generated audio.

At a first glance, the generation of game audio, music and sounds might not seem to be particularly different from any other type of audio generation outside games. Games are interactive, however, and that particular feature makes the generation of audio a rather challenging task. When it comes to the definition of procedural audio in games, a progressive stance has been that its procedurality is caused by the very interaction with the game. (For instance, game actions that cause sound effects of music can be considered as procedural audio creation [220, 128].) Ideally, the game audio must be able to adapt to the current game state and the player behavior. As a result adaptive audio is a grand challenge for composers since the combinations of

all possible game and player states could be largely unknown. Another difficulty for the autonomous generation of adaptive music, in particular, is that it requires real time composition and production; both of which need to be embedded in a game engine. Aside from a few efforts in that direction3 the current music generation models are not particularly designed to perform well in games. In the last decade, however, academic work on procedural game music and sound has seen substantial advancements in venues such as the Procedural Content Generation and the Musical Metacreation workshop series.

Generally speaking, sound can be either diegetic or non-diegetic. A sound is diegetic if its source is within the game's world. The source of the diegetic sound can be visible on the screen (on-screen) or can be implied to be present at the current game state (off-screen). Diegetic sounds include characters, voices, sounds made by items on-screen or interactions of the player, or even music represented as coming from instruments available in the game. A non-diegetic sound, on the other hand, is any sound whose source is outside the game's world. As such non-diegetic sounds cannot be visible on-screen or even implied to be off-screen. Examples include com mentaries of a narrator, sound effects which are not linked to game actions, and background music.

A PCG algorithm can generate both diegetic and non-diegetic sounds including music, sound effects and commentaries. Examples of non-diegetic sound generation in games include the Sonancia horror sound generator that tailors the tension of the game to the desires of the designer based on the properties of the game level [394]. The mapping between tension and sounds in Sonancia has been derived through crowdsourcing [396]. Similarly to Sonancia—and towards exploring the creative space between audio and level design—Audioverdrive generates levels from audio and audio from levels [273]. Notably within diegetic audio examples, Scirea et al. [606] explores the relationship between procedurally generated music and narrative. Studies have also considered the presence of game characters on-display for the composition of game soundtracks [73] or the combination of short musical phrases

that are driven by in-game events and, in turn, create responsive background audio for strategy games [280].

Finally, it is worth mentioning that there are games featuring PCG that use music as the input for the generation of other creative domains rather than music per se. For example, games such as Audio Surf (Fitterer, 2008) and Vib Ribbon (Sony En tertainment, 2000) do not procedurally generate music but they instead use music to drive the generation of levels. AudioInSpace [275] is another example of a side scrolling space shooter game that does not generate music but uses the background music as the basis for weapon particle generation via artificial evolution.

3 For instance, see the upcoming melodrive app at: http://melodrive.com.

4.5.4 Narrative

Many successful games are relying heavily on their narratives; the clear distinction however, between such narratives and traditional stories is the interactivity element that is offered by games. Now, whether games can tell stories [313] or games are instead a form of narrative [1] is still an open research question within game stud ies, and beyond the scope of this book. The study of computational (or procedural) narrative focuses on the representational and generational aspects of stories as those can be told via a game. Stories can play an essential part in creating the aesthetics of a game which, in turn, can impact affective and cognitive aspects of the playing experience [510].

By breaking the game narrative into subareas of game content we can find core game content elements such as the game's plotline [562, 229], but also the ways a story is represented in the game environment [730, 83]. The coupling of a game's representation and the story of the game is of vital importance for player experi ence. Stories and plots are taking place in an environment and are usually told via a virtual camera lens. The behavior of the virtual camera—viewed as a parame terized element of computational narrative—can drastically influence the player's experience. That can be achieved via an affect-based cinematographic representa tion of multiple cameras as those used in Heavy Rain (Quantic Dream, 2010) or through an affect-based automatic camera controller as that used in the Maze-Ball game [780]. Choosing the best camera angle to highlight an aspect of a story can be seen as a multi-level optimization problem, and approached with combinations of optimization algorithms [85]. Games such as World of Warcraft (Blizzard Entertain ment, 2004) use cut scenes to raise the story's climax and lead the player to partic ular player experience states. The creation or semi-automatic generation of stories and narratives belongs to the area of interactive storytelling, which can be viewed as a form of story-based PCG. The story can adjust according to the actions of the player targeting personalized story generation (e.g., see [568, 106] among others). Ultimately, game worlds and plot point story representations can be co-generated as demonstrated in a few recent studies (e.g., see [248]).

Computational narrative methods for generating or adapting stories of exposi tions are typically build on planning algorithms, and planning is therefore essential for narrative [792]. The space of stories can be represented in various ways, and the representations in turn make use of dissimilar search/planning algorithms, includ ing traditional optimization and reinforcement learning approaches [483, 117, 106].

Cavazza et al. [106], for instance, introduced an interactive storytelling system built with the Unreal game engine that uses Hierarchical Task Network planning to sup port story generation and anytime user intervention. Young et al. [792] introduced an architecture called Mimesis, primarily designed to generate intelligent, plan-based character and system behavior at runtime with direct uses in narrative generation. Finally the IDtension engine [682] dynamically generates story paths based on the

player's choices; the engine was featured in Nothing for Dinner,4 a 3D interactive story aiming to help teenagers living challenging daily life situations at home.

Similarly to dominant approaches of narrative and text generation, interactive

storytelling in games relies heavily on stored knowledge about the (game) world. Games that rely on narratives—such as Heavy Rain (Quantic Dream, 2010)—may include thousands of lines of dialog which are manually authored by several writers. To enable interactive storytelling the game should be able to select responses (or paths in the narrative) based on what the player will do or say, as in Fac¸ade [441] (see Fig. 4.19). To alleviate, in part, the burden of manually representing world knowledge, data-driven approaches can be used. For instance, one may crowdsource actions and utterance data from thousand of players that interact with virtual agents of a game and then train virtual agents to respond in similar ways using n-grams [508]. Or instead, one might design a system in which designers collaborate with a computer by taking turns on adding sentences in a story; the computer is able to provide meaningful sentences by matching the current story with similar stories available on the cloud [673]. Alternatively, a designer could use the news of the day from sites, blogs or Wikipedia and generate games that tell the news implicitly via play [137].

Research on interactive narrative and story-based PCG benefits from and influences the use of believable agents that interact with the player and are inter woven in the story plot. The narrative can yield more (or less) believability to agents and thus the relationship between agents and the story they tell is impor tant [801, 401, 531, 106]. In that sense, the computational narrative of a game may define the arena for believable agent design. Research on story-based PCG has also influenced the design and development of games. Starting from popular indepen dent attempts like Fac¸ade [441] (see Fig. 4.19), Prom Week [448] and Nothing for Dinner to the commercial success of The Elder Scrolls V: Skyrim (Bethesda Soft works, 2011), Heavy Rain (Quantic Dream, 2010) and Mass Effect (Bioware, 2007) narrative has traditionally been amongst the key factors of player experience and immersion; particularly in narrative-heavy games as the ones aforementioned. Examples of sophisticated computational narrative techniques crossing over from academia to commercial-standard products include the storytelling system Versu [197] which was used to produce the game Blood & Laurels (Emily Short, 2014). For the interested reader the interactive fiction database[5] contains a detailed list of games built on the principles of interactive narratives and fiction, and the stoy gen.org[6] repository, by Chris Martens and Rogelio E. Cardona-Rivera, maintains existing openly-available computational story generation systems. Finally note that the various ways AI can be used to play text-based adventure games and interactive fiction are covered in Chapter 3.

4 http://nothingfordinner.org

5 http://ifdb.tads.org/

6 http://storygen.org/

Fig. 4.19 A screenshot from the Fac¸ade [441] game featuring its main characters: Grace and Trip. The seminal design and AI technologies of Fac¸ade popularized the vision of interactive narrative and story-based PCG within games.

4.5.5 Rules and Mechanics

The game rules frame the playing experience by providing the conditions of play—for instance, winning and losing conditions—and the actions available to the player (game mechanics). Rules constitute necessary content as they are in a sense the core of any game, and a game's rules pervade it.

For most games, the design of their ruleset largely defines them and contributes to their success. It is common that the rule set follows some standard design patterns within its genre. For example, the genre of platform games is partly defined by run ning and jumping mechanics, whereas these are rare in puzzle games. Evidently, the genre constrains the possibility (design) space of the game rules. While this prac tice has been beneficial—as rule sets are built on earlier successful paradigms—it can also be detrimental to the creativity of the designer. It is often the case that the players themselves can create new successful game variants (or even sub-genres) by

merely altering some rules of an existing game. A popular example is the modification of Warcraft III (Blizzard, 2002) which allowed the player to control a single "hero" unit and, in turn, gave rise to a new, popular game genre named Multiplayer Online Battle Arenas (MOBA).

Most existing approaches to rule generation take a search-based approach, and are thus dependent on some way of evaluating a set of rules [711, 355]. However, accurately estimating the quality of a set of game rules is very hard. Game rules differ from most other types of game content in that they are almost impossible to evaluate in isolation from the rest of the game. While levels, characters, textures, and many other types of content can to some extent be evaluated outside of the game, looking at a set of rules generally gives very little information of how they play. For a human, the only way to truly experience the rules of a game is to play the game. For a computer, this would translate to simulating gameplay in some way in order to evaluate the rules. (In this sense, rules can be said to be more similar to program code than they are to e.g., pictures or music.)
So how can simulated playthroughs be used to judge the quality of the rulesets? Several ideas about how to judge a game depending on how agents play it have been introduced. The first is balance; for symmetric two-player games in particular, bal ance between the winning chances of the two players is generally positive [274].
Another idea is outcome uncertainty, meaning that any particular game should be "decided" as late as possible [76]. Yet another idea is learnability: a good game, including its ruleset, is easy to learn and hard to master. In other words, it should have a long, smooth learning curve for the player, as learning to play the game is part of what makes it fun. This idea can be found expressed most clearly in Koster's "Theory of Fun" [351], but can also be said to be implicit in Schmidhuber's the ory of artificial curiosity [602] and in theories in developmental psychology [204].
Within work in game rule generation, attempts have been made to capture this idea in different ways. One way is to use a reinforcement learning agent to try to learn to play the game; games where the agent improves the most over a long time score the best [716]. Another way of capturing this idea is to use several agents of different skill levels to try to play the game. Games score better when they maximize the performance difference between these agents [491]. This idea is also related to the idea of depth in games, which can be seen as the length of the chain of heuristics that can be acquired in a game [362].
Perhaps the most successful example of game rule generation within academic research is Ludi [76]. Ludi follows a search-based PCG approach and evolves gram mars that represent the rules of board games (see Fig. 4.20). The fitness function that drives the rule generation is composed by several metrics that estimate good design patterns in board games, such as game depth and rule simplicity. A success fully designed game that came out of the Ludi generator is named Yavalath [75] (see Fig. 4.20). The game is played on a 5-by-5 hexagonal board by two or three players. Yavalath's winning and losing conditions are very simple: you win if you make a line of four tokens, whereas you lose if you make a line of three tokens; the game is a draw if the board fills up.
One of the earliest examples of video game rule generation is Togelius and Schmidhuber's experiment with generating simple Pac-Man-like games [716]. In that study rules are evolved to maximize the learnability of player agents as mea sured via simulated playthroughs. Another example is the work by Nielsen et al. in which game rules are represented using the video game description language [492].

Using Answer Set Programming (a solver-based method) [69] rather than search based methods, rules have been generated for simple 2D games that, however, re spect constraints such as playability (i.e., the victory condition is attainable) [637].
It is fair to say that none of these attempts to generate video games have been able to produce good games, i.e., games that anyone (except the creator of the system that creates the games) would want to play. This points to the immense challenge in

accurately estimating the quality of a video game rule set. One of the reasons this seems to be a more challenging problem than estimating the quality of a board game rule set is the time dimension, as human reaction time and ability to estimate and predict unfolding processes play an important role in the challenge of many video games.

For more examples and in-depth analysis on rule and mechanics generation the interested reader is referred to the “rules and mechanics” chapter of the PCG book [486].

4.5.6 Games

Game generation refers to the use of PCG algorithms for computationally design ing new complete games. The vast majority of PCG studies so far, however, have been very specific to a particular game facet or domain. It is, for instance, either a level that is generated or the audio for some level but rarely both. Meanwhile it is surprising to think that the relationship between the different facets is naturally interwoven. A characteristic example of the interwoven nature among game facets is given in [381]: player actions—viewed as a manifestation of game rules—are usually accompanied by corresponding sound effects such as the sound of Mario jumping in Super Mario Bros (Nintendo, 1985). Now let us think of a PCG al gorithm that introduces a new rule to the game—hence a new player action. The algorithm automatically constrains the sound effects that can be associated to this new action based on a number of factors such as the action’s duration, purpose and overall contribution to the game plot. Actions and sounds appear to have a cause and effect (or hierarchical) relationship and a PCG algorithm would naturally prioritize the creation of the action before it generates its sound. Most relationships between facets, however, are not strictly hierarchical or unidirectional. For example, a game level can be successful because of a memorable landmark as much as the gameplay it affords [381]. Similarly, the narrative of a game relies on a multitude of factors including the camera placement as well as the visuals and the sounds.

The game generation topic has attracted a growing interest in recent years even though the relationship between the different games facets is not considered largely. Most game generation projects focus on a single facet of a game and do not inves tigate the interaction between different facets. The rule generator for Pac-Man-like games [716], for instance, evolves different rules for different colored agents but it does not evolve the agents’ color to indicate different playing strategies. Similarly

(a) The Ludi game rule generator

(b) A deluxe version of the Yavalath game

Fig. 4.20 The Ludi game rule generator (a) and its game Yavalath (b). Image (a) is adapted from [720]. Image (b) is published with permission by Cameron Browne and N´estor Romeral Andr´es.

to the ghosts of Pac-Man (Namco, 1981) we could imagine that red represents an aggressive behavior whereas orange represents a passive behavior.

Among the few notable attempts of game generation, Game-O-Matic [723] is a game generation system that creates games representing ideas. More specifically, Game-O-Matic includes an authoring tool in which a user can enter entities and their interactions via concept maps. The entities and interactions are translated, re spectively, into game visuals and game mechanics; the mechanics, however, do not take into account the visuals or semantics of the game objects they are applied on.

One of the first preliminary discussions on how multi-facet integration might hap pen is offered by Nelson and Mateas [484]. In their paper they present a system for matching sprites (visuals) to very simple WarioWare-style mechanics. Their system is somewhat similar to Game-O-Matic, but works a bit differently: instead of the designer specifying verbs and nouns she wants a game to be about, she gives the system constraints on how verbs and nouns relate in the game (for example, a chas ing game needs a “prey” sprite that is something that can do things like “flee” or “be hunted” and so on). The system then uses ConceptNet7 and WordNet8

to generate
games that fit these constraints.
Arguably one of the most elaborate examples of game generation is ANGELINA
[135, 137, 136]. ANGELINA9
is a game generator that has seen several develop ments over the years and is currently capable of evolving the rules and levels of the
game, collecting and placing pieces of visuals and music (that are relevant to the theme and the emotive mood of the game), giving names for the games it creates and even creating simple commentaries that characterize them. ANGELINA is able to generate games of different genres—including platformer games (see Fig. 4.21) and 3D adventure games—some of which have even participated in game design competitions [136].
The systems above make some initial, yet important, steps towards game gener ation and they attempt to interweave the different domains within games in a mean ingful way, mostly in a hierarchical fashion. However, PCG eventually needs to
rise to the challenge of tackling the compound generation of multiple facets in an orchestrated manner [711, 371]. An early study on the fusion of more than one generative facet (domain) in games is the one performed recently by Karavolos et al. [324]. The study employs machine learning-based PCG to derive the common generative space—or the common patterns—of game levels and weapons in first person shooters. The aim of this orchestration process between level design and
game design is the generation of level-weapon couplings that are balanced. The un known mapping between level representations and weapon parameters is learned by
a deep convolutional neural network which predicts if a given level with a particular set of weapons will be balanced or not. Balance is measured in terms of the win-lose ratio obtained by AI bots playing in a deathmatch scenario. Figure 4.22 illustrates the architecture used to fuse the two domains. For the interested reader in the or-
7 http://conceptnet.io/
8 https://wordnet.princeton.edu/
9 http://www.gamesbyangelina.org/

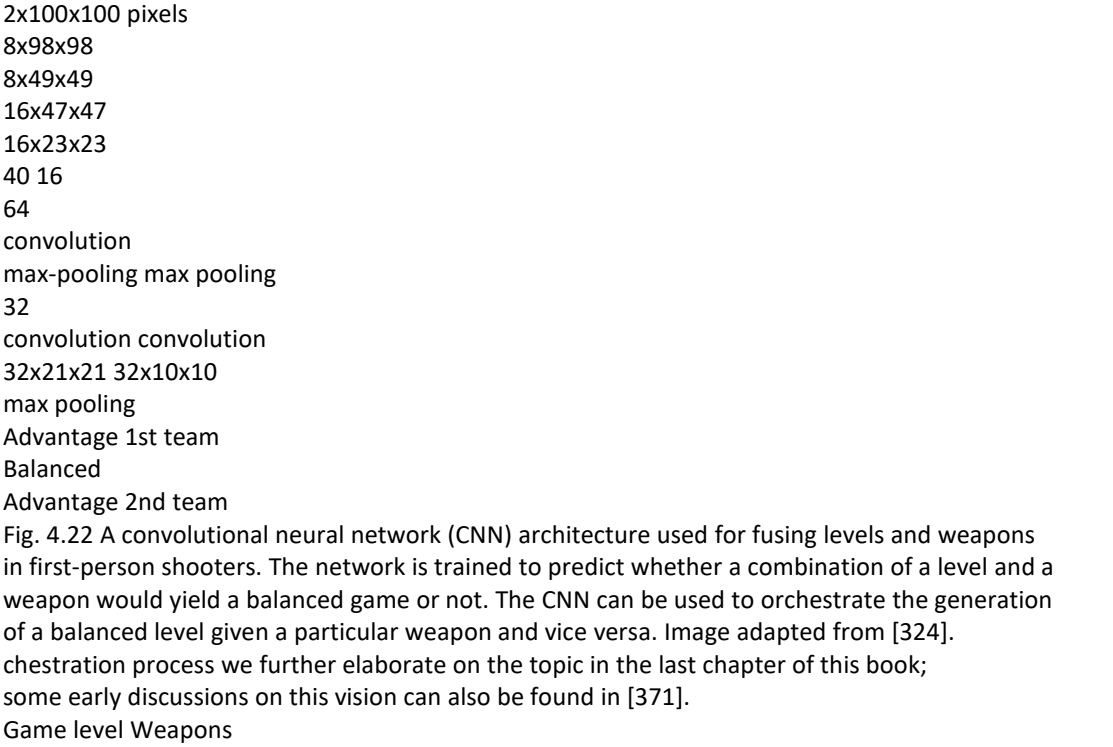
Fig. 4.21 ANGELINA's Puzzling Present game. The game features an invert gravity mechanic that allows a player to overcome the high obstacle on the left and complete the level. Image obtained with permission from http://www.gamesbyangelina.org/.


Fig. 4.22 A convolutional neural network (CNN) architecture used for fusing levels and weapons in first-person shooters. The network is trained to predict whether a combination of a level and a weapon would yield a balanced game or not. The CNN can be used to orchestrate the generation of a balanced level given a particular weapon and vice versa. Image adapted from [324].
chestration process we further elaborate on the topic in the last chapter of this book;
some early discussions on this vision can also be found in [371].

4.6 Evaluating Content Generators
Creating a generator is one thing; evaluating it is another. Regardless of the method followed all generators shall be evaluated on their ability to achieve the desired goals of the designer. Arguably, the generation of any content is trivial; the generation of valuable content for the task at hand, on the other hand, is a rather challenging procedure. (One may claim, however, that the very process of generating valuable content is also, by itself, trivial as one can design a generator that returns a random sample of hand-crafted masterpieces.) Further, it is more challenging to generate content that is not only valuable but is also novel or even inspiring.
4.6.1 Why Is It Difficult?
But what makes the evaluation of content so difficult? First, it is the diverse, stochas tic and subjective nature of the users that experience the content. Whether players or designers, content users have dissimilar personalities, gameplay aims and behavior, emotive responses, intents, styles and goals [378]. When designing a PCG system it is critical to remember that we can potentially generate massive amounts of content for designers to interact with and players to experience. It is thus of utmost impor tance to be able to evaluate how successful the outcomes of the generator might be across dissimilar users: players and designers. While content generation is a cheap process relying on algorithms, design and game-play are expensive tasks relying on humans who cannot afford the experience of bad content. Second, content quality might be affected by algorithms and their underlying stochasticity, for instance, in evolutionary search. Content generators often exhibit non-deterministic behavior, making it very hard to predict a priori what the outcomes of a particular generative system might be.
4.6.2 Function vs. Aesthetics
Particular properties of content can be objectively defined and tested whereas other properties of it can only be assessed subjectively. It is only natural to expect that functional properties of content quality can be objectively defined whereas a large part of its aesthetics can only be defined subjectively. For instance, playability of a level is a functional characteristic that can be objectively measured—e.g., an AI agent manages to complete the level; hence it is playable. Balance and symmetry can also be objectively defined to a degree through estimates of deviation from a norm—it may be a score (balance) or a distance from a central choke point in the map (symmetry). There are games, however, for which content balance, symmetry and other functional properties are not trivially measurable. And of course there are several aspects of content such as the comprehensibility of a narrative, the pleas-

antnesses of a color scheme, the preference for a room's architectural style or the graphics style, and the experience of sound effects and music that are not objectively measured.
Functional, objectively defined, content properties can be expressed either as metrics or as constraints that a generator needs to satisfy. Constraints can be spec ified by the content designers or imposed by other game content already available.
For instance, let us assume that a well-designed generated strategy game level needs to be both balanced and playable. Playability can form a simple binary constraint: the level is playable when an AI agent manages to complete it; it is non-playable oth erwise. Balance can form another constraint by which all items, bases and resources are accessible to similar degrees by all players; if equal accessibility is below a threshold value then the constraint is not satisfied. Next, let us suppose we wish to generate a new puzzle for the map we just generated. Naturally, the puzzle needs to be compatible with our level. A PCG algorithm needs to be able to satisfy these constraints as part of its quality evaluation. Constrained satisfaction algorithms such as the feasible-infeasible two-population genetic algorithm [379, 382], constrained divergent search rewarding content value but also content novelty [382] or surprise [240], and constraint solvers such as answer set programming [638] are able to handle this. The generated results are within constraints, thereby valuable for the

designer. Value, however, may have varying degrees of success and this is where alternative methods or heuristics can help, as we cover in the section below.

4.6.3 How Can We Evaluate a Generator?

Generally speaking, a content generator can be evaluated in three ways: directly by the designer or indirectly by either human players or AI agents. Designers can directly observe properties of the content generator and take decisions based on data visualization methods. Human players can play and test the content and/or provide feedback about the content via subjective reporting. AI agents can do the same: play the content or measure something about the content and report it to us in the form of a quality metric, or metrics. Clearly, machines cannot experience the content but they can, instead, simulate it and provide us estimates of content experience. The overall evaluation process can very well combine and benefit from any of the above approaches. In the remainder of this section we cover the approaches of data visualization, AI automated playtesting and human playtesting in further detail.

4.6.3.1 Visualization

The visualization approach to content quality assessment is associated with a) the computation of meaningful metrics that can assign measurable characteristics to content and b) ways to visualize these metrics. The task of metric design can be viewed as the equivalent of fitness function design. As such, designing good con-

Fig. 4.23 The expressive range of the Ropossum level generator for the metrics of linearity and density. Adapted from [608].

tent generation quality metrics in an ad-hoc manner involves a degree of practical wisdom. Metrics, for instance, can be based on the expressive range of the gener ator under assessment, so-called expressivity metrics [640, 608]. The analysis of a generator's expressivity gives us access to the potential overall quality of the gen erator across its full range of generative space. The generative space can then be visualized as heatmaps or alternative graphical representations such as 2D or 3D scatter plots (see Fig. 4.23 for an example). It is one thing if a level generator is able to create only a few meaningful or playable levels and another if the genera tor is robust and consistent with respect to the playability of its generated levels.

It is also one thing if our generator is only able to generate levels with very spe cific characteristics within a narrow space of its expressive range and another if our level generator is able to express a broad spectrum of level properties, yielding uni formly covered expressive ranges. Such information can be directly visible on the illustrated heatmaps or scatter plots. Alternatively, data compression methods can be used directly on the generated content and offer us 2D or 3D representations of the generative space, thereby, bypassing the limitations of ad-hoc metric design. An example of this approach is the use of autoencoders for compressing the images produced by the DeLeNoX autonomous content generator [373].

4.6.3.2 AI

Using AI to playtest our generator is a safe and relatively cheap way to rapidly retrieve quality metrics for it without relying on human playtesting. In the same way that a search-based PCG method would use AI to simulate the content before generating it, AI agents can test the potential of a generator across a number of metrics and return us values about its quality. The metrics can be in the form of classes—for instance, test checks performed for completing an area of a level—scalar values—e.g., a level's balance—or even ordinal values—e.g., the rank of the level in terms of asymmetry. The relevance of the metrics to the generator's quality is obviously dependent on the designer's ad-hoc decisions. Once again, designing appropriate metrics for our AI agent to compute is comparable to the challenge of designing any utility function. An interesting approach to AI-based testing is the use of procedural personas [267, 269]. These are data-driven inferred models of dissimilar play styles that potentially imitate the different styles of human play. In a sense, procedural personas provide a more human-realistic approach to AI-based testing of a generator. Finally, by visualizing particular game artifacts or simulating

them through the use of AI agents we can have access to the information we might be able to extract from a game, we can understand what is possible within our content space, we can infer how rules and functions operate in whatever we generate, and we can possibly understand how the information we are able to extract relates to data we can extract from human playtesting [481].

4.6.3.3 Human Players

In addition to data visualization and AI-based simulation for the evaluation of a con tent generator a designer might wish to use complementary approaches that rely on quantitative user studies and playtesting. Playtesting is regarded to be an expensive way to test a generator but it can be of immense benefit for content quality assurance, in particular for those aspects of content that cannot be measured objectively—e.g., aesthetics and playing experience. The most obvious approach to evaluate the con tent experienced by players is to explicitly ask them about it. A game user study can involve a small number of dedicated players that will play through various amounts of content or, alternatively, a crowdsourcing approach can provide sufficient data to machine learn content evaluation functions (see [621, 370, 121] among others). Data obtained can be in any format including classes (e.g., a binary answer about the quality of a level), scores (e.g., the likeliness of a sound) or ranks (e.g., a preference about a particular level). It is important to note that the playtesting of content can be complemented by annotations coming from the designers of the game or other experts involved in content creation. In other words, our content generator may be labeled with both first-person (player) and third-person (designer) annotations. Fur ther guidelines about which questionnaire type to use and advice about the design of user study protocols can be found in Chapter 5.

4.7 Further Reading

Extensive versions of most of the material covered in this chapter can be found in dedicated chapters of the PCG in Games Textbook [616]. In particular, all the methods and types of PCG in games are covered in further detail (see Chapters 1 to 9 of [616]). Considering the roles of PCG, the mixed-initiative and the experience driven role of PCG are, respectively, detailed in Chapters 11 [374] and 10 [618] of that textbook [616]. In addition to Chapter 11 of [616] the framework named mixed-initiative co-creativity [774] provides a theoretical grounding for the impact of mixed-initiative interaction on the creativity of both designers and computational processes. Further, the original articles about the experience-driven PCG framework can be found in [783, 784]. Finally, the topic of PCG evaluation is covered also in the final chapter [615] of the PCG in Games textbook [616].

4.8 Exercises

PCG offers endless opportunities for generation and evaluation across the different creativity domains in games and across combinations of those. As an initial step we would recommend the reader to start experimenting with maze generation and platformer level generation (as outlined below). The website of the book contains details regarding both frameworks and potential exercises.

4.8.1 Maze Generation

Maze generation is a very popular type of level generation and relevant for several game genres. In the first exercise we recommend that you develop a maze genera tor using both a constructive and a search-based PCG approach and compare their performance according to a number of meaningful criteria that you will define. The reader may use the Unity 3D open-access maze generation framework which is available at: http://catlikecoding.com/unity/tutorials/maze/. Further guidelines and exercises for maze generation can be found at the book's website.

4.8.2 Platformer Level Generation

The platformer level generation framework is based on the Infinite Mario Bros(Pers son, 2008) framework which has been used as the main framework of the Mario AI (and later Platformer AI) Competition since 2010. The competition featured several different tracks including gameplay, learning, Turing test and level generation. For the exercises of this chapter the reader is requested to download the level generation

framework (https://sites.google.com/site/platformersai/) and apply constructive and generate-and-test methods for the generation of platformer levels. The levels need to be evaluated using one or more of the methods covered in this book. Further details and exercises with the platformer level generation framework can be found at the book's website.

4.9 Summary

This chapter viewed AI as a means for generating content in games. We defined procedural content generation as the algorithmic process of creating content in and for games and we explored the various benefits of this process. We then provided a general taxonomy about content and its generation and explored the various ways one can generate content including search-based, solver-based, grammar-based, ma chine learning-based, and constructive generation methods. The use of the PCG method is naturally dependent on the task at hand and the type of content one wishes to generate. It further depends on the potential role the generator might take within games. We outlined the four possible roles a generator can take in games which are determined by the degree to which they involve the designer (autonomous vs. mixed-initiative) and/or the player (experience-agnostic vs. experience-driven) in the process. The chapter ends with a discussion on the important and rather un explored topic of evaluation, the challenges it brings, and a number of evaluation approaches one might consider.

We have so far covered the most traditional use of AI in games (Chapter 3) and the use of AI for generating parts of (or complete) games (this chapter). The next and final chapter of this part is dedicated to the player and the ways we can use AI to model aspects of her behavior and her experience.

Chapter 5
Modeling Players

This chapter is dedicated to players and the use of AI for modeling them. This area of research is often called player modeling [782, 636]. We take player modeling to mean the detection, prediction and expression of human player characteristics that are manifested through cognitive, affective and behavioral patterns while playing games. In the context of this book, player modeling studies primarily the use of AI methods for the construction of computational models of players. By model we re fer to a mathematical representation—it may be a rule set, a vector of parameters, or a set of probabilities—that captures the underlying function between the charac teristics of the player and her interaction with the game, and the player's response to that interaction. Given that every game features at least one player (with some no table exceptions [50]), and that player modeling affects work on game-playing and content generation, we consider the modeling of player behavior and experience as a very important use of AI in games [764, 785].

Psychology has studied human behavior, cognition and emotions for a long time. Branches of computer science and human-computer interaction that attempt to model and simulate human behavior, cognition, emotion or the feeling of emo tion (affect) include the fields of affective computing and user modeling. Player modeling is related to these fields but focuses on the domain of games. Notably, games can yield dynamic and complex emotions in the player, the manifestations of which cannot be captured trivially by standard methods in empirical psychol ogy, affective computing or cognitive modeling research. The high potential that games have in affecting players is mainly due to their ability to place the player in a continuous mode of interaction, which, in turn, elicits complex cognitive, affective and behavioral responses. Thus, the study of the player may not only contribute to the design of improved forms of human-computer interaction, but also advance our knowledge of human experiences.

As mentioned earlier, every game features at least one user—the player—who controls some aspect of the game environment. The player character could be visible in the game as an avatar or a group of entities [94], or could be invisible as in many puzzle games and casual games. Control may vary from the relatively simple (e.g.,

limited to movement in an orthogonal grid) to the highly complex (e.g., having

to decide several times per second between hundreds of different possibilities in a highly complex 3D world). Given these intricacies, understanding and modeling the interaction between the player and the game can be seen as a holy grail of game design and development. Designing the interaction and the emergent experience right results in a successful game that manages to elicit unique experiences.

The interaction between the player(s) and the game is dynamic, real-time and in many cases highly complex. The interaction is also rich in the sense that many modalities of interaction may be involved and that the information exchange be tween the game and the player may both be fast and entail large amounts of data for us to process. If the game is well-designed, the interaction is also highly engaging for the player. Given the great amount of information that can be extracted through this interaction and used for creating models of the player, the game should be able to learn much about the person playing it, as a player and perhaps as a human in general. In fact, there is no reason why the model should not know more about how you play than you do.

In the remainder of this chapter we first attempt to define the core ingredients of player modeling (Section 5.1) and then we discuss reasons why AI should be used to model players (Section 5.2). In Section 5.3 we provide a high-level taxonomy of player modeling focusing on two core approaches for constructing a player model: top-down and bottom-up. We then detail the available types of data for the model's input (Section 5.4), a classification for the model's output (Section 5.5) and the various AI methods that are appropriate for the player modeling task (Section 5.6). The key components of player modeling as discussed in this chapter (input, output and model) are depicted in Fig. 5.1. We conclude, in Section 5.7, with a number of concrete examples of AI being used for modeling players.

5.1 What Player Modeling Is and What It Is Not

One could arguably detect behavioral, emotional or cognitive aspects of both hu man players and non-human players, or non-player characters (notwithstanding the actual existence of emotions in the latter). However, in this book we focus on as pects that can be detected from, modeled from, and expressed in games with human players [782]. We explicitly exclude the modeling of NPCs from our discussion in this chapter, as in our definition, player modeling is modeling of a human player. Modeling the experience of an NPC would seem to be a futile exercise, as one can hardly say that an NPC possesses actual emotions or cognition. Modeling the be havior of an NPC is also of little interest, at least if one has access to the game's code: a perfect model for the NPC already exists. NPC modeling, however, can be a useful testbed for player modeling techniques, for instance, by comparing the model derived from human players with the hand-crafted one. More interestingly, it can be an integral component of AI that adapts its behavior in response to the dynamics of the NPCs—as in [28]. Nevertheless, while the challenges faced in modeling NPCs

Fig. 5.1 The key components of player modeling as discussed in this chapter. The distinction be tween model-based and model-free approaches is outlined in Section 5.3. The various options for the input of the model are discussed in Section 5.4. The taxonomy for the model's output is dis cussed in Section 5.5—each box represents a dedicated subsection. Finally, the various AI methods (supervised learning, reinforcement learning and unsupervised learning) used for modeling corre sponding output data types are discussed thoroughly in Section 5.6.

are substantial, the issues raised from the modeling of human players define a far more complex and important problem for the understanding of player experience.

Sometimes the terms player modeling and opponent modeling [214, 592, 48] are used interchangeably when a human player is modeled. However, opponent mod eling is a more narrow concept referring to predicting behavior of an adversarial player when playing to win in an imperfect information game like Poker [48] or StarCraft (Blizzard Entertainment, 1988) [504]. Some aspects of modeling NPCs

or simulated playthroughs for winning in a game are discussed in Chapter 3.
We also make a distinction between player modeling [116, 281] and player profiling [782]. The former refers to modeling complex dynamic phenomena during gameplay interaction, whereas the latter refers to the categorization of players based on static information that does not alter during gameplay. Information of static nature includes personality, cultural background, gender and age. We put an emphasis on the former, but will not ignore the latter, as the availability of a good player profile may contribute to the construction of reliable player models.

In summary, player modeling—as we define it in this book—is the study of computational means for the modeling of a player's experience or behavior which is based on theoretical frameworks about player experience and/or data derived from the interaction of the player with a game [782, 764]. Player models are built on dynamic information obtained during game-player interaction, but they could also rely on static player profiling information. Unlike studies focusing on taxonomies of behavioral player modeling—e.g., via a number of dimensions [636] or direct/indirect measurements [623]—we view player modeling in a holistic manner including cognitive, affective, personality and demographic aspects of the player. Moreover, we exclude approaches that are not directly based on human-generated data or not based on empirically-evaluated theories of player experience, human cognition, affect or behavior. The chapter does not intend to provide an exhaustive review of player modeling studies under the above definition, but rather an introduction and a high level taxonomy that explores the possibilities with respect to the modeling approach, the model's input and the model's output.

5.2 Why Model Players?

The primary goal of player modeling is to understand how the interaction with a game is experienced by individual players. Thus, while games can be utilized as an arena for eliciting, evaluating, expressing and even synthesizing experience, we argue that the main aim of the study of players in games is the understanding of players' cognitive, affective and behavioral patterns. Indeed, by the very nature of games, one cannot dissociate games from player experience.
There are two core reasons that drive the use of AI for modeling game players and their play, thereby serving the primary goal of player modeling as stated above. The first is for understanding something about their players' experience during play. Models of player experience are often built using machine learning methods, typically supervised learning methods like support vector machines or neural networks. The training data here consists of some aspect of the game or player-game interaction, and the targets are labels derived from some assessment of player experience, gathered for example from physiological measurements or questionnaires [781]. Once predictors of player experience are derived they can be taken into account for designing the in-game experience. That can be achieved by adjusting the behavior of non-player characters (see Chapter 3) or by adjusting the game environment (see Chapter 4).
The second reason why one would want to use AI to model players is for understanding players' behavior in the game. This area of player modeling is concerned with structuring observed player behavior even when no measures of experience are available—for instance, by identifying player types or predicting player behavior via game and player analytics [178, 186]. A popular distinction in data derived from games [186] is the one between player metrics and game metrics. The latter is a superset of the former as it also includes metrics about the game software (system

metrics) and the game development process as a whole (process metrics). System metrics and process metrics are important aspects of modern game development that influence decision making with respect to procedures, business models, and marketing. In this book, however, we focus on player metrics. The interested reader may refer to [186] for alternative uses of metrics in games and the application of analytics to game development and research—i.e., game analytics.

Once aspects of player behavior are identified a number of actions can be taken to improve the game such as the personalization of content, the adjustment of NPCs or, ultimately, the redesign of (parts of) the game. Derived knowledge about the in-game behavior of the player can lead to improved game testing and game design procedures, and better monetization and marketing strategies [186]. Within behavior modeling we identify four main player modeling subtasks that are particularly rel evant for game AI: imitation and prediction—achieved via supervised learning or reinforcement learning—and clustering and association mining—achieved via un supervised learning. The two main purposes of player imitation is the development of non-player characters with believable, human-like behavioral characteristics, and the understanding of human play per se through creating generative models of it. The prediction of aspects of player behavior, instead, may provide answers to ques tions such as "when will this player stop playing?" or "how often will that player get stuck in that area of the level?" or "which item type will this player pick in the next room?". The aim of clustering is the classification of player behaviors within a number of clusters depending of their behavioral attributes. Clustering is important for both the personalization of the game and the understanding of playing behavior in association with the game design [178]. Finally, association mining is useful in instances where frequent patterns or sequences of actions (or in-game events) are important for determining how a player behaves in a game.

While player behavior and player experience are interwoven notions there is a subtle difference between them. Player behavior points to what a player does in a game whereas player experience refers to how a player feels during play. The feeling of one's gameplay experience is clearly associated with what one does in the game; player experience, however, is primarily concerned with affective and cognitive aspects of play as opposed to mere reactions of gameplay which refer to player behavior.

Given the above aims, core tasks and sub-tasks of player modeling in the next section we discuss the various available options for constructing a player model.

5.3 A High-Level Taxonomy of Approaches

Irrespective of the application domain, computational models are characterized by three core components: the input the model will consider, the computational model per se, and the output of the model (see Fig. 5.1). The model itself is a mapping between the input and the output. The mapping is either hand-crafted or derived from data, or a mix of the two. In this section we will first go through the most

common approaches for constructing a computational model of players, then we will go through a taxonomy of possible inputs for a player model (Section 5.4) and finally we will examine aspects of player experience and behavior that a player model can represent as its output (Section 5.5).

A high-level classification of the available approaches for player modeling can be made between model-based (or top-down) and model-free (or bottom-up) ap proaches [782, 783]. The above definitions are inspired by the analogous classification in RL by which a world model is available (i.e., model-based) or not (i.e., model-free). Given the two ends of this continuum hybrid approaches between them can naturally exist. The gradient red color of the player model box in Fig. 5.1 illus trates the continuum between top-down and bottom-up approaches. The remainder of this section presents the key elements of and core differences among the various approaches for modeling of players.

5.3.1 Model-Based (Top-Down) Approaches

In a model-based or top-down [782] approach a player model is built on a theoret ical framework. As such, researchers follow the modus operandi of the humanities and social sciences, which hypothesize models to explain phenomena. Such hy potheses are usually followed by an empirical phase in which it is experimentally determined to what extent the hypothesized models fit observations; however, such a practice is not the norm within player experience research. While user experience has been studied extensively across several disciplines, in this book we identify three

main disciplines we can borrow theoretical frameworks from and build models of player experience: psychology and affective sciences, neuroscience, and finally, game studies and game research.

5.3.1.1 Psychology and Affective Sciences

Top-down approaches to player modeling may refer to models derived from popu lar theories about emotion [364] such as the cognitive appraisal theory [212, 601].

Further, the player model may rely on well established affect representations such as the emotional dimensions of arousal and valence [200] that define the circumplex model of affect of Russell [539] (see Fig. 5.2(a)). Valence refers to how pleasur able (positive) or unpleasurable (negative) the emotion is whereas arousal refers to how intense (active) or lethargic (inactive) that emotion is. Following a theoretical model, emotional manifestations of players are often mapped directly to specific player states. For instance, by viewing player experience as a psychophysiological phenomenon [779] a player's increased heart rate may correspond to high arousal and, in turn, to high levels of excitement or frustration.

Beyond established theories of emotion, model-based approaches can also be in spired by a general cognitive-behavioral theoretical framework such as the theory

of mind [540] for modeling aspects of social interactions in games. Popular ex ample frameworks for deriving user models in games include the usability theory [489, 290], the belief-desire-intention (BDI) model [66, 224], the cognitive model by Ortony, Clore, and Collins [512] and Skinner's behavioristic approach [633] with its links to reward systems in games. Further we can draw inspiration from social sciences and linguistics in order to model lexical aspects of gameplay interaction (e.g., chatting). Natural language processing, opinion mining and sentiment analysis are normally relying on theoretical models that build on affective and sociological aspects of textual communication [517, 514].

Of particular importance is the concept of flow by Csikszentmihalyi [151, 149, 150] which has been a popular psychological construct for modeling player experi ence in a top-down fashion. When in a state of flow (or else, state of "happiness") during an activity we tend to concentrate on the present moment, we lose our ability of reflective self-consciousness, we feel a sense of personal control over the sit uation or activity, our perception of time is altered, and we experience the activity as intrinsically rewarding. Analogously the optimal experience during play has been associated with a fine balance between boredom and anxiety, also known as the flow channel (see Fig. 5.2(b)). Given its direct relevance to player experience, flow has been adapted and incorporated for use in game design and for the understanding of player experience [678, 675, 473].

5.3.1.2 Neuroscience

A number of studies have relied on the working hypothesis of an underlying map ping between the brain, its neural activity and player experience. However, this re lationship is not well explored and the presumptive mapping is largely unknown.

For example, interest has been associated with activity in the visual cortex and the release of endomorphin whereas the sense of achievement has been linked to dopamine levels [35]. According to [35], neuroscientific evidence suggests that the reward systems of games are directly associated with the dopamine-based reward structures in the brain and that dopamine is released during gameplay [346]. Fur ther, pleasure has been associated with areas in the brain responsible for decision making, thereby revealing the direct links between gameplay experience and deci sion making [575]. Pleasure has also been associated with uncertain outcomes or uncertain rewards [625] as well as with interest and curiosity [43], which are all key elements of successful game design. Stress is also tightly coupled with player expe rience given its clear association with anxiety and fear; stress can be both monitored via physiology and regulated via game design. The testosterone levels of players have also been measured in association to digital game activities [444] and findings reveal particular patters of competition in games as testosterone factors. Finally, it

appears that trust between players in a social gaming setup could be measured indi rectly via oxytocin levels [350].
The degree to which current findings from neuroscience are applicable to player experience research is largely unknown since access to neural activity and brain hor-

(a) Russell's two-dimensional circumplex model of affect. The figure contains a small number of representative affective states (black circles).
(b) An illustration of the flow channel.
Fig. 5.2 Two popular frameworks used for modeling users and their experience in games: (a) the arousal-valence circumplex model of affect and (b) the flow channel concept.

mone levels remains a rather intrusive process at the time of writing. Manifestations of brain activity such as the brain's electrical waves—measured through electroen cephalography on our scalp—or more indirect manifestations such as stress and anxiety—measured through skin conductance—can give us access to approximates of brain activity. These approximates can be used for modeling the experience of play as discussed later in this chapter.

5.3.1.3 Game Studies and Game Research

Theoretical models of user experience in games are often driven by work in game studies and game research. Examples of models that have been used extensively in the literature include Malone's core design dimensions that collectively contribute to 'fun' games [419] defined as challenge, curiosity and fantasy. In particular, challenge refers to the uncertainty of achieving a goal due to e.g., variable difficulty level, multiple level goals, hidden information, and randomness. Curiosity refers to the player's feeling of uncertainty with respect to what will happen next. Finally, fantasy is the ability of the game to show (or evoke) situations or contexts that are not actually present. These three dimensions have been quantified, operationalized and successfully evaluated in prey-predator games [766], physical games [769, 775], preschooler games [320] and racing games [703].
Bartle's [33] classification of player types within games as a form of general player profiles can be used indirectly for modeling players. Bartle identifies four archetypes of players he names killers (i.e., players that focus on winning and are engaged by ranks and leaderboards), achievers (i.e., players that focus on achieving goals quickly and are engaged by achievements), socializers (i.e., players that focus on social aspects of games such as developing a network of friends) and explorers (i.e., players who focus on the exploration of the unknown). Various other method ologies have also been followed to derive specific player experience archetypes for particular classes of games [34, 787].
Other popular and interconnected views of player experience from a game design perspective include the theory of 'fun' by Koster [351], the notion of the 'magic circle' in games [587] and the four "fun" factor model of Lazzaro [365]. Indicatively, Koster's theory relates the concept of fun with learning in games: the more you learn the more you tend to play a game. According to his theory you stop playing a game that is way too easy (no learning of new skills) or way too hard (no learning either). Lazzaro's four fun factors are named hard fun (e.g., playing to win and see how good I am at it), easy fun (e.g., playing to explore new worlds and game spaces), serious fun (e.g., playing to feel better about myself or get better at something that matters to me) and people fun (e.g., playing as an excuse to invite friends over, or having fun merely by watching them play). Within game studies, the theoretical model of incorporation [94] is a notable multifaceted approach for capturing player immersion. The model is composed of six types of player involvement: affective, kinaesthetic, spatial, shared, ludic, and narrative.

With a careful analysis of the models proposed and their subcomponents one could coherently argue that there is one underlying theoretical model of player ex perience after all. While it is not the intention of this book to thoroughly discuss the

interconnections between the aforementioned models it is worth pointing out a num ber of indicative examples of our envisaged overarching player experience model.
An explorer (Bartle), for instance, can be associated with the easy fun factor of Laz zaro and the curiosity dimension of Malone. Further, the achiever archetype (Bartle) can be linked to the serious fun factor (Lazzaro). Accordingly, a killer archetype (Bartle) maps to the hard fun factor (Lazzaro), the challenge dimension of Malone's model, and a number of flow aspects. Finally, a socializer player profile (Bartle) could be associated to people fun (Lazzaro) and, in turn, to the shared involvement facet of Calleja [94].
Even though the literature on theoretical models of experience is rather rich, one needs to be cautious with the application of such theories to games (and game play ers) as the majority of the models have not been derived from or tested on interactive media such as games. Calleja [94], for instance, reflects on the inappropriateness of the concepts of 'fun' and 'magic circle' (among others) for games. At this point it is worth noting that while ad-hoc designed models can be an extremely powerful and expressive they need to be cross-validated empirically to be of practical use for computational player modeling; however, such practices are not as common within the broader area of game studies and game design.

5.3.2 Model-Free (Bottom-Up) Approaches

Model-free approaches refer to the data-driven construction of an unknown map ping (model) between a player input and a player state. Any manifestation of player affect or behavioral pattern could define the input of the model (see more in Section 5.4 below). A player state, on the other hand, is any representation of the player's experience or current emotional, cognitive, or behavioral state; this is es sentially the output of the computational model (see more in Section 5.5). Evidently, model-free approaches follow the modus operandi of the exact sciences, in which observations are collected and analyzed to generate models without a strong initial assumption on what the model looks like or even what it captures. Player data and labels of player states are collected and used to derive the model.
Classification, regression and preference learning techniques adopted from ma chine learning—see Chapter 2—or statistical approaches are commonly used for the construction of the mapping between the input and the output. Examples include studies in which player actions, goals and intentions are modeled and predicted for the purpose of believable gameplay, adaptive gameplay, interactive storytelling or even the improvement of a game's monetization strategy [511, 800, 414, 693, 592].
In contrast to supervised learning, reinforcement learning can be applied when a re ward function, instead, can characterize aspects of playing behavior or experience.
Unsupervised learning is applicable when target outputs are not available for pre-

dictive purposes but, alternatively, data is used for the analysis of playing behavior (see Fig. 5.1).
We meet bottom-up player modeling attempts since the early years of the game AI field in first-person shooters [695, 696], racing games [703] and variants of Pac Man (Namco, 1980) [776]. Recently, the availability of large sets of game and player data has opened up the horizons of behavioral data mining in games—i.e., game data mining [178]. Studies that attempt to identify different behavioral, playing and action patterns within a game are well summarized in [186] and include [36, 176, 687, 690, 750], among many others.

5.3.3 Hybrids

The space between a completely model-based and a completely model-free ap proach can be viewed as a continuum along which any player modeling approach might be placed. While a completely model-based approach relies solely on a the oretical framework that maps a player's responses to game stimuli, a completely model-free approach assumes there is an unknown function between modalities of user input and player states that a machine learner (or a statistical model) may dis cover, but does not assume anything about the structure of this function. Relative to these extremes, the vast majority of studies in player modeling may be viewed

as hybrids that synergistically combine elements of the two approaches. The con tinuum between top-down and bottom-up player modeling approaches is illustrated with a gradient color in Fig. 5.1.

5.4 What Is the Model's Input Like?

By now we have covered the various approaches available for modeling players and we will, in this section, focus on what the input of such a model might be like. The model's input can be of three main types: (1) anything that a player is doing in a game environment gathered from gameplay data—i.e., behavioral data of any type such as user interface selections, preferences, or in-game actions; (2) objective data collected as responses to game stimuli such as physiology, speech and body movements; and (3) the game context which comprises of any player agent interactions but also any type of game content viewed, played, and/or created.

The three input types are detailed in the remainder of this section. At the end of the section we also discuss static profile information on the player (such as personality) as well as web data beyond games that could feed and enhance the capacity of a player model.

5.4.1 Gameplay

Given that games may affect the player's cognitive processing patterns and cogni tive focus we assume that a player's actions and preferences are linked directly to her experience. Consequently, one may infer the player's current experience by an alyzing patterns of her interaction with the game, and by associating her experience with game context variables [132, 239]. Any element derived from the direct inter action between the player and the game can be classified as gameplay input. These interpretable measures of gameplay have also been defined as player metrics [186].

Player metrics include detailed attributes of the player's behavior derived from re sponses to game elements such as NPCs, game levels, user menus, or embodied conversational agents. Popular examples of data attributes include detailed spatial locations of players viewed as heatmaps [177], statistics on the use of in-game menus, as well as descriptive statistics about gameplay, and communication with other players. Figure 5.3 shows examples of heatmaps in the MiniDungeons1 puzzle game. Both general measures (such as performance and time spent on a task) and game-specific measures (such as the weapons selected in a shooter game [250]) are relevant and appropriate player metrics.

A major limitation with the gameplay input is that the actual player experience is only indirectly observed. For instance, a player who has little interaction with a game might be thoughtful and captivated, or just bored and busy doing something else. Gameplay metrics can only be used to approach the likelihood of the presence of certain player experiences. Such statistics may hold for player populations, but may provide little information for individual players. Therefore, when one attempts to use pure player metrics to make estimates of player experiences and make the game respond in an appropriate manner to these perceived experiences, it is advis able to keep track of the feedback of the player to the game responses, and adapt when the feedback indicates that the player experience was gauged incorrectly.

5.4.2 Objective

Computer game players are presented with a wide palette of affective stimuli dur ing game play. Those stimuli vary from simple auditory and visual events (such as sound effects and textures) to complex narrative structures, virtual cinemato graphic views of the game world and emotively expressive game agents. Player emotional responses may, in turn, cause changes in the player's physiology, reflect on the player's facial expression, posture and speech, and alter the player's attention and focus level. Monitoring such bodily alterations may assist in recognizing and constructing the player's model. As such, the objective approach to player model ing incorporates access to multiple modalities of player input.

1 http://minidungeons.com/

(a) In this example the player acts as a com pletionist: succeeding in killing all monsters,

drinking all potions and collecting all treasure.
(b) In this example the player prioritizes reach ing the exit, avoiding any monsters and only collecting potions and treasures that are near
the path to the exit.
Fig. 5.3 Two example heatmaps (human playtraces) in the MiniDungeons game. MiniDungeons is a simple turn-based rogue-like puzzle game, implemented as a benchmark problem for modeling the decision-making styles of human players [267].

The relationship between psychology and its physiological manifestations has been studied extensively ([17, 95, 779, 558] among many others). What is widely evidenced is that the sympathetic and the parasympathetic components of the au tonomic nervous system are involuntary affected by affective stimuli. In general, arousal-intense events cause dynamic changes in both nervous systems: an increase and a decrease of activity, respectively, in the sympathetic and the parasympathetic nervous system. Alternatively, activity at the parasympathetic nervous system is high during relaxing or resting states. As mentioned above, such nervous system activities cause alterations in one's facial expression, head pose, electrodermal ac tivity, heart rate variability, blood pressure, pupil dilation [91, 624] and so on.

Recent years have seen a significant volume of studies that explore the interplay between physiology and gameplay by investigating the impact of different game play stimuli on dissimilar physiological signals ([697, 473, 421, 420, 556, 721, 175, 451] among others). Such signals are usually obtained through electrocardiogra phy (ECG) [780], photoplethysmography [780, 721], galvanic skin response (GSR) [421, 271, 270, 272], respiration [721], electroencephalography (EEG) [493] and electromyography (EMG).

In addition to physiology one may track the player's bodily expressions (mo tion tracking) at different levels of detail and infer the real-time affective responses from the gameplay stimuli. The core assumption of such input modalities is that particular bodily expressions are linked to expressed emotions and cognitive pro cesses. Objective input modalities, beyond physiology, that have been explored ex-

tensively include facial expressions [321, 19, 236, 88, 794], muscle activation (typ ically face) [133, 164], body movement and posture [23, 731, 321, 172, 47], speech [741, 319, 308, 306, 30], text [517, 137, 391], haptics [509], gestures [283], brain waves [559, 13], and eye movement [23, 469].

While objective measurements can be a very informative way of assessing the player's state during the game a major limitation with most of them is that they can be invasive, thus affecting the player's experience with the game. In fact, some types of objective measures appear to be implausible within commercial-standard game development. Pupillometry and gaze tracking, for instance, are very sensitive to distance from screen, and variations in light and screen luminance, which collec tively make them rather impractical for use in a game application. The recent rebirth of virtual reality (VR), however, gives eye gaze sensing technologies entirely new opportunities and use within games [628]; a notable example of a VR headset that features eye-tracking is FOVE.2 Other visual cues obtained through a camera (facial expressions, body posture and eye movement) require a well-lit environment which is often not present in home settings (e.g., when playing video-games) and they can be seen by some players as privacy hazards (as the user is continuously recorded).

Even though highly unobtrusive, the majority of the vision-based affect-detection systems currently available have additional limitations when asked to operate in real-time [794]. We argue that an exception to this rule is body posture, which can both be effectively detected nowadays and provide us with meaningful estimates of player experience [343]. Aside from the potential they might have, however, the appropriateness of camera-based input modalities for games is questionable since experienced players tend to stay still while playing games [22].

As a response to the limitations of camera-based measurements, speech and text (e.g., chat) offer two highly accessible, real-time efficient and unobtrusive modali ties with great potential for gaming applications; however, they are only applicable

to games where speech (or text) forms a control modality (as e.g., in conversational games for children [320, 789]), collaborative games that naturally rely on speech or text for communication across players (e.g., in collaborative first-person shoot ers), or games that rely on natural language processing such as text-based adventure games or interactive fiction (see discussion of Chapter 4).

Within players' physiology, existing hardware for EEG, respiration and EMG require the placement of body parts such as the head, the chest or parts of the face on the sensors, making those physiological signals rather impractical and highly in trusive for most games. On the contrary, recent sensor technology advancements for the measurement of electrodermal activity (skin conductivity), photoplethysmogra phy (blood volume pulse), heart rate variability and skin temperature have made those physiological signals more attractive for the study of affect in games. Real time recordings of these can nowadays be obtained via comfortable wristbands and stored in a personal computer or a mobile device via a wireless connection [779].

At the moment of writing there are a few examples of commercial games that utilize physiological input from players. One particularly interesting example is

2 https://www.getfove.com/

Fig. 5.4 A screenshot from Nevermind (Flying Mollusk, 2015). The game supports several off the-shelf sensors that allow the audiovisual content of the game to adapt to the stress levels of the player. Image obtained from Erin Reynolds with permission.

Nevermind (Flying Mollusk, 2015), a biofeedback-enhanced adventure horror game that adapts to the player's stress levels by increasing the level of challenge it pro vides: the higher the stress the more the challenge for the player (see Fig. 5.4). A number of sensors which detect heart activity are available for affective interac tion with Nevermind. The Journey of Wild Divine (Wild Divine, 2001) is another biofeedback-based game designed to teach relaxation exercises via the player's blood volume pulse and skin conductance. It is also worth noting that AAA game developers such as Valve have experimented with the player's physiological input for the personalization of games such as Left 4 Dead (Valve, 2008) [14].

5.4.3 Game Context

In addition to gameplay and objective input, the game's context is a necessary input for player modeling. Game context refers to the momentanous state of the game during play and excludes any gameplay elements; those are already discussed in the gameplay input section. Clearly, our gameplay affects some aspects of the game context and vice versa but the two can be viewed as separate entities. Viewing this relationship from an analytics lens, the game context can be seen as a form of game metrics, opposed to gameplay which is a form of player metrics.

The importance of the game context for modeling players is obvious. In fact, we could argue that the context of the game during the interaction is a necessary input for detecting reliably any cognitive and affective responses of players. It could also

be argued that the game context is necessary as a guide during the annotation of the player experience; but more of that we will discuss in Section 5.5. The same way that we require the current social and cultural context to better detect the under lying emotional state of a particular facial expression of our discussant any player reactions cannot be dissociated from the stimulus (or the game context) that elicited them. Naturally, player states are always linked to game context. As a result, player models that do not take context into account run a risk of inferring erroneous states for the player. For example, an increase in galvanic skin response can be linked to different high-arousal affective states such as frustration and excitement. It is very hard to tell however, what the heightened galvanic skin response "means" without knowing what is happening in the game at the moment. In another example, a partic ular facial expression of the player, recorded though a camera, could be associated with either an achievement in the game or a challenging moment, and needs to be triangulated with the current game state to be understood. Evidently, such dualities of the underlying player state may be detrimental for the design of the player model.

While a few studies have investigated physiological reactions of players in isola tion, good practice in player modeling commands that any reactions of the play ers is triangulated with information about the current game state. For instance,
the model needs to know if the GSR increases because the player died or com pleted the level. The game context—naturally combined (or fused) with other in put modalities from the player—has been used extensively in the literature for the
prediction of different affective and cognitive states relevant to playing experience
[451, 434, 521, 617, 572, 133, 254, 558, 452, 433].

5.4.4 Player Profile

A player profile includes all the information about the player which is static and it
is not directly (nor necessarily) linked to gameplay. This may include information
on player personality (such as expressed by the Five Factor Model of personality
[140, 449]), culture dependent factors, and general demographics such as gender
and age. A player's profile may be used as input to the player model to comple ment the captured in-game behavior with general attributes about the player. Such
information may lead to more precise predictive models about players.
While gender, age [787, 686], nationality [46] and player expertise level [96]
have already proven important factors for profiling players the role of personality
remains somewhat contentious. On the one hand, the findings of van Lankveld et al.
[736, 737], for instance, reveal that gameplay behavior does not necessarily corre spond to a player's behavior beyond the game. On the other hand, Yee et al. have
identified strong correlations between player choices in World of Warcraft (Blizzard
Entertainment, 2004) and the personalities of its players [788]. Strong correlations
have also been found between the playing style and personality in the first-person
shooter Battlefield 3 (Electronic Arts, 2011) [687]. In general, we need to acknowl edge that there is no guaranteed one-to-one mapping between a player's in-game

behavior and personality, and that a player's personality profile does not necessarily
indicate what the player would prefer or like in a game [782].

5.4.5 Linked Data

Somewhere between the highly dynamic in-game behavior and the static profile in formation about the player we may also consider linked data retrieved from web
services that are not associated with gameplay per se. This data, for instance, may
include our social media posts, emoticons, emojis [199], tags used, places visited,
game reviews written, or any relevant semantic information extracted from diverse
Web content. The benefit of adding such information to player models is many fold but it has so far seen limited use in games [32]. In contrast to current player
modeling approaches the use of massive amounts and dissimilar types of content
across linked online sources would enable the design of player models which are
based on user information stored across various online datasets, thereby realizing
semantically-enriched game experiences. For example, both scores and sentiment analyzed textual reviews [517, 514] from game review sites such as Metacritic3 or
GameRankings4
can be used as input to a model. This model can then be used to
create game content which is expected to appeal to the specific parts of the com munity, based, for instance, on demographics, skill or interests collected from the
user's in-game achievements or favored games [585].

5.5 What Is the Model's Output Like?

The model's output, i.e., that which we wish to model, is usually a representation of
the player's state. In this section we explore three options for the output of the model
that serve different purposes in player modeling. If we wish to model the experi ence of the player the output is provided predominately through manual annotation.
If instead we wish to model aspects of player behavior the output is predominately
based on in-game actions (see Fig. 5.1). Finally, it may very well be that the model
has no output. Section 5.5.1 and Section 5.5.2 discuss the particularities of the out put, respectively, for the purpose of behavioral modeling and experience modeling
whereas Section 5.5.3 explores the condition where the model has no outputs.

3 http://www.metacritic.com
4 http://www.gamerankings.com/

5.5.1 Modeling Behavior
The task of modeling player behavior refers to the prediction or imitation of a partic ular behavioral state or a set of states. Note that if no target outputs are available then we are faced with either an unsupervised learning problem or a reinforcement learn ing problem which we discuss in Section 5.5.3. The output we must learn to predict (or imitate) in a supervised learning manner can be of two major types of game play data: either micro-actions or macro-actions (see Fig. 5.1). The first machine learning problem considers the moment-to-moment game state and player action space that are available at a frequency of frame rates. For example, we can learn to imitate the moves of a player on a frame-to-frame basis by comparing the play traces of an AI agent and a human as e.g., done for Super Mario Bros (Nintendo, 1985) [511, 469]. When macro-actions are considered instead, the target output is normally an aggregated feature of player behavior over time, or a behavioral pat tern. Examples of such outputs include game completion times, win rates, churn, trajectories, and game balance.
5.5.2 Modeling Experience
To model the experience of the player one needs to have access to labels of that experience. Those labels ideally need to be as close to the ground truth of expe rience as possible. The ground truth (or gold standard) in affective sciences refers to a hypothesized and unknown label, value, or function, that best characterizes and represents an affective construct or an experience. Labels are normally provided through manual annotation which is a rather laborious process. Manual annotation is however necessary given that we require some estimate of the ground truth for subjective notions such as the emotional states of the player. The accuracy of that estimation is regularly questioned as there are numerous factors contributing to a deviation between a label and the actual underlying player experience.
Manually annotating players and their gameplay is a challenge in its own right with respect to both the human annotators involved and the annotation protocol cho sen [455, 777]. On one hand, the annotators need to be skilled enough to be able to approximate the actual experience well. On the other hand, there are still many open questions left for us to address when it comes to the annotation tools and protocols used. Such questions include: Who will do the labeling: the person experiencing the gameplay or others? Will the labeling of player experience involve states (dis crete representation) or instead involve the use of intensity or experience dimensions (continuous representation)? When it comes to time, should it be done in real-time or offline, in discrete time periods or continuously? Should the annotators be asked to rate the affect in an absolute fashion or, instead, rank it in a relative fashion? Answers to the above questions yield different data annotation protocols and, in evitably, varying degrees of data quality, validity and reliability. In the following

sections we attempt to address a number of such critical questions that are usually raised in subjective annotations of player states.
5.5.2.1 Free Response or Forced Response?
Subjective player state annotations can be based either on a player'sfree response—retrieved via e.g., a think-aloud protocol [555]—or on forced responses retrieved through questionnaires or annotation tools. Free response naturally contains richer information about the player's state, but it is often unstructured, even chaotic, and thus hard to analyze appropriately. On the other hand, forcing players to self-report their experiences using directed questions or tasks constrains them to specific ques tionnaire items which could vary from simple tick boxes to multiple choice items.
Both the questions and the answers we provide to annotators may vary from single words to sentences. Questionnaires can contain elements of player experience (e.g., the Game Experience Questionnaire [286]), demographic data and/or personality traits (e.g., a validated psychological profiling questionnaire such as the NEO-PI-R

[140]). In the remainder of this section we will focus on forced responses as these are easier to analyze and are far more appropriate for data analysis and player mod eling (as defined in this book).

5.5.2.2 Who Annotates?

Given the subjective nature of player experience the first natural question that comes in mind is who annotates players? In other words, who has the right authority and the best capacity to provide us with reliable tags of player experience? We dis tinguish two main categories: annotations can either be self-reports or reports ex pressed indirectly by experts or external observers [783].

In the first category the player states are provided by the players themselves and we call that first-person annotation. For example, a player is asked to rate the level of engagement while watching her playthrough video. First-person is clearly the most direct way to annotate a player state and build a model based on the solicited annotations. We can only assume there is disparity between the true (inner) expe rience of each player and the experience as felt by herself or perceived by others.

Based on this assumption the player's annotations should normally be closer to her inner experience (ground truth) compared to third-person annotation. First-person annotation, however, may suffer from self-deception and memory limitations [778]. These limitations have been attributed mainly to the discrepancies between "the ex periencing self" and "the remembering self" of a person [318] which is also known as the memory-experience gap [462].

Expert annotators—as a response to the above limitations—may instead be able to surpass the perception of experience and reach out to the inner experience of the player. In this second annotation category, named third-person annotation, an expert—such as a game designer—or an external observer provides the player state

in a more objective manner, thereby reducing the subjective biases of first-person perceptions. For instance, a user experience analyst may provide particular player state tags while observing a first-person shooter deathmatch game. The benefit of third-person annotation is that multiple annotators can be used for a particular game play experience. In fact, the availability of several such subjective perceptions of experience may allow us to approximate the ground truth better as the agreement between many annotators enhances the validity of our data directly. A potential disagreement, on the other hand, might suggest that the gameplay experience we examine is non-trivial or may indicate that some of our annotators are untrained or inexperienced.

5.5.2.3 How Is Player Experience Represented?

Another key question is how player experience is best represented: as a number of different states (discrete) or, alternatively, as a set of dimensions (continuous)? On one hand, discrete labeling is practical as a means of representing player experience since the labels can easily form individual items (e.g., "excited", "annoyed" etc.) in an experience questionnaire, making it easy to ask the annotator/player to pick one (e.g., in [621]). Continuous labeling, on the other hand, appears to be advantageous for two key reasons. First, experiential states such as immersion are hard to capture with words or linguistic expressions that have fuzzy boundaries. Second, states do not allow for variations in experience intensity over time since they are binary: ei ther the state is present or not. For example, the complex notions of fun, or even engagement, cannot be easily captured by their corresponding linguistic represen tation in a questionnaire or define well a particular state of a playing experience.

Instead it seems natural to represent them as a continuum of experience intensity that may vary over time. For these reasons we often observe low agreement among the annotators [143] when we represent playing experience via discrete states.

As discussed earlier, the dominant approach in continuous annotation is the use of Russell's two-dimensional (arousal-valence) circumplex model of affect [581] (see Fig. 5.2(a)). Figure 5.5 illustrates two different annotation tools (FeelTrace and AffectRank) that are based on the arousal-valence circumplex model of affect. Figure 5.6 depicts the RankTrace continuous annotation tool which can be used for the

annota

Annotation can happen either within particular time intervals or continuously. Time continuous annotation has been popularized due to the existence of freely available tools such as FeelTrace [144] (see Fig. 5.5(c)) and GTrace [145], which allows for continuous annotation of content (mostly videos and speech) across the dimensions of arousal and valence. In addition to FeelTrace there are annotation tools like the
5.5. What Is the Model's Output Like? 223
continuous measurement system [454] and EmuJoy [474], where the latter is de signed for the annotation of music content. User interfaces such as wheels and knobs linked to the above annotation tools show further promise for the continu ous annotation of experience in games [125, 397, 97] (see Fig. 5.6). The continuous annotation process, however, appears to require a higher amount of cognitive load compared to a time-discrete annotation protocol. Higher cognitive loads often result in lower levels of agreement between different annotators and yield unreliable data for modeling player experience [166, 418].

As a response to the above limitations, time-discrete annotation provides data at particular intervals when the annotator feels there is a change in the player's state. And changes are best indicated relatively rather than absolutely. AffectRank, for instance (see Fig. 5.5(b)), is a discrete, rank-based annotation tool that can be used for the annotation of any type of content including images, video, text or speech and it provides annotations that are significantly more reliable (with respect to inter-rater agreement) than the annotations obtained from continuous annotation tools such as FeelTrace [777]. The rank-based design of AffectRank is motivated by observations of recent studies in third-person video annotation indicating that ". . . humans are better at rating emotions in relative rather than absolute terms." [455, 777]. Further, AffectRank is grounded in numerous findings showcasing the supremacy of ranks over ratings for obtaining annotations of lower inconsistency and order effects [777, 773, 778, 436, 455, 761].

A recent tool that builds on the relative-based annotation of AffectRank and al lows for the annotation of affect in a continuous yet unbounded fashion is RankTrace (see Fig. 5.6). The core idea behind RankTrace is introduced in [125]: the tool asks participants to watch the recorded playthrough of a play session and annotate in real-time the perceived intensity of a single emotional dimension. The annotation process in RankTrace is controlled through a "wheel-like" hardware, allowing par ticipants to meticulously increase or decrease emotional intensity by turning the wheel, similarly to how volume is controlled in a stereo system. Further, the general interfacing design of RankTrace builds on the one-dimensional GTrace annotation tool [145]. Unlike other continuous annotation tools, however, annotation in Rank Trace is unbounded: participants can continuously increase or decrease the intensity as desired without constraining themselves to an absolute scale. This design deci sion is built on the anchor [607] and adaptation level [258] psychology theories by which affect is a temporal notion based on earlier experience that is best expressed in relative terms [765, 777, 397]. The use of RankTrace has revealed the benefits of relative and unbounded annotation for modeling affect more reliably [397] and has also showed promise for the construction of general models of player emotion across games [97].

5.5.2.5 When to Annotate?

When is it best to annotate experience: before, during or after play (see Fig. 5.1)? In a pre-experience questionnaire we usually ask annotators to set the baseline of
224 Chapter 5. Modeling Players
(a) Annotating facial expressions of players during play. The annotation is context-dependent as the video includes the gameplay of the player (see top left corner).
(b) AffectRank: A time-discrete annotation tool for arousal and valence.
(c) FeelTrace. A time-continuous annotation tool for arousal and valence.

Fig. 5.5 An example of third-person annotation based on videos of players and their gameplay using either (a) the AffectRank (b) or the FeelTrace (c) annotation tool. AffectRank is freely avail able at: https://github.com/TAPeri/AffectRank. FeelTrace is freely available at: http://emotion research.net/download/Feeltrace%20Package.zip.

a player's state prior to playing a game. This state can be influenced by a number of factors such as the mood of the day, the social network activity, the caffeine consumption, earlier playing activity and so on. This is a wealth of information that can be used to enrich our models. Again, what is worth detecting is the relative change [765] from the baseline state of the user prior playing to the game.

5.5. What Is the Model's Output Like? 225

Annotation Timeline Controllable

Reference

Video

Playback

Fig. 5.6 The RankTrace annotation tool. In this example the tool is used for the annotation of tension in horror games. Participants play a game and then they annotate the level of tension by watching a video-recorded playthrough of their game session (top of image). The annotation trace is controlled via a wheel-like user interface. The entire annotation trace is shown for the partici pant's own reference (bottom of image). Image adapted from [397]. The RankTrace tool is available at: http://www.autogamedesign.eu/software.

A during-experience protocol, on the other hand, may involve the player in a first-person think-aloud setup [555] or a third-person annotation design. For the latter protocol you may think of user experience experts that observe and annotate player experience during the beta release of a game, for example. As mentioned earlier, first-person annotation during play is a rather intrusive process that disrupts the gameplay and risks adding experimental noise to annotation data. In contrast, third-person annotation is not intrusive; however, there are expected deviations from the actual first-person experience, which is inaccessible to the observer.

The most popular approach for the annotation of player experience is after a game (or a series of games) has been played, in a post-experience fashion. Post-experience annotation is unobtrusive for the player and it is usually performed by the players themselves. Self-reports, however, are memory-dependent by nature and memory is, in turn, time-dependent. Thus, one needs to consider carefully the time window be tween the experience and its corresponding report. For the reported post-experience to be a good approximation of the actual experience the playing time window needs to be small in order to minimize memory biases, yet sufficiently large to elicit par ticular experiences to the player. The higher the cognitive load required to retrieve the gameplay context is, the more the reports are memory-biased and not relevant to the actual experience. Further, the longer the time window between the real experi ence and the self-report the more the annotator activates aspects of episodic memory associated with the gameplay [571]. Episodic memory traces that form the basis of

self-reports fade over time, but the precise rate at which this memory decay occurs is unknown and most likely individual [571]. Ideally, memory decay is so slow that the annotator will have a clear feeling of the gameplay session when annotating it. Now, if the time window becomes substantial—on the scale of hours and days—the annotator has to activate aspects of semantic memory such as general beliefs about a game. In summary, the more the episodic memory, and even more so the seman tic memory, are activated during annotation, the more systematic errors are induced within the annotation data.

As a general rule of thumb the longer it takes for us to evaluate an experience of ours the larger the discrepancy between the true experience and the evaluation of the experience, which is usually more intense than the true experience. It also seems that this gap between our memory of experience and our real experience is more prominent when we report unpleasant emotions such as anger, sadness and tension rather than positive emotions [462]. Another bias that affects how we report our experience is the experience felt near the end of a session, a game level or a game;

this effect has been named peak-end rule [462].
An effective way to assist episodic memory and minimize post-experience cogni tive load is to show annotators replay videos of their gameplay (or the gameplay of others) and ask them to annotate those. This can be achieved via crowdsourcing [96] in a third-person manner or on a first-person annotation basis [125, 271, 397, 97]. Another notable approach in this direction is the data-driven retrospective inter viewing method [187]. According to that method player behavioral data is collected and is analyzed to drive the construction of interview questions. These questions are then used in retrospect (post-experience) to reflect on the annotator's behavior.

5.5.2.6 Which Annotation Type?

We often are uncertain about the type of labels we wish to assign to a player state or a player experience. In particular, we can select from three data types for our annota tion: ratings, classes, and ranks (see Fig. 5.1). The rating-based format represents a player's state with a scalar value or a vector of values. Ratings are arguably the dom inant practice for quantitatively assessing aspects of a user's behavior, experience, opinion or emotion. In fact, the vast majority of user and psychometric studies have adopted rating questionnaires to capture the opinions, preferences and perceived ex periences of experiment participants—see [78, 442, 119] among many. The most popular rating-based questionnaire follows the principles of a Likert scale [384] in which users are asked to specify their level of agreement with (or disagreement against) a given statement—see Fig. 5.7(a) for an example. Other popular rating based questionnaires for user and player experience annotation include the Geneva Wheel model [600], the Self-Assessment Manikin [468], the Positive and Negative Affect Schedule [646], the Game Experience Questionnaire [286], the Flow State Scale [293] and the Player Experience of Need Satisfaction (PENS) survey [583], which was developed based on self-determination theory [162].

Rating-based reporting has notable inherent limitations that are often over looked, resulting in fundamentally flawed analyses [778, 298]. First, ratings are analyzed traditionally by comparing their values across participants; see [233, 427] among many. While this is a generally accepted and dominant practice it neglects the existence of inter-personal differences as the meaning of each level on a rat ing scale may differ across experiment participants. For example, two participants assessing the difficulty of a level may assess it as exactly the same difficulty, but then one rates it as "very easy to play" and the other as "extremely easy to play". It turns out that there are numerous factors that contribute to the different internal rating scales existent across participants [455] such as differences in personality, culture [643], temperament and interests [740]. Further, a large volume of studies has also identified the presence of primacy and recency order effects in rating-based questionnaires (e.g., [113, 773]), systematic biases towards parts of the scale [388] (e.g., right-handed participants may tend to use the right side of the scale) or a fixed tendency over time (e.g., on a series of experimental conditions, the last ones are rated higher). Indicatively, the comparative study of [773] between ratings and ranks showcases higher inconsistency effects and significant order (recency) effects existent in ratings.

In addition to inter-personal differences, a critical limitation arises when ratings are treated as interval values since ratings are by nature ordinal values [657, 298]. Strictly speaking, any approach or method that treats ratings as numbers by, for instance, averaging their ordinal labels is fundamentally flawed. In most question naires Likert items are represented as pictures (e.g., different representations of arousal in the Self-Assessment Manikin [468]) or as adjectives (e.g., "moderately", "fairly" and "extremely"). These labels (images or adjectives) are often erroneously converted to integer numbers, violating basic axioms of statistics which suggest that ordinal values cannot be treated as interval values [657] since the underlying nu merical scale is unknown. Note that even when a questionnaire features ratings as numbers (e.g., see Fig. 5.7(a)), the scale is still ordinal as the numbers in the in strument represent labels. Thus, the underlying numerical scale is still unknown and

dependent on the participant [657, 515, 361]. Treating ratings as interval values is grounded in the assumption that the difference between consecutive ratings is fixed and equal. However, there is no valid assumption suggesting that a subjective rating scale is linear [298]. For instance, the difference between “fairly (4) ” and “ex tremely (5)” may be larger than the distance between “moderately (3)” and “fairly (4)” as some experiment participants rarely use the extremes of the scale or tend to use one extreme more than the other [361]. If, instead, ratings are treated naturally as ordinal data no assumptions are made about the distance between rating labels, which eliminates introducing data noise to the analysis.

The second data type for the annotation of players is the class-based format. Classes allow annotators to select from a finite and non-structured set of options and, thus, a class-based questionnaire provides nominal data among two (binary) or more options. The questionnaire asks subjects to pick a player state from a particu lar representation which could vary from a simple boolean question (was that game level frustrating or not? is this a sad facial expression? which level was the most

stressful?) to a player state selection from, for instance, the circumplex model of affect (is this a high- or a low-arousal game state for the player?). The limitations of ratings are mitigated, in part, via the use of class-based questionnaires. By not providing information about the intensity of each player state, however, classes do not have the level of granularity ratings naturally have. A class-based questionnaire might also yield annotations with an unbalanced number of samples per class. A common practice in psychometrics consists of transforming sets of consecutive rat ings into separate classes (e.g., see [226, 260] among many). In an example study [255], arousal ratings on a 7-point scale are transformed into high, neutral and low arousal classes using 7-5, 4 and 3-1 ratings, respectively. While doing so might seem appropriate, the ordinal relation among classes is not being taken into account. More importantly, the transformation process adds a new set of bias to the subjectivity bias of ratings, namely class splitting criteria [436].

Finally, rank-based questionnaires ask the annotator to rank a preference among options such as two or more sessions of the game [763]. In its simplest form, the annotator compares two options and specifies which one is preferred under a given statement (pairwise preference). With more than two options, the participants are asked to provide a ranking of some or all the options. Examples of rank-based ques tions include: was that level more engaging than this level? which facial expres sion looks happier?). Another example of a rank-based questionnaire (4-alternative forced choice) is illustrated in Fig. 5.7(b). Being a form of subjective reporting, rank-based questionnaires (as much as rating-based and class-based questionnaires) are associated with the well known limitations of memory biases and self-deception. Reporting about subjective constructs such as experience, preference or emotion via rank-based questionnaires, however, has recently attracted the interest of researchers in marketing [167], psychology [72], user modeling [761, 37] and affective com puting [765, 721, 436, 455, 773] among other fields. This gradual paradigm shift is driven by both the reported benefits of ranks minimizing the effects of self-reporting subjectivity biases and recent findings demonstrating the advantages of ordinal an notation [765, 773, 455].

5.5.2.7 What Is the Value of Player Experience?

Describing, labeling and assigning values to subjective notions, such as player ex perience, is a non-trivial task as evidenced by a number of disciplines including neuroscience [607], psychology [258], economics [630], and artificial intelligence [315]. Annotators can attempt to assign numbers to such notions in an absolute manner, using for instance a rating scale. Annotators can alternatively assign val ues in a relative fashion, using for instance a ranking. There are, however, a mul titude of theoretical and practical reasons to doubt that subjective notions can be encoded as numbers in the first place [765]. For instance, according to Kahneman [317], co-founder of behavioral economics, “...it is safe to assume that changes are more accessible than absolute values”; his theory about judgment heuristics is built

on Herbert Simon's psychology of bounded rationality [630]. Further, an important
5.5. What Is the Model's Output Like? 229
(a) Rating: A 5-point Likert item example (b) Rank: A 4-alternative forced choice exam ple
Fig. 5.7 Examples of rating-based (a) vs. rank-based (b) questionnaires.
thesis in psychology, named adaptation level theory [258], suggests that humans
lack the ability to maintain a constant value about subjective notions and their pref erences about
options are, instead, made on a pairwise comparison basis using an
internal ordinal scale [460]. The thesis claims that while we are efficient at discrimi nating among
options, we are not good at assigning accurate absolute values for the
intensity of what we perceive. For example, we are particularly bad at assigning ab solute values to
tension, frequency and loudness of sounds, the brightness of an im age, or the arousal level of a video.
The above theories have also been supported by
neuroscientific evidence suggesting that experience with stimuli gradually creates
our own internal context, or anchor [607], against which we rank any forthcoming
stimulus or perceived experience. Thus, our choice about an option is driven by our
internal ordinal representation of that particular option within a sample of options;
not by any absolute value of that option [658].
As a remote observation, one may argue that the relative assessment provides
less information than the absolute assessment since it does not express a quantity
explicitly and only provides ordinal relations. As argued earlier, however, any addi tional information
obtained in an absolute fashion (e.g., when ratings are treated as
numbers) violates basic axioms of applied statistics. Thus the value of the additional
information obtained (if any) is questioned directly [765].
In summary, results across different domains investigating subjective assessment
suggest that relative (rank-based) annotations minimize the assumptions made about
experiment participants' notions of highly subjective constructs such as player ex perience. Further,
annotating experience in a relative fashion, instead of an absolute
fashion, leads to the construction of more generalizable and accurate computational
models of experience [765, 436].

5.5.3 No Output

Very often we are faced with datasets where target outputs about player behavioral
or experience states are not available. In such instances modeling of players must
rely on unsupervised learning [176, 244, 178] (see Fig. 5.1). Unsupervised learning,
230 Chapter 5. Modeling Players
as discussed in Chapter 2, focuses on fitting a model to observations by discovering
associations of the input and without having access to a target output. The input is
generally treated as a set of random variables and a model is built through the obser vations of
associations among the input vectors. Unsupervised learning as applied to
modeling players involves tasks such as clustering and association mining which
are described in Section 5.6.3.
It may also be the case that we do not have target outputs available but, neverthe less, we can design
a reward function that characterizes behavioral or experiential
patterns of play. In such instances we can use reinforcement learning approaches
to discover policies about player behavior or player experience based on in-game
play traces or other state-action representations (see Section 5.6.2). In the following
section we detail the approaches used for modeling players in a supervised learning,
reinforcement learning and unsupervised learning fashion.

5.6 How Can We Model Players?

In this section we build upon the data-driven approach of player modeling and dis cuss the application
of supervised, reinforcement and unsupervised learning to
model players, their behavior and their experience. To showcase the difference be tween the three
learning approaches let us suppose we wish to classify player be havior. We can only use
unsupervised learning if no behavioral classes have been
defined a priori [176]. We can instead use supervised learning if, for example, we
have already obtained an initial classification of players (either manually or via clus tering) and we
wish to fit new players into these predefined classes [178]. Finally,
we can use reinforcement learning to derive policies that imitate different types

of playing behavior or style. In Section 5.6.1 we focus on the supervised learning paradigm whereas in Section 5.6.2 and Section 5.6.3 we outline, respectively, the re inforcement learning and the unsupervised learning approach for modeling players. All three machine learning approaches are discussed in Chapter 2.

5.6.1 Supervised Learning

Player modeling consists of finding a function that maps a set of measurable at tributes of the player to a particular player state. Following the supervised learning approach this is achieved by machine learning, or automatically adjusting, the pa rameters of a model to fit a dataset that contains a set of input samples, each one paired with target outputs. The input samples correspond to the list of measurable attributes (or features) while the target outputs correspond to the annotations of the player's states for each of the input samples that we are interested to learn to pre dict. As mentioned already, the annotations may vary from behavioral characteris-

tics, such as completion times of a level or player archetypes, to estimates of player experience, such as player frustration.

As we saw in Chapter 2 popular supervised learning techniques, including artificial neural networks (shallow or deep architectures), decision trees, and support vector machines, can be used in games for the analysis, the imitation and the pre diction of player behavior, and the modeling of playing experience. The data type of the annotation determines the output of the model and, in turn, the type of the machine learning approach that can be applied. The three supervised learning alter natives for learning from numerical (or interval), nominal and ordinal annotations—respectively, regression, classification and preference learning—are discussed in this section.

5.6.1.1 Regression

When the outputs that a player model needs to approximate are interval values, the modeling problem is known as metric or standard regression. Any regression algo rithm is applicable to the task, including linear or polynomial regression, artificial neural networks and support vector machines. We refer the reader to Chapter 2 for details on a number of popular regression algorithms.

Regression algorithms are appropriate for imitation and prediction tasks of player behavior. When the task, however, is modeling of player experience cau tion needs to be put on the data analysis. While it is possible, for instance, to use regression algorithms to learn the exact numeric ratings of experience, in general it should be avoided because regression methods assume that the target values follow an interval (numerical) scale. Ratings naturally define an ordinal scale [765, 773] in stead. As mentioned already, ordinal scales such as ratings should not be converted to numerical values due to the subjectivity inherent to reports, which imposes a non-uniform and varying distance among questionnaire items [778, 657]. Prediction models trained to approximate a real-value representation of a rating—even though they may achieve high prediction accuracies—do not necessarily capture the true reported playing experience because the ground truth used for training and valida tion of the model has been undermined by the numerous effects discussed above.

We argue that the known fundamental pitfalls of self-reporting outlined above pro vide sufficient evidence against the use of regression for player experience modeling [765, 515, 455, 361]. Thus, we leave the evaluation of regression methods on expe rience annotations outside the scope of this book.

5.6.1.2 Classification

Classification is the appropriate form of supervised learning for player modeling when the annotation values represent a finite and non-structured set of classes. Classification methods can infer a mapping between those classes and player at tributes. Available algorithms include artificial neural networks, decision trees, ran-

dom forests, support vector machines, K-nearest neighbors, and ensemble learning among many others. Further details about some of these algorithms can be found in Chapter 2.

Classes can represent playing behavior which needs to be imitated or predicted, such as completion times (e.g., expressed as low, average or high completion time) or user retention in a free-to-play game (e.g., expressed as weak, mild or strong retention). Classes can alternatively represent player experience such as an excited versus a frustrated player as manifested from facial expressions or low, neutral and high arousal states for a player.

Classification is perfectly suited for the task of modeling player experience if discrete annotations of experience are selected from a list of possibilities and pro vided as target outputs [153, 344]. In other words, annotations of player experience need to be nominal for classification to be applied. A common practice, however, as already mentioned in Section 5.5.2.6, is to treat ratings of experience as classes and transform the ordinal scale—that defines ratings—into a nominal scale of separate classes. For example, ratings of arousal that lie between −1 and 1 are transformed into low, neutral and high arousal classes. By classifying ratings not only the ordinal relation among the introduced classes is ignored but, most importantly, the transfor mation process induces several biases to the data (see Section 5.5.2.6). These biases appear to be detrimental and mislead the search towards the ground truth of player experience [765, 436].

5.6.1.3 Preference Learning

As an alternative to regression and classification methods, preference learning [215] methods are designed to learn from ordinal data such as ranks or preferences. It is important to note that the training signal in the preference learning paradigm merely provides information for the relative relation between instances of the phe nomenon we attempt to approximate. Target outputs that follow an ordinal scale do not provide information about the intensity (regression) or the clusters (classifica tion) of the phenomenon.

Generally we could construct a player model based on in-game behavioral pref erences. The information that this player, for example, prefers the mini-gun over a number of other weapons could form a set of pairwise preferences we could learn from. Alternatively we can build a model based on experience preferences. A player, for instance, reported that area X of the level is more challenging than area Y of the same level. Based on a set of such pairwise preferences we can derive a global func tion of challenge for that player.

As outlined in Chapter 2 a large palette of algorithms is available for the task of preference learning. Many popular classification and regression techniques have been adapted to tackle preference learning tasks, including linear statistical models such as linear discriminant analysis and large margins, and non-linear approaches such as Gaussian processes [122], deep and shallow artificial neural networks [430], and support vector machines [302].

Preference learning has already been extensively applied to modeling aspects of players. For example, Mart´ınez et al. [430, 431] and Yannakakis et al. [780, 771] have explored several artificial neural network approaches to learn to predict af fective and cognitive states of players reported as pairwise preferences. Similarly, Garbarino et al. [217] have used linear discriminant analysis to learn pairwise en joyment predictors in racing games. To facilitate the use of proper machine learning methods on preference learning problems, a number of such preference learning methods as well as data preprocessing and feature selection algorithms have been made available as part of the Preference Learning Toolbox (PLT) [198]. PLT is an open-access, user-friendly and accessible toolkit5 built and constantly updated for the purpose of easing the processing of (and promoting the use of) ranks.

As ratings, by definition, express ordinal scales they can directly be transposed to any ordinal representation (e.g., pairwise preferences). For instance, given an anno tator's rating indicating that a condition A felt 'slightly frustrating' and a condition B felt 'very frustrating', a preference learning method can train a model that predicts a higher level of frustration for B than for A. In this way the modeling approach avoids introducing artifacts of what is the actual difference between 'very' and 'slightly' or

the usage of the scale for this particular annotator. Further, the limitation of different subjective scales across users can be safely bypassed by transforming rating reports into ordinal relations on a per-annotator basis. Finally, the problem of the scale varying across time due to episodic memory still persists but can be minimized by transforming only consecutive reports, i.e., given a report for three conditions A, B and C, the player model can be trained using only the relation between A and B, and B and C (dropping the comparison between A and C).

5.6.1.4 Summary: The Good, the Bad and the Ugly

The last section on supervised learning is dedicated to the comparison among the three methods—regression, classification and preference learning—for modeling players. Arguably the discussion is limited when the in-game behavior of players is imitated or predicted. If behavioral data about players follows interval, nomi nal or ordinal scales then naturally, regression, classification and preference learn ing should be applied, respectively. Behavioral data have an objective nature which makes the task of learning less challenging. Given the subjective notion of player ex perience, however, there are a number of caveats and limitations of each algorithm that need to be taken into account. Below we discuss the comparative advantages of each and we summarize the key outcomes of supervised learning as applied to modeling the experience of players.

Regression vs. Preference Learning: Motivated by psychological studies sug gesting that interval ratings misrepresent experience [515, 455, 361], we will not dedicate ourselves an extensive comparison between preference learning and regres sion methods. The performance comparison between a regression and preference-

5 http://sourceforge.net/projects/pl-toolbox/

learned model is also irrelevant as the former is arguably a priori incapable of cap turing the underlying experience phenomenon as precisely as the latter. Such de viations from the ground truth, however, are not trivial to illustrate through a data modeling approach and thus the comparison is not straightforward. The main rea son is that the objective ground truth is fundamentally ill-defined when numbers are used to characterize subjective notions such as player experience.

Regression vs. Classification: Classes are easy to analyze and create player models from. Further, their use eliminates part of the inter-personal biases intro duced with ratings. For these reasons classification should be preferred to regres sion for player experience modeling. We already saw that classification, instead of regression, is applied when ratings are available to overcome part of the limitations inherent in rating-based annotation. For instance, this can be achieved by transform ing arousal ratings to high, neutral and low arousal classes [255]. While this com mon practice in psychometrics eliminates part of the rating subjectivity it adds new forms of data biases inherent in the ad-hoc decisions to split the classes. Further, the analysis of player models across several case studies in the literature has already shown that transforming ratings into classes creates a more complicated machine learning problem [765, 436].

Classification vs. Preference Learning: Preference learning is the supreme method for modeling experience when ranks or pairwise preferences are available. Even when ratings or classes are available comparisons between classification and preference learning player models in the literature suggest that preference learn ing methods lead to more efficient, generic and robust models which capture more information about the ground truth [765]. Indicatively, Crammer and Signer [147] compare classification, regression and preference learning training algorithms in a task to learn ratings. They report the supremacy of preference learning over the other methods based on several synthetic datasets and a movie-ratings dataset. In addition, extensive evidence already shows that preference learning better approxi mates the underlying function between input (e.g., experience manifestations such as gameplay) and output (e.g., annotations) [436]. Figure 5.8 showcases how much closer a preference learned model can reach a hypothesized (artificial) ground truth,

compared to a classification model trained on an artificial dataset. In summary, pref erence learning via rank-based annotation controls for reporting memory effects,
eliminates subjectivity biases and builds models that are closer to the ground truth of player experience [778, 777].
Grounded in extensive evidence our final note for the selection of a supervised learning approach for modeling player experience is clear: Independently of how experience is annotated we argue that preference learning (the good) is a superior supervised learning method for the task at hand, classification (the ugly) provides a good balance between simplicity and approximation of the ground truth of player experience whereas regression (the bad) is based on rating annotations which are of questionable quality with respect to their relevance to the true experience.

(a) Ground Truth
(b) Classification (c) Preference learning
Fig. 5.8 A hypothesized (artificial) ground truth function (z-axis) which is dependent on two player attributes, x1 and x2 (Fig. 5.8(a)), the best classification model (Fig. 5.8(b)) and the best preference learned model (Fig. 5.8(c)). All images are retrieved from [436].

5.6.2 Reinforcement Learning

While it is possible to use reinforcement learning to model aspects of users dur ing their interaction the RL approach for modeling players has been tried mostly in
comparatively simplistic and abstract games [195] and has not seen much applica-

tion in computer games. The key motivation for the use of RL for modeling players is that it can capture the relative valuation of game states as encoded internally by humans during play [685]. At first glance, player modeling with RL may seem to be an application of RL for game playing, and we discuss this in Chapter 3 as part of the play for experience aim. In this section, we instead discuss this approach from the perspective that a policy learned via RL can capture internal player states with no corresponding absolute target values such as decision making, learnability, cog nitive skills or emotive patterns. Further, those policies can be trained on player data
such as play traces. The derived player model depicts psychometrically-valid, ab stract simulations of a human player's internal cognitive or affective processes. The
model can be used directly to interpret human play, or indirectly, it can be featured in AI agents which can be used as playtesting bots during the game design process, as baselines for adapting agents to mimic classes of human players, or as believable, human-like opponents [268].
Using the RL paradigm, we can construct player models via RL if a reward sig nal can adjust a set of parameters that characterize the player. The reward signal
can be based either directly on the in-game behavior of the player—for instance, the decision taken at a particular game state—or indirectly on player annotations (e.g., annotated excitement throughout the level) or objective data (e.g., physiological in dexes showing player stress). In other words, the immediate reward function can be
based on gameplay data if the model wishes to predict the behavior of the player or, instead, be based on any objective measure or subjective report if the model attempts to predict the experience of the player. The representation of the RL approach can be anything from a standard Q table that, for instance, models the decision making behavior of a player (e.g., as in [268, 685]) to an ANN that e.g., models in-game behavior (as in [267]), to a set of behavior scripts [650] that are adjusted to imitate gameplay behavior via RL, to a deep Q network.
We can view two ways of constructing models of players via RL: models can be built offline (i.e., before gameplay starts) or at runtime (i.e., during play). We can also envision hybrid approaches by which models are first built offline and then are polished at runtime. Offline RL-based modeling adds value to our playtesting capacity via, for instance, procedural personas (see Section 5.7.1.3) whereas run time RL-based player modeling offers dynamicity to the model with respect to time.
Runtime player modeling further adds on the capacity of the model to adapt to the

particular characteristics of the player, thereby increasing the degree of personaliza tion. We can think of models of players, for instance, that are continuously tailored to the current player by using the player's in-game annotations, behavioral deci sions, or even physiological responses during the game.

This way of modeling players is still in its infancy with only a few studies existent on player behavioral modeling in educational games [488], in roguelike adventure games [268, 267] via TD learning or evolutionary reinforcement learning, and in first-person shooter games [685] via inverse RL. However, the application of RL for modeling users beyond games has been quite active for the purposes of model ing web-usage data and interactions on the web [684] or modeling user simulations in dialog systems [114, 225, 595]. Normally in such systems a statistical model is

first trained on a corpus of human-computer interaction data for simulating (imitat ing) user behavior. Then reinforcement learning is used to tailor the model towards an optimal dialog strategy which can be found through trial and error interactions between the user and the simulated user.

5.6.3 Unsupervised Learning

The aim of unsupervised learning (see Chapter 2) is to derive a model given a number of observations. Unlike in supervised learning, there is no specified target output. Unlike in reinforcement learning there is no reward signal. (In short, there is no training signal of any kind.) In unsupervised learning, the signal is hidden in ternally in the interconnections among data attributes. So far, unsupervised learning has mainly been applied to two core player modeling tasks: clustering behaviors and mining associations between player attributes. While Chapter 2 provides the general description of these unsupervised learning algorithms, in this section we focus on their specific application for modeling players.

5.6.3.1 Clustering

As discussed in Chapter 2, clustering is a form unsupervised learning aiming to find clusters in datasets so that data within a cluster is similar to each other and dissimilar to data in other clusters. When it comes to the analysis of user behavior in games, clustering offers a means for reducing the dimensionality of a dataset, thereby yielding a manageable number of critical features that represent user behav ior. Relevant data for clustering in games include player behavior, navigation pat terns, assets bought, items used, game genres played and so on. Clustering can be used to group players into archetypical playing patterns in an effort to evaluate how people play a particular game and as part of a user-oriented testing process [176]. Further, one of the key questions of user testing in games is whether people play the game as intended. Clustering can be utilized to derive a number of different playing or behavioral styles directly addressing this question. Arguably, the key challenge in successfully applying clustering in games is that the derived clusters should have an intelligible meaning with respect to the game in question. Thus, clusters should be clearly interpretable and labeled in a language that is meaningful to the involved stakeholders (such as designers, artists and managers) [176, 178]. In the case studies of Section 5.7.1 we meet the above challenges and demonstrate the use of clustering in the popular Tomb Raider: Underworld (EIDOS interactive, 2008) game.

5.6.3.2 Frequent Pattern Mining

In Chapter 2 we defined frequent pattern mining as the set of problems and tech niques related to finding patterns and structures in data. Patterns include sequences and itemsets. Both frequent itemset mining (e.g., Apriori [6]) and frequent se quence mining (e.g., GSP [652]) are relevant and useful for player modeling. The key motivation for applying frequent pattern mining on game data is to find inherent and hidden regularities in the data. In that regard, key player modeling problems, such as player type identification and detection of player behavior patterns, can be viewed as frequent pattern mining problems. Frequent pattern mining can for ex ample be used to to discover what game content is often purchased together—e.g., players that buy X tend to buy Y too—or what are the subsequent actions after dying

in a level—e.g., players that die often in the tutorial level pick up more health packs in level 1 [120, 621].

5.7 What Can We Model?

As already outlined at the beginning of this chapter, modeling of users in games can be classified into two main tasks: modeling of the behavior of players and modeling of their experience. It is important to remember that modeling of player behavior is (mostly) a task of objective nature whereas the modeling of player experience is subjective given the idiosyncratic nature of playing experience. The examples we present in the remainder of this chapter highlight the various uses of AI for modeling players.

5.7.1 Player Behavior

In this section, we exemplify player behavior modeling via three representative use cases. The two first examples are based on a series of studies on player modeling by Drachen et al. in 2009 [176] and later on by Mahlmann et al. in 2010 [414] in the Tomb Raider: Underworld (EIDOS interactive, 2008) game. The analysis includes both the clustering of players [176] and the prediction [414] of their behavior, which make it an ideal case study for the purposes of this book. The third study presented in this section focuses on the use of play traces for the procedural creation of player models. That case study explores the creation of procedural personas in the MiniDungeons puzzle roguelike game.

Fig. 5.9 A screenshot from the Tomb Raider: Underworld (EIDOS interactive, 2008) game fea turing the player character, Lara Croft. The game is used as one of the player behavior modeling case studies presented in this book. Image obtained from Wikipedia (fair use).

5.7.1.1 Clustering Players in Tomb Raider: Underworld

Tomb Raider: Underworld (TRU) is a third-person perspective, advanced platform puzzle game, where the player has to combine strategic thinking in planning the 3D-movements of Lara Croft (the game's player character) and problem solving in order to go through a series of puzzles and navigate through a number of levels (see Fig. 5.9).

The dataset used for this study includes entries from 25,240 players. The 1,365 of those that completed the game were selected and used for the analysis presented below. Note that TRU consists of seven main levels plus a tutorial level. Six features of gameplay behavior were extracted from the data and are as follows: number of deaths by opponents, number of deaths by the environment (e.g., fire, traps, etc.), number of deaths by falling (e.g., from ledges), total number of deaths, game com pletion time, and the times help was requested. All six features were calculated on the basis of completed TRU games. The selection of these particular features was based on the core game design of the TRU game and their potential impact on the process of distinguishing among dissimilar patterns of play.

Three different clustering techniques were applied to the task of identifying the number of meaningful and interpretable clusters of players in the data: k-means, hierarchical clustering and self-organizing maps. While the first two have been cov ered in Chapter 2 we will briefly outline the third method here.

A self-organizing map (SOM) [347] creates and iteratively adjusts a low dimensional projection of the input space via vector quantization. In particular, a type of large SOM called an emergent self-organizing map [727] was used in con junction with reliable visualization techniques to help us identify clusters. A SOM consists of neurons organized in a low-dimensional grid. Each neuron in the grid

(map) is connected to the input vector through a connection weight vector. In ad dition to the input vector, the neurons are connected to neighbor neurons of the map through neighborhood interconnections which generate the structure of the map: rectangular and hexagonal lattices organized in a two-dimensional sheet or a three-dimensional toroid shape are some of the most popular topologies used. SOM training can be viewed as a vector quantization algorithm which resembles the

k-means algorithm. What differentiates SOM, however, is the update of the topo logical neighbors of the best-matching neuron—a best-matching neuron is a neuron for which there exists at least one input vector for which the Euclidean distance to the weight vector of this neuron is minimal. As a result, the whole neuron neighbor hood is stretched towards the presented input vector. The outcome of SOM training is that neighboring neurons have similar weight vectors which can be used for pro jecting the input data to the two-dimensional space and thereby clustering a set of data through observation on a 2D plane. For a more detailed description of SOMs, the reader is referred to [347].

To get some insight into the possible number of clusters existent in the data, k means was applied for all k values less than or equal to 20. The sum of the Euclidean distances between each player instance and its corresponding cluster centroid (i.e., quantization error) is calculated for all 20 trials of k-means. The analysis reveals that the percent decrease of the mean quantization error due to the increase of k is notably high when k = 3 and k = 4. For k = 3 and k = 4 this value equals 19% and 13% respectively while it lies between 7% and 2% for k > 4. Thus, the k-means clustering analysis provides the first indication of the existence of three or four main player behavioral clusters within the data.

As an alternative approach to k-means, hierarchical clustering is also applied to the dataset. This approach seeks to build a hierarchy of clusters existent in the data. The Ward's clustering method [747] is used to specify the clusters in the data by which the squared Euclidean distance is used as a measure of dissimilarity between data vector pairs. The resulting dendrogram is depicted in Fig. 5.10(a). As noted in Chapter 2 a dendrogram is a treelike diagram that illustrates the merging of data into clusters as a result of hierarchical clustering. It consists of many U-shaped lines connecting the clusters. The height of each U represents the squared Euclidean distance between the two clusters being connected. Depending on where the data analyst sets the squared Euclidean distance threshold a dissimilar number of clusters can be observed.

Both k-means and hierarchical clustering already demonstrate that the 1,365 players can be clustered in a low number of different player types. K-means in dicates there are for three or four clusters while the Ward's dendrogram reveals the existence of two populated and two smaller clusters, respectively, in the middle and at the edges of the tree.

Applying SOM, as the third alternative approach, allows us to cluster the TRU data by observation on a two-dimensional plane. The U-matrix depicted in Fig. 5.10(b) is a visualization of the local distance structure in the data placed onto a two-dimensional map. The average distance value between each neuron's weight vector and the weight vectors of its immediate neighbors corresponds to the height

(a) Dendrogram of TRU data using Ward hierarchical clustering. A squared Euclidean distance of 4.5 (illustrated with a horizontal black line) reveals four clusters. Image adapted from [176].
(b) U-matrix visualization of a self-organizing map depicting the four player clusters identified in a population of 1,365 TRU players (shown as small colored squares). Different square colors depict different player clusters. Valleys represent clusters whereas mountains represent cluster borders. Image adopted from [176].

Fig. 5.10 Detecting player types from TRU data using hierarchical (a) and SOM (b) clustering methods.

of that neuron in the U-matrix (positioned at the map coordinates of the neuron).

Thus, U-matrix values are large in areas where no or few data points reside, creating mountain ranges for cluster boundaries. On the other hand, visualized valleys indi cate clusters of data as small U-matrix values are observed in areas where the data space distances of neurons are small.

The SOM analysis reveals four main classes of behavior (player types) as de picted in Fig. 5.10(b). The different colors of the U-matrix correspond to the four different clusters of players. In particular, cluster 1 (8.68% of the TRU players)

corresponds to players that die very few times, their death is caused mainly by the environment, they do not request help from the game frequently and they complete the game very quickly. Given such game skills these players were labeled as Vet erans. Cluster 2 (22.12%) corresponds to players that die quite often (mainly due to falling), they take a long time to complete the game—indicating a slow-moving, careful style of play—and prefer to solve most puzzles in the game by themselves. Players of this cluster were labeled as Solvers, because they excel particularly at solving the puzzles of the game. Players of cluster 3 form the largest group of TRU players (46.18%) and are labeled as Pacifists as they die primarily from active op ponents. Finally, the group of players corresponding to cluster 4 (16.56% of TRU players), namely the Runners, is characterized by low completion times and frequent deaths by opponents and the environment.

The results showcase how clustering of player behavior can be useful to evalu ate game designs. Specifically, TRU players seem to not merely follow a specific strategy to complete the game but rather they fully explore the affordances provided by the game in dissimilar ways. The findings are directly applicable to TRU game design as clustering provides answers to the critical question of whether people play the game as intended. The main limitation of clustering, however, is that the derived clusters are not intuitively interpretable and that clusters need to be represented into meaningful behavioral patterns to be useful for game design. Collaborations be tween the data analyst and the designers of the game—as was performed in this study—is essential for meaningful interpretations of the derived clusters. The ben efit of such a collaboration is both the enhancement of game design features and the effective phenomenological debugging of the game [176]. In other words, we make sure both that no feature of the game is underused or misused and that the playing experience and the game balance are debugged.

5.7.1.2 Predicting Player Behavior in Tomb Raider: Underworld

Building on the same set of TRU player data, a second study examined the possi bilities of predicting particular aspects of playing behavior via supervised learning [414]. An aspect of player behavior that is particularly important for game design is to predict when a player will stop playing. As one of the perennial challenges of game design is to ensure that as many different types of players are facilitated in the design, being able to predict when players will stop playing a game is of interest because it assists with locating potentially problematic aspects of game design. Fur-

ther, such information helps toward the redesign of a game's monetization strategy for maximizing user retention.

Data was drawn from the Square Enix Europe Metrics Suite and was collected during a two month period (1st Dec 2008 – 31st Jan 2009), providing records from approximately 203,000 players. For the player behavior prediction task it was de cided to extract a subsample of 10,000 players which provides a large enough and representative dataset for the aims of the study, while at the same time is manage able in terms of computational effort. A careful data-preprocessing approach yielded 6,430 players that were considered for the prediction task—these players had com pleted the first level of the game.

As in the TRU cluster analysis the features extracted from the data relate to the core mechanics of the game. In addition to the six features investigated in the cluster ing study of TRU the extracted features for this study include the number of times the adrenalin feature of the game was used, the number of rewards collected, the number of treasures found, and the number of times the player changes settings in the game (including player ammunition, enemy hit points, player hit points, and re covery time when performing platform jumps). Further details about these features can be found in [414].

To test the possibility of predicting the TRU level the player completed last a number of classification algorithms are tested on the data using the Weka machine learning software [243]. The approach followed was to experiment with at least one algorithm from each of the algorithm families existent in Weka and to put additional

effort on those classification algorithms that were included in a recent list of the most important algorithms in data mining: decision tree induction, backpropagation and simple regression [759]. The resulting set of algorithms chosen for classification are as follows: logistic regression, multi-layer perceptron backpropagation, variants of decision trees, Bayesian networks, and support vector machines. In the following section, we only outline the most interesting results from those reported in [414]. For all tested algorithms, the reported classification prediction accuracy was achieved through 10-fold cross validation.

Most of the tested algorithms had similar levels of performance, and were able to predict when a player will stop playing substantially better than the baseline. In particular, considering only gameplay of level 1 classification algorithms reach an accuracy between 45% and 48%, which is substantially higher than the baseline per formance (39.8%). When using additional features from level 2, the predictions are much more accurate—between 50% and 78%—compared to the baseline (45.3%). In particular, decision trees and logistic regression manage to reach accuracies of almost 78% in predicting on what level a player will stop playing. The difference in the predictive strength of using level 1 and 2 data as compared to only level 1 data is partly due to increased amount of features used in the latter case.

Beyond accuracy an important feature of machine learning algorithms is their transparency and their expressiveness. The models are more useful to a data analyst and a game designer if they can be expressed in a form which is easy to visualize and comprehend. Decision trees—of the form constructed by the ID3 algorithm [544] and its many derivatives—are excellent from this perspective, especially if pruned to

Fig. 5.11 A decision tree trained by the ID3 algorithm [544] to predict when TRU players will stop playing the game. The leaves of the tree (ovals) indicate the number of the level (2, 3 or 7) the player is expected to complete. Note that the TRU game has seven levels in total. The tree is constrained to tree depth 2 and achieves a classification accuracy of 76.7%.

a small size. For instance, the extremely small decision tree depicted in Fig. 5.11 is constrained to tree depth 2 and was derived on the set of players who completed both levels 1 and 2 with a classification accuracy of 76.7%. The predictive capacity of the decision tree illustrated in Fig. 5.11 is impressive given how extremely simple it is. The fact that we can predict the final played level—with a high accuracy—based only on the amount of time spent in the room named Flush Tunnel of level 2 and the total rewards collected in level 2 is very appealing for game design. What this decision tree indicates is that the amount of time players spend within a given area early in the game and how well they perform are important for determining if they continue playing the game. Time spent on a task or in an area of the game can indeed be indicative of challenges with progressing through the game, which can result in a frustrating experience.

5.7.1.3 Procedural Personas in MiniDungeons

Procedural personas are generative models of player behavior, meaning that they can replicate in-game behavior and be used for playing games in the same role as players; additionally, procedural personas are meant to represent archetypical players rather than individual players [268, 267, 269]. A procedural persona can be defined as the parameters of a utility vector that describe the preferences of a

player. For example, a player might allocate different weight to finishing a game fast, exploring dialog options, getting a high score, etc.; these preferences can be numerically encoded in a utility vector where each parameter corresponds to the persona's interest in a particular activity or outcome. Once these utilities are de-fined, reinforcement learning via TD learning or neuroevolution can be used to find a policy that reflects these utilities, or a tree search algorithm such as MCTS can be used with these utilities as evaluation functions. Approaches similar to the proce dural persona concept have also been used for modeling the learning process of the player in educational games via reinforcement learning [488].

As outlined in Section 5.4.1, MiniDungeons is a simple rogue-like game which

features turn-based discrete movement, deterministic mechanics and full informa tion. The player avatar must reach the exit of each level to win it. Monsters block the way and can be destroyed, at the cost of decreasing the player character's health; health can be restored by collecting potions. Additionally, treasures are distributed throughout levels. In many levels, potions and treasures are placed behind monsters, and monsters block the shortest path from the entrance to the exit. Like many games, it is therefore possible to play MiniDungeons with different goals, such as reaching the exit in the shortest possible time, collecting as much treasure as possible or killing all the monsters (see Figs. 5.3 and 5.12).

These different playing styles can be formalized as procedural personas by at taching differing utilities to measures such as the number of treasures collected, the number of monsters killed or the number of turns taken to reach the exit. Q-learning can be used to learn policies that implement the appropriate persona in single lev els [268], and evolutionary algorithms can be used to train neural networks that implement a procedural persona across multiple levels [267]. These personas can be compared with the play traces of human players by placing the persona in ev ery situation that the human encountered in the play trace and comparing the action chosen by the procedural persona with the action chosen by the human (as you are comparing human actions with those of a Q function, you might say that you are asking "what would Q do?"). It is also possible to learn the utility values for a procedural persona clone of a particular human player by evolutionary search for those values that make the persona best match a particular play trace (see Fig. 5.12). However, it appears that these "clones" of human players generalize less well than designer-specified personas [269].

5.7.2 Player Experience

The modeling of player experience involves learning a set of target outputs that ap proximate the experience (as opposed of the behavior) of the player. By definition, that which is being modeled (experience) is of subjective nature and the model ing therefore requires target outputs that somehow approximate the ground truth of experience. A model of player experience predicts some aspect of the experience a player would have in some game situation, and learning such models is naturally a

(a) An artificial neural network mapping between a game state (input) and plans (output). The input contains blue circles rep resenting distances to various important elements of the game and a red circle representing hit points of the player character.
(b) A level of MiniDungeons 2 depicting the current state of the game.

Fig. 5.12 A example of a procedural persona: In this example we evolve the weights of artificial neural networks—an ANN per persona. The ANN takes observations of the player character and its environment and uses these to choose from a selection of possible plans. During evolution the utility function of each persona is used as the fitness function for adjusting its network weights. Each individual of each generation is evaluated by simulating a full game. A utility function allows us to evolve a network to pursue multiple goals across a range of different situations. The method depends on the designer providing carefully chosen observations, appropriate planning algorithms, and well-constructed utility functions. In this example the player opts to move towards a safe treasure. This is illustrated with a green output neuron and highlighted corresponding weights (a) and a green path on the game level (b).

supervised learning problems. As mentioned, there are many ways this can be done, with approaches to player experience modeling varying regarding the inputs (from what the experience is predicted, e.g., physiology, level design parameters, playing style or game speed), the outputs (what sort of experience is predicted, e.g., fun, frustration, attention or immersion) and the modeling methodology.

In this section we will outline a few examples of supervised learning for model ing the experience of players. To best cover the material (methods, algorithms, uses

of models) we rely on studies which have been thoroughly examined in the litera ture. In particular, in the remainder of this section we outline the various approaches
and extensive methodology for modeling experience in two games: a variant of the popular Super Mario Bros (Nintendo, 1985) game and a 3D prey-predator game named Maze-Ball.

5.7.2.1 Modeling Player Experience in Super Mario Bros

Our first example builds upon the work of Pedersen et al. [521, 520] who modified an open-source clone of the classic platform game Super Mario Bros (Nintendo, 1985) to allow for personalized level generation. That work is important in that it set the foundations for the development of the experience-driven procedural content

generation framework [783] which constitutes a core research trend within proce dural content generation (see also Chapter 4).

The game used in this example is a modified version of Markus Persson's Infinite Mario Bros which is a public domain clone of Nintendo's classic platform game Super Mario Bros (Nintendo, 1985). All experiments reported in this example rely on a model-free approach for the modeling of player experience. Models of player experience are based on both the level played (game context) and the player's play ing style. While playing, the game recorded a number of behavioral metrics of the players, such as the frequency of jumping, running and shooting, that are taken into consideration for modeling player experience. Further, in a follow-up experiment [610], the videos of players playing the game were also recorded and used to extract a number of useful visual cues such as the average head movement during play. The output (ground truth of experience) for all experiments is provided as first-person, rank-based reports obtained via crowdsourcing. Data was crowdsourced from hun dreds of players, who played pairs of levels with different level parameters (e.g., dissimilar numbers, sizes and placements of gaps) and were asked to rank which of two levels best induced a number of player states. Across the several studies of player experience modeling for this variant of Super Mario Bros (Nintendo, 1985) collectively players have been asked to annotate fun, engagement, challenge, frus tration, predictability, anxiety, and boredom. These are the target outputs the player model needs to predict based on the input parameters discussed above.

Given the rank-based nature of the annotations the use of preference learning is necessary for the construction of the player model. The collected data is used to train artificial neural networks that predict the players' experiential states, given a player's behavior (and/or affective manifestations) and a particular game context, using evo lutionary preference learning. In neuroevolutionary preference learning [763], a genetic algorithm evolves an artificial neural network so that its output matches the pairwise preferences in the data set. The input of the artificial neural network is a set of features that have been extracted from the data set—as mentioned earlier, the input may include gameplay and/or objective data in this example. It is worth noting that automatic feature selection is applied to pick the set of features (model input) that are relevant for predicting variant aspects of player experience. The ge netic algorithm implemented uses a fitness function that measures the difference between the reported preferences and the relative magnitude of the model output. Neuroevolutionary preference learning has been used broadly in the player model ing literature and the interested reader may refer to the following studies (among many): [432, 610, 763, 521, 520, 772].

The crowdsourcing experiment of Pedersen et al. [521, 520] resulted in data (gameplay and subjective reports of experience) from 181 players. The best pre dictor of reported fun reached an accuracy of around 70% on unseen subjective reports of fun. The input of the neural network fun-model is obtained through au tomatic feature selection consists of the time Mario spent moving left, the number of opponents Mario killed from stomping, and the percentage of level played in the left direction. All three playing features appear to contribute positively for the prediction of reported fun in the game. The best-performing model for challenge

Fig. 5.13 Facial feature tracking for head movement. Image adapted from [610].
prediction had an accuracy of approximately 78%. It is more complex than the best fun predictor, using five features: time Mario spends standing still, jump difficulty, coin blocks Mario pressed, number of cannonballs Mario killed, and Mario kills by stomping. Finally, the best predictor for frustration reaches an accuracy of 89%. It is indeed an impressive finding that a player experience model can predict (with near-certainty) whether the player is frustrated by the current game by merely cal culating the time Mario spent standing still, the time Mario spent on its last life, the jump difficulty, and the deaths Mario had from falling in gaps. The general findings of Pedersen et al. [520] suggest that good predictors for experience can be found if a preference learning approach is applied on crowdsourced reports of experience and gameplay data. The prediction accuracies, however, depend on the complexity of the reported state—arguably fun is a much more complicated and fuzzier notion to report compared to challenge or frustration. In a follow up study by Pedersen et al. [521] the additional player states of predictability, anxiety and boredom were predicted with accuracies of approximately 77%, 70% and 61%, respectively. The same player experience methodology was tested on an even larger scale, soliciting data from a total number of 780 players of the game [621]. Frequent pattern mining algorithms were applied to the data to derive frequent sequences of player actions. Using sequential features of gameplay the models reach accuracies of up to 84% for engagement, 83% for frustration and 79% for challenge.

In addition to the behavioral characteristics the visual cues of the player can be taken into account as objective input to the player model. In [610] visual features were extracted from videos of 58 players, both throughout whole game sessions and during small periods of critical events such as when a player completes a level or when the player loses a life (see Figs. 5.13 and 5.14). The visual cues enhance the quality of the information we have about a player's affective state which, in turn, allows us to better approximate player experience. Specifically, fusing the game play and the visual reaction features as inputs to the artificial neural network we achieve average accuracies of up to 84%, 86% and 84% for predicting reported en gagement, frustration and challenge, respectively. The key findings of [610] suggest

(a) Winning (b) Losing
(c) Experiencing challenge (d) Experiencing challenge
Fig. 5.14 Examples of facial expressions of Super Mario Bros (Nintendo, 1985) players for differ ent game states. All images are retrieved from the Platformer Experience Dataset [326].

that players' visual reactions can provide a rich source of information for modeling experience preferences and lead to more accurate models of player experience.

5.7.2.2 Modeling Player Experience in Maze-Ball

Our second example for modeling player experience builds largely upon the exten sive studies of Mart´ınez et al. [434, 430, 435] who analyzed player experience us ing a simple 3D prey-predator game named Maze-Ball towards achieving affective driven camera control in games. While the game is rather simple, the work on Maze Ball offers a thorough analysis of player experience via a set of sophisticated tech niques for capturing the psychophysiological patterns of players including prefer ence learning, frequent pattern mining and deep convolutional neural networks. In addition the dataset that resulted from these studies is publicly available for further experimentation and forms a number of suggested exercises for this book.

Maze-Ball is a three-dimensional prey-predator game (see Fig. 5.15); similar to a 3D version of Pac-Man (Namco,1981). The player (prey) controls a ball which

(a) Maze-Ball (b) Space-Maze
Fig. 5.15 Early Maze-Ball prototype (a) and a polished variant of the game (b) that features real-time camera adaptation [345]. The games can be found and played at http://www.hectorpmartinez.com/.

moves inside a maze hunted by 10 opponents (predators) moving around the maze. The goal of the player is to maximize her score by gathering as many tokens, scat tered in the maze, as possible while avoiding being touched by the opponents in a

predefined time window of 90 seconds. A detailed description of Maze-Ball game play can be found in [780].
Gameplay attributes and physiological signals (skin conductance and heart rate variability) were acquired from 36 players of Maze-Ball. Each subject played a pre defined set of eight games for 90 seconds, thus the total number of game sessions available is 288. Gameplay and physiological signals define the input of the player experience model. To obtain the ground truth of experience players self-reported their preference about game pairs they played using a rank-based questionnaire, in particular a 4-alternative forced choice (4-AFC) protocol [780]. They were asked to rank the two games of each game pair with respect to fun, challenge, boredom, frus tration, excitement, anxiety and relaxation. These annotations are the target outputs the player model will attempt to predict based on the input parameters discussed above.
Several features were extracted from the gameplay and physiological signals ob tained. These included features related to kills and deaths in the game as well as features associated with the coverage of the level. For extracting features from the physiological signals the study considered their average, standard deviation, and maximum and minimum values. The complete feature list and the experimental pro tocol followed can be found in [780].
As in the example with the game variant of Super Mario Bros (Nintendo, 1985) the rank-based nature of the annotations requires the use of preference learning for the construction of the player experience model. Thus, the collected data is used to train neural networks that predict the player states, given a player's gameplay be havior and its physiological manifestations, using evolutionary preference learning.
The architecture of the neural network can be either shallow using a simple multi layered perceptron or deep (using convolutional neural networks [430, 435]). Figure 5.16 shows the different ways information from gameplay and physiology can been
5.7. What Can We Model? 251
Fig. 5.16 Three dissimilar approaches to deep multimodal fusion via convolutional neural net works. Gameplay events are fused with skin conductance in this example. The networks illustrated present two layers with one neuron each. The first convolutional layer receives as input a contin uous signal at a high time resolution, which is further reduced by a pooling layer. The resulting signal (feature map) presents a lower time resolution. The second convolutional layer can combine this feature map with additional modalities at the same low resolution. In the convolution fusion network (left figure), the two events are introduced at this level as a pulse signal. In the pooling fusion network (middle figure), the events are introduced as part of the first pooling layer, resulting in a filtered feature map. Finally, in the training fusion network (right figure), the events affect the training process of the first convolutional layer, leading to an alternative feature map. Image adapted from [435].
fused on a deep convolutional neural network which is trained via preference learning to predict player experience in any game experience dataset that contains discrete in-game events and continuous signals (e.g., the player's skin conductance).
Predictors of the player experience states can reach accuracies that vary from 72% for challenge up to 86% for frustration using a shallow multi-layer percep tron player model [780]. Significant improvements are observed in those accuracies when the input space of the model is augmented with frequent sequences of in-game and physiological events (i.e., fusion on the input space). As in [621], Mart´ınez et al. used GSP to extract those frequent patterns that were subsequently used as in puts of the player model [434]. Further accuracy improvements can be observed when physiology is fused with physiological signals on deep architectures of con volutional neural networks [435]. Using deep fusion (see Fig. 5.16) accuracies of predicting player experience may surpass 82% for all player experience states con sidered. Further information about the results obtained in the Maze-Ball game can be found in the following studies: [780, 434, 430, 435, 436].
252 Chapter 5. Modeling Players
5.8 Further Reading
For an extensive reading on game and player analytics (including visualization,

data preprocessing, data modeling and game domain-dependent tasks) we refer the reader to the edited book by El-Nasr et al. [186]. When it comes to player modeling two papers offer complementary perspectives and taxonomies of player modeling and a thorough discussion on what aspects of players can be modeled and the ways players can be modeled: the survey papers of Smith et al. [636] and Yannakakis et al. [782].

5.9 Exercises

In this section we propose a set of exercises for modeling both the behavior and the experience of game players. For that purpose, we outline a number of datasets that can be used directly for analysis. Please note, however, that the book's website will remain up to date with more datasets and corresponding exercises beyond the ones covered below.

5.9.1 Player Behavior

A proposed semester-long game data mining project is as follows. You have to choose a dataset containing player behavioral attributes to apply the necessary pre processing on the data such as extracting features and selecting features. Then you must apply a relevant unsupervised learning technique for compressing, analyz ing, or reducing the dimensionality of your dataset. Based on the outcome of unsu pervised learning you will need to implement a number of appropriate supervised learning techniques that learn to predict a data attribute (or a set of attributes). We leave the selection of algorithms to the reader or the course instructor. Below we dis cuss a number of example datasets one might wish to start from; the reader, however, may refer to the book's website for more options on game data mining projects.

5.9.1.1 SteamSpy Dataset

SteamSpy (http://steamspy.com/) is a rich dataset of thousands of games released on Steam[6] containing several attributes each. While strictly not a dataset focused on player modeling, SteamSpy offers an accessible and large dataset for game analyt ics. The data attributes of each game include the game's name, the developer, the publisher, the score rank of the game based on user reviews, the number of owners

[6] http://store.steampowered.com/

of the game on Steam, the people that have played this game since 2009, the people that have played this game in the last two weeks, the average and median playtime, the game's price and the game's tags. The reader may use an API[7] to download all data attributes from all games contained in the dataset. Then one might wish to apply supervised learning to be able to predict an attribute (e.g., the game's price) based on other game features such as the game's score, release date and tags. Or alterna tively, one might wish to construct a score predictor of a new game. The selection of the modeling task and the AI methods is left to the reader.

5.9.1.2 StarCraft: Brood War Repository

The StarCraft: Brood War repository contains a number of datasets that include thousands of professional StarCraft replays. The various data mining papers, datasets as well as replay websites, crawlers, packages and analyzers have been compiled by Alberto Uriarte at Drexel University.[8]

In this exercise you are faced with the chal lenge of mining game replays with the aim to predict a player's strategy. Some results on the StarCraft: Brood War datasets can be found in [750, 728, 570] among others.

5.9.2 Player Experience

As a semester project on player experience modeling it is suggested you choose a game, one or more affective or cognitive states to model (model's output) and one or more input modalities. You are expected to collect empirical data using your selected game and build models for the selected psychological state of the players that rely on the chosen input modalities.

As a smaller project that does not involve data collection you may opt to choose

one of the following datasets and implement a number of AI methods that will de rive accurate player experience models. The models should be compared in terms of
a performance measure. The two datasets accompanying this book and outlined be low are the platformer experience dataset and the Maze-Ball dataset. The book's
website will be up to date with more datasets and exercises beyond the ones covered
below.

5.9.2.1 Platformer Experience Dataset

The extensive analysis of player experience in Super Mario Bros (Nintendo, 1985)
and our wish to further advance our knowledge and understanding on player expe-

7 http://steamspy.com/api.php

8 Available at: http://nova.wolfwork.com/dataMining.html

rience had led to the construction of the Platformer Experience Dataset [326]. This
is the first open-access game experience corpus that contains multiple modalities of
data from players of Infinite Mario Bros, a variant of Super Mario Bros (Nintendo,
1985). The open-access database can be used to capture aspects of player experi ence based on behavioral and visual recordings of platform game players. In addi tion, the database contains aspects of the game context—such as level attributes—
demographic data of the players and self-reported annotations of experience in two
forms: ratings and ranks.

Here are a number of questions you might wish to consider when attempting to
build player experience models that are as accurate as possible: Which AI methods
should I use? How should I treat my output values? Which feature extraction and
selection mechanism should I consider? The detailed description of the dataset can
be found here: http://www.game.edu.mt/PED/. The book website contains further
details and a set of exercises based on this dataset.

5.9.2.2 Maze-Ball Dataset

As in the case of the Platformer Experience Dataset the Maze-Ball dataset is also
publicly available for further experimentation. This open-access game experience
corpus contains two modalities of data obtained from Maze-Ball players: their
gameplay attributes and three physiological signals: blood volume pulse, heart rate
and skin conductance. In addition, the database contains aspects of the game such as
features of the virtual camera placement. Finally the dataset contains demographic
data of the players and self-reported annotations of experience in two forms: ratings
and ranks.

The aim, once again, is to construct the most accurate models of experience for
the players of Maze-Ball. So, which modalities of input will you consider? Which
annotations are more reliable for predicting player experience? How will your sig nals be processed? These are only a few of the possible questions you will en counter during your efforts. The detailed description of the dataset can be found
here: http://www.hectorpmartinez.com/. The book website contains further details
about this dataset.

5.10 Summary

This chapter focused on the use of AI for modeling players. The core reasons why
AI should be used for that purpose is either to derive something about the players'
experience (how they feel in a game) or for us to understand something about their
behavior (what they do in a game). In general we can model player behavior and
player experience by following a top-down or a bottom-up approach (or a mix of
the two). Top-down (or model-based) approaches have the advantage of solid the oretical frameworks usually derived from other disciplines or other domains than

games. Bottom-up (or model-free) instead rely on data from players and have the
advantage of not assuming anything about players other than that player experience
and behavior are associated with data traces left by the player and that these data
traces are representative of the phenomenon we wish to explain. While a hybrid
between model-based and model-free approaches is in many ways a desirable ap proach to player modeling, we focus on bottom-up approaches, where we provide

a detailed taxonomy for the options available regarding the input and the output of the model, and the modeling mechanism per se. The chapter ends with a number of player modeling examples, for modeling both the behavior of players and their experience.

The player modeling chapter is the last chapter of the second part of this book, which covered the core uses of AI in games. The next chapter introduces the third and last part of the book, which focuses on the holistic synthesis of the various AI areas, the various methods and the various users of games under a common game AI framework.

Part III

The Road Ahead

Chapter 6

Game AI Panorama

This chapter attempts to give a high-level overview of the field of game AI, with particular reference to how the different core research areas within this field inform and interact with each other, both actually and potentially. For that purpose we first identify the main research areas and their sub-areas within the game AI field. We then view and analyze the areas from three key perspectives: (1) the dominant AI method(s) used under each area; (2) the relation of each area with respect to the end (human) user; and (3) the placement of each area within a human-computer (player game) interaction perspective. In addition, for each of these areas we consider how it could inform or interact with each of the other areas; in those cases where we find that meaningful interaction either exists or is possible, we describe the character of that interaction and provide references to published studies, if any.

The main motivations for us writing this chapter is to help the reader understand how a particular area relates to other areas within this increasingly growing field, how the reader can benefit from knowledge created in other areas and how the reader can make her own research more relevant to other areas. To facilitate and foster synergies across active research areas we place all key studies into a taxonomy with the hope of developing a common understanding and vocabulary within the field of AI and games. The structure of this chapter is based on the first holistic overview of the game AI field presented in [785]. The book takes a new perspective on the key game AI areas given its educational and research focus.

The main game AI areas and core subareas already identified in this book and covered in this chapter are as follows:

• Play Games (see Chapter 3) which includes the subareas of Playing to Win and Playing for the Experience. Independently of the purpose (winning or experi ence) AI can control either the player character or the non-player character.

• Generate Content (see Chapter 4) which includes the subareas of autonomous (procedural) content generation and assisted content generation. Please note that the terms assisted (procedural) content generation and mixed-initiative (proce dural) content generation (as defined in Chapter 4) are used interchangeably in this chapter.

• Model Players (see Chapter 5) which includes the subareas of player experience modeling and player behavior modeling, or else, game data mining [178].

The scope of this chapter is not to provide an inclusive survey of all game AI areas—the details of each area have been covered in preceding chapters of the book—but rather a roadmap of interconnections between them via representative examples. As research progresses in this field, new research questions will pop up and new methods be invented, and other questions and methods recede in impor tance. We believe that all taxonomies of research fields are by necessity tentative.

Consequently, the list of areas defined in this chapter should not be regarded as fixed and final.

The structure of the chapter is as follows: In Section 6.1, we start by holistically

analyzing the game AI areas within the game AI field and we provide three alter native views over game AI: one with respect to the methods used, one with respect to the end users within game research and development and one where we outline how each of the research areas fits within the player-game interaction loop of digital games. Then, Section 6.2, digs deeper into the research areas and describes each one of them in detail. With the subsection describing each area, there is a short de scription of the area and a paragraph on the possible interactions with each of the other areas for which we have been able to identify strong or weak influences . The chapter ends with a section containing our key conclusions and vision for the future of the field.

6.1 Panoramic Views of Game AI

Analyzing any research field as a composition of various subareas with intercon nections and interdependencies can be achieved in several different ways. In this section we view game AI research from three high-level perspectives that focus on the computer (i.e., the AI methods), the human (i.e., the potential end user of game AI) and the interaction between the key end user (i.e., player) and the game. In stead in Section 6.2 we outline each game AI area and present the interconnections between the areas.

Game AI is composed of (a set of) methods, processes and algorithms in artificial intelligence as those are applied to, or inform the development of, games. Naturally, game AI can be analyzed through the method used by identifying the dominant AI approaches under each game AI area (see Section 6.1.1). Alternatively, game AI can be viewed from the game domain perspective with a focus on the end users of each game AI area (see Section 6.1.2). Finally game AI is, by nature, realized through systems that entail rich human-computer interaction (i.e., games) and, thus, the different areas can be mapped to the interaction framework between the player and the game (see Section 6.1.3).

Table 6.1 Dominant (•) and secondary (◦) AI methods for each of the core AI areas we cover in this book. The total number of methods used for each area appears at the bottom row of the table.

	Play Games	Generate Content	Model Players		
	Winning	Experience Autonomously	Assisted Experience	Behavior	Behavior
Authoring	•	•			
Tree Search	•	◦	◦	◦	
Evolutionary Computation	•	◦	•	•	•
Supervised Learning	◦	•	•	•	
Reinforcement Learning	•	◦			
Unsupervised Learning	◦	◦	•		
Total (Dominant)	5 (4)	5 (2)	2 (1)	3 (1)	3 (2) 2 (2)

6.1.1 Methods (Computer) Perspective

The first panoramic view of game AI we present is centered around the AI meth ods used in the field. As the basis of this analysis we first list the core AI methods mostly used in the game AI field. The key methodology areas identified in Chapter 2 include ad-hoc behavior authoring, tree search, evolutionary computation, rein forcement learning, supervised learning, and unsupervised learning. For each of the game AI areas investigated we have identified the AI methods that are dominant or secondary in the area. While the dominant methods represent the most popular techniques used in the literature, secondary methods represent techniques that have been considered from a substantial volume of studies but are not dominant.

We have chosen to group methods according to what we perceive as a received taxonomy and following the structure of Chapter 2. While it would certainly be pos sible to classify the various methods differently, we argue that the proposed classi- fication is compact (containing solely key methodology areas) and it follows stan dard method classifications in AI. While this taxonomy is commonly accepted, the lines can be blurred. In particular evolutionary computation, being a very general

optimization method, can be used to perform supervised, unsupervised or reinforce ment learning (more or less proficiently). The model-building aspect of reinforce ment learning can be seen as a supervised learning problem (mapping from action sequences to rewards), and the commonly used tree search method Monte Carlo tree search can be seen as a form of TD learning. The result of any tree search algorithm can be seen as a plan, though it is often not guaranteed to lead to the desired end state. That the various methods have important commonalities and some overlap does not detract from the fact that each of them is clearly defined.

Table 6.1 illustrates the relationship between game AI areas and correspond ing methods. It is evident that evolutionary computation and supervised learning appear to be of dominant or secondary use in most game AI areas. Evolutionary computation is a dominant method for playing to win, for generating content (in an assisted/mixed-initiative fashion or autonomously), and for modeling players; it has also been considered for the design of believable play (play for experience) research. Supervised learning is of substantial use across the game AI areas and appears to be

dominant in player experience and behavioral modeling, as well as in the area of AI that plays for experience. Behavior authoring, on the other hand, is useful solely for game-playing. Reinforcement learning and unsupervised learning find limited use across the game AI areas, respectively, being dominant only on AI that plays to win and player behavior modeling. Finally, tree search finds use primarily in playing to win and it is also considered—as a form of planning—for controlling play for experience and in computational narrative (as part of autonomous or assisted PCG).

Viewing Table 6.1 from the game AI areas' perspective (columns) it seems that AI that plays games (either for wining or for the experience) defines the game AI area with the most diverse and richest palette of AI methods. On the contrary, proce dural content generation is solely dominated by evolutionary computation and tree search to a secondary degree. It is important to state that the popularity of any AI method within a particular area is closely tied to the task performed or the goal in mind. For example, evolutionary computation is largely regarded as a computation ally heavy process which is mostly used in tasks associated with offline training. As PCG so far mainly relies on content that is generated offline, evolutionary compu tation offers a good candidate method and the core approach behind search-based PCG [720]. If online learning is a requirement for the task at hand, however, other methods (such as reinforcement learning or pruned tree-search) tend to be preferred.

Clearly the possibility space for future implementations of AI methods under particular game AI areas seems rather large. While particular methods have been traditionally dominant in specific areas for good reasons (e.g., planning in compu tational narrative) there are equally good reasons to believe that the research in a game AI area itself has been heavily influenced by (and limited to) its correspond ing dominant AI methods. The empty cells of Table 6.1 indicate potential areas for exploration and offer us an alternative view of promising new intersections between game AI areas and methods.

6.1.2 End User (Human) Perspective

The second panoramic view of the game AI field puts an emphasis on the end user of the AI technology or general outcome (product or solution). Towards that aim we investigate three core dimensions of the game AI field and classify all game AI areas with respect to the process AI follows, the game context under which algorithms operate and, finally, the end user that benefits most from the resulting outcome. The classes identified under the above dimensions are used as the basis of the taxonomy we propose.

The first dimension (phrased as a question) refers to the AI process: In general, what can AI do within games? We identify two potential classes in this dimension: AI can model or generate. For instance, an artificial neural network can model a playing pattern, or a genetic algorithm can generate game assets. Given that AI can model or generate the second dimension refers to the context: What can AI methods model or generate in a game? The two possible classes here are content

Fig. 6.1 The end user perspective of the identified game AI areas. Each AI area follows a process (model or generate) under a context (content or behavior) for a particular end user (designer, player, AI researcher or game producer/publisher). Blue and red arrows represent the processes of modeling and generation, respectively. Modified graph from [785].
and behavior. For example, AI can model a players' affective state, or generate a level. Finally, the third dimension is the end user: AI can model, or generate, either content or behavior; but, for whom? The classes under the third dimension are the designer, the player, the AI researcher, and the producer/publisher.
Note that the above taxonomy serves as a framework for classifying the game AI areas according to the end user and is, by no means, inclusive of all potential processes, contexts, and end users. For instance, one could claim that the producer's role should be distinct from the publisher's role and that a developer should also be included in that class. Moreover, game content could be further split into smaller sub-classes such as narrative, levels, etc. Nevertheless, the proposed taxonomy pro vides distinct roles for the AI process (model vs. generate vs. evaluate), clear-cut classification for the context (content vs. behavior) and a high-level classification of the available stakeholders in game research and development (designer vs. player vs. AI researcher vs. producer/publisher). The taxonomy presented here is a modified version of the one introduced in [785] and it does not consider evaluation as a process for AI since it is out of the primary scope of this book.
Figure 6.1 depicts the relationship between the game AI core areas, the subareas and the end users in game research and development. Assisted, or mixed-initiative, content generation is useful for the designer and entails all possible combinations of processes and context as both content and behavior can be either modeled or gener ated for the designer. Compared to the other stakeholders the player benefits directly from more game AI research areas. In particular the player and her experience are affected by research on player modeling, which results from the modeling of ex perience and behavior; research on autonomous procedural content generation, as

a result of generation of content; and studies on NPC playing (for wining or expe rience) resulting from the generation of behavior. The player character (PC)-based game playing (for winning or experience) areas provide input to the AI researcher primarily. Finally, the game producer/publisher is primarily affected by results on behavioral player modeling, game analytics and game data mining as a result of behavior modeling.

6.1.3 Player-Game Interaction Perspective

The third and final panoramic perspective of game AI presented in this section couples the computational processes with the end user within a game and views all game AI areas through a human-computer interaction—or, more accurately, a player-game interaction—lens. The analysis builds on the findings of Section 6.1.2 and places the five game AI areas that concern the player as an end user on a player game interaction framework as depicted in Fig. 6.2. Putting an emphasis on player experience and behavior, player modeling directly focuses on the interaction be tween a player and the game context. Game content is influenced primarily by re search on autonomous procedural content generation. In addition to other types of content, most games feature NPCs, the behavior of which is controlled by some form of AI. NPC behavior is informed by research in NPCs that play the game to win or any other playing-experience purpose such as believability.
Looking at the player-game interaction perspective of game AI it is obvious that the player modeling area has the most immediate and direct impact on the player experience as it is the only area linked to the player-game interaction directly. From the remaining areas, PCG influences player experience the most as all games have some form of environment representation and mechanics. Finally, AI that plays as an NPC (either to win or for the experience of play) is constrained to games that include agents or non-player characters.
The areas not considered directly in this game AI perspective affect the player

rather remotely. Research on AI tools that assist the generative process of content improves the game's quality as a whole and in retrospect the player experience since designers tend to maintain a second-order player model [378] while designing. Fi nally, AI that plays the game as a player character can be offered for testing both the content and the NPC behaviors of a game, but also the interaction between the player and the game (via e.g., player experience competitions), but is mainly directed to AI researchers (see Fig. 6.1).

6.2 How Game AI Areas Inform Each Other

In this section, we outline the core game AI areas and discuss how they inform or influence (the terms are used interchangeably) each other. All research areas could

Fig. 6.2 Game AI areas and sub-areas viewed from a player-game interaction perspective.

be seen as potentially influencing each other to some degree; however, making a list of all such influences would be impractical and the result would be uninteresting. Therefore we only describe direct influences. Direct influences can be either strong (represented by a • as the bullet point style next to the corresponding influence in the following lists) or weak (represented by a ◦). We do not list influences we do not consider potentially important for the informed research area, or which only go through a third research area.

The sections below list outgoing influence. Therefore, to know how area A influences area B you should look in the section describing area A. Some influences are mutual, some not. The notation A → B in the headings of this section denotes that "A influences B". In addition to the text description each section provides a fig ure representing all outgoing influences of the area as arrows. Dark and light gray colored areas represent, respectively, strong and weak influence. Areas with white background are not influenced by the area under consideration. The figures also de pict the incoming influences from other areas. Incoming strong and weak influences are represented, respectively, with a solid line and a dashed line around the game AI areas that influence the area under consideration. Note that the description of the

incoming influence from an area is presented in the corresponding section of that area.

6.2.1 Play Games

The key area in which AI plays a game (as covered in Chapter 3) involves the sub areas of Playing to Win and Playing for Experience. As mentioned earlier in the chapter the AI can control either player or non-player characters of the game. We cover the influences to (and from) these subareas of game AI in this section.

6.2.1.1 Playing to Win (as a Player or as a Non-Player)

As already seen in Chapter 3 research in AI that learns to play (and win) a game focuses on using reinforcement learning techniques such as temporal difference learning or evolutionary algorithms to learn policies/behaviors that play games well—whether it is a PC or an NPC playing the game. From the very beginning of AI research, reinforcement learning techniques have been applied to learn how to play board games (see for example Samuel's Checkers player [591]). Basically, playing the game is seen as a reinforcement learning problem, with the reinforcement tied to some measure of success in the game (e.g., the score, or length of time survived). As with all reinforcement learning problems, different methods can be used to solve the problem (find a good policy) [715] including TD learning [689], evolutionary computation [406], competitive co-evolution [24, 538, 589, 580], simulated anneal ing [42], other optimization algorithms and a large number of combinations between such algorithms [339]. In recent years a large number of papers that describe the application of various learning methods to different types of video games have ap peared in the literature (including several overviews [470, 406, 632, 457]). Finally, using games to develop artificial general intelligence builds on the idea that games can be useful environments for algorithms to learn complex and useful behaviors; thus research in algorithms that learn to win is essential.

It is also worth noting that most existing game-based benchmarks measure how

well an agent plays a game—see for example [322, 404, 504]. Methods for learning to play a game are vital for such benchmarks, as the benchmarks are only meaningful in the context of the algorithms. When algorithms are developed that "beat" existing benchmarks, new benchmarks need to be developed. For example, the success of an early planning agent in the first Mario AI competition necessitated that the software be augmented with a better level generator for the next competition [322], and for the Simulated Car Racing competition, the performance of the best agents on the original competition game spurred the change to a new more sophisticated racing game [710, 392].

Fig. 6.3 Playing to Win: influence on (and from) other game AI research areas. Outgoing influence (represented by arrows): black and dark gray colored areas reached by arrows represent, respectively, strong and weak influence. Incoming influence is represented by red lines around the areas that influence the area under investigation (i.e., AI that plays to win in this figure): strong and weak influences are represented, respectively, by a solid and a dashed line.

Research in this area impacts game AI at large as three game AI subareas are directly affected; in turn, one subarea is directly affecting AI that plays to win (see Fig. 6.3).

• Playing to Win → Playing for the Experience: An agent cannot be believable or existent to augment the game's experience if it is not proficient. Being able to play a game well is in several ways a precondition for playing games in a be lievable manner though well playing agents can be developed without learning (e.g., via top-down approaches). In recent years, successful entries to competi tions focused on believable agents, such as the 2K BotPrize and the Mario AI Championship Turing test track, have included a healthy dose of learning algo rithms [719, 603].

• Playing to Win → Generate Content (Autonomously): Having an agent that is capable of playing a game proficiently is useful for simulation-based testing in procedural content generation, i.e., the testing of newly generated game con tent by playing through that content with an agent. For example, in a program generating levels for the platform game Super Mario Bros (Nintendo, 1985), the levels can be tested by allowing a trained agent to play them; those that the agent cannot complete can be discarded [335]. Browne's Ludi system, which generates complete board games, evaluates these games through simulated playthrough and uses learning algorithms to adapt the strategy to each game [74].

• Playing to Win → Generate Content (Assisted): Just as with autonomous pro cedural content generation, many tools for AI-assisted game design rely on being able to simulate playthroughs of some aspect of the game. For instance, the Sen tient Sketchbook tool for level design uses simple simulations of game-playing agents to evaluate aspects of levels as they are being edited by a human de-

signer [379]. Another example is the automated playtesting framework named Restricted Play [295] which aims mostly at assisting designers on aspects of game balance during game design. A form of Restricted Play is featured in the Ludocore game engine [639].

6.2.1.2 Playing for the Experience (as a Player or as a Non-Player)

Research on AI that plays a game for a purpose other than winning is central to studies where playing the game well is not the primary research aim. AI can play the game as a player character attempting to maximize the believability value of play as, for instance, in [719, 619, 96]. It can alternatively play the game in a role of an NPC for the same purpose [268]. Work under this research subarea involves the study of believability, interestingness or playing experience in games and the inves tigations of mechanisms for the construction of agent architectures that appear to have e.g., believable or human-like characteristics. The approaches for developing such architectures can be either top-down behavior authoring (such as the FAtiMA model used in My Dream Theatre [100] and the Mind Module model [191] used in The Pataphysic Institute) or bottom-up attempting to imitate believable gameplay from human players such as the early work of Thurau et al. in Quake II (Activision,

1997) bots [696], the human imitation attempts in Super Mario Bros (Nintendo, 1985) [511], the Unreal Tournament 2004 (Epic Games, 2004) believable bots of Schrum et al. [603] and the crowdsourcing studies of the Restaurant game [508].
Evidently, commercial games have for long benefited from agent believability re search. Examples of this include popular games such as the Sims (Electronic Arts, 2000) series. The industry puts a strong emphasis on the design of believability in games as this contributes to more immersive game environments. The funding of believability research through game AI competitions such as the 2K BotPrize is one of the many clear indicators of the commercial value of agent believability.
Over the last few years there has been a growing academic (and commercial) interest in the establishment of competitions that can be used as assessment tools for agent believability [719]. Agent believability research has provided input and given substance to those game benchmarks. A number of game Turing competitions have been introduced to the benefit of agent believability research, including the 2K BotPrize on the Unreal Tournament 2004 (Epic Games, 2004) [647, 264] game and the Mario AI Championship: Turing test track [619] on the Super Mario Bros (Nintendo, 1985) game. Recently, the community saw AI agents passing the Turing test in the 2K BotPrize [603].
The study of AI that plays games not for winning, but for other purposes, affects research on three other game AI areas as illustrated in Fig. 6.4, whereas it is affected by four other game AI areas.
• Playing for the Experience → Model Players (Experience and Behavior):
There is a direct link between player modeling and believable agents as research carried out for the modeling of human, human-like, and supposedly believable playing behavior can inform the construction of more appropriate models for

Fig. 6.4 Playing for the Experience: influence on (and from) other game AI research areas.
players. Examples include the imitation of human play styles in Super Mario Bros (Nintendo, 1985) [511] and Quake II (Activision, 1997) [696]. Though computational player modeling uses learning algorithms, it is only in some cases that it is the behavior of an NPC that is modeled. In particular, this is true when the in-game behavior of one or several players is modeled. This can be done using either reinforcement learning techniques, or supervised learning techniques such as backpropagation or decision trees. In either case, the intended outcome for the learning algorithm is not necessarily an NPC that plays as well as possible, but one that plays in the style of the modeled player [735, 511].
• Playing for the Experience → Generate Content (Autonomously): Believ able characters may contribute to better levels [96], more believable stories [801, 401, 531] and, generally, better game representations [563]. A typical ex ample of the integration of characters in the narrative and the drive of the lat ter based on the former includes the FAtiMa agents in FearNot! [516] and My Dream Theater [100]. Another example is the generation of Super Mario Bros (Nintendo, 1985) levels that maximize the believability of any Mario player [96].

6.2.2 Generate Content

As covered in detail in Chapter 4 AI can be used to design whole (or parts of) games in an autonomous or in an assisted fashion. This core game AI area includes the subareas of autonomous (procedural) content generation and assisted or mixed initiative (procedural) content generation. The interactions of these subareas with the remaining areas of game AI are covered in this section.

Fig. 6.5 Generate Content (Autonomously): influence on (and from) other game AI research areas.

6.2.2.1 Generate Content (Autonomously)

As stated in Chapter 4 procedural content generation has been included in limited roles in some commercial games since the early 1980s; however, recent years have seen an expansion of research on more controllable PCG for multiple types of game content [764], using techniques such as evolutionary search [720] and constraint solving [638]. The influence of PCG research beyond games is already evident in

areas such as computational creativity [381] and interaction design (among others). There are several surveys of PCG available, including a recent book [616] and vision paper [702], as well as surveys of frameworks [783], sub-areas of PCG [554, 732] and methods [720, 638].

Autonomous content generation is one of the areas of recent academic research on AI in games which bears most promise for incorporation into commercial games. A number of recent games have been based heavily on PCG, including independent ("indie") game production successes such as Spelunky (Mossmouth, 2009) and Minecraft (Mojang, 2011), and mainstream AAA games such as Diablo III (Blizzard Entertainment, 2012) and Civilization V (2K Games, 2010). A notable example, as mentioned in Chapter 4 is No Man's Sky (Hello Games, 2016) with its quintillion different procedurally generated planets. Some games heavily based on PCG and developed by researchers have been released as commercial games on plat forms such as Steam and Facebook; two good examples of this are Petalz [565, 566] and Galactic Arms Race [249].

Figure 6.5 depicts the three (and five) areas that are influenced by (and influence) autonomous PCG.

• Generate Content (Autonomously) → Play to Win: If an agent is trained to perform well in only a single game environment, it is easy to overspecialize the

training and arrive at a policy/behavior that will not generalize to other levels. Therefore, it is important to have a large number of environments available for training. PCG can help with this, potentially providing an infinite supply of test environments. For example, when training players for the Mario AI Champi onship it is common practice to test each agent on a large set of freshly generated levels, to avoid overtraining [322]. There has also been research on adapting NPC behavior specifically to generated content [332]. Finally, one approach to artificial general intelligence is to train agents to be good at playing games in general, and test them on a large variety of games drawn from some genre or dis tribution. To avoid overfitting, this requires games to be generated automatically, a form of PCG [598]. The generation of new environments is very important for NPC behavior learning, and this extends to benchmarks that measure some aspect of NPC behavior. Apart from the Mario AI Championship, competitions such as the Simulated Car Racing Championship use freshly generated tracks, unseen by the participants, to test submitted controllers [102]. But there is also scope for benchmarks and competitions focused on measuring the capabilities of PCG systems themselves, such as the Level Generation track of the Mario AI Championship [620].

◦ Generate Content (Autonomously) → Play for the Experience: Research on autonomous PCG naturally influences research on agent (PC or NPC) control for believability, interestingness or other aims aside from winning given that these agents are performing in a particular environment and under a specific game context. This influence is still in its infancy and the only study we can point the reader to is the one by Camilleri et al. [96] where the impact of level design on player character believability is examined in Super Mario Bros (Nintendo, 1985). Further, research on interactive narrative benefits from and influences the use of believable agents that interact with the player and are interwoven in the story plot. The narrative can yield more (or less) believability to agents and thus the relationship between the behavior of the agents and the emergent story is strong [801, 401, 531]. In that sense, the computational narrative of a game may define the arena for believable agent design.

• Generate Content (Autonomously) → Generate Content (Assisted): As con tent design is a central part of game design, many AI-assisted design tools incor porate some form of assisted content design. Examples include Tanagra, which helps designers create complete platform game levels which ensure playabil ity through the use of constraint solvers [641], and SketchaWorld [634]. An other example is Sentient Sketchbook, which assists humans in designing strategy

game levels through giving immediate feedback on properties of levels and au tonomously suggesting modifications [379].

6.2.2.2 Generate Content (Assisted)

Assisted content generation refers to the development of AI-powered tools that sup port the game design and development process. This is perhaps the AI research area

Fig. 6.6 Generate Content (Assisted): influence on (and from) other game AI research areas.

which is most promising for the development of better games [764]. In particu lar, AI can assist in the creation of game content varying from levels and maps to game mechanics and narratives. The impact of AI-enabled authoring tools on design and development influences the study of AI that plays games for believability, in terestingness or player experience, and research in autonomous procedural content generation (see Fig. 6.6). AI-assisted game design tools range from those designed to assist with generation of complete game rulesets such as MetaGame [522] or RuLearn [699] to those focused on more specific domains such as strategy game levels [379], platform game levels [642], horror games [394] or physics-based puz zles [613]. It is worth noting that AI tools have been used extensively for supporting design and commercial game development. Examples such as the SpeedTree (Interactive Data Visualization Inc., 2013) generator for trees and other plants [287] have seen uses in several game productions. The mixed-initiative PCG tools mentioned above have a great potential in the near future as most of these are already tested on com mercial games or developed with game industrial partners. Furthermore, there are tools designed for interactive modeling and analysis of game rules and mechanics, which are not focused on generating complete games but on prototyping and un derstanding aspects of complex games; such systems could be applied to existing commercial games [639].

◦ Generate Content (Assisted) → Play for the Experience: Authoring tools in forms of open-world sandboxes could potentially be used for the creation of more believable behaviors. While this is largely still an unexplored area of research and development, notable attempts include the NERO game AI platform where players can train game agents for efficient and believable first-person shooter bot behaviors [654]. An open version of this platform focused on crowdsourcing behaviors has been released recently [327]. A similar line of research is the gen-

eration of Super Mario Bros (Nintendo, 1985) players by means of interactive evolution [648].

• Generate Content (Assisted) → Generate Content (Autonomously): Re search on methods of mixed-initiative co-creation [774] and design can feed input to and spur discussion on central topics in procedural content genera tion. Given the importance of content design in the development process as a whole, any form of mixed-initiative AI assistance in the generation process can support and augment procedural content generation. Notable examples of mixed-initiative PCG include the Tanagra platform game level design AI assis tant [641], and the SketchaWorld [634], the Sentient World [380], the Sentient Sketchbook [379, 774] and the Sonancia [394] systems which generate game maps and worlds in a mixed-initiative design fashion following different ap proaches and levels of human computation. Further, tools can assist the author ing of narrative in games. In particular, drama management tools have long been investigated within the game AI community. An academic example is ABL which has allowed the authoring of narrative in Fac¸ade [441]. Among the few available and well-functional authoring tools the most notable is the Versu [197] storytelling system which was used in the game Blood & Laurels (Emily Short, 2014) and the Inform 7 [480] software package that led to the design of Mystery House Possessed (Emily Short, 2005). More story generation tools as such can be found at the http://storygen.org/ repository, by Chris Martens and Rogelio E. Cardona-Rivera.

6.2.3 Model Players

As already explored in Chapter 5, modeling players involves the subtasks of mod eling their behavior or their experience. Given the interwoven nature of these two tasks we present their influences to (and from) other game AI areas under one com mon section. In player modeling [782, 636], computational models are created for detecting how the player perceives and reacts to gameplay. As stated in Chapter 5 such models are often built using machine learning methods where data con sisting of some aspect of the game or player-game interaction is associated with labels derived from some assessment of player experience, gathered for example from questionnaires [781]. However, the area of player modeling is also concerned with structuring observed player behavior even when no correlates to experience are available—e.g., for identifying player types or predicting player behavior.

Research and development in player modeling can inform attempts for player ex perience in commercial-standard games. Player experience detection methods and algorithms can advance the study of user experience in commercial games. In addi tion, the appropriateness of sensor technology, the technical plausibility of biofeed back sensors, and the suitability of various modalities of human input can inform in dustrial developments. Quantitative testing via game metrics—varying from behav ioral data mining to in-depth low scale studies—is also improved [764, 178, 186].

Fig. 6.7 Model Players: influence on (and from) other game AI research areas.

By now, a considerable number of academic studies use directly datasets from com mercial games to induce models of players that could inform further development of the game. For example, we refer the reader to the experiments in clustering players of Tomb Raider: Underworld (Square Enix, 2008) into archetypes [176] and pre dicting their late-game performance based on early-game behavior [414]. Examples of player modeling components within high-profile commercial games include the arousal-driven appearance of NPCs in Left 4 Dead 2 (Valve Corporation, 2009), the fearful combat skills of the opponent NPCs in F.E.A.R. (Monolith, 2005), and the avatars' emotion expression in the Sims series (Maxis, 2000) and Black and White (Lionhead Studios, 2001). A notable example of a game that is based on player expe rience modeling is Nevermind (Flying Mollusk, 2016); the game adapts its content based on the stress of the player, which is manifested via a number of physiological sensors.

Player modeling is considered to be one of the core non-traditional uses of AI in games [764] and affects research in AI-assisted game design, believable agents, computational narrative and procedural content generation (see Fig. 6.7).

• Model Players → Play for the Experience: Player models can inform and up date believable agent architectures. Models of behavioral, affective and cognitive aspects of gameplay can improve the human-likeness and believability of any agent controller—whether it is ad-hoc designed or built on data derived from gameplay. While the link between player modeling and believable agent design is obvious and direct, research efforts towards this integration within games are still sparse. However, the few efforts made on the imitation of human game play ing for the construction of believable architectures have resulted in successful outcomes. For example, human behavior imitation in platform [511] and racing games [735, 307] has provided human-like and believable agents while similar approaches for developing Unreal Tournament 2004 (Epic Games, 2004) bots (e.g., in [328]) recently managed to pass the Turing test in the 2K BotPrize com-

petition. Notably, one of the two agents that passed the Turing test in 2K BotPrize managed to do so by imitating (mirroring) aspects of human play [535]. A line of work that stands in between player modeling and playing games for the ex perience is the study on procedural personas [268, 267, 269]. As introduced in Chapter 5 procedural personas are NPCs that are trained to imitate realistically the decision making process of humans during play. Their study both influences our understanding about the internal (cognitive) processes of playing behavior and advances our knowledge on how to build believable characters in games.

• Model Players → Generate Content (Autonomously): There is an obvious link between computational models of players and PCG as player models can drive the generation of new personalized content for the player. The experience driven role of PCG [783], as covered in Chapter 4, views game content as an indirect building block of a player's affective, cognitive and behavioral state and proposes adaptive mechanisms for synthesizing personalized game experiences. The "core loop" of an experience-driven PCG solution involves learning a model that can predict player experience, and then using this model as part of an evalua tion function for evolving (or otherwise optimizing) game content; game content is evaluated based on how well it elicits a particular player experience, accord ing to the model. Examples of PCG that are driven by player models include the generation of game rules [716], camera profiles [780, 85] and platform game levels [617]. Most work that goes under the label "game adaptation" can be said to implement the experience-driven architecture; this includes work on adapting the game content to the player using reinforcement learning [28] or semantic constraint solving [398] rather than evolution. Player models may also inform the generation of computational narrative. Predictive models of playing experi ence can drive the generation of individualized scenarios in a game. Examples of the coupling between player modeling and computational narrative include the affect-driven narrative systems met in Fac¸ade [441] and FearNot! [26], and the affect-centered game narratives such as the one of Final Fantasy VII (Square, 1997).

◦ Model Players → Generate Content (Assisted): User models can enhance au thoring tools that, in turn, can assist the design process. The research area that bridges user modeling and AI-assisted design is in its infancy and only a few ex ample studies can be identified. Indicatively, designer models [378] have been employed to personalize mixed-initiative design processes [774, 377, 379]. Such models may drive the procedural generation of designer-tailored content.

6.3 The Road Ahead

This chapter has initially identified the currently most active areas and subareas within game AI and placed them on three holistic frameworks: an AI method map ping, a game stakeholder (end user) taxonomy and the player-game interaction loop. This analysis revealed dominant AI algorithms within particular areas as well as

room for exploration of new methods within areas. In addition, it revealed the dis similar impact of different areas on different end users such as the AI researcher and the designer and, finally, outlined the influence of the different game AI areas on the player, the game and their interaction. From the high-level analysis of the game AI field we moved on to the detailed analysis of the game AI areas that compose it and thoroughly surveyed the meaningful interconnections between the different areas. The total number of strong and weak influences is rather small compared to all possible interconnections between the areas, which clearly signals the research ca pacity of the game AI field for further explorations. We can distinguish a number of connections which are currently very active, meaning that much work currently goes on in one area that draws on work in another area. Here we see, for example, the connection between AI that plays to win in a general fashion in conjunction with the use of tree search algorithms: the MCTS algorithm was invented in the context of board game-playing, proved to be really useful in the general game playing compe tition, and is being investigated for use in games as different as StarCraft (Blizzard Entertainment, 1998) and Super Mario Bros (Nintendo, 1985). Improvements and modifications to the algorithm have been flowing back and forth between the vari ous areas. Another indicative connection that is alive and strong is between player modeling and procedural content generation, where it is now common for newly devised PCG algorithms and experimental studies to include player behavioral or player experience models.

One can also study the currently strong areas by trying to cluster the trending topics in recent iterations of the IEEE CIG and AIIDE conferences. Such studies

always include some form of selection bias, as papers can usually be counted into more than one area (e.g., depending on if you group by method or domain), but if you start from the session groupings made by the program chairs of each conference you achieve at least some inter-subjective validity. According to such a clustering, the most active topics over the last few years have been player (or emotion) model ing, game analytics, general game AI, real-time strategy game playing—especially StarCraft (Blizzard Entertainment, 1998)—and PCG (in general). Another perspec tive of the trend in game AI research is the varying percentage of studies on NPC (or game agent) behavior learning over other uses of AI in games at the two key conferences in the field (IEEE CIG and AIIDE). Our preliminary calculations sug gest that while, initially, AI was mainly applied for NPC control and for playing board/card games well—more than 75% of CIG and AIIDE papers were linked to NPC behavior and agent game playing in 2005—that trend has drastically changed as entirely new (non-traditional) uses of AI became more common over the years—e.g., roughly 52% of the papers in CIG and AIIDE in 2011 did not involve game agents and NPC behavior. These facts indicate a shift in the use of AI in and for games towards multiple non-traditional applications—which tend to be traditional by now—for the development of better games [764].

But it is maybe even more interesting to look at all those connections that are unexploited or underexploited or potentially strong. For example, player modeling is potentially very important in the development of AI that controls believable, in teresting or curious agents, but this has not been explored in enough depth yet; the

same holds for the application of user (or else, designer) modeling principles to wards the personalization of AI-assisted game design. Believable agents have, in turn, not been used enough in content generation (either autonomous or assisted).

A grand vision of game AI for the years to come is to let it identify its own role within game design and development as it sees fit. In the last chapter of this book we discuss frontier research topics as such and identify unexplored roles of AI in games.

6.4 Summary

We hope that with this chapter of this book, we have been able to give our readers a sense of how this—by now rather large and multifaceted—research field hangs together, and what could be done to integrate it further. We realize that this is only our view of its dynamics and interconnections, and that there are (or could be) many competing views. We look forward to seeing those in upcoming studies in the field.

Finally, it is important to note that it would have been impossible to provide a complete survey of all the areas as, first, the game AI field is growing rapidly and, second, it is not the core objective of the book. This means that the bibliography is indicative rather than exhaustive and serves as a general guideline for the reader. The website of the book, instead of the book per se, will be kept up to date regarding important new readings for each area.

The next and final chapter of the book is dedicated to a few long-standing, yet rather unexplored, research frontiers of game AI. We believe that any advances made in these directions will lead to scientific breakthroughs not merely within game AI but largely in both games (their design, technology and analysis) and AI per se.

Chapter 7
Frontiers of Game AI Research

In this final chapter of the book we discuss a number of long-term visionary goals of game AI, putting an emphasis on the generality of AI and the extensibility of its roles within games. In particular, in Section 7.1 we discuss our vision for general behavior for each one of the three main uses of AI in games. Play needs to be come general; generators are required to have general generative capacities across games, content types, designers and players; models of players also need to show case general modeling abilities. In Section 7.2 we also discuss roles of AI that are still unexplored and certainly worth investigating in the future. The book ends with a discussion dedicated to general ethical considerations of game AI (Section 7.3).

7.1 General General Game AI

As evidenced from the large volume of studies the game AI research area has been supported by an active and healthy research community for more than a decade—at least since the start of the IEEE CIG and the AIIDE conference series in 2005. Before then, research had been conducted on AI in board games since the dawn of automatic computing. Initially, most of the work published at IEEE CIG or AI IDE was concerned with learning to play a particular game as well as possible, or using search/planning algorithms to play a game as well as possible without learn ing. Gradually, a number of new applications for AI in games and for games in AI have come to complement the original focus on AI for playing games [764]. Papers on procedural content generation, player modeling, game data mining, human-like playing behavior, automatic game testing and so on have become commonplace within the community. As we saw in the previous chapter there is also a recogni tion that all these research endeavors depend on each other [785]. However, almost all research projects in the game AI field are very specific. Most published papers describe a particular method—or a comparison of two or more methods—for per forming a single task (playing, modeling, generating, etc.) in a single game. This is problematic in several ways, both for the scientific value and for the practical appli-

cability of the methods developed and studies made in the field. If an AI approach is only tested on a single task for a single game, how can we argue that is an ad vance in the scientific study of artificial intelligence? And how can we argue that it is a useful method for a game designer or developer, who is likely working on a completely different game than the one the method was tested on?

As discussed in several parts of this book general game playing is an area that has already been studied extensively and constitutes one of the key areas of game AI [785]. The focus of generality solely on play, however, is very narrow as the possible roles of AI and general intelligence in games are many, including game design, content design and player experience design. The richness of the cognitive skills and affective processes required to successfully complete these tasks has so far been largely ignored by game AI research. We thus argue, that while the focus on general AI needs to be retained, research on general game AI needs to expand beyond mere game playing. The new scope for general general game AI beyond game-playing broadens the applicability and capacity of AI algorithms and our understanding of intelligence as tested in a creative domain that interweaves problem solving, art, and engineering.

For general game AI to eventually be truly general, we argue that we need to extend the generality of general game playing to all other ways in which AI is (or can be) applied to games. More specifically we argue that the field should move towards methods, systems and studies that incorporate three different types of generality:

1. Game generality. We should develop AI methods that work with not just one game, but with any game (within a given range) that the method is applied to.
2. Task generality. We should develop methods that can do not only one task (play ing, modeling, testing, etc) but a number of different, related tasks.
3. User/designer/player generality. We should develop methods that can model, respond to and/or reproduce the very large variability among humans in design style, playing style, preferences and abilities.

We further argue that all of this generality can be embodied into the concept of general game design, which can be thought of as a final frontier of AI research within games. Further details about the notion of general general game AI can be found in the vision paper we co-authored about this frontier research area [718]. It is important to note that we are not arguing that more focused investigations into methods for single tasks in single games are useless; these are often important as proofs-of-concept or industrial applications and they will continue to be important in the future, but there will be an increasing need to validate such case studies in a more general context. We are also not envisioning that everyone will suddenly start

working on general methods. Rather, we are positing generalizations as a long-term goal for our entire research community. Finally, the general systems of game AI that we envision ought to have a real-world use. There is a risk that by making systems too general we might end up not finding applications of these general systems to any specific real-world problem. Thus, the system's applicability (or usefulness) sets our core constraint towards this vision of general game AI. More specifically, we envi-

sion general general game AI systems that are nevertheless integrated successfully within specific game platforms or game engines.

7.1.1 General Play

The problem of playing games is the one that has been most generalized so far. There already exist at least three serious benchmarks or competitions attempting to pose the problem of playing games in general, each in its own imperfect way. The Gen eral Game Playing Competition, often abbreviated GGP [223], the Arcade Learning Environment [40] and the General Video Game AI competition [528]; all three have been discussed in various places in this book. The results from these competitions so far indicate that general purpose search and learning algorithms by far outper form more domain-specific solutions and "clever hacks". Somewhat simplified, we can say that variations of Monte Carlo tree search perform best on GVGAI and GGP [202], and for ALE (where no forward model is available so learning a policy for each game is necessary) reinforcement learning with deep networks [464] and search-based iterative width [389, 301, 390] perform best. This is a very marked difference from the results of the game-specific competitions, indicating the lack of domain-independent solutions.

While these are each laudable initiatives and currently the focus of much re search, in the future we will need to expand the scope of these competitions and benchmarks considerably, including expanding the range of games available to play and the conditions under which gameplay happens. We need game playing bench marks and competitions capable of expressing any kind of game, including puzzle games, 2D arcade games, text adventures, 3D action-adventures and so on; this is the best way to test general AI capacities and reasoning skills. We also need a number of different ways of interfacing with these games—there is room for both benchmarks that give agents no information beyond the raw screen data but give them hours to learn how to play the game, and those that give agents access to a forward model and perhaps the game code itself, but expect them to play any game presented to them with no time to learn. These different modes test different AI capabilities and tend to privilege different types of algorithms. It is worth noting that the GVGAI competition is currently expanding to different types of playing modes, and has a long-term goal to include many more types of games [527].

We also need to differentiate away from just measuring how to play games op timally. In the past, several competitions have focused on agents that play games in a human-like manner; these competitions have been organized similarly to the classic Turing test [263, 619]. Playing games in a human-like manner is important for a number of reasons, such as being able to test levels and other game content as part of search-based generation, and to demonstrate new content to players. So far, the question of how to play games in a human-like manner in general is mostly unexplored; some preliminary work is reported in [337]. Making progress here will likely involve modeling how humans play games in general, including characteris-

tics such as short-term memory, reaction time and perceptual capabilities, and then translating these characteristics to playing style in individual games.

7.1.2 General Game Generation and Orchestration

The study of PCG [616] for the design of game levels has reached a certain matu rity and is, by far, the most popular domain for the application of PCG algorithms and approaches (e.g., see [720, 785, 783] among many). What is common in most of the content generation studies covered in this book, however, is their specificity and strong dependency of the representation chosen on the game genre examined.

For the Mario AI Framework, for instance, the focus on a single level generation problem has been very much a mixed blessing: it has allowed for the proliferation and simple comparison of multiple approaches to solving the same problem, but has also led to a clear overfitting of methods. Even though some limited generalization is expected within game levels of the same genre, the level generators that have been explored so far clearly do not have the capacity of general level design. We argue that there needs to be a shift in how level generation is viewed. The obvious change of perspective is to create general level generators—level generators with general intelligence that can generate levels for any game (within a specified range). That would mean that levels are generated successfully across game genres and players and that the output of the generation process is meaningful and playable as well as entertaining for the player. Further, a general level generator should be able to coor dinate the generative process with the other computational game designers who are responsible for the other parts of the game design.

To achieve general level design intelligence algorithms are required to capture as much of the level design space as possible at different representation resolutions. We can think of representation learning approaches such as deep autoencoders [739] capturing core elements of the level design space and fusing various game genres within a sole representation—as already showcased by a few methods, such as the Deep Learning Novelty Explorer [373]. The first attempt to create a benchmark for general level generation has recently been launched in the form of the Level Genera tion Track of the GVGAI competition. In this competition track, competitors submit level generators capable of generating levels for unseen games. The generators are then supplied with the description of several games, and produce levels which are judged by human judges [338]. Initial results suggest that constructing competent level generators that can produce levels for any game is much more challenging than constructing competent level generators for a single game. A related effort is the Video Game Level Corpus [669] which aims to provide a set of game levels across multiple games and genres which can be used for training level generators for data-driven procedural content generation.

While level generation, as discussed above, is one of the main examples of proce dural content generation, there are many other aspects (or facets) of games that can be generated. These include visuals, such as textures and images; narrative, such

as quests and backstories; audio, such as sound effects and music; and of course all kinds of things that go into game levels, such as items, weapons, enemies and personalities [381, 616]. However, an even greater challenge is the generation of complete games, including some or all of these facets together with the rules of the game. While, as covered in Chapter 4, there have been several attempts to gener ate games (including their rules) we are not aware of any approach to generating games that tries to generate more than two of the facets of games listed above. We are also not aware of any game generation system that even tries to generate games of more than one genre. Multi-faceted generation systems like Sonancia [394, 395] co-generate horror game levels with corresponding soundscapes but do not cater to the generation of rules. It is clear that the very domain-limited and facet-limited as pects of current game generation systems result from intentionally limiting design choices in order to make the very difficult problem of generating complete games tractable. Yet, in order to move beyond what could be argued to be toy domains and start to fulfill the promise of game generation, we need systems that can generate multiple facets of games at the same time, and that can generate games of different kinds.

This process has been defined as facet (domain) orchestration in games [371, 324]. Orchestration refers to the process of harmonizing game generation. Evi dently, orchestration is a necessary process when we consider the output of two or more content type generators—such as visuals and audio—for the generation of a complete game. Drawing inspiration from music, orchestration may vary from a top-down, conductor-driven process to a bottom-up, free-from generation process

[371]. A few years ago, something very much like general game generation and or chestration was outlined as the challenges of “multi-content, multi-domain PCG” and “generating complete games” [702]. It is interesting to note that there has not seemingly been any attempt to create more general game generators since then, per haps due to the complexity of the task. A recent study by Karavolos et al. [324] moves towards the orchestration direction as it fuses level and game design param eters in first-person shooters via deep convolutional neural networks. The trained networks can be used to generate balanced games. Currently the only genre for which generators have been built that can generate high-quality (complete) games is abstract board games. Once more genres have been “conquered”, we hope that the task of building more general level generators can begin.

Linked to the tasks of orchestration and general game generation there are im portant questions with respect to the creative capacity of the generation process that remain largely unanswered. For example, how creative can a generator be and how can we assess it? Is it, for instance, deemed to have appreciation, skill, and imagina tion [130]? When it comes to the evaluation of the creative capacity of current PCG algorithms a case can be made that most of them possess only skill. Does the creator manage to explore novel combinations within a constrained space, thereby resulting in exploratory game design creativity [53]; or, is on the other hand trying to break existing boundaries and constraints within game design to come up with entirely new designs, demonstrating transformational creativity [53]? If used in a mixed initiative fashion, does it enhance the designer’s creativity by boosting the possi-

bility space for her? Arguably, the appropriateness of various evaluation methods for autonomous PCG creation or mixed-initiative co-creation [774] remains largely unexplored within both human and computational creativity research.

7.1.3 General Game Affective Loop

It stands to reason that general intelligence implies (and is tightly coupled with) general emotional intelligence [443]. The ability to recognize human behavior and emotion is a complex yet critical task for human communication that acts as a fa cilitator of general intelligence [157]. Throughout evolution, we have developed particular forms of advanced cognitive, emotive and social skills to address this challenge. Beyond these skills, we also have the capacity to detect affective patterns across people with different moods, cultural backgrounds and personalities. This generalization ability also extends, to a degree, across contexts and social settings. Despite their importance, the characteristics of social intelligence have not yet been transferred to AI in the form of general emotive, cognitive or behavioral models. While research in affective computing [530] has reached important milestones such as the capacity for real-time emotion recognition [794]—which can be faster than humans under particular conditions—all key findings suggest that any success of affective computing is heavily dependent on the domain, the task at hand, and the context in general. This specificity limitation is particularly evident in the domain of games [781] as most work in modeling player experience focuses on particu lar games, under well-controlled conditions with particular, small sets of players (see [783, 609, 610, 435] among many). In this section we identify and discuss two core unexplored and interwoven aspects of modeling players that are both important and necessary steps towards the long-term aim of game AI to realize truly adaptive games. The first aspect is the closure of the affective loop in games; the second as pect is the construction of general models capable of capturing experience across players and games.

As stated at the start of this book, affective computing is best realized within games in what we name the game affective loop. While the phases of emotion elic itation, affect modeling and affect expression have offered some robust solutions by now, the very loop of affective-based interaction has not been closed yet. Aside from a few studies demonstrating some affect-enabled adaptation of the game [772, 617] the area remains largely unexplored. It is not only the complexity of modeling play ers and their experience that is the main hurdle against any advancement. What is

also far from trivial is the appropriate and meaningful integration of any of these models in a game. The questions of how often the system should adapt, what it should alter and by what degree are not easy to answer. As most of the questions are still open to the research community the only way to move forward is to do more research in adaptive games involving affective aspects of the experience. Existing commercial-standard games that already realize the affective loop such as Never mind (Flying Mollusk, 2016) are the ambassadors for further work in this area.

7.2. AI in Other Roles in Games 285

Once the game affective loop is successfully realized within particular games the next goal for game AI is the generality of affect-based interaction across games. The game affective loop should not only be operational; it should ideally be general too. For AI in games to be general beyond game-playing it needs to be able to recog nize general emotional and cognitive-behavioral patterns. This is essentially AI that can detect context-free emotive and cognitive reactions and expressions across con texts and builds general computational models of human behavior and experience which are grounded in a general gold standard of human behavior. So far we have only seen a few proof-of-concept studies in this direction. Early work within the game AI field focused on the ad-hoc design of general metrics of player interest that were tested across different prey-predator games [768, 767]. In other, more recent, studies predictors of player experience were tested for their ability to capture player experience across dissimilar games [431, 612, 97]. Another study on deep multi modal fusion can be seen as an embryo for further research in this direction [435], in which various modalities of player input such as player metrics, skin conductance and heart activity have been fused using stacked autoencoders. Discovering entirely new representations of player behavior and emotive manifestations across games, modalities of data, and player types is a first step towards achieving general player modeling. Such representations can, in turn, be used as the basis for approximating the ground truth of user experience in games.

7.2 AI in Other Roles in Games

The structure of this book reflects our belief that playing games, generating content and modeling players are the central applications of AI methods in games. How ever, there are many variants and use cases of game playing, player modeling or content generation that we have not had time to explore properly in the book, and which in some cases not have been explored in the literature at all. Further, there are some applications of AI in games that cannot be really classified as special cases of our “big three” AI applications in games, despite our best efforts. This section briefly sketches some of these applications, some of which may be important future research directions.

Playtesting: One of the many use cases for AI for playing games is to test the games. Testing games for bugs, balancing player experience and behavior, and other issues is important in game development, and one of the areas where game devel opers are looking for AI assistance. While playtesting is one of the AI capabilities within many of the mixed-initiative tools discussed in Chapter 4, there has also been work on AI-based playtesting outside of that context. For example, Denzinger et al. evolved action sequences to find exploits in sports games, with discouragingly good results [165]. For the particular case of finding bugs and exploits in games, one of the research challenges is to find a good and representative coverage of problems, so as to deliver an accurate picture to the development team of how many problems

there are and how easy they are to run into, and allow prioritization of which prob lems to fix.

Critiquing Games: Can AI methods meaningfully judge and critique games? Game criticism is hard and generally depends on deep understanding of not only games but also the surrounding cultural context. Still, there might be automated metrics that are useful for game criticism, and can provide information to help reviewers, game cu rators and others in selecting which games to consider for reviewing for inclusion in app stores. The ANGELINA game generation system is one of the few examples towards this direction [136] in which AI generates the overview of the game to be

played.

Hyper-formalist Game Studies: AI methods can be applied to corpora of games in order to understand distributions of game characteristics. For example, decision trees can be used to visualize patterns of resource systems in games [312]. There are likely many other ways of using game AI for game studies that are still to be discovered.

Game Directing: The outstanding feature of Left 4 Dead (Valve Corporation, 2008) was its AI director, which adjusted the onslaught of zombies to provide a dramatic curve of challenge for players. While simple and literally one-dimensional (only a single dimension of player experience was tracked), the AI director proved highly effective. There is much room for creating more sophisticated AI directors; the experience-driven PCG framework [783] is one potential way within which to work towards this.

Creative Inspiration: While designing a complete game that actually works likely requires a very complex generator, it can be simpler to generate an idea for new games, that are then designed by humans. Creative ideation tools range from sim ple word-recombination-based tools implemented as card games or Twitter bots, to elaborate computational creativity systems such as the What If-Machine [391].

Chat Monitoring: In-game chats are important in many online multi-player games, as they allow people to collaborate within games and socialize through them. Un fortunately, such chats can also be used to threaten or abuse other players. Given the very large volume of chat messages sent through a successful online game, it becomes impossible for the game developers to curate chats manually. In the efforts to combat toxic behavior, some game developers have therefore turned to machine learning. Notably, Riot Games have trained algorithms to recognize and remove toxic behavior in the MOBA League of Legends (Riot Games, 2009) [413]. Even worse, sexual predation can be seen in some games, where pedophiles use game chats to reach children; there have been attempts to use machine learning to detect sexual predators in game chats too [241].

AI-Based Game Design: Throughout most of the book, we have assumed the ex-

istence of a game or at least a game design, and discussed how AI can be used to play that game, generate content for it or model its players. However, one could also start from some AI method or capability and try to design a game that builds on that method or capability. This could be seen as an opportunity to showcase AI methods in the context of games, but it could also be seen as a way of advancing game de sign. Most classic game designs originate in an era where there were few effective AI algorithms, there was little knowledge among game designers about those AI algorithms that existed, and CPU and memory capacity of home computers was too limited to allow anything beyond simple heuristic AI and some best-first search to be used. One could even say that many classic video game designs are an attempt to design around the lack of AI—for example, the lack of good dialog AI for NPCs led to the use of dialog trees, the lack of AIs that could play FPS games believably and competently led to FPS game designs where most enemies are only on-screen for a few seconds so that you do not notice their lack of smarts, and the lack of level generation methods that guaranteed balance and playability led to game de signs where levels did not need to be completable. The persistence of such design patterns may be responsible for the relatively low utilization of interesting AI meth ods within commercial game development. By starting with the AI and designing a game around it, new design patterns that actually exploit some of the recent AI advances can be found.

Several games have been developed within the game AI research community specifically to showcase AI capabilities, some of which have been discussed in this book. Three of the more prominent examples are based on Stanley et al.'s work on neuroevolution and the NEAT algorithm: NERO, which is an RTS-like game where the player trains an army through building a training environment rather

than controlling it directly [654]; Galactic Arms Race, in which weapons con trolled through neural networks are indirectly collectively evolved by thousands of players [250, 249]; and Petalz, which is a Facebook game about collecting flow ers based on a similar idea of selection-based collective neuroevolution [565, 566].

Other games have been built to demonstrate various adaptation mechanisms, such as Infinite Tower Defense [25] and Maze-Ball [780]. Within interactive narrative it is relatively common to build games that showcase specific theories and methods; a fa mous example is Fac¸ade [441] and another prominent example is Prom Week [447].

Treanor et al. have attempted to identify AI-based game design patterns, and found a diverse array of roles in which AI can be or has been used in games, and a number of avenues for future AI-based game design [724].

7.3 Ethical Considerations

Like all technologies, artificial intelligence, including game AI, can be used for many purposes, some of them nefarious. Perhaps even more importantly, technol ogy can have ethically negative or at least questionable effects even when there is no malicious intent. The ethical effects of using AI with and in games are not always

obvious, and the topic is not receiving the attention it should. This short section looks at some of the ways in which game AI intersects with ethical questions. For general AI research issues, ethics and values we refer the interested reader to the Asilomar AI Principles1 developed in conjunction with the 2017 Asilomar confer ence.

Player modeling is perhaps the part of game AI where the ethical questions are most direct, and perhaps most urgent. There is now a vigorous debate about the mass collection of data about us both by government entities (such as the US Na tional Security Agency or the United Kingdom's GCHQ) and private entities (such as Google, Amazon, Facebook and Microsoft) [64, 502]. With methodological ad vances in data mining, it is becoming possible to learn more and more about in dividual people from their digital traces, including inferring sensitive information and predicting behavior. Given that player modeling involves large-scale data col lection and mining, many of the same ethical challenges exist in player modeling as in the mining of data about humans in general. Mikkelsen et al. present an overview of ethical challenges for player modeling [458]. Below we give some examples of such challenges.

Privacy: It is becoming increasingly possible and even practicable to infer vari ous real-life traits and properties of people from their in-game behavior. This can be done without the consent or even knowledge of the subject, and some of the in formation can be of a private and sensitive nature. For example, Yee and colleagues investigated how player choices in World of Warcraft (Blizzard Entertainment, 2004) correlated with the personalities of players. They used data about players' characters from the Armory database of World of Warcraft (Blizzard Entertainment, 2004) and correlated this information with personality tests administered to players; multiple strong correlations were found [788]. In a similar vein, a study investigated how players' life motives correlated with their Minecraft (Mojang, 2011) log files [101]. That research used the life motivation questionnaires of Steven Reiss, and found that players' self-reported life motives (independence, family, etc.) were expressed in a multitude of ways inside constructed Minecraft (Mojang, 2011) worlds. Using a very different type of game strong correlations have been found between play ing style in the first-person shooter Battlefield 3 (Electronic Arts, 2011) and player characteristics such as personality [687], age [686] and nationality [46]. It is en tirely plausible that similar methods could be used to infer sexual preferences, po litical views, health status and religious beliefs. Such information could be used by advertising networks to serve targeted ads, by criminals looking to blackmail the player, by insurance companies looking to differentiate premiums, or by malevolent political regimes for various forms of suppression. We do not know yet what can be predicted and with what accuracy, but it is imperative that more research be done on this within the publicly available literature; it is clear that this kind of research will also be carried out behind locked doors.

1 https://futureoflife.org/ai-principles/

Ownership of Data: Some player data can be used to recreate aspects of the player's behavior; this is the case for e.g., the Drivatars in Microsoft's Forza Motorsport series, and more generally for agents created according to the procedural persona concept [267]. It is currently not clear who owns this data, and if the game developer/publisher owns the data, what they can do with it. Will the game allow other people to play against a model of you, i.e., how you would have played the game? If so, can it identify you to other players as the origin of this data? Does it have to be faithful to the behavioral model of you, or can it add or distort aspects of your playing behavior?

Adaptation: Much of the research within game AI is concerned with adaptation of games, with the experience-driven PCG framework being the perhaps most complete account on how to combine player modeling with procedural content generation to create personalized game experiences [783]. However, it is not clear that it is always a good thing to adapt games to players. The "filter bubble" is a concept within discussion of social networks which refers to the phenomenon where collaborative filtering ensures that users are only provided with content that is already in line with their political, ethical, or aesthetic preferences, leading to a lack of healthy engagement with other perspectives. Excessive adaptation and personalization might have a similar effect, where players are funneled into a narrow set of game experiences.

Stereotypes: Anytime we train a model using some dataset, we run the risk of reproducing stereotypes within that dataset. For example, it has been shown that word embeddings trained on standard datasets of the English language reproduce gender-based stereotypes [93]. The same effects could be present when modeling player preferences and behavior, and the model might learn to reproduce prejudiced conceptions regarding gender, race, etc. Such problems can be exacerbated or ameliorated by the tools made available to players for expressing themselves in-game. For example, Lim and Harrell have developed quantitative methods for measuring and addressing bias in character creation tools [386].

Censorship: Of course, it is entirely possible, and advisable, to use AI methods to promote ethical behavior and uphold ethical values. Earlier, Section 7.2 discussed the examples of AI for filtering player chats in online multi-player games, and for detecting sexual predators. While such technologies are generally welcome, there are important ethical considerations in how they should be deployed. For example, a model that has been trained to recognize hate speech might also react to normal in-game jargon; setting the right decision threshold might involve a delicate tradeoff between ensuring a welcoming game environment and not restricting communications unduly.

AI Beyond Games: Finally, a somewhat more far-fetched concern, but one we believe still merits discussion is the following. Games are frequently used to train and