1 Introduction

Let us play a little thought game. Get a pen and paper. Choose any game you know, and think about the elements required to make it work. Write down a list of these elements. Be as specific or indiscriminate as you want. Once you have finished, choose another game and think about it. Try to find items in the list of the first game that correspond to the second game and mark them. If there are features in the second game that the first one does not have, add them to the list. Repeat this procedure for two or three more games. Next, take the five most common items in your list and compare them to the following list. For each corresponding item you get one point.

The key elements of a game are:

- players who are willing to participate in the game;
- rules which define the limits of the game;
- goals which the players try to achieve during the game;
- opponents or opposing forces which prevent the player from achieving the goals;
- a representation of the game in the real world.

How many points did you score?

The five components we have listed seem to be present in every game, and the relationships between them form three aspects of a game, which are illustrated in Figure 1.1 (Smed and Hakonen 2003, 2005b):

(i) *Challenge*. Rules define the game and, consequently, the goal of the game. When players decide to participate in the game, they agree to follow the rules. The goal motivates the players and drives the game forward, because achieving a goal in the game gives the players enjoyment.

(ii) *Conflict*. The opponent (which can include unpredictable humans and random processes) obstructs the players from achieving the goal. Because the players do not have a comprehensive knowledge of the opponent, they cannot determine precisely the opponent's effect on the game.

(iii) *Play*. The rules are abstract but they correspond to real-world objects. This representation concretizes the game to the players.

The challenge aspect alone is not enough for a definition of a game, because games are also about conflict. For example, a crossword puzzle may be a challenge in its own right but there is hardly any conflict in solving it – unless someone erases the letters or changes the hints or keeps a record of the time to solve the puzzle. Obviously, the

Algorithms and Networking for Computer Games, Second Edition. Jouni Smed and Harri Hakonen.
© 2017 John Wiley & Sons Ltd. Published 2017 by John Wiley & Sons Ltd.

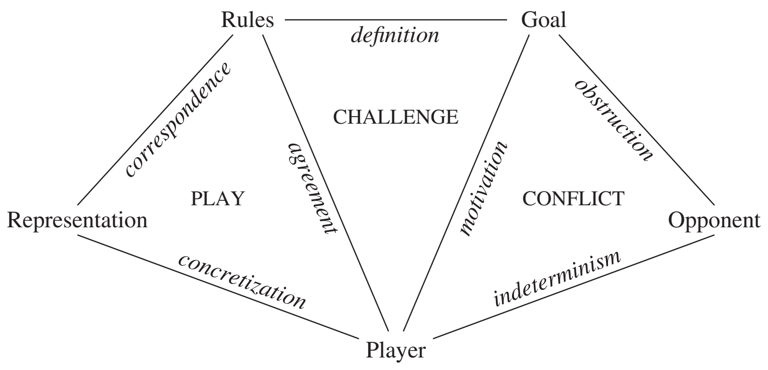

Figure 1.1 Components, relationships, and aspects of a game.

conflict arises from the presence of an opponent, which aims to obstruct the player from achieving the goal. The opponent does not have to be a human but it can be some random process (e.g. throw of dice or shuffling of the deck of cards). The main feature of the opponent is that it is non-deterministic to the player: because the player cannot predict exactly what another human being or a random process will do, outwitting or outguessing the opponent becomes an important part of the game.

Challenge and conflict aspects are enough for defining a game in an abstract sense. However, in order to be played the game needs to be concretized into a representation. This representation can be a board and plastic pieces as well as non-tactile words or three-dimensional graphics rendered on a computer screen. Even the players themselves can be the representation, as in the children's game of tag. Regardless of the representation there must exist a clear correspondence to the rules of the game.

Let us take the game of poker as an example. The players agree to follow the rules, which state (among other things) what cards there are in a deck, how many cards one can change, and how the hands are ranked. The rules also define the goal, having as good a hand as possible when the cards are laid on the table, which is the player's motivation. The other players are opponents, because they try to achieve a better hand to win – or, at least, to give such an impression. Also, the randomness of the deck caused by shuffling opposes the player, who cannot determine what cards will be dealt next. The game takes a concrete form in a deck of plastic-coated cards (or pixels on the screen), which represent the abstractions used in the rules.

One of the earliest written collection of games, *Libro de los juegos* ('Book of games'), commissioned by King Alfonso X of Castile, León and Galicia and completed in Toledo 1283, divides the games into three groups: games of skill (e.g. chess), games of chance (e.g. dice games) and games combining skill and chance (e.g. backgammon). This division reflects the conflict aspect and the type of the opponent.

Huizinga's definition of play from his classical work *Homo Ludens*, the playful human, captures most of the features we listed earlier:

> [Play] is an activity which proceeds within certain limits of time and space, in a visible order, according to rules freely accepted, and outside the sphere of necessity or material utility. The play-mood is one of rapture and enthusiasm, and is sacred or festive in accordance with the occasion. A feeling of exaltation and tension accompanies the action, mirth and relaxation follow. (Huizinga 1955, p. 132)

Moreover, Huizinga's idea of a magic circle tries to capture the complete game (or play) experience, which resides outside ordinary life.

Caillois (2001) builds upon Huizinga's work and divides games further into four forms:

- *agon* (competition) describes games where the aim is to beat the opponent and luck does not play a significant role (e.g. chess);
- *alea* (chance) describes games where luck or chance is the decisive factor on the outcome (e.g. Roulette);
- *mimicry* (role-play) describes games where the players go through an adventure with their characters in a game world (e.g. *Dungeons & Dragons*);
- *ilinx* (vertigo) describes games that affect the player's observations or movements (e.g. *Dance Dance Revolution*).

Games are usually a combination of the aforementioned forms. Moreover, Caillois notes that games form a continuum from structured, rule-governed games (*ludus*) to spontaneous, unstructured play (*paidia*).

Wittgenstein argues that it is impossible to define a game: 'For how is the concept of a game bounded? What still counts as a game and what no longer does? Can you give the boundary? No.' (Wittgenstein 2009, Aphorism 68). Suits responds to Wittgenstein's challenge directly by giving the following definition:

> To play a game is to attempt to achieve a specific state of affairs [prelusory goal], using only means permitted by rules [lusory means], where the rules prohibit use of more efficient in favour of less efficient means [constitutive rules], and where the rules are accepted just because they make possible such activity [lusory attitude]. I also offer the following simple and, so to speak, more portable version of the above: playing a game is the voluntary attempt to overcome unnecessary obstacles. (Suits 2014, p. 43)

Crawford (1984, Chapter 1) defines a game as 'a closed formal system that subjectively represents a subset of reality'. Accordingly, a game is self-sufficient, follows a set of rules, and has a representation in the real world. These observations are echoed by the definitions of Costikyan (2002, p. 24), who sees a game as 'an interactive structure of endogenous meaning that requires players to struggle toward a goal', and by Salen and Zimmerman (2004, p. 80), for whom a game is 'a system in which players engage in an artificial conflict, defined by rules, that results in a quantifiable outcome'. A widely known, practical definition of a game, attributed to the game designer Sid Meier, states that a game is a series of meaningful choices (Rollings and Morris 2000, p. 38). Schell (2015, p. 47) shares this point of view, defining a game as 'a problem-solving activity, approached with a playful attitude'.

Apart from formal features, the gameplay also includes subjective elements such as an immersion in the game world, a sense of purpose, and a sense of achievement from mastering the game. One could argue that the sense of purpose is essential for the immersion. What immerses us in a game (as well as in a book or a film) is the sense that there is a purpose or motive beneath the surface. In a similar fashion, the sense of achievement is essential for the sense of purpose (i.e. the purpose of a game is to achieve goals, points, money, recognition, etc.). From a human point of view, we get satisfaction in the process of nearing a challenging goal and finally achieving it – and then realizing that we can relive that feeling. These aspects, however, are outside the scope of our current discussion, and we turn our focus to a subset of games, namely computer games.

1.1 Anatomy of Computer Games

Computer games are a subset of games. To be more precise, let us define a computer game as a game that is carried out with the help of a computer program. This definition leaves us some leeway, since it does not imply that the whole game takes place in the computer. For example, a game of chess can be played on the screen or on a real-world board, regardless of whether the opponent is a computer program. Also, location-based games (see Chapter 11) further obscure the traditional role of a computer game by incorporating real-world objects into the game world.

In effect, a computer program in a game can act in three roles:

(i) coordinating the game process (e.g. realizing a participant's move in a chess game according to the rules);
(ii) illustrating the situation (e.g. displaying the chessboard and pieces on screen); and
(iii) participating as a fellow-player.

This role division closely resembles the *Model–View–Controller* (MVC) architectural pattern for computer programs. MVC was originally developed within the Smalltalk community (Krasner and Pope 1988) and was later adopted as a basis for object-oriented programming in general (Gamma et al. 1995). The basic idea is that the representation of the underlying application properties (Model) should be separated from the way it is presented to the user (View) and from the way the user interacts with it (Controller). Figure 1.2 illustrates the MVC components and the data flow in a computer game.

The Model part includes software components which are responsible for the coordination role (e.g. evaluating the rules and upholding the game state). The rules and basic entity information (e.g. physical laws) form the core structures. It remains unchanged while the state instance is created and configured for each game process. The core structures need not cover all the rules, because they can be instantiated. For example, the core structures can define the basic mechanism and properties of playing cards (e.g. suits and values) and the instance data can provide the additional structures required for a game of poker (e.g. ranking of the hands, staking, and resolving ties).

The View part handles the illustration role. A proto-view provides an interface into the Model. It is used for creating a synthetic view for a synthetic player or for rendering a view to an output device. The synthetic view can be preprocessed to suit the needs of the synthetic player (e.g. board coordinates rather than an image of the pieces on a board). Although rendering is often identified with visualization, it may also include audification and other forms of sensory feedback. The rendering can have some user-definable options (e.g. graphics resolution or sound quality).

The Controller part includes the components for the participation role. Control logic affects the Model and keeps up the integrity (e.g. by excluding illegal moves suggested by a player). The human player's input is received through an input device filtered by driver software. The configuration component provides instance data, which is used in generating the initial state for the game. The human player participates in the data flow by perceiving information from the output devices and performing actions through

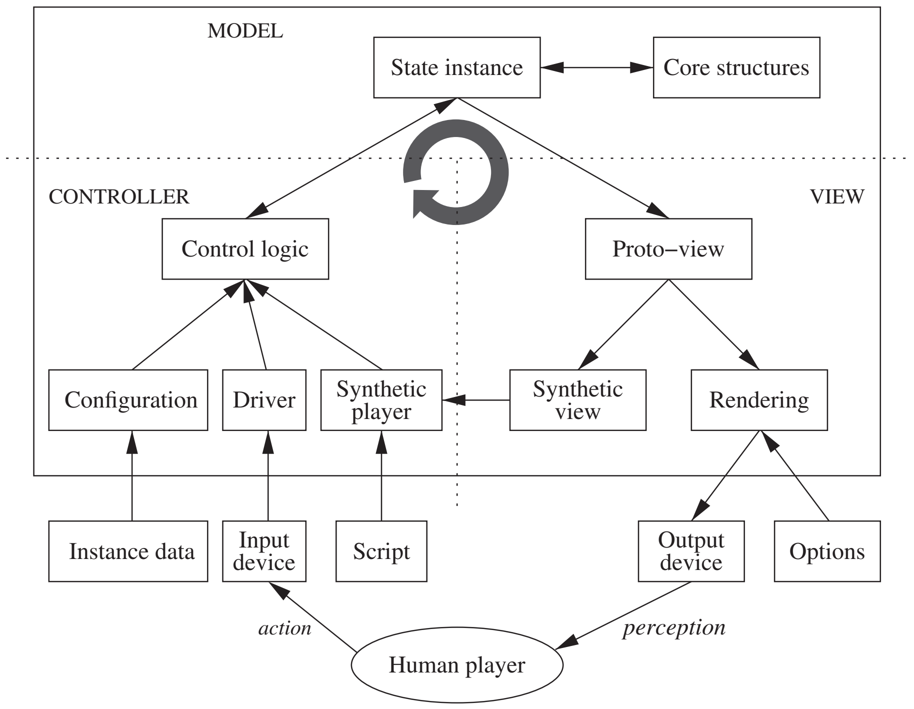

Figure 1.2 Model, View and Controller in a computer game.

the input devices. Although the illustration in Figure 1.2 includes only one player, naturally there can be multiple players participating in the data flow, each with their own output and input devices. Moreover, the computer game can be distributed among several computer nodes rather than residing inside a single node. Conceptually, this is not a problem since the components in the MVC can also be thought to be distributed (i.e. the data flows run through the network rather than inside a single computer). In practice, however, networked computer games provide their own challenges (see Section 1.4).

1.2 Game Development

In the game industry, the production process of games is called game development and the people participating in this process are collectively known as game developers. This group is diverse and houses talents with different skills and backgrounds, but typically the game industry recognizes seven professional disciplines (Novak 2007, pp. 302–321):

- *production* – managing the practical challenges of the game development process;
- *marketing* – raising and maintaining awareness of the game among the (potential) players;
- *testing and quality assurance* – ensuring the stability and playability of the game;
- *design* – handling the mechanics behind the rules and play of the game;
- *art* – creating the visual components of the game;

- *audio* – creating the sounds and aural environments for the game;
- *programming* – implementing the game in a digital form.

During game development, the producer and game designer play the pivotal roles. The game designer is the one with a vision of the game, which is to be carried from inception to conclusion. The producer is the counterpart who has to work with the realities – schedule, budget – to enable the project to materialize. They often work in tandem, the producer being the external link (e.g. to the customer or publisher) and the designer the internal link (i.e. to the rest of the development team). The artists (both visual and audio) design and create the assets that the game uses, and the programmers' task is to implement the game mechanics and the user interfaces. The game testers and quality assurance provide feedback to the development team by taking care that the game is playable, bug-free, enjoyable, and ready to be marketed to the customers.

A large commercial game project can take 2–4 years of work, throughout which the game development involves 50–150 people, possibly in several countries and production sites. For example, the production of *Grand Theft Auto V* took 5 years, involved over 300 people and cost £170 million. Requiring both technical and artistic expertise, even smaller projects require cooperation between several specialized professionals. Nevertheless, the finished game should be a cohesive whole, which delivers the vision of the game designer to the players.

From the game designer's perspective the game can be divided into the basic parts as illustrated in Figure 1.3 (Adams 2014). The three fundamental components are the player who plays the game, the user interface that presents the game to the player, and the core mechanics implementing the rules and the game artificial intelligence (AI). The core mechanics generates challenges that the user interface (through a camera model) converts to output for the player. Conversely, the player's input is conveyed through the user interface (based on the interaction model) and converted to actions for the core mechanics. Gameplay is then the challenges and actions, and together with the user interface they define the gameplay mode, of which a game can have several.

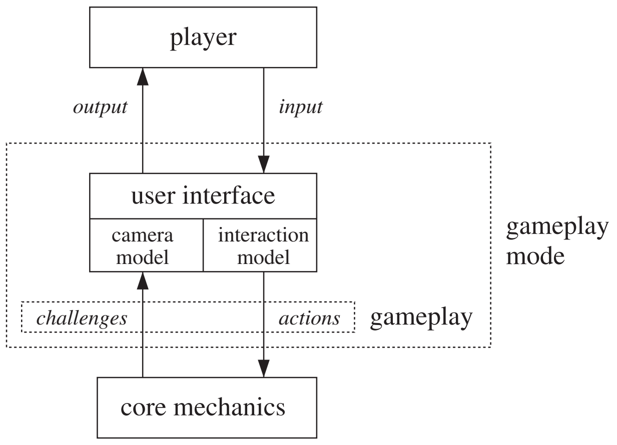

Figure 1.3 Basic design parts of a game.

1.2.1 Phases of development

Figure 1.4 illustrates the phases of a typical game development project (Novak 2007, pp. 334–346). In the concept phase, a game idea is concretized into a concept document, which is used (as a sales pitch) in order to raise funding for the production. If a publisher accepts the concept, the game idea will be refined in the pre-production phase, in which the game designer creates a game design document. This represents a

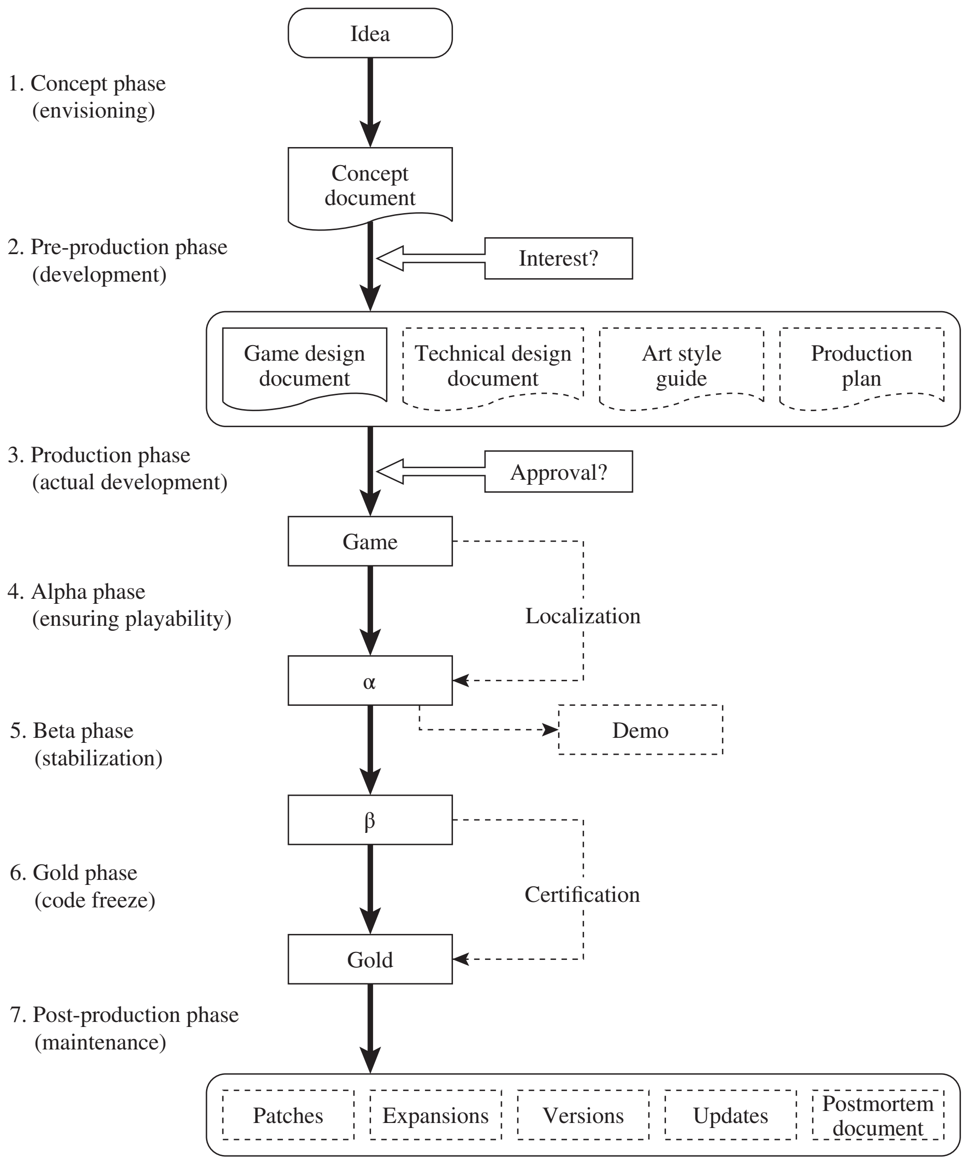
Idea
1. Concept phase (envisioning)
Concept document
2. Pre-production phase (development)
Interest?
Game design document
Technical design document
Art style guide
Production plan
3. Production phase (actual development)
Approval?
Game
4. Alpha phase (ensuring playability)
Localization
α
Demo
5. Beta phase (stabilization)
β
6. Gold phase (code freeze)
Certification
Gold
7. Post-production phase (maintenance)
Patches
Expansions
Versions
Updates
Postmortem document

Figure 1.4 The phases of a game development project.

'blueprint' of the game for the production. Additionally, other documents such as a technical design document, an art style guide or a production plan can be generated during the pre-production phase.

After the pre-production phase, the project must get approval from the publisher before entering the production phase, where the actual game development takes place. Once the production phase is over, the game moves first to the alpha phase, which concentrates on ensuring playability, and then to the beta phase, where the aim is to stabilize the game by eliminating bugs. Finally, in the gold phase the game code and other assets are finalized before publishing. The game might also require certification from the publisher for market acceptance. At the same time, the game can be localized to other languages so that all the versions can be published simultaneously. In order to market the forthcoming game, playable demos can be made public before the final product is ready.

When the game has been published, it enters the post-production phase, where bugs and design flaws can be patched and the game can be updated according to possible new requirements. The game can be ported to other platforms or extended by creating new material for the players. Finally, the game developers can issue a postmortem document where they analyse the project.

Digital distribution and ideas from lean development have changed this model slightly into a more iterative process. Once the game has been published online, the development can revert back to production phase to include new content or even new game mechanisms based on feedback and metric data from players (more on this in Chapter 14). This means that the production process is not as heavy as in the traditional, game-as-a-product distribution model which aims to deliver a finished game. Lean and iterative production, which is common especially in mobile games, changes this into a game-as-a-service distribution model, where the game is never finished but continuously growing and transforming.

1.2.2 Documentation

To maintain the original vision a game development project is built upon game documents, which provide all the departments – from management to engineering and arts – with a single vision (Rouse 2004, pp. 206–319). Therefore, the documentation serves two purposes: it is a record of the design decisions, and it is a means of communication that conveys the game design to all participants in the project. The documentation can have any format suitable for the game production; for example, it can be, apart from text, a collection of thematic images, sounds, video and other items. The purpose is to compile and convey the business idea, product identity and value proposition of the game.

Figure 1.5 summarizes the typical documents created and used during a game development project (Schell 2015, pp. 425–432):

- *game design overview* – a short summary of the game written for the company's management;
- *game design document* – a detailed description of the game mechanics and interfaces;
- *story overview* – a description of the setting, characters, and actions that will take place in the game;
- *technical design document* – a specification of the technology used, for the engineering department;
- *pipeline overview* – instructions on how the art assets will be integrated into the game;

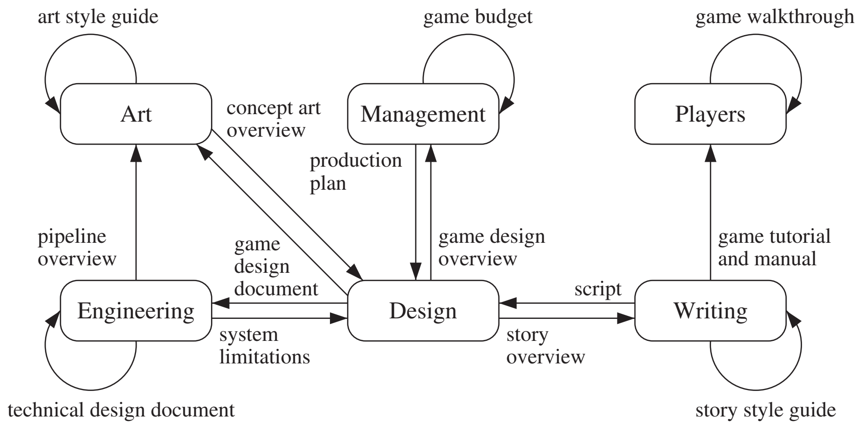

Figure 1.5 The game documents created and used in a game development project.

- *system limitations* – a summary of the technical limitations for the design department;
- *art style guide* – guidelines for the arts department to create a single, consistent look and feel;
- *concept art overview* – a summary of the outlook of the game;
- *game budget* – a spreadsheet for keeping track of costs;
- *production plan* – a schedule for the game project;
- *story style guide* – guidelines of the story-world for the writing department;
- *script* – dialogues for the game characters;
- *game tutorial and manual* – instructions on how to play the game;
- *game walkthrough* – a guide written by players to help other players to play the game.

The game design document (GDD) is the central written resource for the game design. It is typically a 50–200-page reference guide to the whole game development process (Novak 2007, pp. 368–370). It specifies the mechanics of the game, the rules of play, and the theme of the outlook (Chandler 2008, pp. 252–257). A GDD is not static but continually edited and updated by the designers and developers during the production phase. Nevertheless, the GDD should remain up to date even after the production phase because it will still be needed in localizing the game to new languages. If the game is ported to a new platform, the GDD provides a source of information for the (possibly third-party) team carrying out the conversion. Also, a GDD is a valuable asset when designing a follow-up or an extension to the original game.

1.2.3 Other considerations

Although defining what makes a game enjoyable is subjective, we can list some features that alluring computer games seem to have. Of course our list is far from complete and open to debate, but we want to raise certain issues which are interesting in their own right but which – unfortunately – fall outside the scope of this book.

- *Customization*. A good game has an intuitive interface that is easy to learn. Because players have their own preferences, they should be allowed to customize the user interface to their own liking. For example, the interface should adapt dynamically to

the needs of a player so that in critical situations the player has more detailed control. If a player can personalize her avatar (e.g. customize the characteristics to correspond to her real-world persona), it can increase immersion in the game.
- *Tutorial*. The first problem a player faces is learning how the game works, which includes both the user interface and the game world. Tutorials are a convenient method for teaching the game mechanics to the player, where the player can learn the game by playing it in an easier and possibly assisted mode.
- *Profiles*. To keep the game challenging as the player progresses, it should support different difficulty levels which provide new challenges. Typically, this feature is implemented by increasing certain attributes of the enemies: their number, their accuracy, and their speed. The profile can also include the player's preferences of the type of game (e.g. whether it should focus on action or adventure).
- *Modifications*. Games gather communities around them, and members of the community start providing new modifications (or 'mods') and add-ons to the original game. A modification can be just a graphical enhancement (e.g. new textures) or can enlarge the game world (e.g. new levels). Also, the original game developers themselves can provide extension packs, which usually include new levels, playing characters, and objects, and perhaps some improvement of the user interface.
- *Replaying*. Once is not enough. We take pictures and videotape our lives. The same applies also to games. Traditionally, many games provide the option to take screen captures, but replays are also an important feature. Replaying can be extended to cover the whole game, and the recordings allow the players to relive and memorize the highlights of the game, and to share them with friends and the whole game community.

It is important to recognize beforehand what software development mechanisms are published to the players and with what interfaces. The game developers typically implement special software for creating content for the game. These editing tools are a valuable surplus to the final product. If the game community can create new variations of the original game, the longevity of the game increases. Furthermore, the inclusion of the developing tools is an inexpensive way – since they are already implemented – to enrich the contents of the final product.

Let us turn the discussion around and ask what makes a bad computer game. It can be summed up in one word: *limitation*. Of course to some extent limitation is necessary – we are, after all, dealing with limited resources. Moreover, the rules of the game are all about limitation, although their main function is to impose the goals for the game. The art of making games is to balance the means and limitations so that this equilibrium engrosses the human player. How do limitations manifest themselves in the program code? The answer is the lack of parameters: the more things are hard-coded, the lesser the possibilities to add and support new features. Rather than closing down possibilities, a good game – like any good computer program! – should be open and modifiable for both the developer and the player.

1.3 Synthetic Players

A synthetic player is a computer-generated actor in the game. It can be an opponent, a non-player character which participates in some limited way (like a supporting actor), or a *deus ex machina* which can control natural forces or godly powers and thus intervene or generate the game events.

Because everything in a computer game revolves around the human player, the game world is anthropocentric. Regardless of the underlying method for decision-making (see Chapter 9), the synthetic player is bound to show certain behaviour in relation to the human player, which can range from simple reactions to general attitudes and even complex intentions. As we can see in Figure 1.2, the data flow of the human player and the synthetic player resemble each other, which allows us to project human-like features onto the synthetic player.

We can argue that, in a sense, there should be no difference between the players whether they are humans or computer programs; if they are to operate on the same level, both should ideally have the same powers of observation and the same capabilities to cope with uncertainties (see Chapter 10). Ideally, the synthetic players should be in a similar situation to their human counterparts, but of course a computer program is no match for human ingenuity. This is why synthetic players rarely display real autonomy but appear to behave purposefully (e.g. in *Grand Theft Auto III* pedestrians walk around without any real destination).

The more open (i.e. the less restrictive) the game world is, the more complex the synthetic players are. This trade-off between the Model and the Controller software components is obvious: if we remove restricting code from the core structures, we have to reinstate it in the synthetic players. For example, if the players can hurt themselves by walking into fire, the synthetic player must know how to avoid it. Conversely, if we rule out fire as a permitted area, path finding (see Chapter 7) for a synthetic player becomes simpler.

Let us take a look at two external features that a casual player is most likely to notice first in a synthetic player: humanness and stance. These are also relevant to the design of the synthetic player by providing a framework for the game developers and programmers.

1.3.1 Humanness

The success of networked multiplayer games can be, at least in part, explained by the fact that the human players provide something that the synthetic ones still lack. This missing factor is the human traits and characteristics – flaws as much as (or even more than) strengths: fear, rage, compassion, hesitation, and emotions in general. Even minor displays of emotion can make the synthetic player appear more human. For instance, in *Half-Life* and *Halo* the synthetic players who have been taken by surprise do not act with superhuman coolness but show fear and panic appropriate to the situation; actually, the reaction time should be 0.2–0.4 seconds (Rabin 2015). We, as human beings, are quite apt to read humanness into the decisions even when there is nothing but naïve algorithms behind them. Sometimes a game, such as *NetHack*, even gathers around a community that starts to tell stories of the things that synthetic players have done and to interpret them in human terms.

A computer game comprising just synthetic players could be as interesting to watch as a movie or television show (Charles et al. 2002). In other words, if the game world is fascinating enough to observe, it is likely that it is also enjoyable to participate in – which is one of the key factors in games like *The Sims* and *Singles*, where the synthetic players seem to act (more or less) with a purpose and the human player's influence is, at best, only indirect.

There are also computer games that do not have human players at all. Already back in the 1980s *Core War* demonstrated that programming synthetic players to compete with each other can be an interesting game itself (Dewdney 1984). Since then some games have tried to use this approach, but, by and large, AI programming games have been only by-products of 'proper' games. For example, *Age of Empires II* includes the possibility of creating scripts for computer players, which allows games to be organized where programmers compete as to who creates the best AI script. The whole game is then carried out by a computer while the humans remain as observers. Although the programmers cannot affect the outcome during the game, they are more than just enthusiastic watchers: They are the coaches and the parents, and the synthetic players are the protégés and the children.

1.3.2 Stance

The computer-controlled player can have different stances (or attitudes) towards the human player. Traditionally, the synthetic player has been seen only in the role of an enemy. As an enemy the synthetic player must provide challenge and demonstrate intelligent (or at least purposeful) behaviour. Although the enemies may be omniscient or cheat when the human player cannot see them, it is important to keep the illusion that the synthetic player is at the same level as the human player.

When the computer acts as an ally, its behaviour must adjust to the human point of view. For example, a computer-controlled reconnaissance officer should provide intelligence in a visually accessible format rather than overwhelm the player with lists of raw variable values. In addition to accessibility, the human players require consistency, and even incomplete information (as long as it remains consistent) can have some value to them. The help can even be concrete operations as in *Neverwinter Nights* or *Star Wars: Battlefront* where the computer-controlled team-mates respond to the player's commands.

The computer has a neutral stance when it acts as an observer (e.g. camera director or commentator) or a referee (e.g. judging rule violations in a sports game) (Martel 2014; Siira 2004). Here, the behaviour depends on the context and conventions of the role. In a sports game, for example, the camera director program must heed the camera placements and cuts dictated by television programme practice. Refereeing provides another kind of challenge, because some rules can be hard to judge. Finally, synthetic players can be used to carry on the plot, to provide atmosphere, or simply to act as extras (de Sevin et al. 2015). Nevertheless, as we shall see next, they may have an important role in assisting immersion in the game world and directing the gameplay.

1.4 Multiplaying

What keeps us interested is – surprise. Humans are extremely creative at this, whereas a synthetic player can be lacking in humanness. One easy way to limit the resources dedicated to the development of synthetic players is to make the computer game a multiplayer game.

The first real-time multiplayer games usually limited the number of players to two, because the players had to share the same computer by dividing either the screen (e.g. *Pitstop II*) or the playtime among the participating players (e.g. *Formula One Grand Prix*). Also, the first networked real-time games connected two players over a modem (e.g. *Falcon A.T.*). Although text-based networked multiplayer games started out in the early 1980s with multi-user dungeons (Bartle 1990), real-time multiplayer games (e.g. *Quake*) became common in the 1990s as local area networks (LANs) and the Internet became more widespread. These two development lines were connected when online game sites (e.g. *Ultima Online*) started to provide real-time multiplayer games for a large number of players sharing the same game world.

On the technical level, networking in multiplayer computer games depends on achieving a balance between the consistency and responsiveness of a distributed game world (see Chapter 12). The problems are due to the inherent technical limitations (see Chapter 11). As the number of simultaneous players increases, scalability of the chosen network architecture become critical. Although related research work on interactive real-time networking has been done in military simulations and networked virtual environments (Smed et al. 2002, 2003b), the prevention of cheating is a unique problem for computer games due to the conflicting motivations and interests of the participants (see Chapter 13).

Nowadays, commercially published computer games are expected to offer a multiplayer option, and, at the same time, online game sites are expected to support an ever increasing number of users. Similarly, the new game console releases rely heavily on the appeal of online gaming, and a whole new branch of mobile entertainment has emerged with the intention to develop distributed multiplayer games for wireless applications.

The possibility of having multiple players enriches the game experience – and complicates the software design process – because of the interaction between the players, both synthetic and human. Moreover, the players do not have to be opponents but they can cooperate. Although more common in single-player computer games, it is possible to include a story-like plot in a multiplayer game, where the players are cooperatively solving the story (e.g. *No One Lives Forever 2* and *Neverwinter Nights*).

In the design of massively multiplayer online games, the two main game design approaches are called theme-park and playground – or, alternatively, rollercoaster and sandbox. A theme-park (or rollercoaster), such as *World of Warcraft*, provides the players with top-down generated challenges. The set-up and goal of a challenge are preconceived by the game designers, but there is much leeway as to how the players actually reach the goal. In contrast, a playground (or sandbox), such as *Eve Online*, relies on the emergence of player-originated stories and the social media connecting the game community. The game world is like a playground allowing all kinds of plays and events to unfold. The stories are then told by the community (retrospectively) the same way as reporters and historians do in the real world. Let us next look at storytelling from a broader perspective.

1.5 Interactive Storytelling

Storytelling is one of the oldest human activities. We learn from a very young age to use stories and narratives to communicate ideas and to think about possibilities. In the oral

JWST828-c01 JWST828-Smed May 22, 2017 15:11 Printer Name: Trim: 244mm × 170mm

tradition of storytelling, a bard would adapt a story depending on the audience – even the structure of the story could vary within a certain confines. Only with the advent of the written media did storytelling become 'petrified' and come to mean the process of an author crafting a reproducible composition. 'Interactive storytelling' has taken the original meaning emphasizing the reactive and performative aspects of storytelling, where the aim is to generate dramatically compelling stories based on the user's input (Smed 2014).

Research on interactive digital storytelling (IDS) began in the 1980s with the seminal work of Brenda Laurel (1991). She took ideas from the world of theatre and applied them to computer interfaces in general and to IDS in particular. Formally put, an IDS application is 'designed for users (interactors) to take part in a concrete interactive experience, structured as a story represented in a computer' (Peinado and Gervás 2007).

The core question at the heart of interactive storytelling is the *narrative paradox*, in which the 'pre-authored plot structure conflicts with the freedom of action and interaction characteristics of the medium of real-time interactive graphical environment' (Aylett and Louchart 2007), causing a tension between the interactor's freedom and well-formed stories (Adams 2013). Simply put, the more freedom the interactor has, the less control the author has, and vice versa.

1.5.1 Approaches

The research on IDS has revolved around two distinct approaches. The *author-centric* approach likens IDS to theatre, where the author sets up the story-world and a computer-controlled drama manager directs its characters. A drama manager modifies how the computer-controlled characters react and tries to lead the story in a direction that the author has intended. It tries to change the situation so that the user is going in the direction of the intended story. This can be realized, for example, by limiting the stage and possible actions in the story-world such as in *Façade* where story happens in an apartment during a soirée involving an interactor and two characters having domestic problems (Mateas 2002).

The *character-centric* approach to IDS believes in emergence by allowing the characters in the story-world to be autonomous. Therefore, the key question is to model the mental factors that affect on how the characters act. The author's influence is limited in creating and setting up the story-world. After that, the story-world runs without the author's influence, and the story – hopefully – emerges from the interaction between the computer-controlled characters and the human interactor.

To compare the two approaches Riedl (2004, p.12–14) proposes two measures:

- plot coherence – the perception that the main events of a story are causally relevant to the outcome of the story;
- character believability – the perception that the events of a story are reasonably motivated by the beliefs, desires and goals of the characters.

Clearly, the author-centric approach allows us to have strong plot coherence, because of the drama manager's influence. The downside is, however, that character believability is weakened if the actions of the characters seem to be compelled to follow the author's will. The problem is then finding subtlety so that the influence does not feel too forced upon the user. In implementation, the main concern is that an IDS system must observe the reactions of the user as well as the situation in the story-world to recognize what pattern fits the current situation: is the story getting boring and should there be a surprise twist in the plot, or has there been too much action and the user would like to have a moment of peace to rest and regroup? Since we aim to tell a story to the human users, we must ensure that the world around them remains purposeful.

Conversely, the character-centric approach has (and requires) strong character believability. This means that plot coherence is weaker, because the story emerges from the bottom up from the characters' aspirations. Although the idea of emergent narrative of the character-centric approach seems to solve the narrative paradox, it is unlikely that it is enough for implementing a satisfying IDS system. Realistic actions are not necessarily dramatically interesting, if the characters have no dramatic intelligence. Therefore, the argument is that the author's presence is necessary, because without the author's artistic control we would end up having the chaos of everyday life.

Recently, the discussion has evolved to include a hybrid approach, where the characters are autonomous but they can communicate with one another outside of the story-world (Swartjes et al. 2008). These two modes of the character are called in-character (IC) and out-of-character (OOC). They are used, for example, in live action role-playing where the participants can act IC (i.e. within the role they are playing) or drop to OOC when they are being themselves. Also, in improvisational theatre the actors can convey OOC information using indirect communication (e.g. an actor can say 'Hello, son!', cuing the other actor to assume the role of son). For example, Weallans et al. (2012) present a hybrid approach called distributed drama management, where the characters act on an IC level and reflect on their actions on an OOC level. A character proposes a set of possible actions to a drama manager, which selects dramatically the best alternative. Here, the drama manager is no longer pushing the characters to follow its lead but supports their decision-making through OOC communication.

1.5.2 Storytelling in games

The International Game Developers Association (2004) says that '[a]ny game featuring both characters and a story in which one or more narrative aspects changes interactively can be considered an interactive story.' The simplest narrative aspect that can be interactive is the plot, which can vary in response to the player's actions. Another possibility is that the player's actions affect and change the non-player character's attitude and personality (e.g. if the player acts in a friendly manner, the non-player character also becomes more friendly and helpful towards the player). A third possibility is to have a varying theme, where the player's behaviour in the story-world trims the theme, making the story, for example, more romantic, thrilling or violent.

According to Costikyan (2002), a game is not a story: while a story progresses linearly, a game must provide an illusion of free will. Obviously, the player must have a range of actions to choose from at each stage. More formally, let us consider the story in a game as a directed graph where the episodes (or levels) are vertices and the possible transitions edges (Figure 1.6). This means that the greater the fan-out of a vertex is, the more freedom the player has in the story-world. The simplest game stories use a linear structure, where the story unfolds as an episodic sequence. A game may offer only a little room for the story to deviate – as in *Dragon's Lair* where, at each stage, the players can choose from several alternative actions of which all but one lead to certain

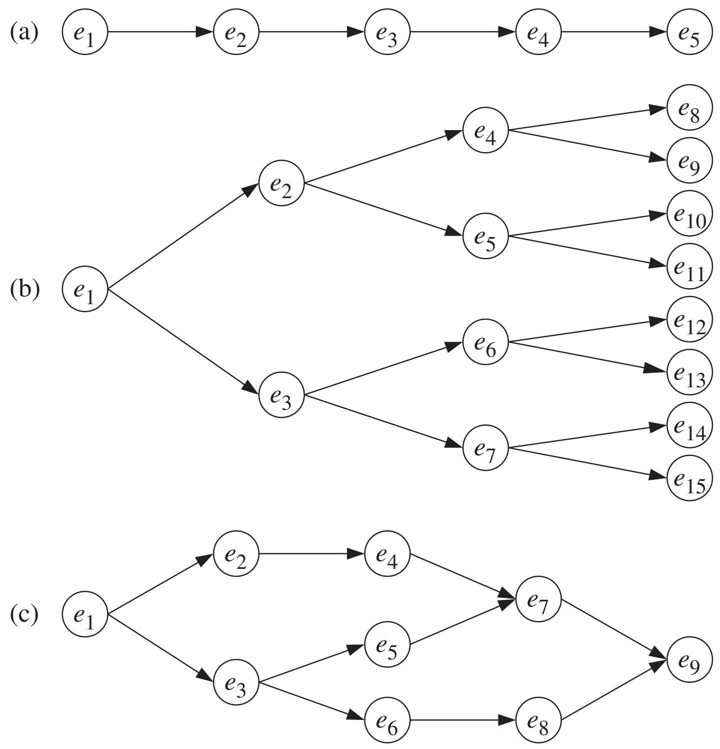

Figure 1.6 The story structure can be illustrated as a graph, where episodes (or levels) are represented as vertices and transitions as directed edges. (a) In a linear structure, the story always unfolds the same way. (b) In a branching structure, each transition leads to a unique episode. (c) In parallel paths, although the episodes can branch, they can also conjoin, limiting the number of episodes.

death. Nevertheless, this is the most commonly used structure to tell a story in computer games: An episode has a fixed starting point and ending point (e.g. a ‘boss monster’ at the end of the level) between which the player can proceed freely. For instance, in *Max Payne* or *Diablo II* the plot lines of the previous chapter are concluded at the transition point, and new alternatives are introduced for the next one. The episodes follow one another linearly in a pre-authored order, and they are linked, for example, by cut-scenes. Linear story structure is the cheapest to produce, which is why it is popular even today. The obvious drawback is that the player has no influence on how the story unfolds and the story can feel like it has been pasted over the game. In the worst case, the elaborate cut-scene videos and complex plot twists only bore the player who skips them in order to get right into the action.

In theory, branching structure would be optimal for IDS, because each decision has a unique outcome. However, combinatorial explosion prevents us from using it in practice. The only way to make it feasible is to have conjoining edges that bring two or more vertices (i.e. episodes) together, leading to a structure called parallel paths. The story is again presented in an episodic manner, but at the transition point, where the story of the previous episode is concluded, the player gets to choose from alternative paths for the next episode. Although the paths can take players to different routes, eventually they conjoin in a major story point. For example, given a task, a player can choose to fight to achieve his aims or to take a diplomatic path avoiding violence altogether. An early example of this is *Indiana Jones and the Fate of Atlantis*, where in mid-game the player has to choose one of three possible paths – team, wits or fists – which converge before the end. Although the paths could lead to the same outcome, the episodes leading there can be totally different. This of course means that the game will have content that a player cannot see in one play. For a game publisher this means that it seemingly gets less value for its investment, which is why there is a pressure for the game designers to limit the amount of parallel paths in a game. However, there are commercially successful games such as *Heavy Rain* and *The Walking Dead* as well as critically acclaimed games such as *80 Days* and *Her Story* using this approach.

1.6 Outline of the Book

The intention of our book is to look at the algorithmic and networking problems present in commercial computer games from the perspective of a computer scientist. As the title implies, this book divides into two parts: algorithms and networking. This emphasis in topic selection leaves out components of Figure 1.2 which are connected to the human-in-the-loop. Most noticeably we omit topics concerning graphics and human interaction – which is not to say that they are in any way less important or less interesting than the current selection of topics. Also, game design as well as ludological aspects of computer games fall outside the scope of this book.

The topics of the book are based on the usual problems that we have seen game developers encountering in game programming. We review the theoretical background of each problem and review the existing methods for solving them. The given algorithmic solutions are not provided in any specific programming language but in pseudocode format, which can be easily rewritten in any programming language and – more importantly – which emphasizes the algorithmic idea behind the solution. The algorithmic notation used is described in detail in Appendix A. We also present a practical approach to vectors and matrices in Appendix B.

We have also included examples from real-world computer games to clarify different uses for the theoretical methods. In addition, each chapter is followed by a set of exercises which go over the main points of the chapter and extend the topics by introducing new perspectives.

1.6.1 Algorithms

Part I of this book concentrates on typical algorithmic problems in computer games and presents solution methods. The chapters address the following questions:

- Chapter 2, 'Random Numbers': How can we achieve the indeterminism required by games using deterministic algorithms?
- Chapter 3, 'Noise': How can we make computation based on mathematics look more life-like?
- Chapter 4, 'Procedural Generation': How can we create game content using algorithms?

- Chapter 5, 'Tournaments': How we can form a tournament to decide a ranking for a set of contestants?
- Chapter 6, 'Game Trees': How can we build a synthetic player for perfect information games?
- Chapter 7, 'Path Finding': How can we find a route in a (possibly continuous) game world?
- Chapter 8, 'Group Movement': How can we steer a group of entities through the game world?
- Chapter 9, 'Decision-Making': How can we make a synthetic player act intelligently in the game world?
- Chapter 10, 'Modelling Uncertainty: How can we model the uncertainties present in decision-making?

1.6.2 Networking

Part II turns our attention to networking. Our aim is to describe the ideas behind different approaches rather than get too entangled in the technical details. The chapters address the following questions:

- Chapter 11, 'Communication Layers': What are the technical limitations behind networking?
- Chapter 12, 'Compensating Resource Limitations': How can we cope with the inherent communication delays and divide the network resources among multiple players?
- Chapter 13, 'Cheating Prevention': Can we guarantee a fair playing field for all players?
- Chapter 14, 'Online Metrics': What can we measure from the online player's behaviour?

1.7 Summary

All games have a common basic structure comprising players, rules, goals, opponents and representation. They form the challenge, play and conflict aspects of a game, which are reflected, for instance, in the Model–View–Controller software architecture pattern. The computer can participate in the game as a synthetic player, which can act in the role of an opponent or a team-mate or have a neutral stance. For example, the synthetic player must take the role of a story-teller, if we want to incorporate story-like features into the game. Multiplaying allows other human players to participate in the same game using networked computers.

Game development has matured from its humble beginnings and now resembles any other industrialized software project. Widely accepted software construction practices have been adopted in game development, and, at the same time, off-the-shelf components (e.g. 3D engines and animation tools) have removed the burden to develop all software components in-house. Moreover, modern game development tools such as CryEngine, Unity and Unreal Engine have democratized the development process and made it possible for people who are not so competent in programming to make games. This maturity, however, does not mean that there is no room for artistic creativity and technical innovations. There must be channels for bringing out novel and possibly radically different games, and, as in music and film industry, independent game publishing can act as a counterbalance to the mainstream. One could even argue that this liberation of the game industry has brought about a fresh evolution pool of ways to make business, already affecting the entertainment industry in the large.

Nevertheless, behind computer games are computer programs propelled by algorithms and networking. Let us see what they have in store for us.

Exercises

1-1 Take any simple computer game (e.g. *Pac-Man*) and discern what forms its challenge aspect (i.e. player, rules and goal), conflict aspect and play aspect.

1-2 A crossword puzzle is not a game (or is it?). What can you do to make it more game-like?

1-3 Why do we need a proto-view component in the MVC decomposition?

1-4 What kind of special skills and knowledge should game programmers have when they are programming
(a) the Model part of the software components,
(b) the View part of the software components, or
(c) the Controller part of the software components?

1-5 Let us look at a first-person shooter game (e.g. *Doom* or *Quake*). Discern the required software components by using the MVC. What kind of modelling does it require? What kind of View-specific considerations should be observed? How about the Controller part?

1-6 *Deus ex machina* (from Latin 'god from the machine') derives from ancient theatre, where the effect of the god's appearing in the sky, to solve a crisis by divine intervention, was achieved by means of a crane. If a synthetic player participates the game as a *deus ex machina*, what kind of role will it have?

1-7 What does 'anthropocentrism' mean? Are there non-anthropocentric games?

1-8 *The Sims* includes an option of free will. By turning it off, the synthetic players do nothing unless the player explicitly issues a command. Otherwise, they show their own initiative and follow, for example, their urges and needs. How much free will should a synthetic player have? Where it would serve best (e.g. in choosing a path or choosing an action)?

1-9 In the movie *Stranger Than Fiction* (2006), the protagonist realizes that his life is happening in a fictional novel, and when he refuses to obey the voiceover, the world tries to force him to follow the intended story. Does this represent an author-centric or character-centric approach to interactive storytelling?

1-10 Take your favourite game and decompose its storytelling. Does it always tell the same story or does it vary from one play instance to another? If the story does not respond to the player's actions, what could be done to make it more interactive?

1-11 Game development includes people with different talents (e.g. artists, programmers, designers and marketing people). What kind of communication problems might arise when they work together on the same game project? What is the role of a game programmer in a game project?

1-12 Consider the differences and similarities of Figures 1.4 and 1.5 for a triple-A game developed by hundreds of persons and an indie game developed by one person.

1-13 Because game documents are living documents that change on almost a daily basis, the biggest debate within the game industry is about the effectiveness of creating extensive game documentation during the pre-production phase. Usually many elements of the game change drastically during the production phase, and the documents cannot keep up with the pace of change. The problems encountered in game documentation can be classified into five categories: (Rouse 2004, pp. 374–379):

- Lack of content: The document does not provide enough reference material for the production phase.
- Misplaced focus: The document provides data (e.g. backstory) that is irrelevant to the production phase.
- Overspecification: The document goes too deeply into details, which will become clear only in the production phase.
- Infeasible content: The document contains design decisions that are impossible to realize in the game.
- Fossilization: The document is out of date and, subsequently, abandoned during production.

Game documentation often does not support but hinders the work, because there are no computer-aided tools for maintaining it. Instead, it comprises a bundle of text documents without a clear maintenance scheme. To complicate matters further, game documents – unlike, for example, film scripts – have no pre-defined formats (Rouse 2004, pp. 355–359). Although there are document templates, many game designers state that documents are different for every game and for every team. For example, Schell (2015, p. 426) says outright that a 'magic template [for game documents] does not exist'.

How could this problem be solved? Think about game documentation in terms of maintainability, accessibility and communicativeness. Also, take the concept of 'document medium' as sufficiently wide to cover, for example, a whiteboard or even an oral discussion.

1-14 Many games are variations of the same structure. Consider *Pac-Man* and Snake. Discern their common features and design a generic game which can be parameterized to be both games.

1-15 Judging rules can be difficult, even for an objective computer program. In football (or soccer as some people call it) the official rules say that the referee can allow play to continue if the team against which an offence has been committed has a chance of an immediate, promising attack (i.e. advantage), and penalize the original offence if the anticipated advantage does not ensue at that time (Fédération Internationale de Football Association 2016). How would you implement this rule? What difficulties are involved in it?

1-16 The progression in the lattice of mission groups in *Wing Commander* resembles the story structure shown in Figure 1.6(c). The player's performance in the missions branches in the story, and piling failures drive the player further away from the hope of a victory. However, with later successes it is still possible to get back to the path of victory.

With such second chances in mind, analyse the aspects of failure and failing in general in computer games. For example, in games like *Super Meat Boy* and *Dwarf Fortress* losing is an integral part of the game experience but, on the other hand, in *NetHack* death is permanent and the game feels intentionally brutally lost. How does the presence of failing define and complement the other features of a game?