PREVIOUS MENU

Game Design Patterns

Jussi Holopainen[1] & Staffan Björk[2]

[1] Nokia Research Center
Visiokatu 1
33721 Tampere
Finland
jussi.holopainen@nokia.com

[2] PLAY, Interactive Institute
Hugo Grauers gata 3
411 33 Göteborg
Sweden
staffan.bjork@tii.se

Table of Contents

Introduction

This text is one half of the notes for the “Game Design Patterns” lecture at GDC 2003. The lecture is the result of the game design community and the research community identifying that the development of semi-formal and formal game design methods would advance the way game design is discussed. We believe that Game Design Patterns offers a viable method to achieve this that can be used both for commercial and academic use and can bridge the difference between the two communities, allowing them to collaborate and share knowledge to a greater extent than before.

As stated in the lecture description, we believe patterns are only useful as long as one can learn and apply them with reasonable effort and that they can be tailored to any given project. In other words, patterns need to support practical work such as developing the initial game design or problem-solving particular interaction elements in a game. Thus, the lecture focuses on practical use, and includes workshop-like elements. These notes go beyond the practical use of patterns and discuss more general aspects of design patterns, e.g. the choice of pattern definition and the advantages and disadvantages of pattern use.

The notes make no attempt at defining a "canonical pattern language". For a language to prove its merit, it has to be used and expanded by speakers. Therefore, we do not wish to present a complete and finalized collection, or even definition, of patterns but instead show our work-in-process so that designers can influence the technique to better suit their needs. One structure for patterns is described in this text while the other text for the lecture focuses on how to apply patterns as a technique, creating one’s own set of patterns. These two approaches have been developed from different perspectives and offer two possibilities for pattern use within game design. Further material can be found in the companion material of the GDC 2002 roundtable (Kreimeier 2002a) about design patterns and in Gamasutra article The Case for Game Design Patterns (Kreimeier 2002b). This part of the notes is a continuation of work presented at Computer Game Design Patterns workshop (Björk&Holopainen 2002).

If patterns are widely adopted by practitioners and researchers we expect several different approaches and pattern collections to appear. Regardless of the specific formalism that becomes the norm this can be seen as a success as the game designers and researchers have gained one language to express the knowledge and experience contained in the community.

Why Design Patterns?

Before presenting the details of our model for using game design patterns, motivations for using design patterns at all are in place. We believe that the following points are valid reasons for using patterns, independently of if one is an experienced project leader leading a large group, a new designer daunted by the task of completing one's first game concept, or an interested hobbyist.

- Problem-Solving for Game Interaction Design

  Patterns contain descriptions of identifiable elements of interaction within a game together with approaches to ensure the presence of those elements. If one as a designer has a problem with a specific part of the interaction within a game concept, the approaches describe in patterns containing the wanted interaction offer several potential solutions.

- Inspiration

  Having a collection of patterns is in essence having a listing of concepts that other game designers have found useful for designing games. Having these concepts at one's fingertips provides a game designer with a knowledge base that can be used to find the core of a new game design or tweaks that make a game different.

- Creative Design Tool

  The original use of design patterns as described by Alexander et al (Alexander 1977) was that of a tool that helped designers (architects in this case) to work through all levels of a design in a structured fashion. A collection of game design patterns can be used in the same way. One begins by selecting a few patterns based on the core game concept (and external requirements). These patterns are analyzed in the specific context for the designers and potential subpatterns are identified. The subpatterns are analyzed and chosen based on their feasibility and their subpatterns are identified. This activity goes on recursively until an initial design has been completed.

- Communicating with peers

  Patterns offer definitions of elements found in many games. Describing one's design in terms of patterns offers one possibility to describe the design in a standardized format to make the understanding and comparison of different designs easier. Further, if readers are unfamiliar with the game concept described by a pattern, the definition of the pattern is available in the same sense definitions of words are available in a dictionary.

- Communicating with other professions

  The method of using design patterns have been introduced in a number of disciplines, including architecture (Alexander 1977), software engineering (Gamma 1994), Human-Computer Interaction (Borchers), and Interaction Design (Erickson). Although not all of these areas are usually part of the competences of developing a game, software engineering and HCI usually are. Using patterns for the design of the actual game play can make the transferal between different parts of a project group easier as the general working technique is familiar and gaining a basic understanding of patterns from another discipline is usually easy.

Template for Game Design Patterns

Gamma (1994) quotes Alexander: "Each pattern describes a problem which occurs over and over again in our environment, and then describes the core of a solution to that problem, in such a way that you can use this solution a million times over, without ever doing it the same way twice" (Alexander 1977). Thus, a pattern can be described as an identifiable characteristic that can found in otherwise totally different designs. A typical template of a pattern consists of its *name*, a description of a *problem* and its *solution* together with the *consequences* of having or applying the pattern. For the design of games, we are currently working with the following template:

- *Name*: Giving short and expressive names to patterns make them possible to use as self-sufficient concepts. "Naming a pattern immediately increases our design vocabulary. It lets us design at a higher level of abstraction" (Gamma 1994). However, naming can be difficult for patterns are connotations can create problems, e.g. the pattern **Paper-Rock-Scissors** describes the strategy of avoiding dominant strategies but could easily be thought to describe the act of secretly choosing a strategy and simultaneously revealing the choices.

- *Description*: Unlike most design patterns we have chosen not to define patterns as a pure problem-solution pairs. This is due to two observations. First, defining patterns from problems creates a risk of viewing patterns as a method for only removing unwanted effects of a design. In other words, using patterns as a tool for problem-solving only and not as a tool to support creative design work. Second, many of the patterns we have identified described a characteristic that more or less automatically guaranteed other characteristics in a game, i.e. the problem described in a pattern might easily be solved by applying a more specific subpattern.

- *Consequences*: Each solution has its own trade-offs and consequences. Solutions can, in turn, cause or amplify other problems. To take a design decision for or against a given solution, its costs and benefits have to be understood and compared against those of alternatives. This section describes the likely or possible consequences of applying the solution suggested by the pattern.

- *Using the Pattern*: As patterns are general solutions the application of a pattern to any given situation requires a number of design choices specific for the current context. However, the high-level choices can often be divided into categories. This section is used to mention the common choices a designer is faced with when applying a pattern, often exemplified by specific game elements from published games.

- *Relations*: Here the relations between different game design patterns are stated. These are basically three forms of relationship: patterns that are superior in the sense that they describe more abstract characteristics (often mentioned in the consequences section) and can be implemented by applying the given pattern, subpatterns that can be used to implement the given pattern (often mentioned in the using the pattern section), and conflicting patterns that are difficult to implement with the given pattern.

Criticism of Patterns

The use of patterns in other fields have received criticism for several different reasons (being a fad, being too formal, being not formal enough, being too arbitrary) and we direct interested readers to the companion material of the GDC 2002 roundtable about design patterns for that discussion (Kreimeier 2002a). However, the use of patterns for game design can be criticized for similar reasons as well as reasons specific for games. Below we address some of the possible objections to the use of pattern in game design.

- Patterns a fad

  Question: Patterns have become popularized in other fields, is the use of patterns in game design simply a way of taking advantage of that popularity?

  Answer: We believe the strength of design patterns is that they represent a general method of helping design processes, in essence helping to define a language for specific design fields. Patterns have become popular within other fields only to be replaced or challenged by other methods. Whatever the cause for this, e.g. trends or the introduction of methods developed specifically for the field, the game community currently lacks a common language to discuss design. We believe patterns may be a solution to this but other solutions may be equally suited, the important point is that a solution is found.

- Pattern methods too formal

  Question: Doesn't the formalizing of design processes risks inhibiting the creative processes of designers and artists?

  Answer: Design Patterns should be seen as tools; one should only use them as long as they are appropriate. By their definition they are open to interpretation, leaving considerable room for creative freedom. Further, they focus on the game interaction so that e.g. the actual content creating characterization and narrative structure is left untouched.

- Patterns too abstract

  Question: It doesn't seem that game design patterns can provide any help with actually writing code for a game. Aren't the patterns too abstract?

  Answer: Game Design Patterns deal with the design of games from an interaction perspective rather than one of expressing games in programming code. Thus, they are by definition more abstract than methods used to formalize game design into code and should not be used for that activity.

- Patterns not formal enough

  Question: The game design patterns do not seem formal enough to be translated e.g. into design patterns for software engineering. Assuming that one has design a game and described it through game patterns, what practical use does one have of the effort?

  Answer: We do not believe that there is a simple one-to-one mapping from game design patterns to design patterns for object-oriented programming. However, having a game description using game patterns gives programmers a blueprint expressed in a form using the same basic structuring model that they use for programming. We believe this translation is easier than one from e.g. scenarios or story-boards.

- Patterns too arbitrary

  Question: How are patterns identified? Is there a formalized selection method?

  Answer: Creating a collection of patterns can be compared to creating a dictionary. One has to study how people use words (and decide if they are talking the language one is interested in or not). Identified words need to be described and related to other words. If the dictionary is to have any influence, it has to be used by the practitioners.

  We are currently collecting patterns by studying the use of concepts within the gaming industry. Looking both at computer games and traditional games, we have identified several patterns (primarily from game genres and game mechanics) by study-

ing written material about games, interviewing game designers and, of course, playing the games. However, even more patterns have been identified that do not currently have names; the evidence for these patterns being present in the final game designs or the activities of designers. All these patterns are currently in the process of being described and analyzed. These will then be presented to the game design community as a selection method. Only the patterns that make sense for designers and are actually used should be kept.

Using Design Patterns

For those that are interested in using game design patterns we can see three important questions that may be asked. In this section we try to briefly answer them; other questions no doubt will occur for interested parties and we invite you to contact us with them.

- Do Patterns work for game design?

  It is still too early to answer this question in a meaningful way. That one can use game patterns in the design process is apparent but do the benefits of using it surpass the disadvantages? We have tried to identify the possible benefits of design patterns as well as locate possible disadvantages and methods for mitigating them but until design patterns have been put into wider practical use these are just speculations.

  However, we do not believe that the success of game patterns can be measured by game sales alone. Instead, we would consider game design patterns as successful if individual designers perceive the patterns as helping them in their design process as well as in communicating their designs to all the other people involved in the process of creating a game.

- Do Design Patterns work for all types of games?

  Our collection of patterns has been constructed by looking at different genres within computer games, board games and card games. Thus, we believe that patterns have a potential for being useful in these areas. Further, one of the research aims of our work is to provide a theoretical framework that will help in the design of new forms of games, including mobile computer games based on position or context, games using new input and output systems, and traditional games augmented with embedded computers.

- How can I use Design Patterns?

  Contact the presenters by email. We are collecting, defining and analyzing patterns in an ongoing process to build a comprehensive collection of interrelated patterns for use by practitioners. The patterns have to be validated by actual use to provide their worthiness and the only people who can do that are game designers.

  The second alternative is to start from scratch and build your own collection of patterns. This requires more work but allows different starting points; our collections start from either user experience or possible interactions but other are possible, e.g. manipulation of the game state or components of game engines.

What Design Patterns exist?

The simple answer to this question is: any that has been identified. However, what useful design patterns exist is a question which is more difficult to answer. We are in the process of identifying patterns which we believe are useful either as problem-solving tools, descriptions of game elements that are useful when analyzing game

designs, or patterns that we believe support design processes. To do this we have studied and played many games (computer games, board games, card games, role-playing games, etc.) but also interviewed professional game designers. Many patterns have been found or distilled when trying to describe other patterns or to relate patterns to each other.

In this section we present a number of candidates for game design pattern that we have identified. Besides describing a number of recurring patterns we have found in games they exemplify our approach to using game patterns. The pattern **Analysis Paralysis** described below is included as an example of a category of patterns that usually is avoided by game designers (another examples include **Invisible Wall** and **King Maker**). However, the patterns can easily occur in a game design by mistake or by the combined effects of other patterns and thus deserves mention. The use of the patterns in a way that makes a game playable may be seen as a design challenge.

Note that in examples below the related patterns are in **bold** and most of the patterns are not described in this paper.

- Perceived Chance to Succeed

Description
Games usually have the possibility to reach certain goals as one of the primary reason for playing them. A game where the goals are perceived as impossible to reach by players often unplayable and the same applies to the game where success is always certain. This is especially true for multiplayer games; if after some period of time one player is perceived as the only possible winner the game usually its appeal to the other players. Therefore many games try to let players have a perceived chance of winning as long as possible, which includes have a chance to succeed with the subgoals necessary to win.
Consequence
If a player perceives that he or she always has a chance to reach a goal, although it may be minute, it will increase the probability that he or she will struggle against the system or the other players. This in turn will increase the tension of the struggle and will make succeeding more rewarding.
Using the Pattern
When using this pattern one has to take precautions that how players play does not become irrelevant until the very end of the game. Thus, players should feel motivated to always do there best as this will improve their chances of winning but this chance should not become a certainty until as late as possible in the game. Monopoly (Parker Brothers) is an example of a game breaking this pattern as in most of the games the winner is clear many turns before the actual end condition. Games that use **Last Man Standing** pattern in victory condition, for example Magic: The Gathering (Wizards of the Coast), violate this pattern in such way that players are removed from the game one by one until only the winner is left. Slowly increasing the difficulty, i.e. providing a **Smooth Learning Curve**, lets player feel that the current subgoal is possible to complete while at the same time allowing the player to learn all skills necessary to complete the whole game. For examples, a **Level-based Game** let players feel that they are making progress by completing levels where the first levels are usually very easy and the last levels impossible for a novice to complete. An interesting subgroup is games where the player will always lose the game in the end (Space Invaders, Asteroids, Tetris) but

Description
which still give the player a feeling of having a chance to succeed. This leads to a strong 'just one more go' experience. See (Loftus&Loftus 1983) for a discussion of a psychological basis of this effect (**Partial Reinforcement**). **Balancing Effect** can be incorporated into game system to try and compensate for unequal player positions. Some games use **Dynamical Difficulty Adjustment** to, for example, making monsters easier to defeat or increasing the number of available power-ups if the player fails constantly. Another example is found in many racing games where the leading players or computer controlled vehicles are automatically slowed down to make it easier for other players to catch up. This makes the game more compelling even to the leading player as it heightens the perceived **Struggle**.
Related patterns
Superior patterns: - Subpatterns: **Balancing Effect, Partial Reinforcement, Smooth Learning Curve** Conflicting patterns: **Early Elimination, King Maker, Last Man Standing**

- Analysis Paralysis

Description
Analysis Paralysis occurs when a player is confronted with so many possibilities that gaining an overview of what the different consequences will be becomes overwhelming and game play is affected negatively. That many possibilities exist is not sufficient for the pattern to occur; the player has to have a sense that analyzing the situation is possible and will give the player an advantage over other players.
Consequence
Analysis Paralysis forces players to spend time on deciding what to do instead of interacting with the game system. This may led to experiences that the game does not **Allow Game Mastery** and the **Perceived Chance to Win** becomes one of pure luck.For other players Analysis Paralysis mean that the game does not have **Reasonable Waiting Times** but may decrease **Tension** as these players do not have a pressure upon them to do something in the game.
Using the Pattern
Limiting players' possibilities to analyze a game state can remove the consequences of Analysis Paralysis. Having a **Time Limit** sets a fixed limit to the amount of time that can be spend analyzing a situation while **Constant Player Activity** and **Constant Movement** forces a player to continuously weigh the benefits of continuing the analysis with reacting to events in the game. All these approaches do not actually remove the problem of analysis for players but simply enforces that action is taken or the players is negatively affected. Gentler approaches, which do not threaten with punishments, are possible by enforcing a **Limited Foresight** so that the consequences of actions are more difficult to calculate. Analysis Paralysis may be mitigated for other players by giving them purposeful things to do while a player is thinking which does not actually affect the game state. Different forms of **Stimulated Social Interaction** are likely candidates: al-

Description
lowing communication for negotiation of tactics or alliances or out-of-game conversations. Other examples are support for studying the game rules or training various aspects of a game while waiting.
Relationships
Superior patterns: -
Subpatterns: -
Conflicting patterns: **Reasonable Waiting Times**, **Allow Game Mastery**, **Reasonable Waiting Times**

- Mutual Goal

Description
The players, or some of players, try to reach the same goal or subgoal within the game. This pattern occurs whenever more than one player has exactly the same goal, e.g. "we both want the red car to come first" and not "we both want our respective cars to come first" (which are **Symmetrical Goals**).
Consequence
Defining a victory condition for several players by giving them a mutual goal creates **Team Play**. Mutual goals that are subgoals in a game can either be used to strengthen the advantages of cooperating within a team or be used to create a **Temporary Alliance** between players. In all cases, the mutual goals promote **Stimulated Social Interaction.**
Using the Pattern
Just as any kind of goal, mutual goals can either be Predefined Goals that are either known or unknown (Unknown Goals) before game play begins or goals that are Player-Constructed Closures with rewards defined by the players. An example of the latter is Diplomacy (Avalon Hill) where players can negotiate and define own Mutual Goals ranging from simple 'support this unit' to far ranging 'defend Italy'. Mutual goals that are unknown may either be unknown to the other players or the players having the goals. In the first case this leads to **Secret Collaboration** which the other players try to disclose while in the second case the players have **Unknown Allies** and one part of the game is to determine who is on your side without revealing your intentions to the other players. One example of this is Royal Turf (Reiner Knizia, Alea 2001) where money is won by betting on the outcome of a horse race. Once a secret betting is done, players take turns moving the horses around the track. Only moving the horse one has bet on gives away ones strategy but at the same time one would like to identify other players that have bet on the same horse in order to cooperate. The division of rewards (if any) from a mutual goals is vital for the level of cooperation between players. **Individual Rewards** can create **Tension** as players compete to be the first to achieve a given game state while **Shared Rewards** promote cooperation depending on the exact sharing procedure. One example of the latter which promotes cooperation is automated distribution of experience points or gold in role-playing games (although the distribution of magical items typically is up to the players). In the case where the reward is vaguely defined, e.g. the mu-

Description

tual goals in Diplomacy, the mutual goal can lead to **Tension** between the players as the value of the goal for the other players, and thereby the effort they will put into fulfilling the goal, is difficult to judge.

If the players can negotiate the other players participating in the mutual goal can lead to a **Balancing Effect** since players lagging behind are more likely to form temporary alliances against the current leaders. This, however, is heavily influenced by the distribution function of **Shared Rewards**.

Relationships

Superior patterns: **Team Play**, **Temporary Alliance**, **Stimulated Social Interaction**, **Narrative Structure**, **Punctuated Equilibrium**

Subpatterns: **Predefined Goals, Unknown Goals, Player-Constructed Closures** , **Secret Collaboration**, **Unknown Allies**, **Individual Reward, Shared Reward, Balancing Effect**

Conflicting patterns: **Asymmetrical Goals**, **Symmetrical Goals**

- Shared Reward

Description

The players which have been involved in some way in reaching a closure in the game usually share some form of reward. It is not necessary for all the players sharing the rewards to actually cooperate as in some cases other players can, for example, bet on the result of an adventuring party on a quest.

Consequence

Shared rewards opens up a number of possibilities for player interaction. In the case of the shared reward being the achieved by **Collaboration**, the shared reward gives players a **Mutual Goal** and promotes **Stimulated Social Interaction**. In the case where the reward has to be shared between players the pattern encourages **Competition** between the players. Further, rewards that can be shared between competitors have a certain **Balancing Effect** as the reward does not benefit only one player. This is especially true in situations where players have to choose whom to share the reward with; one usually chooses the player that is furthest from winning.

Using the Pattern

The use of shared rewards greatly depends on whether the size and distribution of the reward is set or it is depending on the game state. If a player gets the same quantitative reward regardless of the number of other players sharing the reward the player does not lose anything by cooperating. This however requires an **Unlimited Resource Pool**, at least during the actual reward distribution and can lead to **Inflation**.

If the size of the reward is fixed (**Limited Resource Pool**), players have to compete against each other to receive as large part of it as possible. How the distribution function is defined greatly influences game play for these kinds of rewards: a first-come-first-served function creates a **Race**, distribution of rewards based on chance promotes **Investment**, predetermined relationships between shares and reward encourages struggles with a set **Time Limit** when the reward is distributed. In the last case **Geometrical Reward for Investments** creates more competi-

Description

tion than **Arithmetical Reward for Investments**. Note that the number of shares does not need to be fixed even though the size of the reward is fixed.

How players receive shares of the reward can either be by merit or by concession. For players to receive shares by merit simply means that they have to fulfill certain criteria to be entitled to a share, e.g. having stock in a company. Shared by concession means that players have to give away parts of a reward to other players in order to get their part. For example, the board game *Kohle*, *Kies & Knete* (Sid Sackson, Schmidt Spiele 1994) is a game of making deals where each deal gives a certain amount of money. However, deals can rarely be closed by only one player and the lead player of a deal will have to seek help from the other players in exchange for a piece of the payoff.

The actual content of the shared reward can be the results of the process of negation between players. An example of this is **Trading**, which can be seen as a shared reward since the players that trade get a benefit over those players who weren't a part of that trade. In trading rules that allow for **Bluffing**, the actual content of the shared reward is uncertain creating **Uncertain Reward**.

Relationships

Superior patterns: **Stimulated Social Interaction, Collaboration, Mutual Goal**, **Competition**, **Balancing Effect**

Subpatterns: **Trading, Geometrical Reward for Investments, Arithmetical Reward for Investments**

Conflicting patterns: **Individual Reward**

Creative tool for experimental game design

As has been mentioned earlier in the text, we believe that patterns can be used not only to analyze existing games but also to assist in the design process itself. One of the benefits of using patterns is that the designer is able to formulate the design problem in a clear and consistent way thus giving the opportunity to consciously make different kinds of design decisions than in the previous games.

This is especially important in what can be called experimental game design (see http://www.experimental-gameplay.com) where the goal is to create games that break the traditional genres, themes, or game styles of existing games. It is easier to innovate and experiment when the basic elements and building blocks are known and formalized to some extent, even if the innovation lies in *not* using a well-known game element but instead trying out the opposite. One example of how this can be done is given below, describing how one of the authors designed a prototype of a small multiplayer game for mobile phones. Design patterns are indicated with bold type.

The main design requirement was that the game should have **Stimulated Social Interaction** between players. This pattern has **Trading** as a subpattern, given that if players have to trade with each other they have to communicate which easily can turn from purely formal communication to social interaction. In order to have something to trade the **Source/Sink** pattern was used to define the basic game mechanics. As the game was required to be simple only four different resource types were defined. The pattern version of **Asymmetric Distribution** (Falstein 2002) was used in resource generation, the **Source** part of **Source/Sink** pattern, in such way that for each of the players it was randomly biased to one of the resource types. This, combined with the design decision that the players needed all resource types roughly equally to succeed in game, the **Sink** part of the **Source/Sink** pattern, lead to the situation where trading and haggling were necessary parts of game play.

Trading, however, was considered not to be enough so **Mutual Goals** and **Shared Rewards** patterns, also subpatterns of **Stimulated Social Interaction**, were used to create a system where players have the same or supporting subgoals and that players shared the rewards of reaching the goals. In this case the reward was simple victory points.

The **Bluffing** pattern was used to create more **Tension** between the players. The players now secretly selected their own goals from a set that ensured that the there was sufficient overlap. The selection was also connected to the resource generation in a way that made it impossible for a single player to reach any of the goals single handedly. Thus, the game design requires players to cooperate but also makes it possible to trick other players to use their resources for your own good.

Several other patterns for multiplayer games, such as **Spectators**, **Voting** and **Personalization**, were used in the creation of the final design. At least in this experiment, even the preliminary patterns were very helpful in the design process. More information about the final design is available from Jussi Holopainen, see above for contact information.

Bibliography

Alexander, C. et al. (1977): A Pattern Language: Towns, Buildings, Construction. Oxford University Press.

Björk, Staffan & Holopainen, Jussi: Computer Game Design Patterns. One-day workshop Computer Games & Digital Textualities 2002 conference, Tampere, Finland.

Borchers, Jan: The HCI Patterns Pages.
http://www.stanford.edu/~borchers/hcipatterns/

Erickson, Tom: The Interaction Design Patterns Page:
http://www.pliant.org/personal/Tom_Erickson/InteractionPatterns.html

Falstein, Noah (2002): Better by Design: Asymmetrical Distribution in Game Developer Magazine, August 2002.

Gamma, E. et al. (1994): Design Patterns: Elements of Reusable Object-Oriented Software. Addison Wesley Longman

Kreimeier, B. (2002a): Content Patterns in Game Design. GDC 2002 Roundtable.
http://www.onearrow.org/game/pattern/

Kreimeier, B. (2002b): The Case for Game Design Patterns.
http://www.gamasutra.com/features/20020313/kreimeier_01.htm

Loftus, Geoffrey R. & Elizabeth F Loftus (1983): Mind at Play: The Psychology of Video Games. New York: Basic Books

Pattern Practice

A Supplement to the "Game Design Patterns"

GDC 2003 combined lectures by Bernd Kreimeier, Jussi Holopainen, Staffan Björk
Supplement by Bernd Kreimeier

Extended Abstract

There is a wide consensus in the game design community that semi-formal or even formal game design methods must be developed (see e.g. the GDC 2002 roundtable discussion on "Game Design Patterns" [5] as well as the "400 Project" started at GDC 2001 [9], as well as Doug Church's "Formal Abstract Design Tools" [10]). We must advance the way we discuss game design details. However, at this time it is not clear which, if any, of the approaches proposed to date are best suited for practical use in the game development process.

Some game industry professionals already use derivatives of Alexandrian "Design Patterns", usually in the context of software engineering. Christopher Alexander's original pattern language has also been referred to in discussing level design and level architecture (see [4] for details and references). Patterns are also used by parts of the research community concerned with Human-Computer Interaction (HCI), of which games (or at least the design and analysis of game User Interfaces) can be considered a specialization. (For a HCI perspective on patterns see e.g. [7] by Sally Fincher.) The work by Microsoft Game Studios' User Testing group (see [2] for an introduction) is also oriented within an HCI context [1]. The connection between HCI problems and game design issues is not a recent insight: witness the popularity of Brenda Laurel's "Computers as Theatre" [3] within parts of the game designer community.

The authors propose the use of "Game Design Patterns" [4,6] as a tool to analyze, document, and discuss game design techniques and recurring game elements. The game design pattern approach is loosely based on Alexander's work, but deviates from it in several ways essential ways. While pattern-based methods can be used to express imperatives (see e.g. the normative rules of the "400 Project"), one important advantage is their adaptability to the specific needs of a given team or project. The lecture focuses on patterns as a method. Individual patterns, and small pattern collections, will be discussed as examples, but no attempt at defining a "canonical pattern language" is made. There is a variety of game genres, and within each genre, a variety of game design philosophies that make it difficult, possibly counterproductive, to attempt to define a pattern collection that could serve so manifold objectives without becoming unwieldy.

Pattern Practice

Patterns, like any semi-formal method, are only useful as long as reasonable efforts to memorize and apply them suffice. Furthermore, in an artistic context, patterns can be very specific to the project or the individuals involved. Consequently, the lecture focuses

on methods to recognize and harvest patterns, on how to define and refine them for documentation, how to document them, and on their application within small groups. The resulting pattern languages should assist the designers in their day-to-day design work, first as tools to document and evaluate designs, and last, but certainly not least, to enable designers to consciously expand the design spaces of computer games.

Pattern-based methods (in game design) are in their infancy. However, even experienced game designers might find the different perspective offered by pattern-based approaches helpful for their own attempts to develop and improve other formal methods (like FADTs or rules). Ultimately, any practical application of patterns requires the individual designer to define and use design patterns in their day-to-day work, as a semi-formal, flexible method to express, evaluate and reflect upon design choices both within the team, and with others outside the development group. Fincher [7] points out that Alexander specifically intended patterns to permit communication between designers (in his domain, architects) and users. For practical purposes of game development, communication with the programming and engineering members of the team, as well as communication with formal User Testing [2] (or more informal user feedback/playtest) groups, as well as communication with production (see also [8]) and management, could potentially benefit from pattern-based documentation and methodology.

Aspects of practical pattern use to be covered in the lecture include the issue of harvesting/mining for patterns ("in every cliché hides a pattern waiting to be understood"), and issues of documenting and discussing patterns:

- Pattern writing process [13,17]
- Pattern writing styles [12,14]
- Pattern writing templates [4,6]
- Minimal mnemonic representation
  - vs. Use-centered reference
  - vs. Full pattern documentation
- Pattern misconceptions [15]

Specific to the efforts aiming at game design patterns are the topics of:

- Anecdoctical representation of knowledge
- Empiric data and patterns: observation vs. "best practice"

- Pattern names and forms: object-centric vs. problem-centric
- Purpose vs. effects
- Pattern categories: spatial vs. behavioral

- Levels of abstraction: building blocks or guiding principles
- Hierarchy, alternative solutions, aggregation, dependencies
- Composite patterns

- Patterns as narrative [11]

- Pattern extraction from walkthroughs
- Pattern extraction from project post-mortems [4]
- Pattern mining by introspection [16]
- Pattern review (e.g. [17])

Of particular interest are the differences between the rule based approach of the "400" project [9] and a pattern approach:

- Pattern made from rules
- Patterns formulated as rules
- Patters vs. rules: trade-offs vs. trumping hierarchy

Comparisons between FADTs and patterns can also serve to better understand either approach. Other possible topics include support for structured editing by off-the-shelf (e.g. XML) tools, pattern review by inversion, use of patterns as creative (i.e. exploratory) tools, as well as patterns as a tool for introspection.

Outlook

The combined lectures on "Game Design Patterns" and the IGDA roundtable on "Game Design Methods" at GDC 2003 complement each other in an attempt to advance the discussion of how to develop formal, abstract tools for game design, a discourse if not begun, then made explicit by Doug Church in 1999 (see [10]). The authors believe that among the approaches proposed to date, patterns offer the most flexibility and versatility to accommodate the wide spectrum defined by different genres, different design objectives and philosophies, and the different levels of abstraction to match the varying needs of pre-production and core production of an increasingly complex (and increasingly expensive) development process.

Note: an updated version of this document, as well as other documents related to the GDC 2002 and 2003 roundtables, can be found at: http://www.oneArrow.org/pattern/game/, as well as on the official GDC website at http://www.gdconf.com/.

References

[1] Randy J. Pagulayan, K Keeker, D. Wixon, R.L. Romero and T. Fuller, "User-Centered Design in Games", In: J.A. Jacko and A. Sears (eds.), "The Human-Computer Interaction handbook: Fundamentals, Evolving Technologies and Emerging Applications". Lawrence Erlbaum Assoc., 2003. ISBN 0-805-83838-4

[2] Bill Fulton, "Beyond Psychological Theory: Getting Data that Improves Games". Game Developer's Conference 2002 Proceedings, San Jose CA, March 2002.
See http://www.gamasutra.com/gdc2002/features/fulton/fulton_01.htm
and http://www.gamasutra.com/gdce/2002/bill_fulton.zip

[3] Brenda Laurel, "Computers as Theatre", Addison Wesley Longman Inc. 1991, 1993 ISBN 0-201-55060-1.

[4] Bernd Kreimeier, "The Case for Game Design Patterns".
See http://www.gamasutra.com/features/20020313/kreimeier_pfv.htm

[5] Bernd Kreimeier, "Game Pesign Patterns", supplement to GDC 2002 roundtable, GDC 2002 proceedings CDROM, see also http://www.oneArrow.org/game/pattern/

[6] Jussi Holopainen and Staffan Bj"ork", "Game Design Patterns", GDC 2003 lecture, Proceedings CD ROM.

[7] Sally Fincher, "Patterns for HCI and Cognitive Dimensions: two halves of the same story?"
Submitted to 14th annual PPIG workshop, 2003.
See http://www.pliant.org/personal/Tom_Erickson/FincherOnPatterns.pdf

[8] Mark Cerny, "Method". GDCE 2002 Web Lecture
See http://www.gamasutra.com/features/slides/cerny/index.htm
also http://www.gamasutra.com/gdce/2002/mark_cerny.zip

[9] Noah Falstein. "Better By Design: The 400 Project". (Game Developer magazine, Vol. 9, Issue 3, March 2002, p. 26.)

[10] Doug Church. "Formal Abstract Design Tools." (Gamasutra, 1999. Originally Game Developer magazine, Vol 3, Issue 28, July 1999.)
See http://www.gamasutra.com/features/19990716/design_tools_01.htm

[11] Andrew Rollings and Dave Morris. Game Architecture and Design. (The Coriolis Group, 2000.) ISBN 1-57610-425-7

[12] Aamod Sane "The Elements of Pattern Style." December 1995.
See http://choices.cs.uiuc.edu/sane/elem.pdf

[13] John Vlissides. "Pattern Hatching - Seven Habits of Successful Pattern Writers." C++ Report. Nov/Dec 1996, and Pattern Hatching: Design Patterns Applied (Addison Wesley, 1998).

[14] Gerard Meszaros and Jim Doble "A Pattern Language for Pattern Writing"
See http://hillside.net/patterns/writing/patternwritingpaper.htm

[15] John Vlissides. "Patterns: The Top Ten Misconceptions". Object Magazine, March 1997. See http://www.research.ibm.com/designpatterns/pubs/top10misc.html

[16] NN. "TIES Patterns- Pattern Mining". From Jim Coplien's pattern writer's workshop. See http://hillside.net/patterns/patternsmining.htm

[17] Doug Lea, "Pattern Checklist" See http://hillside.net/patterns/writing/checklist.htm
See http://hillside.net/patterns/writing/checklist.htm

Speaker Info

Staffan Björk
Staffan Björk is a Ph.D. in informatics with a background in computing science. He was one of the initial members of the PLAY group of the Interactive Institute in Sweden and has been director for the studio since the autumn of 2001. His research interests include design patterns in games, ubiquitous computing, information visualization, and the use of emergent narratives in computer entertainment. His work has been published in SIGGRAPH, ACM CHI, ACM UIST, IEEE Information Visualization, IEEE ISWC, HUC (now UbiComp), AVI and Interact. He is currently co-chair for the Short Talks & Interactive Posters at SIGCHI 2003, Community of Practive Liaison for the Games Community for the CHI 2003 conference, one of the guest editors for a special issue on Ubiquitous Games in the journal Personal and Ubiquitous Computing, and a member of the executive board of Digital Games Research Association (see http://www.digra.org).

Jussi Holopainen
Jussi Holopainen is a Research Scientist at the Visual Communications Laboratory of Nokia Research Center in Finland. His research interests include entertainment applications for wearable and mobile devices, as well as the aesthetical foundations of game design process. His work has been presented at HUC (Handheld and Ubiquitous Computing), ISWC (International Symposium on Wearable Computing), E-Culture Fair (in conjunction with Doors of Perception), Consciousness Reframed, SIGGRAPH and UbiComp. He is also one of the organizers of Jyväskylä Arts Festival, a member of the programme committee for Computer Games & Digital Cultures conference, one of the guest editors for a special issue on Ubiquitous Games in the journal Personal and Ubiquitous Computing, and a member of the executive board of Digital Games Research Association (see http://www.digra.org).

Bernd Kreimeier
Bernd Kreimeier is a writer, physicist, and coder, currently employed as a Senior Programmer for a game development studio in California. He has pursued the topic of game design patterns since 1997, and is currently working on a related book to be

published by Wordware in 2003. His credits include Linux ports of Heretic2 and Quake3; Team Arena; as well as an Xbox launch title. He has lectured at GDC in 2001 and moderated a GDC Roundtable on Game Design Patterns in 2002, and has been published in Game Developer Magazine and on Gamasutra. He has the equivalent of a masters degree in experimental physics from the University of Dortmund in Germany, and served as research and teaching staff member at the Universities of Duesseldorf and Bonn. He has published several novels and short stories in the german language.

The Case For Game Design Patterns

By Bernd Kreimeier
Gamasutra
March 13, 2002

URL: http://www.gamasutra.com/features/20020313/kreimeier_01.htm

Game design, like any other profession, requires a formal means to document, discuss, and plan. Over the past decades, the designer community could refer to a steadily growing body of past computer games for ideas and inspiration. Knowledge was also extracted from the analysis of board games and other classical games, and from the rigorous formal analysis found in mathematical game theory.

However, while knowledge about computer games has grown rapidly, little progress has made to document our individual experiences and knowledge - documentation that is mandatory if the game design profession is to advance. Game design needs a shared vocabulary to name the objects and structures we are creating and shaping, and a set of rules to express how these building blocks fit together.

This article proposes to adopt a pattern formalism for game design, based on the work of Christopher Alexander. Alexandrian patterns are simple collections of reusable solutions to solve recurring problems. Doug Church's "Formal Abstract Design Tools" [11] or Hal Barwood's "400 Design Rules" [6,7,18] have the same objective: to establish a formal means of describing, sharing and expanding knowledge about game design.

From the very first interactive computer games, game designers have worked around this deficiency by relying on techniques and tools borrowed from other, older media -- predominantly tools for describing narrative media like cinematography, scriptwriting and storytelling. Computer games are a visual medium, and game designers are increasingly relying on design techniques developed for movies - to the detriment of efforts to identify methods genuinely suited for game design. The discussion of narrative techniques has come to dominate the discourse on game design, almost extinguishing alternatives.

If the techniques borrowed from other media were sufficient to express and address game design issues, there would be no need to search for alternatives. However, the metaphors and devices borrowed from narrative media are usually insufficient (or even inadequate) to capture the essence of the interactive game medium. The community is aware of this [12], although the response is often an attempt to reconcile the contradiction by redefining the term "narrative" [24]. Taken to the extreme of an artificial playwright [23], this constitutes proposing a technological solution to a conceptual problem. The real issue is not the shortcomings of narrative techniques with respect to their utility in game design, but the lack of techniques genuinely suited for interactive media.

The game design pattern method proposed here is concerned with content patterns, as opposed to software engineering patterns [19], specializations of which that have been proposed for game

programming [33,20]. Similarly, process patterns to organize and manage game development projects (such patterns could be extracted from [17,28]) are beyond the scope of this article.

What Are Patterns?

In a nutshell, patterns are simply conventions for describing and documenting recurring design decisions within a given context, be it game design or software engineering. Specific patterns are the result of applying this method consistently, leading to collections of design patterns which have been assigned a name and are documented by an anecdotal or abstract description. Pattern methods provide semi-formal tools for problem domains in which rigorously formal methods cannot easily be applied, or are simply not available or even conceivable.

Patterns are traditionally expressions of problem-oriented thinking. The seminal book by Gamma et al., Design Patterns: Elements of Reusable Object-Oriented Software, quotes Alexander [3]: "Each pattern describes a problem which occurs over and over again in our environment, and then describes the core of a solution to that problem, in such a way that can you can use this solution a million times over, without ever doing it the same way twice." There can be several alternative solutions to a given problem, each one defining its own pattern, but the combination of problem statement and solution proposal is the essence of any Alexandrian pattern.

A game design pattern collection would provide a shared design vocabulary that allows experienced designers to:

- communicate efficiently with each other, with less experienced designers, and with members of other professions (like software engineers and game coders)
- document their insights, organizing individual experience as written knowledge
- analyze their own design as well as the designs of others, e.g. for purposes of comparative criticism, re-engineering, or maintenance

It is important to distinguish between pattern-based methods, which are very generic and general, and specific pattern collections created for a given purpose. The decision to use patterns merely determines form, not content. The conventions of any pattern template do not guarantee (or prohibit) that useful patterns will be found and documented. Pattern methods are simply a successful way to express existing knowledge.

A Pattern Template

Pattern templates typically contain these four essential elements:

1. **Name.** "Naming a pattern immediately increases our design vocabulary. It lets us design at a higher level of abstraction". Names have to be mnemonic and evocative, but the connotations also pose problems. "Also Known As", frequently part of pattern templates, is actually an indication of a naming problem: "Finding good names has been one of the hardest parts of developing our catalog" (again, Erich Gamma et al.).
2. **Problem.** This describes the problem, including its inherent trade-offs and the context in which the problem occurs. The description of the problem implies a goal that we want to accomplish, and the obstacles we encounter when we attempt to do so.
3. **Solution.** A description of a general arrangement of entities and mechanisms that can be used to solve the problem. This is not a particular design or concrete implementation, but an abstract structure that describes an entire family of solutions that are essentially the same.
4. **Consequences.** Each solution has its own trade offs and consequences. Solutions can, in turn, cause or amplify other problems. The costs and benefits of a solution should be understood and compared against those of alternatives before making a design decision.

Around this essential core, pattern templates often add other elements, or subdivide a core element.

Gamma et.al. uses Name - Intent - Aliases - Motivation - Applicability - Structure - Participants - Collaborations - Consequences - Implementation - Sample - Known Uses - Related Patterns . Meszaros and Doble describe pattern writing itself in patterns organized as Title - Problem - Context - Forces - Solution - Indications - Resulting Context - Related Patterns - Examples - Samples - Rationale - Aliases - Acknowledgements [25]. Alexander et. al. use Name - Example as Picture - Context within larger patterns - Problem - Solution - Solution as Diagram - Relation to smaller patterns [3]. Ultimately, the details of the format chosen do not matter as much as the fact that one format has been elected and is used consistently (for more details on how to write patterns, see [31,34,5]).

Pattern Examples

To show the method in action, let's look at some pattern candidates. The following examples are admittedly very simple, and weren't selected necessarily for their particular relevance to game design.

PROXY

Problem: It might not be possible or desirable to require that a game action is applied to the target object directly. In other words, we have to avoid proximity to or collision with the ultimate target of a given action.

Solution: Introduce another entity (object) that is used as a proxy, standing in for the target object. The proxy can be placed and moved independently of the target object or objects. The restrictions that apply to the target(s) do not by default apply to the proxy. Multiple proxies can be used for the same target or targets.

Consequence: The link between the proxy and the actual target has to be communicated to the player, if the player is supposed to exploit this connection and indirection. Restrictions on placement of proxy and target with respect to player field of view, and restrictions on player view control might apply, to ensure that the player perceives the effect on the target.

Examples: *Munch's Oddyssee* has several proxies for the player characters: the possession orb which enables Abe to take over other game characters (thus turning them into proxies), and the Snoozer robot and H-Crane, which served as proxies for Munch. One benefit of player character proxies is that of a safe death: loosing a proxy does not end the game. Buttons/Switches/Levers that open doors or start/stop machinery are often placed in different locations, instead of being part of the target object. In these cases, the designer wants to separate trigger from target to permit more elaborate puzzles. Half-Life's final boss had a vulnerable location inside its head. That vulnerable spot could be considered a proxy for the entire (invulnerable) boss. As a second order vulnerability, the boss was dependent on three

crystals that supplied energy for healing and attack. Attacking the crystals destroyed them, thus permitting effective attacks on the boss monster itself.

PROXY was chosen to illustrate potential problems in using the same name for different patterns existing in different contexts. There is a software engineering pattern with the same name, introduced by Gamma et al. However, there is no meaningful connection between the two concepts. On the other hand, in some cases a given pattern language might translate into a sensible OOP implementation hierarchy, if there is a correspondence between the purpose of a given game design entity within the design context, and its implementation in the software that constitutes the game engine.

PREDICTABLE CONSEQUENCE

Problem: The player has to perceive failure as a consequence of her mistakes, not as a random or predestined event unrelated to, and unavoidable by, any reasonable choice of actions on her part.

Solution: The player cannot take a meaningful decision to act (or not to act) if the result of a possible action can not be anticipated. A meaningful player decision is an informed decision: she has to be able to guess the result of her action before she ever takes it. Choose game mechanisms that communicate predictable behavior by visual appearance relying on knowledge the player already has (either from real world experience, or from experience with other games, or from experience with preceding parts of your game). Do not break consistency of your visual language, i.e. make sure that objects that behave identical look identical.

Consequence: It is no longer possible to implement surprises that are impossible to anticipate. The player might rely on the out-of-context assurance to anticipate the designer's ambush, and either outwit the designer, or find herself locked into performing an action against her better judgment as the designer has prevented alternative courses of action. Ultimately, the designer might arrange for actions that the informed player will never perform, as they are predictably to her disadvantage. The designer might have to go to considerable length to communicate the rules that govern the outcome of actions, to the player. The design might be biased towards using mechanisms resembling real world objects, to minimize the need for explanations. A lack of uncertainty on the player's side makes the game more transparent, potentially removing the need for exploration and experimentation. Ultimately, the need for Predictable Consequence is determined by the need for backtracking (severity of failure) and costs of backtracking (latency between choice and consequence, see also behavioral science on timing issues). Another possible consequence is that the design might have to restrict itself to a smaller set of player actions, in order to be able to enable them consistently throughout the

game world. Actions only available once or in a very few locations do not amortize the costs of communication and teaching/learning on the designer's/player's part. The design might also have to exaggerate the differences between distinct objects to avoid ambiguities. Any kind of variation or range has to follow rules that can easily be observed and complied with.

Examples: Doug Church's article relied on *Mario 64* as the textbook example [11] of predictable consequence. This pattern could be applied to any game with a consistent physics approximation, such those exhibited by bouncing grenades in FPS games. Other examples include crates and barrels that are marked as explosive actually explode, fireballs that hurt a player when they hit, burning fire, water that drowns if you do not swim, and lava which kills. Counter-examples are plentiful: switches and buttons that say "Push Me" to unleash a necessary ambush (an ambush that also opens the only way to move on) anticipated only as such. Openings in which to jump without knowing what is on the other side such as those found in *Opposing Force* and *Ico*. This is often a necessity imposed by the level partition/transition mechanism the engine uses.

References: Doug Church introduced the concept of perceived/perceivable consequence [11], which he uses also do describe predictable outcomes. There is a difference between the requirement for immediate feedback that the player can connect to her actions, and the requirement that the player has to have a chance to anticipate the likely outcome of her action. This pattern was derived from the latter, the former would require another pattern.

PREDICTABLE CONSEQUENCE is my rewrite of FADT "Perceivable Consequence" introduced by Doug Church (see [11] for the original description). Aside from changes to the description itself to refine its meaning (as I perceived it), this pattern candidate illustrates that FADTs can expressed as patterns.

PAPER-ROCK-SCISSORS

Problem: Avoid a dominant strategy that makes player decisions a trivial choice.

Solution: Introduce nontransitive relationships within a set of alternatives, as in the game of paper-rock-scissors.

Consequence: The player is no longer able to find a single strategy that will be optimal in all situations and under all circumstances. She has to revisit her decisions, and, depending on the constraints imposed by the game, adjust to changing situations, or suffer the consequences of an earlier decision.

Examples: The example given by Andrew Rollings is the set of warrior-barbarian-archer from the Dave and Barry Murray game *The Ancient Art of War* (Broderbund 1984). He also describes *Quake*'s weapon/monster relations in similar terms: Nailgun beats shambler, shambler beats rocket launcher, rocket launcher beats zombie, zombie beats nailgun [28].

References: Chris Crawford (see "Triangularity" in [15]) provided the first explicit description of the use of nontransitive relationships. Andrew Rollings' discussion of examples uses game theory including detailed payoff, as well as informal fictional designer dialogs.

PAPER-ROCK-SCISSORS is a recurring game design device that may have been the very first to be semi-formally described. It can also be presented as a pattern. There is no new information in this rewrite; the existing knowledge has merely been reorganized to match the other examples. The pattern could be extended to include a description of payoff matrices and examples for sets larger than three (see [28] for such a discussion), but even the minimal description captures its essence.

PRIVILEGED MOVE

Problem: Sometimes a given game entity cannot be permitted to interfere with others, or other entities cannot be permitted to interfere with it.

Solution: Introducing exclusive moves and locations separates and protects game entities. These exclusive moves and locations cannot be entered or left without being able to perform the respective privileged move. By introducing different, separate spaces or media, the game environment is broken down into distinct areas, which can serve e.g. as safe zones or safe passages.

Consequence: Depending on the implementation, this can be a very heavy-handed way to ensure protection. Unless protection is implemented in a way that is at the player's disposal (a door that only she can open and close), the player might perceive the constraint as annoyance and disappointment. If the restriction is meant to be temporary, the mechanism to allow it to be lifted has to be communicated to the player, and initiated by her.

Examples: In *DOOM*, monsters could traverse acid pools, but the player could only at the expense of damage that quickly turned to be lethal. In *Half-Life* and *Halo*, enemy soldiers can be dropped from flying vehicles, a transportation mechanism not available to the player, and impossible to interfere with. *Half-Life* also had a spider-like boss that was capable of breaking through its own nets, which blocked the player's way until she forced the monster to retreat. In *Thief*, the player traverses rooftops in relative safety, permitting preview and outlook on the level

bottom. In *Ico*, the ghostly opponents traverse the world in a different dimension, and emerge from portals without apparent restriction. In *Munch's Oddyssee*, water was inaccessible to enemies, providing one player character a privileged swim move and safe passage (in the absence of snipers and mines). In many games, water enemies are in turn restricted to that medium (e.g. *Quake*, *Half-Life*). In many games flying opponents (e.g. the harpies in *Heretic 2*) can perform moves the player cannot, and thus enjoy some amount of safety.

PRIVILEGED MOVE is a pattern proposal for a mechanism rarely discussed, but frequently used for a multitude of purposes. A single solution can solve or address multiple problems; a situation that is not clearly addressed in Alexandrian pattern languages that emphasize the problem-oriented aspect of pattern use. PRIVILEGED MOVE is also a very close relative of FILTER. This example set is by no means comprehensive. As Gamma et al. wistfully remark: "Finding patterns is much easier than describing them."

FILTER

Problem: If the player is given means to create new game objects, and/or new combination of existing objects, and/or move game objects arbitrarily, the resulting complexity might become overwhelming for the designer.

Solution: Limit local complexity by filtering, thus creating smaller, clearly defined subsets that can be dealt with.

Consequence: Player freedom is restricted by implicit or explicit means. Accidental filtering can lead to game situations that the player cannot solve. Ideally, the filtering is obvious to the player, and inherent part of the game world. FILTER can be an intended consequence or unintended side effect of PRIVILEGED MOVE.

Examples: FILTER as a means of dealing with combinatorial complexity of possible game states is extracted from Jon Blossom's postmortem of *DroidWorks* ([9]). Quote: "Our level designers [..] could never be entirely sure what kind of droid would be walking into their rooms [...] In many cases, they created ingenious physical filtering mechanisms that would guarantee only certain types of droids went beyond certain points in the level: A steep hill would weed out biped droids in favor of droids with tractor treads, a chasm would weed out wheeled droids that couldn't jump, a narrow canyon would weed out wide droids, and a short door would weed out tall droids. The designers used the terrain leading up to key puzzles to reduce the complexity of the puzzle itself [...]"

WEENIE

Problem: Players might loose their sense of direction with respect to how the game world unfolds. This is a particular problem with player-driven scripting proceeding in lockstep with player actions.

Solution: The game has to establish clear leads that communicate to the player where to go, and what actions to attempt once there.

Consequence: The player will over time come to depend on the level of guidance, and will be confused if this guidance is dropped or omitted, which introduces a bias towards WEENIE CHAIN. Depending on the rigidity of the guidance mechanisms, players might become aware the out-of-game agenda behind the in-game setups.

Examples: On occasion this has been called the "carrot on a stick" approach to level design [8]. Stephen Clarke-Wilson [13] explains: "This somewhat bizarre term was coined by Walt Disney, who suggested that when designing massive 3D environments (theme parks), it was necessary to lead visitors through the environment the same way one trains a dog-by holding a wiener and leading the dog by the nose. Obvious weenies at Disneyland are Sleeping Beauty's Castle, which encourages guests to travel from the main entrance to the central hub; the former Rocket Jets, which encourage guests to explore Tomorrowland; the Mark Twain Steamship and dock, which encourage guests to explore Frontierland; and the King Arthur Carousel, which encourages guests to walk over the castle moat and into Fantasyland. [...] Your 3D VR environment needs to have standout landmarks so that it's easy to navigate without a map. The best games, which have typically been designed with very limited graphics, always save a few graphics to denote special and interesting things that should be investigated." Alternative means to communicate to the player the next proposed or mandatory step are explicit messages delivered by NPCs, or brief camera cuts that draw attention to doors opening in nearby locations, like that used in *Munch's Oddyssee*. Cliff Bleszinksi describes as the most subtle and always present WEENIE the incentive of seeing a new area: the player is always inclined to move from "been here, done this" into unknown territory [8].

WEENIE CHAIN

Problem: The player can lose sense of direction in the absence of clear indicators of where to go next. This is particularly a problem in games with large or non-linear environments that do not rely on rails or cut-

offs to constrain player movement, or employ "revisiting" of areas [8].

Solution: Create an uninterrupted chain of WEENIEs, with each link of the chain clearly perceivable from its predecessor, guiding the player from start to finish.

Consequence: WEENIE CHAIN might be limited in employing branches or otherwise introduce ambiguity or player choice. Enforced, WEENIE CHAIN might resemble a rail.

Examples: Noah Falstein refers to *Indiana Jones and the Fate of Atlantis* as an example of how to establish a chain of clear player goals using messenger NPCs [18]. His other examples are *Super Mario 64* and its use of signs and NPCs, as well as Halo. *Munch's Oddyssee* also uses signs, as well as a Shaman NPC for initial guidance.

Pattern candidates can be harvested from many sources. For example, the pattern FILTER is taken verbatim from a game postmortem. WEENIE is taken from a discussion of theme parks and techniques of environmental storytelling, and has been presented under a different name in lectures on level design. WEENIE also is the foundation of WEENIE CHAIN, illustrating pattern dependency, composition, and hierarchy.

Once you start looking for patterns, you start noticing them everywhere. Erich Gamma et al. point out that whether they are aware of it or not, novelists and playwrights already use patterns. Many "narrative devices" (see e.g. [32]) could also be described using a pattern template.

Sometimes the need to differentiate patterns on distinct levels of abstraction leads to different labels: "One person's pattern can be another person's primitive building block" (from [19]). Alexander et al. present a total of 253 patterns, subdivided into three sets for "Towns, Buildings and Construction" [3]. Preferences regarding the virtues of a given purpose might apply: patterns are just recipes, means not ends, and disagreement on the ends is beyond the scope of a pattern. Designers might also have different views on the quality of a given solution: one person's pattern might well be another's "anti-pattern" -- a recurring example of a bad design decision. Ideally, the pattern description itself is just a candid summary of cause and effect, describing one way to reach a given objective.

Alexandrian patterns are not unknown to game developers. Many programmers are familiar with Gamma's concept of software design patterns. Will Wright of Maxis credits the same book as the original inspiration for The Sims, and refers to Christopher Alexander's works (namely [1]). Steven Chen and Duncan Brown, former level designers at LucasArts, refer level designers to the 253 Alexandrian patterns merely as a guideline to create "meaningful environments" [10].

Instead, game developers strive to develop their own forms and templates. The Formal Abstract Design Tools (FADTs) proposed by Doug Church can easily be rewritten to fit the canonical pattern template. Another recent attempt to document game design knowledge in semi-formal ways is "The 400 Project" by Hal Barwood and Noah Falstein, is a project to collect proven game design rules and techniques. This 400 Project dates back to a GDC 2001 lecture by Hal Barwood, in which he also presented four examples.

Their effort has now spawned a column in *Game Developer* magazine, in which Noah Falstein

introduced the rule "Provide clear short-term goals" as the opening example. The 400 Project uses a five-part template: Imperative Statement - Domain of Application - Dominated Rules - Dominating Rules - Examples Aliases (see [18] for details). The concept of dominance (that some rules take precedence over others, and might in turn be trumped themselves) is absent from Alexandrian patterns. An Alexandrian pattern simply lists one possible solution to a given problem, acknowledging that other solutions might exist. Alexander sees each pattern as a recipe that strives to balance competing imperatives and conflicting forces in order to achieve the best possible results with respect to a specific objective. The decision to use a given pattern will also depend on whether there are multiple objectives to fulfill or when aside effects of applying a given solution, influence the. Falstein's rules appear more rigid, implying that all games have the same goals, and the same concerns. The names of the rules in the 400 Project are imperative statements, like Provide clear short-term goals, and their description contains only the solution part of a pattern. The problem to be solved by the pattern is merely implied or presumed obvious in this revision of the rules. WEENIE, as well as WEENIE CHAIN, capture parts of Provide clear short-term goals but also explicitly state the role of the pattern. Like rules, pattern names typically name the solution, not the problem: a matching pattern might just be called SHORT-TERM GOALS.

Outlook

Patterns are a formal means of documentation, and such means open the door to software tools for maintaining and editing game design documents. Take screenplays: word processors can be configured to handle the canonical format for scriptwriting, and some applications have been designed specifically to support that movie industry format. Many game companies and game designers have devised their own internal standards for game design documents. But defining a standard format for game design documents is of limited use unless there are editing and search facilities that support and enforce the format.

A pattern language is a natural match for descriptive markup, e.g. XML, with a huge potential for tool support based on off-the-shelf software. The value of any "living" document is directly related to the ease of its maintenance. Structured editing aids structured thinking: if a design document does not have clear organization, its use of keywords and names is inconsistent, and relevant information can not be located quickly when needed, the document is practically useless.

Consequently, game developers have to make a sustained, conscious effort to define and describe the recurring elements of their daily work - whether as patterns, rules or some other method -- so we can begin to create software tools made or adapted specifically for game design purposes. The case for Alexandrian game design patterns [22,29] seems strong: they have proven themselves in other and diverse professions; they are intuitive, well documented, and a familiar concept to software engineers, yet are flexible enough to permit anecdotak or informal descriptions of artistic choices.

Bibliography

[1] Christopher Alexander. *Notes on the Synthesis of Form*. (Harvard University Press 1964, 1966, 1979.) ISBN 0-674-62750-4 (cloth) ISBN 0-674-62751-2 (paper)

[2] Christopher Alexander, Murray Silverstein, Shlomo Angel, Sara Ishikawa, and Denny Abrams. *The Oregon Experiment*. (Oxford University Press, 1975.) ISBN 0-19-501824-9

[3] Christopher Alexander, Sara Ishikawa, Murray Silverstein, Max Jacobson, Ingrid Fiksdahl-King, and Shlomo Angel. *A Pattern Language: Towns, Buildings, Construction*. (Oxford University Press, 1977.) ISBN 0-19-501919-9

[4] Christopher Alexander. *The Timeless Way of Building.* (Oxford University Press, 1979.) ISBN 0-

[5] Brad Appleton. *Patterns and Software: Essential Concepts and Terminology*. Last update: Feb. 2000.

[6] Hal Barwood. "Four of the Four Hundred 2001". (GDC lecture, 2001.)

[7] Hal Barwood and Noah Falstein. "More of the 400: Discovering Design Rules 2002" (GDC 2002 lecture)

[8] Cliff Bleszinski. "The Art and Science of Level Design." (GDC 2000, pp. 107--118.)

[9] Jon Blossom and Collette Michaud. "Postmortem: LucasLearning's Star Wars DroidWorks" (Gamasutra 1999.) Originally *Game Developer* magazine, Vol 3, Issue 28, pp. 52-58, July 1999.

[10] Steven Chen and Duncan Brown. "The Architecture of Level Design." (GDC 2001 Proceedings, pp. 167--175.)

[11] Doug Church. "Formal Abstract Design Tools." (Gamasutra, 1999. Originally *Game Developer* magazine, Vol 3, Issue 28, July 1999.)

[12] Doug Church. "Abdicating Authorship: Goals and Process of Interactive Design." (GDC 2000, San Jose, Lecture 5403 (not in proceedings).)

[13] Stephen Clarke-Willson. "Applying Game Design to Virtual Environments" (*Digital Illusion*, ACM Press, Vol. 2, Issue 1, January 1, 1998.)

[14] (N/A).

[15] Chris Crawford. *The Art of Computer Game Design*, Chapter 6: "Design Techniques and Ideals." 1984.

[16] Troy Dunniway. "Using the Hero's Journey in Games." (Gamasutra, 1999. Originally published in *Game Developer* magazine, August 1999.)

[17] Troy Dunniway. *Professional Game Design.* (New Riders. To be published June 2002. ISBN 0-7357-1184-4.)

[18] Noah Falstein. "Better By Design: The 400 Project". *(Game Developer magazine*, Vol. 9, Issue 3, March 2002, p. 26.)

[19] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. (Addison Wesley Longman, 1994.) ISBN 0-201-63361-2.

[20] Chris Hecker and Zachary Booth Simpson "Game Programming Patterns & Idioms." *Game Developer magazine*, Sep. 2000.

[21] John Hopson "Behavioral Game Design." Gamasutra, April 2001.

[22] Bernd Kreimeier. *Game Design Patterns*. Wordware Publishing, Inc. To be published March 2003.) ISBN 1-55622-967-4

[23] Brenda Laurel. *Computers as Theatre*. (Addison Wesley Longman, Inc. 1991, 1993 ISBN 0-201-55060-1.)

[24] Marc LeBlanc. "Formal Design Tools: Emergent Complexity, Emergent Narrative." GDC 2000, San Jose, Lecture 5304 (not in proceedings).

[25] Gerard Meszaros and Jim Doble "A Pattern Language for Pattern Writing"

[26] Pierre-Alain Mueller. *Instant UML*. (Wrox Press, Ltd., 1997) ISBN 1-861000-87-1.

[27] Karen Pryor. *Don't Shoot The Dog!* (Bantam Doubleday, 1999) ISBN: 0-55338-039-7 (revised paperback edition)

[28] Andrew Rollings and Dave Morris. *Game Architecture and Design*. (The Coriolis Group, 2000.) ISBN 1-57610-425-7

[29] Andrew Rollings and Ernest Adams. *Patterns in Game Design*. (The Coriolis Group, to be published May 2002.) ISBN 1-57610-873-2

[30] Richard Rouse. *Game Design: Theory & Practice*. (Wordware, Inc., 2000) ISBN 1-55622-735-3

[31] Aamod Sane "The Elements of Pattern Style." December, 1995.

[32] Viktor Shklosvsky. *Theory of Prose Dalkey*. (Archive Press 1990, 1991.) ISBN 0-916583-54-6 (cloth) ISBN 0-916583-54-6 (paper).

[33] Zachary Booth Simpson "Design Patterns for Computer Games." 1998 CGDC Austin, TX, November 1998, also San Jose, CA, May 1999.

[34] John Vlissides. "Pattern Hatching - Seven Habits of Successful Pattern Writers." *C++ Report.* Nov/Dec 1996, and *Pattern Hatching: Design Patterns Applied* (Addison Wesley, 1998).

[35] Christopher Vogler. *The Writer's Journey: Mythic Structure for Writers*. (Michael Wiese Productions, 1998.) ISBN 0-941188-70-1

Game Programming Design Patterns are templates for building modularised code that are generally a repeatable solution to a commonly occurring mechanic applied in computer games. They are general solutions that arenâ€™t tied to a particular problem, making them reusable. One such pattern is object pooling. Whether a game needs an onslaught of All practicing the Design Patterns displayed in Robert Nystrom's 'Game Programming Patterns' for game development. 0 stars. 0 forks.Â Game-Programming-Patterns. All practicing the Design Patterns displayed in Robert Nystrom's 'Game Programming Patterns' for game development. About. All practicing the Design Patterns displayed in Robert Nystrom's 'Game Programming Patterns' for game development. Resources. Readme. game design patterns that describe player interaction while playing. Although the two. parts are the results of an intertwined process they can be used independently; the. structural framework can be used without the patterns to describe games and the use of. design patterns can be based on other structural frameworks. Due to limited space, we. do not present a detailed description of the structural framework and refer interested. This is a tutorial on game programming patterns in Unity with C# code. Another name for the same thing is software design patterns. You will learn the following programming patterns: command pattern, and much more.Â In software engineering, a software design pattern is a general reusable solution to a commonly occurring problem within a given context in software design. It is not a finished design that can be transformed directly into source or machine code. Developer Blog #1: Game Design Patterns and Anti-Patterns. Throughout the design of Arkis Vir, we have been balancing mechanics that are fun for video games, while incorporating a few key mechanics that give board games their allure.Â Not very board-gamey but very fun nonetheless! So here is Part 1 of a 3-part developer blog series that will attempt to illustrate and explain how weâ€™ve decided what gets put in the game, and what doesnâ€™t.