The Hong Kong Polytechnic University

Department of Computing

Adaptive Dynamic Game Balancing Based on Data Mining of Sequential Patterns

CHIU SUK YI, KITTY

A thesis submitted in partial fulfillment of the requirements for the Degree of Master of Philosophy

January, 2007

CERTIFICATE OF ORIGINALITY

I hereby declare that this thesis is my own work and that, to the best of my knowledge and belief, it reproduces no material previously published or written, nor material that has been accepted for the award of any other degree or diploma, except where due acknowledgement has been made in the text.

(Signed)

CHIU SUK YI KITTY (Name of Student)

Abstract

It is the responsibility of a game or game level designer to provide players with a balanced game, one that offers a satisfying level of challenge. This can be done using traditional game programs and artificial intelligence (AI) techniques but it is becoming increasingly common for researchers are using dynamic game balancing, which uses reinforcement learning and focuses on the movement of non-player characters, especially in scripted games. However, this is not suitable for all game genres, such as those that use mazes or require dynamic terrains. In this paper we propose to adjust the level of difficulty of a game by using data mined for sequential patterns that can be used to analyze a player's behaviors. Our method first mines individual gameplay data and then transforms it into a set of sequential patterns. This approach is tested here on a maze game. We capture the behavior of several player's and the parameters of the game environment such as the number of dead ends achieved, hammers used on every level, how quickly players achieve the goal, the size of the maze, number of monsters hit, and the number of walls broken on each level. Feedback from participants in our experiments was very positive as they found the games designed using the proposed approach to be both more interesting and more balanced. This proposed approach differs from existing rule base game AI algorithms in three ways: (1) the game levels are based on the past experience of the player; (2) the approach is data-driven; (3) the game levels are unique and are not predefined, making them more adaptive, which contributes to making the game more interesting and balanced.

Acknowledgements

It is my pleasure to express my sincere thanks to my supervisor, Prof. Keith C.C. Chan, for providing useful guidance, constructive suggestions, and continuous support in my graduate study at The Hong Kong Polytechnic University. Nearly all of the work towards my master's degree originated from Keith. Without his supervision, I could never have had the great satisfaction of completing this project. The experience of working with Keith will be a lifelong asset.

I would also like to acknowledge my thanks to Dr. Korris F.L. Chung and Dr. Robert W.P. LUK. They have provided valuable suggestions on my research.

Most importantly, I would like to thank all of my family members for their patience, understanding and support.

Table of contents

Abstract i

Acknowledgements iii

Table of contents iv

List of figures vi

List of tables viii

Chapter 1 Introduction 1

1.1 The Problem 8

1.2 Overview of Proposed Approach 10

1.3 Outline of Thesis 12

Chapter 2 Game Related Literature 13

2.1 What is a Game? 13

2.2 Game Level Design 18

2.2.1 Guidelines for working with game level design 20

2.2.2 Game balance and level design 24

2.3 Game AI 29

2.3.1 Academic and Game AI 30

2.3.2 Classic AI Techniques 33

Chapter 3 Data Mining on Game Level Design 42

3.1 Data Mining Analysis 42

3.1.1 Association pattern mining 44

3.1.2 Apriori and Other Algorithms 45

3.1.3 Sequential Pattern 47

3.1.4 Sequential / Temporal Pattern Mining Algorithms 49

3.2 Multivariate Analysis of Games 54

Chapter 4 Game AI with Prediction 62

4.1 Sequential Pattern Mining Approach 65

4.2 Decision Tree Approach 82

Chapter 5 Experiment and Implementation 88

5.1 Game Description 88

5.2 Details of the implementation 91

5.2.1 Game Engine 94

5.2.2 Maze Builder 97

5.3 Details of the experiment 98

Chapter 6 Conclusion and Future Research 114

6.1 Summary of the thesis 114

6.2 Future research 115

References/Bibliographies 116

List of figures

Figure 1 Doom 7

Figure 2 Spacewar!, the first computer game. (1962) 14

Figure 3 Pac Man [PAC] 19

Figure 4 Map of “Dungeons and Dragons” [DD] 20

Figure 5 Unfair Game 26

Figure 6 Stalemate 26

Figure 7 Game is balanced, but too short 27

Figure 8 An ideal balance graph 28

Figure 9 Grid 39

Figure 10 Mabinogi 41

Figure 11 Game Flow of the Level design using random number generation 63

Figure 12 Game Flow of the Level design using pre-defined template 64

Figure 13 Game logs data of ***m*** levels 66

Figure 14 Algorithm for Chi-square Testing: 69

Figure 15 Prediction Rule Generation Algorithm 74

Figure 16 Game Flow of the Maze game using sequential pattern mining 75

Figure 17 Basic algorithm of ID3 84

Figure 18 Algorithm of C4.5 84

Figure 19 Game Flow of the Level design with decision tree approach 87

Figure 20 Maze Game 90

Figure 21 System Diagram of the Maze Game 91

Figure 22 System Flow of the Maze Game 92

Figure 23 Class Diagram 93

Figure 24 A tile-based game 96

Figure 25 Maze Game 98

Figure 26 Decision Tree 105

Figure 27 Balance graph of the three versions of the maze game for one player 112

Figure 28 Balance graph of the three versions of the maze game showing the average performance of the ten players 113

List of tables

Table 1 A Two Dimensions Contingency Table with ***I*** Rows and ***J*** Columns 67

Table 2 Description of the Parameter of a level.. 76

Table 3 Description of the Parameters of player’s behavior....................................... 77

Table 4 Game Log Data 77

Table 5 Parameters of levels.. 78

Table 6 Player’s Behavior 79

Table 7 4 X 6 Contingency table for NUMBER OF HAMMERS USED and DEAD ENDS ACHIEVED.. 79

Table 8 Game log data... 90

Table 9 Sample XML for a level.. 94

Table 10 Sample XML file of the player’s behavior... 95

Table 11 Sample XML for the prediction rule.. 95

Table 12 Data representative of the map in Figure 24.. 97

Table 13 Parameter of the Levels with random number 100

Table 14 Player’s Behavior of playing random maze.. 100

Table 15 Parameter of the “Hard” Levels with decision tree approach...................... 102

Table 16 Parameter of the “Easy” Levels with decision tree approach 102

Table 17 Classification Rules 103

Table 18 Player’s behavior on Maze with Decision Tree 106

Table 19 Default Parameters of Maze 107

Table 20 Parameters of Maze with Prediction Rule of a player................................. 108

Table 21 Player’s Behavior in Maze with Prediction Rule... 109

Table 22 Some Prediction Rules at Level 10 109

Table 23 Question of the feedback form 110

Table 24 Player ratings of original Maze Game vs, Predicted Maze Game................111

Chapter 1 Introduction

Computer games provide an extremely popular form of entertainment on a wide range of platforms far beyond the familiar personal computer and include games played on devices as different as the Sony PlayStation, the XBox or a simple mobile phone. There are many game genres including first-person shooter games, real-time strategy games, role-play games, games played with teammates, action, adventure, educational, simulations, sports, fighting games and puzzles [BB2004]. Game development is similar to system development in that it includes game design, game development and game testing. [WIKI1] Every activity of the game development process involves many sub-tasks. The main tasks of game designer are storytelling and the design of characters, levels, scenes, graphics and sound effects. Game programmers focus on the game programming and the design of the game engine. Game testers use a variety of testing methods to ensure players will find a game interesting.

There are many guidelines for designing a perfect computer game. Schuytema[PS2007] says that a good game designer should understand a player's perception and emotions. The player's perceptions include sound effects and music, character design, movement, light and color, and the game flow. The player's emotions are affected by the game flow, by achievement, by the difficulty of the problem and by the element of surprise. When designing a game, designers should pay attention to both the emotional responses of players and to their perceptions.

Game programmers commonly use Java, C#, and C to create a game engine, which is the core software component of the computer game. An engine interprets the game level design and implementation, graphic rendering, collision detection, sound effects, scripting, animation, networking, memory management, artificial intelligence and other elements of the game.[WIKI19] These elements all contribute to a player's perceptions and emotions and therefore to the interest of the game. To make games even more interesting, we can use game artificial intelligence (AI). At a general level, artificial Intelligence refers to using machines or agents in a more human-like fashion and can be classified as either "weak" or "strong". Weak AI is often used in academic work by applying AI technologies to real world problems. Strong AI uses an agent or algorithm that mimics the human thought or behaviors. Almost all game AI is weak AI. [BG2004] Thanks to the rapid development of CPUs and RAM, developers more recently have started to use sophisticated strong AI, usually more familiar in the academic world, such as Genetic Algorithms, to create systems that mimic human thought processes. [BG2004], [MB2005], [WIKI2], [WIKI3]

Today, computer games use a variety of forms of AI. [BG2004] It is used in simple games such as Tetris or Bejeweled to complex online massively multiplayer games (OMMPG). Generally, the game AI is applied to produce the illusion of intelligence on the non-player character (NPC). Some game AI provides a series of challenges or goals that must be met or reached in the course of playing a game. Some game AI is used to catch cheats. Other game AI may provide players with instant feedback.

There are two chief difficulties in using game AI. First, game AI is dependent on the number of processor cycles and the amount of available memory and its programming is mostly platform dependent. Second, the costs of game development in terms of both time and money are huge. Game companies cannot afford to research game AI. A lot of game programmers are still using a lot of classic game AI such as finite state machines and flocking. Genetic Algorithms, neural networks and expert systems are modern techniques in game AI. [RR2005] [SR2002] [AN2004] [BB2004] [BG2004] [BM2005]

A successful computer game can be defined by a variety of factors: Online integration, Technology, Genre, Purpose of Play and Business Model. [MF2003] But the grand challenge of game AI is to be able to use it to make games more fun. “Fun” is the most important determinant of whether a game will be successful. Imagine a massively multiplayer online game on the Internet. The profits of the game provider depend on spending on the player’s weapons and equipment and a monthly fee. If players feel the game is interesting, they will spend more time on the game, and spend more money to stay or to buy equipment. There are many potential sources of “fun”. Fun can come from challenges. Challenges and goals can be many things, for example, victory in a scenario, the accumulation of money, or the right to move to the next level. Goals and challenges may be on the one hand pre-defined by the game engine and it may be a requirement of the game that every player reach or meet them or on the other hand they may be player-specific and be the result of or related to their game-playing activities or preferences.

Challenges are strongly related to the design of game levels, also known in many games as “maps”, “stages” or “missions”. Game levels are part of many genres including puzzle games, adventure games, role-play games and sports games. A level also includes a game environment in the game world of the player. Some game genres such as adventure games or online massively multiple-player games change the game environment and match mission levels to the player’s experience. This is also called a “level”. Game level design is a process of creating game levels and is a sub-task of game design. Game level designers need to create a set of challenges within the level and maximize the fun of the game. The game level engine is a tool or algorithm to establish a game level during game development. Game level designers can design a level using a level editor such as Valve's Hammer Editor, Epic's UnrealEd, Leadwerks 3D World Studio, BioWare's Aurora Toolset and id

Software's Q3Radiant. [WIKI16]. The key to good level design is to challenge the player in a satisfying way, to make the game “playable”. However whatever a level is designed, some players will feel the game is hard but others may feel it is too easy. One way to deal with this is to use an adaptive game engine to generate an adaptive level. This means that the difficulty of the level can be adjusted dynamically according to the individual player’s behavior and the game environment. This adjustment of the difficulty is also called dynamic game balancing. [WIKI14]

There are a number of approaches to ynamic game balancing. One approach is to modify the behavior of the Non-Player Characters (NPCs). Research into dynamic scripting is focused on the NPC’s behavior [PIE2004] [SL2006] [PIE2004] but is suitable only for games that are scripted or imply storytelling. Andrade [GA2005] also used reinforcement learning to modify NPC’s behaviors. Another approach is to change the parameters of the game environment. Hunicke and Chapman’s [RH2004] approach is to control the game environment setting according to the player’s behavior and then adjust the difficulty of the challenges. Buckland [BM2005] uses rule-based AI in fighting games with dynamic game balancing. Game AI is used to predict a player opponent's next strike. This algorithm is based solely on the player’s previous actions and seeks to prediction the NPC’s action. Apart from player behaviors, we can also use a game log to record the parameters of a game environment. Such gameplay data can be use both to predict NPC actions and to adjust the difficulty of the game by changing the parameters of the game environment in a scene. the game more balanced and playable, as itailor–made for the specific player. In this way, a 12-year-old boy will not find the game too difficult to play and an adult will not find it too easy. Here, data mining is a useful tool in the design of game levels.

Data Mining can be used to adjust the level of difficulty of a gamey. Data Mining, also called Knowledge Discovery in Databases (KDD) or Knowledge Discovery, is used to extract or mine knowledge from large amounts of data. [JH2001]. Data mining includes activities such as clustering, classification, association, and sequential pattern discovery. Different activities are for different approaches. Data mining involves several issues and goals [JH2001]. Various datasets can be mined to discover knowledge which can be used to, for example, predict weather conditions at airports or to predict stock prices or trends.

Gameplay data is both sequential and temporal. Sequential pattern mining is used for prediction and is the process of mining frequently occurring patterns that are related to time or other sequences. [JH1999] [JG1993], [HJ2001], [JP2000], [JA2001], [KC1994] We can use it to predict a player's behavior and game environment, to adjust the difficulty of the level of the game, and to make the game more adaptive.

Sequential pattern mining can replace traditional rule-based game AI or pre-defined game environments when we build game scenes. For example, the terrain or maze of a scene is pre-defined on Ultimate Online, Final Fantasy and Doom (Figure 1). After playing several times, a player may become bored with the unchanging environment.

Figure 1 Doom

In this thesis, we propose a new approach to game level design that uses data mining techniques. We propose an approach to using sequential pattern mining to adjust the difficulty of the levels of games that require terrain generation. We do this by using the game log to find the relationship between a player's past behaviors and the parameters of the game environment. The player's behaviors can then be predicted and the game level can be adjusted to the individual player. To replace the traditional rule-based game AI or pre-defined game level template, we use sequential pattern mining. Our proposed approach is as follows. First we detect underlying patterns in an ordered game log over a series of levels. We then use these patterns to construct sequence-generation rules that can predict the attributes of a player and those attributes are used to generate a game environment of a suitable level of difficulty.

1.1 The Problem

A game level is built on a set of small challenges [WIKI5], such as an enemy, a maze, or some hybrid challenge, the game environment [TR1999], or the number of enemies. Most of the elements of the game environment are related to the mission. A balanced game will provide a satisfying level of challenge to the player with a current level being more difficult than a previous level. Although some games allow players to adjust the basic level of difficulty, the levels of most games are pre-defined during the game design process and the overall level of the game is static. As a result, some players will feel a game is easy and others that it is hard. The game level designer may apply game AI to control the behaviors of the NPC, provide instant feedback to players, or generate a balanced game world.

The development of tailor -made game AI is constrained by unpredictable cost and time requirements and the limitations of computer resources. A lot of game programming uses existing game AI, pre-defined game level templates for the development of a game. Exceptions are games [SW1999] [AN2004] [PMIE2006] such as Creatures (CyberLife Technologies, 2001) and Black and White (Electronic Arts, 2001).

Some game level design allows players to adjust the basic level of difficulty but the levels of most games are defined in the design process and the overall level of the game is static. As a result, some players will feel a game is easy and others that it is hard. The adjustment of the difficulty of a game during play is called dynamic game balancing. [WIKI14] Dynamic game balancing uses an automatic algorithm to change parameters, scenarios and behaviors in the game to prevent players feeling frustrated during play. There are a number of different approaches to dynamic game balancing, including modifying the behaviors of NPCs [PIE2004] [SL2006] [PIE2004] [BG2004] and changing the parameters of the game environment. [RH2004]. However, those researches are focusing on the scripting game or first person shooting game (FPS). They are not suitable for other game genres. For example, if the scene is a maze, dynamic scripting cannot be used to create a dynamic terrain for the player.

We propose using a sequential pattern mining approach to analyze the game log, which records player behaviors and the parameters of the game environment. Then we generate a suitable game scene environment for the player at a particular game level by adjusting the parameters of the new level based on positive feedback [EA2002]. This makes the game more balanced and the playability of the game increases, as it seems to be tailor–made for the specific player. In this way, a 12-year-old boy will not find the game too difficult to play and an adult will not find it too easy.

This approach differs from those used in existing algorithms in three ways: (1) the game level is based on the past experience of the player; (2) the game level is data-driven; (3) the game level is unique and is not predefined by changing the parameters of the game environment. Not only can this approach build a maze, it can also help the game to build terrain, which may not be predefined.

1.2 Overview of Proposed Approach

Our proposed solution implies developing a sequential pattern-mining algorithm for the game engine to provide a suitable game environment. Gameplay data of the individual player will be collected and transformed into a series of sequential patterns. These patterns are defined as follows:

In order to mine a sequential pattern, we first suppose that there is an ordered sequence $\boldsymbol{S}$ of M BEHAVIORS which appear in a time series, $\boldsymbol{Behavior_1}$… ,$\boldsymbol{Behavior_p}$,…$\boldsymbol{Behavior_M}$, where $\mathrm{Behavior}_p$ is located at time=$\boldsymbol{p}$ in $\boldsymbol{S}$. Further suppose that each Behavior in the sequence is described by $\boldsymbol{n}$ distinct attributes, $\boldsymbol{Attr_{1p}}$, …, $\boldsymbol{Attr_{jp}}$, $\boldsymbol{Attr_{np}}$, and that in any instantiation of the Behavior description, an attribute $\boldsymbol{Attr_{jp}}$ takes on a specific ordinal property that is defined according to some of the attribute's past values, $\boldsymbol{val_{jp}} \in \boldsymbol{domain(Attr_{jp})}$ – $\{\boldsymbol{v_{jk}} = 1,2,\ldots \boldsymbol{J}\}$, which may be numerical, symbolic, or both. [KC1994]. It is then possible to use the information theoretic measure to identify relationships between various attributes found on different levels.

It is also possible to determine a set of prediction rules that can be used to predict the characteristics of a future Behavior. For example, in a maze game, the game engine can predict that if a player used a hammer three times, the player will meet a dead end two times on the next level. We can do this by using sequential pattern mining that uses a probabilistic inductive method, which consists of three phases:

(1) Detection of underlying patterns in an ordered event sequence of time series $\boldsymbol{t}$.

(2) Construction of sequence-generation rules based on the detected patterns

(3) Use of these rules to predict the attributes of player and generate a suitable environment with those attributes.

1.3 Outline of Thesis

This thesis is organized as follows. Chapter 2 describes current related work done using AI for game development. In chapter 3 we will describe the current technology for mining sequential patterns. In Section 4 we will give an overview of the proposed solution using sequential patterns in game AI. In chapter 5 we describe the implementation on a test model of our approach to analyzing and predicting gameplay. We also provide the implementation details and the pre-defined parameters or rules, and our experimental results. Chapter 6 concludes the thesis.

Chapter 2 Game Related Literature

In Section 2.1, we present the history of the game and game design. In Section 2.2 we present the basic concepts of game level design. Section 2.3 presents some classical game AI.

2.1 What is a Game?

A game is a series of processes that takes a player to a result; it is a series of interesting decisions. [PS2007] But what does it mean for a decision to be "interesting"? To take a classic game, Pac Man, as an example, it would not be interesting if it could wander though the maze without ghosts or without ghosts of varying kinds in various situations. The interest of Pac Man lies in deciding which row of dots to chomp based on the player's current positions and statuses of the ghosts so that the player can eat all dots and finish a level.

A.S. Douglas invented the first graphic computer game "Tic-Tac-Toe" in 1952. [INV] This game was programmed on an EDSAC vacuum-tube computer, which had a cathode ray tube display. In 1962, Steve Russell used a mainframe computer to develop the computer game "SpaceWar!" at MIT.[JJ 2001] "SpaceWar!" was the first game intended for computer use (Figure 2).

Figure 2 Spacewar!, the first computer game. (1962)

Developing computer games is now a truly major industry. Computer games have developed from stand-alone games to massively multiplayer online games. In 1962, the computer games had simple game logic and interacted with the user through a vacuum-tube computer. When Ultima Online was released in 1997, a new section of the computer game market, the online game, was born. Today, a computer game has complicated game AI and players interact with other through the Internet.

There are many guidelines for designing a perfect computer game. In the previous chapter we discussed Schuytema's [PS2007], which focused on understanding a player's perception and emotions. But Schuytema [PS2007] also suggests a set of elements that a game should contain. A game should have a goal and target for the player. The game should allow the player to know their progress, and how to achieve the final goal. Sometimes, as in RPG games, the player may not know the goal on the initial stage. The game will then guide the players' action and their perceptions and help the player recognize the master goal of the game.

The rules of the game need to be clear and understandable, as this will affect the players' responses and actions in the game. Players need to know what they can and cannot do in the game. If the logic of the game rule is not clear, the player will spend more time trying to understand the game than enjoying it.

The game world should have its own rules but a player's responses should not be fully controlled by the game rules, as this is frustrating. Schuytema suggests that variability and malleability can make the game world more fun.

It is important that players be able to understand the context. When players play the game, they want to get a new and exciting experience though the game world. The game world might be quite different from normal life. For example, in the game world, humans can fly without equipment. The differences between the game and real worlds are a source of interest to players as they attempt to understand the world of the game but the game world and the context of the game cannot be unstructured and confusing to the player. This reduces the playability of the game.

The game needs to grab and hold a player's attention by engaging the player's mind and testing his dexterity. There is no game without a challenge to the skill of the player, an adrenaline rush, or a sense of accomplishment.

Feedback can motivate players. Games are full of interactivity and this can be between computer and player or between players. Feedback allows players to determine the amount of success or failure they are achieving. The game should provide vital information about what the player just did and clues for the player to avoid repeating mistakes in the future.

The user interface of the game should be consistent. It should be smooth and easy to understand. Usually, learning a game involves a learning period but this should be as short as possible.

A rest break is important to the flow of a game. In these periods, players can settle back into their character and prepare for the next sequence of action. The game can provide simpler missions between missions of a heavy intensity, or downtime within a mission.

The game flow should not be predictable. Randomness can be used to make a game less predictable.

The game should allow the players to know where they are. Generally, if the terrain is not intended to be a maze, the game should provide interface tools that to navigate them where they are. The navigator may be a map, a guiding arrow or other feature. Players like to explore a game world but they don’t want to get lost in one.

It is important for players to be able to enjoy a sense of accomplishment. The game should provide some small challenges with opportunities for sub-victories for the player. For example, the “Nested Victory” approach is used in a lot of football games. The player should get the most points he or she can before the game finishes in a set time. Such games also provide additional challenges in the form of semis and finals.

Player failures must also be dealt with. Games should include penalties for failure, such as loss of equipment and reductions in strength. Balance is essential here so as to avoid frustrating and hamstringing the player.

Game AI has uses in all of these areas. It can make a game more interesting and interactive. Some game AI can create interesting play and challenging patterns for players. Some AI provides instant feedback about the non-player character (NPC) to the player. Some game AI is built as a “failure counter” into game logic. If the player fails several times, a hint or a power-up provides the player with some help.

2.2 Game Level Design

A game level is also called a “map”, “stage” or “mission” on many games. It is used on a lot of genres, such as puzzle games, adventure games, role-play games and sports games. [WIKI19]A game level is built on a set of small challenges [WIKI] and the game environment [TR1999]. Some challenges may be an enemy, some may be a maze or some challenges may be hybrid. The current challenge of a level should be more difficult than that of the previous level. The game environment includes the terrain design, number of enemies, when and where the player resources are revealed and so on. Most of the elements of the game environment are related to the mission. The purpose of the remaining elements is to enhance the attraction of the game world. For example, a game level in a racing game may have a player and a non-player character (NPC). T The game level designer not only designs the route of the race. He also needs to design scenes such as the position of the NPC, the venue of the route and the path, the audience to enhance the attractive of the level and the position of the power-on.

Pac Man (Figure 3) is a classic game that contains game level design. Although the map never changes, An increase in levels means decreasing the effective time of the power-up and increasing the speed of ghosts’ movement to increase the difficulty of a level. Some role-playing games or advantage games, such as “Dungeons and Dragons” [BE2005] (Figure 4), change the scene along with the level. The tower scenes in Dungeons and Dragons change and become more complex when the game level increases. In both Pac Man and Dungeons and Dragons the levels however are static. The difficulty of the game level cannot be adjusted dynamically according to the player's experience, skill and ability when he or she is playing.

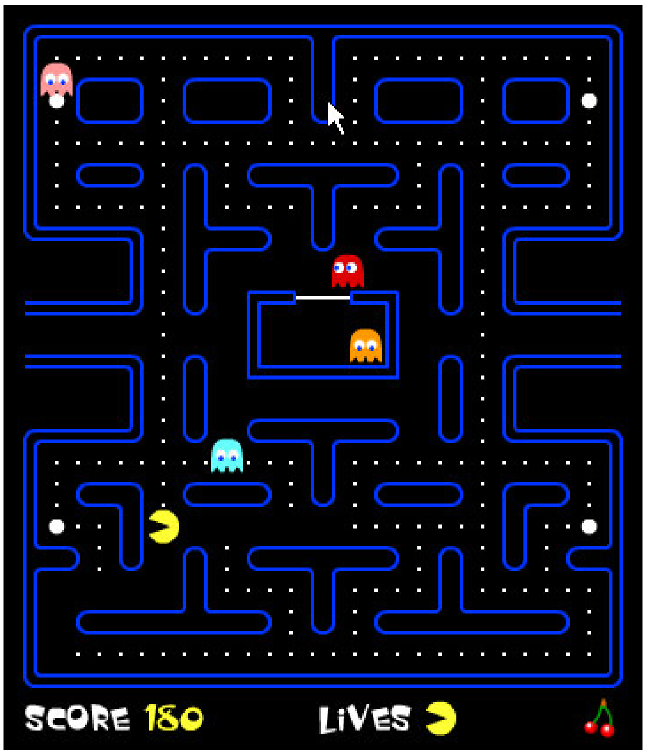
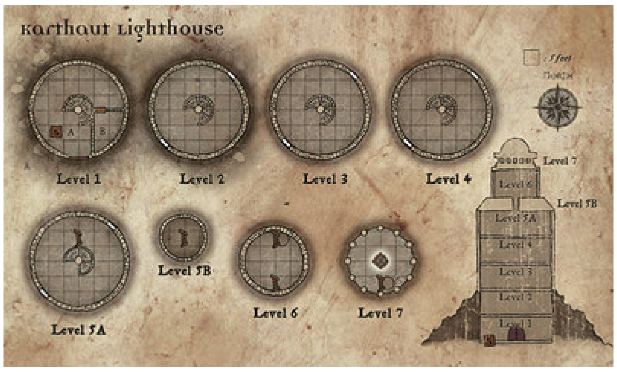


Figure 3 Pac Man [PAC]



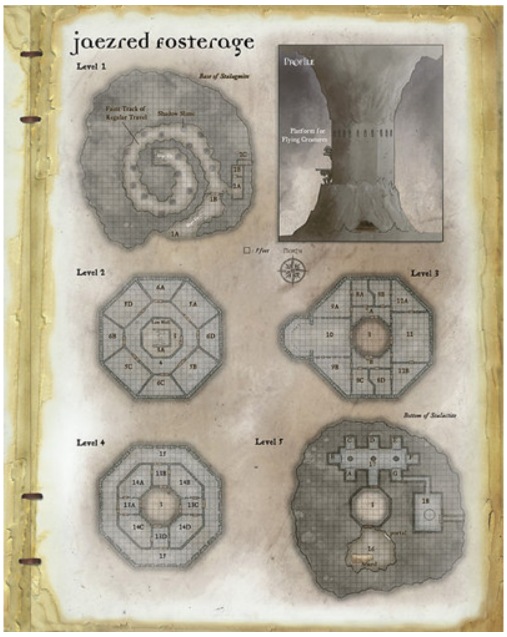


Figure 4 Map of “Dungeons and Dragons” [DD]

In the early years of the computer game industry, game designers had to design every level. In the past ten years, level designers have become responsible for the creation of the game level. [WIKI17]

2.2.1 Guidelines for working with game level design

Game level design makes a game more interesting. The challenges or mission can be different at every level. The major task of the game level designer is to challenge the player and balance the game. Tim Ryan offered the following advice on game level design [TR21999].

1) The game level designer should understand the main idea of the game.

The Level designer should maintain and express the game designer's vision. The Level designer should maintain good communication between the producer and game designer so as to "capture" their concept of the game.

2) The game palette must be clearly designed. The design palette includes all of the art and the game play elements such as power-up, weapons, game play puzzles and possible solutions.

3) Experimentation and implementation is important to game level design. The designer can evaluation a level repeatedly to make sure that the level is interesting enough to the player.

4) Every level should offer different types of challenges, for example, routes, traps, power-up, enemies.

5) Every level should provide multiple or alternative solutions matching players' styles and their learning abilities. For example, some players like playing conservatively but others like to play it risky.

6) Level can include secrets, alternate paths or shortcuts as unexpected rewards.

7) Pacing of a level relates to the conflict, tension and excitement of the player. The designer can consider using time limits, the movement speed and the distance between player and goal to keep the level at a certain pace and interest.

8) The revelation of the assets can keep the player interested in the game. The assets can be terrain BEHAVIORS, enemy BEHAVIORS, power-up equipments and so on. The designer should design the position, ordering and timing of the revelation of the assets.

9) The core idea of the game should be clear.

10) The difference in the difficulty between levels should increase during the game. At the beginning of the game, it should be easy for a player because that is the learning period. The player's ability, skill and equipment should be enhanced after some level is passed. The designer should thereafter test the players' mettle and make them uncertain of victory. At the end of the game, levels should be more difficult because by this time the player has greater skill and more resources. There are several solutions to help designers design a suitable progression of levels. First, designer can scale up or down the difficult in the level without grossly change the game play or the fun factor. Second, designer can reposition the level in the game. Finally, designer can use

"change-of-pace" levels. A "change-of-pace" level is easier than a previous level but allow the player to play with some unusual action or pattern. For example, after fighting with "big brother" on the game level, an easy level shows to player to give more lives or "power up".

11) Try to make the game unique. The level designer should use levels to combine the game elements and present the vision of the game to the players. If the game is unique or novel, players will find it more interesting.

12) Don't assume that players have read the instructions, dialogues, or mission descriptions. A clear picture or layout is often the best way of providing players with guidance as to how to complete a particular.

13) The "event horizon" can be positioned unpredictably or changeably. The "event horizon" is a region that shows the player the upcoming terrain or enemies that must be engaged. If the event horizon is always positioned the same, players will become bored.

14) Players may have expectations of a level based on what they may have already seen or been told. Player's expectations can change as more information is provided. The level should contain surprises.

15) The level should assume a player has a median skill level.

16) Every player has his or her tricks for the strategies and tactics for solving a puzzle or problem in a game. The designer should know those tricks and take them into account when designing the level. This is one of the indices for the level of difficulty of a game level.

17) The increase in difficulty from one level to the next should not be too great.

18) The level designer is also a player’s adversary. Players expect game AI to respond to them as humans. The strategy of the game AI should provoke fear in a player and prey on the player’s weaknesses

19) Designers should accept that the testing cycle in level design can be quite long. This is because game levels must be tested and retested until they are suitable for the players for whom the game is intended.

20) The more time you spend designing a level the better it will be.

2.2.2 Game balance and level design

A balanced game is fair to players and is fair in a way that is balanced between players, in a multiple player game, or between a human player and a computer. On a game, it contains different challenges or decision to allow player to decide. The game level designer should ensure that the game is balanced, does not frustrate players, and retains their interest.

The goal of game balancing is genre-dependent. For example, a role-play game (RPGs) should be designed so that the monsters and other adversaries are not too hard for the player to defeat. Action games should not provide weapons or power-ups that make tasks too easy.

Positive Feedback is [EA2002] uses positive feedback to makes things easier for the player when he is ahead. For example, in a single-player role-playing games, a player starts with poor weapons. He can get better weapons by killing monsters or getting treasure and this will allow the player to get more or better weapons and treasure, and so on.

Ernest Adams [EA2002] uses a balance graph to show how to use positive feedback to balance a game. Time is represented on the horizontal axis. The vertical axis uses a number to show who's ahead and shows the difference in points scored by the two players and whether a game is not balanced. [Figure 5] shows an unfair game where the advantage is with player A and he quickly wins.

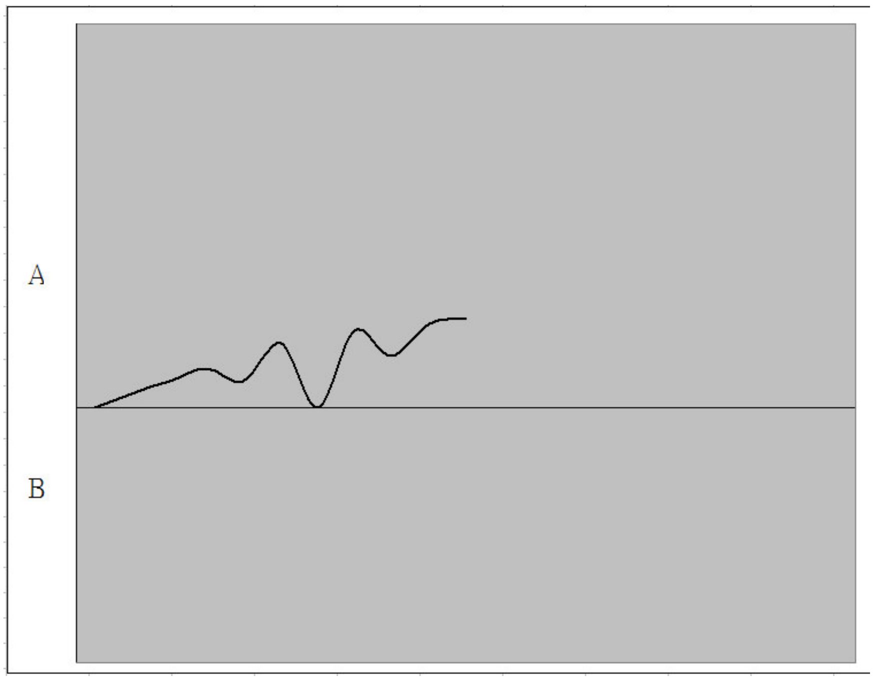


Figure 5 Unfair Game

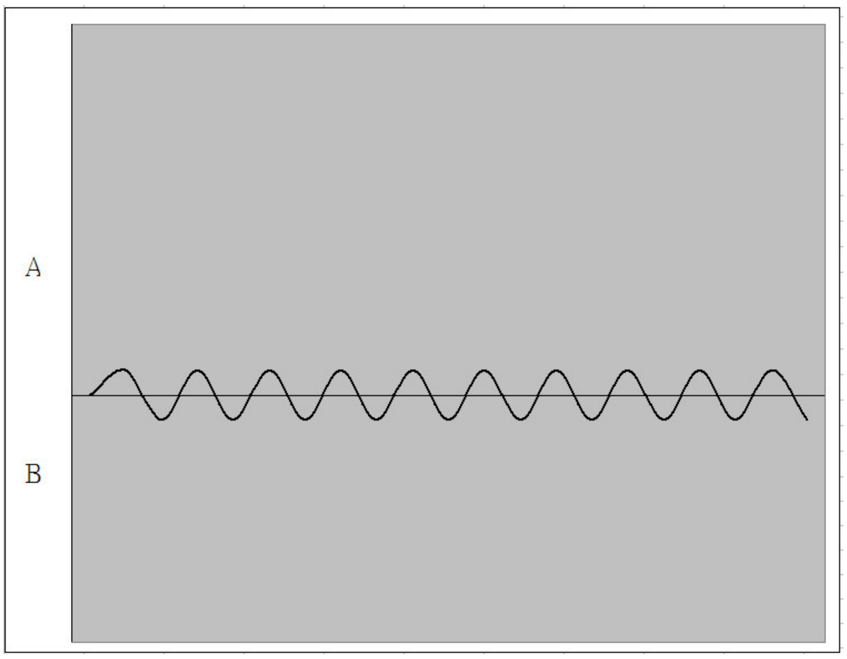


Figure 6 Stalemate

[Figure 6] shows a game in a stalematee. No player will win this game. This will not happen in a balanced game as a better player will receive positive feedback. The graph in [Figure 7] shows a balanced game where the positive feedback is received too soon. Player B is winning at first but player A wins ultimately and too soon.

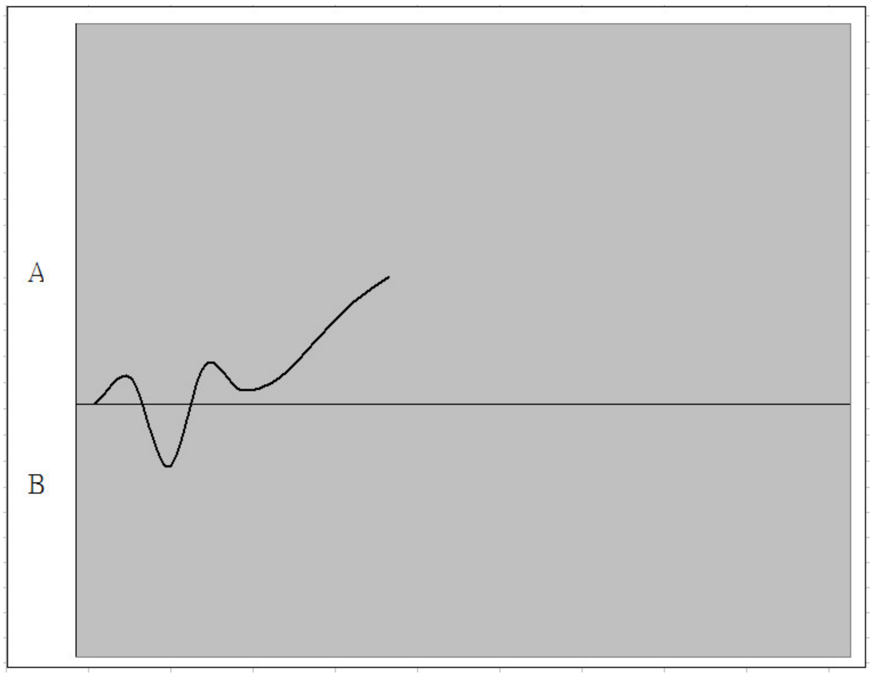


Figure 7 Game is balanced, but too short

Figure 8 shows an ideal balance graph. Player A and player B compete over a long period with the advantage initially changing hands but ultimately one player gets the upper hand and retains it, even while the progress to victory is fraught with regular setbacks, maintaining the hope of the ultimate loser right to the end.

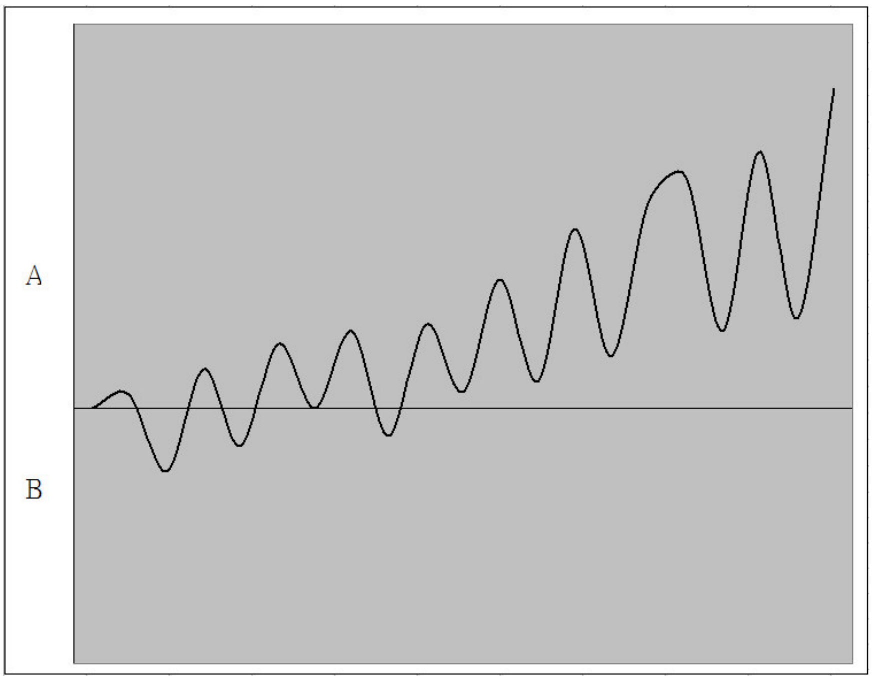


Figure 8 An ideal balance graph

Game balance is applied during of game level design. Game level designer should ensue the game is balance when designing a level. Most game levels are defined during game design. Some games allow players to adjust the basic level of difficulty yet the the overall level of the game is static. Some players will feel that such a game is easy and others that it is hard. Adjustment of the game difficulty is called dynamic game balancing. [WIKI4] Dynamic game balancing uses an automatic algorithm to change parameters, scenarios and behaviour in the game to maintain player interest. Dynamic game balancing may modify NPC's behaviors or the parameter of the game environment.

Hunicke and Chapman [RH2004] adjusted the difficult of the game by changing the game parameters by using a system called "Hamlet". This is a dynamic difficulty adjustment system which uses Inventory Theory and Operations Results to analyse and adjust the supply and the demand of game inventories in order to control the difficulty of the game. Hamlet is used on first person shooting games.

Dynamic game balancing as applied to dynamic scripting focuses on the [PIE2004] [SL2006] [PMIE2006] is suitable only for games that are scripted or imply storytelling. For example, Sangkyung [SL2006] used a Gaussian Mixture Model on the dynamic scripting for a shooter game. He used a Gaussian Mixture Module to model the player's reaction pattern and to maintain the level intended by the level designer in the shooter game. Spronck [PIE2004] [PMIE2006] used reinforcement learning on the dynamic scripting to control the movement of the NPC. Andrade [GA2005] used reinforcement learning to modify NPC behaviors.

2.3 Game AI

This section first compares and provides some background to academic and game AI and then describes in some detail the seven different types of game AI. We describe game the AI of Terrain Builder in even greater detail because of its important role as a game environment and level design technique and provide further detail about three basic algorithms used in Terrain Builder.

2.3.1 Academic and Game AI

At a general level, artificial Intelligence refers to using machines or agents in a more human-like fashion and can be classified as either “weak” or “strong”. Strong AI refers to an agent or algorithm that mimics the human thought or behaviors. The agent may have emotion, consciousness, self-awareness, etc. Weak AI, also called applied AI, is often to the use of software to accomplish specific problem solving task that may not be especially to the human cognitive abilities. John Searle uses “Weak AI” and “Strong AI” to distinguish between two different hypotheses about artificial intelligence.[WIKI3]. Game AI is weak AI because the goal of the game AI is to design agents that provide the illusion of intelligence to enhance interest. [BG2004] In the early days of computer games, game AI was dependent on the number of processor cycles and the amount of available memory. It was also platform-dependent, context-dependent, and nature-dependent [BG2004]. As a result, a lot of academic AI such as Genetic Algorithms may not be suitable for game development. Moreover, some academic AI (including Strong AI and weak AI) are nondeterministic so they are only suitable for the emergent conditions of the game scene or NPC behaviors.

To date, a lot of academic AI has been applied to game development. For example Creatures, Black & White, Battlecruiser 3000AD, Dirt Track Racing, Fields of Battle, and Heavy Gear used nondeterministic AI methods such as decision trees, neural networks, genetic algorithms, and probabilistic methods.

[SR2004] [BG2004] , [MB2005], , [SW1998] , [WIKI2], [WIKI3]

In the early days of computer games, AI was used to reduce predictable behavior. In games such as Pong, Pac-Man, and Donkey, the game designer used very simple rules and scripted sequence action with random decision–making [SR2002]. Nowadays, computer games use a variety of forms of AI. [BG2004]. Game AI is used in a range of games as simple as Tetris or Bejeweled or as complex as massively multiplayer online games. Game AI is used for many different purposes. It can help to make game play more random [RR2005], making it more difficult for players to exploit the patterns of games and making games more challenging. Game AI provides has been used to provide a series of challenges or goals that must be reached in the course of playing a game [MF2003], [BG2004], [BB2004], [BM2005] or in animation control [SR2002], Some game AI focuses on the behavior of non-player characters (NPC), providing players with instant feedback about non-player characters (NPCs) [BG2004], [BB2004], [BM2005], controlling their movement in role-play games (RPG) [MF2003], [BG2004], [BB2004], [BM2005]. Some game AI is used to assist the player [MF2003], [BG2004], [BB2004], [BM2005]. The game can’t control the action of the player but game AI can create situations and mechanisms that will favor the creation of unexpected victories. Game AI can also provide challenges based on the player’s past experience which can be used to train the player for more difficult challenges and goals [TR21999], [BE2005]. These challenges will be based either on the nature of the characters or the skill level of the player. For example, in most online games, such as Ultima Online [UO], players can create their own characters and occupations in the game world and the game AI provides a list of related challenges through which players can improve their skills. Puzzle-type games train players to be more skillful and archive equipment on every level. The difficulty of the level increases when the player plays. The player improves and can then play on the next level.

The ultimate goal of game AI is to make the game experience fun but there are a number of barriers to its wider use and greater innovation in the area. First, the development of game AI is on the end of the game development cycle and is not a main “selling point” in the game market. The attractions of perfect graphic design and sound effects make a more immediate selling point than the fact that opponent characters can reason cleverly [AN2004]. Second, game AI is expensive to develop [AN2004] [SW1999] in terms of both time and money and there is no guarantee of a proportionate return. Finally, AI makes demands of the CPU and main memory on the hardware that restricts its greater use but this factor is becoming less decisive nowadays as more powerful platforms become more common.

Resource issues related to development, tuning, and testing have been one of the main restriction on the use of AI approaches from academia, such as genetic algorithms or neural networks, expert systems, case-based reasoning, finite-state machines, decision trees, flocking, genetic algorithms, fuzzy logic and belief networks [SR2004]. If applied, all of these can make the reactions of NPCs more sophisticated and human-like and scene loading and effects more smooth and attractive. Strategy games are the main area of the academic AI research with the greatest challenges being in the area of AI opponents. Real-Time Strategy (RTS) AI focuses on real-time performance requirements. [RR2005] [SR2002] [AN2004] Ultimately, then, a successful AI technique in game AI is one that is simple to implement and overcomes the constraints of hardware [SR2002].

2.3.2 Classic AI Techniques

The following are some classic AI techniques which are applied in games.

1. Finite State Machines

Finite state machines (also simply called state machines) are often used in game AI for game development [JM2006]. A finite state machine is a control system which consists of a finite number of states and changes its status based on the past behavior or a transition of the Behavior under a particular condition. A tragic signal is a model of a finite state machine.

Pac-Man uses a finite state machine for the movement of its ghosts. [BG2004] In each state, the ghosts' behavior is different and their transitions are determined by the play's action. For example, if the player eats a power pill, the ghosts' states might change from chasing to evading and they return to chasing after the player's power returns to normal.

2. Influence Mapping

Influence Mapping, a common terrain analysis tool [DCP2000], uses a 2D array to represent the area of a terrain. An influence map can operate in almost any type of game world, whether it is represented using a square grid, a hexagonal grid, or a fully 3D environment. Every cell of the grid consists of a value, which is also called a "weighting". A cell has a higher value implies a character should move towards this cell. Influence maps can help NPCs plan their movements by analyzing the value of every cell at every movement of the NPCs. NPCs also identify the area the player controls and plan a starting position to threaten a player-controlled enemy.[SR2002]. Sweetser [SR2004] used Neural Networks to improve the analysis of the weightings of the cells of influence maps.

3. Flocking

Flocking is used to control a group of flocking behaviors of the non-player creatures such as sheep and fish. Bourg [BG2004] describes many examples of the use of the flocking algorithm to control the movements of armies in real-time strategy games. Flocking is also applied to enemies or squads in first-person shooter games and can simulate the behavior of crowds of people, for example, in a town square.

The simple steering behaviors of flocking are separation, alignment, cohesion, and avoidance. [BG2004] The behavior use Separation to avoid hitting its neighbors; use alignment to align itself to the average heading of its neighbors; use cohesion to make a unit steer object toward the average position of its neighbors. Avoidance [SR2002] allows groups of a unit to steer around obstacles and moves so as to avoid enemies.

Formation and Swarming are similar to flocking. Formation is a technique for group movement that mimics military formations. It is similar to flocking but each agent is guided toward a specific goal location and heading that is based on the position of the formation.[SR2004]

Swarming is similar to flocking. Flocking is suitable for small and medium numbers of agents in a game. If the agent of the NPCs is calculated individually, Swarming is used on every NPCs' movement. Swarming is a more computationally efficient method of moving a large numbers of agents.[SR2002]

4. Dead Reckoning

Dead reckoning is used on the network computer game or online multiplayer games. This is a process, which is used to reduce the lag, which is due to network latency and bandwidth problems.

There is a process for predicting the future state of the entity based on the current states and this reduces delays in the action. The current states are estimated using one or more entity such as direction, speed, time or some other mechanical measurement. [LL2002]. Dead Reckoning is used in sports games to reflect the computer opponent’s action in few seconds.[SR2004]

5. Command Hierarchy

Command Hierarchy is a team-based AI. Team-base AI is common used in strategic game and first- and third- person action games. Team AI is used to control a large number of non-player characters (NPCs).

Normally, team AI is used to control a large number of computer-controlled non-player characters. Command Hierarchy is a mechanism for controlling a large number of the agents or non-player characters which are in a levels of complexity. [SR2004]

Reynolds [SR2002] presents a good example of how decisions are made at each level of the hierarchy of levels. Every level has its own responsibilities and lower levels are subordinate to higher levels. An effective communication is provided between up and down the hierarchy so that the information is passed in a sufficient way.

To implement the command hierarchy, the game developer needs to consider the strategy of the agents, a effective message passing algorithm and allow data sharing between agents which are in difference levels and the controlled AI of individual agents for every level.

6. A* Pathfinding

Pathfinding is a searching algorithm for finding paths between two points. NPCs use pathfinding mechanism to move through the game world. A* (pronounced A-Star) pathfinding mechanism can help a non-player character or monster to negotiate a path though the terrain to the player, and then move approaches to the player or leave alone from the player. It is efficiently even when the player is constantly moving throughout the building. A* pathfinding is used to find the shortest path in the game world, especially through mazes. [SR2002] A* algorithm is most common basic ingredient for computing a long-distance route of an NPC.

There is a limitation to using A* algorithm. The A* algorithm assumes that the game world terrain is unchanging and as a result it is not suitable for use on a dynamic terrain, for example, a terrain where monsters are moving in the map and influencing the movement of the player. For pathfinding on changeable terrains, D*, a variation of A* pathfinding, is a better choice.

7. Terrain builder

Terrain builder is used to build the game environment modeled in this work, which is quite typical both in that it is common to use mazes to build game environments and in that Terrain Builder is widely used in maze-building, both at different game levels and in stand-alone games. Mazes are a type of puzzle. They appear in RPG games, shooting games and most adventure games. Pullen [WD2006] introduced the description of the maze and the history of the maze. There are four types of maze: perfect, braid, unicursal, and partial braid. A perfect maze contains one and only one solution. Braid mazes, also called purely multiply connected mazes [WIKI18] [WD2006], contain one or more passages without dead ends and require a player to choose the correct path to a goal. A unicursal maze has just a single path. A labyrinth is a unicursal maze with just one long snake-like passage and spiral on the map. A partial braid maze is made up of a mixture of both loops and dead ends. Terrain Builder can construct mazes using any one of the following three algorithms.

Random Depth-first search

Depth-first search is a recursive function for creating perfect mazes [MW2002]. Assume that there is a rectangular maze; each square of the grid is a cell (Figure 9). The horizontal and vertical lines represent the walls. To start, all walls of the cells are up and we use a depth-first search to selectively knock down walls until the working maze is perfect.

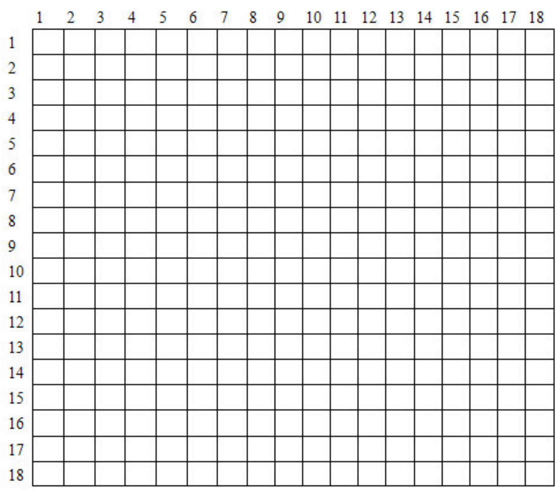


Figure 9 Grid

This is the simplest maze generation algorithm. It works like this:

1) Start at a cell randomly and mark as a current cell.
2) Select its neighbor that has been visited before.
3) If the neighbor cell is found, knock the wall between the current cell and its neighbors. Otherwise, return to the previous cell and mark as a current cell, repeat step 2
4) Mark the neighbor as current cell, repeat step 2 and 3 until all cell is visited.

Finally, all cells have been visited. All cells can be accessed when the player play. This algorithm prevent the creation of any open areas or the path this is lopped back to themselves.

Depth-First Searching is easy to implement and effective so that it is the most common algorithm on maze generation.

Prim's algorithm

Prim's algorithm is a minimal spanning tree algorithm that uses a greedy strategy [WIKI18][WD2006]. The algorithm is as follows[WIKI18]:

1) Select the start and end point of the maze.
2) Select a cell at random and become a current cell, then mark it as a part of the maze. Add the walls of the current cell to the wall list.
3) Draw a cell from the wall list. Break the wall at random of the current cell to make a passage.
4) Make the cell on the opposite side as a part of the maze. Add walls of its neighboring cell to the wall list
5) Move to the new cell become a current cell, and repeat 3 and 4 until there is at least one route between the start and end points.

Randomized Kruskal's algorithm

Kruskal's algorithm is a minimal spanning tree algorithm for a connected weighted graph [WIKI19][WD2006]. The algorithm as follows:

1) Create a list L. This lists all of the small paths between two cells on the grid. Each cell connects to its neighbor cell only once.
2) Select a path from L and connect to the main path by removing the wall. The addition of that path cannot be a circuit.
3) Repeat 2 until the at least one path is connected between the start and end points.

There are also some new maze generation algorithms such as the Aldous-Broder algorithm, the simplest algorithm, the non-cell-based algorithm. and the small-memory algorithm.

Maze and Terrain design is very commonly used in game level design to create game environments. For example, in the classic game, Doom, the maze is in the form of a tower, beginning in a basement. Every level is a pre-defined maze and players need to escape from the basement and go to the elevator. Some scenes in Mabinogi [MAB](Figure 10), a massively multiplayer online role player game (MMORPG), also use mazes.

Figure 10 Mabinogi

Chapter 3 Data Mining on Game Level Design

There is a lot of research on the data mining or statistical method on the graphic representation, animation and sound effect of the game development. However, there is not too much research on the using data mining on the game level design or game engine development. [DK2003] uses data mining on the evaluation the players' behavior. However, it is not about the game level design. Our proposed approach is used data mining to analysis the game. We introduce some concept of the data mining in this section. In section 3.1, we present some classical data mining algorithms. Section 3.2 presents the multivariate analysis which we used in the game level design

3.1 Data Mining Analysis

There are three main issues in the data mining process: Methodology, Performance, and Dataset [JH2001] and each of these has in turn its own specific concerns, which we outline in the following:

Mining methodology

1. The kind of knowledge mined
2. The ability to mine knowledge at multiple granularities

3. The use of domain knowledge
4. Knowledge visualization and presentation

Performance Issues

1. Efficiency and scalability of data mining algorithm
2. Parallel, distributed, and incremental mining algorithm

Issues relating to the diversity of data types

1. Handling of relational and complex types of data
2. Mining information from heterogeneous databases and global information

There are four main approaches for a data mining algorithm in the data mining process: Classification, Clustering, Association, and Prediction. Classification is a grouping process that is used to group similar data. The groups or the models are pre-defined. Usually, the models are used to describe the behavior or the concept of the data, for example, weather data may be related either to “Sunny” or “Raining”. Data will be grouped within these groups by comparing the similarity between the models and data. Classification is used for prediction. In many applications, classification is used to predict missing or unavailable data values.

Clustering analyzes data without consulting a known class label or model.

Clustering is used to identify a model and label it. The simple principle of clustering is applied to maximize the intra-class similarity and minimize the inter-class similarity. Some clusters of a data set are formed so that the data set within a cluster are highly similar within a cluster (model) and are highly dissimilar to other clusters.

Association rule mining finds interesting association rules and correlations between data sets. Association rules can help a business or organization make decisions. Association rule mining finds relationships between data set.

3.1.1 Association pattern mining

A typical example of association rule mining is market basket analysis [RA1994]. Customer-buying behavior is discovered using a set of association rules. This is very useful for businesses, which can use this knowledge of user behavior in product design, cross marketing and promotion. The basic form of association rule is

$$A \Rightarrow B \qquad \text{[support = x\% , confidence = y\%]}$$

An interesting rule is found and it has a support and confidence threshold. Suppose ***I*** is a set of items {i_1, ... i_m}, a database ***D*** that contains a set of transactions. Each transaction contains a set of items, which belong to ***I***. A support threshold presents the percentage of records in a data set that contains $\boldsymbol{A \cup B}$. A confidence threshold presents the percentage of records in a data set that contains ***A*** and also contains ***B***. A set of association rules is created which satisfies both a minimum support threshold and a minimum confidence threshold. An itemset refers to a set that contains ***k*** items which satisfy the minimum support. To find the association rules within a database, a two step procedure is applied as follows:

1. Find all frequent itemset in the database
2. Generate association rules from the frequent itemsets that satisfy the minimum support and minimum confidence.

The general idea is that if, say, ***ABCD*** and ***AB*** are frequent itemsets, we can determine if the rule ***AB CD*** holds by computing the ratio ***r = support(ABCD)/support(AB)***. The rule holds only if ***r*** >= minimum confidence. Note that the rule will have the minimum support because ***ABCD*** is frequent.

3.1.2 Apriori and Other Algorithms

AIS and SETM are early algorithms for finding frequent itemsets, which were used before Apriori was developed. AIS performs level-wise enumeration of sequences in $\boldsymbol{2}^{A}$, where A is the attributes of the sequence, in the most immediate way by terminal extension, [RA1994] the algorithm is simple: to find out all frequent k-itemset with $1 \leq k \leq N$ where N is the cardinal number. [RA1994]

SETM uses SQL to find frequent itemsets. In order to use the standard join operation for candidate generation, SETM separates a sequence generation from counting. However, it is not fundamentally different from AIS. [RA1994]

IBM's Quest project team developed an association rule mining algorithm, Apriori, for rule mining in large transaction databases [JA2001] [JH2001] [RA1994]. The algorithm is based on the Apriori property, which is an itemset property used for generating frequent itemsets. This property is based on the following observation. By definition, if an itemset ***I*** does not satisfy the minimum support threshold, then ***I*** is not frequent. If an item ***A*** is added to the itemset ***I,*** the resulting itemset (i.e., $\boldsymbol{I \cup A}$) cannot occur more frequently than ***I***. Therefore, $\boldsymbol{I \cup A}$ is not frequent. Apriori contains a joining step and a pruning step. The joining step generates candidate (***k+1***)-itemsets from frequent ***k***-itemsets. The pruning step eliminates candidate (***k+1***)-itemsets by counting the data record. The Apriori algorithm is as follows:

1) Find the set of frequent 1-itemsets. This set is denoted L_1.
2) Use L_1 to find L_2, the set of frequent 2-itemsets, which is used to find L_3, and so on, until no more frequent k-itemsets can be found. Apply join step and prune step to create $L_{K+1.}$

AprioriTid is an improvement on Apriori as it allows users to avoid having to scan the entire database after the first pass when counting the support threshold [JA2001]. It uses an addition set, counting_base, which is a set of pairs (***r, S1***), where ***r*** is the first element of the transaction (***r,s***) in ***D*** and ***S1*** is the set containing all attributes in sub-sequence ***s***. In each step, AprioriTid generates a new set ***C*** of candidates that is same as the Aprior does. Then the set counting_base, generated in the previous step, is used instead of the entire database ***D*** to count support. When counting support, AprioriTid generates the set counting_base associated with the current step. Although AprioirTid scans the sets counting_base that are expected to be smaller as they take into account all frequent itemsets discovered up to the previous set. However there is some disadvantage when using AprioirTid. AprioirTid is the complexity of its data structure and operation. And it has the same number of operations as Apriori.

Direct Hashing and Pruning (DHP) [JP1995] is derived from Apriori but makes use of an additional hash table that limits the generation of candidates in candidate set. It also progressively trims the database by discarding attributes in transactions or even by discarding entire transactions when they appear to be useless.

3.1.3 Sequential Pattern

A sequence is a function mapping from a set of integers, described as the index set, onto the real line or into a subset. A sequence containing k elements is called a ***k***-sequence. A time series database consists of sequences of values or events changing with time. The value should be measured at equal time intervals. A time series is a sequence whose index corresponds to consecutive dates separated by a unit time interval.

Sequential pattern mining is the mining of frequently occurring patterns related to time or other sequences. An example of a sequential pattern is ***"Someone who bought a Wii nine month ago is likely to order a NDS within three months". A*** sequence in a set of sequences is maximal if it is not contained in any other sequences. The goal of sequential pattern mining is to identify maximal sequences from among a set of frequent sequences.

Several parameters set and influence the results of sequential pattern mining. The first parameter of a time sequence is duration. The duration may be the entire available sequence in the database. Sequential pattern mining can be confined to the data within a specified duration. The second parameter is the event folding windows, ***w***. A set of events occurring within a specified period of time can be folded into the analysis. The third parameter is the interval. There are three different interval setting modes: interval equals 0 means no interval gap is allowed such as in an $a_{i-1}a_ia_{i+1}$ sequence. If an interval is within a specific period, the pattern is present between a minimal interval and a maximum interval. The last interval setting mode is an exact value such as 2 days, 1 hour. [LS1993']

3.1.4 Sequential / Temporal Pattern Mining Algorithms

Lin [WL2002] described temporal data mining, that is, data mining and the theory of statistical time series analysis, as involving a number of tasks and subtasks:

1. Temporal data mining tasks
    a. Temporal data characterization and comparison
    b. Temporal clustering analysis
    c. Temporal classification
    d. Temporal association mining
    e. Temporal pattern analysis
    f. Temporal prediction and trend analysis

2. The temporal data model may need to be developed based on
    a. Temporal data structures
    b. Temporal semantics

3. The temporal data mining concept must take account of the following concerns:
    a. the task of temporal data mining can be seen as a problem of extracting an interesting part of the logical theory of a model
    b. the theory of a model may be formulated in a logical formulism able to express into a quantitative knowledge and an approximate facts

Most data mining techniques, such as AprioriAll or AprioriSome, are Apriori-like algorithms because the Apriori property can be applied to mining sequential patterns [RA1995]. If a sequential pattern of length k is infrequent, its superset (of length ***k+1***) cannot be frequent.

AprioriAll

AprioriAll is a count-all algorithm that counts all of the frequent sequences, which are including non-maximal sequences. Those non-maximal sequences need to be pruned out. To avoid counting non-maximal sequences by counting longer sequences, the count-some algorithm is applied. [JA2001]

The temporal sequential pattern mining involves six-phase: sort phase, litemset phase, transformation phase, sequence phase, and maximal phase. [JA2001] describe as the follow:

In the sort phase, the original transaction database is converted into a database of sequences by sorting the database with the data-sequences-id. The data-sequences-id is the major key and transaction-time is the minor key. The frequent itemsets are found in the itemset phase. Association rule mining and sequential pattern mining is applied according to the different definitions of support. In sequential pattern mining, the support of an itemset is defined as the fraction of sequences in which the itemset is present. In the transformation phase, each transaction is expressed as a set of itemsets. If a sequence does not contain any itemsets, this sequence is deleted from the transformed database. However, it still contributes to the count of the total number of sequences. A transformed data-sequence is represented by a list of sets of itemsets. In the sequence phase, the frequent sequences are found. The general structure of this phase is to make multiple passes over the data. In each pass, it starts with a seed set of frequent sequences. The seed set is used to generate candidate sequences. The support for these candidate sequences is found during the pass over the data. At the end of the pass, the frequent candidates sequences is found. These frequent candidates become the seed set for the next pass.

AprioriAll and AprioriSome were proposed as ways to find frequent sequences. AprioriAll, is a count-all algorithm, generates the candidate sequences using the frequent sequences from the previous pass and then measures their support by making a pass over the database. At the end of each pass, the support of the candidates is used to find the large sequences. AprioriSome is a count-some algorithm, has a forward phase and a backward phase. In the forward phase, only the sequences, which are certain lengths, are counted. And the remaining sequences are counted in the backward phase. In the final phase, the maximal sequence among the frequent sequence is found.

AprioriSome can find only maximal sequential patterns. If some applications require all pattern and their support, AprioriAll is applied. There are some limitation on AprioriAll and AprioriSome, such as the absence of time constraints, rigid definitions of a transaction, and absence of taxonomies. [JA2001]

Frequent-pattern tree (FP-Tee)

Frequent-pattern tree (FP-Tree) is a tree structure for mining sequential patterns that is supporting for the Apriori algorithm. [JH2001] The Apriori algorithm incurs two costs: It generates a huge number of candidate sets and it has to repeatedly scan a database and check a large set of candidates by pattern matching. FP-tree combined with Apriori mines the complete set of frequent itemsets without generating candidates. [JH2001] FP-tree is a compressed database of frequent items within a frequent-pattern growth (FP-growth) method. After constructing an FP-tree, the itemset association information is retained by the FP-growth, and then a compressed database is divided into a set of conditional databases, each of databases are associated with one frequent item and mines each database separately.

However, there is a weakness on using FP-Tree. Jian Pei [JP2000] said, “However, the items (or subsequences) containing different orderings cannot be reordered or collapsed in sequential pattern mining. Thus, FP-tree structures so generated will be huge that cannot benefit mining”. [JP2000]

FreeSpan

FreeSpan (Frequent pattern projected Sequential pattern mining) uses frequent items project sequence databases into a set of smaller projected databases and grows subsequence in each project database. The process is recusively. [JP2000]

The major cost of FreeSpan solved in the projected databases. If a pattern finds in each sequence of a database, the size of its projected database does not decrease. Moreover, it is costly because if a length-k subsequence can grow at any position, the searching process for ***length- (k+1)*** candidate sequence will need to check every possible combination.[JP2000]

PrefixSpan

Prefix-projected Sequential Pattern mining (PrefixSpan) is used for sequential pattern mining. It examines only the prefix subsequences and projects only its corresponding postfix subsequences into projected databases [JP2000] Sequential patterns in each projected database are grown by exploring only local frequent patterns. The database projections are level-by-level projection and bi-level projection to increase the efficiency of the mining process.. Bi-level projection is used for large databases. [JP2000]

An apriori-like sequential pattern mining algorithm has two weaknesses: [JA2001]

- Large number of candidate sequences.

- Repeated database scans.

3.2 Multivariate Analysis of Games

In multivariate analysis, each observation consists of a vector and the p variables of the vector ***Y= (y1, y2, … yp)*** represent measurements of a single subject or object. Since the variables arise from the sample sampling unit, they are typically inter-correlated. However, in most cases they measure different things, for example, width, height and weight, or resting heart rate. The mainly target of the multivariate analysis techniques is to simplify the data.

Most of the multivariate techniques can be categorized as either descriptive or inferential. [JG1993] Descriptive procedures should characterize the correlation within the observation vectors or show how the variables contribute to the grouping patterns. They may attempt to separate overlapping information from correlated variables by constructing a small number of uncorrelated variables, which also can be linear combinations of the original variables that reveal the essential dimensionality of the system. [JG1993]

Multivariable inferential procedures include hypotheses testing. The testing allows any correlation structure among the variables. It also control the experiment error rates, the rating is independent to the number of variables to be tested. Many multivariable inferential techniques are extensions of univariate procedures such as t-tests or F-tests. [JG1993]

Most procedures analyze only for the continuous random variables. However, multivariate techniques also produce good results when applied to discrete ordinal data. Log linear models, generalized linear models, or correspondence analysis can be applied on the categorical or discrete data.

Multivariate data is a set of observations, which contains several different variables pertaining to a number of set or individuals. Examination results, nutritional studies, medical data, and shopping behavior could all serve as multivariate data [JH2001].

A large set of multivariate data is represented by a set of formulae with few parameters or a matrix (or data matrix). The representation as follow: [JG1993]

Suppose that there are number of variables by p, and the number of individuals by n.

$$X = \begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1p} \\ x_{21} & & \cdots & x_{2p} \\ \vdots & & & \vdots \\ x_{p1} & x_{p2} & \cdots & x_{np} \end{bmatrix}$$

A data matrix can be seen as n row vectors, which denote by x_1^T to x_n^T, or as p column vectors y_1 to y_p.

Multivariate analysis techniques are either variable-directed or individual-directed. [JG1993] Variable-directed analysis is concerned with the relationship between variables. When comparing variables, we only consider column vectors of a data matrix. Matrix algebra is used to find the correlation coefficient. It is a cosine of the angle between the two n-dimensional column vectors. If the two variables are close together in n-dimensional space, then the angle between the two vectors will be small and the cosine will be close to +1 to indicate that two individuals have a positive correlation. Individual-directed analysis is concerned with the relationships between individuals or objects. Row vectors in p-dimensional space are used for analysis. The distance between two row vectors implies the similarity of two individual or objects. If two individual are similar, the distance would be small.

Multivariate techniques are used in medicine, physical and biological sciences, economics and social science, and in many industrial and commercial applications. Principal component analysis is used for exploring data to reduce dimensions. Generally, PCA seeks to represent n correlated random variables by a reduced set of uncorrelated variables, which are obtained by transformation of the original data set into an appropriate subspace. Principal component analysis and factor analysis are closely related techniques. They are used to reduce the dimensionality of multivariate data. The correlations and interactions between the variables are summarized among the variables in terms of a small number of underlying factors.

Factor analysis also takes into account the variation in a number of original variables using a smaller number of index variables or factors. Each original variable is represented a linear combination of these factors plus a residual term. The residual term reflect the extent to which the variable is independent of the other variables. For example there is a set of data with five variables as x_1, x_2, x_3, x_4 and x_5. [CA1980]A two factor model assumes that

$$\begin{aligned}
X_1 &= a_{11}F_1 + a_{12}F_2 + e_1 \\
X_2 &= a_{21}F_1 + a_{22}F_2 + e_2 \\
X_3 &= a_{31}F_1 + a_{32}F_2 + e_3 \\
X_4 &= a_{41}F_1 + a_{42}F_2 + e_4 \\
X_5 &= a_{51}F_1 + a_{52}F_2 + e_5
\end{aligned}$$

Where the a_{ij} values are constants, F1 and F2 are the factors and e_i represents the variation in Xi that is independent of the variation in the other X-variables. The aim of the principal component analysis is similar to factor analysis. PCA produces an orthogonal transformation of the variables and is focus on the explaining the variance. Factor analysis is based on a proper statistical model and is more concerned with explaining the covariance structure of the variables. The idea of the factor analysis is to derive new variables called factors, which provides a better representation of the data.

Most applications of factor analysis have been in psychology and the social sciences [CA1980]. Suppose that information is collected from a wide range of people as to their occupation, type of education, whether or not they own their own home, and so on. A single index of class of the data to represent that multidimensional data is constructed by factor analysis, which use a single underlying factor.

Canonical analysis is used to partition variable into two sets. For example, a first set contains variables that refer to behavior characteristics of neurological patients such as problems in vision, speech, or movement and the other set contains physiological variables derived from an EEG. IN another example, the first set contains answer of respondents to a number of questions and the second set contains their background data. The question is whether the results in the first set are related to results in the second set.

A formal approach is introduced by [JG1993]. Let the m1 variable in the first set as

$$z_{11}, z_{21}, \ldots, z_{m_1 1}$$

and the m2 variables in the second set as

$$z_{12}, z_{22}, \ldots, z_{m_2 2}$$

Canonical analysis looks for a solution of weights so that one can define weighted sum variables

$$\begin{aligned} x_{11} &= (z_{11}w_{111} + z_{21}w_{211} + \ldots + z_{m_1 1}w_{m_1 11}) / m_1 \\ x_{21} &= (z_{12}w_{121} + z_{21}w_{221} + \ldots + z_{m_2 1}w_{m_2 21}) / m_2 \end{aligned}$$

with numbers, ***w_{jls},*** as weights. The first index ***j*** refers to the number of the variable within its own data set, the second to the number of the data set (***l***=1,2), and the third index ***s*** gives the number of the dimension of the solution (***s***=1, so far).

The first dimension of the solution is weighting w_{jl1} when the correlation between ***x_{11}*** and ***x_{21}*** is maximized. This correlation is also called the canonical correlation. The weighted sum vectors ***x_{11}*** and ***x_{21}*** are the first pair of canonical variates. The weight ***w_{jl1}*** are called canonical weights. The correlation between ***x_{11}*** (or ***x_{21}***) and the individual variable ***z_{jl}*** is called the canonical loadings.

A second solution is choosing ***w_{jl2}*** that the second canonical correlation (between ***x_{12}*** and ***x_{22}***) is maximized, under the condition that canonical variates ***x_{12}*** and ***x_{22}*** must be uncorrelated with the canonical variates ***x_{11}*** and ***x_{12}*** of the first dimension. Canonical variates of any dimension must be uncorrelated with the canonical variates of all other dimensions.

Other multivariate techniques include Multidimensional Scaling, Cluster Analysis, and Correspondence Analysis.

Many univariate and multivariate methods assume that the measured variables are all continuous. The distribution of a continuous random variable may be described by the cumulative distribution function (c.d.f) or by its derivative called the probability density function (p.d.f) For continuous multivariate distributions, a suitable multivariate analogues need to be defined for the variables for these functions. Categorical data is a type of discrete data. For categorical data analysis, [JG1993] defines three types of variable on multivariate analysis: Nominal, Ordinal and Numerical.

Nominal variables are label or name to distinguish one category from another. Labels can be numbers, letters or names. If the label is numerical (1, 2, 3 …), there are no differences or priori numerical values for those labels.

A priori order applies in Ordinal variables. For example, a “Height” variable may consist of “Tall”, “Medium” or “Short”. Each label implies an order of height. However, the actual measurements cannot be reflect on the labels or the between the variables.

Numerical labels reflect the a priori order and their difference. For example, A group of people answers the questionnaire. In the questionnaire, age is collected and round off, in age classes of 5 years, with multiples of 5 as their midpoints. Age is a numerical label. A person in category 20 must be older than 17.5 years. The difference between 20 to 25 is same as the difference between 50 and 55 and also are 5 years.

The process of the distinction of the variables is very important. The distinctions depend on a decision made by the data analyst and the researcher.

Chapter 4 Game AI with Prediction

A game engine which is used to build game environments will usually use traditional AI techniques [SW1999], such as random number generation, pre-defined rules, or templates. This is because game level design is a component in the development cycle of a game. Many game designers and programmers focus on the movement of NPCs (including action, instance response and movement), user interface design (design of the characters, rendering of the graphic and layout of the world), game control and sound effects. This is very costly. To reduce this cost, many game level designers use weak AI and some game companies release a beta version of the game to the public or target players to evaluate the game levels. Game companies then can adjust the difficulty of the game level before official release. [SW1999]

For example, almost all maze game AI generate a random number as a parameter and build the mazes on every game level. This process is repeated until the players pass all levels and reach the final goal (Figure 11). To increase the playability of a game, when generating a new level, some game level designers add more criteria to the maze generator to ensure the difficulty of the level increases.

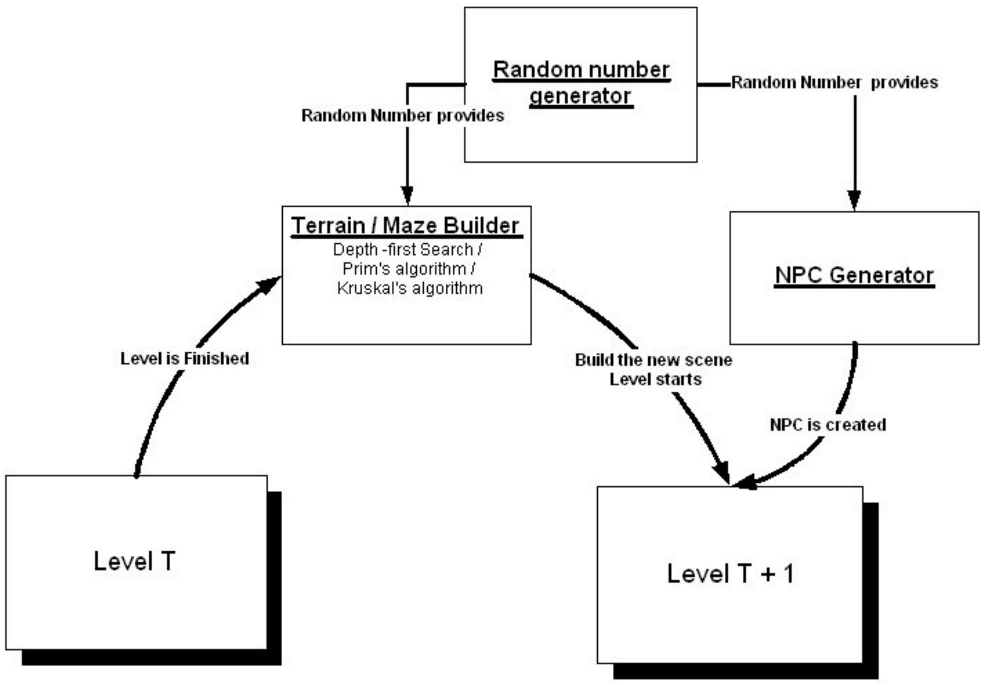


Figure 11 Game Flow of the Level design using random number generation

Some game level designers use pre-defined templates of the game environment. The floor plan of the maze, number of NPCs and their behaviors are pre-defined for every level. After players finish Level T, the game engine immediately loads the new level T+1 (Figure 12).

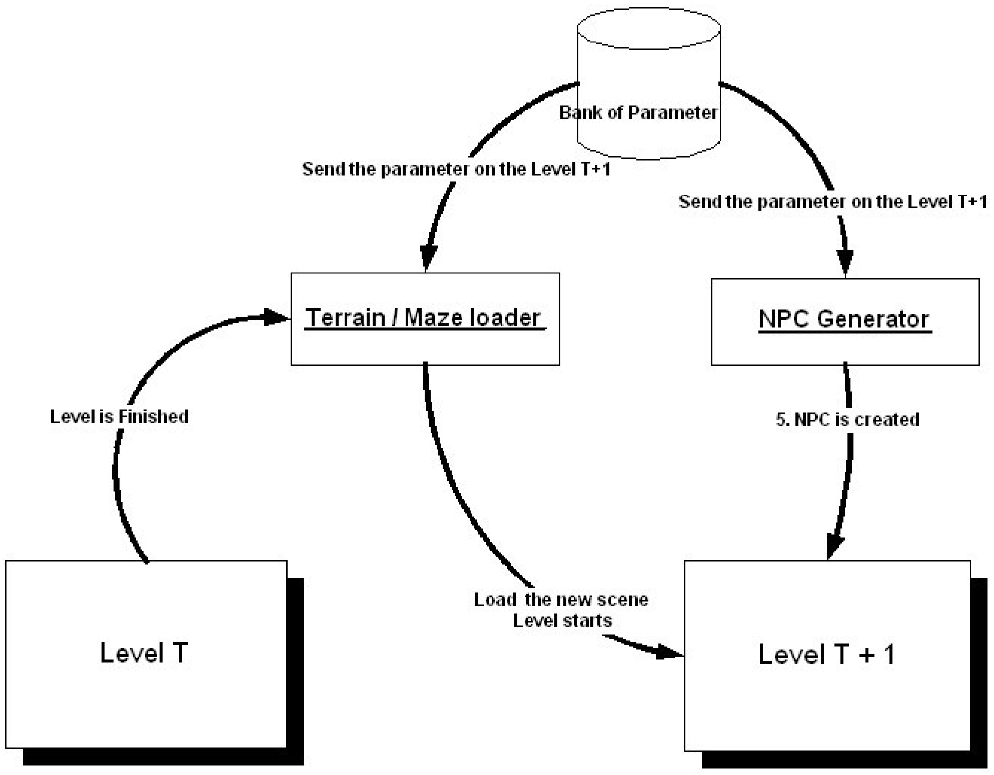


Figure 12 Game Flow of the Level design using pre-defined template

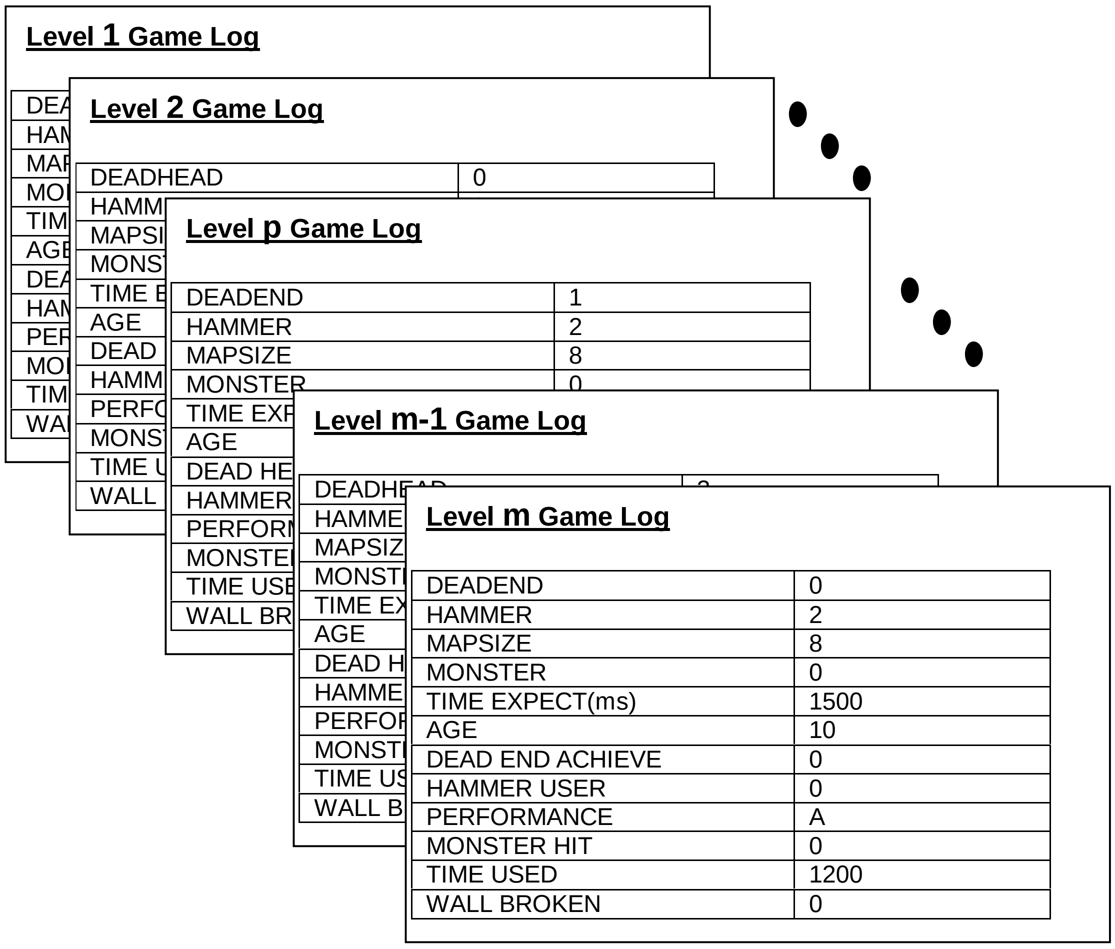
The random generation and pre-defined template approaches are inflexible. To deal with this, an adaptive game engine is applied. This adjustment of the difficulty is also called dynamic game balancing and is discussed in section 2.2.2.

By analyzing the individual player's gameplay data, we can identify characteristic patterns and predict a player's future behavior, adjusting the game level so that the game is more interesting and adapts to the individual player. As an illustration of the prediction tasks, we have given an example of the ongoing process in a maze game. We will now propose a multivariate pattern mining method for analyzing a player's game log. This method also builds a set of prediction rules to predict a player's behavior and then adjusts the difficulty of the level.

In Section 4.1, we present the details of our proposed approach to predicting player behaviors and then adjusting the difficult of the game level. In Section 4.2, we use a decision tree approach to adjust the difficult of the game level for the player.

4.1 Sequential Pattern Mining Approach

Assume the game log is a set of sequential patterns (Figure 13). It can be defined as follows: suppose that there is an ordered sequence $\boldsymbol{S}$ of $\boldsymbol{M}$ BEHAVIORS, $\boldsymbol{Behavior_1}$, …,$\boldsymbol{Behavior_p}$,…,$\boldsymbol{Behavior_m}$, where $\boldsymbol{Behavior_p}$ is located at position $\boldsymbol{p}$ in $\boldsymbol{S}$. Suppose also that each Behavior in the sequence is described by $\boldsymbol{n}$ distinct attributes, $\boldsymbol{Attr_{1p}}$, …, $\boldsymbol{Attr_{jp}}$, …, $\boldsymbol{Attr_{np}}$, and that in any instantiation of the Behavior description, an $\boldsymbol{Attr_{pj}}$ takes on a specific value, $\boldsymbol{val_{jp}} \in$ domain($\boldsymbol{Attr_{jp}}$)={$\boldsymbol{v_{jk}}$| $\boldsymbol{k}$=1,…,$\boldsymbol{J}$}, which may be numerical or symbolic or both.



DEADEND	0
HAMMER	2
MAPSIZE	8
MONSTER	0
TIME EXPECT(ms)	1500
AGE	10
DEAD END ACHIEVE	0
HAMMER USER	0
PERFORMANCE	A
MONSTER HIT	0
TIME USED	1200
WALL BROKEN	0

Figure 13 Game logs data of ***m*** levels

It is important to know the relationship between attributes as it can help to increase the accuracy of the prediction. If the ***i***th attribute of an Behavior that takes on $\boldsymbol{v_{i1}}$, is always preceded at τ level (positions or time units) earlier by an Behavior whose ***j***th attributes takes on the value $\boldsymbol{v_{jk}}$, we can conclude that the $\boldsymbol{v_{j1}}$ is dependent on $\boldsymbol{v_{jk}}$, with a level lag of τ.

To decide if the ***i***th attribute of an Behavior in a sequence is dependent on the

jth attribute ***$Attr_{jp}$*** of the Behavior at τ level earlier, the chi-square test can be employed. The chi-square test uses a two-dimensional contingency table of ***I*** rows and ***J*** columns (I and ***J*** respectively being the total number of values taken on by the ***i***th and the ***j***th attributes).

		Attrjp						
		v_{j1}	v_{j2}	…	v_{jk}	…	v_{jJ}	Totals
	v_{i1}	o_{11} (e_{11})	o_{12} (e_{12})	… …	o_{1k} (e_{1k})	… …	o_{1J} (e_{1J})	o_{1+}
	v_{i2}	o_{21} (e_{21})	o_{22} (e_{22})	… …	o_{2k} (e_{2k})	… …	o_{2J} (e_{2J})	o_{2+}
	⋮	⋮	⋮		⋮		⋮	⋮
Attri(p+τ)	v_{il}	o_{l1} (e_{l1})	o_{l2} (e_{l2})	… …	o_{lk} (e_{lk})	… …	o_{lJ} (e_{lJ})	o_{l+}
	⋮	⋮	⋮		⋮		⋮	⋮
	v_{iI}	o_{I1} (e_{I1})	o_{I2} (e_{I2})	… …	o_{Ik} (e_{Ik})	… …	o_{IJ} (e_{IJ})	o_{I+}
Totals		O_{+1}	o_{+2}		o_{+k}		o_{+J}	M'

Table 1 A Two Dimensions Contingency Table with ***I*** Rows and ***J*** Columns

In Table 1,Let $\boldsymbol{o_{jk}}$ be the total number of BEHAVIORS in ***S*** whose ***i***th attribute $\boldsymbol{Attr_{i(p+\tau)}}$, take on the value $\boldsymbol{v_{il}}$ and are preceded at τ positions earlier by BEHAVIORS that have the characteristic $\boldsymbol{v_{jk}}$. Let $\boldsymbol{e_{lk}}$ be the expected number of such BEHAVIORS, under the assumption that $\boldsymbol{Attr_{i(p+\tau)}}$ and $\boldsymbol{Attr_{jp}}$ are independent. $e_{lk} = \sum_{u=1}^{J} o_{lu} \sum_{u=1}^{I} o_{uk} / M'$ where $M' = \sum_{l,k} o_{lk}$ is less than or equal to M (the total number of BEHAVIORS in the sequence S) due to the possibility of there being missing values in the data.

The chi-square statistic can be defined as

$$X^2 = \sum_{l=1}^{I} \sum_{k=1}^{J} \frac{(o_{lk} - e_{lk})^2}{e_{lk}} = \sum_{l=1}^{I} \sum_{k=1}^{J} \frac{o_{lk}{}^2}{e_{lk}} - M'.$$

The difference between the observed and expected value could have arisen by chance. It can be determined by comparing chi-square statistic $\boldsymbol{X^2}$ with the critical chi-square $\boldsymbol{X^2_{d,\alpha}}$, where ***d***=(***I***-1)(***J***-1) is the degree of freedom and α, usually taken to be 0.05 or 0.01, is the significance level. The confidence level is (1-α)%. If $\boldsymbol{X^2}$ is greater than the critical value, there is enough evidence to conclude that $\boldsymbol{Attr_{i(p+\tau)}}$ is dependent on $\boldsymbol{Attr_{jp}}$. One cannot conclude this if $\boldsymbol{X^2}$ is less than $\boldsymbol{X^2_{d,\alpha}}$.

Finally, we can use multivariate analysis to find a set of prediction rules that describe ***S*** and can be employed to predict the characteristics of a future

Behavior. For example, if a value of an attribute $\boldsymbol{Attr_{jp\,l}}$ is equal v_{jk} , then the value of the attribute $\boldsymbol{Attr_{i(p+\tau)}}$ is equal v_{il} after τ positions.

The following figure (Figure 14) summarizes the steps in Chi-square testing:

```
1)  Chi-square Testing:
2)  Gamelog =Sequence of the log with n Attribute, and one of these attributes represents
    the category of the record, which may be numerical or symbolic.
3)  p= Maximum future level ( p=3 if we calculate the relation between the attribute on
    the current $\tau$ level and ( $\tau$ +p) level.
4)  for (level = 1 to current level –p)
5)      For (attr = 0 to n)
6)          For (pattr = 0 to n)
7)              For (plevel 0 to p)
8)                  Building a contingency table between the $Attribute_{attrl}$ and
                    $Attribute_{(pattrl- plevel)}$ where pattr≠ attr with the observed number
9)                  Building a contingency table between the $Attribute_{attr}$ and
                    $Attribute_{(pattrl- plevel)}$where pattr≠ attr with the expected number
10)                 Calculate critical chi-square $X_2$
11)                 Compare $X_2$ with Critical Values for the Chi-Square
                    Distribution with degree of freedom df
12)                 Identify the dependent of the $Attribute_{attr}$ and $Attribute_{(pattrl-plevel)}$where pattr≠ attr
13)             Loop plevel
14)         Loop pattr
15)     Loop attr
16) Loop level
```

Figure 14 Algorithm for Chi-square Testing:

If the results of chi-square testing are significant, we can conclude that an attribute $\boldsymbol{Attr_{i(p+\tau)}}$ is dependent on another attribute, $\boldsymbol{Attr_{jp}}$. However, it cannot provide information as to how the observed valued of the ***i***th attribute in a sequence is dependent on that of the ***j***th attribute of an Behavior at τ positions earlier

To predict the player's future behavior, we can construct a set of prediction rules. They represent each detected dependence relation between two attributes values by using a rule in the following form:

If<*Condition*> then <*conclusion*> with certainty *W.*

The condition part of the rule shows the characteristic that a Behavior should possess so that the Behavior at a certain position later in the sequence will take on the attribute value predicted in the conclusion. As such a prediction cannot usually be constructed with complete certainty, the degree of certainty has to be reflected by the weight ***W*** associated with the rule.

Suppose the attribute value v_{il}, is found and it is dependent on v_{jk} as described in the previous section, the prediction rule is constructed as follows and shows the following relationship:

> If $Attr_{jp}$ of an Behavior is v_{jk}, then it is with certainty W that $Attr_{i(p+\tau)}$ of an Behavior located at τ positions later in the sequence has the value v_{il},

where $W = W(Attr_{i(p+\tau)}=v_{il} \;/\; Attr_{i(p+\tau)} \neq v_{il} \mid Attr_{jp}=v_{jk})$ measures the amount of positive or negative evidence provided by v_{jk} supporting or refuting the

Behavior at τ positions later to have the characteristic, ***v_{il}*.**

The derivation of W is based on an information theoretic measure known as mutual information and defined between vjk and vik as

$$I(Attr_{i(p+\tau)} = v_{il} : Attr_{jp} = v_{jk})$$
$$= log \frac{Pr(Attr_{i(p+\tau)} = v_{ij} \mid Attr_{jp} = v_{jk})}{Pr(Attr_{i(p+\tau)} = v_{il})}$$

$I(Attr_{i(p+\tau)} = v_{il} : Attr_{jp} = v_{jk})$ is positive if and only if $Pr(Attr_{i(p+\tau)} = v_{ij} \mid Attr_{jp} = v_{jk}) > Pr(Attr_{i(p+\tau)} = v_{il})$. Otherwise it is either negative or has a value of 0.

As v_{il} of $Attr_{i(p+\tau)}$ is dependent on v_{jk} of $Attr_{jp}$, the weight of evidence can be defined as in [DO1974]:

$$W(Attr_{i(p+\tau)} = v_{il} / Attr_{i(p+\tau)} \neq v_{il} \mid Attr_{j[} = v_{jk})$$
$$= I(Attr_{i(p+\tau)} = v_{il} : Attr_{j[} = v_{jk}) - \quad .$$
$$I(Attr_{i(p+\tau)} \neq v_{il} : Attr_{j[} = v_{jk})$$

W can also be expressed as

$$
\begin{aligned}
& W(Attr_{i(p+\tau)} = v_{il} / Attr_{i(p+\tau)} \neq v_{il} \mid Attr_{j[} = v_{jk}) \\
& = I(Attr_{i(p+\tau)} = v_{il} : Attr_{j[} = v_{jk}) - \\
& I(Attr_{i(p+\tau)} \neq v_{il} : Attr_{j[} = v_{jk}) \\
& = log \frac{Pr(Attr_{i(p+\tau)} = v_{il} \mid Attr_{jp} = v_{jk})}{Pr(Attr_{i(p+\tau)} = v_{il})} \\
& - log \frac{Pr(Attr_{i(p+\tau)} \neq v_{il} \mid Attr_{jp} = v_{jk})}{Pr(Attr_{i(p+\tau)} \neq v_{il})} \qquad . \\
& = log \frac{Pr(Attr_{jp} = v_{jk} \mid Attr_{i(p+\tau)} = v_{il})}{Pr(Attrjp = v_{jk})} \\
& - log \frac{Pr(Attrjp \neq v_{jk} \mid Attr_{i(p+\tau)} \neq v_{il})}{Pr(Attr_{ijp} = v_{jk})} \\
& = log \frac{Pr(Attr_{jp} - v_{jk} \mid Attr_{i(p+\tau)} = v_{il})}{Pr(Attr_{jp} = v_{jk} \mid Attr_{i(p+\tau)} \neq v_{il})}
\end{aligned}
$$

The weight of evidence is a measure of the difference in the gain in information when the ***i***th attribute of a Behavior takes on the value v_{il} and when it takes on other values, given that the Behavior that is τ positions in front has the characteristic v_{jk}.

If vj_k provides positive evidence supporting the $_i$th attribute of the behavior at τ positions later in the sequence having the value v_{il}, the weight ***W*** is its position. ***W*** is negative if the evidence of V_{jk} is negative

To illustrate how such predictions can be made, suppose that given a sequence ***S*** of ***M*** Behavior, ***Behavior₁***,… , ***Behavior$_M$***, and the value of the ***i***th attribute ***Attr$_{i(M+h)}$*** of the behavior ***BehaviorM+h***, which is ***h*** positions behind, and that the most recently observed one is ***S***, ***Behavior$_M$***, is predicted. The relationship between attributes within the behavior should be determined to discover whether the value of ***Attr$_{i(M+h)}$*** is dependent on the BEHAVIORS in ***S***.

Assume that a set of BEHAVIORS is generated probabilistically in such a way that the characteristics of the Behavior at a certain position depend on that of a maximum of ***L*** BEHAVIORS before it. The prediction process starts by searching though the prediction rules that determine how the characteristics of ***Behavior***$_{M}$, ***Behavior***$_{M-1}$, ***Behavior***$_{(M-L)+1}$ may affect the value of ***Attr***$_{i(p+\tau)}$ of ***Behavior***$_{M+h}$.

To find the prediction rules, we use the following search process. First, we match the attributes values val_{jp} (where ***j=1,2,…,n and p=M, M-1,(M-L)+1***) of the BEHAVIORS Behavior$_{M}$, Behavior$_{M-1}$, ***Behavior***$_{(M-L)+1}$***,*** against the subset of prediction rules whose conclusions predict what values the ***i***th attribute of an Behavior at h t ***h***, ***h***+1, …, (***h+L)***-1 position later will take on. An attribute value that satisfies the condition part of a rule, affects the value of the ith attribute of the Behavior at ***M***+***h***. As a result, this value provides a certain amount of evidence on the conclusion parts, which either supports or opposes the ***i***th attributes taking on the value. This value is reflected by the weight of the prediction rule.

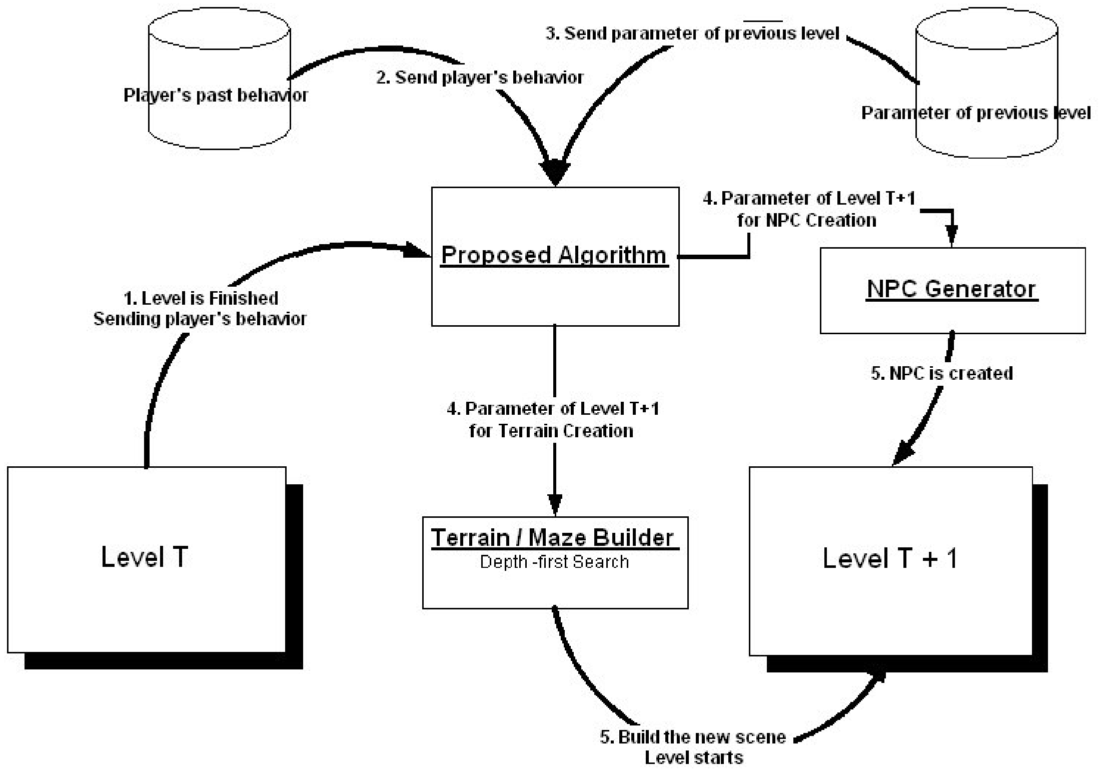
Figure 15 summarizes the steps in building a prediction rule with weight ***W***:

1)	**Prediction Rule:**
2)	Gamelog =Sequence of the log with n Attribute, and one of these attributes represents the *category* of the record, which may be numerical or symbolic.
3)	p= Maximum future level (p=3 if we calculate the relation between the attributes on

```
    current $\tau$ level to ( $\tau$ +p )level.
4)  for (level = 1 to current level –p)
5)      For (attr = 0 to n)
6)          For (pattr = 0 to n)
7)              For (plevel 0 to p)
8)                  Building a contingency table between the $\text{Attribute}_{\text{attrl}}$ and
                    $\text{Attribute}_{(\text{pattrl- plevel})}$ where pattr≠ attr with the observed number
                    Calculate certainty weight $W$
                    Build the prediction rule with the observed value of the
                    $\text{Attribute}_{\text{attrl}}$ and observed valued of $\text{Attribute}_{(\text{pattrl- plevel})}$ with
                    the certainty W where pattr≠ attr and W ≠ 0
9)              Loop plevel
10)         Loop pattr
11)     Loop attr
12) Loop level
```

Figure 15 Prediction Rule Generation Algorithm

Figure 16 shows the game flow for implementing the proposed algorithm. When player finishes Level ***T,*** the game engine collects the Level ***T*** game log and retrieves all past behavior of the player from the database. The parameters of the past levels are also retrieved. Using the Prediction Rule Generation algorithm (Figure 15), the game engine will follow the weight of the rule and the previous (T-τ**+**1) level to predict (τ +1) level. The number of NPCs on the game level is also a parameter of the game environment. Our proposed algorithm can also predict the number of NPCs that should appear at a level. The Maze builder will then create a new maze for the level using depth-first searching and the parameters of (τ +1) level.



Player's past behavior
2. Send player's behavior
3. Send parameter of previous level
Parameter of previous level
Proposed Algorithm
4. Parameter of Level T+1 for NPC Creation
NPC Generator
1. Level is Finished Sending player's behavior
4. Parameter of Level T+1 for Terrain Creation
5. NPC is created
Level T
Terrain / Maze Builder
Depth -first Search
Level T + 1
5. Build the new scene Level starts

Figure 16 Game Flow of the Maze game using sequential pattern mining

We use the maze game as a testing model for the approach. There is an ordered sequence ***S*** of M BEHAVIORS.(***$Behavior_1$***… ,***$Behavior_p$***,…***$Behavior_M$***, where $Behavior_p$ is located at time=***p*** in ***S***). The details of the game logs are in Table 4. For better presentation, we have divided Table 4 into two tables: the parameters of the level (Table 5) and player's behavior (Table 6). The attributes of the parameters of the level are as follows:

LEVEL NUMBER, DEADEND, HAMMER, MAPSIZE, MONSTER and TIME EXPECT TO FINISH

The description of the parameters of the level is as follows:

Name of Parameter	**Description**

LEVEL NUMBER,	Level number
DEADEND	Number of Dead End for the maze
HAMMER	Number of hammer is provides for the player
MAPSIZE	There are three sizes of the map,10,12 and 15. The is defined as number of girds (Column X Row) in the map
MONSTER	Number of monster on the maze
TIME EXPECT TO FINISH	The expected timer that the player should be finished in milliseconds

Table 2 Description of the Parameter of a level

The attributes of the player’s behaviors are as follows:

LEVEL NUMBER, DEAD END ACHIEVE, HAMMER USED, LEVEL PREFORMANCE, MONSTER HIT, BROKEN WALL and TIME USED

The parameters of the game log containing the player’s behavior is as follows:

Name of Parameter	**Description**
LEVEL NUMBER,	Level number
DEAD END ACHIEVE	Number of Dead End that the player met of a level
HAMMER USED	Number of hammer used to hit the monster or break the wall of a level
LEVEL PREFORMANCE	A- Same as expected time ±0.1second B-Time used is less than Time Expected

	W- Time used is more than Time Expected
MONSTER HIT,	Number of monster hit on a level
BROKEN WALL	Number of wall broken on a level
TIME USED	Time to finish the level in milliseconds

Table 3 Description of the Parameters of player's behavior

LEVEL NUMBER	DEADEND	HAMMER	MAP SIZE	MONSTER	TIME EXPECT TO FINISH	DEAD END ACHIEVE	HAMMER USED	LEVEL PERFORMANCE *	MONSTER HIT	BROKEN WALL
1	0	0	15	0	30500	0	0	B	0	0
2	1	0	15	0	31500	0	0	B	0	0
3	1	0	10	0	34000	0	0	B	0	0
4	1	0	10	0	32500	0	0	B	0	0
5	2	0	12	1	31000	0	0	B	0	0
6	2	0	15	1	28000	0	0	W	0	0
7	2	0	12	1	28000	0	0	B	0	0
8	3	0	15	1	29000	0	0	W	0	0
9	3	1	12	1	34500	1	1	W	1	0
10	3	1	10	2	34000	1	0	B	0	1
11	3	1	15	2	28000	1	1	W	1	0
12	4	1	12	2	35000	1	0	B	0	1
13	4	2	10	2	30500	2	1	W	1	1
14	4	2	12	3	35000	2	2	W	2	0
15	4	2	10	3	33500	2	1	B	1	1
16	5	2	15	3	36500	1	1	W	1	0
17	5	3	15	3	34000	3	2	W	2	1
18	5	3	12	4	30500	3	3	W	3	0
19	5	3	15	4	27000	1	1	B	1	0
20	5	3	10	4	29000	1	0	B	0	0
* Performance: A- Same as expected time ±0.1second B-Time used is less than Time Expected W- Time used is more than Time Expected										

Table 4 Game Log Data

LEVEL NUMBER	DEADEND	HAMMER	MAPSIZE	MONSTER	TIME EXPECT TO FINISH
1	0	0	15	0	30500
2	1	0	15	0	31500
3	1	0	10	0	34000
4	1	0	10	0	32500
5	2	0	12	1	31000
6	2	0	15	1	28000
7	2	0	12	1	28000
8	3	0	15	1	29000
9	3	1	12	1	34500
10	3	1	10	2	34000
11	3	1	15	2	28000
12	4	1	12	2	35000
13	4	2	10	2	30500
14	4	2	12	3	35000
15	4	2	10	3	33500
16	5	2	15	3	36500
17	5	3	15	3	34000
18	5	3	12	4	30500
19	5	3	15	4	27000
20	5	3	10	4	29000

Table 5 Parameters of levels

LEVEL NUMBER	DEAD END ACHIEVE	HAMMER USED	LEVEL PERFORMANCE *	MONSTER HIT	TIME USED	BROKEN WALL
1	0	0	B	0	18677	0
2	0	0	B	0	21951	0
3	0	0	B	0	11396	0
4	0	0	B	0	17325	0
5	0	0	B	0	16513	0
6	0	0	W	0	32146	0
7	0	0	B	0	18657	0
8	0	0	W	0	33929	0
9	1	1	W	1	38155	0
10	1	0	B	0	30473	1
11	1	1	W	1	37874	0
12	1	0	B	0	33528	1
13	2	1	W	1	35241	1
14	2	2	W	2	41690	0
15	2	1	B	1	31105	1
16	1	1	W	1	63221	0

17	3	2	W	2	20639	1
18	3	3	W	3	16494	0
19	1	1	B	1	22919	0
20	1	0	B	0	14710	0

Table 6 Player’s Behavior

To predict the characteristics of the player’s behaviors in Table 4, we first determine if the attribute “NUMBER OF HAMMERS USED” is important for predicting the attribute “DEAD END ACHIEVE” on the next three maze levels. We then construct a contingency table with 6 (there is a maximum of 5 dead ends in the maze) columns and 4 (there is a maximum of 3 hammers used) rows (Table 7).

		DEAD ENDS ACHIEVED						
		ZERO	ONE	TWO	THREE	FOUR	FIVE	TOTAL
NUMBER OF HAMMERS USED	ZERO	4	4	0	4	0	0	12
	ONE	4	0	0	0	0	0	4
	TWO	0	0	0	1	0	0	1
	THREE	0	0	0	0	0	0	0
	TOTAL	8	4	0	5	0	0	17

Table 7 4 X 6 Contingency table for NUMBER OF HAMMERS USED and DEAD ENDS ACHIEVED

The value of the chi-square statistic, $\boldsymbol{X^2}$, is 33.15. $\boldsymbol{X^2}$ is greater than the critical chi-square value $\boldsymbol{x^2}_{15,0,05}$ and $\boldsymbol{x^2}_{15,0.01}$. The chi-square test is significant at both the 95% and 99%s. This means the attribute NUMBER OF HAMMERS USED is dependent on the attribute NUMBER OF DEAD ENDS ACHIEVED of the future three levels.

We used the chi-square test and found that the attribute “NUMBER OF HAMMERS USED” is important in determining the attribute “DEAD ENDS ACHIEVED” on the next three levels. It can predict that if a player used a hammer three times, it will meet a dead end twice on the future level. To predict the player’s future behavior, we can construct a set of prediction rules. They represent each detected dependence relation between two attributes values by using a rule in the following form:

If<***Condition***> then <***conclusion***> with certainty ***W.***

$$W(NUMBER\,OF\ HAMMER\,USED = ZERO\ /\ NUMBER\,OF\ HAMMER\,USED \neq S = ZERO$$
$$|\,DEAD\ HEAD\ ACHIEVE = THREE\,)$$

$$= log\frac{Pr(\,NUMBER\ OF\ \ HAMMER\ USED = ZERO\ |\ DEAD\ HEAD\ \ ACHIEVE = THREE\ \ )}{Pr(\,NUMBER\ OF\ \ HAMMER\ USED = ZERO\ |\ DEAD\ HEAD\ \ ACHIEVE \neq THREE\,)}$$
$$= 0.70$$

The prediction rule can be expressed as follows:

If a player did not use a hammer, then the player will, with a certainty of 0.70, meet a dead end three times on the next three levels.

All the rules can be constructed from the other relevant values. A set of prediction rules is constructed as follows:

1) If the player has broken the wall once, the player will, with a certainty of 3.29, use the hammer once on the next level.
2) If the player has met a dead end once, the player will, with a certainty of 3.29, use a hammer once on the next two levels.
3) If the player has met a dead end once, the player will, with a certainty of 3.29, break the wall once on the next two levels.
4) If a player has used a hammer once, the player will, with a certainty of 3.29, meet a dead end three times on next level.
5) If a player's performance is good, the player will, with a certainty of 2.22, meet a dead end three times on the next level.
6) If a player's performance is good, the player will, with a certainty of 2.22, use a hammer twice on the next two levels.
7) If a player's performance is good, the player will, with a certainty of 2.22, break a wall twice on the next two levels.
8) If a player's performance is good, the player will, with a certainty of 1.70, use a hammer twice on the next three levels.
9) If a player's performance is good, the player will, with a certainty of 1.70, break a wall once on the next levels.
10) If a player's performance is good, the player will, with a certainty of 1.70, break a wall once on the next two levels.
11) If a player's performance is good, the player will, with a certainty of 1.70, use a hammer once on the next two levels.
12) If a player's performance is good, the player will, with a certainty of

1.70, use a hammer once on the next levels.

13) If a player’s performance is good, the player will, with a certainty of 1.70, break a wall twice on the next three levels.
14) If the player broke the wall once, the player will, with a certainty of 1.70, use a hammer once on the next two levels.
15) If the player broke the wall once, the player will, with a certainty of 1.70, meet a dead end once on the next level.
16) If a player’s performance is good, the player will, with a certainty of 1.70, meet a dead end three times on the next two levels.
17) If a player’s performance is good, the player will, with a certainty of 1.7, meet a dead end once on the next level.
18) If a player used a hammer once, the player will, with a certainty of 3.29, break a wall once on the next two levels.

These prediction rules can be used to find the prediction parameters for the game environment and player behavior.

4.2 Decision Tree Approach

Future levels can also be predicted by using a decision tree. C4.5 is a decision tree-generating algorithm that is based on the ID3 algorithm by Quinlan [JQ1993]. The decision is grown using a depth-first strategy. ID3

(Induction Tree Decision) builds a decision tree from a fixed set of BEHAVIORS. The decision tree is used to classify future BEHAVIORS. The Behavior has several attributes and belongs to a class (like hot or cold). The leaf nodes of the decision tree contain the class name and a non-leaf node is a decision node. The decision node is an attribute test with each branch (to another decision tree) being a possible value of the attribute [JH2001].

The algorithm is summarized in Figure 17:

```
1)   Function : ID3_Decision_Tree
2)   Input : Sample = discrete-value attributes
4)   Input : Attribute-list= the set of candidate attributes
5)   Output : A decision Tree


6)   Create a node N;
7)   If samples are all of the same class, C then
8)   Return N as a leaf node labeled with the class C;
9)   If attribute-list is empty then
10)  Return N as a leaf node labeled with the most common class of samples;
11)  Select test-attributes, the attribute among attribute-list with the highest information
     gain;
12)  Label node N with test-attributes;
13)  For each known value $a_i$ of test-attribute
14)      Grow a branch from node N for the condition test-attribute-all
15)      Let $s_i$ be the set of samples in samples for which test-attribute-all
16)      If $s_i$ is empty then
```

```
17)                 Attach a leaf labeled with the most common class of samples;
18)            Else
19)                 Attach the node return by ID3_Descision_tress($s_i$,
                    attribute-list-test-attributes)
20)      Looping
```

Figure 17 Basic algorithm of ID3

C4.5 derives from ID3. C4.5 is an extension of the decision-tree. The algorithm consists of the following steps (Figure 18)

```
1)   Build the decision tree from the training set using conventional ID3.
2)   Convert the resulting tree into an equivalent set of classification rules. All possible
     paths from the root to a leaf node are reduced to a possible path
3)   According to a validation set, rules will prune each rule by removing any
     preconditions that result in improving its accuracy,
4)   Sort the pruned rules in descending order according to their accuracy
```

Figure 18 Algorithm of C4.5

A decision tree approach is presented by Duan Fu and Ryan Huletter. [SR2004] A decision tree was used in the game “Black & White”, developed by Lionhead Studios (2001).

Suppose the game log is a set of sequential patterns (Figure 13). It can be defined as follows: suppose that there is an ordered sequence ***S*** of ***M*** BEHAVIORS, $\boldsymbol{Behavior_1}$, …,$\boldsymbol{Behavior_p}$,…,$\boldsymbol{Behavior_m}$, where $\boldsymbol{Behavior_p}$ is located at position ***p*** in ***S***. Suppose also that each Behavior in the sequence is described by ***n*** distinct attributes, $\boldsymbol{Attr_{1p}}$, …, $\boldsymbol{Attr_{jp}}$, …, $\boldsymbol{Attr_{np}}$, and that in any instantiation of the Behavior description, an $\boldsymbol{Attr_{pj}}$ takes on a specific value,

val_{jp} $\in$ domain(***$Attr_{jp}$***)={***v_{jk}***| ***k***=1,…,***J***}, Transform the specific value , ***val_{jp}*** $\in$ domain(***$Attr_{jp}$***)={***v_{jk}***| ***k***=1,…,***J***}, into a category of the ***$Attr_{pj}$***.

When building a decision tree, the information gain measure is used to select the test attribute at each node in the tree. The attribute with the highest information gain (or greatest entropy reduction) is selected as the test attribute for the current node.

Suppose the ***k*** distinct value of class label attributes ***p*** on S. ***k*** distinct values defines ***k*** distinct classes, C_i (for ***i***=1,…,***k***)/. The expected information is given by

$$I(obj_1, obj_2, ...obj_p) = \sum_{i=1}^{m} p_i \log_2(p_i),$$

where ***p_i*** is the probability that an arbitrary sample of the sequence ***S*** belongs to class ***C_i*** and is estimated by ***$Behavior_i/S$***

.

On the attribute A, it contains k distinct values. The entropy is given by

$$E(A) = \sum_{n-1}^{k} \frac{obj_{1n} + ...obj_{mn}}{S} I(obj_1, obj_2, ...obj_p)$$

where ***m*** is the number of distinct classes.

Gain(A), the expected reduction in entropy caused by knowing the value of attribute A, is given by

$$Gain(A) = I(obj_1, obj_2, ...obj_p) - E(A)$$

The algorithm computes the information gain of each attribute. The attribute that has the highest information gain is chosen as the test attribute for the given set S. A node is then created and labeled with the attribute, branches are created for each value of the attribute, and the samples are partitioned.

The representation of the classification rules is in the form of IF-THEN rules. Each attribute-value pair along a given path forms a conjunction in the rule antecedent and is placed on the “IF” part. The leaf node holds the class prediction, forming the rule consequent as a “THEN” part. C4.5 shows the accuracy for every rule as a percentage.

On the testing model of the maze game, every level contains “easy” and “hard” level templates. We analyze the player’s behavior on the previous level. After analyzing the player’s behavior, the game engine predicts that the player should play the “hard” level on the coming level. Then the “hard” level of the coming level shows. Otherwise, the game engine generates an “easy” level for the next level. The parameters of the level are pre-defined. Figure 19 shows the game flow with the decision tree approach.

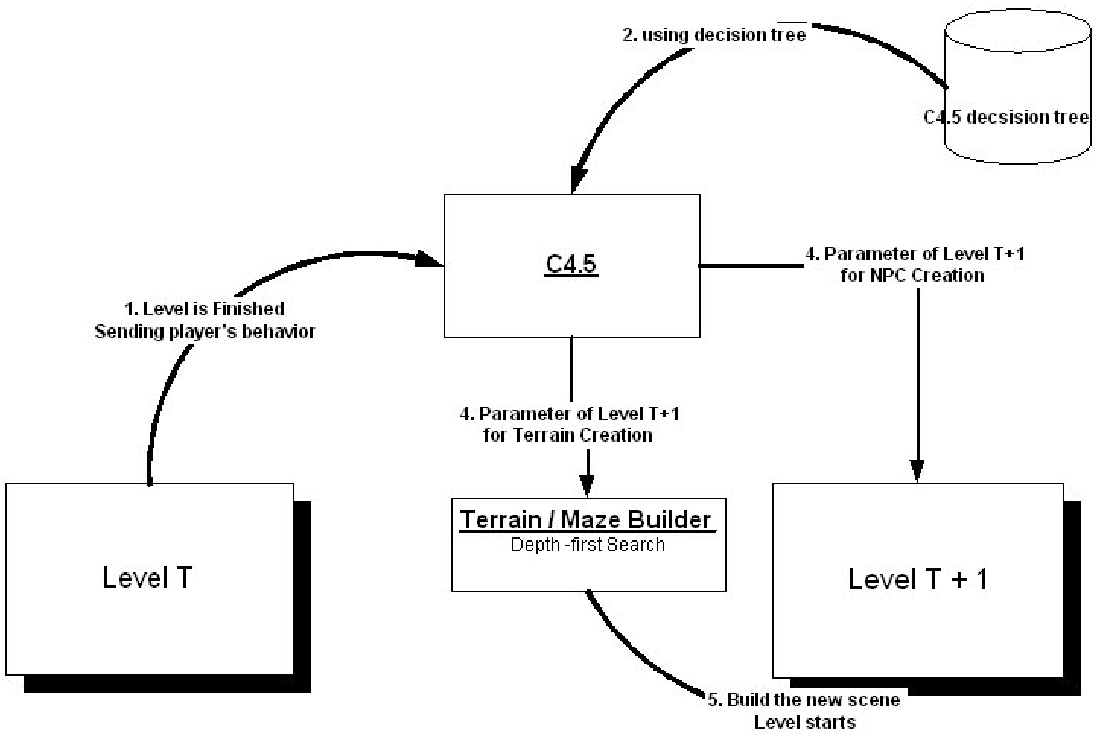


Figure 19 Game Flow of the Level design with decision tree approach

Chapter 5 Experiment and Implementation

In this chapter, we present our implementation of our proposed solution on a testing model – a maze builder. We use Java to implement the backend game engine which is including our proposed algorithm. The game log is saved as an XML file. A number of experiments were performed to verify the performance of the algorithm and determine if it had any advantages over other approaches. Section 5.1 describes the game. Section 5.2 presents the details of the implementation. Section 5.2 shows the performance of the algorithm on real data.

5.1 Game Description

Our testing model is a tile-based game similar to Pac-Man. The maze features a mouse which, to finish a level, must find the cheese and east it within a maximum time for each level (the game has twenty levels) shown on the progress bar. The game is over if the mouse cannot eat the cheese within this time. Figure 20 shows the game layout. There is an “Expected time” on every level. “Expected time” means how long the player should take to finish the current level. If the mouse eats the cheese within the expected time, the mouse is rated “Good” and we give his performance “B” in the game log. If the performance of the mouse is “Bad”, we record “W” in the game log. If the difference between the expected time and finishing time is between +0.1 second and –0.1 second, the mouse gets an “A”.

On every level, monsters appear to block the movement of the mouse but the mouse is provided with hammers to either kill the monsters or break through a wall.   data is captured in the game log as follows:

Name of Parameter	**Description**
LEVEL NUMBER,	Level number
DEADEND	Number of Dead Ends for the maze
HAMMER	Number of hammers provided to the player
MAPSIZE	There are three sizes of map, 10,12, and 15. The number of size is use to build the number of grids (Column X Row) in the map
MONSTER	Number of monsters in the maze
TIME EXPECT TO FINISH	The expected finishing time
DEAD END ACHIEVE	Number of Dead End that the player met of a level
HAMMER USED	Number of hammers used to hit the monster or break the wall of a level
LEVEL PREFORMANCE	A- Same as expected time ±0.1second B-Time used is less than Time Expected W- Time used is more than Time Expected

MONSTER HIT,	Number of monster hit on a level
BROKEN WALL	Number of wall broken on a level
TIME USED	Time taken to finish the level in milliseconds

Table 8 Game log data

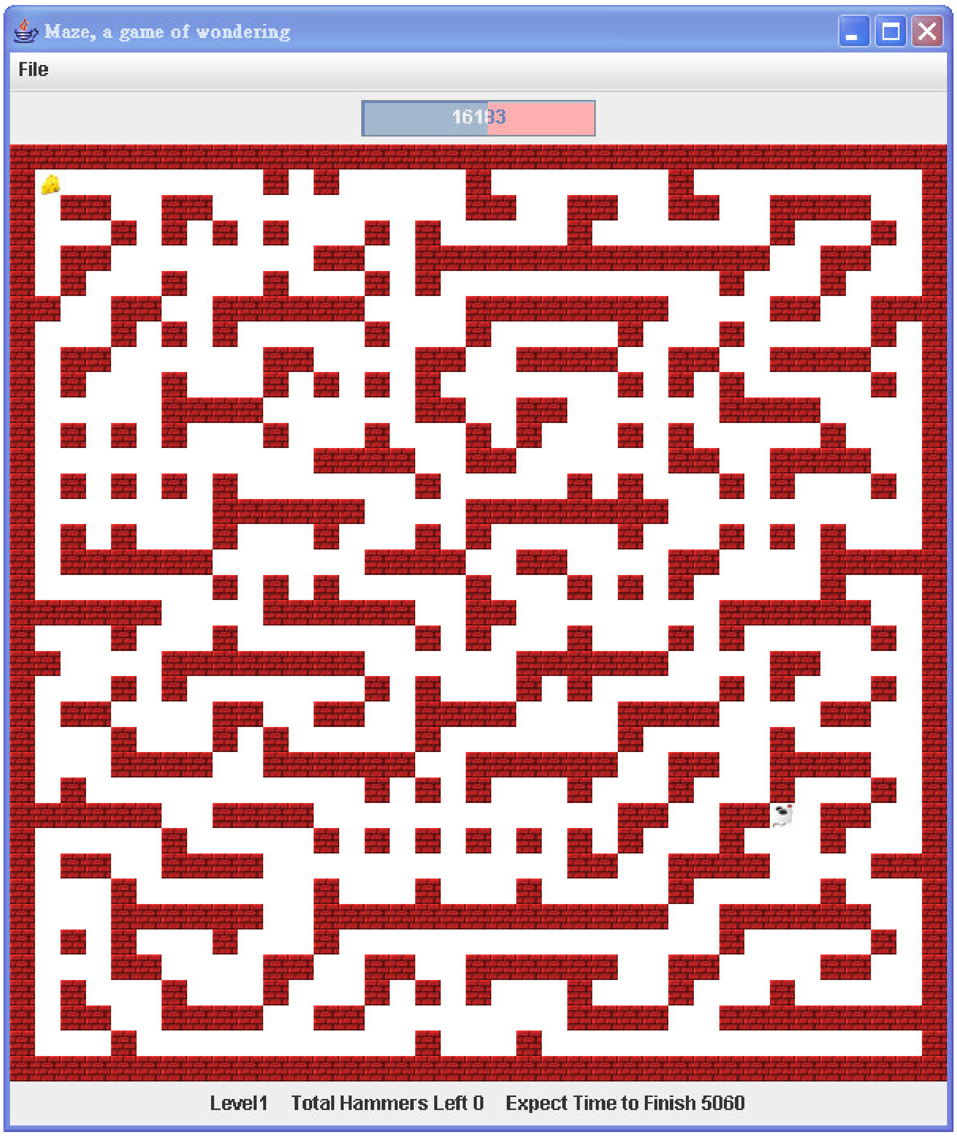

Maze, a game of wondering
File
16133
Level1 Total Hammers Left 0 Expect Time to Finish 5060

Figure 20 Maze Game

5.2 Details of the implementation

The maze game is implemented with Java and XML. When a player plays the game, a set of game logs is stored as an XML file. Figure 21 is a system diagram of the game. There are two parts to the game: the Game Engine and the Maze Builder.

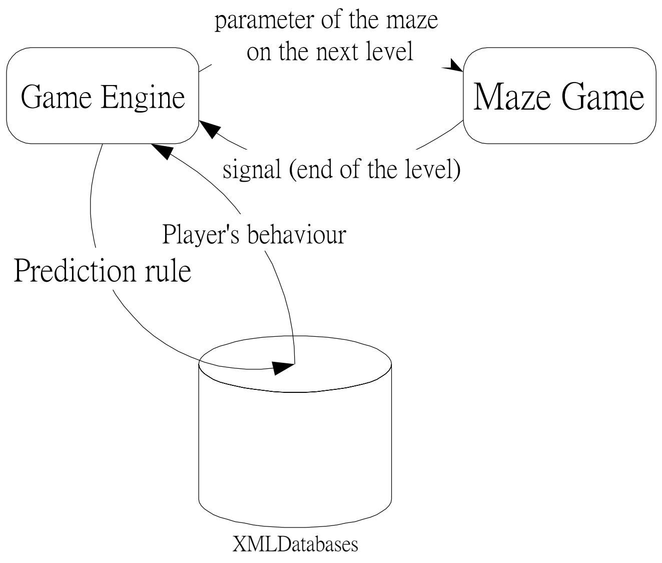


Figure 21 System Diagram of the Maze Game

The function of the game engine is to analysis the game log data and then export the parameters of the game environment to the next level of the game. The main task of the maze game is to generate the maze and the game layout control such as the movement of the mouse, timer control and the maze generation. The prediction rules and the player behaviors are stored in XML format. When a player plays the game, the game engine predicts the player’s behavior at the coming level by analyzing the game log data then passes the parameter of the game environment of the coming level to the maze builder. The maze builder uses these parameters to generate the game world. The general flow of the game is shown in Figure 22 and the class diagram is shown in Figure 23.

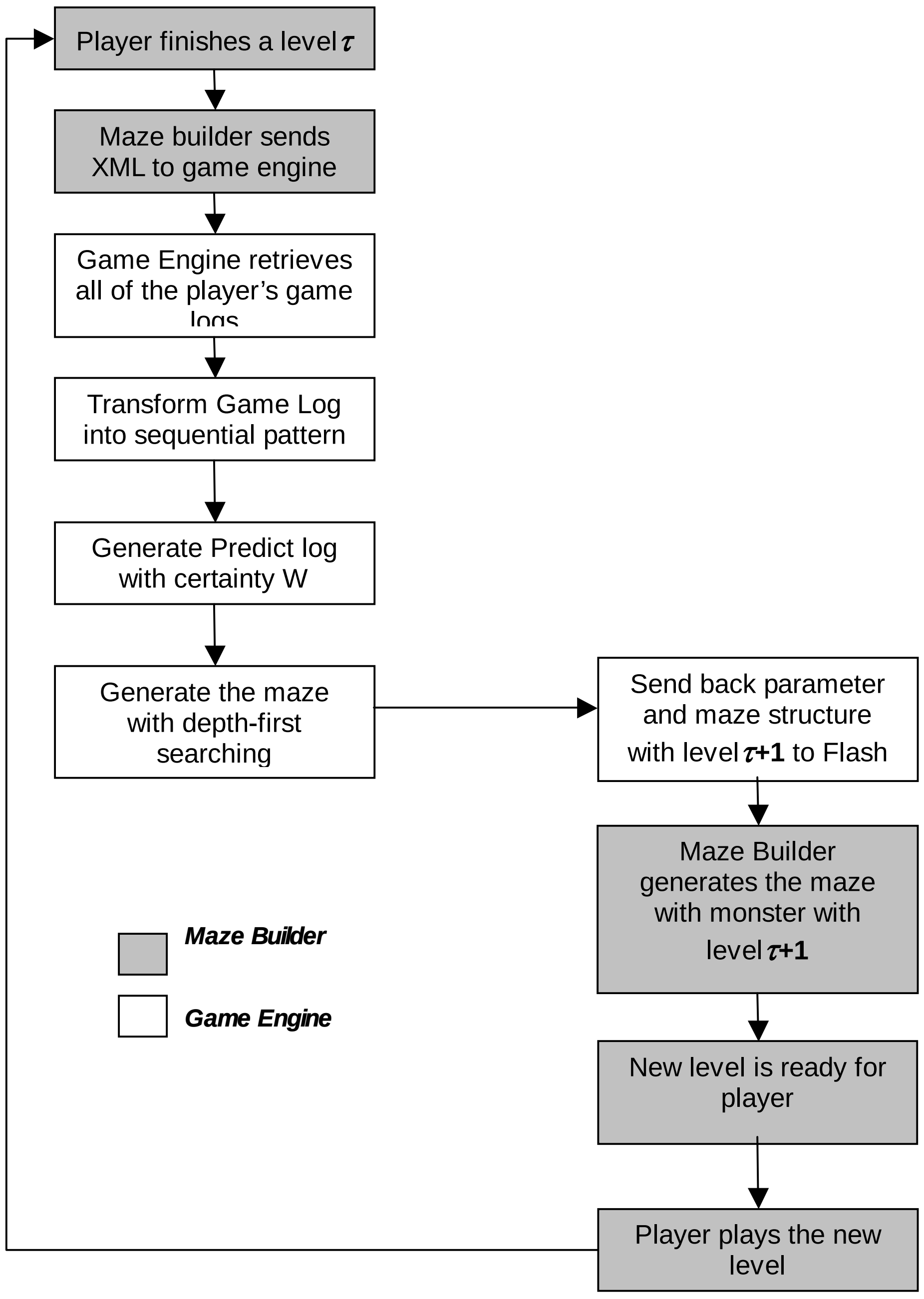

Player finishes a level τ
Maze builder sends XML to game engine
Game Engine retrieves all of the player's game logs
Transform Game Log into sequential pattern
Generate Predict log with certainty W
Generate the maze with depth-first searching
Send back parameter and maze structure with level τ+1 to Flash
Maze Builder generates the maze with monster with level τ+1
New level is ready for player
Player plays the new level
Maze Builder
Game Engine

Figure 22 System Flow of the Maze Game

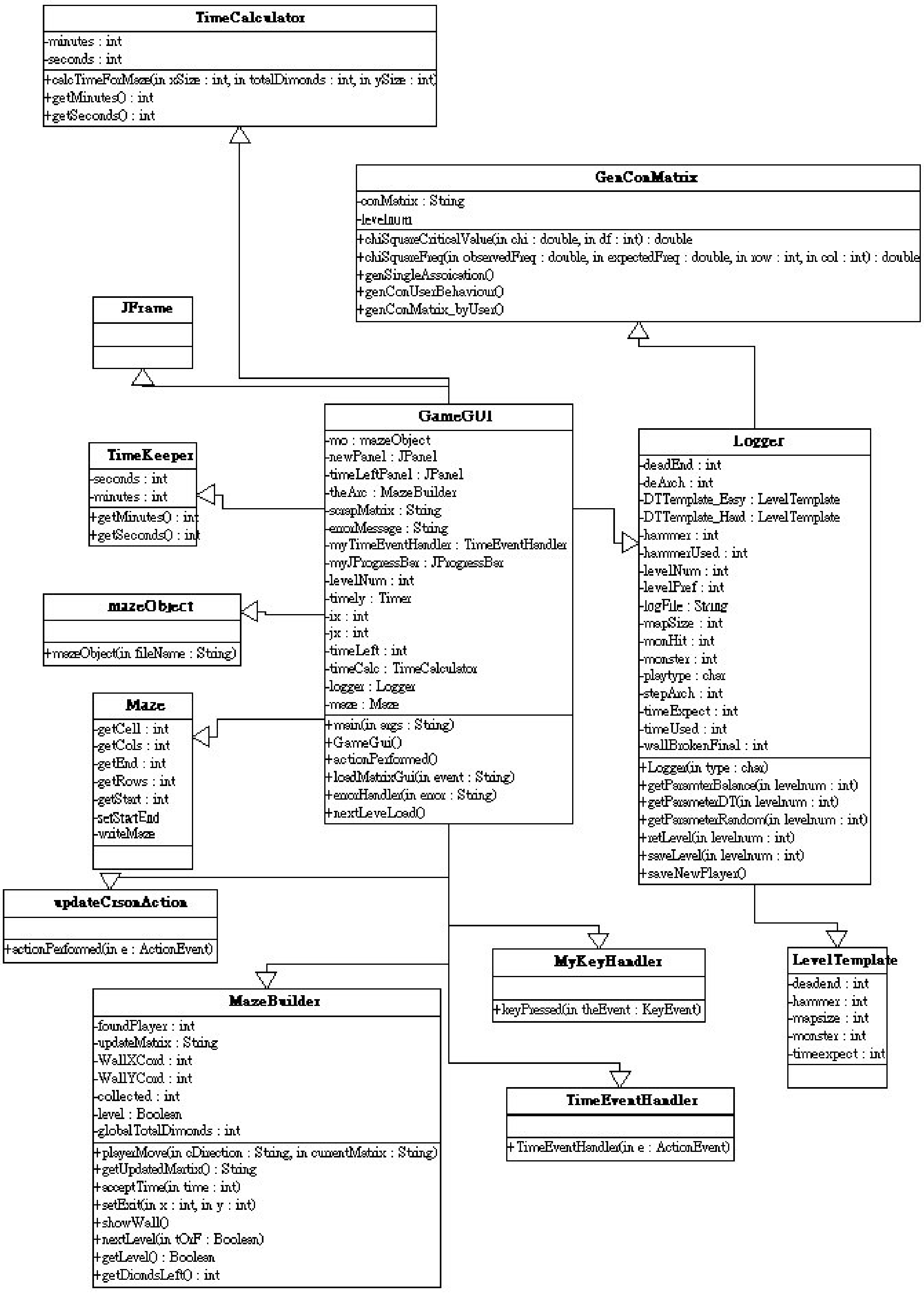


Figure 23 Class Diagram

5.2.1 Game Engine

When the game engine receives the start or complete signal from the maze builder, the game engine reads the player's behavior. Then it saves the current game log data in XML format and analyses that data using the previous game log data. Table 9 and Table 10 show the parameter of the game level and the player's behavior stored in XML format. The prediction rules with their weightings are generated and are saved back to XML for reference (Table 11). Then the game engine uses the prediction rules to adjust the global level parameters and the map of the game. The game engine sends back the parameters of the next maze to the Maze Builder. The main classes of the game engine are Logger, Maze and GenConMatrix.

```
<level>
      <levelNum>1</levelNum>
      <DEADEND>0</DEADEND>
      <hammer>0</hammer>
      <mapsize>12</mapsize>
      <monster>0</monster>
      <pathToTarget>1</pathToTarget>
      <stepToTarget>0</stepToTarget>
      <timeExcept>2700</timeExcept>
</level>
```

Table 9 Sample XML for a level

```
<userBehaviour>
      <levelNum>1</levelNum>
      <dhArch>0</dhArch>
      <hammerUsed>0</hammerUsed>
      <levelPref>W</levelPref>
```

```
        <monHit>0</monHit>
        <stepArch>31</stepArch>
        <timeUsed>6989</timeUsed>
        <monhitFinal>0</monhitFinal>
        <wallBrokenFinal>0</wallBrokenFinal >
</userBehaviour>
<userBehaviour>
        <levelNum>2</levelNum>
        <dhArch>0</dhArch>
        <hammerUsed>0</hammerUsed>
        <levelPref>W</levelPref>
        <monHit>0</monHit>
        <stepArch>31</stepArch>
        <timeUsed>5249</timeUsed>
        < wallBrokenFinal >0</wallBrokenFinal >
</userBehaviour>
```

Table 10 Sample XML file of the player's behavior

```
<Asso>
        <condition>
                <Attr>levelPref=B</Attr>
        </condition>
        <conclusion>
                <Attr>dhArch-2=3</Attr>
        </conclusion>
        <weight>1.700</weight>
</Asso>
```

Table 11 Sample XML for the prediction rule

The maze for our testing model is built using tile-based game theory with depth-first searching. The map is divided into several cells. The game maze is perfect. A cell can be represented as "Wall" or "Path". If the cell is a wall, the maze builder does not allow the player to walk through it. We use the array to represent the map. Figure 24 is a sample of a map with 25 X 25 grids and Table 12 is the array to represent that maze.

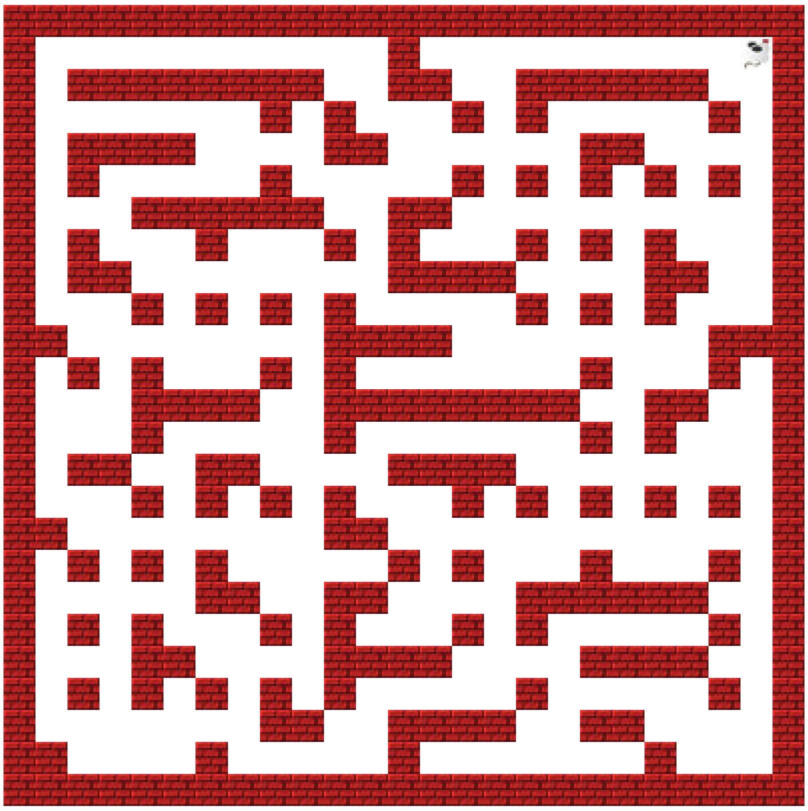


Figure 24 A tile-based game

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
0	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W
1	W												W											P	W
2	W		W	W	W	W	W	W	W	W			W	W			W	W	W	W	W	W			W
3	W								W		W				W		W						W		W
4	W		W	W	W	W					W	W							W	W					W
5	W		W						W						W		W		W		W		W		W
6	W				W	W	W	W	W	W			W	W											W
7	W		W				W				W		W				W		W		W				W
8	W		W	W									W	W	W	W					W	W			W
9	W				W		W		W		W						W		W		W				W
10	W	W									W	W	W	W									W	W	W

11	W		W		W				W		W								W				W		W
12	W				W	W	W	W			W	W	W	W	W	W	W	W			W	W			W
13	W				W						W								W		W				W
14	W		W	W			W	W					W	W	W	W									W
15	W				W		W		W		W				W		W		W		W		W		W
16	W	W									W	W													W
17	W		W		W		W						W		W				W				W		W
18	W						W	W			W	W					W	W	W	W	W	W			W
19	W		W		W				W		W				W		W						W		W
20	W				W	W					W	W	W	W					W	W	W	W			W
21	W		W		W		W		W		W						W						W		W
22	W								W	W			W	W	W	W			W	W					W
23	W	W					W						W								W				W
24	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W

Table 12 Data representative of the map in Figure 24

5.2.2 Maze Builder

The maze builder is used to build the game world and control the movement of the mouse. The main classes of the maze builder are GameGUI and MazeBuilder. Figure 25 is the layout of the game. The Maze builder recives the map and parameters of the game world and then draws the layout of the game. A player uses the arrow key to control the movement of the mouse. The bottom of the game shows the current level, the number of hammer available on the current level and the expected finishing time. Player can use the “Space” bar to use the hammer and break the wall or kill the monster. Then the number of hammers is reduced. There is a progress bar on the top of the game to show the maximum time allowed to complete the current level. If a player takes more than the expected time, the progress bar changes to red.

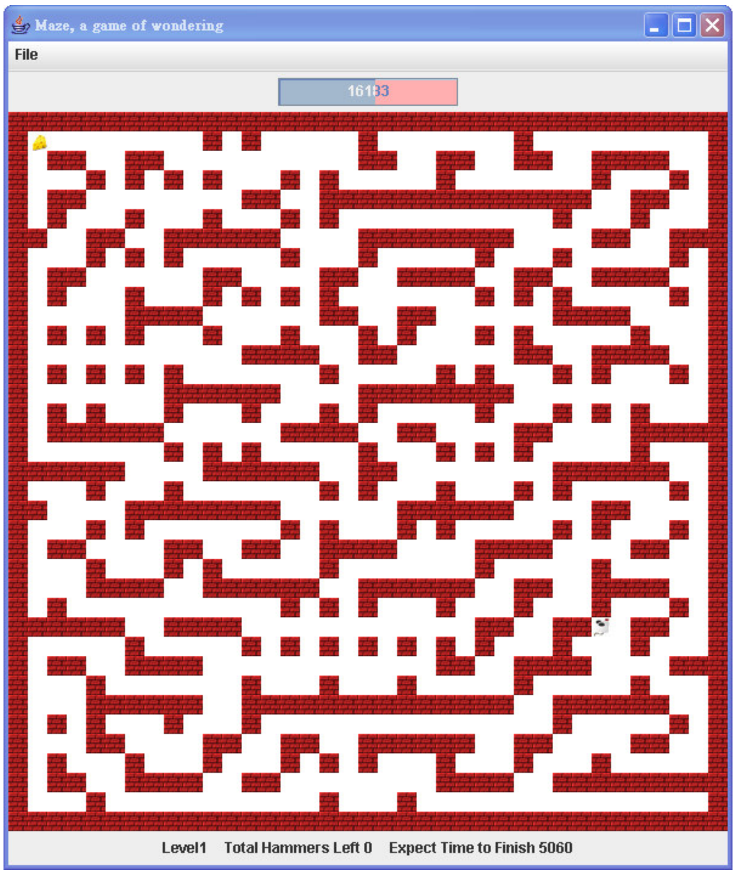

Maze, a game of wondering
File
16133
Level1 Total Hammers Left 0 Expect Time to Finish 5060

Figure 25 Maze Game

5.3 Details of the experiment

We introduced the testing model to ten different players who provided their feedback on the experience after playing. The main goal of testing was to find out if the players felt that the game was playable.

A player will play the three maze games: the original maze game using random number generation; the maze game using a decision tree; and the maze game using our adaptive game engine. They will play 20 levels for every game engine. The map may be 10, 12 or 15 grids; the maze may contain between 0 and 20 dead ends; there may be between 0 and 9 hammers for each player; and there are between 0 and 8 monsters. Table 13 shows the level parameters of the original maze game which is implemented with random numbers approach of one player. Table 14 is the game log of a player that played the maze game with those parameters.

LEVEL_NUMBER	MONSTER	DEAD_END	MAPSIZE	HAMMER	TIME_EXPECT
1	5	5	15	6	29500
2	1	13	10	2	33500
3	7	11	12	8	32500
4	0	8	15	1	31000
5	0	8	12	1	31000
6	2	2	15	3	28000
7	4	8	12	5	31000
8	6	6	10	7	30000
9	6	10	15	7	32000
10	0	16	12	1	35000
11	1	13	10	2	33500
12	3	3	10	4	28500
13	5	17	12	6	35500
14	5	13	12	6	33500
15	1	13	10	2	33500
16	4	8	10	5	31000
17	4	12	12	5	33000
18	6	6	12	7	30000
19	4	7	10	5	2700
20	4	8	15	5	31000

Table 13 Parameter of the Levels with random number

Level Number	DEAD END ACHIEVE e	HAMMER USED	LEVEL PREFORMANCE *	MONSTER HIT,	BROKEN WALL	TIME USED
1	1	3	B	0	3	20810
2	1	1	B	0	1	10545
3	0	2	B	1	1	18456
4	0	1	B	0	0	23554
5	0	1	B	0	0	19408
6	0	1	W	0	1	46977
7	0	2	B	0	2	30534
8	1	5	B	3	2	21501
9	1	3	W	0	3	37704
10	0	1	W	0	0	36723
11	0	1	B	0	1	26448
12	1	2	B	1	1	24615
13	1	3	W	0	3	39106
14	0	0	W	0	0	38766
15	1	1	W	0	1	41950
16	0	1	W	0	1	41740
17	0	3	W	0	3	42581
18	0	2	B	0	2	15813
19	0	1	B	0	1	11296
20	0	2	W	0	2	38740

* Performance:
A- Same as expected time ±0.1second
B-Time used is less than Time Expected
W- Time used is more than Time Expected

Table 14 Player's Behavior of playing random maze

For the maze game using decision tree approach, we trained the decision tree before the players played. Fifty players played the maze game in the random number generation approach. After collecting the data, we compare the parameters of every level to choose the "Easy" template and the "Hard" template. A "Hard" template is defined that the most players lost in the same level and same game parameters. A “Easy” template is defined that the most players win in the same level and same game parameters.Table 15 and Table 16 show level parameters of “Hard” and “Easy” respectively.

To define the decision tree and the classification rules, we transformed the game log into a set of row data and construct a decision tree with the following attributes, ‘Dead End Achieve”, “Monster Hit”, “Wall Broken”, “Hammer Used”, “Map Size”, “Monster”, “Hammer”, “Dead End” and two classes, “Best (B) Performance” and “Worst (W) Performance”. If the player is in the class “Best Performance” on the current level, the game will use the “Hard” template on the next level. If the player is in the class “Worst (W) Performance”, the game will use the “Easy” template on the coming level. Table 18 is the game log of the player.

LEVEL_NUMBER	MONSTER	DEAD_END	MAPSIZE	HAMMER	TIME_EXPECT
1	All player play easy template on Level 1				
2	5	5	12	6	19500
3	2	2	12	3	18000
4	0	0	10	1	17000
5	5	5	10	6	19500
6	2	6	15	3	20000
7	3	3	10	4	18500
8	6	2	12	7	18000
9	6	2	12	7	18000
10	1	5	10	2	19500
11	6	2	10	7	18000
12	5	5	10	6	19500
13	1	1	10	2	17500
14	4	0	10	5	17000
15	4	0	15	5	17000
16	1	1	15	2	17500
17	2	2	10	3	18000

18	1	1	15	2	17500
19	5	5	12	6	19500
20	2	2	10	3	18000

Table 15 Parameter of the “Hard” Levels with decision tree approach

LEVEL_NUMBER	MONSTER	DEAD_END	MAPSIZE	HAMMER	TIME_EXPECT
1	5	9	10	6	21500
2	3	15	15	4	24500
3	7	15	10	8	26500
4	6	14	15	7	24000
5	6	18	10	7	26000
6	5	17	10	6	25500
7	5	17	12	6	25500
8	7	19	12	8	28500
9	7	19	12	8	28500
10	7	19	12	8	28500
11	7	19	12	8	28500
12	7	11	10	8	24500
13	6	18	10	7	26000
14	7	11	12	8	24500
15	7	19	10	8	28500
16	2	14	10	3	24000
17	7	15	10	8	26500
18	7	15	15	8	26500
19	3	19	15	4	26500
20	7	19	12	8	28500

Table 16 Parameter of the “Easy” Levels with decision tree approach

The default class of the player is Easy.

If the player does not meet a dead end, the number of the monster is less than three, and the number of dead end of the map is less than eight, then the game engine will display an “Easy” level on the coming level with 93.6% accuracy.

If the player uses a hammer fewer than six times, the size of the map is 10 and the number of the dead end is less than eight, then the game engine will display an “Easy” level on the coming level with 91.5% accuracy.

If the player meets a dead end more than four times, the number of the

monster is more than four, and the number of the dead end is ten, then the game engine will display an “Easy” level on the coming level with 75.6% accuracy.
If the player meets a dead end more than eleven times, the size of map is 12, the number of the monster is fewer than four, and the number of dead end is less than thirteen, then the game engine will display an “Easy” level on the coming level with 70.7% accuracy.
If the player does not meet a dead end, the number of the monster of map is less than three, and the number of dead end is seven or eight, then the game engine will show a “Hard” level on the coming level with 63.0% accuracy.

Table 17 Classification Rules

```
Decision Tree:

deadEnd <= 6 : W (367.0/10.0)
deadEnd > 6 :
|   deadEnd <= 10 :
|   |   deadEnd <= 8 :
|   |   |   hammerused > 6 : B (4.0)
|   |   |   hammerused <= 6 :
|   |   |   |   monster <= 3 :
|   |   |   |   |   dhArch <= 0 : B (3.0)
|   |   |   |   |   dhArch > 0 :
|   |   |   |   |   |   hammerused > 3 : W (7.0)
|   |   |   |   |   |   hammerused <= 3 :
|   |   |   |   |   |   |   monhit > 0 : W (12.0/3.0)
|   |   |   |   |   |   |   monhit <= 0 :
|   |   |   |   |   |   |   |   mapsize = 10: W (8.0/1.0)
|   |   |   |   |   |   |   |   mapsize = 15: W (6.0/1.0)
|   |   |   |   |   |   |   |   mapsize = 12:
|   |   |   |   |   |   |   |   |   dhArch <= 4 : W (3.0)
|   |   |   |   |   |   |   |   |   dhArch > 4 : B (3.0/1.0)
|   |   |   |   monster > 3 :
|   |   |   |   |   mapsize = 10: W (8.0/2.0)
|   |   |   |   |   mapsize = 12:
|   |   |   |   |   |   monhit > 1 : B (2.0)
|   |   |   |   |   |   monhit <= 1 :
|   |   |   |   |   |   |   monhit <= 0 :
```

```
|   |   |   |   |   |   |   |   dhArch <= 6 : W (7.0/2.0)
|   |   |   |   |   |   |   |   dhArch > 6 : B (3.0)
|   |   |   |   |   |   |   monhit > 0 :
|   |   |   |   |   |   |   |   dhArch <= 3 : B (3.0/1.0)
|   |   |   |   |   |   |   |   dhArch > 3 : W (2.0)
|   |   |   |   |   mapsize = 15:
|   |   |   |   |   |   monster <= 5 : B (3.0)
|   |   |   |   |   |   monster > 5 : W (3.0/1.0)
|   |   deadEnd > 8 :
|   |   |   mapsize = 10:
|   |   |   |   deadEnd <= 9 :
|   |   |   |   |   wallbroken <= 2 : B (12.0/1.0)
|   |   |   |   |   wallbroken > 2 : W (3.0/1.0)
|   |   |   |   deadEnd > 9 :
|   |   |   |   |   hammerused <= 0 : W (2.0)
|   |   |   |   |   hammerused > 0 :
|   |   |   |   |   |   monster <= 4 : B (5.0/1.0)
|   |   |   |   |   |   monster > 4 :
|   |   |   |   |   |   |   dhArch <= 4 : B (5.0/1.0)
|   |   |   |   |   |   |   dhArch > 4 : W (3.0)
|   |   |   mapsize = 12:
|   |   |   |   deadEnd <= 9 :
|   |   |   |   |   monster <= 3 : B (8.0/2.0)
|   |   |   |   |   monster > 3 :
|   |   |   |   |   |   hammerused > 4 : B (4.0)
|   |   |   |   |   |   hammerused <= 4 :
|   |   |   |   |   |   |   monhit <= 1 : B (7.0/2.0)
|   |   |   |   |   |   |   monhit > 1 : W (2.0)
|   |   |   |   deadEnd > 9 :
|   |   |   |   |   wallbroken > 3 : W (3.0)
|   |   |   |   |   wallbroken <= 3 :
|   |   |   |   |   |   hammerused <= 1 : W (2.0)
|   |   |   |   |   |   hammerused > 1 : B (10.0/1.0)
|   |   |   mapsize = 15:
|   |   |   |   monster <= 1 : W (9.0/1.0)
|   |   |   |   monster > 1 :
|   |   |   |   |   deadEnd <= 9 :
|   |   |   |   |   |   dhArch <= 8 : B (9.0/1.0)
|   |   |   |   |   |   dhArch > 8 : W (2.0)
|   |   |   |   |   deadEnd > 9 :
|   |   |   |   |   |   monster <= 4 :
|   |   |   |   |   |   |   wallbroken <= 0 : W (2.0)
|   |   |   |   |   |   |   wallbroken > 0 :
|   |   |   |   |   |   |   |   monhit <= 0 : B (2.0)
|   |   |   |   |   |   |   |   monhit > 0 : W (3.0/1.0)
|   |   |   |   |   |   monster > 4 :
|   |   |   |   |   |   |   wallbroken > 3 : W (2.0)
|   |   |   |   |   |   |   wallbroken <= 3 :
|   |   |   |   |   |   |   |   dhArch <= 4 : B (3.0)
|   |   |   |   |   |   |   |   dhArch > 4 : W (5.0/1.0)
|   deadEnd > 10 :
|   |   deadEnd <= 13 :
|   |   |   mapsize = 10:
|   |   |   |   dhArch <= 6 : B (21.0)
|   |   |   |   dhArch > 6 :
|   |   |   |   |   wallbroken > 0 : B (11.0/1.0)
|   |   |   |   |   wallbroken <= 0 :
|   |   |   |   |   |   deadEnd <= 11 : B (5.0/1.0)
|   |   |   |   |   |   deadEnd > 11 :
|   |   |   |   |   |   |   dhArch <= 8 : W (2.0)
|   |   |   |   |   |   |   dhArch > 8 : B (2.0)
|   |   |   mapsize = 12:
|   |   |   |   monster > 4 : B (13.0)
|   |   |   |   monster <= 4 :
|   |   |   |   |   dhArch > 11 : W (4.0)
|   |   |   |   |   dhArch <= 11 :
|   |   |   |   |   |   dhArch > 3 : B (32.0/7.0)
|   |   |   |   |   |   dhArch <= 3 :
|   |   |   |   |   |   |   deadEnd <= 12 : W (6.0/1.0)
|   |   |   |   |   |   |   deadEnd > 12 : B (2.0)
|   |   |   mapsize = 15:
|   |   |   |   monster <= 4 : B (24.0/11.0)
|   |   |   |   monster > 4 :
|   |   |   |   |   dhArch <= 0 : W (2.0)
|   |   |   |   |   dhArch > 0 : B (22.0/3.0)
|   |   deadEnd > 13 :
|   |   |   deadEnd > 18 : B (50.0/1.0)
|   |   |   deadEnd <= 18 :
|   |   |   |   mapsize = 12: B (78.0/9.0)
|   |   |   |   mapsize = 15: B (90.0/10.0)
|   |   |   |   mapsize = 10:
|   |   |   |   |   wallbroken > 0 : B (57.0/2.0)
|   |   |   |   |   wallbroken <= 0 :
|   |   |   |   |   |   monhit <= 1 : B (25.0/2.0)
|   |   |   |   |   |   monhit > 1 :
|   |   |   |   |   |   |   deadEnd <= 16 : B (5.0/1.0)
|   |   |   |   |   |   |   deadEnd > 16 : W (2.0)
```

Figure 26 Decision Tree

Level Number	DEAD END ACHIEVE	HAMMER USED	LEVEL PREFORMANCE *	MONSTER HIT,	BROKEN WALL	LEVEL TEMPLATE	TIME USED
1	0	0	B	0	0	Easy	18236
2	1	2	B	0	2	Easy	21241
3	1	3	B	0	3	Hard	21291
4	0	0	B	0	0	Hard	18396
5	2	5	B	3	2	Easy	27269
6	1	4	B	1	3	Easy	24175
7	1	1	B	0	1	Easy	24395
8	1	1	B	0	1	Hard	24265
9	0	0	W	0	0	Hard	33508
10	1	7	B	0	7	Easy	30464
11	0	1	B	0	1	Hard	18366
12	0	3	B	0	3	Easy	27630
13	1	3	W	2	1	Easy	42271
14	1	1	B	0	1	Easy	30173
15	0	1	W	0	1	Hard	51554
16	0	2	B	0	2	Easy	24455
17	1	2	B	0	2	Hard	27560
18	0	2	W	0	2	Hard	51500
19	0	1	W	0	1	Easy	60500
20	0	4	W	0	4	Easy	64102
* Performance: A- Same as expected time ±0.1second B-Time used is less than Time Expected W- Time used is more than Time Expected							

Table 18 Player's behavior on Maze with Decision Tree

The maze game using our adaptive game engine also includes the game level template (Table 19). We tested the game with fifty players in a random number approach and collected the game log. We defined the parameters of the game level if the most of the players got a "B" on that corresponding level. Every player plays on the game with our adaptive game engine. The parameters of the first ten levels are the same. The game engine is adaptive on the last ten game levels. The parameters of the game are changed by the prediction rules which are real-time generated and mined from previous player behaviors.

When a player finishes level 10, the game engine analyses the player’s behavior and generates a list of prediction rules. Then the game engine changes the parameters of the next game level with those predictions rules according to their weighting. To make the game perform better, we only select the highest 40 weighting rules. Table 22 shows some prediction rules generated by our proposed solution. Table 20 and Table 21 show the final parameters of the levels and the game log… Table 22 is a prediction rule generated when a player plays beyond Level 10.

LEVEL_NUMBER	DEAD_END	HAMMER	MAPSIZE	MONSTER	TIME_EXPECT
1	0	0	12	0	27000
2	1	2	10	0	27500
3	3	4	10	3	28500
4	2	3	12	2	28000
5	2	3	12	2	28000
6	3	4	15	3	28500
7	2	3	12	3	28000
8	1	2	12	4	27500
9	1	2	10	3	27500
10	2	3	12	0	28000
11	1	2	15	1	27500
12	2	3	12	4	28000
13	0	1	15	2	27000
14	3	4	12	1	28500
15	3	4	12	3	28500
16	0	1	12	2	27000
17	1	2	10	1	27500
18	0	1	12	1	27000
19	2	1	12	3	28000
20	1	2	15	1	27500

Table 19 Default Parameters of Maze

LEVEL_NUMBER	DEAD_END	HAMMER	MAPSIZE	MONSTER	TIME_EXPECT
1	0	0	12	0	27000
2	1	2	10	0	27500
3	3	4	10	3	28500
4	2	3	12	2	28000
5	2	3	12	2	28000
6	3	4	15	3	28500
7	2	3	12	3	28000
8	1	2	12	4	27500
9	1	2	10	3	27500
10	2	3	12	0	28000
11	0	0	12	0	27000
12	1	2	12	2	27500
13	0	0	10	4	27000
14	1	2	15	2	28500
15	2	2	12	2	27500
16	3	0	12	1	27000
17	1	2	12	1	29000
18	2	2	15	0	27500
19	1	3	10	4	27500
20	1	2	12	2	31000

Table 20 Parameters of Maze with Prediction Rule of a player

Level Number	DEAD END ACHIEVE	HAMMER USED	LEVEL PREFORMANCE *	MONSTER HIT,	BROKEN WALL	TIME USED
1	0	0	B	0	0	18046
2	1	2	B	0	0	15091
3	0	3	B	1	2	18326
4	0	1	B	0	1	15112
5	0	1	W	0	1	30183
6	0	2	W	1	1	33558
7	1	1	B	0	1	27209
8	0	1	B	1	0	27229
9	0	1	B	0	1	24225
10	0	2	B	0	0	27490
11	0	0	W	0	0	27389
12	0	1	B	1	0	24455
13	0	0	B	0	0	20690
14	0	2	W	1	1	35496
15	0	1	B	0	1	19337

Level Number	DEAD END ACHIEVE	HAMMER USED	LEVEL PREFORMANCE *	MONSTER HIT,	BROKEN WALL	TIME USED
16	0	0	W	0	0	28714
17	0	1	W	0	1	28500
18	0	2	B	0	2	22471
19	0	2	B	0	2	20500
20	0	2	B	2	0	18500

* Performance:
A- Same as expected time ±0.1second
B-Time used is less than Time Expected
W- Time used is more than Time Expected

Table 21 Player's Behavior in Maze with Prediction Rule

If two hammers are used on level p, then there is a certainly of 2.9 that the player will meet a dead ends on level p+1
If two hammers are used on level p, then there is a certainly of 4.0 that the player break the two walls on level p+1
If a monster is hit on level p, then there is a certainly of 3.9 that the player will not break the wall on the level p+2.
If the maze size is 10 on level p, then there is a certainly of 2.2 that the player's performance is B on level p+2
If a wall didn't break on level p, then there is a certainly of 2.7 that the player will not hit the monster on level p+1

Table 22 Some Prediction Rules at Level 10

When they have finished three maze games, players are asked to rate each game on a scale of 1 to 10 with 1 being the lowest and 10 the highest The feedback form is a modified form of Sweetser's game flow evaluation form [PS2005] The categories of Sweetser's evaluation form include Concentration, Challenge, Player Skills, Control, Clear Goals, Feedback, Immersion and

Social Interaction. Our experiment focuses on Challenge only. The adaptive game engine is effective when the player starts to play Level 11. Some questions are about Levels 11 to 20 and compare the performance of the different approaches.

Question
1. The difficulty is same from Level 11 to Level 20
2. The Level of the game is random generated
3. The game is very easy for every level
4. The difficulty of level should increase as players progress through the game
5. The Path is easy to find
6. The Level shows adaptability
7. Fun
** 1 is the minimum score .10 is the maximum score. **

Table 23 Question of the feedback form

Table 24 shows the questions and the average scores. We can see that the maze game with the adaptive game engine compares favorably with the original and that difficulty adjustment makes the game more fun.

Criterion	Original Maze Game	Maze Game with C4.5	Maze Game with Prediction Rule
1. The difficulty is the same from Level 11 to Level 20	6.5	5.2	3
2. The Level of the game is randomly generated	9	5.8	5.4
3. The game is very easy at every level	5	4	3.8

4. The difficulty of level should increase as players progress through the game	4	6.3	7.2
5. The Path is easy to find	4.8	5	5.5
6. The Levels show adaptability	2.0	5.7	6.9
7. Fun	4.2	5.8	6.9

Table 24 Player ratings of original Maze Game vs, Predicted Maze Game

The players felt that the maze game using our proposed solution adapts to their behavior and is more interesting than either the original maze game or the maze game that uses a decision tree.

Figure 27 shows a balance graph for our three versions of the maze game. The horizontal axis represents the levels of the games and the vertical axis indicates the player’s performance as a comparison between the difference between the actual playing time and the expected playing time. It appears that the maze game using our proposed approach is a better fit for the ideal game progression, making it a more balanced game experience for the player.

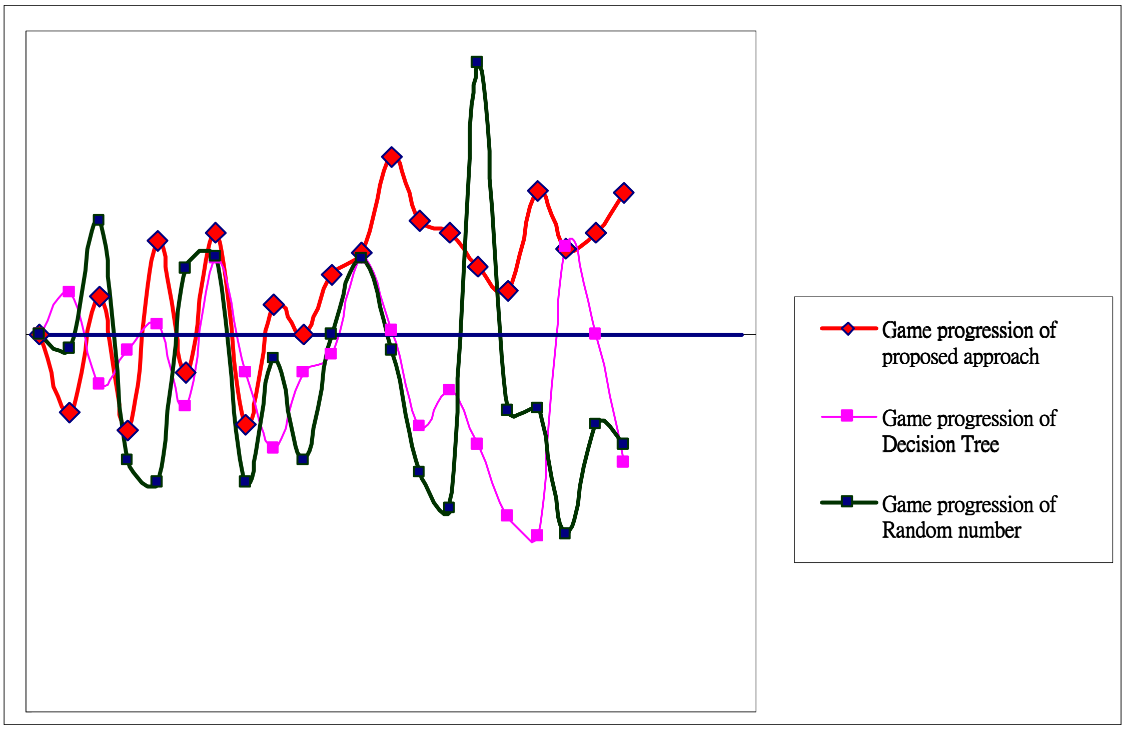


Figure 28, which compares the average performance of the ten players on the three maze games against the curve of our proposed algorithm, also shows them to be a good fit to the ideal game progression.

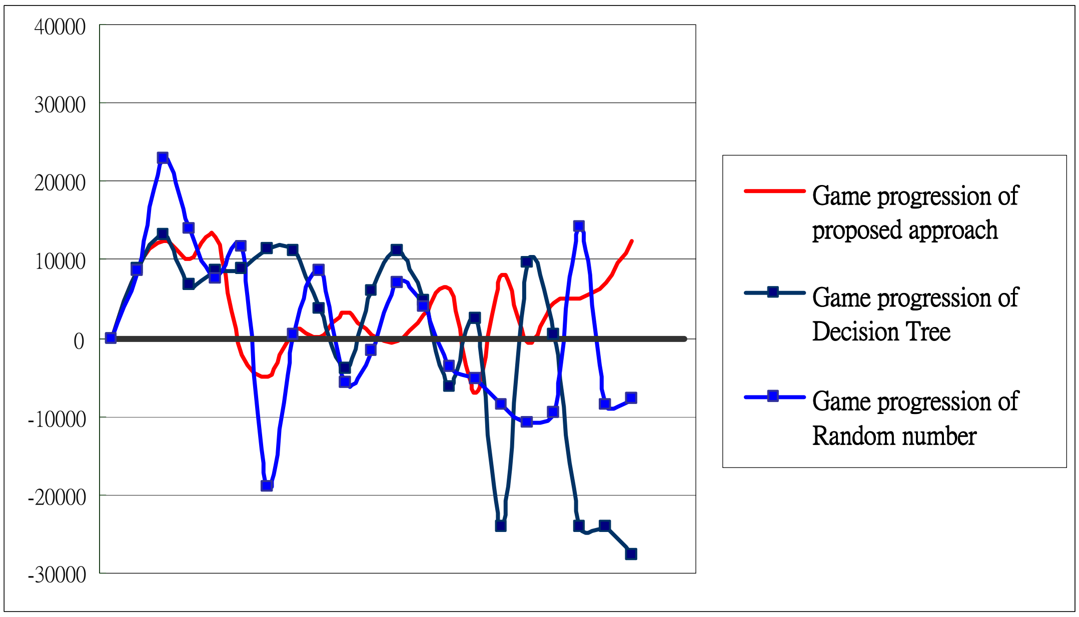


Figure 27 Balance graph of the three versions of the maze game for one

player

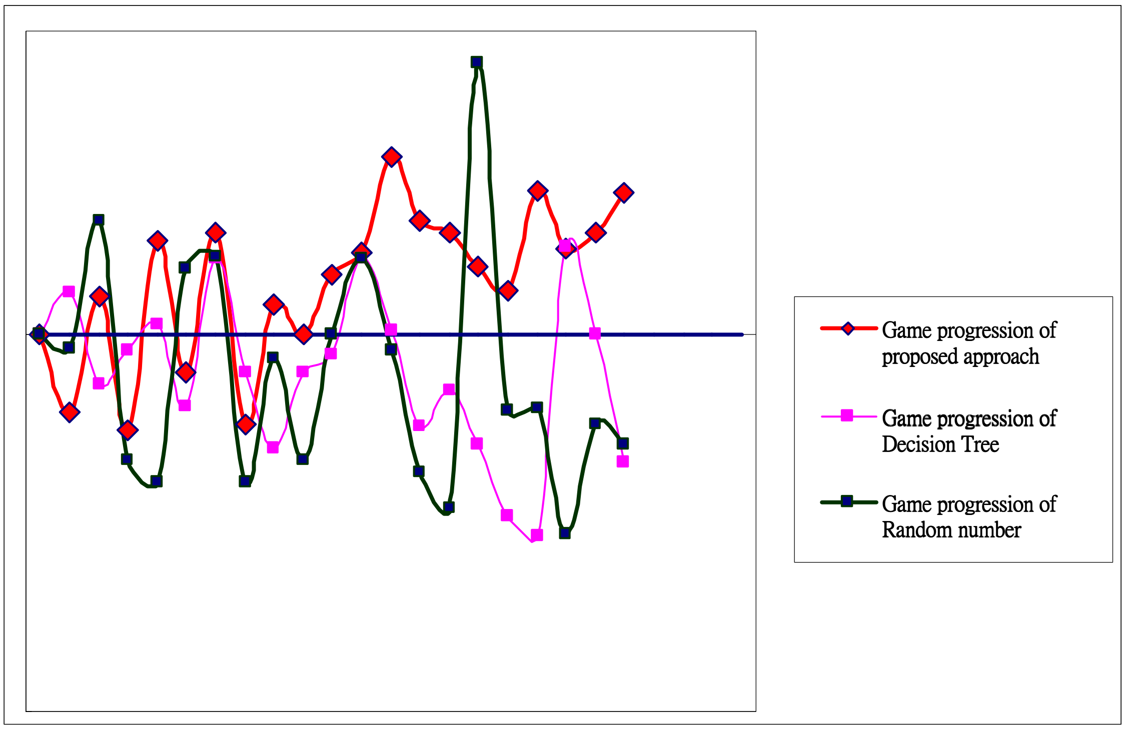


Figure 28 Balance graph of the three versions of the maze game showing the average performance of the ten players

Chapter 6 Conclusion and Future Research

6.1 Summary of the thesis

In this thesis, we have described several commonly used game AI techniques. Different game AI techniques are used in different types of games. Our proposed solution is a type of game AI focusing on dynamic game balancing. It uses a player's past experience to build the game environment. We have proposed a data mining technique that uses sequential pattern mining to build the game environment by using the relationship between characteristics of the player and the environment. The proposed solution is able to (1) detect underlying patterns in an ordered event sequence in a time series t, (2) use detected patterns to construct sequence-generation rules (3) use these rules to predict the attributes of players and generate a suitable environment for those attributes.

Our proposed solution is different from existing algorithms in that the game level is based on the past experience of the player, it is data-driven, and the game environment is unique and is not predefined.

Our proposed solution is used for building game environments. We have used a maze game as a test model. Not only can the solution build a maze, it can also help the game to build terrain, which may not be predefined, all to achieve the main goal of game development, to make games more fun.

6.2 Future research

Our proposed solution can also be applied to robotics training. The adaptive environment is based on the past behavior of the robot. Engineers can easily estimate the performance of the robot and train it.

A web site is also a type of maze in which users can easily get lost. In this case a users' visit log can be transformed into a set of sequential multivariate patterns. Our proposed solution can provide users with an interactive navigation menu and site owners can present users with suitable information or products. As a result, users will enjoy visiting the site and the usability of the web site will increase.

References/Bibliographies

1. [AN2004] A. Nareyek, ***"AI in Computer Game"***, ACM Queue vol. 1, no. 10 - February 2004, http://acmqueue.com/modules.php?name=Content&pa=printer_friendly&pid=117&page=1

2. [BB2004] B. Bates, ***"Game Design"***, Thomson Course Technology PTR, 2004

3. [BE2005] E. Byrne, ***"Game level design,"*** Hingham, Mass.: Charles River Media, 2005.

4. [BG2004] Bourg, David M.,Sebastopol , ***"AI for game developers"*** O'Reilly, 2004

5. [BM2005] Buckland, Mat. *"Programming Game AI By Example"*, Plano, Tex. : Wordware Pub., 2005.

6. [CA1980] C.Chatfield, A. J. Collins, ***"Introduction to Multivariate analysis"***, Chapman & Hall, 1980et

7. [CCH2001] C.H. Chan, ***"The Crazy Maze but Fun"***, http://www.hay.idv.hk/challenge/maze/, 2001

8. [CC2001] C.H. Chang, S.C. Lui and Y.C. Wu, ***"Applying Pattern Mining to Web Information Extraction"***, D. Cheung, G. J. Williams, and Q. Ku (Eds.): PAKDD 2001, LNAI 2035, pp 4 –15 , 2001

9. [CC2003] C.Crawford, "***Chris Crawford on game design*****",** Indianapolis,

Ind. : New Riders, 2003

10. [DCP2000] D.C. Pottinger, ***"Terrain Analysis in Realtime Strategy Games Technical Director, Ensemble Studio***s" http://www.gamasutra.com/features/gdcarchive/2000/pottinger.doc

11. [DD] ***"Dragons of Faerun Map Gallery",*** http://www.wizards.com/default.asp?x=dnd/ag/20060830a

12. [DK2003] D. Kennerl, "***Better Game Design through Data Mining***", Gamasutra, August 15, 2003 http://www.gamasutra.com/features/20030815/kennerly_01.shtml

13. [DO1974] D.B. Osteyee and I.J. Good, ***"Information Weight of Evidence, the Singularity between Probability Measures and Signal Detection"***, Berlin: Springer-Verlag, 1974

14. [EA2002] E Adams, ***"Balancing Games with Positive Feedback"***, Gamasutra, January 4, 2002 http://www.gamasutra.com/features/20020104/adams_01.htm

15. [GA2005] G. Andrade, G. Ramalho et V. Corruble , "***Automatic computer game balancing: a reinforcement learning approach***", In International Conference on Autonomous Agents and Multiagent Systems, pp. 1111--1112.,2005

16. [HJ2001] H. Pinto, J. Han, J. Pei, K. Wang, Q. Chen, and U. Dayal, ***"Multi-Dimensional Sequential Pattern Mining"***, Proc. 2001 Int.

Conf. on Information and Knowledge Management (CIKM'01), Atlanta, GA, Nov. 2001.

17. [INV] “***The history of Computer and Video game "*** http://inventors.about.com/library/inventors/blcomputer_videogames.htm

18. [JA2001] J.M. Adamo, ***“Data Mining for Association Rules and Sequential Patterns- Sequential and Parallel Algorithm”***, Springer, 2001

19. [JG1993] J. P. V. de Geer, ***“Multivariate Analysis of Categorical Data: Theory”,*** SAGE Publications Inc., 1993

20. [JH1999] J. Han, G. Dong and Y. Yin , ***“Efficient Mining of Partial Periodic Patterns in Time Series Database***”, IEEE ,1999

21. [JH2001] J. Han, M. Kamber, ***“Data Mining: Concepts and Techniques”***, Morgan Kaufmann publishers, 2001

22. [JJ2001] J.Juul, ***“A Clash between Game and Narrative”***, http://www.jesperjuul.net/thesis/, 2001

23. [JM2006] ***“The Game AI Page”***, http://www.gameai.com

24. [JP1995] J.S. Park, M.S. Chen, and P.S. Yu, ***“An Effective Hash Based Algorithm for Mining Association Rules”,*** In Proceedings of 1995 ACM SIGMOD International Conference on Management of Data, San Jose, May 1995, pp. 175-186.

25. [JP2000] J. Pei, J. Han, B. Mortaxavi-Asl and H. Pinto, ***“PrefixSpan: Mning Sequential Patterns Efficiently by Prefix- Projected***

Pattern Growth”, IEEE, 2000

26. [JQ1993] J. R. Quinlan,.: ***“C4.5: Programs for Machine Learning”***, Morgan Kauffman, 1993
27. [KC1994] Keith C.C Chan, Andrew, K.C. Wong, David, Y. K Chiu, ***“Learning Sequential Patterns for Probabilities Induction Prediction”***, IEEE Transations on System, Man and Cybernetics, Vol24, No. 10, October, 1994
28. [LL2002] L. Pantel and L. Wolf, ***“On The Suitability of Dead Reckoning Schemes for Games”*** in Proc. of NetGames2002, 2002
29. [LS1993’] L. Swift, ***“Time Series – forecasting, simulation, applications”***, Ellis Horwood Limited , 1993
30. [MAB] ***“Mabinogi”***, http://tw.mabinogi.gamania.com/
31. [MB2005] M. Buckland, ***“Programming Game AI by Example“*** , Wordware publishing, inc. 2005
32. [MF2003] M. Friedl ,“***Online Game interactivity Theory”***. Hingham, Mass : Charles River Media, inc., 2003
33. [MW2002] ***“MazeWorks”,*** http://www.mazeworks.com/mazegen/mazetut/index.htm
34. [MGAW] M. Gagnon and A.Withrow, ***“Road Blocks”***, http://www.2flashgames.com/f/f-861.htm
35. [PAC] ***“Pacman - Flash Game”,*** www.ebaumsworld.com/pacman.html
36. [PC2006] Phil Co, ***“Level Design for Games: Creating Compelling Game Experiences”***, Berkeley, Calif. : New Riders Games, 2006

37. [PIE2004] P. Spronck, I. Sprinkhuizen-Kuyper and E. Postma, "***Difficulty Scaling of Game AI"***. GAME-ON 2004: 5th International Conference on Intelligent Games and Simulation (eds. El Rhalibi, A., and D. Van Welden), pp. 33--37. EUROSIS, Belgium, 2004.

38. [PMIE2006] P Spronck, M Ponsen, I Sprinkhuizen-Kuyper, and E Postma . "***Adaptive Game AI with Dynamic Scripting. Machine Learning"***, Vol. 63, No. 3, pp. 217-248, 2006

39. [PS2005] P. Sweetser, P. Wyeth ***"GameFlow: a model for evaluating player enjoyment in games"***, Computers in Entertainment (CIE), Volume 3 Issue 3 ACM Press ,July 2005

40. [PS2007] P. Schuytema. ***"Game Deisgn: A Practical Approach"***, Charles River Media 2007

41. [RA1994] R. Agrawal, R. Srikant: ***"Fast Algorithms for Mining Association Rules"***, Proc. of the 20th Int'l Conference on Very Large Databases, Santiago, Chile, Sept. 1994

42. [RA1995] R. Agrawal and R. Srikant, ***"Mining Sequential Patterns"***, In Proc. 1995 Int Conf. Data Engineering, page 3-14, Taipei, Taiwan, March, 1995

43. [RE2003] R. E. Pedersen. ,***"Game design foundations "***, Plano, Tex. : Wordware Pub., 2003.

44. [RH2004] R. Hunicke and V. Chapman, "***AI for Dynamic Difficulty Adjustment in Games***". Challenges in Game Artificial

Intelligence AAAI Workshop, pp. 91-96, San Jose, 2004.

45. [RR2005] R. Rouse. ***"Game design: theory & practice"***, Wordware Pub., 2005.

46. [RV2004] R. Hunicke., V. Chapman, "***AI for Dynamic Difficulty Adjustment in Games***". In Proceedings of the Challenges in Game AI Workshop, Nineteenth National Conference on Artificial Intelligence (AAAI '04) (San Jose, California) AAAI Press, 2004.

47. [SL2006] S.Lee, K. Jung, ***"Dynamic Game Level Design Using Gaussian Mixture Model"***, Q.Yang and G. Webb(Eds): PRICAII 2006, LNAI 4099, pp 955-959 2006, Springer-Verlag Bedin Heidilbery, 2006

48. [SR2002] S. Rabin, ***"AI game programming wisdom"***, Hingham, Mass. : Charles River Media, 2002

49. [SR2004] S.Rabin, ***"AI game programming wisdom 2"***, Hingham, Mass. : Charles River Media, 2004

50. [SW1998] S. Woodcock, ***"Game AI: The State of the Industry, Game Developer"*** , Gamasutra, November 20, 1998 http://www.gamasutra.com/features/19981120/gameai_01.htm

51. [SW1999] S. Woodcock, ***"Game AI: The State of the Industry, Game Developer"*** , Gamasutra, August20, 1999 http://www.gamasutra.com/features/19990820/game_ai_01.htm

52. [TR1999] T. Ryan, ***"Beginning Level Design, Part 1: Level Design Theory"***, Gamasutra April 16, 1999. http://www.gamasutra.com/features/19990416/level_design_01.htm

53. [TR21999] T. Ryan, ***"Beginning Level Design, Part 2: Rules to Design By and Parting Advice"***, Gamasutra April 16, 1999. http://www.gamasutra.com/features/19990423/level_design_01.htm

54. [UO] ***"Ultima Online",*** http://www.uo.com/

55. [UU] ***"Usability Glossary Playability",*** http://www.usabilityfirst.com/glossary/term_657.txl

56. [WD2006] W. D. Pullen, ***"Think Labyrinth!"***, http://www.astrolog.org/labyrnth.htm

57. [WDM2006] ***"10 Usability Principles to guide you through the Web Design Maze"***, http://www.humanfactors.com/downloads/10tips.asp

58. [WL2002] W. Lin, M. A. Orgun and G. Williams; ***"An overview of temporal data mining"***. In The Proceedings of The Australasian Data Mining Workshop, Held in Conjunction with The 15th Australian Joint Conference on Artificial Intelligence, 3 December 2002, Canberra

59. [WIKI1] ***"Wikipedia- Game_programming"***, http://en.wikipedia.org/wiki/Game_programming

60. [WIKI] ***"Wikipedia"***, http://en.wikipedia.org/

61. [WIKI2] ***“Wikipedia-Artificial_Intelligence”,***

http://en.wikibooks.org/wiki/Artificial_Intelligence

62. [WIKI3] ***“Wikipedia-Strong AI vs. Weak AI”,***

http://en.wikipedia.org/wiki/Strong_AI

63. [WIKI4] ***“Wikipedia-Dynamic Game Balancing”,***

http://en.wikipedia.org/wiki/Dynamic_game_balancing,

64. [WIKI5] “***Wikipedia- Level design***”,

http://en.wikipedia.org/wiki/Level_design

65. [WIKI16] “***Wikipedia- Level Editor***”,

http://en.wikipedia.org/wiki/Level_editor

66. [WIKI17] “***Wikipedia- Level Designer***”,

http://en.wikipedia.org/wiki/Level_designer

67. [WIKI18] “***Wikipedia- Maze generation algorithm***”

http://en.wikipedia.org/wiki/Maze_generation_algorithm

68. [WIKI19] ***“Wikipedia- Game Engine***”,,

http://en.wikipedia.org/wiki/Game_engine