SUWAN KINGDOM GAME

TAWATCHAI DECHWONGPIBUL

PHETCHARUT LIKITMONGKONSAKUN

A SPECIAL PROJECT SUBMITTED IN PARTIAL FULFILLMENT

OF THE REQUIREMENT FOR THE DEGREE OF BACHELOR OF SCIENCE

IN COMPUTER SCIENCE (INTERNATIONAL PROGRAM)

FACULTY OF SCIENCE

KING MONGKUT'S INSTITUTE OF TECHNOLOGY LADKRABANG

ACADEMIC YEAR 2011

Thesis Title	Suwan Kingdom Game
Students	Mr. Tawatchai Dechwongpibul ID: 50050260
	Miss Phetcharut Likitmongkonsakun ID: 50050277
Degree	Bachelor of Science
Major	Computer Science (International Program)
Academic Year	2011
Advisor	Dr. Rungrat Wiangsripanawan

ABSTRACT

In this special project, a multiplayer game called "Suwan Kingdom Game" was developed. This game contains characteristics of role-playing game (RPGs), turn based, strategy and board game altogether. The game allows up to 4 players to play. In each turn, each player has to fight with monsters, other players or the Big Boss to earn more assets. The game ends when the last target of the game which is the Big Boss is killed and the player who posses the highest amount of assets collected strategically during the game's journey is the winner. The game also allows players to save the game to play later. Some background music (BGM) and some character clothes are from Thai traditional style. The game was developed using Adobe Flash CS3.

Keywords: Flash game, Action script, Game planning, RPGs

ACKNOWLEDGEMENTS

This special project Suwan Kingdom Game would not have been possible without the help from several people.

First and foremost, we would like to thank our parents for their love and support. Thanks for always cheering us up, motivating us and understanding us throughout our studies especially during this special project.

We also would like to give our special thanks to Dr.Rungrat Wiangsripanawan, our supervisor, for her guidance and encouragement throughout the project.

Our grateful thanks are also extended to Dr. Warangkhana Kimpan and Asst.Prof.Dr. Korakot Prachumrak, our committees, for their suggestions on the game design and the improvement of the game function.

Last but not least, we wish to thank all our friends for their support and help while we were doing this special project.

Tawatchai Dechwongpibul

Phetcharut Likitmongkonsakun

Contents

Contents (Cont.)

Contents (Cont.)

List of Tables

List of Figures

List of Figures (Cont.)

List of Figures (Cont.)

List of Figures (Cont.)

List of Figures (Cont.)

List of Figures (Cont.)

Chapter 1

Introduction

1.1 Statement and significance of the problems

Nowadays, computer games are one of the most popular computer applications. From a simple turn based game like tic-tac-toe, today games' technology and playing method have hugely developed. Games' stories are much more complex. Graphics used are massively improved. Many types of game such as action or role playing games were introduced. New released games seem to be a mixture of several types (or sometimes is call genres) together. Since playing game with friends or someone that the players know are more fun and challenging, several of the new games are also multiplayer games. The multiplayer games allow many players to play the game at the same time on the same environment together.

In this special problem project, we are interested in developing a multiplayer game that will allow from 2-4 players playing at the same time on the same machine. Our game is a mixture of role-playing games (RPG), turn based, strategy and board games together. Each player is allowed to choose his job and assume that job role. Each player plays his turn one-by-one (typical turn-based game). During his turn, the player will be moved on the exploration field and perform some actions according to a set of rules (a kind of board game). However, unlike the typical board game in which the person reaches the end point will win, our game justifies the winner from the player who possess the highest amount of asset (strategy game). Strategies and plans with some luck involved are significant factors to win our game. Since, it takes a while to finish our game, we have a save and load function allowing players to save their information so that they can quit the game and continue to play it later at the stage that they leave. This game is developed by using Adobe Flash CS3.

1.2 Goal and objective

1.2.1 Study how to design and develop a game application.

1.2.2 Study how to develop a game using Flash ActionScript by using Adobe Flash CS3.

1.2.3 Study how to use Adobe Photoshop for create graphic.

1.3 Scope of the study

1.3.1 Develop 2d multiplayer RPGs from Adobe Flash CS3.

1.3.2 Game can be played from 2-4 Players.

1.3.3 Game is a combination between RPGs game and board game.

1.3.4 Game can be saved or loaded to play later.

1.4 Expected benefits

1.4.1 Understand how to design and develop a game application.

1.4.2 Improve Flash ActionScript skills.

1.5 Process of the study

1.5.1 Design a game and its story.

- Design characters and scenes.
- Create characters and scenes.

1.5.2 Study and choose tools.

1.5.3 Implement the game using Flash ActionScript.

1.5.4 Test the game.

1.5.5 Write the document of the game.

1.5.6 Fix bugs of the game.

1.5.7 Make installation file and record into CD.

Chapter 2

Background

This chapter begins with an introductory to the game genre. Then, a brief description of some genres such as action, adventure, role-playing, simulation, strategy and some other genres will be described. Next, the description of "Flash games" will be given. Finally, Adobe Flash which is the tool that used to create Flash games will be explained.

2.1 Computer game genres [1]

In fictions such as films or books, the word 'genre' is used for the type of the content. However, in video games, it is classified based on the play interaction rather than visual or narrative differences. A video game genre is defined by a set of game play challenges which is independent from their setting or game-world content.

2.1.1 Action Game [2]

Action game is a video game genre that emphasizes on physical challenges such as hand-eye coordination and reaction-time. Fighting games, shooting games, and platform games are widely considered the most popular action games. Some real-time strategy games can be considered as action games. Examples of this game are New Super Mario Bros, Harry porter, Alice: Madness Returns and etc.

Figure 2.1: An Example of Action Game from New Super Mario Bros.

2.1.1.1 Action-Adventure Game [3]

Action-adventure game is a video game that combines some properties of the adventure game with some action game properties. It might be the most diverge and the broadest genre in gaming. It can include many games which can also categorize under other genres which are narrower. The first known game in this genre is the Atari 2600 game Adventure (1979). Examples of this game are God of war, Uncharted, Zelda and etc.

Figure 2.2: An Example of Action-Adventure Game from God of War 3

2.1.1.2 Shooting Game [4]

Shooting game is a subgenre of action game. The objective of the game is to test the player's ability in speed and reaction time. This is quite very broad subgenres. For example, the First Person Shooter (FPS) game also falls in this group. To clarify, Shooting game also includes games in other subgenres. These games commonly focus on the actions of the avatar using some sort of weapons, which is usually a gun or some other long-range weapon. Ammunition is another common resource found in many games in this type. The most common thing in any variance of the shooting game is the purpose of the game. That is its player's character can shoot his opponents and then proceed through missions without dying. Examples of this game are Battlefield, Crysis, Medal of Honor and etc.

Figure 2.3: An Example of Shooting Game from Battlefield 3

2.1.2 Adventure Game [5]

Adventure game is a video game genre that its player assumes the role of protagonist in an interactive story. Instead of physical challenge, this type of games focuses on the story which is driven by the exploration and puzzle-solving. The story is heavily drawn from other narrative-based media such as literatures and films so it encompasses a wide variety of literary genres. Emphasizing on story and character makes multi-player design for adventure game difficult. Hence, almost all adventure games are designed for a single player. Examples of this game are Prince of Persia, Portal, Heavy rain and etc.

Figure 2.4: An Example of Adventure Game from Prince of Persia

2.1.3 Role-Playing Games [6]

Role-playing games (commonly known as **RPGs**) is a video game genre which has its origin from pen-and-paper role-playing games such as Dungeons & Dragons. The player in RPGs controls one character or several adventuring party members in order to fulfill one or many quests. RPGs use much of terminology, settings and game mechanics from the pen-and-paper games. Their major similarities are the way they developed story-telling and narrative elements, their player character development, their complexities, as well as their abilities to replay and their immersion. Examples of this game are Final Fantasy, Dragon Quest, Shadow Hearts and etc.

Figure 2.5: An Example of RPGs from Final Fantasy XIII

2.1.4 Simulation Game [7]

Simulation game describes a diverse super-category of computer and video games. Typically, it is designed to closely simulate aspects of a real or fictional reality such as construction and management simulation, life simulation and vehicle simulation. Examples of this game are The Sims, SimCity, The Tower and etc.

Figure 2.6: An Example of Simulation Game from SimCity 4

2.1.5 Strategy Game [8]

Strategy game is a video game genre that emphasizes on the player's thinking and planning skills to win the game. Challenges in this type of game are focused on strategic, tactical, and sometimes logistical techniques of the player. Many games also offer economic challenges and exploration. These games sometimes also incorporate physical challenges but these challenges must be ones that can annoy player's mind strategically. Examples of this game are Warcraft, Starcraft, Redalert and etc.

Figure 2.7: An Example of Strategy Game from Warcraft 3

2.1.6 Some Other Game Genres

2.1.6.1 Music Game [9]

Music game commonly challenges its players to follow some sequences of movement or develop some specific rhythms. In some games, players are required to input rhythms by stepping their feet on a dance pad, or using a device similar to a specific musical instrument, like a replica drum set. Examples of this game are Guitar Hero, JamLegend, Rock Band and etc.

Figure 2.8: An Example of Music Game from Guitar Hero 3

2.1.6.2 Puzzle Game [10]

Puzzle game challenges its player to solve puzzles or to navigate through complex locations such as mazes. It is frequently come as a hybrid with adventure or educational games. Examples of this game are Zuma, Bejeweled, Puzzle Bubble and etc.

Figure 2.9: An Example of Puzzle Game from ZUMA

2.1.6.3. Sports Game [11]

Sports game imitates the way the sports in real life are played. In the game player can be either the player of that particular sport himself or the one who uses the strategy behind the players such as the manager of the team. Examples of this game are Winning, FIFA, Championship Manager and etc.

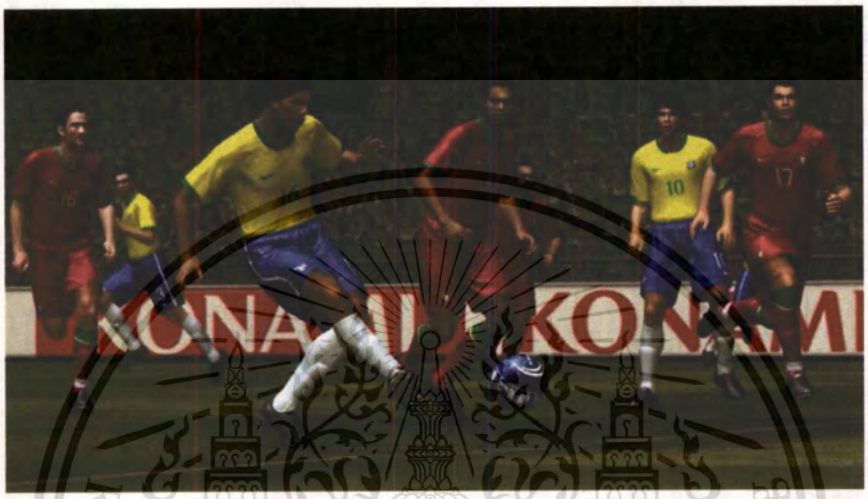


Figure 2.10: An Example of Sport Game from Winning Eleven 2011

2.1.6.4. Trivia Game [12]

Trivia game is a type of game which player play as an individual or a team member tries to answer questions correctly. The popularity of this game has been growing especially on the mobile phones where people do not have much time to play the game. Examples of this game are MobileQ, Brain Ages, Big Brain Academy and etc.

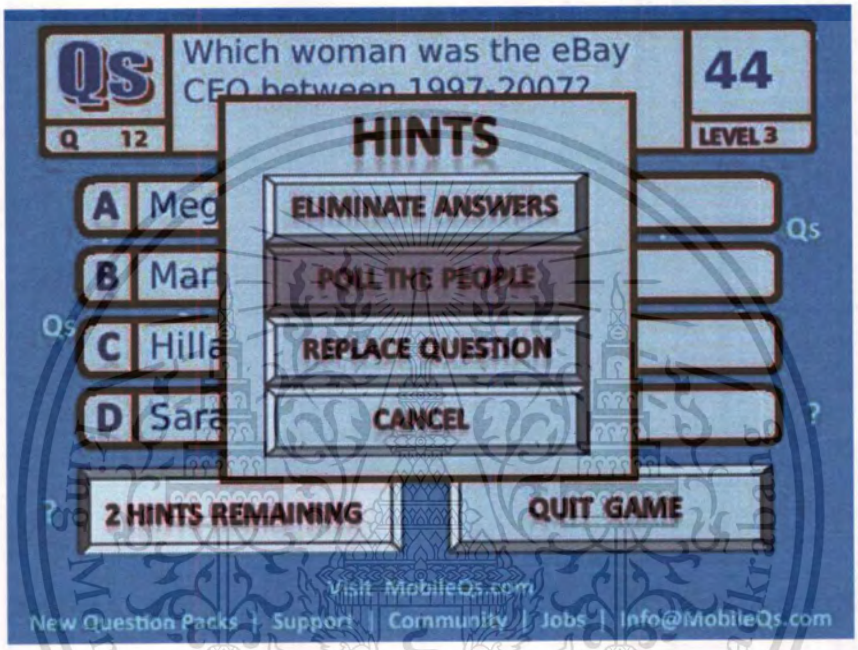


Figure 2.11: An Example of Travia Game from MobileQ

2.1.6.5 Board Game [13]

Board game is a game which players place, remove or move their pieces or counters on a premarked surface or "board" according to the set of game rule. It is originated from a real world "Monopoly game". Usually, players in this game type have their goals to accomplish. In some board games, the players might require purely good strategy or chance to achieve their aims. Examples of this game are Monopoly, Itadaki Streets, Trade Mania and etc.

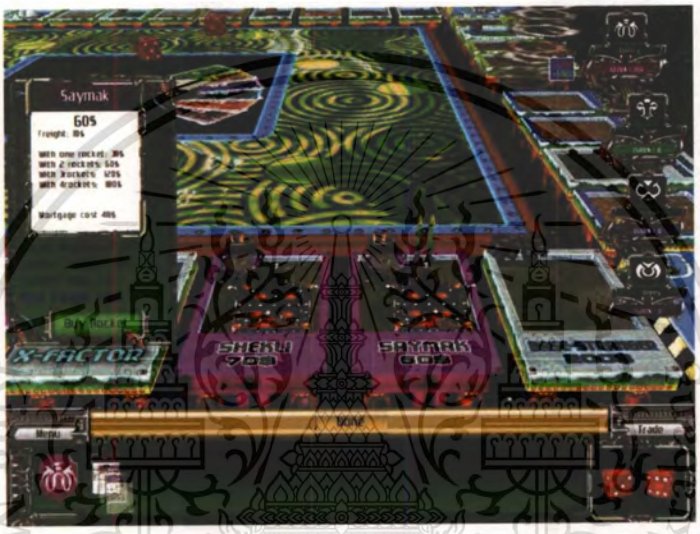


Figure 2.12: An Example of Board Game from Monopoly Galactic Imperia

2.2 Flash game

Flash game is game types that named after the software used to develop them. **Flash game** is developed by using the **Adobe Flash** (formerly known as **Macromedia Flash**) software. Game programmers use Adobe Flash to create interactive games through the ActionscSript. Example of early Flash games is *Pac-Man* and *Tetris*, which were released in the 1980s.

2.3 Adobe Flash [14]

Adobe Flash or **Flash** in short is a multimedia program developed by Macromedia Inc. (now owned by Adobe Systems). It was originally developed to add animation, video and interactivity to web pages. Flash animates text, drawings or still images by manipulating vector and raster graphics. Flash is frequently used to create cartoon advertisements. It is also capable to support bidirectional streaming of audio and video. In addition, Flash provides functions to capture user input via mouse, keyboard, microphone and camera. For an Object-oriented work, Flash also provides an ActionScript (for example AS2 or AS3) which is a Flash's object-oriented scripting language for handle it.

2.3.1 ActionScript [15]

ActionScript is an object-oriented language. It is a dialect of ECMAScript, which is a superset of the syntax and semantics of the more widely known JavaScript. ActionScript is initially used for websites and software development which run on Adobe Flash Player platform. It is appeared on web pages in the form of embedded SWF (Shock Wave File). The language is open-source in the sense that its specification is provided free of charge. It also provides an open source compiler (Adobe Flex) and an open source virtual machine (Mozilla Tamarin).

Initially, ActionScript was designed to control simple 2D vector animations made in Adobe Flash. Since animation was the only main focus, early version of Flash has a limited scripting capacity so that few interactivity features were offered.

ActionScript first appeared as an object-oriented language for Macromedia's Flash authoring tool. There were limited interactive features offered in the first three versions of the tool. The term "action" was used by the early Flash developers for a simple command which they could attach to a button or a frame. The set of actions consisting of commands such as "play", "stop", "getURL", and "gotoAndPlay" was only used for basic navigation controls.

The set of actions together with new capabilities for variables, loops, conditionals, and other basic language components created a small scripting language known internally as the ActionScript in Flash

Player 4. The first version of ActionScript "ActionScript 1.0" was released in Flash Player 5. ActionScript 1.0 used ECMAScript as its prototype-based programming. It allowed full procedural programming and object-oriented programming. Functionality allowing for the creation of Web-based games and rich Internet applications streaming media were added in later versions. ActionScript 2.0 and ActionScript 3.0, the latest version, were released in Flash Player 7 and Flash Player 9 respectively.

By enhancing its capabilities in every new released of the Flash Player, ActionScript is now suitable even to use in some database applications and in basic robotics such as with the Make Controller Kit.

2.3.2 ActionScript 2.0 (AS2) [16]

As a result from user demand for a language with better equipped tool for larger and more complex application, ActionScript 2.0 was introduced with the release of Flash MX 2004 and its corresponding player, Flash Player 7 in September 2003. ActionScript 2.0 played a major role in revising the language. Its main contribution is compile-time type checking and the use of class-based syntax. To illustrate, type annotations were added to allow developers to control variables to a specific type so any type mismatch error could be found at compile-time or the use of keywords such as class and extend. Moreover, class-based inheritance syntax was introduced which allowed developers to create classes and interfaces the same way that they could do in the class-based language such as Java or C++. It is noted that although ActionScript 2.0 provided more structure approach to object-oriented programming than its prior versions, the code after compiled was in ActionScript 1.0 bytecode so it could be used with the previous version (Flash Player 6).

ActionScript 2.0 saves time by scripting something rather than animating it which makes ActionScript 2.0 more flexible to edit. Examples of data types in ActionScript 2.0 which are grouped into top level and complex data types are as follow:

ActionScript 2.0 top level data types

- **String** - A list of characters such as "Hello World".

- **Number** - Any Numeric value.
- **Boolean** - A simple binary storage that can only be "true" or "false".
- **Object** - Object is the data type all complex data types inherit from. It is used for the grouping of methods, functions, parameters, and other objects.

ActionScript 2.0 complex data types

"Complex" data types require more processor and memory and combine many "Simple" data types together. Some of these data types are:

- **MovieClip** - An ActionScript creation that allows easy usage of visible objects.
- **TextField** - A simple dynamic or input text field. Inherits the Movieclip type.
- **Button** - A simple button with 4 frames (states): Up, Over, Down and Hit. Inherits the MovieClip type.
- **Date** - Allows access to information about a specific point in time.
- **Array** - Allows linear storage of data.
- **XML** - An XML object.
- **XMLNode** - An XML node.
- **LoadVars** - A Load Variables object allows for the storing and sends of HTTP POST and HTTP GET variables.
- **Sound** - Class Sound () for control sound in Flash.
- **NetStream** - This class provides methods and properties for playing Flash Video (FLV) files from the local file system or an HTTP address.
- **NetConnection** - This class provides the means to play back streaming FLV files from a local drive or HTTP address.
- **MovieClipLoader** - This class lets you implement listener callbacks that provide status information while SWF, JPEG, GIF, and PNG files are being loaded into movie clips.

2.3.3 ActionScript 3.0 (AS3)

ActionScript 3.0 has a similar syntax to ActionScript 2.0 but a different set of APIs for creating objects. Minimal ActionScript 3.0 programs may be somewhat larger and more complicated due to the increased separation of the programming language and the Flash IDE.

2.4 Other tools

These tools are used to create backgrounds and player's characters.

2.4.1 Adobe Flash CS3

Figure 2.13: Adobe Flash CS3 Icon

Adobe Flash CS3 is one version of the Adobe Flash. It is used to create and edit flash. It is easy and flexible to use because its program interface is designed for the user friendly purpose. The current program version is **Adobe Flash CS5.5**.

2.4.2 Adobe Photoshop CS3

Figure 2.14: Adobe Photoshop CS3 Icon

Adobe Photoshop is a graphics editing program developed and published by Adobe Systems Incorporated. It is used to edit image. The current version of the program is **Adobe Photoshop CS5.5**.

2.4.3 Adobe Illustrator CS3

Figure 2.15: Adobe Illustrator CS3 Icon

Adobe Illustrator is a vector graphics editor developed and marketed by Adobe Systems. A vector graphics editor is a computer program that allows users to compose and edit vector graphics images interactively on a computer (compare with MetaPost) (MetaPost is a powerful language for producing figures for documents to be printed on PostScript printers, either directly or embedded in (La) TeX documents) [17] and save them in one of many popular vector graphics formats, such as EPS, PDF, WMF, SVG, or VML. The current program version is **Adobe Illustrator CS5.5**.

2.4.4 Corel Painter Essentials 4

Figure 2.16: Corel Painter Essentials 4 Icon

Corel Painter Essentials is a home software studio that used for sketching, painting and turning photographs into paintings. The current program version is **Corel Painter Essentials 4**

Chapter 3

Game Design

This chapter is about the game design, game system and game rule. It was classified into 8 sections.

3.1 Game description

"The Suwan Kingdom Game" is design with the following properties and rules:

- The game is a combination between Role-playing games (RPGs) and a Board game.
- The game can be played from 2-4 players on the same computer.
- The game can be saved and loaded.
- Save files can be loaded from the same computer only. It cannot be saved in one computer and then loaded from another computer.
- To win the game, the player has to gain the most assets after the Big Boss has been killed.

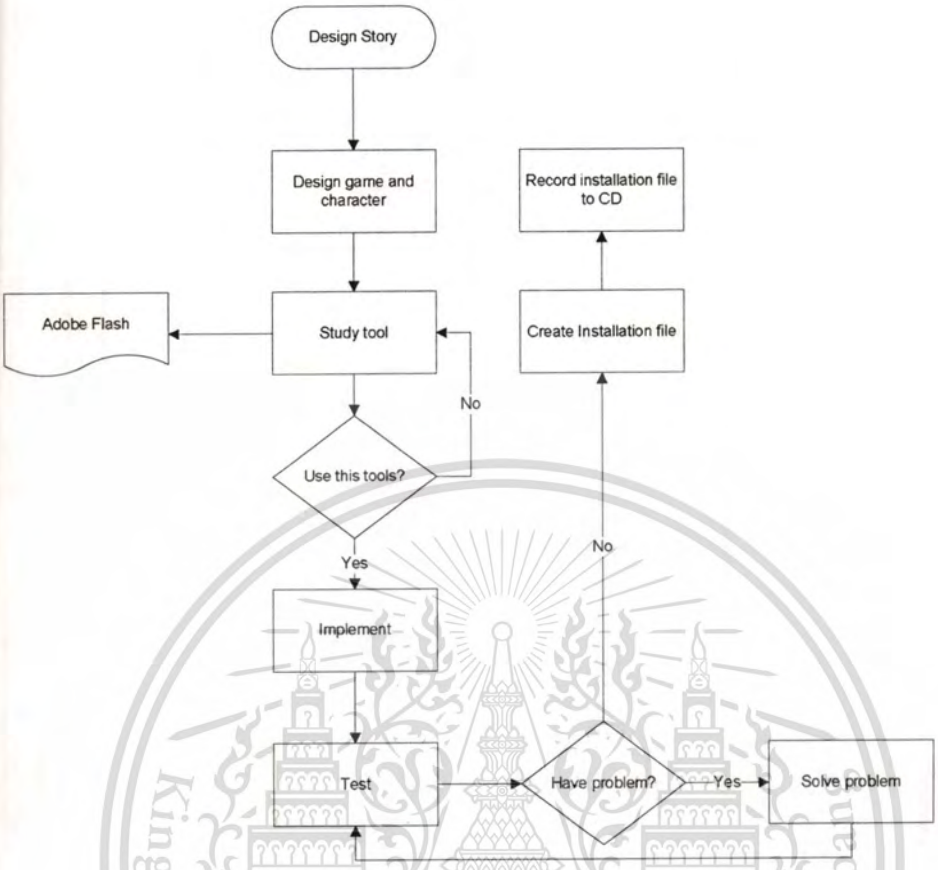


Figure 3.1: Project's Design and Implementation Process Diagram

Figure 3.1 shows a process of how to create and implement our project which started by designing game story, then designing game details and characters. Once we had selected the tools to implement, which finally we chose Adobe Flash CS 3, the implementation process was begun. Testing and problem solving were also done both together with the implementation and after the implementation was finished. When everything was ready, the installation file was created and recorded to the CD.

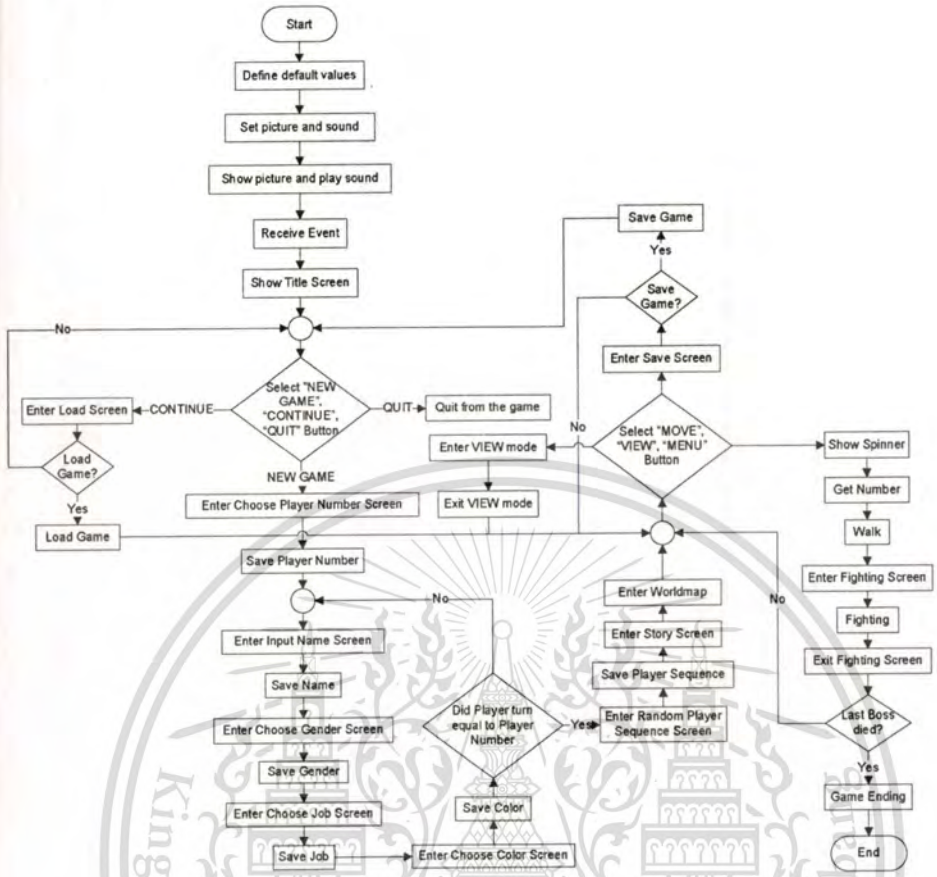


Figure 3.2: Game Working Diagram

Figure 3.2 shows all of the game working processes which will be briefly described as follows. First of all default values, pictures and sound are set and defined by the system. At this stage, players can choose to start the new game or continue playing the saved game or quit from the system as shown in Figure 3.4. If player chooses to play a new game, player has to set up their values, which are player number, names, genders, jobs and colors. The player order will be specified randomly by the system through the menu in Figure 3.17. If player choose to continue the saved data, the loaded screen as in Figure 3.5 will be shown so that player can choose which one they want to continue.

When all players finished all set-up processes, the screen as in Figure 3.18 will appear. Now, the game will begin. Player will see the map and its route which consists of several stops. For each player turn, they start their play by pressing the spinning button to obtain their walk point on the map.

In each visit that player has stops; they need to fight either with the monster or the Boss taking care of the stop, or another player who is on that stop before them. Each player keeps playing for each point until they reaches the point that there is a Big Boss there. Once, the Big Boss has been killed the game will end and the player who has the highest asset value will win. The followings are examples of screen capture from the game.

Figure 3.3: First Screen

Figure 3.4: Title Screen

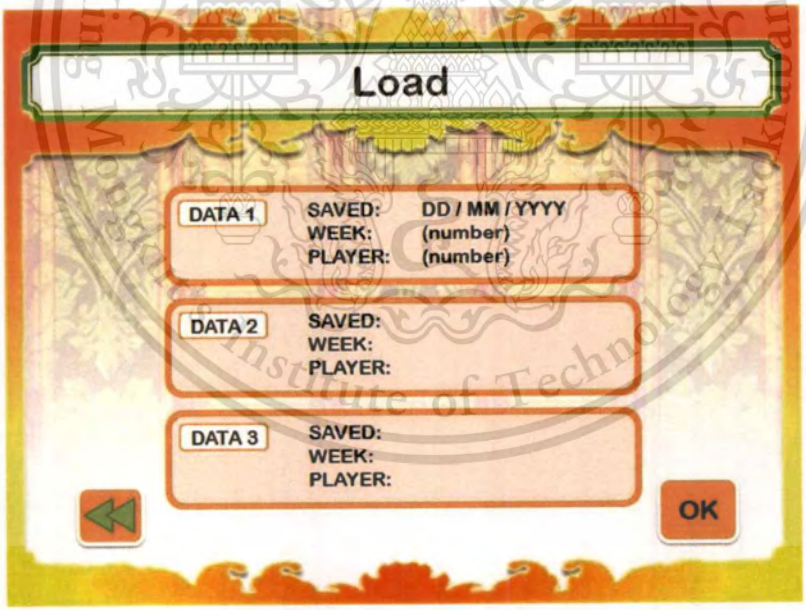


Figure 3.5: Load Screen

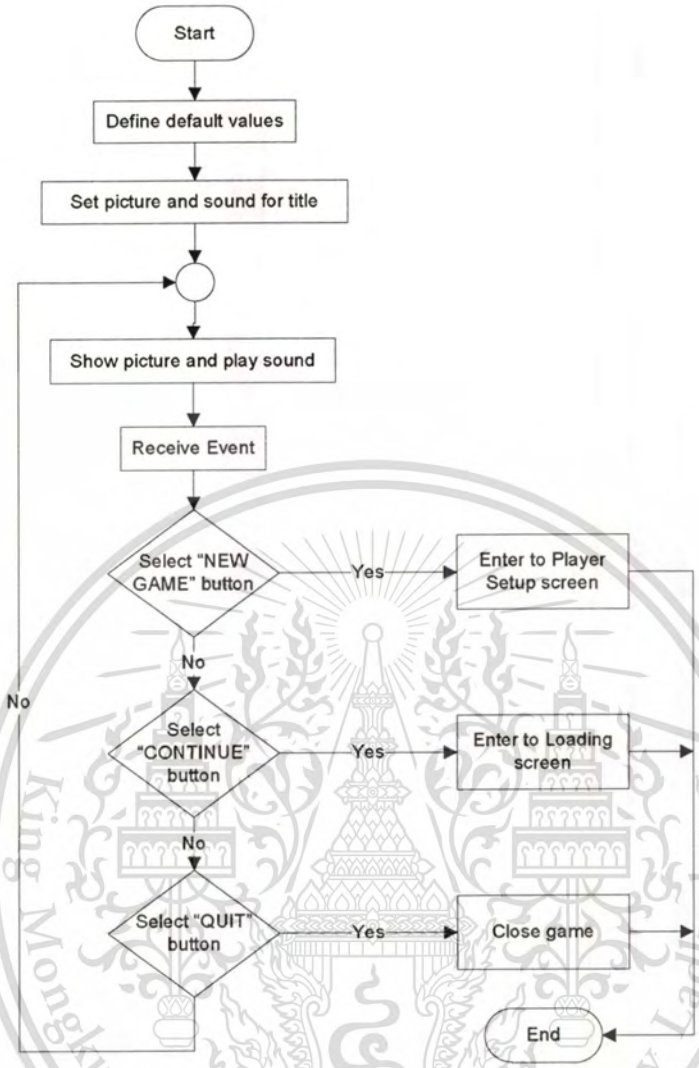


Figure 3.6: Title Screen Diagram

Figure 3.6 shows how the process is done in the Title Screen Diagram.

As previously mentioned, there are three menus in the Title Screen (Figure 3.4).

- New Game – Start a new game.
- Continue – Go to load screen and select saved data that players want to continue.
- Quit – Quit game.

Figure 3.5 shows Load Screen.

3.2 Character selection and settings

Player in this game can choose his character which he also needs to specify they name, gender, job and color. There are four jobs that each player can choose which are a magician, a swordsman, a fighter and a thief. Each character will have different skills which will be described later in the calculation part. Player also has to choose the color of his character. There are four colors to be chosen: **Red, Green, Blue** and **Yellow.** Hence, there are totally 32 characters.

The character selection is designed with the following rules.

- Players can choose to play from 2-4 players together.
- Each player can choose the same gender and job for their character but cannot choose the same name and color.
- Player order is assigned by random.

The diagrams in Figure 3.7 and Figure 3.8 show the flow of character set up process for player number, name, gender and the flow of job selection process.

Screens that involved in the setup process are shown accordingly in Figure 3.9 (Choose Player Number Screen), Figure 3.10 (Enter Name Screen), Figure 3.12 (Gender Selection Screen), Figure 3.13 (Job Selection Screen) and Figure 3.16 (Color Selection Screen). It is noted that in Color Selection Screen, players will choose color for their characters. If any color is selected, it will not allow choosing again. Players are not allowed to enter the same name as other player as shown in Figure 3.11.

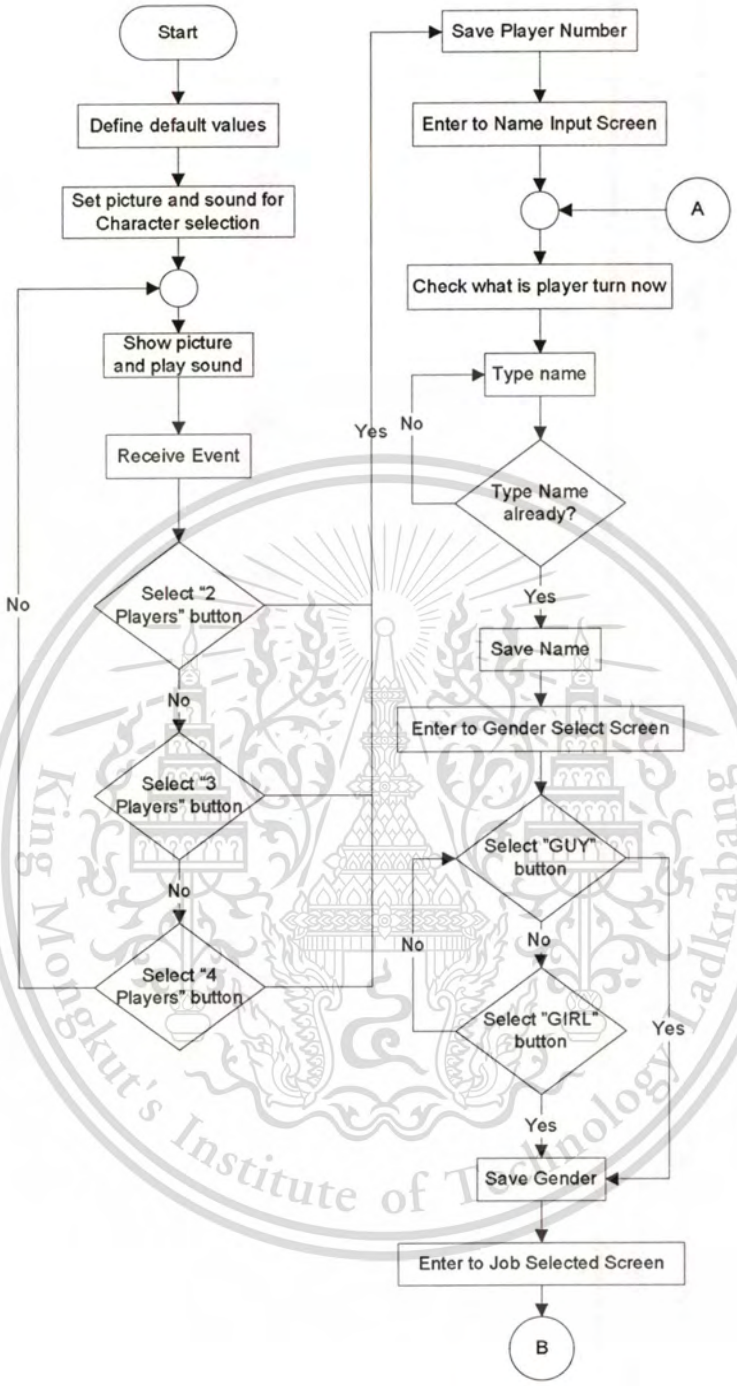


Figure 3.7: Job Selected Diagram A

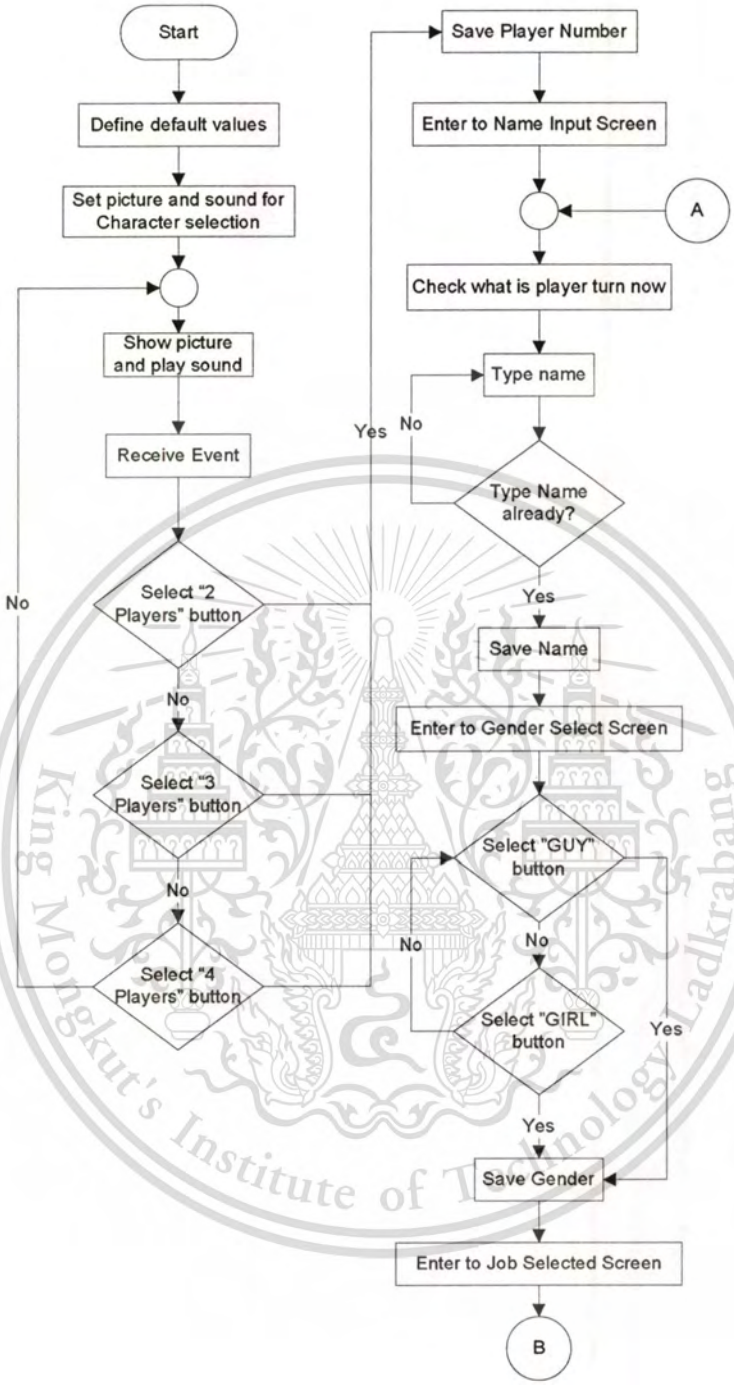


Figure 3.8: Job Selected Diagram B

Figure 3.9: Choose Player Number Screen

Figure 3.10: Enter Name Screen

Figure 3.11: Enter Duplicate Name Screen

Figure 3.12: Gender Selection Screen

Figure 3.13: Job Selection Screen

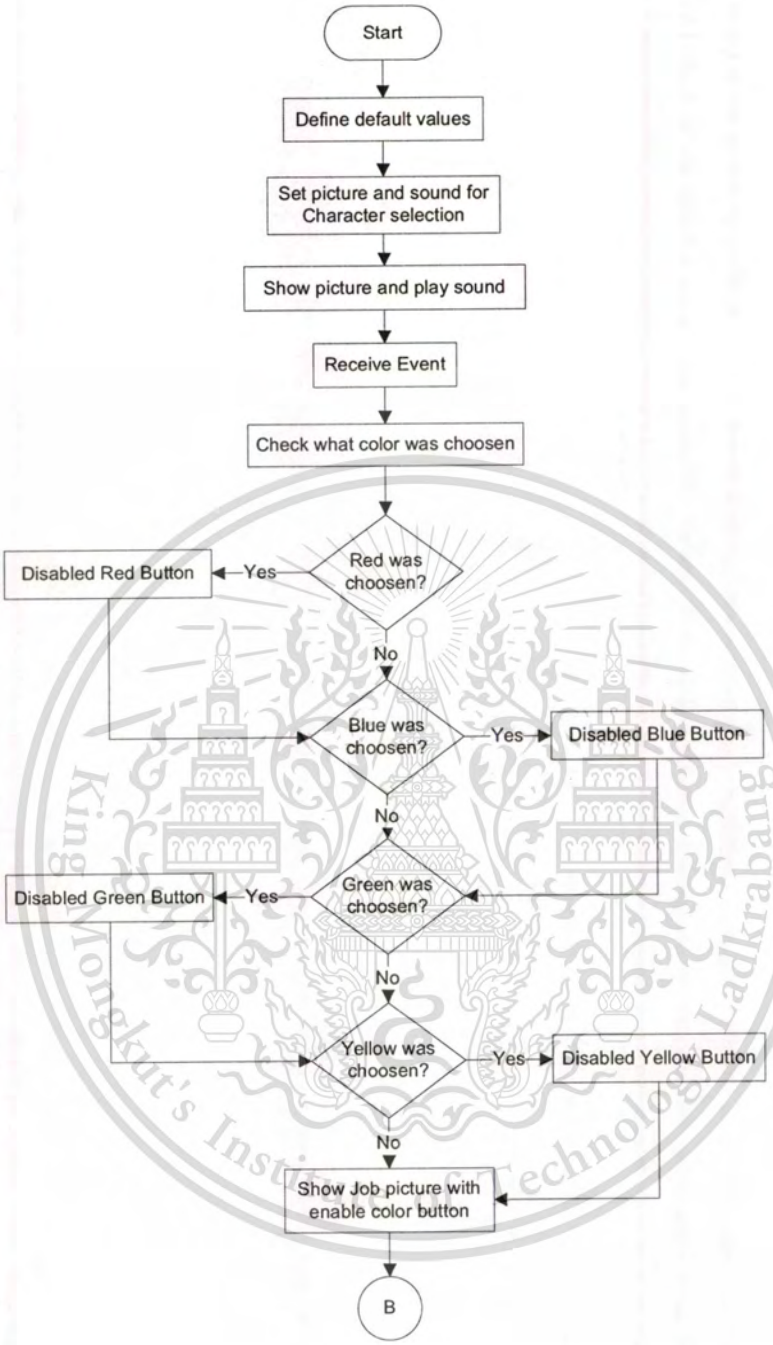


Figure 3.14: Color Selection Screen Diagram

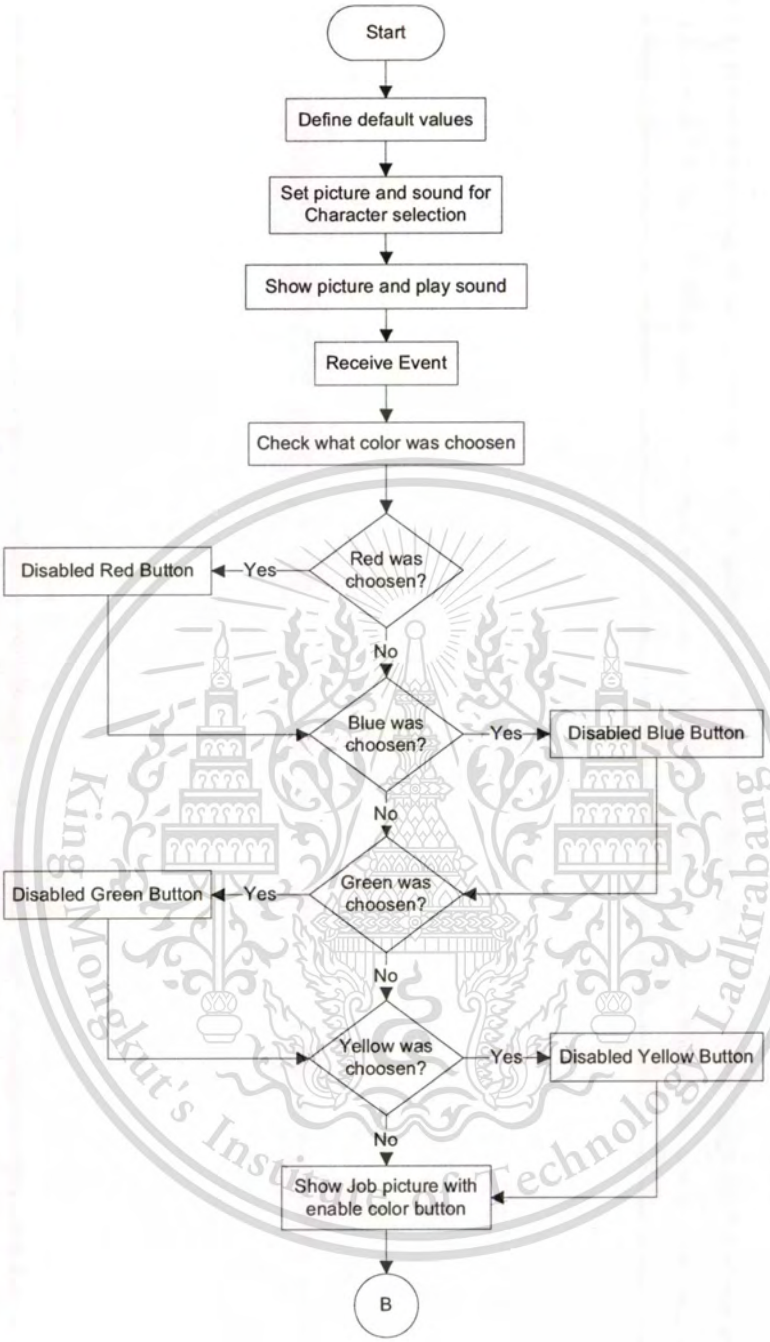


Figure 3.15: Color Selection Screen Diagram (Cont.)

Figure 3.16: Color Selection Screen

Figure 3.17: Random Screen

Figure 3.18: Start Screen

3.3 World map

World map is the map that the player will walk on. There are three types of points on the map.

- Monster point, which is an empty point on the map.
- Boss points, which are represented by the purple star on the point. Each point represents town owned by each Boss.
- Big Boss point, which are represented by the black star on the point. There is only one point and it will be visible only when all Bosses are dead.

World map and how players can walk on the map are designed with these rules.

- Game allows the player to move from point to point on the map.
- Game uses a spinner to randomly obtaining moving number for each player.
- The player can select "VIEW" button to see Boss level (LV), status and hit points (HP) (Figure 3.26).
- The player can select "MENU" button to select save slot (Figure 3.27).
- When the player walks on to any point, they must fight with either monster, Boss or Big Boss on that particular point.
- When the player happens to walk into the point which there is another player standing there, there are two cases:
    - If the first player is not fighting with the monster, the second player will fight with the first player.
    - If the first player is still fighting with the monster, the next player must choose between the monster and the first player to fight with.
- The player is allowed to give up during the fight but he needs to pay the price by stop moving for one turn.
- The player is not allowed to give up during the fight with the Boss or the Big Boss.
- If the player loses during the fight, he will die and will be moved to the castle which after stop moving for two turns, he will revive (start playing from the beginning point again).

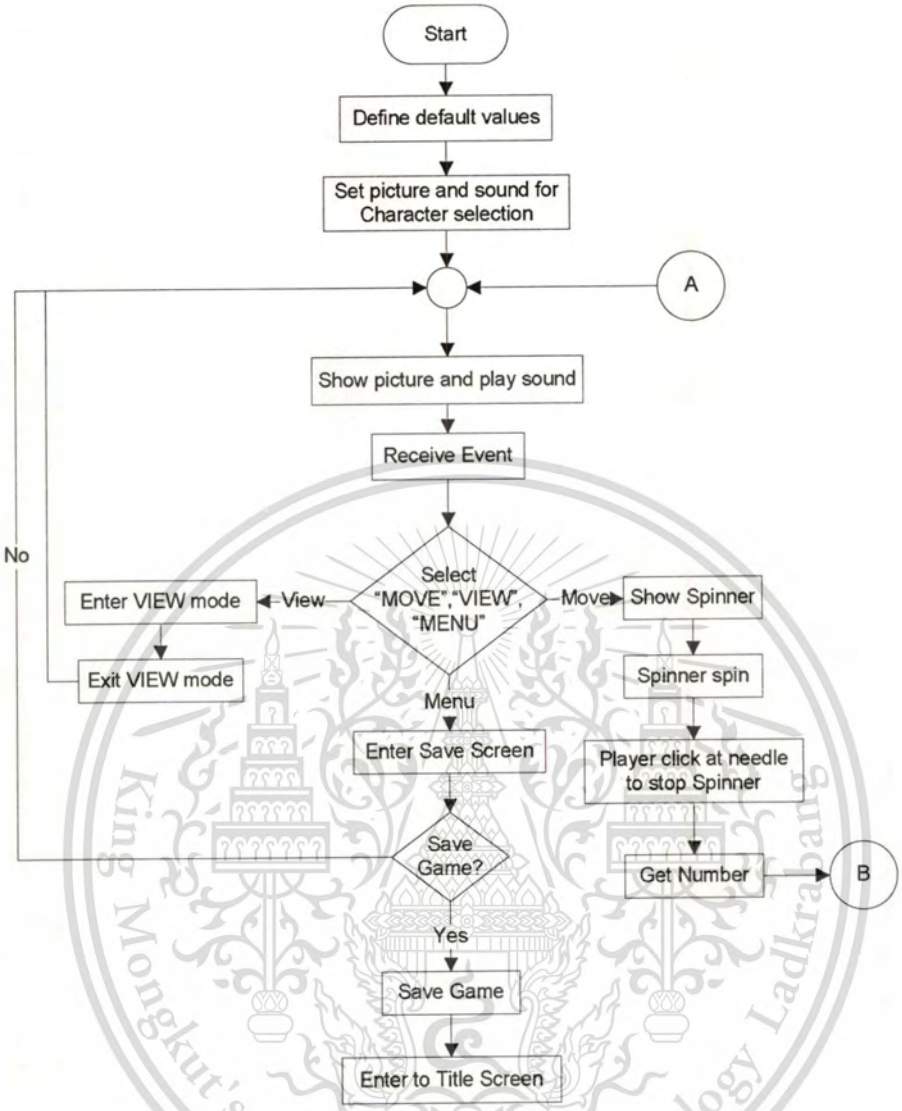


Figure 3.19: Field Explanation Diagram A

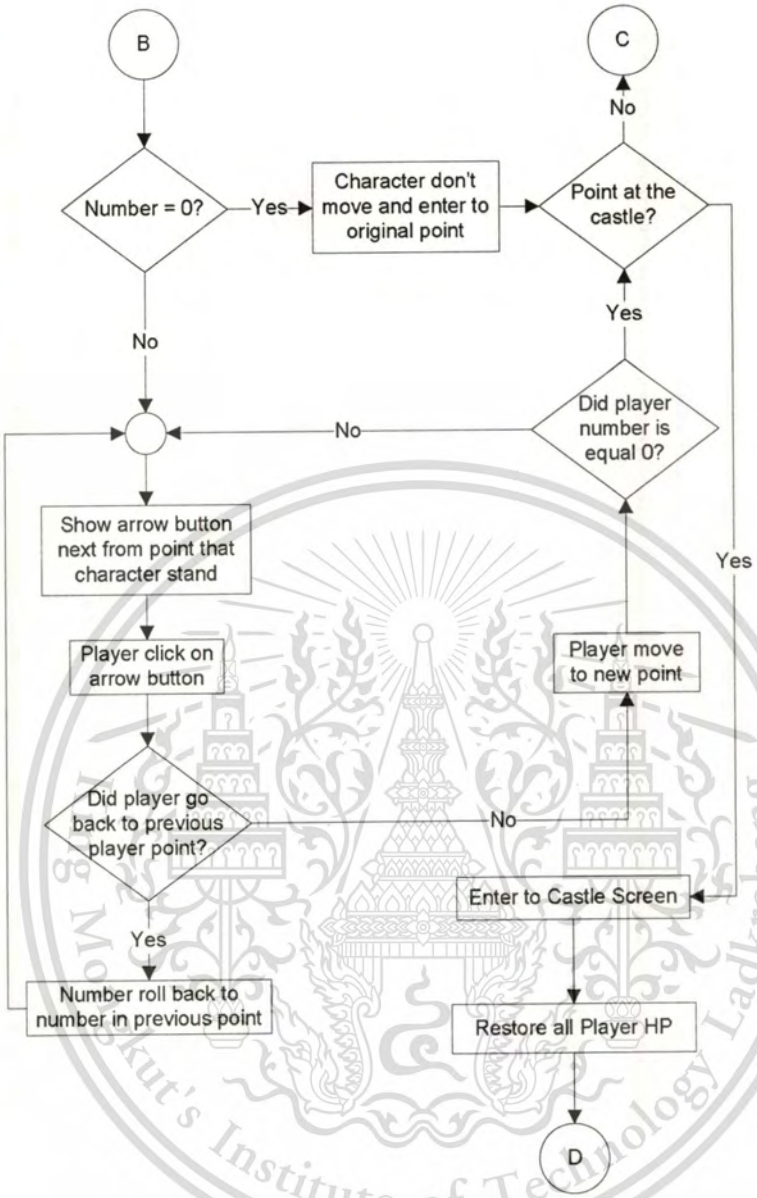


Figure 3.20: Field Explanation Diagram B

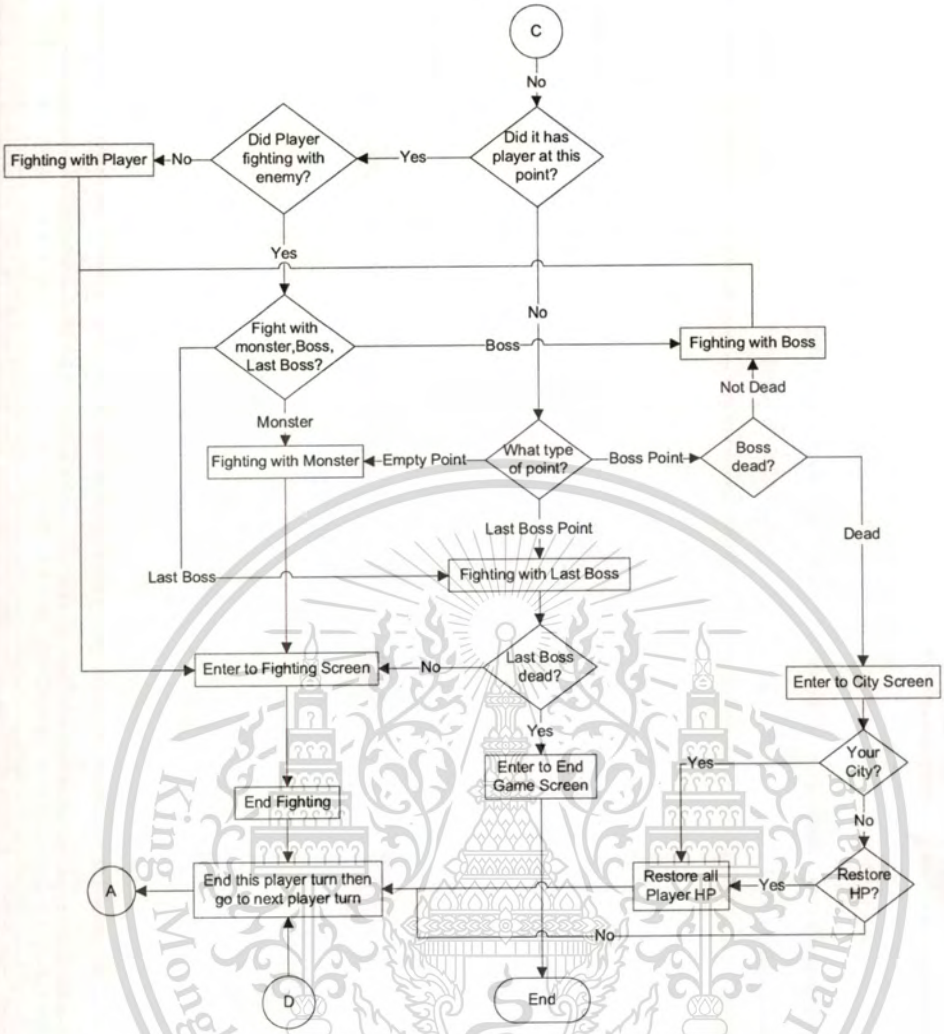


Figure 3.21: Field Explanation Diagram C

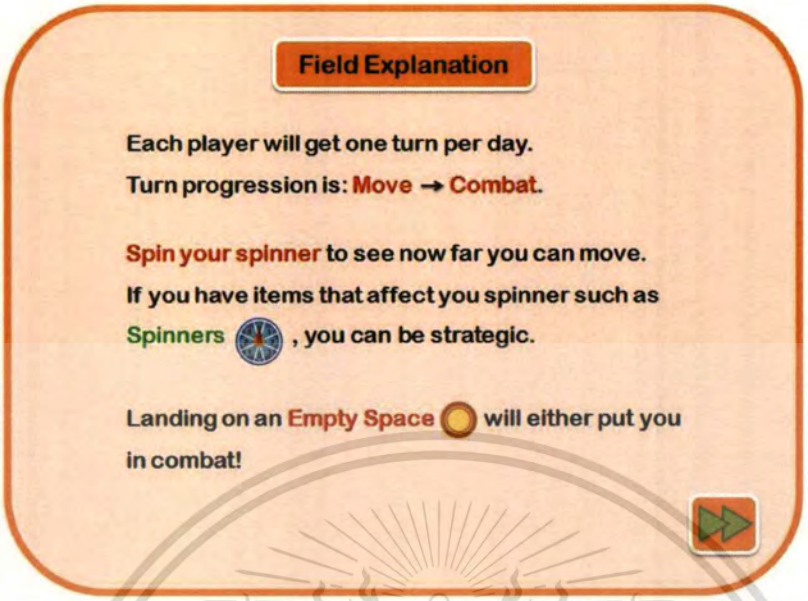


Figure 3.22: Field Explanation Screen

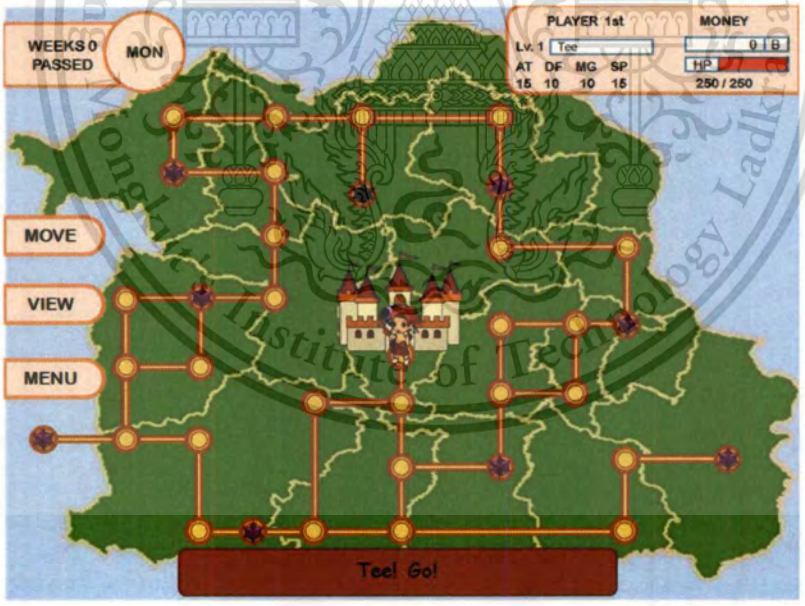


Figure 3.23: World Map Starting Point Screen

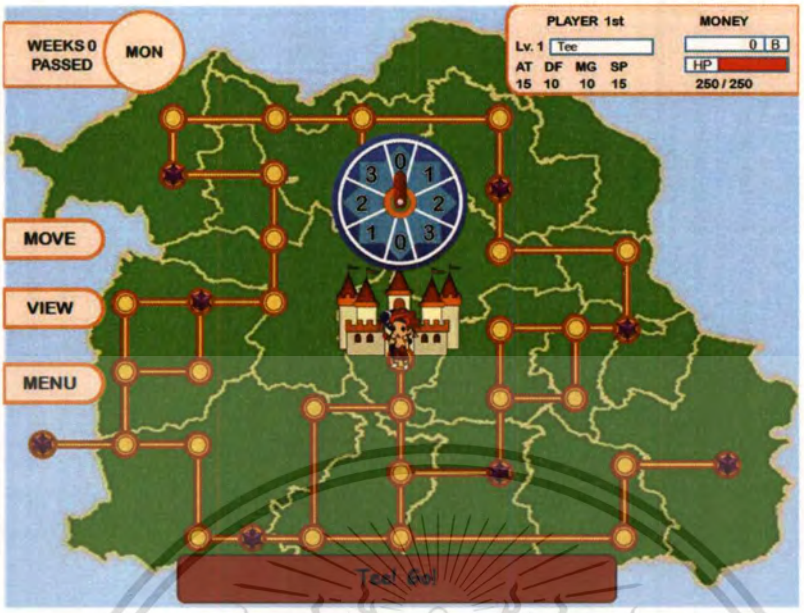


Figure 3.24: Spinner Screen

Figure 3.25: After Spin Spinner Screen

Figure 3.26: View Screen

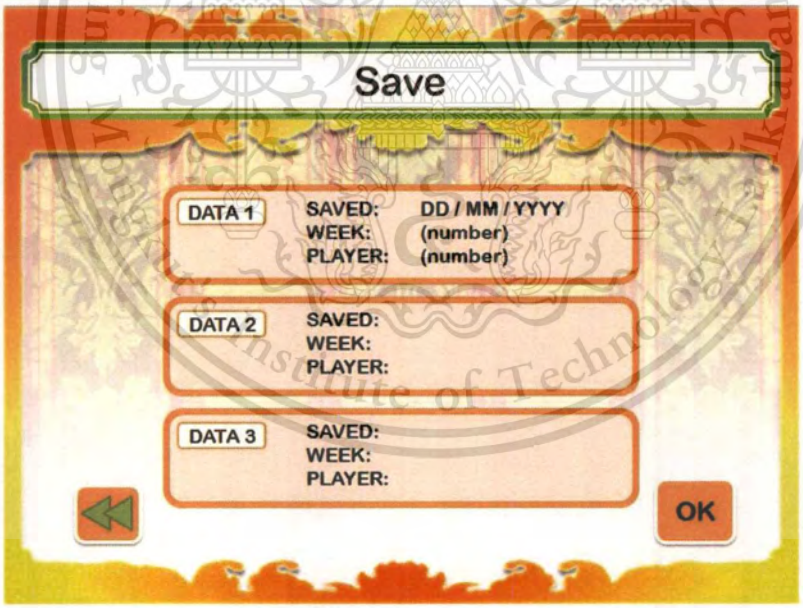


Figure 3.27: Save Screen

Figure 3.19-3.21 show process of the field exploration. Figure 3.22-3.25 show Field Explanation Screen, World Map which the starting point is at the castle, Spinner Screen and the screen after spinning respectively.

3.4 Fighting system

Once the player enters each point, the fighting begins.

Fighting is designed with the following rules:

- There are two sides, the attacker (who hits first) and the defender.
- To get experience for level up, each player must fight with a monster or a Boss.
- The player can also fight with another player.
- If the defender chooses to correct defend, damage value will decreased.
- If the attacker chooses strike attack and the defender chooses counter then damage will return to attacker.

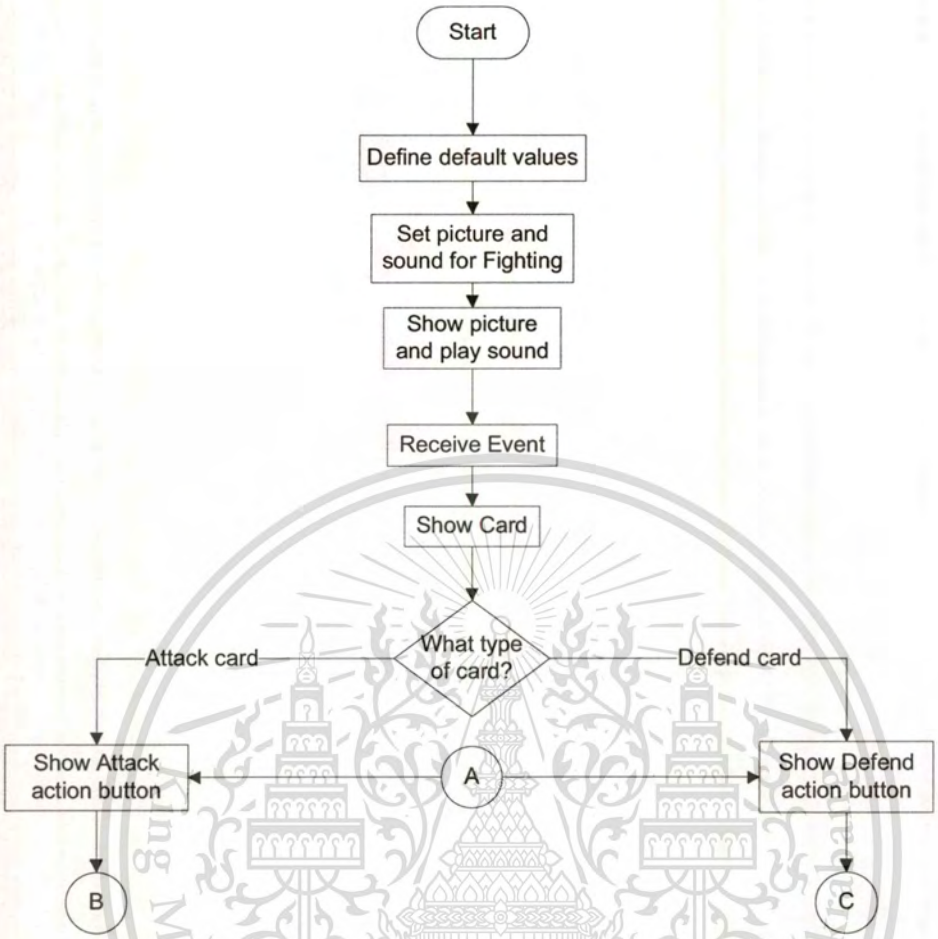


Figure 3.28: Fighting Diagram A

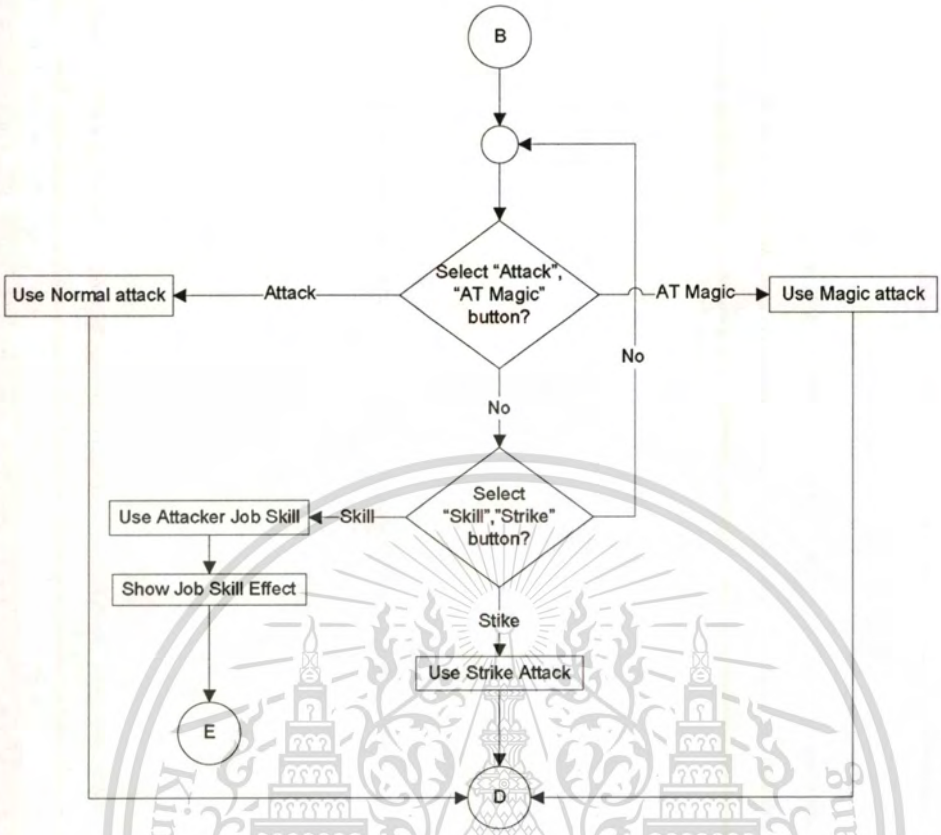


Figure 3.29: Fighting Diagram B

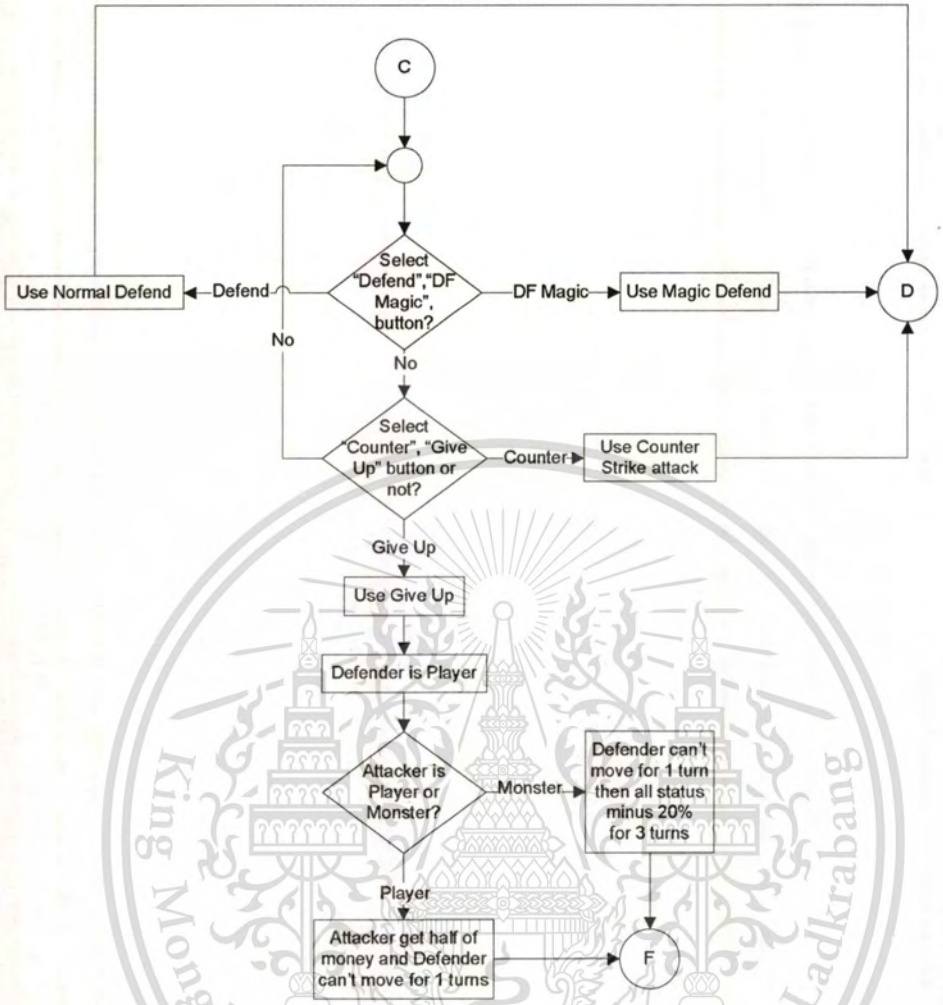


Figure 3.30: Fighting Diagram C

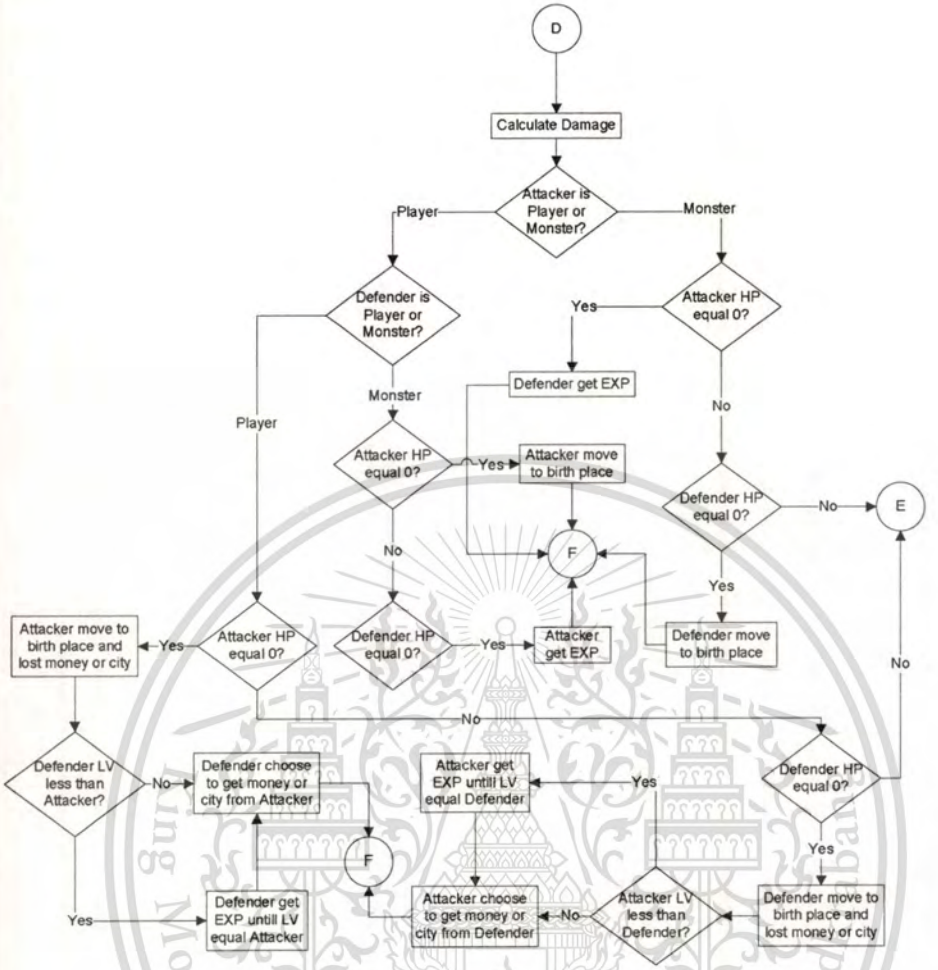


Figure 3.31: Fighting Diagram D

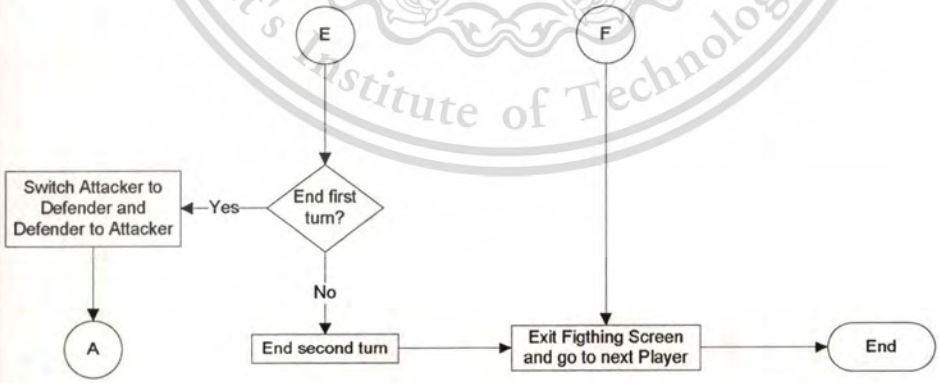


Figure 3.32: Fighting Diagram E, F

Figure 3.33: Choosing Card Screen

Figure 3.34: After Choosing Card Screen

Figure 3.35: Attacker Screen

Figure 3.36: Defender Screen

Figure 3.37: Winner Screen

Figure 3.38: Loser Screen

Figure 3.28-3.32 shows all processes in fighting screen.

Fighting mode's steps

There are 5 steps in fighting mode.

1. The player must choose a card first (Figure 3.33). There are two types of cards: an attacking card and a defending card. If the player gets the attacking card, the player will attack first (he will be an attacker). If the player gets the defending card, the player will defend first (he will be a defender) (Figure 3.34).

2. Attack button will appear to the attacker and Defend button will appear to the defender. There are 4 actions, which are Attack, At Magic, Strike and Skill in the attacker side (Figure 3.35). Similarly, there are 4 actions, which are Defend, Df Magic, Counter and Give up in the defender side (Figure 3.36). The attacker will choose one action from the attack button and the defender will choose one action from the defend button.

3. After both attacker and defender choosing their cards, the actions will be executed and damage to each of them will calculate. If no one dies or gives up then they will switch their roles that is from the attacker to the defender and vice versa.

4. Repeat step 2

5. After either of the defender and the attacker executing his roles, if no one is dead or gives up, this turn ends and goes to next player's turn.

What will happen after the fight in each turn ends?

There are 8 possible ways: five for between the player and the monster and another three for between two players. The first five are as follows:

1. If player HP = 0 and monster HP > 0 then the player is dead and moved to the birth place (castle) and the monster will disappear.

2. If player HP > 0 and monster HP = 0 then the player gains experience (EXP.) and money.

3. If player HP > 0 and monster HP > 0, they will continue fighting in the next player turn.

4. If the player gives up and monster HP > 0 then the player will be barred to walk for one turn.

5. If player HP > 0 and Boss/Big Boss HP = 0 then the player gains EXP., money, and Boss/Big Boss city.

In case that there are more than one player in each stop and they fight with each other.

6. If the first player (Player 1) HP > 0 and the second player (Player 2) HP = 0 then the first player gains EXP until LV equals to the second player. The first player can choose money or city from the second player. If LV of the first player is much more than the second player, the first player will not gain EXP., but still can choose money or city from the second player.

7. If Player 1 HP > 0 and Player 2 > 0 then continue fighting in Player 2 turn.

8. If Player 1 HP > 0 and Player 2 chooses to give up then Player 2 cannot walk for 1 turn.

3.5 Character system

This section will give description and specification of level, experience and skill of each character including monsters, Bosses and the Big Boss.

Experience (EXP.) is a point that each player requires to increase his level (level up). Experience can be gained by killing monster/Boss or killing another player whose level is higher than the attacker. There are 25 levels (LV). The higher level will require more experience than the lower one.

3.5.1 Experience Table

Levels	Experience
2	10
3	20
4	39
5	74
6	137
7	247
8	432
9	734
10	1211
11	1938
12	3004
13	4506
14	6534
15	9148
16	12350
17	16055
18	20069

Table 3.1: Experience Table

Levels	Experience
19	24083
20	27696
21	30466
22	31989
23	33269
24	34267
25	34952

* When LV is up, the player gets 5 status points and restores all HP.

3.5.2 Beginning Character Status

- Swordsman = AT 25 / DF 15 / MG 5 / SP 5
- Magician = AT 5 / DF 10 / MG 25 / SP 10
- Fighter = AT 15 / DF 10 / MG 10 / SP 15
- Thief = AT 10 / DF 10 / MG 5 / SP 25
- AT is Attack. It is used to calculate hit points (HP), attack and strike attack points.
- MG is Magic. It is used to calculate magic attack and magic defend points.
- DF is Defend. It is used to calculate defend point.
- SP is Speed. It is used to calculate Percent Dodge. Percent Dodge is a percent that the defender can dodge attack from the attacker.

3.5.3 Character Skill

Skill is used to increase the player status based on his job and LV. Skill can be used in fighting screen when player is an attacker. Skill is calculated and its effect is shown after the player whose uses skill has played for two turns. After the two turns, skill effect will disappear. Skill of Boss or Big Boss will be used randomly.

3.5.3.1 Swordsman Skill

- LV 1: AT × 1.25 (AT UP I)
- LV 10: AT × 1.5(AT UP II)
- LV 20: AT × 2.0(AT UP III)

* AT UP means AT increase based on LV Skill.

3.5.3.2 Magician Skill

- LV 1: MG × 1.25(MG UP I)
- LV 10: MG × 1.5(MG UP II)
- LV 20: MG × 2.0(MG UP III)

* MG UP means MG increase based on LV Skill.

3.5.3.3 Fighter Skill

- LV 1: AT × 1.1, SP × 1.1 (ATSP UP I)
- LV 10: AT × 1.25, SP × 1.25 (ATSP UP II)
- LV 20: AT × 1.5, SP × 1.5 (ATSP UP III)

* ATSP UP means AT and SP increase based on LV Skill.

3.5.3.4 Thief Skill

- LV 1: SP × 1.25 (SP UP I)
- LV 10: SP × 1.5 (SP UP II)
- LV 20: SP × 2.0 (SP UP III)

* SP UP means SP increase based on LV Skill.

3.6 Calculation

All calculations in this game are calculated from all of the following formulas. Let HP denote Hit points of character.

- HP = (LV × 100) + (AT × 10).
- Attack = (AT × (Random Number (16-20))) + AT × Random Number (0 − 4)
- Strike Attack = Attack × 1.5
- Counter Strike Attack = Strike Attack × 1.5
    - (Damage return to Attacker who uses Strike Attack)
- DEF = (DF × (Random Number (11 − 15))) + DF × Random Number (0 − 4)
- Magic Attack = (MG × (Random Number (16 − 20))) + MG × Random Number (0 − 4)
- Magic Defend = (MG × (Random Number (11 − 15))) + MG × Random Number (0 − 4)
- Percent Dodge = (SP (Attack) ÷ SP (Defend)) × 100
- If Percent dodge is higher than 100, it will calculate at 95.
- Miss is an effect that will appear when the attacker performs his attack but misses. Miss is calculated by using a random number (0-100). If the random number is in percent dodge, the attacker attack will be missed.

3.6.1 Normal ATK

DamageATK = Attack (Attacker) – DEF (Defender)

- Attack -> Defend = DamageATK
- Attack -> Counter = DamageATK × 1.25
- Attack -> DF Magic = DamageATK × 1.25
- Strike -> Defend = Strike Attack
- Strike -> Counter = Counter Strike Attack
- (Damage is returned to the Attacker)
- Strike -> DF Magic = Strike Attack

3.6.2 Magic ATK

Magic ATK = Magic Attack – Magic Defend

- At Magic -> Defend = Magic ATK × 1.25
- At Magic -> Counter = Magic ATK × 1.25
- At Magic -> DF Magic = Magic ATK

3.7 Monster

The monster is an enemy in the empty point. If the monster is an attacker, the monster can use Attack, At Magic and Strike. If the monster is a defender, the monster can use Defend, Df Magic and Counter. If the player can defeat the monster, the player will get EXP. and money. If the player is defeated, he will be moved to the castle and cannot move for 2 turns. If the player uses give up, he cannot move for 1 turn.

3.7.1 Monsters details

Name/LV.	Status(AT/MG/DF/SP)	EXP.	Money
 Slime LV.1	15/10/15/10	5	10
 Orange Mushroom LV.2	20/15/10/10	6	12
 Blue Mushroom LV.3	15/20/15/10	10	20
 Poison Slime LV.4	20/10/25/10	14	28
 Wild Mushroom LV.5	10/30/10/20	22	44

Table 3.2: Monsters Details Table

Name/LV.	Status(AT/MG/DF/SP)	EXP.	Money
 Moss Snail LV.7	20/10/15/30	54	108
 Senile Snail LV.7	30/5/20/20	54	108
 Lichen Mushroom LV.8	30/10/30/10	81	162
 Poison Squid LV.9	5/40/25/15	121	242
 Spine Squid LV.9	25/5/20/35	121	242

Name/LV.	Status(AT/MG/DF/SP)	EXP.	Money
 Giant Clam LV.10	10/30/40/10	176	352
 Happy Bear LV.10	30/10/30/20	176	352
 Happy Lion LV.11	10/40/35/10	250	500
 Zombie Mushroom LV.11	40/5/15/35	250	500
 Rainbow Owl LV.12	40/5/20/35	345	690

Name/LV.	Status(AT/MG/DF/SP)	EXP.	Money
 Moody Lion LV.13	40/15/35/15	466	932
 Angry Lion LV.13	40/15/35/15	466	932
 Angry Bear LV.14	5/45/5/55	609	1,218
 Circus Bear LV.15	50/15/35/15	771	1,542

Name/LV.	Status(AT/MG/DF/SP)	EXP.	Money
 Circus Lion LV.15	50/15/10/40	771	1,542
 Furious Owl LV.16	5/50/40/25	944	1,888
 Mystic Log LV.16	50/15/25/30	944	1,888
 Yellow Troll LV.17	35/35/25/30	1,114	2,228

Name/LV.	Status(AT/MG/DF/SP)	EXP.	Money
 Green Troll LV.18	35/25/35/35	1,384	2,768
 Blue Troll LV.19	35/35/30/35	2,465	4,930

3.7.2 Boss details

Boss is an enemy in the purple star point. Boss has higher status than the monster and can use skill. The players cannot give up when fighting with Boss. If Boss is an attacker, Boss can use Attack, At Magic, Strike and Skill. If Boss is a defender, Boss can use Defend, Df Magic and Counter. If the player can defeat Boss, he will get EXP., money and Town that Boss occupies. If the player is defeated, he will move to the castle and cannot move for 2 turns.

Name/LV.	Status(AT/MG/DF/SP)	EXP	Money	Skill
 Spiny Mushroom LV.10	20/60/35/20	606	1,818	MG UP I
 King Ramos LV.11	50/30/50/13	969	2,907	DF UP I
 Big Toad LV.13	60/18/60/20	2,253	6,759	AT UP I
 Alligator LV.14	20/75/20/50	3,267	9,801	MG UP II

Table 3.3: Boss Details Table

Name/LV.	Status(AT/MG/DF/SP)	EXP	Money	Skill
 Zombie Monk LV.15	50/50/50/23	4,574	13,722	SP UP I
 Leviathan LV.16	70/30/30/50	6,175	18,525	AT UP II
 Horror Tree LV.18	90/40/20/25	10,035	30,105	DF UP II

Name/LV.	Status(AT/MG/DF/SP)	EXP	Money	Skill
 Jasper Horse LV.19	50/50/33/70	12,042	36,126	SP UP II

3.7.3 Big Boss details

Big Boss is an enemy in the black star point. Big Boss has higher status than Boss and can use skill. The players cannot give up when fighting with Big Boss. If Big Boss is an attacker, Big Boss can use Attack, At Magic, Strike and Skill. If Big Boss is a defender, Big Boss can use Defend, Df Magic and Counter. If the player can defeat Big Boss, he will get EXP., money and Town that Big Boss occupies. If the player is defeated, the player will moves to the castle and cannot move for 2 turns.

Name/LV.	Status(AT/MG/DF/SP)	EXP	Money	Skill
 Asura LV.20	90/70/70/50	28,000	140,000	All Status UP (All Status × 1.5)

Table 3.4: Big Boss Details Table

3.7.4 Location of monsters, Bosses and Big Boss on the map plus their levels.

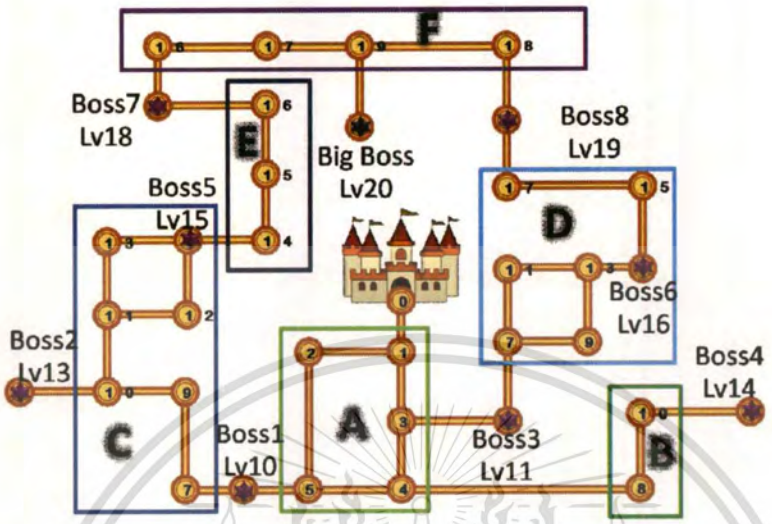


Figure 3.39: Point of Monster, Boss and Big Boss

Figure 3.39 show points and monster level, Boss level and Big Boss level.

3.8 Other game rules

- Money can be taken from the monster, the player or Boss.
- Town can be taken by killing Boss or killing the player who owns that town.
- The player who has the highest asset value is the winner.
- Asset is calculated from money plus town value.
- Every city has its unique value.
- When the player visits a city that belongs to them, he can restore his HP without paying money.
- When the player visits a city that does not belong to him, the player will choose to restore all HP by paying money (which is calculate from LV*100) or doing nothing.

3.9 Suwan Kingdom Game

Each game design page is kept in a different frame. Our game consists of 2 files. The first file is the main game which contains 17 frames listed below:

- Frame 1: intro
- Frame 2: menu
- Frame 3: loadGame
- Frame 4: numPlayer
- Frame 5: character
- Frame 6: BeforeRandomScreen4
- Frame 7: BeforeRandomScreen3
- Frame 8: BeforeRandomScreen2
- Frame 9: AfterRandomScreen4
- Frame 10: AfterRandomScreen3
- Frame 11: AfterRandomScreen2
- Frame 12: startGame
- Frame 13: story
- Frame 14: worldMap
    - Frame 14.1 :Spinner
- Frame 15: continue
- Frame 16: endGuy
- Frame 17: endGirl

The second file is the Fighting file.

3.9.1 Components of each frame

Figure 3.40: Suwan Kingdom Game (Frame 1: intro)

Start Frame codes are shown in Figure 3.41 and Figure 3.42.The main components are explained as follows.

```
var numPlayer:Number = 0;
var nameChar:Array = new Array();
var genChar:Array = new Array();
var jobChar:Array = new Array();
var colorChar:Array = new Array();
var noChar:Array = new Array();
numMem = 0;
// PointX,Y In After Random//
PX1 = 322;
PY1 = 243;
PX2 = 322;
PY2 = 384;
PX3 = 559;
PY3 = 243;
PX4 = 559;
PY4 = 384;

function NextFrame(){
    switch(numPlayer)
    {
    case 2:
    gotoAndPlay("AfterRandomScreen2");
    break;
    case 3:
    gotoAndPlay("AfterRandomScreen3");
    break;
    case 4:
    gotoAndPlay("AfterRandomScreen4");
    break;
    }
}
```

Figure 3.41: Suwan Kingdom Game (Frame 1: intro Code 1)

```
start_btn.onPress = function(){
    gotoAndStop("menu");

}
function getDate(){
    var date:Date = new Date();
    var day = date.getDate();
    var month = date.getMonth();
    var strM = "";
    if(month < 10) strM = "0" + month;
    else strM = month;

    var year = date.getFullYear();
    var str = day + "/" + strM + "/" + year;
    return str;
}
```

Figure 3.42: Suwan Kingdom Game (Frame 1: intro Code 2)

- var numPlayer:Number = 0
  - Create a variable named *numPlayer* with the initial value set to 0. This variable is used to store the player number.
- var nameChar:Array = new Array()
  - Create a new array with the name *nameChar*. Another four new arrays are created in a similar way. They are *genChar, jobChar, colorChar* and *noChar.*
- numMem = 0
  - Create a variable named *numMem* and set its initial value to 0.
- PX1 = 322
  - Assign the value (321) to the variable *PX1*.
  - *PY1, PX2, PY2, PX3, PY3, PX4, PY4* ***values*** are assigned in the same way.
- function NextFrame()
  - Create the new function named *NextFrame()*. Depending on the number of the players chosen, the function will call that number of players' frame.
- var date:Date = new Date()
  - Create a variable named *date* and also create a new class *Date.*
- var day = date.getDate()
  - Create a variable named *day* in which its initial value is obtained from *date.getDate()*. *date.getDate()* retrieves the current day value from the computer. Similarly, the program obtains month and year values from *var month = date.getMonth() and var year = date.getFullYear()* respectively.
- return str
  - The NextFrame() function will return the string name *str* which is the day-month-year value to whoever that calls it.

Figure 3.43: Suwan Kingdom Game (Frame2: menu)

Figure 3.44: Suwan Kingdom Game (Create Button 1)

Figure 3.43 shows some buttons in the game. To create a button, follow these steps: click Insert -> New symbol (Figure 3.44 appears after "New symbol" has been clicked) -> Choose "Button" -> Click "OK".

```
1  new_btn.onPress = function(){
2      gotoAndStop("numPlayer");
3  }
4  quit_btn.onPress = function(){
5      fscommand("quit","true");
6  }
7  con_btn.onPress = function(){
8      gotoAndStop("loadGame");
9
10 }
```

Figure 3.45: Suwan Kingdom Game (Create Button 2)

To add an action to the button, press F9 to display the Action-Button page. Figure 3.45 shows codes for buttons in Figure 3.43. Some of the codes are explained below:

- new_btn.onPress = function ()
  - When the button *new_btn* was pressed, function will be called to execute.
- gotoAndStop("numPlayer")
  - Go to the frame that has name according to what is assigned inside the brackets. For example, the game will go to the frame that its name property is *numPlayer* after *gotoAndStop("numPlayer")* is executed.
- fscommand("quit","true")
  - Change the *quit status* from false to true which will quit the game.

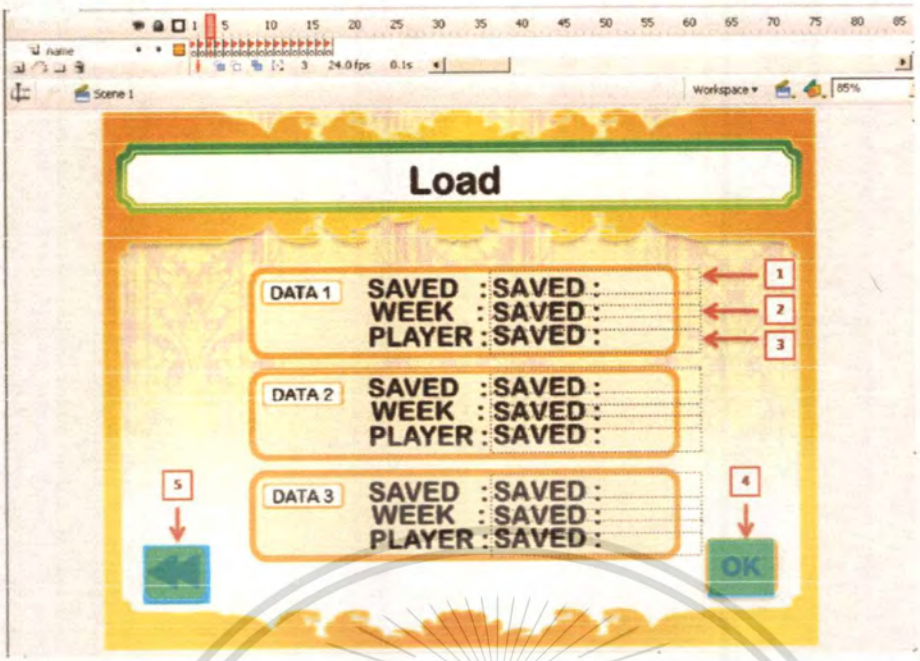


Figure 3.46: Suwan Kingdom Game (Frame 3: loadGame)

This frame consists of three types of buttons: Data button, OK button (Number 4) and previous button (Number 5). There are three components (Number 1, 2 and 3 in Figure 3.46) in the Data button.

- Number 1: Save Date shows what date was saved?
- Number 2: Weeks shows the number of game weeks that the players have played the game so far.
- Number 3: Player Number shows the number of players playing in this saved slot.
- Number 4: OK Button is clicked after the players have decided which saved data they will continue playing which the program will load the chosen saved data.
- Number 5: Back Button is clicked when the player wants to go back to main menu.

```
var select = 0;
var nameData = "";
data1.onPress = function(){
    this.gotoAndStop(2);
    data2.gotoAndStop(1);
    data3.gotoAndStop(1);
    select = 1;
}
data2.onPress = function(){
    this.gotoAndStop(2);
    data1.gotoAndStop(1);
    data3.gotoAndStop(1);
    select = 2;
}
data3.onPress = function(){
    this.gotoAndStop(2);
    data2.gotoAndStop(1);
    data1.gotoAndStop(1);
    select = 3;
}
ok_btn.onPress = function(){
    if(select == 3 && data3.week_txt.text != ""){
            nameData = "slot3";
            gotoAndStop("continue");
    }
    else if(select == 2 && data2.week_txt.text != ""){
            nameData = "slot2";
            gotoAndStop("continue");
    }
    else if(select == 1 && data1.week_txt.text != ""){
            nameData = "slot1";
            gotoAndStop("continue");
```

Figure 3.47: Suwan Kingdom Game (Frame 3: loadGame Code)

```
on(press){
    gotoAndStop(2);
}
```

Figure 3.48: Suwan Kingdom Game (Back Button on Frame 3: loadGame)

This frame consists of the following variables and functions:

- var select = 0
    - Create a variable named *select* which is used to keep the saved data the player chooses and sets its initial value to 0.
- var namedata = “” means

- Create a variable named *data* and set its initial value to blank (no data in it).

- data1.onPress = function()
  - When button of *data1* was pressed the function *data1.onPress* will be called to execute.
  - this.gotoAndStop(2)
    - Go to frame 2 inside data1 button to make the frame appear around save slot.
  - data2.gotoAndStop(1)
    - Go to frame 1 inside data2 button.
  - data3.gotoAndStop(1)
    - Go to frame 1 inside data3 button.
  - select = 1
    - Change *select* value to 1.
- Similarly, depending on the button being pressed, the program will go to either function *data2.onPress = function() or* function *data3.onPress = function()* in which each will go to its frame and set which data set it will use (*select*).
- ok_btn.onPress = function()
  - The value of *select* represents different kinds of condition (select == 1,2,3) in function *ok_btn.onPress*. That is if *select* is :
    - "1" means the players choose save slot1 to continue playing or
    - "2" means the players choose save slot2 to continue playing or
    - "3" means the players choose save slot3 to continue playing.
  - Depending on the *select* value, the program will assign the value either *slot1* or *slot2* or slot3 to the variable *nameData* before going to the *continue* frame.

The Suwan Kingdom Game (Text Tools)

Figure 3.49: Suwan Kingdom Game (Text Tools)

To start creating a Text dialog, click Text Tools (Figure 3.49) then place it on the screen.

- For Text Dialog, there are 3 types (Figure 3.50)
  - Static Text – Use for Text that is not changed.
  - Input Text – Use for input Text. We use this to Enter Name Dialog.
  - Dynamic Text – Use for Text that will be changed when showing.
- Note that we must set name for each Text dialog so that it allows another function to use or call this Text dialog.
- nextFrame() means go to the frame next to this frame.

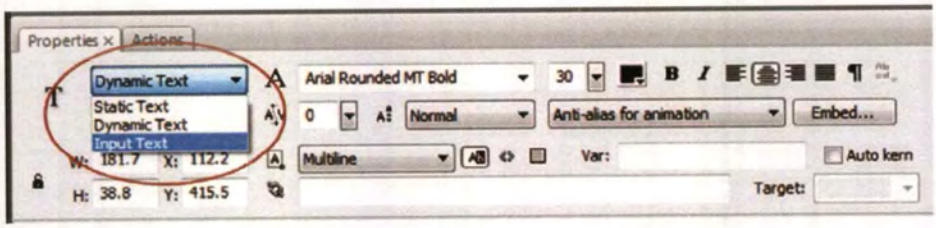


Figure 3.50: Suwan Kingdom Game (Text Type)

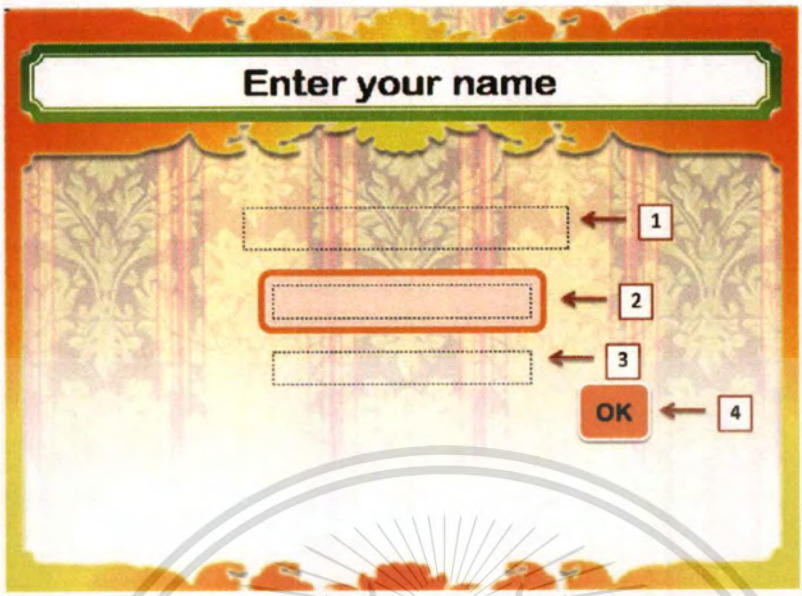


Figure 3.51: Suwan Kingdom Game (Frame 5: character)

As shown in Figure 3.51, there are three Text dialogs (Number 1-3) and one button (Number 4).

- Number 1: Player number shows the number of this player in each turn.
- Number 2: Enter Name shows Enter name dialog in which each player must type his name in this dialog box.
- Number 3: Warning Text shows Warning text dialog.
- Number 4: OK Button is the button that the player clicks after he finishes entering his name in which if the duplicated name is entered, the duplicated warning text will appear.

```
ok_btn.onPress = function() {

    if (name_txt.text != "") {
        var checkName = true;
        for (var i = 0; i<_parent.number; i++) {
            if (name_txt.text.toLowerCase() == _parent.nameChar[i].toLowerCase()) {
                checkName = false;
            }
        }
        if (checkName) {
            _parent.nameChar[_parent.number-1] = name_txt.text;
            nextFrame();
        } else {
            warning_txt.text = "Name not duplicate other player";
        }
    } else {
        warning_txt.text = "Please Enter You name";
    }
};

```

Figure 3.52: Suwan Kingdom Game (Frame 5: character Code)

Figure 3.53: Suwan Kingdom Game (Frame 6: BeforeRandomScreen4)

In this frame as shown in Figure 3.53, we use a random function to randomly obtain each player's order. There are two buttons: the next button (*NextBt.onPress*) function and the next button1 (*Next1.onPress*). *NextBt.onPress* will call function *NextRand4* and *Next1.onPress* will call function *NextFrame*. More codes are shown in Figure 3.54.

```
1  stop();
2  NextBt.onPress = NextRand4;
3  Next1.onPress = NextFrame;
4  RanNum = 0;
5  function NextRand04(){
6      NextBt.enabled = false;
7      a = Math.floor(Math.random()*4);
8      mcRandomScreen.gotoAndPlay(a);
9      RanNum++;
10     switch(RanNum){
11         case 200:
12         clearInterval(myInterval);
13         NextBt._visible = false;
14         trace(NumMem);
15         break;
16     }
17 }
18
19 function NextRand4(){
20     myInterval = setInterval(NextRand04,10);
21 }
```

Figure 3.54: Suwan Kingdom Game (Frame 6: BeforeRandomScreen4 Code)

- function *NextRand*4() is the function used to obtain the player's order by setting the interval of the random number function (NextRand04) to 0.01 second as follows:
    - myInterval = setInterval(NextRand04,10)
        - Means do functions *NextRand04* one times in 0.01 second.
- function *NextRand04()* consists of the following statements:
    - NextBt.enable = false
        - Means disable *NextBt* button.
    - a = Math.floor(Math.random()*4)
        - Means random number from 0-3 and assign the result value to *a*.
    - clearInterval(myInterval)
        - Means stop doing myInterval.
    - NextBt.visible = false
        - Means make *NextBt* button invisible.

```
stop();
PX1 = -78;
PY1 = -57;
PX2 = -78;
PY2 = 84;
PX3 = 159;
PY3 = -57;
PX4 = 159;
PY4 = 84;
_root.NumMem = 1;
_root.noChar = [0,1,2,3];
_root.mcRandomScreen.onEnterFrame = function(){
    switch(_root.genChar[1]){
        case "Guy":
        trace("Guy");
            switch(_root.jobChar[1]){
            case "FighterGuy":
            trace("FighterGuy");
                switch(_root.colorChar[1]){
                case "Red":
                trace(555);
                Player1 = this.attachMovie("mcFightGuyRedBox","Player1",1);
                Player1._x = PX2;
                Player1._y = PY2;
                break;
                case "Blue":
                trace(666);
                Player1 = this.attachMovie("mcFightGuyBlueBox","Player1",1);
                Player1._x = PX2;
                Player1._y = PY2;
                break;
```

Figure 3.55: Suwan Kingdom Game (mcRandomScreen Code)

In Figure 3.55, Codes in mcRandomscreen Frame are displayed as follows:

- *_root.NumMem = 1*
  - Set *Nummem* value to1 in *_root*(Main Frame).
- *_root.noChar = [0,1,2,3]*
  - Set *noChar* array to [0,1,2,3].
- *Switch (_root.genChar[1])*
  - Use switch to compare value in *genChar[1]* in root.
- Case "Guy":
  - If the value in *genChar[1]* is "Guy" then follow all codes in this case.

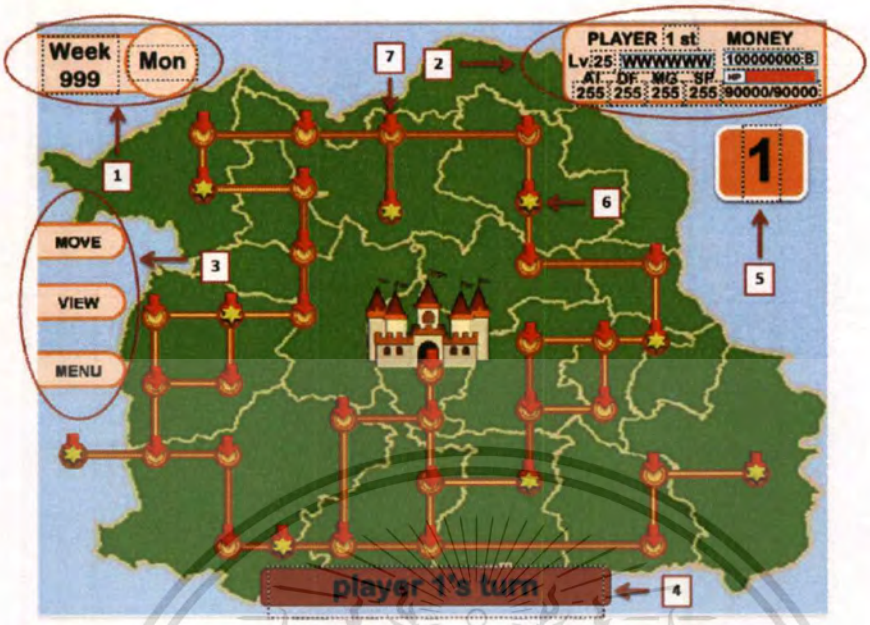


Figure 3.56: Suwan Kingdom Game (Frame 14: worldMap)

As shown in Figure 3.56 this frame consists of the following components:

- Number 1: **Week and Day in Game** shows Week and Day that the game has been played (game time).
- Number 2: **Player Status** shows the status of each player.
- Number 3: **Buttons in World Map** shows buttons that the player can press in the World Map Screen. There are 3 buttons:
  - MOVE: When this button is clicked, the program will call the spinner to obtain the random moving number.
  - VIEW: When this button is clicked, all Bosses' statuses including the Big Boss' level will be shown.
  - MENU: When this button is clicked, the Save Screen which allows the player to save his games to play later will appear.
- Number 4: **Game Dialog** shows dialogs in the game.
- Number 5: **Move Number** shows the moving number that the player has left.

- Number 6: **Town Star** shows the star color of each town.
- Number 7: **Arrow Button** which will appear after the player finishes spinning. Each arrow represents the direction that the player can walk to in each moving step.

```
#include "mc_tween2.as"
var myThis = this;
var winName = "";
var keepPlayer:Array = new Array();
var keepWalk:Array = new Array();
var keepSnap:MovieClip = new MovieClip();
var dataSword:Object = new Object();
var dataMage:Object = new Object();
var dataThief:Object = new Object();
var dataSword:Object = new Object();
var dataFighter:Object = new Object();
var dataMonster:Object = new Object();
var week:Number = 1;
var town_ary:Array = new Array();
var startDay:Number = 0;
var day_ary = ["Sun","Mon","Tue","Wens","Thur","Fri","Sat"];
var nubSnap = 0;
var image_mcl:MovieClipLoader = new MovieClipLoader();
```

Figure 3.57: Suwan Kingdom Game (Frame 14: worldMap Code1)

Figure 3.57 and Figure 3.61 to Figure 3.78 show codes in World Map Frame.

Important codes in Figure 3.57 are as follows:

- #include "mc_tween2.as"
  - Allow functions in mc_tween2.as to use code in this frame.
- var keepSnap:MovieClip = new MovieClip()
  - Create a new MovieClip name *keepSnap*.
- var dataSword: Object = new Object()
  - Create a new Object name *dataSword*.
- var day_ary = ["Sun","Mon","Tue","Wed","Thur","Fri","Sat"]
  - Create an array named *day_ary* containing the day name in a week "Sun","Mon","Tue","Wed","Thur","Fri"and"Sat".
- var image_mcl:MovieClipLoader = new MovieClipLoader()
  - Create a new MovieClipLoader name *image_mcl*.

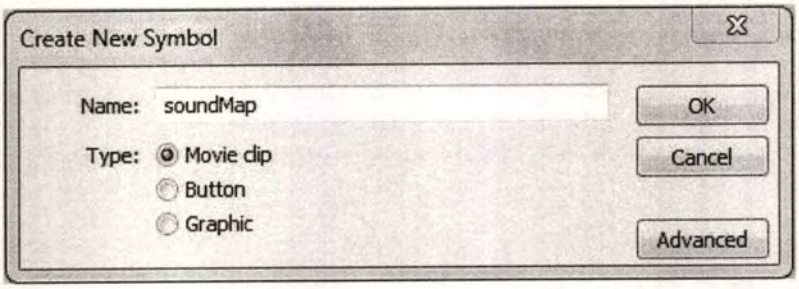


Figure 3.58: Suwan Kingdom Game (Create Sound1)

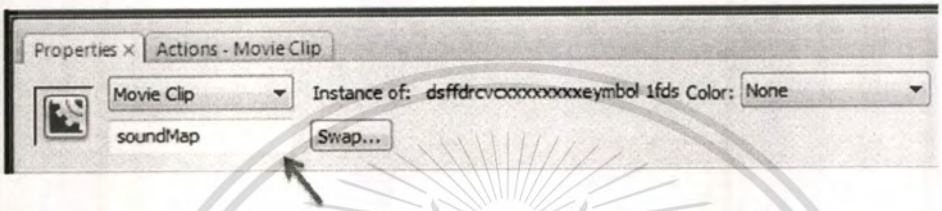


Figure 3.59: Suwan Kingdom Game (Create Sound2)

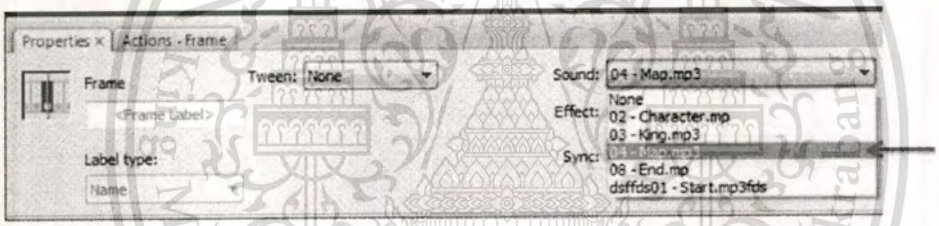


Figure 3.60: Suwan Kingdom Game (Create Sound3)

To create the sound, follow these steps: click Insert -> New symbol (Figure 3.58 appears after "New symbol" has been clicked) -> Choose "Movie clip" -> Click "OK". Click at Movie Clip then select Properties and then click at dialog below Movie Clip to type Instance name (Figure 3.59). Instance name is used to assign name to this Movie Clip so that the clip can be called by other function. Double-click at Movie Clip to go inside that Movie Clip. Click at Properties to add Sound (Figure 3.60). Number of Frames inside the Movie Clip depends on Sound length.

```
function init(){
    soundMap.gotoAndPlay(2);
    mcSave._visible = false;
    setWeek();
    nubMiniBoss = 8;
    bg_boss._visible = false;
    bigBoss._visible = false;
    nubSnap = 1;
    game.swapDepths(1000);
    allow = false;
    keepSnap = snap1;
    point = 0;
    showPoint._visible = false;
    finishedMove = true;
    turn = 0;
    setPlayer();
    setSnap();
    setTown();
    for(var i = 1; i<=9;i++){
        var star = this["star" + i];
        star._visible = false;
    }
}
function setTown(){
    var obj:Object = new Object();
    obj.name = "Prachin Buri";

    ojb.ownTown = "";
    ojb.color = "";
    ojb.value = 100,000;
    town_ary.push(obj);
```

Figure 3.61: Suwan Kingdom Game (Frame 14: worldMap Code2)

Figure 3.61 consists of codes for *function init()* and *function setTown()*.

- function init()
  - Set all initial values the frame needs.
  - the soundMap.gotoAndPlay(2)
    - Set *soundmap* Movie Clip play on Frame 2.
  - mcSave._visible = false
    - Set the Movie Clip *mcSave* be invisible.
  - setWeek()
    - Call function *setWeek*.
  - game.swapDepths (1000)

- Set the depth of game to 1000. (Depth means the depth of the object. For example, if object1 depth is 10 and another object depth is 20 then object2 is shown over object1 in the screen)

- function setTown()
  - obj.name = "Prachin Buri"
    - Set the name value in *obj* object to “Prachin Buri”.
  - town_ary.push(obj)
    - Push *obj* object into *town_ary* array.

```
109  function setWeek(){
110      mcWeek.week_txt.text = "Week " + week;
111      mcWeek.day_txt.text = day_ary[startDay];
112  }
113
114  function setPlayer(){
115      dataSword.AT = 25;
116      dataSword.MG = 5;
117      dataSword.DF = 15;
118      dataSword.SP = 5;
119
120
121      dataMage.AT = 5;
122      dataMage.MG = 25;
123      dataMage.DF = 10;
124      dataMage.SP = 10;
125
126      dataFighter.AT = 15;
127      dataFighter.MG = 10;
128      dataFighter.DF = 10;
129      dataFighter.SP = 15;
130
131      dataThief.AT = 10;
132      dataThief.MG = 5;
133      dataThief.DF = 10;
134      dataThief.SP = 25;
```

Figure 3.62: Suwan Kingdom Game (Frame 14: worldMap Code3)

- function setWeek()
  - Set the date of the game play. As seen in Figure 3.62, there are two statements:
    - mcWeek.week_txt.text = "Week " + week
      - Display text “Week” + *week number*.
    - mcWeek.day_txt.text = day_ary[startDay]
      - Display value in *day_ary* on *day_txt*.

- function setPlayer() (See Figure 3.62 and Figure 3.63)
  - Set the player default values. For example, *dataSword.AT = 25* sets AT value in *dataSword* to *25*.

```
136     for(var i = 0;i < numPlayer;i++){
137         nn = "mc" + jobChar[noChar[i]] + colorChar[noChar[i]] + "Icon";
138         trace(nn);
139         var player = this.attachMovie(nn,"player" + (i + 1),_root.getNextHighestDepth());
140         player.attM = "mc" + jobChar[noChar[i]] + colorChar[noChar[i]];
141         player._x = snap1._x;
142         player._y = snap1._y;
143         player.gen = jobChar[noChar[i]];
144         player.color = colorChar[noChar[i]];
145         player.job = jobChar[noChar[i]];
146         player.lv = 1;
147         player.money = 0;
148         player.skillRound = 0;
149         player.plusAT = 1;
150         player.plusMG = 1;
151         player.plusSP = 1;
152         player.plusDF = 1;
153         player.exp = 0;
154         player.point = 0;
155         player.mode = 1;  // 1/Normal 2/ Fighting 3/ Give up 4/ Dead
156         player.noTurn = 0;
157         player.name = nameChar[noChar[i]];
158         player.snap = snap1;
```

Figure 3.63: Suwan Kingdom Game (Frame 14: worldMap Code4)

- var player = this.attachMovie(nn,"player" + (i + 1),_root.getNextHighestDepth())
  - Attach the Movie clip from the library to stage
  - Assign name to "player"+ *(i+1)* and save it to the variable *player*.
- root.getNextHighestDepth()
  - Set this Movie Clip to the highest depth.

```
175         player.hp = getHp(player.lv,player.stats.AT);
176         player.currHp = player.hp;
177         keepPlayer.push(player);
178         trace(keepPlayer[i]);
179
180     }
181     player1.swapDepths(2000);
182     setPlayerData();
183     showNamePlayer_txt.text = "player "+ keepPlayer[turn].name + "'s turn";
184 }
185 function getHp(lv:Number,at:Number){
186     var hp =  0;
187     hp = (lv*100) + (at*10)
188     return hp;
189 }
190 function setPlayerData(){
191     mcPlayerData.name_txt.text = keepPlayer[turn].name;
192     mcPlayerData.money_txt.text = keepPlayer[turn].money;
193     mcPlayerData.hp_txt.text = keepPlayer[turn].currHp + "/" + keepPlayer[turn].hp;
194     mcPlayerData.bar._xscale = (keepPlayer[turn].currHp * 100) / keepPlayer[turn].hp;
195     mcPlayerData.at_txt.text = keepPlayer[turn].stats.AT;
196     mcPlayerData.df_txt.text = keepPlayer[turn].stats.DF;
197     mcPlayerData.mg_txt.text = keepPlayer[turn].stats.MG;
198     mcPlayerData.sp_txt.text = keepPlayer[turn].stats.SP;
199     mcPlayerData.lv_txt.text = keepPlayer[turn].lv;
200     mcPlayerData.number_txt.text = (turn + 1) + " st";
201 }
```

Figure 3.64: Suwan Kingdom Game (Frame 14: worldMap Code5)

- player.hp = getHp(player.lv,player.stats.AT)
  - Set the player.hp to hit points (HP) calculating from *getHP* function.
- function getHp(lv:Number,at:Number)
  - Create function *getHP* together with 2 input numbers (*lv* and *at*).
- function setPlayerData
  - Set the player data such as the player's *name, money, HP, AT, DF, MG, SP* and *LV*.

```
202 function setSnap(){
203     snap1.near = [snap2];
204     snap2.near = [snap1,snap3,snap4];
205     snap3.near = [snap2,snap5,snap7];
```

Figure 3.65: Suwan Kingdom Game (Frame 14: worldMap Code6)

- function setSnap (See Figure 3.65)
  - Set values in each point.
  - Snap is the number of each point.
  - An example of snap statement is snap1.near = [snap2]
    - Allow point named *snap2* to walk from point named *snap1*.

```
240     snap2.dataObj = new Object();
241     snap2.dataObj.nameMon = "Smile";
242     snap2.dataObj.lv = 1;
243     snap2.dataObj.AT = 15;
244     snap2.dataObj.MG = 10;
245     snap2.dataObj.DF = 15;
246     snap2.dataObj.SP = 10;
247     snap2.dataObj.plusAT = 1;
248     snap2.dataObj.plusMG = 1;
249     snap2.dataObj.plusSP = 1;
250     snap2.dataObj.plusDF = 1;
251     snap2.dataObj.status = "live";
252     snap2.dataObj.exp = 5;
253     snap2.dataObj.money = 10;
254     snap2.dataObj.att = "AT";
255     snap2.dataObj.xx = 636.5;
256     snap2.dataObj.yy = 475.5;
257     snap2.dataObj.map = "map1";
258     snap2.dataObj.type = "normal";
259     snap2.dataObj.player1 = 0;
260     snap2.dataObj.player2 = 0;
261     snap2.dataObj.player3 = 0;
262     snap2.dataObj.player4 = 0;
263     trace("snap2.dataObj.player" + snap2.dataObj.player._name);
264     snap2.dataObj.hp = getHp(snap2.dataObj.lv,snap2.dataObj.AT);
265     snap2.dataObj.currHp = snap2.dataObj.hp;
```

Figure 3.66: Suwan Kingdom Game (Frame 14: worldMap Code7)

- snap2.dataObj = new Object()
  - Create a new object name *dataObj*.
- Other statements in Figure 3.66 are statements for setting values of point *snap2*.

```
1209  function visibleSnap(con:Boolean){
1210      for(var i = 1;i<=36;i++){
1211          if(con){
1212
1213              for(var j = 0;j<keepSnap.near.length;j++){
1214                  if(keepSnap.near[j]._name != "snap30"){
1215                    keepSnap.near[j]._visible = true;
1216                    keepSnap.near[j].swapDepths(1001 + j);
1217                  }
1218                  else if(nubMiniBoss <= 0){
1219                    keepSnap.near[j]._visible = true;
1220                    keepSnap.near[j].swapDepths(1001 + j);
1221                  }
1222              }
1223          }
1224          else{
1225               var snap = this["snap" + i];
1226              snap._visible = false;
1227          }
1228      }
1229  }
```

Figure 3.67: Suwan Kingdom Game (Frame 14: worldMap Code8)

- function visibleSnap(con:Boolean) (See Figure 3.67)
  - Create function *visibleSnap and* assign 1 to the *con* parameter (input condition typed Boolean (true or false)).
  - This function sets snap point to visible when the condition is true.

```
1230  function finishedSpin(){
1231      showPoint._visible = true;
1232      trace("point" + point);
1233      showPoint.numPoint.text = point;
1234      //spin._visible = false;
1235      if(point == 0 && keepPlayer[turn].snap._name != "snap1"){
1236          setTimeout(loadMap,2000);
1237      }
1238      else if(point == 0 && keepPlayer[turn].snap._name == "snap1"){
1239          setTimeout(finishedGame,2000);
1240      }
1241      else{
1242          keepWalk = new Array();
1243         keepWalk.push(keepPlayer[turn].snap._name);
1244         trace(keepWalk.length);
1245         visibleSnap(true);
1246      }
1247  }
1248  function checkBeforeMove(){
1249      if(this._visible){
1250      for(var i = 0;i<keepSnap.near.length;i++){
1251          if(this._name == keepSnap.near[i]._name){
1252              visibleSnap(false);
1253              movePlayer(this);
1254          }
1255      }
1256      }
1257  }
```

Figure 3.68: Suwan Kingdom Game (Frame 14: worldMap Code9)

The following explains codes in Figure 3.68.

- function finishedSpin()
  - Show moving number after spinning and check "Is player number is equal 0".
  - setTimeout(loadMap,2000)
    - Call function *loadMap* in 2 second.
- function checkBeforeMove()
  - Set the player token to be invisible and call function *movePlayer*.

```
1258  function movePlayer(snap:MovieClip){
1259      keepSnap = snap;
1260      keepWalk.push(snap._name);
1261      allow = true;
1262      keepPlayer[turn].slideTo(snap._x,snap._y,1,'easeoutsine');
1263      keepPlayer[turn].onTweenComplete = function() {
1264          if(allow){
1265              keepPlayer[turn].snap = snap;
1266               allow = false;
1267               var checkSnap = 0;
1268
1269                   if(snap._name == keepWalk[keepWalk.length - 3]){
1270                       keepWalk[keepWalk.length - 2] = "";
1271                       checkSnap = 1;
1272                   }
1273
1274               if(checkSnap == 1){
1275                    point++;
1276               }
1277               else{
1278                   point--;
1279               }
1280               showPoint.numPoint.text = point;
1281               if(point > 0){
1282                  visibleSnap(true);
1283               }
1284               else{
1285                  if(keepSnap._name == "snap1"){
1286                      keepPlayer[turn].currHp = keepPlayer[turn].hp;
1287                      setTimeout(finishedGame,1500);
1288                  }
1289                  else
1290                  setTimeout(loadMap,1500);
```

Figure 3.69: Suwan Kingdom Game (Frame 14: worldMap Code10)

- function movePlayer(snap:MovieClip)
  - This function is used to move the player to the point that the player chooses.
  - As seen in Figure 3.69, the parameter of this function is snap (snap is a MovieClip of point).Some important codes in this function are described as follows:
    - keepPlayer[turn].slideTo(snap._x,snap._y,1,'easeoutsine')
      - Move the player's token to point *snap._x* and *snap._y*.
    - keepPlayer[turn].onTweenComplete = function()
      - Perform this function when Tween is completed. (Tween is a class for creating smooth animations using ActionScript by changing the value of the object properties over a period of time or frames.)

```
1296  }
1297  function loadMap(){
1298      trace("loadMap" + nubb++);
1299      soundMap.gotoAndStop(1);
1300      dataMonster = keepPlayer[turn].snap.dataObj;
1301      checkHitSnap(keepPlayer[turn].snap);
1302      var ary_player:Array = new Array();
1303      if(snap.dataObj.player1 == 1) ary_player.push(keepPlayer[0]);
1304      if(snap.dataObj.player2 == 1) ary_player.push(keepPlayer[1]);
1305      if(snap.dataObj.player3 == 1) ary_player.push(keepPlayer[2]);
1306      if(snap.dataObj.player4 == 1) ary_player.push(keepPlayer[3]);
1307      if(ary_player.length <=1 && dataMonster.status == "die"){
1308          dataMonster.status == "live"
1309      }
1310      showPoint._visible = false;
1311      keepPlayer[0]._visible = false;
1312      keepPlayer[1]._visible = false;
1313      keepPlayer[2]._visible = false;
1314      keepPlayer[3]._visible = false;
1315      bg._visible = false;
1316      mcPlayerData._visible = false;
1317      mcWeek._visible = false;
1318      nameFile = "File/Fighting.swf";
1319      var mclListener:Object = new Object();
1320          mclListener.onLoadInit = function(target_mc:MovieClip) {
1321              loadGame.swapDepths(3000);
1322          };
1323
1324       image_mcl.addListener(mclListener);
1325       //image_mcl.loadClip(selectFile,loadFile);
1326       image_mcl.loadClip(nameFile,loadGame);
1327  }
```

Figure 3.70: Suwan Kingdom Game (Frame 14: worldMap Code11)

- function loadMap()
  - Load Fighting screen after the player has finished his moving (move number equals 0).
  - Some codes as can be seen in Figure 3.70 are:
    - checkHitSnap(keepPlayer[turn].snap)
      - Check what point that the player is standing on.
    - nameFile = "File/Fighting.swf"
      - Load files *Fighting.swf* from this path *"File/Fighting.swf"*.
    - mclListener.onLoadInit = function(target_mc:MovieClip)
      - Load Movie Clip into *mcListener* object.
    - image_mcl.addListener(mclListener)
      - Set image_*mcl* to obtain the event from *mcListener*.
    - image_mcl.loadClip(nameFile,loadGame)

- Load clip from *namefile.*

```
1328  function checkHitSnap(snap:MovieClip){
1329      if(snap.hitTest(keepPlayer[0])){
1330          snap.dataObj.player1 = 1;
1331      }
1332      else snap.dataObj.player1 = 0;
1333
1334      if(snap.hitTest(keepPlayer[1])){
1335          snap.dataObj.player2 = 1;
1336      }
1337      else snap.dataObj.player2 = 0;
1338
1339      if(snap.hitTest(keepPlayer[2])){
1340          snap.dataObj.player3 = 1;
1341      }
1342      else snap.dataObj.player3 = 0;
1343
1344      if(snap.hitTest(keepPlayer[3])){
1345          snap.dataObj.player4 = 1;
1346      }
1347      else snap.dataObj.player4 = 0;
1348  }
```

Figure 3.71: Suwan Kingdom Game (Frame 14: worldMap Code12)

- function checkHitSnap(snap:MovieClip)
  - Check if there is any player on the point.
  - if(snap.hitTest(keepPlayer[0]))
    - Check if the player is on the point.
    - Keep checking (see codes in Figure 3.71) until the last player.

```
1349  function finishedGame(){
1350      soundMap.gotoAndPlay(2);
1351      updateStar();
1352      if(nubMiniBoss <= 0){
1353          bg_boss._visible = true;
1354      }
1355      MoveBt._visible = true;
1356      spin.gotoAndStop(1);
1357
1358      keepPlayer[turn].gotoAndStop(keepPlayer[turn].mode);
1359      if(keepPlayer[turn].mode == 4){
1360          keepPlayer[turn]._x = snap1._x;
1361          keepPlayer[turn]._y = snap1._y;
1362      }
1363      image_mc1.unloadClip(loadGame);
1364      bg._visible = true;
1365      mcPlayerData._visible = true;
1366      mcWeek._visible = true;
1367      showPoint._visible = false;
1368      keepPlayer[0]._visible = true;
1369      keepPlayer[1]._visible = true;
1370      keepPlayer[2]._visible = true;
1371      keepPlayer[3]._visible = true;
1372      keepSnap.dataObj = dataMonster;
1373      checkHitSnap(keepSnap);
1374      turn++;
1375      if(turn == numPlayer){
1376          turn = 0;
1377          startDay++;
1378          if(startDay == 7) startDay = 0;
1379          setWeek();
1380      }
```

Figure 3.72: Suwan Kingdom Game (Frame 14: worldMap Code13)

Figure 3.72 and Figure 3.73 displays codes for function *finishedGame()*.

- function finishedGame()
  - Call the World Map Screen after the Fighting is completed.
  - image_mc1.unloadClip(loadGame)
    - Unload clip *loadGame*.

```
1381        setPlayerData();
1382        showNamePlayer_txt.text = "player "+ nameChar[noChar[turn]] + "'s turn";
1383
1384        for(var i = 0;i<4;i++){
1385            if(i == turn){
1386                keepPlayer[turn].swapDepths(2000);
1387            }
1388            else{
1389                keepPlayer[i].swapDepths(400 + i);
1390            }
1391        }
1392        keepSnap = keepPlayer[turn].snap;
1393        finishedMove = true;
1394        if(keepPlayer[turn].mode == 2){
1395            MoveBt._visible = false;
1396            setTimeout(loadMap,5000);
1397        }
1398        else if(keepPlayer[turn].noTurn > 0){
1399            MoveBt._visible = false;
1400            keepPlayer[turn].noTurn--;
1401            keepSnap.dataObj.player = 5;
1402            if(keepPlayer[turn].noTurn == 0){
1403                if(keepPlayer[turn].mode == 4){
1404                    keepPlayer[turn].snap = snap1;
1405                    keepPlayer[turn].currHp = keepPlayer[turn].hp;
1406                    var angle = myThis.attachMovie("mcRevival","mcRevival",500);
1407                    angle.gotoAndPlay(1);
1408                    angle._x = 406.95;
1409                    angle._y = 276.35;
1410                    keepPlayer[turn].mode = 1;
1411                    keepPlayer[turn].gotoAndStop(keepPlayer[turn].mode)
1412                    turn--;
```

Figure 3.73: Suwan Kingdom Game (Frame 14: worldMap Code14)

- var angle = myThis.attachMovie("mcRevival","mcRevival",500)
  - Attach the movie name *mcRevival* from the library, called it *mcRevival* and assigns its depth to 500.

```
1425  function updateStar(){
1426      for(var i = 0;i<town_ary.length;i++){
1427          var star = myThis["star" + (i + 1)];
1428          if(town_ary[i].ownTown != ""){
1429          if(town_ary[i].ownTown == keepPlayer[0]._name || town_ary[i].ownTown == keepPlayer[1]._name
1430          || town_ary[i].ownTown == keepPlayer[2]._name || town_ary[i].ownTown == keepPlayer[3]._name)
1431              star._visible = true;
1432              star.gotoAndStop(town_ary[i].color);
1433          }
1434          }
1435      }
1436  }
1437  function clearStage(){
1438      keepPlayer[0]._visible = false;
1439      keepPlayer[1]._visible = false;
1440      keepPlayer[2]._visible = false;
1441      keepPlayer[3]._visible = false;
1442  }
```

Figure 3.74: Suwan Kingdom Game (Frame 14: worldMap Code15)

Figure 3.74 shows code for *function updateStar()* and *function clearStages()*.

- function updateStar()
  - Update the color of the star to the player's color.
- function clearStage()
  - Make all players' characters invisible.

```
1443  function saveGame(nameData2:String){
1444      returnVisible();
1445      var my_so:SharedObject = SharedObject.getLocal(nameData2, "/");
1446      var obj:Object;
1447  //เช็คว่ามีข้อมูลหรือไม่ ถ้ามีก็ดึงออกมา ถ้าไม่มีก็สร้างอาเรย์ใหม่
1448  if (my_so.data.obj != undefined) {
1449      obj = my_so.data.obj;
1450  } else {
1451      obj = new Object();
1452  }
1453     obj.dateSave = getDate();
1454     obj.week = week;
1455     obj.numPlayer = numPlayer;
1456
1457     var ary_snap = new Array();
1458     for(var i = 1;i<=36;i++){
1459         var snap = myThis["snap" + i];
1460         ary_snap.push(snap.dataObj);
1461     }
1462     obj.startDay = startDay;
1463     obj.turn = turn;
1464     obj.point = point;
1465     obj.nubMiniboss = nubMiniboss;
1466     var savePlayer:Array = new Array();
```

Figure 3.75: Suwan Kingdom Game (Frame 14: worldMap Code16)

Figure 3.75 and Figure 3.76 show codes of function *saveGame()*.

- function saveGame()
  - Create function *saveGame* to save all values for the playing later. And assign 1 for the input String.
  - var my_so:SharedObject = SharedObject.getLocal(nameData2, "/")
    - Create SharedObject *my_so* for saving all current values.

```
1466    var savePlayer:Array = new Array();
1467    for(var i = 0;i<keepPlayer.length;i++){
1468        var objPlayer:Object = new Object();
1469         objPlayer.attM = keepPlayer[i].attM;
1470         objPlayer.color = keepPlayer[i].color;
1471         objPlayer.job = keepPlayer[i].job;
1472         objPlayer.lv = keepPlayer[i].lv;
1473         objPlayer.money = keepPlayer[i].money;
1474         objPlayer.skillRound = keepPlayer[i].skillRound;
1475         objPlayer.plusAT = keepPlayer[i].plusAT;
1476         objPlayer.plusMG = keepPlayer[i].plusMG;
1477         objPlayer.plusSP = keepPlayer[i].plusSP;
1478         objPlayer.plusDF = keepPlayer[i].plusDF;
1479         objPlayer.exp = keepPlayer[i].exp;
1480         objPlayer.point = keepPlayer[i].point;
1481         objPlayer.mode = keepPlayer[i].mode;
1482         objPlayer.noTurn = keepPlayer[i].noTurn;
1483         objPlayer.name = keepPlayer[i].name;
1484         objPlayer.gen = keepPlayer[i].gen;
1485
1486         objPlayer.snap = keepPlayer[i].snap._name;
1487         objPlayer.job = keepPlayer[i].job;
1488         objPlayer.stats = keepPlayer[i].stats;
1489         objPlayer.hp = keepPlayer[i].hp;
1490         objPlayer.currHp = keepPlayer[i].currHp;
1491         savePlayer.push(objPlayer);
1492     }
1493      my_so.data.obj = obj;
1494      my_so.data.keepWalk = keepWalk;
1495      my_so.data.keepPlayer = savePlayer;
1496      my_so.data.ary_snap = ary_snap;
1497      my_so.data.town_ary = town_ary;
1498  }
```

Figure 3.76: Suwan Kingdom Game (Frame 14: worldMap Code17)

- All data required for saving can be seen in Figure 3.76.

```
1499  function returnVisible(){
1500
1501      keepPlayer[0]._visible = true;
1502      keepPlayer[1]._visible = true;
1503      keepPlayer[2]._visible = true;
1504      keepPlayer[3]._visible = true;
1505  }
1506  MoveBt.onPress = function(){
1507      if(finishedMove){
1508          finishedMove = false;
1509          spin.gotoAndStop(2);
1510      }
1511  }
1512  save_btn.onPress = function(){
1513      if(point == 0){
1514      mcSave._visible = true;
1515      mcSave.gotoAndStop(2);
1516      mcSave.swapDepths(3000000);
1517      clearStage();
1518      }
1519  }
1520  ViewBt.onPress = function(){
1521      if(point == 0){
1522      mcView.gotoAndStop(2);
1523      mcView.swapDepths(4000000);
1524      clearStage();
1525      }
1526  }
```

Figure 3.77: Suwan Kingdom Game (Frame 14: worldMap Code18)

Figure 3.77 shows codes of for following functions:

- function returnVisible()
    - Set all players' characters to visible.
- MoveBt.onpress = function()
    - Call a spinner by clicking at *MoveBt* button.
- save_btn.onPress = function()
    - Call Save Screen
- ViewBt.onPress = function()
    - Call View Screen.

```
1527  function gameWin(){
1528      clearStage();
1529      winName = keepPlayer[turn].name;
1530      var winGen = substring(keepPlayer[turn].gen,keepPlayer[turn].gen.length,1);
1531      if(winGen == "1"){
1532          gotoAndStop("endGirl");
1533      }
1534      else{
1535          gotoAndStop("endGuy");
1536      }
1537  }
1538  nubb = 0;
1539  init();
```

Figure 3.78: Suwan Kingdom Game (Frame 14: worldMap Code19)

Figure 3.78shows codes of *function gameWin()*.

- function gameWin()
    - Check the winner's gender and call End Game Screen.

```
1   #include "mc_tween2.as"
2   var my_so:SharedObject = SharedObject.getLocal(nameData, "/");
3   var objData:Object;
4   // Check did it have a new data? if have load it, if not then create new array
5   objData = my_so.data.obj;
6   var winName = "";
7   var oldPlayer:Array = my_so.data.keepPlayer;
8   var old_snap:Array = my_so.data.ary_snap;
9   var town_ary:Array = my_so.data.town_ary;
10  var numPlayer = 0;
11  var myThis = this;
12  var keepPlayer:Array = new Array();
13  var keepWalk:Array = my_so.data.keepWalk;
14  var keepSnap:MovieClip = new MovieClip();
15  var dataMonster:Object = new Object();
16  var week:Number = 0;
17  //var town_ary:Array = new Array();
18  var startDay:Number = 0;
19  var nubMiniBoss = 0;
20  var day_ary = ["Sun","Mon","Tue","Wens","Thur","Fri","Sat"];
21  var nubSnap = 0;
22  var image_mcl:MovieClipLoader = new MovieClipLoader();
```

Figure 3.79: Suwan Kingdom Game (Frame 15: continue Code)

Important codes in Figure 3.79 are as follows:

- var my_so:SharedObject = SharedObject.getLocal(nameData, "/")
    - Load data from the saved data.
- objData = my_so.data.obj
    - If there is data to load, load it. If there is not, create a new array.

Figure 3.80: Suwan Kingdom Game (Frame 14.1: Spinner)

```
1   var rang = random(10) + 7;
2   id_time = setInterval(gg,500);
3
4   mc.onEnterFrame = function(){
5       mc.gotoAndPlay(mc._currentframe += rang);
6
7   }
8   function gg(){
9
10      rang--;
11      if(rang <= 0){
12          clearInterval(id_time);
13          delete mc.onEnterFrame;
14          mc.stop();
15          setTimeout(getPoint,2500);
16
17      }
18  }
19  function getPoint(){
20
21
22      _parent.finishedSpin();
23
24  }
```

Figure 3.81: Suwan Kingdom Game (Frame 14.1: Spinner Code)

Figure 3.81 shows codes in Spinner Frame in movie clip.

- var rang = random(10) + 7
  - Create a random number from 7 to 16 then save its value in variable *rang*.
- mc.onEnterFrame = function()
  - Set *mc* movie clip playing on loop (spin spinner).
- function gg ()
  - Called when want to stop needle in the spinner.
- function getPoint()
  - Used to call function finishedSpin() for showing move number.

Figure 3.82: Suwan Kingdom Game (Frame 1: Fighting Frame)

Monster, Bosses and Big Boss will appear at the left (of the screen) all the time.

As can be seen in Figure 3.82, the fighting frame consists of 7 components as follows:

- Number 1: **Player Levels and Hit Points** shows the player levels and hit point.
- Number 2: **Player Status** shows the player status value (AT/DF/MG/SP).

- Number 3: **Dead** shows Dead picture. It will appear when someone is dead. In this example, it is the monster.
- Number 4: **Player Picture** shows the player's picture.
- Number 5: **Fighting Dialog Box** shows Text message in Dialog Box.

```
#include "mc_tween2.as"
var monster:MovieClip = new MovieClip();
function init(){
    mcDead2._visible = false;
    soundFight.gotoAndPlay(2);
    mcDead._visible = false;
    exp_ary = [34952,34267,33269,31989,30466,27696,24083,20069,16055,12350,9148
              ,6534,4506,3004,1938,1211,734,432,247,137,74,39,20,10];
    turn = "";
    round = 0;
    atkMc.swapDepths(100000);
    dataMonster = _root.dataMonster;
    monsterName = dataMonster.nameMon;
    atM_txt.text = dataMonster.AT;
    dfM_txt.text = dataMonster.DF;
    mgM_txt.text = dataMonster.MG;
    spM_txt.text = dataMonster.SP;
    lvM_txt.text = dataMonster.lv + " " + dataMonster.nameMon;
    des_txt.text = monsterName + " appeared";
    player = _root.keepPlayer[_root.turn];

    if(player.skillRound > 0){
        player.skillRound--;
    }
    else{
        player.plusAT = 1;
        player.plusMG = 1;
        player.plusDF = 1;
        player.plusSP = 1;
    }
```

Figure 3.83: Suwan Kingdom Game (Frame 1: Fighting Frame Code1)

The details of the frame codes are shown in Figure 3.83 – 3.99.

- function init()
  - Assign values to variables needed in the Fighting Frame as shown in Figure 3.83.
  - exp_ary = [34952, 34267, 33269, 31989, 30466, 27696, 24083, 20069, 16055, 12350, 9148, 6534, 4506, 3004, 1938, 1211, 734, 432, 247, 137, 74, 39, 20, 10]
    - Store experience number into *exp_ary* array.

```
function checkParentPlayer(){
    var changeMode = false;
    if(player.snap.player1 == 1 && _root.keepPlayer[0].mode == 2 && _root.keepPlayer[0] != player){
        changeMode = true;
    }
    else if(player.snap.player2 == 1 && _root.keepPlayer[1].mode == 2 && _root.keepPlayer[1] != player
        changeMode = true;
    }
    else if(player.snap.player3 == 1 && _root.keepPlayer[2].mode == 2 && _root.keepPlayer[2] != player
        changeMode = true;
    }
    else if(player.snap.player4 == 1 && _root.keepPlayer[3].mode == 2 && _root.keepPlayer[3] != player
        changeMode = true;
    }
    return changeMode;
}
this.onEnterFrame = function(){
    barP._xscale  = (curHpP * 100) / hpPlayer;
    barM._xscale = (curHpM * 100) / hpMonster;
    hpP_txt.text = curHpP + "/" + hpPlayer;
    hpM_txt.text = curHpM + "/" + hpMonster;
}
function useSkill(){
    sts.gotoAndPlay(2);
    player.skillRound = 2;
    if(player.job == "Fighter"){
        if(player.lv >= 20)
            player.plusAT = 1.5;
            player.plusSP = 1.5;
            des_txt.text = "ATSP UP III";
        }
        else if(player.lv >= 10){
```

Figure 3.84: Suwan Kingdom Game (Frame 1: Fighting Frame Code2)

The following codes are shown in Figure 3.84:

- function checkParentPlayer()
    - Change the player mode from normal to fighting.
- this.onEnterFrame = function()
    - Set values to hit points and display hit points values both in text and graphics (in the bar)
- function useSkill()
    - Set skill for each job.

```
function animagePlayer(type:String){
    atkMc.gotoAndStop(5);
    var mSkill = monsterDefend(type);
    var block = "";
    if(mSkill == "DF" || mSkill == "CT"){
        block = "mcDefend";
    }
    else block = "mcMagicDefend";
    mcBlock = this.attachMovie(block,"block"+this.getNextHighestDepth(),this.getNextHighestDepth());
    mcBlock._x = 533.8;
    mcBlock._y = 447.4;
    if(type == "ST" || type == "AT"){
        if(type == "AT")
        des_txt.text = "Player attack";
        else des_txt.text = "Player Strike";
        var allow = true;
        var allow2 = true;
        mcPlayer.slideTo(mcPlayer._x + 15 ,mcPlayer._y,1,'easeoutsine');
        mcPlayer.onTweenComplete = function() {
           if(allow){
               allow = false;
               mcPlayer.slideTo(mcPlayer._x - 15 ,mcPlayer._y,1,'easeoutsine');
               mcPlayer.onTweenComplete = function() {
                 if(allow2){
                     allow2 = false;
                     removeMovieClip(mcBlock);
                     playerAttack(type,mSkill);
```

Figure 3.85: Suwan Kingdom Game (Frame 1: Fighting Frame Code3)

- function animagePlayer(type:String)
  - Create the player's animation when he is performing his action (attacking or defending). Function details are shown in Figure 3.85.
  - removeMovieClip(mcBlock)
    - Remove Movie Clip *mcBlock*.

```
    else if(type == "AM"){
        des_txt.text = "Player attack magic";
        var magic = this.attachMovie("mcMagicAttack","block"+this.getNextHighestDepth(),this.getNextHi
        magic._x = 244.45;
        magic._y = 429.4;
        magic.onEnterFrame = function(){
            this._x += 5;
            if(this.hit.hitTest(monster)){
                removeMovieClip(this);
                removeMovieClip(mcBlock);
                playerAttack(type,mSkill);
                delete this.onEnterFrame;
```

Figure 3.86: Suwan Kingdom Game (Frame 1: Fighting Frame Code4)

  - magic.onEnterFrame = function()
    - Play the magic attack's animation. More function details can be seen in Figure 3.86.

```
function animageMonster(type:String){
    trace("animageMonster");
    atkMc.gotoAndStop(5);
    var mSkill = monsterSkill();
    trace(mSkill);
    var block = "";
    if(type == "DF" || type == "CT"){
        block = "mcDefend";
    }
    else block = "mcMagicDefend";
    mcBlock = this.attachMovie(block,"block"+this.getNextHighestDepth(),this.getNextHighestDepth());
    mcBlock._x = 223.3;
    mcBlock._y = 439.95;
    if(mSkill == "ST" || mSkill == "AT"){
        if(mSkill == "ST")
        des_txt.text = "Monster attack";
        else des_txt.text = "Monster Strike";
        var allow = true;
        var allow2 = true;
        trace("monster_x" + monster._x);
        monster.slideTo(monster._x - 15 ,monster._y,1,'easeoutsine');
        monster.onTweenComplete = function() {
           if(allow){
               allow = false;
               monster.slideTo(monster._x + 15 ,monster._y,1,'easeoutsine');
               monster.onTweenComplete = function() {
                 if(allow2){
                     allow2 = false;
                     removeMovieClip(mcBlock);
                     playerDefend(type,mSkill);
                 }
               }
```

Figure 3.87: Suwan Kingdom Game (Frame 1: Fighting Frame Code5)

- function animageMonster(type:String)
  - Play the Monster animation when the monster is on action (attacking or defending). See Figure 3.87 for codes in the function.

```
function playerAttack(type:String,mSkill:String){
    var job = player.job;
    var def = calDef(mSkill,dataMonster.DF,dataMonster.MG,dataMonster.plusDF,dataMonster.plusMG);
    var atk = calAtk(type,player.stats.AT,player.stats.MG,player.plusAT,player.plusMG);
    trace("player.stats.AT= " +player.stats.AT);
    trace("type = " + type);
    trace("atk =" + atk);
    trace("def = " + def);
    var percentDodge = (spP / dataMonster.SP) *100;
    var ranDodge = random(100) + 1;
    if(percentDodge>95){
        if(ranDodge>95){
            stas = "miss";
        }
    //if(ranDodge >= percentDodge) stas = "miss";
    }else if(percentDodge<=95){
        if(ranDodge>percentDodge){
            stas = "miss";
        }
    }

    var damageATK = atk - def;
    if(damageATK <= 0) damageATK = 0;
    var dmFinished = 0;
    if(type == "ST" && mSkill == "CT"){
        dmFinished = damageATK * 1.5;
        delPlayer(Math.round(dmFinished),type);
        return;
    }
```

Figure 3.88: Suwan Kingdom Game (Frame 1: Fighting Frame Code6)

- function playerAttack(type:String,mSkill:String)
  - Calculate the fighting result when the player is the attacker. See Figure 3.88 and 3.89 for more code details.

```
301     }
302     else if(type == "AT" && mSkill == "CT"){
303         if(atk > def) dmFinished = damageATK * 1.25;
304     }
305     else if(type == "AT" && mSkill == "DM"){
306         if(atk > def) dmFinished = damageATK * 1.25;
307     }
308     else if(type == "AM" && mSkill == "DF"){
309         if(atk > def) dmFinished = damageATK * 1.25;
310     }
311     else if(type == "AM" && mSkill == "CT"){
312         if(atk > def) dmFinished = damageATK * 1.25;
313     }
314     else if(type == "AM" && mSkill == "DM"){
315         if(atk > def) dmFinished = damageATK;
316     }
317     if(stas == "miss"){
318         des_txt.text = "Player attack miss!";
319         miss = this.attachMovie("miss","miss" + this.getNextHighestDepth(),this.getNextHighestDepth())
320         miss._x = 616.4;
321         miss._y = 286.45;
322         miss.gotoAndPlay(1);
323         setTimeout(nextRound,3000);
324     }
325     else if(dmFinished <= 0){
326         soundEvent.gotoAndPlay("block");
327
328         des_txt.text = "Monster can defend";
329         setTimeout(nextRound,3000);
330     }
331     else{
332         delMonster(Math.round(dmFinished),type);
```

Figure 3.89: Suwan Kingdom Game (Frame 1: Fighting Frame Code7)

```
function playerDefend(type:String,mSkill:String){
    var stas = "";

    var atk = calAtk(mSkill,dataMonster.AT,dataMonster.MG,dataMonster.plusAT,dataMonster.plusMG);
    var def = calDef(type,player.stats.DF,player.stats.MG,player.plusDF,player.plusMG);
    trace("type = " + type);
    trace("atk =" + atk);
    trace("def = " + def);
    var percentDodge = (dataMonster.SP / spP) *100;
    var ranDodge = random(100) + 1;
    if(percentDodge>95){
        if(ranDodge>95){
            stas = "miss";
        }
    //if(ranDodge >= percentDodge) stas = "miss";
    }else if(percentDodge<=95){
        if(ranDodge>percentDodge){
            stas = "miss";
        }
    }
    var damageATK = atk - def;
    if(damageATK <= 0) damageATK = 0;
    var dmFinished = 0;
    trace("Monster Skilll = " + mSkill);
    trace("player Skilll = " + type);
    if(mSkill == "ST" && type == "CT"){
        dmFinished = damageATK * 1.5;
        delMonster(Math.round(dmFinished),mSkill);
        return;
    }
    else if(mSkill == "AT" && type == "DF"){
        if(atk > def) dmFinished = damageATK;
```

Figure 3.90: Suwan Kingdom Game (Frame 1: Fighting Frame Code8)

- function playerDefend(type:String,mSkill:String)
  - Calculate the fighting result when the player is the defender. See Figure 3.90 and 3.91 for more code details.

```
    else if(mSkill == "AT" && type == "CT"){
        if(atk > def) dmFinished = damageATK * 1.25;
    }
    else if(mSkill == "AT" && type == "DM"){
        if(atk > def) dmFinished = damageATK * 1.25;
    }
    else if(mSkill == "AM" && type == "DF"){
        if(atk > def) dmFinished = damageATK * 1.25;
    }
    else if(mSkill == "AM" && type == "CT"){
        if(atk > def) dmFinished = damageATK * 1.25;
    }
    else if(mSkill == "AM" && type == "DM"){
        if(atk > def) dmFinished = damageATK;
    }
    if(stas == "miss"){
        des_txt.text = "Monster attack miss!";
        miss = this.attachMovie("miss","miss" + this.getNextHighestDepth(),this.getNextHighestDepth())
        miss._x = 65.65;
        miss._y = 286.45;
        miss.gotoAndPlay(1);
        setTimeout(nextRound,3000);
    }
    else if(dmFinished <= 0){
        soundEvent.gotoAndPlay("block");
        des_txt.text = "Player can defend";
        setTimeout(nextRound,3000);
    }
    else{
        delPlayer(Math.round(dmFinished),mSkill);
    }
}
```

Figure 3.91: Suwan Kingdom Game (Frame 1: Fighting Frame Code9)

```
403  function monsterDefend(type:String){
404      var ran = random(100) + 1;
405      var mSkill = "";
406      if(type == "AT"){
407          if(ran > 85) mSkill = "DM";
408          else  mSkill = "DF";
409      }
410      else if(type == "AM"){
411          if(ran > 85) mSkill = "DF";
412          else  mSkill = "DM";
413      }
414      else if(type == "ST"){
415          if(ran > 50) mSkill = "CT";
416          else  mSkill = "DF";
417      }
418      return mSkill;
419  }
420  function monsterSkill(){
421      var ran = random(100) + 1;
422      var mSkill = "";
423      if(dataMonster.att == "AT"){
424          if(ran > 80) mSkill = "ST";
425          else if(ran > 70) mSkill = "AM";
426          else  mSkill = "AT";
427      }
428      else if(dataMonster.att == "MG"){
429          if(ran > 90) mSkill = "ST";
430          else if(ran > 800) mSkill = "AT";
431          else  mSkill = "AM";
432      }
433      trace("mSkill " + mSkill)
434      return mSkill;
435  }
```

Figure 3.92: Suwan Kingdom Game (Frame 1: Fighting Frame Code10)

Figure 3.92 shows codes that used to calculate the monster skill which consists of these two functions:

- function monsterDefend(type:String)
  - Calculate the monster's defending skill.
  - var ran = random(100) + 1
    - The random value *ran* generated from the random function (+1) is used for checking which condition is matched for the monster skill value.
- function monsterskill()
  - Calculate the monster's attacking skill.

```
434  function calAtk(type:String,att:Number,mgg:Number,pAT:Number,pMG:Number){
435
436          var atk2 = 0;
437          var at = att * pAT;
438          var mg = mgg * pMG;
439          trace("pAT = " + pAT);
440          trace("pMG = " + pMG);
441          trace("att =" + att);
442          trace("mgg = " + mgg);
443          if(type == "ST"){
444              atk2 = (at *(random(6) + 15)) + (at * (random(5) + 1));
445              atk2 *= 1.5;
446          }
447          else if(type == "AT"){
448              atk2 = (at *(random(6) + 15)) + (at * (random(5) + 1));
449          }
450          else if(type == "AM"){
451              atk2 = (mg *(random(6) + 15)) + (mg * (random(5) + 1));
452          }
453         // if(type == "CT"){
454  //          atk2 = at * 100;
455  //          atk2 *= 1.5;
456  //      }
457  //      else if(type == "AT"){
458  //          atk2 = at *100;
459  //      }
460  //      else if(type == "AM"){
461  //          atk2 = mg * 100;
462  //      }
463          trace("atk2 = " + atk2)
464          return atk2;
465  }
```

Figure 3.93: Suwan Kingdom Game (Frame 1: Fighting Frame Code11)

- function calAtk(type:String,att:Number,mgg:Number,pAT:Number,pMG:Number)
  - Calculate the attack damage. More codes can be seen in Figure 3.93.

```
466  function calDef(type:String,dff:Number,mgg:Number,pDF:Number,pMG:Number){
467
468          var def = 0;
469          var df = dff * pDF;
470          var mg = mgg * pMG;
471          if(type == "DF"){
472              def = (df *(random(5) + 11)) + (df * (random(5) + 1));
473          }
474          else if(type == "DM"){
475              def = (mg *(random(5) + 11)) + (mg * (random(5) + 1));
476          }
477  //          if(type == "DF"){
478  //              def = df * 90;
479  //          }
480  //          else if(type == "DM"){
481  //              def = mg * 90 ;
482  //          }
483          return def;
484  }
```

Figure 3.94: Suwan Kingdom Game (Fighting Code12)

- function calDef(type:String,att:Number,mgg:Number,pAT:Number,pMG:Number)
  - Calculate the defend value. See Figure 3.94 for the code details.

```
485  function delPlayer(dm:Number,mSkill:String){
486      curHpP -= dm;
487
488      if(curHpP <= 0){
489          curHpP = 0;
490      }
491      if(mSkill == "AT" || mSkill == "AM"){
492          mcDamage = this.attachMovie("mcPunch","mcPunch" + this.getNextHighestDepth(),this.getNextHighe
493          mcDamage._x = 210.15;
494          mcDamage._y = 350;
495          mcDamage.gotoAndPlay(1);
496      }
497      else{
498          mcDamage = this.attachMovie("mcPunchStrike","mcPunchStrike" + this.getNextHighestDepth(),this.
499          mcDamage._x = 174.5;
500          mcDamage._y = 371.6;
501          mcDamage.gotoAndPlay(1);
502      }
503      des_txt.text = "Player have damaged " + dm;
504      if(curHpM > 0 && curHpP > 0){
505          setTimeout(nextRound,3000);
506      }
507      else{
508          setTimeout(finishedGame2,2000);
509      }
510  }
```

Figure 3.95: Suwan Kingdom Game (Frame 1: Fighting Frame Code13)

- function delPlayer(dm:Number,mSkill:String)
  - Check the current player hit point (HP). If the player HP is less than 0 then set his HP to 0. After that, check which movieClip will appear when the player is attacked by the monster in each condition. If the player HP is more than 0 and the monster HP is more than 0, then call function *nextRound* to check another condition, otherwise call function *finishedGame2*. Codes of this function are shown in Figure 3.95.

```
513 function delMonster(dm:Number,pSkill:String){
514     curHpM -= dm;
515     if(curHpM <= 0){
516         curHpM = 0;
517     }
518      if((pSkill == "AT" || pSkill == "AM") && player.job == "Sword"){
519         mcDamage = this.attachMovie("mcAttack","mcAttack" + this.getNextHighestDepth(),
520         mcDamage._x = 602.3;
521         mcDamage._y = 429.7;
522         mcDamage.gotoAndPlay(1);
523     }
524     else if(pSkill == "ST" && player.job == "Sword"){
525         mcDamage = this.attachMovie("mcAttackStrike","mcAttackStrike" + this.getNextHig
526         mcDamage._x = 608.9;
527         mcDamage._y = 422.45;
528         mcDamage.gotoAndPlay(1);
529     }
530     else  if(pSkill == "AT" || pSkill == "AM"){
531         mcDamage = this.attachMovie("mcPunch","mcPunch" + this.getNextHighestDepth(),th
532         mcDamage._x = 593.15;
533         mcDamage._y = 437;
534         mcDamage.gotoAndPlay(1);
535     }
536     else{
537         mcDamage = this.attachMovie("mcPunchStrike","mcPunchStrike" + this.getNextHighe
538         mcDamage._x = 609;
539         mcDamage._y = 432;
540         mcDamage.gotoAndPlay(1);
541     }
542     des_txt.text = "Player attack damaged " + dm;
543     if(curHpM > 0 && curHpP > 0){
544          setTimeout(nextRound,3000);
545     }
546     else{
547         setTimeout(finishedGame2,2000);
548     }
549 }
```

Figure 3.96: Suwan Kingdom Game (Frame 1: Fighting Frame Code14)

- function delMonster(dm:Number,pSkill:String)
  - Check the current monster hit point (HP). If the monster HP is less than 0 then set monster HP to 0. After that, check what movieClip will appear when the monster is attacked by the player by checking each condition. If the monster HP is more than 0 and the player HP is more than 0, then call function *nextRound* to check with another condition, otherwise call function *finishedGame2*. Codes of this function are shown in Figure 3.96.

```
548  function nextRound(){
549      round++;
550      if(round != 2){
551          if(turn == "player"){
552              atkMc.gotoAndStop("monster");
553          }
554          else{
555              atkMc.gotoAndStop("player");
556          }
557      }
558
559      else{
560          setTimeout(finishedGame2,3000);
561      }
562  }
563  function setDataPlayer(){
564      atP *= player.plusAT;
565      mgP *= player.plusMG;
566      spP *= player.plusSP;
567
568      atP = Math.round(atP);
569      spP = Math.round(spP);
570      mgP = Math.round(mgP);
571
572      atP_txt.text = atP;
573      dfP_txt.text = dfP;
574      mgP_txt.text = mgP;
575      spP_txt.text = spP;
576      lvP_txt.text = player.lv + " " + player.name;
577  }
```

Figure 3.97: Suwan Kingdom Game (Frame 1: Fighting Frame Code15)

Figure 3.97 shows codes of the following two functions:

- function nextRound()
    - Check whether the player and the monster have already played both of their roles. If it is not then switch the role. Otherwise, call function *finishedGame2* to check another condition.
- function setDataPlayer()
    - Show players' statuses on the Fighting Screen.

```
578  function finishedGame2(){
579      player.currHp = curHpP;
580      dataMonster.currHp = curHpM;
581      if(curHpM > 0 && curHpP > 0){
582          player.mode = 2;
583          saveGame();
584      }
585      else if(curHpM <= 0){
586          mcDead._visible = true;
587          des_txt.text = "monster dead";
588          var changeMode = checkParentPlayer();
589          if(changeMode){
590              player.mode = 2;
591          }
592          else{
593              player.mode = 1;
594          }
595
596          dataMonster.status = "die";
597
598          monsterDie();
599      }
600      else if(curHpP <= 0){
601          player.mode = 4;
602          player.snap = _root.snap1;
603          player.noTurn = 2;
604          playeDie();
605      }
606  }
```

Figure 3.98: Suwan Kingdom Game (Frame 1: Fighting Frame Code16)

- function finishedGame2()
  - Check the current monster hit point (HP) and the player HP.
  - If the player HP and the monster HP is more than 0 then assign the player mode to 2 (mode 2 means player is still fighting).
  - If not check the monster HP, if it is less than 0, the message "monster dead" will appear and the Movie Clip *mcDead* will display on the monster side, then, set the variable *changeMode* to the value of *checkParentPlayer()* function and check if this value equals to *checkParentPlayer()* function's value, if so change the player mode to 2, otherwise change the player mode to 1. Then, change *dataMonster.status* to "die" and call function *monsterDie()*.
  - Otherwise, if the player HP is less than 0, change the player mode to 4 (Player Dead); move player location to the castle point; skip the player turn for 2 times and call function *playerDie()*.

```
607  function monsterDie(){
608      popup_winlose.gotoAndStop(2); // win
609      player.exp += dataMonster.exp;
610      player.money += dataMonster.money;
611      var lvUp = checkLvUp(player.exp);
612      if(lvUp){
613          player.currHp = player.hp;
614          currHp = player.currHp;
615          setTimeout(showLvUp,5000);
616      }
617      else setTimeout(saveGame,5000);
618  }
619  function playeDie(){
620      mcDead2._visible = true;
621      popup_winlose.gotoAndStop(3); // win
622      player.exp -= dataMonster.exp;
623      if(player.exp < 0) player.exp = 0;
624      player.money -= dataMonster.money;
625      if(player.money < 0) player.money = 0;
626      setTimeout(saveGame,5000);
627  }
628  function showLvUp(){
629      lvP_txt.text = player.lv + " " + player.name;
630      popup_winlose.gotoAndStop(4);
631  }
```

Figure 3.99: Suwan Kingdom Game (Frame 1: Fighting Frame Code17)

Figure 3.99 consists of three functions as follows:

- function monsterDie()
  - popup_winlose.gotoAndStop(2)
    - Call Win pop up on screen.
  - player.exp += dataMonster.exp
    - Add the monster's experience to the player's experience.
  - player.money += dataMonster.money
    - Add the monster's money to the player's money.
  - var lvUp = checkLvUp(player.exp)
    - Set value *lvUp* to the value of function *checkLvUP()* and set *player.exp* to the value of *checkLvUp*.
  - if(lvUp)

- Check whether the player experience (exp.) reaches required *exp* for moving up. If so, restore player HP to full and call function *showLvUp*. If not call function *saveGame*.

- function playerDie()
  - Works similarly to *monsterDie()* but this function calls MovieClip *mcDead* and *minus exp.* and *money*.
- function showLvUp()
  - Show level up button for the player to increase his status.

```
632 function checkLvUp(exp){
633     var lvUp  = false;
634     if(player.lv < 25){
635      for(var i = 0;i<exp_ary.length;i++){
636          if(exp >= exp_ary[i]){
637              player.lv = 25 - i;
638              player.point += 5 * (player.lv - oldLv);
639              lvUp = true;
640              break;
641          }
642      }
643     }
644      return lvUp;
645 }
646 function playerGiveup(){
647     atkMc._visible = false;
648     player.mode = 3;
649     player.noTurn = 1;
650     dataMonster.currHp = curHpM;
651     popup_winlose.gotoAndStop(3);
652     setTimeout(saveGame,5000);
```

Figure 3.100: Suwan Kingdom Game (Frame 1: Fighting Frame Code18)

Figure 3.100 and Figure 3.101 consist of the following three functions:

- function checkLvUp(exp)
  - Check the player's experience whether it reaches the value that the player can upgrade his level.
- function playerGiveup()
  - Check whether the player chooses to give up or not. Show the Lose Screen if the player gives up and change the player mode to give up.

```
655 function saveGame(){
656     _root.keepPlayer[_root.turn] = player;
657     if(dataMonster.status == "die"){
658         dataMonster.currHp = dataMonster.hp;
659     }
660     if(player.mode == 4){
661         dataMonster.currHp = dataMonster.hp;
662     }
663     _root.dataMonster = dataMonster;
664
665     _root.finishedGame();
666 }
667
668 init();
669
670 stop();
```

Figure 3.101: Suwan Kingdom Game (Frame 1: Fighting Frame Code19)

- function saveGame()
  - Change turn to the next player turn. If the monster is dead then assign the monster HP to the default monster HP's value. If the player mode equals 4 (mode 4 means the player is dead) then assign the monster HP to the default monster HP's value. After that set function *dataMonster* equal *dataMonster* in *_root* (main frame) and call function *finishedGame* in root.

Chapter 4

Implementation

This chapter will give instructions for installation and game manual.

4.1 How to install

The game requires a Flash Player program. Typically, most computers today have Flash Player installed, so Flash game such as our game can be played without any extra work. However, for the computer that does not have a program, a user must install the Flash Player first. Otherwise the game can also be played through .exe file.

4.2 Suwan Kingdom Game

Figure 4.1: First Screen

Press "**START!**" to start game.

Figure 4.2: Title Screen

After, the player presses "**START!**" to start the game. The setup menu (as shown in Figure 4.2 will appear). This menu contains 3 buttons "**NEW GAME**", "**CONTINUE**" and "**QUIT**".

- "**NEW GAME**" is used to start a new game.
- "**CONTINUE**" is used to continue playing the saving game that the player was not finished playing and saving all information from last time.
- "**QUIT**" is used to quit from the game.

When the player selects "CONTINUE", the screen as in Figure 4.3 will appear. This page shows three sets of save data files that the user can choose to continue. The player can select one of the set. In this page, there are two buttons:

Back button: Click on it will bring the player back to the previous menu.

OK button: The player clicks this button to confirm the set of the saving data that he chooses.

Figure 4.3: Load Screen

Figure 4.4: Choose Player Number Screen

If the player selects "**NEW GAME**", the menu as in Figure 4.4 will appear. In this menu, the player can select the player number which was allowed up to 4 players.

Figure 4.5: Enter Name Screen

After the players have selected the player number, the pages as shown in Figure 4.5-4.7 will appear to ask each player to enter his name, his gender and his job respectively.

Figure 4.6: Gender Selection Screen

Figure 4.7: Job Selection Screen

For the job selection, there are four jobs which the player can choose: swordsman, magician, fighter and thief.

Figure 4.8: Guy Jobs

In Figure 4.8, from left to right are Fighter Guy, Swordsman Guy, Magician Guy and Thief Guy.

Figure 4.9: Girl Jobs

In Figure 4.9, from left to right are Fighter Girl, Swordsman Girl, Magician Girl and Thief Girl.

Figure 4.10: Color Selection Player 1 Screen

Figure 4.11: Color Selection Player 2 Screen

The player is allowed to choose his color which must be different from the already chosen color. As shown in Figure 4.11, if any color was chosen, this color will be disable so that other player cannot choose this color. To confirm his selection, the player must click "OK" button.

Figure 4.12: Before Random Screen

Figure 4.13: After Random Screen

After all player characters have been created, the display as shown in Figure 4.12 will appear. The player (one of them) must press the Next button to randomly determine play order. Then, the play order will be shown (Figure 4.13). After that, the setup process is done. Players can go to the starting page by pressing next button.

Figure 4.14: Start Screen

After the player presses "Start" button, the story will begin as in Figure 4.15. The player must press the next button to continue the story until it is finished.

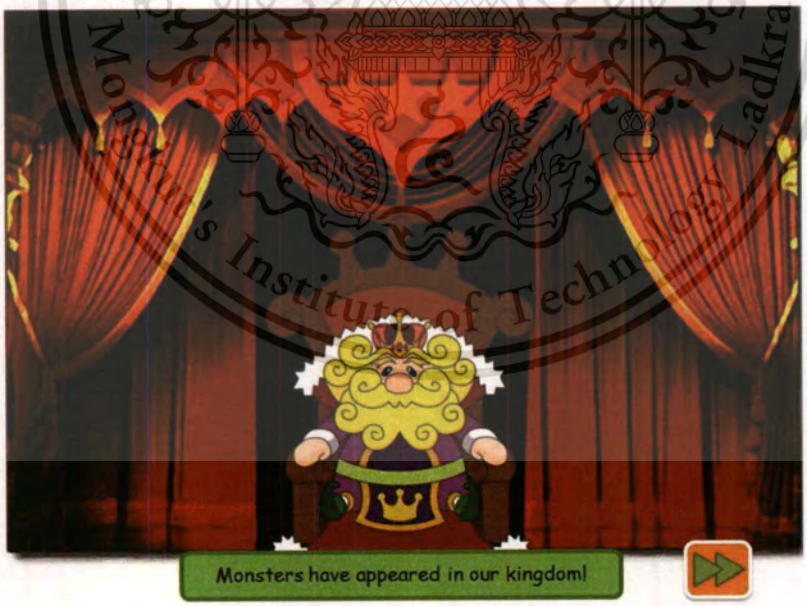


Figure 4.15: Story Screen

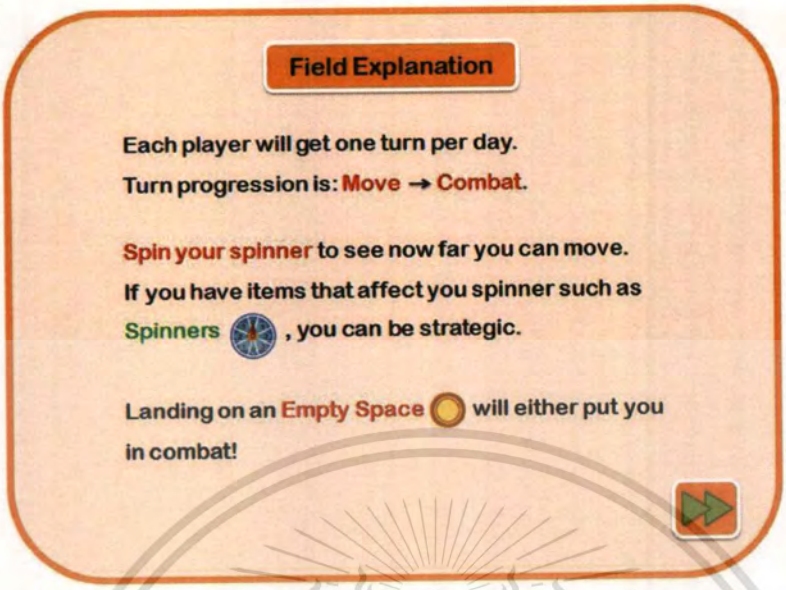


Figure 4.16: Field Explanation Screen

After the story ends, the field explanation screen as in Figure 4.16 which explains how players can explore the map will appear. When every player is ready to start exploring, one of them must press the next button.

Then, the first screen for the first player (player 1) will appear as in Figure 4.17 in which the player's name (Tee in this example) will be on the story dialog box. Apart from the world map which shows points that the player can go, there are 3 buttons in this page, "MOVE", "VIEW" and "MENU". "MOVE" is used for calling a spinner. "View" is used to view information on each point. "MENU" is used to go to save menu when all players agree to continue playing later.

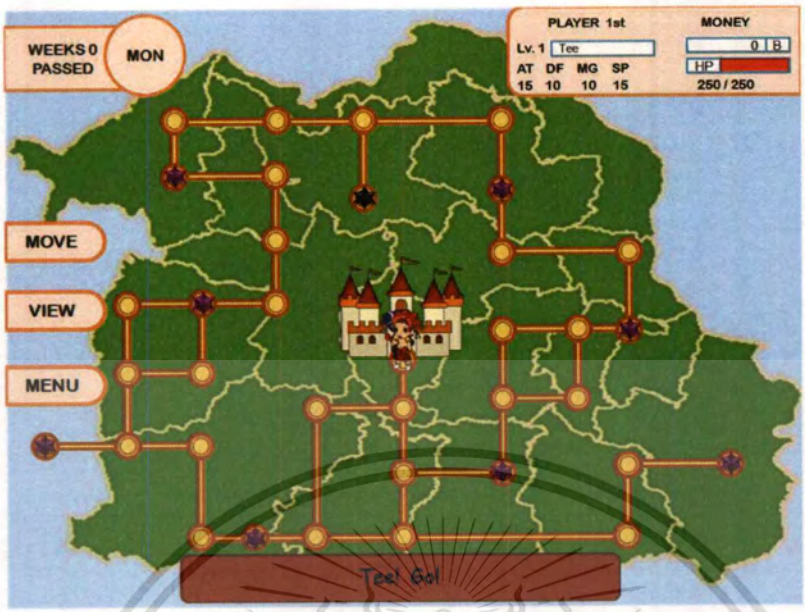


Figure 4.17: World Map Starting Point Screen

When the player clicks on MOVE button, the spinner as in Figure 4.18 will appear. The spinner functions work as a dice (the maximum point that player can walk each time is 3). To use it, the player simply presses it. Then, the moving number will appear on the top right of the screen together with the red arrow. The red arrows show possible points that player can move on each step he walks as shown in Figure 4.19. The red arrow will appear until the moving number equals 0. It is noted that one move represents one day.

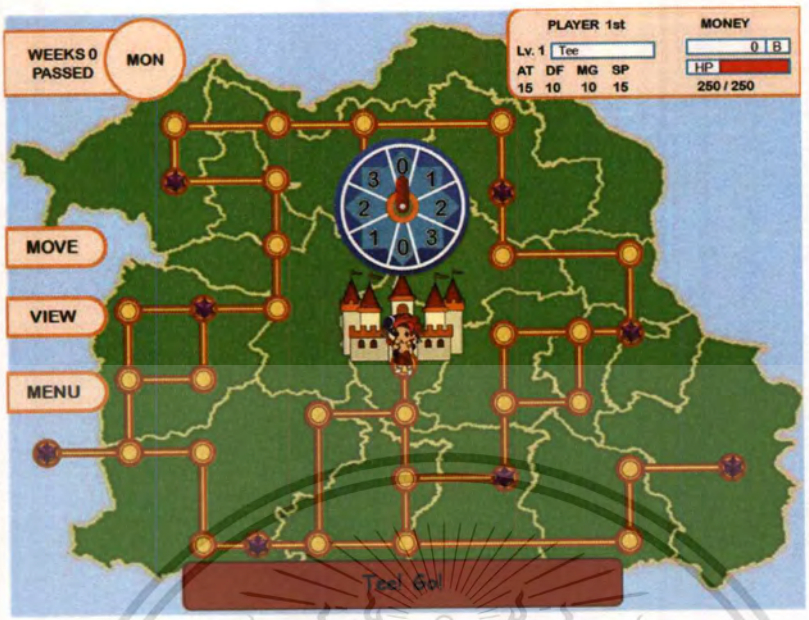


Figure 4.18: Spinner Screen

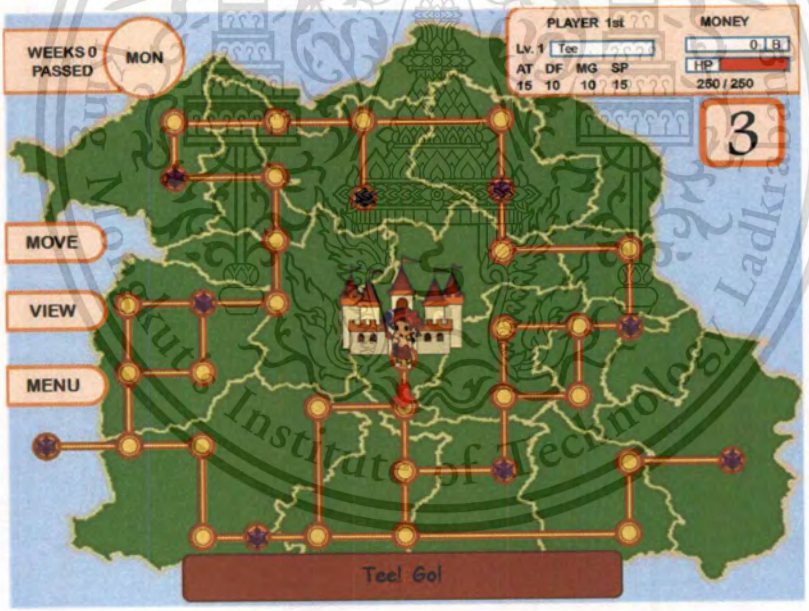


Figure 4.19: After Spinning the spinner Screen

Figure 4.20 shows the world map components when the player is on the move. Each number represents each component. The followings are their details:

- **Number 1:** Week and day
  - It shows the number of days and weeks that have already passed in the game. When all players' turns have passed (all players have played their turns), it means one day is passed. A week in the game also consists of seven days. After a week has passed, the game will restart the day to the beginning day again. Day contains with "Mon","Tue","Wed","Thu","Fri","Sat","Sun".
- **Number 2:** Player status.
  - Player status contains:
    - Player order.
    - Player money.
    - Player level.
    - Player name.
    - Player status like AT, DF, MG and SP.
    - Player hit points.
    - Player hit points bar.
- **Number 3:** How many moves that the player has left (moving number remaining)
- **Number 4:** Boss point which there are 9 points in the map (represented by the purple/black stars). Each boss owns one town. The player objective is to beat the boss to occupy their town.
- **Number 5:** Big boss point will be visible only when all bosses are defeated.
- **Number 6:** Town that the player has already occupied by beating the town boss.
- **Number 7:** Other players that are still fighting.
- **Number 8:** Give up player (Cannot move for one turn).
- **Number 9:** Dead player (wait two turns to revive).

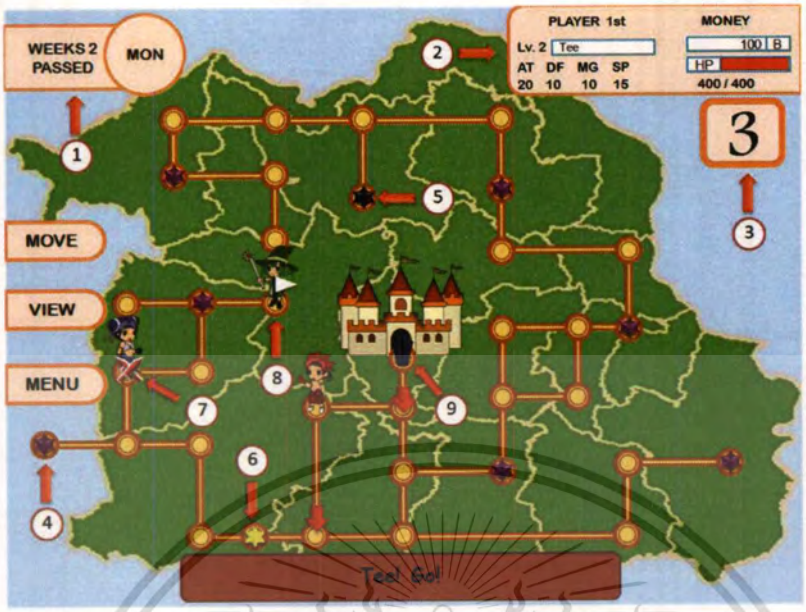


Figure 4.20: World Map Screen

There are four possible cases for the player location which results in a different fighting enemy.

- The player will stop at the empty point. In this case, the player will fight with the monster of that point.
- The player will stop at the boss point. In this case, the player will fight with the boss of that point.
- The player will stop at the big boss point (this point will be visible only when all bosses are defeated). In this case, the player will fight with the big boss.
- The fourth case is a special case. For this case the player will stop on the point that another player is already there.

Before entering the fighting mode, the screen as in Figure 4.21 will appear to explain the player about the battle method. That is in each battle turn, the player must do both attacking and defending roles (one attacking and one defending). This screen will explain what the player can do in each role.

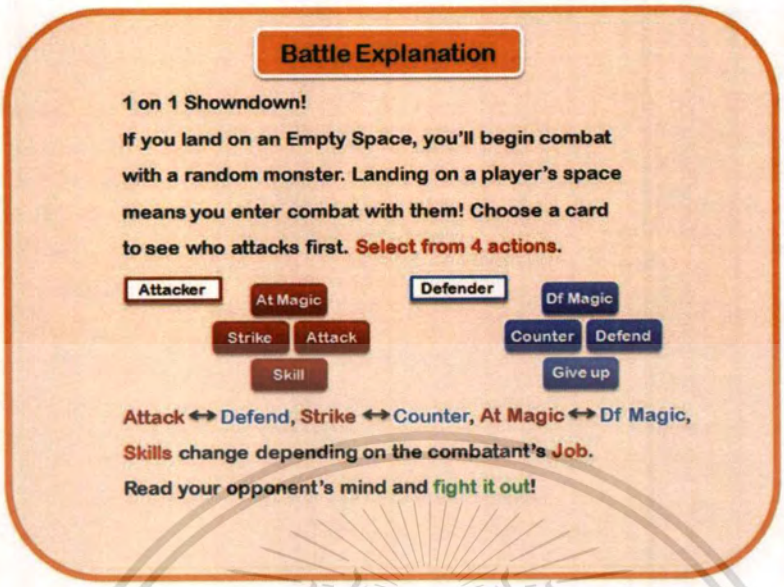


Figure 4.21: Battle Explanation Screen

Then, the next page (Figure 4.22) will determine the role that he will play first from the card that he chooses. In each role, there are four actions that the player can choose. His choice of action depends on his enemy type. Figure 4.23 and Figure 4.24 are examples of the attacking and defending role selection pages of the monster/boss (Slime is a monster).

Figure 4.22: Choose Card Screen

Figure 4.23: Choose Attack Action Screen

Figure 4.24: Choose Defend Action Screen

Figure 4.25: Winner Selection Screen

Figure 4.26: Winner Screen

Figure 4.27: Level up Screen

Figure 4.28: Loser Screen

After the player and his opponent executing both of their roles, the result (win or lose) will be shown. In case that it is a fight between the players versus another player, if the player wins, it will go to Figure 4.25 which the player must choose money or town for a reward before going to the winning summary screen in Figure 4.26. If the player level is also up, the screen as in Figure 4.27 will appear. In case that he loses, the screen as in Figure 4.28 describing what is taken (money or experience) will appear.

Figure 4.29: Choose Rival Screen

If the player goes to the point that another player is still fighting with the monster, the player will have to choose who will fight with before fighting as in Figure 4.29.

Figure 4.30: Town Owner Screen

Figure 4.31: Other Player Town Screen

Figure 4.32: Choose Action in Other Player Town Screen

If the player visits to town that he occupies, Figure 4.30 will display and the player's hit points (HP) will restore to full. If the player goes to another player's town, screens as in Figure 4.30 and Figure 4.31 will be displayed and the player has to choose between "Rest to recover your HP" (Pay (LV*100)) or "Do nothing" (HP is not restore).

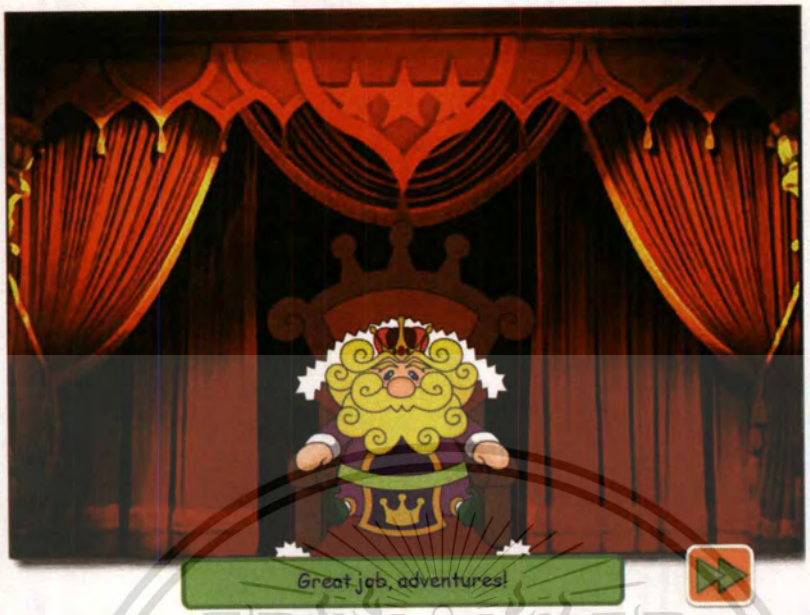


Figure 4.33: Ending Story Screen

Figure 4.34: The Ending Game Screen

After the big boss is killed, screen as in Figure 4.32 will appear. If the next button is clicked, the game ending screen as in Figure 4.33 will be shown.

Chapter 5

Conclusion

The main objective of this project is to create a game for entertainment called "Suwan Kingdom Game". It is a multiplayer turned based game which allows 2-4 players to play together. Some background music (BGM) and some of the characters costumes in the game are from Thai traditional style. Since the game instructions are in English, the game is not limited only to Thais but also encourages foreigners to play. This game also provides an opportunity for Thais to practice their English language skills while they read instructions and the words when they are playing. The game was developed by Adobe Flash CS3.

5.1 Problems

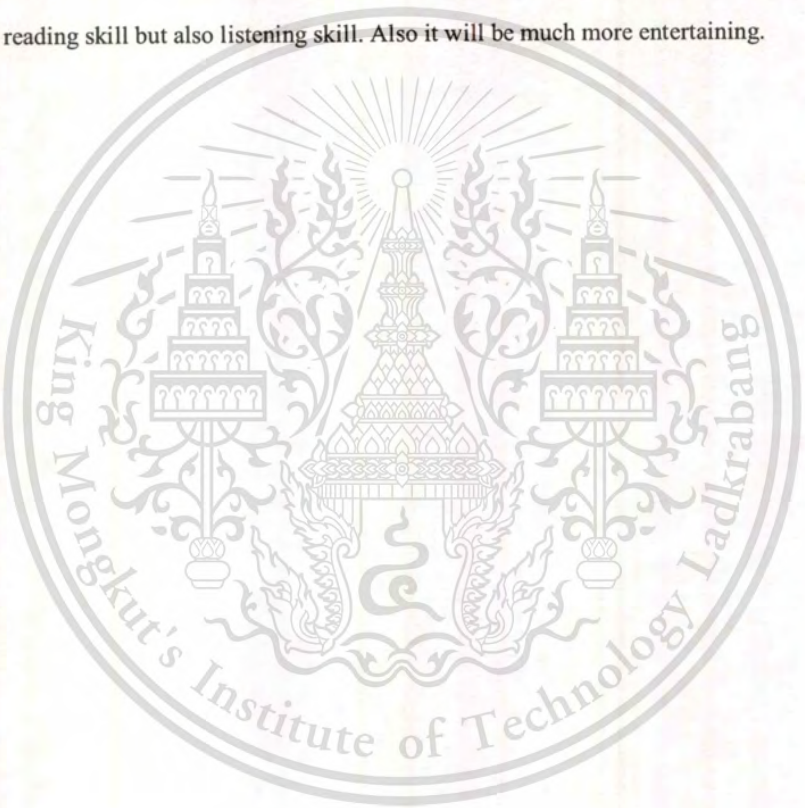
- In the beginning, we tried to implement our game using Irrlicht Engine [18]. However, it requires a strong programming skill in C++. It was quite difficult to implement our game functions on this engine. It took us several months to figure this out and changed to Flash ActionScript.
- Some functions are hard to implement. For example, we had designed our game to automatically walk the players to their targets. However, it was too complicated to include this function in the implementation. We hadn't decided our implementation (data structure) in a way to allow us to do this task.

5.2 Limitations

- Our game can only be played on the same computer one by one at a time.
- Since Flash ActionScript was new for us, it took us sometimes to understand how to write our game using this program. We believe that some of our implementation would be done better and faster if we had more experience in programming in ActionScript.

5.3 Suggestions

- The game should walk the player token automatically to the target that the player chooses.
- The game should be extended to the online game in which each player can play on different computers at the same time.
- For the game feature, we suggest to add the feature which allows the players to buy their weapons from the shop to make the game much more fun.
- We also suggest to add narration onto each screen so that the player does not only practice their English reading skill but also listening skill. Also it will be much more entertaining.



References

[1] http://en.wikipedia.org/wiki/Video_game_genres (Accessed: 11 December 2011)

[2] http://en.wikipedia.org/wiki/Action_game (Accessed: 11 December 2011)

[3] http://en.wikipedia.org/wiki/Action-adventure_game (Accessed: 11 December 2011)

[4] http://en.wikipedia.org/wiki/Shooter_game (Accessed: 11 December 2011)

[5] http://en.wikipedia.org/wiki/Adventure_game (Accessed: 11 December 2011)

[6] http://en.wikipedia.org/wiki/Role-playing_video_game (Accessed: 11 December 2011)

[7] http://en.wikipedia.org/wiki/Simulation_video_game (Accessed: 11 December 2011)

[8] http://en.wikipedia.org/wiki/Strategy_video_game (Accessed: 11 December 2011)

[9] http://en.wikipedia.org/wiki/Music_video_game (Accessed: 11 December 2011)

[10] http://en.wikipedia.org/wiki/Puzzle_video_game (Accessed: 11 December 2011)

[11] http://en.wikipedia.org/wiki/Sports_game (Accessed: 11 December 2011)

[12] http://en.wikipedia.org/wiki/Trivia (Accessed: 11 December 2011)

[13] http://en.wikipedia.org/wiki/Board_game (Accessed: 11 December 2011)

[14] http://en.wikipedia.org/wiki/Macromedia_Flash (Accessed: 11 December 2011)

[15] http://en.wikipedia.org/wiki/ActionScript (Accessed: 11 December 2011)

[16] http://docs.brajeshwar.com/as2/class-summary.html (Accessed: 11 December 2011)

[17] http://www.tex.ac.uk/cgi-bin/texfaq2html?label=MP (Accessed: 11 December 2011)

[18] http://irrlicht.sourceforge.net/ (Accessed: 24 June 2012)

[19] Kampol Leelaporn (2008) *flash ActionScript* PROVISION, Bangkok.

Appendices

Appendix A

How to install Flash Player

This game requires players to install Flash Player before playing the game. Flash player can be downloaded from Adobe website. After downloading, the player clicks on the set up file to install the program.

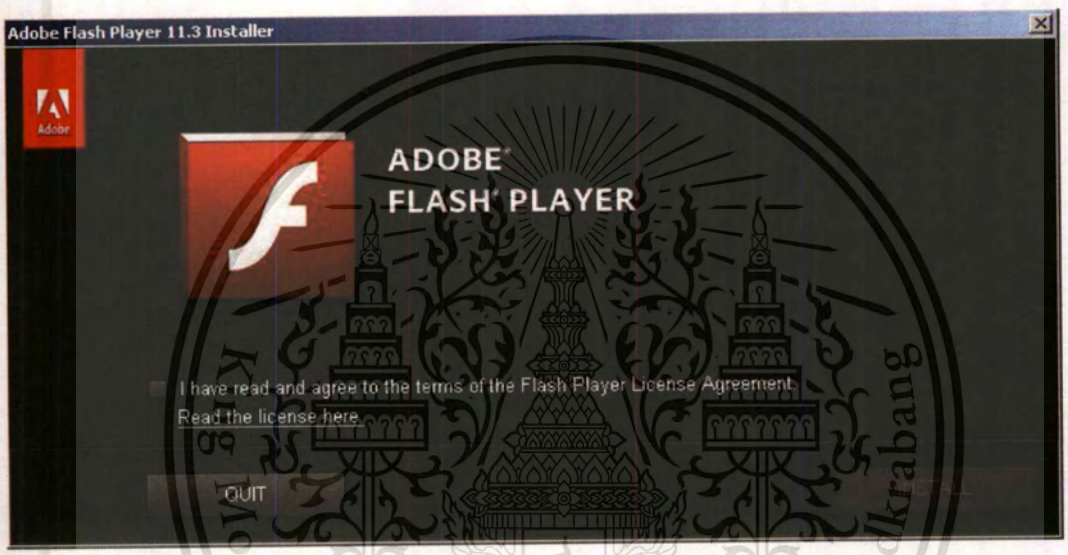


Figure A.1: Adobe Flash Player Installer

Appendix B

How to play game

After installing Flash Player, the player can click at our game file in red box to play a game.

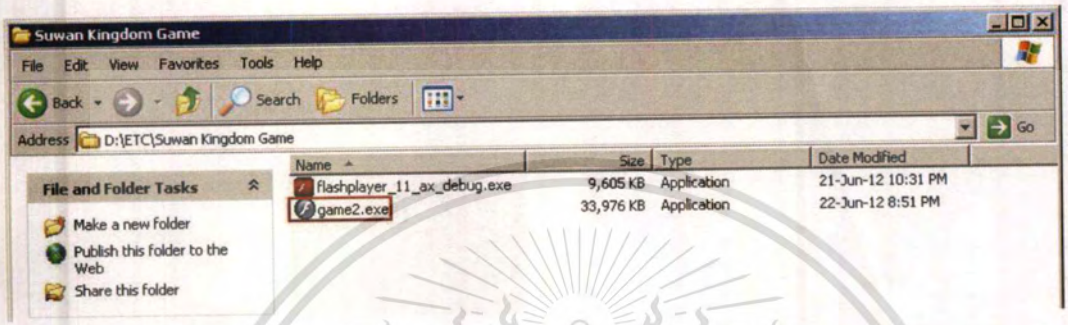


Figure B.1: Game File