THE UNIVERSITY OF AUCKLAND
NEW ZEALAND
Te Whare Wānanga o Tāmaki Makaurau

COMPSYS302 Java Project Report

TaeYoung Hwang

UPI: thwa475

ID: 725590943

KyungHo Chun

UPI: kchu916

ID: 6416306

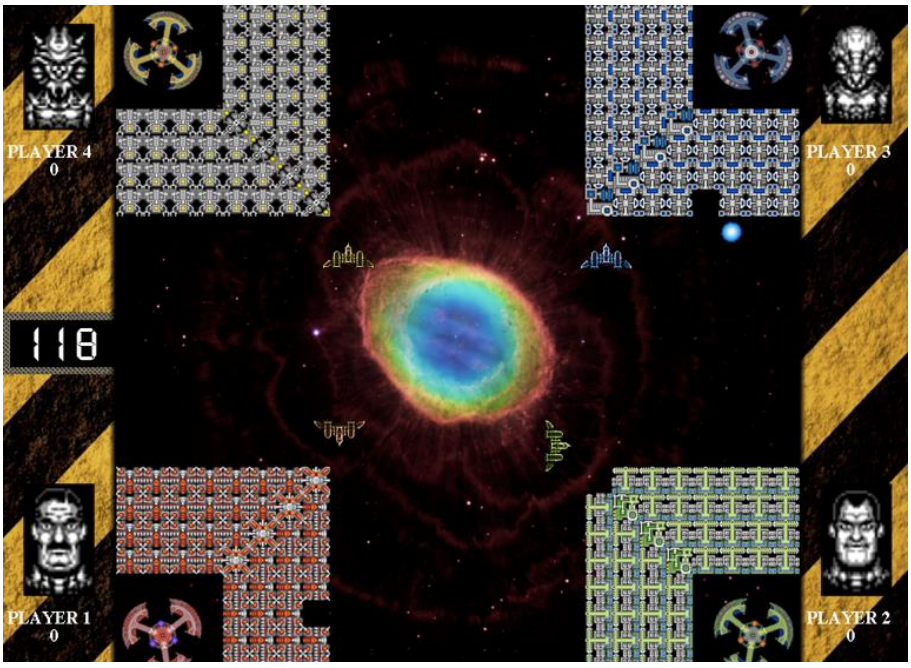

Table of Contents

Table of Figures

Introduction

The aim of this project was to develop a Warlord's game with basic Object-Oriented Programming, Threading and Concurrency, Graphical User Interface (GUI), Algorithm Development, and Software Project Development skills. To meet these goals the developers accepted a client's request to design and create a game for his twelve-year-old son.

This report presents a simple 2D game called "Warlords", which is based off of a video game (Same name) developed by Atari in 1980. A brief description of the requirements met, design aesthetics and general development discussions are presented in the following.

Meeting the requirements

This project was developed using the Java 8 language on the Eclipse IDE (Integrated development environment). The target audience for this game to appeal to the client and his son, the game is not confined to specific age group and is enjoyable by all. The visuals of the game were given a retro theme, but the game mechanics remained modern. This was to give the client who played the original game a feeling of nostalgia with the visuals, but also grab the attention of new consumers such as the client's son with a more modern and addictive gameplay.

When this game was developed, the developers were concentrated on creating the general game base first so we had a working basis to further develop the game on. Extra game features were added in after the basis of the game was complete.

When starting the Java application, the main game menu (Figure 2: Menu Screen) will appear first. Players are given the option to select between single player, multiplayer modes, demo mode, high score display screen and exiting the application. Within the multiplayer options, the player may choose two, three or four player mode. Empty player slots are controlled by an AI computer interface player. The main menu screen is navigated by using the up and down arrow keys (Exact buttons stated in README).

The game play starts with a "Get Ready" message and a countdown from three seconds. During this time, players will not be able to move their paddle. This has been implemented using a timer function which starts from three and decrements every second until it reaches 0. This helps to let the players get ready and not be surprised with a sudden start. Once the timer reaches zero seconds, the controls for each player's paddle is enabled and the game ball will spawn, moving in a random direction. A second timer function is initialized for the game timer, which begins decrementing from 120 seconds. When the game timer reaches zero seconds, the game will end (Figure 3: GamePlay Screen).

The walls of each player is initialized by giving each "brick" a value of one. When a "brick" is removed the value is changed to zero. This value of zero tells the system to not render the specific brick.

The game ball has been made to reflect (bounce) off of any boundaries (paddles, walls, game frame). This was developed by multiplying the X-direction of the ball by -1 when a vertical wall or object collides, this is same for when the ball travelling in the Y-direction hits a horizontal wall or object

collides. When a collision is detected with a wall, the wall is destroyed, when the ball collides with a player's king, this player's king and paddle is destroyed and controls of that player becomes disabled.

During a game a player is able to toggle the pause button by pressing the "P" key. When the game has been paused, the game time is stopped, the ball direction is temporarily saved and the movement is stopped. Pressing the pause button again continues the ball to move.

Detecting win condition has been made into two different methods, first when there is only one player remaining and secondly, when the timer has hit 0. There is an integer value called playerActiveCount which is initially equal to 0. This value is incremented by 1 for each new player initialized within the system (maximum 4 players) and decremented by 1 whenever a player dies. When this value is 1, the program is able to detect that the game is over through an "if" statement and sub sequentially the remaining player is declared the winner. However, in scenarios where the game lasted 2 minutes and there was more than 1 player remaining, there is a wallsRemainingAtEnd function which detects for the player with the most remaining bricks and checks to see if they are alive to even be valid to be declared winner. The player declared winner as a result will be the player who is still alive and has the most bricks remaining.

To build on this, when a player dies, their king and paddle is removed but their walls remain intact. This is to further increase gameplay for the remaining players.

Sound packages such as "java.applet.AudioClip" have been implemented through the use of the "Java.io.File" package. A variety of different sounds have been made to create an audio output whenever the ball has a collision with a wall, paddle or king. All sound clips were from open source websites and have been cited in the README file.

Bonus Game Features

- To give the players a sense of competition and accomplishment, a score counter was added. When an attacking player destroys a wall of another player, the attacking player is awarded five point.
- A new ball initializes when a play's king is destroyed, making the game more difficult. Two extra balls can be added if the player(s) want to by pressing the "1" and "2" keys.
- To help present the retro theme of the game, appropriate images and sounds have been used to help promote a sense of nostalgia for older players and also introducing the concept of retro gaming to the younger audience.
- A demo mode (Using four AI players) has been implemented to demonstrate the game to the clients.
- The menu screen is also functionally with mouse click inputs. This has been done by implementing the mouse listener class.
- The high score screen displays the high scores of players by reading a text file with a list of high scores.
- Paddle is able to curve in/out with corresponding up key for the specific player's controls.
- Threading has been implemented to allow for concurrency for audio output.

Top Level View of How the System Works

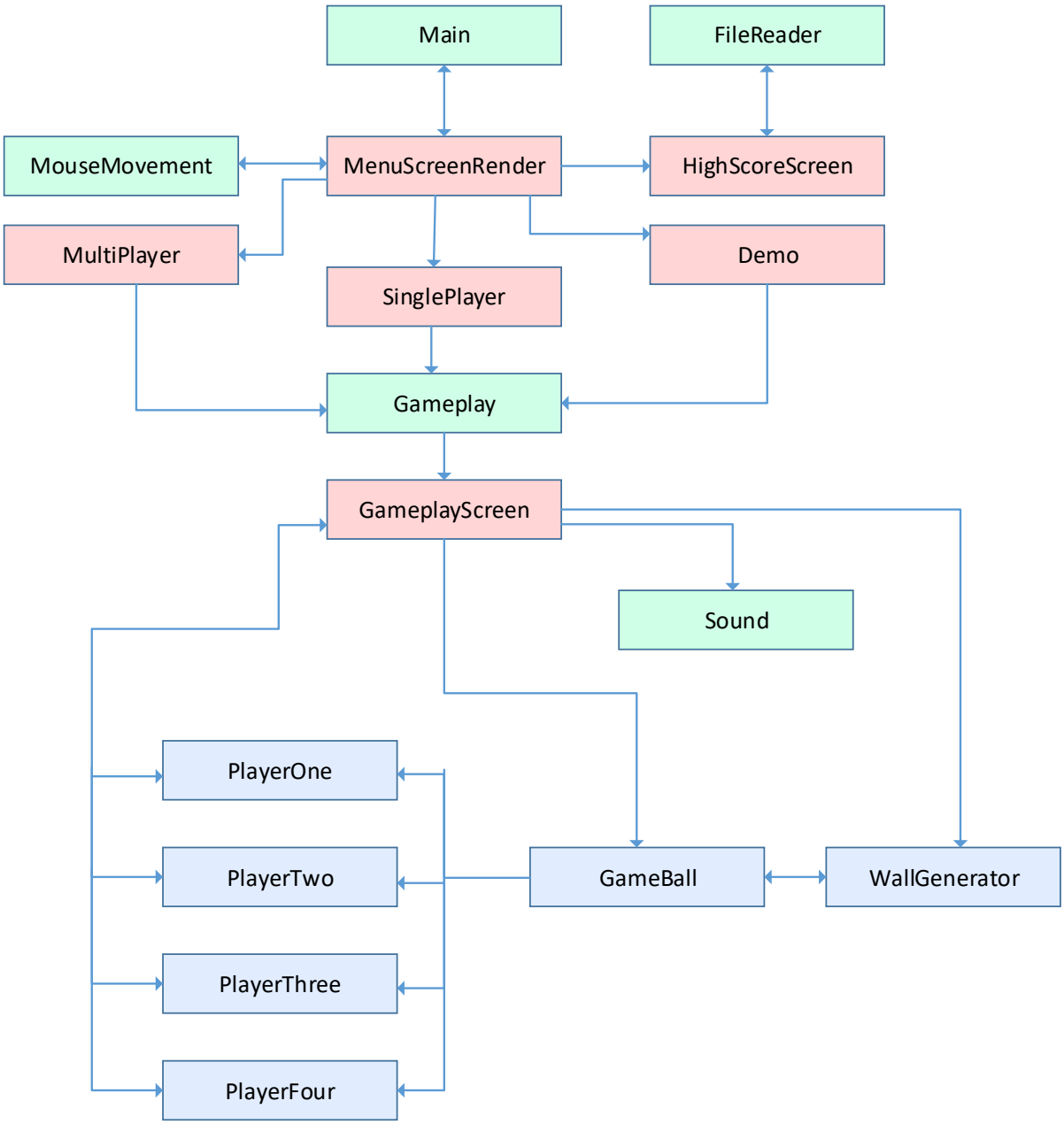

Figure 1: Top Level view

The Main function is initially run which calls the MenuScreenRender and from the MenuScreenRender, different game modes, high score and quit options are able to chosen through the MouseMovement class. If a game mode is chosen it is then gone through the Gameplay class and then the GameplayScreen class to call the players, GameBall, WallGenerator and Sound classes.

Significant Issues

During the development phase, implementing the sound became an issue. Initially for any given sound, it would only play once for the duration of the game. This was overcome by learning about threading and implementing it. This eventually lead to fixing the issue and being able to have sound effects throughout the whole duration of the game.

Another issue appeared during the development of the AI. The developers were very new to the concept of AI. However, this was overcome by a repetition of trial and error. The AI proved to be simpler, by making the paddles follow the all once it had come into a section of a screen, as opposed to following the ball around the whole screen. Another challenge regarding the AI came from making the AI intentionally slow/beatable. This challenge was overcome by slowing the movement based on the system timer.

Improving System Functionality

- Paddle movement is commenced by traversing through two arrays (X-horizontal array and Y-vertical array) of position values that decide the next position of the paddle. This array sets the maximum boundary parameters for the movement of the paddle instead of having to set a condition when the paddle went out of the boundary parameters
- E-numerator values have been used to represent the game's states, multiplayer states and players. The game state were used to change the screen of the game. For example, change to the Multiplayer screen and starting game modes with AIs.
- To improve the fairness of the game, all balls are spawned traveling in a random direction.
- Ball bounces off fairly after all collisions by simply inversing x direction value of the ball. This creates a simple reflective bounce effect of the ball.
- The AI computer players have been designed to be a moderate balance between being hard and easy. This allows for a more interactive and enjoyable gaming experience against the AI bots for the players.
- Use of static variables have only been used in scenarios where there can only be one instance of a class. Some variables and array components were set private where possible to decrease the chances of variables conflicting.
- Implemented a mouse click interface for players to use to navigate through the game menu.

Suitability of Tools Used

Java was a very suitable language for the development of the “Warlords” game. The language is very versatile and there are several active communities made for developers of Java.

JUnit testing was an effective tool in testing the functionality of the game design. For example, by creating a test case where a ball would spawn when a player’s king died was efficient to have just to make sure the implemented function was working correctly. It was also useful to test the code against provided JUnit test cases whenever the code was modified to see whether the specified function was still operating correctly after being modified. The basis JUnit tests provided were nice to base off to write even more test cases to see if everything in the game was functioning correctly.

Java itself was very convenient for game design, because of the inbuilt application development frameworks and various inbuilt packages. An example of this is the “KeyEvent” library, which simply allows the game application to process keyboard inputs.

Java Swing was used over JavaFX in this project this was because it was more fitting for the team member’s coding styles and Java Swing seemed more suitable to use for this game since Warlords is not a heavily demanding game in terms of graphics.

Git was a very useful tool throughout the duration of this project. Initially it took some time for the team members to understand and fully utilize Git. However, it was proven to be a very useful tool in software development when a team is working on a single project. Git allowed both team members to consistently work on the game without having to worry about how to merge and add the new and changed parts of the game code, especially with its capabilities of being able to detect conflicts when merging one another’s code. Git is a very resourceful and efficient tool that will definitely be used in the future for further development in group work projects.

OO Design + Cohesion and Coupling

This game has been developed through object orientated design. As mentioned before, Java is a very strong tool for implementing object orientated design applications because of its wide range of inbuilt import packages and system capabilities. Due to both team members having previous object orientated design practices, it was not difficult transitioning from C++ to Java’s object orientated programming methods and syntax.

Regarding cohesion levels of the game design, classes have been made to deliver high levels of cohesion. This has been done by creating functions in appropriate class even if this required creating more classes. In general, the classes are well fixed on implementing only the functions that they are required to do. For example, the GameBall class only has functions and variables relating to the ball functionality within the game.

Coupling levels of the game design have been kept minimal. This has been done by making sure the classes are well linked with one another whilst providing minimal dependency on each other. This has helped to minimize changing all other classes and functions whenever one small component was changed in a specific class.

Software development methodology

Throughout the project, the team has conformed to the Agile Software Development method. Initially, for the first period of the project, given the time-frame of the project start date to the prototype due date, a lot of work had commenced to try and meet some/most of the set objectives at the time. For example, trying to detect collisions between the ball the paddles.

Both group members had quick reviews of the projects between each stage of development where we identified the next set of objectives and analyzed code components that could be improved. When setting the objectives, the minimum specifications of the game were considered first before any other extra design features were added. As a result, the code developed for the game was very flexible and easily adjustable with minimal consequences if any. Most communication was done face to face which minimized confusion on design concepts and allowed for efficient progress of the development of the game. The team members valued communication as one of the top priorities for this project and this ultimately lead them to a satisfactory result.

Improvement for future development

- When team members are working separately on specific components it will be good to comment what is going on in the functions as they are developed. This creates less confusion for the other team member when the newest version of the project is pulled from Git and less time is occupied on figuring on what this new function is actually doing.
- This was not a huge issue in the duration of the project however in the future it will be good for team members to tell one another what components are being worked on so there is no situation where the team members are working on the same parts of the project. This ensures for the highest level of code development efficiency and less merging conflicts when working with Git.
- For the future it will be good practice to declare functions and variables as static only when absolutely required. It became quite frequent throughout the project to solve multiple errors by simply adding static to many of the functions and variables within the code. This is bad practice and was solved after the group members had become more educated on class accessibility.

- Another improvement for future development would be making a sprite packet made of the images used in the game, rather than calling individual images. Although this is not a particularly big issue within the scope of this project, in the future when designing higher end games, this will help to reduce the memory usage of the game.
- When designing games similar to Warlords, it would be nice to create more game modes such as fun modes i.e several balls spawning at once, two controllable paddles for a given player.

Appendix

Figure 2: Menu Screen

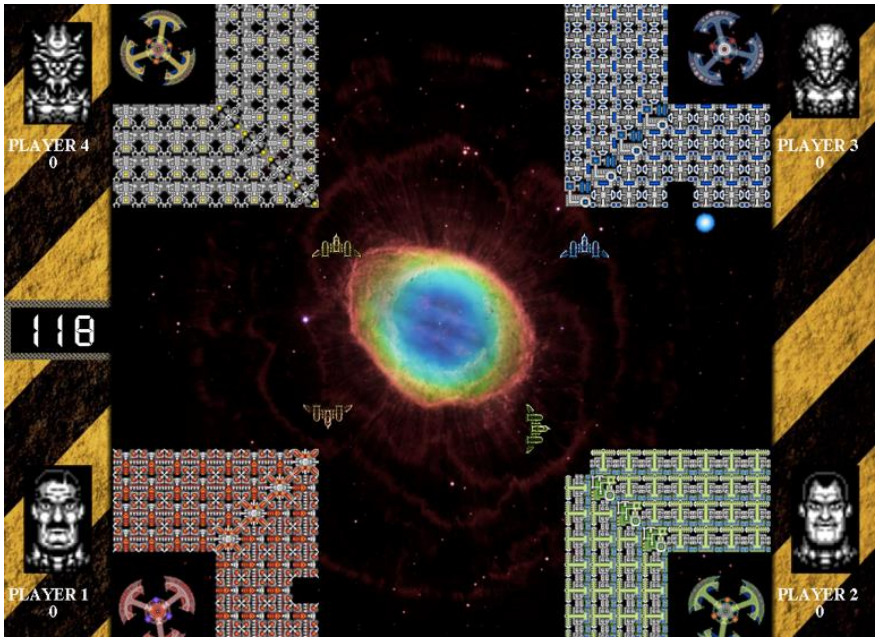

Figure 3: GamePlay Screen

Group Number: 2

	Features (minimum specifications = 50%)	Yes/No ?	Comments	Team member(s)*
0	Compiles and runs fine without errors/Code quality - comments, indenting, etc.	Yes	-	TH(50%) + KC(50%)
1	Welcome screen: select a game mode using keyboard, three game modes: single player (vs AI), local multiplayer	Yes	-	TH(50%) + KC(50%)
2	Start game: stationary paddles, countdown timer from 3, paddles should not be able to move	Yes	-	TH(10%) + KC(90%)
3	At least one ball should spawns with random velocity	Yes	-	TH(40%) + KC(60%)
4	Objects should not exceed 1024x768 boundaries	Yes	-	TH(50%) + KC(50%)
5	Hit registered when ball collides with wall, event(s) follow (e.g. wall being destroyed)	Yes	-	TH(50%) + KC(50%)
6	Ball should bounce off paddles and window edges predictably	Yes	-	TH(50%) + KC(50%)
7	Hit registered when ball collides with base, destroying warlord and related paddle	Yes	-	TH(50%) + KC(50%)
8	Game has two minute time limit (and a way to keep track of this)	Yes	-	TH(20%) + KC(80%)
9	Game can be paused/resumed with ‘p’, exited with ‘Esc’ back to main screen	Yes	-	TH(20%) + KC(80%)
10	Win condition evaluated, exit screen at end of game with summary, PgDn to skip to exit screen	Yes	-	TH(80%) + KC20(%)
11	Appropriate sounds played for any collisions	Yes	-	TH(100%) + KC(0%)
Design Elements (worth 50%)				
1	Scoring Function	Yes	-	TH(80%) + KC(20%)
2	Spawning balls when the king dies	Yes	-	TH(0%) + KC(100%)
3	Able to spawn extra balls when specific buttons are pressed	Yes	-	TH(100%) + KC(0%)
4	Demo Mode	Yes	-	TH(50%) + KC(50%)
5	2,3,4 multiplayer modes	Yes	-	TH(20%) + KC(80%)
6	Retro/indie theme images + sounds	Yes	-	TH(50%) + KC(50%)
7	High Score Screen + File reading	Yes	-	TH(90%) + KC(10%)
8	Mouse click functionality	Yes	-	TH(50%) + KC(50%)
9	Threading which allows for concurrency (Sound)	Yes	-	TH(50%) + KC(50%)
10	Paddle is able to curve in/out with corresponding up key for the specific player	Yes/No	Visuals are fully functional but gameplay functionality slightly off.	TH(0%) + KC(100%)