Sylvain Bechet, Mark Yep

Warlords

Final Submission Technical Report

compsys 302   
java game project

Group 01

Table of Contents

[System Requirements 1](#_Toc479497824)

[System Outline 1](#_Toc479497825)

[Development Issues 1](#_Toc479497826)

[Features 1](#_Toc479497827)

[Tools Used 1](#_Toc479497828)

[Coding 1](#_Toc479497829)

[Software Development Methodology 1](#_Toc479497830)

[Future Improvements 1](#_Toc479497831)

[Appendix 1](#_Toc479497832)

# System Requirements

How the developed system meets the requirements

# System Outline

Our system follows the Model – View – Controller (MVC) software structure. The model classes contain the basic data and implementation for each of the game elements, while the view classes handle the graphical user interface and the various visual gameplay elements. The main controller classes are used to link the models and views together, and provide overall logic and structure. A basic top down class diagram for our system is included in the appendix.

The WarlordsController class is the main controller class for our system. It contains the public static void main() function and is the class that is initially created when the system is run. Additionally, this class is responsible for creating the root window of the game GUI, creating JavaFX views from .fxml files, and linking with every view controller. This class is also responsible for creating instances of the game model and passing them into the game view.

We have model classes for the game, paddles, warlords, balls, and walls. Each of these classes implement their corresponding given interfaces and contain methods which override the given interface functions. These models mostly store data about each of their instances, such as their positions on the game screen, and if they have been destroyed yet. Additionally, they contain some public static variables, such as their sizes, which are utilised by other model and controller classes. These models also contain getters and setters for their private instance variables. The Game class also includes much of the logic for each in game tick. This logic includes the ball movement, ball collision detection, paddle movement, and some usable player abilities. The game logic could be moved to a separate game controller class in the future to more strictly follow the MVC structure.

Each view was designed using Gluon SceneBuilder to create .fxml files, which detailed layouts, shapes, sizes, and more. Each view has a corresponding view controller class, which handled graphical logic for that view. For example, the MainMenuViewController class handled the arrow key movement and dynamic keyboard menu selections for the main menu view, detailed in MainMenuView.fxml. The GameViewController handled the graphical logic for the gameplay itself, by linking the shapes on the screen to the individual objects of the Game instance. All the logic was completed separately from the view controller, which only handled displaying the results of that logic, in true MVC fashion. However, the GameViewController was responsible for counting in to the game, creating event handlers for key presses and releases, pausing and exiting the game, and timing the tick and countdown timer events.

# Development Issues

One or two significant issues during development and how they were overcome

# Features

Features that improve functionality of the system. Timers/multithreading, sounds, sprites, animation

# Tools Used

Discussion of the suitability of the tools for the application (e.g. Java, Git)

# Coding

Discussion on OO design and how cohesion and coupling issues were addressed

# Software Development Methodology

Discussion of the software development methodology (e.g. Test-driven Design)

# Future Improvements

Suggested improvements for future development

# Appendix

