2017

Age of Balls

CS302 JAVA GAME REPORT

HAMISH O'NEILL & ROMAN AMOR

Table of Contents

1 Introduction

We have been given the task of creating a computer game based on the 1980's Warlords Atari 2600 game. The aim of the game is to deflect the ball as much as possible using your paddle, in order to avoid it breaking your walls, and ultimately killing your warlord.

Our take on the game brings the graphics and sounds to a modern standard using Java 8 and JavaFX, whilst still maintaining the root aims of the original game. We also have augmented the classic game features with some extra additions such as powerups and a new story line.

2 Game Overview

Our game follows a theme of the ages through the Neolithic, Medieval and Industrial eras and into Space. Up to four players can play together using the keyboard and share their high scores online. Players move their paddle in a circular arc around their walls trying to deflect up to two balls that rebound along realistic lines.

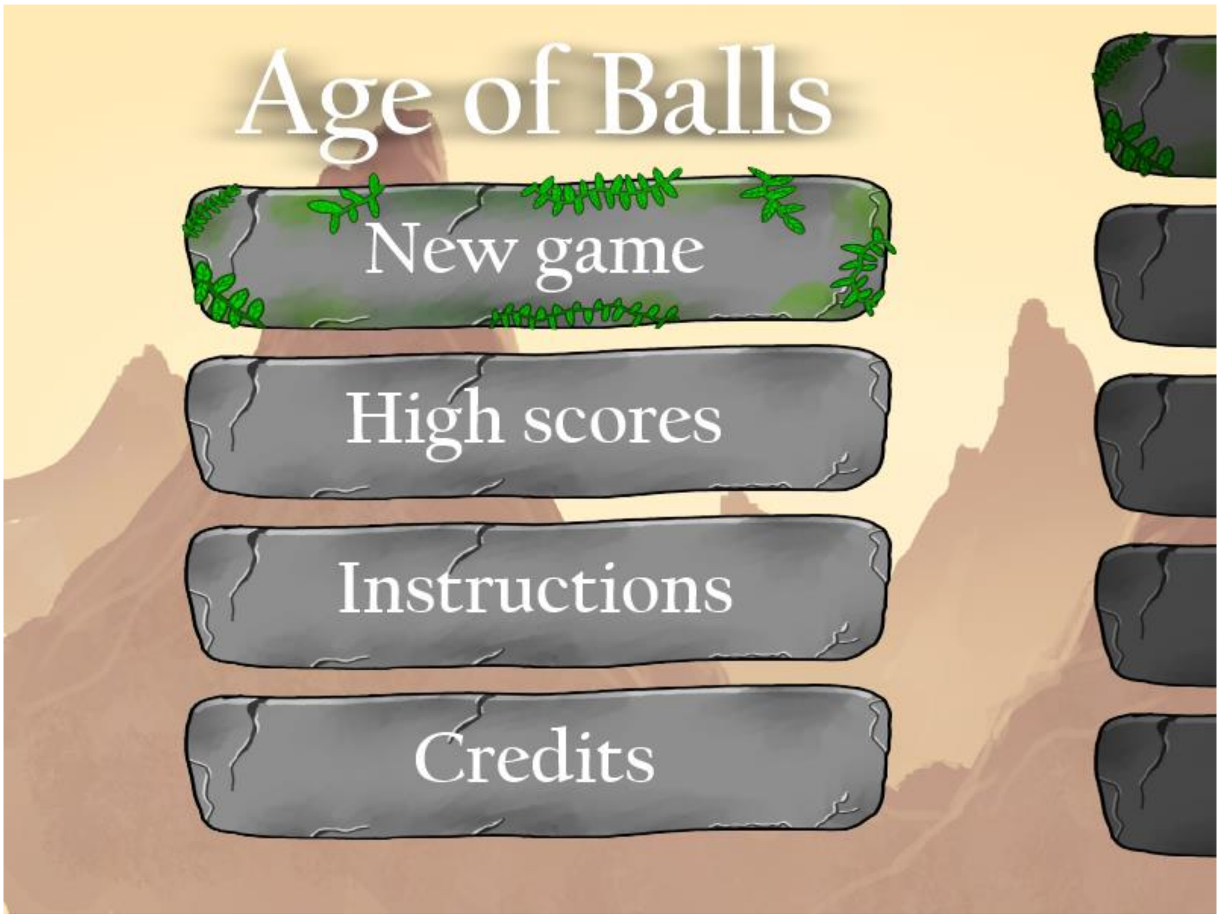
The aim to not let a ball hit your warlord and kill you, whilst trying to kill all of the other players as quickly as possible.

3 Base Features

3.1 Welcome and menu screen

We have built our own menu framework that draws directly to the JavaFX canvas. The user uses the keyboard to select items and move between nested menu screens. The transition between menu screens is animated and buttons are represented by visually pleasing rock images.

From within the menu the user chooses how many people are going to play the game and whether they would like to play a single game our use Campaign mode (see later). If they are playing a single game they can choose which age to use as the theme for the game. They also have the option to view the high score board and a credits screen.

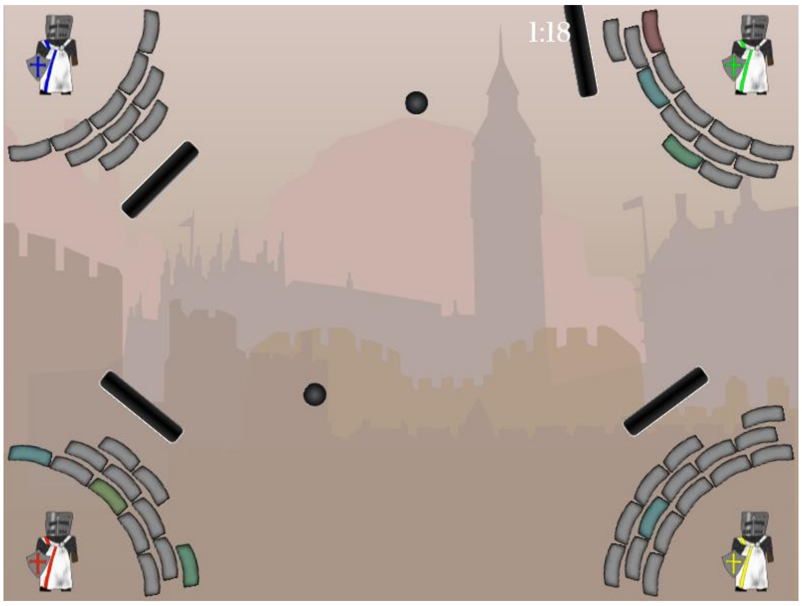


3.2 Start of game (single game mode)

After setting up the game in the menu the game will load with a themed background, walls, warlord and balls. The game will countdown from three then a ball (also in theme) will be spawned from the centre of the screen in a random direction.

3.3 Gameplay

When a ball strikes any surface (a wall, warlord, paddle or the screen edge) it will rebound at a realistic angle. Players move the paddle left and right using two keys each on the keyboard to deflect the ball. When a player's warlord is hit, the warlord dies and the player loses their paddle. When the ball strikes a wall, it is immediately destroyed. A different sound it played for each object hit to enhance the game experience.

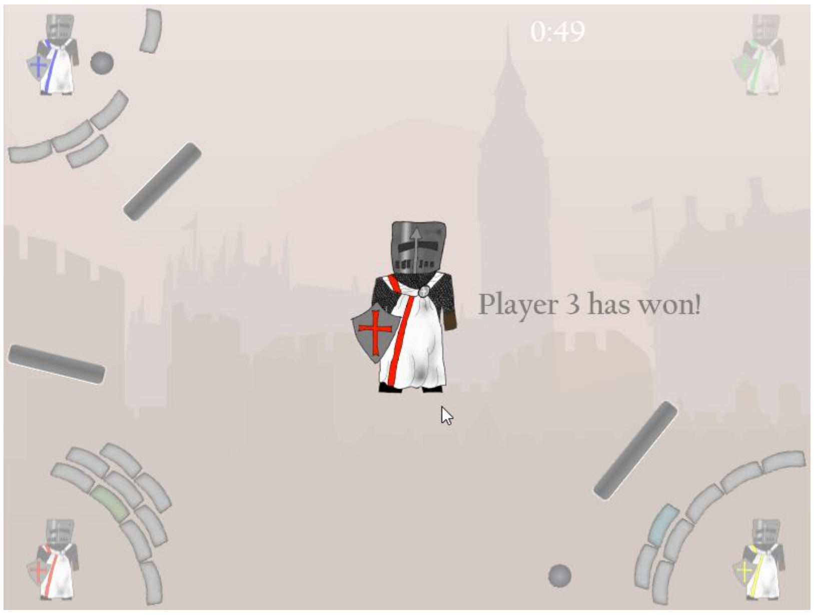


3.4 User control keys

Users can pause the game with the P key and exit the game using Esc. The user will be asked for confirmation when they try to exit the game before it has ended.

3.5 Game ending

The main way for the game to end is when only one player is left alive. The last player standing wins, but if the human players fail to beat our AI there is no winner. The game also has a two-minute time limit. After the time is up, the player with the most walls remaining wins. If multiple players have the same number of walls, it is a draw. Once the game ends the result of the game is overlaid over the game window, including a large version of the winner’s warlord when applicable.



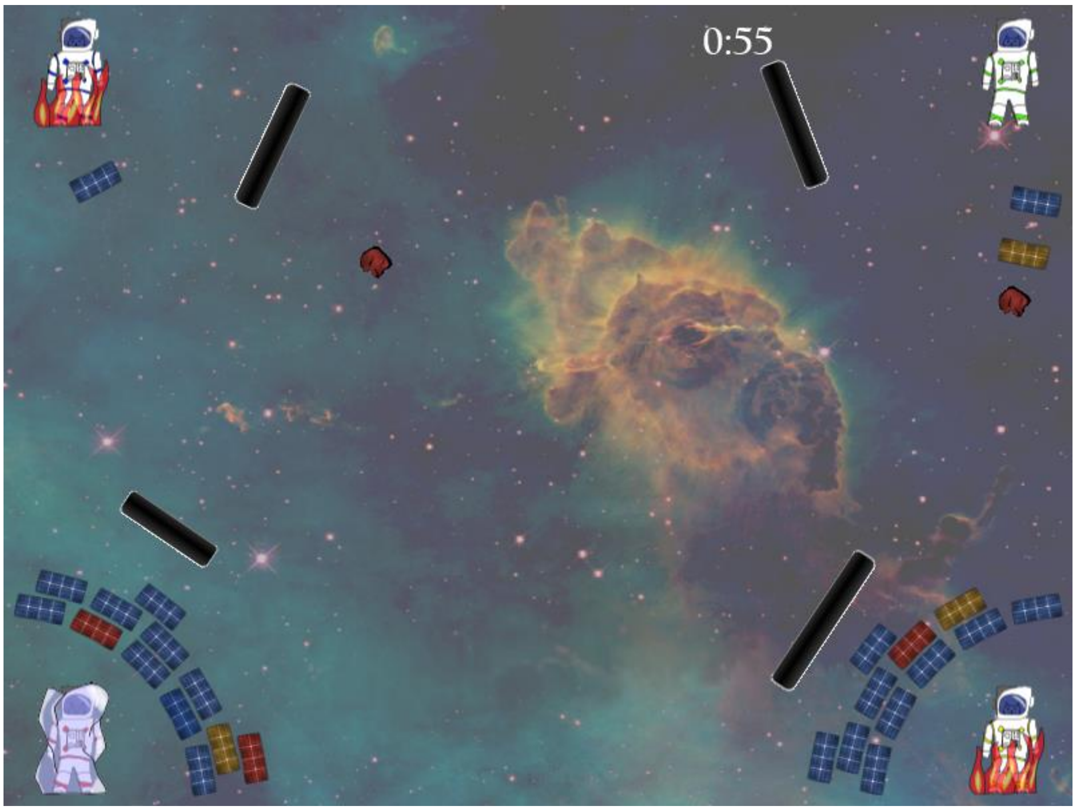
4 Extra features

We have added a number of extra features on to the basics of the game, this is to improve the playing experience and enjoyability of the game. These extra features include:

4.1 Power-up system

The walls surrounding each warlord can be randomly generated with a power-up within them. When a ball breaks any of these wall, it inherits this powerup. If the powerup relates to the ball, it is applied instantly - speeding up or slowing down the ball, if it relates to a paddle, it is applied to the next paddle that that ball touches. The paddle can be made to move faster or slower and to grow or shrink in size.

As seen below, a player with a fast paddle appears on fire, and a player with a slow paddle appears in an ice block.



4.2 Extra balls

More balls can be added throughout the game in order to increase the level of difficulty. In order to keep the difficulty of the game reasonable, we currently add a second ball randomly in the first minute of gameplay.

4.3 Number of players

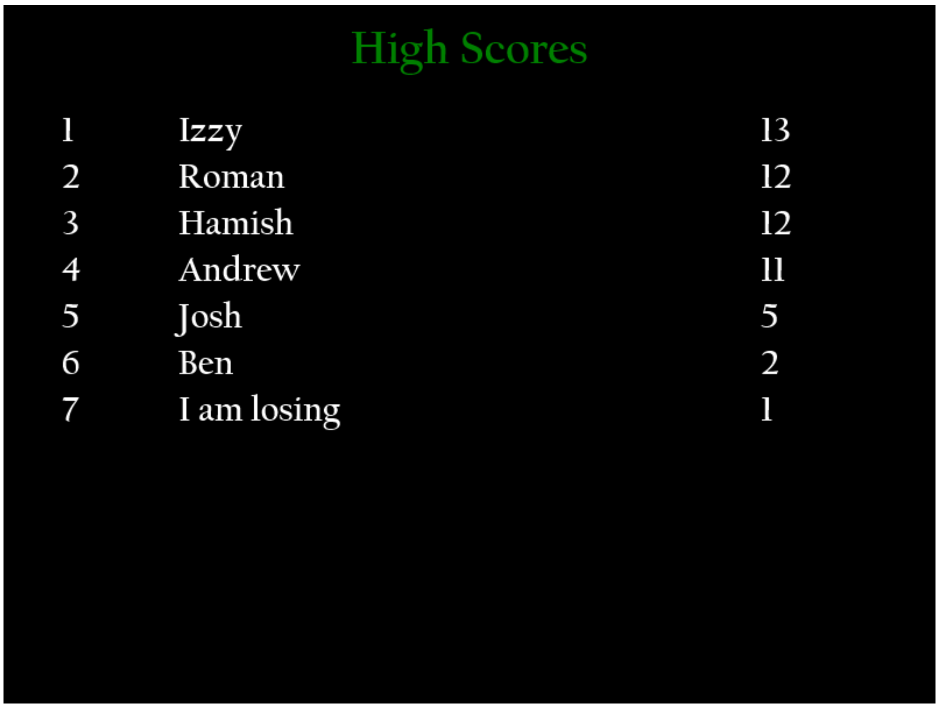
The game can be played with any combination of AI and human players that constitutes 4 total players.

4.4 Game modes

We have implemented 2 different game modes, Campaign and single play. Both of these modes can be played with any number of players with the difference being that games played in single play mode will be considered for high scores and games played in campaign mode will progress through the story line instead.

4.5 High scores

We have implemented a high score system which is web linked, this allows any player to compete with any other players globally, this means that players can still have competitions and championships across their neighbourhood, even if they are not together.

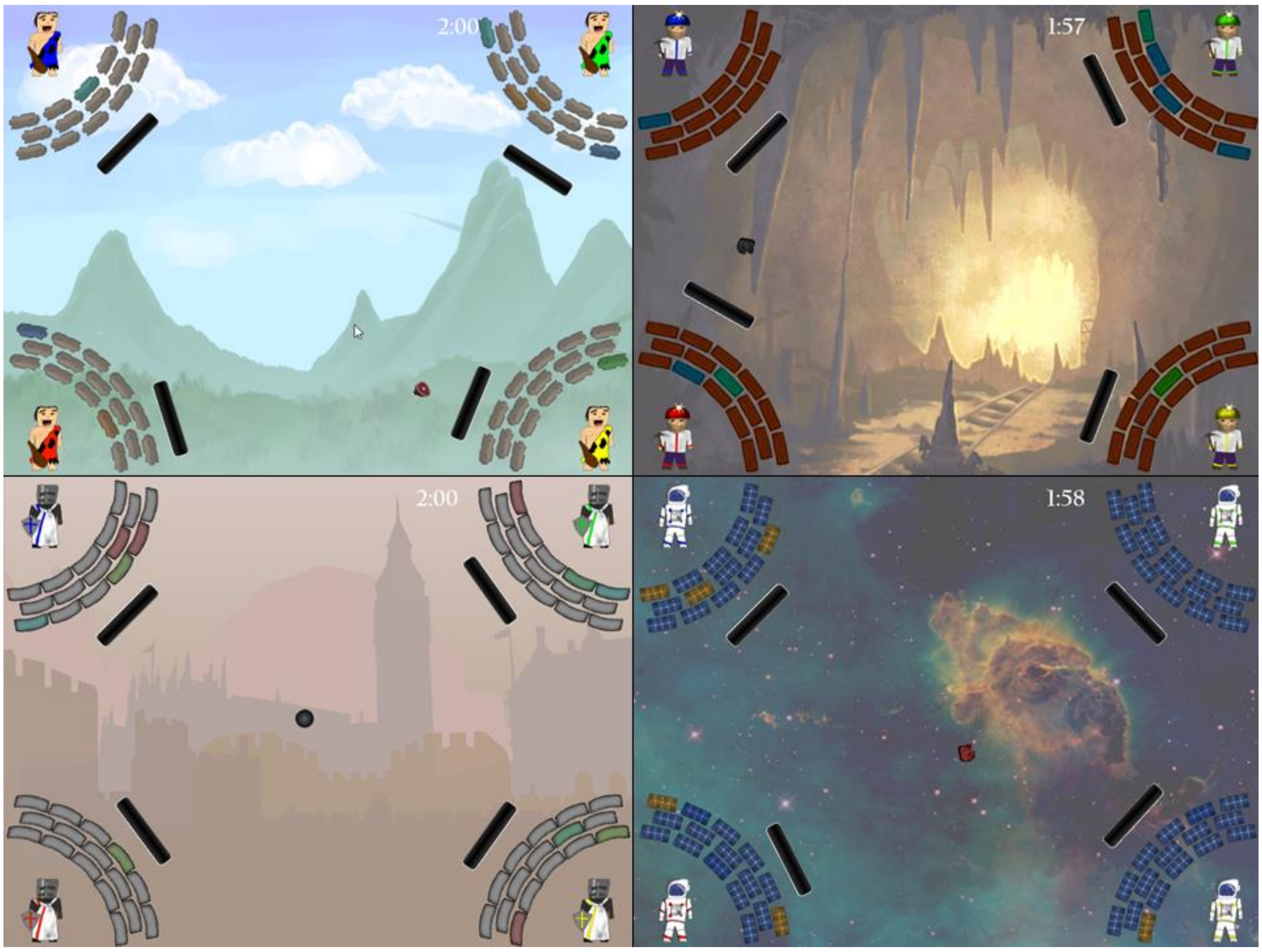


High Scores

1	Izzy	13
2	Roman	12
3	Hamish	12
4	Andrew	11
5	Josh	5
6	Ben	2
7	I am losing	1

4.6 Ages

The game is based around 4 ages in time; Neolithic, Medieval, Industrial, and Space. If the player is using the campaign mode, the player will progress through all of these as they follow the story line.



5 Technical tools used

Our game is written in Java 8 using the JavaFX GUI framework. Within JavaFX we are mostly using the canvas element which gives us low level control over what is drawn where. This means we are not taking advantage of any pre-written layout or transition engines. To make the menu screens we are calculating the pixel coordinates to draw images at ourselves. We were also missing other features found in proper game engines such as a physics engine.

The advantage of this approach was the very small learning curve as the canvas only has a relatively small number of easily understood methods. On the other hand, it often felt like we were reinventing the wheel and doing more work than should be necessary to get the layout and game dynamics working. Being able to build the game using a full game engine and with access to external libraries would we believe have allowed for more rapid development (at least after the initial learning stages).

6 Challenges during development

The biggest issue we encountered during development was implementing our own collision detection, we managed to mostly resolve our issues but it is still not quite perfect. Our development of the collision detection encountered issues right from the first implementation, the basis of all these issues revolved around the fact that the ball could leave the screen or enter a paddle. Initially we were able to mitigate this by implementing a more complex collision detector that calculates a vector of the ball path and determines where this intersects with another object.

We then ran into the issue that a ball can become stuck within a paddle. This was possible because the paddle could move over the top of the ball, thus trapping it within its bounds. We managed to mostly resolve this by stopping a paddle from moving if it’s immediate movement will make it intersect with the ball. While this mitigated the majority of the issues, we did still need to implement some safety net code to move the ball out of a paddle if it ever enters, and move the ball back into the game area if it ever leaves.

We decided that all of our game sprites would be developed in house in order to get our desired theme, we didn’t however expect it to take as much time as it did, and as such was a challenge to get all of them made while still having time to implement features in the game itself.

7 Software design

We have used the Model-View-Controller (MVC) framework along with object orientated design to build the game. We have a large continuous loop running up to 60 times per second (depending on system performance). Once the game has started the following processes occur in each iteration of the loop:

- User input is read (including from computer controlled users).
- The paddles are moved according to this user input.
- Each ball is moved forward one iteration of its velocity. As part of this process we check if the ball will collide with any other objects (using vector intersections) and its motion path is determined accordingly.
- All of the possible game ending conditions are checked.
- All objects are rendered to the screen.

Following MVC we have the game logic within controller classes. There is one coordinating controller (MainController) that passes off control of the screen to MenuController and GameController as appropriate. Information to be passed between these controllers is stored in the Game model class. There are also a number of other model classes, including one representing menu items and one for each type of game object.

Object orientated programming principles have greatly enhanced the quality of our design. All of the drawable game objects extend from an abstract class to avoid repeating the same implementation. The KeyboardInput class is accessed through a generic user input interface which will allow alternative input methods to be easily substituted in the future (including AI). An example of when OO was helpful was when we moved from one ball to multiple. Having a ball class with high cohesion meant we could easily just create another instance of the ball without having to re-write the entire game.

8 Development Methodology

While we only have a team of two developers, we have used agile inspired methodology. Neither of us took any particular role in the software development (although Roman did manage the graphics and audio), instead we worked together to develop the game functionality. We started with an initial plan but evolved the schedule and feature set as time went on.

We used the git version control system which allowed us to work on related parts of the project separately then easily merge the change sets. We made sure to document all tasks and bugs in the Bitbucket issue tracker which ensured we did not accidentally work on the same tasks and kept us on track. We employed a continuous integration (CI) build server so we could easily see if our commits were passing their unit tests.

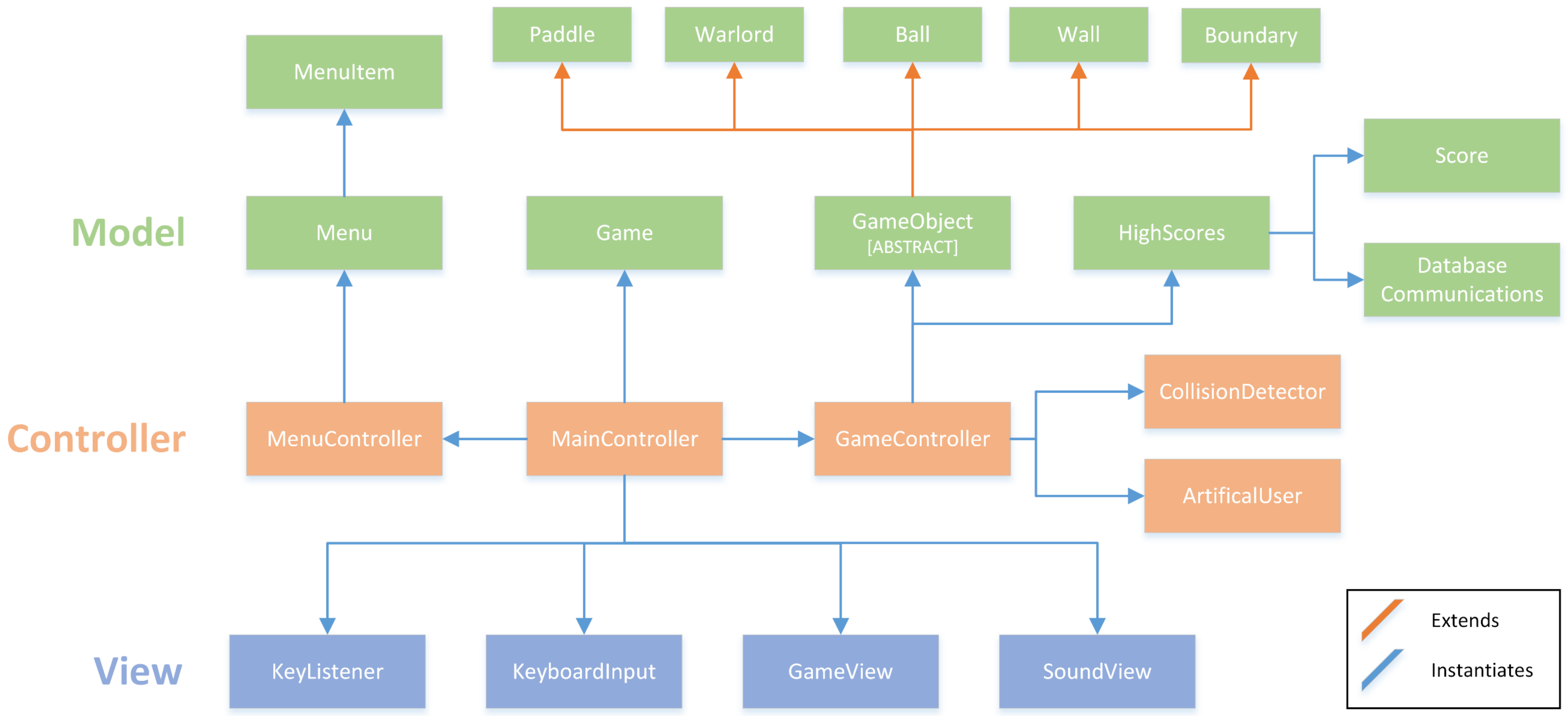
9 Suggested improvements

Firstly, we would like to work to improve the existing game code. We would continue working on the collision detection system to improve the interaction between paddles and balls that are both moving. We would also like to set up much more thorough testing for our classes. One benefit of this would be safe refactoring (because of regression tests), for which the first target would be our GameController class which currently assumes many responsibilities (meaning it is quite long).

There are also additional features we would like to implement. Currently we only have four levels which will not take players long to complete. We would like to complete more levels and also make the game dynamics and difficulty change between levels. A major feature that we believe should be implemented is an online multiplayer mode which will make it possible to play with friends without having them physically present.

Appendices

Class relationship diagram


Model
Controller
View
MenuItem
Paddle
Warlord
Ball
Wall
Boundary
Score
Menu
Game
GameObject [ABSTRACT]
HighScores
Database Communications
CollisionDetector
MenuController
MainController
GameController
ArtificalUser
KeyListener
KeyboardInput
GameView
SoundView
Extends
Instantiates