12 / 04 / 2017

COMPSYS 302 Final Report

Adil Bhayani

Sakayan Sitsabesan

Table of Contents

Aim of Project

The aim of this project was to design a game as a Java application that would be suitable for play by the client's 12-year-old son. The game was to be designed so that it can take input from the user and respond accordingly. A set of minimum specifications was provided in the project brief and our aim was to deliver a game which not only met those requirements; but to also continue to develop the game so that it was unique through the implementation of additional features.

Minimum Specifications

The project brief outlined the basic set of requirements that our project must deliver. These requirements include:

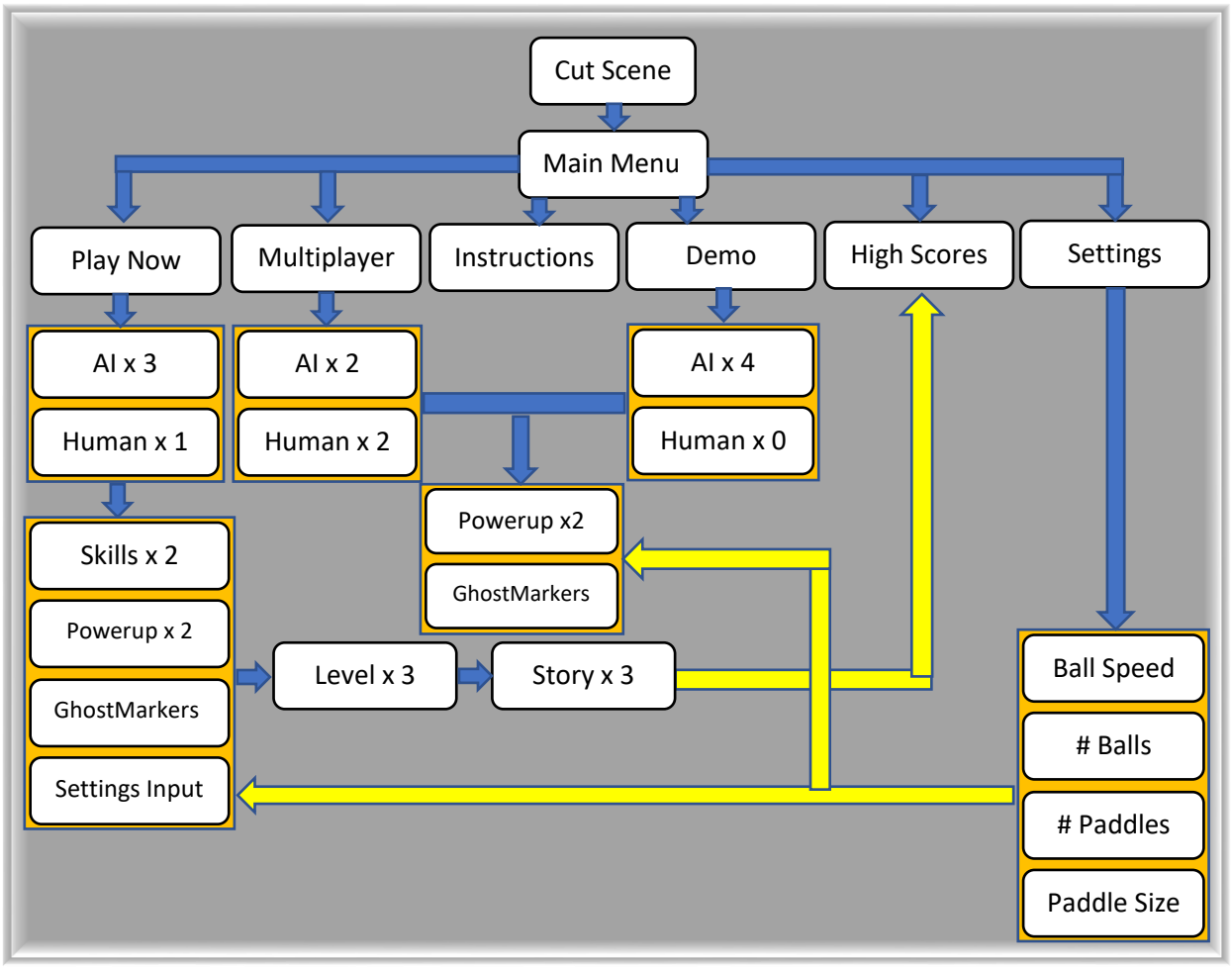
- A welcome screen
- Four stationary paddles and countdown timer before game starts
- Spawn of a ball in the centre of the window with a random direction
- All objects were to be drawn within the 1024 x 768 window
- Destruction of walls when the ball collides with them
- Ball must bounce off paddles in a predictable way and must also bounce off window edges.
- When ball collides with the warlord, the warlord must be destroyed and their paddle must disappear.
- Maximum of two minutes per level and mechanism for keeping track of this must be shown to the user.
- Pause and escape functionality
- Appropriate win conditions
- Appropriate sounds for collision with objects

These minimum specifications were used in our project as a base to build our game upon. It would be difficult to implement any additional features without having first implemented the minimum specifications. Hence the earlier part of our work was focused solely upon building a game which would meet these requirements. The minimum specifications were completed well in advance to allow us enough time to focus on building a game which was truly unique and fun to play.

The win condition of our game is based on the number of bricks that are remaining when a round ends. The player with the most number of bricks is declared the winner upon round completion. Due to the difficulty for a 12-year-old to successfully defeat three AI controlled paddles it was decided that it would suffice to simply have more bricks remaining than the AIs.

The minimum specifications also require that appropriate sounds be played for collisions with objects. Due to the amount of time that would be required to create our own custom sounds it would not be feasible to implement the feature without acquiring the sounds from the web. Care was take to only use images that were available in the public domain or had an attribution copyright. If they had an attribution copyright, they were also given credit to as part of our created game.

System Top Level View



The system has been designed so that the options that are configured by the user in the settings screen are reflected across all 3 game modes, play now, multiplayer and demo mode. Once the user finishes play now mode their high score is added and can be seen through the high scores menu. The main menu also links to the credits scene which has not been included in this diagram. Detailed diagrams included class diagrams can be found in the appendices.

Issues During Development

A significant issue that occurred during development was the lack of proper communication and cooperation among team members. This was a result of the little commenting that both team members did while writing their sections of the code. This meant that when the other person started reading code, they had no idea what was going on and development was stalled till the person who originally wrote the code is available to explain what it does. The lack of cooperation in the development meant that functions implemented by one member were often re-written by the other member as the original was not implemented in a flexible enough manner for future use.

Once this issue was recognised we began to minimise its impact by clearly commenting code especially in more logically complicated segments of code. These comments were being written

not just to write comments for the sake of marks but rather as a means of communicating thinking as the code was written. Later it was also found that well commented code allowed even the author of the code to quickly refresh what the purpose of written code is.

Clear separation on who implements which features was also not made. This could have been problematic if working in a team with a greater number of members. However, since this project was completed in pairs this did not pose much of a problem. But to minimize this risk further we consulted the other person before beginning to work on a feature. This issue is also strongly linked to the lack of proper communication. Solving the issue of communication ultimately leads to solving this issue too.

In terms of difficulty in creating functionality the feature which caused the most issues was being able to create accurately implemented collisions. To be able to model collisions realistically the code needs to consider the direction of the ball, the velocity of the ball and how it hits the paddle. Collisions between the paddle and the ball were the most difficult to create. This is because both objects are in motion and adds another variable into the equation.

To be able to fix this issue a lot of work was put into implementing the Bresenham Line Drawing Algorithm. After this method was implemented collisions were much better than previously and we could identify which axis of movement to invert. However often the speed and angle of the ball are such so that multiple collisions occur and not just one. This meant that the ball would still be bouncing unpredictably and the minimum requirements would not be met. To further improve on our existing code, we added a method in the ball class which would only allow collisions to occur if the ball has not been hit in the last 5 ticks. This would mean that only the first collision would be registered and the ball would bounce accordingly.

Features that Improve Functionality of the System

Our system incorporates many additional features to make the game more than just a task that was completed for a university paper. The added features make the game interesting and above all fun to play. All features have been designed to make every game different from the one that was previously played. The additional features include:

- Embedded YouTube videos which act as the intro scene or the cutscenes between the 3 levels. These videos help to make the story engaging and will be highly appealing to a 12-year-old.
- Two single player skills which can be changed via the up/down arrow keys and activated via shift when the general is alive. An icon next to the user's base indicates which skill is selected and whether it is ready for use or on cool down. The first skill causes all balls in play to become explosive (indicated by yellow ball) and completely destroy a layer of bricks when it hits one. The second skill allows the user to hold a ball and shoot it in a controlled direction through the implementation of a rendered arrow that is drawn on the screen.

- When a user kills an AI general the skills are reset and the user can choose another skill to activate. This helps the user to continue to win and enables strategic use of powerups to be able to obtain victory.
- The settings screen allows the user to configure settings to either make the game easier or much more difficult but fun at the same time. These options include changing the ball speed, increasing the number of balls, the number of paddles and changing the paddle size. When there are two paddles the second paddle reflects the first one. All options configured through settings affect all game modes.
- Two powerups which are activated when a ball passes over them. The first powerup increases the speed of the ball that activated it. To reflect this change in speed the colour of the ball changes. After a few seconds the ball slows down to original speed and the colour change is removed. The second powerup increases the paddle size of all generals that are alive. This powerup when combined with multiple ball gameplay and two paddle gameplay make the game very exciting and fast paced.
- Upon death gameplay can be quiet boring. All game modes incorporate a mechanism where a dead general becomes a ghost marker and generates powerups for the other generals which are still playing. This allows users to influence the game even after death while also increasing gameplay speed as the round progresses. When an AI general dies they too begin controlling a marker and generate speed powerups too. This makes the gameplay even faster since the ball is likely to pass over a speed up.
- It is understandable that all these various features can be confusing for a 12-year-old, thus to demonstrate the gameplay and to help them quickly learn to play an AI demo mode has also been included. The demo mode has all the features that can be used by the player except the single player skills. Since the options configured through settings influence the gameplay in demo mode too, they can provide a good indication to the user in terms of how to play.
- High scores can be viewed through the main menu of the game. These scores are retained across multiple sessions, even after closing the game. They add a sense of competition which is known to make games more addictive. The name for the high score is entered upon the start of single player mode and is then retained permanently using a data file which retains the data of all game plays.

These features have been designed specifically to link together with the other features to form gameplay which feels immersive and entertaining.

Suitability of the Tools Used for the Application

The game was implemented in the Java programming language using JavaFX as the graphics library. Some of the advantages of using Java for this project include its object orientation, cross-platform support, code reuse, ease of development, tool availability, reliability and stability, good documentation. Some disadvantages of Java include non-deterministic memory management which can lead to exceedingly high amounts of RAM usage even for a simple 2D game such as ours. If this game was to be played on a console which many users prefer as opposed to PC then it would not be easy to write the game in Java due to lack of support for it on consoles.

Git was used in the project to make the development of the game project streamlined. Advantages git provides include Backup and Restore, Synchronization, Short-term & Long-term undo, Track Changes, Track Ownership, Branching and merging, and a larger sandbox for testing new features.

In this project git proved to be very useful because it provided us with the ability to always be up to date with the changes that the other person has made. It was also very crucial in aiding with the merging of files when both of us had changed the same file at the same time. Through the use of branches we were also able to separate our code for the prototype from that of the final submission. Features such as these make git ideal for all forms of code related development.

OO design and how Cohesion and Coupling Issues were Addressed

Object oriented design was use in this project along with the MVC pattern. This means that the user can only see information that is displayed by the view and their actions are interpreted solely by the controller. The controller then manipulates the model which then leads to changes in the view. MVC is used because it forces the programmers into planning the code they write to have a better flow. The MVC patterns involves separate types of classes layered together as Models, Views and Controllers. The MVC pattern encourages high cohesion and low coupling.

High cohesion is achieved by logical grouping of related classes under the Model-View-Controller sections. Low coupling is part of the very essence of MVC programming as models, views and controllers should be easily replicable by substitute classes. These patterns and designs were used to structure and model the code in this project. Each of the MVC sections were separated into separate packages and code was kept clearly separated yet together with code of similar nature.

As Java is an object-oriented language, there was little choice to use any other approach to software design. OOP allowed us to ensure that changes when made to a given model are reflected across all instances of that model. E.g. a change to the size of the ball will be reflected across the balls that are present in game. This made it easier to prevent inconsistences in the system which can be very difficult to resolve if not using OOP.

Overall the use of OOP and MVC patterns proved to be quite effective in designing and implementing this game.

Software Development Methodology

The design of this game was done using an Agile development methodology. Agile focuses on minimising risk in software development by working in sort time periods known as sprints. These sprints are of a specified period and each sprint has some tasks which must be completed within it. Since agile is very flexible and can often be used in a manner which allows it to be suitable for most software development projects including ours.

Since our project had a very short deadline our sprints were extremely short and often ended within one to two days only. We would decide on the feature we would like to add or a bug we would like to fix and spend the next couple of days working towards it. Future iterations would then focus on improving something else until our goals were met.

At the start of our project we also used a test-driven development approach. Test-driven development allows us to ensure that what we are building will ultimately be ideal for the cases in which it will be used. When mistakes are made at the start of a project and are left unnoticed till closer to the end serious problems occur and a large amount of refactoring is usually required. To ensure that our code's foundation is written to meet the test cases that it should adhere to, we used test driven development at the start of our project but focused on implementing features closer towards the end.

Suggested Improvements for Future Development

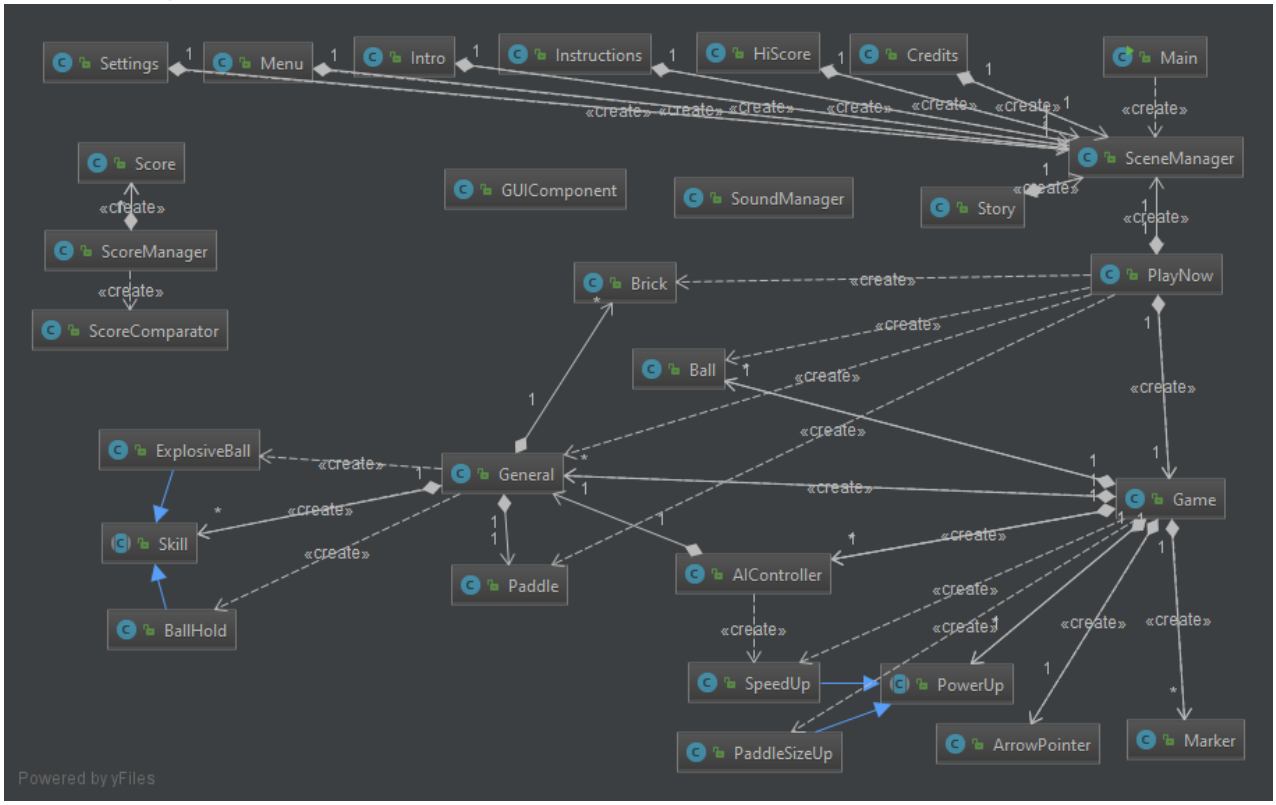
In future iterations of the system, it is planned to add more features which allow the user to be more influential in the gameplay after death. Since the victory condition in our game is that the general with the most bricks remaining wins, we would consider implementing options which allow the user to possibly continue to deflect balls once they become a ghost marker. Or being able to generate powerups which when passed over by a ball cause the bricks to become invincible for a short amount of time.

Currently the skills are only available in single player mode. To make multiplayer mode more exciting and fun we would consider refactoring the code to allow both players to use their skills. The addition of more skills which allow the stealing of enemies' walls or temporally decreasing the speed of movement of the other generals would also be potential development options.

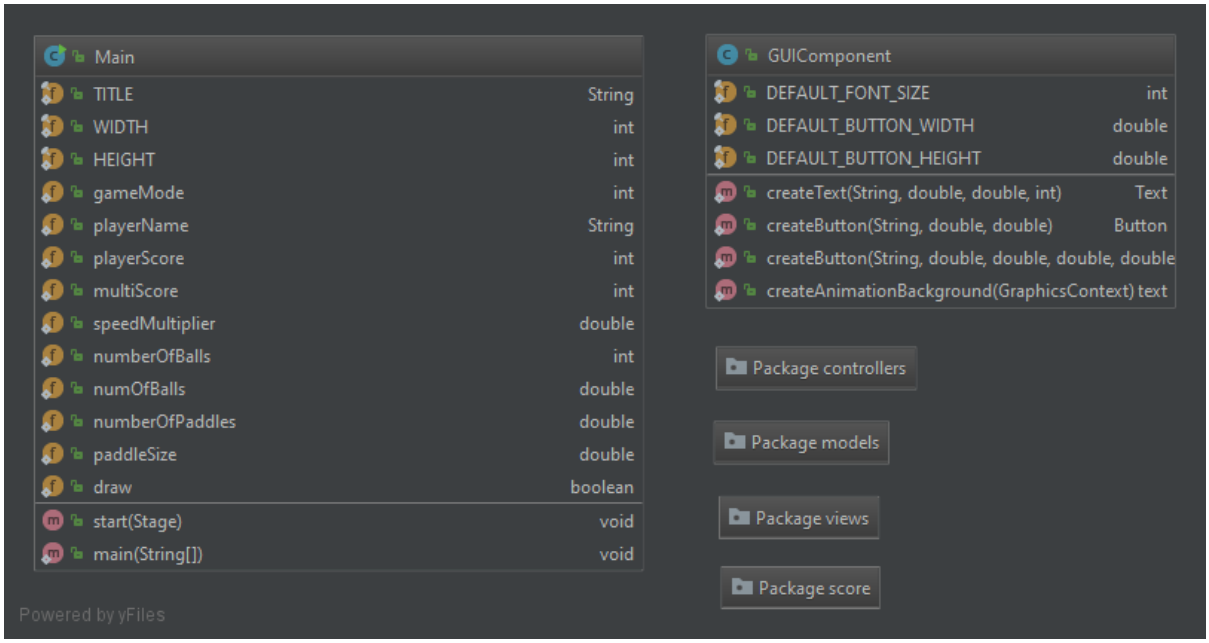
The length of the current single player campaign is relatively short at only 3 levels. To extend the shelf time of the game it would be desirable to add additional levels to keep the user entertained for longer. Once additional levels are added it would also be feasible to use a skill tree to potentially allow the user to choose skills which best correspond to their style of play.

Appendices

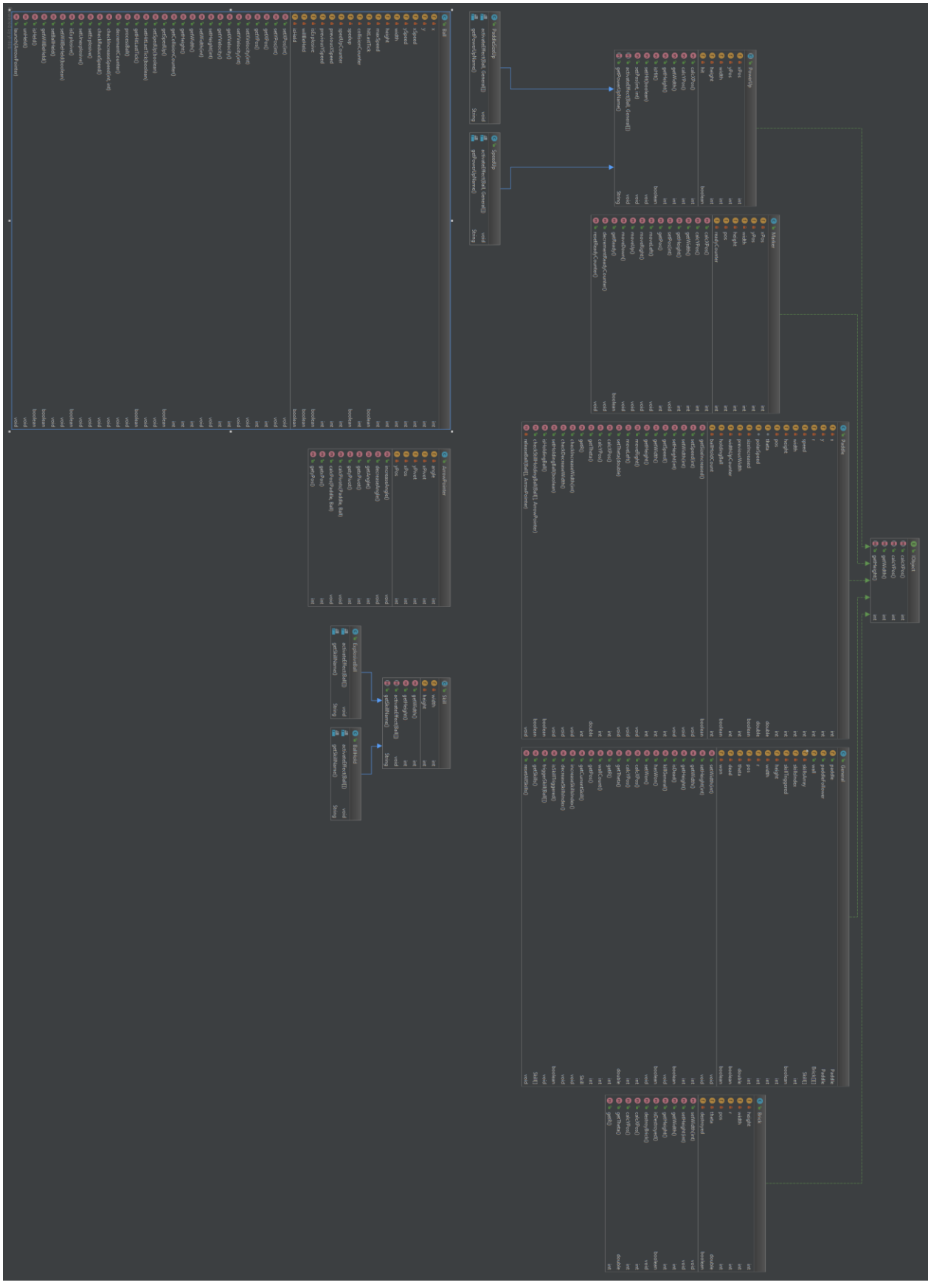
Overall Diagram



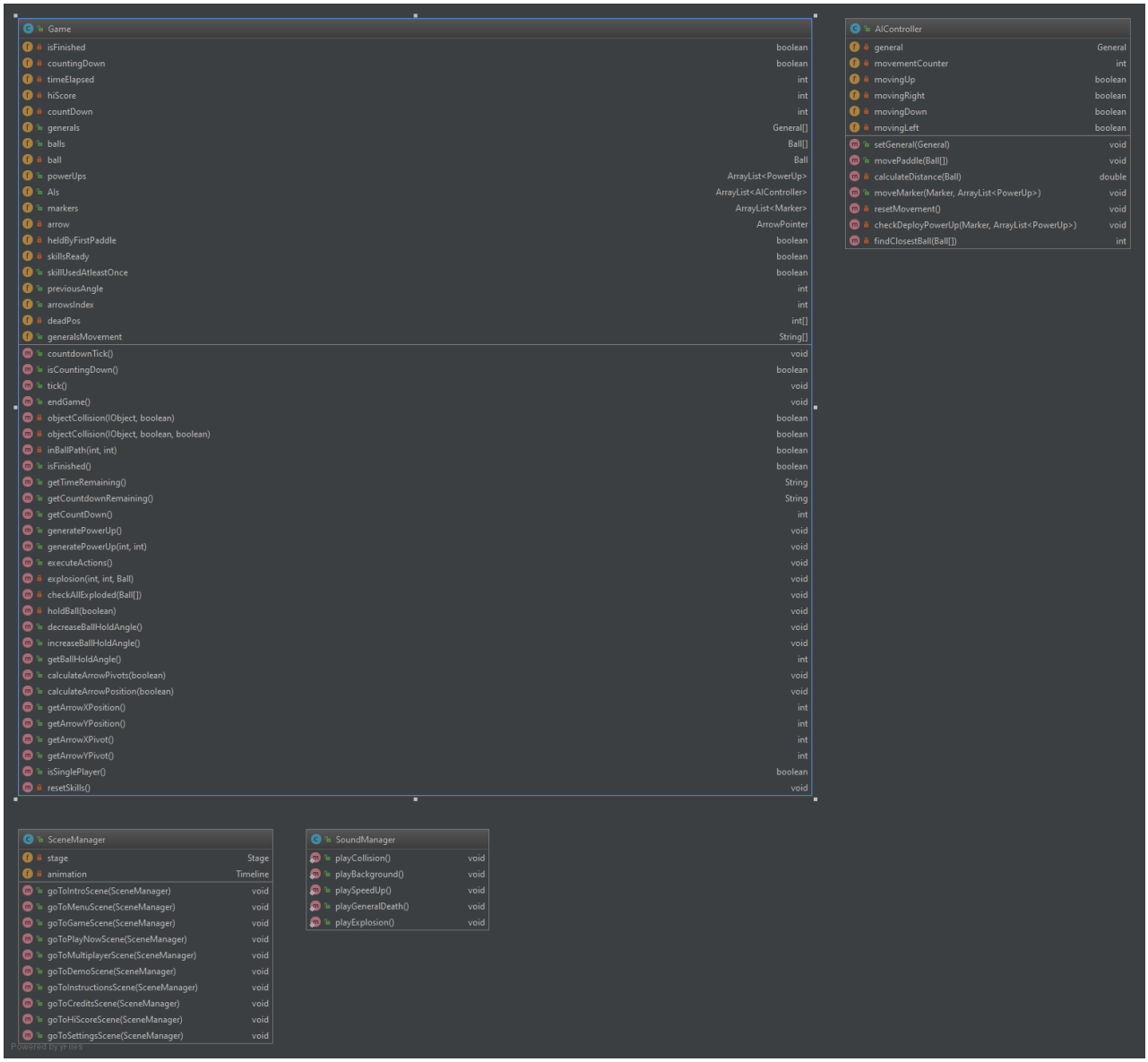
Etruaruta



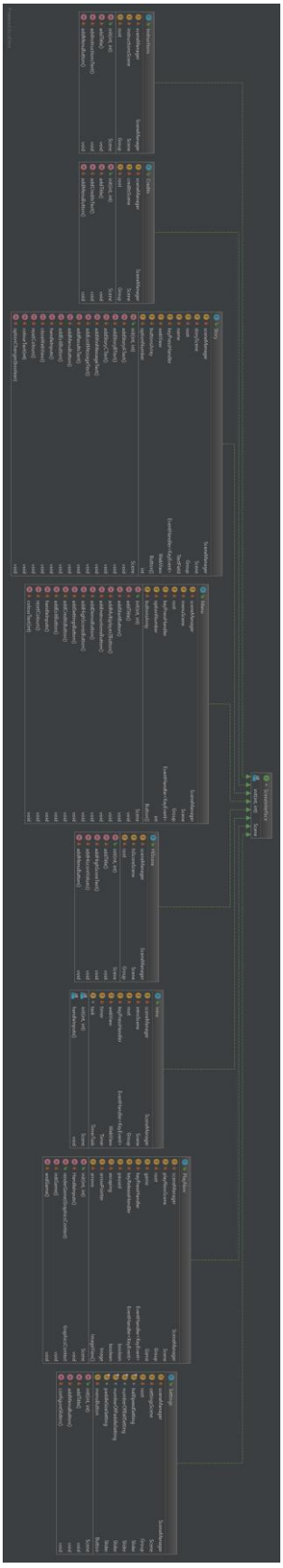
Models



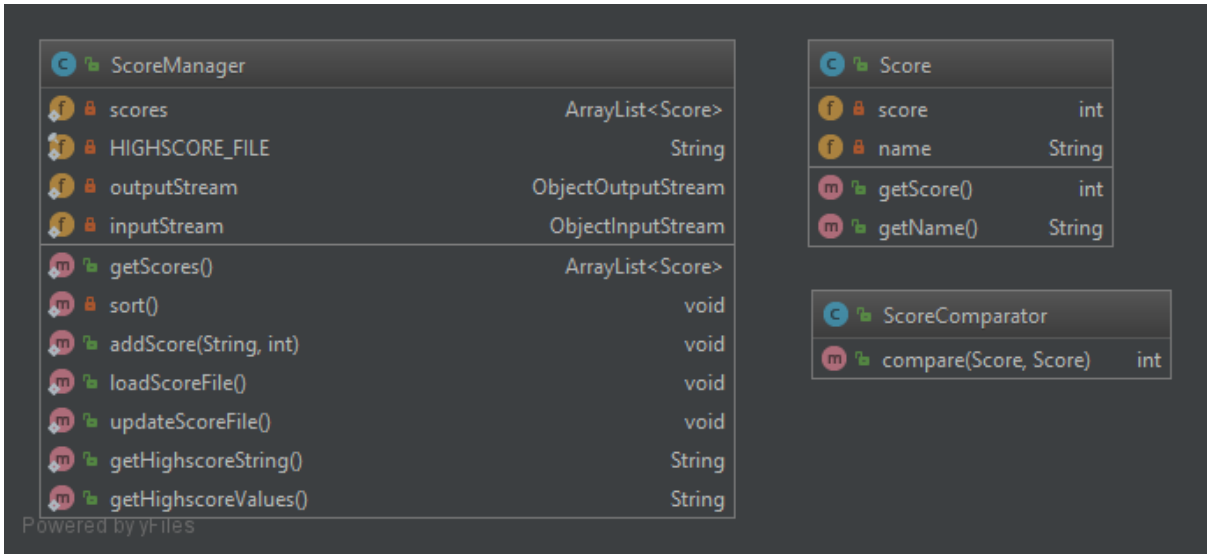
Controllers



Views



Scores


ScoreManager
scores ArrayList<Score>
HIGHSCORE_FILE String
outputStream ObjectOutputStream
inputStream ObjectInputStream
getScores() ArrayList<Score>
sort() void
addScore(String, int) void
loadScoreFile() void
updateScoreFile() void
getHighscoreString() String
getHighscoreValues() String
Score
score int
name String
getScore() int
getName() String
ScoreComparator
compare(Score, Score) int
Powered by yFiles

v