'Balls of Steel'

Group 4

John Laurin

Joakim Åkerlund

Milan Ivanovic

Daniel Öberg

Christoffer Lundell Johansson

Table of Contents

1. Introduction

This document is intended to outline the design considerations and the detailed architecture for the 'Balls of Steel' 1.0 software project. It is intended to outline the system-level design decisions that were developed from the requirements analysis and allow for a successful development of the project implementation.

This document will familiarise you with the overall architecture of the system platform and the system itself, list the dependencies, assumptions and constraints of that platform, outline the graphical User Interface and provide design details and test cases for the various system components.

This document is intended for the purview of professional and semi-professional software developers who have experience developing software with the Java programming language.

This document is a continuation of the development process begun by the requirements analysis and the subsequent Requirements Document. The Requirements Document 1.0 is considered a prerequisite document for this document.

This document will result in the creation of an Implementation Plan, a Testing Plan and ultimately the application itself.

This document will in its entirety show the following:

The goal of the project is to develop a mobile game application according to the description herein. In essence, the goal of the game is to navigate a bouncing ball through a series of obstacles. The game is to be developed with Java Micro Edition and is intended for mobile phones.

This presents a number of obstacles in the development of the application, as the computational resources, programming functionality and internal memory of a mobile phone is limited.

Important terms and acronyms

Java ME – Java Micro Edition.

CPU – Central Processing Unit.

RAM – Random Access Memory.

Sprite – A small graphic that can be moved independently around the screen, producing animated effects.

Framework – In software development, a framework is a defined support structure in which another software project can be organized and developed.

GUI – Graphical User Interface.

UML – Unified Modelling Language (UML) is an Object Management Group (OMG) standard for modelling software artifacts.

2. System Overview

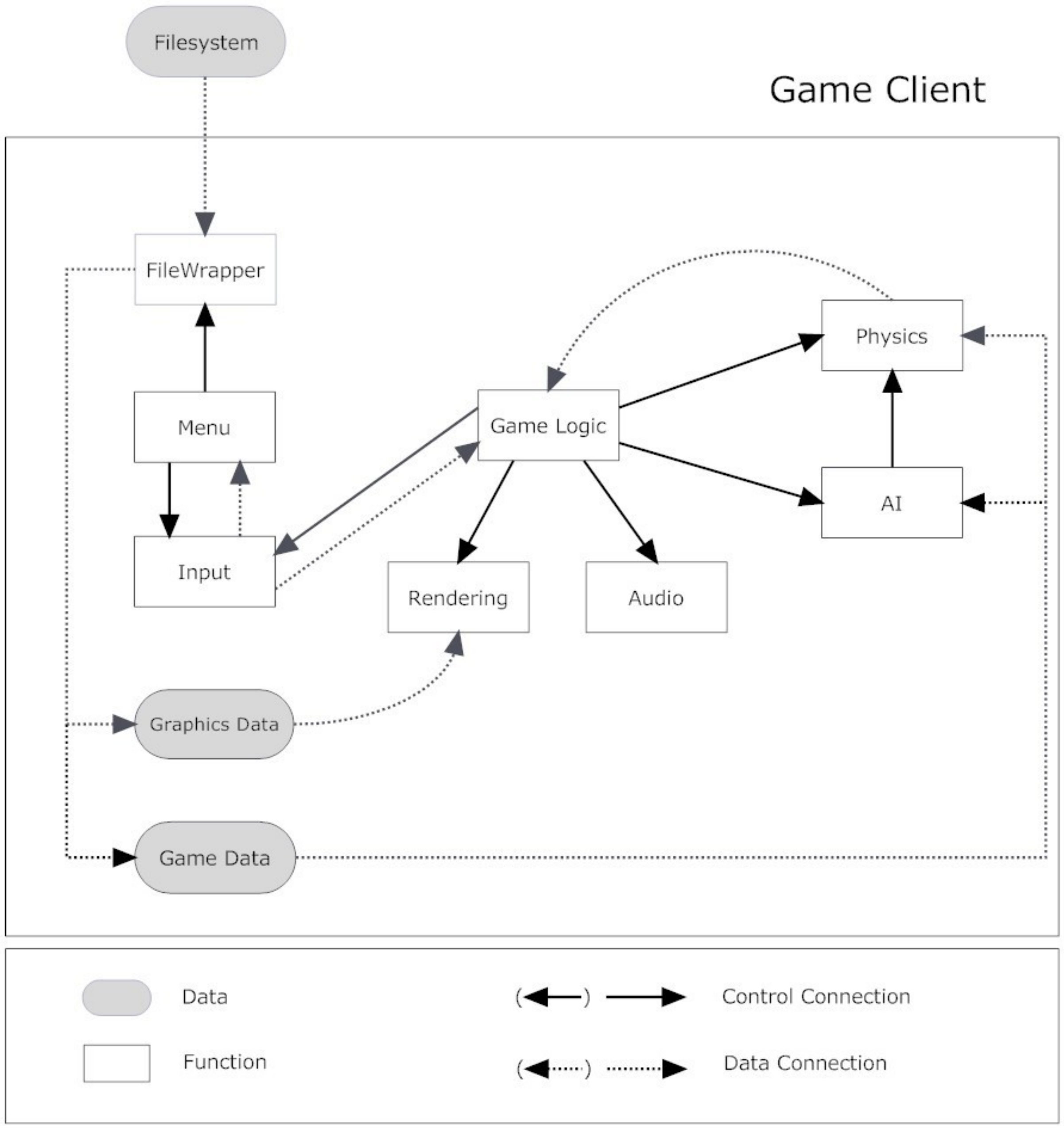
2.1 General Description

What we aim for is an easy to get to game that offers a challenge and entertaining way to pass time. There are issues related to implement applications and games on the mobile platform, like memory restrictions, hardware power and restrictions. The mobile system gives applications very few rights when working against the files and the file system of the platform, making it hard to save data or settings. Due to the limited cpu and RAM capabilities inherited in the mobiles varied brands and design, we must construct the game in a way that makes it adaptable to the greater masses of phones.

The Java ME has many functions, classes and methods for working the mobile environment to its highest potential. We can save and load highscore, data or information with the ME database interfaces. This bypasses the limitations of the access to the file system. We can dynamically change resolution and settings to try to optimize the game for the device that runs it.

In the game itself, your avatar is a ball. A bouncing ball. The goal is to, under the pressure of time, navigate your bouncing avatar through the levels collecting keys to advance to the next level. In doing so chasing the best score possible. Rejoicing in past accomplishments and anticipating challenges on higher levels.

2.2 Overall Architecture Description



AI - Takes care of the enemies AI and their game logic. More specifically in what way the enemies will move and how they will react upon coming into close range of the ball.

Audio - This module is called upon when another module wants to play a sound clip.

File Wrapper - Handles loading and saving of data into the phones memory.

Game Logic - Handles everything that the other modules don't. That includes the balls motion and calling the other modules for updates.

Input - Pulls the state of the input (joystick/keypads) and presents these.

Menu - Presents the menu on screen and gives the user the capability to select whether he wants to continue, change settings, see the high score, create a new game or quit.

Physics - Controls the player's gravity and all collision detection.

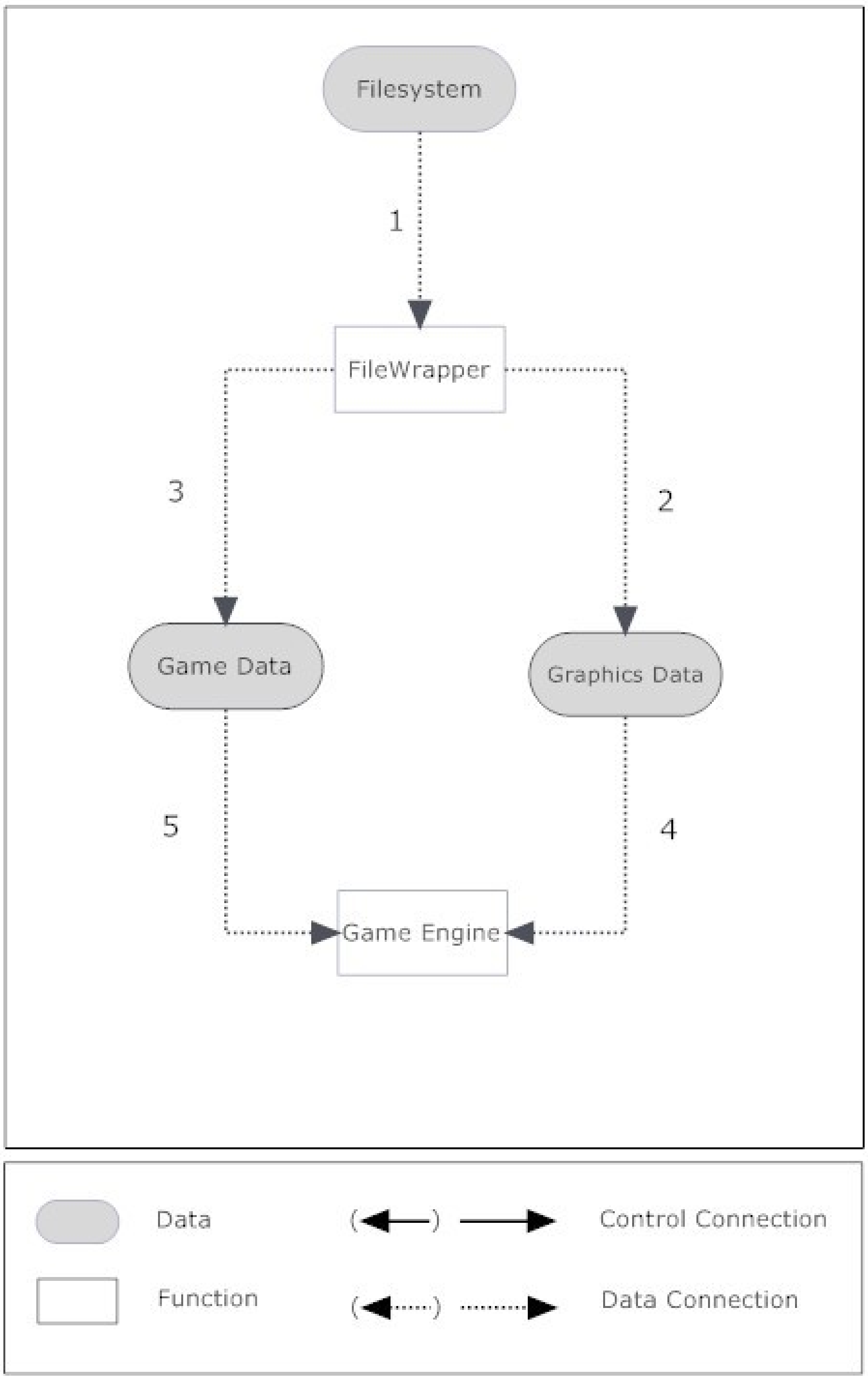
Rendering - Renders the graphics for the ball, the enemies, the items and the map and presents it on the screen.

Game Data - Handles the gamedata received from the filewrapper, e.g. information about enemies position etc.

Graphics Data - Handles the graphics data received from the filewrapper.

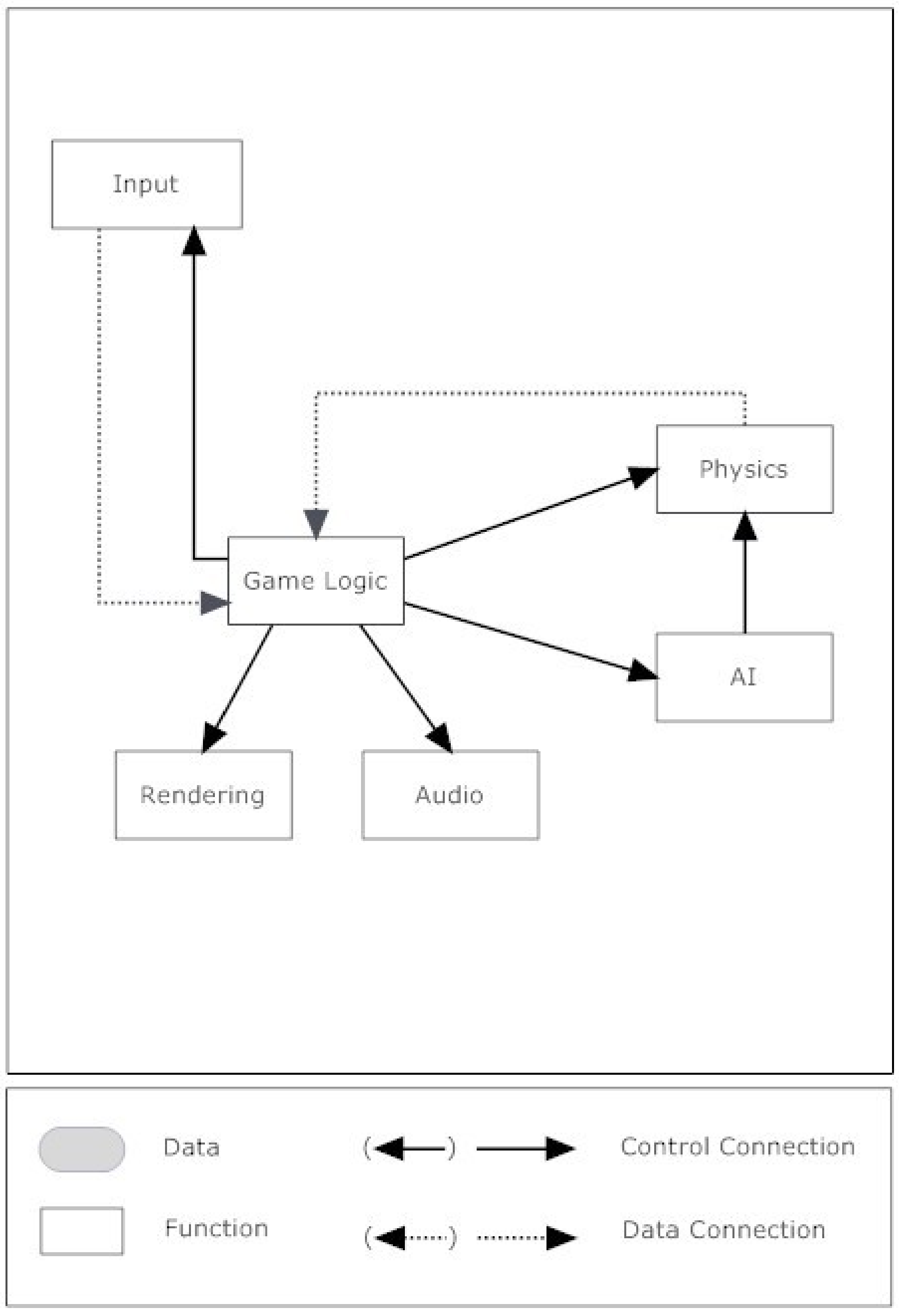
2.3 Detailed Architecture

Dataflow while loading game



1. The FileWrapper gets data from the Filesystem
2. The FileWrapper sends the data regarding the graphics (sprites, position, colors, etc) to Graphics Data.
3. The FileWrapper sends the data regarding the game state (score, level, etc) to Game Data.
4. Graphics Data sends the data to the necessary module in the Game Engine.
5. Game Data sends the data to the necessary module in the Game Engine.

Control flow and data flow while playing the game



1. The Game Logic fetches the input-commands from the input module.

2. The Game Logic calls the functions for the AI module for calculating enemy movement etc.

3. The AI communicates with the Physics module using that data to calculate enemy behavior.

4. The Game Logic calls the functions for the Physics module for calculating collision detection and gravity and returns the resulting data to the Game Logic.

5. The Game Logic calls the Renderer to draw up all the updates in graphic.

6. The Game Logic calls the Audio module for playing up sounds.

3. Design Considerations

3.1 Assumptions and Dependencies

Framework

The game will run under the Java ME mobile platform. Therefore we assume that any mobile that will play our game also have support for Java ME, since without this it won't work.

Operating systems

Since the game is run under a virtual machine, it should work on any operating system supporting Java ME mobile platform of at least version:

Connected Limited Device Configuration

1.1

JSR 139

Mobile Information Device Profile

2.0

JSR 118

3.2 General Constraints

Hardware limitations

Since our intended platform is a mobile phone the hardware is very limited in comparison to a stationary computer. The overall memory capacity in a mobile phone is often very low imposing a severe limitation on a mobile game. It's also often very limited to what applications are allowed to do with the file system concerning reading and writing data.

Java ME limitations

Java ME is like normal Java but with several limits and changes to consume less resources. Due to the Java ME limitations the game will not run on older mobile phones with an obsolete version of Java ME. However the game should run on future versions as long as they don't do any drastic changes to the current framework.

4. Graphical User Interface

Upon booting the game the option menu loads and is presented on the screen. The user can clearly see the different options, which he by normal key/touch input, can move around to different options and thus highlighting them. The user can then choose an option of the following as described by the functional requirements.

Menu Screen GUI

Balls of Steel

Continue Game - Boots up the in-game GUI and continues a previously saved progress from the start of that level. In the case that the game is unable to load the save file an error message will occur and the user will have to restart the game.

New Game - Boots up the in-game GUI and starts a new game.

High Score - Loads up the High Score GUI and presents the play with the top ten scores and their corresponding name tag.

Instructions - Loads up the instructions menu where a short "how to play" will be described and pressing the return key will return to the normal menu screen.

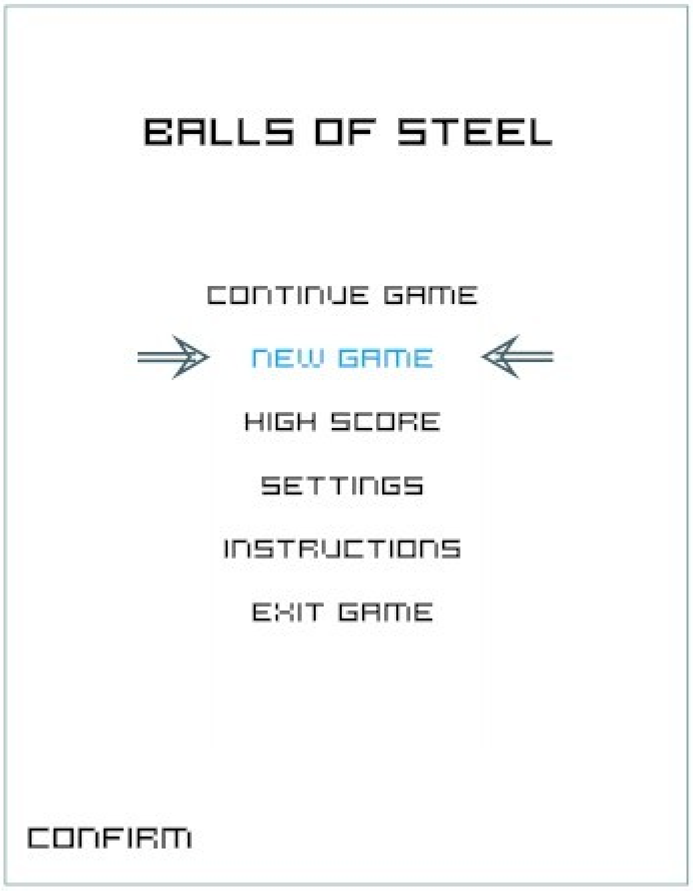
Settings - Loads up the settings screen where you can turn the sound on and off and also return to the normal menu screen.

Exit Game - Exits the game.

Confirm - Performs the selected action.

Functional Requirements describing this: 1

Concept picture of the menu screen:



Instructions menu GUI

How to control the ball:

The ball continuously bounces up and down and you control the movement of the ball by adding spin to the left or right, which will take input from the keypad/touchpad and move the ball to the left or right at a speed defined by how much spin it has in that direction.

Move the ball through the level and collect all the keys until you open up the exit and enter it get to the next level. Do this as fast as possible and try to avoid enemies and obstacles.

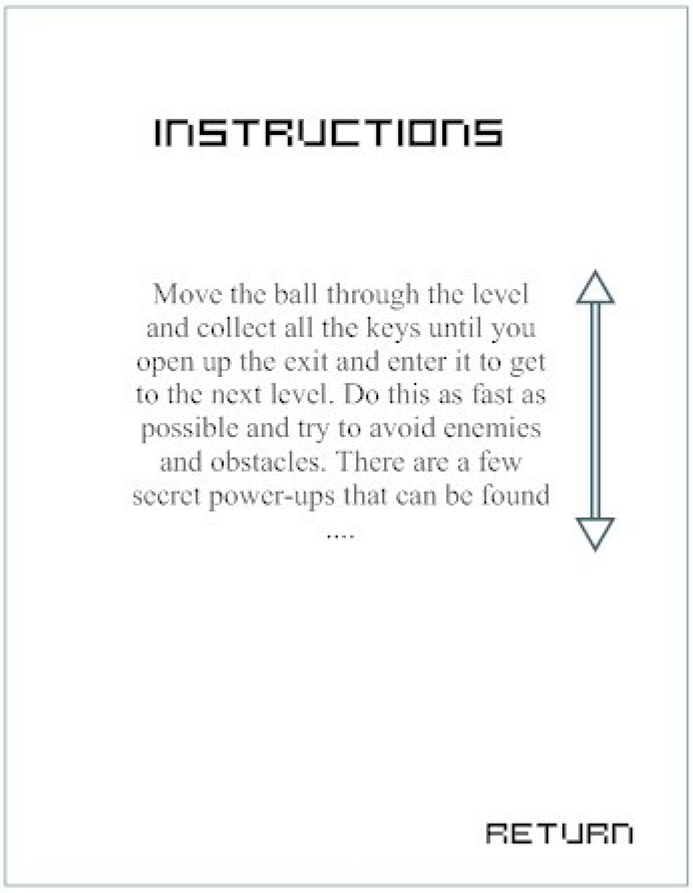
There are a few secret power-ups that can be found throughout the levels which may help you in your quest.

To move you spin the ball to the left or right by using the key/touch pad.

Press the return key to return to the menu screen

Functional Requirements describing this: 3

Concept picture of the instructions menu:



INSTRUCTIONS

Move the ball through the level and collect all the keys until you open up the exit and enter it to get to the next level. Do this as fast as possible and try to avoid enemies and obstacles. There are a few secret power-ups that can be found ....

RETURN

Settings menu GUI

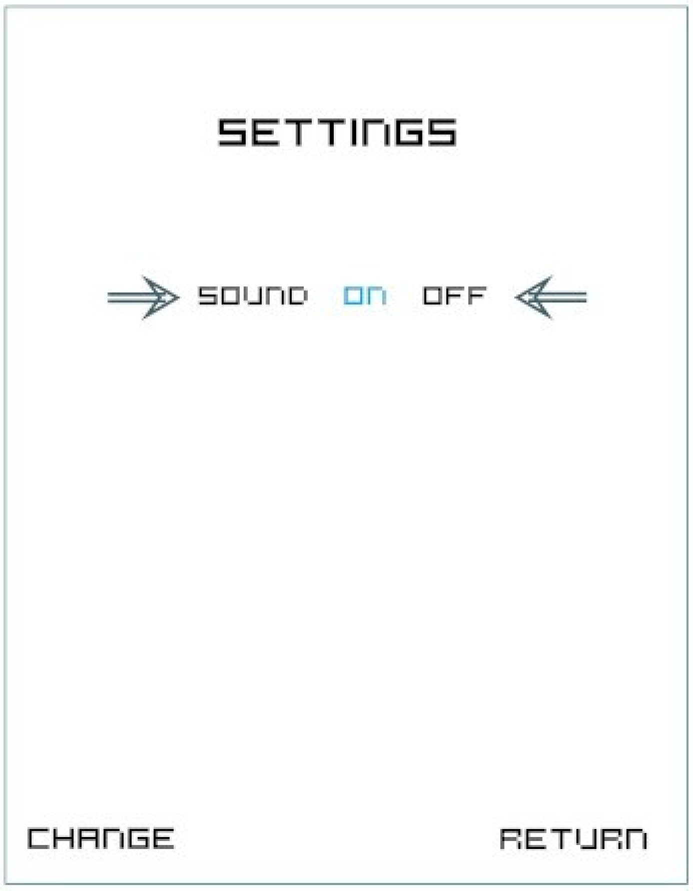
Sounds on/off - Toggling this will switch between sound on and sound off.

Change - Performs the selected action.

Return - Will return the user to the normal menu screen.

Functional Requirements describing this: 2

Concept picture of the settings menu:



High Score GUI

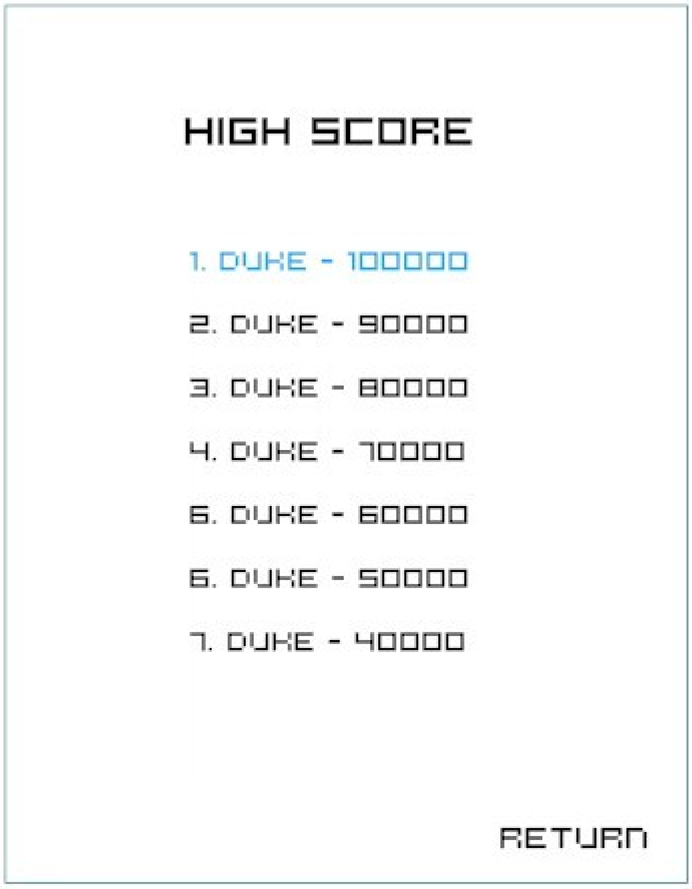
Default High Scores are the following.

1. Duke – 100000
2. Duke – 90000
3. Duke – 80000
4. Duke – 70000
5. Duke – 60000
6. Duke – 50000
7. Duke – 40000

Press the return key to return to the menu screen

Functional Requirements describing this: 4, 23 ,24

Concept picture of the high score list:



HIGH SCORE

1. DUKE - 100000

2. DUKE - 90000

3. DUKE - 80000

4. DUKE - 70000

6. DUKE - 60000

6. DUKE - 50000

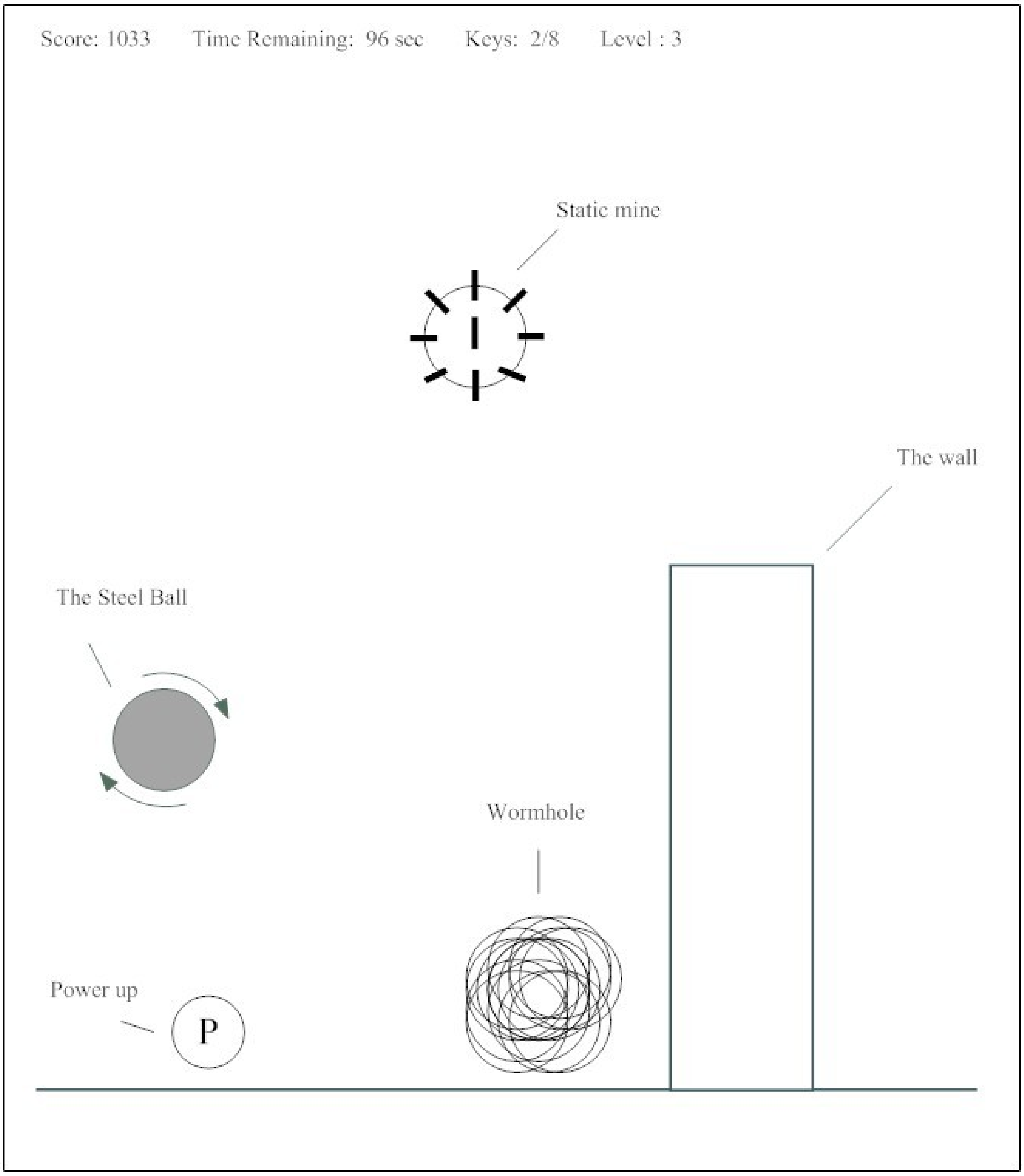
7. DUKE - 40000

In-game GUI

During the game you control the movement as specified and you can return to the menu screen by pressing a pre-defined key. If at any time the phone would receive an sms/phonecall etc the game will pause and the phone will use its standard procedures for handling it.

Functional Requirements describing this: 25, 26, 27 describes sms/phonecall handling etc, and 6-22 define the actual In-game content.

Concept picture of the In-game GUI:



5. Design Details

5.1 Class Responsibility Collaborator (CRC) Cards

FileWrapper	
Responsibilities	Collaborators
Main task is to loads data from the file-system and then distribute it to the collaborators. It also handles saving of high-score and the currently active level and the remaining time.	Menu

Menu	
Responsibilities	Collaborators
Provides different options, including new game, continue game, view high-score, settings, exit, which the player can choose between.	FileWrapper Input

Input	
Responsibilities	Collaborators
Retrieves direct input from the mobile phone's key/touchpad.	Menu Game Logic

Game Logic	
Responsibilities	Collaborators
The game logic will be the central core of the game and will organize and call upon the other classes.	Input Rendering Audio Physics AI

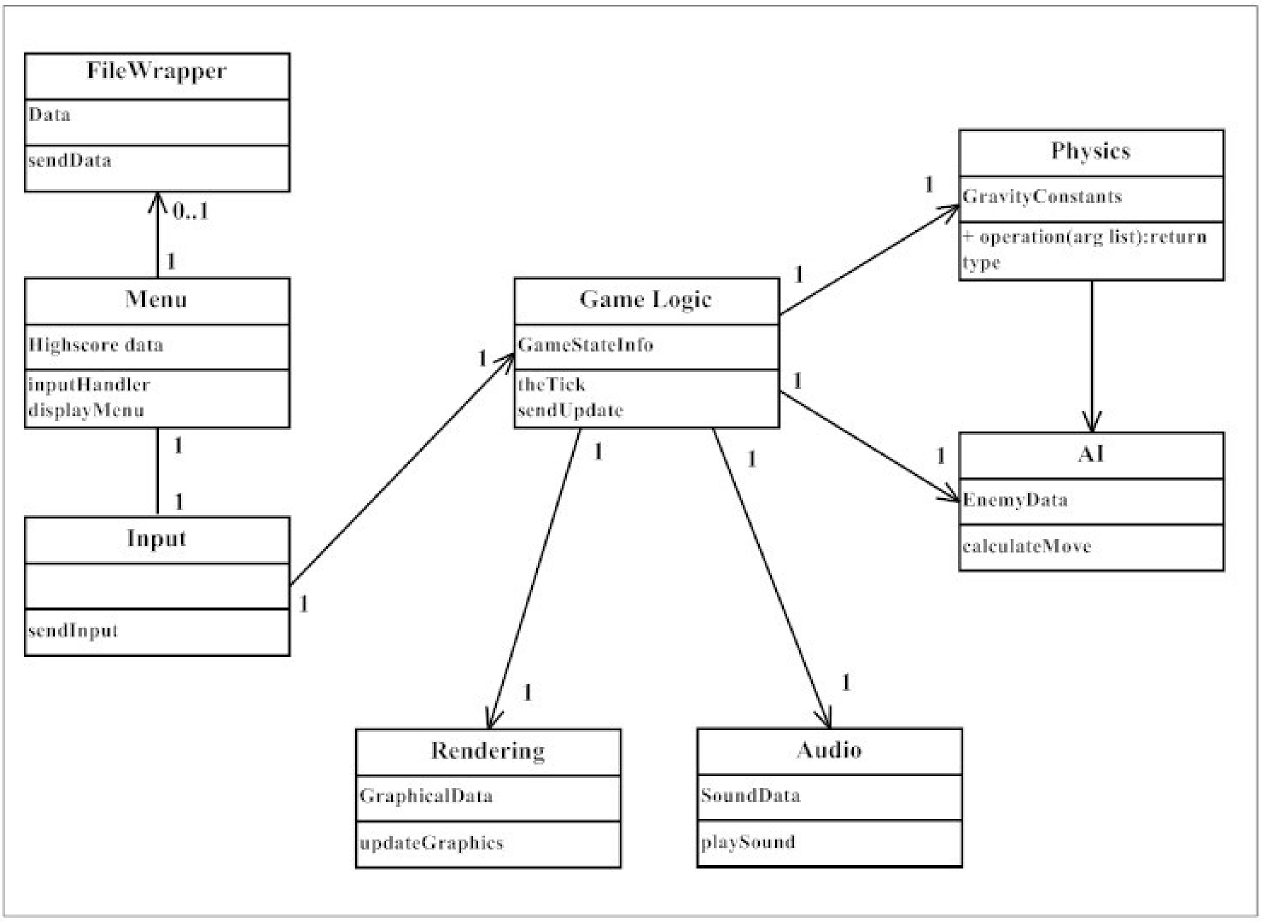
Rendering	
Responsibilities	Collaborators
Presents the graphic upon the screen.	Game Logic

Audio	
Responsibilities	Collaborators
Handles the sound and music output.	Game Logic

AI	
Responsibilities	Collaborators
Handles the movement and action of the obstacles and enemies.	Game Logic Physics

Physics	
Responsibilities	Collaborators
Handles the calculations for the gravity and collision detection.	Game Logic AI

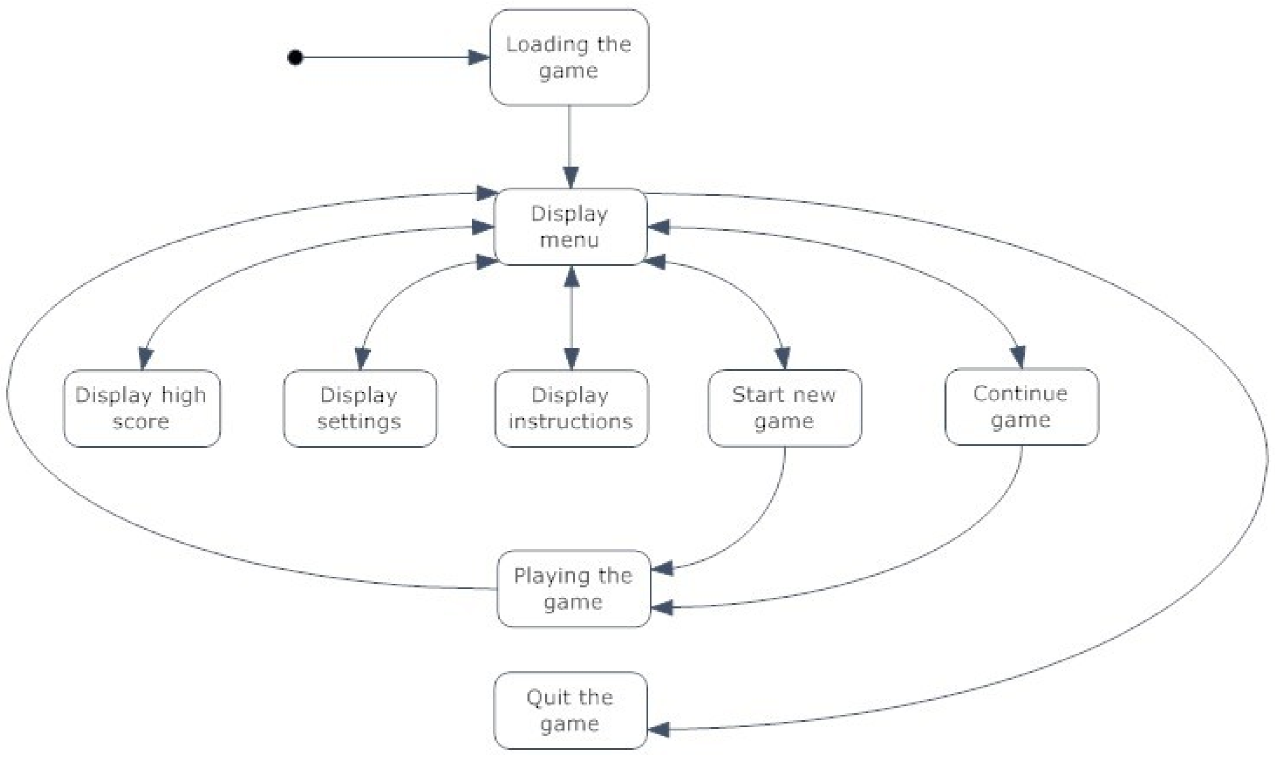
5.2 Class Diagram



0..1	No instances, or one instance (optional, may)
1	Exactly one instance
0..* or *****	Zero or more instances
1..*	One or more instances (at least one)

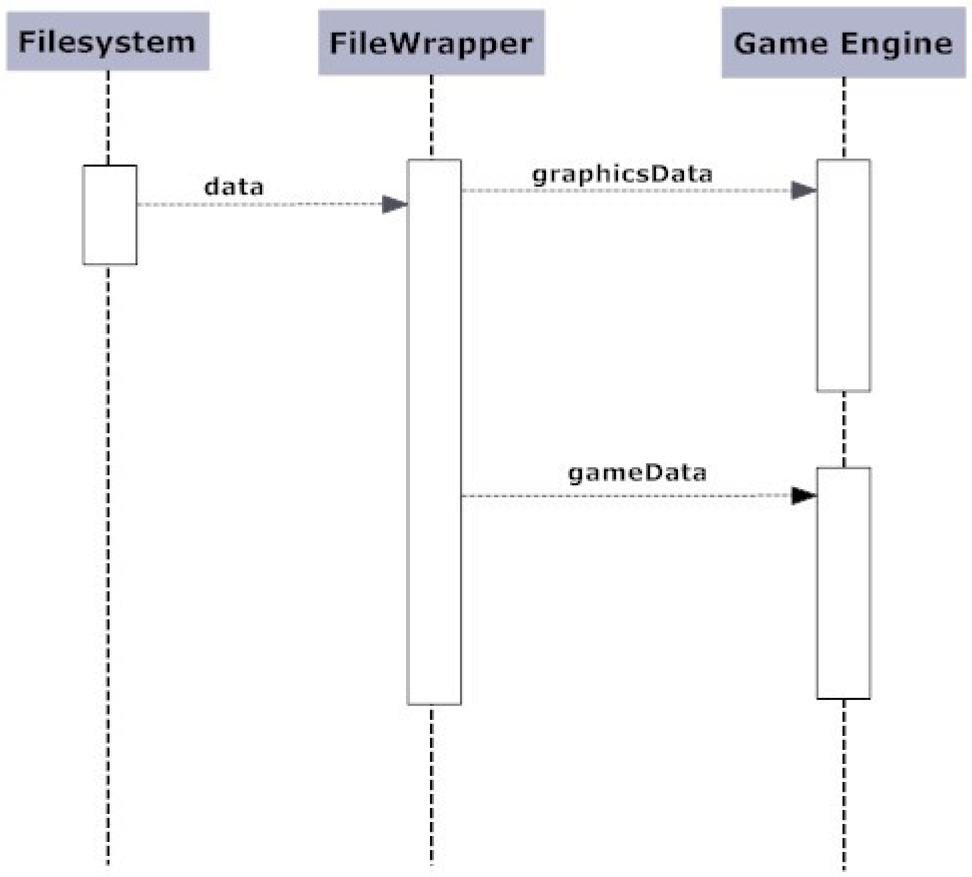
5.3 State Charts

State-chart for one session

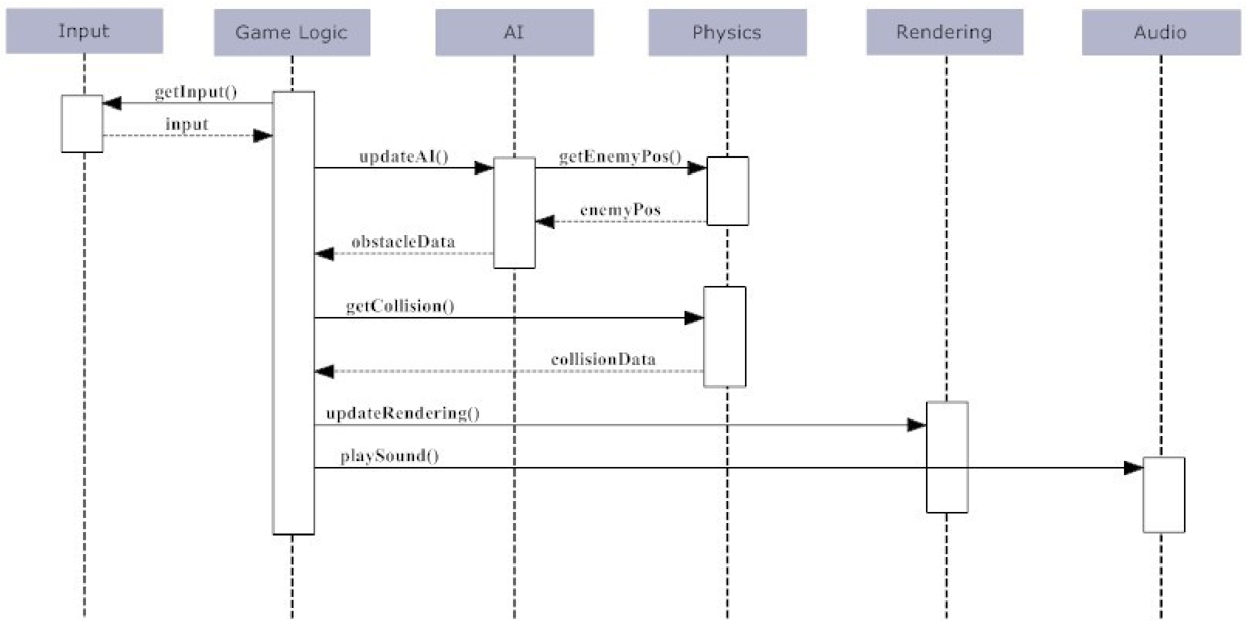


5.4 Interaction Diagrams

Loading data sequence diagram



Gameplay sequence diagram



5.5 Detailed Design

This document describes the detailed design. It will list all public, private and protected methods. The marks are from UML and symbolise the following:

Mark	Visibility type
+	Public
#	Protected
-	Private
~	Package

Menu
+ Constructor() **Parameters:** none **Return Value:** void **Description:** Initializes the menus User Interface (UI). A list is shown with the following elements: New Game, High Score, Settings, Instructions and Exit Game. Checks whether there is an earlier gaming session available and if it is it also presents the user with an extra option Continue Game. **Pre-condition:** None **Post-conditions:** None **Called by:** Main **Calls:** isContinueAvailable()   + ContinueGame() **Parameters:** none **Return Value:** void **Description:**

Restores the screen to gameplay and passes the controlflow to Main which in turn creates GameLogic and starts the game.

Pre-condition:

isContinueAvailable returns true.

Post-conditions:

If GraphicsData and GameData doesn't exist they get constructed.

Called by:

Called by delegates from keypad made by the java api.

Calls:

loadData

+ NewGame()

Parameters: none

Return Value: void

Description:

Restores the screen to gameplay and passes the controlflow to Root which in turn creates GameLogic and starts the game.

Pre-condition:

None

Post-conditions:

Graphics and GameData (leveldata and so on) will be loaded.

Called by:

Called by delegates from keypad made by the java api.

Calls:

loadData

+ Highscore()

Parameters: none

Return Value: void

Description:

Creates a new screen and lists the highscore on it.

Pre-condition:

None

Post-conditions:

None

Called by:

Called by delegates from keypad made by the java api.

Calls:

None.

+ Settings()

Parameters: none

Return Value: void

Description:

Creates a screen that present the user with the option of turning sound on or off.

Pre-condition:

None

Post-conditions:

That GameData will contain these settings.

Called by:

Called by delegates from keypad made by the java api.

Calls:

None

+ Instructions()

Parameters: none

Return Value: void

Description:

Creates a screen that present the user with information about the game on how to play it.

Pre-condition:

None

Post-conditions:

None

Called by:

Called by delegates from keypad made by the java api.

Calls:

None

+ Exit()

Parameters: none

Return Value: void

Description:

Optionally saves the data and exits.

Pre-condition:

None

Post-conditions:

The program shuts down

Called by:

Called by delegates from keypad made by the java api.

Calls:

None

- loadData(continue: boolean)

Parameters:

continue: boolean

Tells whether this function should try to load data from an earlier gaming session, if the value is true then it will.

Return Value: void

Description:

Loads data from the phones memory card and creates or updates the GraphicsData and GameData structures.

Pre-condition:

Images should exist on the same path as the game excutable. If not the program throws an IOException which will lead to that the game shuts down.

Enough memory should exist to load the images into it otherwise throw out of memory exception.

Post-conditions:

The global structures GraphicsData and GameData will be created.

Called by:

NewGame()

CreateGame()

Calls:

FileWrapper.getData()

- isContinueAvailable(): boolean

Parameters: none

Return Value:

Returns true if there exists data to continue an earlier gaming session.

Description:

Checks if there exists data to continue an earlier gaming session.

Pre-condition:

None

Post-conditions:

None

Called by:

Constructor()

Calls:

FileWrapper.saveExists()

GameLogic

+ run()

Parameters: none

Return Value: void

Description:

A loop that calls AI, Physics and Renderings update function. Also updates the time and checks the time left for the player.

Pre-condition:

None

Post-conditions:

None

Called by:

Main()

Calls:

AI.update()

Physics.update()

Rendering.update()

AI
+ update() **Parameters:** none **Return Value:** void **Description:** An loop that calls AI, Physics and Renderings update function. **Pre-condition:** None **Post-conditions:** None **Called by:** GameLogic.run() **Calls:** moveEnemies() - moveEnemies() **Parameters:** none **Return Value:** void **Description:** Moves the enemies according to a pattern. Changes the enemies coordinates. **Pre-condition:** None **Post-conditions:** GameData is updated with the new coordinates for the enemies. **Called by:** update() **Calls:** None - attackPlayer() **Parameters:** none **Return Value:** void

Description:

Removes some of the players time left.

Pre-condition:

None

Post-conditions:

GameData.timeleft() is updated.

Called by:

update()

Calls:

Audio.playSound()

Physics

+ update()

Parameters: none

Return Value: void

Description:

Checks if the player bumped into an object or an enemy. If the player bumped into an object it changes the angle the ball is traveling. If it bumps into an enemy it calls AI to tell it to do something. It also moves the players ball.

Pre-condition:

None

Post-conditions:

None

Called by:

GameLogic.run()

Calls:

isPlayerCloseToEnemy()

isPlayerCloseToObject()

movePlayer()

- isPlayerCloseToObject(): boolean

Parameters: none

Return Value:

returns true if an object and the players ball have overlapped.

Description:

Checks if an object and the players ball have overlapped. If so it checks for what kind of object it is. If it is a wall it changes the angle the ball is traveling. If it is a powerup it changes the GameData depending on what kind of powerup it is.

Pre-condition:

None

Post-conditions:

None

Called by:

update()

Calls:

none

- isPlayerCloseToEnemy(): boolean

Parameters: none

Return Value:

returns true if an enemy and the players ball have overlapped.

Description:

Checks if an enemy and the players ball have overlapped. If so it calls AI.attackPlayer.

Pre-condition:

None

Post-conditions:

None

Called by:

update()

Calls:

AI.attackPlayer()

- movePlayer()

Parameters: none

Return Value: none

Description:

Moves the players ball according to the angle and speed specified in GameData.

Pre-condition:

None

Post-conditions:

GameData is updated with the new coordinate for the player.

Called by:

update()

Calls:

None

Filewrapper

+ getData()

Parameters: Defines what data type is to be loaded (level, save, graphic etc)

Return Value: Data

Description:

Fetches specific data.

Pre-condition:

That the data files are not corrupted.

Post-conditions:

None

Called by:

Called by GameLogic when loading the game.

Calls:

none

+ saveGameData()

Parameters: none

Return Value: void

Description:

Saves the game data.

Pre-condition:

The user shuts down a game in progress.

Post-conditions:

None

Called by:

Called by GameLogic when saving the game.

Calls:

None

+saveExists()

Parameters: none

Return Value: boolean

Description:

Checks if there is any saved data.

Pre-condition:

None

Post-conditions:

None

Called by:

Called by Menu when starting up the game.

Calls:

None

+ sendGraphicsData()

Parameters: none

Return Value: Graphical Data

Description:

Sends the graphical data that was loaded from the filesystem to the rendering class.

Pre-condition:

That the data in GraphicsData is intact and available. If this fails exit the game.

Post-conditions:

None

Called by:

Called by GameLogic when loading the game.

Calls:

Filewrapper.getData()

+ sendGameData()

Parameters: none

Return Value: void

Description:

Sends the game data that was loaded from the filesystem to the rendering class.

Pre-condition:

That the data in GameData is intact and available. If this fails exit the game.

Post-conditions:

none

Called by:

Called by GameLogic when loading the game.

Calls:

Filewrapper.getData()

Input

+ pollInput()

Parameters: none

Return Value: Data regarding for which keycommands are issued

Description:

The method listens for any keypresses to occur

Pre-condition:

The game must be loaded.

Post-conditions:

None

Called by:

Called by GameLogic when loading the game.

Calls:

none

+ sendInput()

Parameters: none

Return Value: void

Description:

Sends data regarding the keypresses to the GameLogic.

Pre-condition:

The game must be loaded and pollInput() must be running.

Post-conditions:

None

Called by:

Called by GameLogic when pollInput() has received data to send.

Calls:

none

Rendering

+ updateGraphics()

Parameters: Data regarding graphics (updated position etc)

Return Value: void

Description:

Updates the graphical output on the mobile phone.

Pre-condition:

The game must running.

Post-conditions:

None

Called by:

Called by GameLogic while running the game.

Calls:

none

Audio

+ playSound()

Parameters: Data regarding the audio (what sounds to play etc)

Return Value: void

Description:

Plays the sound.

Pre-condition:

The game must running.

Post-conditions:

None

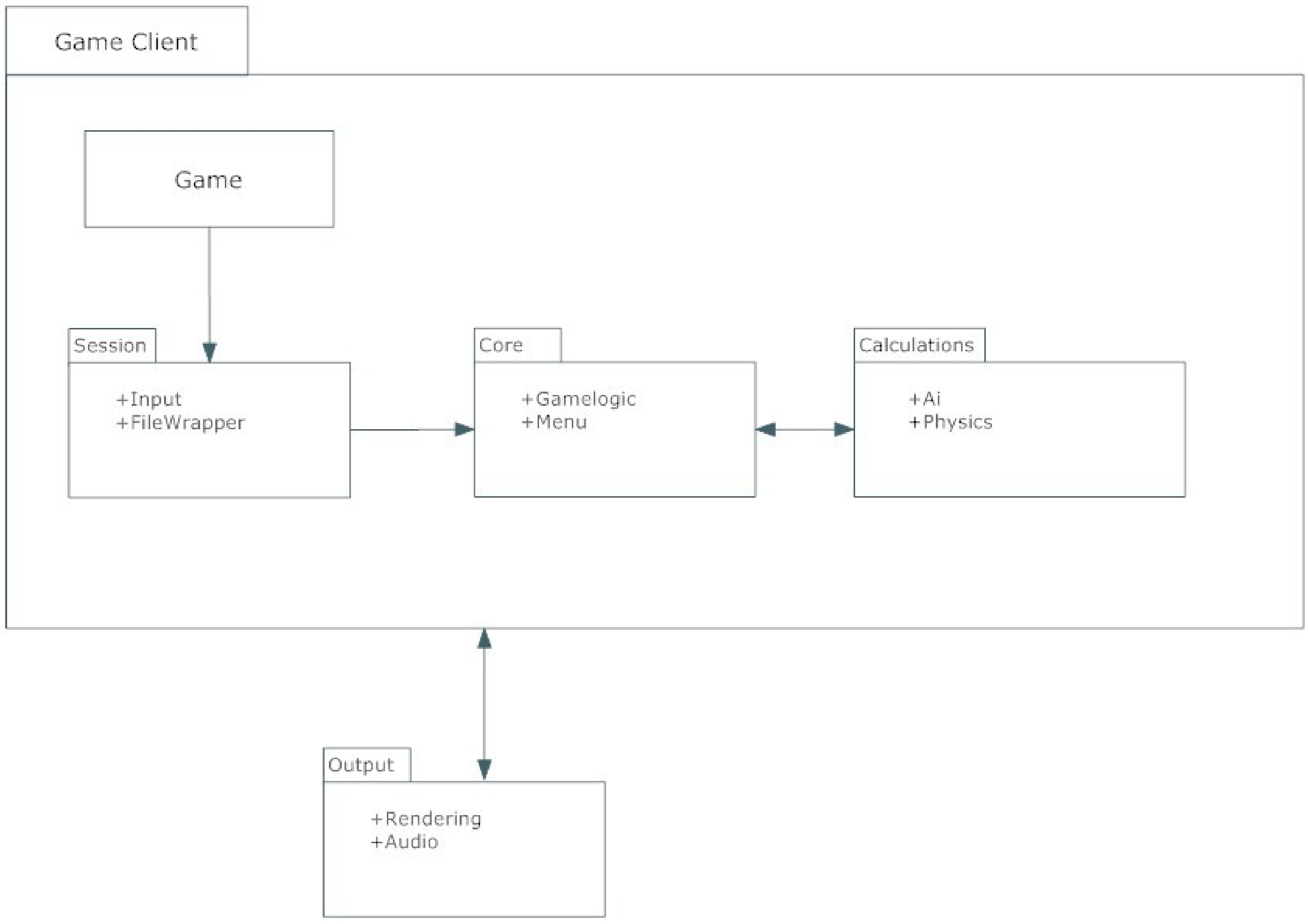
Called by:

Called by GameLogic while running the game.

Calls:

none

5.6 Package Diagram



6. Functional Test Cases

Menu options

The system shall provide a menu-screen on start-up, allowing the user to select to start a new game, to continue one that is in progress, to view the high-score, to make general settings, to view instructions and to exit the program.

Reference: Functional Requirement 1

Input: None.

Output/Observed Effects: Menu is correctly displayed on the screen.

Test:

1. Load up the game.
2. Observe that the relevant menu items are presented on the screen.

Sound option

The system shall allow users to turn on/off sound via the settings-option on the menu-screen.

Reference: Functional Requirement 2

Input: Key/Touchpad press.

Output/Observed Effects: The sound turned on/off.

Test:

1. Enter settings.
2. Toggle sound on or off.
3. Start a new (or continue an old) game.
4. Confirm the loss or existence of sound.

Instructions option

The inexperienced user shall be able to gain instructions on gameplay mechanics and goals via the instructions menu.

Reference: Functional Requirement 3

Input: Key/Touchpad press.

Output/Observed Effects: Instruction text is displayed.

Test:

1. Enter instructions option.
2. Observe the instructions text.

Highscore option

The user shall be able to see the 10 highest times, as well as the signature of the user who achieved it via the high-score option on the menu-screen.

Reference: Functional Requirement 4

Input: Key/Touchpad press.

Output/Observed Effects: Highscore is displayed.

Test:

1. Start a new game.
2. Play the game to the finish (and merit a highscore).
3. Enter your initials for the highscore.
4. Enter highscore option.
5. Observe that your previously written initials and highscore is displayed.

Return to main menu

The user shall be able to return to the main menu at any time during play.

Reference: Functional Requirement 5

Input: Key/Touchpad press.

Output/Observed Effects: The menu screen is displayed.

Test:

1. Start up a game or enter any menu option.
2. Press the return key.
3. Observe that you are returned to the main menu.

Ball motion

The user shall be able to control the motion of the ball.

Reference: Functional Requirement 6

Input: Key/Touchpad press.

Output/Observed Effects: The ball is moved.

Test:

1. Start up a game.
2. Initiate movement by pressing left or right key.
3. Observe ball movement in the specified direction.

Gravitational force

There shall be a gravitational force that propels the ball, and the default direction is downwards.

Reference: Functional Requirement 7

Input: None.

Output/Observed Effects: The ball is propelled in the direction of the gravity.

Test:

1. Start up a game.

2. Use special test code that will display the direction of the gravity and velocity applying to the ball.

3. Observe that the ball is propelled in the same direction at the gravitational force.

Obstacles

The game will provide a number of obstacles that provide special events on collision, including walls, mines, magnets and wormholes.

Reference: Functional Requirement 8

Input: None.

Output/Observed Effects: Observe the specific behavior for collision with an obstacle.

Test:

1. Start up a game.

2. Observe behavior upon collision with an obstacle.

Mines

Mines shall, upon coming in contact with the ball, return the ball to the levels starting point and subtract a preset amount of time from the global countdown. Mines can either be static or dynamic. Dynamic mines shall move in a predetermined pattern, static mines shall remain at their original position at all times.

Reference: Functional Requirement 9 & 10

Input: None.

Output/Observed Effects: Ball is moved to the starting point at the time is correctly updated.

Test:

1. Start up a game.

2. Collide with a mine

3. Observe that the right amount of time is subtracted, and that the position of the ball is updated to the beginning of the map.

Magnets

Magnets shall exercise a force upon the ball if it is within a given distance and alter the trajectory of the ball.

Reference: Functional Requirement 11

Input: None.

Output/Observed Effects: The ball's trajectory is changed.

Test:

1. Start up a game.

2. Move into the range of the magnet.

3. Observe that the balls trajectory and velocity is affected as described by the requirements.

Wormholes

Wormholes shall, upon contact with the ball, transfer the ball to a corresponding wormhole in another location on the level.

Reference: Functional Requirement 12

Input: None.

Output/Observed Effects: The ball's position is updated to the outgoing wormhole.

Test:

1. Start up a game.

2. Move the ball into a wormhole

3. Observe that the ball is moved to the corresponding wormhole as specified by requirements.

Walls

Walls are the standard impediment. Upon contact with the ball, the ball bounces off at a trajectory determined by the angle of the wall and the balls original vector.

Reference: Functional Requirement 13

Input: None.

Output/Observed Effects: The ball bounces off the wall.

Test:

1. Start up a game.

2. Observe that the wall will hinder movement upon collision with the ball.

3. Use specific test code that displays the ball's current velocity and speed to confirm that it bounces off as specified by the requirements.

Keys

The user shall be able to collect keys throughout the levels. Upon collecting all the keys, the exit point for that level shall be unlocked.

Reference: Functional Requirement 14

Input: None.

Output/Observed Effects: The exit point is unlocked.

Test:

1. Start up a game.

2. Collect all the keys in the level by colliding with them.

3. Observe that the exit point is unlocked when all keys are collected by moving the ball into it.

Exit Point

Upon reaching the exit point for a level, the next level shall begin. Players shall receive additional time for the global countdown, determined by which level they just completed.

Reference: Functional Requirement 15

Input: None.

Output/Observed Effects: The new level is loaded and the time is updated.

Test:

1. Start up a game.
2. Collect keys and enter the exit point by colliding with it.
3. Observer that the next level is loaded and that the extra time is added.

Power-ups

The player shall be able to collect power-ups throughout the level. These shall include time-increases, low gravity, reverse gravity and freeze.

Reference: Functional Requirement 16

Input: None.

Output/Observed Effects: The effect caused by the power-up is observed.

Test:

1. Start up a game.
2. Collect a power up by moving into it.
3. Observe if the specified behavior is as expected.

Time Increase power-ups

Time Increase power-ups shall give the player additional time to the global countdown.

Reference: Functional Requirement 17

Input: None.

Output/Observed Effects: Time is added to the global countdown

Test:

1. Start up a game.
2. Collect the power up by moving into it.
3. Observe that the right amount of time is added.

Low Gravity power-ups

Low Gravity power-ups shall lower the gravitational acceleration of the ball, resulting in changes to how the ball behaves when bouncing. This effect ceases after a given time.

Reference: Functional Requirement 18

Input: None.

Output/Observed Effects: The ball will bounce higher.

Test:

1. Start up a game.

2. Collect the power up by moving into it.

3. Observe that the gravity changes by using the same method as specified in the gravitational force and that it returns to normal after given time.

Reverse gravity power-up

Reverse Gravity power-ups shall change the direction of the gravitational force, in effect turning the ceiling into the floor and vice versa. This effect ceases after a given time.

Reference: Functional Requirement 19

Input: None.

Output/Observed Effects: The ball will be propelled in the direction of the new gravitational force.

Test:

1. Start up a game

2. Collect the reverse gravity power-up by colliding with it.

3. Observe that the direction of the gravitational force has changed making the ceiling into the floor and vice versa.

Freeze power-up

Freeze power-ups shall stop the motion of all dynamic mines throughout the level. This effect ceases after a given time.

Reference: Functional Requirement 20

Input: None.

Output/Observed Effects: All the dynamic mines become static for the duration of the power-up.

Test:

1. Start up a game.

2. Collect the freeze power-up by colliding with it.

3. Observe if the dynamic mines become static for the duration of the power-up.

Increasing the difficulty

The game shall provide a number of levels. If the player completes all of these, the player shall return to the first level and shall be subject to increased difficulty. This shall take the shape of increased penalties for hitting mines, less time given upon level completion and less power-ups given across the level. This effect is cumulative until the game ends.

Reference: Functional Requirement 21

Input: None.

Output/Observed Effects: The required changes are in affect.

Test:

1. Start up a game.

2. Complete all the levels.

3. Observe if the game sends the player to the first level.

4. Observe if the difficulty is increased by controlling the time left, damage from mines and number of power-ups during the level.

Game over screen

When the global countdown reaches zero, the game shall display a Game Over screen, on which shall be displayed the total time played. It shall also display whether or not the time was sufficiently high to be displayed on the High score.

Reference: Functional Requirement 22

Input: None.

Output/Observed Effects: The game over screen is displayed together with the additional information.

Test:

1. Start up a game.

2. Play the game for 1-2 minutes and then lose on purpose.

3. Observe if the game over screen is displayed together with the additional information.

Sufficient score at the game end

If the player time was sufficient to be displayed on the High Score, the game shall allow the player to input his signature at the specified place.

Reference: Functional Requirement 23

Input: None.

Output/Observed Effects: The Highs core is displayed and the player's name is on the list.

Test:

1. Observe that the High Score is presented.

2. The player enters his name.

3. Observe that the players score and name are present on the list.

Insufficient score at the game end

If the players time was not sufficient to merit a mention on the High Score, the High Score shall be displayed.

Reference: Functional Requirement 24

Input: None.

Output/Observed Effects: The High Score is displayed and the player's name is not on the list.

Test:

1. Observe that the High Score is presented.

2. Observe that the players score is not present on the list.

Call interrupts

If the phone receives a call during play, the game shall pause, and the phone shall use its standard procedures for handling incoming calls. Upon terminating the call, the game shall be able to continue.

Reference: Functional Requirement 25

Input: System interrupt message.

Output/Observed Effects: The game is paused and resumable.

Test:

1. Send a call to the mobile phone phone.
2. Observe if the game enters a paused state.
3. Observe if the game can be continued.

Message interrupts

It the phone receives an sms, mms or some other form of message, the game will be paused.

Reference : Functional Requirement 26

Input: System interrupt message.

Output/Observed Effects: The game is paused and resumable.

Test:

1. Send an sms and mms to the mobile phone
2. Observe if the game enters a paused state.
3. Observe if the game can be continued.

System interrupts

If the phone issues a warning or another kind of system message, the game shall be paused.

Reference: Functional Requirement 27

Input: System interrupt message.

Output/Observed Effects: The game is paused and resumable.

Test:

1. Force warnings and phone messages to the mobile phone.
2. Observe if the game enters a paused state.
3. Observe if the game can be continued.