Project Flip Jump

Group 17

Mikael Ahlberg
Daniel Ericsson
Axel Stenkula
Johannes Svensson
Fredrik Vretblad

Table of Contents

1 Introduction

Version history

Version	Date	Rational	Changes	Authors
1.0	2008-03-07	Initial version	-	Mikael Ahlberg Daniel Ericsson Axel Stenkula Johannes Svensson Fredrik Vretblad

Purpose
The purpose of this document is to define the system functionalities for the programmers to make the implementation easier. It also makes sure that we implements the requirements from the Requirements Document.

Scope
This document contains all the classes in the system and how interact with each other. Different test cases to ensure that the requirements from the Requirements Document are fulfilled are also included along with the methods of the game classes. The graphical user interface is also defined in this document.

Intended audience
The intended readers of this document are mainly the programmers that will implement the system, but may also include testers who will test the functionality of the system.

Related documents
We expect that the reader is familiar with our Requirements Document.

Abstract
This document describes the expected way to implement Project Flip jump, with detailed descriptions of how the game will look, how the architecture will be and data/control flow of the game.

2 System Overview

2.1 General Description

Flip jump is a 2D-game where the player is intended to jump between blocks in order to avoid falling down. The screen is moving upwards to force the player to move upwards. The player can move sideways, jump and collect special items. If the player moves out from the sides of the screen, then the player appears on the other side of the screen. Another feature of the game is the flip function, which rotates the screen in a random direction. The game will then continue in the new direction. We will also provide a high score list, in order to let the player keep track of his/her best score.

We will use Lightweight Java Game Library (LWJGL) when implementing our system, instead of using Java-2D. The reason for this is that Java-2D doesn't provide good hardware acceleration for graphical intensive software.

2.2 Overall Architecture Description

2.2.1 Graphics manager

This object only handles the graphical part and do not affect any other part of the system. It will receive information from the Game logic object for processing. The processed information is then sent to the graphics card which then draws it on the display.

2.2.2 Audio manager

This object only handles the audio and do not affect any other part of the system. The Audio manager will receive information from the Game logic object for processing and then send the result to the Audio card.

2.2.3 Game logic

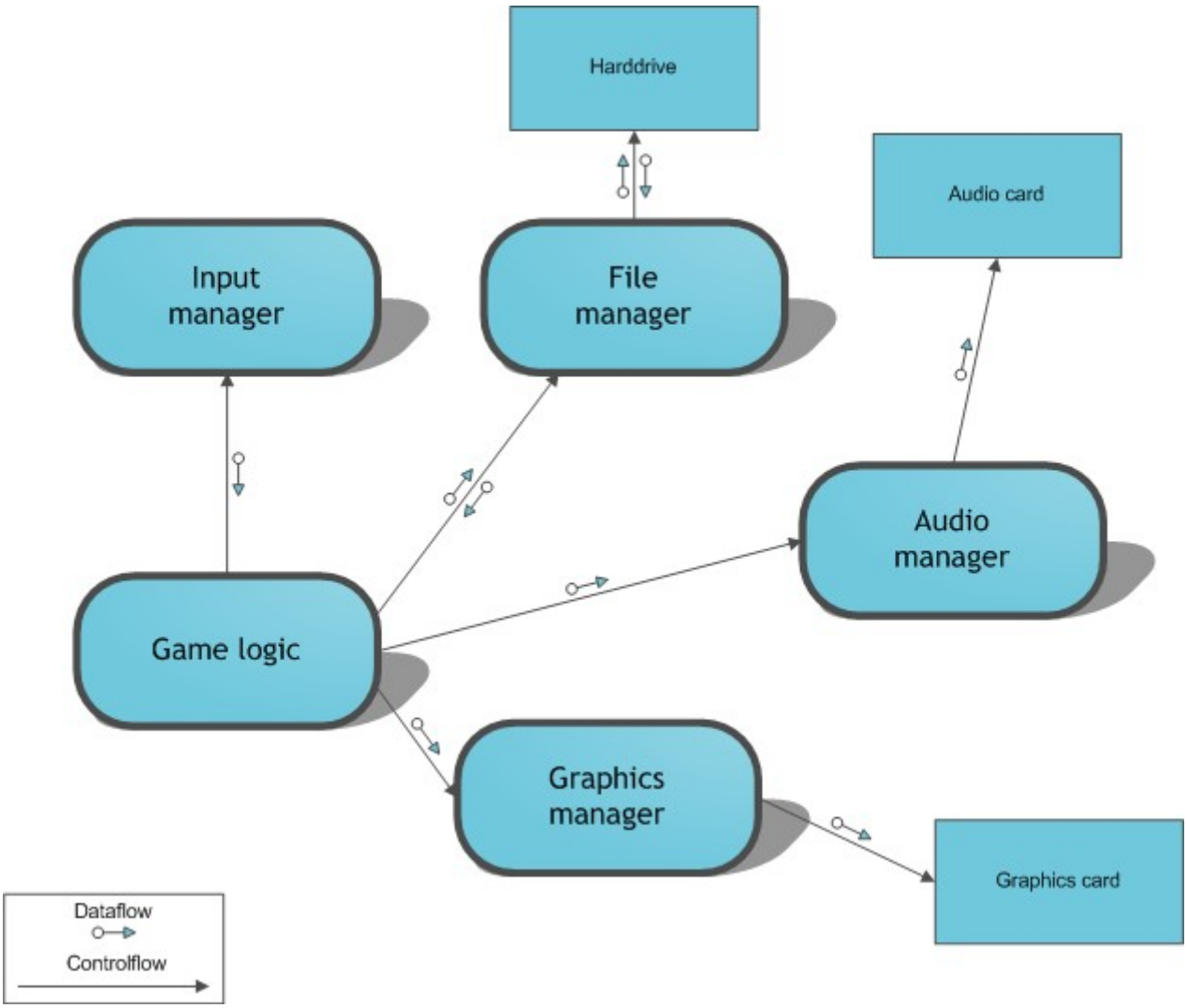
This object controls everything. This will contain the main loop of events. The Game logic will control everything from the game physics and game rules to what is to be heard and seen (audio and graphics). It will receive information from the Input object about what the user is up to, and acts accordingly. The Game logic also sends and receives information to and from the File system object.

2.2.4 Input manager

The Input manager needs to be started by the Game logic object to be able to listen to input. This object interprets the input from the keyboard and sends the information to the Game logic object.

2.2.5 File manager

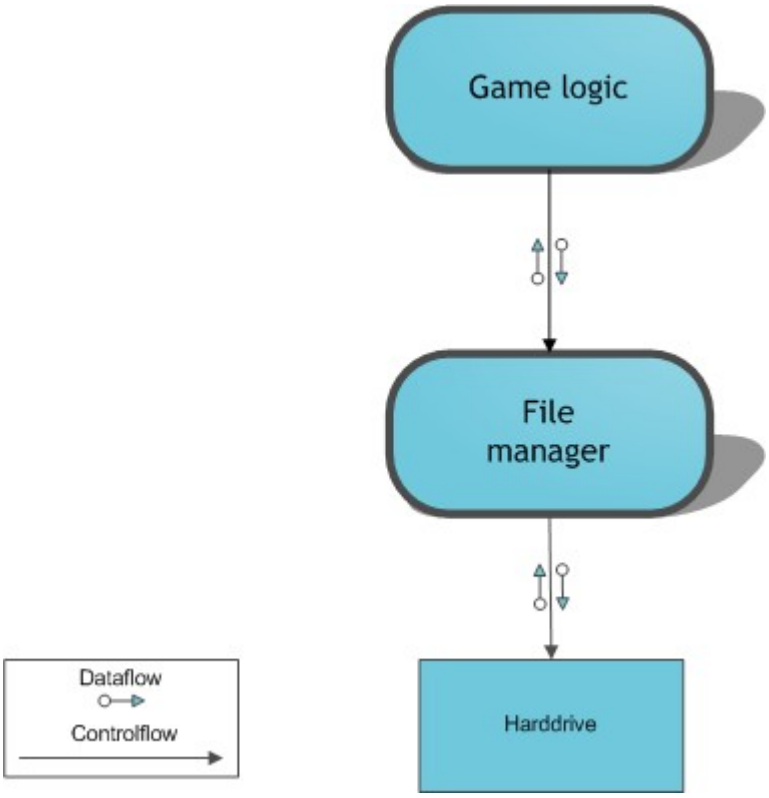
This object controls the input and output from and to files on the hard drive, such as textures, sounds, high score list, settings and similar.



We have used a data- and control flow diagram, which shows how our system works. The rounded rectangles represents the software and the others represent the hardware. We will use the same type of diagram in following sections.

2.3 Detailed Architecture

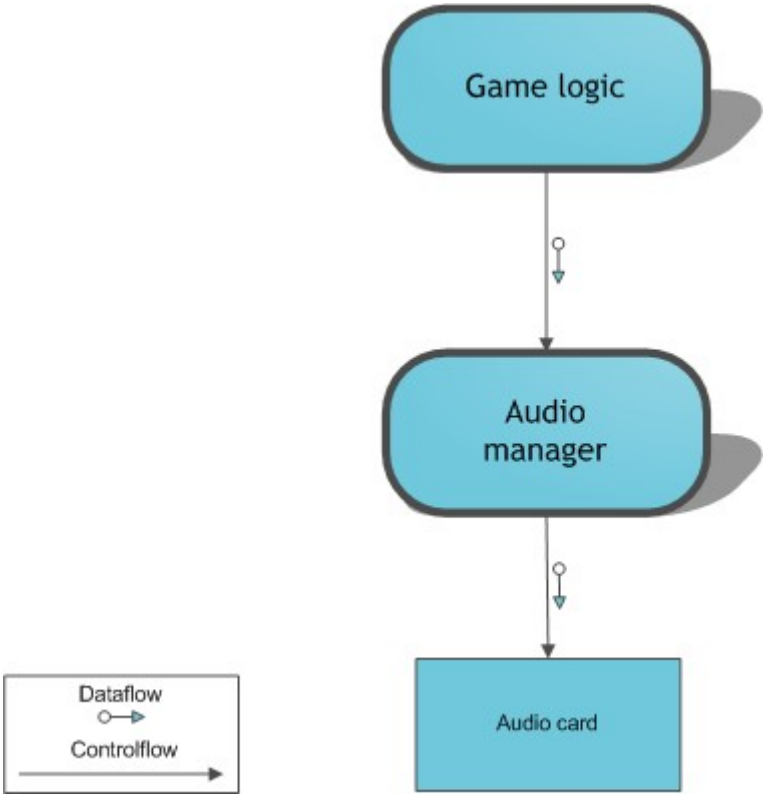
2.3.1 File manager



Control flow
The control flow for the connection between the Game logic components and the File manager is a one-way control channel. The Game logic components will access the hard drive through the File manager layer. Inside the File manager there will be functions to load and store data on the hard drive where all components are stored. This will be controlled by the Game logic since the File manager can't know when data is needed in-game.

Data flow
Data flows in both direction, but most data is transfered from the hard drive through File manager to end up in the Game logic. Typical data that is transfered in this direction is sound files, bitmap textures and stored settings for the game. In other words, everything the game needs to load to be able to run. In the other direction, from the Game logic to the hard drive, we have for example storing of game settings and high score list.

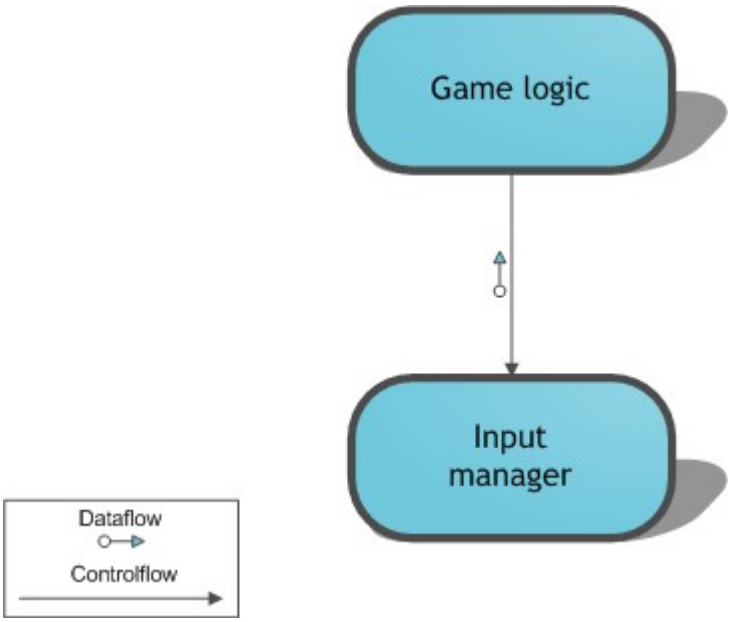
2.3.2 Audio Manager



Control flow
The Game logic components will, when requested, command the Audio manager to play a specified sound clip. The Audio manager will then handle the connection to the computers Audio card and make sure that the sound clip is played correspondingly.

Data flow
The data flow for the Audio manager is a one-way channel, from the Game logics through the Audio manager to the computer Audio card. The data will contain in-game sound effects and/or music from the game. The Game logic has received data from the File manager.

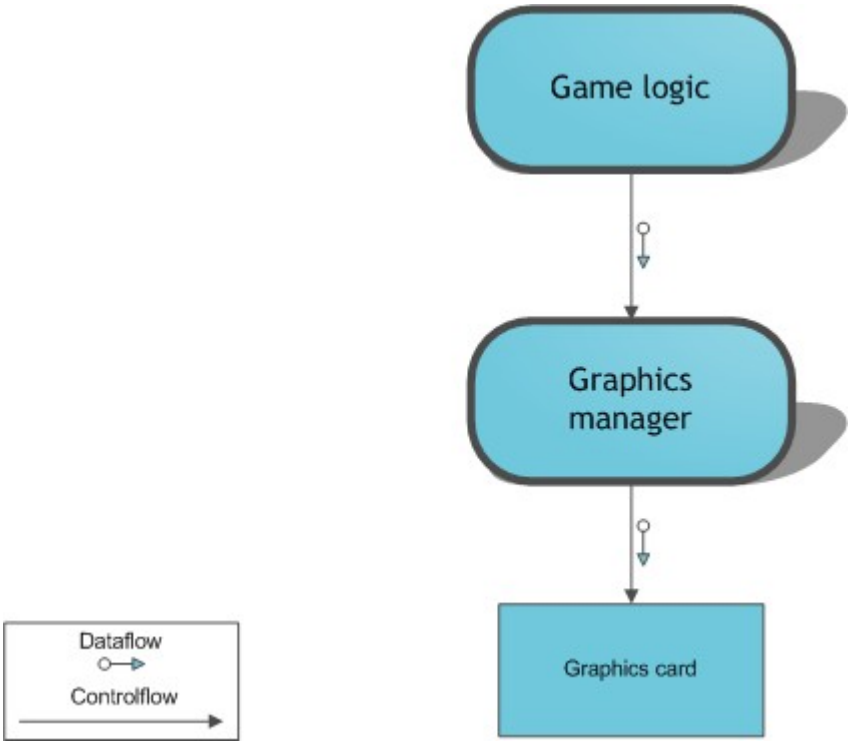
2.3.3 Input manager



Control flow
The Game logic will frequently control with the Input manager if there's any new input from the user. The Game logic will act accordingly to the input. The Input manager will act as a layer between the Game logic components and the system hardware.

Data flow
This object interprets the input from the keyboard and sends the information to the Game logic object. When the users presses keyboard buttons, an event is sent from the Input manager to the Game logic object.

2.3.4 Graphics manager



Control flow
The Game logic components will, when requested, command the Graphics manager to draw the current frame or state on the screen. The Graphics manager will then handle the connection to the computers graphics card and make sure that the frame is drawn correctly on the screen.

Data flow
The Graphics manager will receive information from the Game logic object for processing. The processed information is then sent to the graphics card which then draws it on the display. All the graphical content supposed to be on screen will be drawn by the Graphics manager. This includes the game character, textures, background and menus.

3 Design Considerations

3.1 Assumptions and Dependencies

The assumptions and dependencies regarding our the software and its use are:

- Working computer
  The user must have a working computer to start and play the game. The system will also require that the user has a graphics card which supports OpenGL.

- Keyboard
  The user must have a keyboard connected to the computer to be able to play our game.

- Operating system
  The user has one of these operating systems: Windows XP, Mac OS X and Linux.

- Java >= 1.5
  Since we will write the game for Java version 1.5 the user must have version 1.5 or later of the Java Runtime Environment (JRE).

- Know how to start an application
  The user needs to know how to start the application, which is done either by clicking on the game binary or running the game from a terminal.

- Understand a simple menu system
  The user needs to know how you usually control a normal menu system.

- Understand English
  The user needs to be able to read and understand English, since this game will only feature English text.

Changes in functionality will only appear if our time gets limited and we need to implement the game without some of it's intended functionality. If that happens, we will only implement the high and medium priority requirements in the Requirements Document.

3.2 General Constraints

When implementing the system we need to think of the memory usage of the system. We will load each texture for each in-game object only ones to save memory. This will lower the memory usage of the system.

4 Graphical User Interface

4.1 General overview

When the game starts the user will see the main menu which will display several options. These options or selectable menu items are ”Start Game”, ”High Score”, ”Options” and ”Quit”. If the user presses Quit the game will exit and return to the computers previous state. When pressing the High score-button, a new screen will appear, showing the complete score list. The Options menu-alternative will also open a new screen with a menu of the different selectable settings. A more detailed description for the High score and Options menu item will follow below.

When the Start Game-button is selected, a new menu will occur with the different difficulty levels. After the difficulty level has been selected, a tutorial text will be presented to the user. The user selects ”Start Game” when he or she has read the tutorial text and the game will start.

A feature of the game is the ability to make a special jump. This jump can be made if the player has collected a special item. Another feature is the flip-function which rotates the screen and forces the user to continue in another direction. When the screen speed increases to a higher level an indication will appear on the screen. The user score will be displayed on the screen because the user needs to know his or her current score. These functions will be described in more detail further down.

4.2 Detailed description of the user interface

4.2.1 Main menu

Description
The main menu contains a list of different selectable alternatives concerning the game. As you can see in the screen shot the alternatives are ”Start Game”, ”High score”, ”Options” and ”Quit”. You navigate in this menu by pressing the arrow keys to choose which alternative you want to select. If you choose ”Start Game”, the difficult level will be selectable. If you choose the ”High score” alternative, the high score list will be presented. If you choose ”Options” the options menu will appear and last if you choose ”Quit” the game will end.

Reference to Functional Requirement
None.

4.2.2 Difficulty level

Description
The user has a choice to choose the difficulty level of the game before it starts. There are three different difficulty level alternatives, ranging from easy to hard, that affects the toughness of the game. There is also the option to go back to the main menu if the user wishes to do that.

Reference to Functional Requirement
Functional Requirement 4.1.2 Game: 7. Difficulty level

4.2.3 Tutorial text

Description
When the user has selected the difficulty level for the game, this tutorial menu will appear. It contains a short text that explains how to play the game and when the user is ready, he or she can press the ”Start game” button.

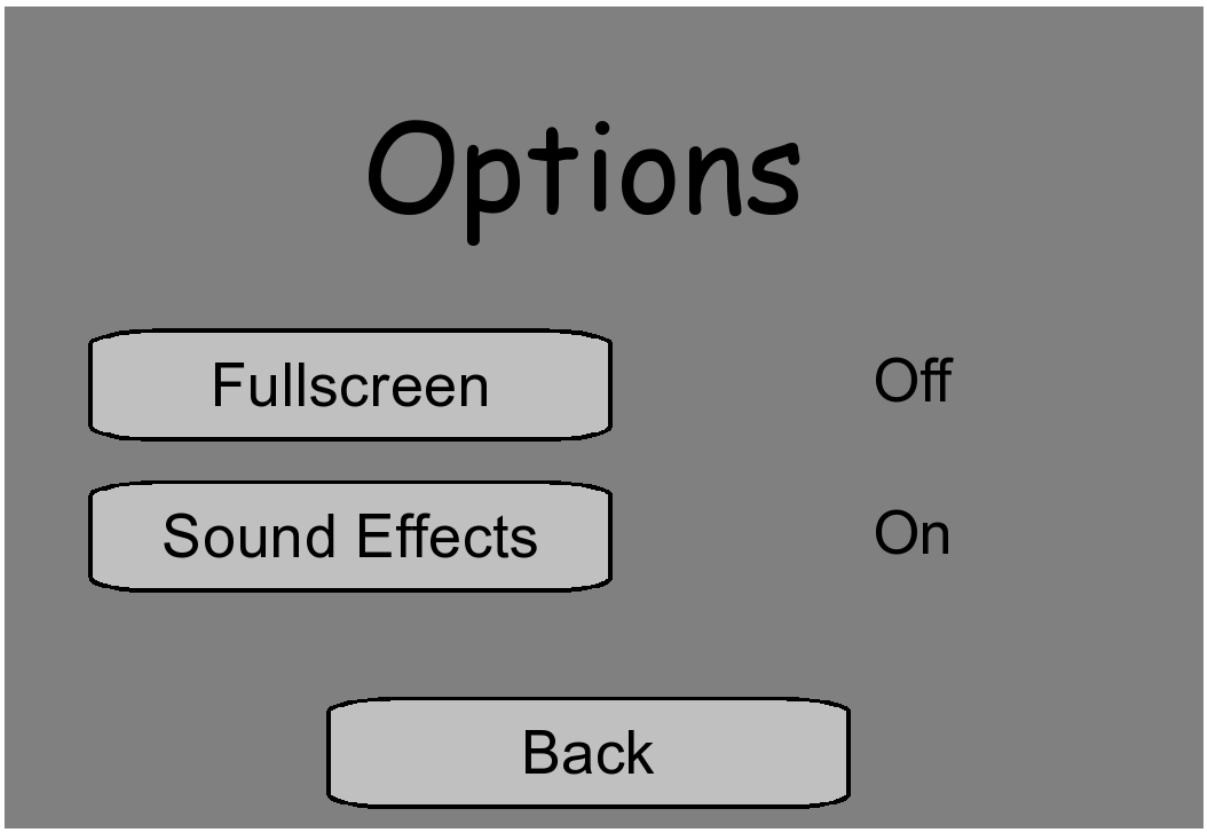
Reference to Functional Requirement
Functional Requirement 4.1.1 General: 1. Tutorial text

4.2.4 High score list

Description
The high score list contains information regarding the top players score in a descending order. The list is shown when the user selects the “High score” alternative from the main menu or when the user gets game over in-game. It can at a maximum contain the 10 best players. The “Back” alternative will take the user back to the main menu.

Reference to Functional Requirement
Functional Requirement 4.1.3 High Score: 3. Check High Score

4.2.5 Options menu



Description
When the user enters the options menu, he or she will be presented with the following alternatives: ”Fullscreen”, ”Sound Effects” and ”Back”. When the user toggles the Fullscreen button, the window of the game will switch from window-mode to fullscreen mode or vice versa. The current status of the setting is, as you can see, displayed on the right side of the alternatives. The ”Sound Effects” button will switch all sounds in the game on or off and the ”Back” alternative will take the user back to the main menu.

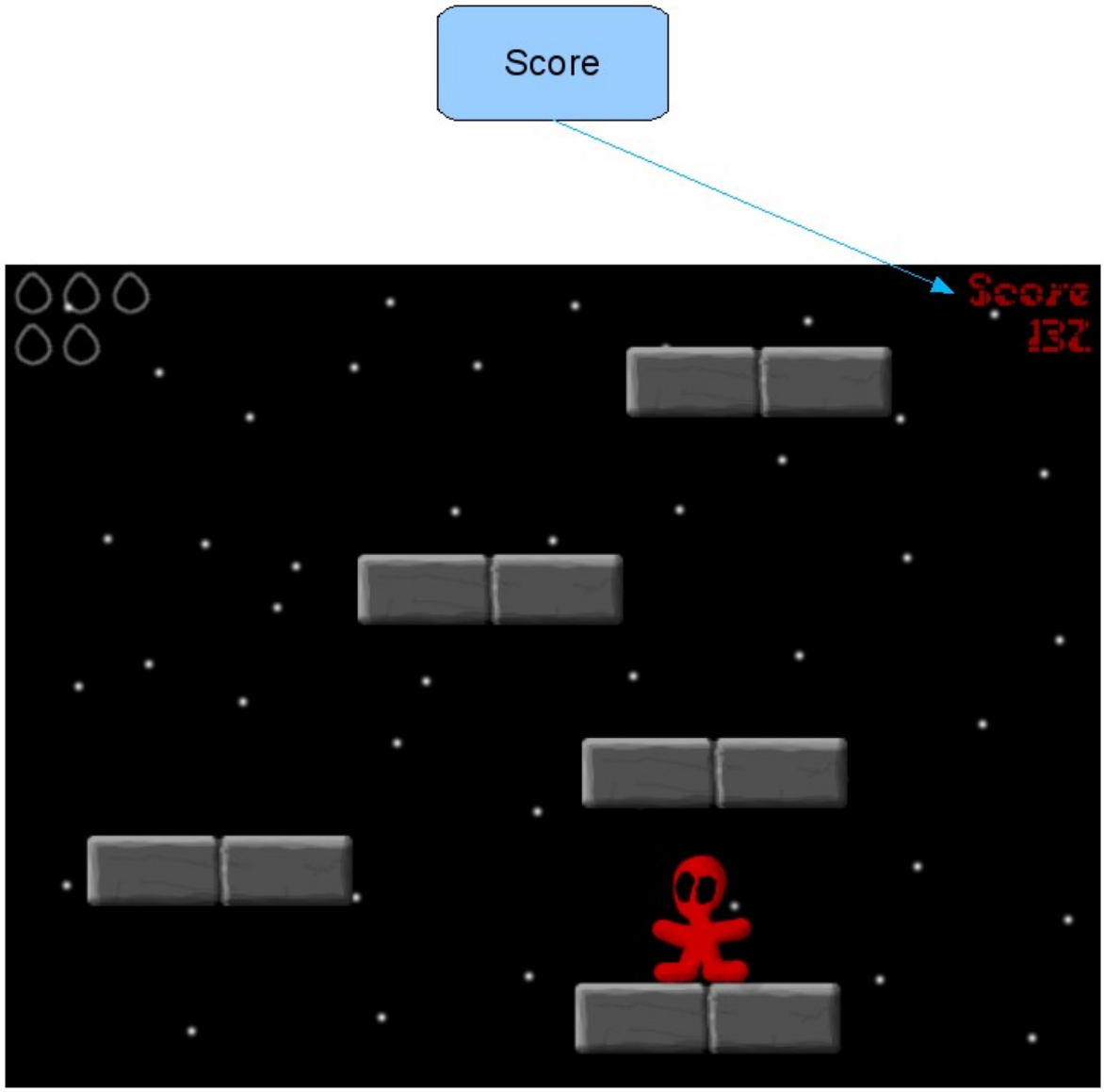
Reference to Functional Requirement
Functional Requirement 4.1.1 General: 3. Screen size
Functional Requirement 4.1.1 General: 4. Appearance of sound

4.2.6 Enter high score list

Description
If the user has beaten a score on the high score list, the user will be presented with the following menu where the user can enter his or her name. There is also a “Main menu” alternative that will take the user back to the main menu.

Reference to Functional Requirement
Functional Requirement 4.1.3 High Score: 2. Enter High Score list

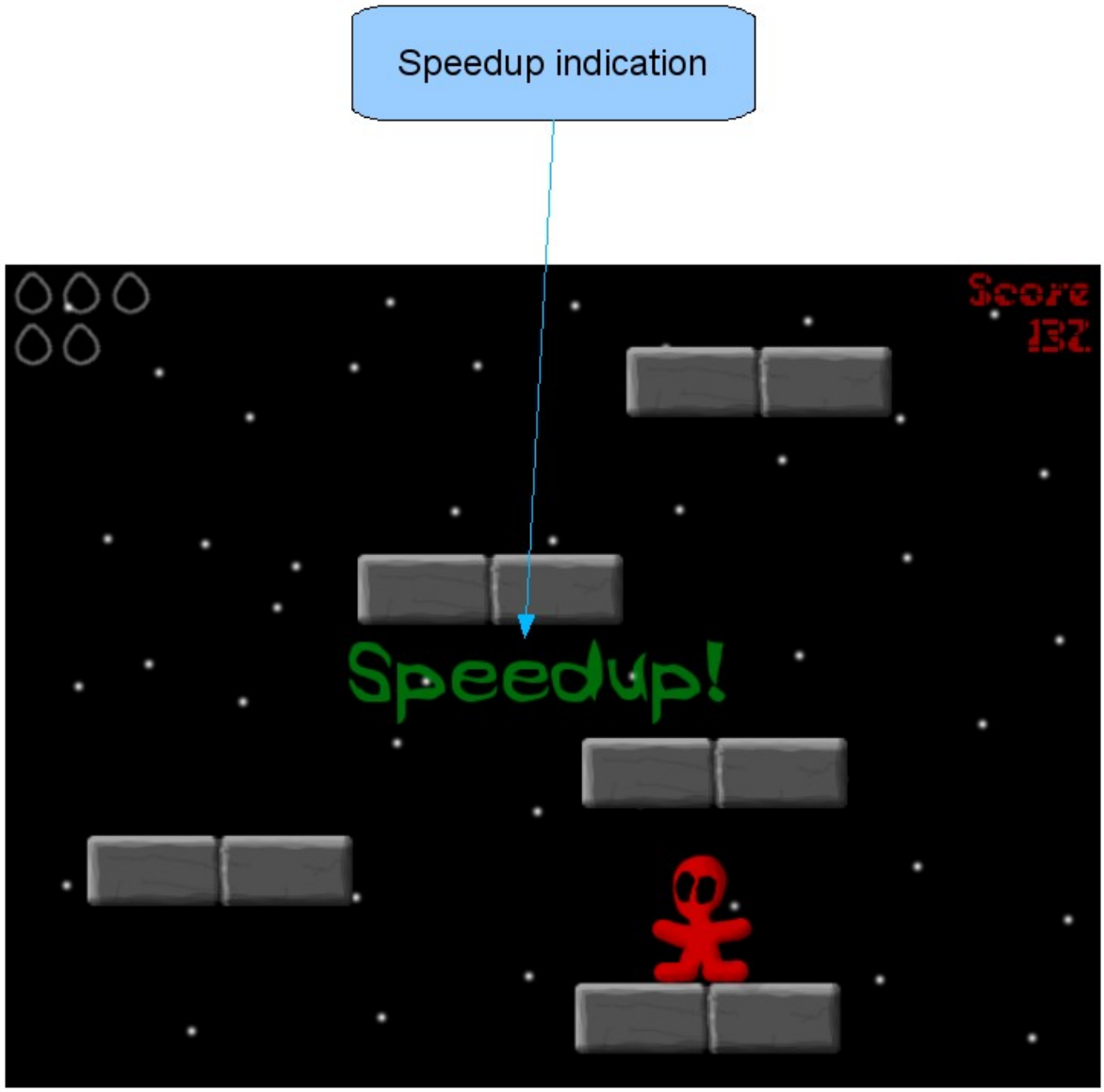
4.2.7 Showing score



Description
The user will receive points when reaching a higher block. The current score will be displayed in the top right corner to let the user know how many points he or she has collected so far.

Reference to Functional Requirement
Functional Requirement 4.1.2 Game: 10. Receiving points
Functional Requirement 4.1.2 Game: 11. Showing score

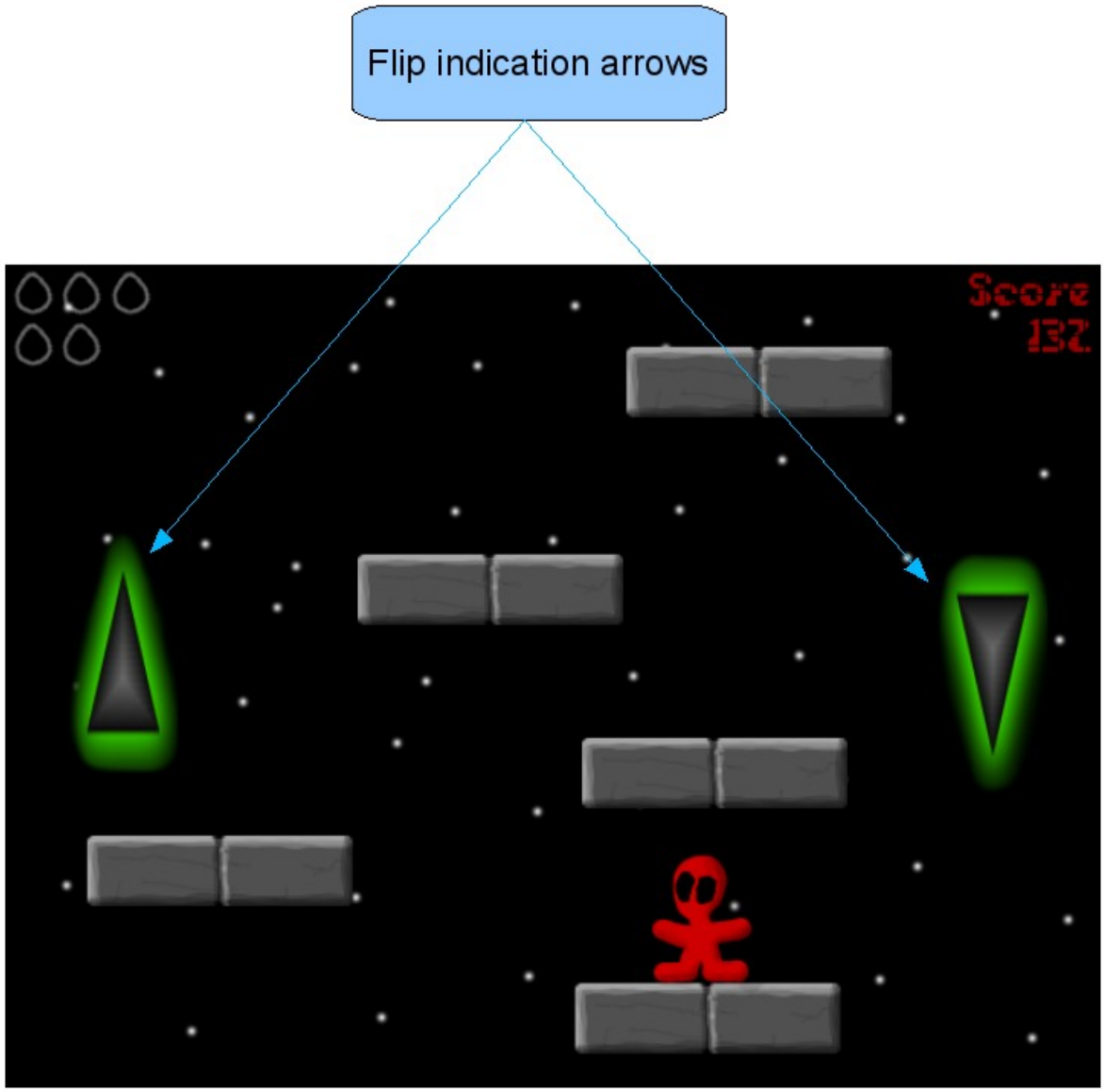
4.2.8 Screen speed increase indication



Description
When playing the game, the screen will move upwards in a certain speed and the user must jump to higher blocks in order to survive. The screen speed will at certain times increase. The increment will be indicated by the Speedup indication in the middle of the screen and also a sound.

Reference to Functional Requirement
Functional Requirement 4.1.2 Game: 8. Screen speed
Functional Requirement 4.1.2 Game: 9. Screen speed increase indication

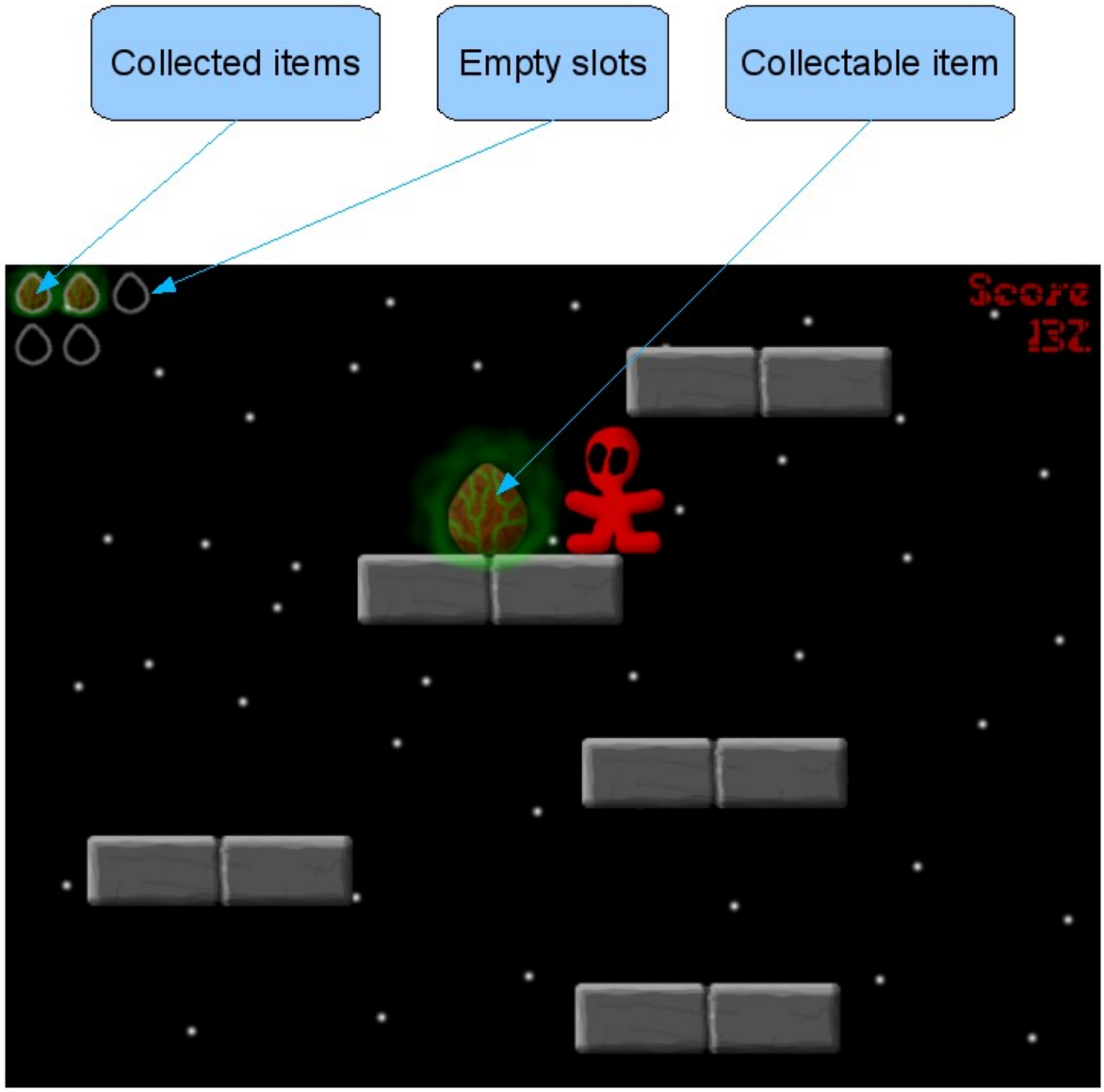
4.2.9 Flip indication



Description
A special feature of the game is the flip function. When the user plays the game the flip indication arrows may suddenly appear at the sides of the screen showing in which direction the screen will flip. The screen will flip shortly after the arrows appeared and the user must continue its journey in a new direction.

Reference to Functional Requirement
Functional Requirement 4.1.2 Game: 12. Flip function
Functional Requirement 4.1.2 Game: 13. Flip indication

4.2.10 Special item



Description
The empty slots indicates how many more items you can collect. If the character touches the collectable item, the item will disappear from the screen and appear as a collected item in the top left corner. The collected items can be used to make a special jump. When an item has been used the corresponding collected item will disappear from the screen.

Reference to Functional Requirement
Functional Requirement 4.1.2 Game: 3. Collect special item
Functional Requirement 4.1.2 Game: 4. User special item

5 Design Details

5.1 Class Responsibility Collaborator (CRC) Cards

Abstract Class Entity	
Responsibilities:	Collaborators:
• Stores it's own coordinates. • Checks for collision with another object.	• Item • Block • Player • World

Class Player	
Responsibilities:	Collaborators:
• Extends Entity. • Keeps track of it's own textures. • Keeps track of it's current movement direction. • Keeps track of amount of items carried by the player.	

Class Item	
Responsibilities:	Collaborators:
• Extends Entity. • Keeps track of it's own textures. • Keeps track of it's current movement direction.	

Class Block	
Responsibilities:	Collaborators:
• Extends Entity. • Keeps track of it's own textures. • Keeps track of it's current movement direction.	

Class Settings	
Responsibilities:	Collaborators:
• Contains game settings. • Reads and writes settings from and to a settings file.	

Class HighScore	
Responsibilities:	Collaborators:
• Keeps track of current high score. • Reads and writes settings from and to a high score file.	

Class Menu	
Responsibilities:	Collaborators:
• Contains coordinates for buttons (placement). • Keeps track of current menu alternative. • Keeps track of textures for the different parts of the menu.	• HighScore

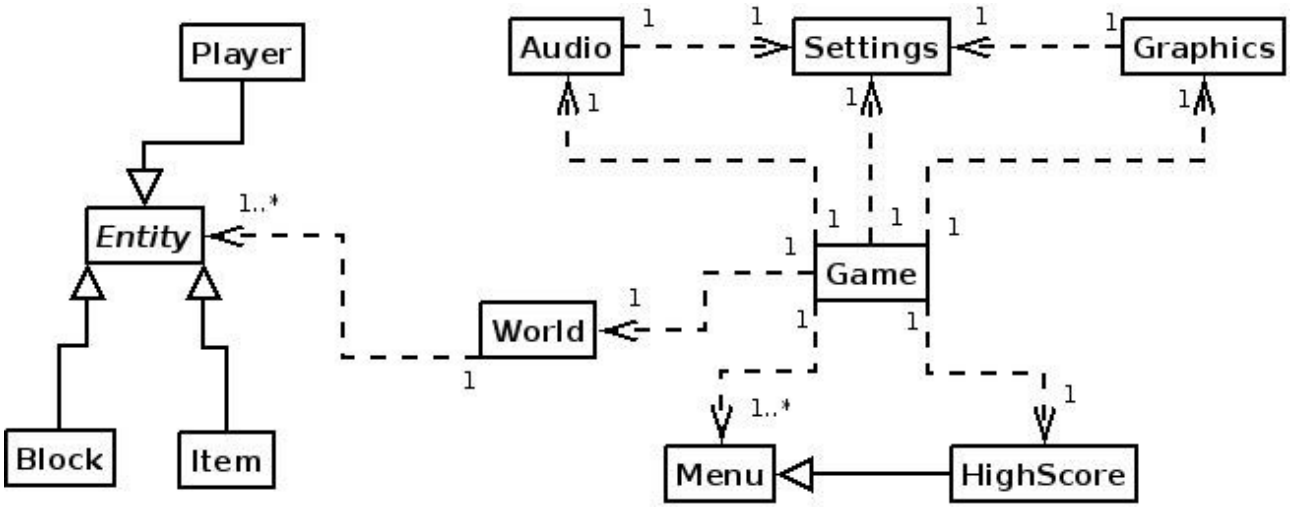
Class World	
Responsibilities:	Collaborators:
• Keeps track of the objects of the world. • Update coordinates for the world objects (item, block) in the game. • Animate objects which needs to be animated.	• Player • Item • Block

Class Audio	
Responsibilities:	Collaborators:
• Initiates audio card. • Sends sound to the audio card.	• Settings

Class Graphics	
Responsibilities:	Collaborators:
• Initiates OpenGL. • Handles writing of the textures to the graphics card. • Handles the rotation of the camera.	• World • Settings

Class Game	
Responsibilities:	Collaborators:
• Handles the input from the user. • Handles movement of the in-game character. • Keeps track of the users score. • Contains the game loop, which updates everything in the game.	• Menu • Entity • Graphics • Audio • World • Settings • High score

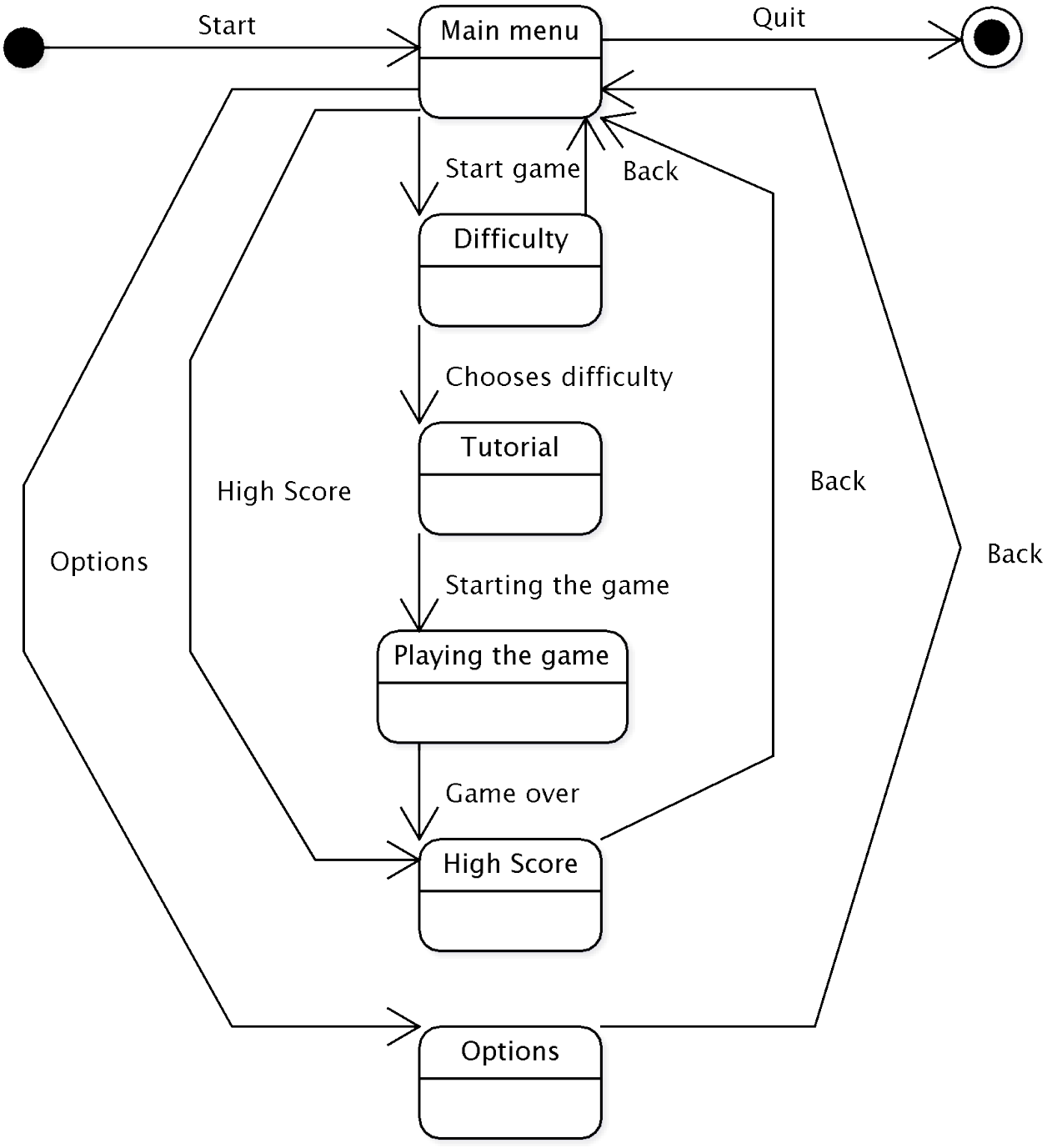
5.2 Class Diagram



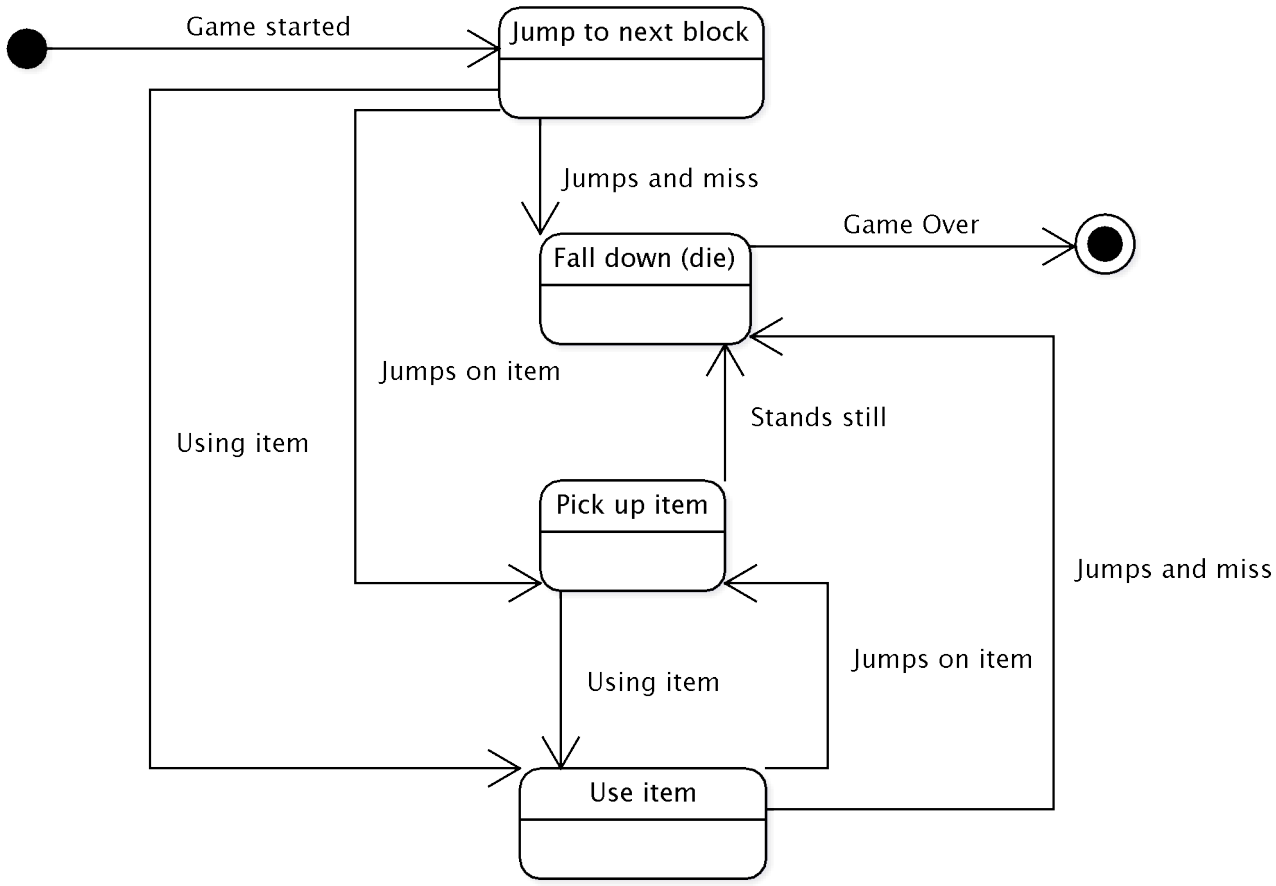
This is an UML diagram representing our class hierarchy.

5.3 State Charts

5.3.1 Menu state chart

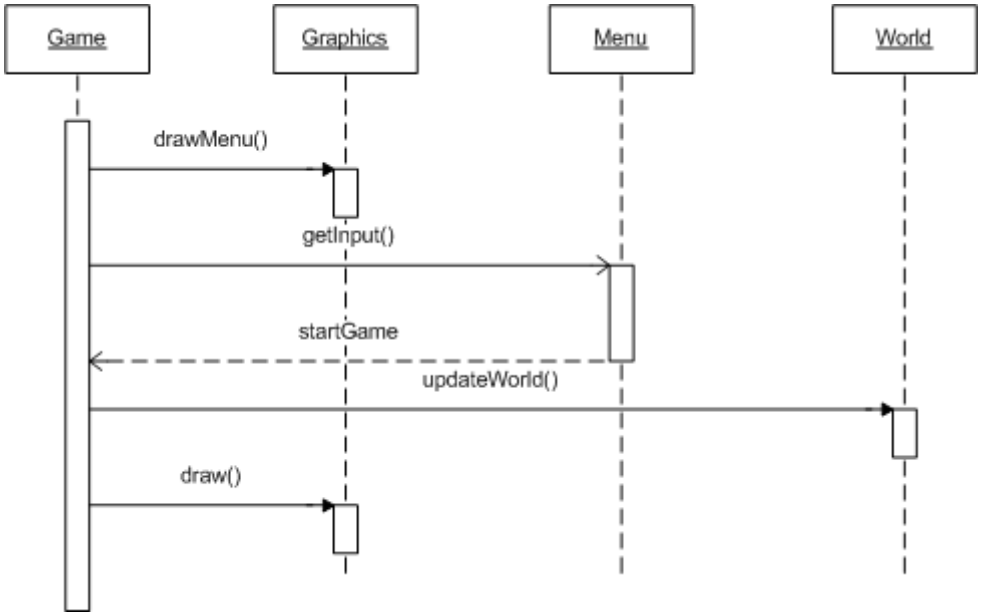


5.3.2 In-game state chart

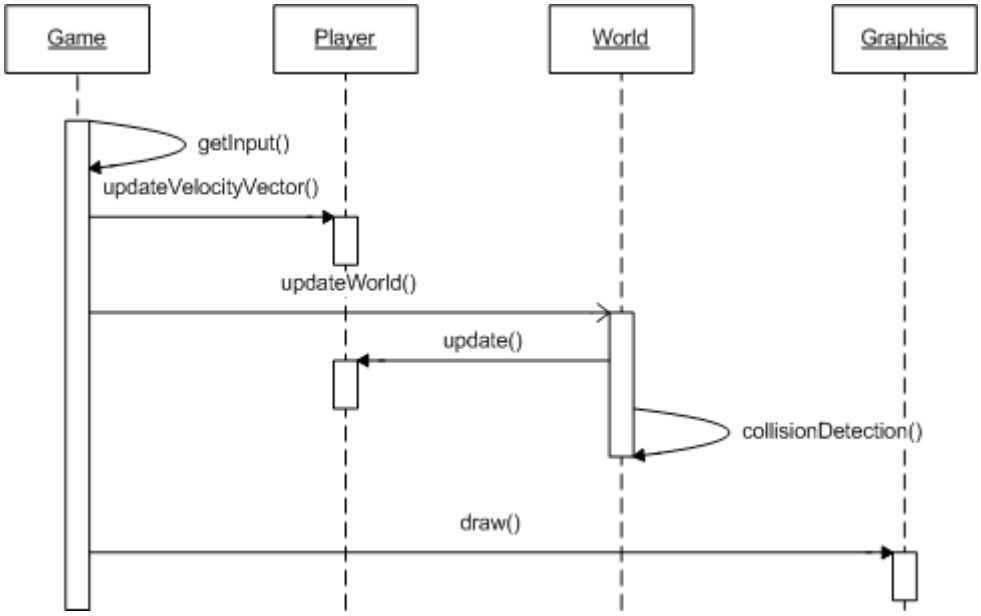


5.4 Interaction Diagrams

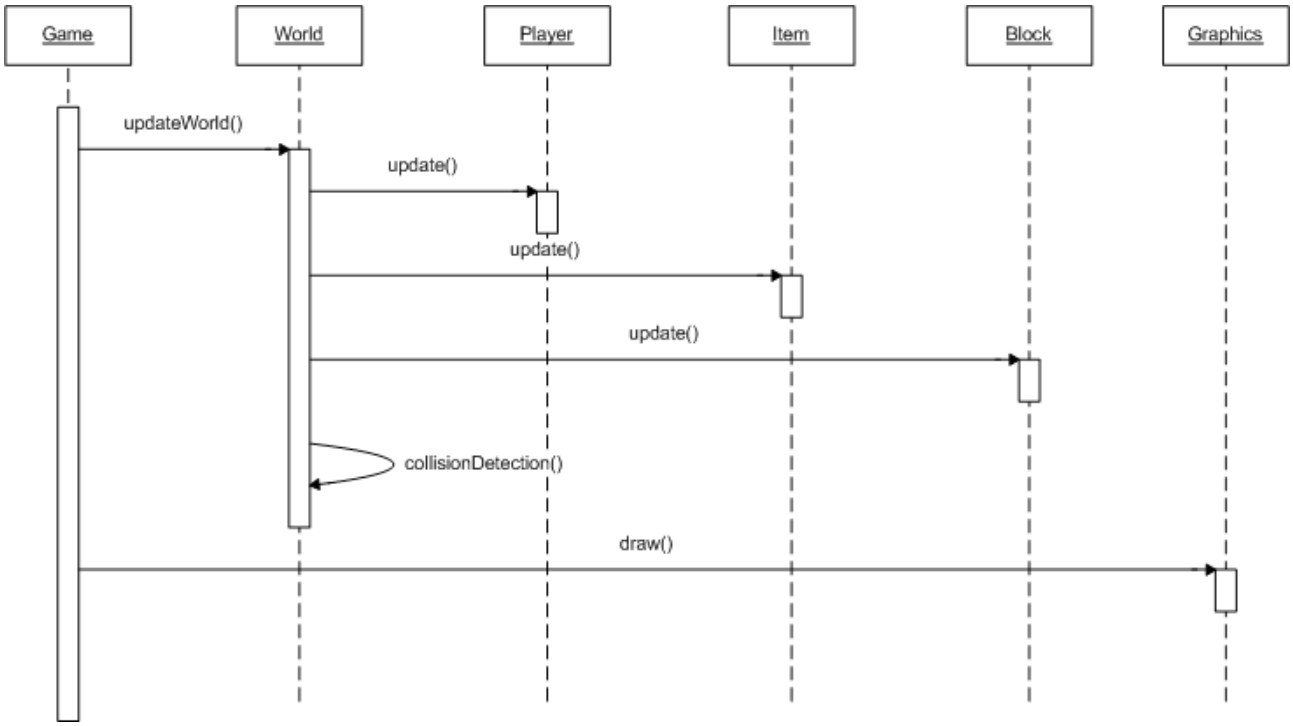
5.4.1 Start game



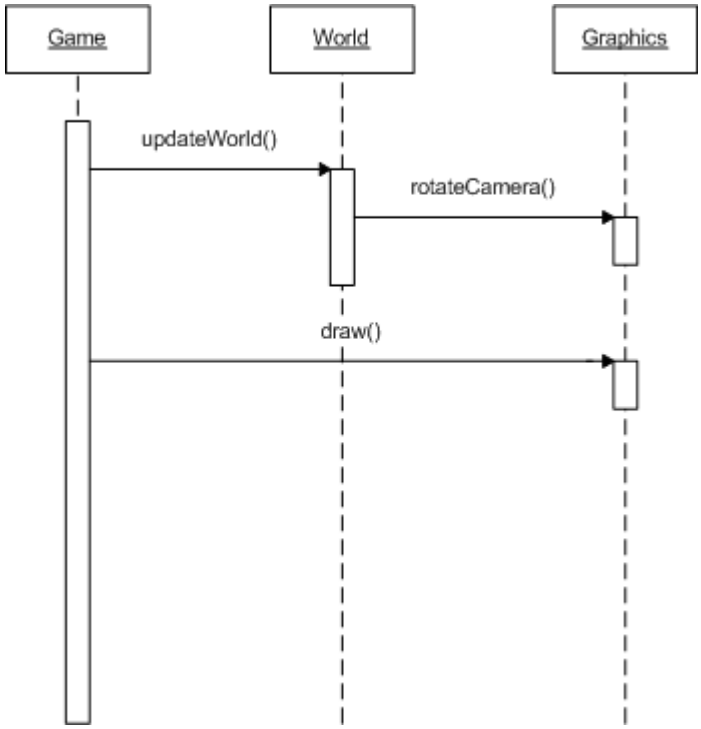
5.4.2 Player movement



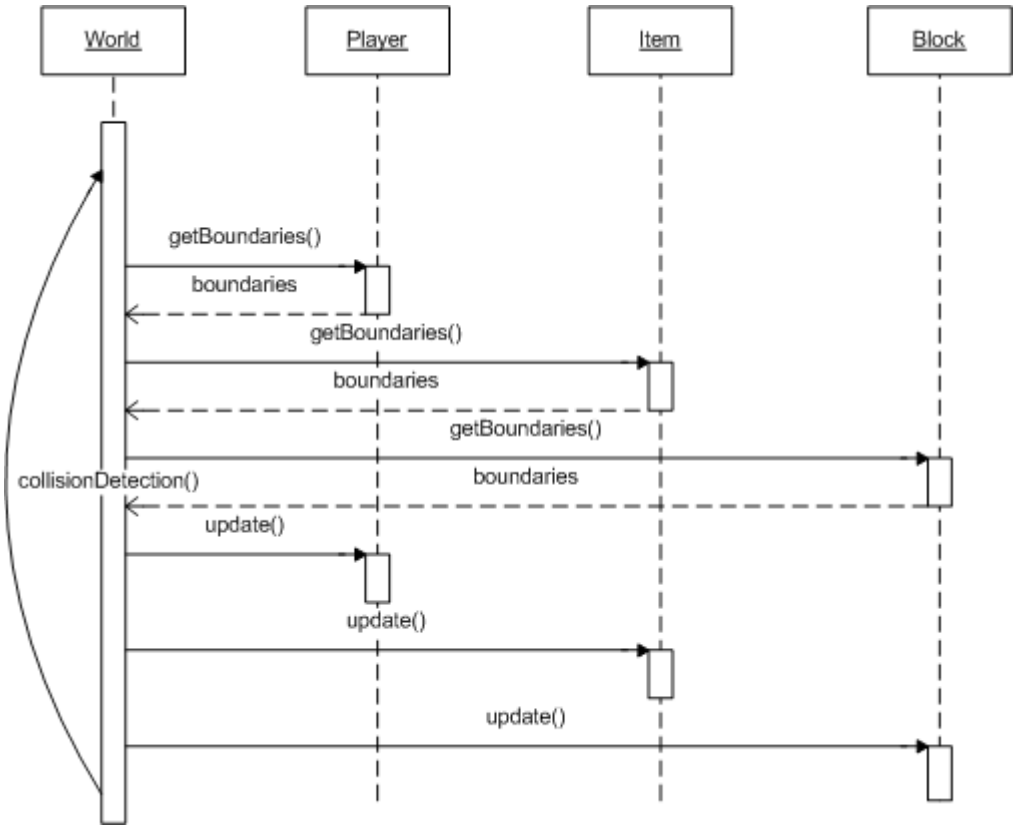
5.4.3 Screen movement



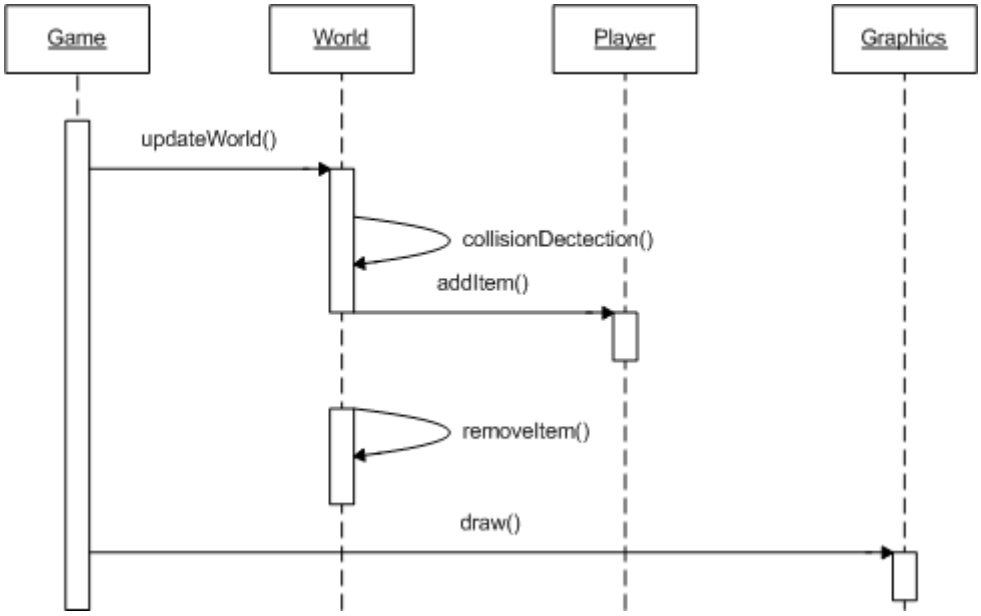
5.4.4 Flip screen



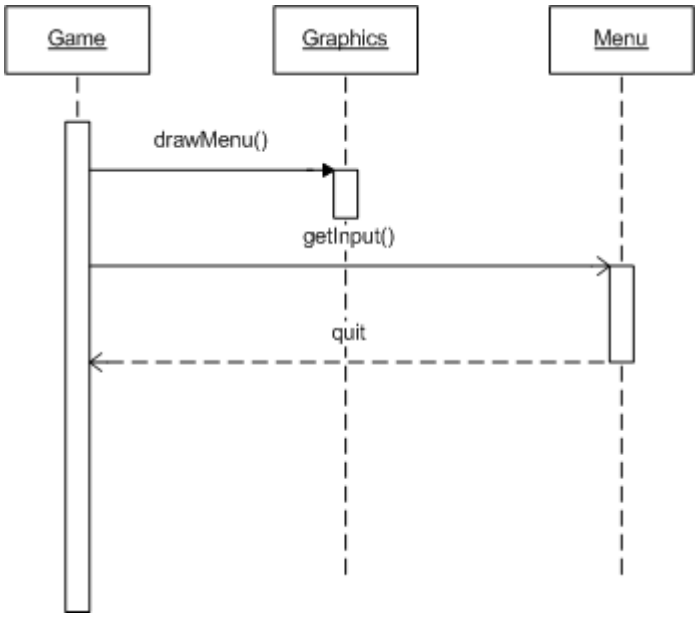
5.4.5 Collision



5.4.6 Special item



5.4.7 Shutdown



5.5 Detailed Design

5.5.1 Class Audio

Fields

Field name: private OpenALSource soundSource
Comment: Handles the sound connection for the class.

Field name: private Settings settings;
Comment: A link/reference to the settings class for easier retrieval of game settings.

Methods

Method name: public Audio(Settings settings)
Parameter (Settings): The link/reference to the settings class.
Return value: None
Description: The constructor for the class. It initiates all OpenAL code so the game is ready to play sounds.
Pre-condition: The settings class has been initiated and settings loaded.
Post-condition: The OpenAL-engine has been initialized.
Called by: The Game class constructor.
Implements: -

Method name: public void play(OpenALWave soundClip)
Parameter (OpenALWave): The sound clip to be played.
Return value: None
Description: The method plays the specified sound clip that's sent as a parameter to the method.
Pre-condition: The OpenAL-enginge has been initialized.
Post-condition: The sound clip has been played.
Called by: Called by speedup-event, highscore-event and gameover-event.
Calls: Calls OpenALSource play()-method.
Implements: 6.1.3.5 High score sound

Method name: public void stopCurrent()
Parameter: None.
Return value: None.
Description: The method stops all current sound effects.
Pre-condition: The OpenAL-enginge has been initialized.
Post-condition: All sound clips are stopped.
Called by: The Game class menuLoop()-method.
Calls: Calls OpenALSource stop()-method.
Implements: 6.1.3.5 High score sound

Method name: public boolean toggleMute()
Parameter: None.
Return value (Boolean): Returns true if the sound is switched on and vice versa.
Description: The method changes the mute settings for the game.
Pre-condition: The OpenAL-enginge has been initialized.
Post-condition: The sound has been switched on or off by the method.
Called by: The Game class menuLoop()-method.
Calls: Calls the Settings class setMute()-method.
Implements: 6.1.1.4 Appearance of sound

Method name: public boolean isMuted()
Parameter: None.
Return value (Boolean): Returns true if the sound is muted and vice versa.
Description: The method returns the current state of the mute settings.
Pre-condition: The Settings for the mute function has been initiated.
Post-condition: The value has been returned.
Called by: Called by play-event and toggleMute-event.
Calls: Calls the Settings class isMute()-method.
Implements: 6.1.1.4 Apperance of sound

5.5.2 Class Block

Fields

None.

Methods

Method name: public Block(int x, int y, int height, int width)
Parameter (int): The blocks initial x value.
Parameter (int): The blocks initial y value.
Parameter (int): The blocks height value.
Parameter (int): The blocks width value.
Return value: None
Description: The constructor for the class. It makes sure that the initial values for the block is set corrected, like dimension, texture and position in the world.
Pre-condition: The game world has been created.
Post-condition: A new block has been created in the world.
Called by: The World class constructor.
Calls: Calls the setTexture()-method in the superclass.
Implements: -

5.5.3 Abstract Class Entity

Fields

Field name: private int x;
Comment: The X-position of the entity in the world.

Field name: private int y;
Comment: The Y-position of the entity in the world.

Field name: private int velocityX;
Comment: The current speed in the X-direction.

Field name: private int velocityY;
Comment: The current speed in the Y-direction.

Field name: private int height;
Comment: The entitys height in the world.

Field name: private int width;
Comment: The entitys width in the world.

Field name: private String texture;
Comment: A path to the entitys texture.

Methods

Method name: public Entity(int x, int y, int height, int width)
Parameter (int): The entity's initial x value.
Parameter (int): The entity's initial y value.
Parameter (int): The entity's height value.
Parameter (int): The entity's width value.
Return value: None
Description: The constructor for the class. This class is an abstract class and is not instanced.
Pre-condition: The game world has been created.
Post-condition: A new entity of some kind has been created in the world.
Called by: The classes that inherits from this class.
Implements: -

Method name: public void setTexture(String texture)
Parameter (String): The path to the texture.
Return value: None
Description: The method sets the texture for the object to the specified path.
Pre-condition: A object that inherits from entity has been created.
Post-condition: The texture has been set.
Called by: The specified objects constructor.
Calls: None.
Implements: -

Method name: public String getTexture()
Parameter: None
Return value (String): Returns the path to the objects texture.
Description: The method returns the path to the objects texture.
Pre-condition: An object that inherits from entity has been created and texture has been set.
Post-condition: The texture path has been returned.
Called by: The Graphics class draw()-methods.
Calls: None.
Implements: -

Method name: public String getTexture()
Parameter: None
Return value (String): Returns the path to the objects texture.
Description: The method returns the path to the objects texture.
Pre-condition: An object that inherits from entity has been created and texture has been set.
Post-condition: The texture path has been returned.
Called by: The Graphics class draw()-methods.
Calls: None.
Implements:

Method name: public void setX(int x)
Parameter (int): The entity's X-position.
Return value: None.
Description: The method sets the entity's X-position in the world.
Pre-condition: An object that inherits from entity has been created.
Post-condition: The X-position has been set.
Called by: The World class collisionDetection()-method and the local update()-method.
Calls: None.
Implements: -

Method name: public int getX()
Parameter: None.
Return value (int): Returns the entity's X-position in the world.
Description: The method returns the entity's X-position.
Pre-condition: An object that inherits from entity has been created.
Post-condition: The X-position has been returned.
Called by: The World class collisionDetection()-method.
Calls: None.
Implements: -

Method name: public void setY(int y)
Parameter (int): The entity's Y-position.
Return value: None.
Description: The method sets the entity's Y-position in the world.
Pre-condition: An object that inherits from entity has been created.
Post-condition: The Y-position has been set.
Called by: The World class collisionDetection()-method and the local update()-method.
Calls: None.
Implements: -

Method name: public int getY()
Parameter: None.
Return value (int): Returns the entity's Y-position in the world.
Description: The method returns the entity's Y-position.
Pre-condition: An object that inherits from entity has been created.
Post-condition: The Y-position has been returned.
Called by: The World class collisionDetection()-method.
Calls: None.
Implements: -

Method name: public void setVelocityX(int velocityX)
Parameter (int): The entity's speed in X-direction.
Return value: None.
Description: The method sets the entity's speed in X-direction.
Pre-condition: An object that inherits from entity has been created.
Post-condition: The X-velocity has been set.
Called by: The World class updateWorld()-method.
Calls: None.
Implements:
6.1.2.1 Sideways movement
6.1.2.2 Character jump
6.1.2.5 Screen movement
6.1.2.8 Screen speed

Method name: public int getVelocityX()
Parameter: None.
Return value (int): Returns the entity's speed in X-direction.
Description: The method returns the entity's speed in X-direction.
Pre-condition: An object that inherits from entity has been created.
Post-condition: The speed in X-direction has been returned.
Called by: The World class updateWorld()-method.
Calls: None.
Implements:
6.1.2.1 Sideways movement
6.1.2.2 Character jump
6.1.2.5 Screen movement
6.1.2.8 Screen speed

Method name: public void setVelocityY(int velocityY)
Parameter (int): The entity's speed in Y-direction.
Return value: None.
Description: The method sets the entity's speed in Y-direction.
Pre-condition: An object that inherits from entity has been created.
Post-condition: The Y-velocity has been set.
Called by: The World class updateWorld()-method.
Calls: None.
Implements:
6.1.2.1 Sideways movement
6.1.2.2 Character jump
6.1.2.5 Screen movement
6.1.2.8 Screen speed

Method name: public int getVelocityY()
Parameter: None.
Return value (int): Returns the entity's speed in Y-direction.
Description: The method returns the entity's speed in Y-direction.
Pre-condition: An object that inherits from entity has been created.
Post-condition: The speed in Y-direction has been returned.
Called by: The World class updateWorld()-method.
Calls: None.
Implements:
6.1.2.1 Sideways movement
6.1.2.2 Character jump
6.1.2.5 Screen movement
6.1.2.8 Screen speed

Method name: public void setHeight(int height)
Parameter (int): The entity's height.
Return value: None.
Description: The method sets the entity's height.
Pre-condition: An object that inherits from entity has been created.
Post-condition: The height has been set.
Called by: The Entity's class constructor.
Calls: None.
Implements: -

Method name: public int getHeight()
Parameter: None.
Return value (int): Returns the entity's height.
Description: The method returns the height.
Pre-condition: An object that inherits from entity has been created.
Post-condition: The height has been returned.
Called by: The World class collisionDetection()-method.
Calls: None.
Implements: -

Method name: public void setWidth(int width)
Parameter (int): The entity's width.
Return value: None.
Description: The method sets the entity's width.
Pre-condition: An object that inherits from entity has been created.
Post-condition: The width has been set.
Called by: The Entity's class constructor.
Calls: None.
Implements: -

Method name: public int getWidth()
Parameter: None.
Return value (int): Returns the entity's width.
Description: The method returns the width.
Pre-condition: An object that inherits from entity has been created.
Post-condition: The width has been returned.
Called by: The World class collisionDetection()-method.
Calls: None.
Implements: -

Method name: public void update()
Parameter: None.
Return value: None.
Description: The method converts the velocity variables to actual coordinates for each frame in the game and thereby moves the object in the world.
Pre-condition: An object that inherits from entity has been created.
Post-condition: The coordinates has been updated with respect to the velocity vectors.
Called by: The World class updateWorld()-method and the collisionDetection()-method.
Calls: setX(), setY(), getX(), getY(), setVelocityX(), setVelocityY(), getVelocityX() and getVelocityY()-methods.
Implements: -

5.5.4 Class Game

Fields

Field name: private World world;
Comment: The world object.

Field name: private Graphics graphics;
Comment: The graphics object.

Field name: private Audio audio;
Comment: The audio object.

Field name: private Menu currentMenu;
Comment: Keeps track of the current menu that's displayed to the screen.

Field name: private Menu startMenu;
Comment: The startMenu object.

Field name: private Menu optionsMenu;
Comment: The optionsMenu object.

Field name: private Menu difficultyMenu;
Comment: The difficultyMenu object.

Field name: private Menu tutorialMenu;
Comment: The tutorialMenu object.

Field name: private HighScore highScore;
Comment: The highScore object.

Field name: private Settings settings;
Comment: The settings object.

Field name: private boolean enter;
Comment: Keeps track if the enter key has been pressed.

Field name: private boolean escape;
Comment: Keeps track if the escape key has been pressed.

Field name: private boolean up;
Comment: Keeps track if the up key has been pressed.

Field name: private boolean down;
Comment: Keeps track if the down key has been pressed.

Field name: private boolean right;
Comment: Keeps track if the right key has been pressed.

Field name: private boolean left;
Comment: Keeps track if the left key has been pressed.

Field name: private boolean space;
Comment: Keeps track if the space key has been pressed.

Methods

Method name: public Game()
Parameter: None
Return value: None
Description: The constructor for the class. It initializes all objects and contains the game loop. In other words, the core of the game.
Pre-condition: The program has been started.
Post-condition: All vital objects has been initialized.
Called by: The main()-method.
Calls: initiateMenus()-method and object constructors.
Implements: -

Method name: public void initiateMenus()
Parameter: None
Return value: None
Description: The method sets up all menu items..
Pre-condition: The Game class has been created.
Post-condition: All menus has been set-up.
Called by: The Game constructor.
Calls: Set-up functions in menu.
Implements: -

Method name: public void actOnInput()
Parameter: None
Return value: None
Description: The method checks for input and acts accordingly.
Pre-condition: The gameLoop()-method is running.
Post-condition: Input has been checked.
Called by: The gameLoop()-method.
Calls: The Player class setVelocityX() and setVelocityy()-methods
Implements:
6.1.2.1 Sideways movement
6.1.2.2 Character jump
6.1.2.4 Use special item
6.1.3.2 Enter high score list

Method name: public void resetInput()
Parameter: None
Return value: None
Description: The method resets the input booleans.
Pre-condition: The gameLoop()-method is running.
Post-condition: Input has been reset.
Called by: The gameLoop()-method and the menuLoop()-method.
Calls: None
Implements:
6.1.2.1 Sideways movement
6.1.2.2 Character jump
6.1.2.4 Use special item
6.1.3.2 Enter high score list

Method name: public void keyPressed(KeyEvent arg0)
Parameter (KeyEvent): A key code for the pressed key
Return value: None
Description: The method is a part of the KeyListener-events.
Pre-condition: The Game class has been created.
Post-condition: Input has been confirmed.
Called by: Java underlying system.
Calls: None
Implements:
6.1.2.1 Sideways movement
6.1.2.2 Character jump
6.1.2.4 Use special item
6.1.3.2 Enter high score list

Method name: public void keyReleased(KeyEvent arg0)
Parameter (KeyEvent): A key code for the released key
Return value: None
Description: The method is a part of the KeyListener-events.
Pre-condition: The Game class has been created.
Post-condition: Input has been confirmed.
Called by: Java underlying system.
Calls: None
Implements:
6.1.2.1 Sideways movement
6.1.2.2 Character jump
6.1.2.4 Use special item
6.1.3.2 Enter high score list

Method name: public void keyTyped(KeyEvent arg0)
Parameter (KeyEvent): A key code for the typed key
Return value: None
Description: The method is a part of the KeyListener-events.
Pre-condition: The Game class has been created.
Post-condition: Input has been confirmed.
Called by: Java underlying system.
Calls: None
Implements:
6.1.2.1 Sideways movement
6.1.2.2 Character jump
6.1.2.4 Use special item
6.1.3.2 Enter high score list

Method name: public boolean isGameOver()
Parameter: None
Return value (boolean): Returns true if the game is over and vice versa.
Description: The method checks if the game is over, typically the player is dead or the user has pressed the escape key.
Pre-condition: The gameLoop() is running.
Post-condition: The state of the game has been checked.
Called by: The gameLoop()-method.
Calls: The Players isAlive()-method.
Implements: 6.1.1.2 Durance of play

Method name: public void menuLoop()
Parameter: None
Return value: None
Description: The method handles the menu system and constantly checks for input and calls the appropriete methods. It also calls the graphics draw()-method to draw the menu to the screen.
Pre-condition: The Game class has been created.
Post-condition: The Game has exited.
Called by: The main()-method.
Calls: Methods concerning the game play, like the gameLoop(), and menu functions.
Implements: 6.1.1.1 Tutorial text

Method name: public void gameLoop()
Parameter: None
Return value: None
Description: The method handles the in game functionality and constantly checks for input through the actOnInput()-method. It also make sure that the world is updated and that the game is drawn to the screen.
Pre-condition: The game is running.
Post-condition: The menu is running.
Called by: The menuLoop()-method.
Calls: actOnInput(), isGameOver(), resetInput(), world.updateWorld() and the graphics.draw()-methods.
Implements: -

Method name: public void main(String[] args)
Parameter (String[]): Unused input to the program
Return value: None
Description: The main()-method for the program. It creates an Game object.
Pre-condition: The program has been started.
Post-condition: The menuLoop() is running.
Called by: The Java subsystem.
Calls: menuLoop()-method
Implements: -

5.5.5 Class Graphics

Fields

Field name: private HashMap<String,Object> textures;
Comment: Keeps track of the texture used in game.

Field name: private Settings setting;
Comment: A reference back to the settings object.

Methods

Method name: public Graphics(Settings settings)
Parameter (Settings): None
Return value: None
Description: The constructor for the class. It initializes all OpenGL code.
Pre-condition: The Game class has been created.
Post-condition: All OpenGL systems has been initialized.
Called by: The Game class constructor.
Calls: None
Implements: -

Method name: public void initOpenGL()
Parameter: None
Return value: None
Description: Intitalizes OpenGL systems.
Pre-condition: The Graphics class has been created.
Post-condition: All OpenGL systems is ready for use.
Called by: The Graphics constructor.
Calls: None
Implements: -

Method name: public void loadTexture(String textureName)
Parameter (String): A path to the texture to be loaded.
Return value: None
Description: Loads the specified texture from disk into the HashMap.
Pre-condition: The Graphics class has been created.
Post-condition: The texture has been loaded into the HashMap.
Called by: The Graphics draw()-method.
Calls: None
Implements: -

Method name: public void drawMenu(Menu currentMenu)
Parameter (Menu): The menu to be drawn to the screen.
Return value: None
Description: The method draws the specified menu to the screen.
Pre-condition: The menuLoop() is running.
Post-condition: The menu has been drawn to the screen.
Called by: The menuLoop() in the Game class.
Calls: None
Implements: 6.1.1.1 Tutorial text

Method name: public void fadeOut(int time)
Parameter (int): The time it takes before the screen becomes completely black.
Return value: None
Description: The method fades all graphics to black background.
Pre-condition: The Graphics class has been created and the screen is visible.
Post-condition: The screen is black.
Called by: The menuLoop() in the Game class.
Calls: None
Implements: -

Method name: public void fadeIn(int time)
Parameter (int): The time it takes before the screen becomes completely visible.
Return value: None
Description: The method fades in all graphics from a black background.
Pre-condition: The Graphics class has been created and the screen is black.
Post-condition: The screen is visible.
Called by: The menuLoop() in the Game class.
Calls: None
Implements: -

Method name: public void draw(World world)
Parameter (World): A reference back to the world object.
Return value: None
Description: The method draws a frame of the world to the screen (player, items, blocks, background etc).
Pre-condition: The gameLoop() is running.
Post-condition: The world has been drawn to the screen.
Called by: The gameLoop() in the Game class.
Calls: drawWorld() and drawObjects()
Implements: -

Method name: public void drawObjects(LinkedList<Entity> entities)
Parameter (LinkedList): A reference to the list of entities placed in the world.
Return value: None
Description: The method draws a frame of the world objects to the screen.
Pre-condition: The gameLoop() is running.
Post-condition: The world objects has been drawn to the screen.
Called by: The draw()-method.
Calls: None
Implements: -

Method name: public void drawWorld(World world)
Parameter (World): A reference back to the world object.
Return value: None
Description: The method draws a frame of the world to the screen.
Pre-condition: The gameLoop() is running.
Post-condition: The world has been drawn to the screen.
Called by: The draw()-method.
Calls: None
Implements: -

Method name: public void rotateCamera(boolean left)
Parameter (boolean): Direction of the flip, left = true, right = false
Return value: None
Description: The method rotates the camera/viewport in the given direction.
Pre-condition: The gameLoop() is running.
Post-condition: The camera has been rotated.
Called by: The updateWorld()-method in the World class.
Calls: None
Implements: 6.1.2.12 Flip function

Method name: public boolean toggleFullscreen()
Parameter: None
Return value (boolean): Returns true if fullscreen is activated.
Description: The method switches fullscreen on or off.
Pre-condition: The menuLoop() is running.
Post-condition: The fullscreen mode has been changed.
Called by: The menuLoop().
Calls: setFullscreen() in the Settings class.
Implements: 6.1.1.3 Screen size

Method name: public boolean isFullscreen()
Parameter: None
Return value (boolean): Returns true if fullscreen is activated.
Description: The method returns the current state of the fullscreen value.
Pre-condition: The menuLoop() is running.
Post-condition: The fullscreen value has been returned.
Called by: The menuLoop().
Calls: getFullscreen in the Settings class.
Implements: 6.1.1.3 Screen size

5.5.6 Class Highscore

Fields

Field name: private LinkedList<ScoreData> score
Comment: Keeps track of the different high scores.

Methods

Method name: public HighScore(String title)
Parameter (String): Title name for the graphical interface.
Return value: None
Description: The constructor for the class. It initializes all components used for the high score list.
Pre-condition: The high score option has been selected.
Post-condition: All components has been created.
Called by: The Game class constructor.
Calls: None
Implements: -

Method name: public void save()
Parameter: None
Return value: None
Description: Saves the current high score list down to a locally stored file.
Pre-condition: The high score list is loaded into memory.
Post-condition: The high score list is stored to file.
Called by: Entered new highscore-event.
Calls: None
Implements:
6.1.3.1 Storing of high score
6.1.3.2 Enter high score list
6.1.3.4 Reset high score

Method name: public void load()
Parameter: None
Return value: None
Description: Loads the locally stored list to memory.
Pre-condition: The high score list is in a readable file.
Post-condition: The high score list is loaded.
Called by: The Game constructor.
Calls: None
Implements: 6.1.3.3 Check high score

Method name: public boolean isHighscore()
Parameter: None
Return value (boolean): Always returns true
Description: An identifier method for the high score list.
Pre-condition: The high score object has been created.
Post-condition: The value true has been returned.
Called by: The Graphics drawMenu()-method.
Calls: None
Implements: 6.1.3.3 Check high score

5.5.7 Class Item

Fields

None.

Methods

Method name: public Item(int x, int y, int height, int width)
Parameter (int): The items initial x value.
Parameter (int): The items initial y value.
Parameter (int): The items height value.
Parameter (int): The items width value.
Return value: None
Description: The constructor for the class. It makes sure that the initial values for the item is set corrected, like dimension, texture and position in the world.
Pre-condition: The game world has been created.
Post-condition: A new item has been created in the world.
Called by: The World class constructor.
Calls: Calls the setTexture()-method in the superclass.
Implements: -

5.5.8 Class Menu

Fields

Field name: private ArrayList<MenuData> optionList
Comment: Keeps track of the different selectable menu options.

Field name: private String title
Comment: The name of the menu screen.

Field name: private int currentSelection
Comment: Keeps track of the current selected menu alternative.

Methods

Method name: public Menu(String title)
Parameter (String): Title name for the graphical interface.
Return value: None
Description: The constructor for the class. It initializes all components used for the menu list.
Pre-condition: The Game class has been created.
Post-condition: All components for the menu has been created.
Called by: The Game class constructor.
Calls: None
Implements: -

Method name: public void addComponent(String name, Menu menuLink)
Parameter (String): The text to be displayed for the component in the GUI.
Parameter (Menu): Link to a new menu window.
Return value: None
Description: Adds a new selectable menu component that links to a new menu window.
Pre-condition: The menu object has been created.
Post-condition: A new component has been added to the menu window.
Called by: initiateMenus() in the Game class.
Calls: None
Implements: -

Method name: public void addComponent(String name, int funcIndex)
Parameter (String): The text to be displayed for the component in the GUI.
Parameter (int): A func index so the propper function gets called when selecting this alternative.
Return value: None
Description: Adds a new selectable menu component that doesn't link to a new menu window.
Pre-condition: The menu object has been created.
Post-condition: A new component has been added to the menu window.
Called by: initiateMenus() in the Game class.
Calls: None
Implements: -

Method name: public void changeComponent(String name, String newName)
Parameter (String): The name for the component to change (GUI name).
Parameter (String): The new name for the component (GUI name).
Return value: None
Description: Changes an already defined component in the menu.
Pre-condition: The menu object has been created.
Post-condition: The component has changed its name.
Called by: menuLoop() in the Game class.
Calls: None
Implements: -

Method name: public void changeCurrentComponent(String newName)
Parameter (String): The new name for the component (GUI name).
Return value: None
Description: Changes the currently selected component in the menu.
Pre-condition: The menu object has been created and a component has been selected.
Post-condition: The component has changed its name.
Called by: menuLoop() in the Game class.
Calls: None
Implements: -

Method name: public void removeComponent(String name)
Parameter (String): The name for the component to be removed (GUI name).
Return value: None
Description: Removes the specified component in the menu.
Pre-condition: The menu object has been created.
Post-condition: The component has been removed.
Called by: menuLoop() in the Game class.
Calls: None
Implements: -

Method name: public ArrayList<MenuData> getComponentList()
Parameter: None
Return value (ArrayList): The list of components in this menu object.
Description: Returns the component list containing all objects for this menu.
Pre-condition: The menu object has been created.
Post-condition: The component list has been returned.
Called by: drawMenu() in the Graphics class.
Calls: None
Implements: -

Method name: public int getAction(boolean enter, boolean space, boolean escape, boolean up, boolean down)
Parameter (all booleans): Is true if a key has been pressed.
Return value (int): Returns the funcIndex for the currently selected menu option.
Description: Returns an index so that the caller can call the right method when a menu option has been selected.
Pre-condition: The menu object has been created.
Post-condition: The funcIndex has been returned.
Called by: menuLoop() in the Game class.
Calls: None
Implements: -

Method name: public Menu getNewMenu()
Parameter: None
Return value (Menu): Returns a reference to a new menu window.
Description: If the currently selected menu option links to a new menu window, this method will return the new menu window. The function should only be called when that's true.
Pre-condition: The menu object has been created and the currently selected menu option links to a new menu window.
Post-condition: The reference to the new menu window has been returned..
Called by: menuLoop() in the Game class.
Calls: None
Implements: -

Method name: public boolean isHighscore()
Parameter: None
Return value (boolean): Always returns false
Description: An identifier method for the high score list.
Pre-condition: The menu object has been created.
Post-condition: The value false has been returned.
Called by: The Graphics drawMenu()-method.
Calls: None
Implements: -

5.5.9 Class Player

Fields

Field name: private String t_facingLeft
Comment: The path to the texture for the character when facing left.

Field name: private String t_facingRight
Comment: The path to the texture for the character when facing right.

Field name: private String t_jumpLeft
Comment: The path to the texture for the character when jumping left.

Field name: private String t_jumpRight
Comment: The path to the texture for the character when jumping right.

Field name: private int collectedItems
Comment: The amount of items the user currently have.

Field name: private int score
Comment: The users current score.

Field name: private boolean isAlive
Comment: True if the user is still alive (in the game).

Methods

Method name: public Player(int x, int y, int height, int width)
Parameter (int): The players initial x value.
Parameter (int): The players initial y value.
Parameter (int): The players height value.
Parameter (int): The players width value.
Return value: None
Description: The constructor for the class. It makes sure that the initial values for the player is set corrected, like dimension, texture and position in the world.
Pre-condition: The game world has been created.
Post-condition: A new item has been created in the world.
Called by: The World class constructor.
Calls: Calls the setTexture()-method in the superclass.
Implements: -

Method name: public String getTexture()
Parameter: None
Return value(String): Returns the path to the current picture.
Description: Overloads Entitys getTexture() to return the picture that represents the current state of the player.
Pre-condition: The game been initialized.
Post-condition: The texture path has been returned.
Called by: The Graphics class draw()-methods.
Calls: None.
Implements: -

Method name: public void addScore(int score)
Parameter (int): The score to be added.
Return value: None.
Description: Adds the specified score to the users score.
Pre-condition: The game been initialized.
Post-condition: The score has been updated.
Called by: The World class collisionDetection() method.
Calls: None.
Implements:
6.1.2.10 Receiving points
6.1.2.11 Showing high score

Method name: public int getScore()
Parameter: None
Return value(int): Returns the current score.
Description: Returns the current score for the user.
Pre-condition: The game been initialized.
Post-condition: The score has been returned.
Called by: The Graphics class drawWorld() method.
Calls: None.
Implements: 6.1.2.11 Showing high score

Method name: public boolean hasMaxItems()
Parameter: None
Return value(boolean): Is true if the user has filled up his/hers items stock.
Description: Checks if the user has filled up his/hers item stock.
Pre-condition: The game been initialized.
Post-condition: Has returned true or false.
Called by: The World class collisionDetection() method.
Calls: None.
Implements: 6.1.2.3 Collect special item

Method name: public void addItem()
Parameter: None
Return value: None.
Description: Adds one more item to the users stock of items.
Pre-condition: The game been initialized.
Post-condition: The new item has been added.
Called by: The World class collisionDetection() method.
Calls: None.
Implements: 6.1.2.3 Collect special item

Method name: public void removeItem()
Parameter: None
Return value: None.
Description: Removes one more item to the users stock of items.
Pre-condition: The game been initialized.
Post-condition: One item has been removed.
Called by: The World class collisionDetection() method.
Calls: None.
Implements: 6.1.2.4 Use special item

Method name: public int getCollectedItems()
Parameter: None
Return value(int): Returns the amount of items the user has in stock.
Description: Returns the amount of items the has in stock.
Pre-condition: The game been initialized.
Post-condition: The amount of items has been returned.
Called by: The Graphics class drawWorld() method.
Calls: None.
Implements: -

Method name: public boolean isAlive()
Parameter: None
Return value(boolean): Returns true if the character is alive.
Description: Returns true if the character is alive.
Pre-condition: The game been initialized.
Post-condition: The caller knows if the character is alive.
Called by: The Game class isGameOver() method.
Calls: None.
Implements: 6.1.1.2 Durance of play

Method name: public void setAlive(boolean isAlive)
Parameter (boolean): Is true if the player should be alive.
Return value: None.
Description: Can change the state of the character (if the character is dead or alive).
Pre-condition: The game been initialized.
Post-condition: The users alive state has been changed.
Called by: The World class collisionDetection() method.
Calls: None.
Implements: 6.1.1.2 Durance of play

5.5.10 Class Settings

Fields

Field name: private boolean fullscreen
Comment: Is true if the game is in fullscreen mode.

Field name: private boolean mute
Comment: Is true if the game is in mute mode.

Methods

Method name: public Settings()
Parameter: None
Return value: None.
Description: The constructor of Settings class. Initiates the settings.
Pre-condition: The game been initialized.
Post-condition: The settings has been initialized.
Called by: The Game class constructor.
Calls: None.
Implements: -

Method name: public void load()
Parameter: None
Return value: None.
Description: Loads settings from a file.
Pre-condition: The game been initialized.
Post-condition: The settings has been loaded from file.
Called by: The Game class constructor.
Calls: None.
Implements: -

Method name: public void save()
Parameter: None
Return value: None.
Description: Saves settings to a file.
Pre-condition: The game been initialized.
Post-condition: The settings has been saved to a file.
Called by: The Settings class setFullscreen(boolean fullscreen) and setMute(boolean mute).
Calls: None.
Implements: -

Method name: public boolean getFullscreen()
Parameter: None
Return value(boolean): Returns true if the game is in fullscreen mode.
Description: Checks if the game is in fullscreen mode.
Pre-condition: The game been initialized.
Post-condition: The calling method knows if the game is in fullscreen.
Called by: Graphic class draw methods.
Calls: None.
Implements: 6.1.1.3 Screen size

Method name: public void setFullscreen(boolean fullscreen)
Parameter (boolean): The value to set the fullscreen to, false for window mode.
Return value: None.
Description: Sets the fullscreen variable to the specified value.
Pre-condition: The game been initialized.
Post-condition: The fullscreen value has been changed to the new value.
Called by: menuLoop() in the Game class.
Calls: None.
Implements: 6.1.1.3 Screen size

Method name: public boolean getMute()
Parameter: None
Return value (boolean): Returns the value of the mute settings..
Description: The method returns the value of the mute settings.
Pre-condition: The game been initialized.
Post-condition: The mute value has been returned.
Called by: menuLoop() in the Game class.
Calls: None.
Implements: 6.1.1.4 Appearance of sound

Method name: public void setMute(boolean mute)
Parameter (boolean): The value to set the mute to, true for mute.
Return value: None.
Description: Sets the mute variable to the specified value.
Pre-condition: The game been initialized.
Post-condition: The mute value has been changed to the new value.
Called by: menuLoop() in the Game class.
Calls: None.
Implements: 6.1.1.4 Appearance of sound

5.5.11 Class World

Fields

Field name: private Player player
Comment: The player object.

Field name: private LinkedList<Entity> entityList
Comment: Contains all entity's in the world, both blocks, items and the player.

Field name: private LinkedList<Block> blockList
Comment: Contains all blocks in the world.

Field name: private LinkedList<Item> itemList
Comment: Contains all items in the world.

Field name: private String t_collectedItem
Comment: Contains a path to the corresponding texture.

Field name: private String t_itemPlaceHolder
Comment: Contains a path to the corresponding texture.

Field name: private String t_rotateLeft
Comment: Contains a path to the corresponding texture.

Field name: private String t_rotateRight
Comment: Contains a path to the corresponding texture.

Field name: private String t_speedup
Comment: Contains a path to the corresponding texture.

Field name: private String t_gameover
Comment: Contains a path to the corresponding texture.

Field name: private OpenALWave s_speedup
Comment: Contains a path to the corresponding audio file.

Field name: private OpenALWave s_highScore;
Comment: Contains a path to the corresponding audio file.

Field name: private OpenALWave s_gameOver;
Comment: Contains a path to the corresponding audio file.

Field name: private int difficultylevel
Comment: Keeps track of the current difficulty level.

Field name: private int gravity
Comment: The constant gravity for the game.

Methods

Method name: public World(Graphics graphics)
Parameter (Graphics): A reference to the graphic object.
Return value: None
Description: The constructor for the class. It initializes all world objects and the player.
Pre-condition: The Game class has been created.
Post-condition: All world objects has been initialized and set-up.
Called by: The Game class constructor.
Calls: None
Implements: -

Method name: public LinkedList<Entity> getEntityList()
Parameter: None
Return value (LinkedList): Returns the entity list.
Description: Returns the list of entity's placed in the world.
Pre-condition: The world object has been initialized.
Post-condition: The entity list has been returned.
Called by: drawObjects() in the Graphics class.
Calls: None
Implements: -

Method name: public Player getPlayer()
Parameter: None
Return value (Player): Returns a reference to the player object.
Description: Returns the reference to the player.
Pre-condition: The player object has been created.
Post-condition: The player reference has been returned.
Called by: actOnInput() and isGameOver() in the Game class.
Calls: None
Implements: -

Method name: public void updateWorld()
Parameter: None
Return value: None
Description: Updates all the worlds objects, like positions, speed etc in the world. It also keeps track of when to increase the difficulty level.
Pre-condition: The gameLoop()-method is running,
Post-condition: The objects has been updated in the world.
Called by: gameLoop()-method in the Game class.
Calls: update()-method for each object in the world, collisionDetection()-method, rotateCamera(), speedUp and createNewItem()-methods.
Implements:
6.1.2.5 Screen movement
6.1.2.7 Difficulty level
6.1.2.10 Receiving points
6.1.2.12 Flip function
6.1.2.13 Flip indication
6.1.2.14 Mirroring sideways movement

Method name: public void collisionDetection()
Parameter: None
Return value: None
Description: Check each object against the player object to handle collisions in the game. It also make sure that items are collected if they collide with the player.
Pre-condition: The gameLoop()-method is running,
Post-condition: The objects has been updated in the world.
Called by: updateWorld()-method.
Calls: Each objects move methods (setX, setY, getX, getY).
Implements: 6.1.2.6 Ability to jump through blocks

Method name: public void blockGenerator(Block block)
Parameter (Block): The block to be moved on the screen to a new position.
Return value: None
Description: The method moves used blocks to a new position above the screen for reuse.
Pre-condition: The gameLoop()-method is running,
Post-condition: The objects has been updated in the world.
Called by: updateWorld()-method.
Calls: Each objects move methods (setX, setY, getX, getY).
Implements: -

Method name: public void removeItem()
Parameter: None
Return value: None
Description: The method removes an item from the world, or typically, the item and entity list.
Pre-condition: The gameLoop()-method is running,
Post-condition: The objects has been removed from the world.
Called by: collisionDetection()-method.
Calls: None.
Implements: 6.1.2.3 Collect special item

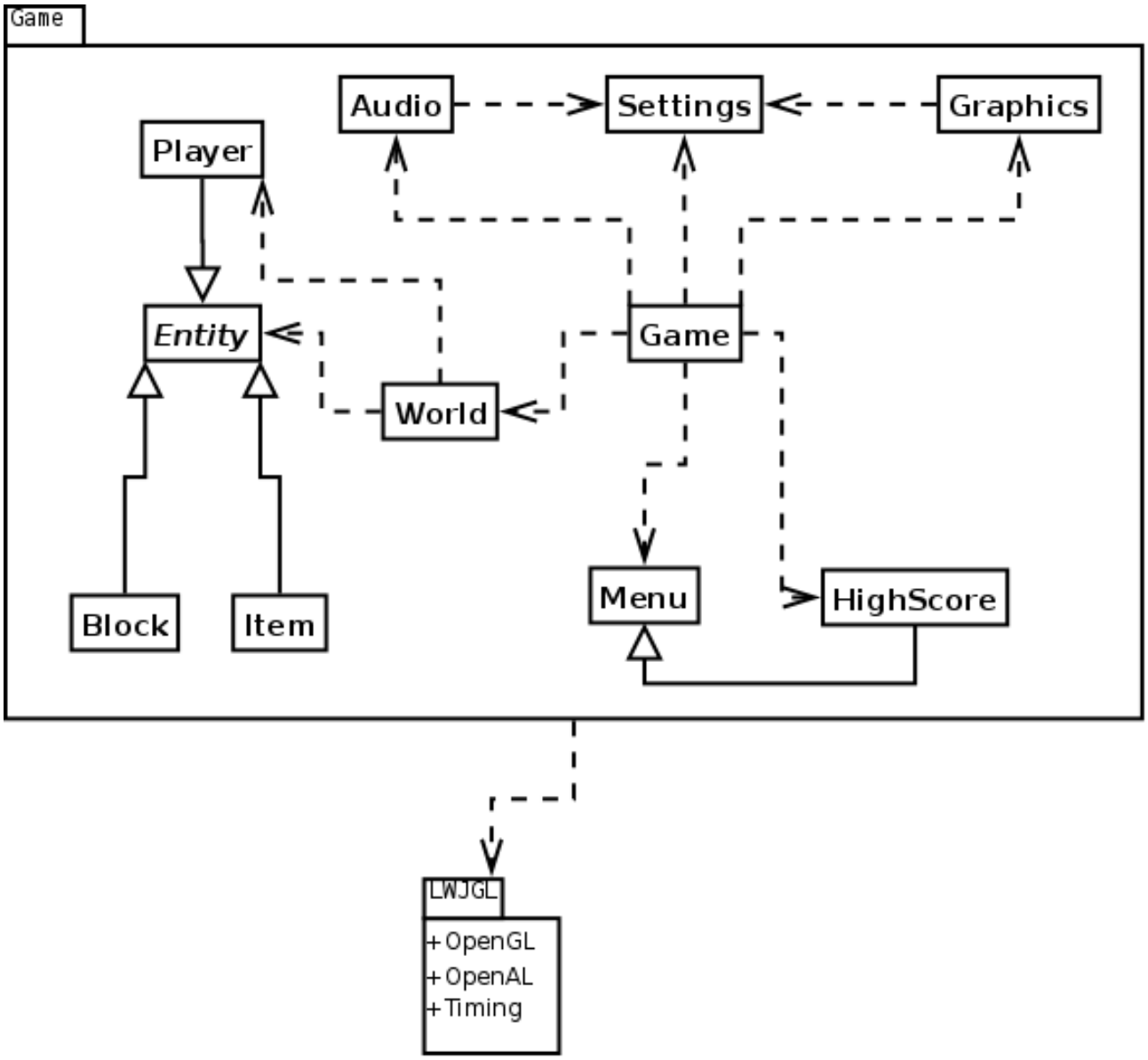
Method name: public void createNewItem()
Parameter: None
Return value: None
Description: The method creates a new item in the world at a random position.
Pre-condition: The gameLoop()-method is running and it's time for a new item to be created.
Post-condition: The objects has been created and positioned in the world.
Called by: updateWorld()-method.
Calls: None.
Implements: -

Method name: public void speedup()
Parameter: None
Return value: None
Description: The method increases the motion speed of the world screen.
Pre-condition: The gameLoop()-method is running and it's time for a speedup.
Post-condition: The screen has speeded up.
Called by: updateWorld()-method.
Calls: None.
Implements:
6.1.2.8 Screen speed
6.1.2.9 Screen speed increase indication

5.5.12 Cross referencing index

	getVelocityX()	page 38
	setVelocityY()	page 39
	getVelocityY()	page 39
	updateWorld()	page 63
6.1.2.6 Ability to jump through blocks	collisionDetection()	page 64
6.1.2.7 Difficulty level	updateWorld()	page 63
6.1.2.8 Screen speed	setVelocityX()	page 38
	getVelocityX()	page 38
	setVelocityY()	page 39
	getVelocityY()	page 39
	speedup()	page 65
6.1.2.9 Screen speed increase indication	speedup()	page 65
6.1.2.10 Receiving points	addScore()	page 57
	updateWorld()	page 63
6.1.2.11 Showing high score	addScore()	page 57
	getScore()	page 57
6.1.2.12 Flip function	rotateCamera()	page 48
	updateWorld()	page 63
6.1.2.13 Flip indication	updateWorld()	page 63
6.1.2.14 Mirroring sideways movement	updateWorld()	page 63
6.1.3.1 Storing of high score	save()	page 50
6.1.3.2 Enter high score list	actOnInput()	page 42
	resetInput()	page 43
	keyPressed()	page 43
	keyReleased()	page 43
	keyTyped()	page 44
	save()	page 50
6.1.3.3 Check high score	load()	page 50
	isHighscore()	page 51
6.1.3.4 Reset high score	save()	page 50
6.1.3.5 high score sound	play()	page 33
	stopCurrent()	page 33

5.6 Package Diagram



This is an UML diagram representing our package hierarchy.

6 Functional test cases

6.1 General

6.1.1 Tutorial text

Function: Display a tutorial text.
Description: Displays a tutorial text for the user to learn how to play the game and to make the user develop his or her skills.
Pre-condition: The user has chosen a difficulty level.
Input: None.
Expected output: The tutorial text is shown.
Output destination: The game screen.
Testing procedure:

- Check that the tutorial text is shown.

Reference: *Requirements Document 6.1.1.1 Tutorial text*

6.1.2 Durance of play

Function: Play time.
Description: Allow the user to play until the character falls to the bottom of the screen.
Pre-condition: The user has started a game session.
Input: None.
Expected output: Game session ends.
Output destination: -
Testing procedure:

- Check that the game session doesn't end until the character falls to the bottom of the screen.

Reference: *Requirements Document 6.1.1.2 Durance of play*

6.1.3 Screen size

Function: Adjusting the screen size.
Description: Changes the size of the screen to oblige the user preferences.
Pre-condition: The user has chosen the Options menu option.
Input: Chosen screen size.
Expected output: Change in screen size.
Output destination: The game screen.
Testing procedure:

- Toggle the fullscreen option on/off.
- Check that the game goes to/from fullscreen.

Reference: *Requirements Document 6.1.1.3 Screen size*

6.1.4 Appearance of sound

Function: Option of sound.
Description: Changes the appearance of sound in the game, muted or full sound.
Pre-condition: The user has chosen the Options menu option.
Input: Chosen sound option.
Expected output: Sound is played or not played.
Output destination: Computer speakers.
Testing procedure:

- Toggle the fullscreen option on/off.
- Check that the game goes to/from fullscreen.

Reference: *Requirements Document 6.1.1.4 Appearance of sound*

6.2 Game

6.2.1 Sideways movement

Function: Move sideways.
Description: Moves the character sideways as the user commands.
Pre-condition: The user has started a game session.
Input: User input.
Expected output: The character moves sideways.
Output destination: The game screen.
Testing procedure:

- Move sideways with the character.
- Check that the character moves in the specified direction.

Reference: *Requirements Document 6.1.2.1 Sideways movement*

6.2.2 Character jump

Function: Character jumping.
Description: The character jumps as the user commands.
Pre-condition: The user has started a game session.
Input: User input.
Expected output: The character jumps.
Output destination: The game screen.
Testing procedure:

- Press the jump button.
- Check that the character jumps.

Reference: *Requirements Document 6.1.2.2 Character jump*

6.2.3 Collect special item

Function: Pick up item.
Description: The character picks up a special item.
Pre-conditions: The user has started a game session. The character have jumped up until a special item appears on the screen. The character has at least one free item slot.
Input: User input.
Expected output: Item disappears and reappears in the upper left corner where all the characters special items are shown.
Output destination: The game screen.
Testing procedure:

- Jump up to the item and touch it with the character.
- Check that the item appears in the upper left corner and that it disappears from the screen.

Reference: *Requirements Document 6.1.2.3 Collect special item*

6.2.4 Use special item

Function: Use item.
Description: The character uses a special item and gets a special jump.
Pre-condition: The user has started a game session. The character has jumped up in the are and has at least on special item.
Input: User input.
Expected output: The character makes a special jump and one item is removed from the inventory.
Output destination: The game screen.
Testing procedure:

- Jump one time and then jump once again before touching the ground.
- Check that a special jump is performed.

Reference: *Requirements Document 6.1.2.4 Use special item*

6.2.5 Screen movement

Function: Move the screen.
Description: Moves the screen in a certain speed upwards, forcing the character to move upwards.
Pre-condition: The user has started a game session. The character jumps.
Input: None.
Expected output: The screen is moving.
Output destination: The game screen.
Testing procedure:

- Check that the screen starts to move upwards.

Reference: *Requirements Document 6.1.2.5 Screen movement*

6.2.6 Ability to jump through blocks

Function: Jump through blocks.
Description: The character jumps through blocks on it's way up.
Pre-condition: Choose Start game
Input: None.
Expected output: None.
Output destination: The game screen.
Testing procedure:

- Jump towards a block from any side but from above.
- Check that the character passes through the block.

Reference: *Requirements Document 6.1.2.6 Ability to jump through blocks*

6.2.7 Difficulty level

Function: Choose difficulty level.
Description: The user selects difficulty level for the game.
Pre-condition: Choose Start game
Input: Chosen difficulty level.
Expected output: The corresponding screen speed to the selected difficulty level.
Output destination: The game screen.
Testing procedure:

- Check that for each higher difficulty level the screen speed increases.

Reference: *Requirements Document 6.1.2.7 Difficulty level*

6.2.8 Screen speed

Function: Speed increment.
Description: The screen speed increases to higher rate.
Pre-condition: The game has started. Jump upwards until a screen speed indication appears.
Input: Current speed level.
Expected output: New speed level.
Output destination: The game screen.
Testing procedure:

- Check that the speed increases after the indication of a speed increase.

Reference: *Requirements Document 6.1.2.8 Screen speed*

6.2.9 Screen speed increase indication

Function: Indication of speed change.
Description: Indicates the user that the screen speed has changed to a higher rate.
Pre-condition: The game has started. Jump upwards.
Input: Speed increment.
Expected output: Visual and audible indication.
Output destination: The game screen and computer speakers.
Testing procedure:

- Check that a screen speed indication appears (visual and audible) after a specified amount of time.

Reference: *Requirements Document 6.1.2.9 Screen speed increase indication*

6.2.10 Receiving points

Function: Receive points.
Description: The user receives points for every block passed.
Pre-condition: The game has started. Jump upwards.
Input: User input.
Expected output: The application.
Output destination: The game screen.
Testing procedure:

- Check that when the character passes a block the user receives corresponding points for the block.

Reference: *Requirements Document 6.1.2.10 Receiving points*

6.2.11 Showing score

Function: Show current score.
Description: Displays the user's current score in-game.
Pre-condition: The game has started.
Input: None.
Expected output: A presented score.
Output destination: The game screen.
Testing procedure:

- Check that there are a in-game score counter that corresponds to the current score of the user.

Reference: *Requirements Document 6.1.2.11 Showing score*

6.2.12 Flip function

Function: Flip the screen.
Description: The screen flips (rotates) in a certain direction.
Pre-condition: A new game has been started and the user is currently playing.
Input: None.
Expected output: Rotated screen.
Output destination: The computer screen.
Testing procedure:

- Play the game until the screen flips around.
- Check if the screen rotated 90 degrees in any direction.

Reference: *Requirements Document 6.1.2.12 Flip function*

6.2.13 Flip indication

Function: Indication of rotate direction.
Description: Indicates the flip direction that the screen will rotate too.
Pre-condition: A flip rotation will occur.
Input: Flip rotation.
Expected output: Flip indication.
Output destination: The computer screen.
Testing procedure:

- Play the game until the screen flips around.
- Notice if there was a flip indication before the rotation.

Reference: *Requirements Document 6.1.2.13 Flip indication*

6.2.14 Mirroring sideways movement

Function: Mirror movement.
Description: Moves the character from one side to the other when crossing the side of the screen.
Pre-condition: A new game has started and the user is currently playing.
Input: User input.
Expected output: A mirror movement.
Output destination: The computer screen.
Testing procedure:

- Walk or jump towards the side of the screen.
- Notice if the player appears on the other side.

Reference: *Requirements Document 6.1.2.14 Mirroring sideways movement*

6.3 High score

6.3.1 Storing of high score

Function: Store the high score list.
Description: Stores the high score list to the local computers hard drive.
Pre-condition: High score has been beaten.
Input: User input.
Expected output: Store high score list.
Output destination: The computer hard drive.
Testing procedure:

- Enter your name and confirm it.
- Restart the program.
- Check if the name is still in the high score list.

Reference: *Requirements Document 6.1.3.1 Storing of high score*

6.3.2 Enter high score list

Function: Enter the high score.
Description: Checks the high score list and lets the user enter a name if a score was beaten.
Pre-condition: A game has ended.
Input: A high score list.
Expected output: User name.
Output destination: The computer screen.
Testing procedure:

- Enter your name.
- Confirm it with the enter button.
- Check to see if the name was stored in the high score list.

Reference: *Requirements Document 6.1.3.2 Enter high score list*

6.3.3 Check high score

Function: Check the high score list.
Description: Presents the current locally stored high score list to the user.
Pre-condition: The system has started and the user is in the menu.
Input: High score list.
Expected output: The reset high score list.
Output destination: The current high score list.
Testing procedure:

- Select the high score option.
- Press the enter button.
- Check the high score list.

Reference: *Requirements Document 6.1.3.3 Check high score*

6.3.4 Reset high score

Function: Reset high score list.
Description: Resets all the scores on the high score list.
Pre-condition: The system has started and the user is in the menu.
Input: High score list.
Expected output: The reset high score list.
Output destination: The computer screen.
Testing procedure:

- Select the high score option.
- Press the enter button.
- Select the reset high score button.
- Press the enter button.

Reference: *Requirements Document 6.1.3.4 Reset high score*

6.3.5 High score sound

Function: Play high score sound.
Description: Plays a sound effect when a previous score on the high score list was beaten.
Pre-condition: A game has ended and a score on the high score list has been beaten.
Input: High score list.
Expected output: Special sound effect.
Output destination: The computer speakers.
Testing procedure:

- Check the speakers for sound.

Reference: *Requirements Document 6.1.3.5 High score sound*

6.4 Non-functional requirements

6.4.1 Usability requirements

6.4.1.1 Tutorial text

Function: Learn the game in one minute.
Description: The tutorial text shall provide the user with the knowledge of how to play the game. The text shall be readable in one minute.
Pre-condition: The tutorial text is loaded on the screen.
Input: A user.
Expected output: The user has learned how to play the game.
Output destination: The user.
Testing procedure:

- Read the tutorial text during a minute.

Reference: *Requirements Document 6.2.1.1 Tutorial text.*

6.4.1.2 Easy to install

Function: Install the game with ease.
Description: The game installed with minimum effort. It shall only require an installation command.
Pre-condition: The installation file is reachable.
Input: Installation command.
Expected output: The game is installed with only an installation command.
Output destination: The computer.
Testing procedure:

- Locate installation executable.
- Start the installation with the OS specific installation command.

Reference: *Requirements Document 6.2.1.2 Easy to install.*

6.4.2 Performance requirement

6.4.2.1 6.4.2.1. Low start up time

Function: Start a game with limited time required.
Description: The game starts in five seconds after the user has viewed the tutorial text and pressed the start game button.
Pre-condition: The tutorial text is shown on the screen.
Input: Enter/Select command to press the start game button.
Expected output: The game starts within five seconds.
Output destination: The computer.
Testing procedure:

- Select the start button.
- Press the button.
- Count the amount of time for the game to start. One Mississippi, two Mississippi...

Reference: *Requirements Document 6.2.2.1 Low start up time*

6.4.3 Space requirement

6.4.3.1 Game size

Function: Install the game with limited space requirement.
Description: The system size is at most 20MB, so the textures and audio files needs to be small.
Pre-condition: The installation file is reachable and you have at least 20MB of usable space.
Input: Installation command.
Expected output: The program has been installed, using less than or equal to 20MB of space.
Output destination: The computer.
Testing procedure:

- Locate installation executable.
- Start the installation with the OS specific installation command.
- Check the used space for the program.

Reference: *Requirements Document 6.2.3.1 Game size*

6.4.4 Portability requirement

6.4.4.1 Multi platform playability

Function: Play on different platforms.
Description: The system works on a set of different platforms supporting the system requirements, at least OSX, Windows and Linux.
Pre-condition: The program is located on the users computer.
Input: Start command.
Expected output: The program has started.
Output destination: The computer.
Testing procedure:

- Locate the game.
- Find the executable.
- Start the executable with the OS specific start command.

Reference: *Requirements Document 6.2.4.1 Multi platform playability.*

7 Glossary

2D
Two dimensions.

3D
Three dimensions.

API
Application Programming Interface, an interface that other applications can access to be able to use functions from the software that provides the API.

Functional requirement
This is a feature or function that should be in the system.

Interface
This is the connection into an application or similar to access and control, for example; a graphical user interface has buttons and menus as the interface while a command line interface has text commands as the interface.

Java
Object oriented programming language that compiles and runs in the Java Virtual Machine, also known as the Java Runtime Environment, JRE. It's not open source yet, but is in the process of making it open source. The new version 7 will be completely free and open source under the GNU General Public License Version 2 (GPLv2).

Linux
A free, open-source and UNIX-like operating system.

Linux distribution
A Linux distribution is comprised of a Linux kernel, utilities from the GNU project and a lot of packages (applications). All these packages are normally managed by a packet managing system.

LWJGL
LightWeight Java Game Library, is a free and open-source library that provides an API for developers to use. This contains game, graphics and sound functionality that helps developers to faster and easier develop games in Java. The OpenGL and OpenAL libraries are included in this package.

Mac OS
A UNIX-like operating system from Apple.

Non-functional requirement
Expected behaviour or constraint on the system.

OpenAL
Open Audio Library is a free and open-source library for audio output.

OpenGL
Open Graphics Library is a free and open-source library for graphics output.

Open-source
Software which have its source code available for everyone and which everyone can download and change. Depending on the licence, the source code can be distributed.

Operating system
Software that controls the hardware and provides a platform, API, for applications to run on.

Terminal
It's a text-based command line interface where the user can type in commands to do different things like starting a program, in contrast to a graphical user interface where you click on the desired program.

Texture
An image applied to a surface of an object.

Ubuntu Linux
A Linux distribution from Canonical.

UNIX
An operating system from 1969 which was widely used because of its multi-user and multiprogramming abilities.

Windows
A operating system from Microsoft.