Bilkent University

Department of Computer Engineering

Object Oriented Software Engineering Project

Academic Warfare : The Conflict in Bilkent

Design Report

Doğukan Yiğit Polat, Selin Fildiş, Yasin Erdoğdu, Onur Elbirlik

Instructor: Uğur Doğrusöz

Design Report

Nov 12, 2016

Table of Contents

INTRODUCTION **4**

Purpose of the System 4

Design Goals 4

PROPOSED SOFTWARE ARCHITECTURE **4**

Overview 4

Subsystem Decomposition 4

Hardware and Software Considerations 5

Peripheral Modules Management 5

SUBSYSTEM SERVICES **5**

Game Logic Design 5

Overview 5

Game-Time Procedures 6

Example Scenarios 7

Detailed Class Design 8

GUI Design 16

Overview 16

GUI Components 16

The Main Menu 16

The Load Game 17

Options 17

HighScores 18

Tutorial 18

In game 19

GUI Class Design 19

GUI Activity Flow 19

Sound Design 20

SoundFX 20

Music 20

1. INTRODUCTION

1.1. Purpose of the System

The purpose of our system for the users to have fun by playing our game, Academic Warfare: The Conflict in Bilkent which is a tower defense game. Tower defense games function with waves of enemies coming to a base which the user has to defend. In every wave the enemies become harder to defeat therefore the user has to develop strategies in order to pass through the waves. If the base takes too much damage the game will end.

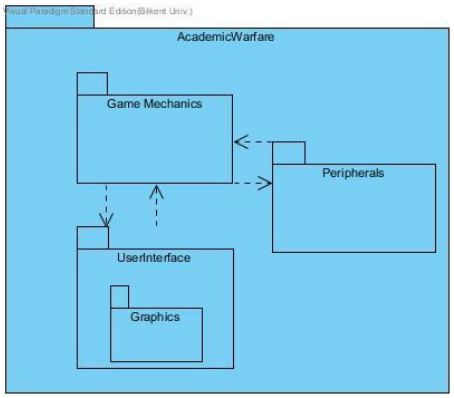
1.2. Design Goals

Our design goal is to develop a smooth, scalable, conveniently-designed and well-engineered computer game that serves fun to its players. We tend to design every part of the system in a complete object-oriented manner. Furthermore, we want to keep every subsystem of the project simple, understandable and easy to manage. All of the building blocks and modules should be modified and upgraded without any problems.

2. PROPOSED SOFTWARE ARCHITECTURE

2.1. Overview

In this section, the software architecture system of our game project has been decomposed into subsystems to provide maintenance. The essential goal of this decomposition is to reduce the redundancy among subsystems and improve interaction between components of subsystems. The other main goal is to provide convenience and clarity for designing the game.



2.2. Subsystem Decomposition

The system decomposition is shown on the right. Our decomposed system consists of AcademicWarfare our main packet with three sub-packets GameMechanics, UserInterface, Peripherals respectively. The UserInterface packet also has a subpacket of itself which is the Graphics packet.

The classes of the UserInterface packet are built to provide an easily understandable, simple design. The other packets are independent from the UserInterface class and purposes to enable functionalities of the game. T

The UserInterface Package the components of the GUI will be implemented. This and it's subpackage graphics will contain the details of the functionality of the GUI and the screens for the GUI as well. This package will communicate with the GameMechanics package to provide a UI to the game. The GameMechanics package will contain the game mechanics such as the game engine. This package will also be in touch with the peripherals class.

2.3. Hardware and Software Considerations

We will use JAVA programming language as for implementing the main design of our game project. JDK and J2SE platforms will be used to develop the program and required JRE package will be used to run our Java application as stated. Our Java application will be available for Windows,Linux and Macintosh systems. Since academic warfare has no multiplayer option, it will be executed in one computer at a time. An executable file and some required other files will be enough to execute the our program. Player will use mouse to interact with game and play the game. Keyboard will be used only to enter the name of the highscores.

2.4. Peripheral Modules Management

The game will have peripheral modules used for host system related interactions such as file system accesses, sound or system time. File system accesses are required for storing sounds, textures, sprites, configuration files, game state files and high-score files. Sound driver should also be accessed properly in order to play sounds of the game properly. System time is used for synchronization purposes and leaderboard entry registration. High scores will be kept in a formatted text file, game states are stored as java object binaries. Game assets such as required graphics and visuals are loaded from the file system as images. Together with graphics, game sound and music data are also loaded from the file system. They will be encoded/decoded as usual sound files so we can utilize internal Java libraries for handling sound files.

3. SUBSYSTEM SERVICES

3.1. Game Logic Design

3.1.1. Overview

In this chapter, with the help of the class diagram, classes of our project will be explained in detail. Trivial methods such as set and get methods will be omitted, since those will be added at implementation stage.

Our game engine initializes with the flow of information from GUI components such as difficulty of the game, current level, name of the player and so on,With this main relations between components starts. Game engine is the administrative component of the logical structure.Game engine and game map communicates for flow of information about the surroundings of the player with respect to players input. Then the engine decides what to do with those information available. For example, if player tries to place a weapon to somewhere in a map, game engine takes the information of corresponding coordinate from game map. If couple of enemies have already reach to end,engine corresponds to that with decreasing the health points of the base. Also we have a collision mechanic in our game. We solved this issue by designing the game map as a grid map. Every movement of any dynamic object occurs as discrete grid coordinate movements with respect to our grid map design. So we can check any collision by checking coordinate checks,easily. If an enemy's bullets current coordinate overlaps with towers coordinate, towers health point will be decremented. We planned to use grid map design not just for simplifying complex problems, also since our Academic Warfare game is very suitable for this design choice.Most of the tower defence games are using this kind of design too. It makes possible to increase the smoothness of animation like view of the moving objects. Java is capable of doing these kind of things.Main relations between classes depicted in figure 5.

3.1.2. Game-Time Procedures

A game program is supposed to update itself regularly at a certain frequency. This frequency should be smaller or equal to the screen refresh rate of the system so that game can respond and be viewed properly. So game engine will repeatedly run an update routine. Once it is provided that game is updated with real time interactions of the player, development process can concentrate on basic game mechanics.

Game mechanics involve player input handling and processing interactions between game objects. Player can interact with the game objects with mouse input and objects will interact with each other depending on the rules that are set by provided game mechanics such as physics, events etc.

Update routine should not block input interrupts. Everything related “with game should be concurrent. So it will run on a separate thread. There will be several concurrent threads. Another thread will handle player input and update input related memory. Update routine involves combining various real time changes such as player input, screen refresh, physics. Once update routine makes sure that all required event handling procedure is completed properly, it will safely combine the changes and wait for a new update call.

Anything related with a game frame should be updated synchronously for avoiding race conditions and false event processing.

Movements of the game objects are provided by the PhysicsManager class. By default, all game objects can have a velocity. PhysicsManager class will look for all moving objects that have a velocity, it will update their positions according to their current velocity vector. Once the coordinates are updated, DisplayManager will draw them to their new coordinates when the new frame refresh event happens.

There will be other more specific interactions between objects such as a weapon firing to an enemy and making the enemy lose health. These interactions are controlled inside the GameEngine class according to various Event classes such as an enemy being inside the range of a weapon. Such Event condition will trigger the functionality section of that particular event and make the weapon fire to that enemy.

3.1.3. Example Scenarios

Scenario 1: Player wants to check the leaderboard.

At the beginning game program starts and shows a main menu GUI. Then, player clicks the Leaderboard button in the main menu. This triggers the screen change event and currently displayed main menu screen will be replaced with leaderboard screen, inside the DisplayManager class. Leaderboard screen loads the data from the formatted files and fills the highscore table. After proper parsing of the high score data including time, scores and names; GameEngine will update the screen with the proper information.

Scenario 2: Player wants to build a weapon, places it and defends the tower.

Player chooses the weapon of his or her choice by clicking its icon on the left side of game screen. Player drrags and drops the weapon icon onto where he or she wants to place it (on the map). This interaction will be handled by the InteractionManager class. GameEngine receives the input information on each time quantum from the InteractionManager and then updates the objects statuses accordingly. Once players declares that he pr she is ready for he next wave by clicking the “ready” button (this is handled the same way as scenario one), enemies will start to spawn from the beginning of their path. All enemies have velocities defined by their paths at certain time instances. Their position is updated by the GameEngine class acoording to their velocity vector (point per second). When an enemy gets inside the range of a weapon placed by the player, That weapon will start to fire. These kinds of conditional events are handled by GameEngine class throusgh processing all Event classes associated with the game.

Event classes have condition check methods and an action method that is triggered when the condition is satisfied. So if player places weapons properly for that particular wave, the wave will be eliminated.

3.1.4. Detailed Class Design

● Game Engine:

The central executive class of the project is Game Engine class. It manages all game-dynamics and operations between classes. It coordinates interactions between classes. In any games, Game engine is designed to be unique class to controls the whole game.

Properties:

"*gameObjects*" includes any objects (player, enemy, weapon, tower, powerUp, path, tile) of the GameObject class. It means that after GameEngine instance is created, all objects in GameObject class are created and they exists until the game is over.

"physicsManager" controls the motion of enemies and collisions. This means that from the enemies appear in the screen until they are killed, physicsManager object updates the positions of enemies and weapons' directions. It also checks the collision of enemy and bullet, then, if bullet hits the enemy, it gets collision

"displayController" is responsible for drawing objects. After the GameEngine is instantiated, displayController object is created and update the screen according to the render time.

"inputController" is object which is responsible for taking events tom keyboard and mouse. It gets current input from the user.

"gameEngineThread" run always during the game in order to update interaction between game objects.

Methods:

"run" is used for initializing every objects in GameEngine class. It works only the beginning of the game execution.

"update" is used after run method is executed and it updates every object status through game execution until the game is finished.

In addition to above declarations, the size of code for GameEngine class will be much more than our expectation. Therefore, there may be some modification about this class during implementation.

● Vector2

2D vector implementation containing 2 public float variables x and y.

● GameObject

GameObject is a fundamental generic class in which all game related objects can be extended from.

Properties:

Position is a Vector2 instance. It is to determine the position of a game object on the screen.

Size is a Vector2 instance. It şs to determine the size of the game object to be created and displayed.

Velocity is a Vector2 instance. It also is a Vector2 and it contains information about the game object's movement which is to be handled by PhysicsManager class.

Texture is an Image instance and it is the image of the game object. For example an enemy's Texture will be instantiated from something like enemy.gif.

Methods:

Getter and **Setter** methods for above properties.

drawEntity(Graphics g) for defining how that game object will be drawn on the screen.

● Event <<interface>>

Event interface is for providing conditional triggers to the GameEngine. For example, weapons will shoot to the nearest enemy if there is an enemy inside their firing range. This conditional action is defined through a class that is implementing Event interface.

● Player

Player class is another game object in the design. It is created at the starting game and it exists until game is over. İt is unique object and starts with given budget.

Properties:

"budget" holds the earning money during the by killing enemy. It has default value before starting game and it increases after any enemy is killed during the game.

Methods:

"decreaseBudget" is function which is responsible for increase the current budget by spending money to buy weapons.

"increaseBudget" is a function which is for increasing budget with earning money by killing enemy.

● Enemy

This class is a parent class for different enemies types. It includes common features of different enemies such as health, path and maxSpeed.

Properties:

"health" is health tube which is responsible for the number of lives of the enemy. Coming to end of the tube means that enemy is dead and it's score is added the current score of the player and meantime, it will be immediately cleaned up from game.

"path" is property which is shared commonly by all the enemies because there will be one path for enemies in order to reach tower.

"maxSpeed" is responsible for the speed of motion. This property can differ within enemies. During the game, this feature of the enemy can be decreased by hitting some weapon in specific time during.

Methods:

● Weapon

There will be four different weapons in the game. This class is parent class for all of them. It includes features of weapons which are different for each weapon type.

Properties:

"damage" holds the amount damage which weapon has. When the weapon hits the enemy, its health will be decreased by amount of damage of weapon.

"fireRate" which is responsible for rate of weapons' fire. It shows that how many fires a weapon can do per second.

"cost" holds the cost amount for buying a weapon during the game. There are four cost amount for different four weapons.

"unlockCost" holds the cost amount to unlock a weapon type for the first time. There are four unlock cost amount for different four weapons.

Methods:

● SpecialWeapon

This class is for weapons which damage enemy in different ways. There will be special weapons which does not just decrease the health. For example, one special weapon will not decrease enemy health, it will slow down enemy motion for a specific time.

Properties:

"effect" is a event properties which determines action which will be performed.

● Path

This class is responsible for motion way for the enemies. According the game map, it can differ from each other.

Properties:

"points" is an ArrayList instance holds the coordinates as Vector2 objects for indicating a way which is from starting point to tower.

● Tile

This class responsible for checking any grid cell in the game screen. It is controls any cell whether it is appropriate for weaponizing or it includes blocks.

Properties:

"isBlocking" is responsible for determining block cells in game screen. İt is true if the grid cell is used for block.

"isWeaponizing" is responsible for checking the given cell is used for weapon. If the cell is used for weaponizing, it returns true.

● Tower

This class is used for fulfilling main aim of the game. Since, the main of the game is defensing tower against from enemies attacks. It is responsible tower game object which player defends.

Properties:

"health" holds the number of lives of the tower. When it is hit by enemy fire, health will be decreased. İf it reaches zero, it means that game is over.

● PowerUp

This class is responsible for performing action when some situations are encountered. For example, if the enemy is come to the tile which includes mine in that cell, this class performs a explode action and kill the enemy.

Properties:

"effect" is property which determines the which action will be performed.

● HighScoresManager

This class is responsible of management of the highscores of the players. Since the aim of the game is surviving from the enemies and getting higher scores, it is important for players to see their highscores.

Properties:

"highScoresTable" is a property which holds the highscores table itself.

"highScoresFile" is a property which is for the text file that includes hig hscores and the names of the highscore owners.

Methods:

"loadHighScores()" is a method which loads the high scores from highScoresFile.

"resetHighScores()" is for resetting the high scores in terms of the users demand.It will reset all the scores.

"createTable()" creates the high scores table with the user demand. It can be used to save high scores.

"saveHighScore(score,name)" is a method that gets the users score and name. Later on it saves that high score to high score file.

● HighScoresScreen

When user click the high scores button, this class will print out the high scores from highScoreFile in descending order.

Properties:

"highScoresManager" is a property for HighScoresManager to do all the required things to show the high scores table.

● HighScoresTable

This class is for getting the names of the players that have high score and their names. These information are going to be used at HighScoresManager

Properties:

"highScoresNames" is a property to hold the names of the players who did high score.

"highScores" is a property for to hold the name of the high scorer.

● Screen

This class is for getting the names of the players that have high score and their names. These information are going to be used at HighScoresManager

Properties:

"gameObjects" is a property that is coming from Game engine. It holds core features of the game, which can be told as "renders the game itself".

"layoutManager" is a property which is for the layout of the screen, borders of the screen and layout of the screen.

"background" is a property to set the background theme of the game.

Methods:

"update()" is a method to refresh the screen and for updating the screen layout.

● MainMenuScreen

This class is for to show the main menu screen and available buttons for the user.

Properties:

"menuItems" is a property for the items in the main menu screen such as new game, load game, tutorial , options etc.

● MapSelectScreen

This class is for player to choose the map with available options.

Properties:

"mapThumbnails" is a property that includes the options about the maps in the game.

● TutorialScreen

This class is for displaying information about how to play game.

Properties:

"tutorialInformation" includes information about the game playing. It shows what the game objects are and what they do and how they will be used.

● EventManager

This class is for handling and organizing events which occurs during the gameplay.

Properties:

"events" is an object for event types which occurs during the gameplay

Methods:

"processEvents" is a function that recognizes the event and carries out this event.

● InputController

This class is responsible to take inputs from user. User can play the game with using mouse and keyboard. Keyboard will be just used for entering player name for highscore table. Rest of the controls is done by mouse.

Properties:

"mouselistener" is an object for mouse inputs. The possible inputs can be clicking, scrolling and moving.

"keyboardListener" is an object for keyboard inputs. It determines which buttons on the keyboard are pressed.

Methods:

"getCurrentInputs" is a function that takes current inputs which are entered from user by using mouselistener and keyboardlistener.

● InteractionManager

This class is responsible to process the possible interaction of the objects. Since during the game there will be interaction this class will be used to handle them.

Properties:

"gameObjects" property comes from the game engine. We need engine to process interactions.

Methods:

"processInteractions(input)" method is for the processing interactions with the given input.

● PhysicsManager

This class is responsible of physics management of the game. Uses game objects and motion manager and collision manager which are the physics part of the game.

Properties:

"gameObjects" property comes from the game engine. We need engine to process interactions.

"dTime" is a property which is called delta time and used for updating scene based on elapsed time since the game last updated.

Methods:

"process()" is a method for continuing processes of the game.

● CollisionManager

This class for determining whether there is a collision between game object and fires. It is also responsible to handle these collisions

Properties:

"gameObjects" property comes from the game engine. We need engine to process interactions.

"collisions" is property for indicating whether there is a collision or not.

Methods:

"getCollisions" function checks the fire and game object location and if there is overlapping, it returns collision.

● MotionManager

This class for processing the motions of the objects during the game.

Properties:

"gameObjects" property comes from the game engine. We need engine to process interactions.

Methods:

"updatePositions()" is a method to update positions of the objects during the game.

● DisplayController

This class is to handle displaying of current screen of the game. It determines render time for displaying.

Properties:

"screen" is object of Screen class and is responsible for displaying game screen.

"renderTime" is property for rendering screen in specific time.

Methods:

"drawObjects" is function to draw game objects at their current location in the screen.

3.2. GUI Design

Commented [1]: BURADAN AŞAĞISI TAMAM!!!!!! ^^

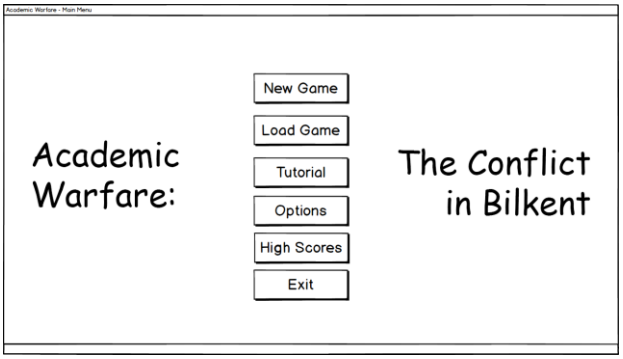
3.2.1. Overview

In this part, the screens will be explained via mockups. Our game will use Java's Swing library. The action flow of the screens will be explained.

3.2.2. GUI Components

3.2.2.1. The Main Menu

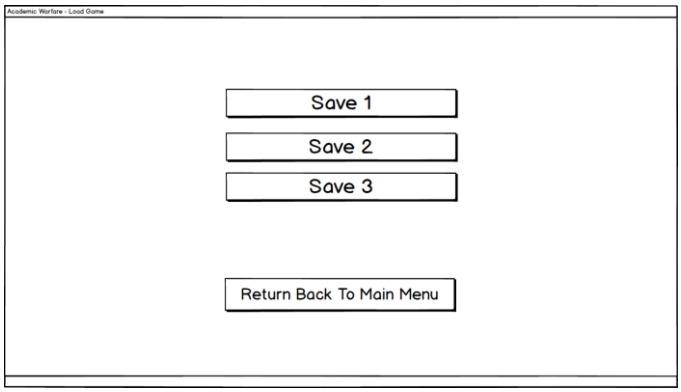
The Main Screen is the main component of a navigational GUI. This screen is shown below and is the opening screen for the game. Through this



interface the user can open up a new game, load a game, watch the tutorial, set options, view highscores and exit the game. The background will contain a screenshot from the game.

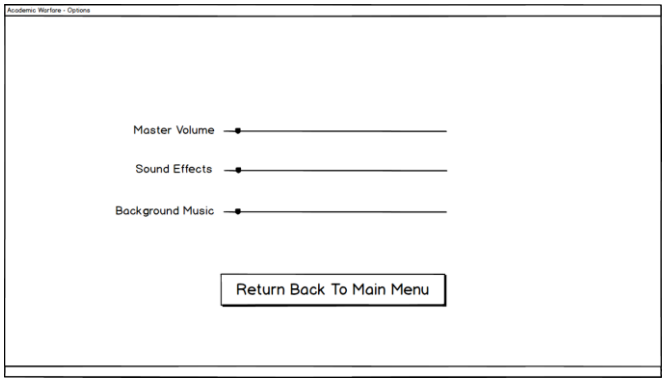
3.2.2.2. The Load Game

In this screen there is an interface for the users to select the previously saved games. In our game, the user can save at maximum 3 games. The background of this screen will also have a screenshot from the game.



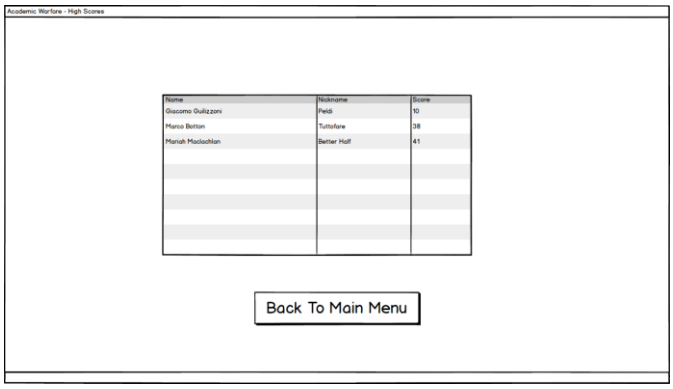
3.2.2.3. Options

In Options menu, the user can adjust the volume settings of the game. The user can return back to the main menu after he or she is done adjusting.



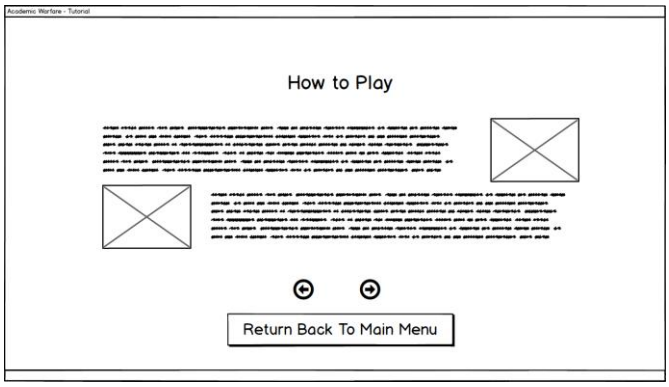
3.2.2.4. HighScores

In this menu the highscores will be displayed. The user can return to the main menu when he/she wants.



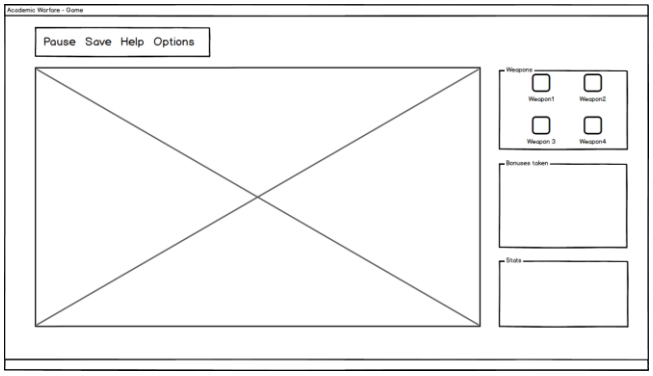
3.2.2.5. Tutorial

The user can view how to play the game via this menu.



3.2.2.6. In game

The in game screen will be as such. Some of the functionalities of the main game can be used via clicking on these buttons



3.2.3. GUI Class Design

In the project, we decided to use the swing library of Java (javax.swing) as it is programmer friendly. Since this project has many GUI components, these components will be instantiated in many classes which would create a large class

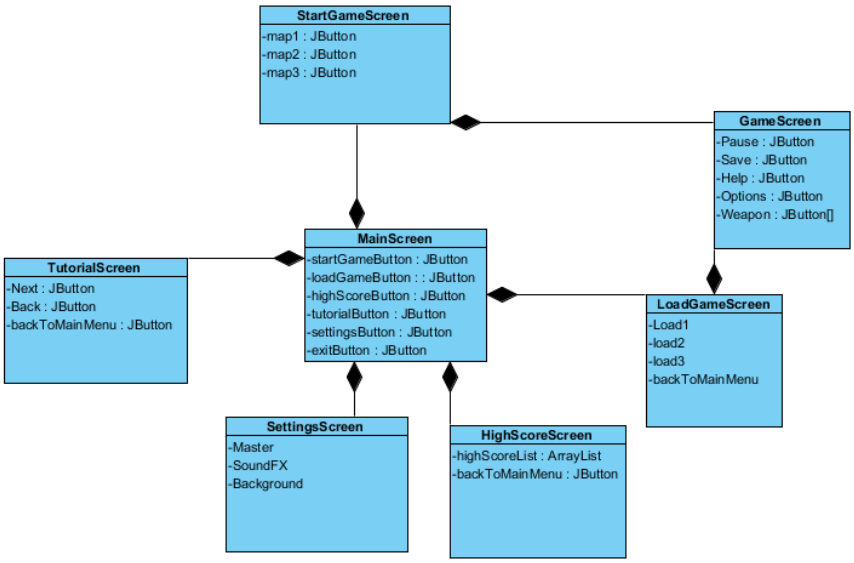


diagram.

Figure : GUI Class Diagram

3.2.4. GUI Activity Flow

The GUI starts with the game initialization. With this initialization the main menu opens and the user can choose one of six options which are New Game, Load Game, Tutorial, Options, High Score, Exit.

Upon clicking the New Game, the game starts. When Load game is clicked the user continues with the previously saved game. The user can return to the Main Menu in the Load Game screen via the button “Return to main menu”.

Commented [2]: abi diğer projeden kopyala direkt açıklamaları değiştiririz.

Commented [3]: la isimlerinin de değişmesi lazım bakayım bir anlar plagarism yeriz

Commented [4]: Değiştirek?

Commented [5]: bakıyom şimdi

When the user clicks on Tutorial, a brief tutorial of the game appears. The tutorial screen has more than one pages, each similar in regards of format but different in content.

After clicking Options, the user comes across an options menu where he can adjust sound settings, which are SoundFX and Music. The user can also alter the Master Volume, muting the game completely.

When the user clicks High Score, he or she can see his or hers highest scores.

And after clicking the exit button the game is terminated.

In game features will be added as a drop-down menu to the game screen.

3.3. Sound Design

There are 2 types of sounds in the program, SoundFX and the Music. Java has classes to help play the music while the program is running which will be implemented into the program. The songs will be taken from a royalty-free website.

3.3.1. SoundFX

The game will have warning effects for the beginning of a new wave and if the tower is being attacked and some effects for the guns.

3.3.2. Music

There will also be a background music playing throughout the program.