Bilkent University

Department of Computer Engineering

CS 319 Term Project

Section 02 Group 2c

Space Invaders

Analysis Report

Iteration 2

Project Group Members

1-Yalchin Aliyev
2-Enes Erol
3-Koray Gürses
4-Arda Gültekin

Supervisor: Uğur Doğrusöz

April 3, 2018

Contents

1. Introduction

For the CS319 term project, we decided to design and implement “Space Invaders” game. This is a well-known game first released in 1978 by Taito, in Japan. This game has re-created many times for different consoles such Atari and Tetris.

Game structure had a crucial part on our decision. We think that “Space Invaders” game is well suited for CS319 because it is suitable for object oriented design. It is also a good choice because we can add extra features to differentiate it from the original game. That way, we can improve our knowledge on object oriented design.

Current features for the game are:

- Single Player Mode
- Obstacles which work in different ways then each other.
- Increasing difficulty on further levels.
- Variety of levels and enemies.
- Buffs/bonuses for the user.
- Sound option.

For the implementation, we will use Java, Swing specifically. We are aiming to make a smooth design to ease our implementation and learn as much as possible on object oriented design by making this project as a team.

2. Overview

2.1 Gameplay

It is a two-dimensional game that a player controls a ship that has a cannon to shoot alien ships by moving the ship left to right or vice versa at the bottom of the screen. Alien ships are also move horizontally across the screen as they approach to bottom of the screen.

The main objective of the game is defeat all the alien ships by shooting them with your controlled ship. The player earns point as they defeat the alien ships and their movement speed and shooting speed increase as the number of the alien ships get destroyed. After defeating the aliens, new and more difficult to beat level of aliens have been brought. When the player finish a level, extra life point has been added. When the enemy ships reach the bottom of the screen or the player loses all the extra life, the game will end. There are also various walls that protect the player from the aliens' attacks, some of them can be destroyed after getting too much hit from both player and alien ships. Further features might be added during the development stage.

2.2 Level

Levels of the game are planned to start easy but with more and more level are done, the next level will always be harder than the previous one and introduce the user to new enemies, buffs and obstacles. For example, user can hide behind iron walls in further levels

but the enemies will move faster and new enemy ships with more health will be added to further levels. Some ships and obstacles will provide buffs/upgrades to the user's ship. Each level will be different from other ones in terms of difficulty, enemies and obstacles.

2.3 The Player

Player will be able to select different buttons to control their ship. Players are able to move in certain directions in order to shoot enemy ships and obstacles and collect the upgrades when they drop from other objects in the game. Players will have a certain amount of health and they will be destroyed when they use all of their health.

2.4 The Enemy

Enemy ships will be controlled by the game's AI. There will be different kind of enemies with different difficulties. Some of these enemies will be able to drop specific buffs when they are destroyed.

2.5 Bullets

Bullets will be shot from the enemies and the player. These bullets will move in a vertical line. With certain buffs, the bullets' speed can increase in order to make the game different from the original. When the player's bullet and an enemy's bullet got hit to each other, that won't be count as a hit, they will continue moving vertically until they hit the enemy on the other side, bullets will disappear when they reach the end of that level's map or hit an enemy ship/obstacle.

2.6 Settings

Players can change the buttons of the game from the settings. Users can also change the volume of the game's sound.

3. Functional Requirements

- Start game; mandatory option for all of the games.
- Select a level; players are free to choose between different levels that we will design. In order to access all of the levels, user have to finish the game first.
- On default, user will control the ship with left arrow, right arrow keys and can shoot with the space bar. User can change these keys anytime they want by using the settings menu.
- User can learn how to play the game in the how to play option, in the main menu.
- Game will offer various buffs and upgrades in further levels. Also, enemies will get stronger in further levels.
- User can use some obstacles to get buffs or hide behind them. This means sometimes obstacles are there to make the game harder for the user but a better player can use these obstacles for his/her benefit.
- There is a pause game option. User can press that anytime in game. Default key for pause is ESC.
- There is a high score table for player to see his top scores.

4. Non-Functional Requirements

4.1. Game Performance

In this game there will be animations, various of space ships including our own and music. We are trying to make the game work smoothly, so that the components of the game do not slow down the game. Frame rate is one of the key component that effects the user's experience from the game, so the game should run with high FPS (Frame per Second). Furthermore, we will try to minimize the system requirements of the game to be compatible with the vast majority of the computer systems.

4.2. User Friendly Interface

Significance of the User Friendly Interface is to create good first impressions to the Players. So, we are trying to design the interface neat, clear and smooth. Features of the game will be easy-accessible to the players such as How to Play, Sound Settings and High Score Menu.

4.3. Competitive Game Experience

In main menu there is an option called "High Scores". All the achieved score will be stored there, so players will try to get higher score from the game in order to be on top of the High Score list.

4.4. Extendibility

4.4.1 Co-op Mode

Space Invaders is a single player game and the purpose of the game is to achieve the higher score than the previous attempts. We are planning to make the Co-op Mode with 2 players or more players. Furthermore, there would be another High Score list for 2 players

4.4.2 Upgrades

In the regular Space Invaders game, when the player destroys the alien ships, they only reward the player with points. We are planning to bring some upgrades in the game. When the player destroys the enemy ship, there will be a chance that, the enemy ship might drop an upgrade. The probability that the enemy ships are dropping upgrades will determined by random number generator. Those upgrades would be;

1) Move Speed increase

2) Attack Speed increase

3) Enemy Ships' attack speed decrease

4) Enemy Ships' move speed decrease

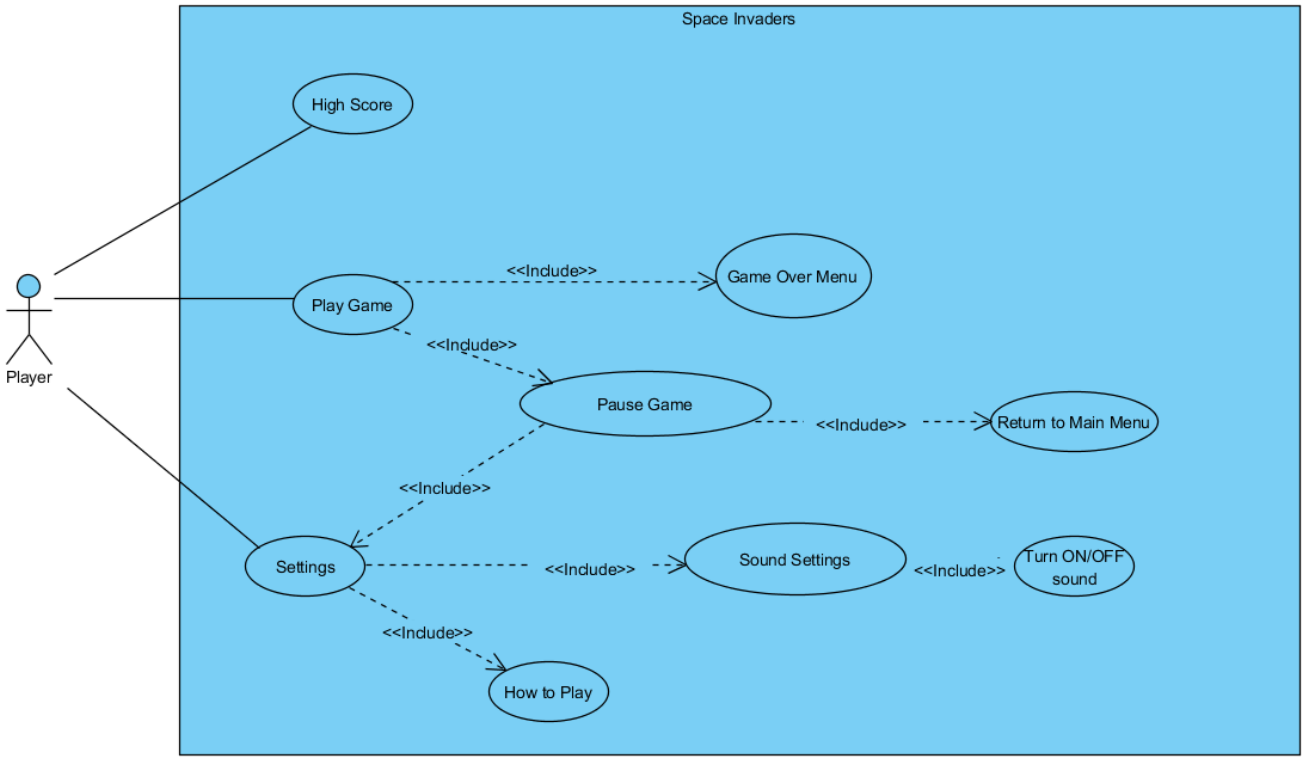
We are planning to bring more upgrades in the future.

4.4.3 Bonus Enemy Ships

We are planning to make a bonus ship that reward the player with a huge amount of points and possible upgrades. The spawn probability of these bonus ships will be determined by random number generators.

5. System Models

5.1 Use case model:



Use Case No.1

Use Case: Play Game

Primary Actor: Player

Entry Conditions:

1) Player must be in the main menu and select the “Play Game” button.

Exit Conditions:

1) When the player loses all the life points.
2) When the player pauses the game and selects the “Return to Main Menu” button.

Flow of Events:

1) Player selects the “Play Game” button.
2) The game has unlimited levels so player should lose to return to main menu.
3) Player proceeds to “Game Over Menu” and enter his/her nickname so that their score will be added to the high scores.
4) Player returns to the main menu.

Alternative Flow of Events:

1) During the game player can pause the game by pressing “Esc” button and choose the “Return to Main Menu” to return to main menu.

Use Case No.2

Use Case: Settings

Primary Actor: Player

Entry Conditions:

1) Player must be in main menu.
2) Player must be in pause menu.

Exit Conditions:

1) Player must press the “Esc” button.

Flow of Events:

1) Player chooses “Settings” button from the **Main Menu.**
2) Player can choose How to Play and Sound Settings.
3) Player chooses “Sound Settings” and turns on or off the music.
4) Player returns to the game or the main menu.

Alternative Flow of Events:

1) Player chooses “Settings” button from the **Pause Menu.**
2) Player can choose How to Play and Sound Settings.
3) Player chooses “How to Play” and reads the instructions of the game.
4) Player returns to the game or the main menu.

Use Case No.3

Use Case: How to Play

Primary Actor: Actor

Entry Conditions: Player must be in the settings and must select the “How to Play” button.

Exit Conditions: Player presses the “Esc” button.a

Main Flow of Events:

1) Player chooses “Settings” button from Main Menu or Pause Menu.
2) Player chooses “How to Play” button.
3) Player reads and learns the instructions.
4) Player returns to the game or the main menu.

Use Case No.4

Use Case: High Score

Primary Actor: Player

Entry Conditions:

1) Player must be in the “Settings”.

2) Player must finish the game to be in the Game Over Menu.

Exit Conditions: Player presses the “Esc” button.

Main Flow of Events:

1) Player chooses “High Score” from the Main Menu.
2) Player can see all the scores that recorded from the game.
3) Player presses “Esc” button to return to Main Menu.

Use Case No.5

Use Case: Pause Game

Primary Actor: Player

Entry Conditions: Player must be in the game.

Exit Conditions: Player can return the game by pressing “Esc” button or return the main menu by choosing “Return to Main Menu” option.

Main Flow of Events:

1) Player chooses the “Start Game” button in the main menu.
2) During the game player can pause the game by pressing “Esc” button.
3) Player selects the “Setting” button.
4) Player can select “How to Play” or “Sound Settings” button.
5) Player selects “How to Play” button and read the instructions of the game.
6) Player returns the game or return to main menu.

Alternative Flow of Events No.1:

1) Player chooses the “Start Game” button in the main menu.
2) During the game player can pause the game by pressing “Esc” button.
3) Player selects the “Setting” button.
4) Player can select “How to Play” or “Sound Settings” button.
5) Player selects “Sound Settings” button and can mute or unmute the music.
6) Player returns the game or return to main menu.

Alternative Flow of Events No.2:

1) Player chooses the “Start Game” button in the main menu.
2) During the game player can pause the game by pressing “Esc” button.
3) Player returns to the main menu by pressing “Return to the Main Menu” button.

Use Case No.6

Use Case: Game Over Menu

Primary Actor: Player

Entry Conditions: Player must lose all the life points.

Exit Conditions: Player must insert a name/nickname and then press “OK” to proceed.

Main Flow of Events:

1) Player loses all the life points.
2) Game Over Menu pops up.
3) Player inserts name/nickname and clicks “OK” after.
4) Player returns to Main Menu.

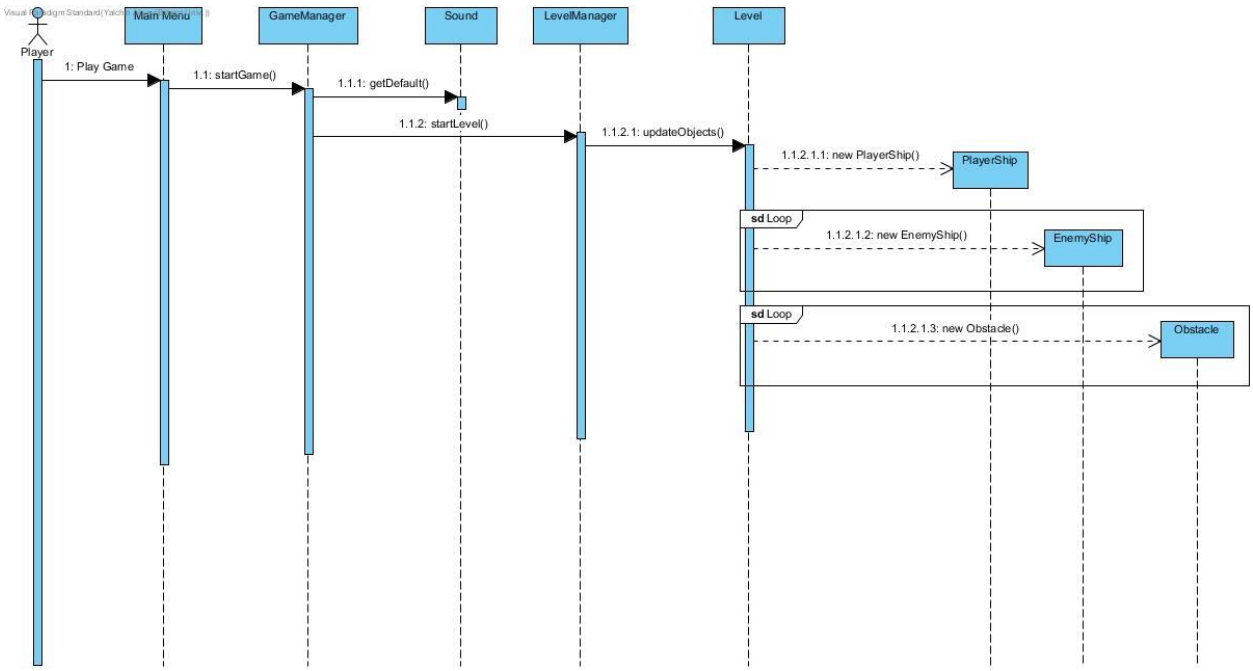
5.2 Dynamic Models

5.2.1 Sequence Diagrams

5.2.1.1 Start Game

Scenario: Player starts the game

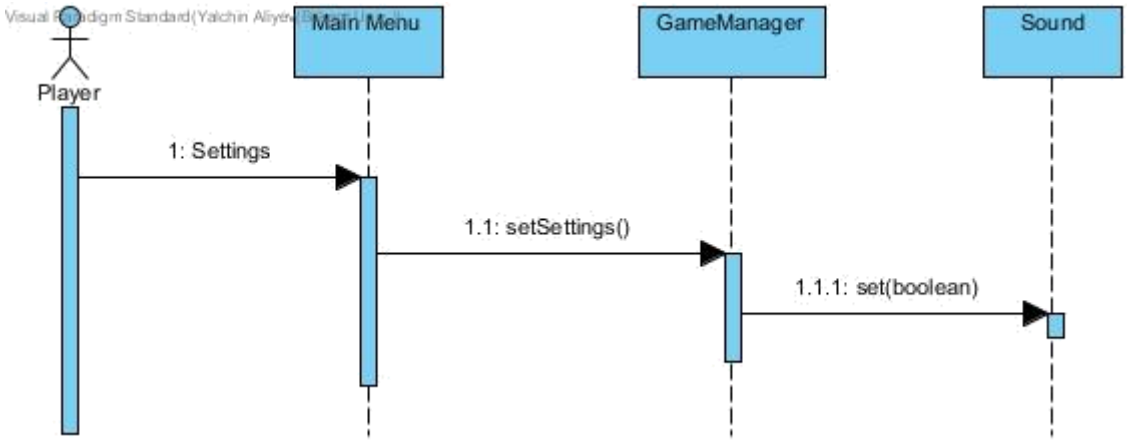
Player chooses Play Game button from the Main Menu. GameManager starts initializing the game. It first gets default sound settings from Sound class. Then GameManager invokes LevelManager to start the Level. Level class initializes all the GameObjects: PlayerShip, EnemyShips and Obstacles.



5.2.1.2 Settings

Scenario: Player changes sound settings from Main menu

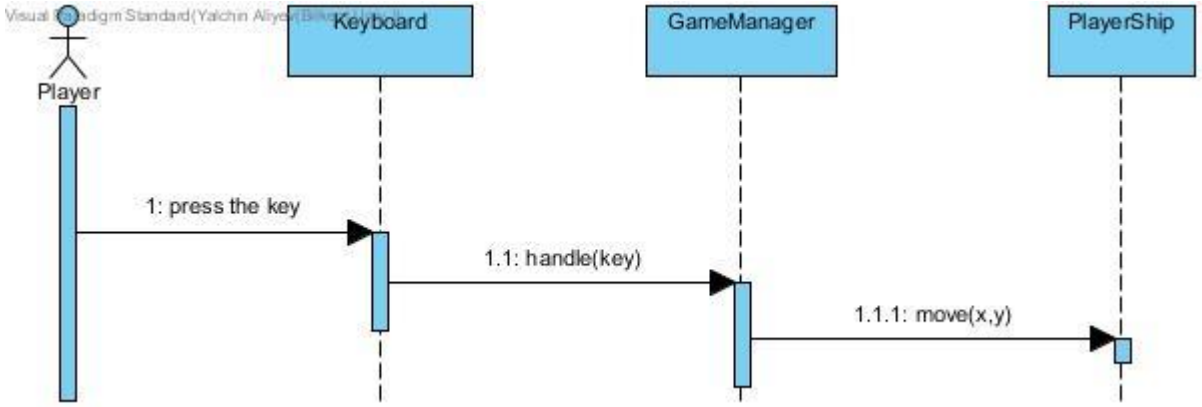
If the Player wants to change sound settings, he/she chooses Settings from Main Menu. If any settings have been changed it will be recognized by GameManager and GameManager will set Sound invoking the method set(Boolean).



5.2.1.3 Move

Scenario: Player wants to move the ship

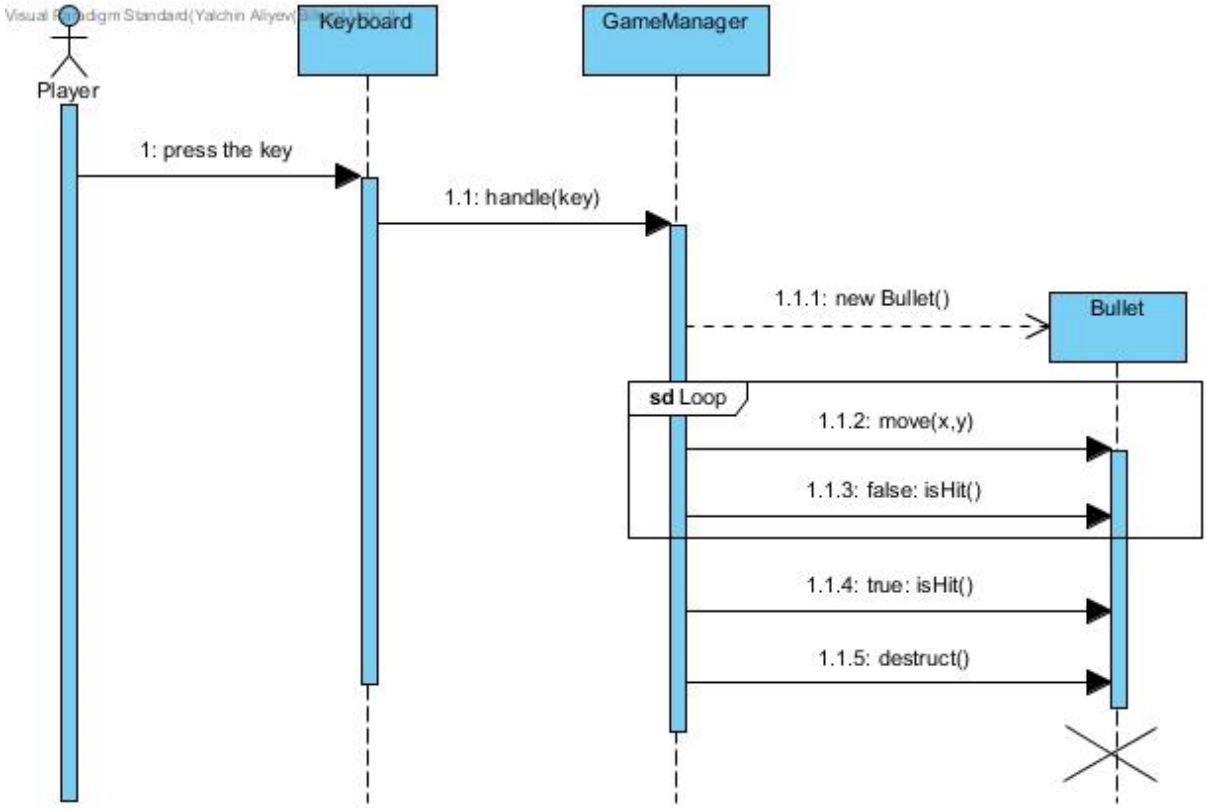
Player can move the ship in the horizontal direction. He/She uses either left or right arrow on the keyboard. It invokes the handle method on GameManager. Then GameManager moves the PlayerShip accordingly left or right invoking move(x,y) method on PlayerShip.



5.2.1.4 Shoot

Scenario: Player wants to shoot

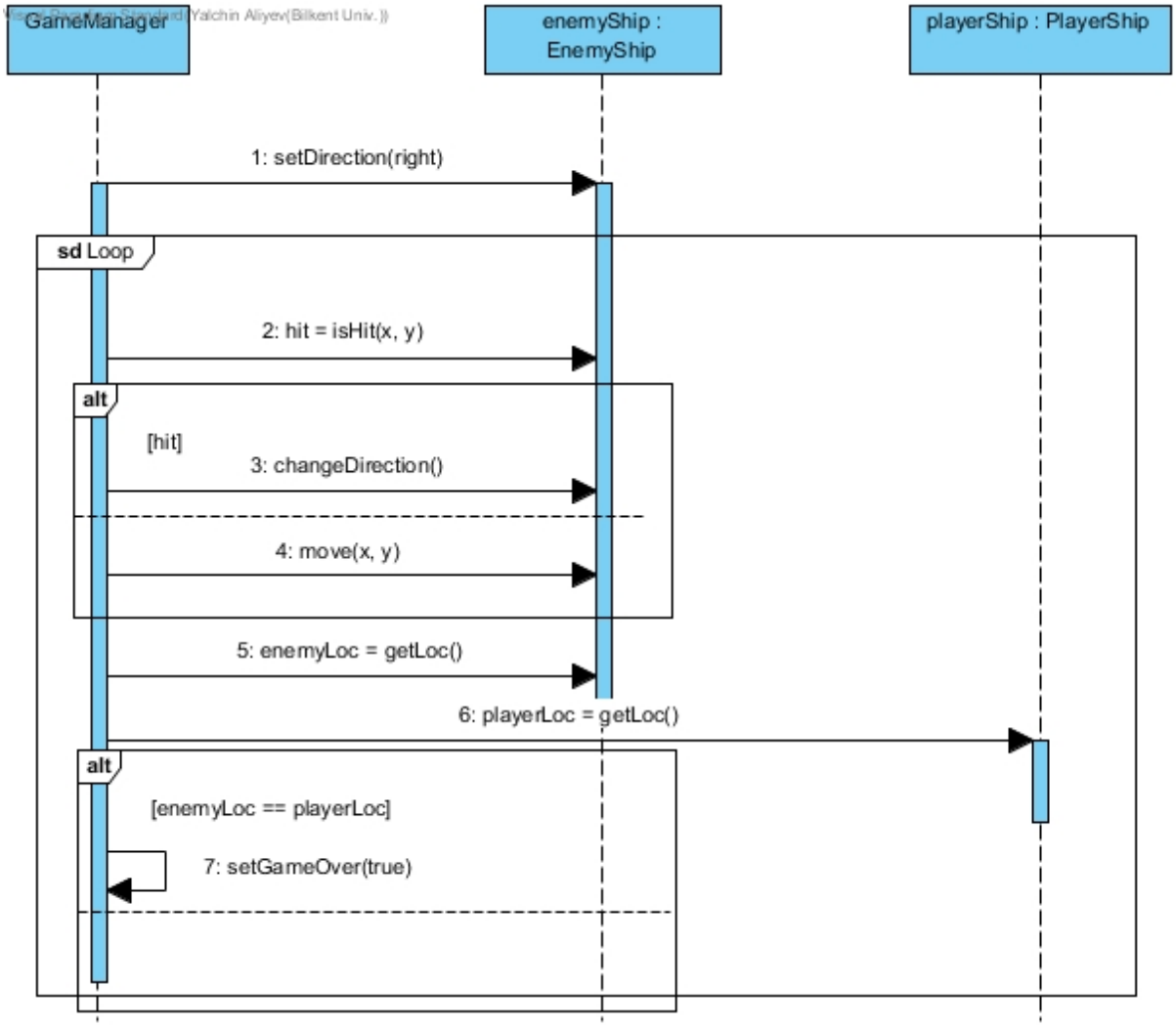
When the player wants to shoot, he/she presses the key on keyboard. It invokes the handle method on GameManager. GameManager creates a new Bullet at the same location as the PlayerShip. Then it keeps moving the Bullet until it hits something. Eventually GameManager destroys the Bullet.



5.2.1.5 Movement of Enemy Ship

Scenario: Enemy Ship wants to move

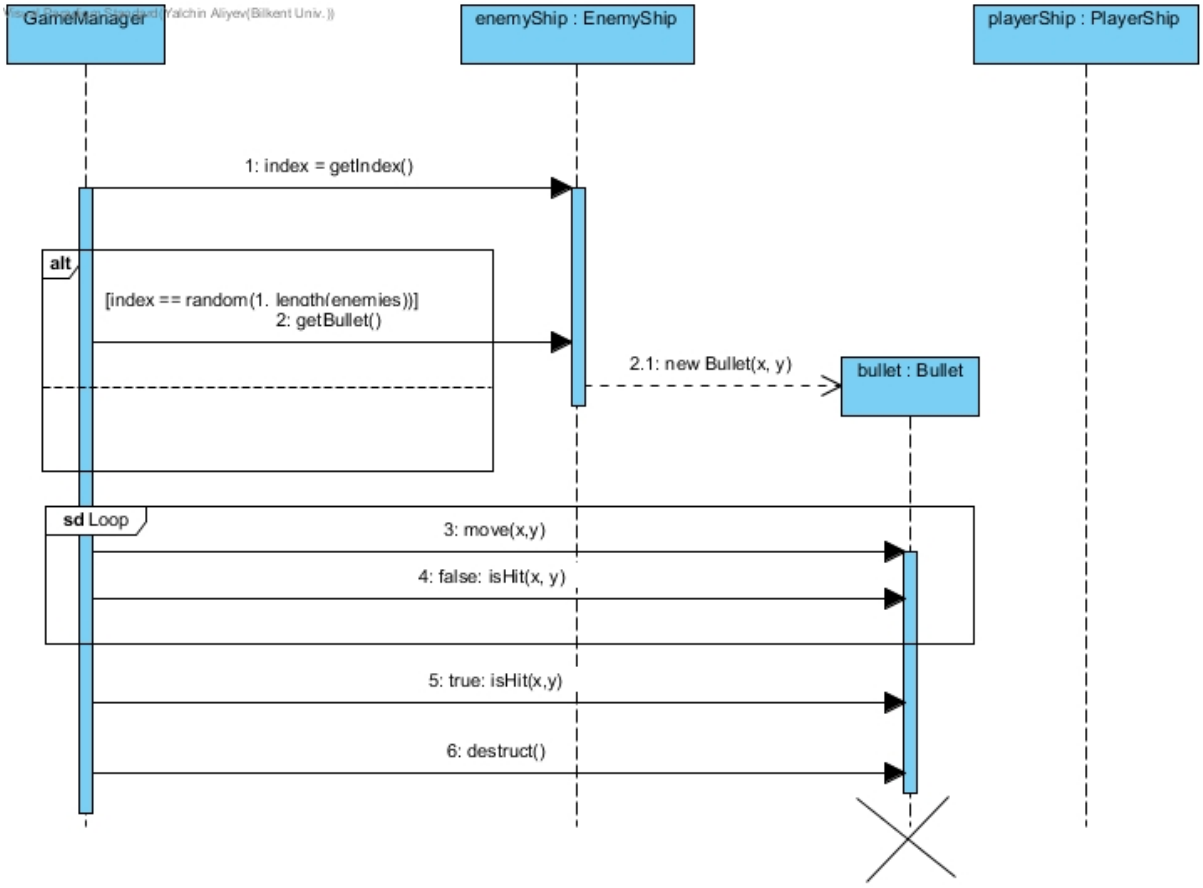
Enemy ship's movement is controlled by the game manager. Enemy ship can move only left, right and down towards to the player. When it is moving right or left, game manager moves it down for one step towards the user and changes its direction to left or right. When enemy ship collides with the player ship, game finishes and the player lose the game.



5.2.1.6 Fire Mechanism of Enemy Ship

Scenario: Enemy Ship wants to fire a bullet

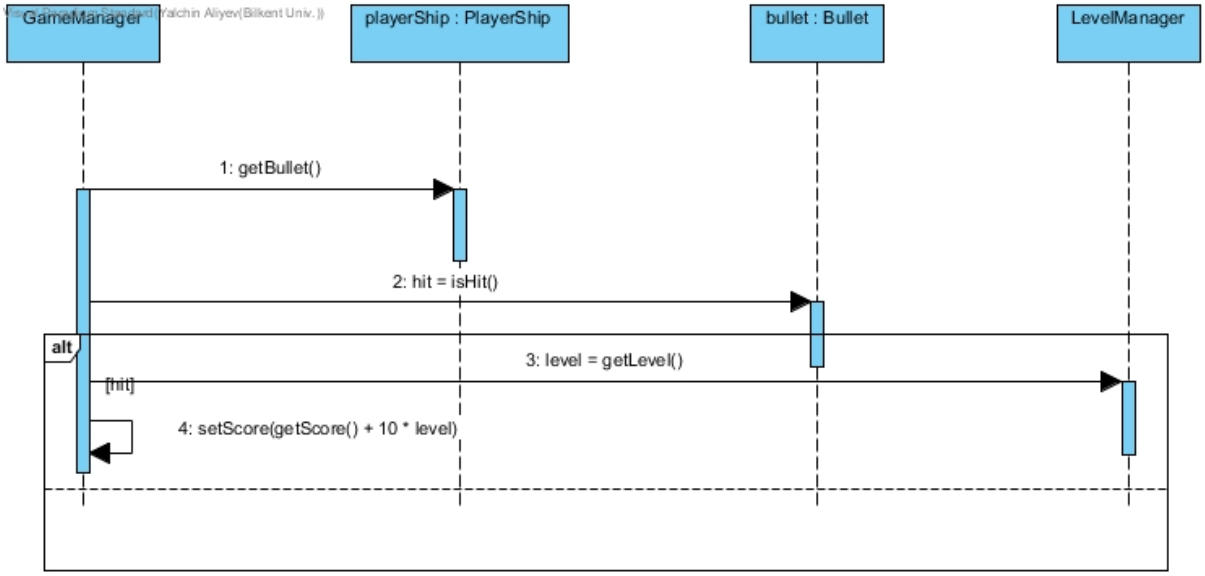
Enemy ship's fire mechanism is controlled by the game manager. There are several enemy ships in the game and each of them has a unique index. Game manager randomly chooses an enemy ship to fire a bullet. Bullet moves towards the player ship and stays alive unless it collides with the player ship or game screen border. If it collides with the user the game finishes and player lose.



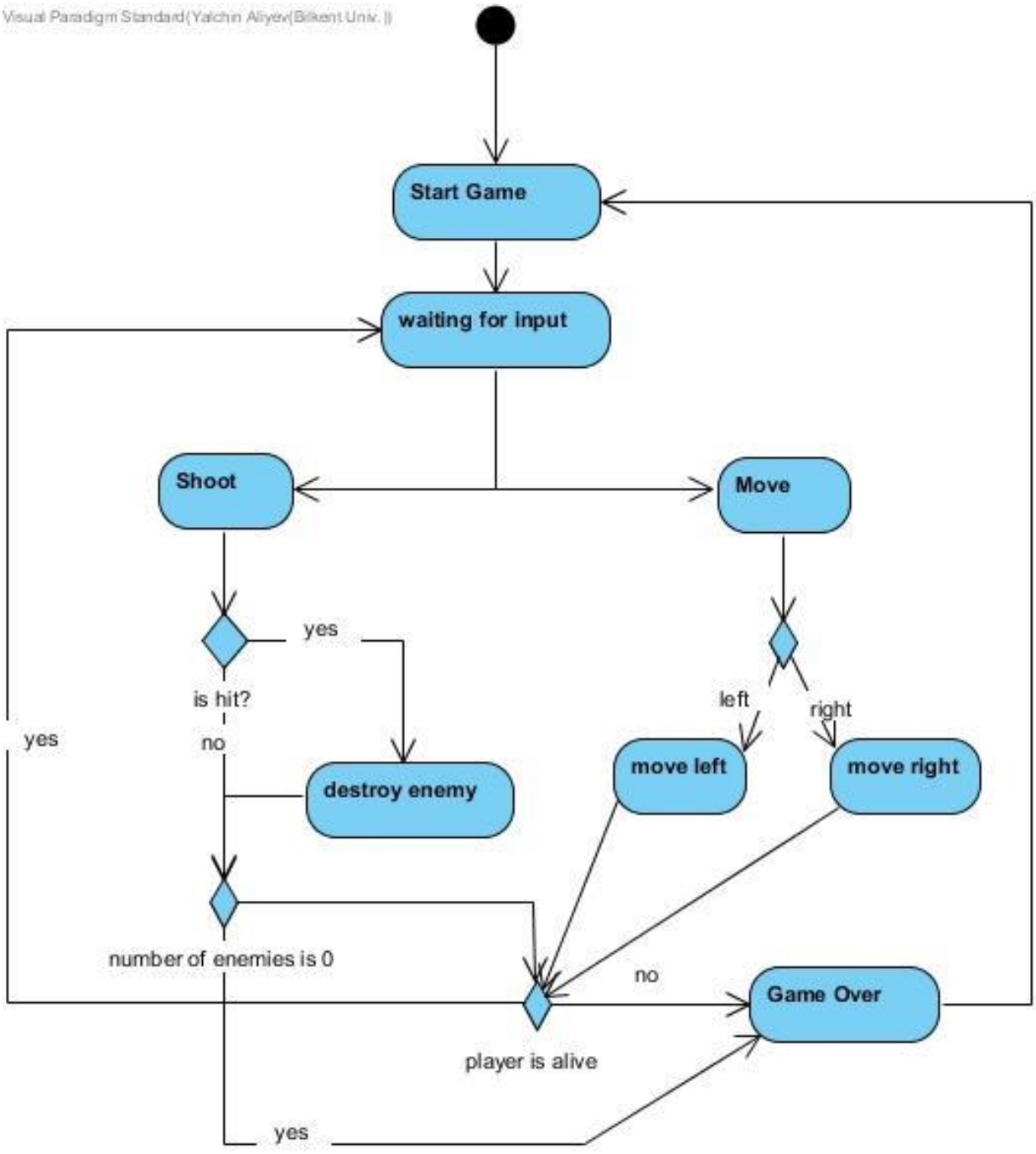
5.2.1.7 Score system

Scenario: Player gets score

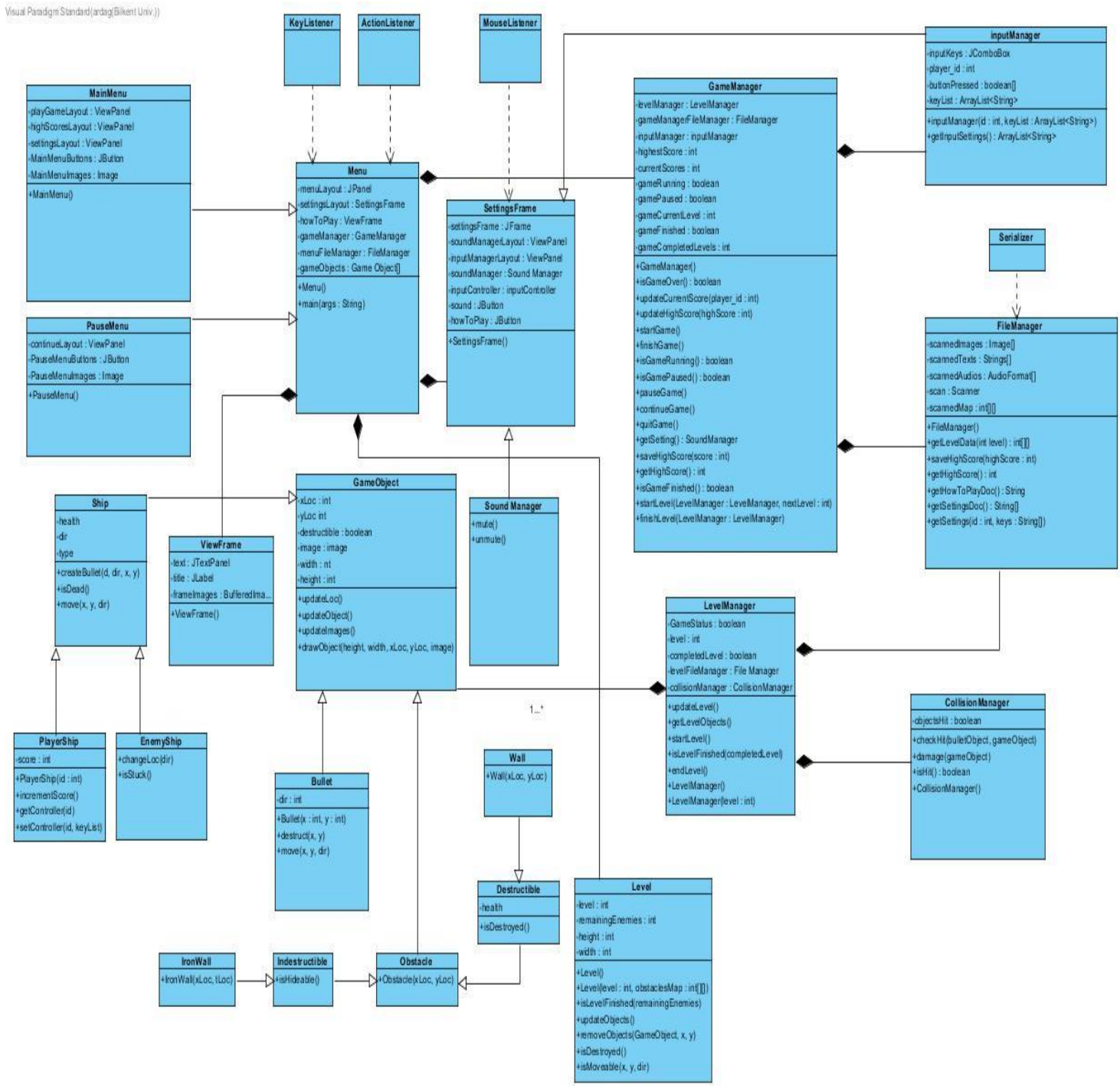
Game manager calculates, and increments score every time when the player ship hits at least one of the enemy ships with the bullet. The calculation of the score also depends on the level of the game. The score that the user gets is calculated as 10 * level.



5.2.2 Activity Diagram



5.3 Final Object and class Model



This is the final class diagram for the “Space Invaders” game.

“Menu” class has two subclasses called “MainMenu” and “PauseMenu”. Menu is the first thing the users will see when they start the game (main interface). “MainMenu” will start the game whenever the user choses “Play” after that “Level Manager” will be initialized by “GameManager”. Whenever the user choose to pause the game “PauseMenu” will appear.

There are also functional classes which are “inputManager”, “SoundManager”, “LevelManager” and “CollisionManager”.

- Purpose of the “inputManager” class is to keep the user’s settings for movement buttons and fire button. This class reports the new buttons’ information to the game. Therefore, they can be updated.
- “SoundManager” class will keep the settings for the in-game sound. User can change the volume of the sounds from there.
- “LevelManager” class initializes a new level when the user starts the game. This class also keeps track of the game data. It gets the data from “CollisionManager” and it request “Level” class to be updated. When the level ends, “LevelManager” starts the next level. When the user finishes all the levels in the game, this class finishes the game.
- “CollisionManager” looks for a hit between the game objects and the bullet coming out from the user’s ship. If there is a collision, it should check if the object got hit is destroyable or not. If it is destroyable it requests that object to be damaged. This class should work together with “LevelManager” class.
- These four were the current functional classes of the “Space Invaders” game. Next on the list is to explain how the classes that provides objects work.
- “GameObject” class collects all of the features of the game objects.

  We need this class to simplify the control on game objects when the game is running. This class has x coordinate and y coordinate of the object (all objects share these two attributes). “Obstacle”, “Ship” and “Bullet” classes are the subclasses of this class.

- “Ship” class if for all of the ships in the game. There is one ship controlled by the user and all of the other ships will be controlled by the games AI. All of the ships have a certain amount of health (some of them have more health than others). All of the ships can also fire bullets and they have a direction.
- “PlayerShip” this class is designed for the user’s ship. This ship will be controlled by the user who is playing the game. This class will have a set of input keys. This ship can get buffs from some of the enemy ships when it destroys them.
- “EnemyShip” class is designed for the enemy ships which are controlled by the game’s AI.
- “Obstacle” class is designed for the obstacles in the game. There are two kinds of obstacles in the game. First kind of obstacles are destroyable by the player’s ship. Second kind of obstacles are cannot be destroyable by the player’s ship, but player can use these obstacles to hide behind them and take cover from enemy ships’ bullets. These 2 types of obstacles are defined in “Indestructible” and “Destructible” classes which are the subclasses of the “Obstacle” class.
- “Bullet” is another subclass of the “GameObject” class. Both of the player’s ship and enemy ships can create bullets and fire them to each other. Every ship in the “Space Invaders” game can use instances of this class. Bullets will always shot directly in front of the ships and will move in a vertical direction. If any objects in game get

hit by the bullets, “CollisionManager” class checks the hit and request for an update accordingly.

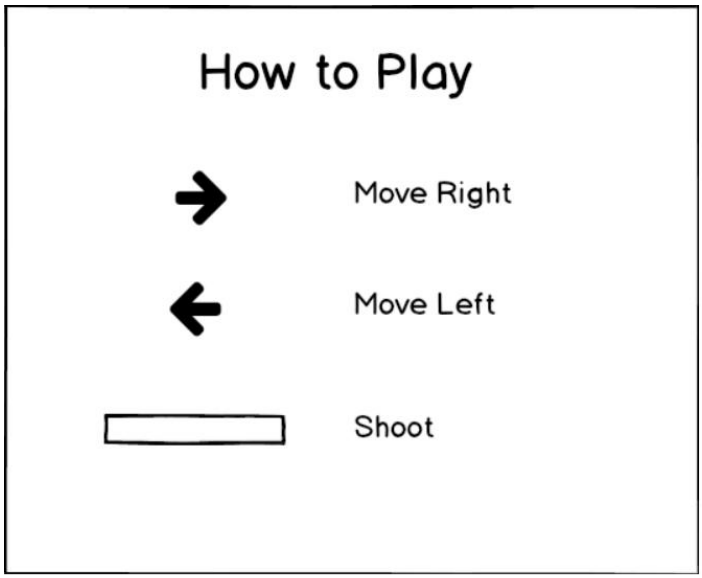
- “Wall” is the subclass of “Destructible”. This name might change in the future. We’re planning user to receive buffs when they destroy these walls. When they get hit, they will drop a buff and be removed from the level.
- “IronWall” is the subclass of “Indestructible”. This name might change in the future. These kind of walls cannot be destroyed, no matter how much they got hit. User can use these as a cover from enemy bullets.

5.4 User interface and screen mock – ups

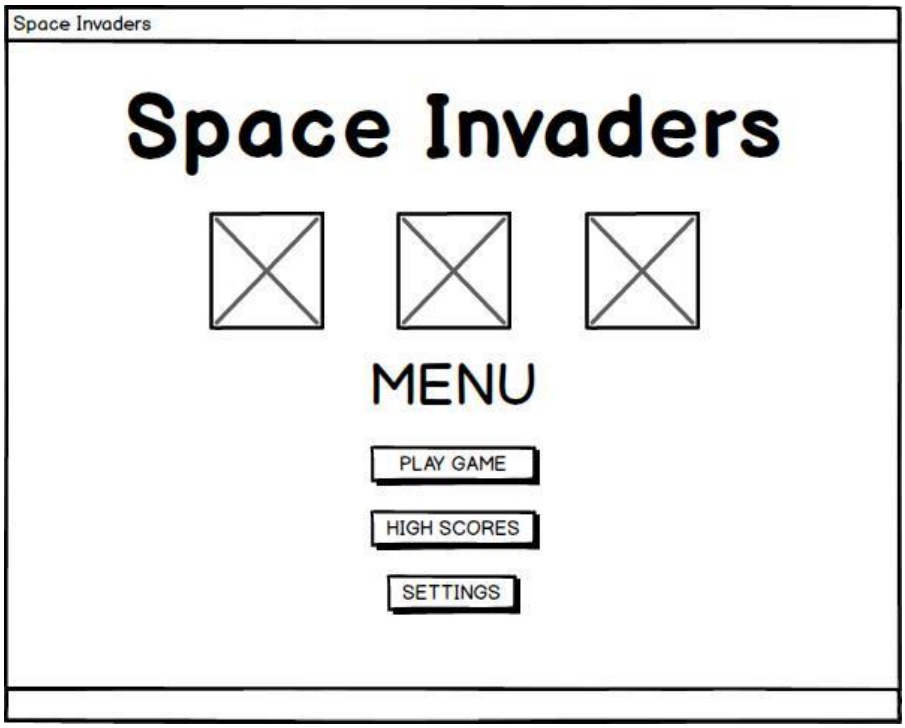
Game Over Screen:

In Game Screen:

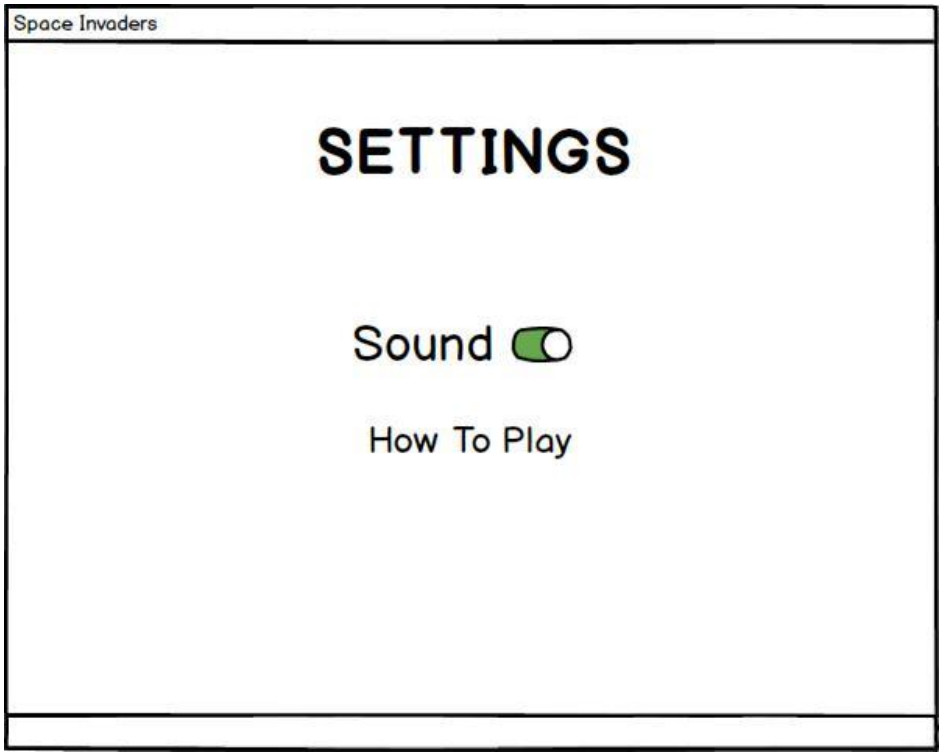
How to Play Screen:



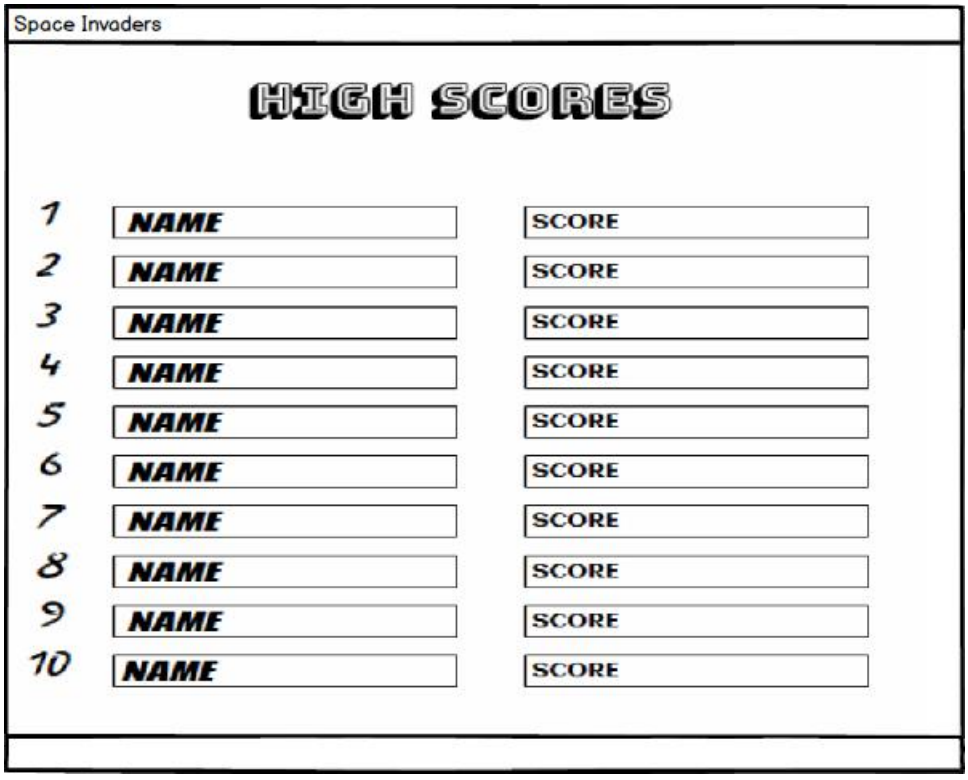
Main Menu:



Settings:



High Scores Screen:



6. Gloassary & References

[1] http://www.pacxon4u.com/space-invaders