Bilkent University

Department of Computer Engineering

CS 319 Term Project

Section 1

Group 1B

Battle City

Analysis Report

Project Group Members

1-Kaan Sancak

2-Mahin Khankishizade

3-Ozan Kerem Devamlı

Supervisor: Ugur Dogrusoz

Contents

1 Introduction

As a group 1B, we decided to design and implement nostalgic “Battle City” game. Battle City game was first released in 1985 by Namco Games. It was one of the most popular games during the 80’s and also 90’s. As children of 90’s, we decided to make this game as project because we realized that there were many instances which we can implement as objects in the game like enemy bots, different obstacles, different map organizations and also it seemed like it is fun to do this game.

As mentioned before, our driving motivation to choose this game comes from the structure of the game. We realized that “Battle City” is very suitable for object oriented design. This suitability also allowed us to differentiate the game and add some additional features as we desire. Currently game will have the following features:

- Single Player Mode
- Multiplayer Mode
- Sound Options
- Adjustable Game Speed
- Variety of Maps (in different difficulty level)
- Different Obstacles
- Bonuses
- Smooth Graphics

The player will be represented as a tank, which will be able to fire, move and hide. In addition to player’s own tank, there will also be CPU controlled bot tanks, which will be main enemies of the player. By passing each level from five, the game atmosphere and challenge will be increased.

For example, the number of enemies, the span of their lives will rise. Moreover, the map will get more complicated by the turns. The game is keyboard and mouse controlled, for instance, the player will be able to shoot the enemy by pressing the spacebar, or any key he/she chose in the settings. (Look at 3.1.3 for clarification). The objective of the game is divided into two, depending on which mode user chose to play in.

To implement the game we will use Java platform, more precisely we are planning to use JavaFX because of its superiority on graphics and UI design over standard Java distribution. Our goal is to implement this game by using the object oriented design principles are taught in CS 319 as much as possible.

2 Overview

2.1 Gameplay

Battle City is both single and multiplayer 2D game. A player of this game is going to be represented by a tank. Each player can control only one tank, the other tanks are controlled by the computer. The goal of the player is to survive through each level, in other words, to shoot all enemy tanks. The number of the enemy tanks increases by the level. Player can shoot other tanks and with the help of obstacles, he/she can hide behind the walls, lakes, forests or move among them for preventing the enemy's attack. The level ends when all enemy tanks are destroyed. For the whole level sequence, the player is going to have three 'life' points. The 'life' bonus may occur randomly in the level, for increasing the life.

2.2 Map

The map of the game is going to consist of brick and stone walls, lakes and forests. For each level the map is going to be different, namely the arrangement of the obstacles will change. Brick walls are easily destructible, whereas stone ones cannot be destroyed. The player will be able to hide in the forest to be invisible. As for lakes, the player will not be able to cross the lake; however, the bullet will be able to pass it.

2.3 Player's initial items

From the beginning of the game, each player will own three 'lives' and unlimited pack of bullets.

2.4 Modes

There are two different modes, which are single and multiplayer modes. In the single mode, the player is going to be alone against all the enemies, whereas in the multiplayer mode, two players will fight together as a team.

2.5 Bonus

There is a 'life' bonus, which will appear randomly in the middle of the game, to increase the life of the player.

2.6 Bullets

Shot bullets will move in the straight line until they reach the obstacle or the enemy tank. If the obstacle is lake, the bullet will pass it. If it is a brick wall, the wall will be destroyed, if it is a stone wall, it will disappear and if it is an enemy tank, the tank will be destroyed. In addition, the player can decline the enemy's bullet with its own one. In other words, if player will shoot against upcoming bullet, that bullet will disappear.

2.7 Tanks

The tanks will move and shoot among the obstacles, with the help of the keyboard. 'up' for moving forward, 'down' for moving back, 'left' for moving to the left, 'right' for moving to the right and spacebar for shooting. For taking a bonus, the tank should simply run over it. When the tank is turning around, it's head will move accordingly In addition, the bullet shot by the tank will move in the line which the head of tank is pointing.

2.8 Settings

The player will be able to turn the volume down or up. In addition, the player will be able to choose the mode of the game.

3. Functional Requirements

3.1 Play Game

There will be two game modes; single player game and multiplayer.

3.1.1 Single Player Game

The players may access this screen from the Main Menu. This is one of the main screens in the Main Menu. By entering this screen, user will be able to start the game.

The map will be generated randomly, and the user's tank will appear on the map. In the single player game, the user's tank should survive through each level by destroying all other enemy tanks. The level ends when the last enemy tank is destroyed. The tank will also be able to enlarge its life span by the level, and fill it with the life bonus.

3.1.2 Multiplayer Game

Multiplayer Game is the second type of the Play Game. By entering this screen, the players will be able to play in multiplayers. At this point, the logic of the game will be the same; however, two players will fight together against the enemies, which will increase the fun and the challenge of the game. Both tanks will have the same amount of capabilities. However, their lives will be independent. In other words, if the first tank takes the life bonus or loses the life, it does not influence the life score of the second tank. Thus, if the first player loses, the second one still can play until his/hers life is not finished.

3.2 How to Play

The players may access this screen from the Main Menu. By entering this screen, users will be informed about:

- Rules

- Button Explanations
- Keyboard Controls
- Life Bonus
- Levels (How life, gun and speed changes by the level)

This screen will help the new players to adapt to the game, and play without errors. For example, the user will not be aware of the pausing with the help of ‘P’ key, if he/she does not read the “How to Play” section.

3.3 Options

The players may access this screen from the Main Menu. By entering this screen, users will be able to change some settings of the game. Changeable settings are below:

- Adjust the volume
- Mute/unmute the volume
- Change the keys for the players (1 or 2)

3.4 Credits

The players may access this screen from the Main Menu. By entering this screen, the players will be informed about the identities of the game’s developers and their e-mail addresses, in order to send a feedback.

3.5 Additional Requirements

3.5.1 Bonuses

There are two types of bonuses in Battle City; life bonus and speed bonus.

In Battle City, each player tank will have three lives. It means, if the enemy tank shoots the player tank three times, the player tank will be destroyed and the game will be finished. However, the player will be able to increase the life score with the life bonus, which will appear during the game at random times and places. The life bonus will appear only twice or less during each level. The player tank will increase the life score for one point after passing over the life bonus.

Speed is the essential thing needed to the player tank in this game. In order to avoid the enemy bullets or tanks, the player tank should have a high speed. However, in Battle City, the default speed of the tanks is not high and in order to make it high, the speed bonuses will appear once during each level, which will increase the speed of the player tank and help it to win.

3.5.2 Portal

Portal will help the player tanks to easily transport on the map. For example, there will be two portals on the map, one in bottom left corner, other one on the top right corner. The player entered from the first portal, will be able to pass to another portal and continue the game from the second portal.

3.6 Extendibility

3.6.1 Player mode

We are now going to make Single Player and Multiplayer (2 players) modes for the first release of our program. However, we are planning to make the Multiplayer mode with 3 and more users which can be one of the extendible features of our game for future. As we will have a class for keeping track of users, it will be easy for us to increase the number of players.

3.6.2 Artificial Intelligence

In the Single Player mode of the game, we are planning to make our bot tanks moving randomly, and shooting exact times in a second without the help of AI. However, the implementation of AI can improve the logic quality of the game in future releases. For example, tanks will be able to move and shoot considering the positions of obstacles and enemies which will make the game more realistic.

4. Non-Functional Requirements

4.1. Game Performance

In order to make our game working in high quality, we are planning to make the extra components (sounds, music and other extra features) which will not decrease the speed of the game. Since the frame rate is also affecting the general appearance of the game to user, we will try to make it to work at as high FPS (frame per second) as it is not decreasing the performance of the computer. Other than that, the system requirements will be as low as possible considering the case that it should be compatible with the computers of different system requirements of users' computers.

4.2. User-Friendly Interface

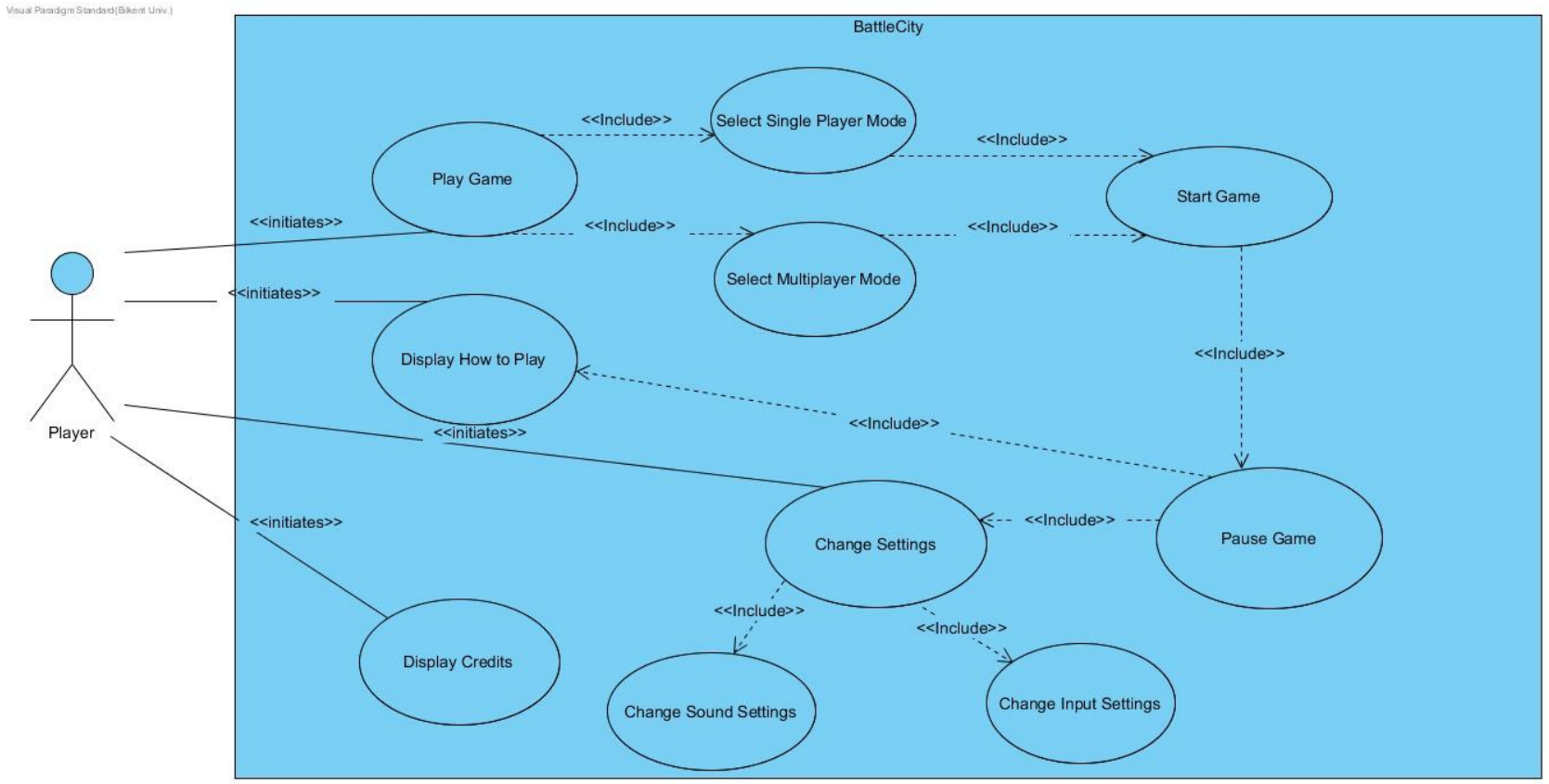
As the quality of User Interface is one of the crucial points which determine the first impression of users about the game. Therefore, we are planning to draw the objects of the game as neatly and smoothly as possible to make it graphically good looking. Options and extra features will be easily-accessible in order to make the user to experience a comfortable game atmosphere. Moreover, we will have an option to change button configuration of commands in order to make the game user-friendly.

4.3. Compatible User Interface with Real World

We are planning to make User Interface having different features which are taken from real life conditions in order to make it much more interesting for the users. Obstacles inside the game will have features taken from the real life. For example, the bricks will be destroyable however the stones which are another type of obstacle will be not destroyable. The users will have life balance which is not realistic but with that feature the game will have competitive environment which is the very common in real life

5 System Models

5.1 Use-Case Model



Use Case #1

Use Case: Play Game

Primary Actor: Player

Stakeholders and Interests:

- Player/Players selects play the game
- System creates the game and starts the game.

Pre-conditions:

- Player must be in the main menu.

Post-conditions:

Entry-conditions:

- Player should choose the game mode (Single Player / Multiplayer).

Exit conditions:

- Player selects “Quit Game” button from the pause menu.
- Player successfully finished the game by completing all levels.

Success Scenario Event Flow:

1. Player chooses “Single Player Mode”.
2. System starts game
3. Player moves its tanks by using direction key.
4. System updates the tank’s view.
5. Player fires by using fire key.
6. System creates and handles bullet.
7. Player destroys a bot tank.
8. System deallocates bullet and removes from map
9. Player destroys all bots.
10. System finishes the level and initializes the next level.

Alternative Event Flows:

1. If Player wants to return main menu:
    a. Player pauses the game by pressing pause button from game screen.

b. System displays pause menu
c. Player selects “Quit Game”
d. System asks confirmation
e. Player selects ‘Yes’.
f. System exits the game.

Use Case #2

Use Case: Pause

Primary Actor: Player

Stakeholders and Interests:

- Player presses the pause button during the game.
- System displays Pause Menu.

Pre-conditions:

- Player must be playing the game.

Post-conditions:

- No post condition

Entry-conditions:

- Player selects “Pause” button from game screen.
- Player presses “P” from keyboard.

Exit conditions:

- Player selects “Exit” button from the pause menu.
- Player presses “Esc” from keyboard.

Success Scenario Event Flow:

1. Player selects “Pause” button from game screen or presses “P”
2. System displays pause menu.

Alternative Event Flows:

1. If user wants to return to the game screen:
    a. Player presses “Esc” from keyboard.
    b. System continues the game.

Use Case #3

Use Case: How to Play

Primary Actor: Player

Stakeholders and Interests:

- Player selects how to play
- System display how to play screen

Pre-conditions:

- Player must pause the game.

Post-conditions:

- No post condition

Entry-conditions:

- Player selects “How to Play” button from pause menu.

Exit conditions:

- Player selects “Exit” button from the pause menu.
- Player presses “Esc” from keyboard.

Success Scenario Event Flow:

1. Player selects “How to Play” button from pause menu.
2. System displays How to Play screen.

Alternative Event Flows:

1. If Player selects how to play:
    a. Player selects “How to Play” button from main menu.
    b. System displays How to Play screen.
2. If Player wants to return to pause menu:
    a. Player presses “Esc” button from pause menu.
    b. System displays Pause menu.
3. If Player selects the button to return to main menu:
    a. User selects “Exit” button.
    b. System displays main menu

Use Case #4

Use Case: Credits

Primary Actor: Player

Stakeholders and Interests:

- Player selects credits from main menu.
- Battle City system displays credits.

Pre-conditions:

- Player must be in the main menu.

Post-conditions:

- No post condition

Entry-conditions:

- Player selects “Credits” button from main menu.

Exit conditions:

- Player selects “Exit” button from the credits screen.
- Player presses “Esc” from keyboard.

Success Scenario Event Flow:

3. Player selects “Credits” button from main menu.
4. System displays pause menu.

Alternative Event Flows:

1. If Player selects the button to return to main menu:
    a. Player selects “Exit” button.
    b. System displays main menu.

Use Case #5

Use Case: Settings

Primary Actor: Player

Stakeholders and Interests:

- Player selects the button to change sound settings.
- Player selects the button to change the input keys
- System changes sound settings.
- System changes the input keys

Pre-conditions:

- Player must be in main menu.
- Player must be in pause menu.

Post-conditions:

- Sound settings are updated.
- Input keys are updated.

Entry-conditions:

- Player selects “Settings” button from main menu.
- Player selects “Settings” button from pause menu.

Exit conditions:

- Player selects “Exit” button from the pause menu.
- Player presses “Esc” from keyboard.

Success Scenario Event Flow:

2. Player selects “Settings” button from main menu.
3. System displays settings.
4. Player mutes the game by pressing “mute” button.
5. System mutes the game.
6. Player changes the input keys.
7. System changes input keys for input controller.
8. Player saves the changes by pressing the “submit” button.
9. System mutes the game and changes the input keys.

Alternative Event Flows:

1. If Player selects the button to return main menu:
    a. Player presses “Esc” from keyboard.
    b. System displays main menu.

2. If Player selects the button to change settings:
    a. Player selects “Settings” from Pause menu.
    b. System displays settings.
    c. Player changes volume by using volume bar.
    d. System gets the new volume and waits for submission.
    e. Player changes the input keys.
    f. System gets the input keys and waits for submission.
    g. Player saves the changes by pressing “submit” button.
    h. System applies the changes.
3. If Player selects the button to return pause menu:
    a. Player presses “Exit” button from Settings menu.
    b. System displays pause menu.

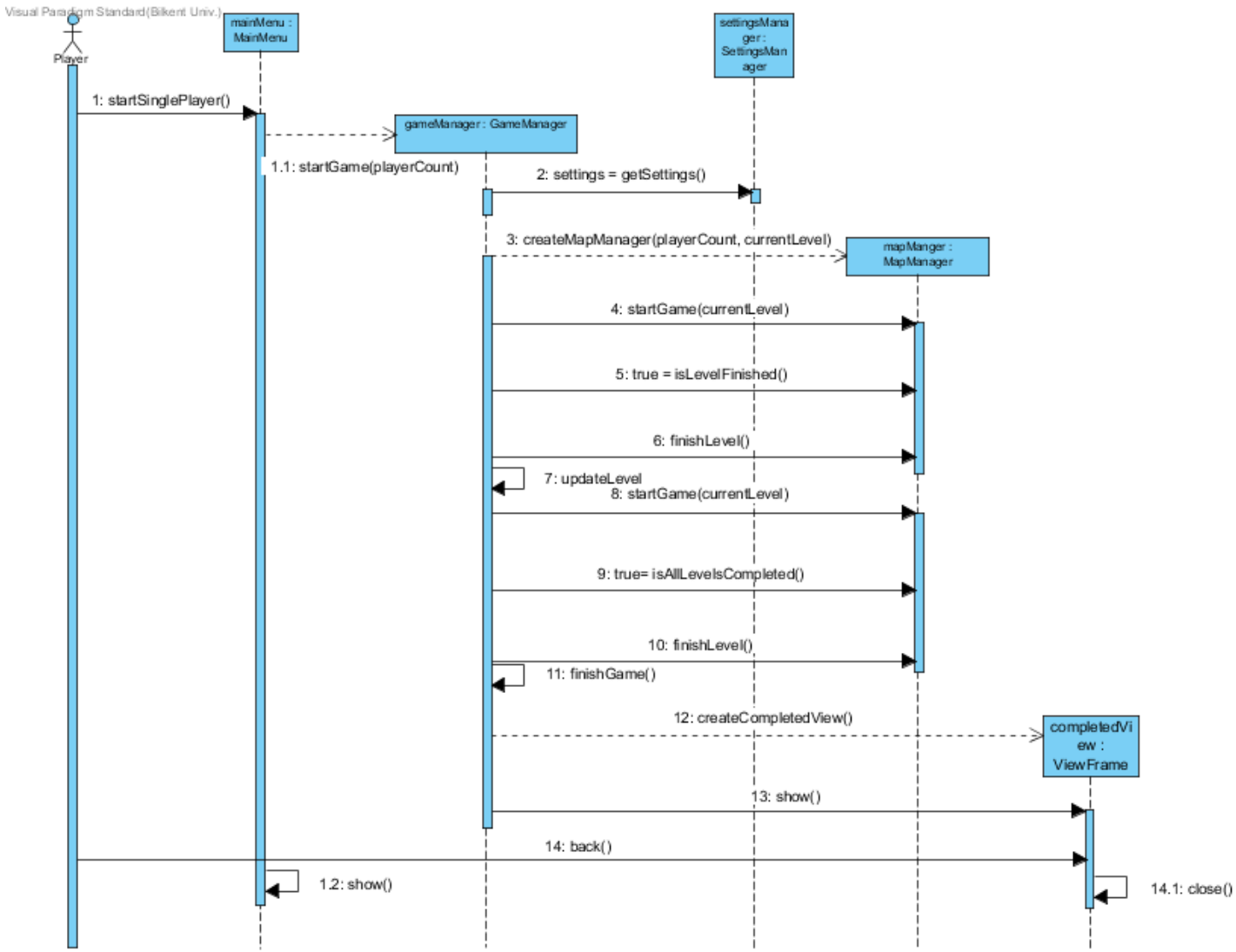
5.2 Dynamic Models

5.2.1 Sequence Diagrams

5.2.1.1 Starting a New Game

Scenario: Starting a new game

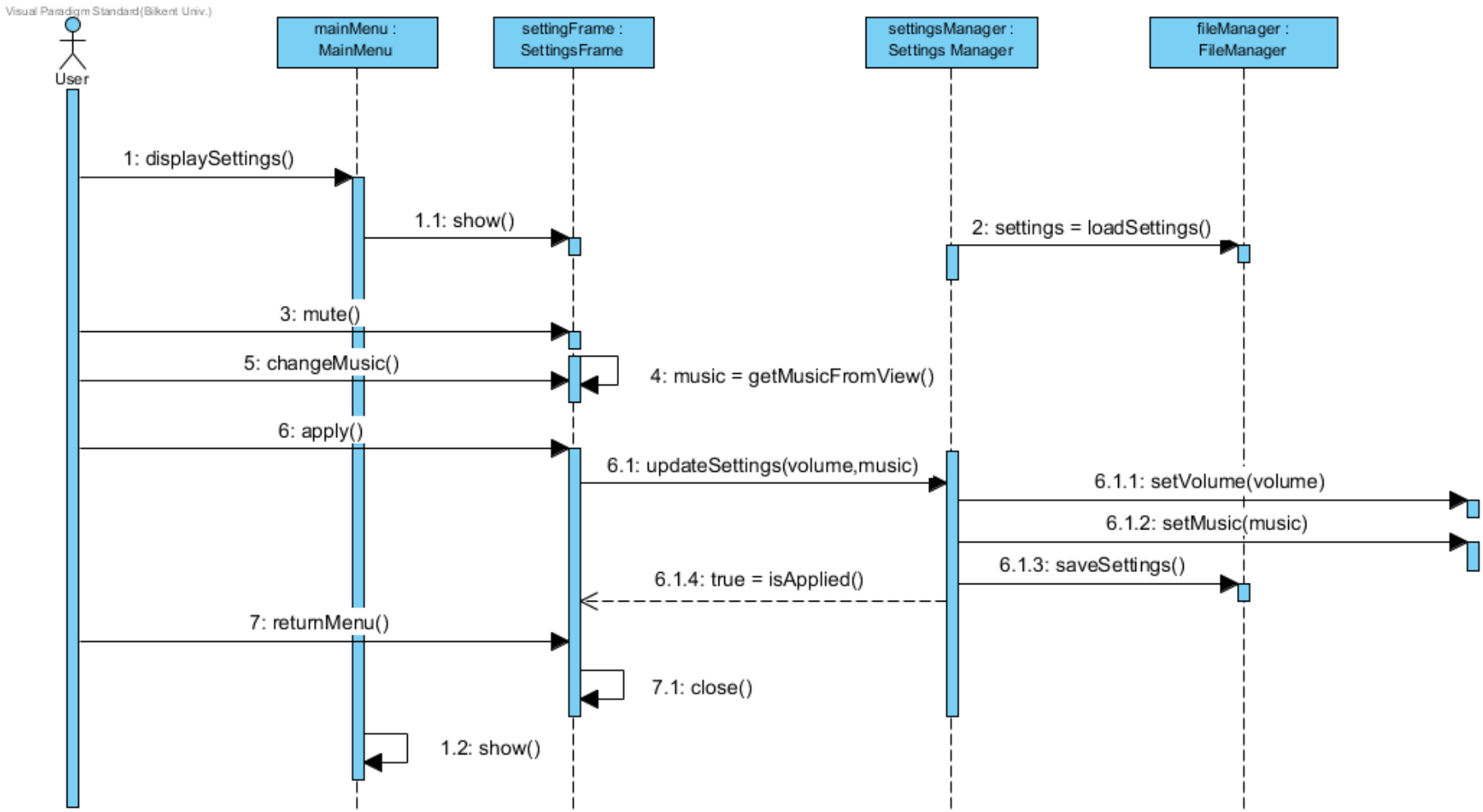
User wants to start the game. He/she enters the Main Menu and chooses one of the modes: single or multiplayer game. The game is initialized by the Game Manager. The creation of the map, the objects on the map and the level of the map is identified through the Map Manager. Gui and the window settings are created with the help of Window Manager. Sound and music of the game is initialized with the help of the Sound Manager. After creating a map, it creates the necessary obstacles with the help of the Obstacle class. The tank is created with the help of the Tank class. Player and Bot tanks are created with the help of the Player and Bot classes. Lastly bullets are created with the help of the Bullet class. All created objects: map, obstacles, tanks, bullets are inserted into the Game, and the game starts after that.


Visual Paradigm Standard(Bilkent Univ.)
Player
mainMenu : MainMenu
gameManager : GameManager
settingsMana ger : SettingsMan ager
mapManger : MapManager
completedVi ew : ViewFrame
1: startSinglePlayer()
1.1: startGame(playerCount)
2: settings = getSettings()
3: createMapManager(playerCount, currentLevel)
4: startGame(currentLevel)
5: true = isLevelFinished()
6: finishLevel()
7: updateLevel
8: startGame(currentLevel)
9: true= isAllLevelsCompleted()
10: finishLevel()
11: finishGame()
12: createCompletedView()
13: show()
14: back()
1.2: show()
14.1: close()

5.2.1.2 Changing Settings

Scenario: User enters the ‘Settings’ screen from the Main Menu

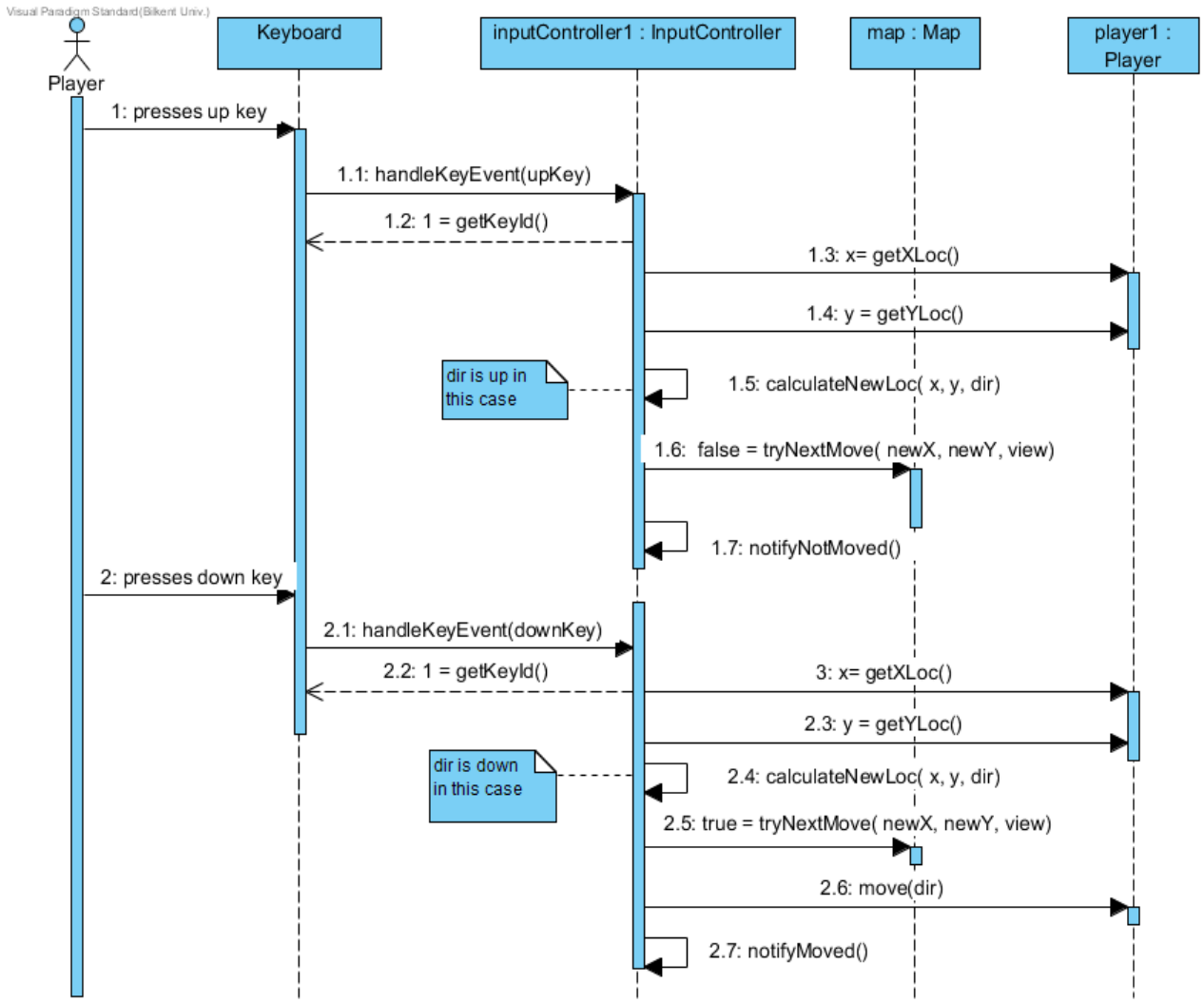
User wants to adjust the settings to his/her preferences, thus he/she enters the Main Menu. Here, if the user does not want to do any changes, he/she returns to Main Menu. Else, the user changes the settings with getNewSettings() method, managed by Settings Manager. Later changed settings are being applied with applyNewSettings(settings). The new adjusted music and volume will be managed by Sound Manager, later it will be set in GameManager with the help of setMusic(music) and setVolume(volume) methods.


Visual Paradigm Standard(Bilkent Univ.)
User
mainMenu : MainMenu
settingFrame : SettingsFrame
settingsManager : Settings Manager
fileManager : FileManager
1: displaySettings()
1.1: show()
2: settings = loadSettings()
3: mute()
5: changeMusic()
4: music = getMusicFromView()
6: apply()
6.1: updateSettings(volume,music)
6.1.1: setVolume(volume)
6.1.2: setMusic(music)
6.1.3: saveSettings()
6.1.4: true = isApplied()
7: returnMenu()
7.1: close()
1.2: show()

5.2.1.3 Moving Player's Tank.

Scenario: User wants to move

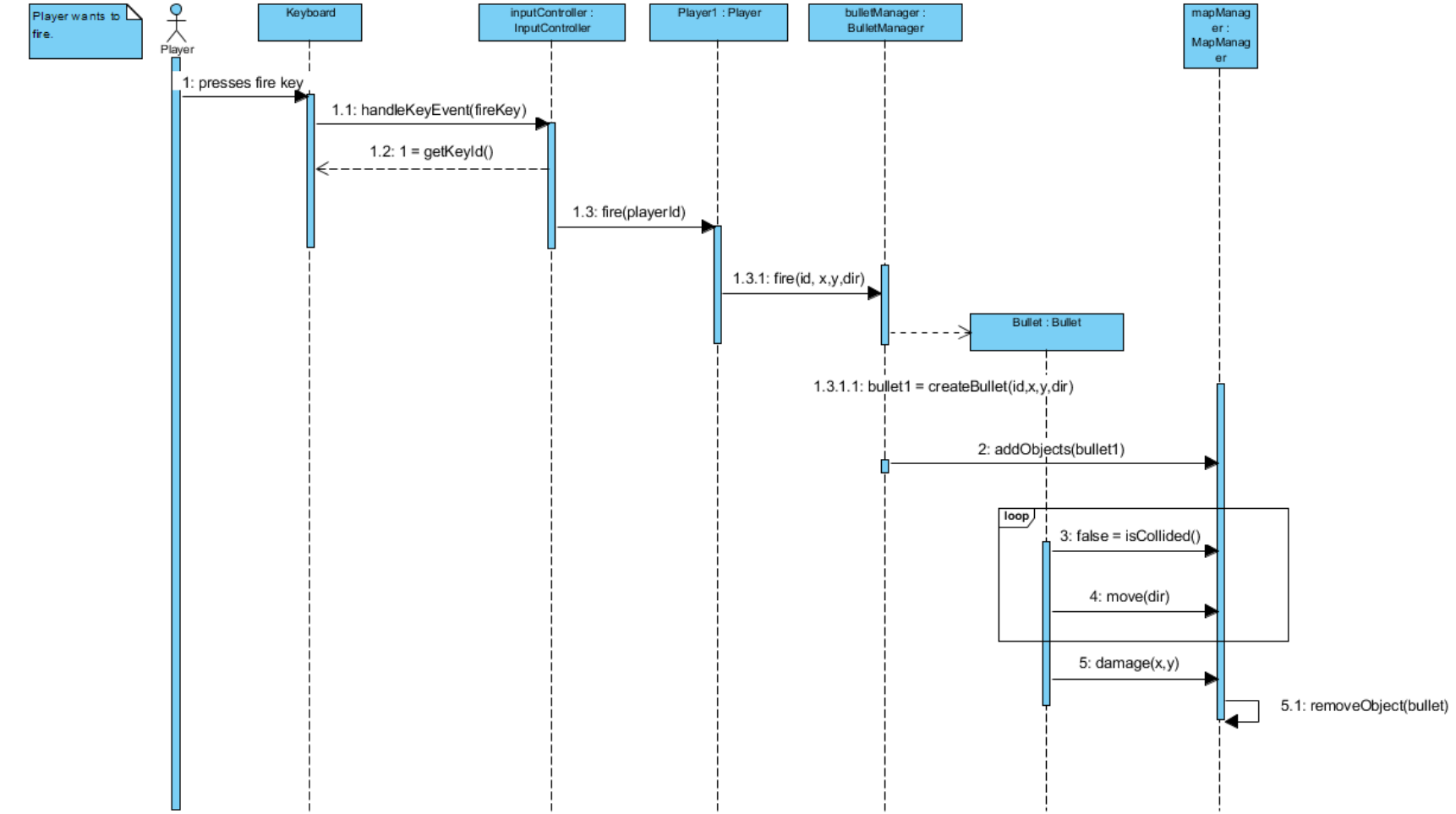
The user wants to move and presses the key from Keyboard. The command of Keyboard calls the move() method from Tank Manager class in order to apply move instruction. Player class gets the (x,y,dir) and the Id (in order to determine which user wants to move) inputs from Tank Manager class. Towards the end of the movement process Player class calls move(up) (with checking isPassable()) to update the map by Map Manager.



5.2.1.4 Fire Interaction

Scenario: User wants to shoot

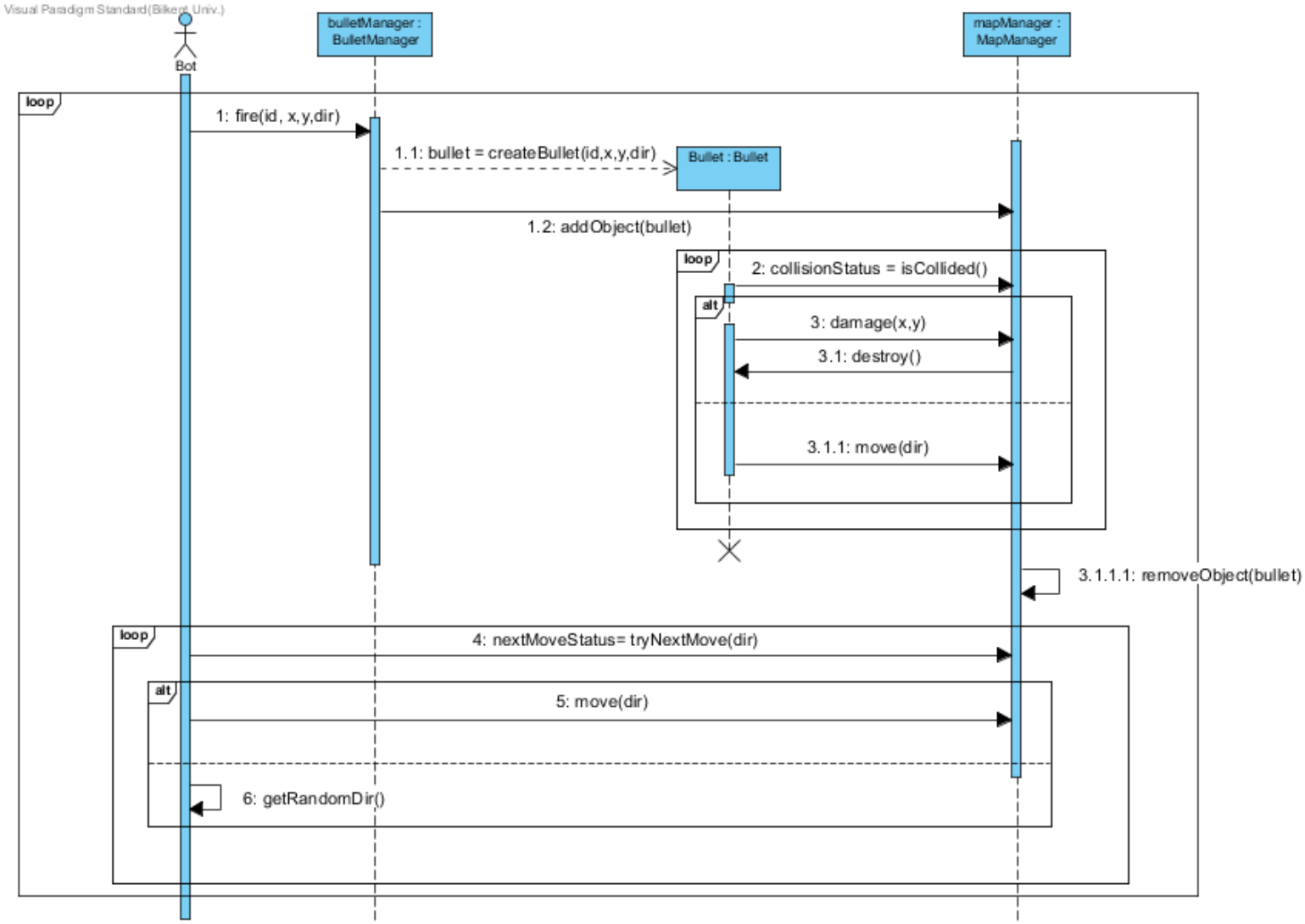
User wants to shoot and gets the (x.y) coordinates and the direction with corresponding methods. The bullet is created with these coordinates and direction by Bullet Manager. After that Bullet class gives addObjects(botBullet) and move(x,y,dir) (with checking if it is passable or not) commands to Map Manager to update the map. Besides, it gives info about if the obstacle is destructible or not and gives removeObjects(botBullet) (if it is destructable) to update the map again and all these instructions will repeat in a loop to check and update the map sequentially.



5.2.1.5 Bot Move and Fire Mechanism

Scenario: The scenario of bot instructions

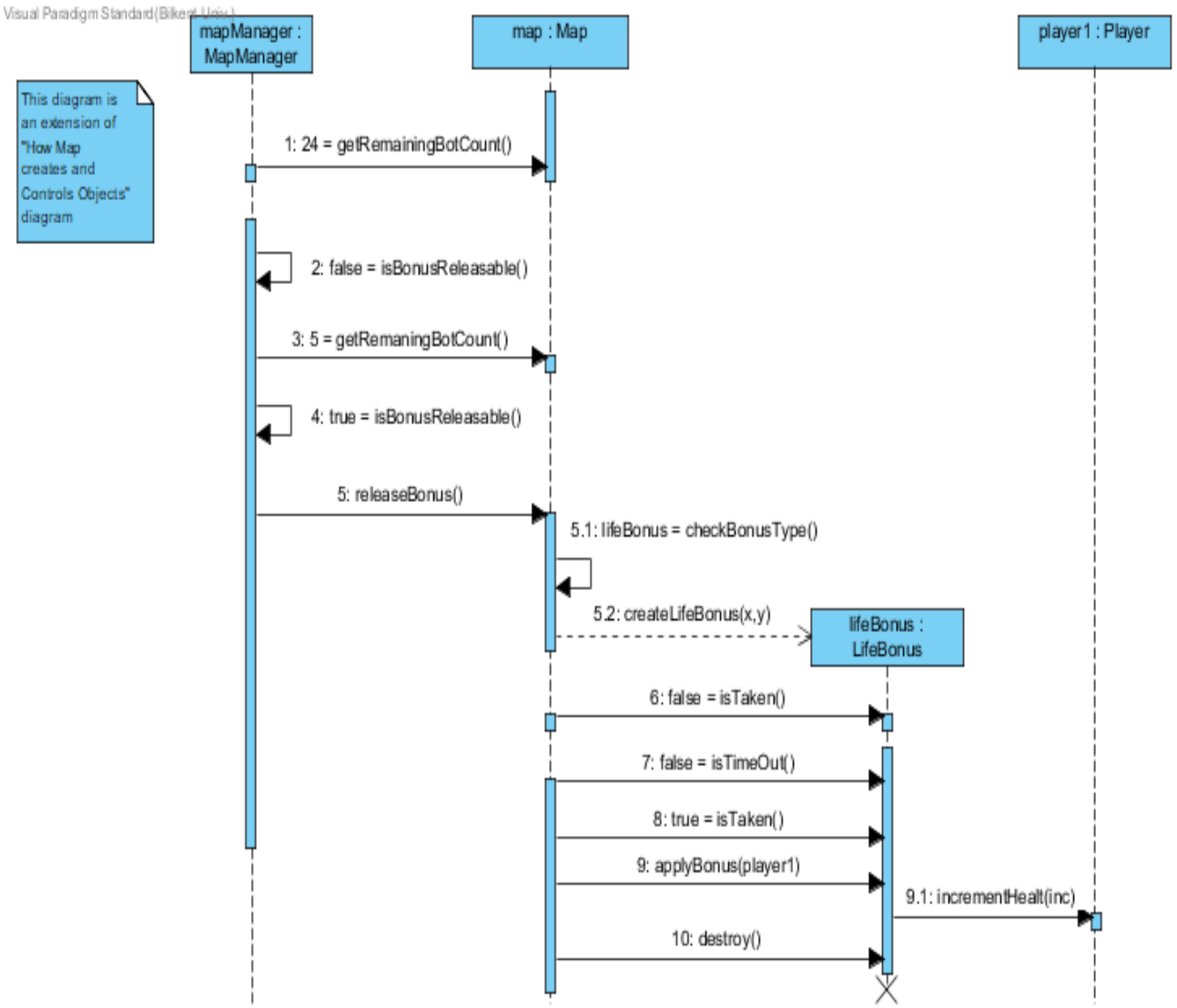
Bot wants to move and gives the move(dir) command with checking isStuck() or isMonotone() functions and by the output of these functions Map Manager updates the map. If the Bot wants to shoot Bullet Manager takes the (x,y,dir) inputs and gives to Map Manager via Bullet class.


Visual Paradigm Standard(Bilkent Univ.)
Bot
bulletManager : BulletManager
Bullet : Bullet
mapManager : MapManager
loop
1: fire(id, x,y,dir)
1.1: bullet = createBullet(id,x,y,dir)
1.2: addObject(bullet)
loop
2: collisionStatus = isCollided()
alt
3: damage(x,y)
3.1: destroy()
3.1.1: move(dir)
3.1.1.1: removeObject(bullet)
loop
4: nextMoveStatus= tryNextMove(dir)
alt
5: move(dir)
6: getRandomDir()

5.2.1.6 Bonus Release

Scenario: Bonus is released

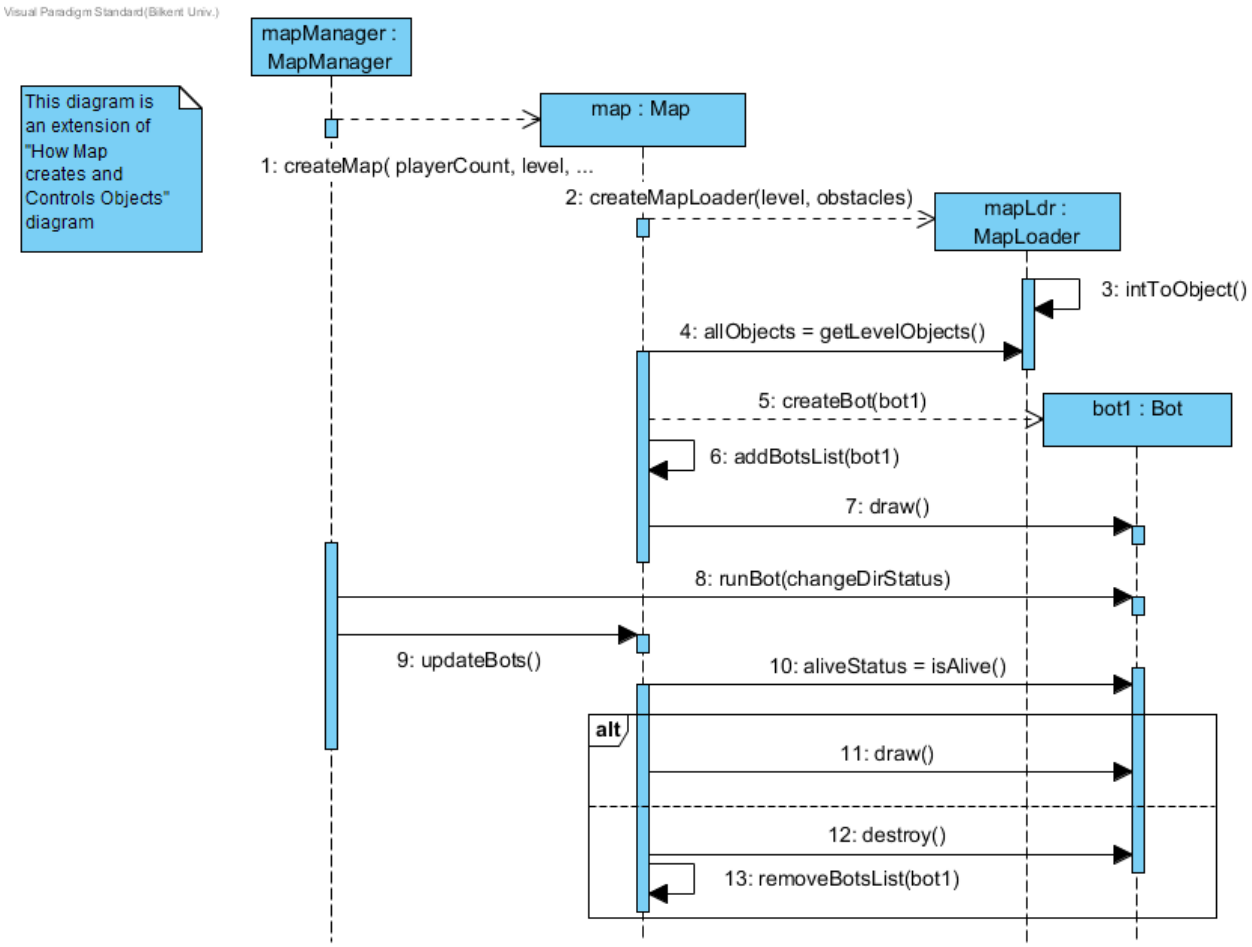
First, Map checks if there are any bots remained. If yes, then map releases the bonus, since the game have not finished yet. After the release, if the player tank passes over the bonus, the life score in Player class is being updated.



5.2.1.7 Addition and Deletion of bots

Scenario: Bot tanks are added or deleted

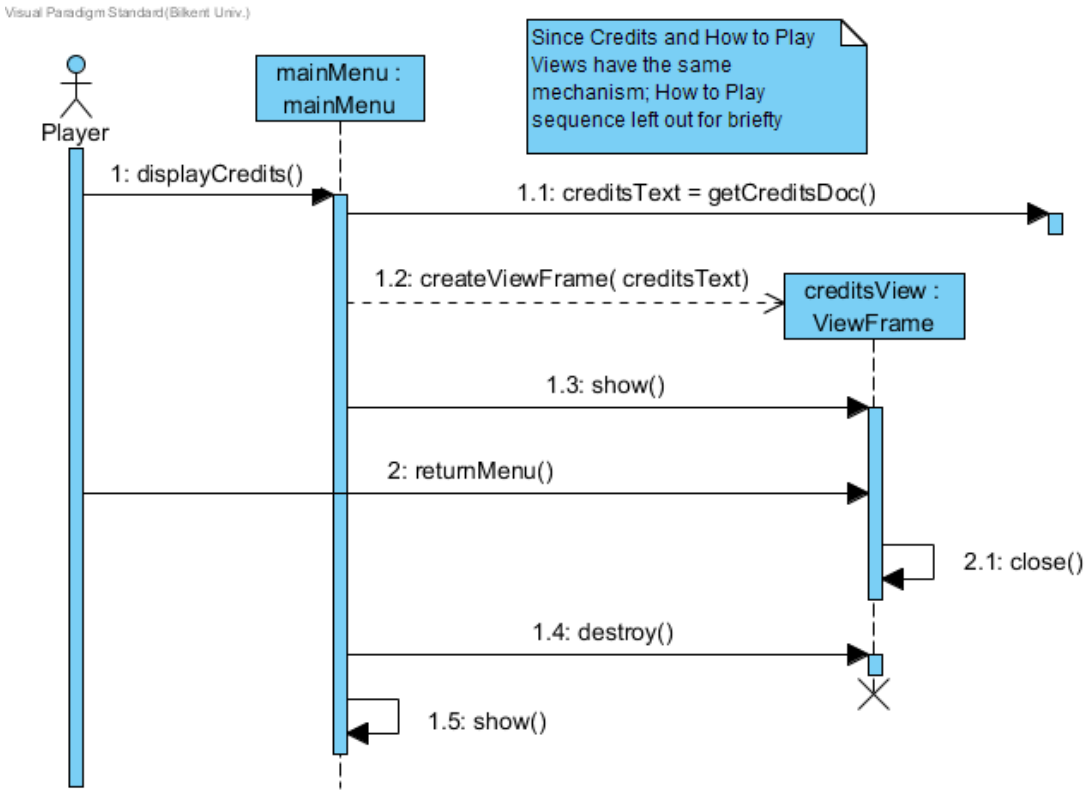
Bot tanks are created with the help of MapManager and Map. First, MapManager sends the information about the map style, namely, if it is going to be single or multi player game. Then, Map loads the level and creates the obstacles on the map. After the creation of the map, Map loads the bots with the help of draw() method of the Bot class. If the bot tank was destroyed, it is being destroyed with the destroy() method.



5.2.1.8 Display of Credits/How to Play frame

Scenario: Credits/How to Play frame is displayed.

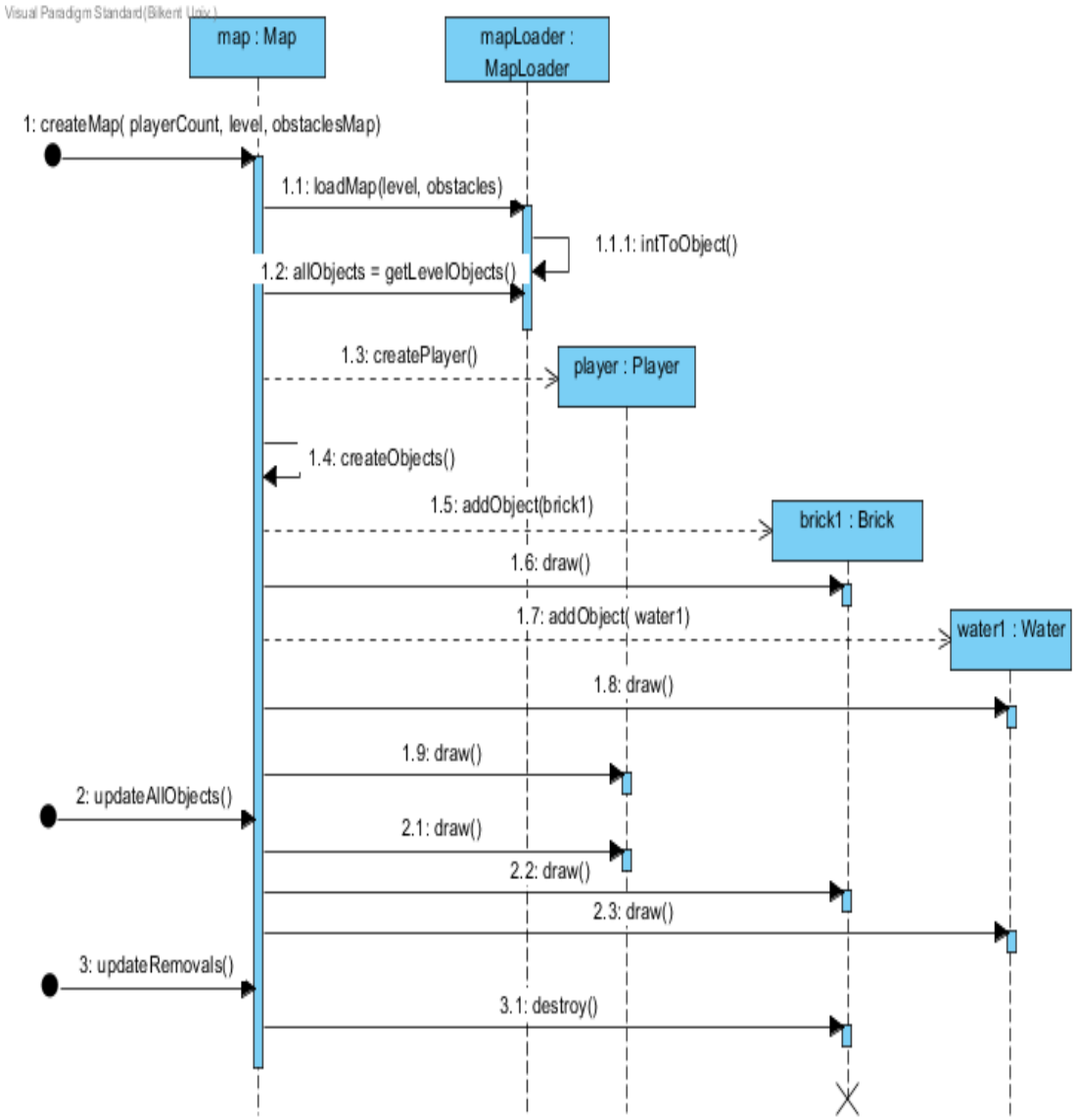
In the Menu class, there is an instance of the ViewFrame class, which provides the interface with the Credits and HowtoPlay frames. When the user clicks the buttons (either Credits or How to Play), the Credits frame loads from the ViewFrame, and the content of the frame is loaded from the according text file, which is being stored in an ArrayList<String>. Then the ViewFrame class, fills the frame.



5.2.1.9 Control of GameObjects by Map

Scenario: Map controls GameObjects.

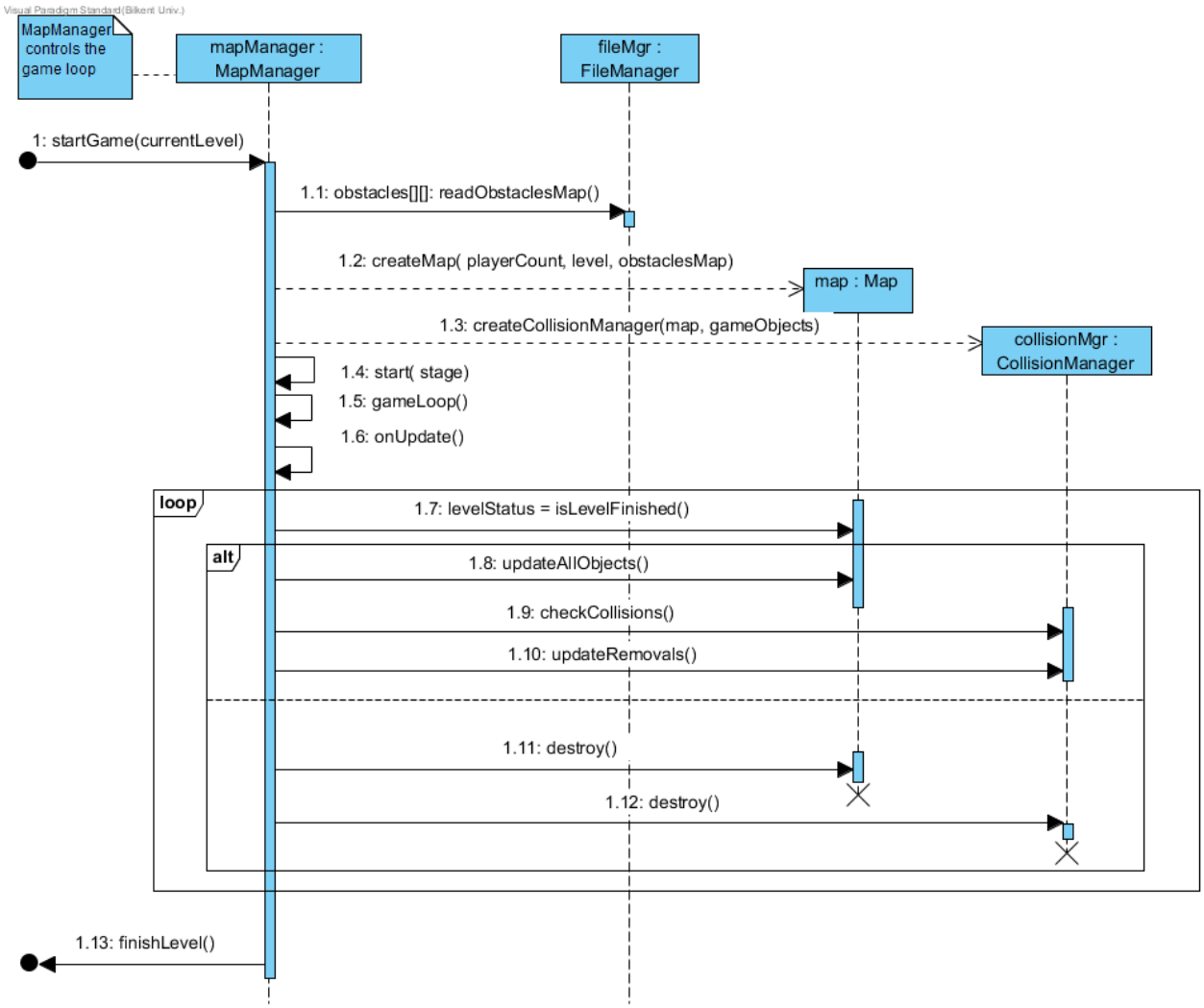
Each game object class as Brick, Bush etc. has a draw() method, which helps the MapLoader class to load the objects to the map.



5.2.1.10 Control of Map by MapManager

Scenario: MapManager is controlling Map.

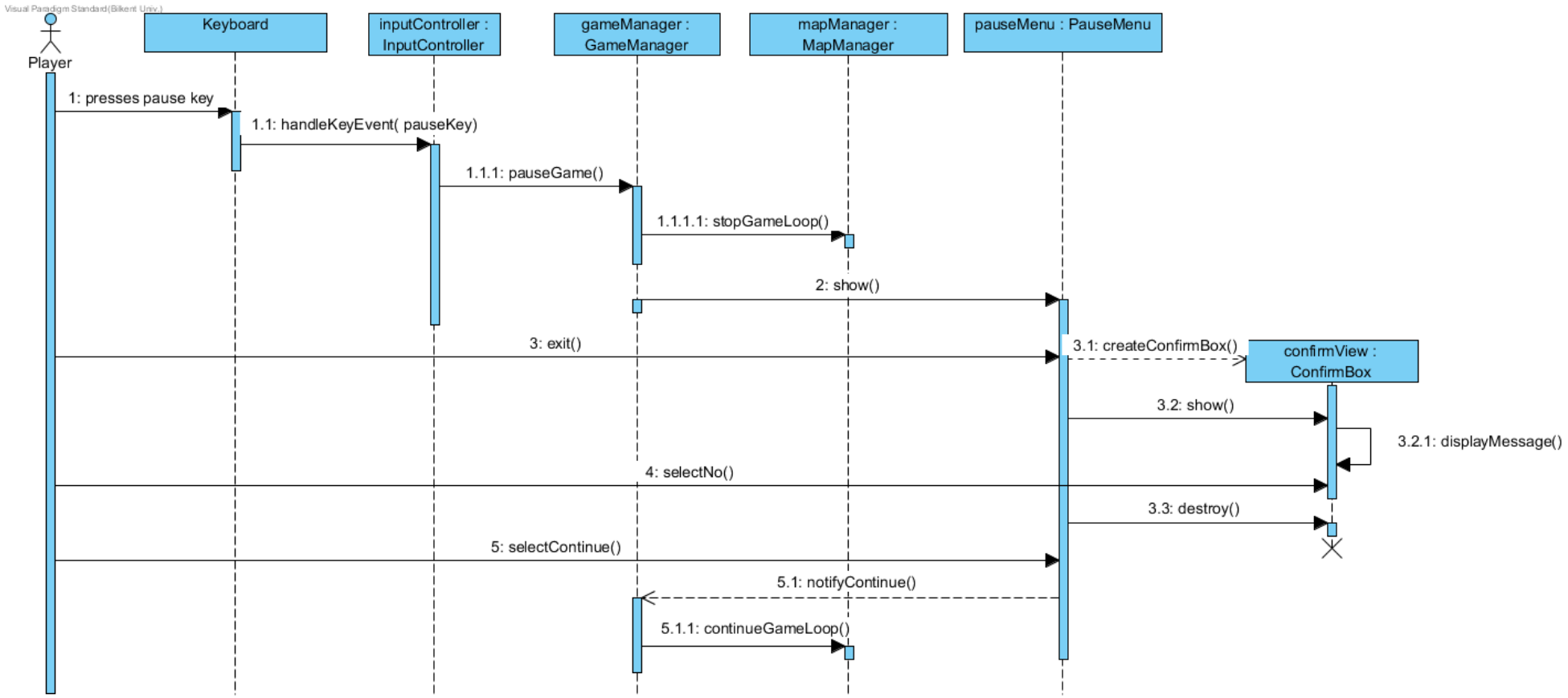
MapManager loads the level information from the folder, via FileManager class. Each time when the object is moving or is destroyed, MapManager updates the Map.



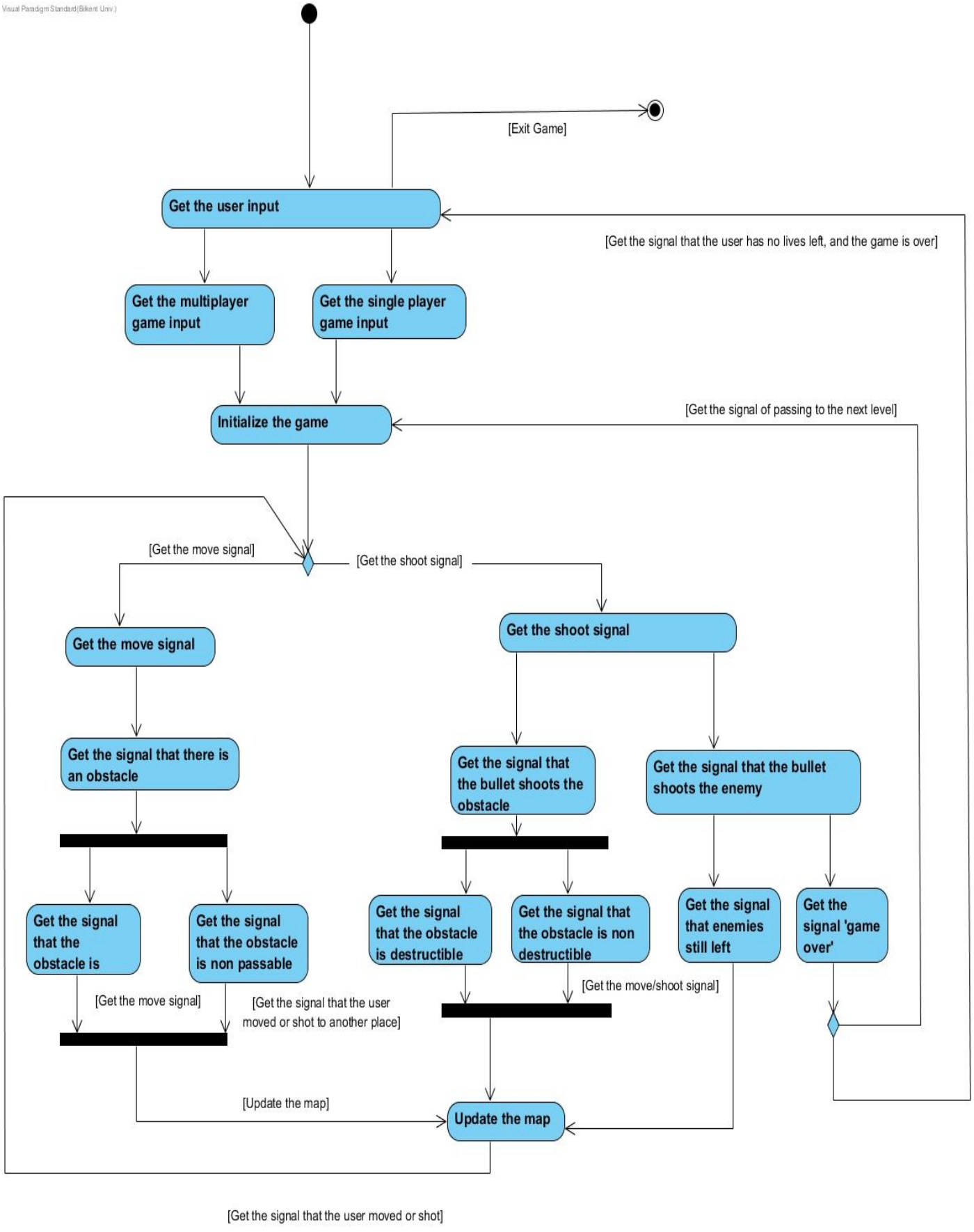
5.2.1.11 Pause/Continue Game

Scenario: User pauses/continues the game**.**

GameManager class checkes whether the game is finished, continuing or paused. When the pause button is pressed, PauseMenu appears on screen. The user may exit or continue the game. When the user presses exit, the ConfirmBox frame is appeared.


Visual Paradigm Standard(Bilkent Univ.)
Player
Keyboard
inputController : InputController
gameManager : GameManager
mapManager : MapManager
pauseMenu : PauseMenu
confirmView : ConfirmBox
1: presses pause key
1.1: handleKeyEvent(pauseKey)
1.1.1: pauseGame()
1.1.1.1: stopGameLoop()
2: show()
3: exit()
3.1: createConfirmBox()
3.2: show()
3.2.1: displayMessage()
4: selectNo()
3.3: destroy()
5: selectContinue()
5.1: notifyContinue()
5.1.1: continueGameLoop()

5.2.2 Activity Diagram



The activity diagram above, illustrates how system runs the game.

Activity diagram explanation:

In the beginning, the system waits the user to input. At this point, user may start the game, or exit it. When the user exits, system goes to the final node. If the user starts the game, it can be either multiplayer or single player game. In both cases, the next thing the system does is to initialize the game. After the initialization, the game starts.

When the game stars, the user may either move his/her tank, or start shooting right away. Therefore, in Activity diagram, the movement and the shooting processes are divided into two activities.

First case: the user moves. When the user moves, the system checks if there are any obstacles. If there are obstacles, the system check if they are passable or non-passable. In both cases, the tank moves, in other words, the object on the map changes its position, direction or place. The system updates the map first and then, the user may move or shoot again. So the process repeats this way.

Second case: the user shoots. When the user shoots, the system should check if the tank's bullet hits any obstacle or enemy.

Case 1: the bullet hits the obstacle. If the bullet hits the obstacle, the system checks if the obstacle is destructible or non-destructible. In both cases, bullet moves, changes position and place. Especially, if the obstacle is destructible, the bullet destroys it. Thus, the system updates the map again.

Case 2: the bullet hits the enemy. When the bullet hits the enemy, the system checks if there are any enemies left.

Case 2.1: If there is no enemy left. If the user's tank has enough life span to pass to the next level, the next level game initializes. If the user's tank has no life left, the game is over, thus the system waits for the user input to start a new game.

Case 2.2: If there is enemies left. The game continues, the shot enemy tank disappears. Thus, the system updates the map.

The system continues this process until the user exits the game.

5.3 Object and Class Model

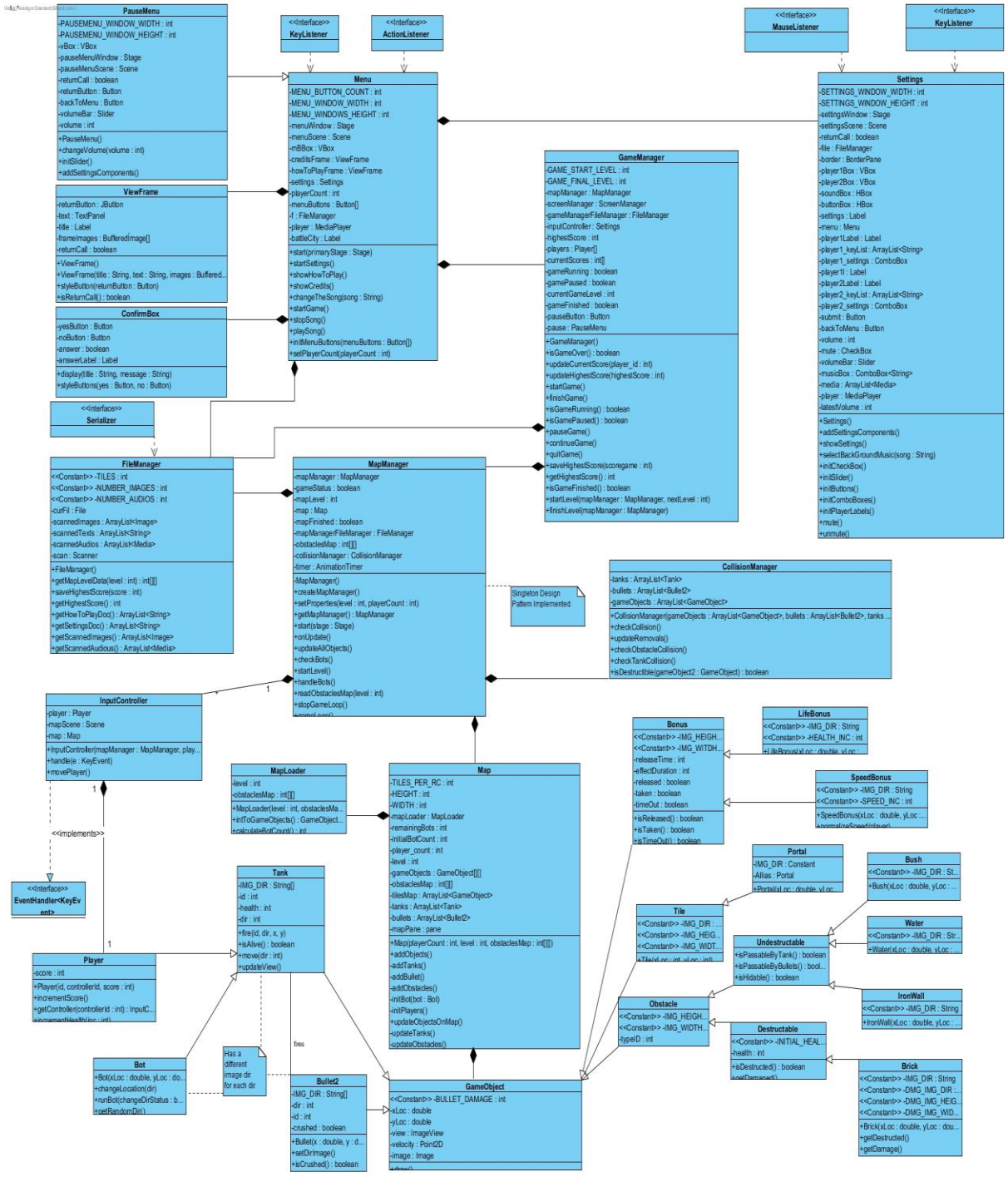


Figure 1-Complete Object Diagram

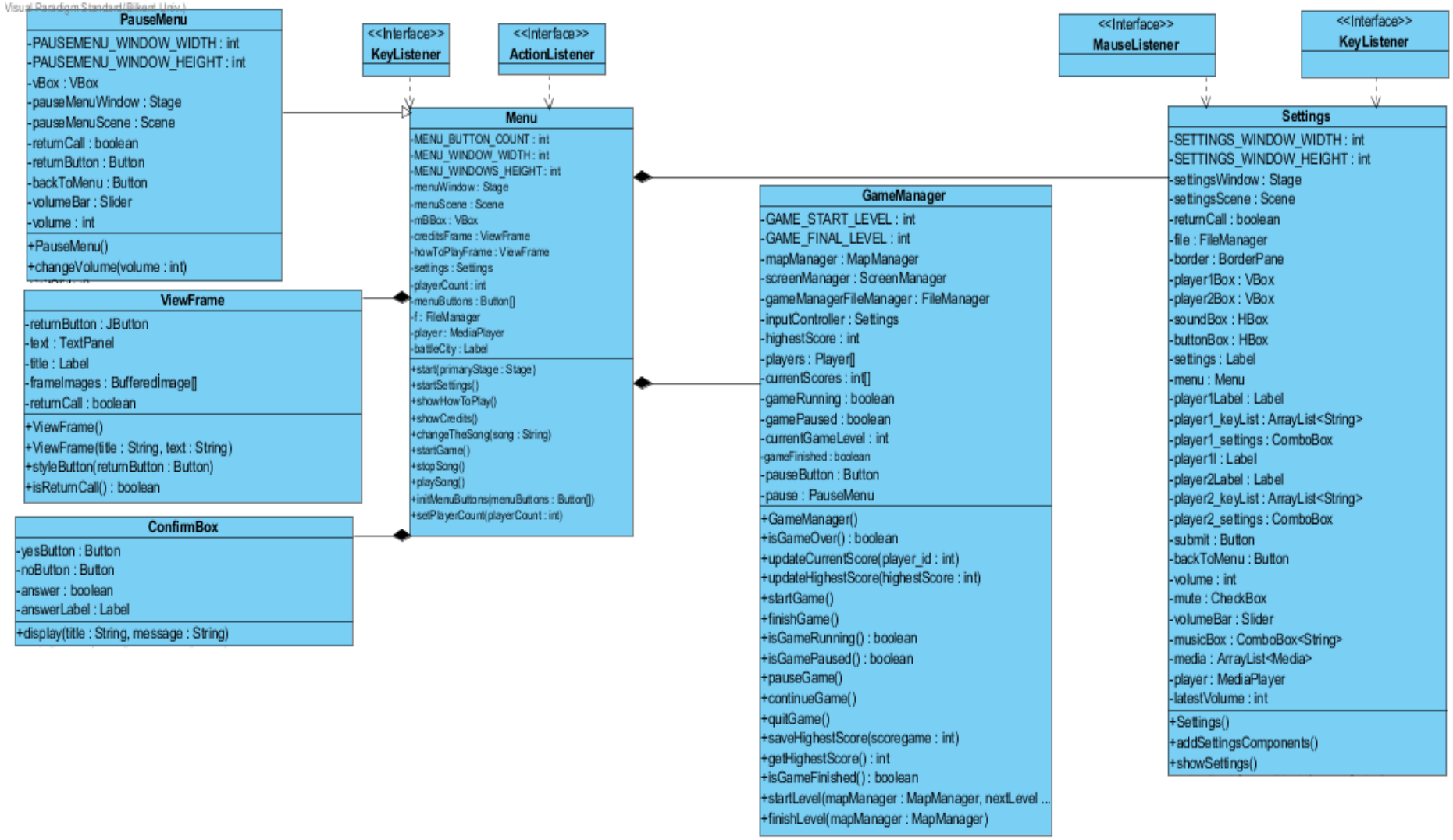


Figure 2-Detailed Object Diagram 1

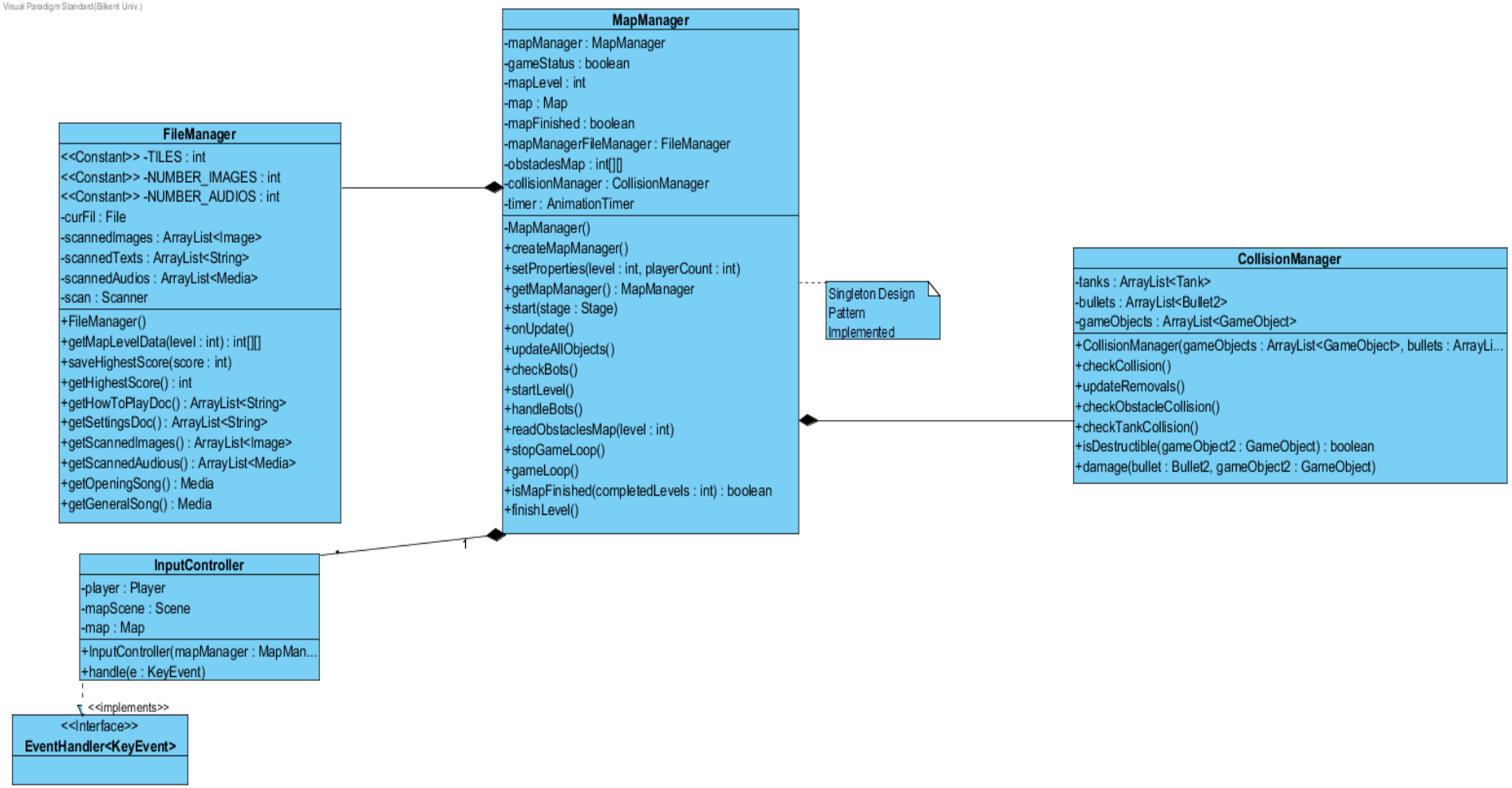


Figure 3-Detailed Object Diagram 2

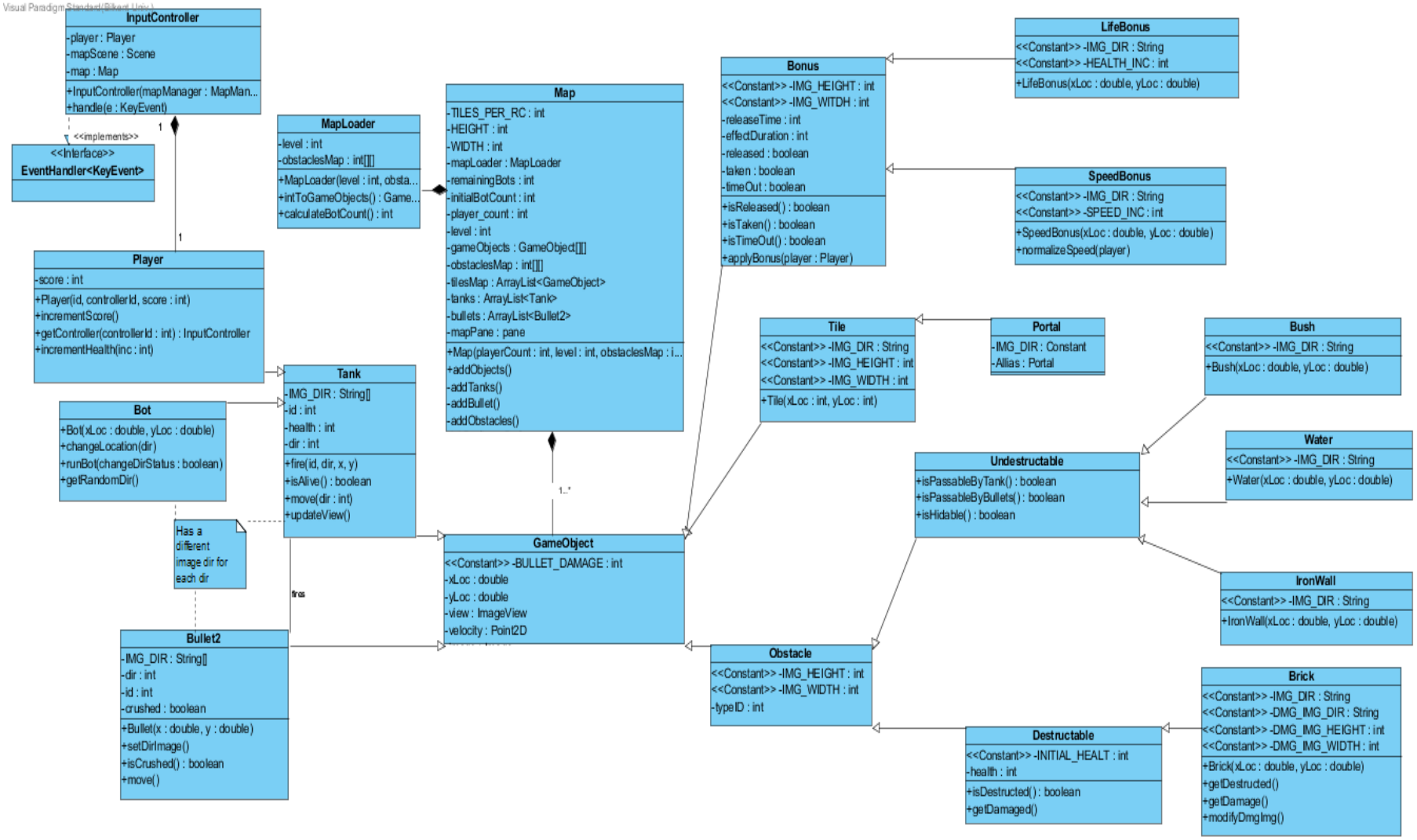


Figure 4-Detailed Object Diagram 3

The class diagram of “Battle City” game illustrated above. Currently, the class diagram has 30 classes.

At the beginning, we have Menu class and its two subclases MainMenu and PauseMenu. Basically, purpose of the Menu classes is to provide user a interface. MainMenu starts the game when user presses “Play” button and then GameManager initalizes the Map Manager class. Moreover, “PauseMenu” class is the user interface class for “pause” action of the game.

After the Menu classes “Battle City” has its functional classes which are GameManager class, InputManager class, SoundManager class, MapManager class and CollisionManager class.

GamaManager class:

Basically, this class gets the settings of the game and inializes a new game. Meaning that, it gets the user input keys, user sound settings and initalizes a new game from level 1. GameManager class starts the game according to input coming from Main Menu.

InputController:

In “Battle City” game, input manager has two purposes:

1. Keeping the user input key settings.
2. Reporting the pressed key information to the game, so that locations of the players can be updated.

Settings Class:

This class keeps the sound settings. These sound settings are music settings and volume settings. User can change the volume and music of the game by using this class.

MapManager Class:

This class is one of the most important functional class of the “Battle City” game. This class inializes a new map when user starts the game and keeps track of the game data. Moreover, this class has gameLoop method which is a method runs the game. According to the data coming from the CollisionManager, the MapManager class may request Map class to update itself and object status(one object may have been destroyed). If one level is finished, MapManager initalizes the next level. If the last completed level is the last level of the game,MapManager finished the game.

CollisionManager Class:

This class checks if there is collision between a bullet and a game object. If there is it checks whether the collided game object is destructable or not. Finally, if it is destructable it request game object to be damaged. CollisionManager simultineusly works with MapManager class.

After discussing about functional classes of the game, now classes that provides instances and objects of the will be disscussed.

On the top of the instance classes, “Battle City” has GameObject class.

GameObject Class:

Basically this is the main class which gathers all the common attributes of the game objects. The main iade is to have a easy control over the game object during game time. This class has 2 main attributes which all game objects have:

1. X cordinate of the object
2. Y cordinate of the object

Moreover, Game object class has 3 subclasses which extend it.

Tank Class:

Tank class is a subclass of GameObject class. It is defined for the tanks of the games. Basically, there are two types of tanks in the game:

1. The tanks which are controlled by players.
2. The tanks which are bots.

These two types of tanks can both fire bullets, they both have id, health and direction.

Player Class:

Player class is a subclass of Tank class. It is designed for the tanks which are controlled by players. Since these tanks are controlled by players, they have a set of input keys which provides control over tanks. These tanks are also have score which increments when they destroy a bot tank.

Bot Class:

This is an another subclass of Tank class. It is designed for bot which are controlled by game bot algorithm.

Bullet Class:

Bullet class is a subclass of GameObject class. Basically, every tank in the game can create instances of this class. According to the location and direction of the tanks, tanks create bullets and fire them. Bullet moves according to the given direction. If it hits an game object, CollisionManager class checks the type of the collision and decides what to do next according to the colllision type.

Obstacles:

Obstacles class is a subclass of GameObject class and it is defined for obstacles of the game. Basically, in “Battle City” game there are different type of obstacles. According to the common attributes of these obstacles, we have divided obstacles to different subclasses.

Destructable:

This class is a subclass of Obstacles class. It is defined for the obstacles which can be destructed by bullets.

Brick:

This is a subclass of destructable obstacles. Basically, each brick can be destructed by two bullets of the game. When first bullet hits, the brick loses half of its health and also size. When second bullet hits it gets destructed and removed from the game map.

Undestructable:

This class is a subclass of Obstacles class. It is defined for the obstales which cannot be destructed by bullets. In the “Battle City” game, there are 3 types of undestructable obstacles.

Water:

Water is one of the subclasses of undestrubtable obstacles. Basically, no tank can destroy these objects by firing bullets however bullets can pass through water objects. On the other hand, tanks cannot pass through water objects.

IronWall:

IronWall is one of the subclasses of undestrubtable obstacles. Basically, no tank can destroy these objects by firing bullets and no bullet can pass through them. Also, no tank can pass through these objects by moving.

Bush:

Bush is one of the subclasses of the undestrubtable obstacles. Basically, no tank can destroy these objects by firing bullets. However, bullets can pass through these objects and also tanks can pass through these objects. When a tank goes into a bush, it hides itself from other tanks and becomes invisible in the map.

Portal:

Portal class will help to create the portal on the map, from where the player tank will be able to pass and transport through the map,

Bonus:

Bonus class is the class for creating the Bonus object in the game. By passing over the bonus, user will be able to either increase life or speed.

LifeBonus:

LifeBonus is one of the subclasses of Bonus class, which will increase the life of the player tank during the game, if the player tank passed over the life bonus.

SpeedBonus:

SpeedBonus is one of the subclasses of Bonus class, which will increase the speed of the player tank during the game, if the player tank passed over the speed bonus.

PauseMenu:

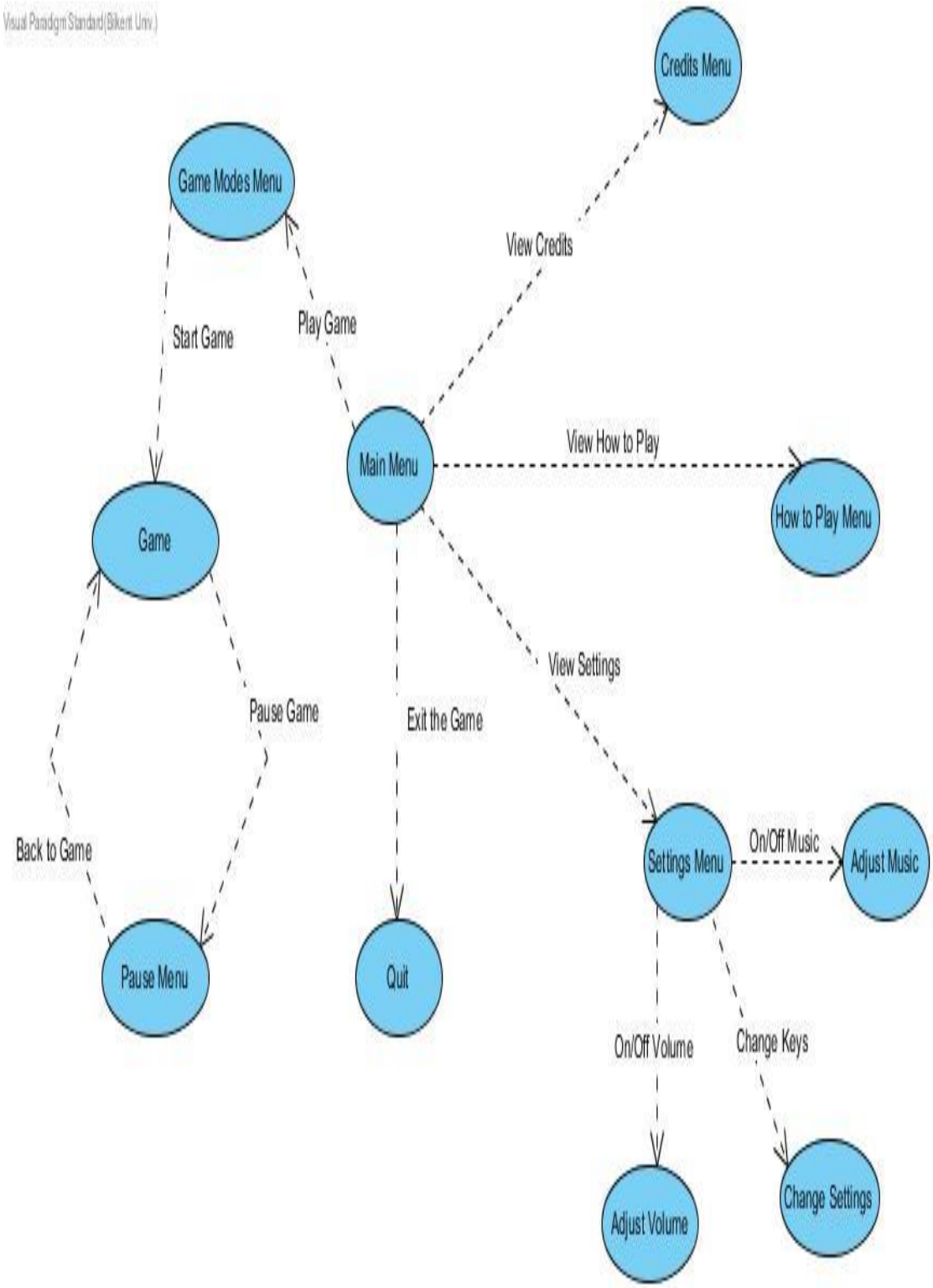
PauseMenu class provides the pause menu frame for the game, when the pause button is pressed. In this frame, user will be able change the settings, quit the game or return to the game.

MapLoader:

MapLoader class is the helper class of the Map class, which keeps track of the objects; if they are destroyed, if they are moving, how many tanks left etc.

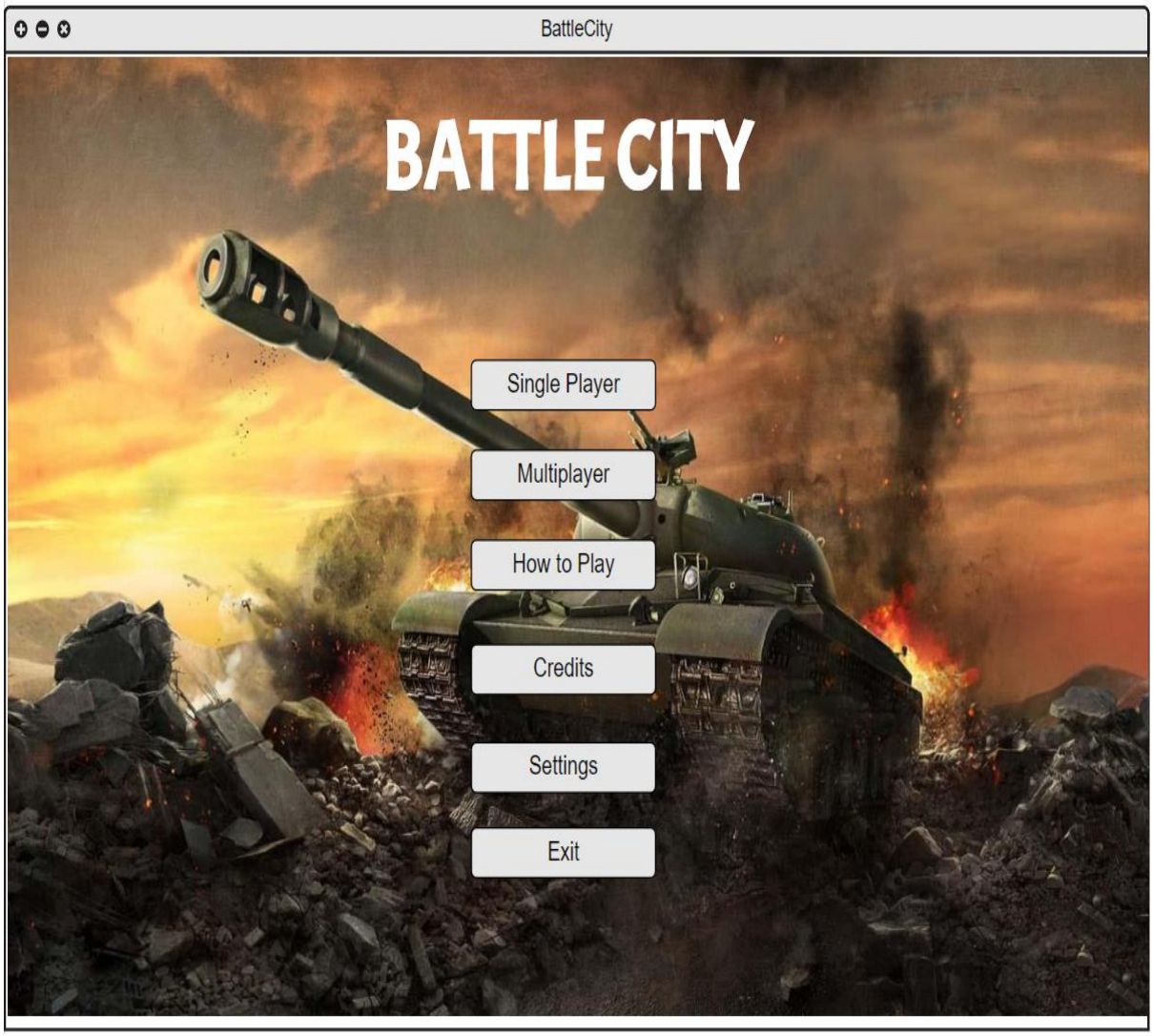
5.4 User interface – navigational paths and screen mock-ups

5.4.1 Navigational Path



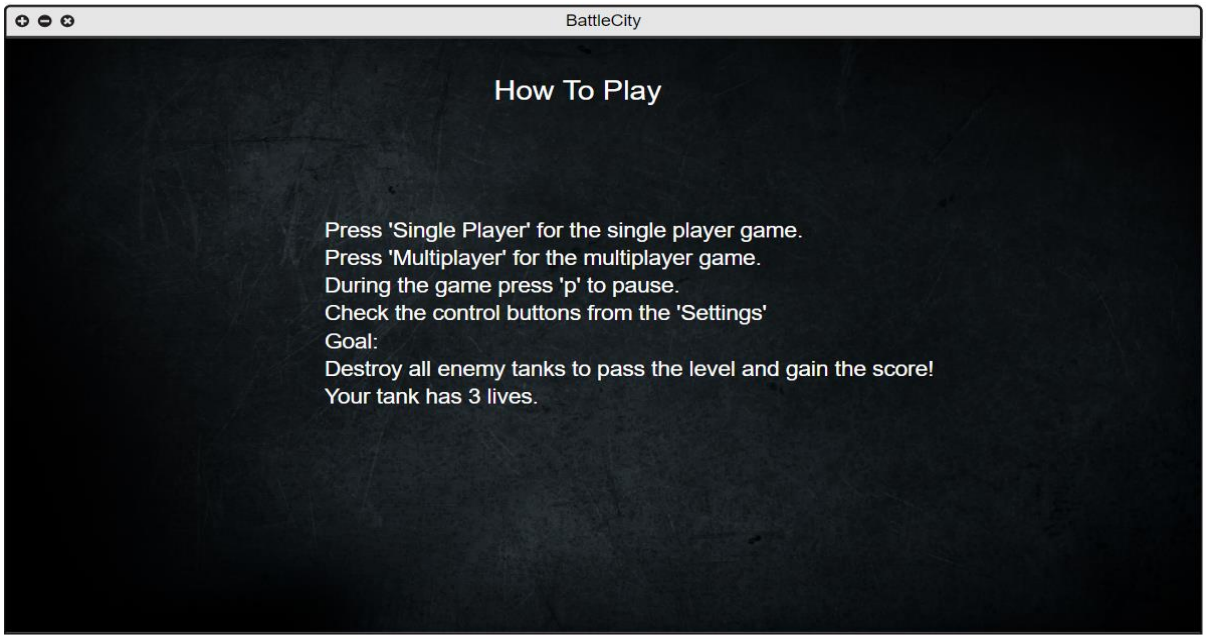
5.4.2 Screen Mock-ups

5.4.2.1 Main Menu



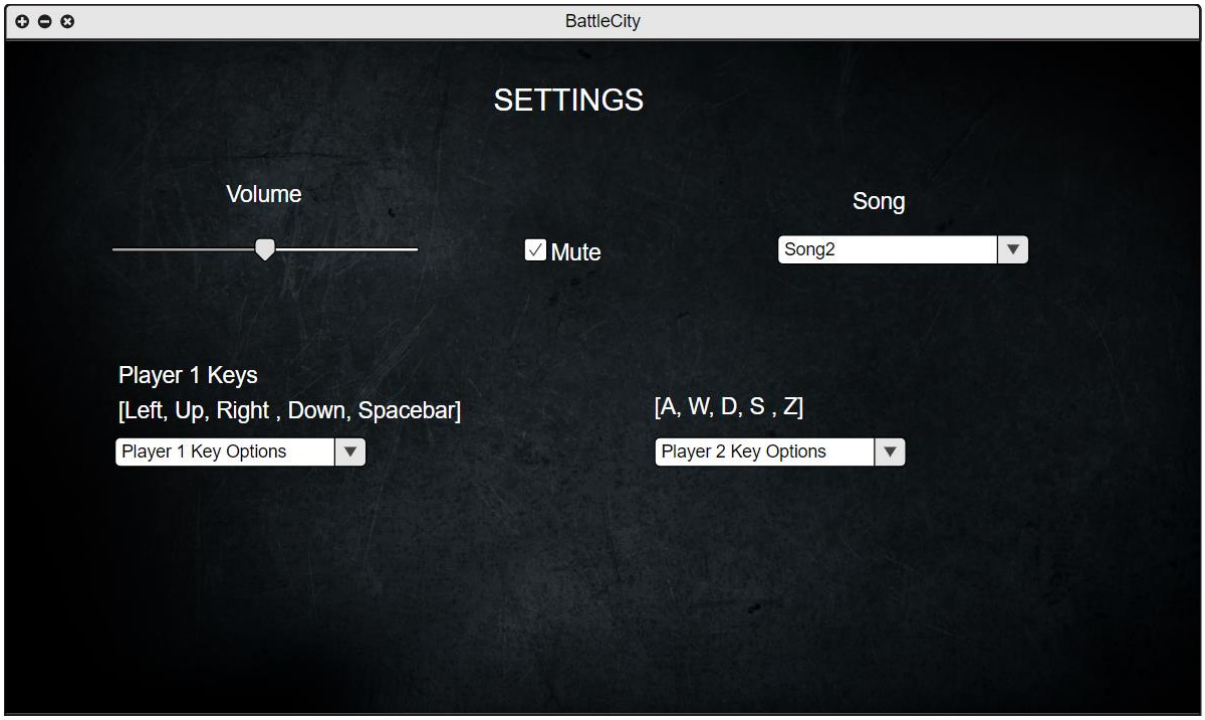
Main Menu is the start menu of the game, which consists of 6 buttons; 1 player, 2 player, how to play, settings, credits and quit. If the user chooses 1 player game, the single player game initializes, if the user chooses 2 players, multiplayer game initializes.

5.4.2.2 How to Play



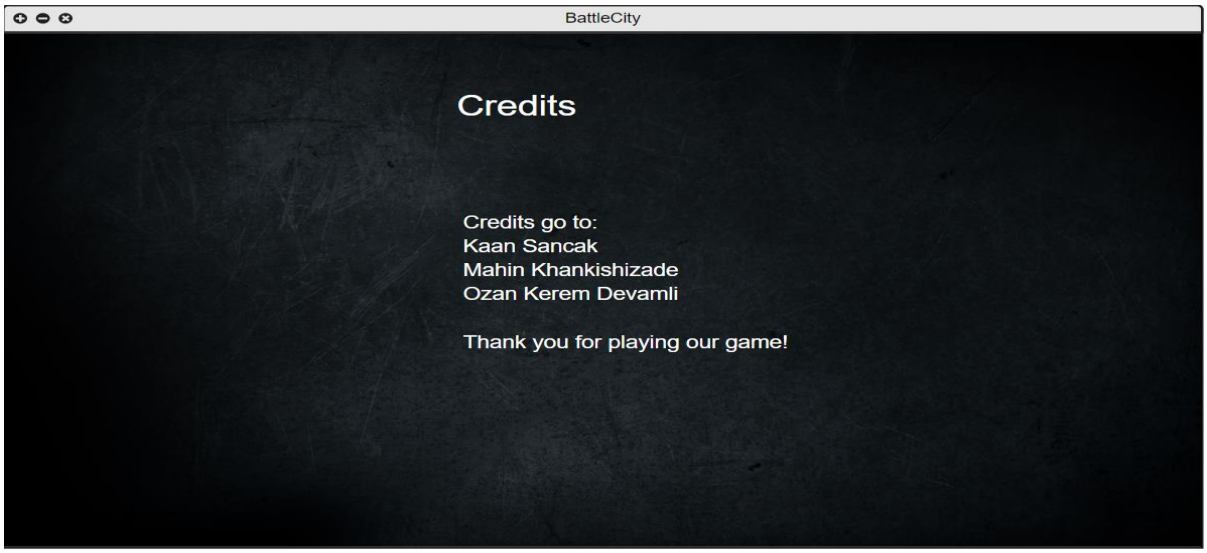
“How to Play” screen provides the users with the information about buttons, rules, controls etc. As it is seen from the figure below, the user gets notified about the objective of the game, the pause button and the control buttons.

5.4.2.3 Settings



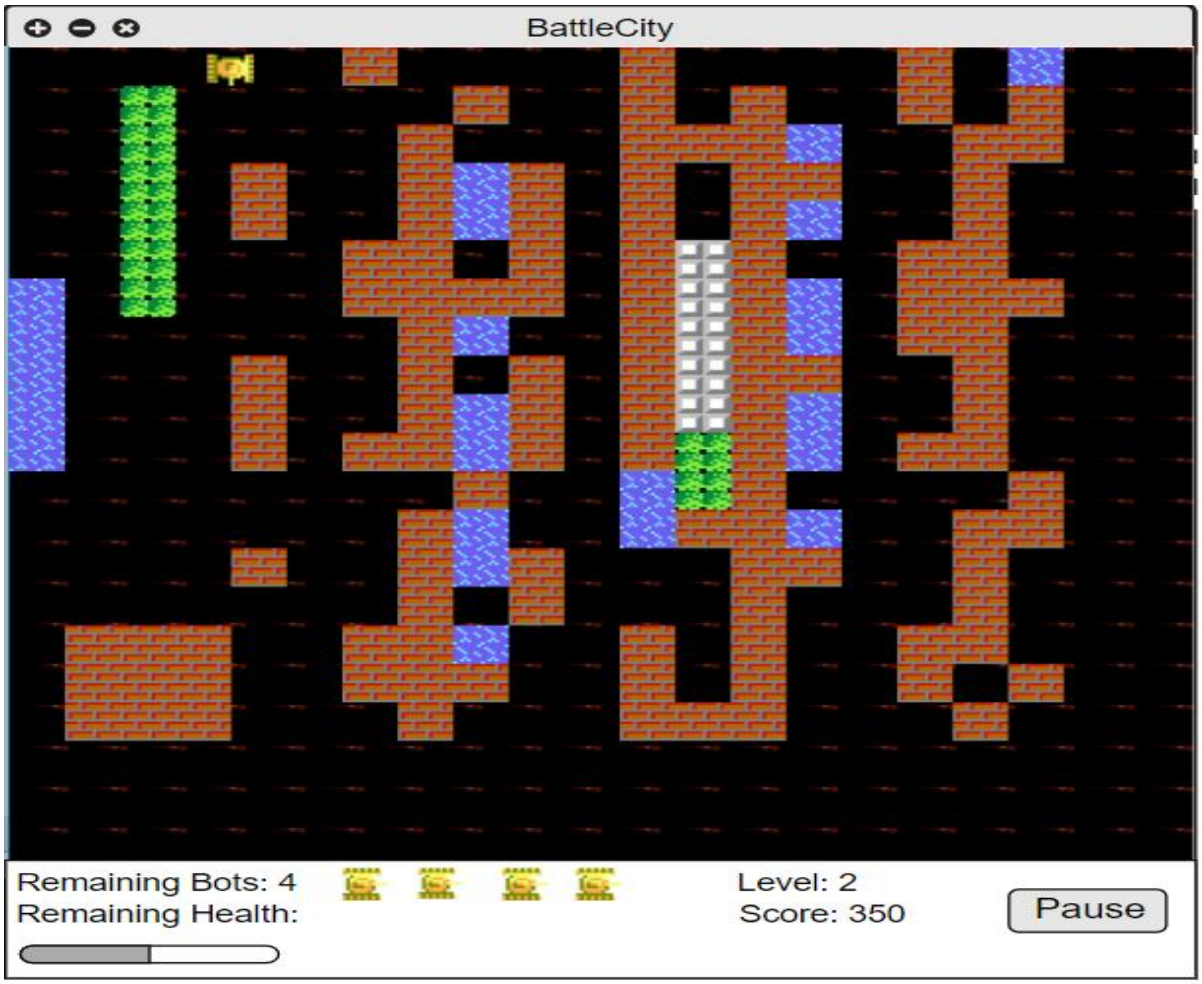
Settings screen provides the user with the opportunity to adjust the music and volume for their preferences, and change the game control settings.

5.4.2.4 Credits



Credits screen gives the user information about the developers of the game.

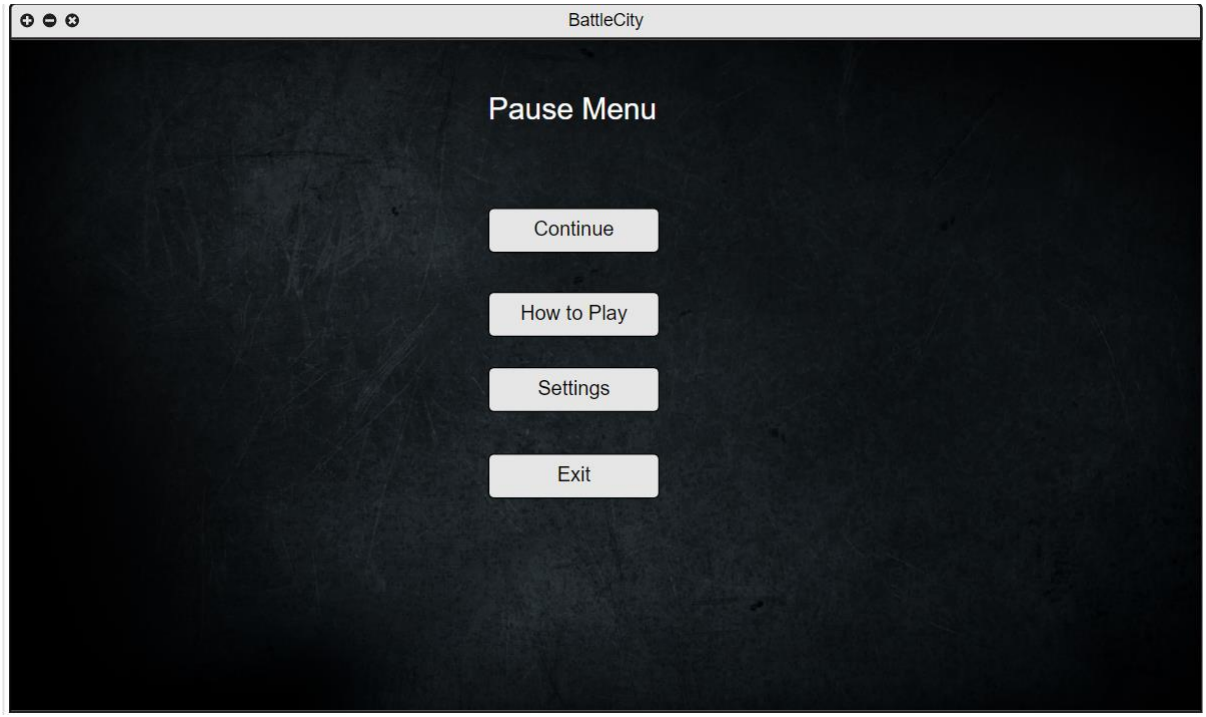
5.4.2.5 Game Screen



In Game Screen the users are going to see the map of the game, the obstacles, their own tank distinguished with the different color and the enemy bot tanks. The actual game will be played in this screen, so the user is going to see the direction, movement and the fired bullets of their own, as well as the enemy tanks. In addition, the users will see the level, their score, the amount of the enemies left, their life span and the pause button.

In more detail, the map contains the user tank, the enemy tanks, and obstacles. The enemy tanks are usually white and identical, so that the user would not be confused. However, the main tank is in another color for the distinction. The obstacles are divided into some parts; destructible and not destructible, passable and not passable, and the obstacles in which the tanks may hide. Destructible obstacles are the obstacles with bricks shown in the figure abowe. The iron-walls are non-destructible obstacles. The forests or trees are the hidable obstacles and the river ones are the non-passable obstacles. Forests are also identified as passable obstacles. Other than obstacles, user sees the bullets on the map.

5.4.2.6 Pause Menu



Pause game screen have buttons with linked to to how to play screen, settings screen and game screen, and if player desires to exit, an exit button.

6 Conclusion

In this analysis report, we have clarified our analysis in order to design and implement classic tank ware game “Battle City”. Basically this report consists of 3 main parts. The first part is “Functional Requirements” in which the necessary functionalities of the game are explained. The second main part is “Non-Functional Requirements” in which other functionalities of the game are explained. Finally, the last main part consists of system models of the game, “Battle City”.
Firstly, in the “Functional Requirements” (Section 3), we tried to explain all the functionalities which a user/player can perform. In our functional requirements section, we tried to be as clear as possible and also as realistic as possible. We did not want to put any functionality which we cannot or may not implement.
Secondly, in our “Non-Functional Requirements” (Section 4), we tried to explain what is the best parts of our design, what we offer to our players. Additionally, we put extendable missions to our project which means if we can finish our main design on time, we will try to extend the game by implementing these extendible missions.
Lastly, after deciding requirements of the game, we designed our system models. Our system model consists of 4 main parts:

1. Use case model
2. Dynamic models
    a. Activity Diagram
    b. Sequence Diagrams
3. Class and Object model
4. User interface- Navigational Paths and Screen Mock-ups

During the design process of use case model, we based on the requirements which we promised in previous sections. We watched some videos about similar games and also played them ourselves, so that we can clearly decide what the use cases of our game are.
Our dynamic models have two main parts which are “Activity Diagram” and “Sequence Diagrams”. In the activity diagram, we tried to explain how our systems runs and maintains the game. Moreover, in we have 5 sequence diagrams. In these

diagrams, we tried to explain the fundamental decision mechanism and actions of our game.

We spent, a considerable time on class and object model of our game. Before rushing into writing our diagram, we sit and discussed about how we can design it more efficiently. We tried to find the common attributes of our objects and management classes, so that in the implementation stage we can implement our game as good as possible.

As a result, we have created this analysis report to make our job easier in the implementation stage. We tried to make it as clear as possible. We hope that, this report will help us a lot for future reports and implementations.

7 Improvement Summary

For the second iteration, we added following things:

1. Two functional requirements: bonuses and portal. We could not add specific good functional requirements, since from the beginning of the first iteration we had set of good functional requirements (multiplayer, change sound/song etc.), thus we could only add bonuses. In the first iteration, we have talked about the bonuses, but it was in the list of extendable things, so we decided to actually add it.
2. We moved the “Extendibility” section to the “Functional Requirements” section, since it was more about use cases rather than the system functionality.
3. The Play Game explanation in section 3.1 was moved to the Introduction section, because the text was more like explanatory of the game, rather than the functional requirement.
4. Since two new functional requirements were added, we updated the use case diagram. Namely, the description of the use case diagram was also updated. In use case diagram, player and user were two separate actors, we decided to make them one single actor – Player. Thus, in the description

of the diagram the statements where the user was acting, were deleted or updated with the player actor.

5. Class Diagram was updated; new classes were added and explained in the explanation part.
6. Old sequence diagrams were updated and new sequence diagrams (bonus release, creation of map, control of game objects etc.) were added.
7. Content was updated
8. Activity diagram was updated; the fork nodes were added for the activities running simultaneously and the activities are written from the point of view of the system.

8 Contributions

- All chapters except 5.4 revised by Kaan Sancak and Mahin Khankishizade.
- Chapter 5.4 revised by Kaan Sancak, Mahin Khankishizade and Ozan Kerem Devamlı

9 References & Glossary

*http://www.retrogames.cz/play_014-NES.php

*https://app.moqups.com