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
4-Teymur Bakhishli

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 (extendable)

· Smooth Graphics

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

By entering this screen, players will be able to start the game. 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. There are two modes:

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.

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.

4.4. Extendibility

4.4.1 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.

4.4.2 Artificial Intelligence

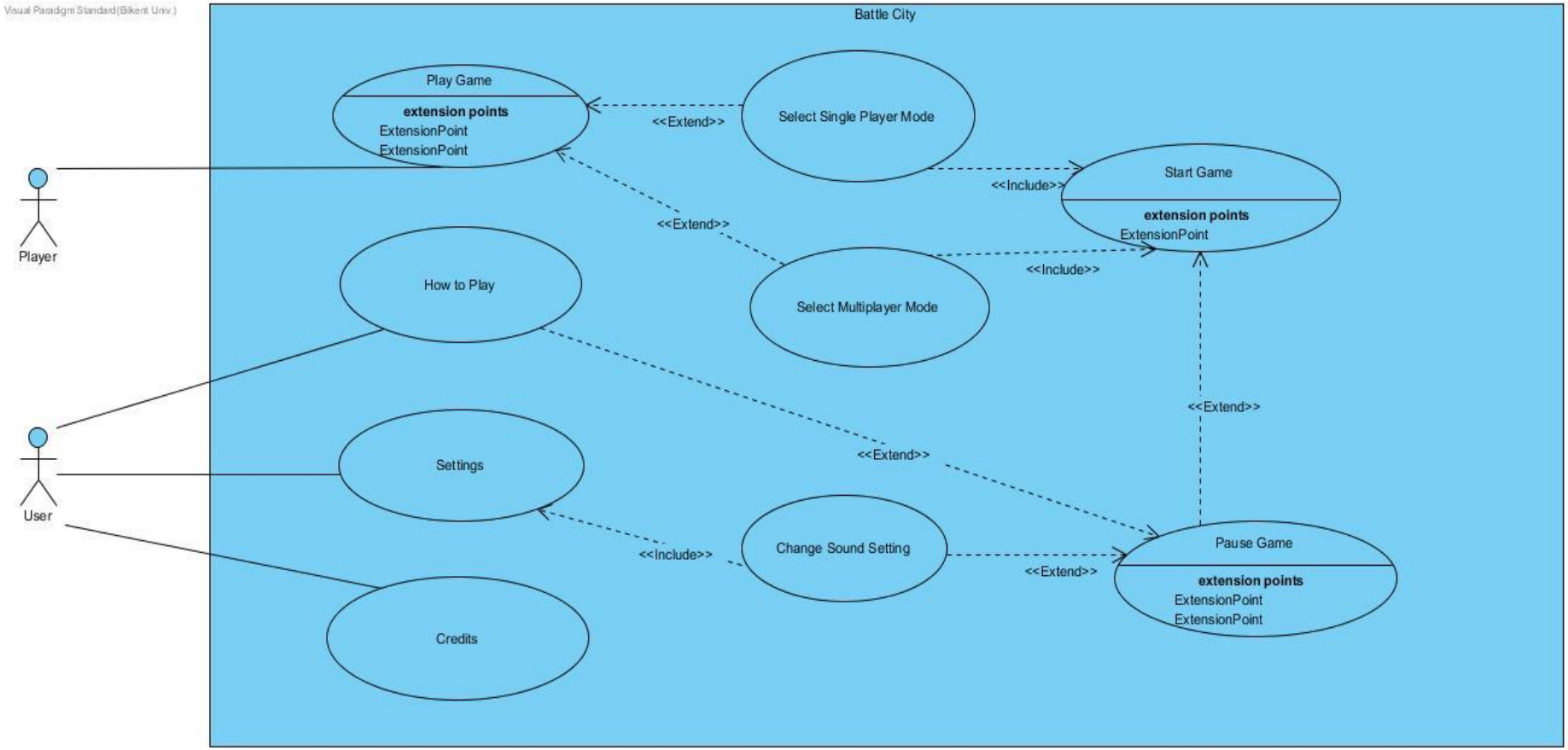
In the Single Player mode of the game, we are planning to make our bot tanks moving randomly, and shooting exact times in a second without the help of AI. However, the implementation of AI can improve the logic quality of the game in future releases. For example, tanks will be able to move and shoot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

4.4.3 Life-ups & Bonuses

These are features that can be added in order to make our game more colorful and having a variety of interesting objects. We can add Bonuses such as shell to the tanks; strengthen the power of tank bullets and different types of Life-up with 1, 2 and more lives.

5 System Models

5.1 Use-Case Model



Use Case #1

Use Case: Play Game

Primary Actor: Player

Stakeholders and Interests:

- Player/Players want to 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 selects “Play” button from main menu.
- 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 selects “Play Game” button from main menu.
2. Player chooses “Single Player Mode”.
3. Player presses “Start” button.
4. Player moves its tanks by using direction key.
5. Player fires by using fire key.
6. Player destroys a bot tank.
7. Player continues to play.

8. Player destroys all bots.
9. Player gets to next level.
10. Player repeats the steps 13 to 18.
11. Player finishes the game.

Alternative Event Flows:

1. If Player wants to return main menu:
    a. Player pauses the game by pressing pause button from game screen.
    b. Player selects “Quit Game”
    c. System quits the game.

Use Case #2

Use Case: Pause

Primary Actor: Player

Stakeholders and Interests:

- Player wants to pause the game.
- Battle City 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:

12. Player selects “Pause” button from game screen or presses “P” from keyboard.
13. System displays pause menu.

Alternative Event Flows:

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

Use Case #3

Use Case: How to Play

Primary Actor: User, Player

Stakeholders and Interests:

- Player wants to learn how to play Battle City
- User want to learn how to play Battle City
- System display how to play screen

Pre-conditions:

- Player must pause the game.
- User must be in the main menu.

Post-conditions:

- No post condition

Entry-conditions:

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

Exit conditions:

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

Success Scenario Event Flow:

14. Player selects “How to Play” button from pause menu.
15. System displays How to Play screen.

Alternative Event Flows:

1. If User wants to learn how to play:
    a. User 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 User wants to return to main menu:
    a. User selects “Exit” button.
    b. System displays main menu

Use Case #4

Use Case: Credits

Primary Actor: User

Stakeholders and Interests:

- User wants to see credits of Battle City.
- Battle City system displays credits.

Pre-conditions:

- User must be in the main menu.

Post-conditions:

- No post condition

Entry-conditions:

- User selects “Credits” button from main menu.

Exit conditions:

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

Success Scenario Event Flow:

16. User selects “Credits” button from main menu.
17. System displays pause menu.

Alternative Event Flows:

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

Use Case #5

Use Case: Settings

Primary Actor: User

Stakeholders and Interests:

- User/Player wants to change sound settings.
- System changes sound settings.

Pre-conditions:

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

Post-conditions:

- Sound settings are updated.

Entry-conditions:

- User selects “Change settings” button from main menu.
- Player selects “Change settings” button from pause menu.
- User changes sound settings by pressing “mute” button or using “volume” bar.
- Player changes sound settings by pressing “mute” button or using “volume” bar.

Exit conditions:

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

Success Scenario Event Flow:

18. User selects “Settings” button from main menu.
19. User mutes the game by pressing “mute” button.
20. System mutes the game.

Alternative Event Flows:

1. If User wants to return main menu:
    a. Player presses “Esc” from keyboard.
    b. System displays main menu.
2. If Player wants to change settings:
    a. Player selects “Settings” from Pause menu.
    b. Player changes volume by using volume bar.
3. If Player wants to return pause menu:
    a. Player presses “Exit” button from Settings menu.
    b. System displays pause menu.

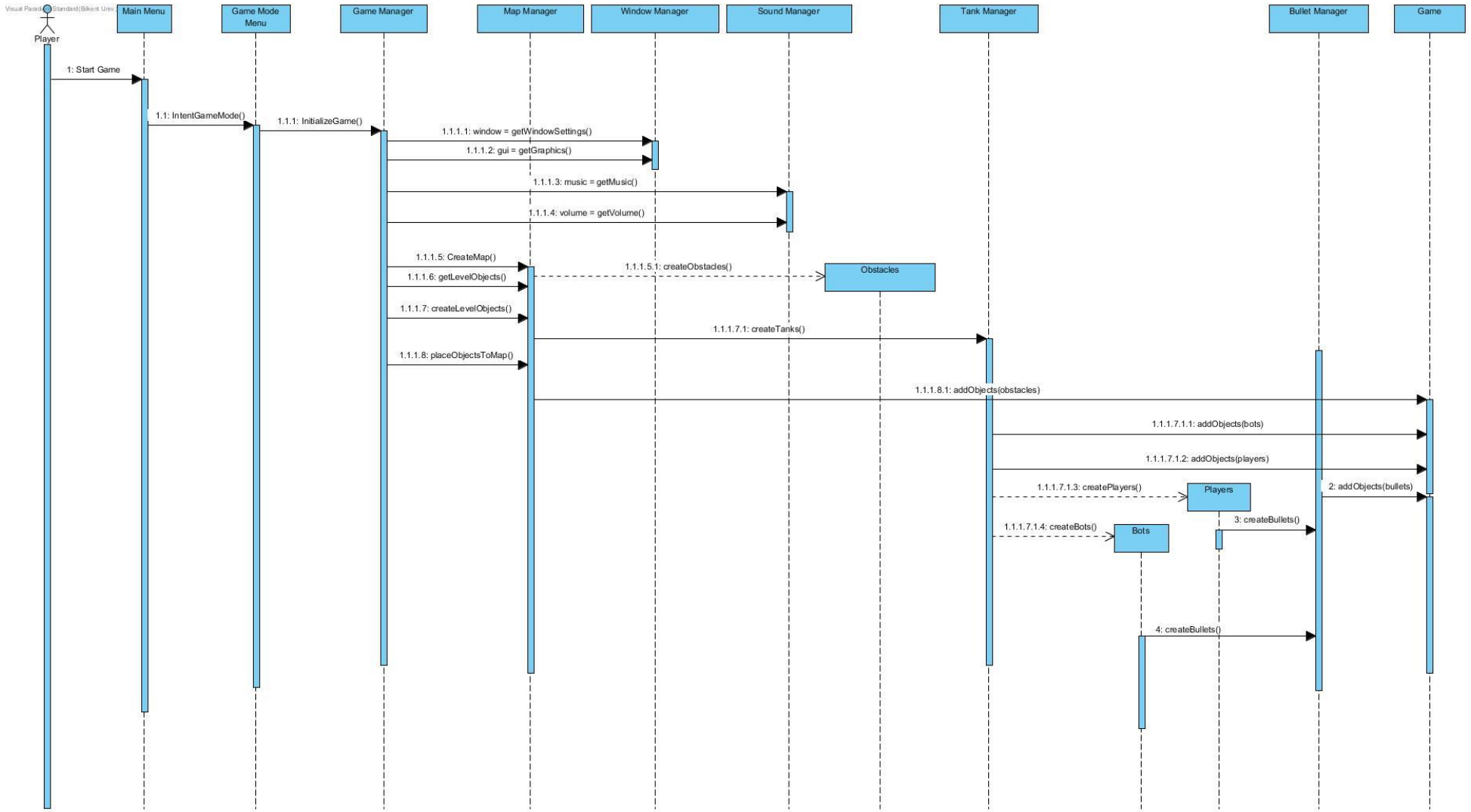
5.2 Dynamic Models

5.2.1 Sequence Diagrams

5.2.1.1 Start Game

Scenario: User starts the game

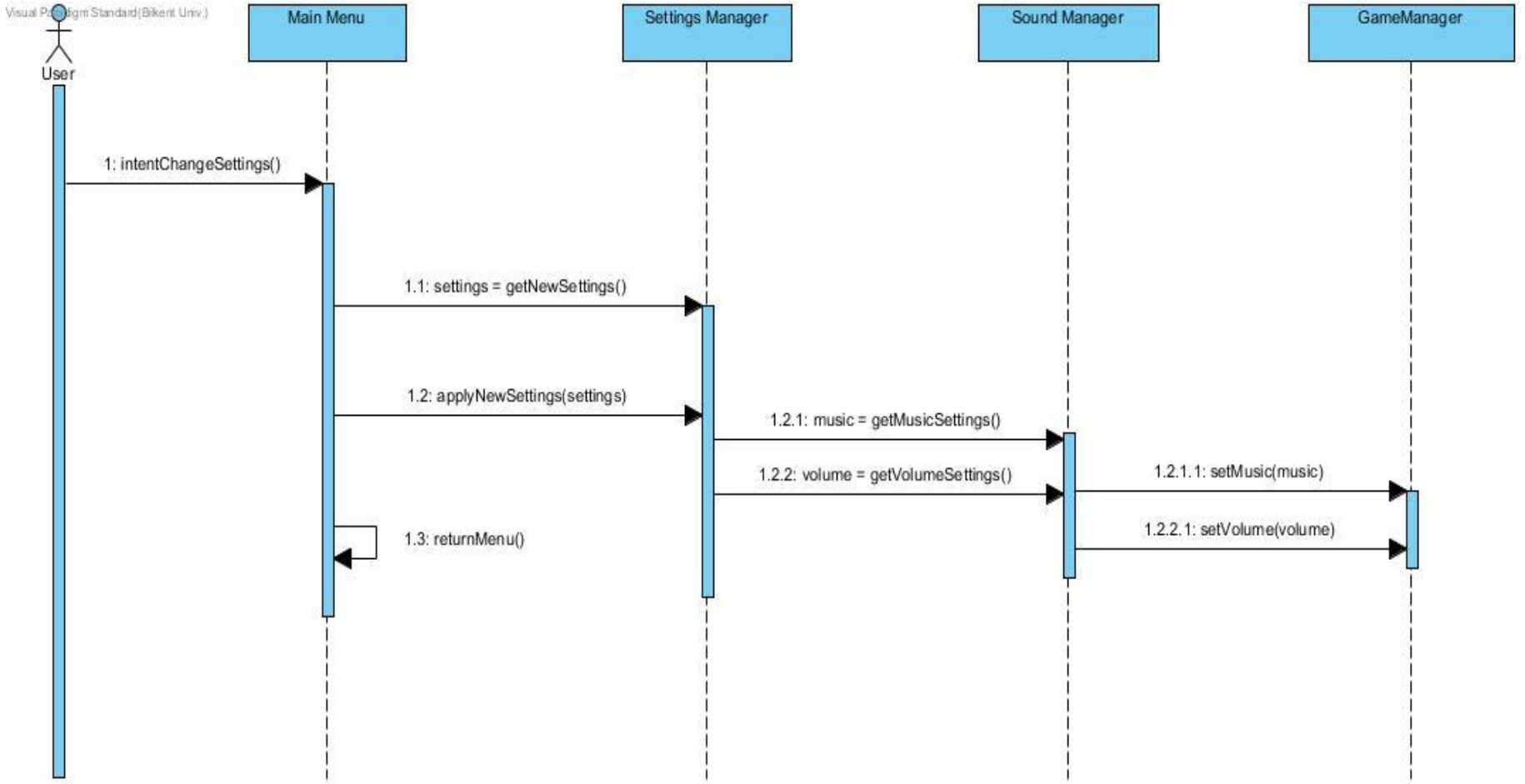
User wants to start the game. He/she enters the Main Menu and chooses one of the modes: single or multiplayer game. The game is initialized by the Game Manager. The creation of the map, the objects on the map and the level of the map is identified through the Map Manager. Gui and the window settings 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.


Player
Main Menu
Game Mode Menu
Game Manager
Map Manager
Window Manager
Sound Manager
Tank Manager
Bullet Manager
Game
Obstacles
Players
Bots
1: Start Game
1.1: IntentGameMode()
1.1.1: InitializeGame()
1.1.1.1: window = getWindowSettings()
1.1.1.2: gui = getGraphics()
1.1.1.3: music = getMusic()
1.1.1.4: volume = getVolume()
1.1.1.5: CreateMap()
1.1.1.5.1: createObstacles()
1.1.1.6: getLevelObjects()
1.1.1.7: createLevelObjects()
1.1.1.7.1: createTanks()
1.1.1.8: placeObjectsToMap()
1.1.1.8.1: addObjects(obstacles)
1.1.1.7.1.1: addObjects(bots)
1.1.1.7.1.2: addObjects(players)
1.1.1.7.1.3: createPlayers()
2: addObjects(bullets)
3: createBullets()
1.1.1.7.1.4: createBots()
4: createBullets()

5.2.1.2 Settings

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

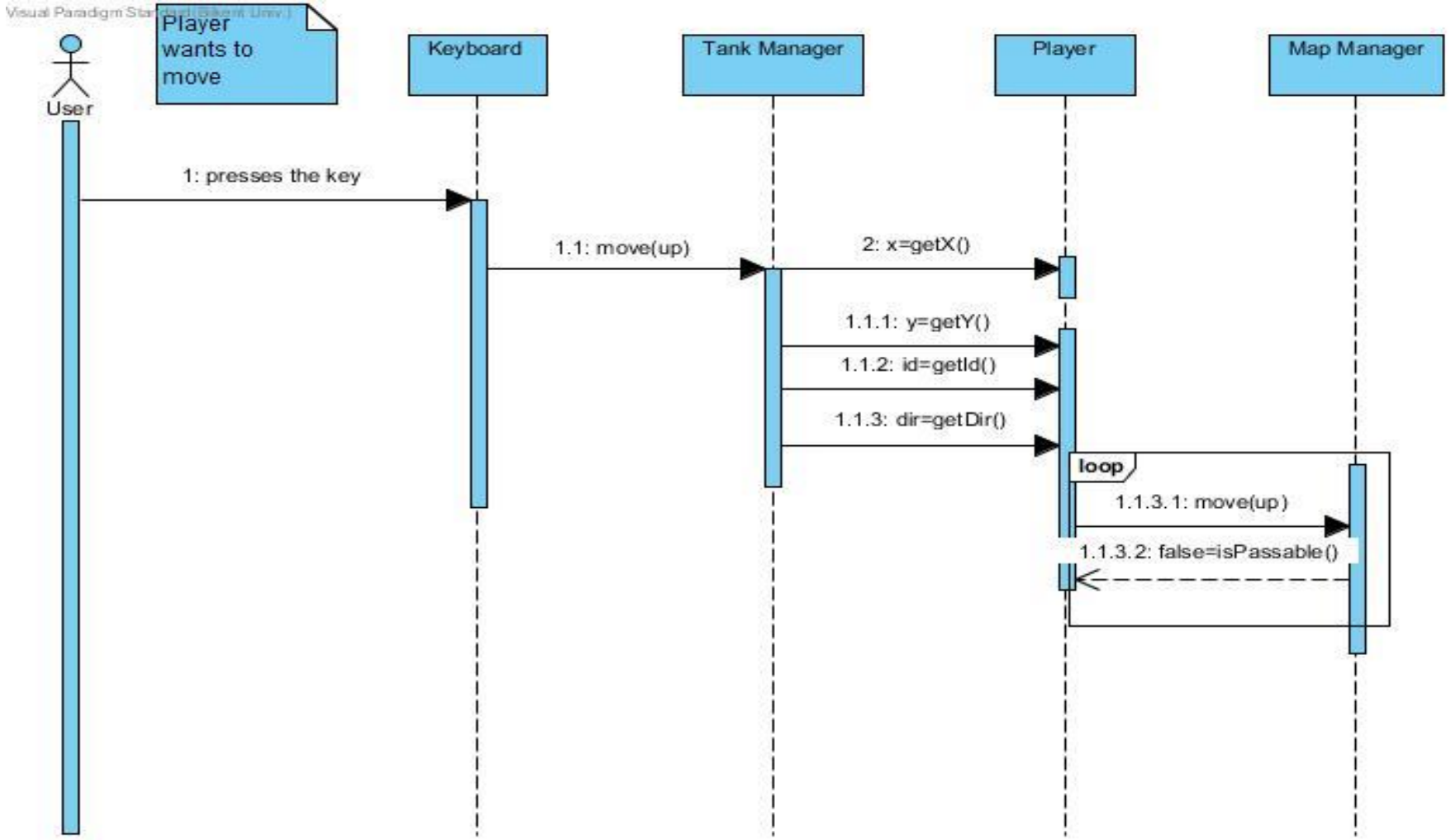
User wants to adjust the settings to his/her preferences, thus he/she enters the Main Menu. Here, if the user does not want to do any changes, he/she returns to Main Menu. Else, the user changes the settings with getNewSettings() method, managed by Settings Manager. Later changed settings are being applied 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
Main Menu
Settings Manager
Sound Manager
GameManager
1: intentChangeSettings()
1.1: settings = getNewSettings()
1.2: applyNewSettings(settings)
1.2.1: music = getMusicSettings()
1.2.2: volume = getVolumeSettings()
1.2.1.1: setMusic(music)
1.2.2.1: setVolume(volume)
1.3: returnMenu()

5.2.1.3 Movement Mechanism

Scenario: User wants to move

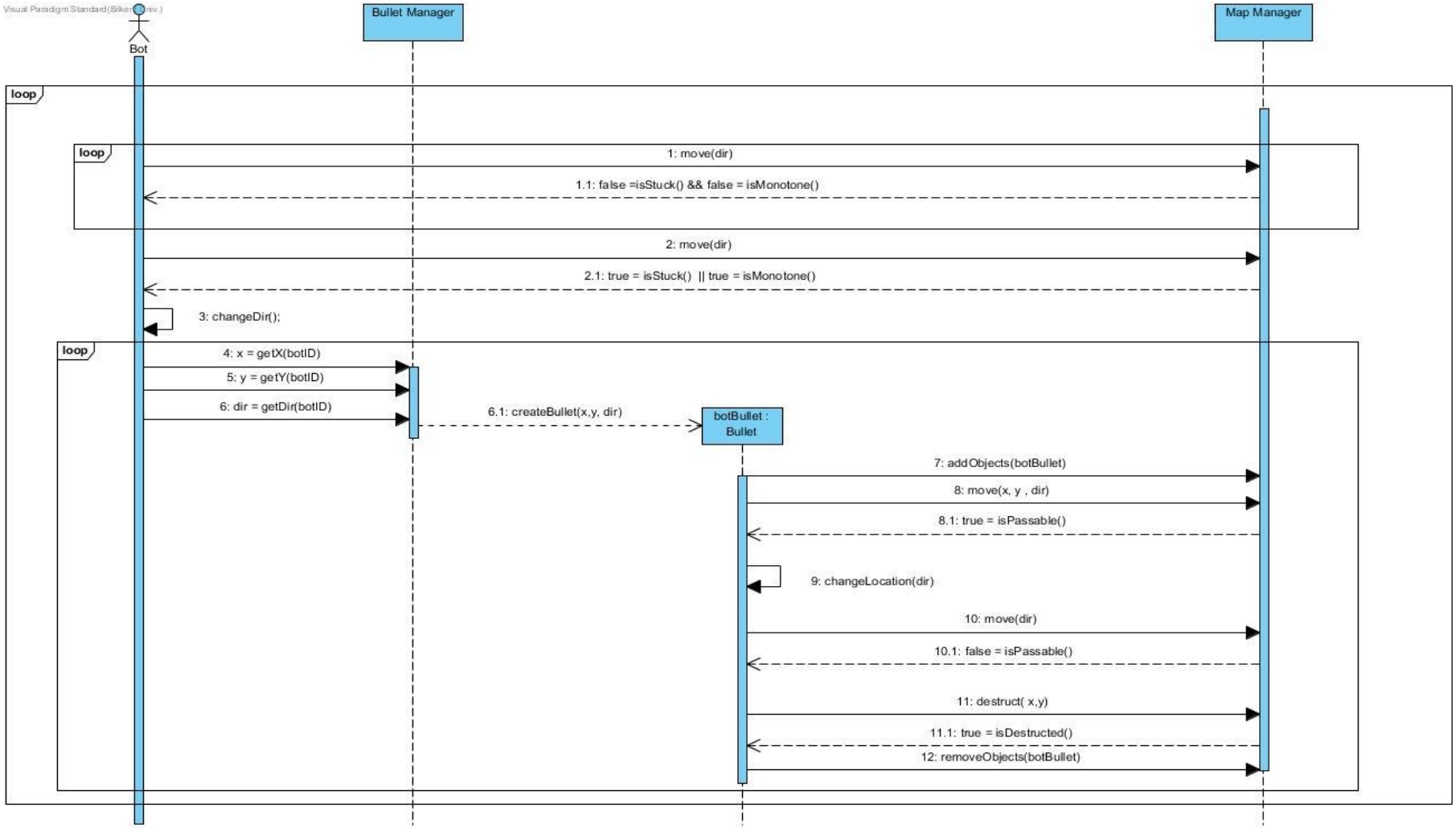
The user wants to move and presses the key from Keyboard. The command of Keyboard calls the move() method from Tank Manager class in order to apply move instruction. Player class gets the (x,y,dir) and the Id (in order to determine which user wants to move) inputs from Tank Manager class. Towards the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


Player wants to move
User
Keyboard
Tank Manager
Player
Map Manager
1: presses the key
1.1: move(up)
2: x=getX()
1.1.1: y=getY()
1.1.2: id=getId()
1.1.3: dir=getDir()
loop
1.1.3.1: move(up)
1.1.3.2: false=isPassable()

5.2.1.4 Bullet Creation and Fire Interaction

Scenario: User wants to move.

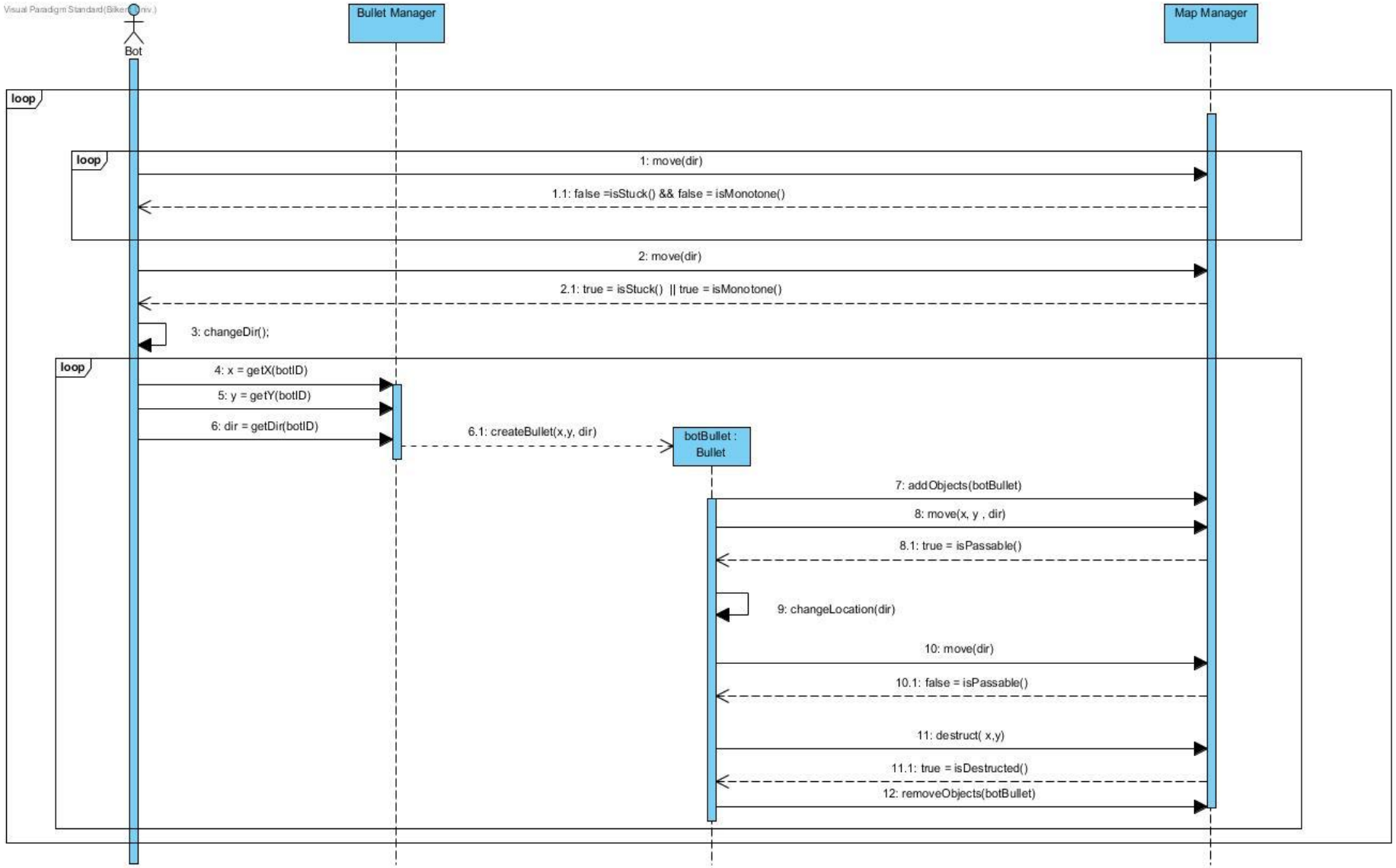
The user wants to move and presses the key from Keyboard. The command of Keyboard calls the move() method from Tank Manager class in order to apply move instruction. Player class gets the (x,y,dir) and the Id (in order to determine which user wants to move) inputs from Tank Manager class. Towards the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


Visual Paradigm Standard(Bikerr (Non-Commercial))
Bot
Bullet Manager
Map Manager
loop
loop
1: move(dir)
1.1: false =isStuck() && false = isMonotone()
2: move(dir)
2.1: true = isStuck() || true = isMonotone()
3: changeDir();
loop
4: x = getX(botID)
5: y = getY(botID)
6: dir = getDir(botID)
6.1: createBullet(x,y, dir)
botBullet : Bullet
7: addObjects(botBullet)
8: move(x, y , dir)
8.1: true = isPassable()
9: changeLocation(dir)
10: move(dir)
10.1: false = isPassable()
11: destruct(x,y)
11.1: true = isDestructed()
12: removeObjects(botBullet)

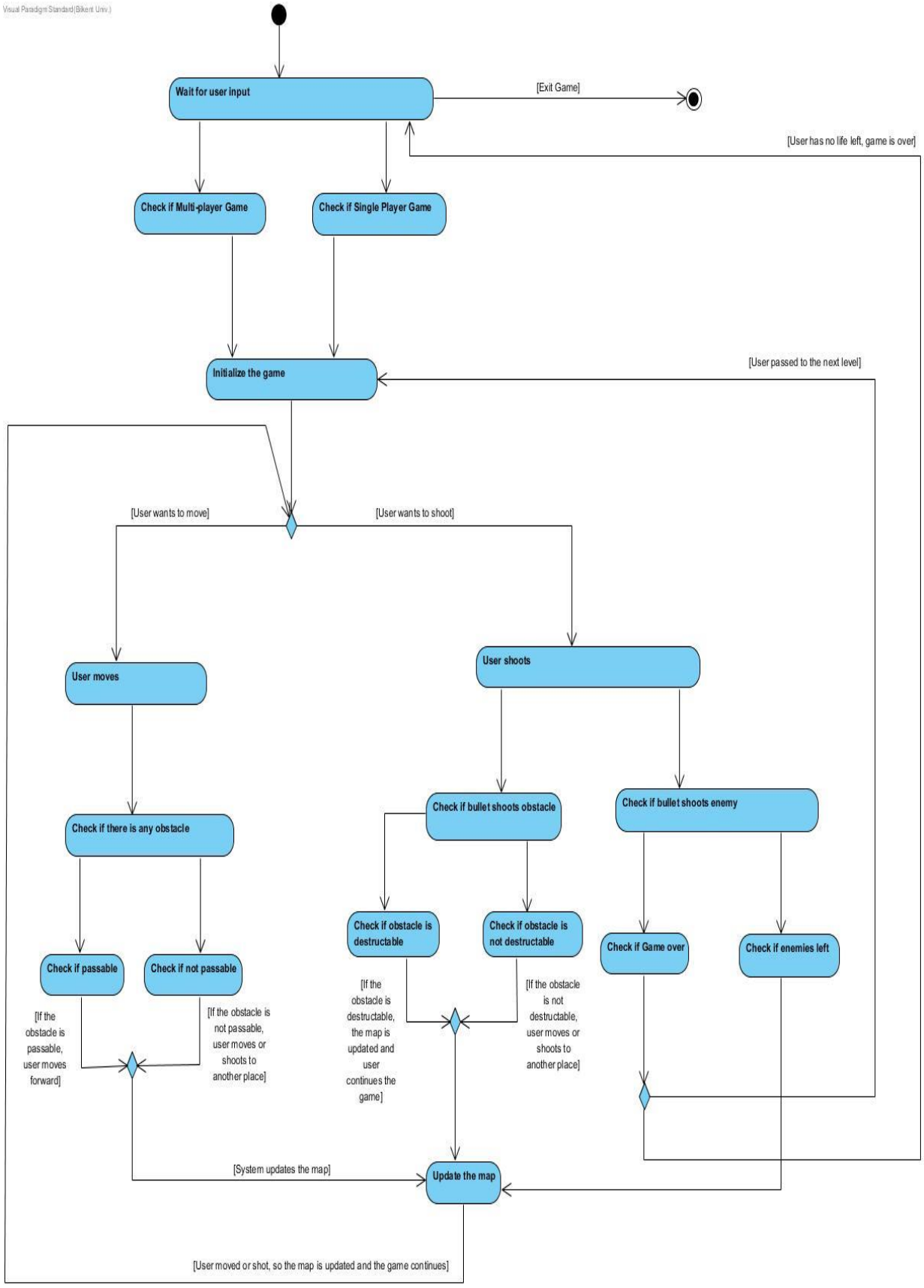
5.2.1.5 Bot Mechanism

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(Biker (Univ.)
Bot
Bullet Manager
Map Manager
loop
loop
1: move(dir)
1.1: false =isStuck() && false = isMonotone()
2: move(dir)
2.1: true = isStuck() || true = isMonotone()
3: changeDir();
loop
4: x = getX(botID)
5: y = getY(botID)
6: dir = getDir(botID)
6.1: createBullet(x,y, dir)
botBullet : Bullet
7: addObjects(botBullet)
8: move(x, y , dir)
8.1: true = isPassable()
9: changeLocation(dir)
10: move(dir)
10.1: false = isPassable()
11: destruct(x,y)
11.1: true = isDestructed()
12: removeObjects(botBullet)

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.

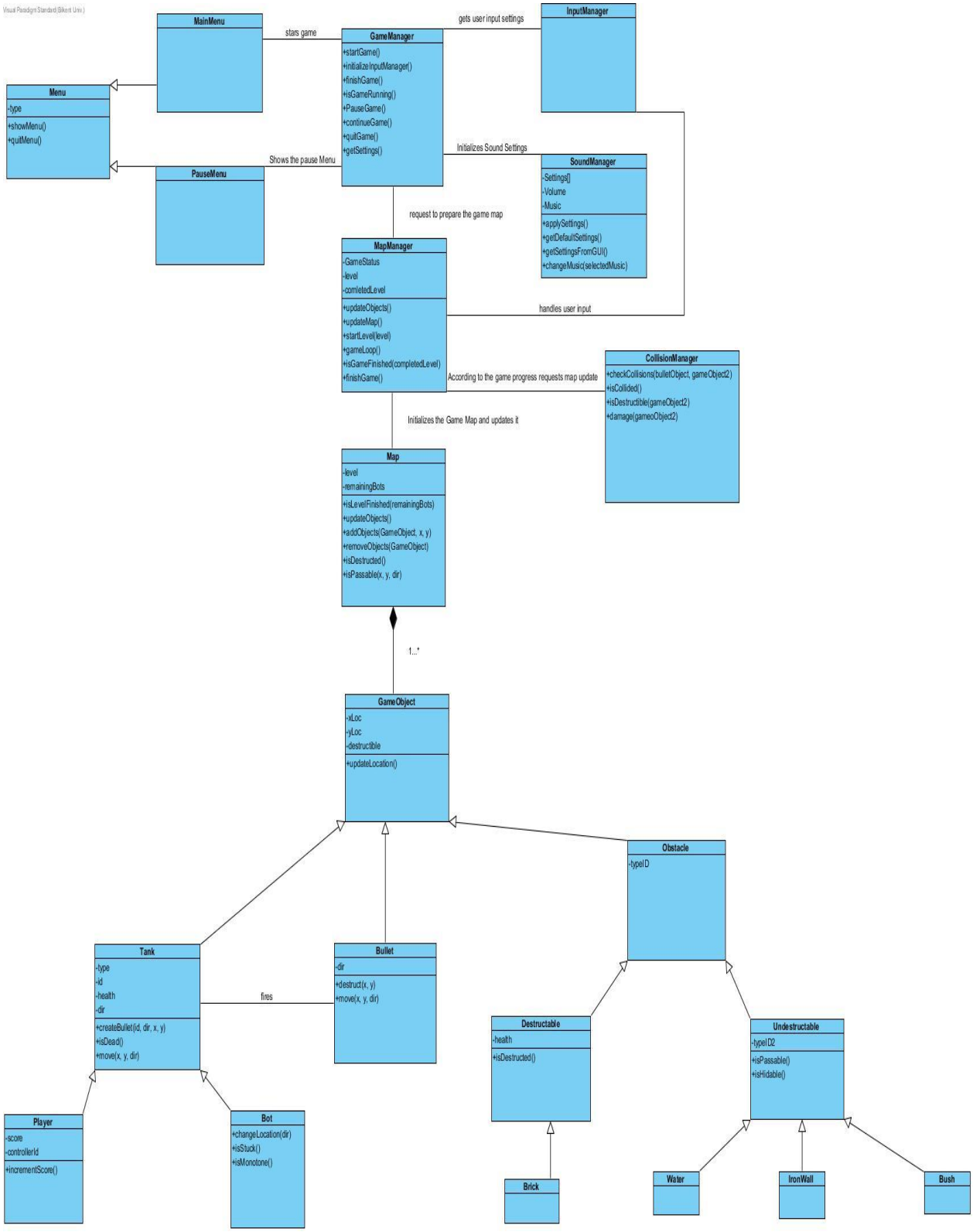
Case 2: the bullet hits the enemy. When the bullet hits the enemy, the system checks if there are any enemies left.

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. However, if there are

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



The class diagram of “Battle City” game illustrated above. Currently, the class diagram has 20 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.

InputManager:

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.

SoundManager 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:

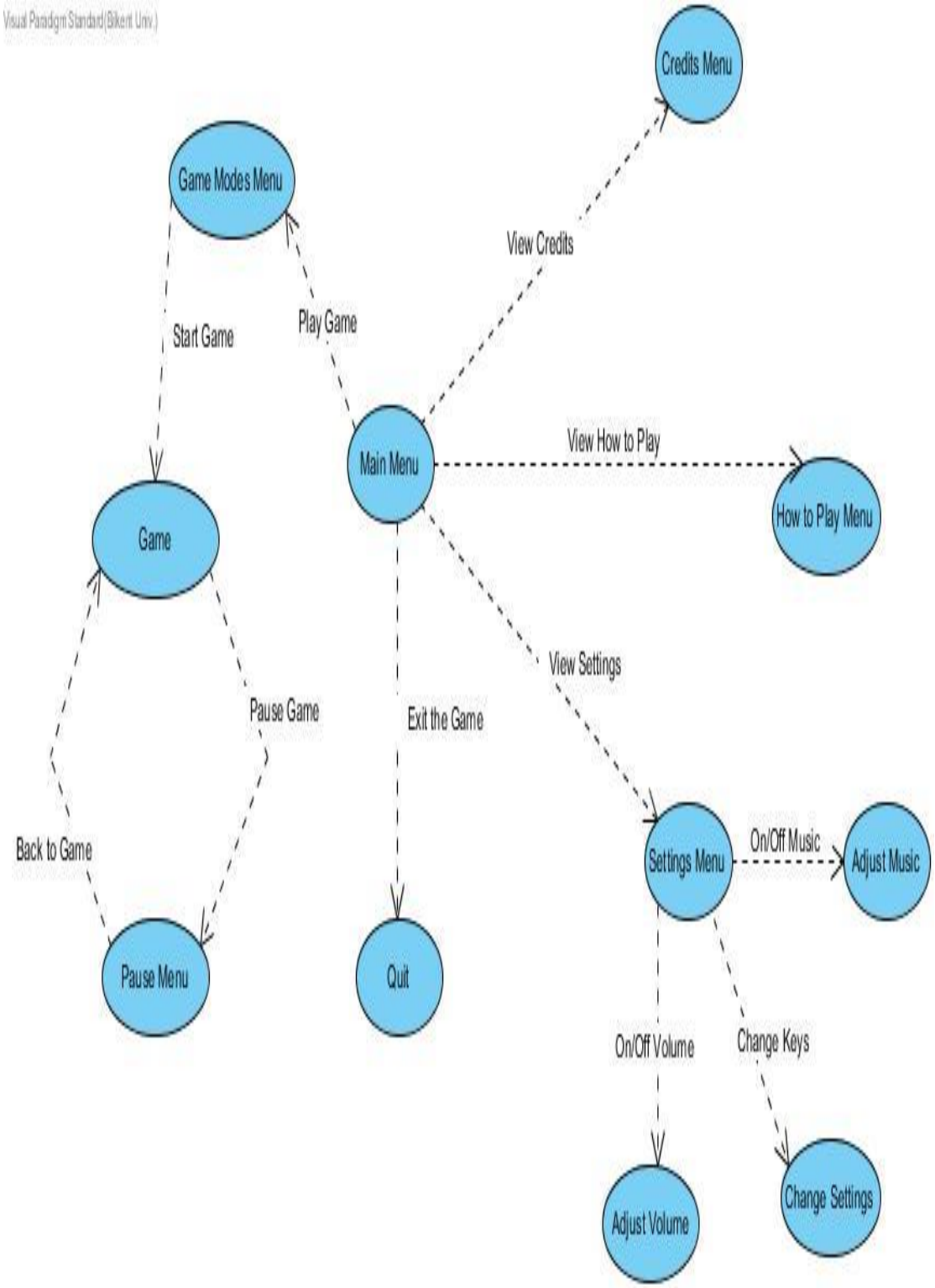
IronWall is one of the subclasses of undestrubtable obstacles. Basically, no tank can destroy these objects by firing bullets and no bullet can pass through them. Also, no tank can pass through these objects by moving.

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.

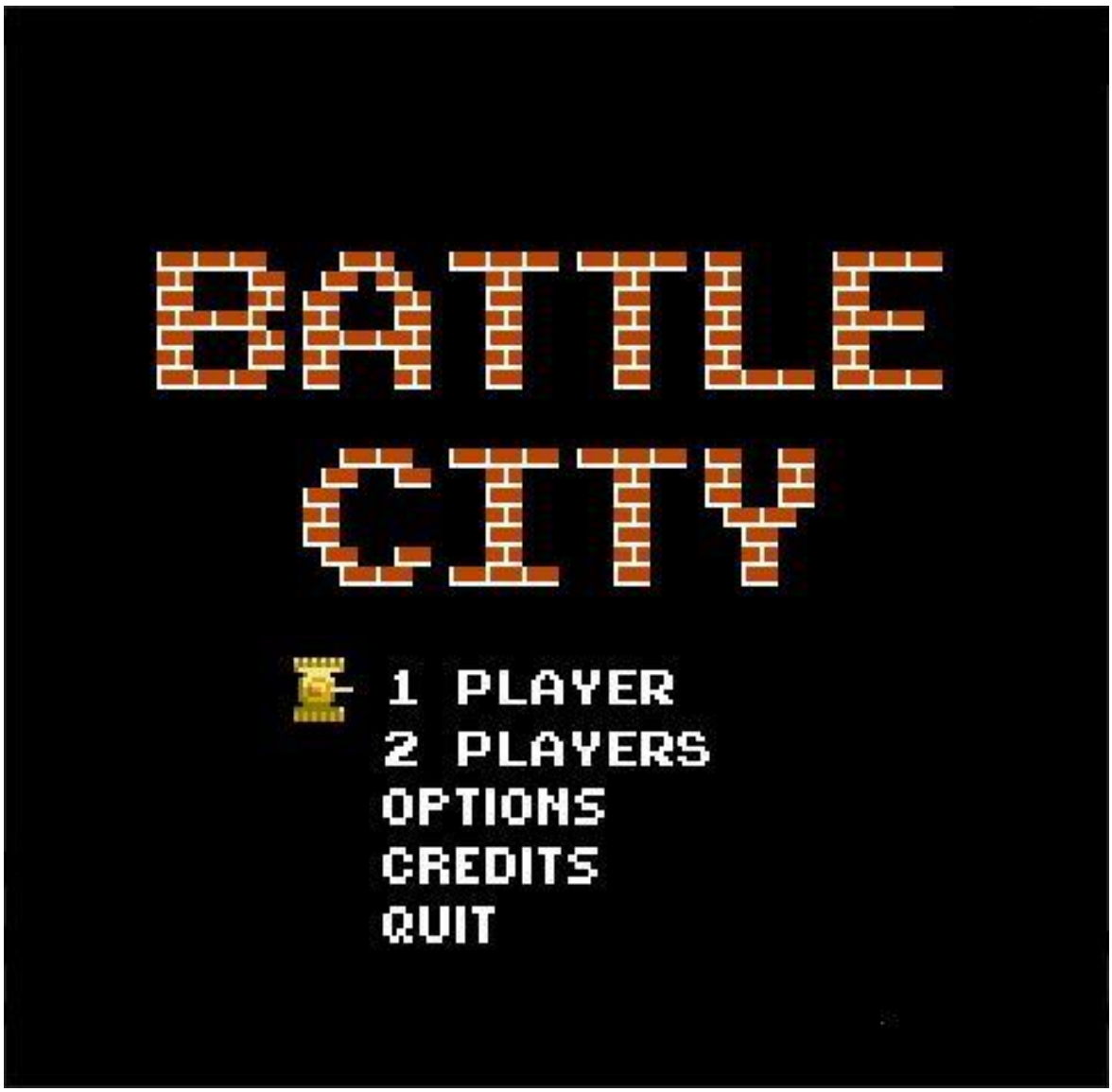
5.4 User interface – navigational paths and screen mock-ups

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 (Options)

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 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 user tank may hide. Destructible obstacles are the obstacles with bricks shown in the figure below. 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.

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 References & Glossary

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