Bilkent University

Department of Computer Engineering

CS 319 Term Project

Group 1H

War of Domination

Analysis Report

Project Group Members:

Akant Atılgan

Unas Sikandar Butt

Kazım Ayberk Tecimer

Teymur Bakhishli

Supervisor: Eray Tüzün

Contents

1. Introduction

As members of group 1H, we decided to design and implement an arcade game called “War of Domination”. We chose this game as this game allows us to work with the Object & Oriented Programming principles, as there are many instances that we can implement as objects in game, different type of obstacles, different game modes, different maps, different type of characters.

As mentioned above, the main reason for choosing such game is its structure. We think that War of Domination is suitable for the object oriented environment. Moreover, such suitability allows us to enlarge the features of the game and add new features as we need them. For now the feature list of the game is shown below.

- Single Player Mode
- Local Multiplayer Mode
- Different Game Mode Types
- Variety of Obstacles
- Weapons (knife, rifle, bomb)
- Fluent Graphics
- Different Maps

We will be implementing our game in Java, as Java is the language we are most comfortable with, we are planning to use Java Swing library

as it allows us to design graphics and UI design over standard Java distribution. Our main aim by implementing this game is to get familiar with the object oriented design principles and the concepts of the principles that we will be learning in CS 319 through the semester.

2. Overview

2.1 Gameplay

War of Domination will be a both single and multiplayer 2D game. A player of this game is going to be represented by a warrior. Each player will be allowed to control his/her warrior. The aim of the player is to survive by killing the other player, by other meaning killing the all enemies. This game will be turn based strategy game, where two opposing teams fight in a battleground until one team is eliminated. Each team gets 2-4 characters each round depending on the map. The game will be played with a fixed camera and an overhead view, so players can keep track of everything at all times. As it is a turn based game, players will alternate between turns and each turn the player has to choose a character to move/shoot/attack, once the player is done, or runs out of time, the opposing team's turn starts. The game will have different map consisting of various elements.

2.2 Map

The map of this game will be consisting of tiles, bricks, and obstacles. The most basic component of the map will be the tiles. The map will then be further built on those tiles. For different game modes, there will be different types of maps consisting of different kind of tiles. Each turn the player can choose to move a character across the tiles, however there will be a limit to

the amount of distance a character can move each turn. The map will consist of obstacles which will restrict movement of characters across them.

2.3 Weapons

All characters will be equipped with a standard gun and a grenade. These weapons will be used to defeat the opposing team. The map may provide some power-ups which can upgrade weapons or present special weapons. There will be no limit to the ammunition of the standard weapons.

2.4 Power- Ups

The map will summon power-ups randomly. These power ups will provide players with a time-limited advantage. The power-ups can include: HP-Regen, Weapon-Upgrade, Wall-Breaker etc.

2.5 Traps

The map will consist of different randomly placed. These traps can be visible or hidden. Some examples of these traps can be ditches, mines, etc.

2.6 Menu

The Main-Menu in the game will be a button based title screen. The options will include, NEW GAME, INSTRUCTIONS, SETTINGS and CREDITS.

2.7 Obstacles

Obstacles will be a major part of the map. The map will be designed in way such that the players will have to navigate their way through these obstacles, and try to keep control of advantageous points throughout the map. The map will consist of various types of obstacles. Some obstacles will be breakable through the use of weapons, some obstacles will not be breakable but will be pierce-able through bullets, in this way the player can shoot

through these obstacles to hit the enemy, but they will not be able to pass through it. There will also be some unbreakable bullet proof obstacles, which can be used by players to avoid incoming enemy shots.

2.8 Settings

Player will be able to change the settings of the game. He/she will be able to change the sound and the keyboard configuration.

3. Functional Requirements

3.1 Main Menu

This screen will allow the user to navigate to other mandatory functions like “play game” or optional functions like "how to play" "credits" etc. It is the basic screen user will encounter when they first launch the game.

3.2 Start Game

When players press the start button, by default player 1 will get first turn. There will be an indicator on a player's characters to let the user(s) know whose turn is being played at that moment. Players will be able to pick one character from their team and play it; until there are no more idle characters in their disposal. When a player has no available characters to play, turn will switch to the opponent. The player with no-characters left alive loses the game resulting in opponent getting victorious. Players will have a limited space of movement each turn, when they finish the movement phase they will have variety of attacking options listed in hotkeys from 1 to 9 (available options might change in future iterations as well as characters might have classes in future iterations). The map itself will contain both obstacles and passable objects as well as pits/water holes (which would result in the death of the player) etc. There will be 3 basic weapon types in players’ disposal; direct line of sight (los) weapons which results in instant damage and a possibility to headshot for a multiplier to their damage. Explosive weaponry will create a knockback

effect which will cause any characters in the field of effect to be washed off slightly. Third weapon type will be melee weapons which will also have a knockback effect. The game can be played with mouse only but it will also have keyboard support for hotkeys if players want to play faster. Mainly the objective of the game is to eliminate other player's all characters by any means. Currently there will be only 1 mode:

3.2.1 Local Co-Op Multiplayer Mode:

This mode will be played by default when user selects "Start Game". This mode will enable user to pick a map from the list of available maps; it will also let the user to select the amount of characters will be in the game. This mode will put teams to separate corners or pre-defined spawns in the selected map.

3.3 Save/Load Options

3.3.1 Load Game

This option will be only available during main-menu. It will load a previously stored game from its latest state and player turn. Load game might not work properly for different versions of the game.

3.3.2 Save Game

This option will only be available if there is a game going on at that moment. Players will be able to save their games to any directory they pick to load them up later.

3.4 How To Play

User will be able to access this screen from the main menu or while playing the game by pressing "esc" to open a pop-up menu window.

This screen will contain:

-Game Mechanics (Knockback,Headshot,Melee, weapon types etc.)

-Game Objective(s)

-Turn management (Movement limits/Shooting/Available options (guard/stand still/aim etc.))

-Map Mechanics (Obstacles, holes, passable objects).

This screen will let users to start playing right away and since they can access this information even while playing they will have a most basic gamepedia at their disposal. This will let players to learn the game much faster.

3.5 Options

This screen will contain following options:

-Volume (SFX/Music)

-Resolution (Will re-open the game frame when exiting options)

-Aim Assist (Will enable/disable a line of your aim while using Line Of Sight weaponry.)

-Gamma (Adjust Brightness)

-Change hotkeys for item access, character actions. (Note: These changes will be global and affect all players in the game.)

3.6 Feedback

This screen will allow the user to send feedback to the developers about the game.

3.7 Credits

This screen will let players know better about the developers behind the game; and contact them if they want to by giving their e-mail addresses. It will also contain a sliding text which will list the individual parts done to form the game by the developers.

4 Non-Functional Requirements:

4.1 Game Performance

Since we are planning to add animations to every action in the game to some extend; we are going to make sure these animations won't cause any performance drops in the gameplay. The gameplay should be always smooth according to users eye's which means it should have a higher FPS rate than the monitor refresh rate (Which is usually 60~ by most standards). The game also shouldn't strain the computer too much since it should be compatible with all systems available in the market that can run java applications.

4.2 Tile Options (HD Tiles/Shadowing Tiles/Animation Tiles):

The tiles we are going to use to form our maps will be picked from a high-quality open-source tiles for 2d games. These tiles can even contain shadows on their own drawings which will enhance the quality of the game remarkably while keeping as little memory space as possible. Some tiles will also contain animations which is basically 2 images flipping constantly. This will allow the game to be much more real-world a-like in the eyes of the user.

4.3 Non-Complex User-Interface:

Every user should be able to understand the basic options given that the players know English. While doing this the UI shouldn't look

too empty, so there will be some visual decorations as long as it doesn't make the UI look too complicated.

Another problem is the nature of turn-based games. The in-game screen usually scares people most of the time when they first see it. We will try to make it as simple as possible by adding simple options but a lot of flexible mechanics to these options to keep the gameplay deep enough and simple enough at the same time.(Like grenade tossing from a wall, headshot option, pushing a player into a hole with melee option, etc.)

4.4 Extending Options

4.4.1 More Teams & Ally Options:

For the moment the game will support only 2 teams; but it could be easily manipulated by changing map spawns and team limitations; user(s) would be able to select the amount of teams to play a team game 2 versus 2 or Free-For-All.

4.4.2 Map Maker:

The best way to keep a game alive is to give tools to the community that is compatible with the game. If community can get a free map-making tool which would allow them to create their own maps from a pre-defined tile list, re-playability of the game will increase significantly with new unique maps.

4.4.3 Game Modes:

We will try to add depth into the game by adding minimal complexity and maximum gain.

By adding new victory conditions; or new default starting conditions we will implement mods; which will let players to experience a different kind of gameplay.

Example Mods:

-Insta-Death (Every Character has 1 hit point.)

-No Legs (Every character has %90 less movement range.)

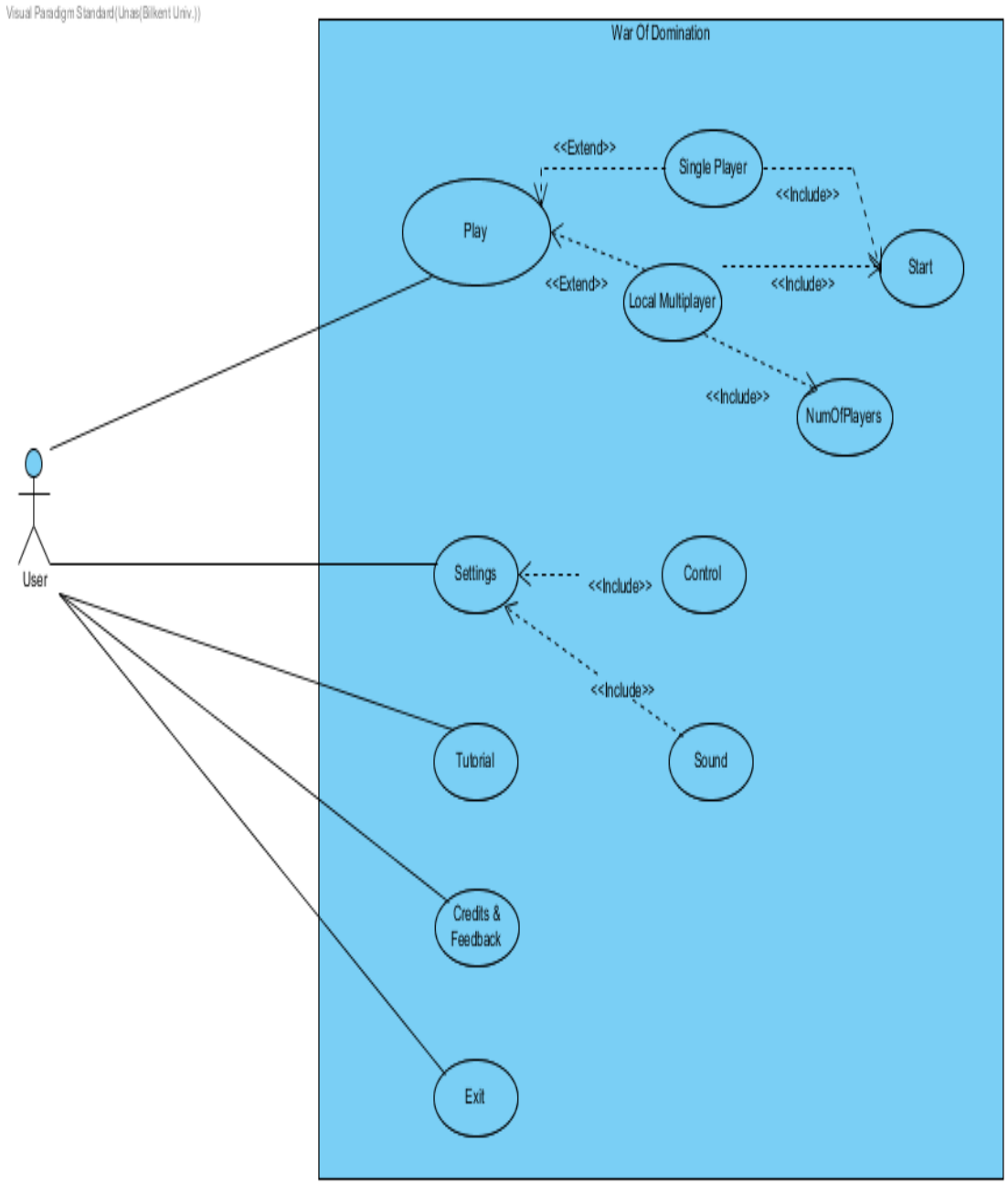
-None Left Behind (First player to lose a character loses the game.)

-Control the Beacon (The first player who fills the "beacon" bar, by staying next to the beacon will win the game.)

etc.

5 System Models

5.1 Use-Case Model



Use Case: Play

Primary actor: User

Interests:
1. User/s want to play the game.
2. System initializes the game.

Pre-conditions:
1. User must start the game.
2. User has to be in the main menu.

Post-conditions: NULL

Entry conditions:
1. User selects “Play” button in the main menu.
2. User selects the desired game mode (single/multiplayer).

Exit conditions:
1. User selects quit game button from the pause menu.
2. User finishes the round by either winning or losing.

Success scenario event flow:

1. User selects “Play” in the main menu.
2. User selects “Single Player Mode”.
3. User selects “Start” button.
4. User moves the character using the direction keys on the keyboard.
5. User uses the fire key to shoot bullets at enemy.
6. Enemy dies.
7. Repeat steps 4-6.
8. All enemies are dead
9. User finishes the round.

Alternative Event flow:

a. User presses pause key while in game.
b. User selects quit game option.
c. System closes the game.

Use Case: Settings

Primary Actor: User

Interests:
1. User wishes to view settings.
2. User wants to change sound settings.
3. User wants to change keyboard configurations.

Pre-conditions:
1. User must be in the main menu.

OR

1. User must be in the pause menu.

Post-conditions:
1. Settings are updated.

Entry conditions:
1. User selects settings button in the main menu.
2. User selects settings button from the pause menu.

Exit conditions:
1. User selects back button from the settings menu.
2. User presses the “esc” key from the keyboard.

Success scenario event flow:

1. User selects the settings button from the main menu.
2. User selects the “disable game music” check-box.
3. The system disables the music from the game.
4. User presses “esc” key to exit the settings menu.

Alternate flow of events:

1. User selects the settings button from the pause menu.
2. User selects the change keyboard configuration button from the settings menu.
3. User selects the fire key.
4. User presses the “space” key on the keyboard to assign it.

5. User presses the “esc” key to go back to settings menu.
6. User presses the “esc” key to go back to continue the game.

Use Case: Tutorial

Primary Actor: User

Interests: 1. User wishes to learn how to play the game.
2. User wants to know the keyboard configurations.

Pre-conditions: 1. User must be in the main menu.

Post-conditions: NULL

Entry conditions: 1. User selects the “tutorial” button from the main menu.

Exit conditions: 1. User selects back button from the tutorial page.
2. User presses the “esc” key from the keyboard.

Success scenario event flow:

1. User selects the tutorial button from the main menu.
2. User reads the tutorial page, to learn the game.
3. User presses the “esc” key to go back to the main menu.

Use Case: Credits & Feedback

Primary Actor: User

Interests: 1. User wishes to know the creators of the link.
2. User wants to give the creator his feedback on the game.

Pre-conditions: 1. User must be in the main menu.

Post-conditions: NULL

Entry conditions: 1. User selects the “Credits & Feedback” button from the main menu.

Exit conditions: 1. User selects back button from the credits & feedback page.
2. User presses the “esc” key from the keyboard.

Success scenario event flow:

1. User selects the “Credits & Feedback” button from the main menu.
2. User reads the credits & feedback page, to find out who created the game.

Alternate events flow:

1. User selects the “give feedback” button.
2. User takes the link to the project’s Github page.
3. User exits the game and uses the link to give feedback.

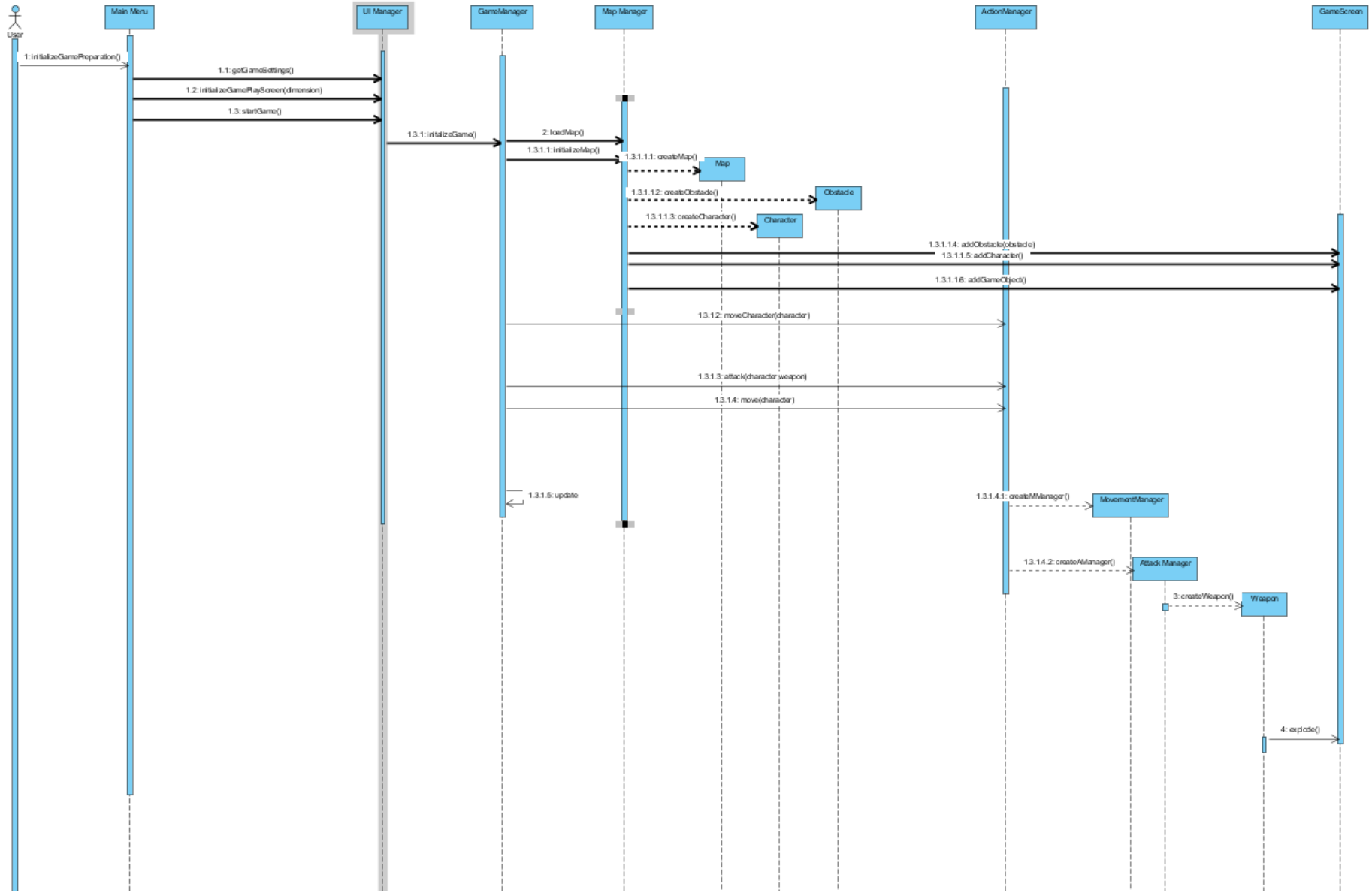
5.2 Dynamic Models

5.2.1 Sequence Diagrams

5.2.1.1 Start Game

Scenario: User starts the game

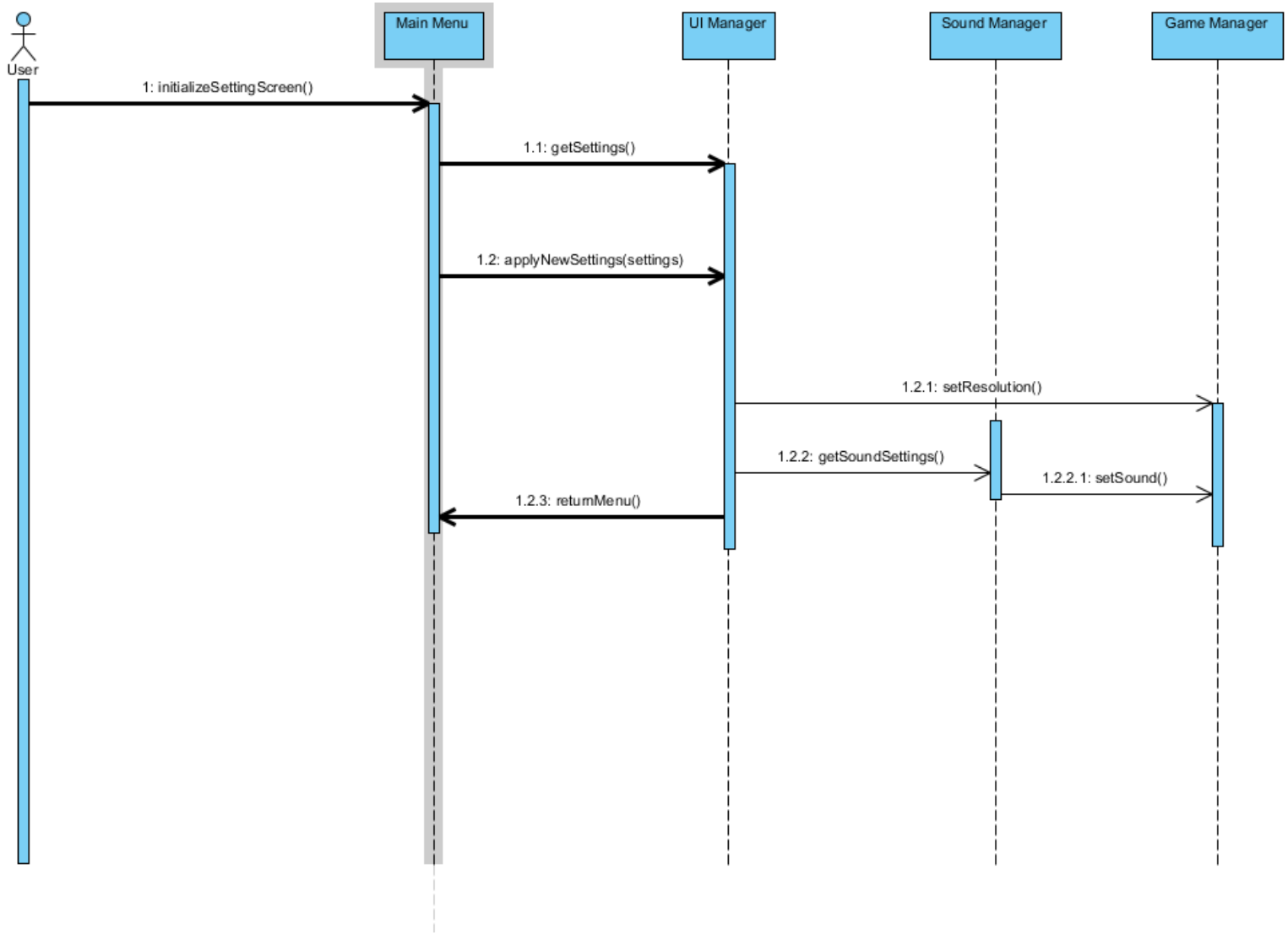
User wants to start the game. He/she enters the main menu. Chooses to start the game, by the help of the UI Manager, user is taken to the game settings screen. Then, after user picks up the settings, that the user prefers these settings are sent to the Game Manager. Game Manager initializes the game by firstly telling the map manager to load the selected map by the user. After the map is loaded and initialized Map manager creates the map and every object inside the map. After the creation of map is completed, by the help of the Action Manager Movement and attack managers are created. Attack manager allows the game to create unique weapons at any given time in the game. To create, unique direction attacks that have an explosion effect on its own collision. This explosion then sent to the Game Screen.


User
Main Menu
UI Manager
GameManager
Map Manager
ActionManager
GameScreen
1: initializeGamePreparation()
1.1: getGameSettings()
1.2: initializeGamePlayScreen(dimension)
1.3: startGame()
1.3.1: initializeGame()
2: loadMap()
1.3.1.1: initializeMap()
1.3.1.1.1: createMap()
Map
1.3.1.1.2: createObstacle()
Obstacle
1.3.1.1.3: createCharacter()
Character
1.3.1.1.4: addObstacle(obstacle)
1.3.1.1.5: addCharacter()
1.3.1.1.6: addGameObject()
1.3.1.2: moveCharacter(character)
1.3.1.3: attack(character,weapon)
1.3.1.4: move(character)
1.3.1.5: update
1.3.1.4.1: createMManager()
MovementManager
1.3.1.4.2: createAManager()
Attack Manager
3: createWeapon()
Weapon
4: explode()

5.2.1.2 Settings

Scenario: User wants to change settings

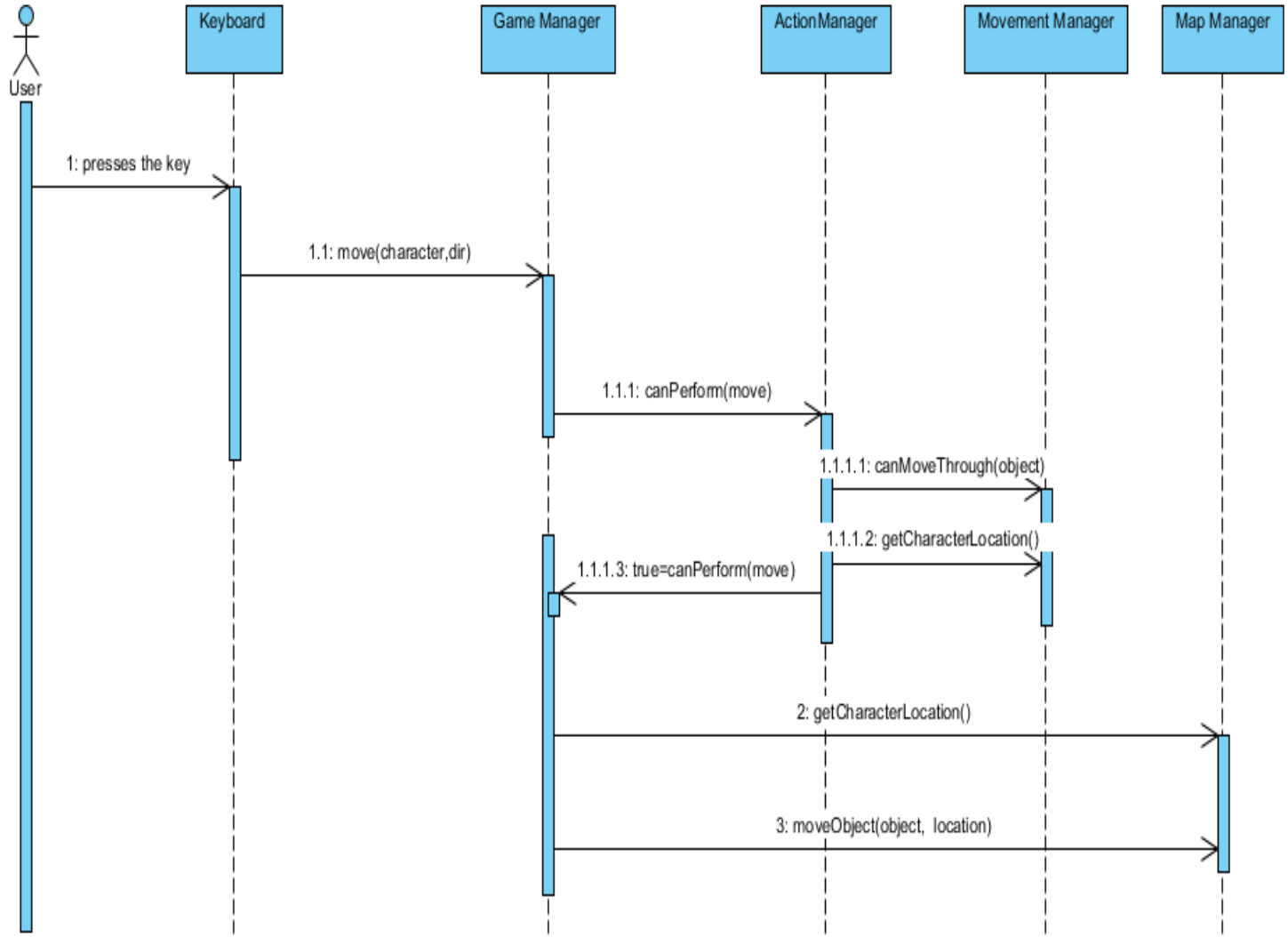
User wants to apply new settings to the game. He/She will enter the setting screen by the help of Main Menu. UI Manager will collect new settings according to the inputs from the user. When user finishes the applying of the game settings, UI Manager will send new resolution to the Game Manager and Sound Manager passes the sound based settings to Game Manager to apply them.


User
Main Menu
UI Manager
Sound Manager
Game Manager
1: initializeSettingScreen()
1.1: getSettings()
1.2: applyNewSettings(settings)
1.2.1: setResolution()
1.2.2: getSoundSettings()
1.2.2.1: setSound()
1.2.3: returnMenu()

5.2.1.3 Movement Mechanism

Scenario: User wants to move to a direction

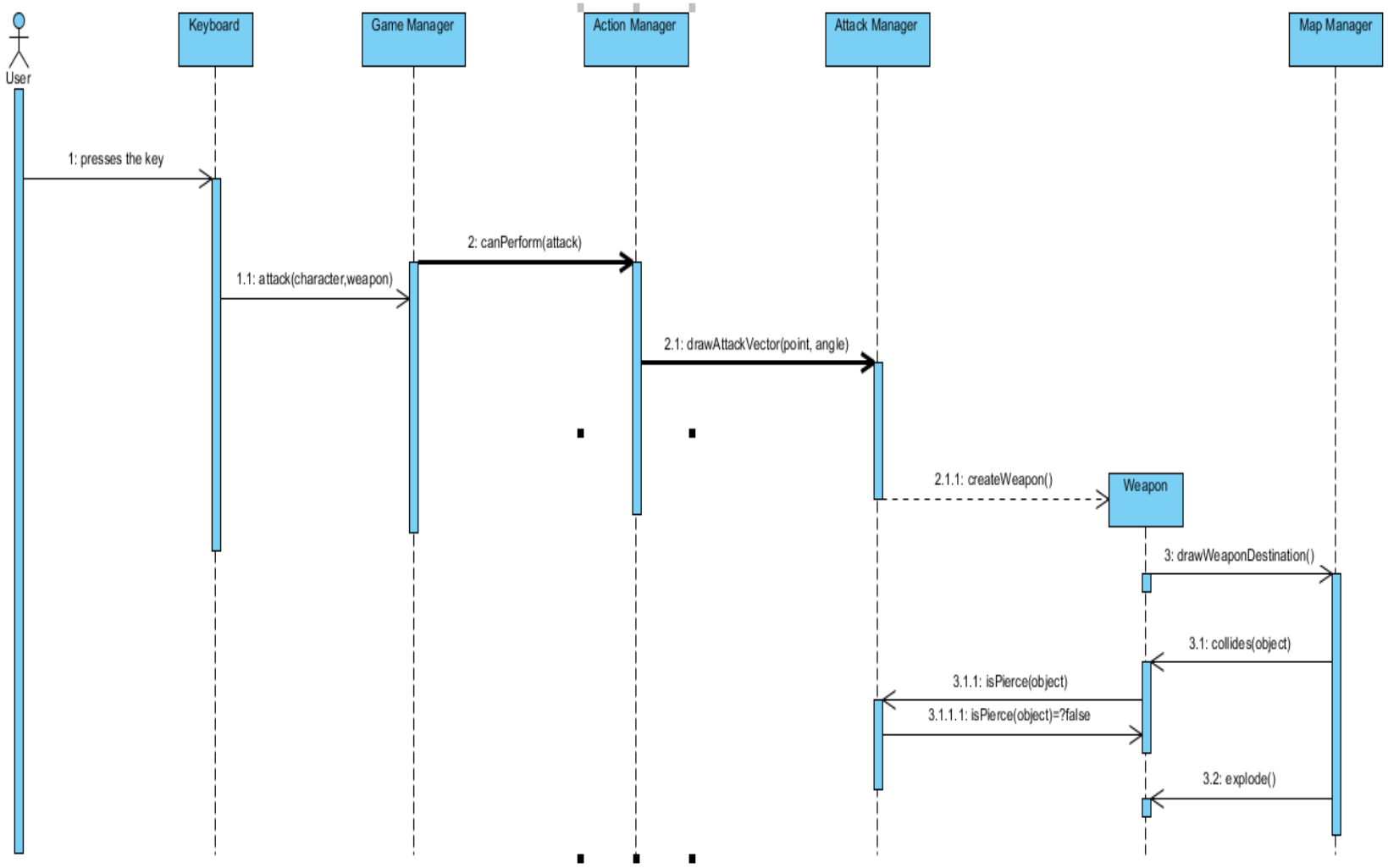
The user wants to move and presses the key from keyboard. The command of keyboard calls the move(character, dir) from Game Manager. Game Manager checks if this action can be performed by sending the parameters to Movement Manager. If this action can be performed, Game Manager gives move object command to the Map Manager and the specified character moves to that direction.


User
Keyboard
Game Manager
ActionManager
Movement Manager
Map Manager
1: presses the key
1.1: move(character,dir)
1.1.1: canPerform(move)
1.1.1.1: canMoveThrough(object)
1.1.1.2: getCharacterLocation()
1.1.1.3: true=canPerform(move)
2: getCharacterLocation()
3: moveObject(object, location)

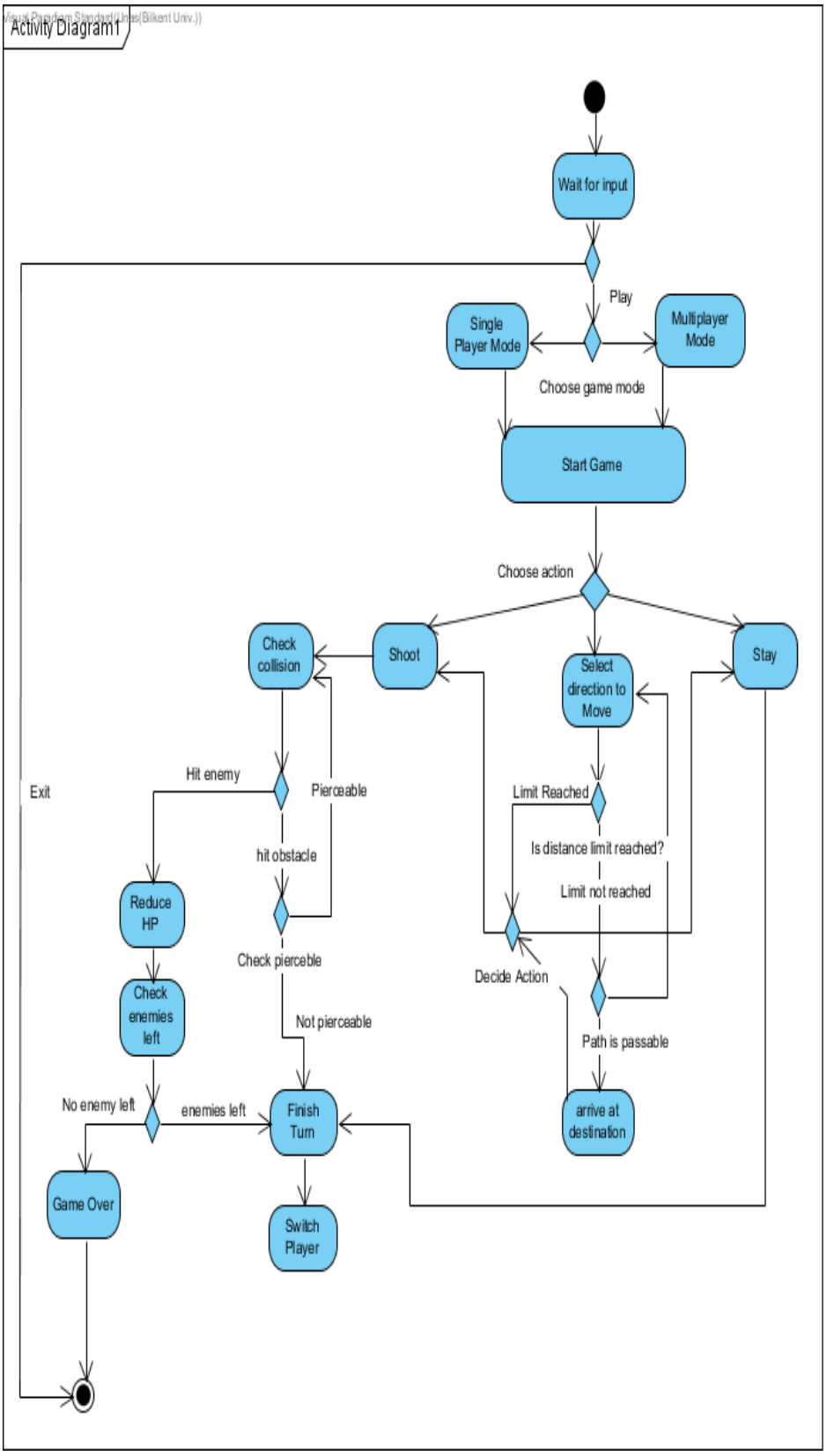
5.2.1.4 Bullet Creation and Fire Interaction

Scenario: User wants to shoot

User presses the keyboard and aims for a specific direction. These parameters are passed to game manager from Input Manager. Game manager checks if this action can be performed and sends the action to the action manager. Action manager calculates the travel of the particle by the help of Attack manager. Attack manager creates the weapon; and draws the weapons attack animation on map. Attack manager checks all tile's on the way of the weapon. When a collision occurs; with the help of map manager it determines if the particle destroys / pierces the object. If it does not pierce; map manager calls the explode function of the weapon and ends the weapons life span.


User
Keyboard
Game Manager
Action Manager
Attack Manager
Map Manager
1: presses the key
2: canPerform(attack)
1.1: attack(character,weapon)
2.1: drawAttackVector(point, angle)
2.1.1: createWeapon()
Weapon
3: drawWeaponDestination()
3.1: collides(object)
3.1.1: isPierce(object)
3.1.1.1: isPierce(object)=?false
3.2: explode()

5.2.2 Activity Diagram



The activity diagram above, illustrates the running system of the game.

Activity diagram explanation:

To start with, the system waits for the user input to check if the user wants to play singleplayer, or multiplayer. User can directly exit the game, where the system goes to final node. If the user starts the game, it can be either multi or single player game. After this choice, game starts, and system initializes the game.

After the game started, user has to either choose move, shoot, or stay. Hence in activity diagram, the process of this is divided into three parts.

In the first case, move, as the user tries to move, system checks if there is any obstacle in the path. If there is, system will check, if it is passable or non passable. Player moves and the player's position, direction changes according to the move. If user reaches the destination he/she can either stay or start shooting.

In the second case, shoot, as the user chooses shooting system checks for the collision. If there is a collision there will be 3 subcases, player, obstacle, and border.

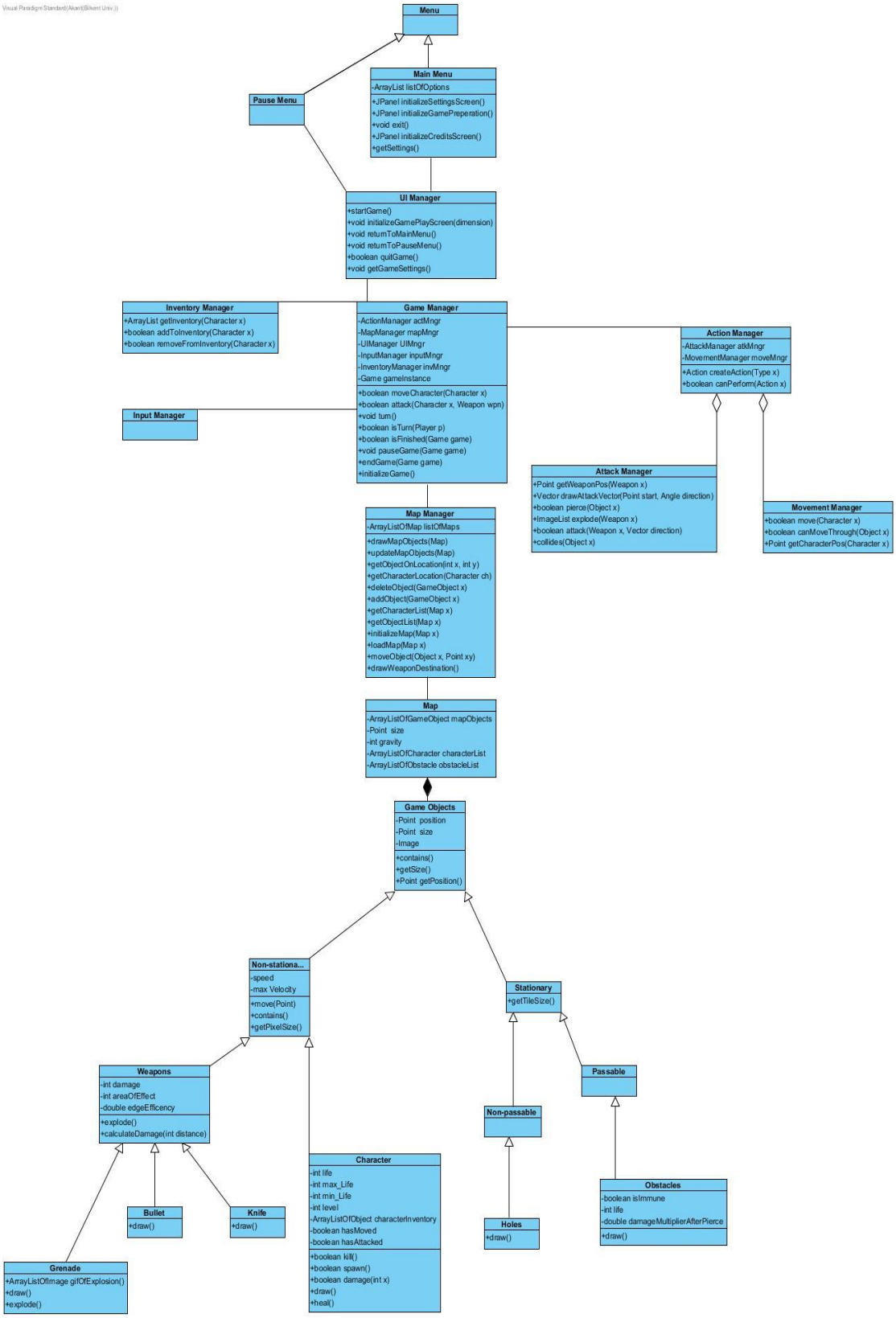
1st case: The bullet hits the player. System reduces the health point of the user and then check if there are any enemies left. If there are no enemies left, game ends, game reaches the final node. If not user's turn finishes.

2nd case: The bullet hits the obstacle. System will check either the obstacle is pierceable or not. If the obstacle is pierceable then system goes back for checking the collision. If not user's turn finishes, and the players are switched.

3rd case: If the user chooses to stay. System will finish the turn and switch the player.

System continues this process until the user exits the game or the game is over.

5.2.3 Object and Class Model:



Explanation:

Map:

This class contains all game objects and stores them. This class also holds a fixed sized display for these objects.

Map Manager:

This class allows the manipulation of any object;

-On a specific location

-Has a specific ID

-This way other classes can access to an objects location and properties given that they know the object ID

In addition this class is also responsible for adding / removing objects to map. Rendering the map by calling draw() methods of each object. Loading map from a saved directory.

Attack Manager:

This class calculates the attack vector by specifying a start and end point. With help of game manager class this class calculates the travel path of projectile and; processes all objects on the path. Determines if the projectile passes / pierces or explodes when contacting these obstacles.

Movement Manager:

This class basically does the checking of a movement action for game manager class to determine the next position of a character after a movement takes place.

Action Manager:

This is a parent class to movement manager and attack manager. It creates the actions temporarily and stores them while the action is being performed in

case of an attack action.

It also helps game manager class by allowing the knowledge of a certain action can be performed or not. This has many parameters like.

-Is there an obstacle

-Is it right players turn

-Can the character move/shoot?

This class checks all these and returns a boolean.

Inventory Manager:

This class allows the management of character inventories. It gets a character as a parameter and manipulates that character's inventory by

-Adding

-Removing

Input Manager:

Gets user input from keyboard and Mouse and sends them to game manager to check if the key pressed needs to be followed with a response.

UI Manager:

This class lets the user communicate with game manager while the game has not started. It sends all the user choices like game settings; map settings;

Main Menu:

Main Menu class initializes the different screens in main menu:

-Settings

-Credits

-Start

-Exit

In addition it sends the user inputs to specific manager classes such as:

-Sound settings go to Sound manager

-Hotkey settings go to Input Manager

-Resolution settings go to Map Manager

All these happen with the help of UI manager class.

Game Manager:

Game manager is the class that does all the logic of the game with association of all manager classes in the project.

It is responsible for the following tasks:

-Start the Game by initializing map for specified settings and spawning the character(s).

-Keep the map manager updated for all the changes done to the objects as game escalates.(Some objects might get destroyed or damaged etc.)

-Keep the inventory manager informed about the changes done to characters ammunitions etc.

While doing these it should also keep associating with input manager to make sure user inputs doesn't get stalled too much while doing other stuff.

Game Object:

This class holds all basic properties for objects in any map. Like:

-position

-size

-image

It can also call contains() method from each predecessor class to get a list ArrayList of points that are inside the boundaries of the object.(This will make it easier to do collusion checks in game manager.)

This class also does the similar call for draw() method which can be used to

draw many different objects with the same method because of inheritance.

Most importantly this class divides into 2 sub classes which are

-Stationary

-NonStationary

This is a crucial division because our non-stationary objects are not tile based while all of our stationary objects are tile based. Tile based means that they are some sort of a rectangle without special curves or edges.

NonStationary :

NonStationary class holds extra speed and max velocity values from its ancestor Game Object. In addition to these since these objects are not stationary they reqiure some sort of movement.

With the help of move() function any class that uses these objects can easly manipulate the location of non-stationary objects.

Since there are currently 2 different type of non-stationary objects in our game this class naturally divides into 2 as well. These are:

-Character

-Weapon

Weapon:

This class holds general variables for any weapon in the game in addition to all the properties it's ancestor classes has.

-damage holds the value that would damage any destructible obstacle or tile for the specified value.

-areaOfEffect holds the value that would apply the damage for each single pixel in the specified radius. areaOfEffect will create a imaginary circle with radius defines as that variable. Every damageable object is going to be

damaged inside that imaginary circle when weapon is exploded.

AreaOfEffect can be set to 0 to make an effect of a rifle for example.Or 10 to make an effect of a grenade.

-edgeEfficency is a value that will only matter if weapon has areaOfEffect. This variable will hold a value from 0.1 to 1.0. It will be basically multiplied with the damage according to the distance from the center of explosion.

This class also calculates the damage done by performing some simple comparisons as follows;

1-It checks if areaOfEffect is 0; if it is then it returns the damage reduced by distance of the shot.

2-If areaOfEffect is not 0 it calculates the damage according to the position of the object to be damaged with respect to the center of explosion. It applies the edge effectiveness value based on the value of areaOfEffect and the distance of object to the center. After this it returns the damage value.

Knife:

This object basically holds a unique image for the "knife" that is going to be shown on the map after the player performs the "attack" which is going to throw a rotating knife at specified point.

Bullet:

Bullet object holds a unique image for the "bullet" to be represented on map when a player chooses to attack with a rifle. It will show a bullet with a trial behind on the map when attack is performed.

Grenade:

Grenade holds a unique image for the "grenade" to be represented on map when a player chooses to throw a grenade.

In addition to other weapons this weapon holds a series of images that simulates an explosion when changed in a rapid succession. This will be invoked when explode() method is called by map manager to show an explosion on map.

Character:

Character class holds all variables that is necessary for the map manager to access in case we need to change these variables.(These variables are an addition to already obvious parameters like position that is inherited from the classes ancestor.(GameObject)).

These additional properties that are unique to a character are:

-inventory array

-current life value

-hasMoved,hasAttacked booleans.

To simplify the class; we can say that it basically holds the current state of the character that is necessary for the game to progress.

Stationary:

This class is the second type of Game Object class. It holds all stationary objects in a map. Since every stationary object in our game is tile based; it has an extra method

-getTileSize() that differs this class from non-stationary objects.

This class has 2 types.

1- Passable

2-Non Passable.

Non Passable:

This class is a non-mandatory class for now; but is necessary for inheritance if

we want to add some more options to the game later on.

Currently it has 2 Types.

1-Holes

2-Obstacles

Holes:

Holes are basically a non-passable terrain that could add depth into the game later on. This class holds a basic image of a hole. For now it has only 1 type.

Obstacles:

This class enables us to calculate the collisions in the game since obstacles are the only tiles that can collide with a weapon projectile. This class has different types according to their variables; but currently it has only 1 image. Later on this class will enable us to create different tiles that can do following:

-An obstacle that could be damaged and pierced

-An obstacle that cannot be pierced or damaged

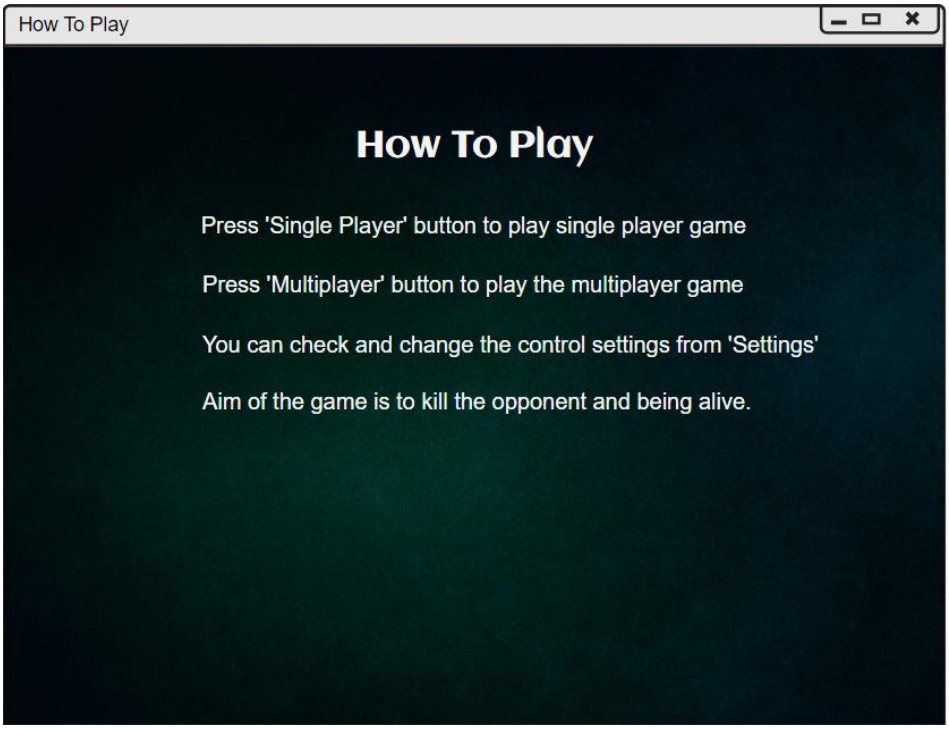
-An obstacle that could be damaged but cannot be pierced.

etc.

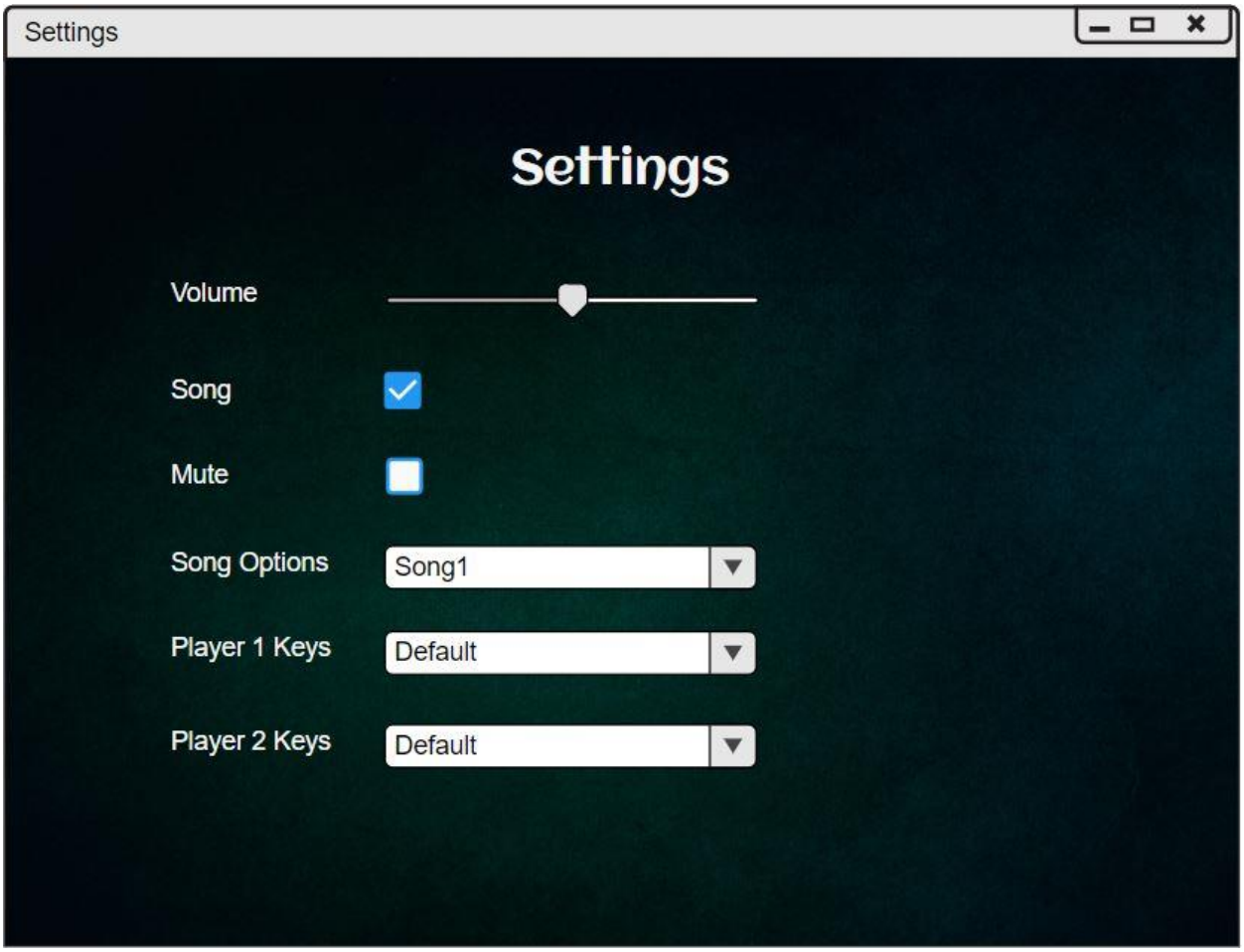
5.3 User interface and Screen Mock-Ups

“Main Menu” is the screen when the game starts and shows the options for the user which are: Single Player, Multiplayer game modes, How to Play, Credits, Settings and Quit. By selecting Single or Multiplayer, user can choose whether to play against computer or 2 players.

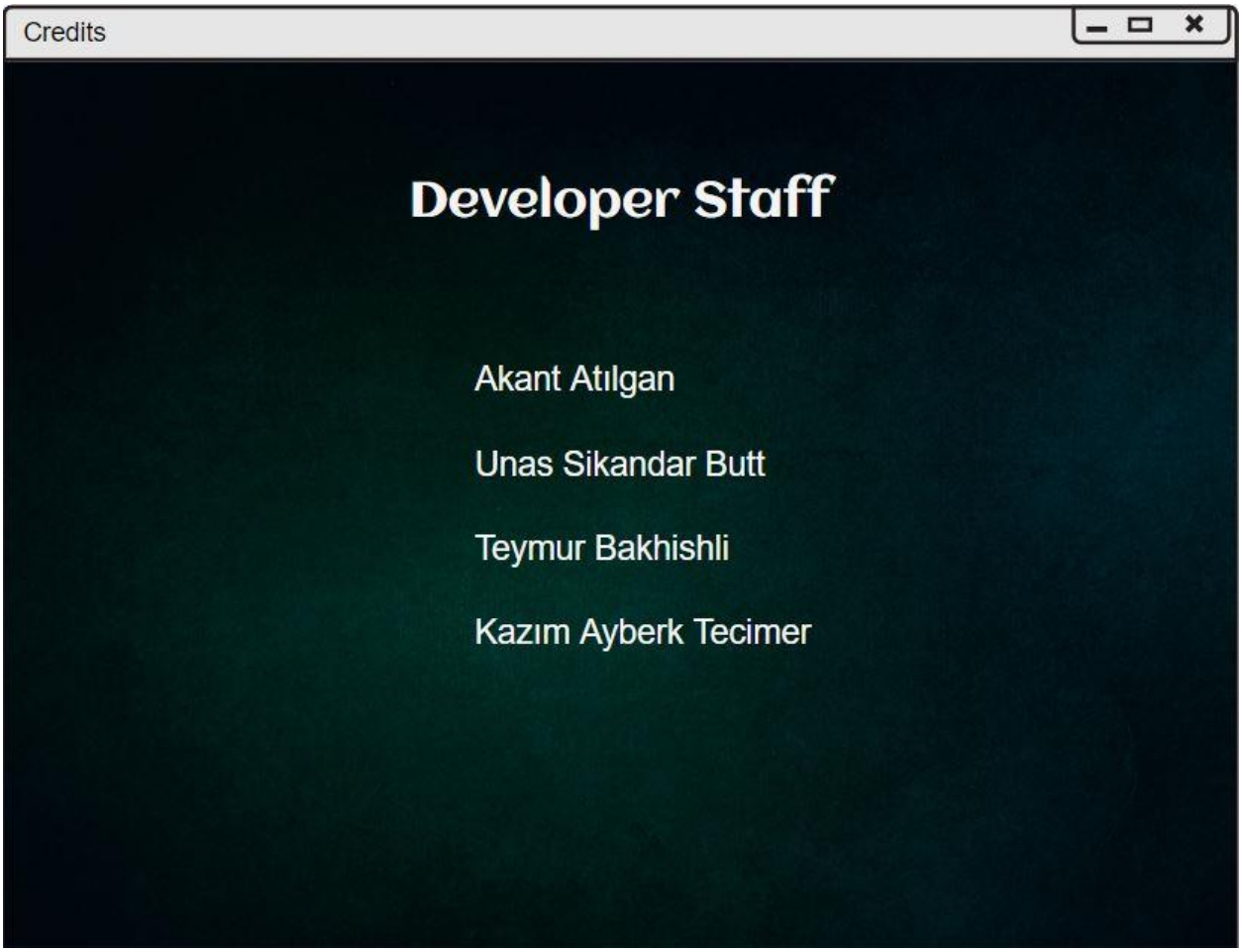
“How To Play” screen gives the info to user about the how and what to do, the aim of the game and the default instructional info to the user.



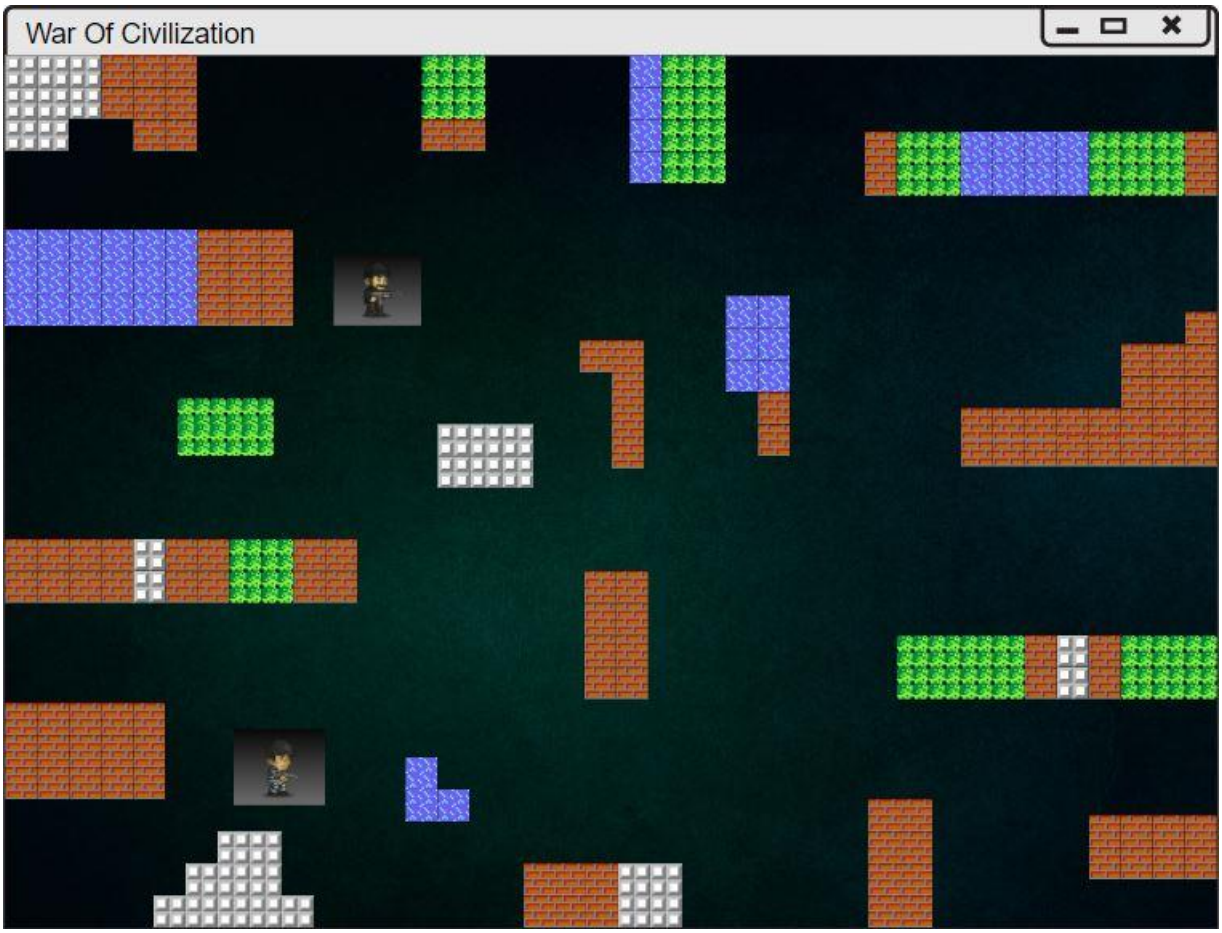
“Settings” screen gives the options to change the music tone, adjust the volume, mute the voice of the game and changing the player buttons for movement, shooting and etc.



“Credits” screen shows the names of game developers.



Game screen will show the map of the game where all the objects will be on this map and the life and inventory bar. In detail, the map will contain the destroyable (stones) and not destroyable (bricks, bushes, water and etc.) Player and his opponent’s character. Other than that bushes will be passable and the other obstacles will be non-passable. The characters of the enemies will be in the black color while the character of the player will be white color.



6. Conclusion

In this report, we basically clarified our analysis to design and implement our game, “War of Domination”. This report includes 3 main parts. First part is the Requirements consist of 2 sub parts (Functional and Non-Functional), where we define especially refine functionalities of our project. In the functional requirements phase, we tried to explain most of the functionalities that the game includes. To do so, we tried to answer “what” questions. Beside this in the non-functional requirements phase we tried to answer “how” questions. After refining the requirements of the project, we designed our system models and drew some models. Our system includes 4 main models.

1. Use case
2. Dynamic models
   2.1 Sequence diagrams
   2.2 Activity diagram
3. Class and Object Model
4. User interface and Screen Mock-ups

To be successful in the design process of use case model, we based on the requirements where we refined before. We examined some use case models on our textbook and it leads us to draw our use case model. Our dynamic models include 2 diagrams, Sequence diagrams and an Activity diagram. In the Sequence diagrams, we try to explain the fundamental decision mechanism and actions of our game. In the Activity diagram, how our system runs and maintains the game is explained.

To sum up, the main aim of the analysis report is to make our life easier in the coding stage and this report will help us a lot for future reports. Therefore, we spent much time on the brainstorming and also on the diagrams.

7 References & Glossary

https://balsamiq.com/

https://www.visual-paradigm.com/