CS102	Fall 2019/20	Project Group	G12
Assistant:	Aydamir Mirzayev		

~ Tower Defense ~

Apocalyptic Games

Sebahattin Utku Sezer

Osman Can Yıldız

Bedirhan Sakinoğlu

Gökhan Taş

Criteria	TA/Grader	Instructor
Presentation		
Overall		

Detailed Design Report

(Version 2.0)

26 December 2019

1. Introduction

Our project is a tower-defense game. Basically, the player builds towers strategically in order to protect his/her territory. Those towers are conceived to eliminate enemies and prevent them from reaching the territory. When enemies reach the territory, they decrease the health of the player. If the player's health reaches zero, the user loses the game.

Before starting the implementation, we need a template that we see which classes and properties we need for our project. In order to do that, we have done Model part so far in MVC. In this paper, our design is described with diagrams and their explanations.

2. System Overview

Elemental Tower Defense game will be written in Java by using the swing, awt, util, io and imageio libraries. Java has convenient libraries such as swing and awt that we use to create our game. Therefore, we are going to employ it. First of all, we have a user interface which is a desktop app created with javax.swing library. User input goes to back-end of the game. The input from the user will be converted to data by back-end program then it will be saved in .txt file which will be used as a data storage. When it is necessary, the data is taken by back-end again and back-end responds to the user in the user interface (front-end).

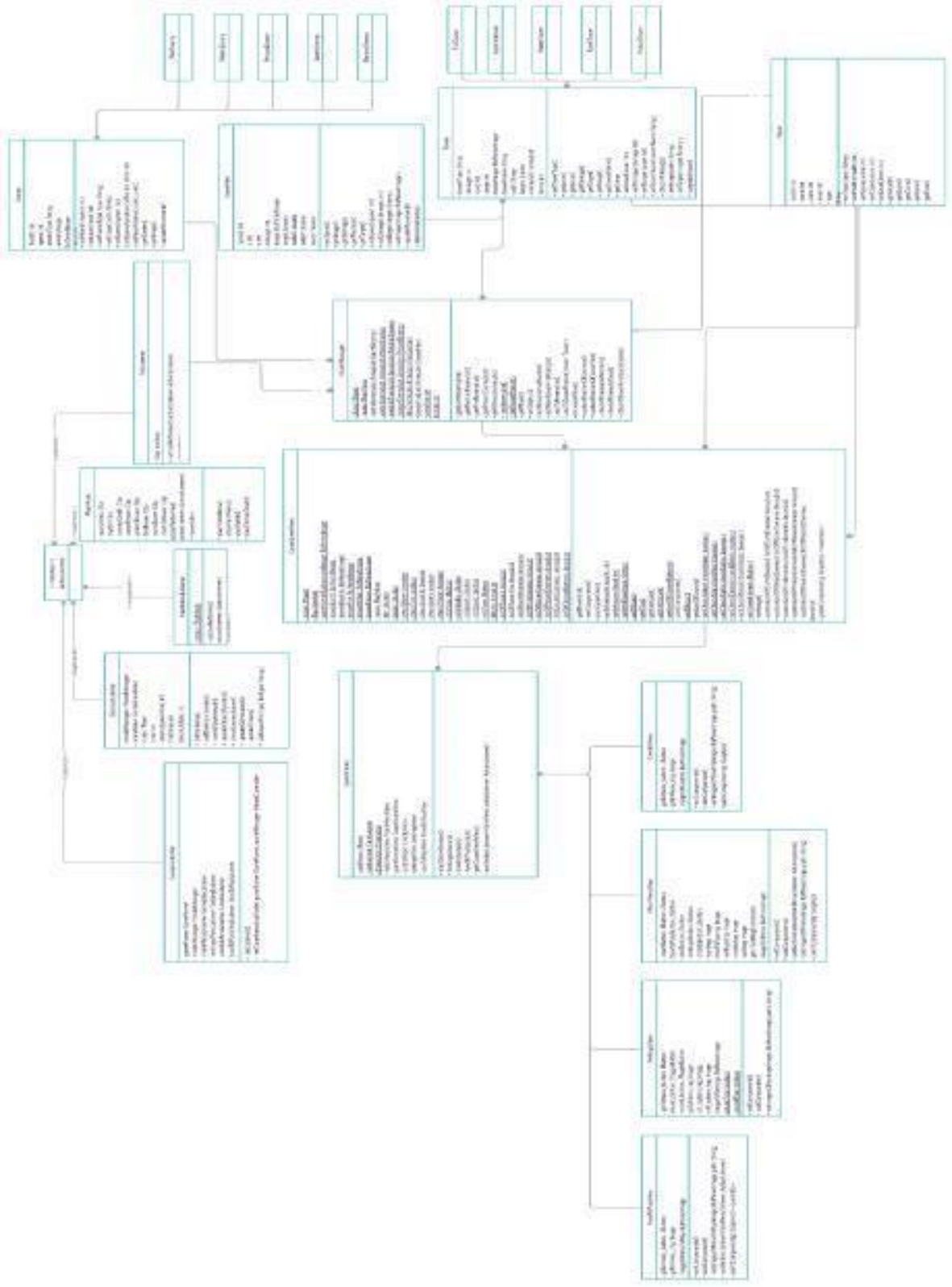
Moreover, we are planning to use IntellIJ. It's simple interface and features will help us to develop our game. In addition, we think that using different program can make us develop ourselves with coding and get used to other coding programs.

To sum up, our game is going to be a desktop application. So, users can play this game on the desktops that Java Environment is installed in it. Also, as a team, we believe that games become much more enjoyable on a desktop because we have some features that user can only see them on the big screen otherwise, they cannot see in mobile. Therefore, it helps us to make our game easily accessible all the time.

3. Core Design Details

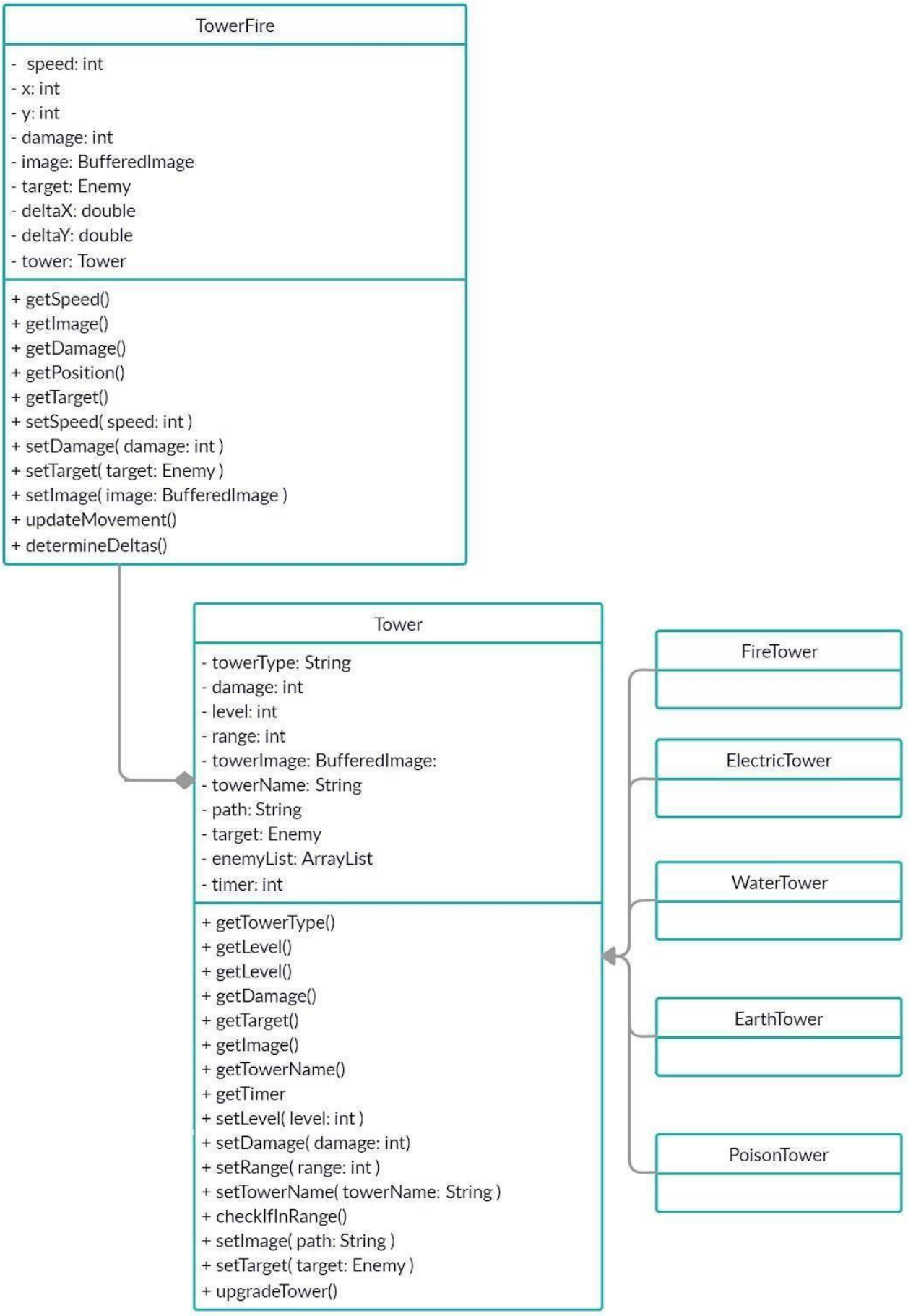
3.1. Class Diagram

Our class diagram is seen below. It includes all model classes for this iteration. Controller and View parts are also added to the class diagram.



In order to see the class diagram clearly, we divided it into pieces to explain one by one.

Piece 1: Tower & TowerFire

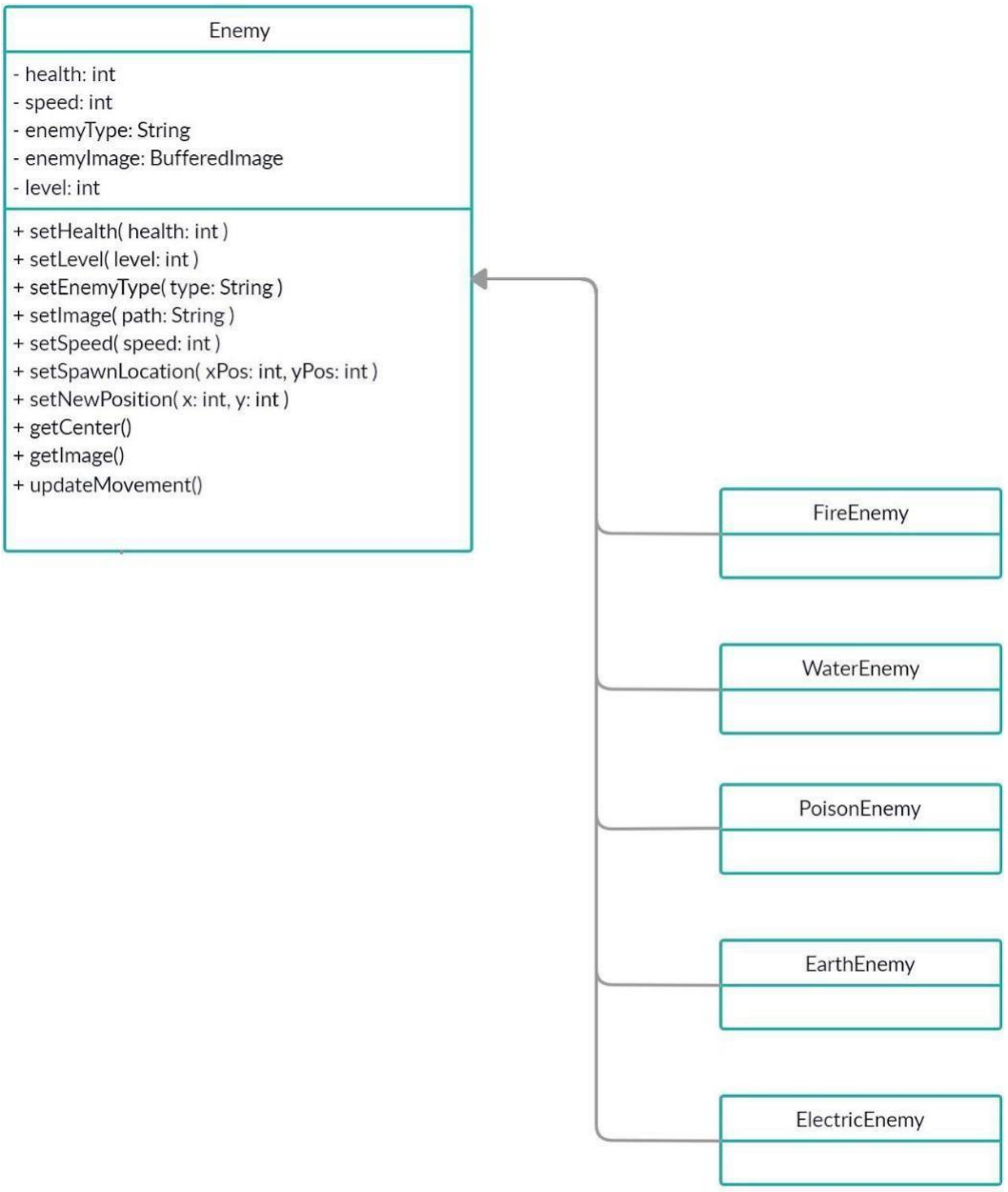


In this class diagram piece, the tower class includes TowerFire as an object. Five different types of tower classes extend the tower class.

TowerFire class is used to make Towers shoot Enemy objects. In order to do this, upateMovement and determineDeltas methods are used to make towerFire objects follow the target enemy. The target, which is determined by the setTarget method, is followed by towerFire object and when it reaches the center of an enemy, towerFire object damages the enemy. In addition, towerFire object deals damage according to enemies' and towers' elements. In order to make it work properly, getDamage method is written to return damage of the tower according to elements.

Tower class has variables xCoordinate, yCoordinate, towerType, towerFireType, speed, level, range, towerFire and image. The initial coordinates of the tower are described as xCoordinate and yCoordinate. The variable towerType and towerFiretype is set to empty string in Tower class. They will be set by setTowerType method and setTowerFireType respectively. These two methods are responsible for setting tower type and fire type according to the given tower type. Speed is a double variable that indicates the fire speed of the towers. When the level, which is an integer variable and indicates the level of towers, is increased the speed of the fire of the towers will be increased automatically. setUpgrade method will be used to upgrade towers. Upgrading the towers increases their levels. When a tower has a higher level than before, it gets additional upgrades. These additional powers differ from tower to tower. For instance, when an electric tower is upgraded, its range will be increased. setPosition method will set the x and y coordinates of the instances and we will get the position of them by using the getPosition method. Moreover, all these types of towers will be overridden which means all towers will have the same properties and methods. Finally, the variable image is the image of the tower object.

Piece 2: Enemy



In this class diagram piece, the enemy class is shown above and it will be extended by 5 different types of enemies.

Firstly, Enemy class has its xCoordinate and yCoordinate integer variables, which will be used in setSpawnPosition in order to determine the spawn location of the enemy and setNewPosition to make enemy moving. Also, there are method to update the location of the enemy along the map, named updateMovement. There are method to determine the enemy image (setImage) and also each type of enemy changes its image according to their type and level. Enemy types determined by the

player’s wave type, randomly. Enemies spawn during the game according to the level of the player wave and the type of the player type. Other classes gets and sets the enemy’s health by using getHealth and setHealth methods.

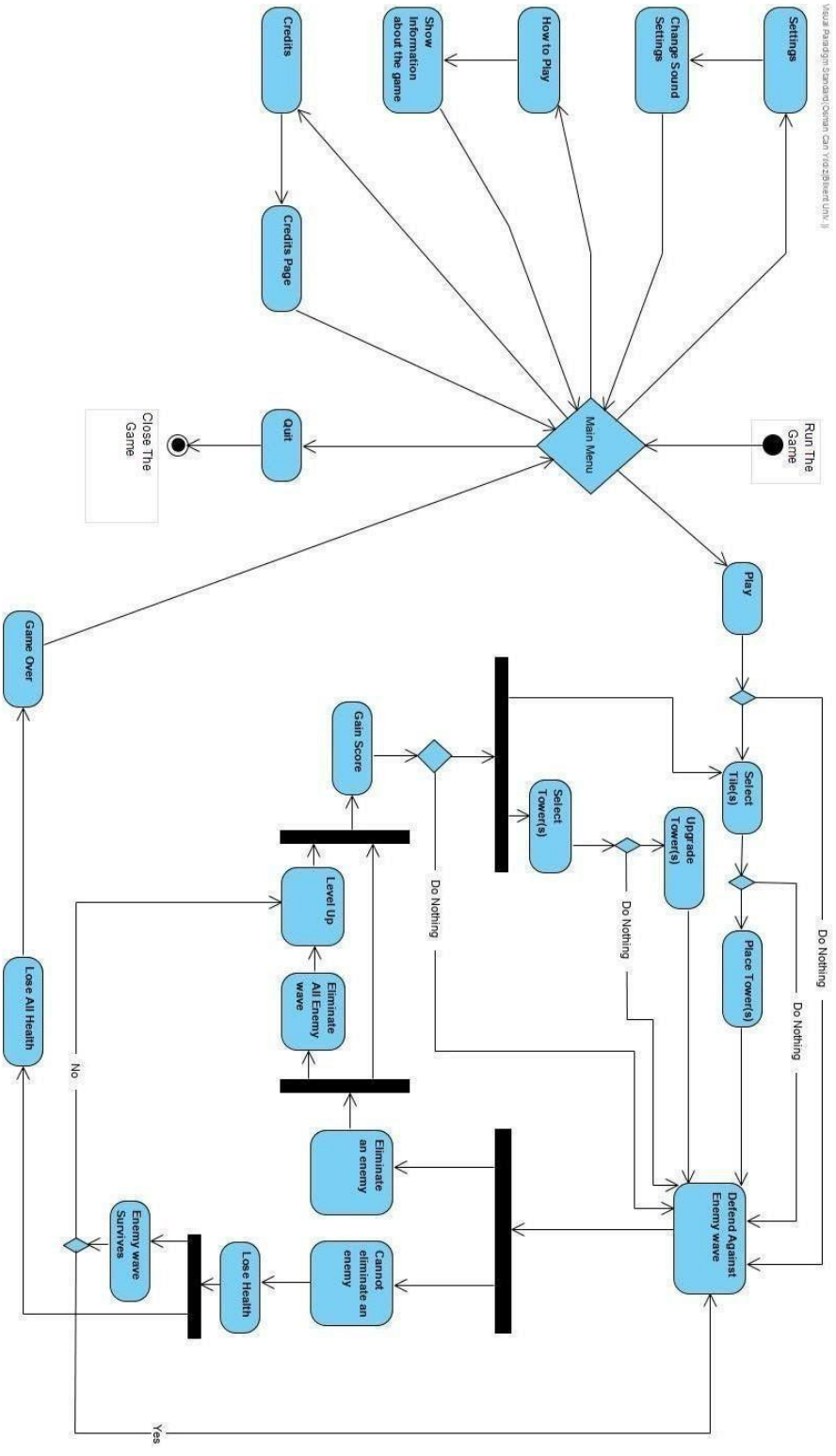
Piece 3: Model Manager

Model Manager Class includes the object Player to set the game according to the Player's type, level, health, coin and score. This class contains many ArrayLists to keep track of TowerFires and different types of enemies. This class has many methods; such as updatePositionOfEnemies which repeatedly updates the location of enemies, updatePositionOfTowerFire which updates the location of the tower fires, addToEnemyList which creates and adds enemies to the specified enemy list and limits the number of enemies for each wave and so on.

Model Manager Class simply keeps track of every object and does many operations according to the object's locations and properties.

This class helped us to store every object and does operations for every frame in the game.

3.2. Activity Diagram



In this diagram, all activities that are in the game are mentioned. First of all, the player runs the game. In the main menu screen, there are five activities that the player can choose: settings, how to play, credits and quit. Settings, how to play and credits screens have “go to menu” buttons. To quit, the user clicks on the quit button on the main menu. To play the game, the user clicks on the play button. In the game, the user has two options which are to select a tile and to do nothing. If the player chooses to do nothing, the player cannot defend their base against enemy wave and loses health. After losing all health, the game is over and the players is directed to main menu. If the player chooses another option, selecting tile after clicking on the play button, the player has another two options which are to select a tower or do nothing. If the player chooses to do nothing, the player defends their territory against the enemy and loses health as there is no tower. After losing all health, the game is over and the user is directed to the main menu. On the other hand, if the player chooses the other option, putting a tower, the player defends their territory against the enemy wave. If the player eliminates an enemy, the player gains score. If the player eliminates all the enemy wave, the stage levels up and the player gains score. However, if the player cannot eliminate an enemy, the player loses health and continues to defend his/her territory until eliminating all waves or not eliminating all waves, which is resulted in losing health or eliminating some of the enemies which are resulted in losing health and gaining score. After gaining score, the player has three options which are to do nothing, to select a tower and to select a tile. If the player selects tower, the user has two options which are to do nothing or upgrade tower. If the player does nothing, he/she continues to defend his/her territory against enemy waves with the player’s initial tower. All possibilities are shown in the activity diagram above.

4. Task Assignment

As a team, we decided to assign the implementation of the program’s parts to each of us, individually.

- Sebahattin Utku Sezer is assigned the implementation of the Enemy, Player, PlayMusic, SessionListener and ModelManager class.
- Bedirhan Sakinoğlu is assigned the implementation of the TowerFire, SessionListener and ModelManager classes.
- Gökhan Taş is assigned the implementation of the GameSceneView, MyListener and Tower and ModelManager classes.

- Osman Can Yıldız is assigned the implementation of View, Assets and UML classes

5. Summary & Conclusion

Consequently, we designed our UML diagram user-friendly and comprehensively to make our game compatible with every type of user. In order to complete our project in the best way, we made task distribution which we think that it might help us to implement the program code properly. As a result, we hope our program will be flawless in the end.