Tower Defense

Autumn 2022

ELEC - A7151

Viivi Alitalo

Antonio Skondras

Samuli Öhman

Edris Hakimi

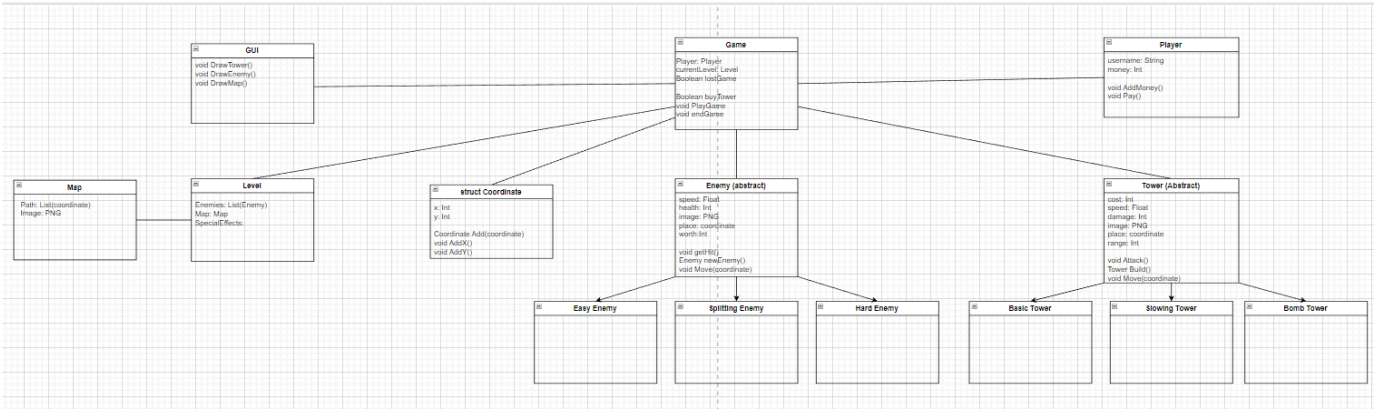
Advisor: Duong Tran

Scope of Work:

Our project is to create a game of tower defense. Tower defense is a game where the player aims to defend their territory from enemy attackers. In the game enemies usually move in waves from some position on the map to another through a predetermined path visible to the player. Whilst the enemies try to get to the end of their path the player tries to prevent it from happening. They do this by placing towers along the enemy's path to attack and hopefully destroy the enemies. Basically, the players objective is to survive by preventing the enemies from getting to the end. If the enemies reach the end of their path the player loses.

We are going to implement all the basic requirements of the project, since that is the requirement to pass. As a team we decided to first fulfill the minimum requirements and after that go for additional features. This is to ensure we get the basics done first in order to pass and then go for a better grade by improving the game. Some improvements we already discussed were creating mor different kinds of enemies and towers, making diverging paths and a high score list. These were some implementations that our groups were interested in and were such that they can be built after the basic implementation is done.

UML:



Credits: Viivi Alitalo

Structure explained:

Game:

The Game class is the root of our classes, so the very top of our class hierarchy. This means it is meant to connect our complete game. Game is the class that is created when the program first runs. The Game-object then proceeds to initialize the conditions of the current game by creating the Player-object and choosing the level and the map on which the game is played.

After that it starts the GUI-object which in turn starts to render the images and visuals on the screen.

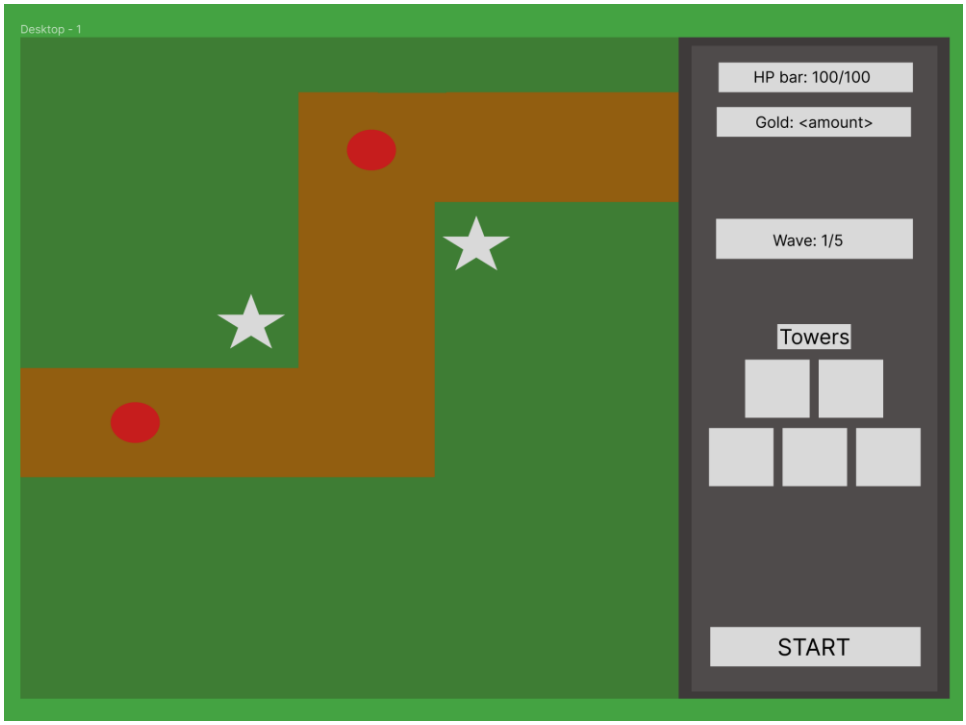
All the towers and the enemies on the screen are stored in a list in the Game-object. Also, the players choices like buying a tower and placing it somewhere goes through the object. When the player destroys the enemies the Game-object rewards them with money so that they can buy new towers to defend themselves with. In the instance of the player buying something the game would first check if they have the money and if they do provide them with the tower they can now place on the map.

The main loop which coordinates the movement and destruction of enemies and towers and checks that the game is not lost also happens in the Game-object. Nearly everything goes through the Game-object. Therefore, it is not useful to explain everything in detail in this plan since it will soon be implemented and explained in the documentation anyway. This is after all a project plan and the outline of the class structure, and the classes are more important.

Graphic design:

In our group the main responsibility for graphic design is on Edris. He is to plan what the game maps, enemies and towers should look like. Also he will think about the size of the enemies and the kinds of effects the enemies can do. The type of background music, the kind of impression that happen if the user loses and so on are included in the graphic design proportion. For this, he will try to use suitable drawing programs to help me draw the characters and other things well. This makes it easier to code the GUI because the parts of it are done and what is left is to put it together piece by piece.

GUI:



Credits: Antonio Skondras

The person responsible for implementing the game's GUI will be Antonio. He will construct the panels that will contain buttons by which the game will be played. He will connect all functions available for the player to the right buttons.

The GUI layout would look something like what is above with some possible changes. We will of course make some adjustments to make the game look better. This picture is an illustration of the layout for the buttons, for example, the start –button and buttons for towers. The panel for buttons will also have bars that will show the player how much health they have and money for building additional towers. The red circles indicate the position of the enemies and how they proceed to the player's fortress. White stars indicate the position of the towers placed by the player. We will have different buttons for towers that can be pressed by the player in order to build them. The buttons will be altered when the icons for towers are ready.

Level:

The Level class is the full outline of what will happen in each level of the game. It has three variables; enemies, which has a container (list) of Enemies, a map of the playing field, and any special effects that we might use. No two levels should be the same, there needs to be a unique factor. It does not matter if it is the map, the number of enemies, the speed of the enemies or what, but there needs to be something different.

Our original plan is to fulfill the minimum requirement and form 5 levels. One of our possible upgrades to our game is to add more levels so this could change. As the played level increases, so does the number of enemies and new special effects could be brought in. The plan is, for example, to develop special effects in such a way that when the level increases, enemies would move faster, and they would not die as easily.

Map

The class Map is the actual outline of the environment the game will be played in. It will have the background, the path the enemies will walk and the towers that will be later placed.

The map class has two variables: an image and a coordinate list. The image represents the background on top of which the enemies and towers are drawn. The coordinate list keeps track of the path the enemy's travel. In practice it has the coordinates in order that the enemies will follow. Our thought process behind the list of coordinates is that the enemies travel towards the next coordinate on the list until they are on the coordinate. Then the direction of the enemy changes towards the next coordinate on the list. If the coordinates are close enough to one another, the traveling of the enemy should look smooth even on round paths.

Struct Coordinates

Coordinates are used almost everywhere in our game. That’s why we decided it would be easiest to make a simple class or a struct to represent them. Because the game is 2-dimensional, the Coordinate-class should have two components: x and y. These are used to describe where on the map the enemies, towers, path, safe zone and such are.

In addition, the Coordinates class should have functions for adding and removing them from each other. This is in order to determine sums and differences of coordinates which in turn help us in moving enemies on the map. Also adding constants to x and y variable should be possible. This is used to change the location of the enemies whilst they move on the map. Also, with this class we can see up to what coordinate the towers attack can reach and if the enemy’s position is that coordinate it is damaged.

Enemy:

Our Enemy class is an abstract class which describes the different enemies we have. In accordance with the minimum requirements, we have three basic enemies. An enemy that is killed immediately when it is hit, another needs multiple hits to be killed and an enemy which splits into multiple enemies that are killed with first hit. In the future if we have time and the resources, we could possibly create more different types of enemies.

The class has at least five variables speed, health, place, worth and image. The speed is how fast the enemy moves which can be increased with the difficulty of the level. Health is the number of times the enemy needs to get hit before it is destroyed, for example once. Place, which is the coordinates of the enemy, so where it is at this moment. The worth of the enemy is how much money the player will receive from destroying it. Lastly, the image, which is what the certain enemy is to look like.

The class also needs some functions. Some of the basic ones are Move, Slow, getHit, destroyEnemy and newEnemy. Move should work as expected and move the enemy to the new position. Slow on the other hand decreases the player's movement speed. The method getHit should deal damage to the enemy so decrease its health. NewEnemy of course creates enemies of that kind so that they can be sent to the level. DestroyEnemy will destroy the enemy when the health of the enemy has run out.

Tower:

The tower class is also abstract, and it will describe the different types of towers that are in the game. Same as with enemies the basic implementation asks for three towers that we can then build upon in the future. The towers will be the defense of the player and attack the enemies. The towers that are going to be implemented are basic, slowing and bombing towers. The basic tower shoots enemies that are within its range and deals them damage. The slowing tower slows the enemy down either for the rest of the path or only in range, that is yet to be decided. The bombing tower will shoot bombs when enemies are in range and can damage multiple enemies.

The class has variables cost, speed, damage, image, place, range and others if need be. The cost is how much money the player needs to buy the tower for their defense. The speed is how fast the tower fires at the enemies and the image of what the tower looks like. The damage is how much the tower hurts the enemy. It can for example take away two health's from the enemy. The place is where the tower has been placed which can change during the game. Range is the radius of the circle that is the tower's range, so all the enemies inside it can be hit by the tower.

The class also needs to have some functions that are attack, build and move. The attack function attacks the enemies in its range, while build makes a new tower that can be built onto the map. Lastly move, which enables the player to move the tower during the game.

Player:

Player is the person who is playing the game. The class is used to store the amount of money the player has and its username. The amount of money is used so that we know if the player can build certain towers or not. The methods can increase the player's money (addMoney) when defeating enemies or decrease it (Pay) when buying a tower.

The username is included in the player for further development of the game. For example, if we wanted to make the leader board, we would need usernames for the players. We would also need points for them. We have not included a variable points yet because we do not have a method in place either which would calculate them. This can be done in the future.

External Libraries:

The use of external libraries in this project addresses the libraries we are intending on using for graphics and multimedia libraries needed for drawing graphics, processing keyboard inputs and so on. In our project it was recommended to use SFML, SDL or Qt. Our group came to the decision to use Qt with the leadership of Antonio who will be implementing the GUI. We chose Qt because Antonio wants to learn to use it, it seemed like the best choice to create a diverse GUI and it is a Finnish company.

Schedule and division of work:

This is the plan of how ideally our project would go. We are aware that this will most likely change and evolve throughout the project. The names that are attached to different tasks does not mean for that person to complete the task alone. The attached name means to just divide the main responsibility for that task to someone so that it will get done. Of course, all team members will help.

Week:	**Due dates:**	**To do:**
Week 45	**11.10.** 23.59 Project plan submission to git	• Create Git – Antonio • UML - Viivi

		• All team members clone repository - everyone • Create project plan – everyone
Week 46	Project plan review with your advisor	• Agree on a meeting time with advisor • Start project • Coordinate – Samuli • GUI – Antonio • Graphic Design – Edris • Player – Viivi • Enemy(1-3) – Viivi • Map – Samuli
Week 47		• Tower(1-3) – Viivi • Game – Samuli • GUI – Antonio • Level – Edris • Extra features?
Week 48		• Gameplay design – everybody • Left over work + bug fixing • Testing class • Extra features?
Week 49	**Week 49 and 50** Project demo to advisor **9.10.** 23.59 Project final commit to git	• Final touches + left over work • Project documentation finalization – everyone (ongoing) • Divide presentation between group members