Project Report

(Part III)

Group 4

Saime Nur **Benliler**
2019402045

Mustafa Can **Coşkun**
2019402033

Fethi **Kahvecioğlu**
2019402051

Muhammed Ikbal **Baykal**
2019402216

IE 201

Z. Caner **Taskin**

A. Tamer **Unal**

January 23, 2022

Table of Contents

1. Game Description

Plants vs. Zombies is a strategy and tower defense game where zombies attack your base and you, as the houseowner, are trying to defend by placing various types of plants.

a. Gameplay

At the start of the game, the player is given some amount of game money which is called “sun”, and the player is given “sun” periodically. Also, the “sun” can be gathered by planting certain plants. The player is permitted to place different types of plants to the playing field which is called “the backyard” in order to stop the zombie invasion by spending required amount of “sun” of every plant has. The playing field consists of 5 horizontal lanes which every zombie and plant are obliged to attack and defend the base from the lane it spawned.

Which plant to place can be chosen from in-game menu located in the up-left of the screen where images of plants are. When a type of plant is placed, it has a cooldown of 4 seconds to place another same type of plant.

Plant Types:

- Sunflower (creates sun with a given interval)
- Ice (shout bullets and zombies move slower with that bullet)
- Peashooter (shout bullets)

Zombie Types:

- Weak Zombie (initial zombie)
- Strong Zombie (has more health than weak zombie)
- Quick Zombie (moves faster than initial zombie and has more health than weak zombie)

The horde of zombie spawn 5 seconds after the game started from the right side of playing field. Different zombies have different attributes such as speed, attack damage, and durability.

Each killed zombie increases the score count by one. The game is won if the predetermined 50 scores are reached. If a zombie reaches the base before this number is reached, the game is lost.

2. Class Diagrams

Here you can see the class diagrams. Definitions of classes and relations among them will be explained later.

Figure 1. Class Diagrams

a. Definition of classes

Definitions of all classes can be seen in the code.

b. Relations among them

Game-MenuBar: It is an aggregation relation. Cardinality of the relation is one to one. In the game, one menu bar is created to place plants on the backyard.

Game-Block: It is an aggregation relation. Cardinality of the relation is one to many. Many blocks are created in the game where flowers can be placed on the backyard.

Game-Slot: It is an aggregation relation. Cardinality of the relation is one to many. Many slots are created in the game to be able to use on Menu Bar.

Game-Sun: It is an aggregation relation. Cardinality of the relation is one to many. In the game, many suns are created on the backyard to gather.

Game-Flower: It is an aggregation relation. Cardinality of the relation is one to many. In the game, many plants are created to be placed on the backyard.

Game-Zombie: It is an aggregation relation. Cardinality of the relation is one to many. In the game, many zombies are created to fight against plants on the backyard.

Game-Bullet: It is an aggregation relation. Cardinality of the relation is one to many. In the game, there are types of bullet for different types of flower.

Zombie-Bullet: It is an association relation. Cardinality of the relation is many to many. Any zombie can be hit by many bullets and any bullet can hit many zombies.

Flower-Slot: It is an association relation. Cardinality of the relation is one to one. Each slot can contain a flower type and each flower type can be placed in a slot.

Flower-Sunflower/Ice/Peas: They are generalization/specialization (inheritance) type of relations. Flower is the parent class; others are the child classes.

Zombie-Weak/Strong/QuickZombie: They are generalization/specialization (inheritance) type of relations. Zombie is the parent class; others are the child classes.

3. Use-case Diagram

Here you can see the use-case diagram below:

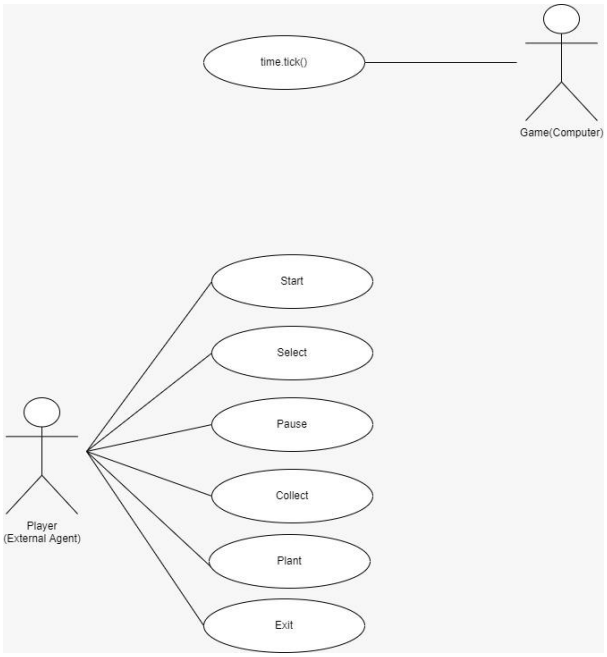

Figure 2. Use-case Diagram

In Use-case Diagram, Player, the external agent, has following controls on the game:

- ***Start:*** When the user starts the game, the background is created, the menu bar is created, and the sun and zombie objects are created at certain intervals according to the given parameters. The sun and zombie objects are created when the check function is true according to the information coming from the time.tick().
- ***Select:*** When the user would like to use "select" feature, the flower image in the selected slot is highlighted (lifting the slot up) if the player has enough money and the slot is not on cooldown.
- ***Pause:*** When the user would like to use "pause" feature, a different screen appears and everything else moving in the game stops.
- ***Collect:*** When the user would like to use "collect" feature, if s/he touches a sun object on the screen, this sun object disappears, and an equivalent amount of sun coin is added to the user's budget.
- ***Plant:*** When the user would like to use "plant" feature, s/he adds the selected flower object to certain blocks created on the screen. The selected flower is also added to the list of flowers.
- ***Exit:*** When the user would like to use "exit" feature, all created objects are deleted and the game is exited.

Computer Interface has the following control:

- ***Time.tick:*** Computer keeps track of time and ensures that changes occur at certain times on a regular basis. It also calculates the accumulated time to create some objects such as zombies, suns, bullets, and help them to continue their movement.

4. Collaboration Diagrams

When it comes to the design part of the project, collaboration diagrams are very important tools to understand the dynamics of the system. Below you can see the collaboration diagrams prepared for the previously created use-cases:

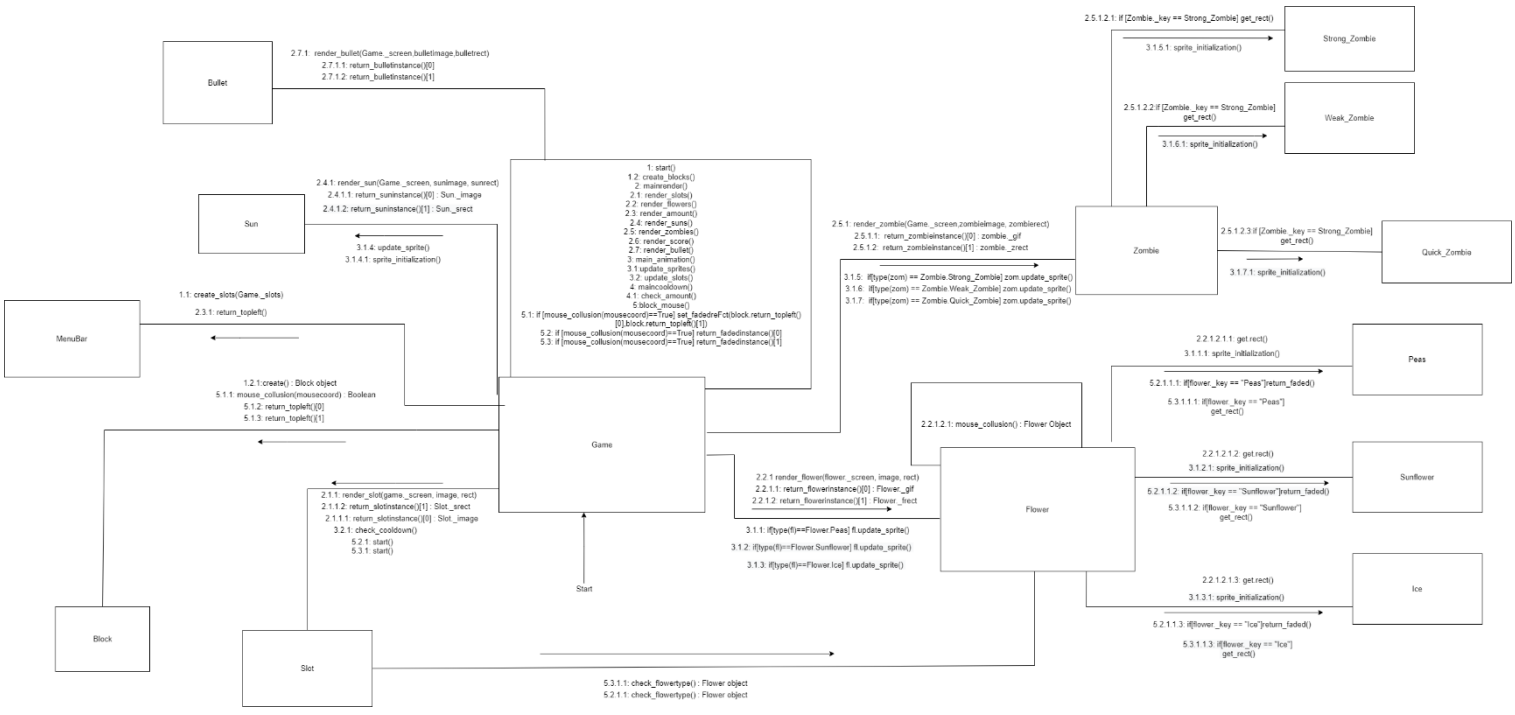

Figure 3. Collaboration Diagram of Start

In Figure 3, you can see the collaboration diagram of "start". First, Game calls the *start* function inside itself. After start, Game calls *create_slots* from MenuBar and *create_blocks* from Block respectively. Thus, areas that can be placed on the screen are created.

Then Game calls *mainrender* function inside itself. With this function, the background is placed first, the MenuBar is added to the screen, and other render functions are called. After the necessary parameters and values are given, the render functions in the Game call the render functions in the relevant classes. *Render_slots, render_flowers, render_amount, render_suns, render_zombies* functions in the relevant class return the necessary images (gifs) and rects to the Game with the *return_...instance* function, thus completing the rendering process and completing the displays on the screen.

Later, Game calls the *mainanimation* function within itself to make the necessary image changes on the screen. Sun, Flower and Zombie objects are animated objects and the update_sprites function called in Game calls *update_sprite* and *sprite_initializaiton* functions in other classes (if the other class is the main class, this function is called from the child classes) and allows the desired moving images to be reflected on the screen. With *update_slots* called in Game, the cooldown status of the slots is checked, and necessary changes are made according to the situation.

Then Game calls the *maincooldown* function internally. This function allows Sun and Zombie objects to be created and displayed on the screen at certain time intervals by calling *sun_spawn* from Sun and *zombie_spawn* from Zombie.

Finally, Game calls *block_mouse* function within itself. Then, with the *mouse_collusion* function from Block and Slot, it checks the appropriate ones, and if the mouse is moved on a suitable block, it takes the faded images of the Flowers and reflects them on the screen.

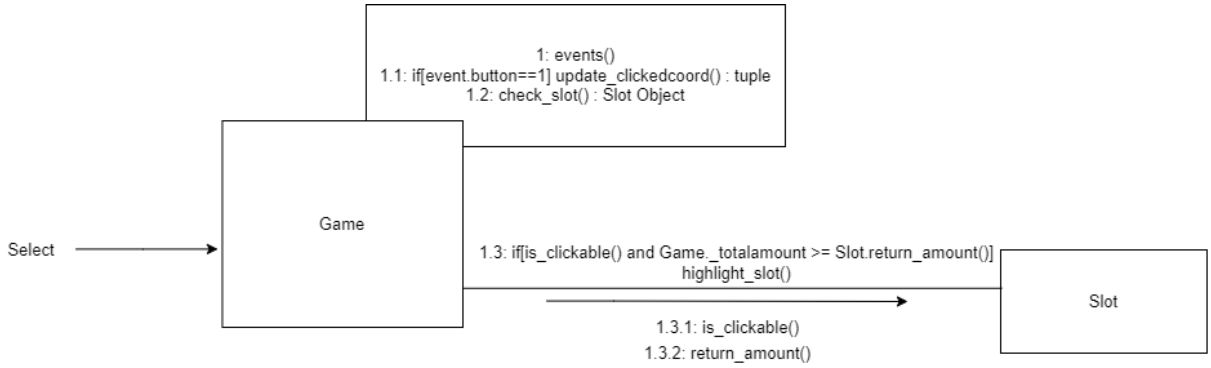

Figure 4. Collaboration Diagram of Select

In Figure 4, you can see the collaboration diagram of "select". Game first calls *events* function to select a flower. To do this, it takes coordinates with *update_clickedcoord* function and with *check_slot* function, it is checked whether the item in these coordinates is a slot or not and returns item.

Then, if the necessary conditions are met, Game calls *highlight_slot* function from the Slot. To do this, Slot checks whether these conditions are met through *is_clickable* and *return_amount* functions.

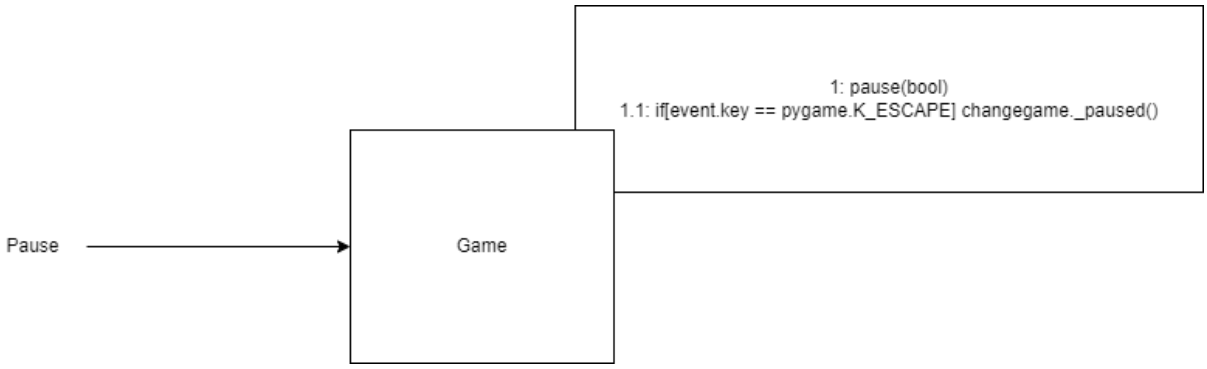

Figure 5. Collaboration Diagram of Pause

In Figure 5, you can see the collaboration diagram of "pause". Game calls the *pause* function within itself to pause the game. In order to do this, Game pauses the screen with *changegame._paused* by taking help from the features of pygame. The game pauses when the ESCAPE key is pressed.

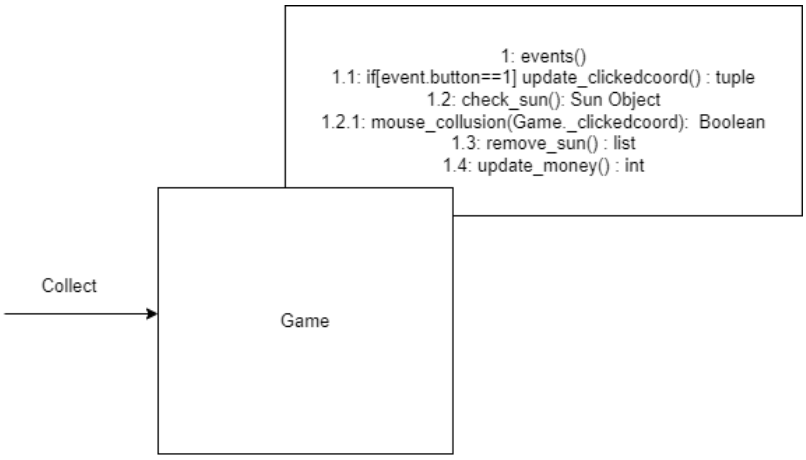

Figure 6. Collaboration Diagram of Collect

In Figure 6, you can see the collaboration diagram of "collect". Game first calls the *events* function within itself to perform the collect operation. With *check_sun* function, it is checked whether the item in these coordinates is a sun or not and returns item (*mouse_collusion* function is used here). Finally, with *remove_sun*, the sun object is deleted from the screen and added to the current money with *update_money* function, and the collect process is completed.

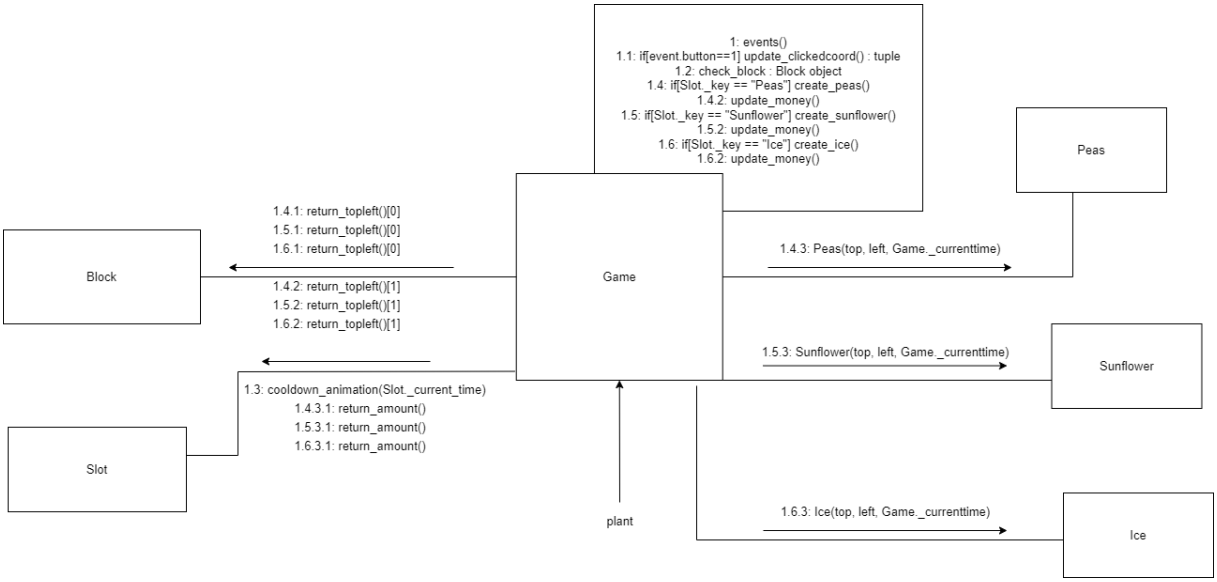

Figure 7. Collaboration Diagram of Plant

In Figure 7, you can see the collaboration diagram of "plant". Game first calls the *events* function within itself to perform the collect operation. To realize the plant, first *update_clickedcoord* function is called within itself, then the used or available status of the selected block is checked with the *check_block* function.

If the conditions are right, the Flower object is created with the *create_*... (Flower name) function. To do this, first the necessary information of the selected block is obtained by calling *return_topleft* from Block. Then the object is created at that point and the balance is updated by calling *update_money* function in Game.

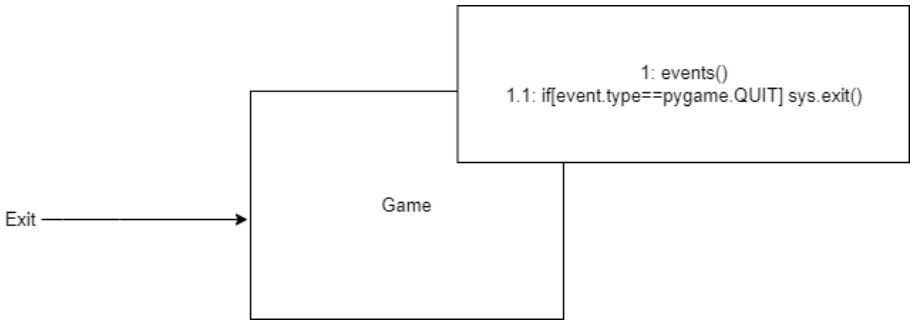

Figure 8. Collaboration Diagram of Exit

In Figure 8, you can see the collaboration diagram of "exit". Game calls the *sys.exit* function (a pygame function) within itself to close the game.

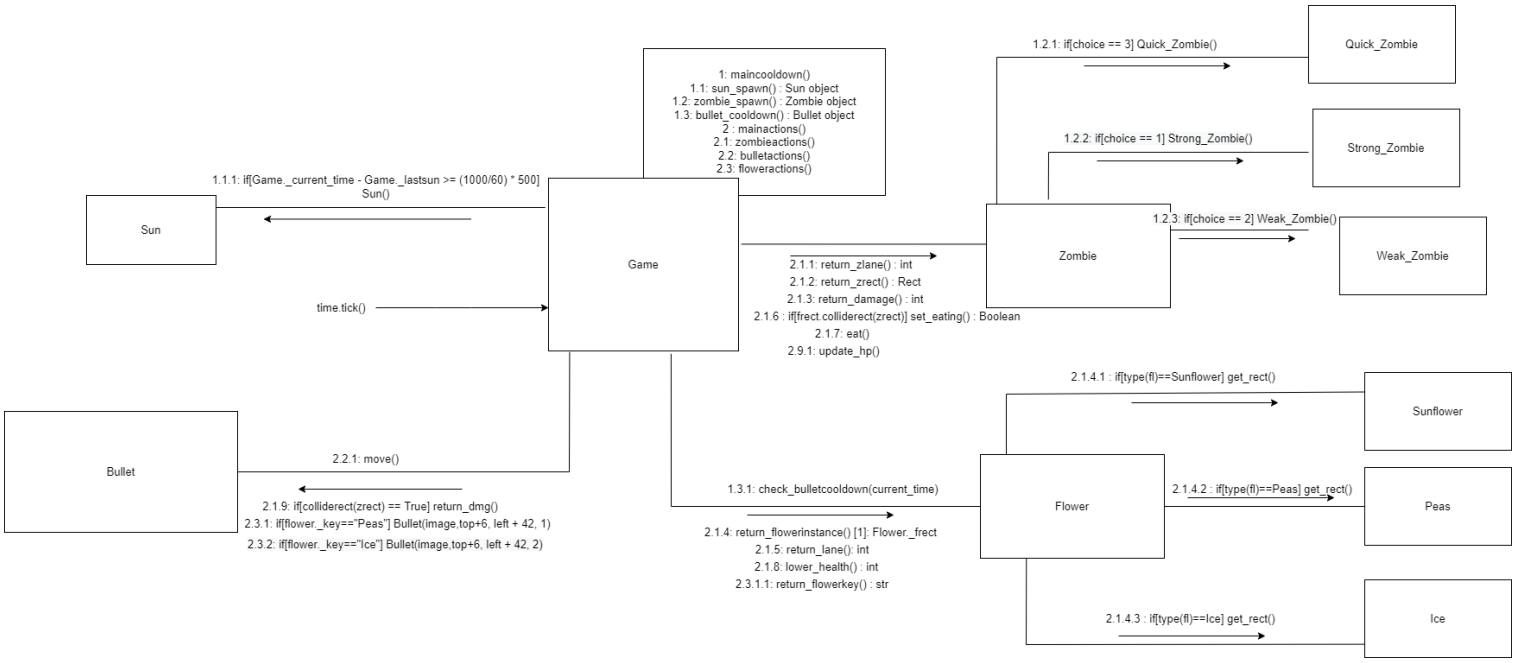

Figure 9. Collaboration Diagram of Time.Tick

In Figure 9, you can see the collaboration diagram of "time.tick". Game keeps time internally. Game calls *maincooldown* function within itself to perform certain events at certain times. These events are the creation of Sun, Zombie and Bullet objects. Then Game calls the *sun_spawn* function inside itself and creates a new Sun object if enough time has passed. After that, Game calls the *zombie_spawn* function inside itself. A Quick, Strong or Weak Zombie object is created according to the selected zombie type.

Another thing time.tick controls is the action of Zombies and Bullets. For this, Game calls *mainactions* function within itself. First it calls some *return_...* functions from Zombie and gets necessary information of Zombie. If this Zombie encounters a flower, *return_...* functions are called from Flower. Then *set_eating* function for this encounter is called from Zombie. *eat* function is called in the same way. *lower_health* is called from the Flower to lower the health of the flower. If it falls below a certain point, the flower is deleted.

To check if a bullet has hit, Game calls the *return_damage* function from Bullet. A collision must occur for this function. Then *update_hp* is called from Zombie to reduce the health of the Zombie object. A Zombie object that takes enough damage dies and is deleted.

bullet_action function is called within Game itself, which triggers *move* from Bullet. In this way, the movement of bullets is ensured.

As a final main action, *flower_action* function is called within Game itself, which triggers a bullet creation from Bullet. To do this, Game calls *return_flowerkey* function from Flower and the appropriate bullet is created based on this information. All of this happens at certain time intervals.

5. User Documentation

"Hey Commander! Welcome to the Plants and Zombies!"

Zombies are invading our home, and you have an army of plants to defend. Think fast and strategically so that we can beat these evil zombies! Now, let's go over the instructions:

Figure 10. Battlefield

As a commander, you have a seat where you can observe the entire scene and have control of the plants and where you can place them. In the beginning, you have 250 Suns, whenever you decide to place the plants, it will cost you the corresponding amount of sun that is written below the slot. From the Menu Bar, you can click and place the plant on the blocks.

Figure 11. Battlefield with a Plant

Now, let's go over the plant types. A good commander is the one who knows its soldiers the best. We have three types of plants: Sunflower, Peas and Daisies. Let's get started:

Sunflower: Sunflower is our sun resource. Without it, this army cannot sustain itself. If you place a sunflower in the battlefield, it will produce suns that will provide you the resource that you need. Cost is 50 Suns.

Figure 12. Sunflower

Peas: It is one of our attacking plants. When you plant the Peas, it will shoot a bullet to the Zombies. Cost is 100 Suns.

Figure 13. Peashooter

Ice: It is also one of our attacking plants. Little bit more costly, 175 Suns, than Peas because they are very precious. Use them wisely!

Figure 14. Ice

Then let's look at our Suns, our resources in the battle. You can either gain Suns from sunflower or collect them when they fall from the sky. To collect them, you must click on it. It will increase your balance so that you can onboard new plants to the army!

Figure 15. Sun

Also, remember that the best commander is the one that knows his enemy. Here, let's look at our evil and disgusting Zombies. There are three types of it: fast, strong, and regular. They are emerging out of their caves and running through our home. When they interact with our plants, they are starting to eat them. But our strong soldiers can successfully beat them with your strategic commands. Remember that if the Zombies reach to our home, we lose the game.

Figure 16. Zombies

Good luck Commander! This home believes in your ability to coordinate the army and win the battle! We wish you the best!

6. References

- https://plantsvszombies.fandom.com/wiki/Plants_vs._Zombies
- https://stackoverflow.com/