FRUIT NINJA

Yosra Emad - 6206
Nouran Ehab - 6216
Omar Alaa - 6294

Farhah Adel - 6318

Introduction

Fruit Ninja is a world-wide arcade game. The game idea is about slicing fruits that jumps out on your screen and slice them with style and combos while avoiding to slice the bombs.

Description

Players:

Users can create new accounts to have their scores saved at the game, and so they can compete with each other in the leaderboard scores

Modes:

- Classic mode:

  You start the game with three lives, you should slice the fruits that appear on the screen and if a fruit falls without slicing you lose a life. Two types of bombs appear on your screen, Fatal and dangerous. If you slice the dangerous bomb you lose a life and if you slice the fatal bomb you lose

- Arcade mode:
  You have a timer of 60 seconds when you start the game. The target of this mode is to compete with the high score you got from previous games or other players got. Missing to slice a fruit won't decrease your score or make you lose, but you will be avoiding the chances to increase your score before the time end, However there are dangerous bombs that will decrease your score by 10 points. You should avoid them. This mode contains special fruits that earns you more score points than ordinary fruits.

- Zen mode:
  You have a timer of 90 seconds when you start the game. Zen mode is exactly like the arcade mode except it doesn't contain special fruits.

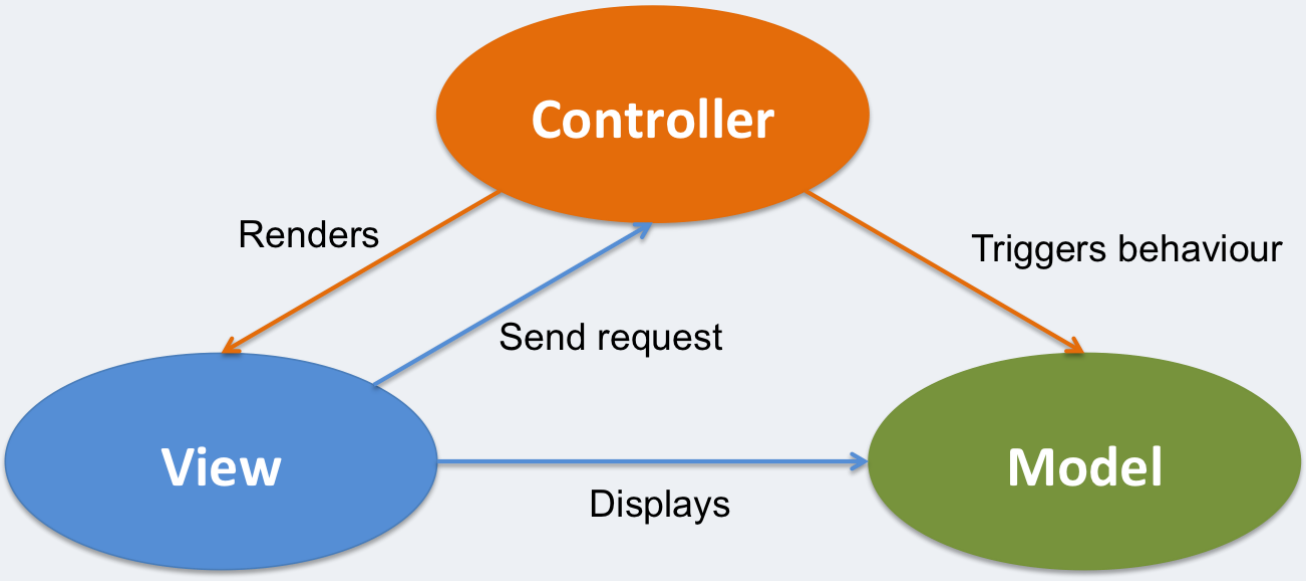
Combos:

You can perform a combo if you sliced 3 or more fruits in the same mouse drag. Fruit combos increase your score by the amount of the fruits that was slices in the combo. Say you slice-combo 4 fruits, Hence you take 4 points for slicing the 4 fruits and extra 4 points for the combo

Project Structure

Architecture Pattern:

The project is using the MVC Pattern to help manage the application's components and maintaining a clean code and more core reusability within the application.



Views:

Contains the application views and pages and the needed utilities and tools to run and navigate between them.

- pages contains the application views (fxml pages) with its code behind associated class.
- CSS contains the needed stylesheets for the fxml pages
- fonts contains the needed fonts for the application
- guiUtils contains Utility function that are related to the gui and the navigation

Controllers:

Contains the application controllers that acts as the middle-man between the models and the views as it fetches the data and logic from the models and updates and updates the views. And to maintain clean code and decrease coupling between classes and controllers. Hence, every controller is associated with only one page.

Models:

The Models package contains all of the application entities and the logic of the game.
The Models and logic are separated to multiple layers. Each layer is concerned with a specific set of tasks that are related and it's responsible for delivering the needed input and data for the next layer so that the game cycle is completed successfully.

- **Audio:** contains the needed logic to run music and sound effects in the game.
- **Entities:** contains the game objects and model classes needed for the application like Fruits, Bombs or Special Fruits.
- **guiUpdate:** contains a set of classes set applies the observer design pattern which helps us update the gui in the game page.
- **Logic:** contains the game logic, game loop and cycle that runs the game while keeping track of the user input and injecting it through it.
- **Modes:** contains a set of classes that applies the strategy pattern to choose a specific algorithm at run time (Different Game mode like classic, Zen or arcade) so that each one of them can run with different properties
- **States:** contains set of classes that applies the state design pattern, that helps us in changing the state of the game from pausing to playing.
- **Users:** contains the models and entities related to the user and his player profile and keeps track of his score.
- **Waves:** contains a set of classes that applies the builder pattern that we use to create specific wave type for each mode on multiple steps.

Data:

The data package contains two layers:

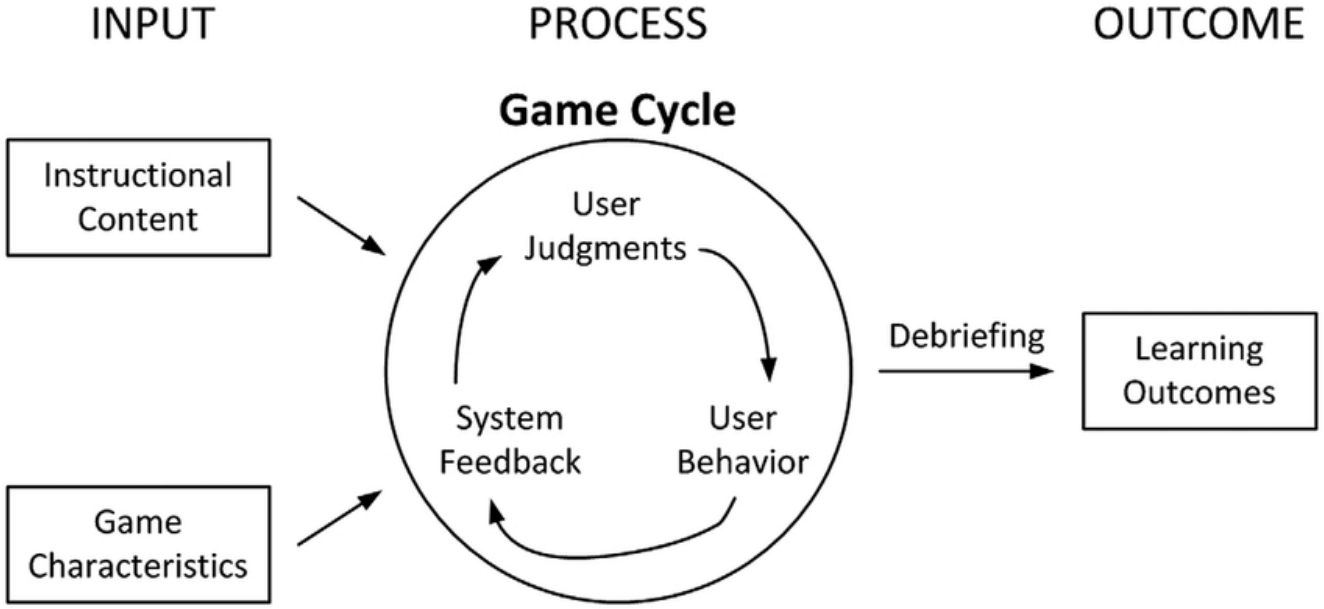
- **Services:** contains the classes that represents the layer that deals directly with the XML files in the application.
- **Repository:** Contains the classes that are above that data access layer and acts as the middleman between the services and the application models that requires these types

Workflow

When a new game starts the program go through a cycle that keeps it in the game cycle with the needed conditions.

First a strategy is of game is selected based on the user selected mode. Each strategy performs differently than the others in the gameplay and hence, the actions that happens in the game.

The Game Cycle:



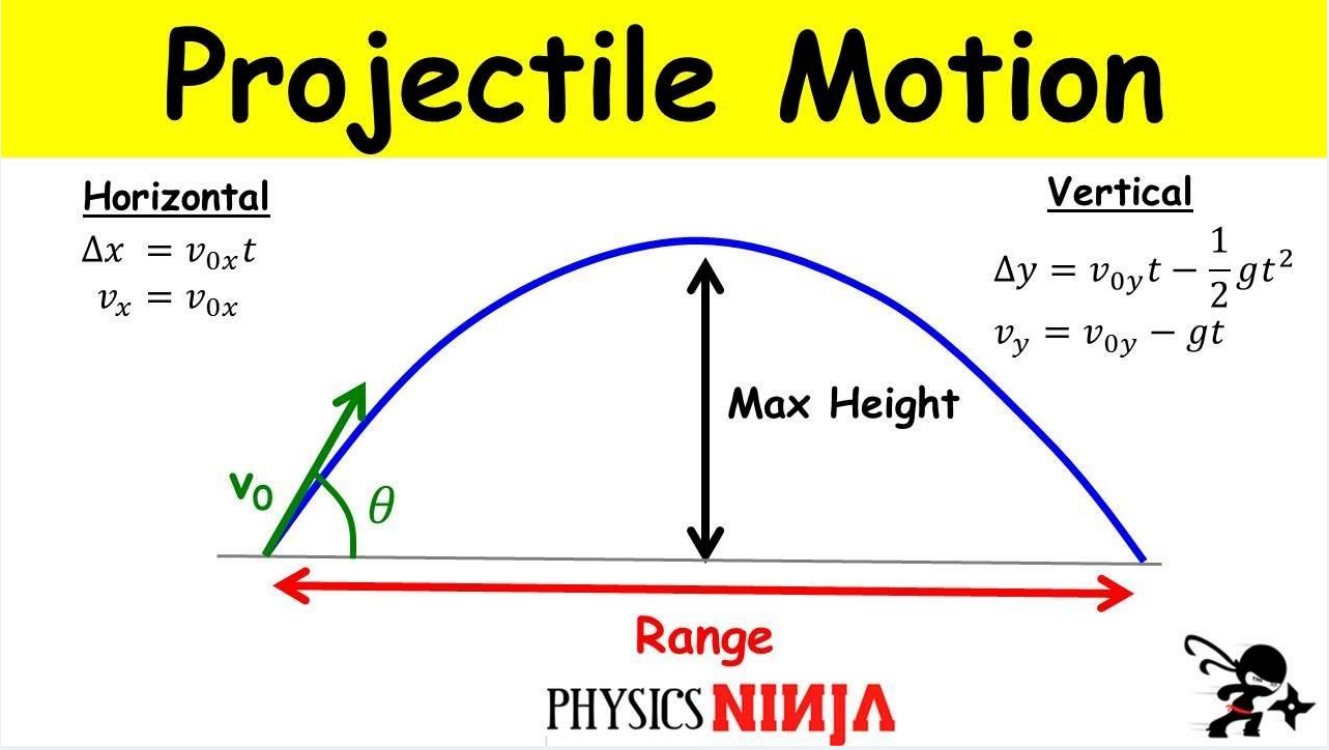
The game consists of waves of different fruits and bombs that comes up every specific time. To generate the fruits, we use the rounds. A game is a set of repeated rounds.

A **round** is something that keeps happening over and over again till some condition is met.

Round Steps:

- Calling the **GameProperties** which keeps track of the wave number and game current status to check the number of the wave and determining the difficulty depending on it.
- Calling the **GameLogic** to start the process of the round.
- *GameLogic* calls the *GamePlayService* that encapsulates the logic of creating a wave and Generating a wave of fruits and bombs depending on the difficulty (the more you play, the harder it gets).
- Calling the **GamePlayActions** to start the process of throwing the game objects to make them visible to the user.
- Repeat all the steps all over again until the user loses or time ends.

The Game Objects Movement



GamePlayActions Class acts the middle-main between the animation package and the process of creating the fruit wave and throwing it

The Game uses a special type of movement, We use the projectile rules to throw the fruits from random different places with random different angle; which makes the gameplay more fun and more realistic.

Animations Package generates a random coordinate for object, and throws it with specific velocity and acceleration depending on the difficulty.

Hence, Gameplay actions rule is to associate every sprite (Game Object) with a projectile object the move on a specific path, and updates the movement of the sprites while keeping track of the game logic.

Detailed Movement Info:

Projectile class creates an object that will be thrown. This object has initial coordinates which gets updated in the movement depending on the time. It also has a velocity coordinates to keep track of the object velocity during the movement.

ProjectileShooter class takes care of the specific angle, speed and acceleration that the projectile should be thrown with.

ProjectileUtilities class helps defining the required speed and acceleration based on the difficulty

Design Patterns

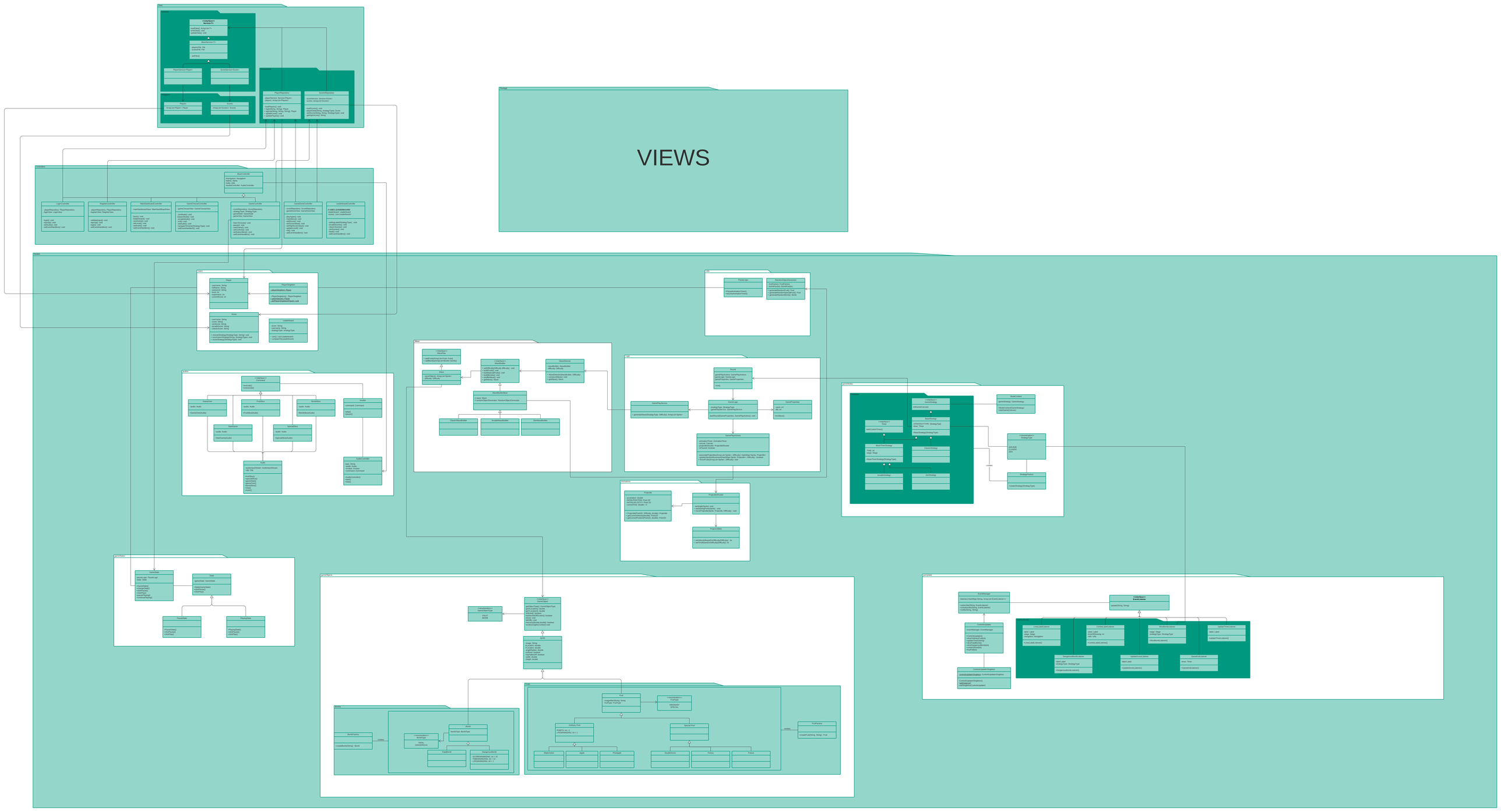
The application used multiple design patterns that helped us. The application used multiple design patterns that helped us in solving lots of common problem and maintaining more cleaner and efficient ways of managing the code in solving lots of common problem and maintaining more cleaner and efficient ways of managing the code.

Design Patterns used:

- **Factory Design Pattern:** used multiple times in creating the game objects like fruits and bombs.
- **Builder Design Pattern:** used to create different wave types depending on the mode of the strategy and to stage the process of creating the wave on multiple steps to make it easier and more maintainable and customizable.
- **Observer Design Pattern:** used to communicate between classes and to notify certain controls when some actions happen and it was mainly used to update the controls like score, lives etc.
- **Strategy Design Pattern:** used to specific the run-time algorithm (aka the mode of the game) when the user chooses it, and change behavior depending on the user preferred mode.
- **State Design Pattern:** used to save the game state and to create ease in swapping between different states like the pausing state that have different behavior in the game. And to allow us to the save the game state and reload it later.
- **Singleton Design Pattern:** used to create a one object of specific class that will be used through the program. we used it to create the profile for the player that contains his level and experience, and the controls updater that manages the events which are being subscribed to all around the application.
- **Command Design Pattern:** used for playing background audio throughout the game and playing sounds when slicing fruits, special fruits or bombs.

The UML Diagrams.

(Class and Sequence Diagrams)


VIEWS