Assignment 3

We decided to use the **model-view-controller** architecture. It divides the application into three main categories:

- **Models:** these are the objects we are using in the game (such as the asteroids and the player).
- **Views:** represented by the GUI of the game which is composed of different screens
- **Controllers:** which are the 'bridges' between views and models.

There are multiple advantages of using this architecture instead of another one:

1. **High Cohesion:** we were able to separate our classes as much as possible so that each class has a well defined job.
2. **Reusable code:** separating the code as much as possible helped us to increase reusability as there are functions that are used across the different screens, for example, the *exit()* function. Moreover, both Register and Login screens are leading to the Menu screen and so on.
3. **Different types of components:** as we have different types of components we can also use different types of files. The view part of the application is represented by FXML files which make it easier to create the graphics of the game whereas, the Java code is handling the user requests and the objects used in the application. Moreover, by default, each FXML file can get a controller assigned to it. In this way, whenever the user triggers an event, the controller will call a function and it will use the right model to fulfill the request.
4. **Easy testing:** Unit testing becomes easier with the MVC model. That is because the graphics of the game is hard to test and we noticed it when we tried to use the ***Node abstract class*** *(to add the visual representation of each object on the screen).* To solve this problem we had to decouple the visual part of each element from its other attributes, that helped us to finally be able to test the components efficiently. Moreover, we were able to vary our testing techniques, as we used integration tests for the interaction between models, views and controllers.

The Models we used are:

- **User:** stores the username and password and is used by the AuthenticationService for registering a new user or logging in an existent user.
- **SpaceEntity**: the abstract superclass that contains the main structure of all the elements that are showing up during the game. These are: player, asteroids, hostiles, shield and bullets.
- **Game:** contains the information about each game. This is then saved in the database.
- **LeaderBoardGame**: stores the information of each game that is visible on the LeaderBoard Screen. This object hides the Game information that needs to be private: the *game Id* and the *Username* of the player (when they use an *Alias*).

- **Shield:** displays a shield surrounding the player whenever they die or they press the **DOWN** arrow key (under certain conditions). The shield protects the user from the collisions with the hostiles and the asteroids.
- **Player:** differs from the User model as it stores the attributes needed during the Game. These make the spaceship move on the screen, shoot, collide with other objects. It also helps tracking the remaining lives and the points earned during the game.
- **Hostile:** contains the information about the enemies. It has a similar structure to the Player model, however, a Hostile is not able to activate a Shield.
- **Asteroid:** contains the information about the type of asteroids to be displayed on the screen (small, medium, large) based on that the player will earn a different amount of points. Each asteroid will also move at different speeds.
- **Bullet:** bullets are shot by hostiles and the player. Whenever a bullet is shot we address it in the player's or enemy's direction and, in case of a collision with another element, it will trigger a different event.

The Views we used are:

- **Main Screen:** contains the buttons leading to the Login Screen, Register Screen or the exit from the application.
- **Login Screen:** contains the fields to enter the username and password, it is directly connected to the AuthenticationService that will check if the information inserted is valid so that it will redirect the user to the Menu Screen. Otherwise, it will display an error message on the screen.
- **Register Screen:** works similarly to the Login Screen with the difference that we have an extra field for validating the inserted password. After the registration is completed and the data is stored in the Database, using the AuthenticationService, the user gets redirected to the Menu Screen.
- **Menu Screen:** contains the Play button which redirects to the Game Screen and starts the actual game, the LeaderBoard button that leads to the LeaderBoard Screen and the Exit button to exit the application
- **Game Screen:** contains the game view where the player, asteroids, hostiles, bullets and shield are shown. It gets updated based on the different events triggered that are handled by the GameScreen Controller.
- **LeaderBoard Screen:** it displays the LeaderBoardGame model in a table with the top 12 scores ordered by the amount of points earned by the player.

The Controllers we used are:

- **Main Controller:** it handles the connection between the different controllers so that we can switch the current view.
- **LoginScreen Controller:** it handles the login page so that the user can be authenticated.
- **Register Controller:** it handles the register page so that the user can be registered.
- **Menu Controller:** it helps switching between the Leaderboard and Game screens.
- **Game Controller:** it updates the Game view whenever certain events are triggered.
- **LeaderBoard Controller:** it helps displaying the data of each Game parsed from the Database.