Module Code:	CS2JA16
Assignment report Title:	Drone Simulation GUI Description
Student Number:	28000050
Date:	10/12/2020
Actual hrs spent for the assignment:	38
Assignment evaluation:	Interesting, Educational, Informative

Giorgos Philippou

1. Introduction

1.1 Abstract

The following report is a description of the Drone Simulation GUI project. The procedure of making such a project required a lot of time of reading and understanding. The assignment targets on learning, is an educational and interesting project, there were many new techniques and skills that was gained by it. The project was practical, as we had the chance to build our program, and theoretical, as we could search and read for several coding usage.

1.2 Introduction

The program that is descripted by this report extends application using Javafx libraries. The program implements the Object-Oriented Programming, different techniques are used to piece together the entire program and be understandable by everyone. The usage of abstract and inheritance classes is shown, and different usage for appropriate visibility of attributes and methods will be described. The program is a game that everyone can play, it has an OOP design that results on the well-structured design, and it takes user's input to give him the opportunity to do some task. On the report you will see a further description of the program's structure and design, a class diagram and the actual runtime-experience description, using use case examples and screenshots.

2. OOP Design & Class Diagram

2.1 OOP Design

At this stage, the program's design and structure is explained. We will briefly go through each class explaining their functionality, the visibility of attributes and methods and some external libraries that are used.

The first class that will be explained is the Drone class. Drone class is the superclass(parent) of the inheritance and is an abstract class, the NormalDrone, MonsterDrone and TargetDrone classes are the subclasses(child). The attributes of that class are in protected visibility as they are used by the subclasses. The constructor takes as parameters the x and y position for a drone and the direction that is moving, as these are attributes that every type of drone need to have. Getters are produsedfor the position of the drone, with a public visibility obviously. Then the public method, tryToMove() is called in other classes to check if there is space for a drone to move at a specific direction, otherwise the direction is changed. This method is a recursive method that uses further methods from other classes to either check for arena boundaries or change to the appropriate(logical) direction. Moreover, we have a method Boolean that checks if there is drone is at a specific position. The next method, boolean isTouching(), is shown twice inside the class, with a private visibility on the first case and a public visibility on the second. The private one is used to compare two drones that coming to the same position, extended by their radius, and returns true or false in case they touch. The public one is used in other classes taking as parameter the type of the drone that we compare and returns a call of the private one giving it the appropriate position of that drone. At the end, we have 3 declarations of methods that we will see on the subclasses of the inheritance, the checkDrone() and toString() methods are abstract as they must be used for any type of drone.

The three subclasses, NormalDrone, MonsterDrone and TargetDrone, extend the Drone class. They have their one constructor where they give their information using super to the drone constructor and their unique speed. The MonsterDrone and TargetDrone classes have an extra id attribute. The classes implement the abstract methods that are declared in Drone class, as shown in figure 1, and the NormalDrone class implements the tryToMoveNormal() method that is similar with the tryToMove() method with the difference that the direction is specified by user's input.

Figure 1

```
/**
 * Calling appropriate methods related with the drone
 *
 * @param dar the current arena we are using
 */
@Override
protected void checkDrone(DroneArena dar) {
    if(dar.checkPoint(this)) {
        dar.Point();
    }
}

/**
 * Takes all the information in a String
 */
@Override
public String toString() {
    return "Your Drone is at ( " + x/10 + " , " + y/10 + " ) moving at " + direct;
}
```

Moving on to Direction enum class, this class have a double functionality as it repossesses the direction to the drones in both normal and random cases. The class contains the constants for the specific directions and public methods for moving to the next/previous direction that are used in Drone.tryToMove() method.

The DroneArena class start with its constructor where it assigns the dimensions of the arena and creates an arraylist to save the drones. The getter methods for the arena dimensions follow, and next the rNum() function creates random values using java.util.Random libraries. The addDrone(), addMonsterDrone() and addTargetDrone() have a public visibility because they must be called in another class, and they create a drone of its type using restrictions on the number of that drones availability and adds it to the arraylist and the ListView. Then the checkDrones() is used repeatedly in the program and it calls the checkDrone() method for each type of drone to see if there are specific tasks that must be completed. Then three methods are performed counting the created drones of each type and inform the user about the restrictions. The game gives the opportunity to the user to play with different number of MonsterDrones, the method setPointsValue() performs a ratio for the points gained by each TargetDrone with the Number of MonsterDrones that are currently in the game. The boolean methods checkPoint() and checkDeath() checks if two drones of a specific type are touching each other and saves the specific drone to be removed by the next functions Points() Death(). Furthermore, we have a function that reset the score to 0 and a getter function which returns the score. The canMoveHere() method is the one used in tryToMove() to chack for space availability and boundaries of x and y axis of the current arena. The last two methods are the methods where they actually move(redraw) the drones to the new appropriate positions by calling tryToMove() to make the check and updateCanvas() to redraw the drones. Almost all the methods in that class are public as they must be used in other classes.

The next class is called MyCanvas and is responsible to gather the information and draw the actual arena. This class uses graphic context, by javafx.scene.canvas.GraphicsContext. It inputs the images and assign them to each type of drone. The constructor gets the canvas dimension and using the method fillCanvas() we draw the arena with specified colours dimensions and style, using the graphic context. Then, the updateCanvas() uses again the graphic context to redraw the arena and calls the next method, showDrones() that gives the current position of each drone with their size to be printed, as the assigned image, in the current arena. In this class the attributes are private as there is no need to be used outside the class, but all the methods are public, because they must be called in specific occasions, when the arena must be drawn/re-drawn.

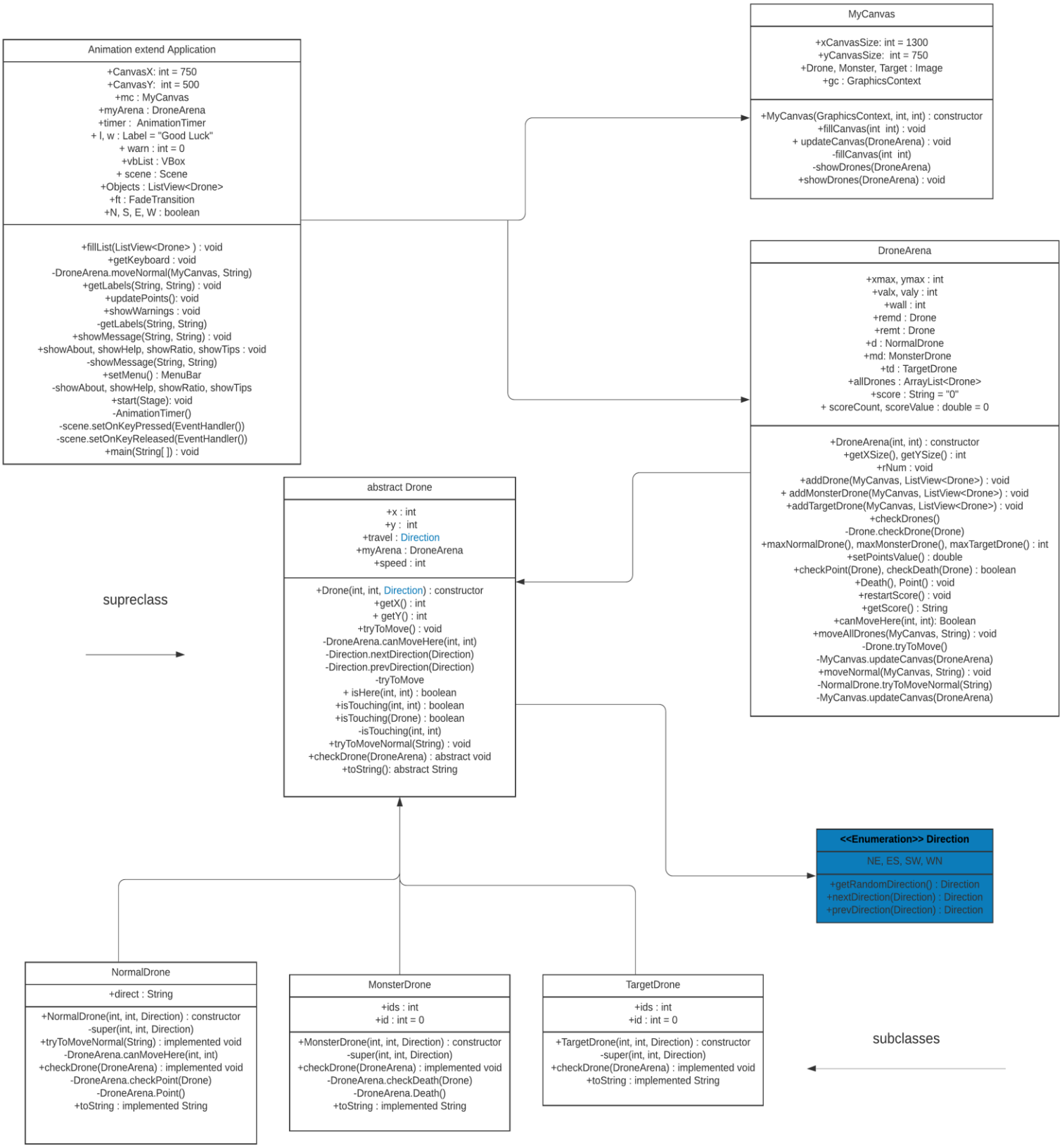
Last but not least, the Animation class extends Application using javafx libraries. Almost every attribute and method is private, as the class takes everything together and in to action. The first method fillList() adds all the drones' information into the ListView to be shown. The getKeyboard() moves the NormalDrone as the user press the according keys and its called later in the timer. We have a vBox to add labels and the getLabel() method that creates a label with specified text and colour, the text is also faded out as time passes. The updatePoints() get the current score of the player and continuously show it during the gameplay using labels and adding them to the vBox. The showWarnings() method is created in order to inform the user for restrictions and it calls the getLabels() to add the appropriate message, an example is shown in figure 2. A menu bar is created with information about the game, the showMessage() function creates the window to be appeared with the appropriate description to inform the user and is called by the showAbout(), showHelp(), showRatio() and showTips() methods. This method contains the description of each menu item. The setMenu() method is where the menu bar and its items are created and set up, it takes all the information regarding the menu and returns it. The next function start is the place where everything comes together and the stage starts. In that function the canvas, the scene, the vBox, the ListView and the Hbox are created and set up. Finally, we have our main() method calling Application.launch(args) to launch the game.

Figure 2

```
/**
 * create warnings to inform the user in different cases
 */
private void showWarnings() {
    switch (warn) {
    case 1:
        if (myArena.maxNormalDrone() >= 1) {
            getLabels("You can only insert one Normal Drone ", "RED"); // create the label and add it to vBox
        }
        warn = 0;
        break;
```

Taking everything into consideration, the OOP design starts with Animation class, where it takes everything together, and it interact with MyCanvas class to create the canvas and with DroneArena to gather all the information about the drones and the arena. The DroneArena class then interacts with Drone abstract method, therefore it interacts with the inheritance of NormalDrone, MonsterDrone and TargetDrone subclasses. The Drone class finally interacts with the Direction enumeration class to assign to its drone the appropriate directions.

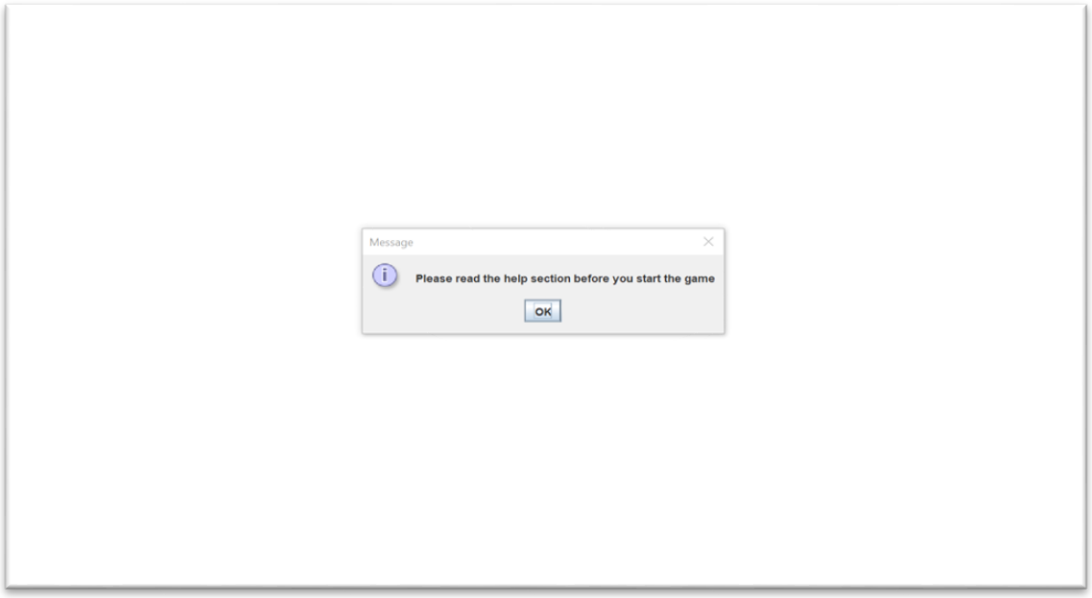
2.2 Class Diagram



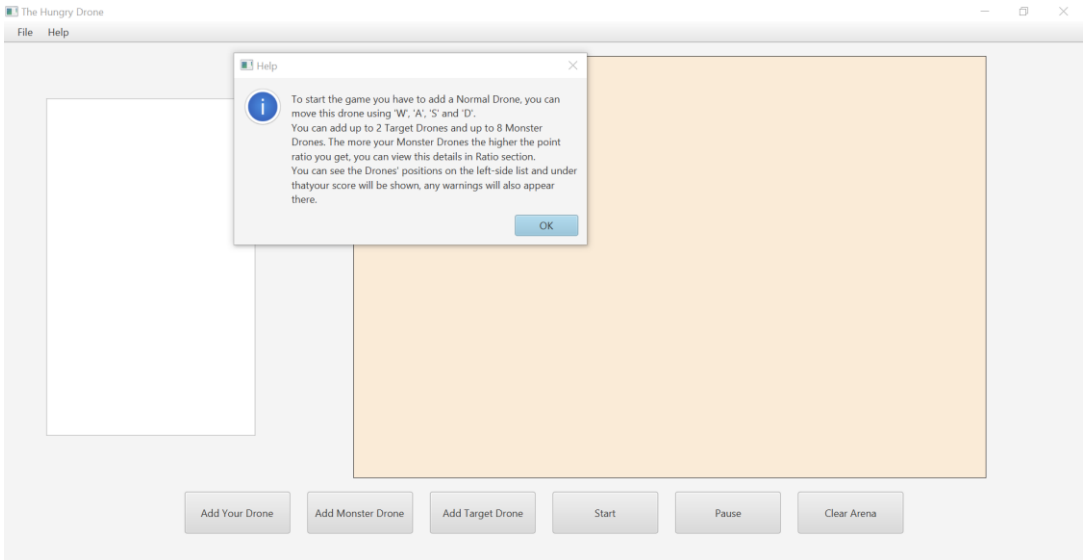
3. User Manuals & Screenshots

When the program is run, the user first gets a message that tells him to read the instructions before he starts the game. Then the window with menu bar, buttons, ListView and the arena is shown. At this point the user can go through the menu items or use the buttons (add Drones, start the game, pause the game or clear the arena). Once a drone is added the specific drone is immediately added to the arena and its information on the ListView. Some restrictions appear below the ListView if he procced to some specific situations. When a NormalDrone is added the user can move it using 'W', 'A', 'S' and 'D'. Other types of drone can start moving by clicking the start button. After the game starts the players score is permanently shown below the ListView and is updated every time he gets a TargetDrone. If the player loses then a window message appears showing him his score. Then the score is reset, and he can add a NormalDrone to start again or clear the arena and add all the drones again.

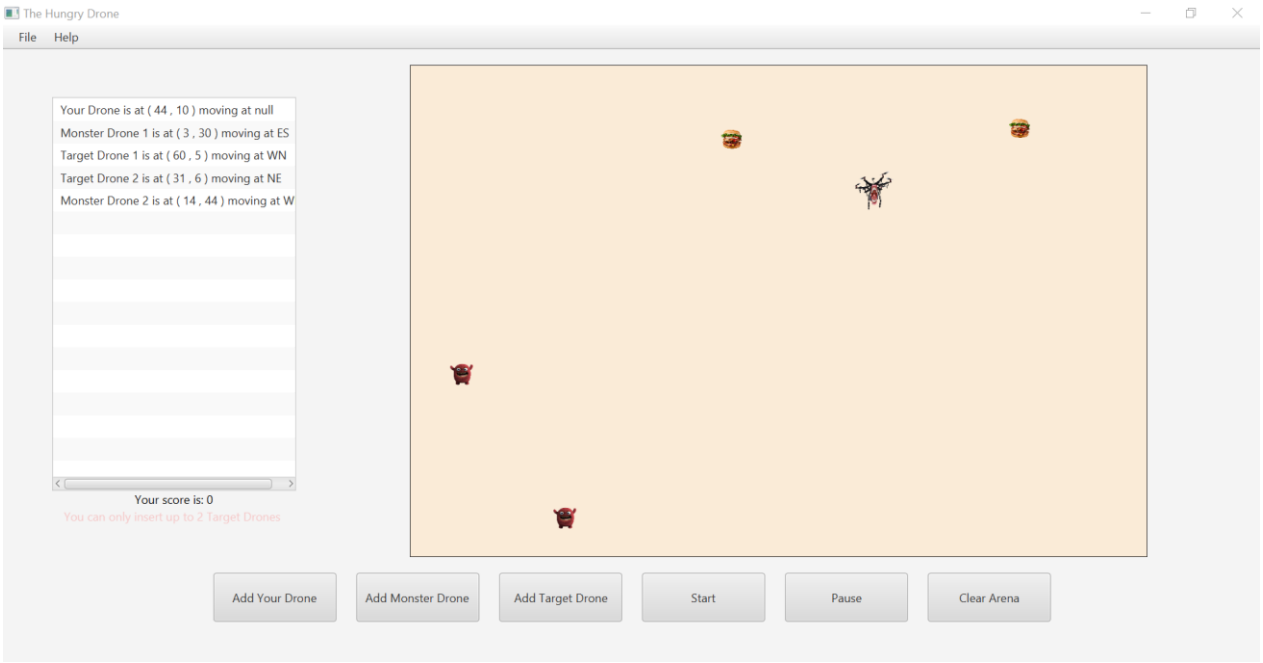
First window:



Menu item example:



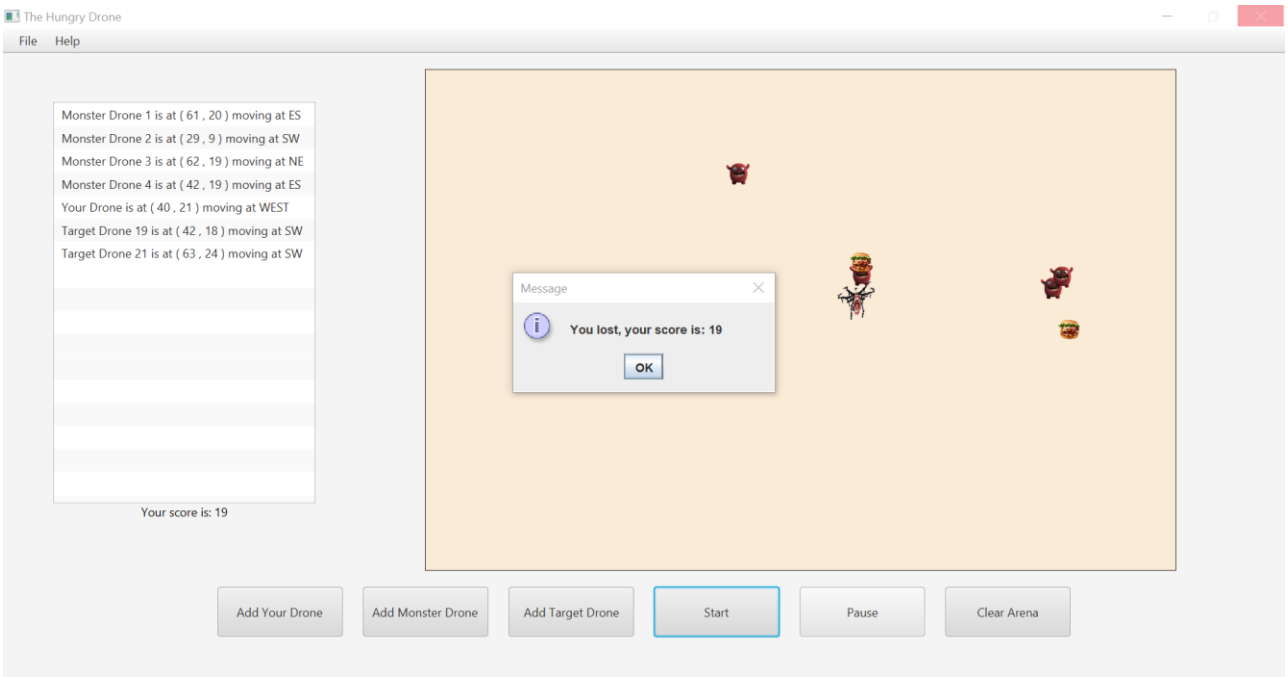
Gameplay:



4. Test Cases

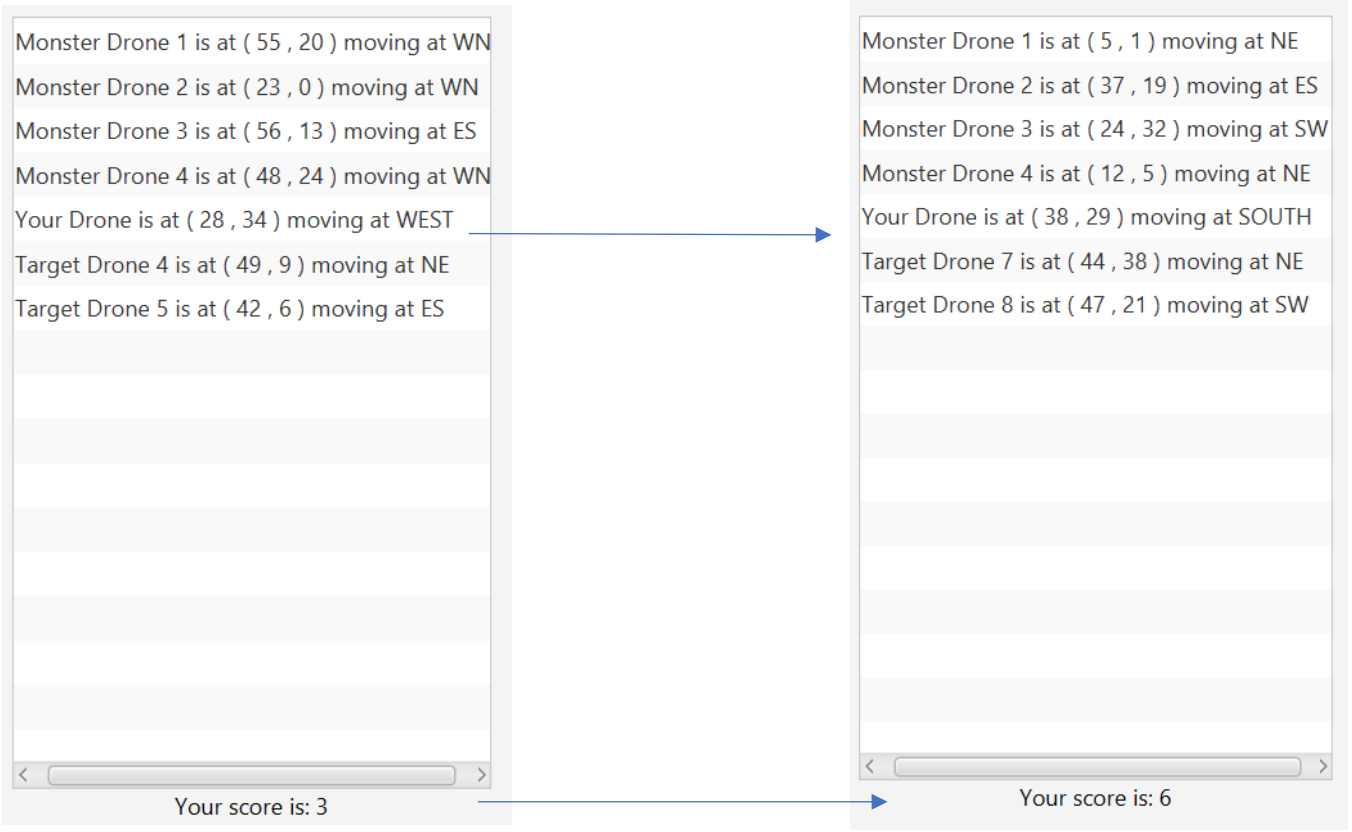
The program went through several testing in order to give to the user the best experience and below I will show you some of these test cases.

A) At first, I wanted to inform the user when he loses that the game is over and show him his score.



B) Set restrictions and inform the user about them as he tries to go through the restricted scenario, there I wanted to prevent the game for starting if a NormalDrone is added to the arena.

C) The information must be updated as the drones move.



Test	Expectation	Succeed
1	Inform the user when he loses that the game is over and show him his score.	**YES**
2	Set restrictions and inform the user about them with warning.	**YES**
3	Update the information while the game is running.	**YES**

5. Conclusion

In conclusion the project was something new to me, introducing the Object-Oriented Design, the use of inheritance, the well-structure programs, enumeration and many more. I really liked working on such project and discovering new techniques. There were some difficulties but that's what made me search more and so enhance my learning. The result is exciting, and I think I did a good work, although there are many things in addition I can learn and improve my coding and my thinking. One last thing that turned out to be very useful and I must mention is the Javadoc comments, where it helps the understanding of a program and it makes easier the fixing of problems later.

CSGitLab Link

https://csgitlab.reading.ac.uk/hz000050/drone-simulation-gui/-/tree/master