Java Individual Coursework

CS2JA16: Drone Simulator GUI

University of Reading

Created by:
Shavin Croos

Module Code: CS2JA16

Assignment report Title: Java Individual Coursework Drone Simulator GUI

Student Number: 27015244

Date (when the work completed): 11/12/20

Actual hrs spent for the assignment: 28 hours

Assignment evaluation (3 key points):

- Allow for more time to complete the GUI version, so that more meaningful features could have been made and added

Introduction

For this Java GUI project, I'm looking to create a drone simulator where in a given arena, multiple drones can move about. In order to create this GUI application, I will need to call upon the console version I had created, by inheriting the classes and methods that I had used to initially get the mechanics of the drone simulation working. Some of the classes will be modified in order to get the graphics needed for the GUI version. Along the way of creating the GUI, I will carry out some tests to see if key aspects of the GUI work properly in the way they should.

OOP design

In order to turn the drone simulation from the console version into a nicer looking GUI, I had to call upon the classes that I had used in the console version. These classes were Drone, DroneArena, ConsoleCanvas, Direction and DroneInterface. The classes Drone, DroneArena and Direction stay pretty much the same as in the console version, whereas the ConsoleCanvas and DroneInterface classes are adapted to produce a graphical outcome, the Java application.

The Drone class is responsible for creating an instance of a drone, which stores the information about its ID, position and direction, as well as determining where the drone will go next and the drone counter. The drone's positions, ID and direction were kept as private classes as they were not needed to be necessarily used by the other classes and only the Drone class should interact and alter, if necessary, these variables. On the other hand, the drone count is kept as a public static class since this value is needed to be seen by the user when they run the application when they insert a drone into the arena.

DroneArena class handles the creation of the drone arena within the specified canvas dimensions of the application in the DroneInterface class, so that drones can be added on to it and move around inside. This class stores details on the dimensions of the drone arena itself, the position details of the drone being added to the arena, a list for the drones to be added to and a drone counter to count how many active drones there are within the created arena. The arena dimensions and drone positions are kept as

private as they are not meant to be altered by the other classes and should be changed during the simulation. In contrast, the drone counter and drone list are made public as this is what the user will see when they interact with the application upon startup.

Direction class is in charge of handling the drone's current direction and which direction it should change to next, should it encounter the arena wall or another drone. Since this is an enumeration class within a class, there are no proper defined variables to consider when using this class. This class stores all the possible directions the drone can go based on a 16-point compass. This class is made public as the Drone class is highly dependent on this class to determine which direction the drone should try to move next, should it find the space occupied by either a wall or a drone.

Next, the MyCanvas class takes care of creating the actual canvas for the arena and drones to be added onto, so that it appears on the application window. This class stores the details of the canvas dimensions. The canvas dimensions do not have any specified access modifiers, since they are set values based on the values inputted to create the canvas from the DroneInterface class. These values are used to tell the user the size of the arena, since the size of the arena will be the same size of that of the canvas it's been given.

Finally, we have the DroneInterface class, which is where everything is culminated together to create a nice graphical output in which the user can interact with. This class calls upon the classes MyCanvas and DroneArena in order to build the drone arena onto the application. At the same time, an abstract class called AnimationTimer is used to animate the drone's movements within the arena, as well as stop that movement once it's not needed anymore.

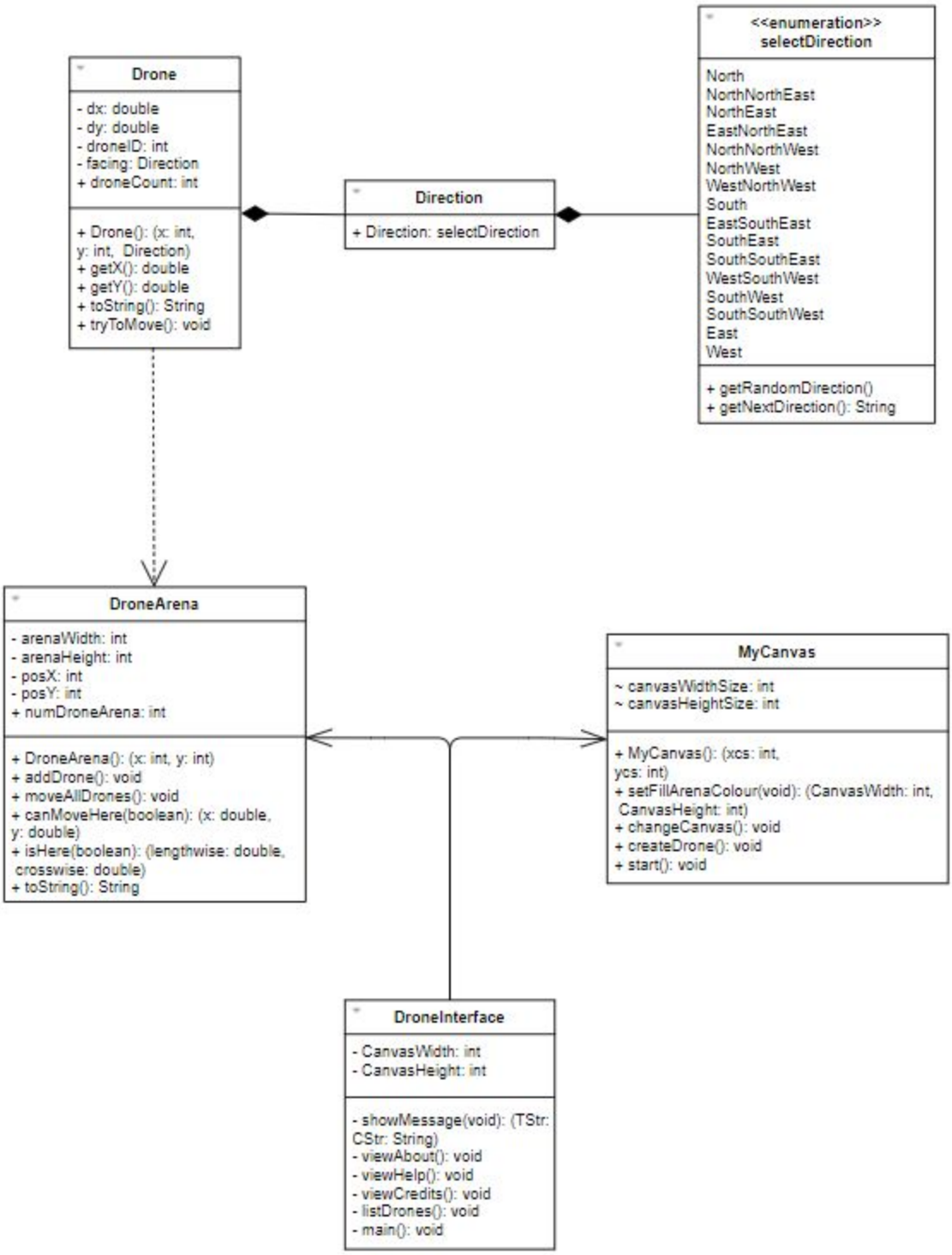
Drone
- dx: double
- dy: double
- droneID: int
- facing: Direction
+ droneCount: int
+ Drone(): (x: int, y: int, Direction)
+ getX(): double
+ getY(): double
+ toString(): String
+ tryToMove(): void
Direction
+ Direction: selectDirection
<<enumeration>>
selectDirection
North
NorthNorthEast
NorthEast
EastNorthEast
NorthNorthWest
NorthWest
WestNorthWest
South
EastSouthEast
SouthEast
SouthSouthEast
WestSouthWest
SouthWest
SouthSouthWest
East
West
+ getRandomDirection()
+ getNextDirection(): String
DroneArena
- arenaWidth: int
- arenaHeight: int
- posX: int
- posY: int
+ numDroneArena: int
+ DroneArena(): (x: int, y: int)
+ addDrone(): void
+ moveAllDrones(): void
+ canMoveHere(boolean): (x: double, y: double)
+ isHere(boolean): (lengthwise: double, crosswise: double)
+ toString(): String
MyCanvas
~ canvasWidthSize: int
~ canvasHeightSize: int
+ MyCanvas(): (xcs: int, ycs: int)
+ setFillArenaColour(void): (CanvasWidth: int, CanvasHeight: int)
+ changeCanvas(): void
+ createDrone(): void
+ start(): void
DroneInterface
- CanvasWidth: int
- CanvasHeight: int
- showMessage(void): (TStr: CStr: String)
- viewAbout(): void
- viewHelp(): void
- viewCredits(): void
- listDrones(): void
- main(): void

Figure A

User Manual

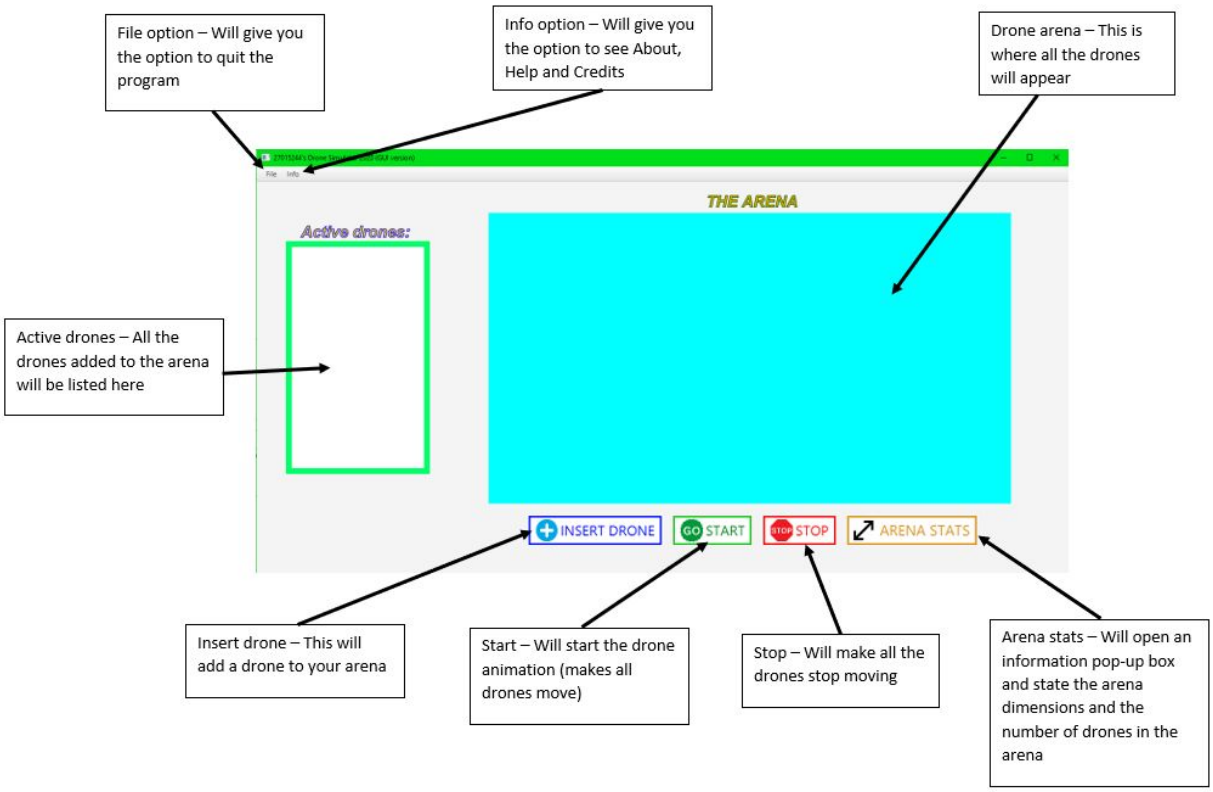

Figure B

To run this program, all you need to do is add a few drones to the arena by clicking the 'INSERT DRONE' button several times. Once you've done that, hit the 'START' button and watch the drones get animated in the arena. They'll bounce off the walls and off each other in a cool manner. When you don't want to see the drones move anymore, simply click the 'STOP' button to stop the drones from moving. 'ARENA STATS' will allow you to see the arena dimensions and the number of drones you have added to the arena, which will save you from scrolling down on the Active drones list.

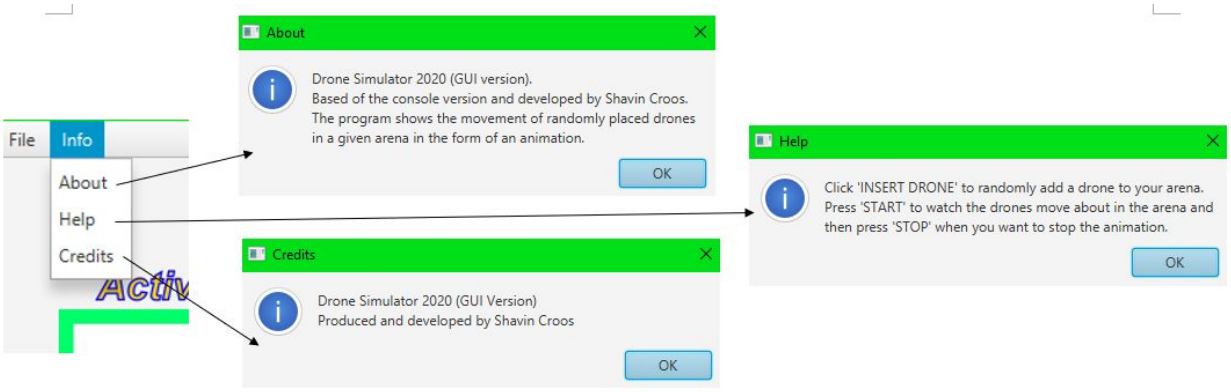

Figure C

Tests executed

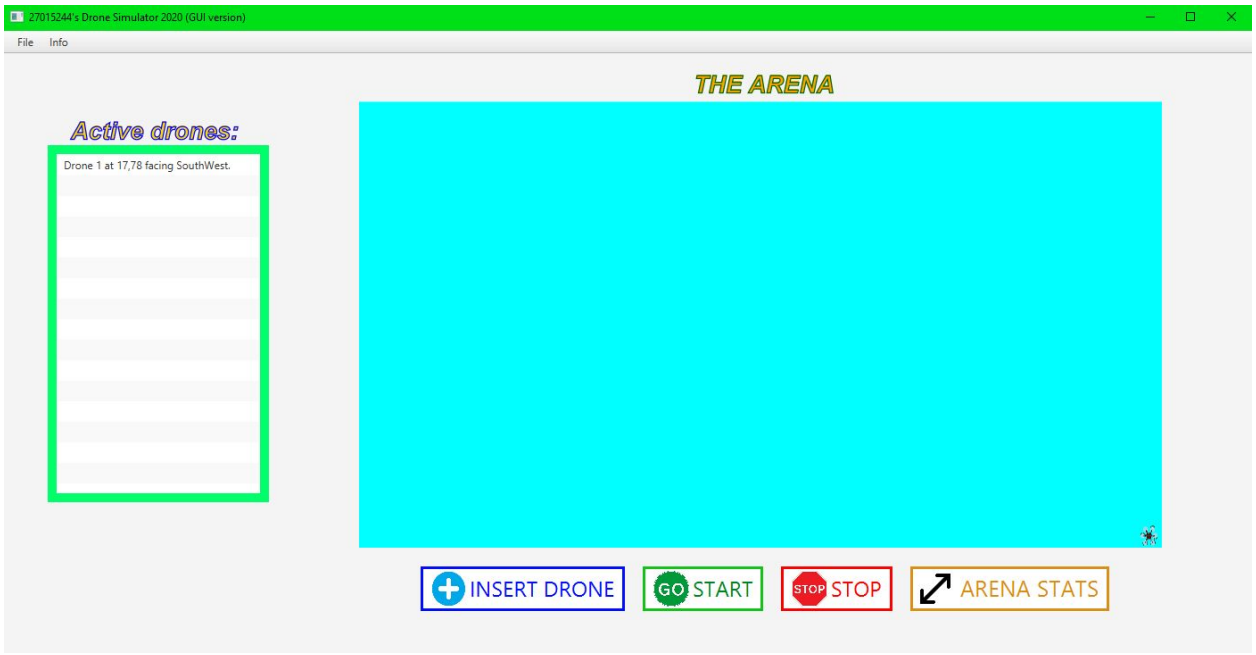

Figure D

I first started my testing by seeing if I could add one drone to the arena and get it to animate (including hitting the walls), without crashing the program. The drone was successfully added and after pressing ‘START’, the drone successfully moved around in the arena, bouncing off the walls without crashing the program.

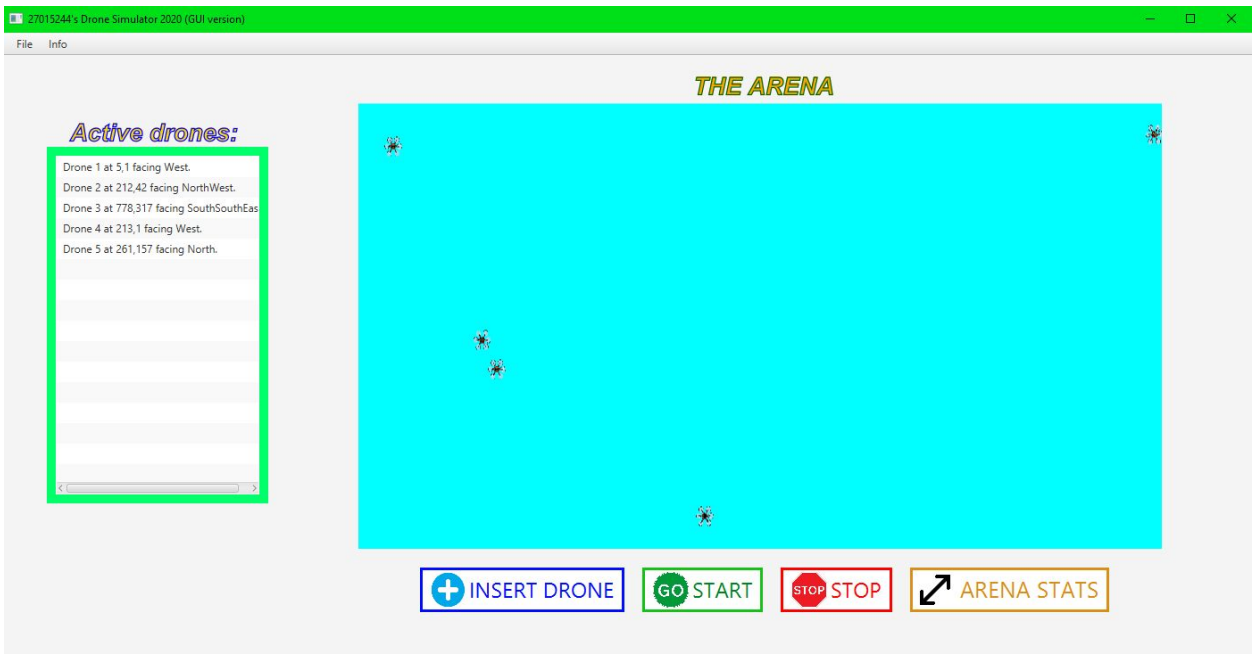

Figure E

Next, I tested to see if the same result would occur (but this time with hitting other drones) with adding more than one drone to the arena. Again, when pressing ‘START’, the program did not crash and the drones successfully moved around the arena, bouncing off the walls and other drones.

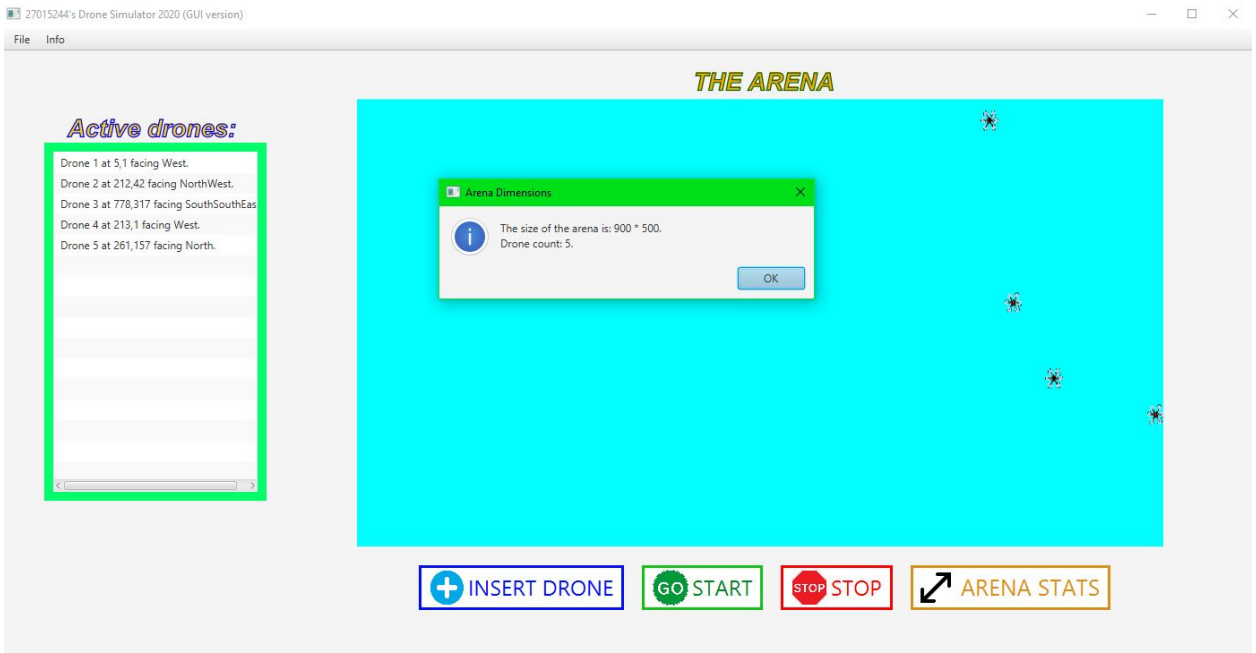

Figure F

Finally, I carried out a test to see if the ‘ARENA STATS’ button worked correctly in the way it should. After pressing the button, the pop-up box correctly displayed the arena dimensions and the number of drones that were inside the arena.

Conclusion

Overall, this project has allowed me to understand the mechanics of how the drone moves within the arena and how the simulation could be displayed nicely as a GUI. To some extent, it has allowed me to incorporate some software engineering techniques I used in other modules such as System Design. I think I could have improved on the GUI version by adding some obstacles and enemy drones to make the simulation look more interesting to the user. Furthermore this project could have been improved by allowing for more extra time over the holidays to be able to think and develop some nice features to the GUI version of the drone simulator.

Reference to my repository

https://csgitlab.reading.ac.uk/at015244/cs2ja16_drone_simulation_gui.git