Major Coursework #1 (Programming report)

Introduction

For this programming coursework, we were required to implement a simulator of different types of robot - like WhiskerBots and Light Detectors - that would be able to move around an arena in which there would also be lights and other obstacles. This had to be displayed in a graphical user interface(GUI) using JavaFX.

The application has been provided with a suitable menu, which allows the arena to be saved, loaded and configured and also provides information about the simulator. The user is able to save every arena that has been created with all its contents to a file as well as load any arena that has been previously saved.

A toolbar is added to the bottom of the window with corresponding buttons to control the simulation. ‘Start’ and ‘Pause’ control the movement in the arena, ‘Robot’, ‘Whisker Bot’ and ‘Light Detector’ add robots and similarly, ‘Light’ and ‘Wall’ add obstacles. Lastly, ‘Clear All’ deletes everything that exists in the arena on that given moment, leaving the arena empty.

Class hierarchy

Robot is the abstract class that all the other classes inherit from. GameRobot, LightRobot and Wall inherit from it directly while WhiskerRobot and LightDetectorRobot are inheriting from Robot indirectly through GameRobot.

Encapsulation is used in my program as part of the object oriented design in combination with inheritance. This allows me to hide private data variables which can be accessed through specific functions.

List of classes :

Robot : is the abstract class that all the other robots in the arena inherit functions and variables from

RobotArena : includes all the functions that are necessary for the building of the arena

RobotInterface : sets up the GUI and includes all the functions that set up this interface

GameRobot : consists of all the methods that are needed in order to make the robots move around the arena and turn to avoid obstacles. Inherits from Robot

WhiskerRobot : has the functions that create a Whisker Bot. Inherits from GameRobot

LightDetectorRobot : has the functions and variables that create a Light Detector. Inherits form GameRobot

LightRobot : includes methods that set up lights in the arena. Inherits from Robot

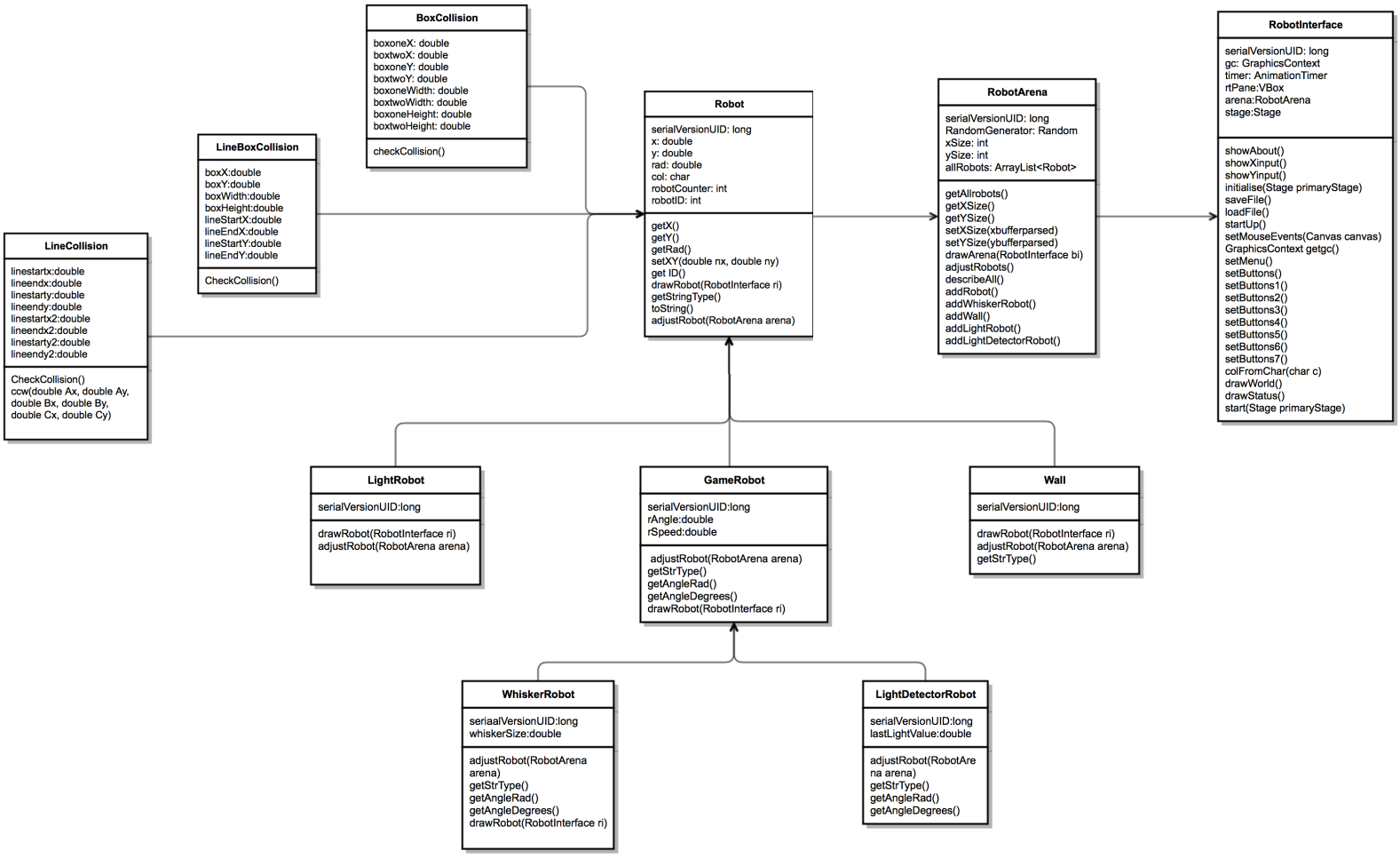
Wall : has all the methods that are needed to create a random wall in the arena. Inherits from Robot

BoxCollision : checks if two boxes are colliding

LineCollision : checks if a line intersects a line

LineBoxCollision : checks if a line intersects a box

UML Diagram :



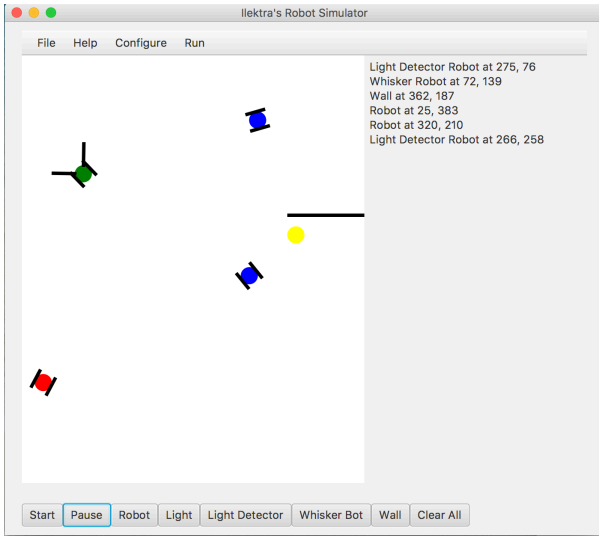
*The constructors have not been included in the method boxes

Design

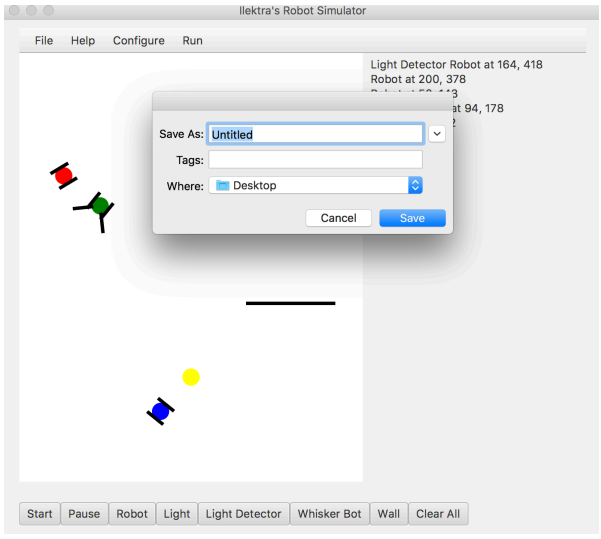
Overall, the design used was strongly object oriented for many different classes inherited variables and functions from the abstract class that was created. The design can be improved further to ensure the program has a stronger hierarchical structure by creating a new abstract collision super class and having the three collision methods inherit from it. This would be feasible as all of the collision classes share eight private variables and have a boolean check collision.

Final application

The final application presents the arena with the menu bar on top and the buttons on the bottom. By clicking Start on the bottom toolbar, the robots that have been added begin to move around the arena while by pressing Pause, they stop their movement.

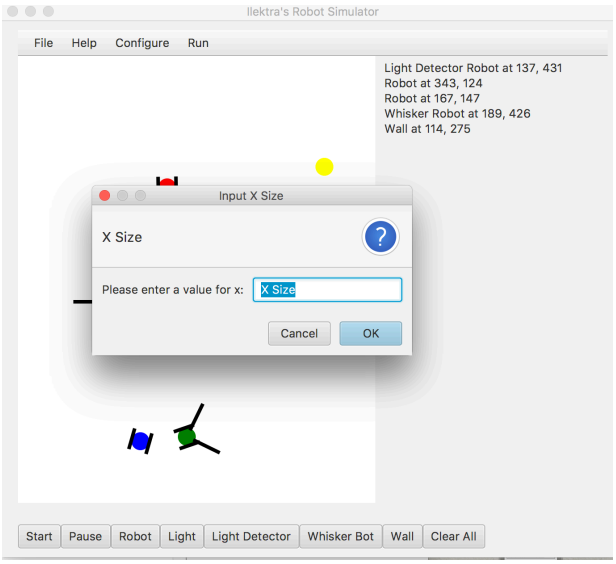


If the user would like to add more robots or obstacles to the arena, they can do so by clicking one of the five buttons on the bottom. For example, by pressing Whisker Bot more robots with whiskers are added to the arena. The Clear All button on the bottom right corner of the screen can be pressed to clear all the existing robots from the interface and create a new empty arena. While the animation is in progress, a pane on the right hand side keeps track of the coordinates of each robot on the screen.



The user is also able to interact with the interface by using the menu at the top of the screen. Selecting the File menu provides the options to save the current arena, load a previously saved one or exit the application.

The Help menu gives the option About which provides the user with information about the application.



By selecting Configure, the user is able to set up the size of the arena by entering a new value for x and/or y. A different window will open depending on which value the user chooses to change.

The Run menu implements the same functionality as the Start button, by clicking on it the robots in the arena begin to move around.

Testing

Testing was extremely important for the development of this program and it included thorough examination of the code as well as execution of that code in different conditions. A considerable amount of methods became functional through testing and using the debugger to find errors in the code. Below there are listed the final round of unit tests that were carried out on the program :

Buttons	**Expectation**	**Result**
btnStart	Starts robot movement	Test passed
btnStop	Pauses robot movement	Test passed
btnAdd	Adds robots to the arena	Test passed
btnLight	Add lights to the arena	Test passed
btnLightDet	Adds light detecting robots	Test passed
btnWhiskerBot	Adds whisker robots	Test passed
btnWall	Adds a wall to the arena	Test passed
btnCls	Clears the arena	Test passed
Menu		
Exit	Exits the application	Test passed
Save	Saves current arena in file	Test passed
Load	Loads arena from file	Test passed
Configure X Size	Allows user to set new x value	Test passed
Configure Y Size	Allows user to set new y value	Test passed
About	Provides info about the app	Test passed
Methods		
checkCollision()	Checks if two robots collide	Test passed
CheckCollison()	Checks if two lines collide	Test passed
drawRobot(RobotInterface ri)	Draws a robot into the interface	Test passed
adjustRobot(RobotArena arena)	Moves a robot inside the arena	Test passed
setMenu()	Sets up the menu for the GUI	Test passed
setButtons()	Sets up box with relevant buttons	Test passed

Conclusions

All in all, the main purpose of this program was to be built using object orientation, something that was accomplished through the way it was designed. In more detail, an abstract class was created in order for the project's numerous different classes to be able to inherit variables and methods from. The brief's specifications were met as all of the required features were successfully implemented in the code.

Improvements can be made, such as adjusting the wall obstacles in the arena so that the robots will no longer be able to cross over them on the sides. In addition, as mentioned before, a new abstract collision super class could be created that will have the three collision methods inherit from it. That would ensure that the program has a stronger hierarchical structure.

To conclude, having used some of the fundamental OOP concepts has contributed greatly to the development of this program. Through encapsulation, the variables of a class were hidden from other classes, and could be accessed only through the methods of the encapsulated class. That was achieved by declaring the variable of the class as private and providing public setter and getter methods to modify and view the variables values. Furthermore, through abstraction it was feasible to hide the implementation details from the user and only provide them with the functionality. By creating an abstract class, the classes that inherit from it have to provide implementations for all its abstract methods. Through the use of these concepts, this project has helped me to come to a deeper understanding of Object Oriented Programming.