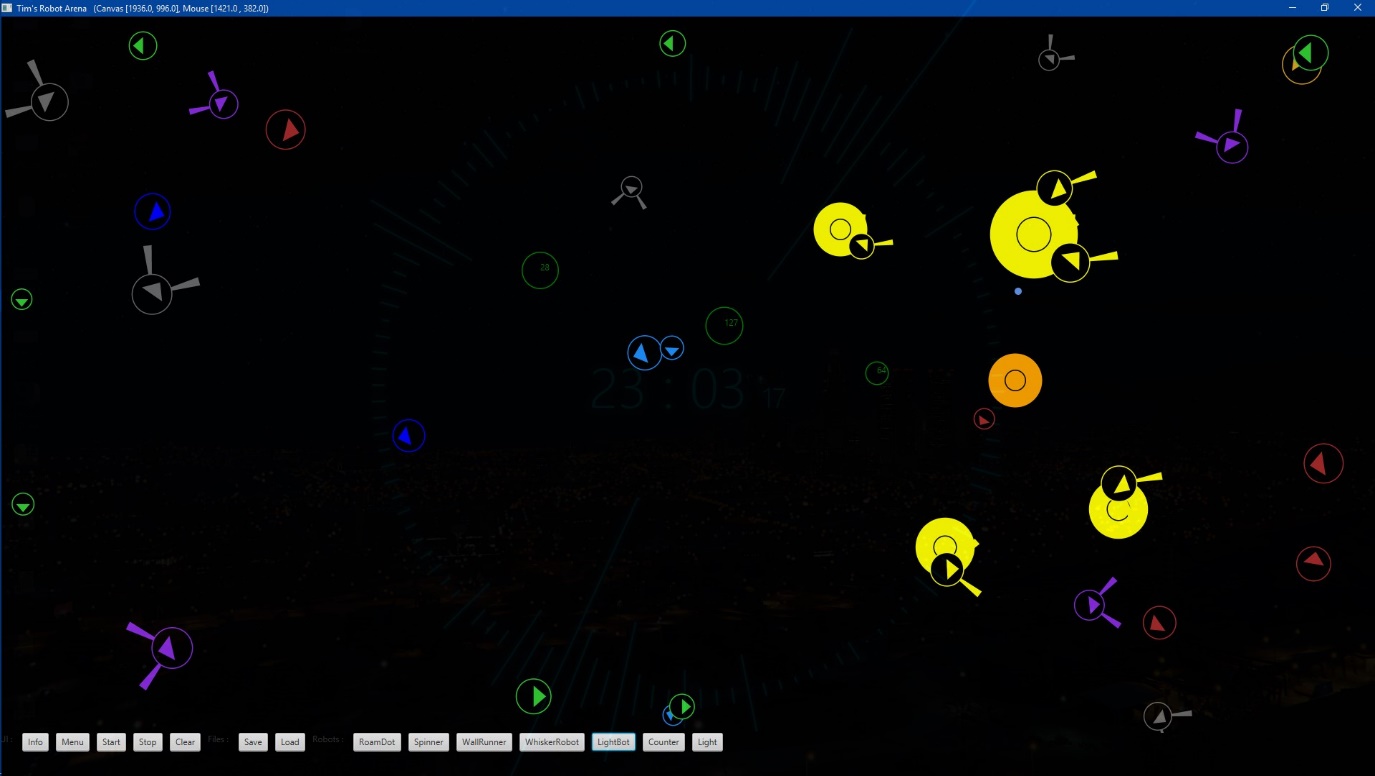
Timothy Nicholls

java  University of Reading

Java fx robot arena

# Introduction

My Robot Arena – Click here for [Javadoc](Java%20FX%20Robot%20Arena/doc/allclasses-frame.html) and [Specification](Java%20FX%20Robot%20Arena/doc/specification.pdf)



The aim of this coursework was to provide a functioning robot arena, with multiple types of robots, both static and active. These robots were designed to have interactions between each other to simulate how robots in an arena would behave.

For my own project, I decided that I wanted to make my Robot Arena have a Tron feel, where everything is dark except the parts which light up, and the parts that light up are colourful and clear. I achieved this by drawing each robot as an outline with a black inside, on a black background, so the base for each robot was just a circle outline. Then a full arrow was filled in the centre. The maths for a hollow arrow Is very possible in the style I decided, would have been not exactly hard, but would involve a lot of trigonometry, and I like to keep code brief with as little processing as possible. Adding potentially twice as many Sin and Cos functions to the code (which are not cheap functions to run) would slow down code.

I was very content with how the project turned out. There are lots of little things like conservation of momentum and changing of colour when robots are active which I enjoyed programming and playing with. I also learnt how to do a function with infinite overloads on parameters which I thought was very cool.

# Classes

## Window

The window class is a JavaFX application, which is the main window that opens. All UI elements (except for some other windows which are opened by the application) are contained within the Window created by this class.

### Main and Start

When the jar is run, the main code starts, which launches the application. Variables, which have been initialised in the class are set, the window and arena are defined, then everything is shown. Animation is also defined here, with calculation, drawing and showing of debug. When the window is resized, the animation timer also deals with resizing of the window.

### UI control (Info, Menu, Start, Stop, Clear, Save, Load, Robot Buttons)

The user interface control is found at the bottom of the window. The info button displays information about the program. Menu shows a button which can be used to toggle debug on and off. Start starts the animation timer, which is started by default and stop pauses it. Clear will purge the arena of any robots. The save and load functions save and load arenas to and from files.

Robot buttons when pressed set the variable type to a value. On the first time pressing them, no robot is placed but rather the type of robot which will be placed is changed. If the user clicks the mouse on the screen, the type of robot of the last button pressed will be placed. Upon pressing the button a second time, a generic robot of that type will be placed in the centre of the arena.

### Mouse Functions (Draw Pointer, MouseEvents, MouseX, MouseY)

When the arena is animated (not paused) a dot is drawn to keep track of where the mouse is. This dot is displayed using the getMouseX() and getMouseY() functions, which should retrieve the mouse location. If the dot is not in the same place as the mouse, then the Mouse X and Y are not being properly calculated. Also, if robots are being placed not on the pointer, then the dot can be used as a debug. The Mouse X and Y are calculated factoring in the Window title height (approx. 20 pixels) and the offset position of the window which is why getMouseX() and getMouseY() are required.

For the clicking of the mouse, I decided to use the mouse click as a signal to either place or remove a robot. The variable type will determine what type of robot is placed. If the mouse pointer is currently over a robot, then that robot is removed. This system allows for a clean adding and removal of robots. If the robots are travelling too fast, pause the arena with stop and then try removing the robot.

### Save and Load

When the save button is pressed, the robots in the current arena are converted into a string. The string is created by taking each parameter, converted to a string, then placing a splitter character in between each parameter, and a new line between each robot. The string is decoded by loading the file into a string, separating it into a string array of robot data by new line, then converting each datum into the correct format, creating a robot with the data. When a file is loaded, the same arena that was saved will appear.

## Robot Arena

The arena has a width and height dependent on the canvas size. It stores a Robot array of all robots created.

### Calculation of Robots

Robots are set as inactive before each frame is calculated, which means there is no code for calculating if a robot should be switched on or off, only code for checking if a robot should be on. Looping through each robot, the robots new position is calculated. There is a check to see if the robot has hit a wall, if so the direction is changed to mirror the wall it just hit so it appears to deflect away from the wall. Another loop goes through each remaining robot in the arena, which when the whole calculate function is run means that every robot has checked for collision with every other robot.

The next section of code checks if the robots are either a light robot and light, or both light robots. If a light robot and light interact, then they are both set active, and if two light robots bump, they simply pass through each other to avoid any strange bumping behaviours around the lights.

If neither robots are lights or light sensitive, it is checked whether they have collided by using Pythagoras to check the distance between them, and seeing if the distance is smaller than both radii added together. If they have collided, the line they have collided on (or the normal to the line between the two centres is calculated, and each robot will deflect from the reflection axis. Both robots are then calculated, which is an attempt to prevent robots from colliding on themselves multiple times.

Robots follow conservation of momentum, so if they have bumped, these laws will apply evening out their momentum. This was done as it felt unnatural that smaller robots should be able to affect the bigger robots in the same way as another big robot would, and smaller robots would bounce no further having bumped into a large robot.

After all is calculated, the robot’s position is checked to ensure it is not stupidly far away from the arena. Robots at (-7000, 20000) on a 1920x1080 arena are clearly lost.

### Draw System and Draw Info

Draw System clears the screen by making it black and clearing the canvas for robots to be drawn. The lights are sent to the beginning of the robots array meaning they will be drawn first. This is because robots can get lost hiding behind lights, and the brightness of a light is not physical so robots would be able to go underneath if they were not drawn above.

Each robot is drawn in a for loop, running the draw function for each robot. If Draw Info is also enabled, the debug area of the Window in the top right will show the each robot including its type and any data about it.

### Add Robot

Add Robot has two overloads. The first will take the integer type which defines the type of robot which should be placed, then the data as a list of parameters separated by a comma. The second overload takes the integer type, then a double array which does the same function, but takes data as a double array, not as a list of parameters.

Once the data has been collected from either function, a new array is created which is 5 long for taking x, y, size, speed, and direction. Depending on how many parameters have been specified, the function will generate any unspecified data which will then be used to create a new robot. If there are too many robots, or the robot type is not found then no robot will be created.

The ID of a robot, or its type is defined as follows. Abstract classes such as Robot and Whisker Robot have the ID 0. Any moving robots have positive IDs, and any static robots have negative IDs.

## Robot

Robots have data for their ID, X position, Y position, size, speed, direction, mass, momentum, inactive colour, active colour, background colour, and a state for whether they are active.

Mass and momentum are related to speed, with Momentum equal to Mass multiplied by Velocity (speed in this case as direction is calculated elsewhere so it is a scalar not a vector). The colours are set default here as white, white and black, however as robot is an abstract class this should never be seen.

### Exporting of Robot Data (Robot Info, Export Robot, Separate)

The aim of these functions is to separate the data of a robot into a string, for the default function in robot, the ID, x, y, size, speed, and direction are exported using the character c (which is § in my code) to separate the data. For other types of robot, with extra data, there is an overload which will add the extra data to the end using the same separate function.

### Colours and Drawing (Set Colour, Set BG, Draw, Draw Circle Border, Draw Circle)

A robot’s colour is set based on whether the robot is active. If the robot has active set to true (which is changed when the robot is calculated), then the colour will be set to the active colour, if not the robot colour is set to inactive. The default draw function first sets the drawing colour to the main colour (active or inactive), then draws a circle border, which is the circle using the radius of size. Next the colour is set to the back colour, which is black always, like the canvas colour. The inside circle is drawn, which is a circle with a radius of a fixed number of pixels, so the border does not scale with the robot size. This looks cooler.

### Colours and Drawing Continued (Draw Direction Arrow)

For the case of any moving robots, a direction arrow is shown to show the direction of which the robot is moving. This is calculated using trigonometry of 3 smaller triangles from the centre to a location a specific distance away from the centre in a direction based on the direction of the robot to calculate each point, to draw the triangle. This triangle is drawn in the main colour of the robot.

### Movement

For non-static robots, the direction is fixed before calculation. A new X and Y coordinate are calculated using sin and cos on the direction multiplied by the speed. After, the momentum is calculated using the mass and the speed depending on if the speed has changed.

### Keeping the Robot in the Arena

The function onWallBounce() checks if the robot has gone outside the bounds of the arena. Firstly, it checks if the X or Y coordinates of the centre of the robot, outside the bounds of the arena, adjusted by the radius of the robot. If the robot is outside the arena, a check on the direction the robot is currently facing is performed, checking if the robot is pointing in the right direction to be sent back to the arena. This is to prevent robots outside the arena from trying to turn away from the arena once they have turned back.

## Other Robots

### Roam Dot

The activity of this robot is calculated in calculate as an overload, the activity depends on the momentum.

### Spinner

Spinners have an extra overload for calculate which adds a spin to the direction each calculation. The direction of the spin depends on whether the value is positive or negative. An overload for export robot has been adjusted to support spin.

### WallRunner

Wall Runners have an overload for when they hit a wall, they will turn 90 degrees to the wall instead of bouncing. Also they overload for no collision code, as things get messy when they collide.

### Counter

Upon colliding, counters increment count by 1, rather than moving, when robots collide with counters, the mass of the counter is simulated to be that of the robot’s own mass, so that the robot loses no momentum.

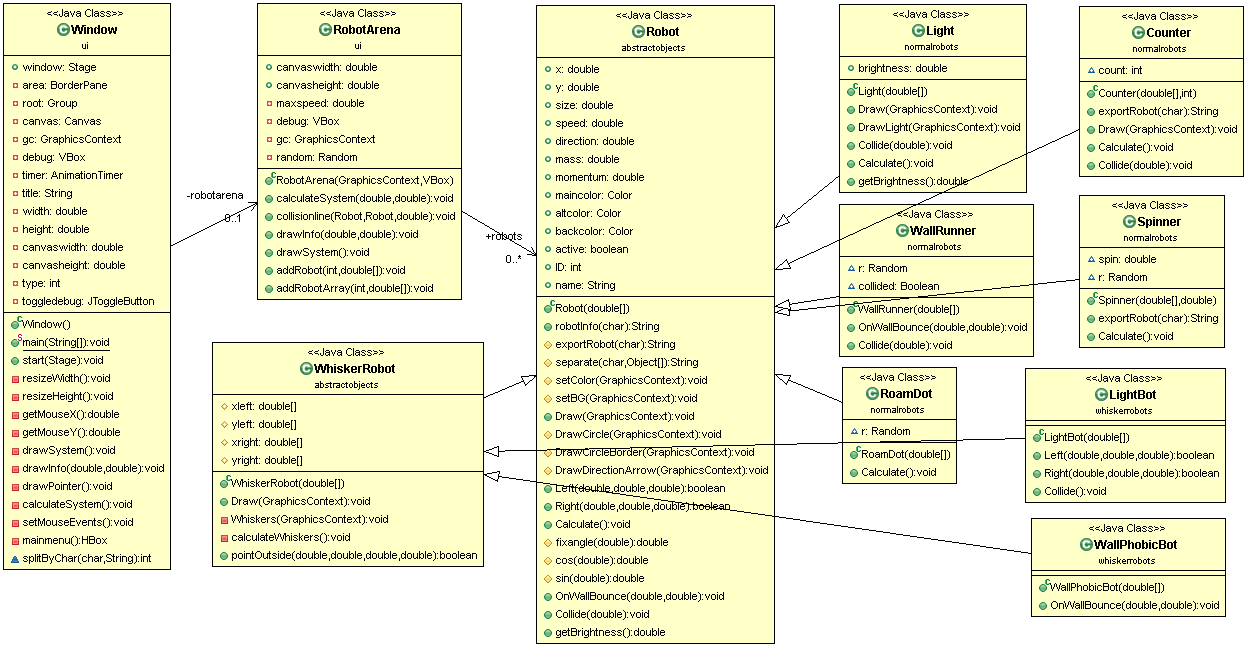
### Whisker Robot

Whisker robots are a special type or robot which have radial whiskers calculated. Only the outer far point of each whisker is used to detect collision however.

### Light Bot

Light Bots have a left and right function that determines whether a point will be inside any circle given the x, y, and radius of the circle.

# UML Diagram



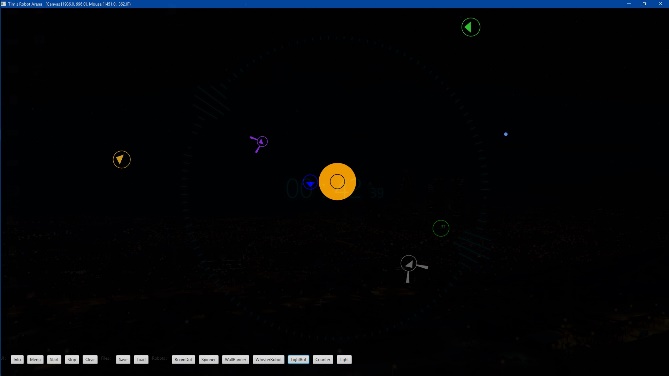
# Testing Section

## Test Plan

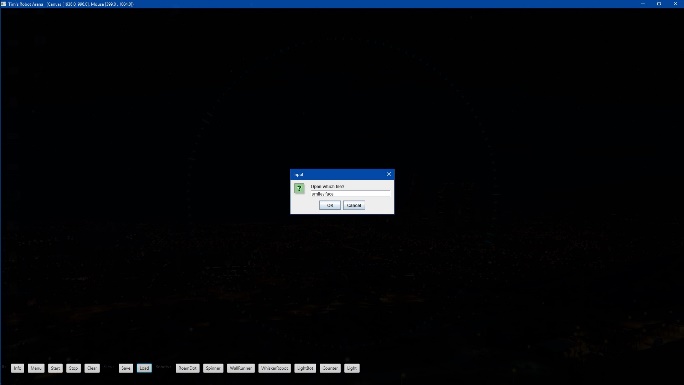
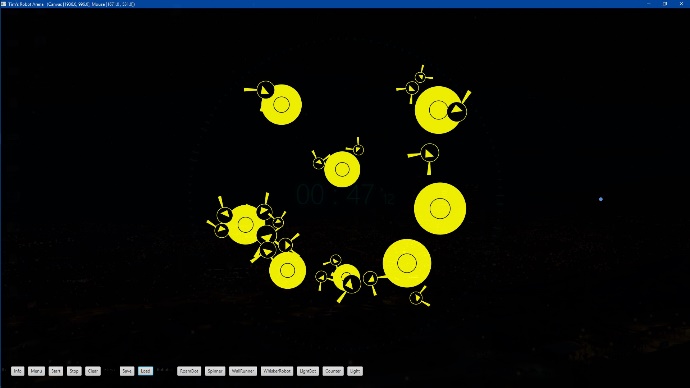
1. Every type of robot can be placed without errors
2. Saving an arena to file, then loading that file displays the same arena.
3. If a robot somehow escapes the arena, it will be returned to the centre of the arena.
4. Arena can resize without crashing and the robot arena continues to work as intended

## Test Result

### 1



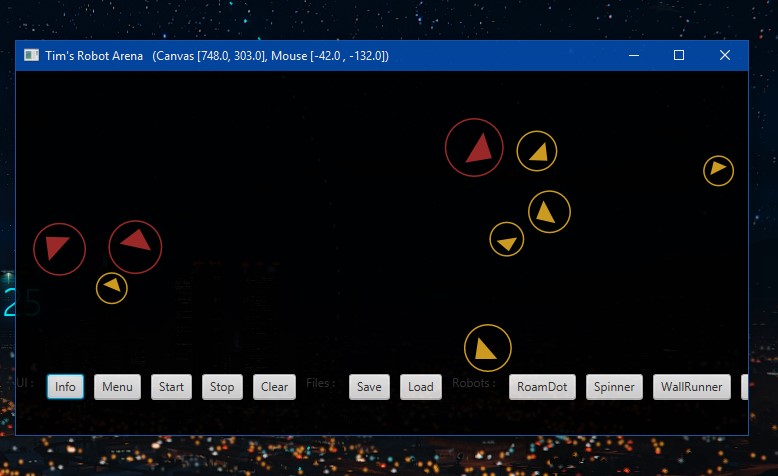
### 2

### 3



### 4



## Testing and Conclusion

As the document needs to be quite short, there is not much space to put screenshots of testing however a lot more testing was done. The debug menu works, the mouse pointer location works well, and robots will be placed and removed in the right location, of where the mouse is. When robots are clicked on they will place and remove as expected, however a small bug is when a robot is placed, which overlaps another robot, they can end up bouncing off each other. There is a limit to the number of robots which may be in the arena at any one time to stop the UI from being pushed off the bottom of the screen when the debug is shown and to keep the robot arena clear. The file handling will check if the name of the file is null or does not exist, and will handle both the saving and loading of files appropriately. If the file is corrupted the program will not accept the file however. When the windows appear for info and debug, they can appear behind the application, which can be annoying. There is also a bug with collision when 2 robots collide too close and get stuck on each other than keep bouncing off themselves.

As far as I can tell, the program works fully as I have intended so I am pleased with the overall result.

# User Guide

Welcome to Tim’s Robot Arena! Get started by running the .jar file

[Press here to launch the program!](TimsRobotArena.jar)

Upon launching the program you will be presented with a dark screen with a user interface and a dot to represent your mouse location.

## User Interface

Info – Displays information about the program

Menu – Toggle debug

Start – Start the animation

Stop – Stop the animation

Clear – Clear the arena of robots

Save – Save the current robot arena to file

Load – Load a robot arena from file

Robot Buttons – Each of these will set the robot type to the chosen robot

## Using the Arena

By default the arena is un-paused, simply clicking anywhere in the arena will place a robot, the default placed robot is a roam dot. The type of robot placed can be changed by pressing a different button with the name of the desired robot. Pressing it once will set the robot type to that robot, pressing it further will place robots in the centre of the screen.

To remove robots from the arena, they must be clicked on. To make this easier just press the stop button, the click on the robots you wish to remove.

