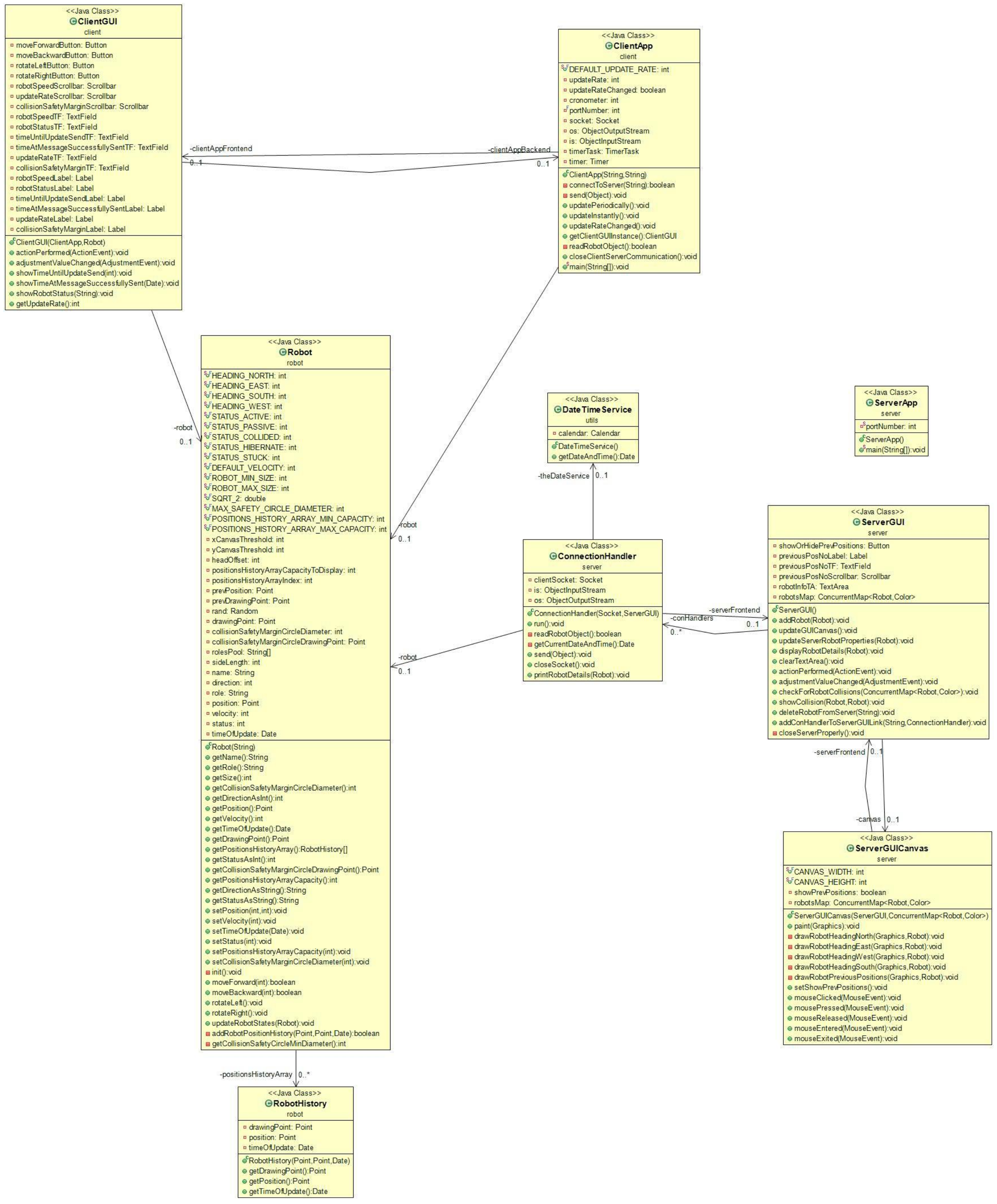


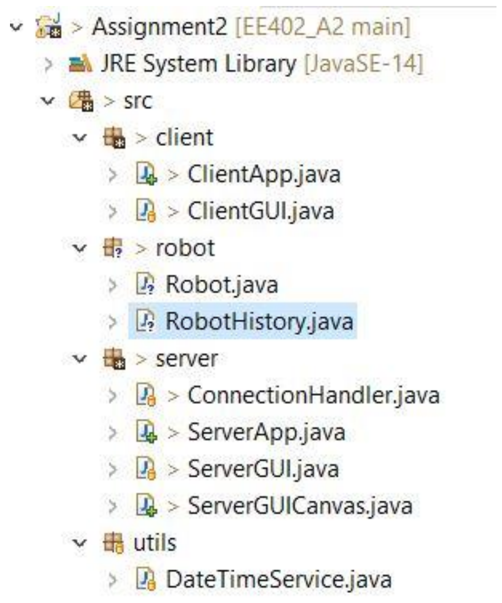
<<Java Class>>
ClientGUI
client
moveForwardButton: Button
moveBackwardButton: Button
rotateLeftButton: Button
rotateRightButton: Button
robotSpeedScrollbar: Scrollbar
updateRateScrollbar: Scrollbar
collisionSafetyMarginScrollbar: Scrollbar
robotSpeedTF: TextField
robotStatusTF: TextField
timeUntilUpdateSendTF: TextField
timeAtMessageSuccessfullySentTF: TextField
updateRateTF: TextField
collisionSafetyMarginTF: TextField
robotSpeedLabel: Label
robotStatusLabel: Label
timeUntilUpdateSendLabel: Label
timeAtMessageSuccessfullySentLabel: Label
updateRateLabel: Label
collisionSafetyMarginLabel: Label
ClientGUI(ClientApp,Robot)
actionPerformed(ActionEvent):void
adjustmentValueChanged(AdjustmentEvent):void
showTimeUntilUpdateSend(int):void
showTimeAtMessageSuccessfullySent(Date):void
showRobotStatus(String):void
getUpdateRate():int
-clientAppFrontend
0..1
-clientAppBackend
0..1
<<Java Class>>
ClientApp
client
DEFAULT_UPDATE_RATE: int
updateRate: int
updateRateChanged: boolean
cronometer: int
portNumber: int
socket: Socket
os: ObjectOutputStream
is: ObjectInputStream
timerTask: TimerTask
timer: Timer
ClientApp(String,String)
connectToServer(String):boolean
send(Object):void
updatePeriodically():void
updateInstantly():void
updateRateChanged():void
getClientGUIInstance():ClientGUI
readRobotObject():boolean
closeClientServerCommunication():void
main(String[]):void
-robot
0..1
<<Java Class>>
Robot
robot
HEADING_NORTH: int
HEADING_EAST: int
HEADING_SOUTH: int
HEADING_WEST: int
STATUS_ACTIVE: int
STATUS_PASSIVE: int
STATUS_COLLIDED: int
STATUS_HIBERNATE: int
STATUS_STUCK: int
DEFAULT_VELOCITY: int
ROBOT_MIN_SIZE: int
ROBOT_MAX_SIZE: int
SQRT_2: double
MAX_SAFETY_CIRCLE_DIAMETER: int
POSITIONS_HISTORY_ARRAY_MIN_CAPACITY: int
POSITIONS_HISTORY_ARRAY_MAX_CAPACITY: int
xCanvasThreshold: int
yCanvasThreshold: int
headOffset: int
positionsHistoryArrayCapacityToDisplay: int
positionsHistoryArrayIndex: int
prevPosition: Point
prevDrawingPoint: Point
rand: Random
drawingPoint: Point
collisionSafetyMarginCircleDiameter: int
collisionSafetyMarginCircleDrawingPoint: Point
rolesPool: String[]
sideLength: int
name: String
direction: int
role: String
position: Point
velocity: int
status: int
timeOfUpdate: Date
Robot(String)
getName():String
getRole():String
getSize():int
getCollisionSafetyMarginCircleDiameter():int
getDirectionAsInt():int
getPosition():Point
getVelocity():int
getTimeOfUpdate():Date
getDrawingPoint():Point
getPositionsHistoryArray():RobotHistory[]
getStatusAsInt():int
getCollisionSafetyMarginCircleDrawingPoint():Point
getPositionsHistoryArrayCapacity():int
getDirectionAsString():String
getStatusAsString():String
setPosition(int,int):void
setVelocity(int):void
setTimeOfUpdate(Date):void
setStatus(int):void
setPositionsHistoryArrayCapacity(int):void
setCollisionSafetyMarginCircleDiameter(int):void
init():void
moveForward(int):boolean
moveBackward(int):boolean
rotateLeft():void
rotateRight():void
updateRobotStates(Robot):void
addRobotPositionHistory(Point,Point,Date):boolean
getCollisionSafetyCircleMinDiameter():int
-robot
0..1
-robot
0..1
-positionsHistoryArray
0..*
<<Java Class>>
RobotHistory
robot
drawingPoint: Point
position: Point
timeOfUpdate: Date
RobotHistory(Point,Point,Date)
getDrawingPoint():Point
getPosition():Point
getTimeOfUpdate():Date
<<Java Class>>
DateTimeService
utils
calendar: Calendar
DateTimeService()
getDateAndTime():Date
-theDateService
0..1
<<Java Class>>
ServerApp
server
portNumber: int
ServerApp()
main(String[]):void
<<Java Class>>
ConnectionHandler
server
clientSocket: Socket
is: ObjectInputStream
os: ObjectOutputStream
ConnectionHandler(Socket,ServerGUI)
run():void
readRobotObject():boolean
getCurrentDateAndTime():Date
send(Object):void
closeSocket():void
printRobotDetails(Robot):void
-serverFrontend
0..1
-conHandlers
0..*
<<Java Class>>
ServerGUI
server
showOrHidePrevPositions: Button
previousPosNoLabel: Label
previousPosNoTF: TextField
previousPosNoScrollbar: Scrollbar
robotInfoTA: TextArea
robotsMap: ConcurrentMap<Robot,Color>
ServerGUI()
addRobot(Robot):void
updateGUICanvas():void
updateServerRobotProperties(Robot):void
displayRobotDetails(Robot):void
clearTextArea():void
actionPerformed(ActionEvent):void
adjustmentValueChanged(AdjustmentEvent):void
checkForRobotCollisions(ConcurrentMap<Robot,Color>):void
showCollision(Robot,Robot):void
deleteRobotFromServer(String):void
addConHandlerToServerGUILink(String,ConnectionHandler):void
closeServerProperly():void
-serverFrontend
0..1
-canvas
0..1
<<Java Class>>
ServerGUICanvas
server
CANVAS_WIDTH: int
CANVAS_HEIGHT: int
showPrevPositions: boolean
robotsMap: ConcurrentMap<Robot,Color>
ServerGUICanvas(ServerGUI,ConcurrentMap<Robot,Color>)
paint(Graphics):void
drawRobotHeadingNorth(Graphics,Robot):void
drawRobotHeadingEast(Graphics,Robot):void
drawRobotHeadingWest(Graphics,Robot):void
drawRobotHeadingSouth(Graphics,Robot):void
drawRobotPreviousPositions(Graphics,Robot):void
setShowPrevPositions():void
mouseClicked(MouseEvent):void
mousePressed(MouseEvent):void
mouseReleased(MouseEvent):void
mouseEntered(MouseEvent):void
mouseExited(MouseEvent):void

Fig. Class Diagram

General

Starting from the provided code for client and server, I built on top of it with the idea of a functional, easy to use and manage, application in mind.

To facilitate communication between client and server I chose to create Robot class(robot.Robot) which carry all information the client and server need to process. Outside of the name of the robot, all other properties are randomly generated or user-adjustable through client and server GUIs.



```
> Assignment2 [EE402_A2 main]
  JRE System Library [JavaSE-14]
  > src
    > client
      > ClientApp.java
      > ClientGUI.java
    > robot
      Robot.java
      RobotHistory.java
    > server
      > ConnectionHandler.java
      > ServerApp.java
      > ServerGUI.java
      > ServerGUICanvas.java
    utils
      DateTimeService.java
```

Project structure

I chose as the design of a robot a shape of arrow carved out from a square with the length of the side randomly generated, being constraint by a minimum and a maximum length specified by 2 of Robot class states(Robot.ROBOT_MIN_SIZE and Robot.ROBOT_MAX_SIZE).

For helping the robot carrying info about his previous positions, I create RobotHistory class (RobotHistory.java). Each robot will have an array of RobotHistory objects and the capacity of the array is indicated by Robot.POSITIONS_HISTORY_ARRAY_MAX_CAPACITY. The user can choose how many previous positions to be shown on server side in [Robot.POSITIONS_HISTORY_ARRAY_MIN_CAPACITY, Robot.POSITIONS_HISTORY_ARRAY_MAX_CAPACITY] range.

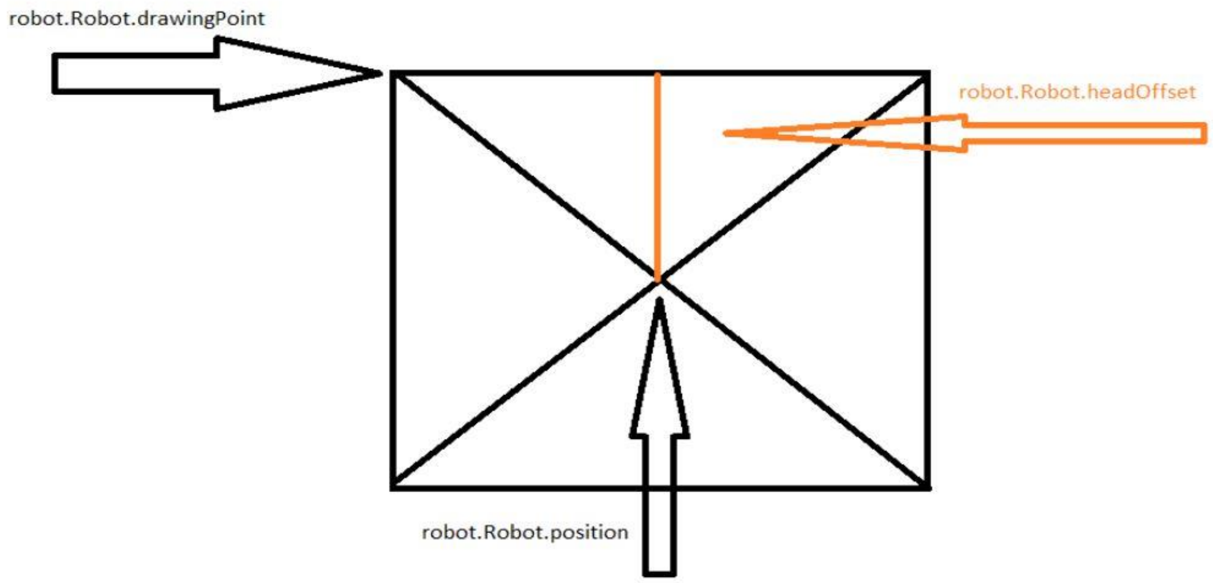


Fig. Robot layout

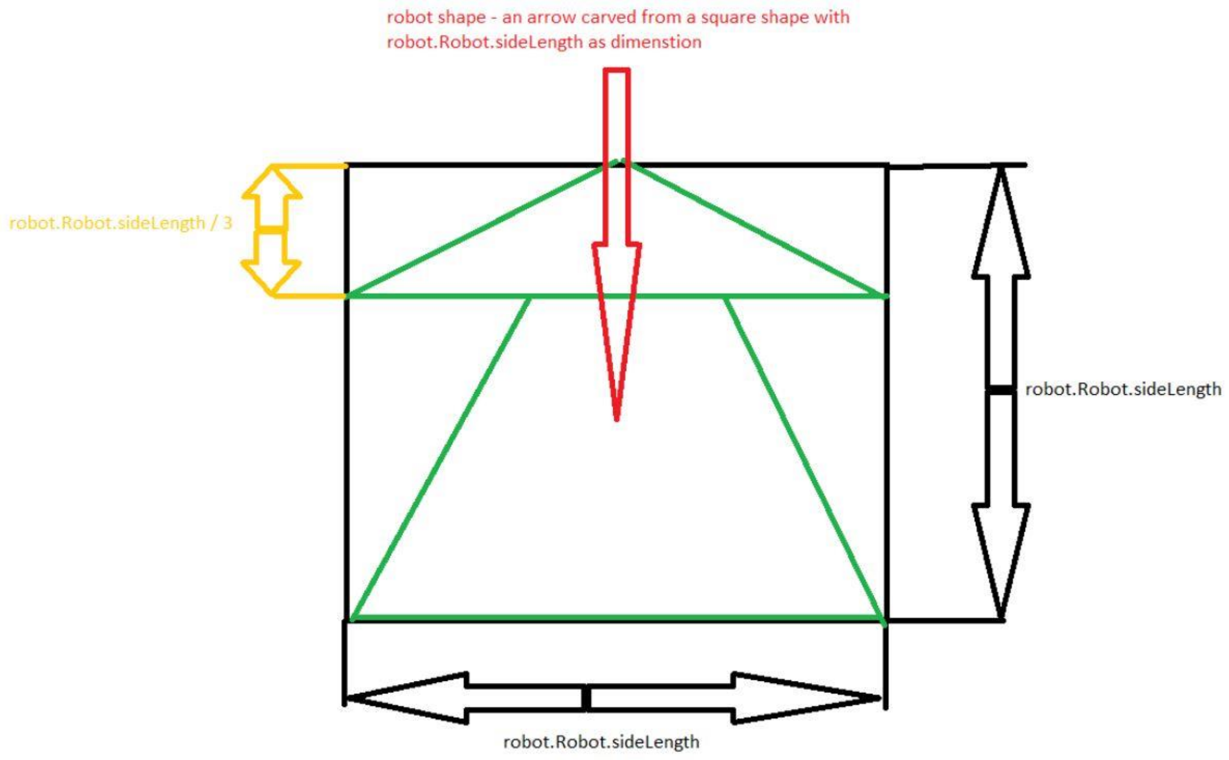
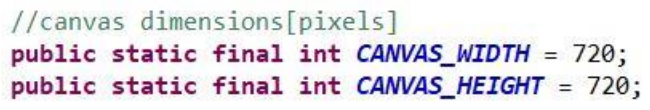

robot shape - an arrow carved from a square shape with robot.Robot.sideLength as dimenstion
robot.Robot.sideLength / 3
robot.Robot.sideLength
robot.Robot.sideLength

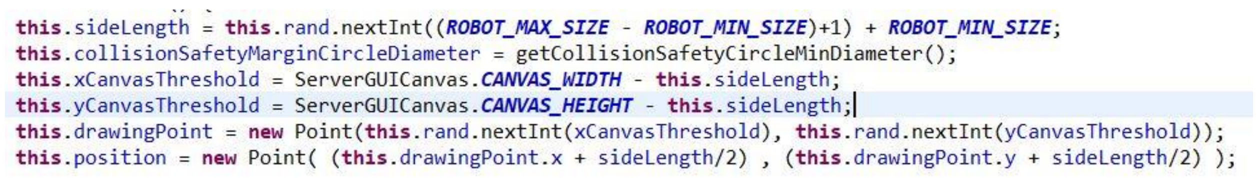
Fig. Robot layout

Robot.drawingPoint it is randomly generated, with constraints to not exceed canvas dimensions and based on it, the position of the robot it is computed to be the intersection point of the square diagonals from which robot arrow shape is carved.



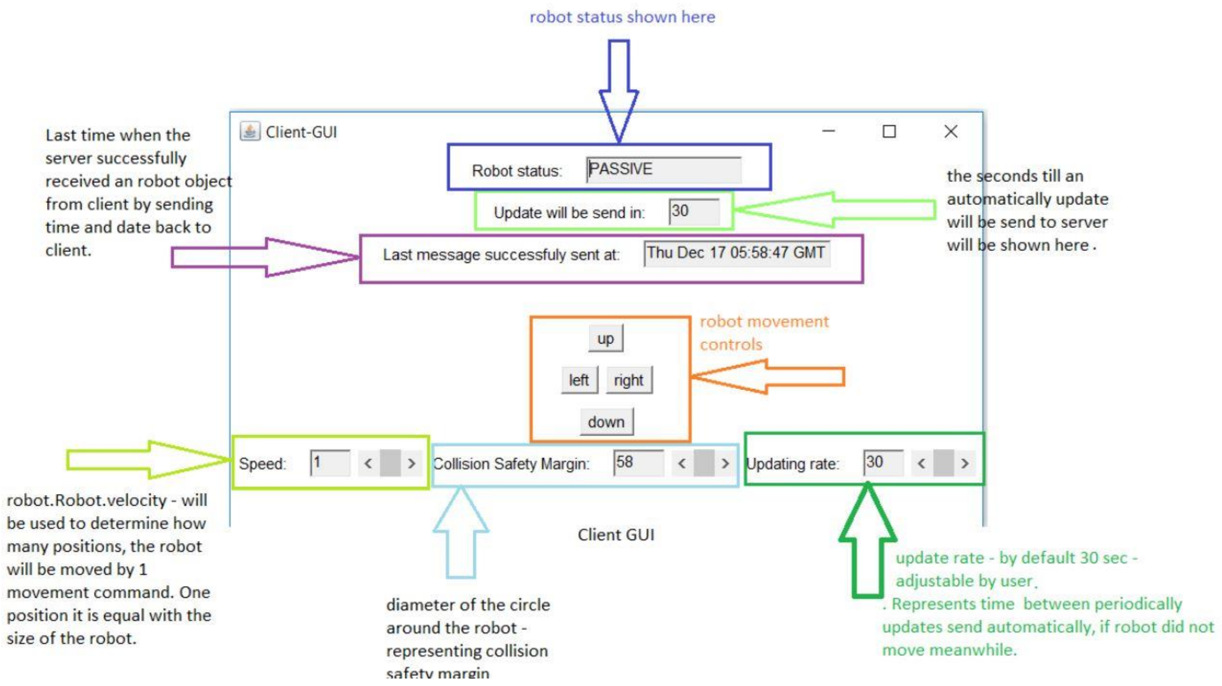
```
//canvas dimensions[pixels]
public static final int CANVAS_WIDTH = 720;
public static final int CANVAS_HEIGHT = 720;
```

Fig. canvas dimensions



```
this.sideLength = this.rand.nextInt((ROBOT_MAX_SIZE - ROBOT_MIN_SIZE)+1) + ROBOT_MIN_SIZE;
this.collisionSafetyMarginCircleDiameter = getCollisionSafetyCircleMinDiameter();
this.xCanvasThreshold = ServerGUICanvas.CANVAS_WIDTH - this.sideLength;
this.yCanvasThreshold = ServerGUICanvas.CANVAS_HEIGHT - this.sideLength;
this.drawingPoint = new Point(this.rand.nextInt(xCanvasThreshold), this.rand.nextInt(yCanvasThreshold));
this.position = new Point( (this.drawingPoint.x + sideLength/2) , (this.drawingPoint.y + sideLength/2) );
```

Fig. drawingPointConstraints



robot status shown here

Last time when the server successfully received an robot object from client by sending time and date back to client.

Robot status: PASSIVE

Update will be send in: 30

Last message successfuly sent at: Thu Dec 17 05:58:47 GMT

the seconds till an automatically update will be send to server will be shown here .

robot movement controls

Speed: 1

Collision Safety Margin: 58

Updating rate: 30

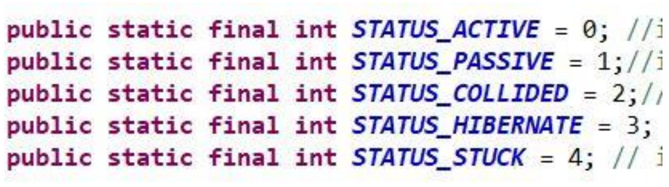
robot.Robot.velocity - will be used to determine how many positions, the robot will be moved by 1 movement command. One position it is equal with the size of the robot.

diameter of the circle around the robot - representing collision safety margin

update rate - by default 30 sec - adjustable by user.
. Represents time between periodically updates send automatically, if robot did not move meanwhile.

Fig. Client – GUI

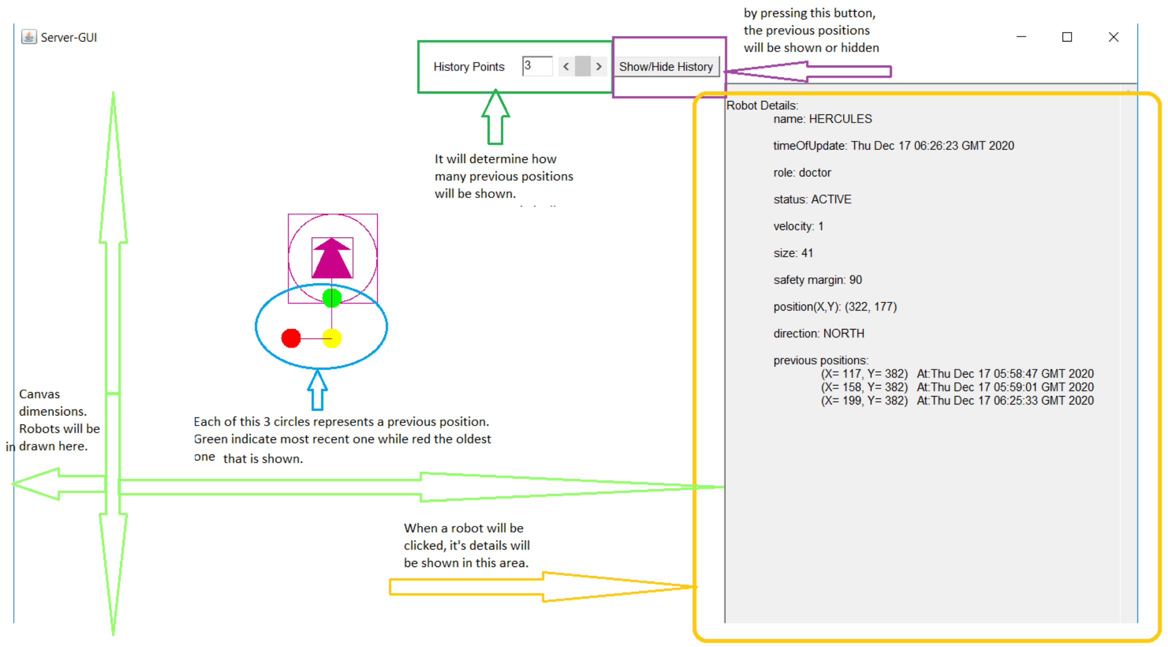
Robot can have 1 of the following statuses:



```
public static final int STATUS_ACTIVE = 0; //i
public static final int STATUS_PASSIVE = 1;//i
public static final int STATUS_COLLIDED = 2;//
public static final int STATUS_HIBERNATE = 3;
public static final int STATUS_STUCK = 4; // i
```

Fig. robot statuses

- **STATUS_HIBERNATE** – it is the default one, when a client just start, before any movement to be performed.
- **STATUS_COLLIDED** – when 2 robots are collided, both of them will show this status to the client and a message about collision will be displayed on server GUI too.
- **STATUS_ACTIVE** – when robot is moved by the user a with a faster rate than the automatically updates are sent.
- **STATUS_PASSIVE** – when robot do not changed its position and automatically updates were send.
- **STATUS_STUCK** – will be shown when a robot cannot go further(forward or backward) because of the canvas dimensions.



by pressing this button, the previous positions will be shown or hidden

It will determine how many previous positions will be shown.

Each of this 3 circles represents a previous position. Green indicate most recent one while red the oldest one that is shown.

Canvas dimensions. Robots will be in drawn here.

When a robot will be clicked, it's details will be shown in this area.

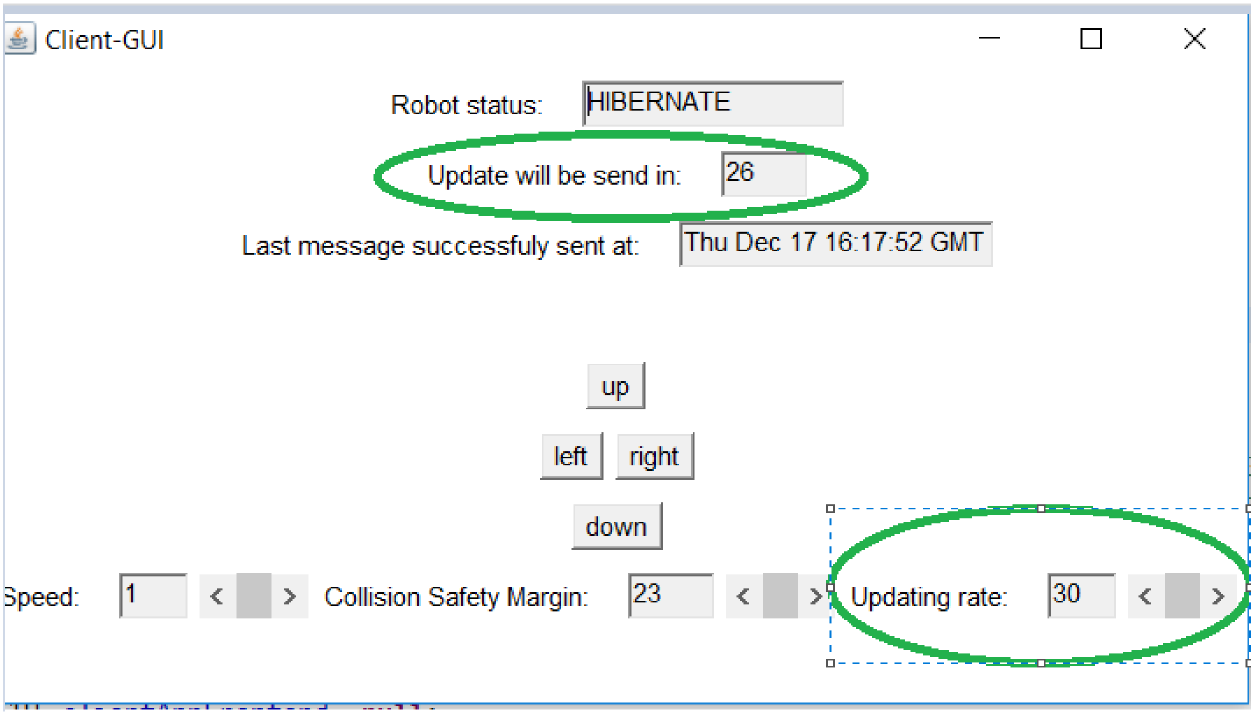
```
Robot Details:
        name: HERCULES

        timeOfUpdate: Thu Dec 17 06:26:23 GMT 2020

        role: doctor

        status: ACTIVE

        velocity: 1

        size: 41

        safety margin: 90

        position(X,Y): (322, 177)

        direction: NORTH

        previous positions:
                (X= 117, Y= 382)   At:Thu Dec 17 05:58:47 GMT 2020
                (X= 158, Y= 382)   At:Thu Dec 17 05:59:01 GMT 2020
                (X= 199, Y= 382)   At:Thu Dec 17 06:25:33 GMT 2020
```

Fig. Server – GUI

Number of previous positions can be shown up to Robot.POSITIONS_HISTORY_ARRAY_MAX_CAPACITY but I chose only 3 of them to be colored with different colors(red, yellow, green) and the other ones to be the same color as the object being easily to be observer on Canvas. If more than 3 previous positions will be shown, the colored ones(red, yellow, green) will be the least recent, marking the end of the history of the object.

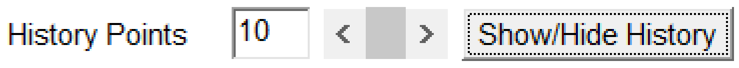
Features

- On the client side, the update rate is shown on client-GUI and it can be adjusted by user.

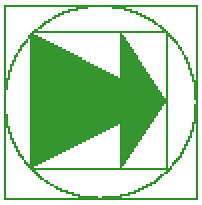


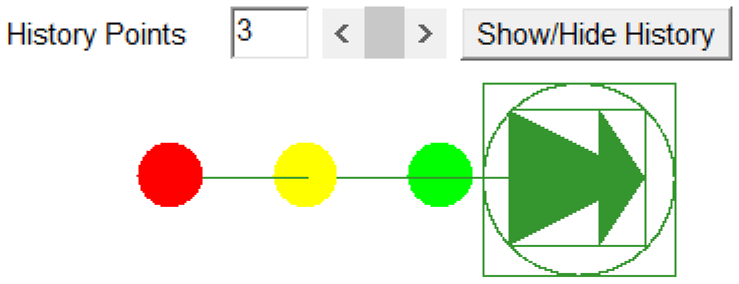
Robot status: HIBERNATE
Update will be send in: 26
Last message successfuly sent at: Thu Dec 17 16:17:52 GMT
Speed: 1
Collision Safety Margin: 23
Updating rate: 30

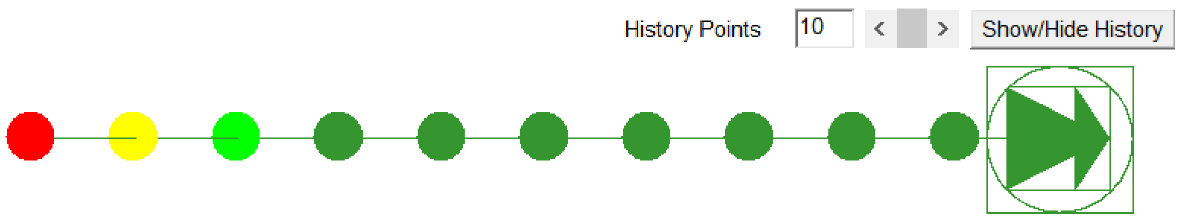
- On the server side, number of previous positions to be shown can be adjusted by user and the user can choose if to show previous positions or not.

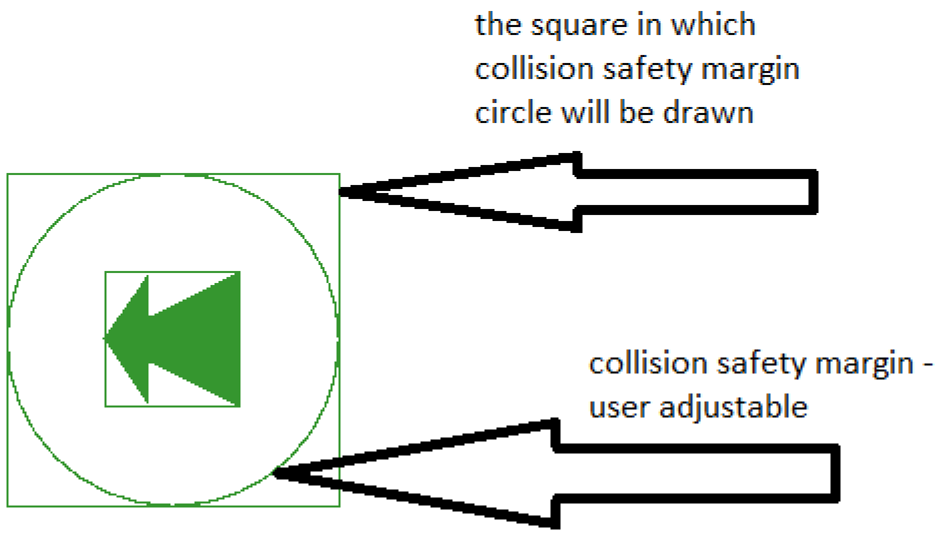


History Points 10





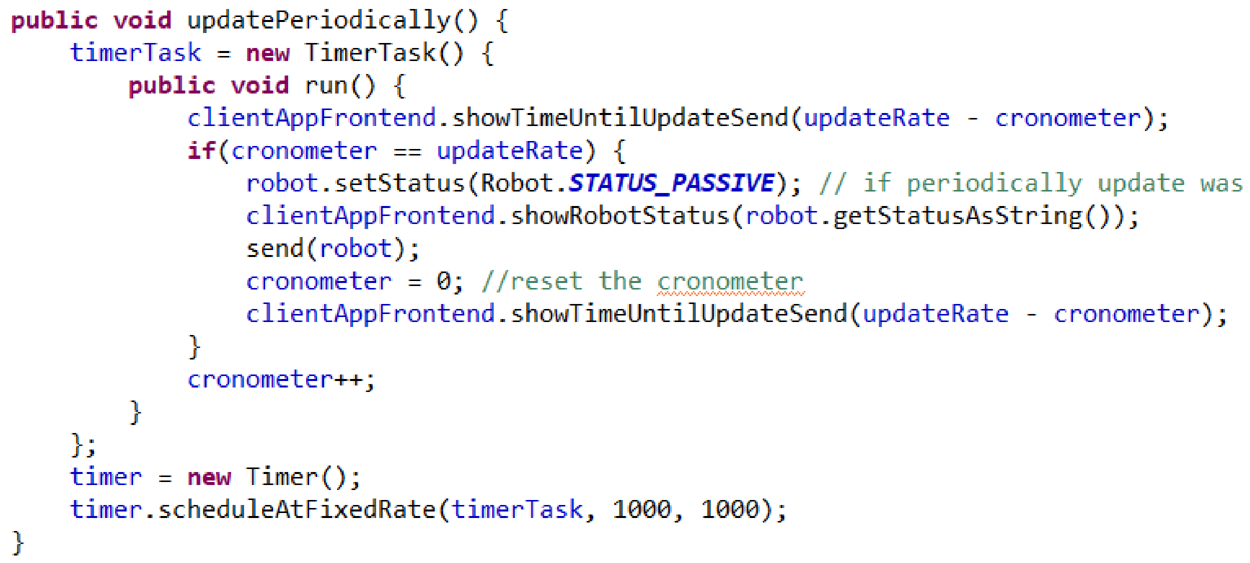



the square in which collision safety margin circle will be drawn
collision safety margin - user adjustable

I drew the additional rectangles(robot body and collision safety margin) making things more visible, as collision points.

Code Segments

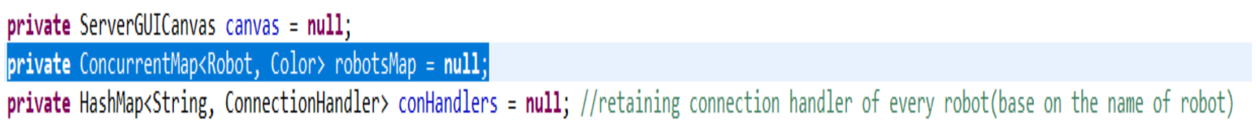
For periodically updated I used TimerTask and Time objects.



```java
public void updatePeriodically() {
    timerTask = new TimerTask() {
        public void run() {
            clientAppFrontend.showTimeUntilUpdateSend(updateRate - cronometer);
            if(cronometer == updateRate) {
                robot.setStatus(Robot.STATUS_PASSIVE); // if periodically update was
                clientAppFrontend.showRobotStatus(robot.getStatusAsString());
                send(robot);
                cronometer = 0; //reset the cronometer
                clientAppFrontend.showTimeUntilUpdateSend(updateRate - cronometer);
            }
            cronometer++;
        }
    };
    timer = new Timer();
    timer.scheduleAtFixedRate(timerTask, 1000, 1000);
}
```

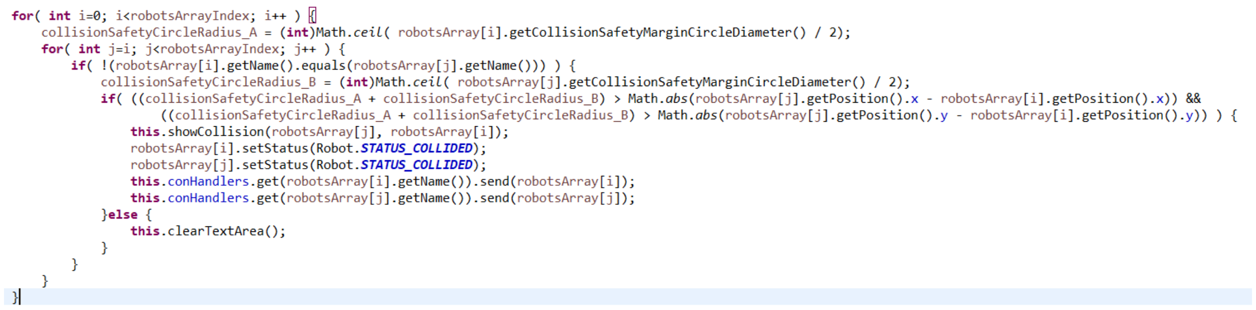
Fig. PeriodicallyUpdates

On server side, I used ConcurrentMap object as data structure to keep a track of all connected robots. Every time a robot connected to the server, a color was assigned to it and bind them in the form of <key, value> using ConcurrentMap<Robot, Color>. The ConnectionHandler for each of robots were stored in a HashMap<String, ConnectionHandler> collection for later use, as when the server want to indicate STATUS_COLLIDED to the client.



```java
private ServerGUICanvas canvas = null;
private ConcurrentMap<Robot, Color> robotsMap = null;
private HashMap<String, ConnectionHandler> conHandlers = null; //retaining connection handler of every robot(base on the name of robot)
```

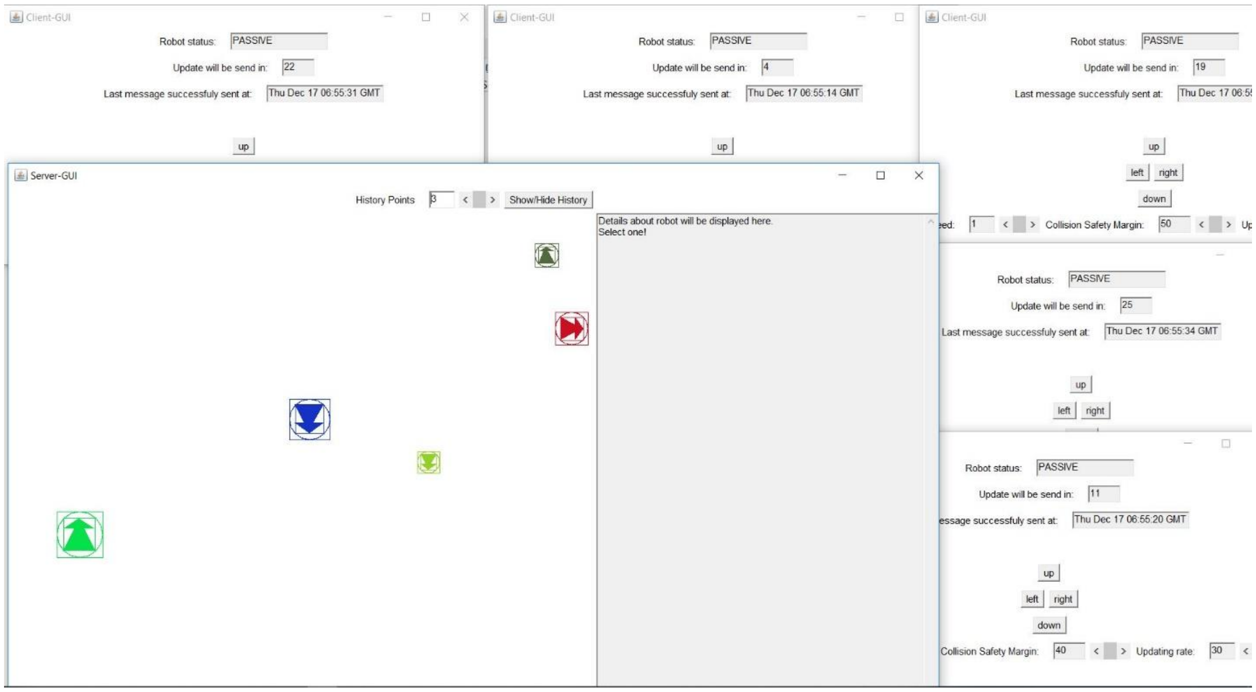
I checked for robots collisions based on each robot position(center of the robot) and the radius of the collision safety margin circle.



```java
for( int i=0; i<robotsArrayIndex; i++ ) {
    collisionSafetyCircleRadius_A = (int)Math.ceil( robotsArray[i].getCollisionSafetyMarginCircleDiameter() / 2);
    for( int j=i; j<robotsArrayIndex; j++ ) {
        if( !(robotsArray[i].getName().equals(robotsArray[j].getName())) ) {
            collisionSafetyCircleRadius_B = (int)Math.ceil( robotsArray[j].getCollisionSafetyMarginCircleDiameter() / 2);
            if( ((collisionSafetyCircleRadius_A + collisionSafetyCircleRadius_B) > Math.abs(robotsArray[j].getPosition().x - robotsArray[i].getPosition().x)) &&
                    ((collisionSafetyCircleRadius_A + collisionSafetyCircleRadius_B) > Math.abs(robotsArray[j].getPosition().y - robotsArray[i].getPosition().y)) ) {
                this.showCollision(robotsArray[j], robotsArray[i]);
                robotsArray[i].setStatus(Robot.STATUS_COLLIDED);
                robotsArray[j].setStatus(Robot.STATUS_COLLIDED);
                this.conHandlers.get(robotsArray[i].getName()).send(robotsArray[i]);
                this.conHandlers.get(robotsArray[j].getName()).send(robotsArray[j]);
            }else {
                this.clearTextArea();
            }
        }
    }
}
```

Testing

For testing the application I used 5 robots connected simultaneously.



Client-GUI
Robot status: PASSIVE
Update will be send in: 22
Last message successfuly sent at: Thu Dec 17 06:55:31 GMT

Client-GUI
Robot status: PASSIVE
Update will be send in: 4
Last message successfuly sent at: Thu Dec 17 06:55:14 GMT

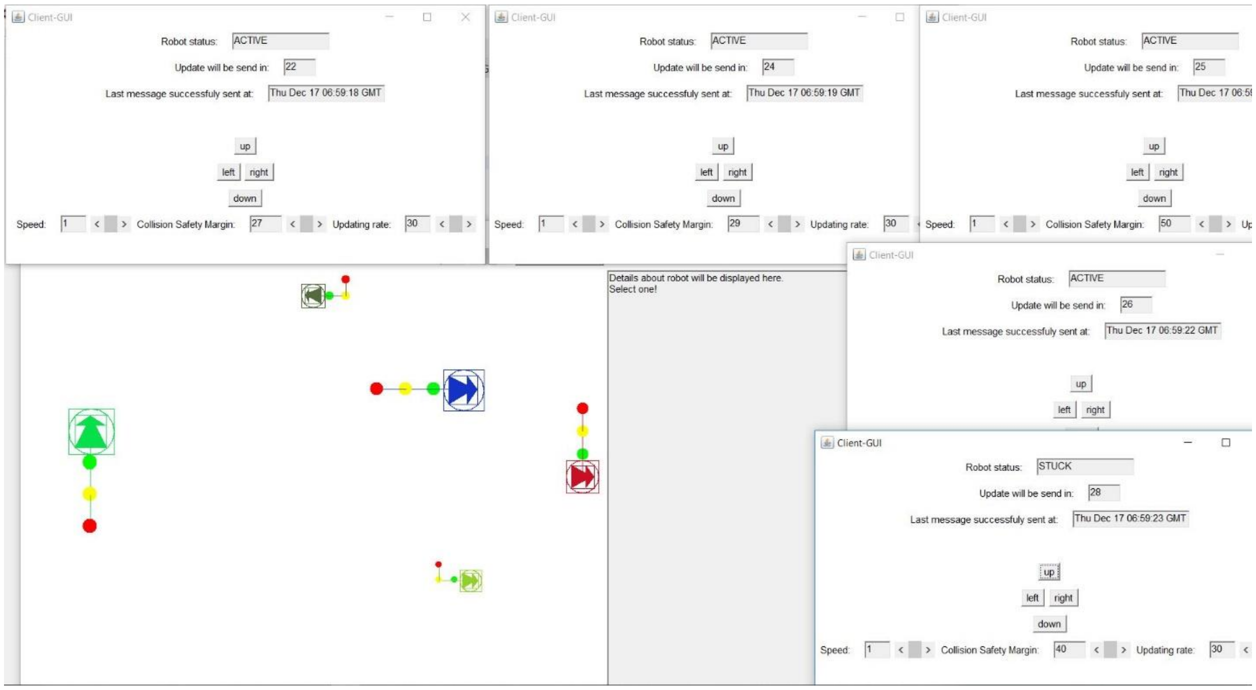
Client-GUI
Robot status: PASSIVE
Update will be send in: 19
Last message successfuly sent at: Thu Dec 17 06:5
eed: 1 Collision Safety Margin: 50 Up

Server-GUI
History Points 3
Details about robot will be displayed here.
Select one!

Robot status: PASSIVE
Update will be send in: 25
Last message successfuly sent at: Thu Dec 17 06:55:34 GMT

Robot status: PASSIVE
Update will be send in: 11
essage successfuly sent at: Thu Dec 17 06:55:20 GMT
Collision Safety Margin: 40 Updating rate: 30

Firg. RunningApp_1



Client-GUI
Robot status: ACTIVE
Update will be send in: 22
Last message successfuly sent at: Thu Dec 17 06:59:18 GMT
Speed: 1 Collision Safety Margin: 27 Updating rate: 30

Client-GUI
Robot status: ACTIVE
Update will be send in: 24
Last message successfuly sent at: Thu Dec 17 06:59:19 GMT
Speed: 1 Collision Safety Margin: 29 Updating rate: 30

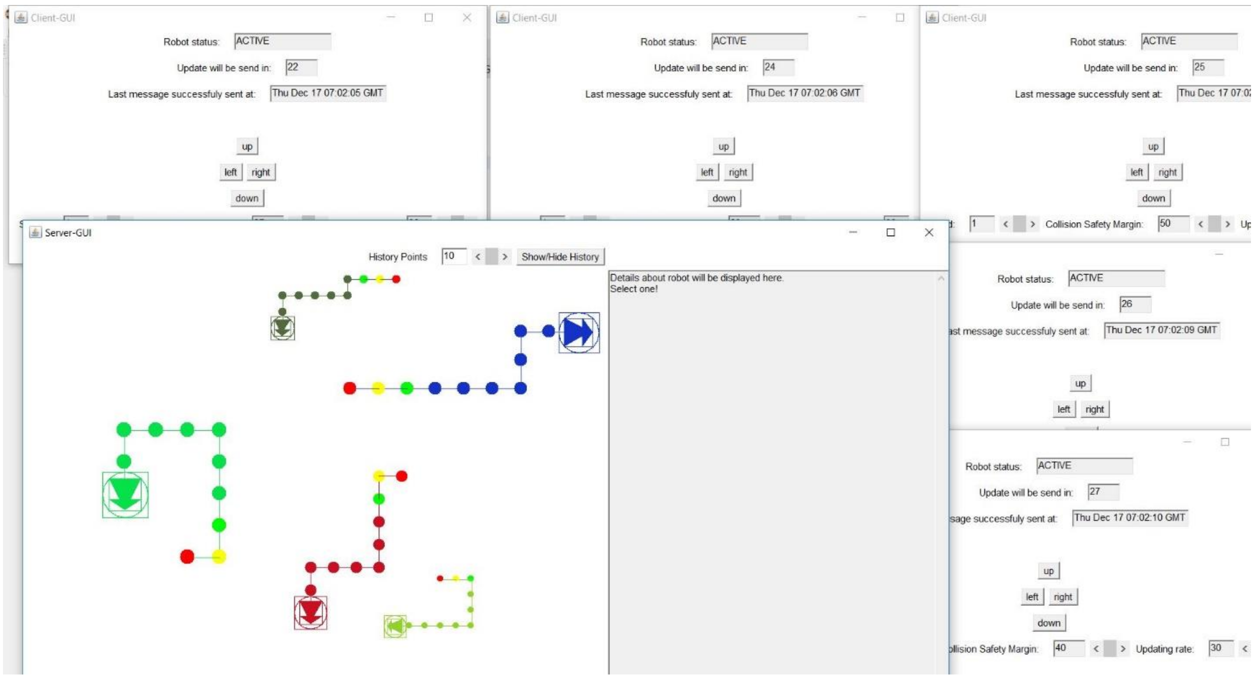
Client-GUI
Robot status: ACTIVE
Update will be send in: 25
Last message successfuly sent at: Thu Dec 17 06:5
Speed: 1 Collision Safety Margin: 50 Up

Details about robot will be displayed here.
Select one!

Client-GUI
Robot status: ACTIVE
Update will be send in: 26
Last message successfuly sent at: Thu Dec 17 06:59:22 GMT

Client-GUI
Robot status: STUCK
Update will be send in: 28
Last message successfuly sent at: Thu Dec 17 06:59:23 GMT
Speed: 1 Collision Safety Margin: 40 Updating rate: 30

Fig. RunnigApp_2



Robot status: ACTIVE
Update will be send in: 22
Last message successfuly sent at: Thu Dec 17 07:02:05 GMT

Robot status: ACTIVE
Update will be send in: 24
Last message successfuly sent at: Thu Dec 17 07:02:06 GMT

Robot status: ACTIVE
Update will be send in: 25
Last message successfuly sent at: Thu Dec 17 07:0

1 Collision Safety Margin: 50 Up

History Points 10

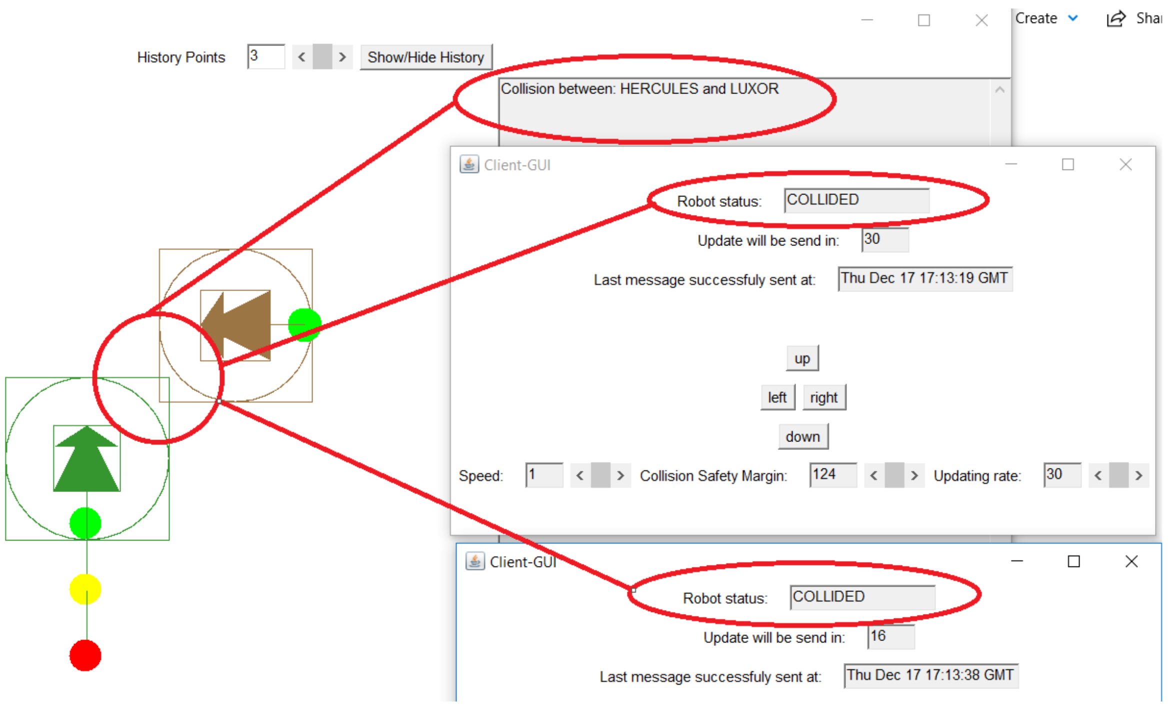
Details about robot will be displayed here.
Select one!

Robot status: ACTIVE
Update will be send in: 26
ast message successfuly sent at: Thu Dec 17 07:02:09 GMT

Robot status: ACTIVE
Update will be send in: 27
sage successfuly sent at: Thu Dec 17 07:02:10 GMT

llision Safety Margin: 40 Updating rate: 30

Fig. RunningApp_3



History Points 3

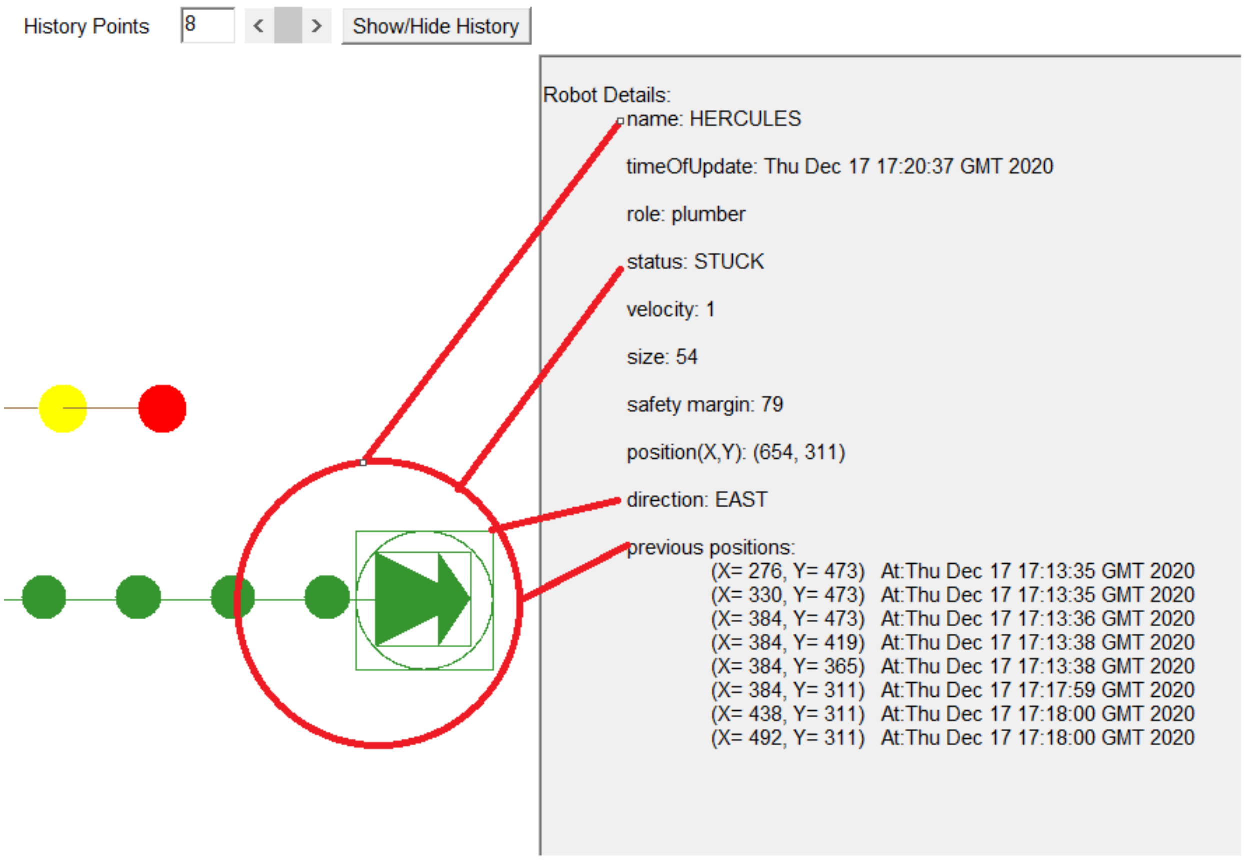
Collision between: HERCULES and LUXOR

Robot status: COLLIDED
Update will be send in: 30
Last message successfuly sent at: Thu Dec 17 17:13:19 GMT

Speed: 1 Collision Safety Margin: 124 Updating rate: 30

Robot status: COLLIDED
Update will be send in: 16
Last message successfuly sent at: Thu Dec 17 17:13:38 GMT

Fig.RunnigApp_4_Collision

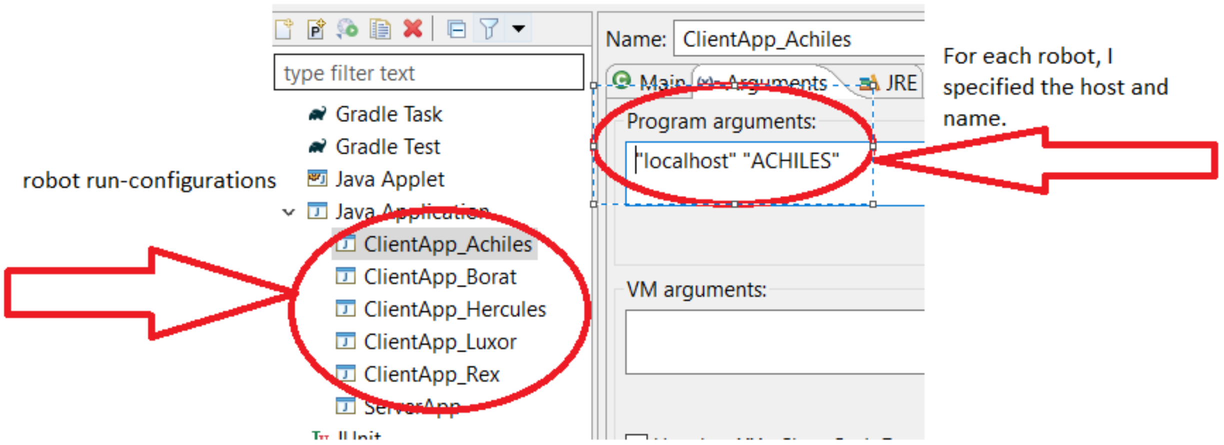


```
Robot Details:
    name: HERCULES
    timeOfUpdate: Thu Dec 17 17:20:37 GMT 2020
    role: plumber
    status: STUCK
    velocity: 1
    size: 54
    safety margin: 79
    position(X,Y): (654, 311)
    direction: EAST
    previous positions:
            (X= 276, Y= 473)   At:Thu Dec 17 17:13:35 GMT 2020
            (X= 330, Y= 473)   At:Thu Dec 17 17:13:35 GMT 2020
            (X= 384, Y= 473)   At:Thu Dec 17 17:13:36 GMT 2020
            (X= 384, Y= 419)   At:Thu Dec 17 17:13:38 GMT 2020
            (X= 384, Y= 365)   At:Thu Dec 17 17:13:38 GMT 2020
            (X= 384, Y= 311)   At:Thu Dec 17 17:17:59 GMT 2020
            (X= 438, Y= 311)   At:Thu Dec 17 17:18:00 GMT 2020
            (X= 492, Y= 311)   At:Thu Dec 17 17:18:00 GMT 2020
```

How To Run

Entry point for running server is represented by class server.ServerApp.java and entry point for client is client.ClientApp.java. Server should be started with no command line argument while for client, the host and name of the robot should be specified.

I ran the application from Eclipse IDE and created 5 run-configuration for the robots.



Name: ClientApp_Achiles

Program arguments:

```
"localhost" "ACHILES"
```

I attached .java and .class files as an archive to my submission.