/\*\*

\* This class creates a normal road block.

\*/

**public** **class** **Block** {

**private** **TrafficLight** t;

**private** **DropOff** d;

**private** **RoadTile** r;

**private** **boolean** hasTrafficLight;

**private** **boolean** hasDropOff;

**private** **boolean** hasRoadTile;

/\*\*

\* This method sets a normal block.

\*/

**public** **Block**() {

hasTrafficLight = **false**;

hasDropOff = **false**;

hasRoadTile = **false**;

}

/\*\*

\* This method sets a traffic light block.

\* **@param** t retrieves traffic light settings from TrafficLight class

\*/

**public** **void** **setTrafficLight**(**TrafficLight** t) {

**this**.t = t;

hasTrafficLight = **true**;

}

/\*\*

\* This method sets a drop off point block.

\* **@param** d settings from DropOff class

\*/

**public** **void** **setDropOff**(**DropOff** d) {

**this**.d = d;

hasDropOff = **true**;

}

/\*\*

\* This method sets up a RoadTile block.

\* **@param** r settings from RoadTile class

\*/

**public** **void** **setRoadTile**(**RoadTile** r) {

**this**.r = r;

hasRoadTile = **true**;

}

/\*\*

\* This method returns traffic light settings.

\* **@return** traffic light settings

\*/

**public** **boolean** **getHasTrafficLight**() {

**return** hasTrafficLight;

}

/\*\*

\* This method returns drop off point settings.

\* **@return** drop off point settings

\*/

**public** **boolean** **hasDropOff**() {

**return** hasDropOff;

}

/\*\*

\* This method returns road tile settings.

\* **@return** road tile settings

\*/

**public** **boolean** **hasRoadTile**() {

**return** hasRoadTile;

}

/\*\*

\* This method returns a traffic light block.

\* **@return** traffic light block

\*/

**public** **TrafficLight** **getTrafficLight**() {

**return** t;

}

/\*\*

\* This method returns a drop off point block.

\* **@return** drop off point block

\*/

**public** **DropOff** **getDropOff**() {

**return** d;

}

/\*\*

\* This method returns a road tile block.

\* **@return** road tile block

\*/

**public** **RoadTile** **getRoadTile**() {

**return** r;

}

}

import java.awt.Color;

import java.util.concurrent.ThreadLocalRandom;

/\*\*

\* This class creates a bus.

\*/

public class Bus extends Vehicle {

private int x;

private int y;

private int x2;

private int y2;

private int lastDirection;

private boolean hasMoved;

private boolean[] neighbor; //This array holds the status of the neighbors of the car.

private int[] road; //This array holds the directions of the surrounding roads.

private int time;

private int speed = chooseNum(1, 2, 3);

private boolean canMove;

private boolean sideStepped;

/\*\*

\* Constructor method for class Bus.

\* @param x x coordinate of front part of bus

\* @param y y coordinate of front part of bus

\*/

public Bus(int x, int y) {

this.x = x;

this.y = y;

x2 = x;

y2 = y;

super.chooseColor();

sideStepped = false;

}

/\*\*

\* This methods updates the values of the Bus.

\*/

public void update() {

if(!canMove) {

if(time > speed) {

canMove = true;

time = 0;

}

time++;

}

}

/\*\*

\* This method gives the bus the direction it should go.

\* @param direction integer that represents direction

\*/

public void move(int direction) {

update();

if(canMove) {

canMove = false;

switch(direction) {

//If the direction is up, move up.

case 0 :

if(road[0] != 1 && neighbor[0] == false) {

x2 = x;

y2 = y;

y--;

lastDirection = 0;

sideStepped = false;

}

else if(road[2] == 0 && !neighbor[2] && !sideStepped && x2 != x - 1) {

x2 = x;

y2 = y;

x--;

lastDirection = 0;

sideStepped = true;

}

else if(road[3] == 0 && !neighbor[3] && !sideStepped && x2 != x + 1) {

x2 = x;

y2 = y;

x++;

lastDirection = 0;

sideStepped = true;

}

break;

//If the direction is down, move down.

case 1:

if(road[1] != 0 && neighbor[1] == false) {

x2 = x;

y2 = y;

y++;

lastDirection = 1;

}

else if(road[2] == 1 && !neighbor[2] && !sideStepped && x2 != x - 1) {

x2 = x;

y2 = y;

x--;

lastDirection = 1;

sideStepped = true;

}

else if(road[3] == 1 && !neighbor[3] && !sideStepped && x2 != x + 1) {

x2 = x;

y2 = y;

x++;

lastDirection = 1;

sideStepped = true;

}

break;

//if the direction is left, move left.

case 2:

if(road[2] != 3 && neighbor[2] == false) {

x2 = x;

y2 = y;

x--;

lastDirection = 2;

}

else if(road[0] == 2 && !neighbor[0] && !sideStepped && y2 != y - 1) {

x2 = x;

y2 = y;

y--;

lastDirection = 2;

sideStepped = true;

}

else if(road[1] == 2 && !neighbor[1] && !sideStepped && y2 != y - 1) {

x2 = x;

y2 = y;

y++;

lastDirection = 2;

sideStepped = true;

}

break;

//If the direction is right, move right.

case 3:

if(road[3] != 2 && neighbor[3] == false) {

x2 = x;

y2 = y;

x++;

lastDirection = 3;

}

else if(road[0] == 3 && !neighbor[0] && !sideStepped && y2 != y - 1) {

x2 = x;

y2 = y;

y--;

lastDirection = 3;

sideStepped = true;

}

else if(road[1] == 3 && !neighbor[1] && !sideStepped && y2 != y - 1) {

x2 = x;

y2 = y;

y++;

lastDirection = 3;

sideStepped = true;

}

break;

//If the car is on an intersection, have fun.

case 4:

canMove = true;

switch(lastDirection) {

//If the car was from SOUTH, do this.

case 0:

if(road[0] != 1 && road[2] != 3 && road[3] != 2 && !neighbor[0] && !neighbor[2] && !neighbor[3])

move(chooseNum(0, 2, 3));

else if(road[0] != 1 && road[2] != 3 && !neighbor[0] && !neighbor[2])

move(chooseNum(0, 2));

else if(road[0] != 1 && road[3] != 2 && !neighbor[0] && !neighbor[3])

move(chooseNum(0, 3));

else if(road[2] != 3 && road[3] != 2 && !neighbor[2] && !neighbor[3])

move(chooseNum(2, 3));

else if(road[0] != 1 && (road[2] == 3 || neighbor[2]) && (road[3] == 2 || neighbor[3]) && !neighbor[0])

move(0);

else if((road[0] == 1 || neighbor[0]) && road[2] != 3 && (road[3] == 2 || neighbor[3]) && !neighbor[2])

move(2);

else if((road[0] == 1 || neighbor[0]) && (road[2] == 3 || neighbor[2]) && road[3] != 2 && !neighbor[3])

move(3);

else if(road[0] != 1 && !neighbor[0])

move(0);

else if(road[2] != 3 && !neighbor[2])

move(2);

else if(road[3] != 2 && !neighbor[3])

move(3);

break;

//If the car was from NORTH, do this.

case 1:

if(road[1] != 0 && road[2] != 3 && road[3] != 2 && !neighbor[1] && !neighbor[2] && !neighbor[3])

move(chooseNum(1, 2, 3));

else if(road[1] != 0 && road[2] != 3 && !neighbor[1] && !neighbor[2])

move(chooseNum(1, 2));

else if(road[1] != 0 && road[3] != 2 && !neighbor[1] && !neighbor[3])

move(chooseNum(1, 3));

else if(road[2] != 3 && road[3] != 2 && !neighbor[2] && !neighbor[3])

move(chooseNum(2, 3));

else if((road[1] == 0 || !neighbor[1]) && road[2] != 3 && (road[3] == 2 || !neighbor[3]) && !neighbor[3])

move(3);

else if((road[1] == 0 || !neighbor[1]) && (road[2] == 3 || !neighbor[2]) && road[3] != 2 && !neighbor[2])

move(2);

else if(road[1] != 0 && (road[2] == 3 || !neighbor[2]) && (road[3] == 2 || !neighbor[3]) && !neighbor[1])

move(1);

else if(road[1] != 0 && !neighbor[1])

move(1);

else if(road[2] != 3 && !neighbor[2])

move(2);

else if(road[3] != 2 && !neighbor[3])

move(3);

break;

//If the car was from EAST, do this.

case 2:

if(road[0] != 1 && road[1] != 0 && road[2] != 3 && !neighbor[0] && !neighbor[1] && !neighbor[2])

move(chooseNum(0, 1, 2));

else if(road[0] != 1 && road[1] != 0 && !neighbor[0] && !neighbor[1])

move(chooseNum(0, 1));

else if(road[0] != 1 && road[2] != 3 && !neighbor[0] && !neighbor[2])

move(chooseNum(0, 2));

else if(road[1] != 0 && road[2] != 3 && !neighbor[1] && !neighbor[2])

move(chooseNum(1, 2));

else if((road[0] == 1 || !neighbor[0]) && (road[1] == 0 || !neighbor[1]) && road[2] != 3 && !neighbor[2])

move(2);

else if((road[0] == 1 || !neighbor[0]) && road[1] != 0 && (road[2] == 3 || !neighbor[2]) && !neighbor[1])

move(1);

else if(road[0] != 1 && (road[1] == 0 || !neighbor[1]) && (road[2] == 3 || !neighbor[2]) && !neighbor[0])

move(0);

else if(road[0] != 1 && !neighbor[0])

move(0);

else if(road[1] != 0 && !neighbor[1])

move(1);

else if(road[2] != 3 && !neighbor[2])

move(2);

break;

//If the car was from WEST, do this.

case 3:

if(road[0] != 1 && road[1] != 0 && road[3] != 2 && !neighbor[0] && !neighbor[1] && !neighbor[3])

move(chooseNum(0, 1, 3));

else if(road[0] != 1 && road[1] != 0 && !neighbor[0] && !neighbor[1])

move(chooseNum(0, 1));

else if(road[0] != 1 && road[3] != 2 && !neighbor[0] && !neighbor[3])

move(chooseNum(0, 3));

else if(road[1] != 0 && road[3] != 2 && !neighbor[1] && !neighbor[3])

move(chooseNum(1, 3));

else if((road[0] == 1 || !neighbor[0]) && (road[1] == 0 || !neighbor[1]) && road[3] != 2 && !neighbor[3])

move(3);

else if(road[0] != 1 && (road[1] == 0 || !neighbor[1]) && (road[3] == 2 || !neighbor[3]) && !neighbor[0])

move(0);

else if((road[0] == 1 || !neighbor[0]) && road[1] != 0 && (road[3] == 2 || !neighbor[3]) && !neighbor[1])

move(1);

else if(road[0] != 1 && !neighbor[0])

move(0);

else if(road[1] != 0 && !neighbor[1])

move(1);

else if(road[3] != 2 && !neighbor[3])

move(3);

break;

}

break;

}

}

}

/\*\*

\* this method checks if the bus had moved to the next block

\* @param moved settings to check if the bus had moved

\*/

public void setHasMoved(boolean moved) {

hasMoved = moved;

}

/\*\*

\* this method checks if the blocks before and after the bus have vehicles

\* @param neighbor vehicle on the blocks adjacent to the bus

\*/

public void setNeighbors(boolean[] neighbor) {

this.neighbor = neighbor;

}

/\*\*

\* this method sets the path of the bus

\* @param road array of the path the bus had took

\*/

public void setRoad(int[] road) {

this.road = road;

}

/\*\*

\* this method returns the current x coordinate of the front part of the bus

\* @return x coordinate of the front part of the bus

\*/

public int getX() {

return x;

}

/\*\*

\* This method returns the current y coordinate of the front part of the bus

\* @return y coordinate of the front part of the bus

\*/

public int getY() {

return y;

}

/\*\*

\* this method returns the x coordinate of the back part of the bus

\* @return x coordinate of the back part of the bus

\*/

public int getX2() {

return x2;

}

/\*\*

\* this method returns the y coordinate of the back part of the bus

\* @return y coordinate of the back part of the bus

\*/

public int getY2() {

return y2;

}

/\*\*

\* This method checks if the bus had moved to the next block

\* @return if the bus had moved

\*/

public boolean getHasMoved() {

return hasMoved;

}

/\*\*

\* This method returns the color of the bus

\* @return color of the bus

\*/

public Color getColor() {

return super.color;

}

}

import java.awt.Color;

import java.util.concurrent.ThreadLocalRandom;

/\*\*

\* This class creates a car.

\*/

public class Car extends Vehicle{

private int x;

private int y;

private int lastDirection;

private boolean hasMoved;

private boolean[] neighbor; //This array holds the status of the neighbors of the car.

private int[] road; //This array holds the directions of the surrounding roads.

private boolean sideStepped;

private int time;

private int speed = chooseNum(1, 2, 3);

private boolean canMove;

/\*\*

\* Constructor method for Car class.

\* @param x current x coordinate of the car

\* @param y current y coordinate of the car

\*/

public Car(int x, int y) {

this.x = x;

this.y = y;

super.chooseColor();

sideStepped = false;

time = 0;

canMove = true;

}

/\*\*

\* This method updates the values of the Car.

\*/

public void update() {

if(!canMove) {

if(time > speed) {

canMove = true;

time = 0;

}

time++;

}

}

/\*\*

\* This method sets the direction of the car based on the block.

\* @param direction integer that represents the direction the car needs to move

\*/

public void move(int direction) {

update();

if(canMove) {

canMove = false;

switch(direction) {

//If the direction is up, move up.

case 0 :

if(road[0] != 1 && neighbor[0] == false) {

y--;

lastDirection = 0;

sideStepped = false;

}

else if(road[2] == 0 && !neighbor[2] && !sideStepped) {

x--;

lastDirection = 0;

sideStepped = true;

}

else if(road[3] == 0 && !neighbor[3] && !sideStepped) {

x++;

lastDirection = 0;

sideStepped = true;

}

break;

//If the direction is down, move down.

case 1:

if(road[1] != 0 && neighbor[1] == false) {

y++;

lastDirection = 1;

sideStepped = false;

}

else if(road[2] == 1 && !neighbor[2] && !sideStepped) {

x--;

lastDirection = 1;

sideStepped = true;

}

else if(road[3] == 1 && !neighbor[3] && !sideStepped) {

x++;

lastDirection = 1;

sideStepped = true;

}

break;

//if the direction is left, move left.

case 2:

if(road[2] != 3 && neighbor[2] == false) {

x--;

lastDirection = 2;

sideStepped = false;

}

else if(road[0] == 2 && !neighbor[0] && !sideStepped) {

y--;

lastDirection = 2;

sideStepped = true;

}

else if(road[1] == 2 && !neighbor[1] && !sideStepped) {

y++;

lastDirection = 2;

sideStepped = true;

}

break;

//If the direction is right, move right.

case 3:

if(road[3] != 2 && neighbor[3] == false) {

x++;

lastDirection = 3;

sideStepped = false;

}

else if(road[0] == 3 && !neighbor[0] && !sideStepped) {

y--;

lastDirection = 3;

sideStepped = true;

}

else if(road[1] == 3 && !neighbor[1] && !sideStepped) {

y++;

lastDirection = 3;

sideStepped = true;

}

break;

//If the car is on an intersection, have fun.

case 4:

canMove = true;

switch(lastDirection) {

//If the car was from SOUTH, do this.

case 0:

if(road[0] != 1 && road[2] != 3 && road[3] != 2 && !neighbor[0] && !neighbor[2] && !neighbor[3])

move(chooseNum(0, 2, 3));

else if(road[0] != 1 && road[2] != 3 && !neighbor[0] && !neighbor[2])

move(chooseNum(0, 2));

else if(road[0] != 1 && road[3] != 2 && !neighbor[0] && !neighbor[3])

move(chooseNum(0, 3));

else if(road[2] != 3 && road[3] != 2 && !neighbor[2] && !neighbor[3])

move(chooseNum(2, 3));

else if(road[0] != 1 && (road[2] == 3 || neighbor[2]) && (road[3] == 2 || neighbor[3]) && !neighbor[0])

move(0);

else if((road[0] == 1 || neighbor[0]) && road[2] != 3 && (road[3] == 2 || neighbor[3]) && !neighbor[2])

move(2);

else if((road[0] == 1 || neighbor[0]) && (road[2] == 3 || neighbor[2]) && road[3] != 2 && !neighbor[3])

move(3);

else if(road[0] != 1 && !neighbor[0])

move(0);

else if(road[2] != 3 && !neighbor[2])

move(2);

else if(road[3] != 2 && !neighbor[3])

move(3);

break;

//If the car was from NORTH, do this.

case 1:

if(road[1] != 0 && road[2] != 3 && road[3] != 2 && !neighbor[1] && !neighbor[2] && !neighbor[3])

move(chooseNum(1, 2, 3));

else if(road[1] != 0 && road[2] != 3 && !neighbor[1] && !neighbor[2])

move(chooseNum(1, 2));

else if(road[1] != 0 && road[3] != 2 && !neighbor[1] && !neighbor[3])

move(chooseNum(1, 3));

else if(road[2] != 3 && road[3] != 2 && !neighbor[2] && !neighbor[3])

move(chooseNum(2, 3));

else if((road[1] == 0 || !neighbor[1]) && road[2] != 3 && (road[3] == 2 || !neighbor[3]) && !neighbor[3])

move(3);

else if((road[1] == 0 || !neighbor[1]) && (road[2] == 3 || !neighbor[2]) && road[3] != 2 && !neighbor[2])

move(2);

else if(road[1] != 0 && (road[2] == 3 || !neighbor[2]) && (road[3] == 2 || !neighbor[3]) && !neighbor[1])

move(1);

else if(road[1] != 0 && !neighbor[1])

move(1);

else if(road[2] != 3 && !neighbor[2])

move(2);

else if(road[3] != 2 && !neighbor[3])

move(3);

break;

//If the car was from EAST, do this.

case 2:

if(road[0] != 1 && road[1] != 0 && road[2] != 3 && !neighbor[0] && !neighbor[1] && !neighbor[2])

move(chooseNum(0, 1, 2));

else if(road[0] != 1 && road[1] != 0 && !neighbor[0] && !neighbor[1])

move(chooseNum(0, 1));

else if(road[0] != 1 && road[2] != 3 && !neighbor[0] && !neighbor[2])

move(chooseNum(0, 2));

else if(road[1] != 0 && road[2] != 3 && !neighbor[1] && !neighbor[2])

move(chooseNum(1, 2));

else if((road[0] == 1 || !neighbor[0]) && (road[1] == 0 || !neighbor[1]) && road[2] != 3 && !neighbor[2])

move(2);

else if((road[0] == 1 || !neighbor[0]) && road[1] != 0 && (road[2] == 3 || !neighbor[2]) && !neighbor[1])

move(1);

else if(road[0] != 1 && (road[1] == 0 || !neighbor[1]) && (road[2] == 3 || !neighbor[2]) && !neighbor[0])

move(0);

else if(road[0] != 1 && !neighbor[0])

move(0);

else if(road[1] != 0 && !neighbor[1])

move(1);

else if(road[2] != 3 && !neighbor[2])

move(2);

break;

//If the car was from WEST, do this.

case 3:

if(road[0] != 1 && road[1] != 0 && road[3] != 2 && !neighbor[0] && !neighbor[1] && !neighbor[3])

move(chooseNum(0, 1, 3));

else if(road[0] != 1 && road[1] != 0 && !neighbor[0] && !neighbor[1])

move(chooseNum(0, 1));

else if(road[0] != 1 && road[3] != 2 && !neighbor[0] && !neighbor[3])

move(chooseNum(0, 3));

else if(road[1] != 0 && road[3] != 2 && !neighbor[1] && !neighbor[3])

move(chooseNum(1, 3));

else if((road[0] == 1 || !neighbor[0]) && (road[1] == 0 || !neighbor[1]) && road[3] != 2 && !neighbor[3])

move(3);

else if(road[0] != 1 && (road[1] == 0 || !neighbor[1]) && (road[3] == 2 || !neighbor[3]) && !neighbor[0])

move(0);

else if((road[0] == 1 || !neighbor[0]) && road[1] != 0 && (road[3] == 2 || !neighbor[3]) && !neighbor[1])

move(1);

else if(road[0] != 1 && !neighbor[0])

move(0);

else if(road[1] != 0 && !neighbor[1])

move(1);

else if(road[3] != 2 && !neighbor[3])

move(3);

break;

}

break;

}

}

}

/\*\*

\* This method checks if the car had moved to the next block.

\* @param moved settings to check if the car had moved

\*/

public void setHasMoved(boolean moved) {

hasMoved = moved;

}

/\*\*

\* This method checks adjacent blocks.

\* @param neighbor nearby cars or buses

\*/

public void setNeighbors(boolean[] neighbor) {

this.neighbor = neighbor;

}

/\*\*

\* This method sets the path the car takes.

\* @param road array that stores the path the car took

\*/

public void setRoad(int[] road) {

this.road = road;

}

/\*\*

\* This method returns the current x coordinate of the car.

\* @return x coordinate of the car

\*/

public int getX() {

return x;

}

/\*\*

\* This method returns the current y coordinate of the car.

\* @return y coordinate of the car

\*/

public int getY() {

return y;

}

/\*\*

\* Checks if the car had already moved

\* @return settings if the car had already moved

\*/

public boolean getHasMoved() {

return hasMoved;

}

/\*\*

\* This method returns the color of the car

\* @return color of the car

\*/

public Color getColor() {

return super.color;

}

}

import java.awt.Color;

import java.awt.Graphics;

import java.awt.Graphics2D;

import java.util.ArrayList;

import javax.swing.JComponent;

/\*\*

\* This class displays the elements of the simulation.

\*/

public class Display extends JComponent {

private int width;

private int height;

private int windowWidth;

private int windowHeight;

private Map s;

private ArrayList <Car> carList;

private ArrayList <Bus> busList;

private ArrayList <Wall> wallList;

private ArrayList <Spawner> spawnerList;

private ArrayList <Road> roadList;

private ArrayList <DropOff> dropOffList;

private ArrayList <TrafficLight> trafficLightList;

private ArrayList <RoadTile> roadTileList;

/\*\*

\* Constructor method for Display class

\* @param s settings from Simulation class

\*/

public Display(Map s) {

this.s = s;

Thread animationThread = new Thread(new Runnable() {

public void run() {

while (true) {

repaint();

try {Thread.sleep(10);} catch (Exception ex) {}

}

}

});

animationThread.start();

}

/\*\*

\* This method displays("paints") the elements of the traffic simulation

\*/

public void paintComponent(Graphics gg) {

Graphics2D g = (Graphics2D) gg;

super.paintComponent(gg);

width = s.getWidth();

height = s.getHeight();

windowWidth = this.getWidth();

windowHeight = this.getHeight();

carList = new ArrayList <Car> (s.getCarList());

busList = new ArrayList <Bus> (s.getBusList());

wallList = new ArrayList <Wall> (s.getWallList());

spawnerList = new ArrayList <Spawner> (s.getSpawnerList());

roadList = new ArrayList <Road> (s.getRoadList());

dropOffList = new ArrayList <DropOff> (s.getDropOffList());

trafficLightList = new ArrayList <TrafficLight> (s.getTrafficLightList());

roadTileList = new ArrayList <RoadTile> (s.getRoadTileList());

float xWidth = windowWidth / width;

float yHeight = windowHeight / height;

int xstart;

int ystart;

int i;

int x, y, x1, y1, x2, y2;

//This is where the road tiles are painted.

for(i = 0; i < roadTileList.size(); i++) {

x = roadTileList.get(i).getX();

y = roadTileList.get(i).getY();

x1 = x \* (int) xWidth;

y1 = y \* (int) yHeight;

if(roadTileList.get(i).getCondition() == 1) {

gg.setColor(Color.LIGHT\_GRAY);

gg.fillRect(x1, y1, (int) xWidth, (int) yHeight);

}

else if(roadTileList.get(i).getCondition() == 2) {

gg.setColor(Color.CYAN);

gg.fillRect(x1, y1, (int) xWidth, (int) yHeight);

}

else if(roadTileList.get(i).getCondition() == 3) {

gg.setColor(Color.DARK\_GRAY);

gg.fillRect(x1, y1, (int) xWidth, (int) yHeight);

}

}

//This is where the drop off points are painted.

for(i = 0; i < dropOffList.size(); i++) {

x = dropOffList.get(i).getX();

y = dropOffList.get(i).getY();

if(dropOffList.get(i).isOccupied())

gg.setColor(Color.GRAY);

else

gg.setColor(Color.LIGHT\_GRAY);

x1 = x \* (int) xWidth;

y1 = y \* (int) yHeight;

gg.fillRect(x1, y1, (int) xWidth, (int) yHeight);

}

//This is where the traffic lights are painted.

for(i = 0; i < trafficLightList.size(); i++) {

x = trafficLightList.get(i).getX();

y = trafficLightList.get(i).getY();

if(trafficLightList.get(i).getGo())

gg.setColor(Color.GREEN);

else

gg.setColor(Color.RED);

x1 = x \* (int) xWidth;

y1 = y \* (int) yHeight;

gg.fillRect(x1, y1, (int) xWidth, (int) yHeight);

}

//This is where the spawners are created.

gg.setColor(Color.PINK);

for(i = 0; i < spawnerList.size(); i++) {

x = spawnerList.get(i).getX();

y = spawnerList.get(i).getY();

x1 = x \* (int) xWidth;

y1 = y \* (int) yHeight + ( (int) yHeight / 2);

x2 = x1 + (int) xWidth;

y2 = y1;

gg.drawLine(x1, y1, x2, y2);

x1 = x1 + ( (int) xWidth / 2);

y1 = y1 - ( (int) yHeight / 2);

x2 = x1;

y2 = y1 + (int) yHeight;

gg.drawLine(x1, y1, x2, y2);

}

//This is where the road directions are created.

gg.setColor(Color.ORANGE);

for(i = 0; i < roadList.size(); i++) {

int direction;

x = roadList.get(i).getX();

y = roadList.get(i).getY();

direction = roadList.get(i).getDirection();

if(direction == 4) {

x1 = x \* (int) xWidth;

y1 = y \* (int) yHeight;

x2 = x1 + (int) xWidth;

y2 = y1 + (int) yHeight;

gg.drawLine(x1, y1, x2, y2);

y1 = y2;

y2 = y1 - (int) yHeight;

gg.drawLine(x1, y1, x2, y2);

}

//This makes an arrow pointing up.

else if(direction == 0) {

x1 = x \* (int) xWidth + ( (int) xWidth / 2);

y1 = y \* (int) yHeight;

x2 = x1 - ( (int) xWidth / 2);

y2 = y1 + ( (int) yHeight);

gg.drawLine(x1, y1, x2, y2);

x2 = x2 + ( (int) xWidth);

gg.drawLine(x1, y1, x2, y2);

}

//This makes an arrow pointing down.

else if(direction == 1) {

x1 = x \* (int) xWidth + ( (int) xWidth / 2);

y1 = y \* (int) yHeight + (int) yHeight;

x2 = x1 - ( (int) xWidth / 2);

y2 = y1 - (int) yHeight;

gg.drawLine(x1, y1, x2, y2);

x2 = x1 + ( (int) xWidth / 2);

gg.drawLine(x1, y1, x2, y2);

}

//This makes an arrow pointing left.

else if(direction == 2) {

x1 = x \* (int) xWidth;

y1 = y \* (int) yHeight + ( (int) yHeight / 2);

x2 = x1 + (int) xWidth;

y2 = y1 - ( (int) yHeight / 2);

gg.drawLine(x1, y1, x2, y2);

y2 = y2 + ( (int) yHeight);

gg.drawLine(x1, y1, x2, y2);

}

//This makes an arrow pointing right.

else if(direction == 3) {

x1 = x \* (int) xWidth + (int) xWidth;

y1 = y \* (int) yHeight + ( (int) yHeight / 2);

x2 = x1 - (int) xWidth;

y2 = y1 - ((int) yHeight / 2);

gg.drawLine(x1, y1, x2, y2);

y2 = y2 + (int) yHeight;

gg.drawLine(x1, y1, x2, y2);

}

//This makes an intersection.

else if(direction == 4) {

x1 = x \* (int) xWidth + ( (int) xWidth / 2);

y1 = y \* (int) yHeight + ( (int) yHeight - ( (int) yHeight / 8 ) );

gg.drawString("" + direction, x1, y1);

}

}

//This is where the vertical lines are printed.

gg.setColor(Color.BLACK);

for(i = 0; i < width; i++) {

xstart = i\*(windowWidth/width);

gg.drawLine(xstart, 0, xstart, windowHeight); //x1, y1, x2, y2

}

//This is where the horizontal lines are printed.

gg.setColor(Color.BLACK);

for(i = 0; i < height; i++) {

ystart = i\*(windowHeight/height);

gg.drawLine(0, ystart, windowWidth, ystart);

}

//This is where the walls are printed.

gg.setColor(Color.BLACK);

for(i = 0; i < wallList.size(); i++) {

x1 = wallList.get(i).getX() \* (int) xWidth;

y1 = wallList.get(i).getY() \* (int) yHeight;

x2 = (int) xWidth;

y2 = (int) yHeight;

gg.fillRect(x1, y1, x2, y2);

}

//This is where the cars are printed.

for(i = 0; i < carList.size(); i++) {

x = carList.get(i).getX();

y = carList.get(i).getY();

gg.setColor(carList.get(i).getColor());

x1 = x \* (int) xWidth + ( (int) xWidth / 4);

y1 = y \* (int) yHeight + ( (int) yHeight / 4);

gg.fillRect(x1, y1, (int) xWidth / 2, (int) yHeight / 2);

}

//This is where the bus are painted.

for(i = 0; i < busList.size(); i++) {

x = busList.get(i).getX();

y = busList.get(i).getY();

x2 = busList.get(i).getX2();

y2 = busList.get(i).getY2();

gg.setColor(busList.get(i).getColor());

x1 = x \* (int) xWidth + ( (int) xWidth / 4);

y1 = y \* (int) yHeight + ( (int) yHeight / 4);

//Paint this if the bus is going SOUTH.

if(x == x2 && y > y2) {

gg.fillRect(x1, y1 - (int) yHeight, (int) xWidth / 2, 3\*((int) yHeight / 2));

}

//Paint this if the bus is going NORTH.

else if(x == x2 && y < y2) {

gg.fillRect(x1, y1, (int) xWidth / 2, 3 \* ((int) yHeight / 2));

}

//Paint this if the bus is going EAST.

else if(x > x2 && y == y2) {

gg.fillRect(x1 - (int) xWidth, y1, 3 \* ((int) xWidth / 2), (int) yHeight / 2);

}

//Paint this if the bus is going WEST.

else if(x < x2 && y == y2) {

gg.fillRect(x1, y1, 3\*((int) xWidth / 2), (int) yHeight / 2);

}

else

gg.fillRect(x1, y1, (int) xWidth / 2, (int) yHeight / 2);

}

repaint();

}

}

/\*\*

\* Traffic Flow Simulator.

\*

\* Road traffic has been a perennial problem in the Metro. Is has been the caused of delays and stress to commuters.

\* Estimates by a Japanese study in 2012 found that time lost by people stuck in traffic and the extra cost of operating vehicles in gridlock amount to 2.4 billion pesos ($51 million) a day.

\*

\* @author Cyrus A. Vatandoost Kakhki

\* @author Julius Ceasar Librada

\*/

import java.io.IOException;

public class Driver {

public static void main(String[] args) throws IOException {

MainMenu m = new MainMenu();

m.run();

}

}

/\*\*

\* These are tiles of the map where vehicles can stop for a specified time duration.

\* The time is assigned as the drop-off time points is placed on the road map.

\*

\* **@author** user

\*

\*/

**public** **class** **DropOff** {

**private** **int** x;

**private** **int** y;

**private** **int** limit;

**private** **int** time;

**private** **boolean** occupied;

**private** **boolean** free;

/\*\*

\* This method sets a Drop Off block.

\* **@param** x x coordinate of the block

\* **@param** y y coordinate of the block

\* **@param** limit time the vehicle has to drop off

\*/

**public** **DropOff**(**int** x, **int** y, **int** limit) {

**this**.x = x;

**this**.y = y;

**this**.limit = limit;

time = 0;

free = **true**;

occupied = **false**;

}

/\*\*

\* This method updates the drop off point synchronized to the simulation time.

\*/

**public** **void** **update**() {

**if**(time >= limit && !free) {

free = **true**;

time = 0;

}

time++;

}

/\*\*

\* This method checks if the drop off point block has a car standing on top of it.

\*/

**public** **void** **occupy**() {

free = **false**;

occupied = **true**;

}

/\*\*

\* This method moves the car on top off it after a period of time which is preset by us or set by the user.

\*/

**public** **void** **unOccupy**() {

occupied = **false**;

free = **true**;

}

/\*\*

\* This method checks if the drop off point is free.

\* **@return** the current condition of the block.

\*/

**public** **boolean** **isFree**() {

**return** free;

}

/\*\*

\* This method checks and returns if the block is free.

\* **@return** the current condition of the block.

\*/

**public** **boolean** **isOccupied**() {

**return** occupied;

}

/\*\*

\* This method returns the x coordinate of the block.

\* **@return** x coordinate of the block

\*/

**public** **int** **getX**() {

**return** x;

}

/\*\*

\* This method returns the y coordinate of the block.

\* **@return** y coordinate of the block

\*/

**public** **int** **getY**() {

**return** y;

}

/\*\*

\* This returns the limit (or rate for dropping off) of the drop off.

\* **@return** limit of the drop off

\*/

**public** **int** **getLimit**() {

**return** limit;

}

}

import java.awt.BorderLayout;

import java.awt.EventQueue;

import javax.swing.JFrame;

import javax.swing.JPanel;

import javax.swing.border.EmptyBorder;

import javax.swing.JTable;

import javax.swing.JLabel;

import javax.swing.JRadioButton;

import javax.swing.JTextField;

import javax.swing.JSeparator;

import javax.swing.JButton;

import javax.swing.JComboBox;

import javax.swing.SwingConstants;

import java.awt.event.ActionListener;

import java.io.FileNotFoundException;

import java.io.IOException;

import java.io.UnsupportedEncodingException;

import java.net.URISyntaxException;

import java.util.ArrayList;

import java.awt.event.ActionEvent;

import javax.swing.JRadioButtonMenuItem;

import javax.swing.JMenu;

import javax.swing.JMenuBar;

import javax.swing.JMenuItem;

import javax.swing.JOptionPane;

import javax.swing.JToolBar;

import javax.swing.JInternalFrame;

import javax.swing.JDesktopPane;

import javax.swing.JFileChooser;

import javax.swing.JToggleButton;

import org.eclipse.wb.swing.FocusTraversalOnArray;

import java.awt.Component;

import java.awt.event.MouseAdapter;

import java.awt.event.MouseEvent;

import java.awt.event.MouseMotionAdapter;

import java.awt.event.KeyAdapter;

import java.awt.event.KeyEvent;

import java.awt.Font;

/\*\*

\* This class provides the settings for the menu where the user can edit the components of the simulation.

\*/

public class EditorMenu extends JFrame{

private JPanel contentPane;

private String mapName;

private int mouseX;

private int mouseY;

EditorModel m = new EditorModel();

MainMenu mm;

private JTextField roadX;

private JTextField wallY;

private JTextField wallX;

private JTextField roadY;

private JTextField spawnerX;

private JTextField spawnerY;

private JTextField spawnerRate;

private JTextField trafficLightX;

private JTextField trafficLightY;

private JTextField trafficLightRate;

private JTextField dropOffX;

private JTextField dropOffY;

private JTextField dropOffRate;

private JTextField width;

private JTextField height;

private JTextField roadTileX;

private JTextField roadTileY;

private boolean running;

/\*\*

\* Constructor method for EditorMenu class

\*/

public EditorMenu() {

setResizable(false);

setTitle("Editor Menu");

setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

setBounds(100, 100, 700, 600);

contentPane = new JPanel();

contentPane.addMouseListener(new MouseAdapter() {

@Override

public void mousePressed(MouseEvent e) {

if(e.getButton() == e.BUTTON3) {

mouseX = m.getMouseX();

mouseY = m.getMouseY();

roadX.setText(mouseX + "");

wallX.setText(mouseX + "");

spawnerX.setText(mouseX + "");

trafficLightX.setText(mouseX + "");

dropOffX.setText(mouseX + "");

roadTileX.setText(mouseX + "");

roadY.setText(mouseY + "");

wallY.setText(mouseY + "");

spawnerY.setText(mouseY + "");

trafficLightY.setText(mouseY + "");

dropOffY.setText(mouseY + "");

roadTileY.setText(mouseY + "");

}

}

});

contentPane.setBorder(new EmptyBorder(5, 5, 5, 5));

setContentPane(contentPane);

contentPane.setLayout(null);

setLocationRelativeTo(null);

JButton newButton = new JButton("New");

JButton clear = new JButton("Clear");

JButton deselect = new JButton("Deselect");

JButton display = new JButton("Display");

JButton addRoadTile = new JButton("Add");

JButton addDropOff = new JButton("Add");

JButton removeRoad = new JButton("Remove");

JButton removeWall = new JButton("Remove");

JButton removeSpawner = new JButton("Remove");

JButton removeDropOff = new JButton("Remove");

JButton removeTrafficLight = new JButton("Remove");

JButton removeRoadTile = new JButton("Remove");

JLabel message = new JLabel("");

message.setBounds(10, 535, 660, 25);

contentPane.add(message);

JLabel mapNameLabel = new JLabel("no file selected");

mapNameLabel.setFont(new Font("Tahoma", Font.ITALIC, 11));

mapNameLabel.setHorizontalAlignment(SwingConstants.CENTER);

mapNameLabel.setBounds(10, 27, 100, 25);

contentPane.add(mapNameLabel);

JButton roadDirection = new JButton("Up");

roadDirection.setFont(new Font("Tahoma", Font.PLAIN, 10));

roadDirection.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e) {

if(roadDirection.getText() == "Up")

roadDirection.setText("Down");

else if(roadDirection.getText() == "Down")

roadDirection.setText("Left");

else if(roadDirection.getText() == "Left")

roadDirection.setText("Right");

else if(roadDirection.getText() == "Right")

roadDirection.setText("Intersection");

else if(roadDirection.getText() == "Intersection")

roadDirection.setText("Up");

}

});

roadDirection.setBounds(230, 136, 100, 25);

contentPane.add(roadDirection);

JButton addRoad = new JButton("Add");

addRoad.setEnabled(false);

addRoad.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e) {

if(roadDirection.getText() == "Up")

m.getMap().addRoad(Integer.parseInt(roadX.getText()), Integer.parseInt(roadY.getText()), 0);

else if(roadDirection.getText() == "Down")

m.getMap().addRoad(Integer.parseInt(roadX.getText()), Integer.parseInt(roadY.getText()), 1);

else if(roadDirection.getText() == "Left")

m.getMap().addRoad(Integer.parseInt(roadX.getText()), Integer.parseInt(roadY.getText()), 2);

else if(roadDirection.getText() == "Right")

m.getMap().addRoad(Integer.parseInt(roadX.getText()), Integer.parseInt(roadY.getText()), 3);

else if(roadDirection.getText() == "Intersection")

m.getMap().addRoad(Integer.parseInt(roadX.getText()), Integer.parseInt(roadY.getText()), 4);

}

});

addRoad.setBounds(230, 171, 100, 25);

contentPane.add(addRoad);

JButton addWall = new JButton("Add");

addWall.setEnabled(false);

addWall.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e) {

m.getMap().addWall(Integer.parseInt(wallX.getText()), Integer.parseInt(wallY.getText()));

}

});

addWall.setBounds(400, 171, 100, 25);

contentPane.add(addWall);

JButton addSpawner = new JButton("Add");

addSpawner.setEnabled(false);

addSpawner.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e) {

m.getMap().addSpawner(Integer.parseInt(spawnerX.getText()), Integer.parseInt(spawnerY.getText()), Integer.parseInt(spawnerRate.getText()));

}

});

addSpawner.setBounds(570, 171, 100, 25);

contentPane.add(addSpawner);

JButton trafficLightStart = new JButton("Go");

trafficLightStart.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e) {

if(trafficLightStart.getText() == "Go")

trafficLightStart.setText("Stop");

else

trafficLightStart.setText("Go");

}

});

trafficLightStart.setBounds(400, 423, 100, 25);

contentPane.add(trafficLightStart);

JButton addTrafficLight = new JButton("Add");

addTrafficLight.setEnabled(false);

addTrafficLight.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e) {

if(trafficLightStart.getText() == "Go")

m.getMap().addTrafficLight(Integer.parseInt(trafficLightX.getText()), Integer.parseInt(trafficLightY.getText()), Integer.parseInt(trafficLightRate.getText()), 0);

else

m.getMap().addTrafficLight(Integer.parseInt(trafficLightX.getText()), Integer.parseInt(trafficLightY.getText()), Integer.parseInt(trafficLightRate.getText()), 1);

}

});

addTrafficLight.setBounds(400, 459, 100, 25);

contentPane.add(addTrafficLight);

JLabel lblAddSpawner = new JLabel("Add Spawner");

lblAddSpawner.setFont(new Font("Tahoma", Font.PLAIN, 10));

lblAddSpawner.setHorizontalAlignment(SwingConstants.CENTER);

lblAddSpawner.setBounds(570, 27, 100, 25);

contentPane.add(lblAddSpawner);

JLabel lblAddWall = new JLabel("Add Wall");

lblAddWall.setFont(new Font("Tahoma", Font.PLAIN, 10));

lblAddWall.setHorizontalAlignment(SwingConstants.CENTER);

lblAddWall.setBounds(397, 32, 88, 14);

contentPane.add(lblAddWall);

JLabel lblAddDropOff = new JLabel("Add Drop Off");

lblAddDropOff.setFont(new Font("Tahoma", Font.PLAIN, 10));

lblAddDropOff.setHorizontalAlignment(SwingConstants.CENTER);

lblAddDropOff.setBounds(230, 279, 100, 25);

contentPane.add(lblAddDropOff);

JLabel lblAddTrafficLight = new JLabel("Add Traffic Light");

lblAddTrafficLight.setFont(new Font("Tahoma", Font.PLAIN, 10));

lblAddTrafficLight.setHorizontalAlignment(SwingConstants.CENTER);

lblAddTrafficLight.setBounds(400, 279, 100, 25);

contentPane.add(lblAddTrafficLight);

JMenuBar menuBar = new JMenuBar();

menuBar.setBounds(0, 0, 694, 21);

contentPane.add(menuBar);

JMenu mnTools = new JMenu("File");

menuBar.add(mnTools);

JButton scan = new JButton("Scan");

scan.setEnabled(false);

scan.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e) {

try {

if(!m.isEmpty(mapName)) {

m.setMapName(mapName);

try {

m.setMap();

} catch (URISyntaxException e1) {

// TODO Auto-generated catch block

e1.printStackTrace();

}

}

else {

message.setText("file is empty");

}

} catch (FileNotFoundException e1) {

// TODO Auto-generated catch block

e1.printStackTrace();

}

m.displayMap();

addRoad.setEnabled(true);

addWall.setEnabled(true);

addSpawner.setEnabled(true);

addDropOff.setEnabled(true);

addRoadTile.setEnabled(true);

addTrafficLight.setEnabled(true);

removeRoad.setEnabled(true);

removeWall.setEnabled(true);

removeSpawner.setEnabled(true);

removeDropOff.setEnabled(true);

removeRoadTile.setEnabled(true);

removeTrafficLight.setEnabled(true);

scan.setEnabled(false);

newButton.setEnabled(true);

display.setEnabled(true);

clear.setEnabled(true);

deselect.setEnabled(true);

running = true;

}

});

JMenuItem mntmSave = new JMenuItem("Save");

mntmSave.setEnabled(false);

mntmSave.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e) {

try {

m.save();

} catch (FileNotFoundException e1) {

e1.printStackTrace();

} catch (UnsupportedEncodingException e1) {

e1.printStackTrace();

}

}

});

mnTools.add(mntmSave);

JMenu mnSelectMap = new JMenu("Select Map");

menuBar.add(mnSelectMap);

JMenuItem map1 = new JMenuItem("Map 1");

mnSelectMap.add(map1);

JMenuItem map2 = new JMenuItem("Map 2");

mnSelectMap.add(map2);

JMenuItem map3 = new JMenuItem("Map 3");

mnSelectMap.add(map3);

JMenuItem map4 = new JMenuItem("Map 4");

mnSelectMap.add(map4);

JMenuItem map5 = new JMenuItem("Map 5");

mnSelectMap.add(map5);

map5.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e) {

mapNameLabel.setText("map\_005.txt");

mapName = mapNameLabel.getText();

message.setText("map\_005.txt selected");

scan.setEnabled(true);

mntmSave.setEnabled(true);

}

});

map4.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e) {

mapNameLabel.setText("map\_004.txt");

mapName = mapNameLabel.getText();

message.setText("map\_004.txt selected");

scan.setEnabled(true);

mntmSave.setEnabled(true);

}

});

map3.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e) {

mapNameLabel.setText("map\_003.txt");

mapName = mapNameLabel.getText();

message.setText("map\_003.txt selected");

scan.setEnabled(true);

mntmSave.setEnabled(true);

}

});

map2.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e) {

mapNameLabel.setText("map\_002.txt");

mapName = mapNameLabel.getText();

message.setText("map\_002.txt selected");

scan.setEnabled(true);

mntmSave.setEnabled(true);

}

});

map1.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e) {

mapNameLabel.setText("map\_001.txt");

mapName = mapNameLabel.getText();

message.setText("map\_001.txt selected");

scan.setEnabled(true);

mntmSave.setEnabled(true);

}

});

JMenu mnHelp = new JMenu("Help");

menuBar.add(mnHelp);

JMenuItem mntmHelp = new JMenuItem("Help Contents");

mntmHelp.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e) {

JOptionPane frame = new JOptionPane();

JOptionPane.showMessageDialog(frame,

"STEPS TO EDIT:\n" +

"1. Select a map from the Select Menu.\n" +

"2. Scan the map.\n" +

"3. To display the map, press the display button\n" +

"4. You can use the add and remove buttons to add the respective entities\n" +

"5. If you would left click on the map and right click in the Layout Manager Menu, it would automatically input the x and y for you." +

"6. Don't forget to save!"

);

}

});

mnHelp.add(mntmHelp);

display.setEnabled(false);

display.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e) {

m.toggleVisibility();

}

});

scan.setBounds(10, 63, 100, 25);

contentPane.add(scan);

display.setBounds(10, 99, 100, 25);

contentPane.add(display);

roadX = new JTextField();

roadX.setBounds(230, 63, 100, 25);

contentPane.add(roadX);

roadX.setColumns(10);

wallY = new JTextField();

wallY.setBounds(400, 99, 100, 25);

contentPane.add(wallY);

wallY.setColumns(10);

wallX = new JTextField();

wallX.setBounds(400, 63, 100, 25);

contentPane.add(wallX);

wallX.setColumns(10);

JLabel lblY = new JLabel("y");

lblY.setFont(new Font("Tahoma", Font.PLAIN, 10));

lblY.setHorizontalAlignment(SwingConstants.RIGHT);

lblY.setBounds(120, 99, 100, 25);

contentPane.add(lblY);

clear.setEnabled(false);

clear.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e) {

m.getMap().clear();

}

});

clear.setBounds(10, 171, 100, 25);

contentPane.add(clear);

roadY = new JTextField();

roadY.setBounds(230, 99, 100, 25);

contentPane.add(roadY);

roadY.setColumns(10);

JLabel lblX = new JLabel("x");

lblX.setFont(new Font("Tahoma", Font.PLAIN, 10));

lblX.setHorizontalAlignment(SwingConstants.RIGHT);

lblX.setBounds(120, 63, 100, 25);

contentPane.add(lblX);

JLabel lblDirection = new JLabel("direction");

lblDirection.setFont(new Font("Tahoma", Font.PLAIN, 10));

lblDirection.setHorizontalAlignment(SwingConstants.RIGHT);

lblDirection.setBounds(120, 135, 100, 25);

contentPane.add(lblDirection);

JLabel label = new JLabel("x");

label.setFont(new Font("Tahoma", Font.PLAIN, 10));

label.setHorizontalAlignment(SwingConstants.RIGHT);

label.setBounds(340, 63, 50, 25);

contentPane.add(label);

JLabel lblY\_1 = new JLabel("y");

lblY\_1.setFont(new Font("Tahoma", Font.PLAIN, 10));

lblY\_1.setHorizontalAlignment(SwingConstants.RIGHT);

lblY\_1.setBounds(340, 99, 50, 25);

contentPane.add(lblY\_1);

spawnerX = new JTextField();

spawnerX.setColumns(10);

spawnerX.setBounds(570, 63, 100, 25);

contentPane.add(spawnerX);

spawnerY = new JTextField();

spawnerY.setColumns(10);

spawnerY.setBounds(570, 99, 100, 25);

contentPane.add(spawnerY);

JLabel label\_1 = new JLabel("x");

label\_1.setFont(new Font("Tahoma", Font.PLAIN, 10));

label\_1.setHorizontalAlignment(SwingConstants.RIGHT);

label\_1.setBounds(510, 63, 50, 25);

contentPane.add(label\_1);

JLabel label\_2 = new JLabel("y");

label\_2.setFont(new Font("Tahoma", Font.PLAIN, 10));

label\_2.setHorizontalAlignment(SwingConstants.RIGHT);

label\_2.setBounds(510, 99, 50, 25);

contentPane.add(label\_2);

spawnerRate = new JTextField();

spawnerRate.setColumns(10);

spawnerRate.setBounds(570, 135, 100, 25);

contentPane.add(spawnerRate);

JLabel lblRate = new JLabel("rate");

lblRate.setFont(new Font("Tahoma", Font.PLAIN, 10));

lblRate.setHorizontalAlignment(SwingConstants.RIGHT);

lblRate.setBounds(510, 135, 50, 25);

contentPane.add(lblRate);

trafficLightX = new JTextField();

trafficLightX.setColumns(10);

trafficLightX.setBounds(400, 315, 100, 25);

contentPane.add(trafficLightX);

trafficLightY = new JTextField();

trafficLightY.setColumns(10);

trafficLightY.setBounds(400, 351, 100, 25);

contentPane.add(trafficLightY);

JLabel label\_3 = new JLabel("x");

label\_3.setFont(new Font("Tahoma", Font.PLAIN, 10));

label\_3.setHorizontalAlignment(SwingConstants.RIGHT);

label\_3.setBounds(340, 315, 50, 25);

contentPane.add(label\_3);

JLabel label\_4 = new JLabel("y");

label\_4.setFont(new Font("Tahoma", Font.PLAIN, 10));

label\_4.setHorizontalAlignment(SwingConstants.RIGHT);

label\_4.setBounds(340, 351, 50, 25);

contentPane.add(label\_4);

trafficLightRate = new JTextField();

trafficLightRate.setColumns(10);

trafficLightRate.setBounds(400, 387, 100, 25);

contentPane.add(trafficLightRate);

JLabel lblRate\_1 = new JLabel("rate");

lblRate\_1.setFont(new Font("Tahoma", Font.PLAIN, 10));

lblRate\_1.setHorizontalAlignment(SwingConstants.RIGHT);

lblRate\_1.setBounds(340, 387, 50, 25);

contentPane.add(lblRate\_1);

JLabel lblStart = new JLabel("start");

lblStart.setFont(new Font("Tahoma", Font.PLAIN, 10));

lblStart.setHorizontalAlignment(SwingConstants.RIGHT);

lblStart.setBounds(340, 423, 50, 25);

contentPane.add(lblStart);

dropOffX = new JTextField();

dropOffX.setColumns(10);

dropOffX.setBounds(230, 315, 100, 25);

contentPane.add(dropOffX);

dropOffY = new JTextField();

dropOffY.setColumns(10);

dropOffY.setBounds(230, 351, 100, 25);

contentPane.add(dropOffY);

dropOffRate = new JTextField();

dropOffRate.setColumns(10);

dropOffRate.setBounds(230, 387, 100, 25);

contentPane.add(dropOffRate);

JLabel label\_5 = new JLabel("x");

label\_5.setFont(new Font("Tahoma", Font.PLAIN, 10));

label\_5.setHorizontalAlignment(SwingConstants.RIGHT);

label\_5.setBounds(120, 315, 100, 25);

contentPane.add(label\_5);

JLabel label\_6 = new JLabel("y");

label\_6.setFont(new Font("Tahoma", Font.PLAIN, 10));

label\_6.setHorizontalAlignment(SwingConstants.RIGHT);

label\_6.setBounds(134, 352, 86, 23);

contentPane.add(label\_6);

JButton btnMenu = new JButton("Menu");

btnMenu.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e) {

if(running) {

m.setVisible(false);

}

running = false;

dispose();

mm = new MainMenu();

mm.run();

}

});

btnMenu.setBounds(10, 495, 100, 25);

contentPane.add(btnMenu);

newButton.setEnabled(false);

newButton.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent arg0) {

try {

m.newMap(mapName, Integer.parseInt(width.getText()), Integer.parseInt(height.getText()));

} catch (NumberFormatException | FileNotFoundException | UnsupportedEncodingException e) {

// TODO Auto-generated catch block

e.printStackTrace();

}

}

});

newButton.setBounds(10, 315, 100, 25);

contentPane.add(newButton);

JLabel lblRate\_2 = new JLabel("rate");

lblRate\_2.setFont(new Font("Tahoma", Font.PLAIN, 10));

lblRate\_2.setHorizontalAlignment(SwingConstants.RIGHT);

lblRate\_2.setBounds(120, 387, 100, 25);

contentPane.add(lblRate\_2);

addDropOff.setEnabled(false);

addDropOff.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent arg0) {

m.getMap().addDropOff(Integer.parseInt(dropOffX.getText()), Integer.parseInt(dropOffY.getText()), Integer.parseInt(dropOffRate.getText()));

}

});

addDropOff.setBounds(230, 459, 100, 25);

contentPane.add(addDropOff);

width = new JTextField();

width.setColumns(10);

width.setBounds(70, 243, 40, 25);

contentPane.add(width);

JLabel label\_7 = new JLabel("Add Road");

label\_7.setFont(new Font("Tahoma", Font.PLAIN, 10));

label\_7.setHorizontalAlignment(SwingConstants.CENTER);

label\_7.setBounds(230, 27, 100, 25);

contentPane.add(label\_7);

height = new JTextField();

height.setColumns(10);

height.setBounds(70, 279, 40, 25);

contentPane.add(height);

JLabel lblWidth = new JLabel("width");

lblWidth.setHorizontalAlignment(SwingConstants.RIGHT);

lblWidth.setBounds(10, 243, 50, 25);

contentPane.add(lblWidth);

JLabel lblY\_2 = new JLabel("height");

lblY\_2.setHorizontalAlignment(SwingConstants.RIGHT);

lblY\_2.setBounds(10, 279, 50, 25);

contentPane.add(lblY\_2);

removeRoad.setEnabled(false);

removeRoad.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent arg0) {

m.getMap().removeRoad(Integer.parseInt(roadX.getText()), Integer.parseInt(roadY.getText()));

}

});

removeRoad.setBounds(230, 207, 100, 25);

contentPane.add(removeRoad);

removeWall.setEnabled(false);

removeWall.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e) {

m.getMap().removeWall(Integer.parseInt(wallX.getText()), Integer.parseInt(wallY.getText()));

}

});

removeWall.setBounds(400, 207, 100, 25);

contentPane.add(removeWall);

removeSpawner.setEnabled(false);

removeSpawner.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e) {

m.getMap().removeSpawner(Integer.parseInt(spawnerX.getText()), Integer.parseInt(spawnerY.getText()));

}

});

removeSpawner.setBounds(570, 208, 100, 25);

contentPane.add(removeSpawner);

removeDropOff.setEnabled(false);

removeDropOff.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e) {

m.getMap().removeDropOff(Integer.parseInt(dropOffX.getText()), Integer.parseInt(dropOffY.getText()));

}

});

removeDropOff.setBounds(230, 495, 100, 25);

contentPane.add(removeDropOff);

removeTrafficLight.setEnabled(false);

removeTrafficLight.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e) {

m.getMap().removeTrafficLight(Integer.parseInt(trafficLightX.getText()), Integer.parseInt(trafficLightY.getText()));

}

});

removeTrafficLight.setBounds(400, 495, 100, 25);

contentPane.add(removeTrafficLight);

JLabel lblAddRoadTile = new JLabel("Add Road Condition");

lblAddRoadTile.setFont(new Font("Tahoma", Font.PLAIN, 10));

lblAddRoadTile.setHorizontalAlignment(SwingConstants.CENTER);

lblAddRoadTile.setBounds(570, 279, 100, 25);

contentPane.add(lblAddRoadTile);

roadTileX = new JTextField();

roadTileX.setColumns(10);

roadTileX.setBounds(570, 315, 100, 25);

contentPane.add(roadTileX);

roadTileY = new JTextField();

roadTileY.setColumns(10);

roadTileY.setBounds(570, 351, 100, 25);

contentPane.add(roadTileY);

JLabel label\_9 = new JLabel("x");

label\_9.setFont(new Font("Tahoma", Font.PLAIN, 10));

label\_9.setHorizontalAlignment(SwingConstants.RIGHT);

label\_9.setBounds(510, 315, 50, 25);

contentPane.add(label\_9);

JLabel label\_10 = new JLabel("y");

label\_10.setFont(new Font("Tahoma", Font.PLAIN, 10));

label\_10.setHorizontalAlignment(SwingConstants.RIGHT);

label\_10.setBounds(510, 351, 50, 25);

contentPane.add(label\_10);

JLabel lblCondition = new JLabel("condition");

lblCondition.setFont(new Font("Tahoma", Font.PLAIN, 10));

lblCondition.setHorizontalAlignment(SwingConstants.RIGHT);

lblCondition.setBounds(510, 387, 50, 25);

contentPane.add(lblCondition);

JButton roadTileCondition = new JButton("Damaged");

roadTileCondition.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent arg0) {

if(roadTileCondition.getText() == "Damaged")

roadTileCondition.setText("Flooded");

else if(roadTileCondition.getText() == "Flooded")

roadTileCondition.setText("Closed");

else if(roadTileCondition.getText() == "Closed")

roadTileCondition.setText("Damaged");

}

});

roadTileCondition.setBounds(570, 387, 100, 25);

addRoadTile.setEnabled(false);

addRoadTile.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent arg0) {

if(roadTileCondition.getText() == "Damaged")

m.getMap().addRoadTile(Integer.parseInt(roadTileX.getText()), Integer.parseInt(roadTileY.getText()), 1);

else if(roadTileCondition.getText() == "Flooded")

m.getMap().addRoadTile(Integer.parseInt(roadTileX.getText()), Integer.parseInt(roadTileY.getText()), 2);

else if(roadTileCondition.getText() == "Closed")

m.getMap().addRoadTile(Integer.parseInt(roadTileX.getText()), Integer.parseInt(roadTileY.getText()), 3);

}

});

addRoadTile.setBounds(570, 459, 100, 25);

contentPane.add(addRoadTile);

removeRoadTile.setEnabled(false);

removeRoadTile.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e) {

m.getMap().removeRoadTile(Integer.parseInt(roadTileX.getText()), Integer.parseInt(roadTileY.getText()));

}

});

removeRoadTile.setBounds(570, 495, 100, 25);

contentPane.add(removeRoadTile);

contentPane.add(roadTileCondition);

deselect.setEnabled(false);

deselect.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent arg0) {

scan.setEnabled(false);

mapNameLabel.setText("no map selected");

mapName = mapNameLabel.getText();

message.setText("no file selected");

addRoad.setEnabled(false);

addWall.setEnabled(false);

addSpawner.setEnabled(false);

addDropOff.setEnabled(false);

addRoadTile.setEnabled(false);

addTrafficLight.setEnabled(false);

removeRoad.setEnabled(false);

removeWall.setEnabled(false);

removeSpawner.setEnabled(false);

removeDropOff.setEnabled(false);

removeRoadTile.setEnabled(false);

removeTrafficLight.setEnabled(false);

deselect.setEnabled(false);

display.setEnabled(false);

clear.setEnabled(false);

newButton.setEnabled(false);

if(running) {

m.setVisible(false);

}

running = false;

running = false;

}

});

deselect.setBounds(10, 136, 100, 25);

contentPane.add(deselect);

contentPane.setFocusTraversalPolicy(new FocusTraversalOnArray(new Component[]{message, mapNameLabel, addWall, addSpawner, lblAddDropOff, trafficLightStart, addRoad, addTrafficLight, lblAddSpawner, lblAddWall, lblAddTrafficLight, menuBar, mnTools, mntmSave, map1, map2, map3, scan, display, roadX, wallY, wallX, lblY, clear, roadY, lblX, lblDirection, label, lblY\_1, spawnerX, spawnerY, label\_1, label\_2, spawnerRate, lblRate, trafficLightX, trafficLightY, label\_3, label\_4, trafficLightRate, lblRate\_1, lblStart, dropOffX, dropOffY, dropOffRate, label\_5, label\_6, btnMenu, newButton}));

}

/\*\*

\* This method lets the user edit elements of the traffic simulation

\*/

public void run() {

EventQueue.invokeLater(new Runnable() {

public void run() {

try {

EditorMenu frame = new EditorMenu();

frame.setVisible(true);

} catch (Exception e) {

e.printStackTrace();

}

}

});

}

}

import java.awt.event.MouseEvent;

import java.awt.event.MouseListener;

import java.io.FileNotFoundException;

import java.io.IOException;

import java.io.UnsupportedEncodingException;

import java.net.URISyntaxException;

import java.util.ArrayList;

import javax.swing.JFrame;

import javax.swing.JOptionPane;

/\*\*

\* This class retrieves the settings from different components of the simulation.

\*/

public class EditorModel extends Thread{

private String mapName;

private int mouseX;

private int mouseY;

private int[][] map;

private int[][] objectMap;

private Block[][] block;

private Map m = new Map();

private JFrame f;

/\*\*

\* This is the constructor method of EditorModel.

\*/

public EditorModel() {

}

/\*\*

\* This method saves the alterations the user made to the map.

\*/

public void save() throws FileNotFoundException, UnsupportedEncodingException {

Printer p = new Printer(m);

p.setMapName(mapName);

p.print();

}

/\*\*

\* This method lets the user choose a map.

\*/

public void setMap() throws URISyntaxException {

int i, x, y, direction;

m.readMap();

int width, height;

width = m.getWidth();

height = m.getHeight();

ArrayList <Wall> wallList = new ArrayList <Wall> (m.getWallList());

ArrayList <Road> roadList = new ArrayList <Road> (m.getRoadList());

ArrayList <TrafficLight> trafficLightList = new ArrayList <TrafficLight> (m.getTrafficLightList());

map = new int[m.getWidth()][m.getHeight()];

objectMap = new int[m.getWidth()][m.getHeight()];

block = new Block[m.getWidth()][m.getHeight()];

//This clears the map.

for(x = 0; x < width; x++) {

for(y = 0; y < height; y++) {

map[x][y] = 0;

objectMap[x][y] = 0;

block[x][y] = new Block();

}

}

//This adds walls into the map.

for(i = 0; i < wallList.size(); i++) {

x = wallList.get(i).getX();

y = wallList.get(i).getY();

map[x][y] = 9;

objectMap[x][y] = 9;

}

//This adds roads into the map.

for(i = 0; i < roadList.size(); i++) {

x = roadList.get(i).getX();

y = roadList.get(i).getY();

direction = roadList.get(i).getDirection();

map[x][y] = direction;

}

//This adds traffic lights into the map.

for(i = 0; i < trafficLightList.size(); i++) {

x = trafficLightList.get(i).getX();

y = trafficLightList.get(i).getY();

block[x][y].setTrafficLight(trafficLightList.get(i));;

}

}

/\*\*

\* This method sets the area of the display.

\* @param width1 width of the simulation

\* @param height1 height of the simulation

\*/

public void setDisplay(int width1, int height1) {

f = new JFrame("Map Editor");

f.setSize(width1, height1);

f.setLocationRelativeTo(null);

f.getContentPane().add(new Display(m));

f.getContentPane().addMouseListener(new MouseListener() {

@Override

public void mouseClicked(MouseEvent e) {

mouseX = e.getX();

mouseY = e.getY();

int xWidth = (int) f.getWidth() / m.getWidth();

int yHeight = (int) f.getHeight() / m.getHeight();

mouseX = mouseX / xWidth;

mouseY = mouseY / yHeight;

System.out.println(mouseX + "," + mouseY);//these co-ords are relative to the component

}

@Override

public void mouseEntered(MouseEvent arg0) {

// TODO Auto-generated method stub

}

@Override

public void mouseExited(MouseEvent arg0) {

// TODO Auto-generated method stub

}

@Override

public void mousePressed(MouseEvent arg0) {

// TODO Auto-generated method stub

}

@Override

public void mouseReleased(MouseEvent arg0) {

// TODO Auto-generated method stub

}

});

f.setVisible(true);

}

/\*\*

\* This method sets the visibility of the Map.

\* @param visibility

\*/

public void setVisible(boolean visibility) {

f.setVisible(visibility);

}

public void toggleVisibility() {

if(f.isVisible())

f.setVisible(false);

else

f.setVisible(true);

}

/\*\*

\* This method displays the simulation

\*/

public void displayMap() {

Thread t = new Thread() {

public void run() {

setDisplay(500, 500);

}

};

t.start();

}

/\*\*

\* This method creates a new map.

\* @param mapName

\* @param width

\* @param height

\* @throws FileNotFoundException

\* @throws UnsupportedEncodingException

\*/

public void newMap(String mapName, int width, int height) throws FileNotFoundException, UnsupportedEncodingException {

m.setMapName(mapName);

m.newMap(width, height);

save();

}

/\*\*

\* This method sets a map name

\* @param mapName name of the map

\*/

public void setMapName(String mapName) {

m.setMapName(mapName);

}

/\*\*

\* This method checks if the map is empty or not.

\* @param mapName name of the map

\* @return boolean of the state of the text file

\* @throws FileNotFoundException

\*/

public boolean isEmpty(String mapName) throws FileNotFoundException {

Reader r = new Reader();

return r.isEmpty(mapName);

}

/\*\*

\* This method gets the assigned Map.

\* @return Map

\*/

public Map getMap() {

return m;

}

/\*\*

\* This method returns the map

\* @return structure of the map

\*/

public int[][] getMap1() {

return map;

}

/\*\*

\* This method returns the x value of the mouse pointer in relative to the frame and the map.

\* @return x value of the mouse

\*/

public int getMouseX() {

return mouseX;

}

/\*\*

\* This method returns the y value of the mouse pointer in relative to the frame and the map.

\* @return y value of the mouse

\*/

public int getMouseY() {

return mouseY;

}

/\*\*

\* This method returns all the coordinates of the walls.

\* @return the coordinates of the walls

\*/

public ArrayList <Wall> getWallList() {

return m.getWallList();

}

/\*\*

\* This method returns all the coordinate of the spawners

\* @return the coordinates of the spawners

\*/

public ArrayList <Spawner> getSpawnerList() {

return m.getSpawnerList();

}

/\*\*

\* This method returns the coordinates of all the roads

\* @return the coordinates of the roads

\*/

public ArrayList <Road> getRoadList() {

return m.getRoadList();

}

/\*\*

\* This method returns the coordinates of all the Drop Off blocks

\* @return the coordinates of the Drop Off blocks

\*/

public ArrayList <DropOff> getDropOffList() {

return m.getDropOffList();

}

/\*\*

\* This method returns the coordinates of all the Traffic Light blocks

\* @return the coordinates of the Traffic Light blocks

\*/

public ArrayList <TrafficLight> getTrafficLightList() {

return m.getTrafficLightList();

}

public ArrayList <RoadTile> getRoadTileList() {

return m.getRoadTileList();

}

}

import java.awt.BorderLayout;

import java.awt.EventQueue;

import javax.swing.JFrame;

import javax.swing.JPanel;

import javax.swing.border.EmptyBorder;

import javax.swing.JLabel;

import javax.swing.SwingConstants;

import java.awt.Font;

import javax.swing.JButton;

import java.awt.event.ActionListener;

import java.awt.event.ActionEvent;

/\*\*

\* This class displays the Main Menu where the user can choose to do two things which is run a map or alter and create a new map.

\*/

public class MainMenu extends JFrame {

private JPanel contentPane;

private SimulationMenu m1 = new SimulationMenu();

private EditorMenu m2 = new EditorMenu();

/\*\*

\* Constructor method for Main Menu class

\*/

public MainMenu() {

setTitle("Traffic Flow Simulator");

setResizable(false);

setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

setBounds(100, 100, 450, 300);

contentPane = new JPanel();

contentPane.setBorder(new EmptyBorder(5, 5, 5, 5));

setContentPane(contentPane);

contentPane.setLayout(null);

setLocationRelativeTo(null);

JLabel lblTrafficFlowSimulator = new JLabel("Traffic Flow Simulator");

lblTrafficFlowSimulator.setFont(new Font("Tahoma", Font.PLAIN, 30));

lblTrafficFlowSimulator.setBounds(5, 5, 429, 50);

lblTrafficFlowSimulator.setHorizontalAlignment(SwingConstants.CENTER);

contentPane.add(lblTrafficFlowSimulator);

JLabel lblNewLabel = new JLabel("Cyrus A. Vatandoost Kakhki");

lblNewLabel.setHorizontalAlignment(SwingConstants.CENTER);

lblNewLabel.setBounds(5, 211, 429, 14);

contentPane.add(lblNewLabel);

JLabel lblJuliusCeasarLibrada = new JLabel("Julius Ceasar Librada");

lblJuliusCeasarLibrada.setHorizontalAlignment(SwingConstants.CENTER);

lblJuliusCeasarLibrada.setBounds(5, 236, 429, 14);

contentPane.add(lblJuliusCeasarLibrada);

JButton btnNewButton = new JButton("Simulator");

btnNewButton.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e) {

setVisible(false);

m1.run();

}

});

btnNewButton.setBounds(5, 66, 204, 134);

contentPane.add(btnNewButton);

JButton btnRunLayoutManager = new JButton("Layout Manager");

btnRunLayoutManager.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e) {

setVisible(false);

m2.run();

}

});

btnRunLayoutManager.setBounds(224, 66, 210, 134);

contentPane.add(btnRunLayoutManager);

}

/\*\*

\* This method runs the simulation based on the input of the user

\*/

public void run() {

EventQueue.invokeLater(new Runnable() {

public void run() {

try {

MainMenu frame = new MainMenu();

frame.setVisible(true);

} catch (Exception e) {

e.printStackTrace();

}

}

});

}

}

import java.io.IOException;

import java.net.URISyntaxException;

import java.util.ArrayList;

import javax.swing.JOptionPane;

/\*\*

\* This class retrieves the settings and components of a map.

\*/

public class Map {

private String mapName;

private boolean hasMap;

private int width;

private int height;

private int[][] numMap;

private int[][] objectMap;

private Block[][] blockMap;

private ArrayList <Car> carList = new ArrayList <Car> ();

private ArrayList <Bus> busList = new ArrayList <Bus> ();

private ArrayList <Wall> wallList = new ArrayList <Wall> ();

private ArrayList <Road> roadList = new ArrayList <Road> ();

private ArrayList <Spawner> spawnerList = new ArrayList <Spawner> ();

private ArrayList <DropOff> dropOffList = new ArrayList <DropOff> ();

private ArrayList <TrafficLight> trafficLightList = new ArrayList <TrafficLight> ();

private ArrayList <RoadTile> roadTileList = new ArrayList <RoadTile> ();

/\*\*

\* Constructor method for Map class.

\*/

public Map() {

mapName = null;

hasMap = false;

}

/\*\*

\* This method reads a map.

\* @throws URISyntaxException

\*/

public void readMap() throws URISyntaxException {

int i, x, y, direction;

if(mapName != null) {

Reader r = new Reader();

try {

r.read(mapName);

} catch (IOException e) {

e.printStackTrace();

}

width = r.getWidth();

height = r.getHeight();

//This gets the values of the different entities from the text file.

wallList = new ArrayList <Wall> (r.getWallList());

spawnerList = new ArrayList <Spawner> (r.getSpawnerList());

roadList = new ArrayList <Road> (r.getRoadList());

dropOffList = new ArrayList <DropOff> (r.getdropOffList());

trafficLightList = new ArrayList <TrafficLight> (r.getTrafficLightList());

roadTileList = new ArrayList <RoadTile> (r.getRoadTileList());

numMap = new int[width][height];

objectMap = new int[width][height];

blockMap = new Block[width][height];

hasMap = true;

//This clears the map.

for(x = 0; x < width; x++) {

for(y = 0; y < height; y++) {

numMap[x][y] = 0;

objectMap[x][y] = 0;

blockMap[x][y] = new Block();

}

}

//This adds walls into the map.

for(i = 0; i < wallList.size(); i++) {

x = wallList.get(i).getX();

y = wallList.get(i).getY();

numMap[x][y] = 9;

objectMap[x][y] = 9;

}

//This adds roads into the map.

for(i = 0; i < roadList.size(); i++) {

x = roadList.get(i).getX();

y = roadList.get(i).getY();

direction = roadList.get(i).getDirection();

numMap[x][y] = direction;

}

//This adds traffic lights into the map.

for(i = 0; i < trafficLightList.size(); i++) {

x = trafficLightList.get(i).getX();

y = trafficLightList.get(i).getY();

blockMap[x][y].setTrafficLight(trafficLightList.get(i));;

}

for(i = 0; i < dropOffList.size(); i++) {

x = dropOffList.get(i).getX();

y = dropOffList.get(i).getY();

blockMap[x][y].setDropOff(dropOffList.get(i));

}

for(i = 0; i < roadTileList.size(); i++) {

x = roadTileList.get(i).getX();

y = roadTileList.get(i).getY();

blockMap[x][y].setRoadTile(roadTileList.get(i));

}

}

else {

JOptionPane frame = new JOptionPane();

JOptionPane.showMessageDialog(frame, "Error! No assigned map to be read.");

}

}

/\*\*

\* This method deletes all the elements of the map.

\*/

public void delete() {

width = 0;

height = 0;

carList.clear();

busList.clear();

wallList.clear();

roadList.clear();

spawnerList.clear();

dropOffList.clear();

trafficLightList.clear();

roadTileList.clear();

hasMap = false;

}

/\*\*

\* This method clears the map.

\*/

public void clear() {

int x, y;

wallList.clear();

for(y = 0; y < height; y++) {

for(x = 0; x < width; x++) {

addWall(x, y);

}

}

carList.clear();

busList.clear();

roadList.clear();

spawnerList.clear();

dropOffList.clear();

trafficLightList.clear();

roadTileList.clear();

}

/\*\*

\* This method converts the elements of the map into integers.

\*/

public void convertToNumMap() {

int i, x, y;

for(y = 0; y < height; y++) {

for(x = 0; x < width; x++) {

numMap[x][y] = 9;

}

}

for(i = 0; i < roadList.size(); i++) {

x = roadList.get(i).getX();

y = roadList.get(i).getY();

numMap[x][y] = roadList.get(i).getDirection();

}

}

/\*\*

\* This method retrieves elements using the integers stored in the map file.

\*/

public void convertFromNumMap() {

int x, y;

for(x = 0; x < width; x++) {

for(y = 0; y < height; y++) {

if(numMap[x][y] == 9)

addWall(x, y);

else {

addRoad(x, y, numMap[x][y]);

}

}

}

}

/\*\*

\* This method opens a new map.

\* @param width width of the map

\* @param height height of the map

\*/

public void newMap(int width, int height) {

carList = new ArrayList <Car> ();

busList = new ArrayList <Bus> ();

wallList = new ArrayList <Wall> ();

roadList = new ArrayList <Road> ();

spawnerList = new ArrayList <Spawner> ();

dropOffList = new ArrayList <DropOff> ();

trafficLightList = new ArrayList <TrafficLight> ();

roadTileList = new ArrayList <RoadTile> ();

this.width = width;

this.height = height;

numMap = new int [width][height];

hasMap = true;

convertFromNumMap();

clear();

}

/\*\*

\* This method adds a new car.

\* @param x x coordinate of the car

\* @param y y coordinate of the car

\*/

public void addCar(int x, int y) {

Car c = new Car(x, y);

carList.add(c);

c = null;

}

/\*\*

\* This method adds a new bus.

\* @param x x coordinate of the bus

\* @param y y coordinate of the bus

\*/

public void addBus(int x, int y) {

Bus b = new Bus(x, y);

busList.add(b);

b = null;

}

/\*\*

\* This method adds a new wall.

\* @param x x coordinate of the wall

\* @param y y coordinate of the wall

\*/

public void addWall(int x, int y) {

int i, x2, y2;

if(x >= 0 && x < width && y >= 0 && y < height) {

for(i = 0; i < wallList.size(); i++) {

x2 = wallList.get(i).getX();

y2 = wallList.get(i).getY();

if(x == x2 && y == y2)

wallList.remove(i);

}

Wall w = new Wall(x, y);

for(i = 0; i < roadList.size(); i++) {

x2 = roadList.get(i).getX();

y2 = roadList.get(i).getY();

if(x == x2 && y == y2)

roadList.remove(i);

}

wallList.add(w);

w = null;

}

else {

JOptionPane frame = new JOptionPane();

JOptionPane.showMessageDialog(frame, "Error! Invalid values dectected.");

frame = null;

}

}

/\*\*

\* This method adds a new road.

\* @param x x coordinate of the road

\* @param y y coordinate of the road

\* @param direction direction pointed to by the road

\*/

public void addRoad(int x, int y, int direction) {

int i, x1, y1, x2, y2;

if(x >= 0 && x < width && y >= 0 && y < height) {

for(i = 0; i < roadList.size(); i++) {

x2 = roadList.get(i).getX();

y2 = roadList.get(i).getY();

if(x == x2 && y == y2)

roadList.remove(i);

}

Road r = new Road(x, y, direction);

x1 = r.getX();

y1 = r.getY();

for(i = 0; i < wallList.size(); i++) {

x2 = wallList.get(i).getX();

y2 = wallList.get(i).getY();

if(x1 == x2 && y1 == y2)

wallList.remove(i);

}

roadList.add(r);

r = null;

}

else {

JOptionPane frame = new JOptionPane();

JOptionPane.showMessageDialog(frame, "Error! Invalid values dectected.");

frame = null;

}

}

/\*\*

\* This method adds a new spawner.

\* @param x x coordinate of the spawner

\* @param y y coordinate of the spawner

\* @param rate rate at which vehicles spawn

\*/

public void addSpawner(int x, int y, int rate) {

int i, x1, y1;

if(x >= 0 && x < width && y >= 0 && y < height) {

for(i = 0; i < spawnerList.size(); i++) {

x1 = spawnerList.get(i).getX();

y1 = spawnerList.get(i).getY();

if(x == x1 && y == y1)

spawnerList.remove(i);

}

Spawner s = new Spawner(x, y, rate);

spawnerList.add(s);

s = null;

}

else {

JOptionPane frame = new JOptionPane();

JOptionPane.showMessageDialog(frame, "Error! Invalid values dectected.");

frame = null;

}

}

/\*\*

\* This method adds a new traffic light.

\* @param x x coordinate of the traffic light

\* @param y y coordinate of the traffic light

\* @param rate rate at which the light changes

\* @param start start time of the simulation

\*/

public void addTrafficLight(int x, int y, int rate, int start) {

int i, x1, y1;

if(x >= 0 && x < width && y >= 0 && y < height) {

for(i = 0; i < trafficLightList.size(); i++) {

x1 = trafficLightList.get(i).getX();

y1 = trafficLightList.get(i).getY();

if(x == x1 && y == y1)

trafficLightList.remove(i);

}

TrafficLight t = new TrafficLight(x, y, rate, start);

trafficLightList.add(t);

t = null;

}

else {

JOptionPane frame = new JOptionPane();

JOptionPane.showMessageDialog(frame, "Error! Invalid values dectected.");

frame = null;

}

}

/\*\*

\* This method adds a new drop off point.

\* @param x x coordinate of the drop off point

\* @param y y coordinate of the drop off point

\* @param rate rate at which the cars stay at the drop off point

\*/

public void addDropOff(int x, int y, int rate) {

int i, x1, y1;

if(x >= 0 && x < width && y >= 0 && y < height) {

for(i = 0; i < dropOffList.size(); i++) {

x1 = dropOffList.get(i).getX();

y1 = dropOffList.get(i).getY();

if(x == x1 && y == y1)

dropOffList.remove(i);

}

DropOff d = new DropOff(x, y, rate);

dropOffList.add(d);

d = null;

}

else {

JOptionPane frame = new JOptionPane();

JOptionPane.showMessageDialog(frame, "Error! Invalid values dectected.");

}

}

/\*\*

\* This adds a new road tile.

\* @param x x coordinate of the road tile

\* @param y y coordinate of the road tile

\* @param condition condition of the road tile (e.g. Broken, Flooded, Blocked)

\*/

public void addRoadTile(int x, int y, int condition) {

int i, x1, y1;

if(x >= 0 && x < width && y >= 0 && y < height) {

for(i = 0; i < roadTileList.size(); i++) {

x1 = roadTileList.get(i).getX();

y1 = roadTileList.get(i).getY();

if(x == x1 && y == y1)

roadTileList.remove(i);

}

RoadTile rt = new RoadTile(x, y, condition);

roadTileList.add(rt);

rt = null;

}

else {

JOptionPane frame = new JOptionPane();

JOptionPane.showMessageDialog(frame, "Error! Invalid values dectected.");

}

}

/\*\*

\* This method removes a car.

\* @param x x coordinate of the car

\* @param y y coordinate of the car

\*/

public void removeCar(int x, int y) {

int i, x1, y1;

for(i = 0; i < carList.size(); i++) {

x1 = carList.get(i).getX();

y1 = carList.get(i).getY();

if(x == x1 && y == y1)

carList.remove(i);

}

}

/\*\*

\* This method removes a car from the car list

\* @param i position of the car

\*/

public void removeCar(int i) {

carList.remove(i);

}

/\*\*

\* This method removes a bus.

\* @param x x coordinate of the bus.

\* @param y y coordinate of the bus.

\*/

public void removeBus(int x, int y) {

int i, x1, y1;

for(i = 0; i < busList.size(); i++) {

x1 = busList.get(i).getX();

y1 = busList.get(i).getY();

if(x == x1 && y == y1)

busList.remove(i);

}

}

/\*\*

\* This method removes a bus from the bus list.

\* @param i position of the bus

\*/

public void removeBus(int i) {

busList.remove(i);

}

/\*\*

\* This method removes a road.

\* @param x x coordinate of the road

\* @param y y coordinate of the road

\*/

public void removeRoad(int x, int y) {

int i, x1, y1;

if(x >= 0 && x < width && y >= 0 && y < height) {

for(i = 0; i < roadList.size(); i++) {

x1 = roadList.get(i).getX();

y1 = roadList.get(i).getY();

if(x == x1 && y == y1)

roadList.remove(i);

}

}

else {

JOptionPane frame = new JOptionPane();

JOptionPane.showMessageDialog(frame, "Error! Invalid values dectected.");

frame = null;

}

}

/\*\*

\* This method removes a wall.

\* @param x x coordinate of the wall

\* @param y y coordinate of the wall

\*/

public void removeWall(int x, int y) {

int i, x1, y1;

if(x >= 0 && x < width && y >= 0 && y < height) {

for(i = 0; i < wallList.size(); i++) {

x1 = wallList.get(i).getX();

y1 = wallList.get(i).getY();

if(x == x1 && y == y1)

wallList.remove(i);

}

}

else {

JOptionPane frame = new JOptionPane();

JOptionPane.showMessageDialog(frame, "Error! Invalid values dectected.");

frame = null;

}

}

/\*\*

\* This removes a spawner.

\* @param x x coordinate of the spawner

\* @param y y coordinate of the spawner

\*/

public void removeSpawner(int x, int y) {

int i, x1, y1;

if(x >= 0 && x < width && y >= 0 && y < height) {

for(i = 0; i < spawnerList.size(); i++) {

x1 = spawnerList.get(i).getX();

y1 = spawnerList.get(i).getY();

if(x == x1 && y == y1)

spawnerList.remove(i);

}

}

else {

JOptionPane frame = new JOptionPane();

JOptionPane.showMessageDialog(frame, "Error! Invalid values dectected.");

frame = null;

}

}

/\*\*

\* This method removes a drop off point.

\* @param x x coordinate of the drop off point

\* @param y y coordinate of the drop off point

\*/

public void removeDropOff(int x, int y) {

int i, x1, y1;

if(x >= 0 && x < width && y >= 0 && y < height) {

for(i = 0; i < dropOffList.size(); i++) {

x1 = dropOffList.get(i).getX();

y1 = dropOffList.get(i).getY();

if(x == x1 && y == y1)

dropOffList.remove(i);

}

}

else {

JOptionPane frame = new JOptionPane();

JOptionPane.showMessageDialog(frame, "Error! Invalid values dectected.");

frame = null;

}

}

/\*\*

\* This method removes a traffic light.

\* @param x x coordinate of the traffic light

\* @param y y coordinate of the traffic light

\*/

public void removeTrafficLight(int x, int y) {

int i, x1, y1;

if(x >= 0 && x < width && y >= 0 && y < height) {

for(i = 0; i < trafficLightList.size(); i++) {

x1 = trafficLightList.get(i).getX();

y1 = trafficLightList.get(i).getY();

if(x == x1 && y == y1)

trafficLightList.remove(i);

}

}

else {

JOptionPane frame = new JOptionPane();

JOptionPane.showMessageDialog(frame, "Error! Invalid values dectected.");

frame = null;

}

}

/\*\*

\* This method removes a road tile.

\* @param x x coordinate of the road tile

\* @param y y coordinate of the road tile

\*/

public void removeRoadTile(int x, int y) {

int i, x1, y1;

if(x >= 0 && x < width && y >= 0 && y < height) {

for(i = 0; i < roadTileList.size(); i++) {

x1 = roadTileList.get(i).getX();

y1 = roadTileList.get(i).getY();

if(x == x1 && y == y1)

roadTileList.remove(i);

}

}

else {

JOptionPane frame = new JOptionPane();

JOptionPane.showMessageDialog(frame, "Error! Invalid values dectected.");

frame = null;

}

}

/\*\*

\* This method sets a map name.

\* @param mapName name of the map

\*/

public void setMapName(String mapName) {

this.mapName = mapName;

}

/\*\*

\* This method sets a width of the map.

\* @param width width of the map

\*/

public void setWidth(int width) {

this.width = width;

}

/\*\*

\* This method sets a height of the map.

\* @param height height of the map

\*/

public void setHeight(int height) {

this.height = height;

}

/\*\*

\* This method sets an integer based map.

\* @param numMap integer based map

\*/

public void setNumMap(int[][] numMap) {

this.numMap = numMap;

}

/\*\*

\* This method sets the objects of the map.

\* @param objectMap objects of the map

\*/

public void setObjectMap(int[][] objectMap) {

this.objectMap = objectMap;

}

/\*\*

\* This method sets the blocks of the map.

\* @param blockMap blocks of the map

\*/

public void setBlockMap(Block[][] blockMap) {

this.blockMap = blockMap;

}

/\*\*

\* This method sets the list of cars.

\* @param carList list of cars

\*/

public void setCarList(ArrayList <Car> carList) {

this.carList = carList;

}

/\*\*

\* This method sets the list of buses.

\* @param busList list of buses

\*/

public void setBusList(ArrayList <Bus> busList) {

this.busList = busList;

}

/\*\*

\* This method sets the list of roads.

\* @param roadList list of roads

\*/

public void setRoadList(ArrayList <Road> roadList) {

this.roadList = roadList;

}

/\*\*

\* This method sets the list of walls.

\* @param wallList list of walls

\*/

public void setWallList(ArrayList <Wall> wallList) {

this.wallList = wallList;

}

/\*\*

\* This method sets the list of traffic lights.

\* @param trafficLightList list of traffic lights

\*/

public void setTrafficList(ArrayList <TrafficLight> trafficLightList) {

this.trafficLightList = trafficLightList;

}

/\*\*

\* This method sets the list of drop off points.

\* @param dropOffList list of drop off points

\*/

public void setDropOffList(ArrayList <DropOff> dropOffList) {

this.dropOffList = dropOffList;

}

/\*\*

\* This method sets the list of road tiles.

\* @param roadTileList list of road tiles

\*/

public void setRoadTiles(ArrayList <RoadTile> roadTileList) {

this.roadTileList = roadTileList;

}

/\*\*

\* This method sets the list of spawners.

\* @param spawnerList list of spawners

\*/

public void setSpawnerList(ArrayList <Spawner> spawnerList) {

this.spawnerList = spawnerList;

}

/\*\*

\* This method returns the current map.

\* @return current map

\*/

public boolean hasMap() {

return hasMap;

}

/\*\*

\* This method returns the map name.

\* @return map name

\*/

public String getMapName() {

return mapName;

}

/\*\*

\* This method returns the integer based map.

\* @return integer based map

\*/

public int[][] getNumMap() {

convertToNumMap();

return numMap;

}

/\*\*

\* This method returns the objects of the map.

\* @return objects of the map

\*/

public int[][] getObjectMap() {

return objectMap;

}

/\*\*

\* This method returns the blocks of the map.

\* @return blocks of the map

\*/

public Block[][] getBlockMap() {

return blockMap;

}

/\*\*

\* This method returns the width of the map.

\* @return width of the map

\*/

public int getWidth() {

return width;

}

/\*\*

\* This method returns the height of the map.

\* @return height of the map

\*/

public int getHeight() {

return height;

}

/\*\*

\* This method returns the list of cars.

\* @return list of cars

\*/

public ArrayList <Car> getCarList() {

return carList;

}

/\*\*

\* This method returns the list of buses.

\* @return list of buses

\*/

public ArrayList <Bus> getBusList() {

return busList;

}

/\*\*

\* This method returns the list of walls.

\* @return list of walls

\*/

public ArrayList <Wall> getWallList() {

return wallList;

}

/\*\*

\* This method returns the list of roads

\* @return list of roads

\*/

public ArrayList <Road> getRoadList() {

return roadList;

}

/\*\*

\* This method returns the list of spawners.

\* @return list of spawners

\*/

public ArrayList <Spawner> getSpawnerList() {

return spawnerList;

}

/\*\*

\* This method returns the list of drop off points.

\* @return list of drop off points

\*/

public ArrayList <DropOff> getDropOffList() {

return dropOffList;

}

/\*\*

\* This method returns the list of traffic lights.

\* @return list of traffic lights

\*/

public ArrayList <TrafficLight> getTrafficLightList() {

return trafficLightList;

}

/\*\*

\* This method returns the list of road tiles.

\* @return list of road tiles

\*/

public ArrayList <RoadTile> getRoadTileList() {

return roadTileList;

}

}

import java.io.File;

import java.io.FileNotFoundException;

import java.io.IOException;

import java.io.PrintWriter;

import java.io.RandomAccessFile;

import java.io.UnsupportedEncodingException;

import java.util.ArrayList;

/\*\*

\* This class displays("prints") the objects of the map.

\*/

public class Printer {

private int width;

private int height;

private int[][] map;

private ArrayList <Wall> wallList;

private ArrayList <Spawner> spawnerList;

private ArrayList <Road> roadList;

private ArrayList <DropOff> dropOffList;

private ArrayList <TrafficLight> trafficLightList;

private ArrayList <RoadTile> roadTileList;

private String mapName;

private Map m;

/\*\*

\* Constructor method for Printer class

\* @param m settings from EditorModel class

\* @throws FileNotFoundException

\* @throws UnsupportedEncodingException

\*/

public Printer(Map m) throws FileNotFoundException, UnsupportedEncodingException {

this.m = m;

}

/\*\*

\* This method prints the list of elements of the simulation

\* @throws FileNotFoundException

\* @throws UnsupportedEncodingException

\*/

public void print() throws FileNotFoundException, UnsupportedEncodingException {

int i, x, y;

copy();

PrintWriter p = new PrintWriter(mapName, "UTF-8");

p.println(width);

p.println(height);

for(y = 0; y < height; y++) {

for(x = 0; x < width; x++) {

p.print(map[x][y]);

}

p.println();

}

for(i = 0; i < trafficLightList.size(); i++) {

p.println("6");

p.println(trafficLightList.get(i).getX());

p.println(trafficLightList.get(i).getY());

p.println(trafficLightList.get(i).getRate());

p.println(trafficLightList.get(i).getStart());

}

for(i = 0; i < dropOffList.size(); i++) {

p.println("7");

p.println(dropOffList.get(i).getX());

p.println(dropOffList.get(i).getY());

p.println(dropOffList.get(i).getLimit());

}

for(i = 0; i < spawnerList.size(); i++) {

p.println("8");

p.println(spawnerList.get(i).getX());

p.println(spawnerList.get(i).getY());

p.println(spawnerList.get(i).getRate());

}

for(i = 0; i < roadTileList.size(); i++) {

p.println("10");

p.println(roadTileList.get(i).getX());

p.println(roadTileList.get(i).getY());

p.println(roadTileList.get(i).getCondition());

}

p.println(-1);

p.close();

}

/\*\*

\* This method lets the user duplicate a map

\*/

public void copy() {

width = m.getWidth();

height = m.getHeight();

mapName = m.getMapName();

map = m.getNumMap();

spawnerList = new ArrayList <Spawner> (m.getSpawnerList());

dropOffList = new ArrayList <DropOff> (m.getDropOffList());

trafficLightList = new ArrayList <TrafficLight> (m.getTrafficLightList());

roadTileList = new ArrayList <RoadTile> (m.getRoadTileList());

}

/\*\*

\* This method sets a map name

\* @param mapName name of the map

\*/

public void setMapName(String mapName) {

this.mapName = mapName;

}

}

import java.io.BufferedReader;

import java.io.File;

import java.io.FileNotFoundException;

import java.io.FileReader;

import java.io.IOException;

import java.net.URISyntaxException;

import java.net.URL;

import java.util.ArrayList;

import java.util.Scanner;

import org.omg.CORBA.portable.InputStream;

/\*\*

\* This method reads the integer based map from a file.

\*/

public class Reader {

private int width;

private int height;

private ArrayList <Road> roadList = new ArrayList <Road> ();

private ArrayList <TrafficLight> trafficLightList = new ArrayList <TrafficLight> ();

private ArrayList <Spawner> spawnerList = new ArrayList <Spawner> ();

private ArrayList <DropOff> dropOffList = new ArrayList <DropOff> ();

private ArrayList <Wall> wallList = new ArrayList <Wall> ();

private ArrayList <RoadTile> roadTileList = new ArrayList <RoadTile> ();

/\*\*

\* Constructor method for Reader class.

\*/

public Reader() {

}

/\*\*

\* This method reads a file

\* @param fileName name of the file

\* @throws IOException

\*/

public void read(String fileName) throws IOException, URISyntaxException {

int y;

int num;

Road r;

Spawner p;

DropOff d;

TrafficLight t;

Wall w;

RoadTile rt;

File f = new File(fileName);

try {

Scanner s = new Scanner(f);

width = s.nextInt(); //This scans the width.

height = s.nextInt(); //This scans the height.

for(y = 0; y <= height; y++) {

if(s.hasNextLine()) {

String line = s.nextLine();

for(int i = 0; i < line.length(); i++) {

num = Character.getNumericValue(line.charAt(i));

//This adds a new Road.

if(num >= 0 && num <= 4) {

r = new Road(i, y - 1, num);

roadList.add(r);

r = null;

}

//This adds a new Wall.

else if(num == 9) {

w = new Wall(i, y - 1);

wallList.add(w);

w = null;

}

}

}

}

while(s.hasNextLine()) {

num = s.nextInt();

if(num == -1) {

break;

}

//If the object type is a TrafficLight, scan this.

if(num == 6) {

t = new TrafficLight(s.nextInt(), s.nextInt(), s.nextInt(), s.nextInt());

trafficLightList.add(t);

t = null;

}

//If the object type is a DropOff, scan this.

else if(num == 7) {

d = new DropOff(s.nextInt(), s.nextInt(), s.nextInt());

dropOffList.add(d);

d = null;

}

//If the object type is a Spawner, scan this.

else if(num == 8) {

p = new Spawner(s.nextInt(), s.nextInt(), s.nextInt());

spawnerList.add(p);

p = null;

}

//If the object type is a RoadTile, scan this.

else if(num == 10) {

rt = new RoadTile(s.nextInt(), s.nextInt(), s.nextInt());

roadTileList.add(rt);

rt = null;

}

}

s.close();

} catch (FileNotFoundException e) {

e.printStackTrace();

}

}

/\*\*

\* This method checks if the file contains anything.

\* @param fileName name of the file

\* @return the current condition of the file

\* @throws FileNotFoundException

\*/

public boolean isEmpty(String fileName) throws FileNotFoundException {

File f = new File(fileName);

Scanner s = new Scanner(f);

if(s.hasNext())

return false;

else

return true;

}

/\*\*

\* This method returns the width of the map

\* @return width of the map

\*/

public int getWidth() {

return width;

}

/\*\*

\* This method returns the height of the map

\* @return height of the map

\*/

public int getHeight() {

return height;

}

/\*\*

\* This method returns all the coordinates of the Road Blocks

\* @return the coordinates of the Road Blocks

\*/

public ArrayList <Road> getRoadList() {

return roadList;

}

/\*\*

\* This method returns all the coordinates of the Traffic Light blocks

\* @return the coordinates of the traffic light blocks

\*/

public ArrayList <TrafficLight> getTrafficLightList() {

return trafficLightList;

}

/\*\*

\* This method returns all the coordinates of the Spawner blocks

\* @return the coordinates of the spawner blocks

\*/

public ArrayList <Spawner> getSpawnerList() {

return spawnerList;

}

/\*\*

\* This method returns all the coordinates of the Drop Off blocks

\* @return the coordinates of the Drop Off blocks

\*/

public ArrayList <DropOff> getdropOffList() {

return dropOffList;

}

/\*\*

\* This method returns all the coordinates of the Walls

\* @return the coordinates of the walls

\*/

public ArrayList <Wall> getWallList() {

return wallList;

}

public ArrayList <RoadTile> getRoadTileList() {

return roadTileList;

}

}

/\*\*

\* This class creates a new road.

\*/

**public** **class** **Road** {

**private** **int** x;

**private** **int** y;

**private** **int** direction;

/\*\*

\* Constructor method for Road class

\* **@param** x - x coordinate of the road

\* **@param** y - y coordinate of the road

\* **@param** direction - direction the road goes

\*/

**public** **Road**(**int** x, **int** y, **int** direction) {

**this**.x = x;

**this**.y = y;

**this**.direction = direction;

}

/\*\*

\* This method returns the x coordinate of the road

\* **@return** x coordinate of the road

\*/

**public** **int** **getX**() {

**return** x;

}

/\*\*

\* This method returns the y coordinate of the road

\* **@return** y coordinate of the road

\*/

**public** **int** **getY**() {

**return** y;

}

/\*\*

\* This method returns the direction of the road

\* **@return** direction of the road

\*/

**public** **int** **getDirection**() {

**return** direction;

}

}

/\*\*

\* This method sets the condition of a Road.

\*/

**public** **class** **RoadTile** {

**private** **int** x;

**private** **int** y;

**private** **int** condition;

**private** **int** time1;

**private** **int** time2;

**private** **boolean** occupied;

**private** **boolean** free;

/\*

\* conditions

\* 0 - normal - 100% speed rate

\* 1 - damaged - 50% speed rate

\* 2 - flooded - 50% speed rate

\* 3 - closed - 0% speed rate

\*/

/\*\*

\* Constructor method for RoadTile class.

\* **@param** x x coordinate of the road

\* **@param** y y coordinate of the road

\* **@param** condition chosen condition of the road

\*/

**public** **RoadTile**(**int** x, **int** y, **int** condition) {

**this**.x = x;

**this**.y = y;

**this**.condition = condition;

time1 = 0;

time2 = 0;

free = **true**;

occupied = **false**;

}

/\*\*

\* This method updates the settings of the road.

\*/

**public** **void** **update**() {

**if**(time1 > 99 && condition == 2) {

condition = 0;

time1 = 0;

}

time1 = time1 + 1;

**if**(occupied) {

**if**(time2 > 0) {

free = **true**;

}

time2 = time2 + 1;

}

**if**(condition == 3) {

free = **false**;

occupied = **true**;

}

}

/\*\*

\* This method makes a road unpassable.

\*/

**public** **void** **occupy**() {

free = **false**;

occupied = **true**;

time2 = 0;

}

/\*\*

\* This method makes a road passable.

\*/

**public** **void** **unOccupy**() {

free = **true**;

occupied = **false**;

}

/\*\*

\* This method sets the condition of a road.

\* **@param** condition integer that represents a condition (e.g. Broken, Flooded, Blocked)

\*/

**public** **void** **setCondition**(**int** condition) {

**this**.condition = condition;

}

/\*\*

\* This method returns the current setting of the road.

\* **@return** current setting of the road

\*/

**public** **boolean** **isFree**() {

**return** free;

}

/\*\*

\* This method returns the current setting of the road.

\* **@return** current setting of the road

\*/

**public** **boolean** **isOccupied**() {

**return** occupied;

}

/\*\*

\* This method returns the current condition of the road.

\* **@return** current condition of the road

\*/

**public** **int** **getCondition**() {

**return** condition;

}

/\*\*

\* This method returns the x coordinate of the road.

\* **@return** x coordinate of the road

\*/

**public** **int** **getX**() {

**return** x;

}

/\*\*

\* This method returns the y coordinate of the road.

\* **@return** y coordinate of the road

\*/

**public** **int** **getY**() {

**return** y;

}

}

import java.io.FileNotFoundException;

import java.io.IOException;

import java.net.URISyntaxException;

import java.util.ArrayList;

import java.util.concurrent.ThreadLocalRandom;

import javax.swing.JFrame;

/\*\*

\* This class sets the settings for the simulation.

\*/

public class Simulation extends Thread {

private int time;

/\*

\* 0 - road direction down

\* 1 - road direction up

\* 2 - road direction left

\* 3 - road direction right

\* 4 - road intersection

\* 5 - vehicle

\* 6 - traffic light

\* 7 - drop off

\* 8 - spawner

\* 9 - wall

\* 10 - road tile

\*/

private Map m = new Map();

private boolean pause;

private boolean opened;

private int timeSpeed;

private JFrame f;

private Thread t;

/\*\*

\* Constructor method for Simulation class

\*/

public Simulation() {

pause = false;

opened = false;

timeSpeed = 500;

}

/\*\*

\* This method sets the elements of the simulation

\* @param width - width of the simulator

\* @param height - height of the simulator

\*/

public void setDisplay(int width, int height) {

f = new JFrame("Traffic Flow Simulation");

//f.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

f.setSize(width, height);

f.setLocationRelativeTo(null);

f.add(new Display(m));

f.setVisible(true);

}

/\*\*

\* This method updates the simulation events based on the simulation clock

\*/

public void updateSimulation() {

int i, x, y;

ArrayList <Car> carList = new ArrayList <Car> (m.getCarList());

ArrayList <Bus> busList = new ArrayList <Bus> (m.getBusList());

ArrayList <Road> roadList = new ArrayList <Road> (m.getRoadList());

ArrayList <Wall> wallList = new ArrayList <Wall> (m.getWallList());

ArrayList <DropOff> dropOffList = new ArrayList <DropOff> (m.getDropOffList());

ArrayList <RoadTile> roadTileList = new ArrayList <RoadTile> (m.getRoadTileList());

ArrayList <TrafficLight> trafficLightList = new ArrayList <TrafficLight> (m.getTrafficLightList());

ArrayList <Spawner> spawnerList = new ArrayList <Spawner> (m.getSpawnerList());

int width = m.getWidth();

int height = m.getHeight();

int objectMap[][] = m.getObjectMap();

int map[][] = m.getNumMap();

Block block[][] = m.getBlockMap();

//This updates the road tiles.

for(i = 0; i < roadTileList.size(); i++) {

roadTileList.get(i).update();

}

//This updates the drop off points.

for(i = 0; i < dropOffList.size(); i++) {

dropOffList.get(i).update();

}

//This updates the traffic lights into the Block.

for(i = 0; i < trafficLightList.size(); i++) {

trafficLightList.get(i).update();

}

//This clears the object map.

for(x = 0; x < width; x++) {

for(y = 0; y < height; y++) {

objectMap[x][y] = 0;

}

}

//This adds walls into the object map.

for(i = 0; i < wallList.size(); i++) {

x = wallList.get(i).getX();

y = wallList.get(i).getY();

objectMap[x][y] = 5;

}

//This will reset all the cars.

for(i = 0; i < carList.size(); i++) {

carList.get(i).setHasMoved(false);

x = carList.get(i).getX();

y = carList.get(i).getY();

objectMap[x][y] = 5;

}

for(i = 0; i < busList.size(); i++) {

busList.get(i).setHasMoved(false);

x = busList.get(i).getX();

y = busList.get(i).getY();

objectMap[x][y] = 5;

x = busList.get(i).getX2();

y = busList.get(i).getY2();

objectMap[x][y] = 5;

}

boolean remove = false;

//This is where the cars will move.

for(i = 0; i < carList.size(); i++) {

carList = m.getCarList();

if(remove)

i = 0;

remove = false;

carList.get(i).setNeighbors(checkNeighbors(carList.get(i)));

carList.get(i).setRoad(checkRoads(carList.get(i)));

x = carList.get(i).getX();

y = carList.get(i).getY();

if(!carList.get(i).getHasMoved()) {

if(block[x][y].hasRoadTile() && block[x][y].getRoadTile().getCondition() == 3){

carList.get(i).setHasMoved(true);

}

else {

if(block[x][y].hasRoadTile() && (block[x][y].getRoadTile().getCondition() == 1 || block[x][y].getRoadTile().getCondition() == 2 )){

if(block[x][y].getRoadTile().isOccupied()) {

if(block[x][y].getRoadTile().isFree()) {

carList.get(i).move(map[x][y]);

block[x][y].getRoadTile().unOccupy();

}

}

else if(block[x][y].getRoadTile().isFree() && !block[x][y].getRoadTile().isOccupied())

block[x][y].getRoadTile().occupy();

else

carList.get(i).move(map[x][y]);

}

else if(block[x][y].hasDropOff()) {

if(block[x][y].getDropOff().isOccupied()) {

if(block[x][y].getDropOff().isFree()) {

carList.get(i).move(map[x][y]);

block[x][y].getDropOff().unOccupy();

}

}

else if(block[x][y].getDropOff().isFree() && !block[x][y].getDropOff().isOccupied())

block[x][y].getDropOff().occupy();

else

carList.get(i).move(map[x][y]);

}

else if(block[x][y].getHasTrafficLight()) {

if(block[x][y].getTrafficLight().getGo())

carList.get(i).move(map[x][y]);

}

else

carList.get(i).move(map[x][y]);

carList.get(i).setHasMoved(true);

if(carList.get(i).getX() < 0 || carList.get(i).getX() > width - 1 || carList.get(i).getY() < 0 || carList.get(i).getY() > height - 1) {

m.removeCar(i);

remove = true;

}

else {

x = carList.get(i).getX();

y = carList.get(i).getY();

objectMap[x][y] = 5;

}

}

}

else {

x = carList.get(i).getX();

y = carList.get(i).getY();

objectMap[x][y] = 5;

carList.get(i).setHasMoved(true);

}

}

//This is where the buses will move.

for(i = 0; i < busList.size(); i++) {

busList = m.getBusList();

if(remove)

i = 0;

remove = false;

busList.get(i).setNeighbors(checkNeighbors(busList.get(i)));

busList.get(i).setRoad(checkRoads(busList.get(i)));

x = busList.get(i).getX();

y = busList.get(i).getY();

if(!busList.get(i).getHasMoved()) {

if(block[x][y].hasRoadTile() && (block[x][y].getRoadTile().getCondition() == 1 || block[x][y].getRoadTile().getCondition() == 2)){

if(block[x][y].getRoadTile().isOccupied()) {

if(block[x][y].getRoadTile().isFree()) {

busList.get(i).move(map[x][y]);

block[x][y].getRoadTile().unOccupy();

}

}

else if(block[x][y].getRoadTile().isFree() && !block[x][y].getRoadTile().isOccupied())

block[x][y].getRoadTile().occupy();

else

busList.get(i).move(map[x][y]);

}

else if(block[x][y].hasDropOff()) {

if(block[x][y].getDropOff().isOccupied()) {

if(block[x][y].getDropOff().isFree()) {

busList.get(i).move(map[x][y]);

block[x][y].getDropOff().unOccupy();

}

}

else if(block[x][y].getDropOff().isFree() && !block[x][y].getDropOff().isOccupied())

block[x][y].getDropOff().occupy();

else

busList.get(i).move(map[x][y]);

}

else if(block[x][y].getHasTrafficLight()) {

if(block[x][y].getTrafficLight().getGo())

busList.get(i).move(map[x][y]);

}

else

busList.get(i).move(map[x][y]);

busList.get(i).setHasMoved(true);

if(busList.get(i).getX() < 0 || busList.get(i).getX() > width - 1 ||busList.get(i).getY() < 0 || busList.get(i).getY() > height - 1) {

m.removeBus(i);

remove = true;

}

else {

x = busList.get(i).getX();

y = busList.get(i).getY();

objectMap[x][y] = 5;

x = busList.get(i).getX2();

y = busList.get(i).getY2();

objectMap[x][y] = 5;

}

}

else {

x = busList.get(i).getX();

y = busList.get(i).getY();

objectMap[x][y] = 5;

x = busList.get(i).getX2();

y = busList.get(i).getY2();

objectMap[x][y] = 5;

}

}

//This is where the spawners will spawn cars.

for(i = 0; i < spawnerList.size(); i++) {

x = spawnerList.get(i).getX();

y = spawnerList.get(i).getY();

spawnerList.get(i).update();

if(spawnerList.get(i).getSpawn() && objectMap[x][y] != 5) {

int num1;

num1 = chooseNum(0, 1, 2);

if(num1 == 0)

m.addBus(x, y);

else

m.addCar(x, y);

spawnerList.get(i).setSpawn(false);

}

}

time++; //This adds 1 to the simulation clock.

m.setNumMap(map);

m.setObjectMap(objectMap);

m.setBlockMap(block);

/\*

m.setCarList(carList);

m.setBusList(busList);

m.setRoadList(roadList);

m.setWallList(wallList);

m.setTrafficList(trafficLightList);

m.setDropOffList(dropOffList);

m.setRoadTiles(roadTileList);

m.setSpawnerList(spawnerList);

\*/

}

public void setVisible(boolean visibility) {

f.setVisible(visibility);

}

public void toggleVisibility() {

if(f.isVisible())

f.setVisible(false);

else

f.setVisible(true);

}

/\*\*

\* This method randomizes between three numbers

\* @param a - 1st number

\* @param b - 2nd number

\* @param c - 3rd number

\* @return randomized number

\*/

public int chooseNum(int a, int b, int c) {

int rand;

rand = ThreadLocalRandom.current().nextInt(1, 4);

if(rand == 1)

return a;

else if(rand == 2)

return b;

else if(rand == 3)

return c;

return a;

}

/\*\*

\* This method checks for adjacent cars

\* @param c - car

\* @return adjacent cars

\*/

public boolean[] checkNeighbors(Car c) {

int x, y;

boolean[] neighbor = new boolean[4];

int height = m.getHeight();

int width = m.getWidth();

int objectMap[][] = m.getObjectMap();

x = c.getX();

y = c.getY();

neighbor[0] = false;

neighbor[1] = false;

neighbor[2] = false;

neighbor[3] = false;

if(y > 0) {

if(objectMap[x][y-1] == 5)

neighbor[0] = true;

}

if(y < height - 1) {

if(objectMap[x][y+1] == 5)

neighbor[1] = true;

}

if(x > 0) {

if(objectMap[x-1][y] == 5)

neighbor[2] = true;

}

if(x < width - 1) {

if(objectMap[x+1][y] == 5)

neighbor[3] = true;

}

return neighbor;

}

/\*\*

\* This method checks for adjacent buses

\* @param c - bus

\* @return adjacent buses

\*/

public boolean[] checkNeighbors(Bus c) {

int x, y;

boolean[] neighbor = new boolean[4];

int height = m.getHeight();

int width = m.getWidth();

int objectMap[][] = m.getObjectMap();

x = c.getX();

y = c.getY();

neighbor[0] = false;

neighbor[1] = false;

neighbor[2] = false;

neighbor[3] = false;

if(y > 0) {

if(objectMap[x][y-1] == 5)

neighbor[0] = true;

}

if(y < height - 1) {

if(objectMap[x][y+1] == 5)

neighbor[1] = true;

}

if(x > 0) {

if(objectMap[x-1][y] == 5)

neighbor[2] = true;

}

if(x < width - 1) {

if(objectMap[x+1][y] == 5)

neighbor[3] = true;

}

return neighbor;

}

/\*\*

\* This method checks the road a car is in

\* @param c - car

\* @return the road the car is in

\*/

public int[] checkRoads(Car c) {

int x, y;

int road[] = new int[4];

int height = m.getHeight();

int width = m.getWidth();

int map[][] = m.getNumMap();

x = c.getX();

y = c.getY();

road[0] = 0;

road[1] = 1;

road[2] = 2;

road[3] = 3;

if(y > 0)

road[0] = map[x][y-1];

if(y < height - 1)

road[1] = map[x][y+1];

if(x > 0)

road[2] = map[x-1][y];

if(x < width - 1)

road[3] = map[x+1][y];

return road;

}

/\*\*

\* This method checks the road the bus is in

\* @param c bus

\* @return road the bus is in

\*/

public int[] checkRoads(Bus c) {

int x, y;

int road[] = new int[4];

int height = m.getHeight();

int width = m.getWidth();

int map[][] = m.getNumMap();

x = c.getX();

y = c.getY();

road[0] = 0;

road[1] = 1;

road[2] = 2;

road[3] = 3;

if(y > 0)

road[0] = map[x][y-1];

if(y < height - 1)

road[1] = map[x][y+1];

if(x > 0)

road[2] = map[x-1][y];

if(x < width - 1)

road[3] = map[x+1][y];

return road;

}

/\*\*

\* This method runs the simulation

\*/

public void displaySimulation() {

runSimulation();

pause = true;

}

/\*\*

\* This method returns the greatest common factor of the two selected numbers.

\* @param a number 1

\* @param b number 2

\* @return the greatest common factor of the two selected numbers

\*/

public int gcf(int a, int b)

{

while (a != b) // while the two numbers are not equal...

{

// ...subtract the smaller one from the larger one

if (a > b) a -= b; // if a is larger than b, subtract b from a

else b -= a; // if b is larger than a, subtract a from b

}

return a; // or return b, a will be equal to b either way

}

/\*\*

\* This method returns the least common multiple of the two selected numbers.

\* @param a the first number

\* @param b the second number

\* @return the least common multiple of the two selected numbers.

\*/

public int lcm(int a, int b)

{

// the lcm is simply (a \* b) divided by the gcf of the two

return (a \* b) / gcf(a, b);

}

/\*\*

\* This method launches the simulation

\*/

public void runSimulation() {

t = new Thread() {

public void run() {

if(opened == false) {

try {

m.readMap();

} catch (URISyntaxException e) {

// TODO Auto-generated catch block

e.printStackTrace();

}

int lcd;

int windowWidth;

int windowHeight;

int num1;

int num2 = 750;

int height = m.getHeight();

int width = m.getWidth();

lcd = lcm(width, height);

windowWidth = lcd / width;

windowHeight = lcd / height;

if(windowWidth > windowHeight) {

num1 = num2 / windowWidth;

windowWidth = windowHeight \* num1;

windowHeight = num2;

}

else {

num1 = num2 / windowHeight;

windowHeight = windowWidth \* num1;

windowWidth = num2;

}

setDisplay(windowWidth, windowHeight);

opened = true;

}

while(pause == false) {

try {

Thread.sleep(timeSpeed);

} catch (InterruptedException e) {

e.printStackTrace();

}

updateSimulation();

}

}

};

t.start();

}

/\*\*

\* This method lets the user pause and resume the simulation

\*/

public void togglePausePlay() {

if(pause == false)

pause = true;

else {

pause = false;

runSimulation();

}

}

/\*\*

\* This method lets the pause the simulation and proceed to the next frame

\*/

public void nextStep() {

if(pause) {

updateSimulation();

}

}

/\*\*

\* This method stops the simulation

\*/

public void stopSimulation() {

if(t.isAlive()) {

t.interrupt();

t.suspend();

t = null;

}

f.dispose();

opened = false;

m.delete();

}

/\*\*

\* This method sets the speed to update the simulation

\* @param time - speed to update

\*/

public void setTimeSpeed(int time) {

timeSpeed = time;

}

/\*\*

\* This sets the name of the map used in the simulation

\* @param mapName - name of the map

\*/

public void setMapName(String mapName) {

m.setMapName(mapName);

}

/\*\*

\* This method sets the rate at which a car stays in a road block before moving

\* @param miliseconds - rate at which the car moves

\*/

public void wait(int miliseconds) {

//1000 milliseconds is equal to one second.

try {

Thread.sleep(miliseconds);

} catch(InterruptedException ex) {

Thread.currentThread().interrupt();

}

}

/\*\*

\* This method checks whether or not the selected map has values in it or not.

\* @param mapName the name of the text file to check

\* @return the boolean value of whether or not there is text in that text file or not

\* @throws FileNotFoundException

\*/

public boolean isEmpty(String mapName) throws FileNotFoundException {

Reader r = new Reader();

return r.isEmpty(mapName);

}

/\*\*

\* This method returns the Map.

\* @return the Map

\*/

public Map getMap() {

return m;

}

/\*\*

\* This method returns the current status of the simulation

\* @return current status of the simulation

\*/

public boolean getPause() {

return pause;

}

/\*\*

\* This method lists all the positions of all the cars present in the simulation

\* @return cars present in the simulation

\*/

public ArrayList <Car> getCarList() {

return m.getCarList();

}

/\*\*

\* This method returns all the bus present in the simulation

\* @return buses present in the simulation

\*/

public ArrayList <Bus> getBusList() {

return m.getBusList();

}

/\*\*

\* This method returns the position of all the walls in the simulation

\* @return the walls in the simulation

\*/

public ArrayList <Wall> getWallList() {

return m.getWallList();

}

/\*\*

\* This method returns the positions of all the Spawner in the simulation

\* @return the spawners in the simulations

\*/

public ArrayList <Spawner> getSpawnerList() {

return m.getSpawnerList();

}

/\*\*

\* This method returns the positions of all the Roads in the simulations

\* @return the roads in the simulation

\*/

public ArrayList <Road> getRoadList() {

return m.getRoadList();

}

/\*\*

\* This method returns the positions of all the Drop Off blocks in the simulation

\* @return the Drop Off blocks in the simulation

\*/

public ArrayList <DropOff> getDropOffList() {

return m.getDropOffList();

}

/\*\*

\* This method returns the positions of all the Traffic Light Blocks in the simulation

\* @return the Traffic Light blocks in the simulation

\*/

public ArrayList <TrafficLight> getTrafficLightList() {

return m.getTrafficLightList();

}

/\*\*

\* This method gets all the Road Tiles.

\* @return the ArrayList of all the Road Tiles.

\*/

public ArrayList <RoadTile> getRoadTileList() {

return m.getRoadTileList();

}

}

import java.awt.BorderLayout;

import java.awt.EventQueue;

import javax.swing.JFrame;

import javax.swing.JPanel;

import javax.swing.border.EmptyBorder;

import java.awt.GridLayout;

import javax.swing.JButton;

import javax.swing.BoxLayout;

import java.awt.GridBagLayout;

import java.awt.GridBagConstraints;

import java.awt.event.ActionListener;

import java.awt.event.ActionEvent;

import net.miginfocom.swing.MigLayout;

import java.awt.FlowLayout;

import javax.swing.JToggleButton;

import javax.swing.JSlider;

import javax.swing.JLabel;

import java.awt.event.MouseAdapter;

import java.awt.event.MouseEvent;

import java.awt.Insets;

import javax.swing.GroupLayout;

import javax.swing.GroupLayout.Alignment;

import javax.swing.LayoutStyle.ComponentPlacement;

import javax.swing.JSpinner;

import javax.swing.JTree;

import javax.swing.SpinnerNumberModel;

import java.awt.event.InputMethodListener;

import java.awt.event.InputMethodEvent;

import javax.swing.JTextField;

import com.jgoodies.forms.layout.FormLayout;

import com.jgoodies.forms.layout.ColumnSpec;

import com.jgoodies.forms.layout.FormSpecs;

import com.jgoodies.forms.layout.RowSpec;

import java.awt.CardLayout;

import javax.swing.JMenuBar;

import javax.swing.JMenu;

import javax.swing.JMenuItem;

import javax.swing.JOptionPane;

import java.beans.PropertyChangeListener;

import java.beans.PropertyChangeEvent;

import javax.swing.event.ChangeListener;

import javax.swing.event.ChangeEvent;

import javax.swing.SwingConstants;

import java.awt.Font;

/\*\*

\* This class controls the Simulation Model.

\*/

public class SimulationMenu extends JFrame {

private MainMenu mm;

private JPanel contentPane;

private Simulation s = new Simulation();

private boolean running;

/\*\*

\* Constructor method for SimulationMenu class.

\*/

public SimulationMenu() {

setAutoRequestFocus(false);

setResizable(false);

setTitle("Traffic Flow Simulator");

setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

setBounds(100, 100, 455, 220);

setLocationRelativeTo(null);

JMenuItem map1 = new JMenuItem("Map 1");

JMenuItem map2 = new JMenuItem("Map 2");

JMenuItem map3 = new JMenuItem("Map 3");

JMenuItem map4 = new JMenuItem("Map 4");

JMenuItem map5 = new JMenuItem("Map 5");

JButton startButton = new JButton("Start");

JButton stop = new JButton("Stop");

JMenuBar menuBar = new JMenuBar();

setJMenuBar(menuBar);

JMenu mnFile = new JMenu("Select Map");

menuBar.add(mnFile);

JLabel mapName = new JLabel("no file selected");

mapName.setFont(new Font("Tahoma", Font.ITALIC, 11));

mapName.setBounds(10, 10, 100, 25);

JLabel lblSpeed = new JLabel("Speed");

lblSpeed.setBounds(10, 45, 100, 25);

JLabel speedLabel = new JLabel("");

speedLabel.setHorizontalAlignment(SwingConstants.CENTER);

speedLabel.setBounds(10, 90, 200, 25);

JSlider speed = new JSlider();

speed.addChangeListener(new ChangeListener() {

public void stateChanged(ChangeEvent arg0) {

s.setTimeSpeed(speed.getValue());

speedLabel.setText(speed.getValue() + "");

}

});

speed.setValue(100);

speed.setMinimum(1);

speed.setMaximum(500);

speed.setBounds(10, 70, 200, 25);

startButton.setEnabled(false);

startButton.setBounds(340, 10, 100, 25);

startButton.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent arg0) {

s.setTimeSpeed(speed.getValue());

s.runSimulation();

startButton.setEnabled(false);

running = true;

}

});

JButton display = new JButton("Display");

display.setEnabled(false);

display.setBounds(340, 100, 100, 25);

display.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent arg0) {

s.toggleVisibility();

}

});

JButton next = new JButton("Next Step");

next.setEnabled(false);

next.setBounds(340, 70, 100, 25);

next.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent arg0) {

s.nextStep();

}

});

JButton pause = new JButton("Pause");

pause.setEnabled(false);

pause.setBounds(340, 40, 100, 25);

pause.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent arg0) {

s.togglePausePlay();

if(s.getPause()) {

pause.setText("Play");

next.setEnabled(true);

}

else {

pause.setText("Pause");

next.setEnabled(false);

}

}

});

stop.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e) {

mapName.setText("no map selected");

startButton.setEnabled(false);

display.setEnabled(false);

pause.setEnabled(false);

stop.setEnabled(false);

next.setEnabled(false);

map1.setEnabled(true);

map2.setEnabled(true);

map3.setEnabled(true);

map4.setEnabled(true);

map5.setEnabled(true);

if(running) {

s.setVisible(false);

s.stopSimulation();

}

running = false;

}

});

stop.setEnabled(false);

map1.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e) {

s.setMapName("map\_001.txt");

mapName.setText("map\_001.txt");

startButton.setEnabled(true);

display.setEnabled(true);

pause.setEnabled(true);

stop.setEnabled(true);

map1.setEnabled(false);

map2.setEnabled(false);

map3.setEnabled(false);

map4.setEnabled(false);

map5.setEnabled(false);

}

});

mnFile.add(map1);

map2.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e) {

s.setMapName("map\_002.txt");

mapName.setText("map\_002.txt");

startButton.setEnabled(true);

display.setEnabled(true);

pause.setEnabled(true);

stop.setEnabled(true);

map1.setEnabled(false);

map2.setEnabled(false);

map3.setEnabled(false);

map4.setEnabled(false);

map5.setEnabled(false);

}

});

mnFile.add(map2);

map3.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e) {

s.setMapName("map\_003.txt");

mapName.setText("map\_003.txt");

startButton.setEnabled(true);

display.setEnabled(true);

pause.setEnabled(true);

stop.setEnabled(true);

map1.setEnabled(false);

map2.setEnabled(false);

map3.setEnabled(false);

map4.setEnabled(false);

map5.setEnabled(false);

}

});

mnFile.add(map3);

map4.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent arg0) {

s.setMapName("map\_004.txt");

mapName.setText("map\_004.txt");

startButton.setEnabled(true);

display.setEnabled(true);

pause.setEnabled(true);

stop.setEnabled(true);

map1.setEnabled(false);

map2.setEnabled(false);

map3.setEnabled(false);

map4.setEnabled(false);

map5.setEnabled(false);

}

});

mnFile.add(map4);

map5.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e) {

s.setMapName("map\_005.txt");

mapName.setText("map\_005.txt");

startButton.setEnabled(true);

display.setEnabled(true);

pause.setEnabled(true);

stop.setEnabled(true);

map1.setEnabled(false);

map2.setEnabled(false);

map3.setEnabled(false);

map4.setEnabled(false);

map5.setEnabled(false);

}

});

mnFile.add(map5);

JMenu mnHelp = new JMenu("Help");

menuBar.add(mnHelp);

JMenuItem mntmHelp = new JMenuItem("Help Contents");

mntmHelp.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e) {

JOptionPane frame = new JOptionPane();

JOptionPane.showMessageDialog(frame,

"STEPS TO RUN A MAP:\n" +

"1. Select a map from the Select Menu.\n" +

"2. Start the map.\n" +

"3. Use the buttons on the right to perform their assigned tasks as shown\n" +

"4. Click the Stop button before loading another map\n" +

"5. Happy Simulating!"

);

}

});

mnHelp.add(mntmHelp);

JMenuItem mntmAboutProgram = new JMenuItem("About Program");

mntmAboutProgram.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e) {

JOptionPane frame = new JOptionPane();

JOptionPane.showMessageDialog(frame,

"This program was coded by:\n" +

"Cyrus A. Vatandoost Kakhki and Julius Ceasar Librada\n" +

"as part of their requirement for OBJECTP."

);

}

});

mnHelp.add(mntmAboutProgram);

contentPane = new JPanel();

contentPane.setBorder(new EmptyBorder(5, 5, 5, 5));

setContentPane(contentPane);

contentPane.setLayout(null);

contentPane.add(speedLabel);

contentPane.add(startButton);

contentPane.add(display);

contentPane.add(pause);

contentPane.add(next);

contentPane.add(mapName);

contentPane.add(speed);

JButton btnMenu = new JButton("Menu");

btnMenu.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e) {

if(running) {

s.setVisible(false);

s.stopSimulation();

}

dispose();

mm = new MainMenu();

mm.run();

}

});

btnMenu.setBounds(10, 130, 100, 25);

contentPane.add(btnMenu);

contentPane.add(lblSpeed);

stop.setBounds(340, 130, 100, 25);

contentPane.add(stop);

}

/\*\*

\* This method runs the simulation using the settings.

\*/

public void run() {

EventQueue.invokeLater(new Runnable() {

public void run() {

try {

SimulationMenu frame = new SimulationMenu();

frame.setVisible(true);

} catch (Exception e) {

e.printStackTrace();

}

}

});

}

}

/\*\*

\* This class creates a spawner.

\*/

**public** **class** **Spawner** {

**private** **int** x;

**private** **int** y;

**private** **int** rate;

**private** **int** time;

**private** **boolean** spawn;

/\*\*

\* Constructor method for Spawner class

\* **@param** x - x coordinate of spawner

\* **@param** y - y cooridnate of spawner

\* **@param** rate - rate at which a car spawns

\*/

**public** **Spawner**(**int** x, **int** y, **int** rate) {

**this**.x = x;

**this**.y = y;

**this**.rate = rate;

time = 0;

spawn = **false**;

}

/\*\*

\* This method spawns a car after a certain time

\*/

**public** **void** **update**() {

**if**(time > 99) {

spawn = **true**;

time = 0;

}

time += rate;

}

/\*\*

\* This method identifies a block as a spawner

\* **@param** spawn - identifies a certain block as spawner

\*/

**public** **void** **setSpawn**(**boolean** spawn) {

**this**.spawn = spawn;

}

/\*\*

\* This method returns the x coordinate of a spawner

\* **@return** x coordinate of the spawner

\*/

**public** **int** **getX**() {

**return** x;

}

/\*\*

\* This method returns the y coordinate of the spawner

\* **@return** y coordinate of the spawner

\*/

**public** **int** **getY**() {

**return** y;

}

/\*\*

\* This method returns the position of the spawner

\* **@return** the position of the spawner

\*/

**public** **boolean** **getSpawn**() {

**return** spawn;

}

/\*\*

\* This method returns the rate at which cars spawn

\* **@return** rate at which cars spawn

\*/

**public** **int** **getRate**() {

**return** rate;

}

}

/\*\*

\* This class creates a Traffic Light.

\*/

**public** **class** **TrafficLight** {

**private** **int** x;

**private** **int** y;

**private** **int** time;

**private** **int** rate;

**private** **int** start;

**private** **boolean** go;

/\*\*

\* Constructor method for TrafficLight class

\* **@param** x - x coordinate of traffic light block

\* **@param** y - y coordinate of traffic light block

\* **@param** rate - rate at which the color of the traffic light changes

\* **@param** start - start time after light change

\*/

**public** **TrafficLight**(**int** x, **int** y, **int** rate, **int** start) {

**this**.x = x;

**this**.y = y;

**this**.rate = rate;

time = 0;

**if**(start > 0)

go = **false**;

**else**

go = **true**;

**this**.start = start;

}

/\*\*

\* This method updates the color of the traffic light depending on the rate

\*/

**public** **void** **update**() {

**if**(time > 450) {

changeLight();

time = 0;

}

time += rate;

}

/\*\*

\* This method changes the color of the traffic light

\*/

**public** **void** **changeLight**() {

**if**(go) {

go = **false**;

}

**else** {

go = **true**;

}

}

/\*\*

\* This method returns traffic light permissions

\* **@return** traffic light permissions

\*/

**public** **boolean** **getGo**() {

**return** go;

}

/\*\*

\* This method returns the x coordinate of the traffic light block

\* **@return** x coordinate of traffic light block

\*/

**public** **int** **getX**() {

**return** x;

}

/\*\*

\* This method returns the y coordinate of the traffic light block

\* **@return** y coordinate of traffic light block

\*/

**public** **int** **getY**() {

**return** y;

}

/\*\*

\* This method returns the rate at which the color of the traffic light changes

\* **@return** rate at which the color of the traffic light changes

\*/

**public** **int** **getRate**() {

**return** rate;

}

/\*\*

\* This method returns the start time after a light changes

\* **@return** start time after light changes

\*/

**public** **int** **getStart**() {

**return** start;

}

}

import java.awt.Color;

import java.util.concurrent.ThreadLocalRandom;

/\*\*

\* This class has the settings for inheritance for Bus and Car.

\*/

public class Vehicle {

protected int x;

protected int y;

protected int lastDirection;

protected boolean hasMoved;

protected boolean[] neighbor;

protected int[] road;

protected Color color;

/\*\*

\* Constructor method for Vehicle class.

\*/

public Vehicle() {

chooseColor();

}

/\*\*

\* This method chooses a color for the vehicle.

\*/

protected void chooseColor() {

int num;

num = ThreadLocalRandom.current().nextInt(0, 6);

Color PURPLE = new Color(153, 51, 255);

Color DARKGREEN = new Color(0, 204, 102);

switch(num) {

case 0:

color = Color.PINK;

break;

case 1:

color = Color.ORANGE;

break;

case 2:

color = DARKGREEN;

break;

case 3:

color = PURPLE;

break;

case 4:

color = Color.MAGENTA;

break;

case 5:

color = Color.BLUE;

break;

}

}

/\*\*

\* This method randomized between two numbers that represent a color.

\* @param a 1st number

\* @param b 2nd number

\* @return randomized number

\*/

protected int chooseNum(int a, int b) {

int rand;

rand = ThreadLocalRandom.current().nextInt(1, 3);

if(rand == 1)

return a;

else if(rand == 2)

return b;

return a;

}

/\*\*

\* This method randomizes between three numbers.

\* @param a 1st number

\* @param b 2nd number

\* @param c 3rd number

\* @return randomized number

\*/

protected int chooseNum(int a, int b, int c) {

int rand;

rand = ThreadLocalRandom.current().nextInt(1, 4);

if(rand == 1)

return a;

else if(rand == 2)

return b;

else if(rand == 3)

return c;

return a;

}

}

/\*\*

\* This class creates a Wall.

\*/

**public** **class** **Wall** {

**private** **int** x;

**private** **int** y;

/\*\*

\* Constructor method for class Wall

\* **@param** x - x coordinate of the wall block

\* **@param** y - y coordinate of the wall block

\*/

**public** **Wall**(**int** x, **int** y) {

**this**.x = x;

**this**.y = y;

}

/\*\*

\* This method returns the x coordinate of the wall

\* **@return** x coordinate of the wall

\*/

**public** **int** **getX**() {

**return** x;

}

/\*\*

\* This method returns the y coordinate of the wall

\* **@return** y coordinate of the wall

\*/

**public** **int** **getY**() {

**return** y;

}

}