МИНИСТЕРСТВО ЦИФРОВОГО РАЗВИТИЯ, СВЯЗИ И МАССОВЫХ КОММУНИКАЦИЙ РОССИЙСКОЙ ФЕДЕРАЦИИ

Ордена Трудового Красного Знамени федеральное государственное бюджетное образовательное учреждение высшего образования

Московский технический университет связи и информатики

(МТУСИ)

**Лабораторная работа №3 по теме**

**«Алгоритм A\*»**

Выполнил:

Студент группы БФИ1801

Сиротин Н. С.

**Москва 2020**

**Цель работы:** Java позволяет использовать объекты. В данной лабораторной работе необходимо использовать классы по одному на файл, чтобы описать, как эти объекты работают.

**Задачи:** Для начала необходимо подготовить класс Location для совместного использования с классами коллекции Java.

Поскольку вы будете использовать контейнеры для хеширования для выполнения данного задания, то для этого необходимо:

• Обеспечить реализацию метода equals ().

• Обеспечить реализацию метода hashcode().

Добавьте реализацию каждого из этих методов в класс Location, следуя шаблонам в классе. После этого вы можете использовать класс Location в качестве ключевого типа в контейнерах хеширования, таких как HashSet и HashMap

Состояния А\* После того, как класс Location готов к использованию, вы можете завершить реализацию класса AStarState. Это класс, который поддерживает наборы открытых и закрытых вершин, поэтому он действительно обеспечивает основную функциональность для реализации алгоритма А\*. Как упоминалось ранее, состояние А\* состоит из двух наборов вершин, один из открытых вершин и другой из закрытых. Чтобы упростить алгоритм, вершины будут храниться в хэш-карте, где местоположение вершин является ключом, а сами вершины являются значениями. Таким образом, у вас будет такой тип: HashMap (Очевидный вывод из всего этого заключается в том, что с каждым местоположением на карте может быть связана только одна вершина.) Добавьте два (нестатических) поля в класс AStarState с таким типом, одно для "открытых вершин" и другой для "закрытых вершин". Кроме того, не забудьте инициализировать каждое из этих полей для ссылки на новую пустую коллекцию.

После создания и инициализации полей, вы должны реализовать следующие методы в классе AStarState:

1) public int numOpenWaypoints() Этот метод возвращает количество точек в наборе открытых вершин.

2) public Waypoint getMinOpenWaypoint() Эта функция должна проверить все вершины в наборе открытых вершин, и после этого она должна вернуть ссылку на вершину с наименьшей общей стоимостью. Если в "открытом" наборе нет вершин, функция возвращает NULL.

Не удаляйте вершину из набора после того, как вы вернули ее; просто верните ссылку на точку с наименьшей общей стоимостью.

3) public boolean addOpenWaypoint(Waypoint newWP) Это самый сложный метод в классе состояний А\*.

Данный метод усложняет то, что он должен добавлять указанную вершину только в том случае, если существующая вершина хуже новой. Вот что должен делать этот метод:

• Если в наборе «открытых вершин» в настоящее время нет вершины для данного местоположения, то необходимо просто добавить новую вершину.

• Если в наборе «открытых вершин» уже есть вершина для этой локации, добавьте новую вершину только в том случае, если стоимость пути до новой вершины меньше стоимости пути до текущей. (Убедитесь, что используете не общую стоимость.)

Другими словами, если путь через новую вершину короче, чем путь через текущую вершину, замените текущую вершину на новую Как вы могли заметить, что в таком случае вам потребуется извлечь существующую вершину из «открытого набора», если таковая имеется. Данный шаг довольно прост - замените предыдущую точку на новую, используя метод HashMap.put(), который заменит старое значение на новое. Пусть данный метод вернет значение true, если новая вершина была успешно добавлена в набор, и false в противном случае.

4) public boolean isLocationClosed(Location loc) Эта функция должна возвращать значение true, если указанное местоположение встречается в наборе закрытых вершин, и false в противном случае. Так как закрытые вершины хранятся в хэш-карте с расположениями в качестве ключевых значений, данный метод достаточно просто в реализации.

5) public void closeWaypoint(Location loc) Эта функция перемещает вершину из набора «открытых вершин» в набор «закрытых вершин». Так как вершины обозначены местоположением, метод принимает местоположение вершины.

Процесс должен быть простым:

• Удалите вершину, соответствующую указанному местоположению из набора «открытых вершин».

• Добавьте вершину, которую вы удалили, в набор закрытых вершин. Ключом должно являться местоположение точки. Компиляция и тестирование Как только вы реализуете вышеуказанную функциональность, запустите программу поиска пути, чтобы проверит правильность ее выполнения. Если вы реализовали все правильно, то у вас не должно возникнуть проблем при создании препятствий и последующим поиском путей вокруг них.

Скомпилируйте и запустите программу также, как и всегда: javac \*.java java AStarApp

**Код класса Location:**

import java.util.Objects;  
  
public class Location  
{  
 public int xCoord;  
 public int yCoord;  
  
 public Location(int x, int y) {  
 xCoord = x;  
 yCoord = y;  
 }  
  
 public Location()  
 {  
 this(0, 0);  
 }  
  
 @Override  
 public boolean equals(Object obj) {  
 if (obj == this)  
 return true;  
 if (obj == null || obj.getClass() != this.getClass())  
 return false;  
  
 Location location = (Location) obj;  
 return xCoord == location.xCoord && yCoord == location.yCoord;  
 }  
  
 @Override  
 public int hashCode() {  
 int *result* = 17;  
 *result* = 37 \* *result* + xCoord;  
 *result* = 37 \* *result* + yCoord;  
 return *result*;  
 }  
}

**Код класса AStarState:**

import java.util.HashMap;  
import java.util.Iterator;  
import java.util.Map;  
import java.util.Set;  
import java.util.concurrent.ForkJoinWorkerThread;  
  
public class AStarState  
{  
 private Map2D map;  
 private HashMap<Location, Waypoint> openWaypoints = new HashMap<>();  
 private HashMap<Location, Waypoint> closeWaypoints = new HashMap<>();  
  
 public AStarState(Map2D map) {  
 if (map == null)  
 throw new NullPointerException("map cannot be null");  
 this.map = map;  
 }  
  
  
 public Map2D getMap()  
 {  
 return map;  
 }  
  
 public Waypoint getMinOpenWaypoint() {  
 if (numOpenWaypoints() == 0) {  
 return null;  
 }  
 Waypoint *minWaypoint* = null;  
 float *min* = Float.MAX\_VALUE;  
 for (Waypoint waypoint : openWaypoints.values()) {  
 float cost = waypoint.getTotalCost();  
 if (cost < *min*) {  
 *min* = cost;  
 *minWaypoint* = waypoint;  
 }  
 }  
 return *minWaypoint*;  
 }  
  
  
 public boolean addOpenWaypoint(Waypoint newWP) {  
 Waypoint openWP = openWaypoints.get(newWP.loc);  
  
 if (openWP == null || newWP.getPreviousCost() < openWP.getPreviousCost()) {  
 openWaypoints.put(newWP.loc, newWP);  
 return true;  
 }  
 return false;  
 }  
  
  
 public int numOpenWaypoints()  
 {  
 return openWaypoints.size();  
 }  
  
 public void closeWaypoint(Location loc) {  
 Waypoint waypoint = openWaypoints.remove(loc);  
 if (openWaypoints != null) {  
 closeWaypoints.put(loc, waypoint);  
 }  
 }  
 public boolean isLocationClosed(Location loc)  
 {  
 return closeWaypoints.containsKey(loc);  
 }  
}

**Код класса для тестирования AStarApp:**

import java.awt.\*;  
import java.awt.event.\*;  
import javax.swing.\*;  
  
  
//класс для алгоритма AStar  
public class AStarApp {  
 //количество ячеек  
 private int width;  
 private int height;  
 //стартовая локация  
 private Location startLoc;  
 //точка назначения  
 private Location finishLoc;  
 //карта из ячеек  
 private JMapCell[][] mapCells;  
  
   
 //обработчик событий кликов на карту ячеек  
 private class MapCellHandler implements MouseListener {  
 //нажата ли клавиша мыши true/false  
 private boolean modifying;  
 //является ли ячейка стеной ?  
 private boolean makePassable;  
   
  
 public void mousePressed(MouseEvent e) {  
 modifying = true;  
 JMapCell cell = (JMapCell) e.getSource();  
 //если мы кликаем по стенке - она перестает быть стеной  
 //если кликаем по пустому метсу - оно становится стенкой  
 makePassable = !cell.isPassable();  
 cell.setPassable(makePassable);  
 }  
  
 public void mouseReleased(MouseEvent e) {  
 modifying = false;  
 }  
   
 //чтобы зажать мышь и непрерывно рисовать стены  
 public void mouseEntered(MouseEvent e) {  
 if (modifying){  
 JMapCell cell = (JMapCell) e.getSource();  
 cell.setPassable(makePassable);  
 }  
 }  
  
  
 public void mouseExited(MouseEvent e){  
 // ??написано не обращать внимание  
 }  
  
 public void mouseClicked(MouseEvent e){  
 // ??написано не обращать внимание  
 }  
 }  
   
   
 //для того чтобы созздавать карты из ячеек заданого размера  
 public AStarApp(int w, int h) {  
 if (w <= 0)  
 throw new IllegalArgumentException("w must be > 0; got " + w);  
   
 if (h <= 0)  
 throw new IllegalArgumentException("h must be > 0; got " + h);  
   
 width = w;  
 height = h;  
 startLoc = new Location( 5, 5);  
 finishLoc = new Location(w -5, h- 5);  
 }  
   
   
 //для интерфейса  
 private void initGUI() {  
 JFrame frame = new JFrame("AStar 2021 супер алгоритм");  
 frame.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);  
 Container contentPane = frame.getContentPane();  
 contentPane.setLayout(new BorderLayout());  
 GridBagLayout gbLayout = new GridBagLayout();  
 GridBagConstraints gbConstraints = new GridBagConstraints();  
 gbConstraints.fill = GridBagConstraints.BOTH;  
 gbConstraints.weightx = 1;  
 gbConstraints.weighty = 1;  
 gbConstraints.insets.set(0, 0, 1, 1);  
 JPanel mapPanel = new JPanel(gbLayout);  
 mapPanel.setBackground(Color.GRAY);  
 mapCells = new JMapCell[width][height];  
 MapCellHandler cellHandler = new MapCellHandler();  
 for (int *y* = 0; *y* < height; *y*++) {  
 for (int *x* = 0; *x* < width; *x*++) {  
 mapCells[*x*][*y*] = new JMapCell();  
 gbConstraints.gridx = *x*;  
 gbConstraints.gridy = *y*;  
 gbLayout.setConstraints(mapCells[*x*][*y*], gbConstraints);  
 mapPanel.add(mapCells[*x*][*y*]);  
 mapCells[*x*][*y*].addMouseListener(cellHandler);  
 }  
 }  
   
 contentPane.add(mapPanel, BorderLayout.CENTER);  
 JButton findPathButton = new JButton("Рассчитать путь");  
 findPathButton.addActionListener(new ActionListener() {  
 public void actionPerformed(ActionEvent e) { findAndShowPath(); }  
 });  
   
 contentPane.add(findPathButton, BorderLayout.SOUTH);  
   
 frame.pack();  
 frame.setVisible(true);  
  
 mapCells[startLoc.xCoord][startLoc.yCoord].setEndpoint(true);  
 mapCells[finishLoc.xCoord][finishLoc.yCoord].setEndpoint(true);  
 }  
  
 //создаем поле с ячейками и алгоритмом поиска  
 private void start() {  
 SwingUtilities.invokeLater(new Runnable() {  
 public void run() { initGUI(); }  
 });  
 }  
   
  
 private void findAndShowPath() {  
 Map2D map = new Map2D(width, height);  
 map.setStart(startLoc);  
 map.setFinish(finishLoc);  
 for (int *y* = 0; *y* < height; *y*++) {  
 for (int *x* = 0; *x* < width; *x*++) {  
 mapCells[*x*][*y*].setPath(false);  
  
 if (mapCells[*x*][*y*].isPassable())  
 map.setCellValue(*x*, *y*, 0);  
 else  
 map.setCellValue(*x*, *y*, Integer.MAX\_VALUE);  
 }  
 }  
   
 //если путь найден, то можно отметить все точки  
 Waypoint *wp* = AStarPathfinder.computePath(map);  
 while (*wp* != null) {  
 Location loc = *wp*.getLocation();  
 mapCells[loc.xCoord][loc.yCoord].setPath(true);  
   
 *wp* = *wp*.getPrevious();  
 }  
 }  
   
   
 //входная точка для программы  
 public static void main(String[] args) {  
 AStarApp app = new AStarApp(30, 30);  
 app.start();  
 }   
}

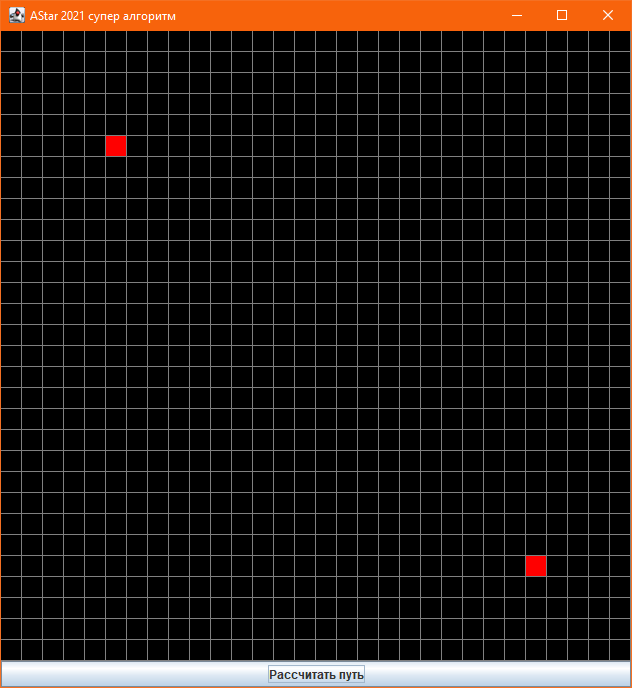


Рисунок 1 — Интерфейс программы

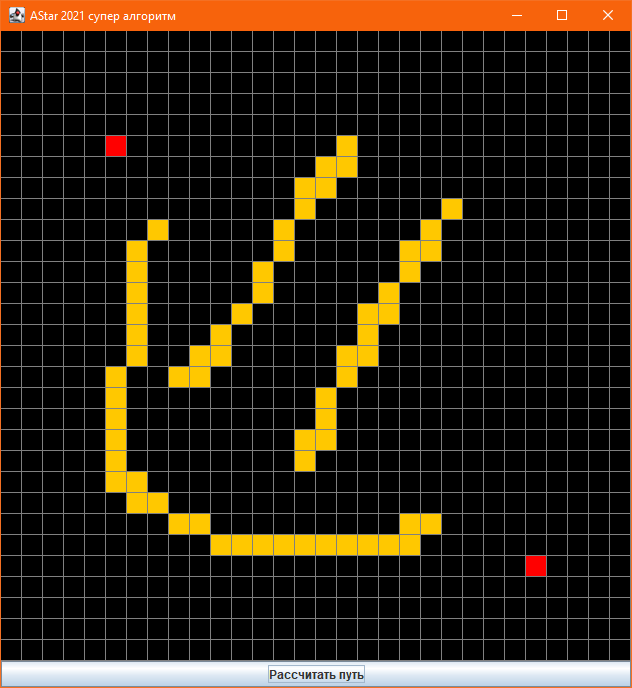


Рисунок 2 — Пример «стен», установленных пользователем

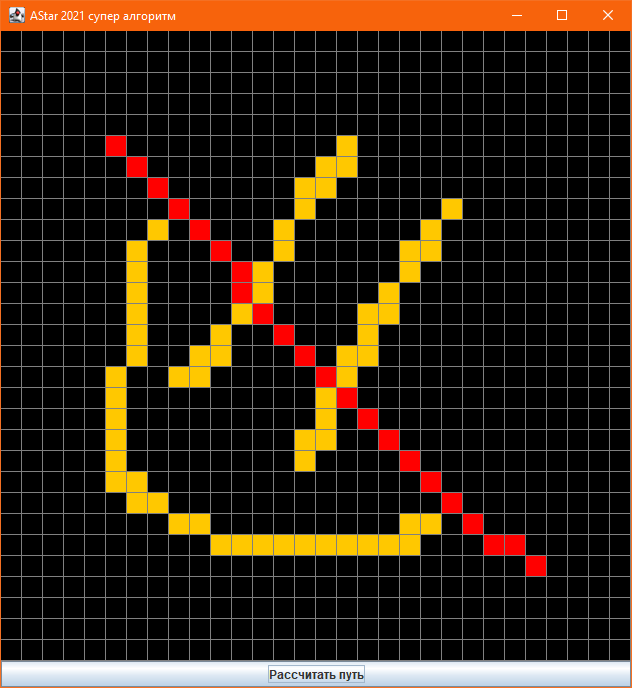


Рисунок 3 — Построение алгоритмом пути

**Вывод:** Мы добавили всю необходимую функциональность в классы, чтобы реализовать работу алгоритма astar в соответствии с поставленным заданием.