Московский технический университет связи и информатики

Лабораторная работа 3

**Я-звезда (A\*)**

Выполнил студент группы БСТ1601

Кузнецов Александр

Москва 2018

Если вы когда-нибудь играли в какую-либо карточную игру на компьютере, вы, вероятно, столкнулись с подразделениями с компьютерного управления, которые знают, как добраться из пункта А в точку Б сами по себе. На самом деле это обычная проблема как в играх, так и в других видах программного обеспечения - как сгенерировать путь от начального местоположения до нужного пункта назначения, который успешно преодолевает препятствия. Один очень широко используемый алгоритм для этого вида проблемы называют А\* (произносится "Я-звезда"), и это - очень эффективный алгоритм для новаторства в компьютерной программе. Алгоритм концептуально довольно прост. Начиная свой путь (самурая) с начального расположения, алгоритм постепенно создает путь от источника до места назначения, всегда используя "до сих пор лучший путь ", чтобы сделать следующий шаг. Это гарантирует, что завершающий путь также будет оптимальным. К счастью, вам не придется выполнять А\* алгоритм; это было уже сделано за вас. На самом деле, есть удобный пользовательский интерфейс для вас, чтобы поэкспериментировать с этим алгоритмом.

Про данный алгоритм написано достаточно большое количество статей, к примеру на Wiki [https://ru.wikipedia.org/wiki/A\*](https://ru.wikipedia.org/wiki/A*)

Каждая итерация А\* алгоритма довольно проста: найти наименее дорогостоящий пункт из набора открытых точек, сделать шаг в любом направлении от этой точки для создания новой открытой точки, а затем переместить путевую точку из открытого набора в закрытый набор. Это повторяется до тех пор, пока текущая точка не достигнет пункта назначения

В нашей программе будет использоваться следующий набор исходных файлов из учебного пособия:

Map2D.java - представляет собой карту, по которой А\* алгоритм двигается, в том числе проходимы ли клетки.

Location.java - этот тип представляет собой координаты конкретной ячейки на карте.

Waypoint.java - представляет отдельные точки в созданный путь.

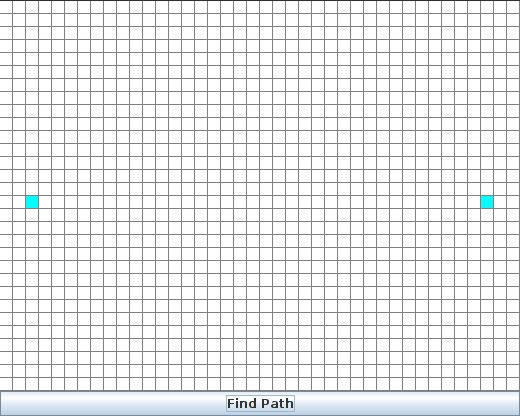
AStarPathfinder.java - этот тип реализует А\* алгоритм поиска пути как статический метод.

AStarState.java - этот тип хранит набор открытых точек и закрытых точек и обеспечивает базовые операции, необходимые для функционирования алгоритма поиска А\*.

AStarApp.java - простое Swing - приложение, которое обеспечивает редактируемый вид 2D карты, и запускает поиск пути по запросу.

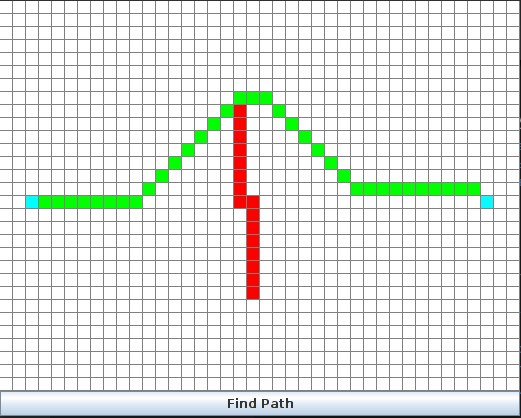
JMapCell.java - это Swing - компонент, который используется для отображения состояния клеток на карте.

Далее, если вы следовали указаниям и программа скомпилировалась и запустилась, у вас появится следующее окно



Далее мы выставляем красные квадратики, так называемый барьер, в обход которого наша программа будет выстраивать кратчайший путь.

В результате получаем следующий результат.



Код программы

+import java.awt.\*;

import java.awt.event.\*;

import javax.swing.\*;

public class AStarApp {

private int width;

private int height;

private Location startLoc;

private Location finishLoc;

private JMapCell[][] mapCells;

private class MapCellHandler implements MouseListener

{

private boolean modifying;

private boolean makePassable;

public void mousePressed(MouseEvent e)

{

modifying = true;

JMapCell cell = (JMapCell) e.getSource();

// If the current cell is passable then we are making them

// impassable; if it's impassable then we are making them passable.

makePassable = !cell.isPassable();

cell.setPassable(makePassable);

}

public void mouseReleased(MouseEvent e)

{

modifying = false;

}

public void mouseEntered(MouseEvent e)

{

if (modifying)

{

JMapCell cell = (JMapCell) e.getSource();

cell.setPassable(makePassable);

}

}

public void mouseExited(MouseEvent e)

{

// This one we ignore.

}

public void mouseClicked(MouseEvent e)

{

// And this one too.

}

}

public AStarApp(int w, int h) {

if (w <= 0)

throw new IllegalArgumentException("w must be bigger than 0; got " + w);

if (h <= 0)

throw new IllegalArgumentException("h must be bigger than 0; got " + h);

width = w;

height = h;

startLoc = new Location(2, h / 2);

finishLoc = new Location(w - 3, h / 2);

}

private void initGUI()

{

JFrame frame = new JFrame("Pathfinder");

frame.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

Container contentPane = frame.getContentPane();

contentPane.setLayout(new BorderLayout());

// Use GridBagLayout because it actually respects the preferred size

// specified by the components it lays out.

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("Find Path");

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);

}

private void start()

{

SwingUtilities.invokeLater(new Runnable() {

public void run() { initGUI(); }

});

}

private void findAndShowPath()

{

// Create a Map2D object containing the current state of the user input.

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);

}

}

// Try to compute a path. If one can be computed, mark all cells in the

// path.

Waypoint wp = AStarPathfinder.computePath(map);

while (wp != null)

{

Location loc = wp.getLocation();

mapCells[loc.xCoord][loc.yCoord].setPath(true);

wp = wp.getPrevious();

}

}

public static void main(String[] args) {

AStarApp app = new AStarApp(40, 30);

app.start();

}

}

AstarPathfinder

+import java.util.HashMap;

import java.util.HashSet;

public class AStarPathfinder

{

public static final float COST\_LIMIT = 1e6f;

public static Waypoint computePath(Map2D map)

{

// Variables necessary for the A\* search.

AStarState state = new AStarState(map);

Location finishLoc = map.getFinish();

// Set up a starting waypoint to kick off the A\* search.

Waypoint start = new Waypoint(map.getStart(), null);

start.setCosts(0, estimateTravelCost(start.getLocation(), finishLoc));

state.addOpenWaypoint(start);

Waypoint finalWaypoint = null;

boolean foundPath = false;

while (!foundPath && state.numOpenWaypoints() > 0)

{

// Find the "best" (i.e. lowest-cost) waypoint so far.

Waypoint best = state.getMinOpenWaypoint();

// If the best location is the finish location then we're done!

if (best.getLocation().equals(finishLoc))

{

finalWaypoint = best;

foundPath = true;

}

// Add/update all neighbors of the current best location. This is

// equivalent to trying all "next steps" from this location.

takeNextStep(best, state);

// Finally, move this location from the "open" list to the "closed"

// list.

state.closeWaypoint(best.getLocation());

}

return finalWaypoint;

}

private static void takeNextStep(Waypoint currWP, AStarState state)

{

Location loc = currWP.getLocation();

Map2D map = state.getMap();

for (int y = loc.yCoord - 1; y <= loc.yCoord + 1; y++)

{

for (int x = loc.xCoord - 1; x <= loc.xCoord + 1; x++)

{

Location nextLoc = new Location(x, y);

// If "next location" is outside the map, skip it.

if (!map.contains(nextLoc))

continue;

// If "next location" is this location, skip it.

if (nextLoc == loc)

continue;

// If this location happens to already be in the "closed" set

// then continue on with the next location.

if (state.isLocationClosed(nextLoc))

continue;

// Make a waypoint for this "next location."

Waypoint nextWP = new Waypoint(nextLoc, currWP);

// OK, we cheat and use the cost estimate to compute the actual

// cost from the previous cell. Then, we add in the cost from

// the map cell we step onto, to incorporate barriers etc.

float prevCost = currWP.getPreviousCost() +

estimateTravelCost(currWP.getLocation(),

nextWP.getLocation());

prevCost += map.getCellValue(nextLoc);

// Skip this "next location" if it is too costly.

if (prevCost >= COST\_LIMIT)

continue;

nextWP.setCosts(prevCost,

estimateTravelCost(nextLoc, map.getFinish()));

// Add the waypoint to the set of open waypoints. If there

// happens to already be a waypoint for this location, the new

// waypoint only replaces the old waypoint if it is less costly

// than the old one.

state.addOpenWaypoint(nextWP);

}

}

}

private static float estimateTravelCost(Location currLoc, Location destLoc)

{

int dx = destLoc.xCoord - currLoc.xCoord;

int dy = destLoc.yCoord - currLoc.yCoord;

return (float) Math.sqrt(dx \* dx + dy \* dy);

}

}

Astarstate

import java.util.HashMap;

public class AStarState

{

private Map2D map;

private HashMap<Location, Waypoint> openWaypoints;

private HashMap<Location, Waypoint> closeWaypoints;

public AStarState(Map2D map)

{

if (map == null)

throw new NullPointerException("map cannot 0");

this.map = map;

openWaypoints = new HashMap<Location, Waypoint>();

closeWaypoints = new HashMap<Location, Waypoint>();

}

public Map2D getMap()

{

return map;

}

public Waypoint getMinOpenWaypoint()

{

Waypoint minWp = null;

float min = Float.POSITIVE\_INFINITY;

for (Waypoint wp: openWaypoints.values()) {

float cost = wp.getTotalCost();

if (cost < min) {

min = cost;

minWp = wp;

}

}

return minWp;

}

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 openWP = openWaypoints.remove(loc);

if (openWP != null) {

closeWaypoints.put(loc, openWP);

}

}

public boolean isLocationClosed(Location loc)

{

return closeWaypoints.containsKey(loc);

}

}

JMAPCELL

import java.awt.\*;

import javax.swing.\*;

import javax.swing.border.\*;

public class JMapCell extends JComponent

{

private static final Dimension CELL\_SIZE = new Dimension(12, 12);

boolean endpoint = false;

boolean passable = true;

boolean path = false;

public JMapCell(boolean pass)

{

// Set the preferred cell size, to drive the initial window size.

setPreferredSize(CELL\_SIZE);

setPassable(pass);

}

public JMapCell()

{

// Call the other constructor, specifying true for "passable".

this(true);

}

public void setEndpoint(boolean end)

{

endpoint = end;

updateAppearance();

}

public void setPassable(boolean pass)

{

passable = pass;

updateAppearance();

}

public boolean isPassable()

{

return passable;

}

public void togglePassable()

{

setPassable(!isPassable());

}

public void setPath(boolean path)

{

this.path = path;

updateAppearance();

}

private void updateAppearance()

{

if (passable)

{

// Passable cell. Indicate its state with a border.

setBackground(Color.WHITE);

if (endpoint)

setBackground(Color.CYAN);

else if (path)

setBackground(Color.GREEN);

}

else

{

// Impassable cell. Make it all red.

setBackground(Color.RED);

}

}

protected void paintComponent(Graphics g)

{

g.setColor(getBackground());

g.fillRect(0, 0, getWidth(), getHeight());

}

}

Location

+public class Location

{

/\*\* X coord \*\*/

public int xCoord;

/\*\* Y coord \*\*/

public int yCoord;

public Location(int x, int y)

{

xCoord = x;

yCoord = y;

}

public Location()

{

this(0, 0);

}

public boolean equals(Object obj)

{

if (obj == null) {

return false;

}

if (obj instanceof Location) {

Location other = (Location) obj;

return xCoord == other.xCoord &&

yCoord == other.yCoord;

}

return false;

}

public int hashCode()

{

int result = 11;

result = 17 \* result + (xCoord \* 17);

result = 7 \* result + (yCoord \* 23);

return result;

}

}

Map2D

public class Map2D

{

private int width;

private int height;

private int[][] cells;

private Location start;

private Location finish;

public Map2D(int width, int height)

{

if (width <= 0 || height <= 0)

{

throw new IllegalArgumentException(

"wdth & hgts shoud be positive; got " + width +

"x" + height);

}

this.width = width;

this.height = height;

cells = new int[width][height];

// Make up some coordinates for start and finish.

start = new Location(0, height / 2);

finish = new Location(width - 1, height / 2);

}

private void checkCoords(int x, int y)

{

if (x < 0 || x > width)

{

throw new IllegalArgumentException("x dolzhno byt' in range [0, " +

width + "), got " + x);

}

if (y < 0 || y > height)

{

throw new IllegalArgumentException("y dolzhno byt' in range [0, " +

height + "), got " + y);

}

}

public int getWidth()

{

return width;

}

public int getHeight()

{

return height;

}

public boolean contains(int x, int y)

{

return (x >= 0 && x < width && y >= 0 && y < height);

}

public boolean contains(Location loc)

{

return contains(loc.xCoord, loc.yCoord);

}

public int getCellValue(int x, int y)

{

checkCoords(x, y);

return cells[x][y];

}

public int getCellValue(Location loc)

{

return getCellValue(loc.xCoord, loc.yCoord);

}

public void setCellValue(int x, int y, int value)

{

checkCoords(x, y);

cells[x][y] = value;

}

public Location getStart()

{

return start;

}

public void setStart(Location loc)

{

if (loc == null)

throw new NullPointerException("loc ne mozhet be 0");

start = loc;

}

public Location getFinish()

{

return finish;

}

public void setFinish(Location loc)

{

if (loc == null)

throw new NullPointerException("loc ne mozhet be 0");

finish = loc;

}

}

Waypoint

+public class Waypoint

{

Location loc;

Waypoint prevWaypoint;

private float prevCost;

private float remainingCost;

public Waypoint(Location loc, Waypoint prevWaypoint)

{

this.loc = loc;

this.prevWaypoint = prevWaypoint;

}

public Location getLocation()

{

return loc;

}

public Waypoint getPrevious()

{

return prevWaypoint;

}

public void setCosts(float prevCost, float remainingCost)

{

this.prevCost = prevCost;

this.remainingCost = remainingCost;

}

public float getPreviousCost()

{

return prevCost;

}

public float getRemainingCost()

{

return remainingCost;

}

public float getTotalCost()

{

return prevCost + remainingCost;

}

}