Artificial Intelligence Report

Darwon Rashid
40280334@napier.ac.uk
Edinburgh Napier University - Artificial Intelligence (SET08118)

1 Introduction

Pathfinding algorithms are used in many everyday scenarios such as looking up routes on Google Maps, planning bus routes , and even ordering items from online services such as Amazon. They take a starting and ending point and will figure out the best possible path defined by its Heuristics. There is a variety of pathfinding algorithms that work in different ways to achieve its goal.

2 A* Algorithm

Invented in 1968 by AI researchers who were trying to improve the path finding that existed at that time. Dijkstra's algorithm was then the best method for path finding until they came up with the A* algorithm. It is an informed search algorithm or as frequently called a best-first search, which means that it finds the shortest path by searching for paths that have the smallest "cost" which is defined by the Heuristic function which is a rule such as the least distance travelled to goal or shortest time to goal. The algorithm is a combination of heuristic approaches like Greedy Best-First-Search and more formal approaches like Dijsktra's algorithm. Some advantages of this search is that it will always find a path if one exists and it is always the shortest path, however it does require memory for it saves the nodes in an Open and Close list. A* algorithm isn't good to use in the context of having more than one goal node.

2.1 How A* Works

At the start, it checks what the start node can move to and then it checks which node it should move to by using the heuristic function. It determines which node is the best one to move to by calculating each node's total cost which is G + H. G being the total distance to that specific node from the start and H being the total distance from that specific node to the goal node. So when it is time to move from the start node, it checks which possible node has the least cost, and then moves to that node. This process is repeated until the goal node has been reached.

3 Iterative Deepening A*

First described by Richard Korf in 1985, this algorithm is a variant of the iterative deepening depth-first search, however it is based off the A* algorithm in the sense that it uses heuristic approaches to help speed up its process. Given it is a depth-first search, this means that it uses less memory than the A* algorithm for it doesn't save visited nodes, but unlike other iterative deepening searches, it focuses on nodes with the least cost and therefore doesn't have equal depth everywhere along the search path (tree). On top of this, it isn't dynamic like the A* algorithm for it doesn't save the visited nodes so it often ends up evaluating the same nodes numerous times. One last point is that this needs more processing power and time than A* algorithm.

3.1 How IDA* Works

Every iteration performs a depth-first search, but it cuts off a branch when its cost (which is G + H) goes over a given "threshold". The threshold starts off as the cost of the start node, but increases for each iteration. The way the threshold increases depends on the cost of all the nodes, so the node with the minimum cost becomes the threshold for the next iteration. This process is repeated till the goal node has been reached.

4 Dijkstra Algorithm

published by Edsger W. Dijkstra in 1959, this algorithm uses weighted graphs to determine the shortest distance between the start and goal nodes. It finds the the shortest path between the start node and every other node in the graph and produces a tree with the shortest distance to the goal node if given a goal node. While this is good for certain contexts, this takes more time than A* algorithm for it processes every node in the weighted graph till it finds the goal node. This algorithm is an uninformed algorithm in the sense that it does not know what the goal node is beforehand for it checks every node till it finds the goal node with no heuristic function. While A* algorithm has two cost functions (G + H), Dijkstra only has one which is G (the distance from the start node to that specific node). This can be good when you have multiple goal nodes and you do not know which is the closest.

4.1 How Dijkstra Works

It starts at the start node with the G value of 0, for there is zero distance at the moment. Then it calculates the G costs for each neighbouring node and the start node is added to the unvisited queue. Next, it removes the start node from the unvisited queue and marks it as visited. The condition for each of its neighbouring node then is to check if that neighbour node (and their neighbouring node) is contained in the unvisited queue and, if so, is it leading to a better path? If this is the case then the costs for the neighbour node lessen (by checking the G costs up until that node), otherwise we check if this neighbour node is already visited. If it has not been visited, it is then then marked as visited and removed from the unvisited queue. This process repeats till every node has been marked as visited or until the goal node is marked as visited.

Instructions to start program:

Extract the zip file that says READY EXECUTABLE and make sure the cav files are within the same directory as the executable jar file. When you are done extracting just click on the executable jar file to start the program.

```
package Business;


import java.util.ArrayList;
import java.util.Collections;
import Data.*;

public class AStarCaves
{

    private SortedCaveList open = new SortedCaveList();
    private SortedCaveList closed = new SortedCaveList();

    private boolean flag = true;
    private boolean complete = false;

    private Cave startCave;
    private Cave endCave;
    private Cave currentCave;


    public AStarCaves()
    {
    }

    public void setStartAndEndCave(Cave startCave, Cave endCave)
    {
        this.startCave = startCave;
        this.endCave = endCave;
        open.emptyList();
        closed.emptyList();
        open.add(startCave);
        complete = false;
    }


    public ArrayList<Cave> generateSolutionPath()
    {
        return aStarsHelper();
    }


    public ArrayList<Cave> aStarsHelper()
    {
        Cave tempEndCave = endCave;

            while (this.flag)
            {
                currentCave = open.pop();


                if (currentCave.equals(tempEndCave))
                {
                    closed.emptyList();
                    open.emptyList();

                    return printSolutionPath(tempEndCave);


                }

                closed.add(currentCave);

                ArrayList<Cave> children = currentCave.getChildren();

                checkChildren(children);
            }
```

```java
      return null;
    }

    public void checkChildren(ArrayList<Cave> children)
    {
        Cave tempEndCave = endCave;

        for (Cave currentChild : children)
        {
            if (!closed.contains(currentChild))
            {
                if (!open.contains(currentChild))
                {
                    currentChild.setParentCave(currentCave);
                    currentChild.setHeuristicCostToGoal(tempEndCave);
                    currentChild.setGCostsTo(currentCave);
                    open.add(currentChild);
                }
                else
                {
                    if (currentChild.getGCosts() >
currentChild.calculateGCostsTo(currentCave))
                    {
                        currentChild.setParentCave(currentCave);
                        currentChild.setGCostsTo(currentCave);
                    }
                }
            }
        }
    }



    private ArrayList<Cave> printSolutionPath(Cave endCave)
    {

        Cave solutionCave = endCave;
        ArrayList<Cave> solutionPath = new ArrayList<>();

        while (solutionCave != null)
        {

            solutionPath.add(solutionCave);
            solutionCave = solutionCave.getParentCave();
        }

        Collections.reverse(solutionPath);

        return  solutionPath;
    }

}
```

Cave Class below and AStarCaves class above ^:

```java
package Business;

import javafx.beans.property.SimpleStringProperty;
import javafx.beans.property.StringProperty;


import java.util.ArrayList;

public class Cave
{

    private int x;
    private int y;
    private int caveID;

    private StringProperty caveName;
    private double g;
    private double heuristic;

    private Cave parentCave;
    private ArrayList<Cave> children = new ArrayList<>();

    public Cave(int x, int y, int caveID)
    {
        this.x = x;
        this.y = y;
        this.caveID = caveID;
        caveName = new SimpleStringProperty();
        updateCaveData();

    }

    private void updateCaveData()
    {
        caveName.setValue("Cave ID: " + Integer.toString(caveID)  + " G: " +
String.format("%.3f", this.g) + " Heuristic: " + String.format("%.3f",
this.heuristic));
    }

    public double getGCosts()
    {
        return g;
    }

    public StringProperty getCaveName() {
        return caveName;
    }

    public double calculateGCostsTo(Cave e)
    {
        double distance = Math.hypot(this.getX() - e.getX(), this.getY() -
e.getY());
        return (e.g + distance);
    }

    public void setGCostsTo(Cave e)
    {
        double distance = Math.hypot(this.getX() - e.getX(), this.getY() -
e.getY());

        this.g = (e.g + distance);
    }

    public void setHeuristicCostToGoal(Cave e)
    {
        double distance = Math.hypot(this.getX() - e.getX(), this.getY() -
e.getY());
        this.heuristic = distance;
    }

    public double getTotalCost()
    {
        updateCaveData();
        return g + heuristic;
    }

    public int getX()
    {
        return x;
    }
```

```java
    public int getY()
    {
        return y;
    }

    public int getCaveID()
    {
        return caveID;
    }


    public void setParentCave(Cave parentCave)
    {
        this.parentCave = parentCave;

    }

    public Cave getParentCave()
    {
        return parentCave;
    }

    public void addChildCave(Cave e)
    {
        this.children.add(e);
    }

    public ArrayList<Cave> getChildren()
    {
        return children;
    }

    @Override
    public String toString() {
        return "Cave{" +
                "x=" + x +
                ", y=" + y +
                ", caveID=" + caveID +
                '}';
    }

    public boolean equals(Cave e)
    {

        if(e == null)
        {
            return false;
        }


        if(this.x == e.getX() && this.y == e.getY() && this.caveID == e.caveID)
        {
            return true;
        }
        else
        {
            return false;
        }
    }
}
```

cavParser class below:

```java
package Data;

import Business.Cave;
import java.io.BufferedReader;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;
import java.util.ArrayList;

public class cavParser
{

    private static cavParser ourInstance;
    private static String fileName;

    private cavParser()
    {
    }

    public static cavParser getInstance(String file)
    {

        if(ourInstance == null)
        {
            ourInstance = new cavParser();
        }
        fileName = file;
        return ourInstance;
    }



    public boolean[][] buildConnectivityMatrix(int noOfCaves, String[] data) throws
FileNotFoundException, IOException
    {

        //Build connectivity matrix

        //Declare the array
        boolean[][] connected = new boolean[noOfCaves][];

        for (int row= 0; row < noOfCaves; row++){
            connected[row] = new boolean[noOfCaves];
        }
        //Now read in the data - the starting point in the array is after the
coordinates
        int col = 0;
        int row = 0;

        for (int point = (noOfCaves*2)+1 ; point < data.length; point++){
            //Work through the array

            if (data[point].equals("1"))
                connected[row][col] = true;
            else
                connected[row][col] = false;

            row++;
            if (row == noOfCaves){
                row=0;
                col++;
            }
        }
        //Connected now has the adjacency matrix within it
        return connected;
    }

    public ArrayList<Cave> buildCaves() throws FileNotFoundException, IOException
    {
        // Open input.cav
        ArrayList<Cave> caves = new ArrayList<>();

        BufferedReader br = new BufferedReader(new FileReader(fileName));

        //Read the line of comma separated text from the file
        String buffer = br.readLine();
      //  System.out.println("Raw data : " + buffer);

        br.close();

        //Convert the data to an array
        String[] data = buffer.split(",");

        //Now extract data from the array - note that we need to convert from
String to int as we go
        int noOfCaves = Integer.parseInt(data[0]);
        System.out.println ("There are " + noOfCaves + " caves.");

        int caveID = 1;
        //Get coordinates

        for (int count = 1; count < ((noOfCaves*2)+1); count=count+2){
            //System.out.println("Cave at " + data[count] +"," +data[count+1]);
            int x = Integer.parseInt(data[count]);
            int y = Integer.parseInt(data[count + 1]);

            Cave cave = new Cave(x, y, caveID);

            caves.add(cave);
            caveID++;
        }

        boolean[][] connectedMatrix = this.buildConnectivityMatrix(noOfCaves,
data);

        generateChildrenForCaves(caves, connectedMatrix);


        return caves;
    }

    public void generateChildrenForCaves(ArrayList<Cave> caves, boolean[][]
connectedMatrix)
    {

        for(int i = 0; i < caves.size(); i++)
        {
            Cave tempCave = caves.get(i);
           // System.out.println("current cave: " +tempCave.toString());
            for(int j = 0; j < caves.size(); j++)
            {
                Cave possibleChild = caves.get(j);
                boolean areConnected = connectedMatrix[tempCave.getCaveID() -
1][possibleChild.getCaveID() - 1];

                if(!tempCave.equals(possibleChild) && areConnected)
                {
                   // System.out.println("child: " + possibleChild.toString());
                    tempCave.addChildCave(possibleChild);
                }
            }
        }
    }

}
```

sortedCaveList class below:

```java
package Data;

import Business.Cave;

import java.util.Comparator;
import java.util.PriorityQueue;

public class SortedCaveList
{
    PriorityQueue<Cave> priorityQueue;

    public SortedCaveList()
    {
        priorityQueue = new PriorityQueue<>(new Comparator<>()
        {
            @Override
            public int compare(Cave o1, Cave o2) {

                return (int)o1.getTotalCost() - (int)o2.getTotalCost();
            }
        });
    }

    public void add(Cave e)
    {
       priorityQueue.add(e);
    }

    public boolean contains(Cave e)
    {
        if(priorityQueue.contains(e))
        {
            return true;
        }

        return false;
    }

    public Cave pop()
    {
        Cave e = priorityQueue.poll();
        return  e;
    }

    public void emptyList()
    {
        while (priorityQueue.size() != 0)
        {
            priorityQueue.poll();
        }
    }
}
```

```java
Controller class below: package Presentation;

import Business.AStarCaves;
import Business.Cave;
import Data.cavParser;
import javafx.application.Application;
import javafx.collections.FXCollections;
import javafx.collections.ObservableList;
import javafx.collections.ObservableListBase;
import javafx.fxml.FXML;
import javafx.geometry.Point2D;
import javafx.scene.Group;
import javafx.scene.canvas.Canvas;
import javafx.scene.canvas.GraphicsContext;
import javafx.scene.chart.*;
import javafx.scene.control.*;
import javafx.scene.control.Button;
import javafx.scene.control.Label;
import javafx.scene.layout.Pane;
import javafx.scene.layout.StackPane;
import javafx.scene.paint.Color;
import javafx.scene.shape.Circle;
import javafx.scene.shape.Line;
import javafx.scene.text.Font;


import java.io.FileNotFoundException;
import java.io.IOException;
import java.util.ArrayList;


public class Controller
{

    @FXML
    private TableView<Cave> openTable;


    @FXML
    private TableView<Cave> solutionTable;

    @FXML
    private TableColumn<Cave, String> childrenList;


    @FXML
    private TableColumn<Cave, String> solutionList;

    @FXML
    private Pane pane;

    @FXML
    private Label lblTotalScore;

    @FXML
    private ChoiceBox<String> cavFiles;

    @FXML
    private Label lblCurrentCave;


    private String input;

    private ArrayList<Cave> caves;

    private ArrayList<Cave> solutionPath;

    private Cave startCave;

    private Cave endCave;

    private int i;

    private AStarCaves aStarCaves;

    private ObservableList<Cave> openCaveData =
FXCollections.observableArrayList();
```

```
    private ObservableList<Cave> solutionCaveData =
FXCollections.observableArrayList();


    public Controller()
    {
    }

    @FXML
    private void initialize()
    {
        aStarCaves = new AStarCaves();
        cavFiles.setItems(FXCollections.observableArrayList(
                "Cave files 1", "Cave files 2", "Cave files 3"));


    }


    public void loadCavFile()  throws FileNotFoundException, IOException
    {

        try
        {

            pane.getChildren().removeAll(pane.getChildren());

            solutionCaveData.removeAll(solutionCaveData);
            openCaveData.removeAll(openCaveData);

            lblTotalScore.setText("");
            lblCurrentCave.setText("");

            input = cavFiles.getValue();

            if(input.equals("Cave files 1"))
            {
                input = "input1.cav";
            }
            else if(input.equals("Cave files 2"))
            {
                input = "input.cav";
            }
            else if(input.equals("Cave files 3"))
            {
                input = "input3.cav";
            }
            else
            {
               throw new Exception("Select Cav file!");
            }


            cavParser data = cavParser.getInstance(input);

            caves = data.buildCaves();

            startCave = caves.get(0);

            endCave = caves.get(caves.size() - 1);

            aStarCaves.setStartAndEndCave(startCave, endCave);

            solutionPath = aStarCaves.generateSolutionPath();


            for(int i = 0; i < caves.size(); i++)
```

```java
            {

                Cave e = caves.get(i);

                if(e.equals(startCave))
                {
                    Label startCaveLabel = new Label("START CAVE");
                    pane.getChildren().add(startCaveLabel);
                    startCaveLabel.relocate((pane.getPrefWidth() / 2) + ((e.getX()
* 10) - 30), (pane.getPrefHeight() / 2) + ((e.getY() * 10) - 5));
                    startCaveLabel.rotateProperty().setValue(180);


                }

                if(e.equals(endCave))
                {
                    Label endCaveLabel = new Label("END CAVE");
                    pane.getChildren().add(endCaveLabel);
                    endCaveLabel.relocate((pane.getPrefWidth() / 2) + ((e.getX() *
10) - 30), (pane.getPrefHeight() / 2) + ((e.getY() * 10) - 5));
                    endCaveLabel.rotateProperty().setValue(180);

                }


                Circle circle = new Circle(4, Color.GREEN);
                circle.setCenterX((pane.getPrefWidth() / 2) + (e.getX() * 10));
                circle.setCenterY((pane.getPrefHeight() / 2) + (e.getY() * 10));

                pane.getChildren().add(circle);


                generateLines(e);

            }

            i = 0;
            aStarCaves.setStartAndEndCave(startCave, endCave);


        }
        catch (Exception ee)
        {
            System.out.println(ee.fillInStackTrace());
           Alert alert = new Alert(Alert.AlertType.INFORMATION);
           alert.setTitle("Cave Files");
           alert.setHeaderText("Cave Files Error!");
           alert.setContentText("Must select a cave file first!");
           alert.showAndWait();
        }
    }


    public void generateLines(Cave e)
    {
        for(Cave ee : e.getChildren())
        {
            double currentCaveX = (pane.getPrefWidth() / 2) + (e.getX() * 10);
            double currentCaveY = (pane.getPrefHeight() / 2) + (e.getY() * 10);

            double currentChildX = (pane.getPrefWidth() / 2) + (ee.getX() * 10);
            double currentChildY = (pane.getPrefHeight() / 2) + (ee.getY() * 10);

            Line line = new Line(currentCaveX, currentCaveY, currentChildX,
currentChildY);

```

```
            Point2D start = new Point2D(currentCaveX, currentCaveY);


            double midPointX = start.midpoint(currentChildX, currentChildY).getX();

            double midPointY = start.midpoint(currentChildX, currentChildY).getY();


            Line connectedLine = new Line(midPointX, midPointY, currentChildX,
currentChildY);
            connectedLine.setStroke(Color.PURPLE);
            connectedLine.setStrokeWidth(3);


            pane.getChildren().add(connectedLine);
            pane.getChildren().add(line);
        }
    }

    public void stepThrough()
    {
        solutionPath = aStarCaves.generateSolutionPath();
        openCaveData.removeAll(openCaveData);

        if((i < solutionPath.size()))
        {

            Circle circle = new Circle(4, Color.RED);


            circle.setCenterX((pane.getPrefWidth() / 2) +
(solutionPath.get(i).getX() * 10));
            circle.setCenterY((pane.getPrefHeight() / 2) +
(solutionPath.get(i).getY() * 10));

            if(!(i + 1 > solutionPath.size() - 1))
            {
                double currentCaveX = (pane.getPrefWidth() / 2) +
(solutionPath.get(i).getX() * 10);
                double currentCaveY = (pane.getPrefHeight() / 2) +
(solutionPath.get(i).getY() * 10);

                double nextCaveX = (pane.getPrefWidth() / 2) + (solutionPath.get(i
+ 1).getX() * 10);
                double nextCaveY = (pane.getPrefHeight() / 2) + (solutionPath.get(i
+ 1).getY() * 10);

                Line redLine = new Line(currentCaveX, currentCaveY, nextCaveX,
nextCaveY);
                redLine.setStroke(Color.RED);

                if(pane.getChildren().contains(redLine))
                {
                    pane.getChildren().remove(redLine);
                }

                Point2D currentCave = new Point2D(currentCaveX, currentCaveY);


                double CurrentCaveMidPointX = currentCave.midpoint(nextCaveX,
nextCaveY).getX();

                double CurrentCaveMidPointY = currentCave.midpoint(nextCaveX,
nextCaveY).getY();


                Line currentCaveConnectedLine = new Line(CurrentCaveMidPointX,
CurrentCaveMidPointY, nextCaveX, nextCaveY);

                Cave currentTempCave = solutionPath.get(i);




                if(pane.getChildren().contains(currentCaveConnectedLine))
                {

                    pane.getChildren().remove(currentCaveConnectedLine);


                }
                currentCaveConnectedLine.setStroke(Color.RED);
                currentCaveConnectedLine.setStrokeWidth(3);


                pane.getChildren().add(currentCaveConnectedLine);
                pane.getChildren().add(redLine);

                if(solutionPath.get(i + 1).getChildren().contains(currentTempCave))
                {

                    Point2D nextCave = new Point2D(nextCaveX, nextCaveY);


                    double nextCaveMidPointX = nextCave.midpoint(nextCaveX,
nextCaveY).getX();

                    double nextCaveMidPointY = nextCave.midpoint(nextCaveX,
nextCaveY).getY();

                    Line nextCaveConnectedLine = new Line(nextCaveMidPointX,
nextCaveMidPointY, currentCaveX, currentCaveY);

                    if(pane.getChildren().contains(nextCaveConnectedLine))
                    {
                        pane.getChildren().remove(nextCaveConnectedLine);
                    }

                    nextCaveConnectedLine.setStroke(Color.RED);
                    nextCaveConnectedLine.setStrokeWidth(3);

                    pane.getChildren().add(nextCaveConnectedLine);

                }

            }

            Circle tempCave = circle;

            if(pane.getChildren().contains(tempCave))
            {
                pane.getChildren().remove(tempCave);
            }

            pane.getChildren().add(circle);


lblTotalScore.setText(Double.toString(solutionPath.get(i).getTotalCost()));
            lblCurrentCave.setText(solutionPath.get(i).toString());
            solutionCaveData.add(solutionPath.get(i));
            solutionTable.setItems(solutionCaveData);
            solutionList.setCellValueFactory(cellData ->
cellData.getValue().getCaveName());

            openCaveData.addAll(solutionPath.get(i).getChildren());
            openTable.setItems(openCaveData);
            childrenList.setCellValueFactory(cellData ->
cellData.getValue().getCaveName());
        }
        i++;
        aStarCaves.setStartAndEndCave(startCave, endCave);
    }

    public void stepThroughAll()
    {

        for(int i = 0; i < solutionPath.size(); i++)
        {
            stepThrough();
        }
    }

}
```

Main class below:

```
package Presentation;

import javafx.application.Application;
import javafx.fxml.FXMLLoader;
import javafx.scene.Parent;
import javafx.scene.Scene;
import javafx.stage.Stage;

import java.io.FileNotFoundException;
import java.io.IOException;

public class Main extends Application {

    @Override
    public void start(Stage primaryStage) throws Exception{
        FXMLLoader loader = new FXMLLoader(getClass().getResource("sample.fxml"));
        Parent root = loader.load();
        primaryStage.setTitle("A Star Caves");
        primaryStage.setScene(new Scene(root, 1200, 500));
        primaryStage.show();

        Controller controller = loader.getController();


    }


    public static void main(String[] args) throws FileNotFoundException,
IOException
    {
        launch(args);


    }



}
```

sample.fxml file below:

```xml
<?xml version="1.0" encoding="UTF-8"?>

<?import javafx.scene.*?>
<?import javafx.embed.swing.*?>
<?import javafx.scene.canvas.*?>
<?import javafx.scene.chart.*?>
<?import javafx.scene.shape.*?>
<?import javafx.scene.control.*?>
<?import java.lang.*?>
<?import javafx.scene.layout.*?>
<?import javafx.geometry.Insets?>
<?import javafx.scene.layout.GridPane?>
<?import javafx.scene.control.Button?>
<?import javafx.scene.control.Label?>

<AnchorPane maxHeight="-Infinity" maxWidth="-Infinity" minHeight="-Infinity"
minWidth="-Infinity" prefHeight="500.0" prefWidth="1200.0"
xmlns="http://javafx.com/javafx/8" xmlns:fx="http://javafx.com/fxml/1"
fx:controller="Presentation.Controller">
   <children>
      <SplitPane dividerPositions="0.49165275459098495" layoutY="81.0" maxHeight="-
Infinity" maxWidth="-Infinity" minHeight="-Infinity" minWidth="-Infinity"
prefHeight="500.0" prefWidth="1370.0" AnchorPane.bottomAnchor="0.0"
AnchorPane.leftAnchor="0.0" AnchorPane.rightAnchor="0.0"
AnchorPane.topAnchor="0.0">
        <items>
          <AnchorPane maxHeight="-Infinity" maxWidth="-Infinity" minHeight="0.0"
minWidth="0.0" prefHeight="498.0" prefWidth="664.0">
               <children>
                  <Label layoutX="14.0" layoutY="14.0" prefHeight="59.0"
prefWidth="112.0" text="Select Cav File" AnchorPane.leftAnchor="14.0"
AnchorPane.topAnchor="14.0" />
                  <ChoiceBox fx:id="cavFiles" layoutX="291.0" layoutY="11.0"
prefHeight="39.0" prefWidth="128.0" />
                  <Button fx:id="btnLoad" layoutX="428.0" layoutY="11.0"
mnemonicParsing="false" onAction="#loadCavFile" prefHeight="39.0" prefWidth="139.0"
text="Load/Reset" />
                  <Button fx:id="btnStep" layoutX="428.0" layoutY="228.0"
mnemonicParsing="false" onAction="#stepThrough" prefHeight="39.0" prefWidth="139.0"
text="Step" />
                  <Label layoutX="14.0" layoutY="125.0" text="Current Cave: " />
                  <Label fx:id="lblCurrentCave" layoutX="235.0" layoutY="125.0" />
                  <Label layoutX="11.0" layoutY="183.0" text="Total Distance:" />
                  <Label fx:id="lblTotalScore" layoutX="239.0" layoutY="183.0"
maxHeight="-Infinity" maxWidth="-Infinity" minHeight="-Infinity" minWidth="-
Infinity" />
                  <TableView fx:id="openTable" layoutX="-1.0" layoutY="300.0"
prefHeight="200.0" prefWidth="322.0">
                    <columns>
                      <TableColumn fx:id="childrenList" prefWidth="321.0"
text="Current Cave's Children" />
                    </columns>
                  </TableView>
                  <TableView fx:id="solutionTable" layoutX="322.0" layoutY="300.0"
prefHeight="200.0" prefWidth="272.0">
                    <columns>
                      <TableColumn fx:id="solutionList" prefWidth="271.0"
text="Solution Path" />
                    </columns>
                  </TableView>
                  <Button fx:id="btnStepAll" layoutX="280.0" layoutY="228.0"
mnemonicParsing="false" onAction="#stepThroughAll" prefHeight="39.0"
prefWidth="139.0" text="Step Through All" />
               </children>
            </AnchorPane>
            <Pane fx:id="pane" maxHeight="-Infinity" maxWidth="-Infinity"
minHeight="-Infinity" minWidth="-Infinity" prefHeight="500.0" prefWidth="601.0"
rotate="-180.0">
```

```
               <children>
                  <Label layoutX="224.0" layoutY="41.0" rotate="180.0" text="Purple
lines indicate connection" />
               </children></Pane>
        </items>
      </SplitPane>
   </children>
</AnchorPane>
```