§CS 520: Assignment 1
Fast Trajectory Replanning

Xiaoyang Xie
167008240

Yikun Xian
168000142

Department of Computer Science
Rutgers University, New Brunswick, NJ

09 Octorber 2015

Abstract

Heuristic search algorithms like A* can be adapted to solving path planning problems of directed goal and unknown environment In this report, we mainly discuss and evaluate three variants of A* algorithms, namely Repeated Forward A*, Repeated Backward A* and Adaptive A*. In the experiment, we first generate two sets of 1000 random grid-based maps, where one follows the assignment requirement containing approximate 30% obstacles and 70% roads, and the other is the set of corridor-like mazes generated by DFS with randomly expanded nodes. Then we compare three algorithms by such evaluation indices as number of expanded nodes, number of explored nodes, total cost of steps, optimal cost of steps, etc. The result shows that 1) the average number of expanded nodes per map and explored nodes per map of Adaptive A* are respectively 28.06% and 24.00% less than those of Repeated Forward A*; 2) cost of steps for all three algorithms is pretty much the same. This indicates that Adaptive A* is greatly optimized in path replanning phase, while there is no significant improvement in actual moving phase. Finally, we discuss and calculate how to optimize data structures to store states as many as possible within only 4M memory. This is the practical problem in the situation where computational resources are rare and precious.

Part 0 Setup Environment

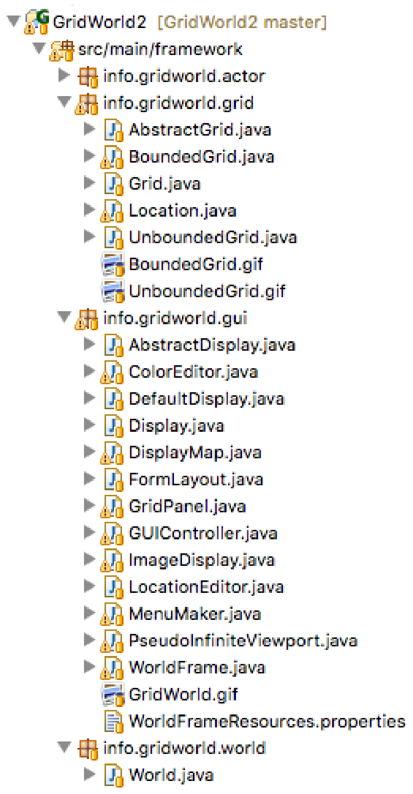
We simulate all path finding processes based on the framework of GridWorld[1], an AP case study project from collegeboard [1]. It provides graphical user interface based on Java AWT where visual objects can interact and perform customized actions in a two-dimensional grid map. In the next part, we will first illustrate original GridWorld framework and our enhancement of displaying colored path. This mainly involves the engineering work, so if you want to directly delve into algorithm analysis, please skip it.

0.1 GridWorld Architecture and Modification

The source code of original GridWorld project is placed in *src/main/framework* folder and its structure can be divided into four parts, as shown in Figure 1a.

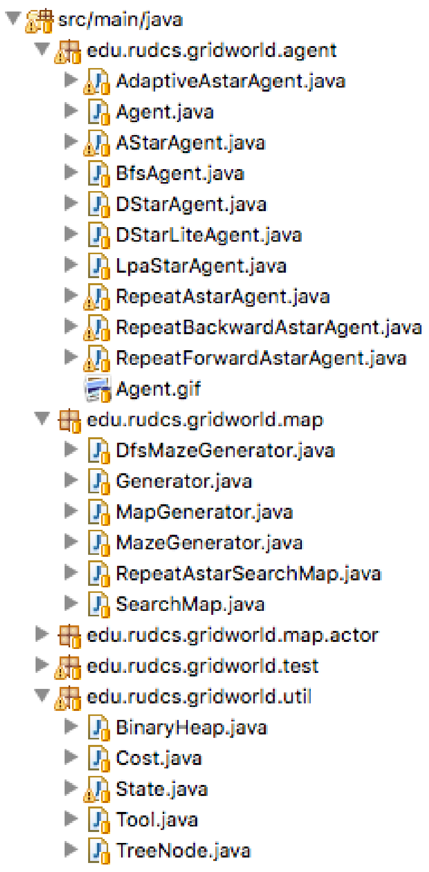
The *actor* package contains objects whose behavior on the map can be arbitrarily defined by rewriting *act* method in each inherited class:

[1]https://www.collegeboard.org/



```
▼ GridWorld2 [GridWorld2 master]
  ▼ src/main/framework
    ▶ info.gridworld.actor
    ▼ info.gridworld.grid
      ▶ AbstractGrid.java
      ▶ BoundedGrid.java
      ▶ Grid.java
      ▶ Location.java
      ▶ UnboundedGrid.java
        BoundedGrid.gif
        UnboundedGrid.gif
    ▼ info.gridworld.gui
      ▶ AbstractDisplay.java
      ▶ ColorEditor.java
      ▶ DefaultDisplay.java
      ▶ Display.java
      ▶ DisplayMap.java
      ▶ FormLayout.java
      ▶ GridPanel.java
      ▶ GUIController.java
      ▶ ImageDisplay.java
      ▶ LocationEditor.java
      ▶ MenuMaker.java
      ▶ PseudoInfiniteViewport.java
      ▶ WorldFrame.java
        GridWorld.gif
        WorldFrameResources.properties
    ▼ info.gridworld.world
      ▶ World.java
```

(a) GridWorld structure



```
▼ src/main/java
  ▼ edu.rudcs.gridworld.agent
    ▶ AdaptiveAstarAgent.java
    ▶ Agent.java
    ▶ AStarAgent.java
    ▶ BfsAgent.java
    ▶ DStarAgent.java
    ▶ DStarLiteAgent.java
    ▶ LpaStarAgent.java
    ▶ RepeatAstarAgent.java
    ▶ RepeatBackwardAstarAgent.java
    ▶ RepeatForwardAstarAgent.java
      Agent.gif
  ▼ edu.rudcs.gridworld.map
    ▶ DfsMazeGenerator.java
    ▶ Generator.java
    ▶ MapGenerator.java
    ▶ MazeGenerator.java
    ▶ RepeatAstarSearchMap.java
    ▶ SearchMap.java
  ▶ edu.rudcs.gridworld.map.actor
  ▶ edu.rudcs.gridworld.test
  ▼ edu.rudcs.gridworld.util
    ▶ BinaryHeap.java
    ▶ Cost.java
    ▶ State.java
    ▶ Tool.java
    ▶ TreeNode.java
```

(b) Project structure

Figure 1: Structure of GridWorld framework and path-finding project

```
@Override
public void act()
```

The *grid* package defines features on bounded and unbounded grid map as well as connections between map and actors. The *gui* package encapsulates low-level Java AWT to provide APIs for visualization and interactions of map and actors. The *world* package provides high-level integration of actors and world.

In order to visualize the presumed unblocked path and the set of nodes expanded and explored after each planning, we enhance the *GridPanel* class in the GUI package by implementing the method below:

```
private void drawColoredLocations(Graphics2D g2)
```

Meanwhile, following five abstract methods related to colored grid are added to *Grid* interface and classes like *BoundedGrid* and *UnboundedGrid* are responsible to implement color configuration on each grid.

```java
ArrayList<Location> getColoredLocations();
Color getColor(Location loc);
void putColor(Location loc, Color color);
void removeColor(Location loc);
void resetColors();
```

The source code related to assignment is placed in *src/main/java* folder, as shown in Figure 1b, which are divided into five packages. The *agent* package defines the simulated agents equipped with specific navigation algorithm for solving goal-directed path-finding problem. The *map* package provides APIs for generation of random map and corridor-like maze. The *map.actor* package defines some static object on the map like obstacles and goal. The *test* package contains simulation program for experiment and testing. The *util* package includes some self-defined data structures to be used in storing intermediate result of algorithm.

0.2 How to Run

Our project is built and packed by Gradle[2], an open-source build automation tool. First, you should install and setup Gradle environment on your computer, and install Gradle plugin for Eclipse, named "Gradle Integration for Eclipse". Then, download complete source code of the project, which is attached in Sakai or available on the Github[3]. Finally, inside Eclipse, click *File* → *Import* → *Gradle* → *Gradle Project* to import project. All runnable programs are placed in *test* package.

[2]http://gradle.org/

[3]https://github.com/orcax/GridWorld2

0.3 Maze Generation Algorithm

Part 1 Understanding the Methods

Part 2 The Effects of Ties

Part 3 Forward vs. Backward

Part 4 Heuristics in the Adaptive A*

Part 5 Heuristics in the Adaptive A*

Part 6 Memory Issues

References

[1] CollegeBoard. AP central gridworld case study. http://apcentral.collegeboard.com/apc/public/courses/teachers_corner/151155.html. [Online; accessed 2-October-2015].

[2] Sven Koenig and Maxim Likhachev. Real-time adaptive a*. In *Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems*, pages 281–288. ACM, 2006.

[3] Wikipedia. Maze generation algorithm — wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Maze_generation_algorithm&oldid=679876968. [Online; accessed 4-October-2015].