EECS 391: Introduction to AI (Spring 2020) Programming Exercise 2

General Instructions:

Please comment your code extensively so we can understand it, write efficient code using good data structures and sensible variable names. We will use the git logs to ensure that each person is contributing equally to all exercises. Therefore, any code written by you must be clearly logged under your own name/network ID by git. You will not receive credit for code committed by anyone other than yourself even if you wrote it, or if we cannot make out who committed the code. **There will be NO exceptions to this rule.**

The redmine system used by csevcs has issues with some Unicode characters. Please ensure your git names use only standard ASCII characters.

Your commits are due on csevcs.case.edu by 11:59pm on the due date specified after the question. You will receive a 10% bonus for any solution turned in a week or more in advance of the due date. You must notify us of such a commit by email. You can use one late day each week (up to Saturday 11:59pm) with a penalty of 20%. Submissions after Saturday 11:59pm for any week will not be graded. Other bonus points may be awarded for exceptionally well-written and commented, easy to read, clean and efficient code.

These programming exercises will use SEPIA ("Strategy Engine for Programming Intelligent Agents"), a game environment written by other CWRU students tailored to writing AI players. SEPIA is a real-time strategy (RTS) game (though we will not use the real-time aspects in these exercises). RTS games generally have "economic" and "battle" components. In the economic game, the goal is to collect different types of resources in the map. Typical resources are "Gold" and "Wood." Resources are collected using units called "Peasants." Having resources allows the player to build other buildings (Peasants can be used to build things) and produce more units. Some of these units are battle units that can be used to fight the opponent. Games generally end when one player has no more units left; however, in SEPIA, a variety of victory conditions can be declared through XML configuration files. For example we can declare a victory condition to be when a certain amount of Gold and Wood have been collected, some number of units of a certain type built, etc.

You need Java 1.8 to run SEPIA. Note that, although the components of SEPIA you will use have been fairly well tested by now, there is still a possibility of bugs remaining. If you encounter behavior that seems strange, please let me or the TAs know.

Programming 2: Pathfinding (First Commit 2/14 (50 points), Final Commit 2/21 (50 points))

The data for this exercise is in the ProgrammingExercise2.zip file on Canvas.

Write an agent that can move around a given map by implement the A* search algorithm discussed in class. In the file AstarAgent.java, find the function AstarSearch.java and fill it in. This function should return a path from the starting location to the goal. The rest of the agent code has already been filled in so that once you implement the search, the agent will execute it in the game. During this step, it will output its progress with helpful messages that should let you debug your code. You can also watch VisualAgent (the GUI window showing the map) to see how your found path is being followed. The terminal output will also show the total time taken by the process. You should try to reduce this as much as possible (i.e. write efficient code!). For a proper estimate of the time taken, you can stop VisualAgent from running by deleting or commenting out the corresponding <agentclass> lines from the configuration file.

For A*, you will need a heuristic function. The Chebyshev distance is a good heuristic for this purpose. This is defined as follows: $D((x1,y1),(x2,y2))=\max(|x2-x1| , |y2-y1|)$. To test your algorithm, use the maze maps provided. In each map, a Footman is trapped in a maze. Somewhere in the maze is an enemy Townhall. The provided code will use your implementation to find a path that takes the Footman to the Townhall and attack it. When the Townhall is destroyed, the game will end. If it is not possible to guide the Footman to the Townhall, print “No available path.” in the terminal and quit (call System.exit(0)).

You can use VisualAgent to check that your agent is behaving correctly. You can run the maze maps just using VisualAgent, left click on the Footman and right click on the Townhall to see the solution according to the built in pathfinding routines.

A* allows to you to do basic pathfinding, but in more complex scenarios, many units will be wandering around, and pathfinding is complicated. We will now simulate this using a simple scenario.

One of the provided maps (maze_16x16h_dynamic) is an environment with an enemy footman, controlled by the wicked EnemyBlockerAgent. This agent will try to prevent your agent from destroying the enemy Townhall by blocking their path (it won’t attack you otherwise). To get around this, use the “shouldReplanPath()” function in AstarAgent. Here, you can check to see if the current path is blocked. If so, you should return true and the agent will redo the search from that point. Try to write a nice function which is smart about when it needs to redo the search so you minimize the total time spent in searching and execution.

We may award up to 10 bonus points if (i) your code is clean, elegant and comprehensible and (ii) your total runtimes are among the top three in the class. Please do NOT use map-

specific or heuristic-specific optimizations in your A* implementation. This means your implementation should be able to handle any map and any heuristic.

Your git repository is located at https://csevcs.case.edu/git/2020_spring_391_N, where N is your Canvas group number. **Because this is a long assignment, you are expected to make intermediate weekly commits to the repository.** Each week, we expect to see substantial progress made in the implementation by each person. Progress must be *proportional to the points allocated that week*. Create a separate subdirectory "P2agents" in your git repository, place your agent in it and push to csevcs. Include a short README file containing (i) a summary of the work done for the week and (ii) your experience with the API and documentation, and anything you found confusing. Do **NOT** commit any files other than the java files containing your agent and the README.