EECS 391: Introduction to AI (Spring 2020) Programming Exercise 5

General Instructions:

This is an individual assignment. Each student must complete this assignment separately. As a result, we will no longer use csevcs for submission. Instead, after writing your agents, please submit the agent code as a zip file on canvas. As always, do not submit anything other than the agent code you wrote.

Please comment your code extensively so we can understand it, write efficient code using good data structures and sensible variable names.

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 5: Enhanced Automated Resource Collection (Due 4/10, 75 points)

The data for this exercise is in the ProgrammingExercise5.zip file on Canvas.

Use the midas*BuildPeasant config files for this assignment. This is a continuation of resource collection. Now suppose you have an additional action, BuildPeasant (the Townhall can execute this action). This action requires 400 gold and 1 food. The Townhall supplies 3 food and each peasant currently on the map consumes 1 food. If successful, this operator will deduct 400 gold from the current gold tally and result in one additional peasant on the map, which can be subsequently used to collect gold and wood. Since your planner is a simple state space planner, it only produces sequential plans, which will not benefit from the parallelism possible with multiple peasants. To solve this, define additional actions as follows. Write additional Move, Harvest and Deposit operators, Move_k, Harvest_k and Deposit_k, that need k peasants to execute and have the effect of k=1 to 3 parallel Moves, Harvests and Deposits, but will only add the cost of a single action to the plan. To execute such operations, your PEA should then find k "idle" peasants and allocate them to carrying out the Move_k, Harvest_k and Deposit_k operator by finding the nearest goldmine/forest/townhall to go to. Note that your PEA can further heuristically parallelize your found plans, though this reduction in cost cannot be accounted for by the planner. For example, with 3 peasants, suppose you have a Move_1(townhall,goldmine) and a Move_2(townhall,forest) in sequence. Your PEA can parallelize these actions to execute at the same time by noticing that their preconditions can be simultaneously satisfied. This sort of behavior by the execution agent falls under scheduling, a part of automated planning that we did not discuss in class. Be careful when writing heuristics for the BuildPeasant operator. Note that it has an immediate negative effect, i.e. it moves the plan farther from the goal. Somehow your heuristic needs to trade this off against the longer-term positive effect that the parallelism will allow.

(a) Set the goal state to be a gold tally of 1000 and a wood tally of 1000. Produce a plan and execute it in SEPIA. (b) Set the goal state to be a gold tally of 3000 and a wood tally of 2000. Produce a plan and execute it in SEPIA. In each case, output the total time taken to actually execute the plan found. As before, be careful not to "pre-plan" by using your knowledge of the game.

If you feel ambitious, think about how to incorporate an additional action, BuildFarm. This action creates a new Farm that supplies additional food, which can be used to build even more peasants. At this point, however, you will need a proper scheduler to handle the parallelized action dispatching at each time step.

We will award up to 10 bonus points for well written code that is able to quickly find a short plan so that the total runtime is fast relative to the rest of the class (e.g. if you are in the top

three runtimes to finish a scenario with a fixed large resource target). Note that we will test your code with other maps than the ones provided with this assignment.

As a reminder, you should no longer use csevcs. You should turn in your agents (only) in a zip file on canvas by 11:59pm on the due date. No partial intermediate submissions or READMEs are required (you are welcome to include a README if you did anything beyond what the exercise asks you, such as trying BuildFarm, to let us know your observations, but it is not required).