EECS 391 (Spring 2015) Programming Assignment 3 (Max Points: 100).

Assigned Thursday March 17, due midnight Tuesday March 31. Turn in your code using blackboard. Please comment your code extensively so we can understand it, and use sensible variable names. If two people worked on a submission, please include both your names and IDs in a README. You may only turn in an assignment as a pair if both of you have contributed equally to it.

In this assignment you will write a forward state space planner to solve resource collection scenarios in SEPIA.

1. Problem Setup

The scenarios you will solve are built around the "rc_3m5t.xml" map in Sepia and the "midas*" configuration files. In this map, there is a townhall, a peasant, three goldmines and five forests. Assume the peasant can only move between these locations. When the peasant is next to a goldmine, it can execute a HarvestGold operation. This requires the peasant to be carrying nothing and the goldmine to have some gold. If successful, it removes 100 gold from the goldmine and results in the peasant carrying 100 gold. The three goldmines in this map have capacities 100 (nearest to townhall), 500 and 5000 (farthest from townhall) respectively. When the peasant is next to a forest, it can execute a HarvestWood operation. This requires the peasant to be carrying nothing and the forest to have some wood. If successful, it removes 100 wood from the forest and results in the peasant carrying 100 wood. The five forests in this map each contain 400 wood. Finally, when the peasant is next to the townhall, it can perform a Deposit[Gold/Wood] operation. This requires the peasant to be carrying something. If successful, it results in the peasant being emptyhanded, and adds to the total quantity of gold or wood available by the amount carried by the peasant.

Note that this description goes beyond the STRIPS language in that we are describing numeric quantities. For this assignment, you should ignore this aspect and write a planner to handle this scenario assuming a "STRIPS-like" semantics according to the description above. Your code should consist of two parts: a planner and a plan execution agent (PEA). The planner will take as input the action specification above, the starting state and the goal and output a plan. The PEA will read in the plan and execute it in SEPIA.

Implement a forward state space planner using the A* algorithm that finds minimum makespan plans to achieve a given goal. Here makespan is time taken by the action sequence when executed. Most actions take unit time, but note that for some actions, such as compound moves, you will only be able to estimate the makespan---that is fine for this assignment. Make sure to design good heuristics for the planner to guide it towards good actions---note that the branching factor of this (simple) search space is quite large once you instantiate all the ground operators! Once a plan is found, write it out to a plain text file as a list of actions, one per line, along with any parameters. Your PEA can then execute this plan in SEPIA. (The PEA does not have to read the text file, you can just directly pass it the plan.)

In order to execute the found plans, you will have to translate the plan actions into SEPIA actions. Since you will be planning at a fairly high level, you may need to write some code to automatically choose target objects if needed so actions can execute properly. You are welcome to do this in a heuristic manner.

If at some point the plan read in by the PEA is not executable in the current state, it should terminate with an error. Else, if all actions could be executed, it should terminate with a “success” output.

2. Resource Collection 1 (40 points)

Use the midasSmall and midasLarge config files for this part of the assignment. Set the initial state to be: the peasant is emptyhanded, the gold and wood tallies are zero and the capacities of all mines and forests are as above. Write STRIPS-like descriptions of the actions. (a) Set the goal state to be a gold tally of 200 and a wood tally of 200. Produce a plan and execute it in Sepia. (b) 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. In each case, output the total number of steps taken to actually execute the plan.

3. Resource Collection 2 (60 points)

Use the midas*BuildPeasant config files for this part of the assignment. 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. One way to solve this is 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.

Write a STRIPS-like description of the BuildPeasant action. Use the same initial state as above. (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.

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.

4. Code and Data Structures

We are providing the following files and data structures.

There is an included StripsAction interface which has two functions. preconditionsMet(GameState) takes in a GameState and returns true if that state satisfies all of the preconditions of the action. apply(GameState) takes in a GameState and applies that actions effects, returning the resulting game state. You can use this to define different classes that implement actions like Move and Harvest if you like. This is similar to SEPIA's Action class, but specific to this assignment.

The GameState class is similar to the previous assignment. It is intended to capture the abstract state the planner reasons over, computed from SEPIA’s state.

The PlannerAgent class contains an empty AstarSearch function that takes in a GameState and returns a Stack of objects implementing the StripsAction Interface. The PlannerAgent includes a predefined method that writes the stack to a file. It calls the toString method on each Strips Action in the plan and writes the output to a line. The PlannerAgent also includes an instance of the PEAgent which is instantiated with the plan found by AstarSearch. There is a createSepiaAction function in PEAgent that takes in a StripsAction and returns a SEPIA Action where you will construct an implementable action corresponding to your plan actions.

The Position class abstracts the position of a unit.

5. What to turn in

Prepare a ZIP file with any java files you modified (maintaining the directory structure) and a README (do NOT include any class files, or any SEPIA code). Include a README with your name(s) and ID(s) and any other comments you have. Name your file as “yournetworkID(s)_PA3.zip” and use Blackboard to submit it. If you worked as a pair, ensure that both your IDs appear in your submission. Only one submission is required if you worked as a pair.