MONASH University

ASSIGNMENT 2 [20%]

Agent Planning

SUBMISSION DUE: Thursday Week 7, 11:55 PM

Learning Outcomes

Completion of this assessment item addresses competency in the following learning outcomes:

- Apply - through practice-based learning - design, development, execution and validation of real-time interactive software using AI techniques.
- Select, evaluate and apply AI and/or ALife software techniques to model simple intelligent behaviour in 2D discrete simulations and games.

Brief

For this assignment, you will be managing the planning component of a collection of agents to complete a number of objectives within a virtual environment. The agents and the environment will be provided within base code for the project, and you will focus on the implementation of a planning system and dynamic collision avoidance. The virtual environment will be fully observable to the agent navigating within a discrete world.

This assignment is based on the completion and implementation of Assignment 1. You should therefore aim to use your existing code as the base for this assignment. Alternatively, base code is provided by the teaching team with a Breadth First Search implementation for those who wish to use it.

Task Overview

The pirate ships (agents) must collect resources from surrounding islands to build up their home island within the virtual environment. There are 3 types of pirate ship, each assigned a different resource to collect: wood, stone or fruit. Each pirate ship must navigate the world and collect resources from surrounding islands, whilst ensuring they avoid collisions with other ships.

Pirate ships start with 100 morale points. This total decreases by 1 point for each action taken, such as moving 1 tile or collecting a resource. When morale reaches 0 points a mutiny occurs! The pirate ship stops all actions for the remainder of the simulation. Pirate ships can collect treasure to refresh their morale.

Your task is to implement a planning system for completing sequences of actions to ensure the pirate ships collect resources required for the home island whilst managing their individual morale. As part of this, the ships should not collide with other ships. This may require you to implement changes to the path finding algorithm.

Marks will be allocated for:

1. Appropriate algorithm choices for planning actions
2. Implementing ship collision avoidance with other ships
3. Correct collection of resources upon travelling to location
4. Appropriate documentation and coding standards

Detail on each of the requirements for points 1-4 above and the marks allocated to each component, appear in sections below.

You have been provided with base code that contains starter files to assist you in developing your pathfinding algorithms. The base code does the following:

1. Loads a map file located in the “mapfiles” folder to generate the world
2. Spawns a collection ship into the virtual environment at runtime
3. Spawns a collection treasure item into the virtual environment at runtime
4. Once a treasure has been collected another will be spawned until all treasures contained within the map file are exhausted
5. The ships each have a morale statistic/parameter to represent the crew's morale
6. An implementation of the Breadth First Search algorithm is included for pathfinding for those who wish to use it

The base code does not handle behaviour for consumption of treasure. This must be performed as part of your implementation of the pathfinding algorithm.

Extensions, Tasks and Steps

As an extended task you will be required to alter the code to include an additional pirate ship agent type, the *merchant*. The merchant agent must travel to islands to collect rum needed by other ships. Other ships can only complete their collection tasks if they have rum on board. The total rum barrel tally on a ship decreases by 1 whenever a collection task is completed (pirates drink a barrel of rum in celebration each time they complete a task).

Note: The extension tasks / steps will only be assessed provided the complete base functionality for all standard steps has been completed successfully.

Marks will be allocated for meeting the following criteria:
- Correct implementation of merchant agent’s actions and goals
- Alteration of existing agents to require the use of rum barrels for resource collection

Detailed task breakdown and mark allocation

1. Documentation and coding standards [20%]

a. All methods must be explained in comments within the code files **[5]**
b. A justification of your chosen algorithms including planning and collision avoidance. This must be included in a separate readme file. **[6]**
c. Variable names must be indicative of their purpose **[2]**
d. . You have a choice of style (eg. camelCase or PascalCase) however, consistency of naming conventions and code style is required **[5]**
e. Readability and code structure should be of a high quality (e.g. Indenting, classes, methods, separate files, header files, general neat formatting, etc.) **[2]**

2. Planning [30% + ext. 15%]

a. Step 1 **[2]**
Create a virtual new class to represent a single action that an agent can take within the virtual environment. This action class must include a map of preconditions, a

map of effects, an action cost, a pointer to a target actor. Include functions for adding/removing preconditions and effects.

b. Step 2 **[3]**
Create a number of subclasses of the action class created above to represent each action an agent can take. These should include actions for collecting a resource, dropping off a resource and collecting treasure.

c. Step 3 **[10]**
Create a new class that will calculate plans for agents based on the current world state, the desired world state and a set of available actions. This class should calculate the plan starting from the desired world state working backwards to find the fastest path towards the current world state.

d. Step 4 **[15]**
Modify the existing agent class to use a Finite State Machine with 4 states; Nothing, Idle, Move and Action. Each state should have an associated OnEnter, OnUpdate and OnExit function.

The **Idle State** should attempt to calculate a new plan where one does not exist. If a valid state is calculated it should transition to the Action State.

The **Move State** should handle movement from the agent's current position to the goal position (calculated in the Action State). This state should include the existing movement code from Assignment 1

The **Action State** should attempt to complete the current action as determined by the planner. If the action requires the agent to move to a specific location then a path should be calculated using the existing pathfinding algorithm before transitioning to the Move State. If there are no actions to complete, or the current action cannot be completed, then it should transition to the Idle State.

e. Extension Step 1 **[+5]**
Create new actions to support the new merchant agent. This should include the collection of rum barrels from island ports and depositing them on the home island.

f. Extension Step 2 **[+10]**
Modify the existing actions and agent class to require rum for completion of resource collection tasks. If rum barrels are not available, the agents should only collect treasure when morale is lower than 40

3. Collision Avoidance [20%]

a. Step 1 **[5]**
Update the agent's pathfinding algorithm to avoid spaces where other agents are currently located.

b. Step 2 **[15]**
Update the agent's movement functions to check for nearby agents that may collide with the current agent. Create an algorithm for avoiding collision between agents and allowing each to still execute their actions.

4. Resource Collection [15%]

a. Step 1 **[10]**
Add functionality to each of the Resource Collection Actions that takes resources from the island node and adds it to the agent's resources at a rate of 1 resource per second. Agents can hold a total of 50 resources.

b. Step 2 **[5]**
   Add functionality to the Resource Drop Off Action that takes resources from the agent and deposits them in the home island node at a rate of 2 resources per second.