Homework 3 - Games

Introduction

In this exercise, you will take the role of the head of one of two competing pirate fleets. You will collect and deposit as many treasures as you can, while your rival tries to do the same. Your goal is to beat them by getting a higher score. The harder you beat them the better!

Environment

The environment is like the previous exercises. The grid world is identical to the previous exercises. You still have sea tiles, Island tile and a Base tile.

Key differences:

- There are two separate pirate fleets, one for each player.
- The actions are taken in turns.
- The treasures are now unique. Once a treasure is collected, its location **becomes the ship.** Each treasure can only be picked up by a single ship. (Further details and clarifications below)
- New treasures will appear with some probability.
- The state now includes a key called “base” whose value is the base location.

There are some key features that remain from exercise 2:

- The agent is required to return an action, given a state.
- The marines behave like in exercise 2. (More on the marines later)
- The execution has a limited number of turns.

Actions

New atomic action: plunder.

This exercise has a new action called **plunder**. If your ship is at the same location as another ship, you can use plunder to sink all its treasure, like a collision with a marine. There is no reward for this action and the sole purpose is to interrupt your adversary. The syntax is the following: (“plunder”, “ship_1”, “ship_2”). This means that ship_1 is plundering ship_2.

The action **deposit** is now different from before. Because the treasures are now unique, you will have to state which treasure you are depositing. The new syntax for deposit is: **(“deposit”, “ship_name”, “treasure_name”).**

The following actions remain unchanged: (“sail”, “collect”, “wait”).

The actions “reset”, and “Terminate” are irrelevant to this exercise.

Notice: Now you can only deposit a single treasure at a time.

Dynamics of the system:

The game consists of two rounds. Each round then takes 100 turns.

At the start of the round, each player is assigned a fleet of pirates they control, the round is initialized, and then the turns take play. At each turn the first player takes an action, the resulting state is given to the second player, who then decides on its action. **After** both players have taken their actions, the marines move.

When the first round is done, the state is set back to the initial state. Then the players switch fleets, and again play for 100 turns.

The player who accumulates the most points after 2 rounds wins.

Pseudo-code example for game dynamics:

For 2 rounds:

1. For 100 rounds:
   1.1. Player1 plays a move.
   1.2. Add treasures to map with some probability.
   1.3. Player2 plays a move.
   1.4. Add treasures to map with some probability.
   1.5. Check collisions with marines.
   1.6. Move marines.
2. Switch players.

Treasures

As we established before the treasures are now unique. If before the location of the treasure did not change after pickup, now it is attached to the ship. After pickup the location of the treasure will be the ship name:

```
"treasures": {'treasure_1': {"location": "pirate_ship_1",
                             "reward": 4}
```

After the treasure is deposited, it is deleted from the state. If a pirate ship collided with a marine ship, all the treasures it collected will be deleted!

Each treasure gives a different reward upon being deposited. Each turn a treasure has the probability of appearing on some island tile.

Points:

Now each treasure gives a different reward upon depositing. This is the only way to obtain points. Collision with marines will take away points in the same way it did in HW2.

The task

Your task is to implement two agents:

1) UCT-based agent: this agent implements UCT as taught in class. The agent must use random rollouts. **Note**: the UCT tree node does not have to contain a state. It may just contain the actions the agent took to get there. **Further Clarification in the Clarifications section**.
2) A general agent: This agent can implement any algorithm you see fit to achieve the best results.

Code:

Your agents are to be implemented in exp3.py. For each agent you must implement:

- A constructor `def __init__(self, initial_state, player_number):`
  Takes in the initial state and an indication of whether you go first or second**. Must finish within 60 seconds.**
- Function `act(self, state):` takes the current state of the game and returns an action. **Must finish within 5 seconds.**
- For the UCT agent you must also implement:
  - `selection(self, UCT_tree)` - selects a tree node to expand
  - `expansion(self, UCT_tree, parent_node)` - updates the tree with a new node
  - `simulation(self)` - performs a simulation from a new node
  - `backpropagation(self, simulation_result)` - updates the UCT tree with new information
- Python libraries: You can use every library in the standard library. If you want to use another library, ask in the forum.

The code consists of 5 files:

- exp3.py. Contains your agents, the only file that should be modified.
- main.py. Runs the game, use it to check your results.
- simulator.py. Contains a simulator of the game. You may use it in your UCT agent.
- sample_agent.py. Contains an agent that uses random actions.

We advise you to read the documentation of functions.

Submission and grading:

You are to submit only the file named ex3_ID1_ID2.py as a python file (no zip, rar, etc.) where ID1 and ID2 are your IDs. We will run the game with our own agents, against your ex3_ID1_ID2.py.

The check is fully automated, so it is important to be careful with the syntax. The grades will be assigned as follows:

- Grading:
    - 80% for a correct implementation of the UCT agent.
    - 25% for a relative performance of your non-UCT agent compared to our bots of increasing difficulty (Competitive):
        - 10% for defeating a simple UCT agent with random rollouts, as you are required to submit.
        - 10% for defeating a UCT agent with light heuristic rollout – using h_1 from Homework 1
        - 5% for defeating a UCT agent with heavy heuristic rollout
- The submission is due on 4.4.2024, at 23:59.
- Submission in singles/pairs only.
- Write your ID numbers in the appropriate field ('ids' in ex3.py) as strings. If you submit alone, leave only one string.
- The name of the submitted file should be "ex3_ID1_ID2.py". If you submit alone, the name should be "ex3_ID.py" where ID is your ID.

To clarify: In the competitive part you are measured by the score difference between your agent and the adversary.

Clarifications:

Clarifications on UCT:

- For the UCT based agent, during rollout, the simulation of the node (the function simulate() in your code) should run **until the end of the game, from that node.** You shouldn't limit the depth of the tree or limit the run time of a specific rollout.
- Regarding the amount of simulations, you should run UCT for as many iterations as possible, under the time limit of act(). In the code example for UCT in tic tac toe that you have in the moodle, UCT runs for 1000 iterations. For this exercise you should run it for as long as you can under the time limit.
- To further clarify the point above, the **entire process** of UCT should run for as long as you can, but each usage of UCT has a **single rollout.**
- The base of the logarithm in the UCB1 formula should be **e.**