CS 165A – Artificial Intelligence, Spring 2024
Machine Problem 2

Responsible TA: Xuan Luo, xuan_luo@ucsb.edu

Due: June 8, 2024 11:59pm

1 Notes

■ Make sure to read the "Policy on Academic Integrity" in the course syllabus.
■ Any updates or corrections will be posted on Canvas.
■ You must work individually on this assignment.
■ Each student must submit their report and code electronically.
■ If you have any questions about MP2, please post them on the Canvas MP2 Q&A.
■ Plagiarism Warning: We will use software to check for plagiarism. You are expected to complete the project independently without using any code from others.

2 Problem Definition

In this assignment, you are required to develop a program that can autonomously play a rogue-like dungeon crawler, inspired by the classic game *Rogue* (`https://en.wikipedia.org/wiki/Rogue_(video_game)`) and *Pixel Dungeon* (`https://github.com/watabou/pixel-dungeon`). The primary goal of your AI program is to amass a high score by collecting as many coins as possible within the game. To achieve this, you should implement an intelligent strategy using search algorithms such as Minimax, Expectimax, or other suitable methods. If you opt for the Minimax algorithm, consider employing alpha-beta pruning to enhance performance, allowing for deeper search levels and potentially higher scores.

3 Gameplay and Mechanics

3.1 The Dungeon

As depicted in Figure 1, Pixel Dungeon features a dynamically generated map of 40x30 blocks. Within this environment, two agents—the 'Warrior' and the 'Magician'—compete to collect coins. You will develop the AI for the 'Warrior', clad in silver armor, while the 'Magician', dressed in a red robe, will operate automatically.

3.2 Player Actions

Each agent can perform one of five actions per time step:

- **W**: Move up one block.
- **A**: Move left one block.
- **S**: Move down one block.
- **D**: Move right one block.
- **I**: Remain idle (no movement).

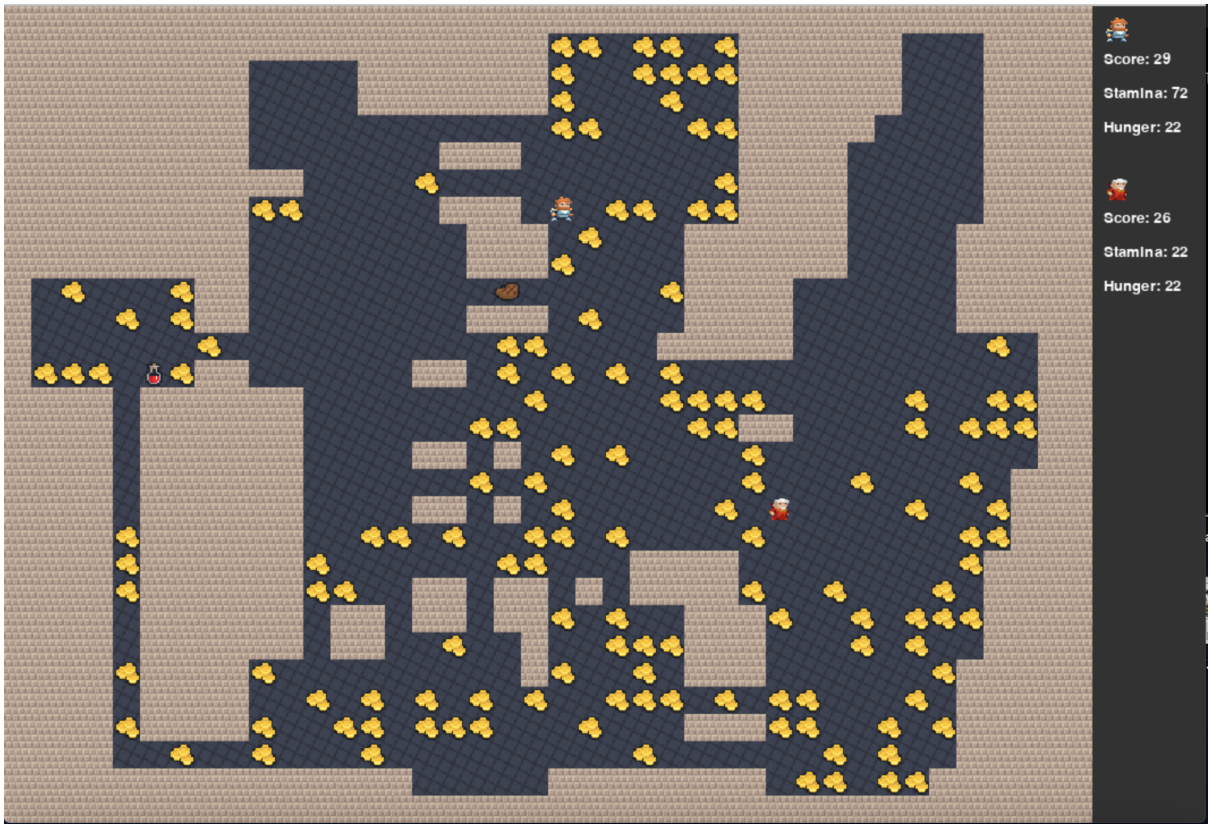

Figure 1: Example of Pixel Dungeon gameplay.

3.3 Agent Status

Agents have three primary states:

1. **Score**: Incremented by 1 for each coin collected.
2. **Stamina**: Depleted by 1 with each movement ('W', 'A', 'S', 'D'); remaining idle ('I') conserves stamina. Stamina is also consumed when attempting to move into a wall (e.g. take the movement action 'W' but hit a wall). The initial stamina is 50.
3. **Hunger**: Decreases by 1 every two turns. Every decrement in hunger restores 1 stamina. If hunger reaches zero, an additional stamina point is consumed per turn. The initial hunger is 50.

3.4 Map Elements

The map consists of several elements (See Figure 2):

- **Floor**: Blocks where agents can move.
- **Wall**: Impassable blocks.
- **Coin**: Blocks that, when stepped on, consume it and increase the agent's score by 1.
- **Food**: Consuming food from these blocks increases the agent's hunger by 30. It cannot exceed the maximum hunger 50.
- **Potion**: Consuming a potion from these blocks increases the agent's stamina by 20. It cannot exceed the maximum stamina 50.

3.5 Termination

The game terminates when all agents' stamina is depleted (<= 0) or all coins are collected. Your score for this machine problem will be determined by $\frac{your\_score}{opponent\_score}$.

Figure 2: Map Elements.

4 Program Specification

4.1 Task Overview

Your primary task is to complete a script named `player1.py`. This script must contain a function called `player1_logic`, which defines the movement logic of your AI agent, the 'Warrior'. This function is invoked at each time step during the game simulation and determines the agent's action based on the current state of the game environment.

4.2 Function Parameters

The `player1_logic` function receives the following parameters:

- **coins**: A list of tuples representing the (x, y) coordinates of all coins present on the map. Example: `[(3, 4), (4, 5)]`.
- **potions**: A list of tuples with the (x, y) coordinates of all potions, structured similarly to `coins`.
- **foods**: A list of tuples indicating the (x, y) coordinates of all food items, formatted identically to `coins`.
- **dungeon_map**: A 40x30 two-dimensional list representing the game map, where each cell contains a string ('wall' or 'floor') that indicates the type of the cell.
- **self_position**: A tuple containing the current (x, y) coordinates of your agent, the 'Warrior'.
- **other_agent_position**: A tuple indicating the (x, y) coordinates of the competing agent, the 'Magician'.

4.3 Return Value

The `player1_logic` function must return a single character ('W', 'A', 'S', 'D', or 'I') that specifies the next action your agent should take:

- **W**: Move up.
- **A**: Move left.
- **S**: Move down.
- **D**: Move right.
- **I**: Remain idle.

4.4 Running the Simulation

To initiate the simulation, execute the following command in your terminal:

```
python run_game.py --speed 5 --seed 777
```

The first parameter 'speed' specifies the simulation speed, which determines how quickly the game progresses. The second parameter 'seed', the random seed, influences the map generation. Using the

same seed will consistently regenerate the same map configuration. To thoroughly test your agent's adaptability, vary the seed to expose it to different map layouts. Notice that you have to install the package 'pygame' to run the code.

Here is an example of program output:

```
Potion count: 1
Food count: 3
Coins collected: 300
Game Over! Your Score: 68, Opponent's Score: 78
Percentage: 0.8717948717948718
```

The first three lines display the quantities of potions, food, and coins available on the map. Your final score is calculated based on the percentage achieved in our confidential evaluations.

5 Submission Guidelines

Please follow these instructions to ensure your submissions are processed correctly by the autograder and eligible for manual grading if necessary.

5.1 Gradescope Submission

We will create two distinct assignments on Gradescope for this module:

MP2: Submit all code-related documents here, including:

- A python script named `player1.py`
- Any additional scripts required for execution

MP2_report: Submit only your PDF report to this assignment.

Important: Do not submit any data files. Also, do not place your scripts or code inside a directory or zip file. The autograder should be able to run your code and provide accuracy results within a few minutes. While there is no limit to the number of submissions, only the last submission will be considered for grading.

Note: If the Gradescope autograder does not support a package you require, please make a post on Canvas. We aim to accommodate all necessary tools to ensure your code runs smoothly.

6 Grading Criteria

6.1 Grading Breakdown

The final grade for this project will be determined based on the following criteria:

- **Complete Report:** 20%
- **Execution Specifications:** 10%
- **Correct Program Specifications:** 10%
  - Reads file names from command-line arguments
  - Produces correct standard output results
  - No segfaults or unhandled exceptions
- **Test Accuracy and Runtime:** 60%

6.2 Leaderboard and Bonuses

- A real-time leaderboard will be available on Gradescope for tracking rankings.
- The top 16 results from the leaderboard will be selected, and your implementations will replace the default opponent agent for this assignment (all of the two agents are implemented by students). We will adopt a knockout tournament format to determine the first, second, and third places.
- The first place will receive a 50% bonus on their project grade.
- The second and third places will each receive a 25% bonus.

Note: We have a limit of 100 submissions on Gradescope.

6.3 Accuracy and Time

The majority of the grade (60%) is based on the testing accuracy and runtime:

- **Testing Accuracy:** Assessed on a private dataset.
- **Runtime:** Your code must complete execution within 10 minutes; otherwise, it will be terminated, and the accuracy score will be zero.

Score Calculation

- Scores based on the average percentage of $\frac{your\_score}{opponent\_score}$ (see part 4.4):
  - $0.0 \leq \frac{your\_score}{opponent\_score} < 1.0$: Scores will be linearly interpolated between 0% and 90%.
  - $1.0 \leq \frac{your\_score}{opponent\_score} \leq 1.1$: Scores will be linearly interpolated between 90% and 100%.
  - $1.1 \leq \frac{your\_score}{opponent\_score} \leq 1.5$: Scores will be linearly interpolated between 100% and 120%.
- A correct implementation of Expectimax Algorithm should achieve 1.0 average $\frac{your\_score}{opponent\_score}$.

7 Report Guidelines

The project report should adhere to the following specifications:

- **Length:** The report must be between 1 and 3 pages, with no exceptions exceeding this limit.
- **Font Size:** The text must be at least 11pt in size.
- **Content Requirements:** Your report should comprehensively cover the following sections:
  1. **Algorithm:** Provide a concise explanation of your code's algorithm, including a description of classes and their basic functionalities.
  2. **Results:** Discuss your results on the provided datasets, including accuracy and running time.
  3. **Challenges:** Detail the challenges you encountered during the project and how you addressed them.
- **Submission Format:** Submit only one PDF report containing all the sections listed above. Do not include additional README files.
- **Identification:** Your report must include your name, email, and perm number at the top of the first page.

Please ensure all sections are clearly defined and thoroughly addressed to meet the project's evaluation criteria.