CS 165A – Artificial Intelligence, Fall 2021

Machine Problem 2
100 points

Due *Tuesday, Nov 30, 2021 11:59pm*

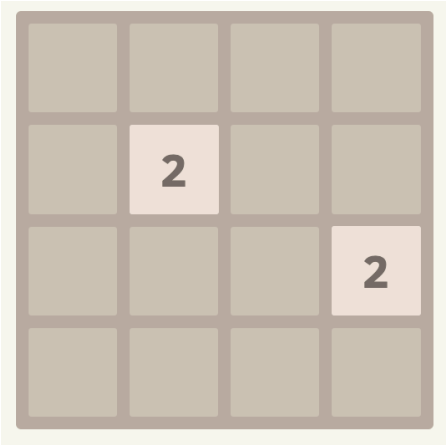
Notes:

- Make sure to read the "Policy on Academic Integrity" on the course syllabus.
- Any updates or corrections will be posted on Piazza.
- You must work individually for this assignment.
- Each student must turn in their report and code electronically.
- Responsible TA for Machine Problem 2: Shiyang Li shiyangli@ucsb.edu
- *Plagiarism Warning: We are going to use software to check plagiarism. You are expected to finish the project by yourself without using any code from others.*

1. Problem Definition

You are going to implement a program that plays **2048** (https://play2048.co/). Your program should be able to search for the best move so that it ends up achieving high game score. Towards this end, you should explore the **Minimax** or **Expectmax** search algorithm. If you use **Minimax** search algorithm, you may consider to use alpha-beta pruning, in order to quickly calculate the program's next move and eventually achieve possibly high score by searching deeper.

2. Game 2048



Game 2048 is a single-player sliding tile puzzle video game and the objective of the game is to slide numbered tiles on a grid to combine them to create a tile with the number 2048; however, one can continue to play the game after reaching the goal, creating tiles with larger numbers. [1]

2048 is played on a plain 4×4 grid, with numbered tiles that slide when a player moves them using the four arrow keys, namely UP, DOWN, LEFT, RIGHT. Every turn, a new tile randomly appears in an empty spot on the board with a value of either 2 or 4. Tiles slide as far as possible in the chosen direction until they are stopped by either another tile or the edge of the grid. If two tiles of the same number collide while moving, they will merge into a tile with the total value of the two tiles that collided. The resulting tile cannot merge with another tile again in the same move. The highest possible tile is 131,072. [1]

If a move causes three consecutive tiles of the same value to slide together, only the two tiles farthest along the direction of motion will combine. If all four spaces in a row or column are filled with tiles of the same value, a move parallel to that row/column will combine the first two and last two. When the player has no legal moves (there are no empty spaces and no adjacent tiles with the same value), the game ends. [1]

3. Technical Details

3.1 Rules

Initially, you will have a 4x4 gird, where two randomly different tiles have value 2 and the rest tiles (empty spots) have value 0. An example is shown in the figure of Page 1. You need to implement a function that takes the grid and current step of this game, and decide which direction you want to slide tiles, and return the direction. After your movement, possible tiles may merge according to rules in section 2 and then a random tile with value 0 (empty spot) will be changed into 2 with probability 0.9 or 4 with probability 0.1 by the system. Then you come to next step and decide which direction to slide until game is over.

Game is over only when one of the following conditions is satisfied (1) you can slide into none of four directions (2) Every element in non-zero in the gird after you slide tiles so that the system cannot change a tile with 0 into 2 or 4 (3) You manually output quit movecode in this game (4) An error occurs when call your function. (5) Gradescope kills your submission due to timeout.

3.2 Format

Specifically, you ***must*** provide a function written by Python3 with the following format.

```
def NextMove(Grid: List[List[int]], Step: int) -> int:
    search algorithm
    return MoveCode
```

NextMove takes Grid: List[List[int]] and Step as input, where Grid is a 4x4 matrix and Step is the current step for playing 2048 and begins with 1. Step may serve as a time measure and be useful if you want to quit current game and save your running time for following test cases. NextMove must return an integer, where 0 means UP, 1 means DOWN, 2 means LEFT, 3 means RIGHT, 4 means QUIT GAME and other integers mean DO NOTHING. If you output a movecode that is not applicable or means DO NOTHING, the grid will keep the same after executing your movecode but then system will randomly change a tile from 0 to 2 or 4. For example, if you output 0, but the grid actually cannot move up, the

grid will first be the same as before executing movecode 0 and then the system will randomly change a tile from 0 to 2 or 4.

3.3 Implementation Restrictions

You ***must*** implement your **NextMove** in **Python3** and name your file including **NextMove** function as **search.py** so that I can use **from search import NextMove** statement to call your function. During grading, the grade script will generate 10 different random initialized grids and it will call your function. Finishing these 10 test cases must be within **40 minutes**, otherwise Gradescope will kill grade script and you will get 0.

You can generate your test cases and call your **NextMove** for debugging purpose before submission. Although our test cases will be most likely different from yours, you will have a basic understanding how faster your search algorithm is and how many scores it can get.

4. Judging

The grade script will generate ten different random initialized grids. During playing for each test case, your final game score of it will be the **maximum** number in the grid, e.g. 1024, 2048 or 4096 after game is over. **This score calculation method is different from original game score**. Grade script will continue to run even after 2048 if you don't output QUIT GAME movecode to encourage you to push the limits. Your final game score will be the average among the 10 different test cases.

5. Report Preparation

As for the report, it should be between 1 and 2 pages in length (no more than 2 pages) with at least 11pt font size and should consist of at least the following sections:

- **Architecture:** Brief explanation of your code architecture (describe the classes and basic functionality)
- **Search:** How you search for the best move (which algorithm you use, what heuristics, optimizations, etc)
- **Challenges:** The challenges you faced and how you solved them
- **Weaknesses:** Weaknesses in your method (you cannot say the method is without flaws) and suggest ways to overcome them.

Note that the most important part is the Search section. There should be only one pdf report which includes all the above (no additional readme file).

6. Grading

Grade Breakdown:

- **20%** Complete Report
- **20%** includes executable code that matches specification.
- **60%** Gameplay performance

6.1 Leaderboard and Bonuses

The top-3 scorer will get extra credit for this assignment. You can view the current leaderboard on Gradescope.

Bonus tier 1: The student whose game score is ranked at the top will get 50% extra credit.
Bonus tier 2: The students whose game score is ranked among the top-3 will get 25% extra credit.

There is no limit on the number of submissions, but only the last submission will count.

6.2 Performance

You will be graded based on your final game score averaged across ten test cases! The main weight of the grade (60%) will be based on your program's performance so you should make your search is as optimal as possible. Your grade score will be linearly interpolated between the scores listed below:

Table 1: Percentage score calculation for different game score range

Game Score	Percentage score for the game score component
<128	0%
128	20%
1024	80%
2048	100%
2500	110%
>2500	110%

The grading scale is designed so that any search algorithm achieving less than 128 game score will get **0** score for game score component. For game score between 128 and 1024, scores will be linearly interpolated between 20% and 80%. For game score between 1024 and 2048, scores will be linearly interpolated between 80% and 100%. For game score between 2048 and 2500, scores will be linearly interpolated between 100% and 110%. For game scores more than 2500, score will be 110%.

7. Submission

Complete as specified in the project instructions. Then upload your code and report to Gradescope.

Since we will evaluate your code with a variety of test cases, the hard-coded solutions will not work on gradescope. The autograder on Gradescope might take a while but don't worry: so long as you submit before the due date, it's not late.

References

[1] https://en.wikipedia.org/wiki/2048_(video_game)#cite_note-8