Semester Project

Xinran Dai, Yixing Hu[1]*

Abstract
In the project, we built the game of 2048 from scratch, developed the logic of the game, built the user interface, and developed an AI that would try to solve this game.
The project is written in Python.

Keywords
2048 — expectimax — Tkinter

[1] *Computer Science, School of Informatics , Computing and Engineering, Indiana University, Bloomington, IN, USA*

Contents

1. General Guidelines

The project contains three parts: the code, the report and the poster. Below is the detailed description of how this project is constructed and how to run this game.

1.1 Code

- **2048.py**
  This file contains the GUI to interact with a player. It uses Tkinter to draw labels and boards on window and provides a way to interact with the player, either by clicking on the 'New Game' button or the 'AI Player' button.

  Save 2048.py, Board.py and AIPlayer.py under the same directory. Run the module in 2048.py, and a window would pop up. There are three frames in this file: the top frame, the board frame, and the bottom frame.

  Top Frame: The icon of the game, '2048' would show in the upper left corner of the window, with the current score and the best score showing to its right. This frame also contains the buttons for a new game or for the AI player.

  Board Frame: This frame contains the actual board of the game.

  Bottom Frame: This frame displayers instructions on how to play this game.

- **Board.py**
  This file contains the fundamentals of this game. With methods to make moves, check the state of the game, etc, Board.py builds the foundation of the game.

- **AIPlayer.py**
  This file contains the class for the AI player. The project uses a heuristic to calculate the best score of a board, and by using expectimax algorithm, the project is able to make moves to get the best result.

  Detailed explanation would be included in this paper below.

1.2 Others

- **Poster**
  This poster is used for presentation and demonstration.

- **Report**
  This report is to explain in details how this project was initiated and constructed, including instructions on how to run the code.

2. Rules and Constraints

2.1 Rules

Please refer to the previous part (General Guidelines) for rules of this game.

2.2 Constraints

The main constraint of building an AI for 2048 is to deal with randomly generated 2s and 4s, on random positions on board.

Another obstacle is to come up with a heuristic that would guarantee to solve this game. As many may already know, the game has a rule to follow and many approaches have been suggested. To find the most efficient heuristic is very important in solving this game.

3. Heuristics and Algorithm

3.1 Heuristic

3.1.1 Basic Logic

Based on our research, the best way to merge the board is to keep the tile with largest value at one corners and keep push other tiles towards it. To achieve that, we used a weight matrix: a 4*4 matrix with larger value in one corner (top left corner in our case) and smaller values in the diagonal corner (bottom right corner in our case). Then, the basic heuristic value would be the product of the value at the board and its corresponding value in the weighted matrix. By doing so, closer the large value is to the top left corner, larger the heuristic value would be. In other words, to get larger heuristic value, the AI would push the large-valued tile towards top left corner.

3.1.2 Penalty Logic

Since the goal of 2048 is to merge tiles with same value into a tile with larger value, the core of the game would be merge two tiles into one. Based on that thought, we developed a penalty rule, which checks if the movement makes two tiles with same value to be neighbors. So the penalty value would be the sum of the difference between every tile and its neighbor tiles. So the final heuristic value returned would be the basic heuristic value from previous step deduct the penalty value.

3.2 Algorithm - Expectimax

3.2.1 Dealing with random pop-ups

Instead of using Alpha-Beta pruning and Minimax algorithm, we chose Expectimax was a way to solve this problem.

Expectiminimax deals with problems with random difficulties. In this project, the player cannot decide if a 2 or 4 would be inserted to the board, and also no way to know where on the board it would be put. Expectiminimax thus has become our first choice to solve this problem.

3.2.2 One-player game

As we were solving Connect-4 game, there are two players in this game. In this project, only one player plays 2048. As min and max scores are used to compare situations suitable for each player, we only need to use the max score in order to make the best move for our player.

3.2.3 Logics behind it

We break the problem into two parts: when it's time for the player to make the move, and when it's time for the board to generate a 2 or a 4 in a random position.

Player's time

As there are four directions, we make four children boards and calculate the scores for each of the child. We keep record of the best score so far, and only update it when a higher score comes up.
We also keep track of the move in order to get this score.

Board generating 2/4

When it is time for the board to put a 2 or a 4, we simulate all the situations with all the possibilities.

For each empty cell on board, we would generate a 2 to put it there, and calculate the score by multiplying the one we get from heuristics and its weight, which is 0.9.

We then repeat this step, but instead of putting 2 on board, we put a 4 there, and the weight now is 0.1. After obtaining both scores for the two conditions, we would add them up and keep a record of the score.

In the end, we get the weighted average by dividing the sum by the number of empty cells.

3.3 Running time and consumed space

Based on our several test running, our AI can solve the puzzle in around 1 second, but only with a guarantee of a tile with at least 64. Usually the largest valued tile would be distributed among 128, 256 and 512.

4. Challenges

4.1 Building the Game

Even though the game is fairly easy and simple, it still took us some time to build this game from scratch. Instead of borrowing from existing code, we first constructed the board class (Board.py). The reason is that this way, we would be able to work on the AI part.

The little challenge we encountered during this process was how to merge the tiles to one direction. There is hardly any better way than to break the matrix into rows and merge the tile row by row, during which we need to sometimes transpose the matrix in order to get all four directions.

As for the GUI, since none of us had experiences in building GUI with Python, we had to learn from the beginning, from the layout to the mouse events and keyboard events.

One challenge in particular that has not been solved is to show how the AI gets to the final board step by step. Though the game itself works perfectly fine, we had little time left to improve our GUI with our AI.

4.2 Implementing the AI

4.2.1 Expectimax

The AI is the hard part of this project, and also the core of our purpose. Understanding expectimax was not hard, since it is very similar to minimax which we already implemented. However, it was a big challenge for us to figure out a way to consider all the random boards generated.

4.2.2 Random assignments of numbers

In the end, we realised that the only way to find the best move was to take all the possibilities into account. We used for loops to consider these: all the random positions and both of the situations in which a 2 or a 4 is put on board.

4.2.3 Coordinating all the files

Moreover, coordinating three classes could take a very long time. Though this was not a technical problem, it was very time-consuming to debug the code.

5. Experiments and Results

5.1 Experiments

We did the test by letting the AI solving the 2048 game form the original board, where there are only two tiles. To eliminate board bias, the start boards are randomly formed. Then we would start the AI to let it run and return the result board, with clocking beside.

5.2 Results

The highest tile value from the most recent ten tests are: 64, 256, 64, 128, 256, 256, 256, 128, 256, 512. Overall the test we did, we had 60% possibility to get 256, 26% possibility to get 128, 9% possibility to get 64 and 5% to get 512.

5.3 Comparison to others

Obviously, our AI for solving 2048 cannot successfully solve the game. Most of the time, our AI can only solve the game till 256. From the data we collected online, AI using expectimax algorithm has an average of forming the 2048 tile 9 times out of 10.

6. Summary and Conclusions

6.1 Alternative Approaches

An alternative approach to solve this game is to use minimax with Alpha-Beta pruning. This algorithm is similar to what has been used (expectimax). The reason we decided not to use this is that we have implemented minimax with Alpha-Beta pruning in Assignment 4, and doing so means that we'd lose the point in completing the final project.

Besides our algorithm, another way to implement this AI is to use bit operations which would definitely be less time consuming. This would involve many details of the algorithm rather than changing the algorithm itself. And even though we are not implementing the project in a bit-operation matter, it does not take long to run the project.

6.2 Summary

This was a fun project to work on and had moderate amount of challenges for us. On the one hand, it helped us to practice the ability to build a mini project by making us to implement 2048 from scratch. On the other hand, as a popular game, 2048 has drawn a lot of interests in its artificial intelligence solver, and this provides us with plenty of ideas to learn from.

While there are many ways to solve this game, we only had time to implement one of them. That being said, this project gives us a chance to study more about artificial intelligence on our own and to convert what we have learned into what we can do.

6.3 Notes

This game does not show the process of AI taking steps. Once the AI button is clicked, the project would run in a very short period of time (less than 2 seconds) and draw the board on screen. The reason to it is that we ran out of time when combining 2048.py with AIPlayer.py.

In order to see the steps, uncomment the code to print out boards, moves and scores in expectimax() in AIPlayer.py.

Acknowledgments

This is the final project for CSCI-B 351, Introduction to Artificial Intelligence. The professor is **Professor Saul Blanco**.

We would like to thank the professor and all the teaching assistants for their help throughout the semester. They have provided tremendous help to us and have been greatly helpful. We would like to especially thank our two mentors, **Daniel Iglarsh** and **Ben Lewis**, who kindly spent their time providing us guidance during this project.

(Other teaching assistants are **Jack Langston, Tristan Rogers, Scott Mathews, Kyle Yohler, Jacob Lin, and Matthew Ploetz**. All of the instructores dedicated their time to help us in office hours and we truly appreciate it.)

Moreover, we would like to thank **Gabriele Cirulli** who created this game. Even though we used a very different way to implement this game, all the credits of this game go to him.