2048 Pseudo-AI Term Project Proposal

**Project Description**

The purpose of this project “2048 Pseudo-AI” is to create an open-sourced educational game of 2048 that introduces the concept of algorithmic AI relationship with customizable algorithmic parameters written in the Python language. By exploring the optimal strategies for solving the popular game 2048, this game will offer a complete, playable game, on top of which users are able to learn how the expectimax AI algorithm works and design their own parameters to alter the efficiency of the algorithm.

Players will be given guidelines on how the top-down design of the AI algorithm is implemented, be able to alter the totally customizable parameters of each helper function of the algorithm, and finally, test the efficiency of their strategies by running the algorithm many times and obtain the probability of highest scores. At the later stages of the game, users may even be able to change the fundamental design of the game—for example, to have 3s and 5s, or to have multiple boards simultaneously—in order to learn the general purpose and mechanisms of this AI algorithm in solving complex problems.

**Competitive Analysis**

Several similar projects exist online and on GitHub, most of which are open-sourced and written in either JavaScript or C++. This project, however, uses a different language (Python) to design and solve the AI algorithm because Python is beginner-friendly for other learners to improve my source code. The approach will also differ, since most online projects do not simulate a board but rather use an online web game and register only key strokes to solve the game. This project will solve the algorithm by iteratively constructing the board and precisely monitor the complexity of the game state.

Moreover, most open-sourced 2048 projects on GitHub have limited readability, meaning their algorithmic design is unclear and accessibility for beginners is slow. To avoid this “black box” of AI algorithm, this project will aim to create a user-friendly UI with customizable parameters so even novice programmers will be able to be inspired and re-create the codes by themselves. Furthermore, this game will add levels to change the fundamental design of the game—for example, to have 3s and 5s, or to have multiple boards simultaneously—in order to encourage algorithmic thinking.

**Structural Plan**

The final project will consist of multiple files:

1. A main \_\_init\_\_.py file for the basic, functional game constructed in tkinter. The main class will be the game state, namely the board matrix. The animation code will be written with tkinter functions. This file will also include guidelines on how to customize both the game and the algorithm.
   1. Later this main file will be improved to make the game entirely customizable, including features like different numbers of base tiles and multiple boards. This will include more similar functions and objects, but with more customizable parameters all considered.
2. An ai.py file to solve the basic game of 2048. There will be a few functions for scoring individual aspects of a single game state, a sum function to evaluate and compute the final score of a single board, and a final recursive function to implement the AI algorithm and give a single recommended step.
   1. This file will be improved later to solve the advanced game mechanisms. It will make the individual scoring functions more customizable and the whole recursive process more tuned to the customizable parameters that the users will want to specify.
3. A dataVisualization.py file to visualize the results of test runs by utilizing matplotlib. Separating this external module will help make the structure clean and organized. It will also help debugging if anything goes wrong with the matplotlib code.

**Algorithmic Plan**

The hardest parts of making the game include merging, making the board entirely customizable, and introducing a general-purpose AI algorithm.

Merging the board will require a top-down approach to solve how the game works fundamentally. Similarly, making the board customizable also requires to means to use OOPy animation, clear design instructions, and maybe multi-threading to accelerate the testing process. Moreover, there must be functions to take and validate the users’ inputs and create new objects using these parameters accordingly.

The general-purpose AI algorithm is determined to be expectimax (a variant of expectiminimax, or minimax.) By inventing a scoring scheme, each game state of the board will be assigned a score. The algorithm will recursively find the single branch that scores highest in all possible moves within a certain number of depths.

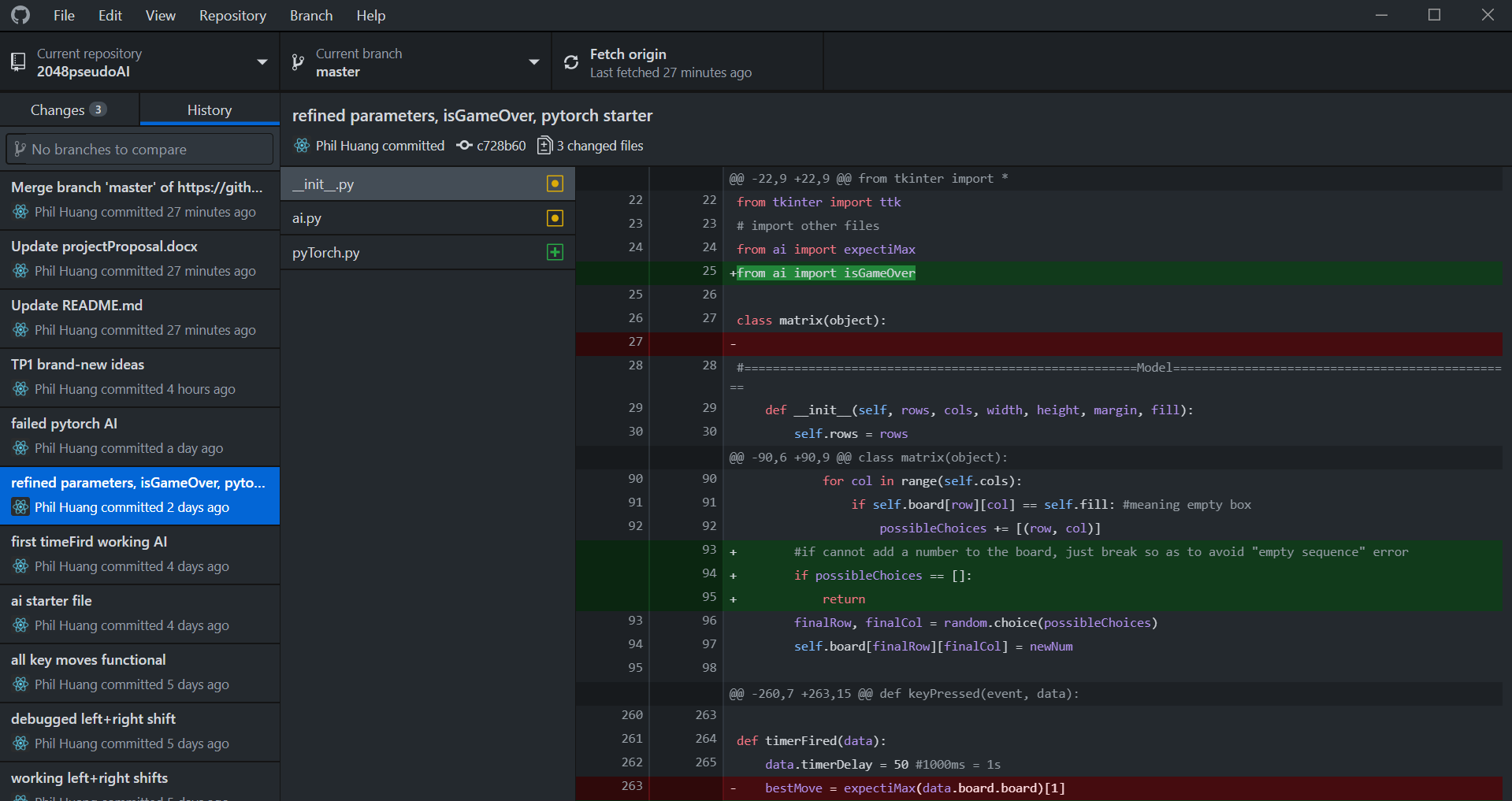
Finally, there will be a test run function that runs the game created by the users’ input for some number of times, pass along the results to a new file, and open a new window to visualize the data using matplotlib.

**Timeline Plan**

1. By TP1, the project should feature:
   1. A functional game of 2048
   2. A working expectimax algorithm with a general evaluation function supported by a number of scoring helper functions.
   3. The game should be entirely playable, and the AI should perform as reasonably well as a normal human would do.
2. By TP2, the project should feature:
   1. A GUI that allows users to input their customized parameters that will also be validated to have a reasonable game.
   2. Functions and objects that take in valid inputs and re-construct the game accordingly. Now the game state should be able to have different base numbers (3 or 5,) different sizes of boards, and multiple boards simultaneously.
   3. The algorithm should be able to adapt to the inputs and calculate the best moves accordingly. This will require OOP design for customizable calculations of many changes in boards, as mentioned above.
3. By TP3, the project should feature:
   1. A user-friendly collection of guidelines that explain the fundamental game mechanisms, the AI algorithms, and how they can change and test the parameters.
   2. There will also be general strategic instructions on how to get higher scores. More advanced instructions will also be provided on how to change the source code.
   3. Most importantly, debug the already complex customization and improve overall efficiency and accessibility. For example, advanced tkinter design may be used such as buttons, scrollers, and pop-up windows.

**Version Control Plan**

The entire project will be open-sourced on GitHub. I will consistently back up and document my progress using GitHub Desktop.



URL: <https://github.com/philxhuang/2048pseudoAI>

**Module & Dependency List**

Internal: basics, tkinter, threading

External: matplotlib

\*Storyboarding is attached as an image file within the same folder.

TP 2 Update

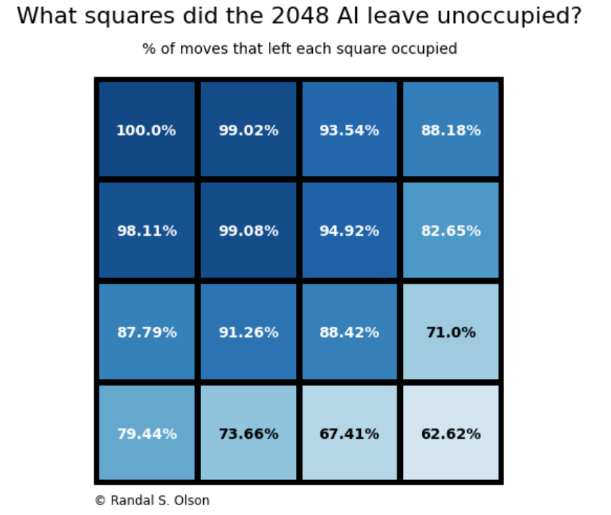
**Structural Plan Update**

The ai.py file should feature exactly 3 AI algorithms: expectimax, minimax, and reinforcement learning (RL). The parameters, such as max recursion depth and some evaluation functions will still be customizable.

The documentations will be more thorough because it will include how all algorithms work and their customizable parameters. The pages in the documentations will try to use recursive fractals to demonstrate how the AI algorithms work.

**Algorithmic Plan**

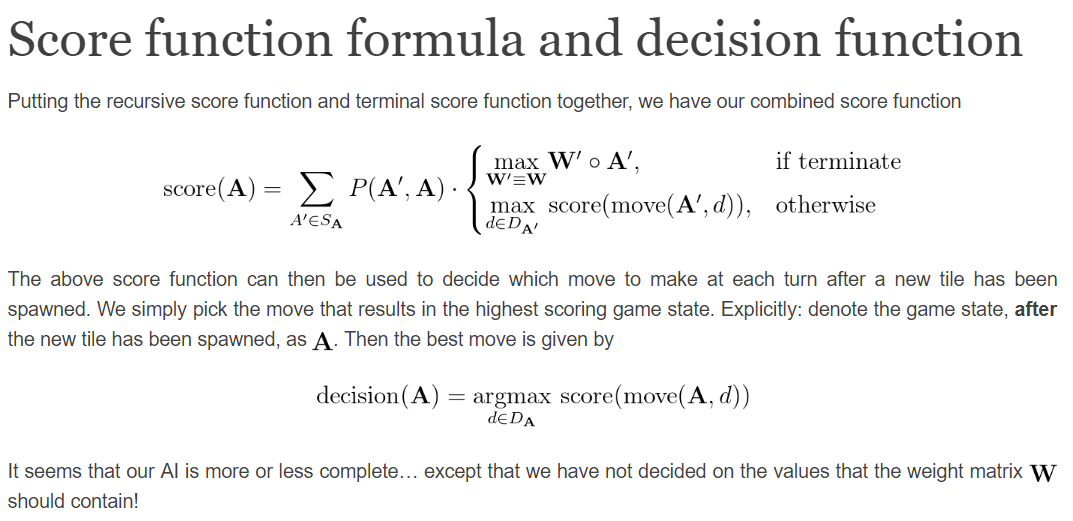
In addition to the first proposal, this project now will have an evil/hard mode where an algorithm will place the tile at the worst place possible. This is done by finding the empty tile that is closest to the largest tiles on the board.

 Now the game will also feature 3 algorithms. The first two, expectimax and minimax, will have some customizable parameters, allowing users to change as they wish. The third, which is the RL algorithm, will be self-learning as it runs. This is done by using a pre-set and customizable weight matrix that will be modified for every move in the game.

In this RL algorithm, essentially, the base matrix (general guideline) may look like base-matrix is [[1000,100,10,1],[0,0,0,0],[0,0,0,0],[0,0,0,0]]. The AI will try to modify it to get the best strategy after every single move in every single game and ultimate learn where to position the 1st, 2nd, 3rd largest tiles, ect. The end matrix after a few dozen games may look like [[564,240,105,49],[222,140,67,9],[7,4,2,1],[2,-5,-12,-23]].

More information about this RL: <http://www.randalolson.com/2015/04/27/artificial-intelligence-has-crushed-all-human-records-in-2048-heres-how-the-ai-pulled-it-off/>

A more math-y source for this RL: <https://codemyroad.wordpress.com/2014/05/14/2048-ai-the-intelligent-bot/>



**Timeline Plan**

1. By TP2, updates include:
   1. Finish both expectimax and minimax algorithm (with alpha-beta pruning) that perform reasonably well.
   2. Make the game entirely customizable, at least internally.
   3. Finish evil mode, which puts the tile intentionally at the worst tile possible.
2. By TP3, updates include:
   1. Massive documentations, maybe with fractals to illustrate how the algorithms work (draw nodes.)
   2. GUI for users to input their desire parameters.
   3. RL algorithm should be up and running, allowing this AI to learn on its own, along with other pre-set parameters.
   4. Matplotlib data visualization for testing purposes and show results in pie charts, if threading works well.

TP 2 Update

Final updates include:

* 1. Documentations with fractals done.
  2. GUI done.
  3. RL algorithm totally functionally, but beginner users are not able to change the evaluation functions. The documentation will allow more advanced users to go into source code and change the evaluation as they wish.
  4. Matplotlib data visualization for pre-recorded data for each algorithm. No threading needed.