CS 182 Final Project Proposal

Aaron Sachs, Everett Sussman & Jan Geffert

November 2, 2017

Problem: *2048* is a puzzle game which gained popularity in 2014. Played on a 4x4 grid, the goal of the game is to attain a puzzle piece with as high a value as possible by moving and merging the tiles on the board in intelligent ways [1]

The game mechanics are interesting as they incorporate both deterministic actions shifting the tiles to the top, left, bottom, or right of the grid and probabilistic elements, in the form of randomly appearing tiles with uncertain values.

Solutions: We intend to build several agents that solve *2048*, including:

1. a **greedy search** agent that picks the move which maximizes the highest-valued tile,
2. a **greedy search** agent that picks the move which maximizes the number of empty squares (empty square),
3. other **greedy search** agents based on the monotonicity of tile values,
4. a **heuristics-based expectimax** agent with a finely tuned evaluation function that is a linear combination of features. We might want to try to find the optimal weigthing of the different features by running local search algorithms on top of expectimax,
5. a pure **Monte Carlo simulation search** agent,
6. an **MDP** agent with knowledge about the underlying probability distributions determing the placement and value of new tiles,
7. a **Q-learning reinforcement learning agent** that learns these distributions / is robust to changes,
8. further explorations could involve a **Deep Q-learning** reinforcement learning agent and a **Monte Carlo Tree Search agent** if time permits,
9. and possibly, an agent learning from successful human play.

[1] Try it for yourself on `http://gabrielecirulli.github.io/2048/`

Metrics: For each algorithm, we plan to measure the following characteristics:

- What is the performance of the algorithm, i.e. the distribution of **final scores** and the distribution of the **values of the maximum tile** at the end of a game?
- What is the distribution of the **number of moves** that the algorithm takes to achieve the final solution?
- Are there certain **patterns** or **particular strategies** that emerge?
- How **complex** are the algorithm and the underlying data structures? What is the runtime, theoretically and practically, and how much space is required?
- In the case of learning algorithms: How many **iterations** are **needed** to reliably reach certain scores?
- We will also be interested in how each agent adapts when certain aspects of the game are changed. For instance we might change the probabilities that certain values appear on newly generated blocks, or we might increase the board size from 4 x 4 to 6 x 6.

We furthermore plan to conduct a non-representative study of the average level of human play to be used as a benchmark.

Development: The game mechanics are fairly simple allowing us to re-implement the game in a local Python testing environment via the Pygame package [2]. Having done that, we will first focus on designing a highly performant *2048* player and then try to optimize the algorithm to reduce runtime and/or space usage.

Expected Behavior of System: We expect the basic greedy algorithm to return very low maximum-value tiles due to many plateaus in the state space. That is, it will often be impossible to increase the value of the highest-valued tile across all possible actions, and so the local maximum that the greedy search agent concludes is a solution will likely be quite low.

We expect our tuned heuristics-based expectimax agent to outperform the greedy algorithm for reasonable feature choices, in the sense that the heuristics-based agent will yield a solution with a higher maximum-value tile. This is because the agent will be able to consider other important features of the state space in determining its next move, rather than just the value of the highest block alone. Possible features might include the number of blank tiles remaining on the board (with a preference for keeping this value high), the highest difference between adjacent blocks across all block pairs on the board (with a preference for keeping this value low).

How we intend to divide work on the project: Everett has already started work on building the environment in Python in which we will run our AI agent and play games of 2048. We will likely split up the implementation of the different algorithms amongst ourselves, but we will have consistent code reviews. Thus in the presentation and essay portion of the project, each member will write/present the algorithm he implemented and the findings across all of the different agents will be summarized in the end of both the presentation and essay.

Resources:

- The official *2048* implementation which is available under the MIT license at https://github.com/gabrielecirulli/2048.
- The relevant chapters of AIMA
- Silver, David (2009). Reinforcement Learning and Simulation-Based Search in Computer Go `http://papersdb.cs.ualberta.ca/~papersdb/uploaded_files/1029/paper_thesis.pdf`
- Silver, David; Huang, Aja; Maddison, Chris J.; Guez, Arthur; Sifre, Laurent; van den Driessche, George; Schrittwieser, Julian; Antonoglou, Ioannis; Panneershelvam, Veda (2016-01-28). Mastering the game of Go with deep neural networks and tree search. `http://www.nature.com/nature/journal/v529/n7587/full/nature16961.html`
- Artificial Intelligence has crushed all human records in 2048. Here's how the AI pulled it off. `http://www.randalolson.com/2015/04/27/artificial-intelligence-has-crushed-all-human-re`
- An artificial intelligence for the 2048 game `http://iamkush.me/an-artificial-intelligence-for-the-2048-game/`