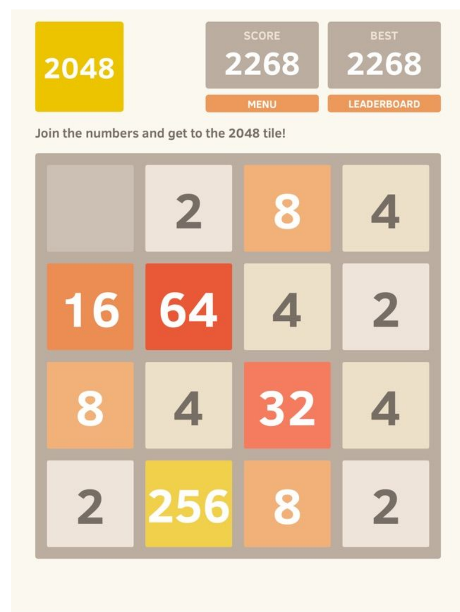
AI Agent for Winning 2048

Ricky Grannis-Vu
rickygv@stanford.edu

2048

Problem Definition

- My goal is to create an AI agent which can create the 2048 tile in **every** 2048 game it plays without significant time delay
- I will further attempt to surpass the score 819404, which is the highest recorded legitimate human score in 2048
- I will **not** attempt to beat the highest human score consistently, only once, since higher scores are very dependent on **luck**



Motivation

- Similar to games such as chess and Go, 2048 proves an interesting challenge for AI to play well since it has a very large state space
- I used to play 2048 semi-competitively with friends online and thought it would be interesting to apply strategies I learned to designing heuristics for an AI agent

Challenges

- Accounting for the **randomness** of the location and values (either 2 or 4) of new tiles spawning
- The state space for 2048 is **extremely large**, rendering my initial attempts to build an AI agent **extremely slow**
  - Need to design an appropriate **evaluation function** to approximate the expected score
  - Need to discover **optimal weights** for features in evaluation function
  - Need to avoid getting stuck in **local maxima**
- Ensuring that the AI agent prioritizes long-term strategic position over short-term increases in score
  - Need to keep game state flexible and organized in **monotone sequences** to allow **merge combos**

Game playing

- 2048 can be modeled as a **two-player zero-sum game** between the player and the computer
  - Model random generation as another player with a **fixed and known** policy
- AI agent evaluates moves using **depth-limited expectimax**:

$$V_{exptmax} = \begin{cases} Utility(s) & IsEnd(s) \\ Eval(s) & depth = 8 \\ max_{a \in Actions(s)} V_{exptmax}(Succ(s,a,d)) & Player(s) = agent \\ \frac{\sum_{a \in Actions(s)} V_{exptmax}(Succ(s,a,d-1))}{|Actions(s)|} & Player(s) = game \end{cases}$$

- Chooses action corresponding to $V_{exptmax}$

Approximation

- **Initial approach**: Eval(s) returns game score at state s

```
score = 0
for tile_value in list_of_tile_values:
    power_of_two = log(tile_value, 2)
    score += (power_of_two - 1) * tile_value
```

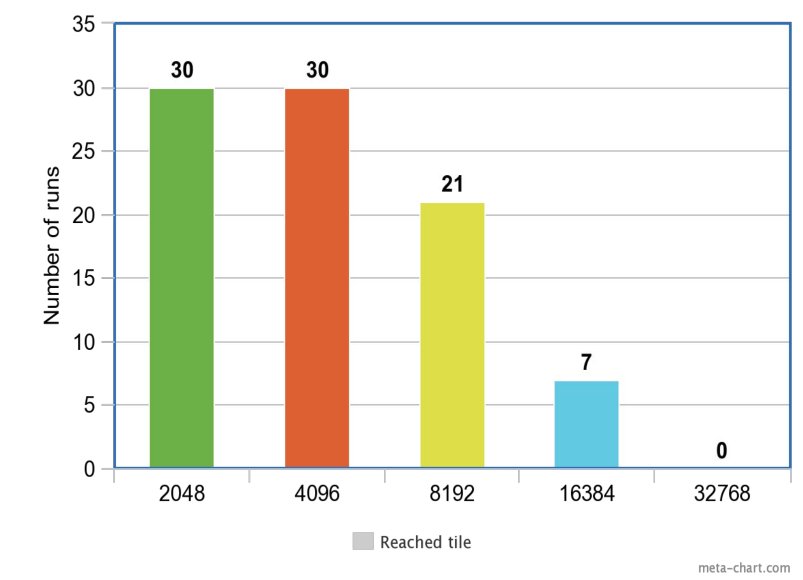
  - **Problem**: This heuristic is **greedy** and prioritizes short-term gains over long-term strategic position
- **Solution**: Implement other heuristics to counteract the greedy heuristic and weight them appropriately using temporal difference (TD) learning
  1. More **unoccupied** cells represents a better position
  2. Tiles should be adjacent to other tiles of **similar value**
  3. Tiles should be **grouped** in the same corner, with **higher tiles closer** to the corner
- At the end of each move, we update the weight vector:

$$w \leftarrow w - \eta[w \cdot \phi(s) - (r + w \cdot \phi(s'))]\phi(s)$$

- Epsilon-greedy approach to avoid getting stuck in **local maxima**
  - Before moving, randomly generate a value 0 <= p <= 1:
  - If p < 0.9, perform optimal action. Else, perform random action

Results

- After learning the weight vector through TD learning, I ran my AI agent on **thirty games**



- The AI agent was **successful** in creating the 2048 tile in every game it played. Furthermore, it was always able to create the 4096 tile
- The highest tile reached was 16384
- The highest score reached was 347000, which is the **same order of magnitude** as the highest human score but **still falls short**
- The average score reached was approximately 221620, which is also the same order of magnitude as the highest human score
- We have achieved **tremendous improvement** over our baseline performance, in which our highest score reached was 6832 and our average score reached was 1885

Analysis

- Qualitative evaluation of my AI agent revealed that it tended to play somewhat **conservatively**
  - Possible consequence of using expected value to model random tile generation
  - Normally, aiming for expected value is optimal. However, may be detrimental in this case because AI agent does not take **risks** to get a high score
  - In future, can introduce a new hyperparameter to represent **risk seeking / risk aversion** in calculation of expected value
- Additionally, in future, can tune existing hyperparameters (step size, epsilon) to improve TD learning