CandyCrush.ai: An AI Agent for Candy Crush

Niranjan Balachandar, Jay Lee, & Karan Singhal
Mentor: Adam Abdulhamid

CANDY CRUSH

- One of the most popular games ever
- Deceptively simple: state space is on the order of 10^{50}
- Finding an optimal score is NP-hard
- Difficult to predict the future value of moves, given the cascading effect of eliminating pieces
- Standard learning algorithms given our problem definition are computationally infeasible

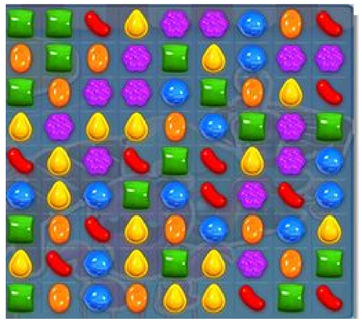
PROBLEM DEFINITION

- Modeled the game as a Markov Decision Process
  - The state is the current board (2D array) and turns left
  - The set of actions is valid moves from a state
  - A valid move creates a row/column of 3 or more pieces in a row to be deleted)
  - Transitions: not explicit, given by game mechanics
  - Reward: Score function

$$Score(x) = 10(X^2 - X) * Combo$$

where X is the number of pieces deleted

and combo specifies number of consecutive "deletions" after 1 move



```
[[3 4 1 4 2 4 2 5 1]
 [1 4 3 2 5 4 4 5 3]
 [4 2 5 5 3 1 5 4 3]
 [2 1 1 2 2 1 2 2 5]
 [1 2 3 3 1 3 2 1 1]
 [2 4 2 5 2 2 1 3 2]
 [5 1 2 2 3 5 5 1 1]
 [2 1 5 1 2 5 2 3 3]
 [3 4 4 3 5 1 3 5 1]]
```

Sample Input

BASELINE/ORACLE

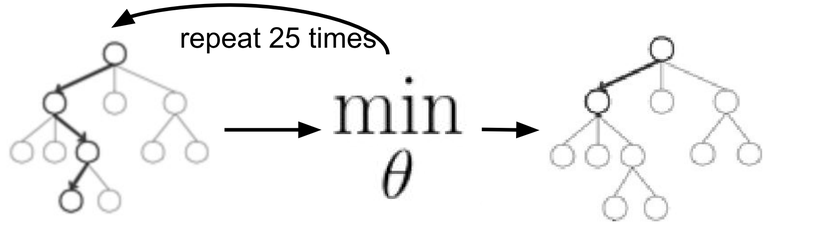
- Baseline was making the first valid move that the agent sees
- Initially oracle was human player (Jay Lee)
- Quickly outperformed by even the greedy algorithm (make move with biggest immediate reward)
- Oracle is the greedy algorithm, similar to that implemented by Alex Ene (https://github.com/AlexEne/CCrush-Bot)

APPROACH/MOTIVATION

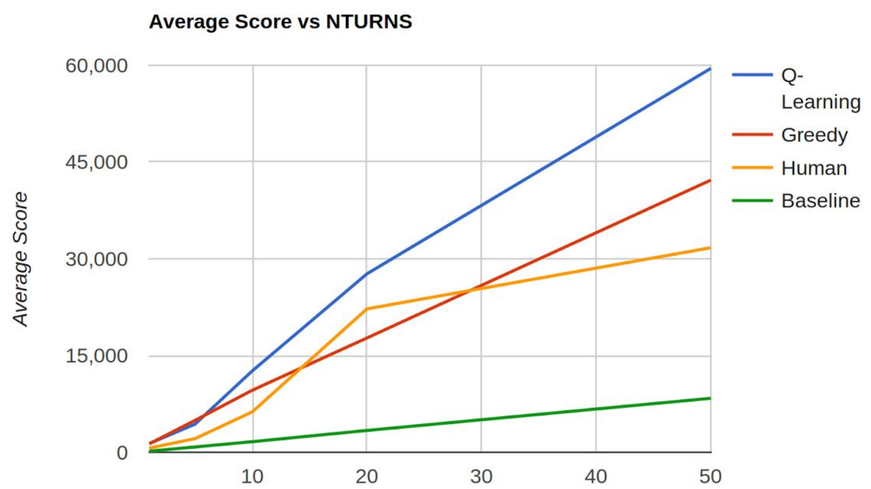
- Need a model-free approach given the complexity of transitions and rewards
- Want to learn values only for relevant states, otherwise computationally infeasible
- We propose a modified Q-learning approach in which our value function is learned on the fly:
  - For every move, simulate the rest of the game 25 times using an epsilon-greedy exploration policy
  - Record the states, actions, and rewards in the episode. and use them to update our model mapping states and actions
  - Our model is a neural network mapping state-action pairs to Q values, trained to minimize the loss function:

$$\min_{\theta} \sum_{(s,a,r,s')} (Q_{opt}(s,a;\theta) - (r + \gamma \max_{a'} Q_{opt}(s',a';\theta)))^2$$

  - Choose the action best action according to current model



RESULTS



We ran each of these algorithms multiple times for each number of turns (NTURNS). This is a graph of the average score vs the number of turns for each of our algorithms. Q-Learning tends to excel the more turns there are to play.

FUTURE STEPS

- Improve function approximation using features such as number of "almost" deletions on the board, number of turns left, numbers of a certain color in a certain row/column, etc.
- Improve run time (Q-Learning takes approximately 2 seconds per turn)
- Experiment with different exploration policies like softmax