AI for Paper.io - Solving Adversarial Games with Imperfect Information

Junshen Kevin Chen | jkc1@stanford.edu

Objective and Motivation

The objective of this project is to construct various agents to solve a game, **paper.io**, and evaluate their performance.

Paper.io is a game of **imperfect information**. In the game, opponent's state and actions are only partially observed, and therefore presents interesting challenges to model the game.

Modeling the Game

The game is modeled as a **simultaneous adversarial zero sum game**, played within an arena by multiple agents.

To win the game, an agent either:

- Eliminate the adversary by stepping on its trail
- Maintain a larger territory when the round ends

```
   000 001 002 003 004 005
00  x   x   x       [X]      00
01  x   x   x   .x. .x.      01     arena_size  (6 , 6)
02  x   .o. .o. [O]          02         agents  [X]  [O]
03      .o.       o   o   o   03     territory   x    o
04      .o. .o.   o   o   o   04         trail  .x.  .o.
05                o   o   o   05
   000 001 002 003 004 005
```

At `vision_radius=1`, agent `[O]` obtains **observation** of only the cells around it (left); it keeps a **memory** of where its own territory and trails are (right).

Finally, it holds a calculated **"belief"** that the adversary is near the position `?x?` , calculated by running the agent's algorithm (e.g. minimax) based on previous observation of the adversary.

```
   000 001 002 003 004 005       000 001 002 003 004 005
00                          00                    ?X?       00
01           x  .x. .x.     01            x   .x. .x.       01
02          .o. [O]         02       .o. .o. [O]            02
03              o   o       03       .o.       o   o   o    03
04                          04       .o. .o.   o   o   o    04
05                          05                 o   o   o    05
   000 001 002 003 004 005       000 001 002 003 004 005
```

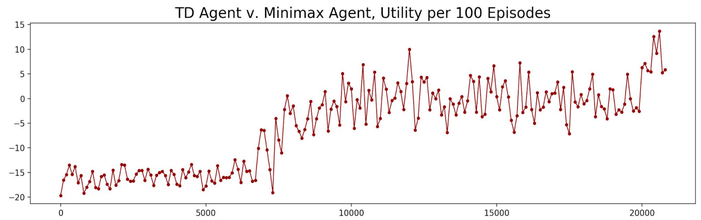
A **game simulator** is built to run games and train agents.

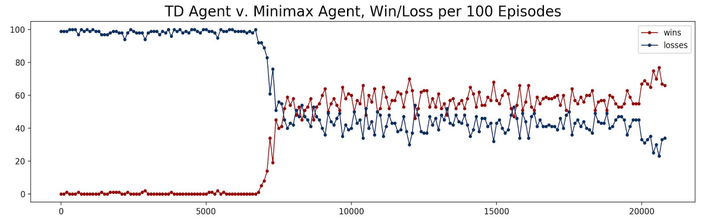
Reinforcement Learning

A **temporal difference (TD) learning agent** with features extracted from the game state: $Vw(s)=w \cdot \phi(s)$, then weights are learned by simulating an episode against minimax / expectimax agent then perform SGD:

$$w := w - \eta\,[V(s;w) - (r + \gamma V(s';w))]\,\phi(s)$$

Features: distance to own territory, adversarial distance to trail, min distance from arena walls, etc.
Reward: $r = (MaxAgentTerritory - MinAgentTerritory) * 1[IsEnd(s)]$
Discount: $\gamma = 0.7$
Epsilon-greedy exploration-exploitation: $\epsilon = 0.02$





Performance

Table of **expected utility** of each matchup simulated 5000 games, with the row agent being the max agent. A positive value means the row agent is expected to win.

	Random	Minimax4	Expectimax4
Random	-0.3342	-	-
Minimax4	31.43	-1.7666	-
Expectimax4	31.88	-4.0233	2.5567
TD	29.07	2.0533	4.6667

Adversarial Approach

Minimax agents are implemented with depth-limited search. Then alpha-beta pruning is added and performance increase is measured. **Expectimax** agents are constructed similarly.

Eval functions defined for agents of different behavior, such as :

- "Builder": $Eval(s)=TerritorySize$
- "Safe Builder":
  $Eval(s) = \lambda * TerritorySize +$
  $(1-\lambda) * (OpponentDistToTrail - DistToOwnTerritory)$

Utility is similarly defined as $MaxAgentTerritory - MinAgentTerritory$ (i.e. zero-sum); if one agent kills the other, the entire map becomes the winner's territory.

agent	depth	mean computation time
minimax	2	4.0015 ms
minimax	3	39.1651 ms
minimax_alpha_beta	3	18.5131 ms
expectimax	3	49.9342 ms

Conclusion and Discussion

- Minimax / expectimax agent sees a performance increase as search depth increases, and alpha-beta pruning enables deeper search by halving computation time. Expectimax agent outperforms minimax agent of the same depth due to imperfect information making minimax agent unable to perfectly predict the adversary's move.
- A TD agent with custom features and learned weights is able to outperform the agent it is trained against, and generalize well to outperform other agents.
- A TD agent outperforms its opponent more as vision radius decreases, and generalizes well when trained under infinite vision, then play under limited vision.

Future Work

- A **Monte Carlo Tree Search** (MCTS) agent
- More complex TD agent features, with multiple layers of neurons and activation
- Devise methods to encode **decaying confidence** (as inversely proportional to time since last seen) in an agent's memory, and use that information to perform better search