New Approach on 2048

Game Optimization

Mon. May 1st, 2023

Group 2 (Jason Andrade, Minho Cho, Alyssa Lowe, Ethan Weaver)

Team Members

- Minho Cho (MEng, Data Analytics Concentration)
- Jason Andrade (MEng, Software Engineering Concentration)
- Alyssa Lowe (MEng, Software Engineering Concentration)
- Ethan Weaver (MEng, Data Analytics Concentration)

Presentation Outline

- Introduction
- Problem Statement
- Approaches
- Implementation Details
- Demo
- Results
- Lesson Learned
- Future Work

Introduction

- The game 2048 is a popular web based strategy game
- The objective of the game is to create a tile with the value of 2048, or greater
- A player can move all tiles on the board in one of 4 direction: up, down, left, or right
- If two tiles collide, they combine to form a tile of twice their value
- After each move, a new tile of value 2 or 4 is generated in a random empty space

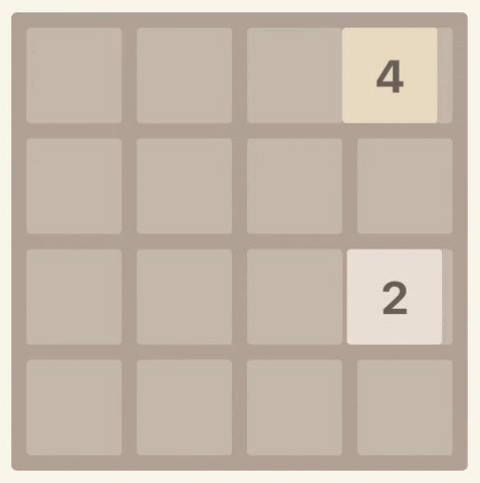


Figure 1. Game Play

VIRGINIA TECH

Problem Statement

- Build an Expert Agent based on a Learning Agent using **different algorithms** to maximize game score making **optimal movements**.

- To compare different types of AI algorithms to see which performs best at the game 2048.

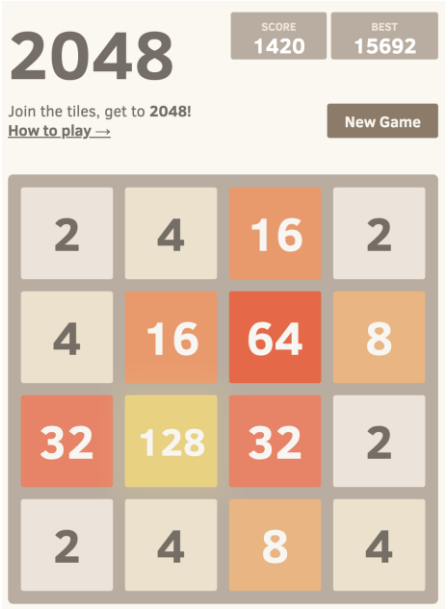


Figure 2. Losing Game Play

VIRGINIA TECH

Random Agent (Baseline)

Makes completely random moves

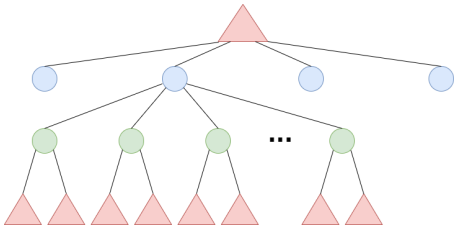
Average Score	Standard Deviation
962	473

Tile Value	16	32	64	128	256
Success Rate	99.99%	99.52%	88.7%	46.05%	4.22%

VIRGINIA TECH

Approaches

- Expectimax Algorithm



- Q Learning Algorithm

```
1: function PLAY GAME
2:     score ← 0
3:     s ← INITIALIZE GAME STATE
4:     while ¬IS TERMINAL STATE(s) do
5:         a ← arg max_{a'∈A(s)} EVALUATE(s, a')
6:         r, s', s'' ← MAKE MOVE(s, a)
7:         if LEARNING ENABLED then
8:             LEARN EVALUATION(s, a, r, s', s'')
9:         score ← score + r
10:        s ← s''
11:    return score
12:
13: function MAKE MOVE(s, a)
14:    s', r ← COMPUTE AFTERSTATE(s, a)
15:    s'' ← ADD RANDOM TILE(s')
16:    return (r, s', s'')
```

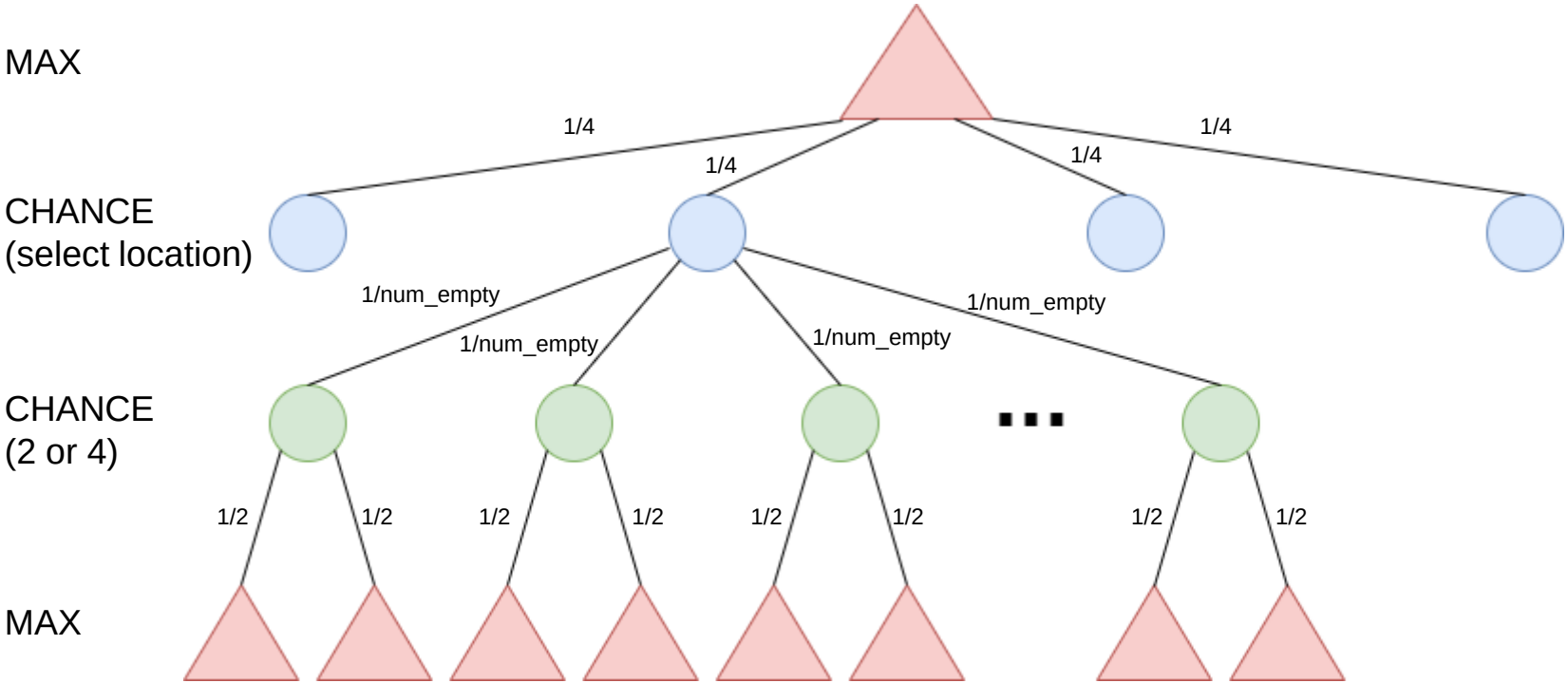
```
1: function EVALUATE(s, a)
2:     return V_a(s)
3:
4: function LEARN EVALUATION(s, a, r, s', s'')
5:     v_next ← max_{a'∈A(s'')} V_a'(s'')
6:     V_a(s) ← V_a(s) + α(r + v_next − V_a(s))
```

- Temporal Difference (TD) Algorithm

```
1: function EVALUATE(s, a)
2:     s', r ← COMPUTE AFTERSTATE(s, a)
3:     return r + V(s')
4:
5: function LEARN EVALUATION(s, a, r, s', s'')
6:     a_next ← arg max_{a'∈A(s'')} EVALUATE(s'', a')
7:     s'_next, r_next ← COMPUTE AFTERSTATE(s'', a_next)
8:     V(s') ← V(s') + α(r_next + V(s'_next) − V(s'))
```

VIRGINIA TECH

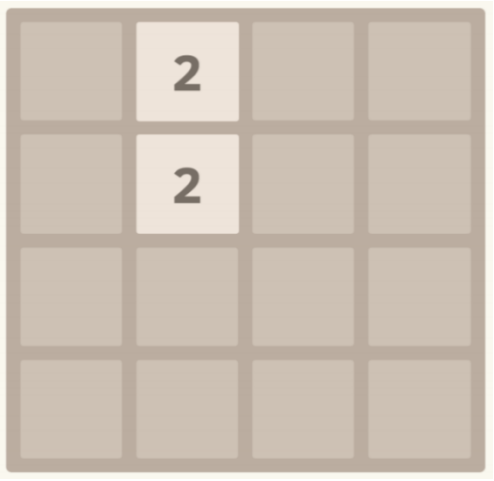
Expectimax Implementation



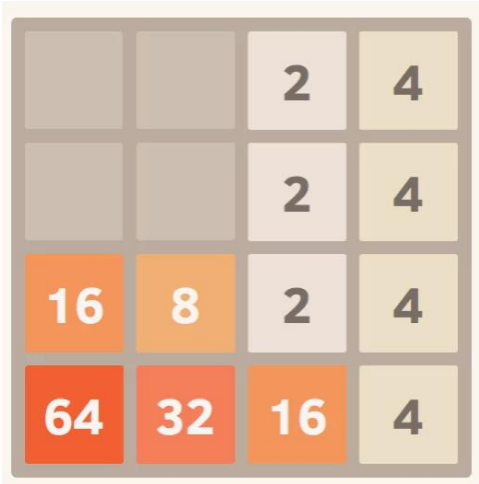
VIRGINIA TECH

Expectimax Evaluation Function

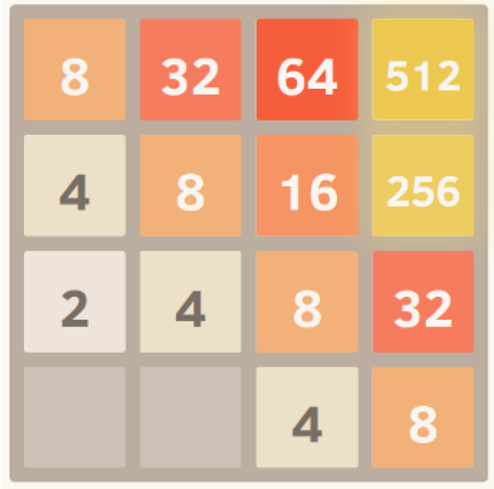
Number of Empty Cells on the Board



Number of Merges Possible



Monotonicity of Rows and Columns

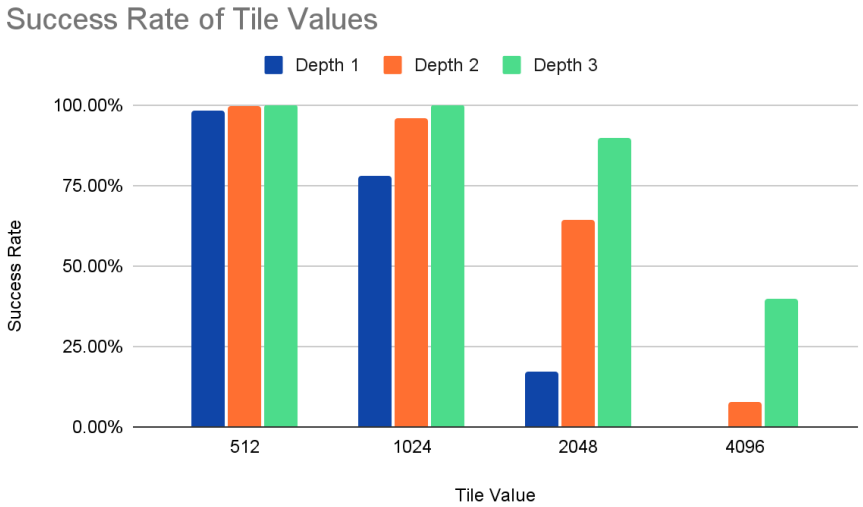


$$E(x) = 350 * num\_empty + 800 * merges\_possible + 20 * monotonicity$$

VIRGINIA TECH

Expectimax Results

Depth	Average Score	Standard Deviation
1	15,883	6,841
2	27,962	12,632
3	40,369	16,109



VIRGINIA TECH

Q-Learning

- Two states considered:
  - “current state” - s,
  - “nextstate” - s’’

- Q-learning evaluates the state by tracking the rewards for a state given an action.

- The “Learn function” adjusts the current states reward within the given action

```
1: function PLAY GAME
2:     score ← 0
3:     s ← INITIALIZE GAME STATE
4:     while ¬IS TERMINAL STATE(s) do
5:         a ← arg max_{a'∈A(s)} EVALUATE(s, a')
6:         r, s', s'' ← MAKE MOVE(s, a)
7:         if LEARNING ENABLED then
8:             LEARN EVALUATION(s, a, r, s', s'')
9:         score ← score + r
10:        s ← s''
11:    return score
12:
13: function MAKE MOVE(s, a)
14:    s', r ← COMPUTE AFTERSTATE(s, a)
15:    s'' ← ADD RANDOM TILE(s')
16:    return (r, s', s'')
```

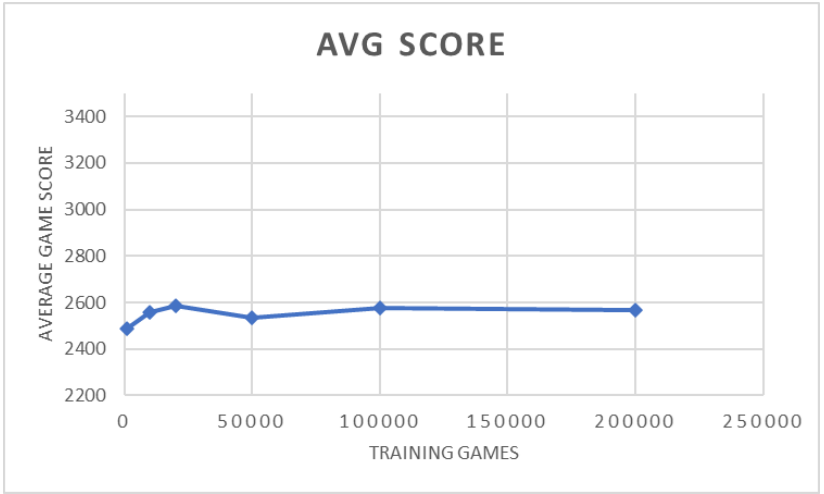
```
1: function EVALUATE(s, a)
2:     return V_a(s)
3:
4: function LEARN EVALUATION(s, a, r, s', s'')
5:     v_next ← max_{a'∈A(s'')} V_{a'}(s'')
6:     V_a(s) ← V_a(s) + α(r + v_next − V_a(s))
```

VIRGINIA TECH

Marcin Szubert, Wojciech Jaskowski - www.cs.put.poznan.pl/wjaskowski/pub/papers/Szubert2014_2048.pdf

Q-Learning Algorithm Results

alpha = 0.0025, real games = 1000



Number of Training Games	Average Score	Standard Deviation	% to reach 512
1,000	2486.62	1213.5	4.6%
10,000	2557.628	1220.7	4.3%
100,000	2,577.9	1,214.1	4.7%
200,000	2568.39	1210.6	5.2%

VIRGINIA TECH

Temporal Difference Afterstate Learning

- Three states considered:
  - “current state” - s,
  - “afterstate” - s’
  - “nextstate” - s’’

- TD afterstate learning evaluates the result of moves based on the reward assigned to an afterstate.

- The “Learn function” adjusts an afterstates value by determining the possible reward of the next afterstate if an optimal move is made.

```
1:  function PLAY GAME
2:      score ← 0
3:      s ← INITIALIZE GAME STATE
4:      while ¬IS TERMINAL STATE(s) do
5:          a ← arg max_{a'∈A(s)} EVALUATE(s, a')
6:          r, s', s'' ← MAKE MOVE(s, a)
7:          if LEARNING ENABLED then
8:              LEARN EVALUATION(s, a, r, s', s'')
9:          score ← score + r
10:         s ← s''
11:     return score
12:
13: function MAKE MOVE(s, a)
14:     s', r ← COMPUTE AFTERSTATE(s, a)
15:     s'' ← ADD RANDOM TILE(s')
16:     return (r, s', s'')
```

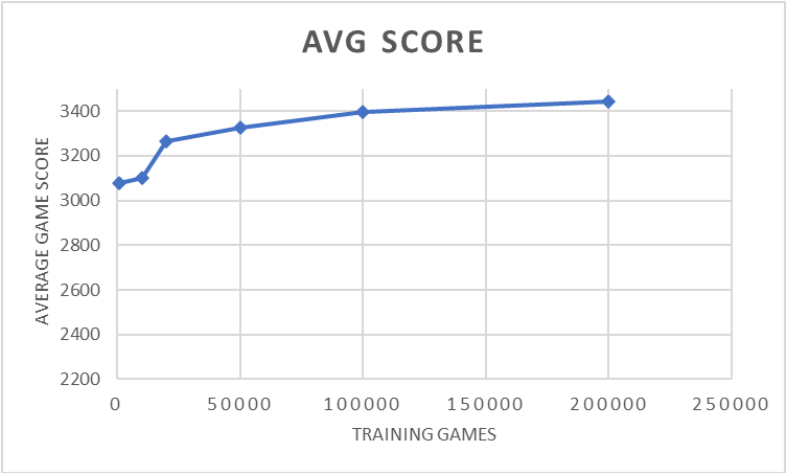
```
1:  function EVALUATE(s, a)
2:      s', r ← COMPUTE AFTERSTATE(s, a)
3:      return r + V(s')
4:
5:  function LEARN EVALUATION(s, a, r, s', s'')
6:      a_next ← arg max_{a'∈A(s'')} EVALUATE(s'', a')
7:      s'_next, r_next ← COMPUTE AFTERSTATE(s'', a_next)
8:      V(s') ← V(s') + α(r_next + V(s'_next) − V(s'))
```

VIRGINIA TECH

Marcin Szubert, Wojciech Jaskowski - www.cs.put.poznan.pl/wjaskowski/pub/papers/Szubert2014_2048.pdf

TD-Learning Algorithm Results

alpha = 0.0025, real games = 1000



Number of Training Games	Average Score	Standard Deviation	% to reach 512	% to reach 1024
1,000	3077.408	1551.6	13.7%	0.1%
10,000	3098.74	1568.2	13.7%	0.1%
100,000	3394.1	1652.3	18.5%	0.03%
200,000	3441.46	1683.82	19.1%	0.5%

VIRGINIA TECH

Results Comparison

Algorithm	Average Score (Depth 3/100,000 Games)	Standard Deviation (Depth 3/100,000 Games)
Expectimax	40,369	16,109
TD-Learning	3,394	1,652
Q-Learning	2,577	1,214
Random	962	473

- Expectimax far outperforms the other methods since it has more information in the form of its evaluation function
- TD-learning has an overall better average score per learning games than Q-learning.
- TD-learning has a better learning rate than Q-learning
- Overall, results coincide with the amount of information that each algorithm has access to.

VIRGINIA TECH

Lessons Learned

- Time and computing resource limitations:
  - When using personal computing devices, some group members are not able to run all algorithms implemented
    - For example: running expectimax at depth 3 for one game would take up to 3 hours for some, or approx 30 minutes for others
    - Having to run games overnight to generate result
- Different algorithms perform better or worse depending on the specific game mechanics and data available for training
  - Small changes to the Expectimax Evaluation Function can have huge effects on the performance of the agent

Future Work

- Improve the efficiency of all algorithms or at least improve upon best performing algorithm, expectimax, to maximize the expected outcome with other evaluation metrics such as max tile reached vs moves taken vs average score
- Similar to Pacman course projects, can continue to experiment with other reinforcement learning techniques that are best leveraged for gamification
- Use cloud computing resources proposed by Professor during class instead of using personal devices

VIRGINIA TECH

Q&A