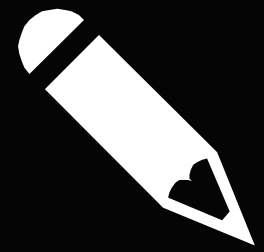
# PLAYING 2048 WITH AI

DMAI

Peiyu Hu, Wei Lin, Yuntong Li, Jiaqi Wan, Yifei Yan

Prof. Sebastian Wandelt
TA: Weibin Dai



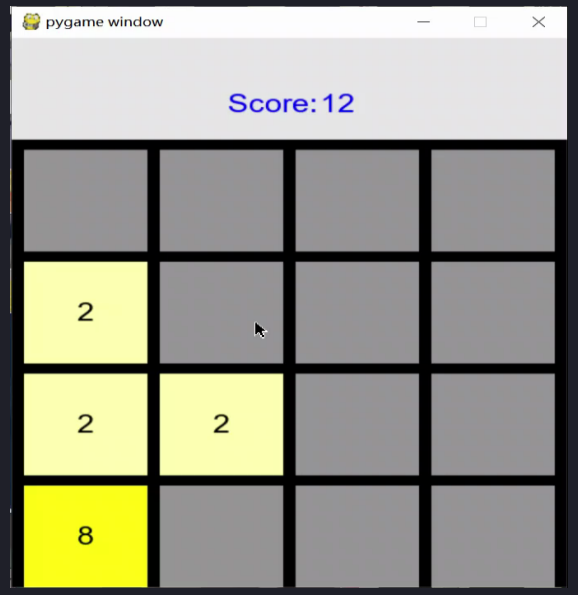
# PART ONE

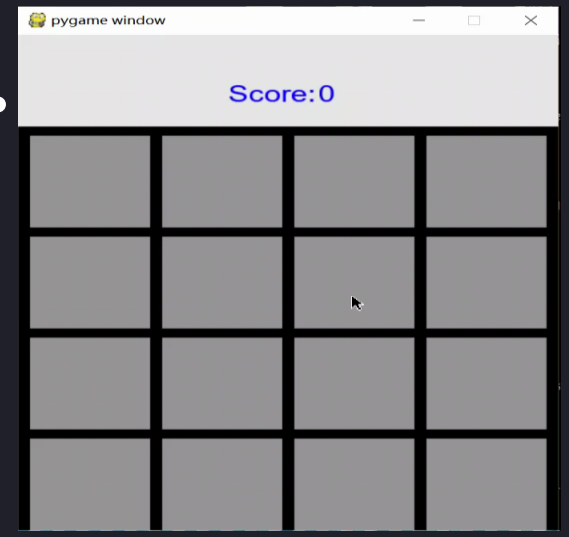
Game Introduction

01

# What is 2048 & How to play it

Operation

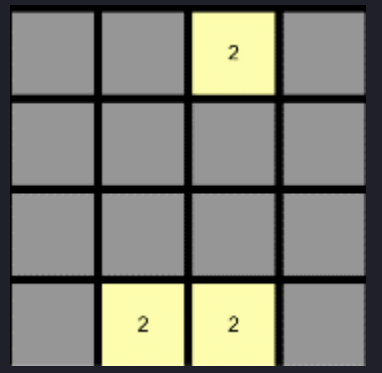



Human player

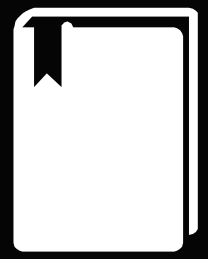
Right--Down--Left--Up

Score += New summed numbers




Initial


final



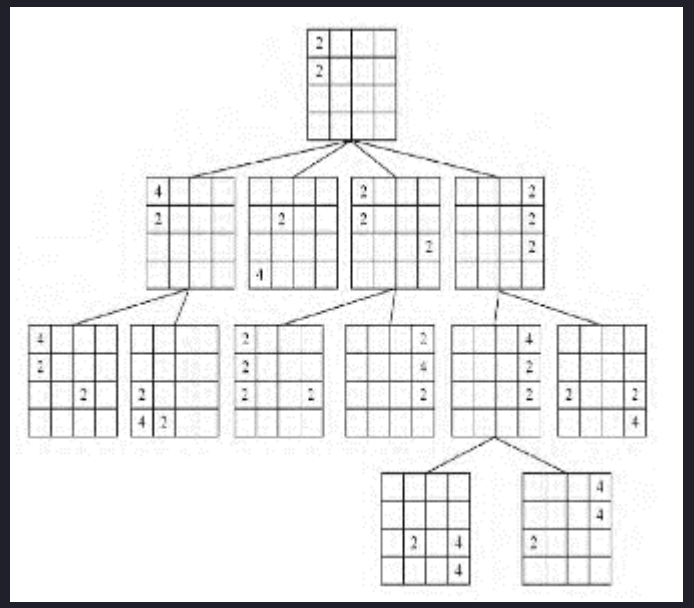
# PART TWO

## Difficulties and Overview

02

# Difficulties Analysis

## 1 Large Game Tree



$S_{n+1} = 4S_n x2\{B\}$

S: number of nodes in one layer
{B}: number of blank tiles

## 2 Situation Assessment


**High score ⇔ Criticality**

- ☐ Numerical coherence
- ☐ Monotonicity
- ☐ Number of blank tiles
- ☐ Maximum

## 3 Randomness

New tile is generated randomly.

- ☐ **Unpredictable**
- ☐ **Unrepeatable**

## 4 Human players

Even for human, 2048 is not a easy game. Among ordinary players, one who can reach 1024 or 2048 are considered master.

02

# Overview

1

AI:

**Receive :** a current state

**Return :** a best operation

2

GAME:

**Receive :** an operation

**Return :** a current state

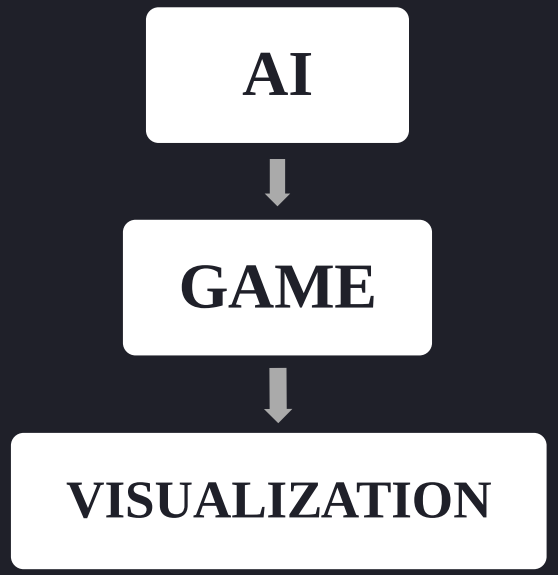
3

Visualization:

**Receive :** a current state

**Return :** a image

## Flowsheet

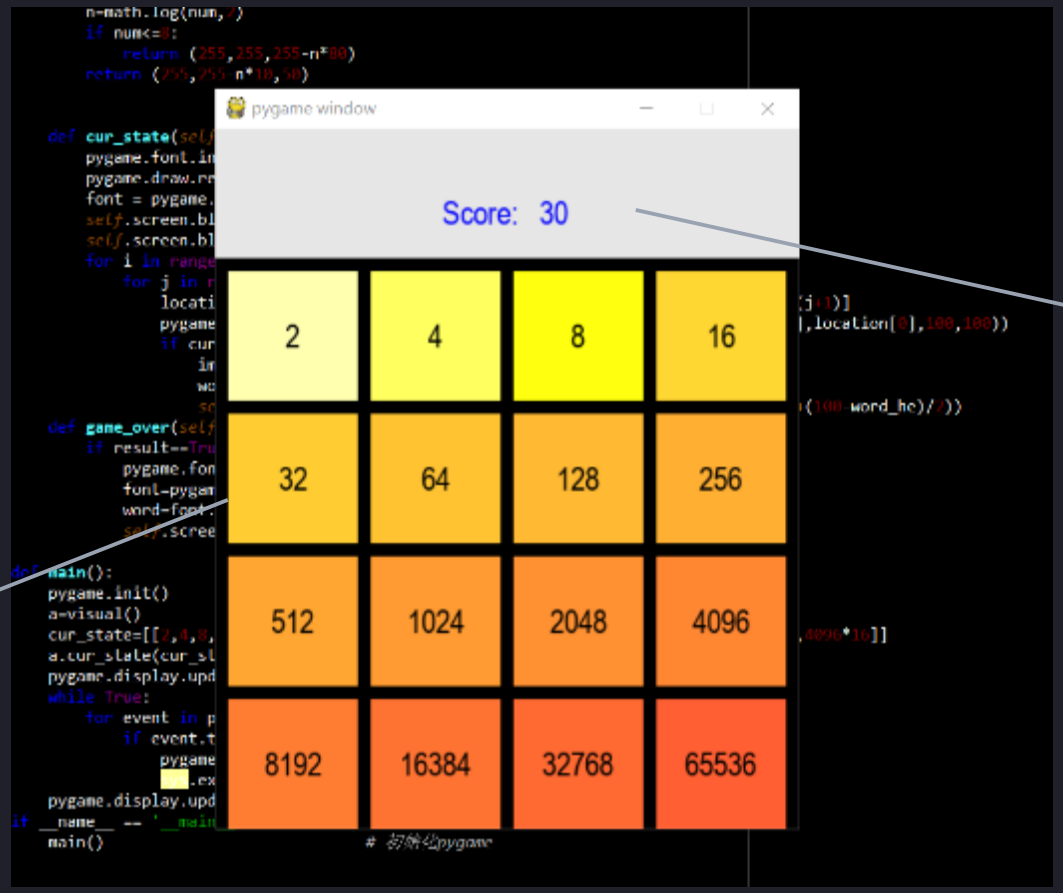


# PART THREE

## Visualization

03

# Visualization




Score

4x4
Matrix

03

# Visualization


**The Start Interface**

Choose the playing mode:

AI or Human player


**The End Interface**

When game is over or

player presses key ‘e’


**Selection Interface**

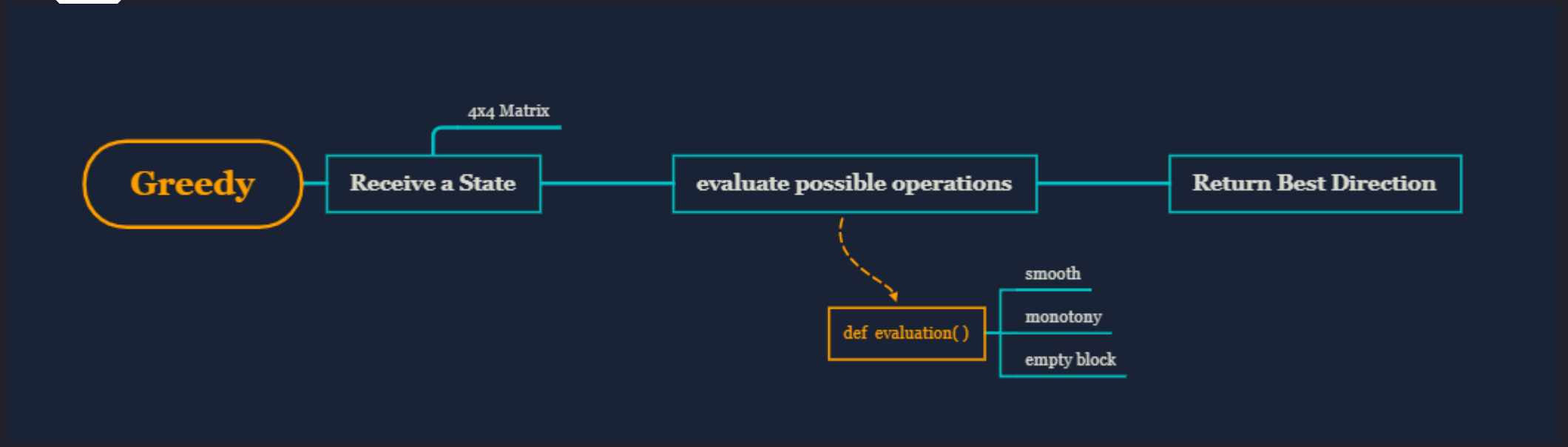
When game reaches

2048 for the first time

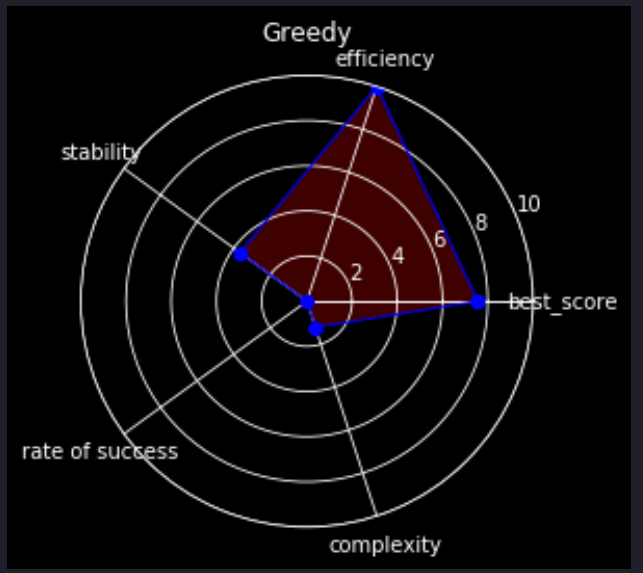
# PART FOUR

# AI Algorithms

# 4.1 Greedy

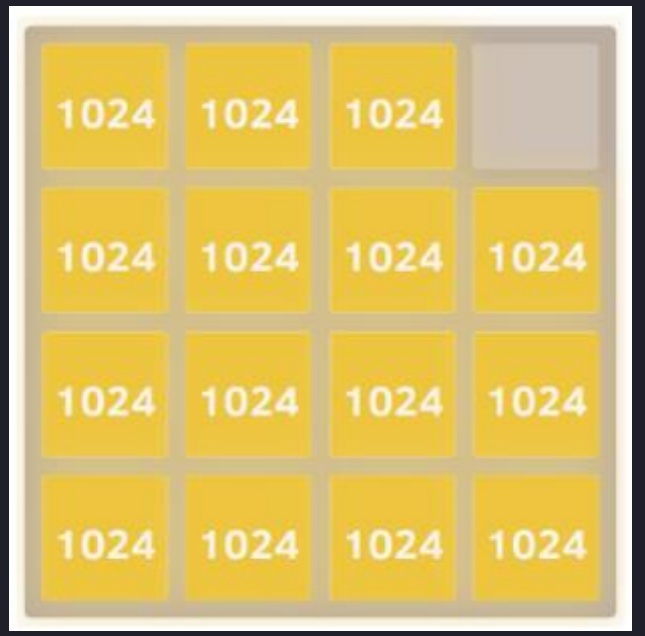



- Best Game: 1024
- Average Score: 5839.9
- Average time for a step: 0.0007s



# 4.1 Evaluation Function

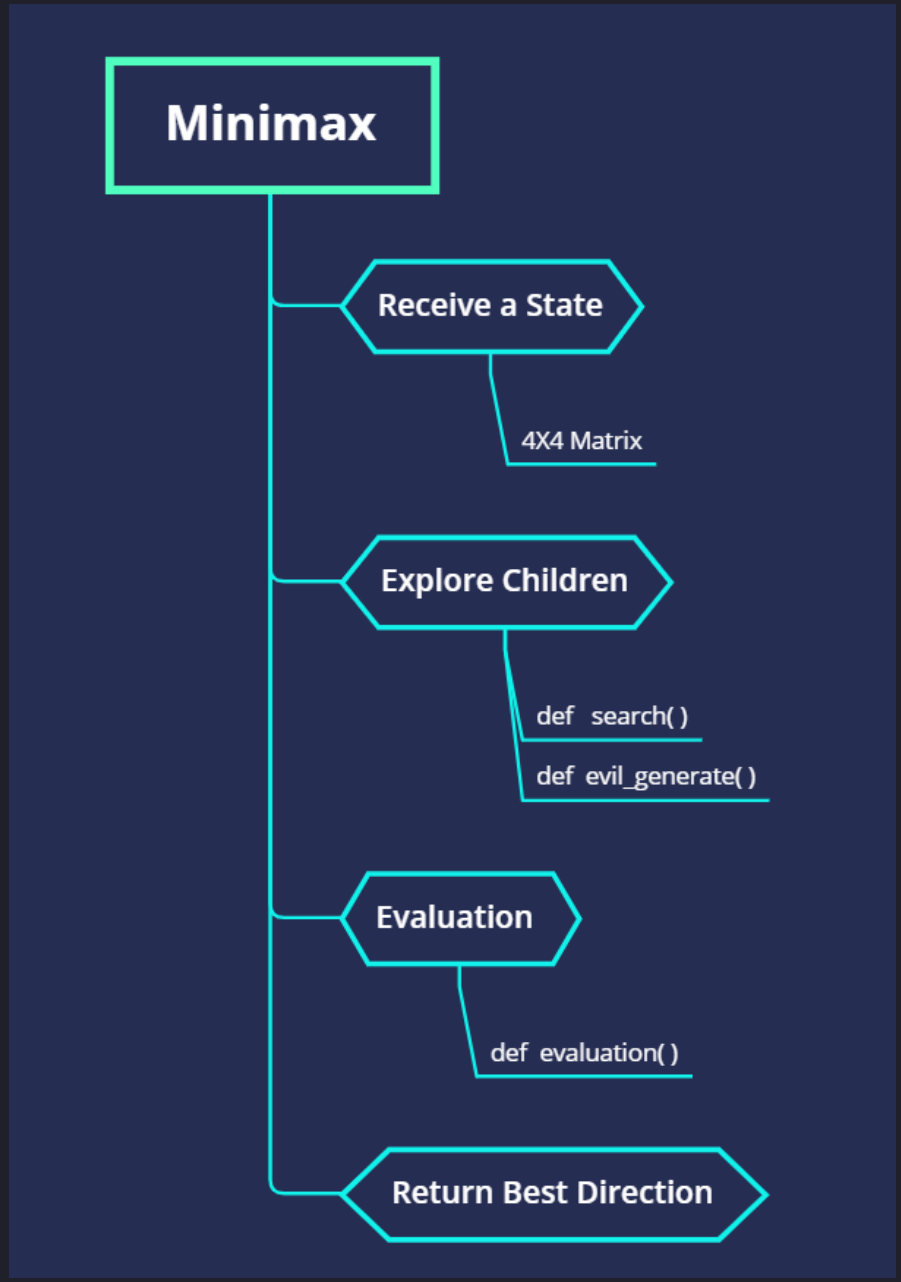
- ✓ Smooth
- ✓ Monotony
- ✓ Number of Empty space




4.2

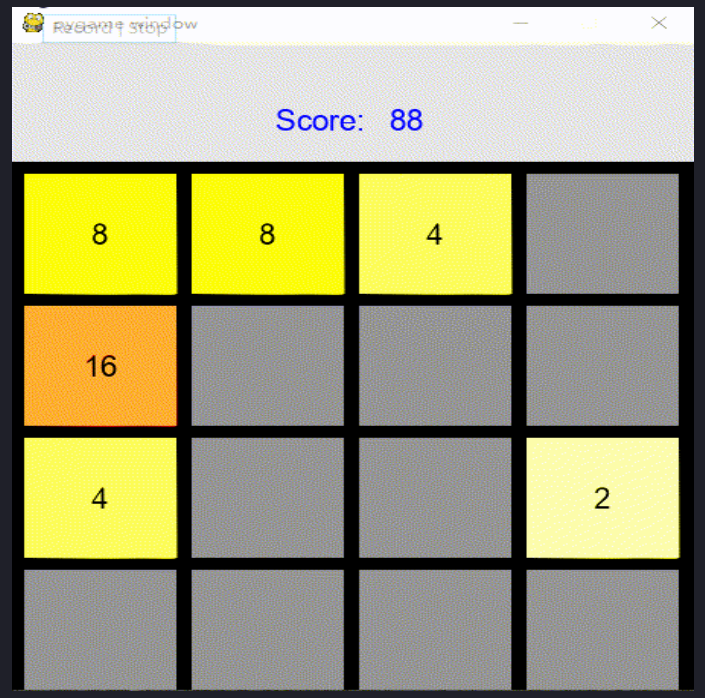
# Minimax

- 1 vs 1 Battle Game
- Pessimistic Algorithm

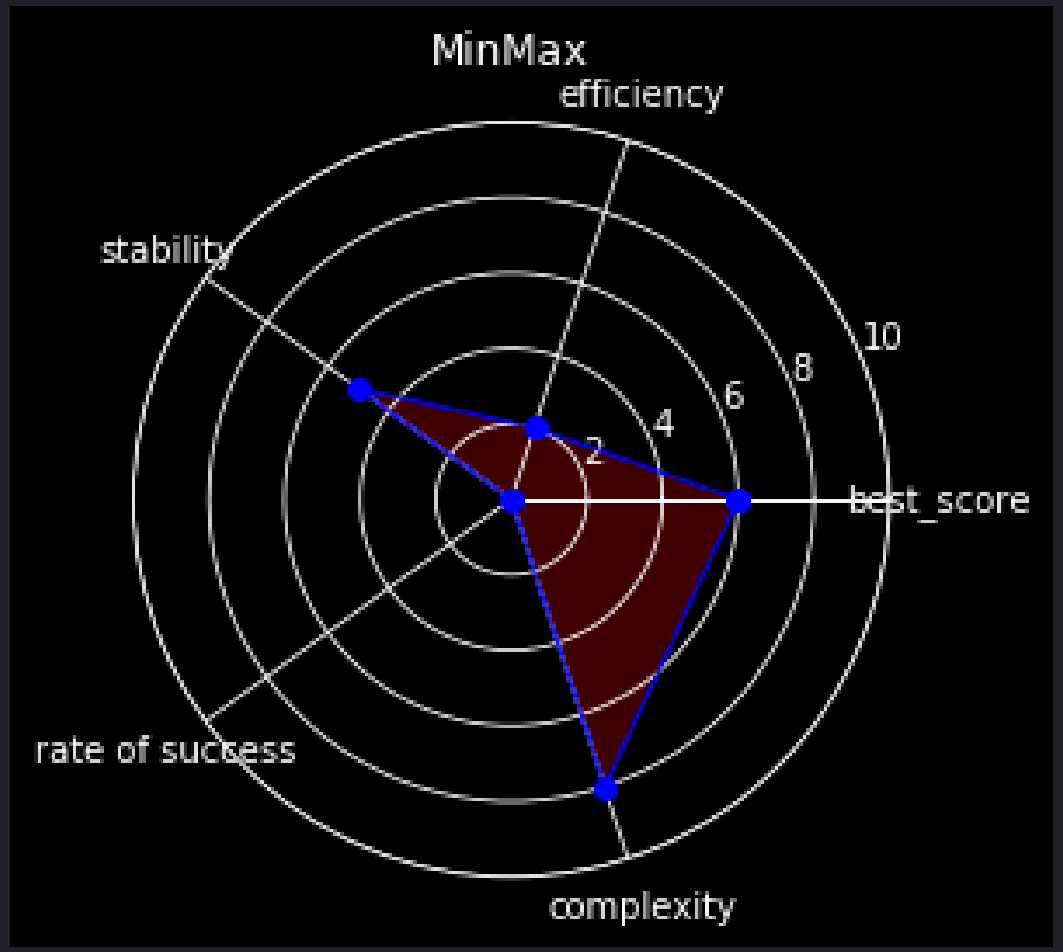


# 4.2 Minimax

- Best Game: 1024
- Average Score: 4785.4
- Average Time for a Step: 1.989s




Played by minimax agent



4.2

# Minimax

## 1,Bad Evaluation Function

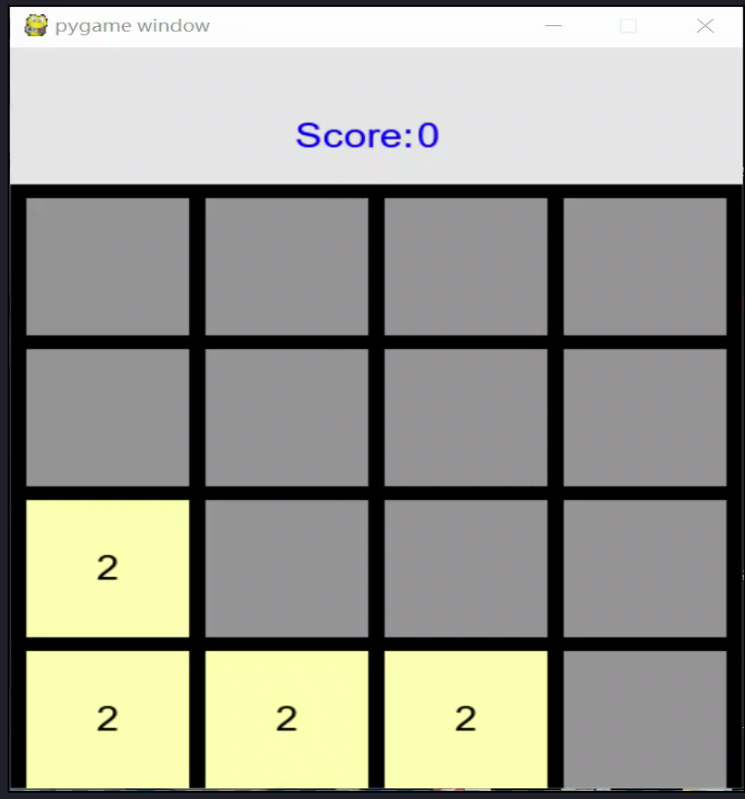
- X Too general standard
- X Bad weight of standards

## 2, Complexity

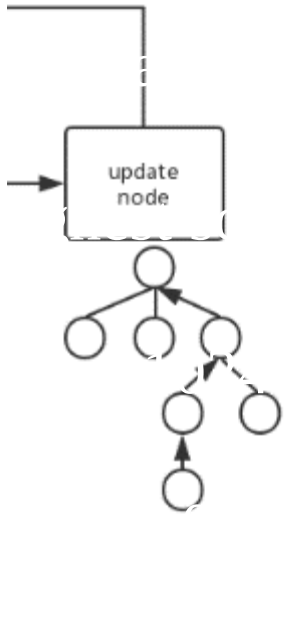
- X $O(2^{2(n+1)})$
- X DFS

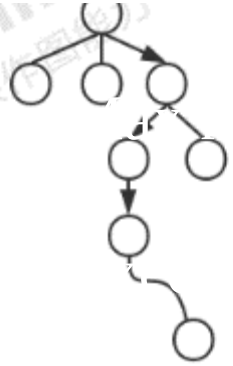
# 4.3 MCTS Algorithm core elements

Played by MCTS agent




| No | score | time | steps | frequwncy | max |
|---|---|---|---|---|---|
| 1 | 7036 | 323.509 | 478 | 0.6767972 | 512 |
| 2 | 3104 | 177.056 | 250 | 0.7082242 | 256 |
| 3 | 12352 | 504.6365 | 731 | 0.6903373 | 1024 |
| 4 | 2900 | 175.4609 | 228 | 0.7695654 | 256 |
| 5 | 3276 | 141.3431 | 271 | 0.5215611 | 256 |
| 6 | 7484 | 247.2539 | 522 | 0.4736665 | 512 |
| 7 | 4332 | 152.6242 | 286 | 0.5336512 | 512 |
| 8 | 12452 | 372.4305 | 740 | 0.5032845 | 1024 |
| 9 | 10696 | 320.82 | 608 | 0.5276644 | 1024 |
| 10 | 12136 | 381.8343 | 714 | 0.5347819 | 1024 |
| 11 | 6864 | 259.5156 | 465 | 0.558098 | 512 |
| 12 | 2932 | 133.0771 | 235 | 0.5662857 | 256 |
| 13 | 3048 | 149.8186 | 251 | 0.5968867 | 256 |
| 14 | 7068 | 255.1662 | 483 | 0.5282945 | 512 |
| 15 | 7072 | 245.4807 | 480 | 0.5114182 | 512 |
| 16 | 4872 | 186.9844 | 336 | 0.5565013 | 512 |
| Average | 6726.5 | 251.6882 | 442.4 | 0.5785636 | |








## Result

We find when c=0.3, the AI get highest score.

Compared with Greedy, MCTS can get more information about the future benefit and do a wiser choice.

Advantage:

Disadvange:

Tree search causes larger time complexicty.

and the range of vi(or vp) can be out of control.

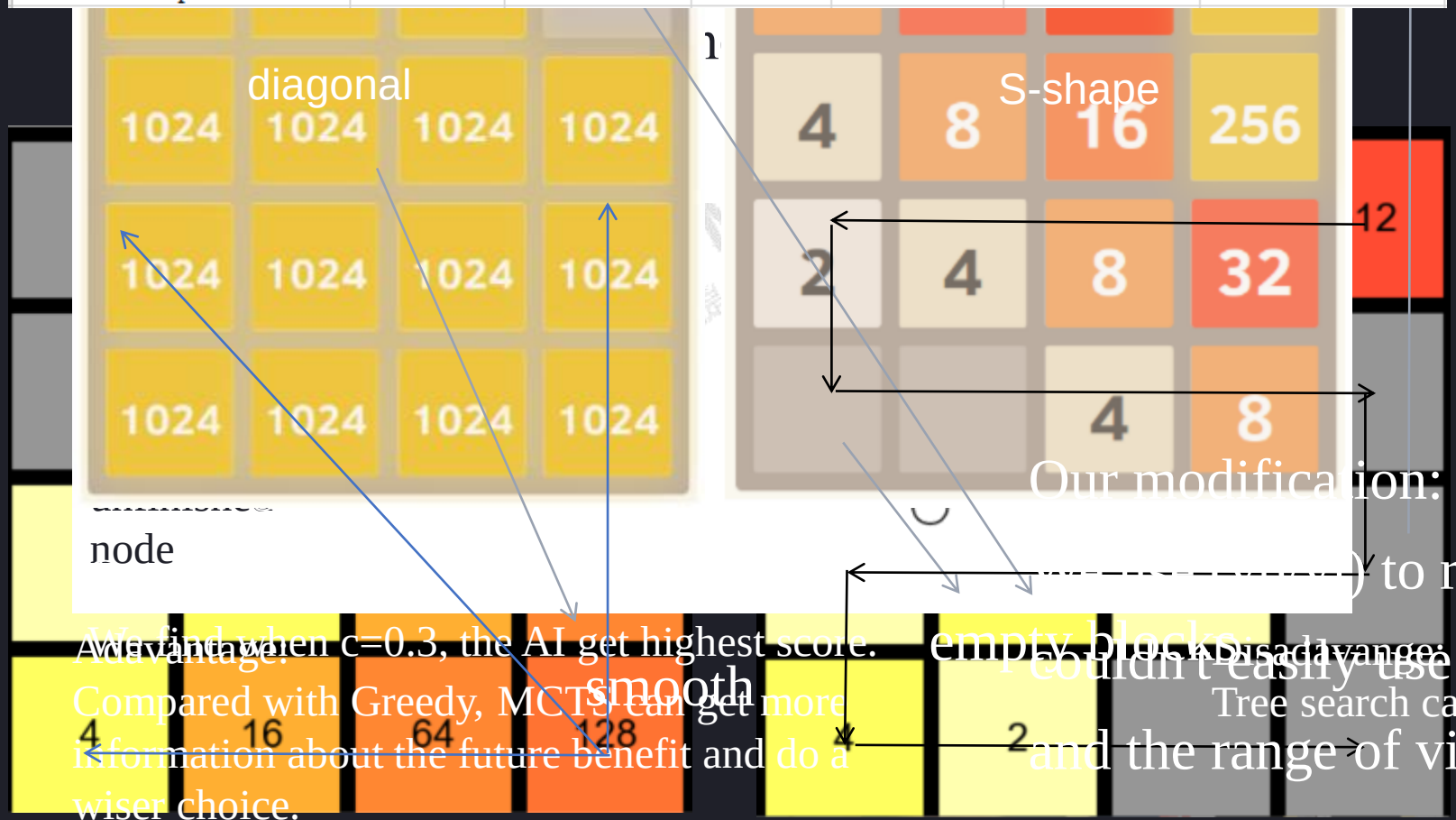
4.3

## Snake-shape Heuristic function score elements

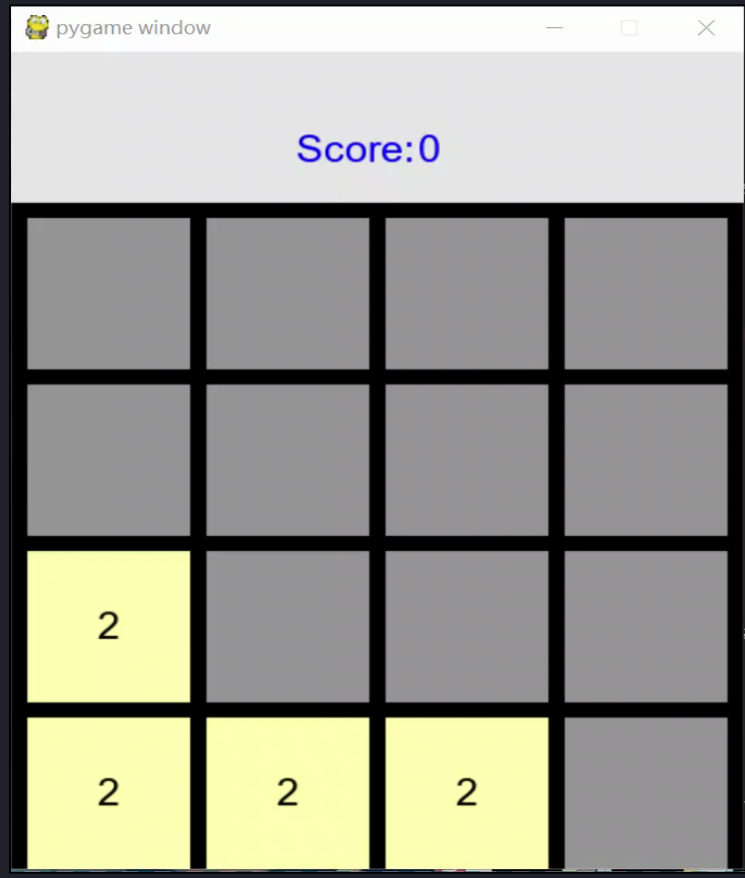
max number

monotony

| Heuristic function | score | time | steps | frequency | max number | highest score |
|---|---|---|---|---|---|---|
| Diagonal-shape | 6726.5 | 251.688195 | 442.375 | 0.5785636 | 1024 | 15908 |
| Snake-shape | 6600.25 | 240.737748 | 433.125 | 0.5612331 | 1024 | 12468 |




Played by MCTS agent




Advantage:
Compared with Greedy, MCTS can get more information about the future benefit and do a wiser choice.

We find when c=0.3, the AI get highest score.

Disadvange:
Tree search causes larger time complexity.

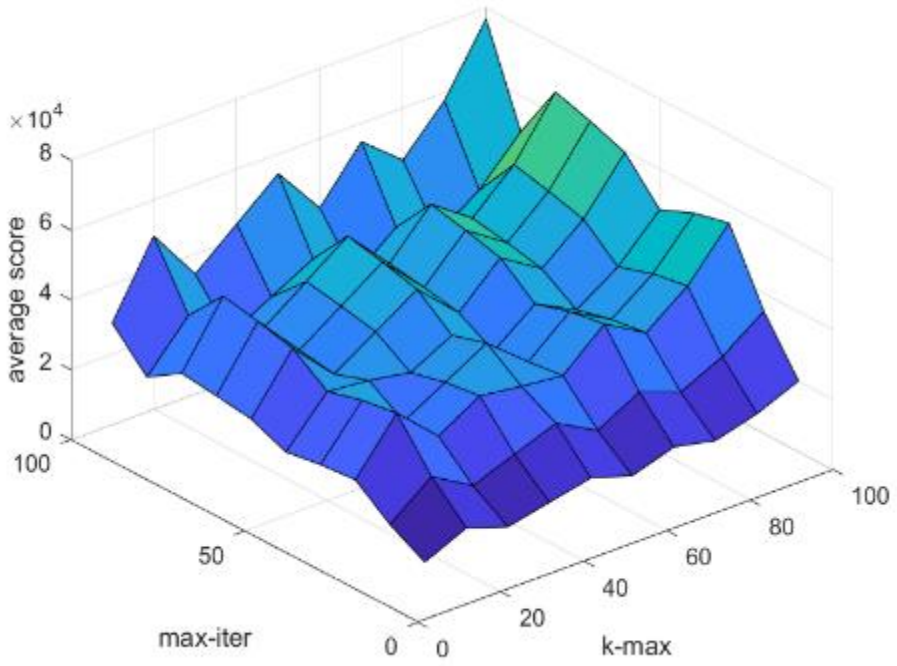
couldn't easily use +1,0,-1 to assess the game state and the range of vi(or vp) can be out of control.

# 4.4 Monte Carlo Method

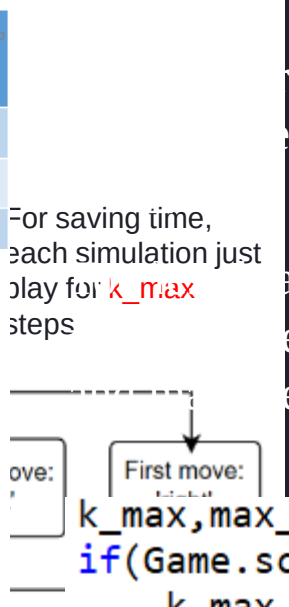
Use randomness against randomness!

k_max and max_iter adjustment

| k_max | max_iter | average score | time(per game) | high score | low score | 2048 rate |
|---|---|---|---|---|---|---|
| 10 | 10 | 8638.12 | 4.095 | 16184 | 1372 | 0% |
| 10 | 100 | 27141.76 | 111.895 | 55880 | 6912 | 71% |
| 10 | 200 | 29831 | 237.8 | 67748 | 7200 | 80% |




Algorithm Process

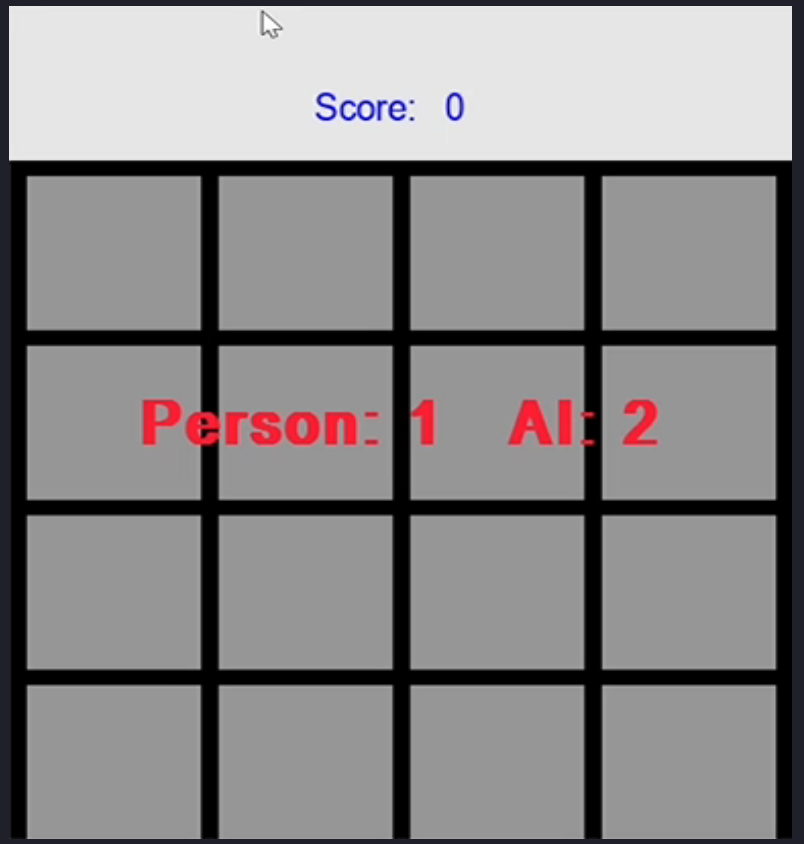



rent score:
k_max and

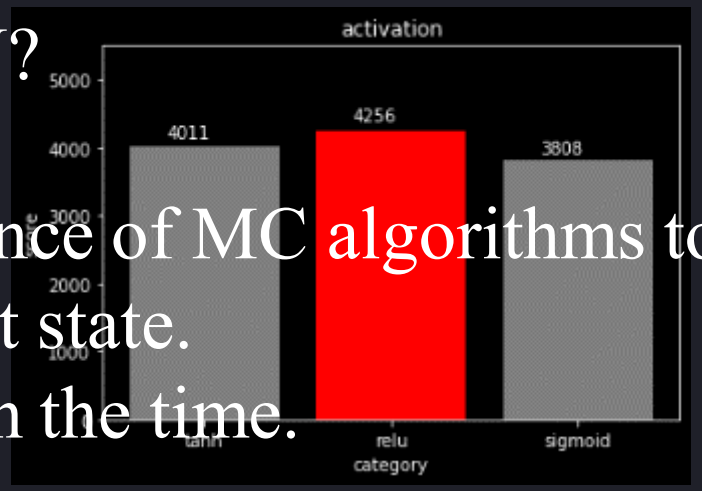
ank tile:
e k_max
er

```
k_max,max_iter=10,100
if(Game.score<5000):
    k_max,max_iter=10,10
elif(Game.score<10000):
    k_max,max_iter=8,50
elif(Game.score<15000):
    k_max,max_iter=10,70
else:
    k_max,max_iter=15,200
if(Game.empty()<=2):
    k_max,max_iter=15,300
```

Played by MC agent



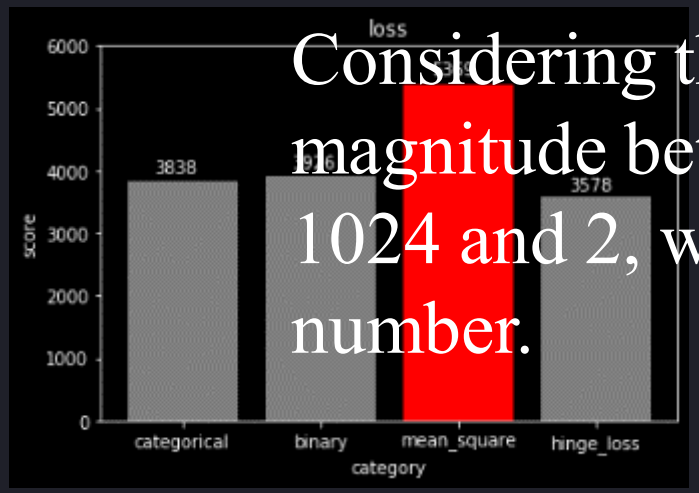
# 4.5 Neural Network

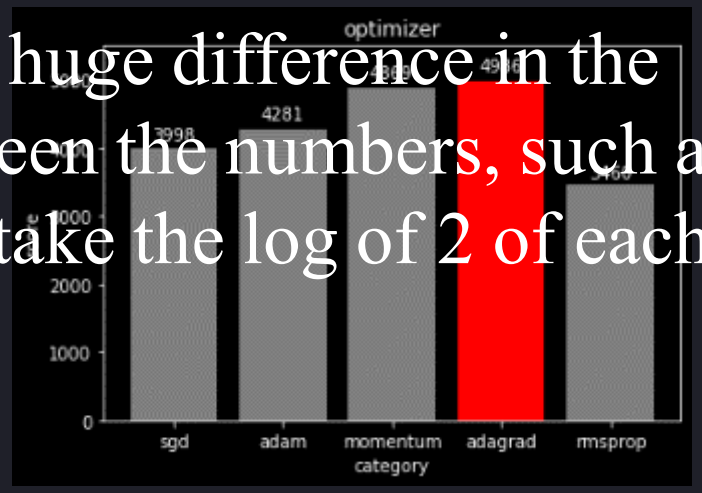




Why we use NN?

--Learn the essence of MC algorithms to judge the current state.

--Further shorten the time.




Considering the huge difference in the magnitude between the numbers, such as 1024 and 2, we take the log of 2 of each number.

Played by NN agent

4.5

# Result of NN

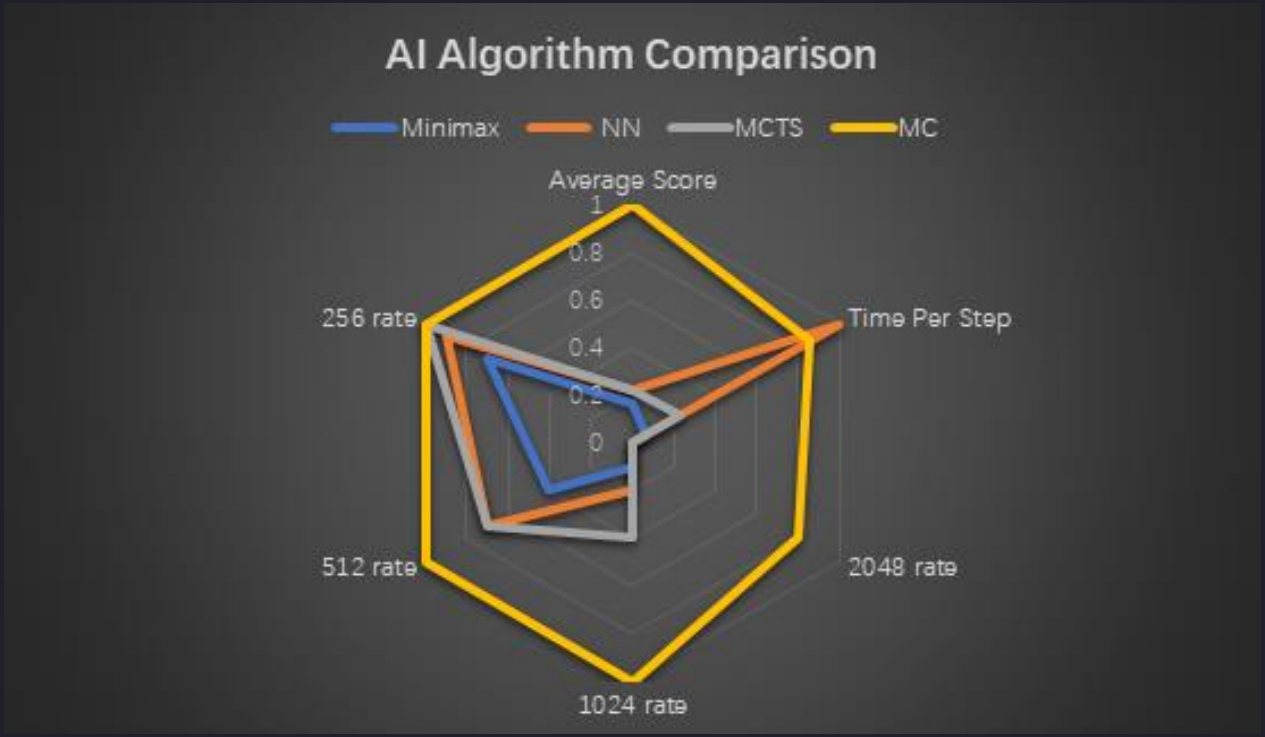
| Algorithms | Average Score | Time/ Step(s) | The rate of success | | | |
|---|---|---|---|---|---|---|
| | | | 2048 | 1024 | 512 | 256 |
| **Random** | 1063.3 | 0.0001 | 0 | 0 | 0 | 0.09 |
| **NN** | 6656.4 | 0.1385 | 0 | 0.2 | 0.7 | 0.9 |
| **MC** | 29831.4 | 0.1600 | 0.8 | 1 | 1 | 1 |

Comparing to MC, it’s quicker but the score and the probability of higher numbers are much lower.

04

# Summary

## MC wins!




MC algorithm can get a very high score at a relatively high speed, so we choose it to be the best algorithm playing the 2048 game.

In the future:

- ✓ Heuristic function can be optimized.
- ✓ Hyperparameter optimization
- ✓ Better Neural Network

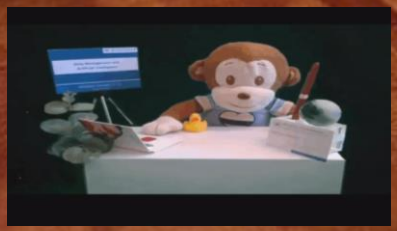










# Thanks for listening!