TECHNICAL REPORT

Strategies used by the computer player.

BORNET Floryan A2
BRENDLE Corentin A2
IMMER Alex A1
KHNAGUI Adam A1
Group 12
IT department
Academic Year 2023/2024

24/05/2023

Contents

1. Introduction. 3
2. Principle Description 3
    a. NAIVE AI. 3
    b. SMART AI. 4
3. “IT” assessment. 4
4. Evaluation of efficiencies. 5
    a. NAIVE AI vs NAIVE AI 5
    b. SMART AI vs NAIVE AI 5
    c. SMART AI vs SMART AI 6
5. Input files. 6
    1) Player X win. 6
    2) Player O win. 7
    3) Player X stuck, pass turn. 7
    4) Draw Game. 8

1. Introduction.

This report aims to present an analysis of two distinct AI (Artificial Intelligence) strategies utilized by the computer player in the game. The focus will be on the SMART and NAIVE strategies, delving into their underlying principles, computational performance, and effectiveness in gameplay.

The SMART strategy is designed to make informed decisions based on a comprehensive evaluation of the game state. It employs techniques such as decision trees to predict and counter the opponent's moves. Conversely, the NAIVE strategy relies on simpler, heuristic-based decision-making, focusing on immediate, short-term gains without a deep analysis of future consequences.

To provide an evaluation, this report will assess the computational aspects of each strategy, including code complexity, memory usage, and average execution time. Furthermore, the effectiveness of each strategy will be measured by their win rates in simulated games, accompanied by an analysis explaining the observed performance differences.

Additionally, we will examine various input files that were used to test and compare the strategies. These files, along with their intended purposes and outcomes, will be described and illustrated with screenshots to demonstrate their results. This evaluation will offer insights into the strengths and limitations of each AI strategy, contributing to a better understanding of their practical applications in game development.

2. Principle Description

a. NAIVE AI.

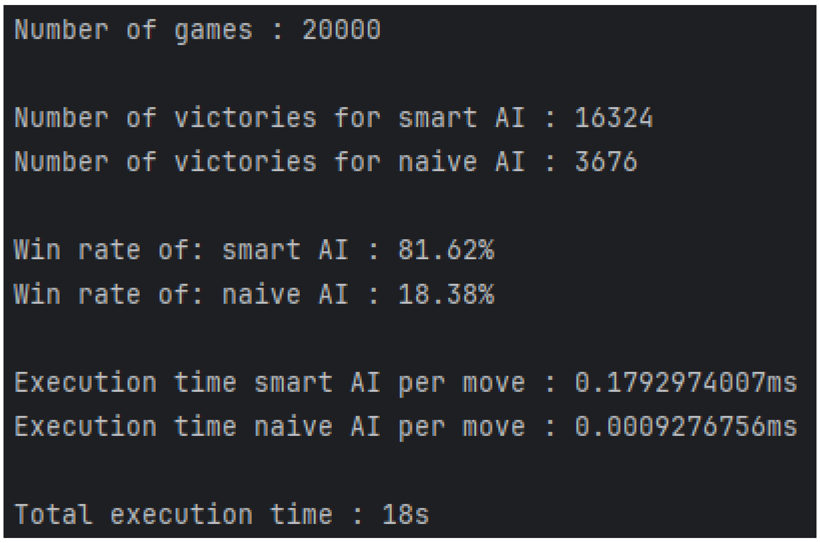
The 'HoleNaiveDecider' uses a simple and naive strategy for move selection, focusing on randomness and basic validation. It checks if the current player can make a move from a specific origin point, validates this move, then retrieves all possible moves from that point. It adds these moves as nodes in a tree, assigning a value of 10 to nodes for movements on the same column and a value of 0 to nodes for diagonal movements. A movement is then chosen among the nodes with a value of 10, and if there are none, a movement among the nodes with a value of 0 is selected.

The logic ensures that the game can continue with valid moves or stop if no moves are possible. The approach does not involve in-depth analysis of the game, hence the term "naive".

b. SMART AI.

The ‘HoleSmartDecider’ employs a more sophisticated strategy for move selection, incorporating both offensive and defensive tactics. It evaluates all possible moves from the current position, assigns points based on their strategic value, and selects the best move to optimize the player's chances of winning while minimizing the risk of losing.

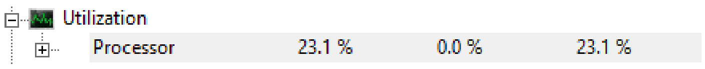
3. “IT” assessment.



```
Number of games : 20000

Number of victories for smart AI : 16324
Number of victories for naive AI : 3676

Win rate of: smart AI : 81.62%
Win rate of: naive AI : 18.38%

Execution time smart AI per move : 0.1792974007ms
Execution time naive AI per move : 0.0009276756ms

Total execution time : 18s
```

Running 20,000 games between the SMART AI and the NAIVE AI reveals that the average execution time per move for the SMART AI is approximately 0.18ms, while for the NAIVE AI, it is about 0.0009ms. This discrepancy can be attributed to the difference in complexity between the two AIs, with the SMART AI employing more advanced calculations and strategies.

We then notice that there's a significant difference in execution time in favor of the naive AI, with the naive AI being approximately 200 times faster. However, in terms of winning efficiency, the smart AI is 80% more effective. Additionally, in terms of utilization, a latency of 0.18ms per move is negligible.



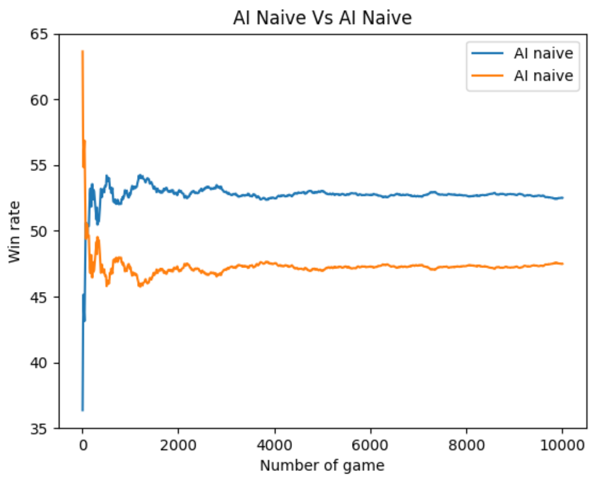
Utilization			
Processor	23.1 %	0.0 %	23.1 %

The observed peak CPU utilization of 23.1% during the generation of 20,000 games indicates that the computational demand imposed by the SMART and NAIVE AIs is relatively modest. Although the SMART AI has a higher average execution time per move (0.18 ms) compared to the NAIVE AI (0.0009 ms), the overall system performance remains efficient.

Overall, the total execution time for 20,000 games is around 18 seconds.

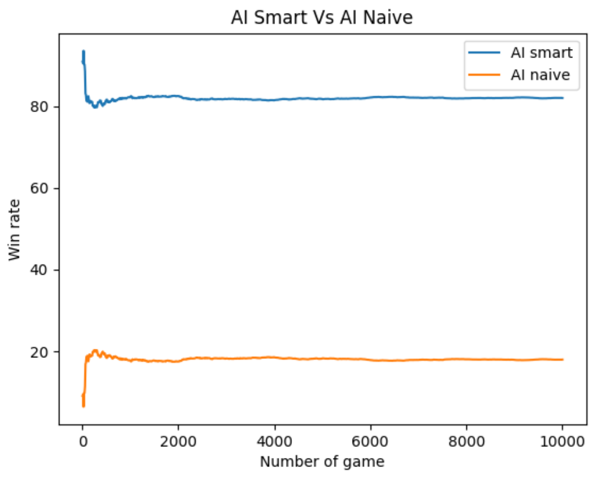
4. Evaluation of efficiencies.

a. NAIVE AI vs NAIVE AI



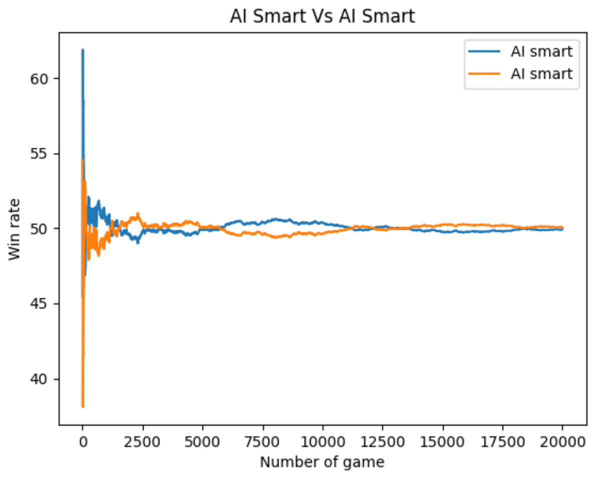
In a sample of 10,000 games, initially, the second AI wins most of the time. However, over time, the AI that plays first begins to win more frequently. The reason for this is straightforward: the first AI has the advantage of making the initial move, allowing it to block a move right from the start. Despite this, there is a noticeable parallelism in the win rates, which can be attributed to both AIs employing the same game strategy.

b. SMART AI vs NAIVE AI



In a sample of 10,000 games, we observe a significant difference in the win rates between the naive AI and the smart AI in the initial games. This disparity is due to the contrasting strategies: naive AIs make simple moves, while smart AIs employ predictive strategies to avoid critical mistakes. As the number of games increases, a symmetry in the win rates emerges, which can be attributed to the large sample size.

c. SMART AI vs SMART AI



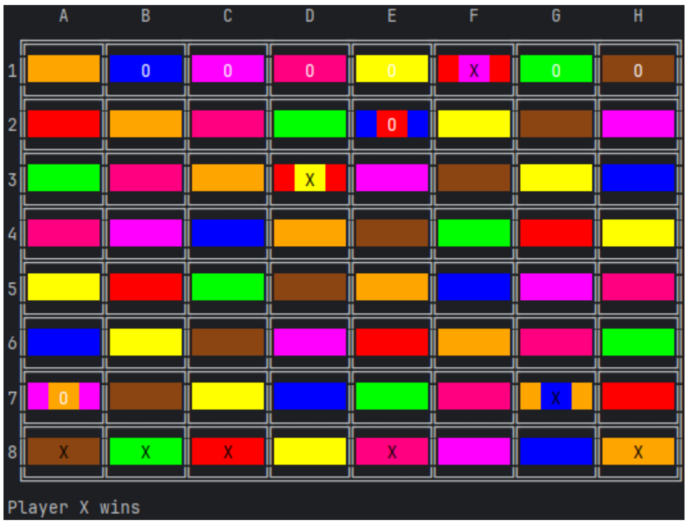
In a sample of 20,000 games, as seen above for naive AI Vs naive AI, at the start there is a significant difference in the win rates of the two smart AIs. However, as more games are played, this difference fades. On the other hand, the symmetry of victory rates is less perceptible, and the difference in victory rates is very small. One reason for this is that both AI use a strategy that predicts the opponent's next move.

5. Input files.

1) Player X win.

Moves:

1. D8
2. D3
3. E2
4. G7
5. A7
6. F1



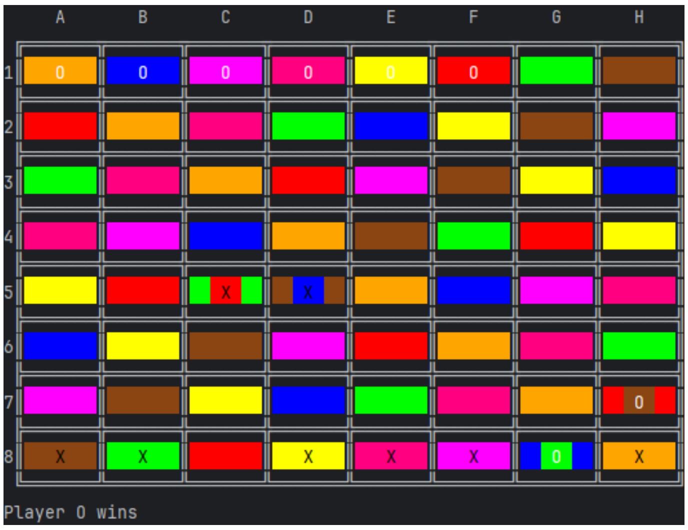
```
    A    B    C    D    E    F    G    H
1        0    0    0    0    X    0    0
2                       0
3                  X
4
5
6
7   0                             X
8   X    X    X         X              X
Player X wins
```

This sequence of moves demonstrates a scenario where Player X achieves a winning position. Each move progresses towards a strategy that leads to a win for Player X.

2) Player O win.

Moves:

1. G8
2. D5
3. H7
4. C5
5. G8



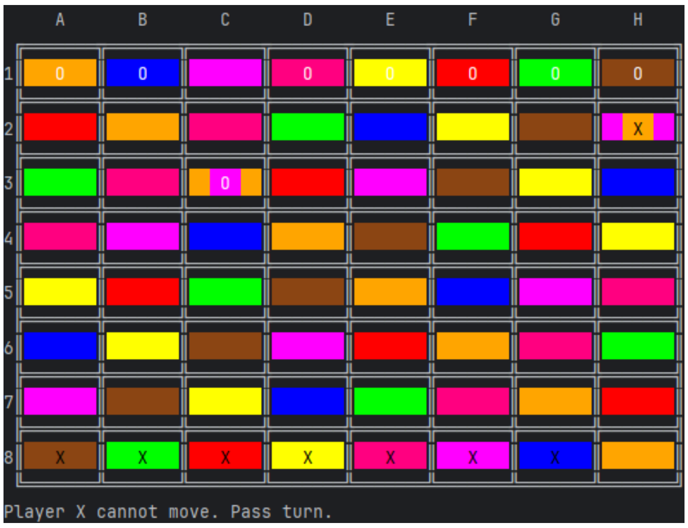
```
Player O wins
```

This sequence of moves demonstrates a scenario where Player O achieves a winning position. The moves illustrate how Player O navigates the board to secure a win.

3) Player X stuck, pass turn.

Moves:

1. H8
2. H2
3. C3



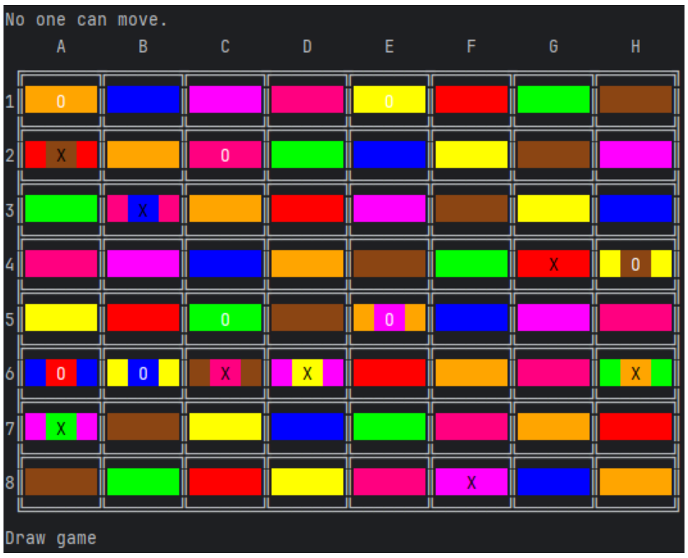
```
Player X cannot move. Pass turn.
```

This sequence shows Player X reaching a position where no valid moves are available, forcing them to pass their turn. This situation highlights the game's handling of a player being stuck.

4) Draw Game.

Moves:

1. G8
2. A2
3. F2
4. G6
5. F7
6. D4
7. H2
8. B2



```
No one can move.
   A     B     C     D     E     F     G     H
1  0                       0
2  X           0
3        X
4                                      X     0
5              0           0
6  0     0     X     X                       X
7  X
8                                X
Draw game
```

This sequence of moves demonstrates a scenario where the game reaches a state where no player can make a valid move, leading to the end of the game. This input file showcases the game's ability to detect and handle a stalemate situation.