Dots and Boxes: A "smart" opponent.

Roberto Falcone

r.falcone13@studenti.unisa.it

1 Introduction

1.1 Background and Purpose of the Project

This report describes the project aimed at realizing an al- gorithm capable of efficiently playing a game, with the goal of achieving good performance against a human adversary. The core of the algorithm is based on the minmax decision tree search technique and alpha-beta pruning. This artificial intelligence approach nowadays is widely used to address challenges concerning games and other applications that require optimal strategic reasoning.

1.2 The Relevance of Game Theory.

Game theory is a fundamental field in artificial intelligence, in which the strategies and behaviors of rational agents in com- petitive or cooperative situations are studied. Game theory offers a mathematical and formal approach to analyzing decision-making situations, identifying the best strategies, and understanding the outcomes that emerge from complex interactions among players. This theory is of fundamental importance in the development of intelligent algorithms because it allows us to model the rational behavior of agents and find the best actions to take.

1.3 Applications of Games in Complex Problems

Games are not only a source of entertainment, but also provide a valuable study platform for developing and evaluating artifi- cial intelligence algorithms. The wide range of games, from deterministic to nondeterministic, from perfectly informed to partially informed, allows for a multiplicity of scenarios that reflect the challenges present in many real-world problems.

1.4 The Choice of the Game

As a case study for this project, the game "Dots and Boxes" was selected, which offers a simple yet complexity-rich platform for testing the effectiveness of the alpha-beta pruning algorithm.

2 Background

2.1 Dots and Boxes

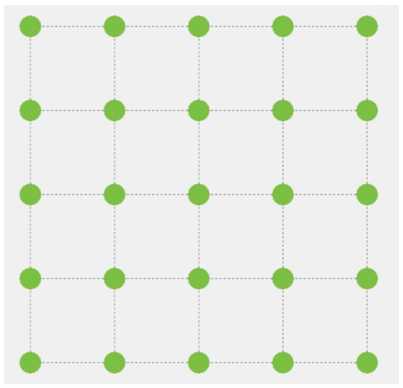


Figure 1: Empty Dots and Boxes board.

"Dots and Boxes" is a paper and pencil game for two or more players. Starting with an empty grid consisting only of dots, players can, in turn, insert a horizontal or vertical line between two dots not yet connected. When either player draws the fourth side of a square he gets a point and marks the square as conquered. The game ends when it is no longer possible to draw lines within the grid, and the player with the most points, that is, the one who has conquered the most squares, wins.

There are no characters or objects in the game, which makes the game **abstract**. Since there are no random or random elements in the game, the outcome of each move is entirely predictable based on the current situation on the grid and the players' actions, so it is **deterministic**. Each player can see the complete grid and the lines already drawn. This means that both players have a clear and complete view of the game situation, which affects their tactical and strategic decisions, from this it follows that the game is **full-information**.

There are two variations of the game: one stipulates that once a square is completed, the turn remains with the player who earned the point, who can then complete other squares as long as he can. The other, on the other hand, maintains alternating turns between players, regardless of who wins a point during the game.

The first variant was chosen to be analyzed because it was considered certainly more challenging.

2.1.1 The game in detail

Most of the arguments in this section come from the studies of University of California mathematics and computer science professor ***Elwyn Berlekamp***, who has devoted his career to the study of games such as "*Fox and Geese*," "*Go,*" and, indeed, "*Dots and Boxes.*"

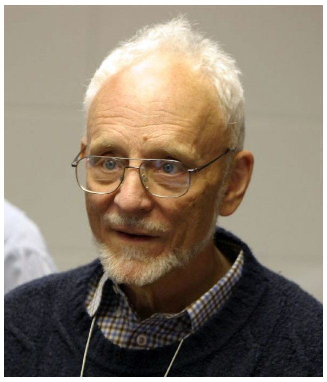


Figure 2: Elwyn Berlekamp

The game, according to Professor Berlekamp, has a couple of mechanics that make it quite unique and complex to analyze: first, the possibility of continuing to move once a square has been conquered: second, the two players' goal is to score as many points as possible and not to make the last move, however, as we will see later in the report, these two goals are implicitly linked.

First of all, there is a need to give the definition of a **chain**. When a player completes a square belonging to a chain, he or she captures that square and can continue to capture squares throughout the length of the chain. Chains can vary in length, from a single cell to a longer sequence of adjacent squares, and can change direction or even in- terse. The "hidden" objective of the game, as we shall see, is to create long and complex chains to capture the maximum number of squares and score more points than the opponent.

2.1.2 The two phases of the game

The game is divided into 2 phases:

- The first phase is the phase in which players try to create chains and avoid giving away points to the opponent (thus marking the third side of a square, a move that will cause the opponent to complete the square itself in addition to continuing with the usual move).

- During the second phase, on the other hand, players are only concerned with com- plete the chains formed, trying to win the longest chains.

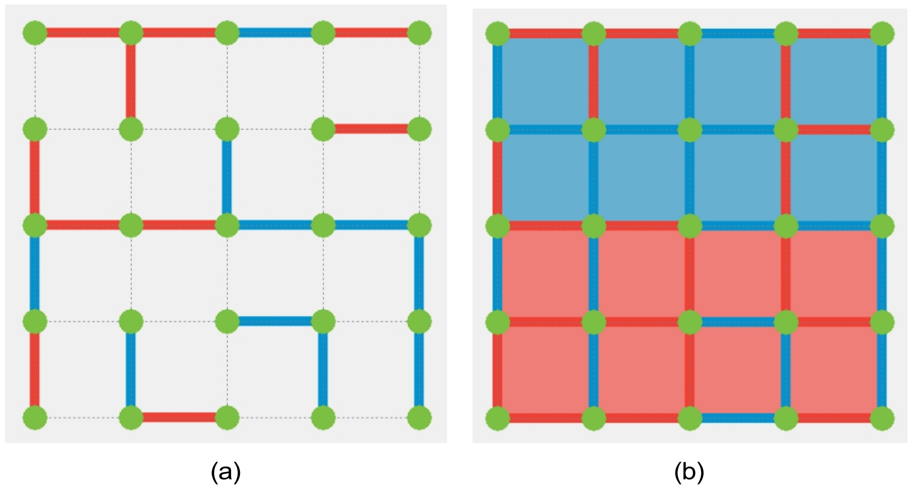


Figure 3: Figure (a) shows the end of the first phase; Figure (b) shows the end of the second phase

This division into two distinct phases is not easily recognized by a novice player.

The more "intuitive" stage of the two turns out to be the first. In this phase the goal, in the eyes of a novice player, is to try not to give away points to the opponent by scoring the third side of a square.

The "manners" at this stage, therefore, are:

- If possible, mark the fourth side of a square
- Avoid marking the third side of a square

For an experienced player, on the other hand, the most interesting phase is precisely the second one, because it is in view of this phase that the latter tries to build more or less long chains.

2.1.3 Long chain rule

There is a rule that, if exploited properly, can bring the player who applies it closer to victory, namely ***the long-chain rule***: suppose the playing field is a rectangle of m rows and n columns and thus consists of m - n squares. If both m and n are even, then the first player should try to make the number of long chains odd. If even one of m or n is odd, then the first player should try to make the number of long chains even. Long chains are defined as any chain that exceeds the length of 3 squares.

Moreover, in the case of a grid with $m = n$, the two dimensions depend strictly on the number of points with which it was constructed:

$$m = n = \#dots - 1$$

So if the number of points is odd then m and n will be even, just as if the number of points is even then m and n will be odd.

Formally summarized, the number of desired long chains is:

- EQUAL: for the first player if the number of points is even, for the second player if the number of points is odd
- Uneven: for the first player if the number of points is odd, for the second player if the number of points is even

3 Implementation

The aim of the present project was the design and implementation of an algor- itm that demonstrates effective game skill in the specific context. The algorithm must demonstrate the ability to execute the game with precision and proficiency while avoiding ceding advantages to the opponent during the chain-building phase. It is required that the algorithm adhere to the game conventions previously outlined while aiming at the goal of creating an optimal number of long chains, thus coming close to winning the game. These criteria are detailed in section **2.1.2** and **2.1.3** of the paper. Of course, the algorithm must finally demonstrate the ability to accumulate a maximum possible quant- itative number of points.

3.1 GUI

For the implementation of the graphical user interface (GUI) in the context of this project, the solution proposed by Aqueel Anwar was adopted, which is available in the GitHub repository Dots-and-Boxes by aqueelanwar.

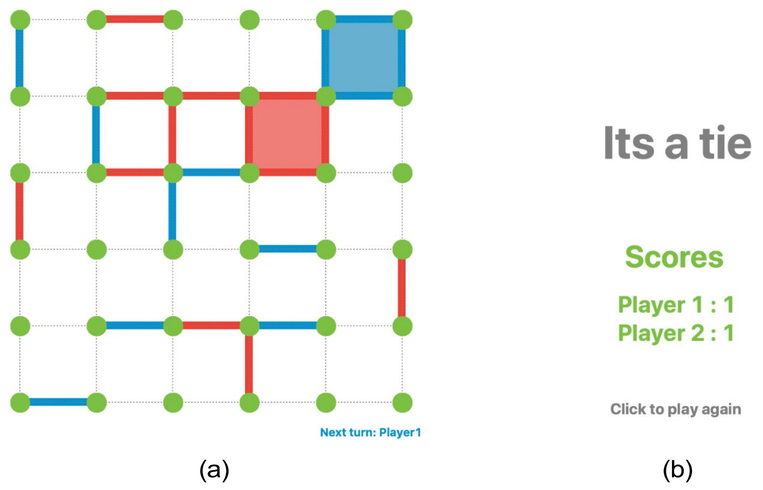
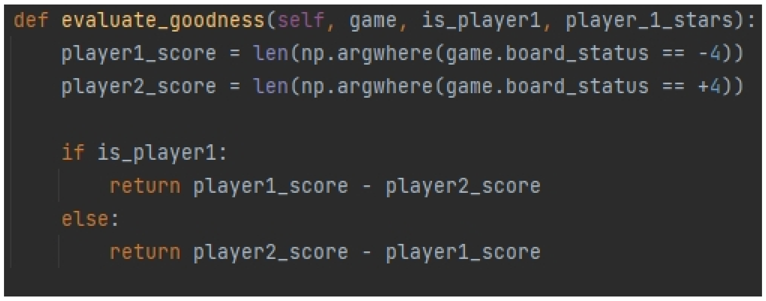


Figure 4: Figure (a) shows the GUI during the game phase; Figure (b) shows how the GUI displays the game winner

The interface is very simple, allowing the player to click on the line they want to mark and coloring any completed cells with the color of the player who scored the point, blue for the first player, red for the second player. The moment all cells have been conquered the GUI switches to displaying the result of the game, announcing the winner and showing the scores of both players. To make another match simply click anywhere on the dashboard.

3.2 CPU only alpha-beta

The first bot made makes use of only the alpha-beta pruning algorithm.
It is implemented by class E1 CPU() and for the evaluation of each move we limits to calculating the score difference between the player who is playing and the opponent.

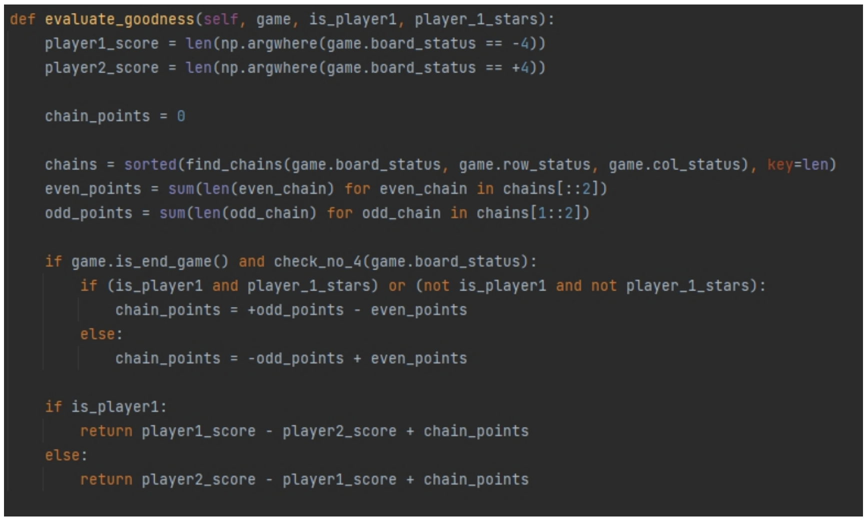


```python
def evaluate_goodness(self, game, is_player1, player_1_stars):
    player1_score = len(np.argwhere(game.board_status == -4))
    player2_score = len(np.argwhere(game.board_status == +4))

    if is_player1:
        return player1_score - player2_score
    else:
        return player2_score - player1_score
```

Figure 5: Function for evaluating a move of the first bot.

3.3 CPU alpha-beta and long chain rule

The second bot adds to the alpha-beta pruning algorithm the long chain heuristic, presented in section **2.1.3**. The heuristic was implemented by weighting more the moves that would bring the player closer to a correct number of long chains. In addition, a CHAIN POINTS parameter was added to properly weigh the influence of the heuristic against the bot's initial behavior.

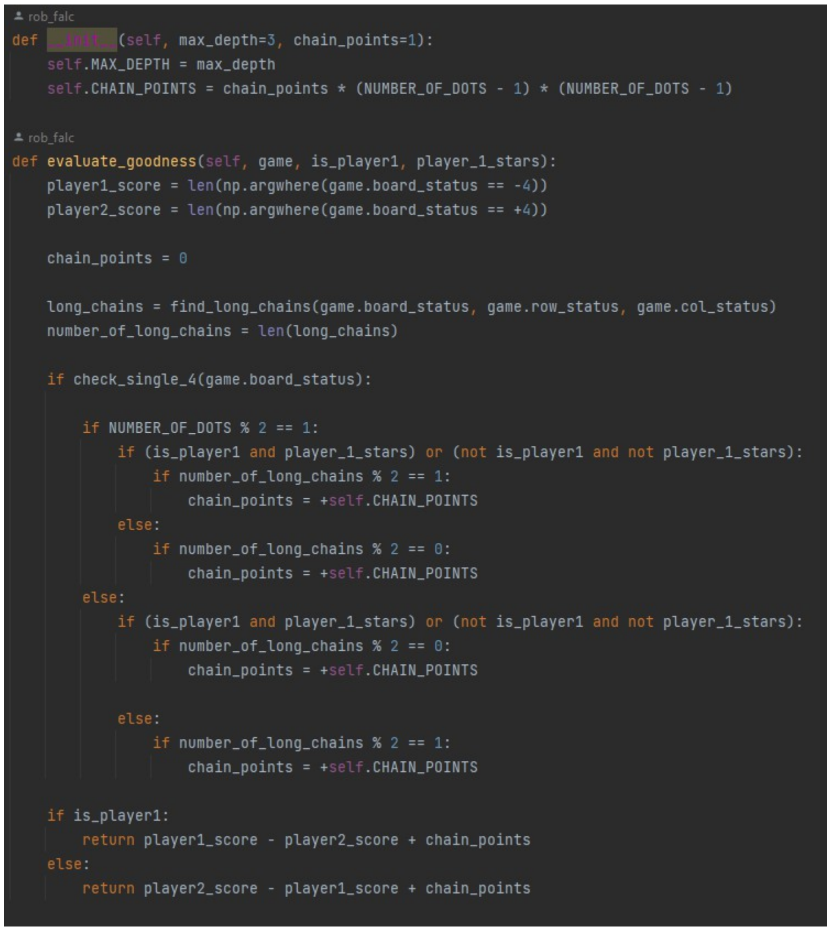


```python
def evaluate_goodness(self, game, is_player1, player_1_stars):
    player1_score = len(np.argwhere(game.board_status == -4))
    player2_score = len(np.argwhere(game.board_status == +4))

    chain_points = 0

    chains = sorted(find_chains(game.board_status, game.row_status, game.col_status), key=len)
    even_points = sum(len(even_chain) for even_chain in chains[::2])
    odd_points = sum(len(odd_chain) for odd_chain in chains[1::2])

    if game.is_end_game() and check_no_4(game.board_status):
        if (is_player1 and player_1_stars) or (not is_player1 and not player_1_stars):
            chain_points = +odd_points - even_points
        else:
            chain_points = -odd_points + even_points

    if is_player1:
        return player1_score - player2_score + chain_points
    else:
        return player2_score - player1_score + chain_points
```

Figure 6: Function for evaluating a move of the second bot.

3.4 CPU with prediction of chain conquest

The third bot, on the other hand, tries to predict which chains will be conquered by the two players. In this case to the difference between the players' scores are added the scores related to the completion of all odd or even chains, depending on whether the player will be the first to start completing the chains or not. This reasoning is based on the fact that once a chain is completed, it is necessary to yield another chain to the opponent (obviously leaving
to the latter the smallest on the grid). For the implementation in fact, it is sufficed to sort the chains within a list and then score those in odd-numbered positions to the first player to make the move and, consequently, score the even-numbered ones to the opposing player.



```
rob_falc
def __init__(self, max_depth=3, chain_points=1):
    self.MAX_DEPTH = max_depth
    self.CHAIN_POINTS = chain_points * (NUMBER_OF_DOTS - 1) * (NUMBER_OF_DOTS - 1)

rob_falc
def evaluate_goodness(self, game, is_player1, player_1_stars):
    player1_score = len(np.argwhere(game.board_status == -4))
    player2_score = len(np.argwhere(game.board_status == +4))

    chain_points = 0

    long_chains = find_long_chains(game.board_status, game.row_status, game.col_status)
    number_of_long_chains = len(long_chains)

    if check_single_4(game.board_status):

        if NUMBER_OF_DOTS % 2 == 1:
            if (is_player1 and player_1_stars) or (not is_player1 and not player_1_stars):
                if number_of_long_chains % 2 == 1:
                    chain_points = +self.CHAIN_POINTS
            else:
                if number_of_long_chains % 2 == 0:
                    chain_points = +self.CHAIN_POINTS
        else:
            if (is_player1 and player_1_stars) or (not is_player1 and not player_1_stars):
                if number_of_long_chains % 2 == 0:
                    chain_points = +self.CHAIN_POINTS

            else:
                if number_of_long_chains % 2 == 1:
                    chain_points = +self.CHAIN_POINTS

    if is_player1:
        return player1_score - player2_score + chain_points
    else:
        return player2_score - player1_score + chain_points
```

Figure 7: Function for evaluating a move of the third bot.

3.5 Player selection

Within the main class you can select the type of the two players: PvP mode (player vs. player); PvB mode (player vs. bot); BvB mode (bot vs. bot)

4 Testing

Tests were conducted to evaluate the performance of the developed algorithm and heuristics as they varied:

- Δ (MD): Maximum-Depth. Parameter of the alpha-beta prun- ing algorithm that defines how low to go within the move search tree.
- ρ: chain points. Defines the weight that is given to the long chain rule for bot **B2** in the first phase of the game.
- Γ: grid_size. Defines the size of the grid in terms of how many quad- rati it consists of.

In addition, a test was conducted to evaluate the time it takes for the algorithm to make a decision on which move to make.

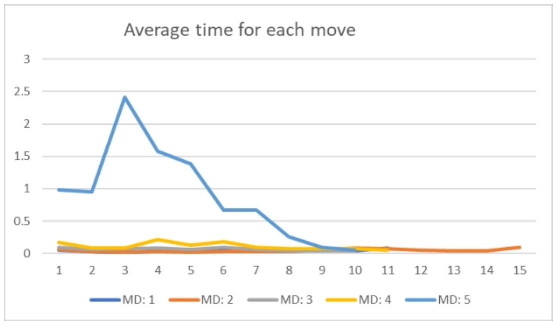
Finally, tests were carried out by having the various bots play with each other in order to understand whether the added heuristics had an impact on the presets of the algorithm.

4.1 B1 Δ - time analysis

A time analysis was performed to understand how long it took the first bot on average to make a move, and how this time changed during the various phases of the game.

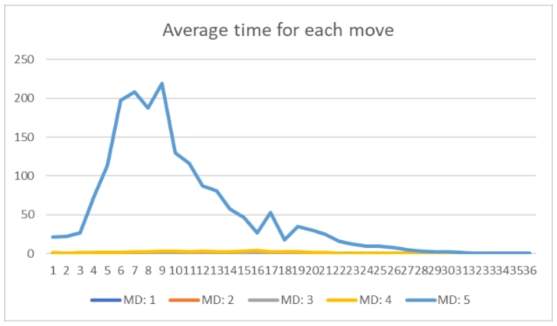
4.1.1 3x3 board

Δ: 1	**Δ: 2**	**Δ: 3**	**Δ: 4**	**Δ: 5**
0.0419	0.0442	0.0625	0.1102	0.9050



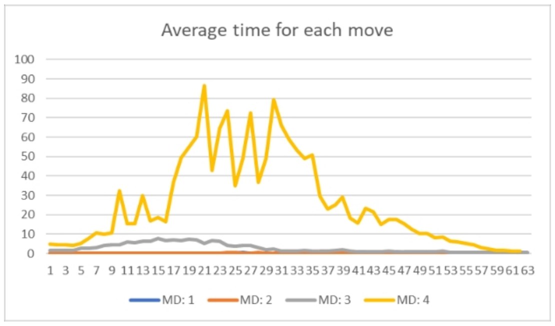
4.1.2 5x5 board

Δ: 1	Δ: 2	Δ: 3	Δ: 4	Δ: 5
0.1125	0.1281	0.3725	1.6655	51.3330



4.1.3 7x7 board

Δ: 1	Δ: 2	Δ: 3	Δ: 4
0.40417	0.37348	2.73431	25.9777



4.1.4 Considerations

The execution times for choosing a single move seem to be ab- lutely limited for values of Δ = {1, 2, 3} in the case of small and medium tables, while they start to increase for values of Δ = {4, *5}*, especially in the case of large ta- belle (*7x7*), a case in which it is impossible to perform tests with Δ = 5. In addition, we can see how the execution times seem to increase as the first phase of the game continues, and then decrease once the players start completing the chains (a factor that greatly reduces the number of moves to be analyzed).

4.2 B1 Δ - performance analysis

Γ	**Δ:1**	**Δ:2**
5x5	4	**96**

Γ	**Δ:2**	**Δ:3**
3x3	37	**63**
5x5	38	**62**

Γ	**Δ:3**	**Δ:4**
3x3	49	**51**
5X5	40	**60**

Γ	**Δ:4**	**Δ:5**
3X3	47	**53**
5X5	49	**51**

Increasing Δ does indeed result in improvements on the performance of the algorithm in the totality of cases. However, the differences tend to taper off for values of Δ = 3*,* 4*,* 5 depending on the size of the grid. Considering also the well-known increase in execution time related to the choice of a single move, it is evident that it is convenient to keep to values of Δ = 3 for small grids and Δ = 4 for medium-sized grids to achieve a better trade-off between execution time and performance.

4.3 B1 vs B2

To evaluate the impact of the addition of heuristics on the performance of the algorithm, the two bots **B1** and **B2** were made to play against each other. The following were evaluated performance as the parameter ρ varies for **B2**, in addition to varying the grid dimen- sions as mentioned earlier. Instead, the max depth parameter was set at $\Delta = 3$

Γ	**B1**	**B2(0.1)**
3x3	41	59
5x5	52	48

Γ	**B1**	**B2(0.2)**
3x3	50	50
5x5	53	47

Γ	**B1**	**B2(0.5)**
3x3	50	50
5x5	51	49

Γ	**B1**	**B2(1)**
3x3	50	50
5x5	45	55

Γ	**B1**	**B2(2)**
3x3	51	49
5x5	46	54

From the results obtained we can see that as ρ increases, the performance of **B2** approaches that of **B1** and then slightly exceeds it, in medium grid cases (*5x5*), for values of $\rho = 1,\ 2$, while in small grid cases (*3x3*) the performance is almost identical.

4.4 B1 vs B3

As in the previous case, to evaluate the performance of bot **B3,** it was made to play against bot **B1** with fixed $\Delta = 3$.

Γ	**B1**	**B3**
3X3	51	49
5X5	46	54
7X7	47	53

It is easy to see from the results how the performance of **B3** exceeds that of **B1** for increasing size of the game grid.

4.5 B2 vs B3

As a final test, bots **B2** and **B3** were made to play against each other. The value of ρ for **B2 was** set to 1.

Γ	**B2**	**B3**
3X3	42	**58**
5X5	47	**53**
7X7	**51**	49

From the table it is easy to see that the performance of **B3** is slightly higher than that of **B2** as the grid size increases, while, for small grids, the two bots are almost equivalent. This probably stems from the fact that B3 also takes into account points obtained from small chains (less than 3 squares in length), which are obviously critical for small grid sizes.

5 Conclusion

Summarizing the test results, both the changes added in **B2** and **B3** led to improvements for the alpha-beta pruning algorithm present in **B1**. The optimal values for the parameters Δ and ρ were found and the best situations for each of the bots were analyzed. Finally, by evaluating the behavior of the bots against human opponents, it was possible to see that all the "good rules" of the game, outlined in section **2.1.2,** were respected.

6 Future developments

In the future, one might consider incorporating within the algorithm a way to teach the bot how to sacrifice cells in order to obtain a higher score. Es- iste in fact a technique within the game that allows any player to win in most situations. The technique, which is all the more effective the larger the game board (and thus the chains that are formed), can be applied once all the short chains have been conquered (chains less than 3 squares in length) and consists of sacrificing the last two squares of a chain being conquered, this will force the opponent to claim those same squares (rather than other longer chains) and surrender another chain to the opponent (certainly larger than 2 squares). This technique, if repeated with each chain conquered, definitely leads to victory for the player who uses it, however, it is considered by players to be an unfair way to play and therefore to be avoided so as not to ruin the game experience for their opponents.