Yolah Board Game

Building a Two-Player Perfect-Information Game with AI Players

Pascal Garcia

November 28, 2025

MrCoder

C’est en forgeant qu’on devient forgeron

À Sarah, Hugo et Célya ❤

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 The Yolah Game

I created the Yolah game to illustrate effective techniques for implementing board games and artificial intelligences for my students. I was inspired by the Pingouins game, whose box you can see in Figure 1.1 (I highly recommend it ☺)

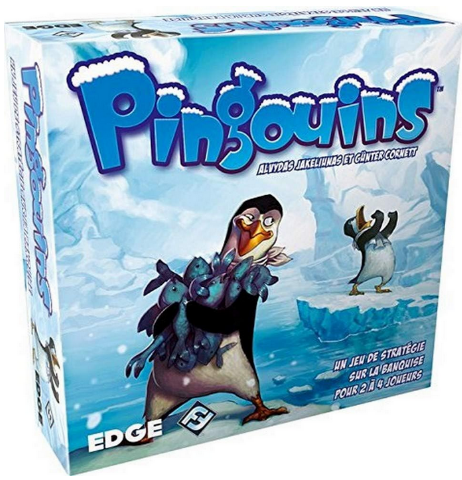


Figure 1.1: The Pingouins game box

Important

I have done my best with my current knowledge (*ars longa, vita brevis*) to implement my game and the associated AIs. But like any good scientist, you should look at my work with a critical eye. I wrote the book in French (easier for me) and asked an AI assistant (Claude [1]) to translate it for me.

I will now describe the rules of the game, then I will explain why I chose these rules, I will give an example of a game between two AIs and then I will present the rest of the book.

1.1.1 Game Rules

The Yolah game board is shown in Figure 1.2. You can see four black pieces and four white pieces placed symmetrically. Black starts by choosing one of their four

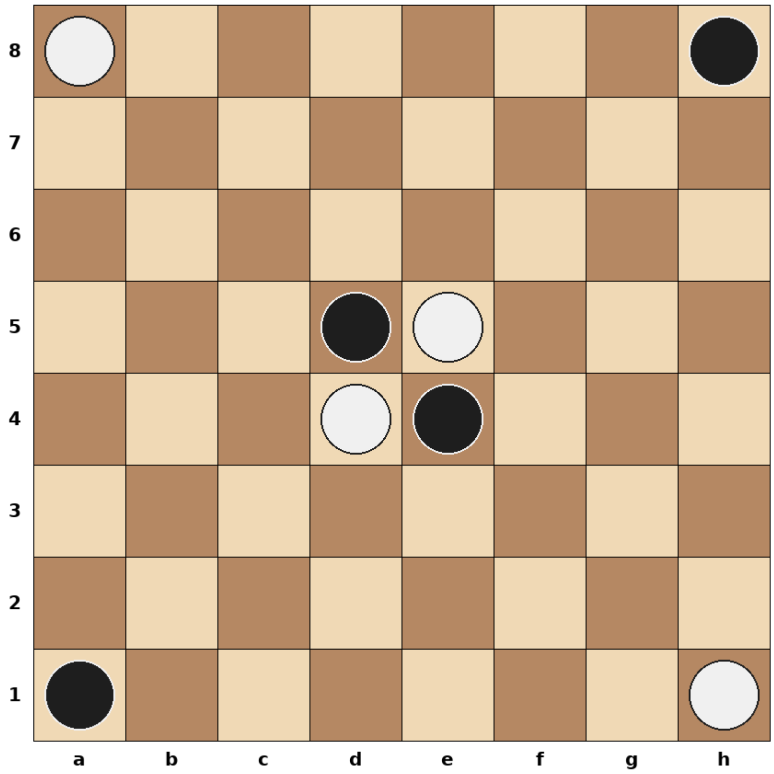


Figure 1.2: The initial configuration of the Yolah game

pieces. A piece can never disappear from the board because Yolah is a game without captures. A piece moves in all eight directions as far as it wishes as long as it is not blocked by another piece or a hole (a concept we will soon discuss). For example, if black chooses to move their piece located at `d5`, the squares where it can land are indicated by small black crosses in Figure 1.3.

Now, if the black piece at `d5` moves to `b7`, which we will denote as `d5:b7`, we get the configuration shown in Figure 1.4. Notice that the starting square of the black piece disappears and becomes a hole! This square (this hole) becomes inaccessible and impassable for the rest of the game! This will create opportunities to block the opponent and try to create areas where the opponent cannot go.

A move earns one point for the player who just moved. For example, in the configuration of Figure 1.4, the black player has one point and the white player who has not yet moved has zero points. The goal of the game is quite simple to summarize: you must move longer than your opponent!

Now it is white's turn to play. They must decide which white piece they will move. Suppose it is the piece at `e5`. The possible moves for this white piece are shown in Figure 1.5. If white decides to make the move `e5:f5`, we end up in the configuration of Figure 1.6 and the score is one point each (each player has played one move).

To summarize, the rules of Yolah are as follows:

- The game is a two-player game (black and white) played in turns.

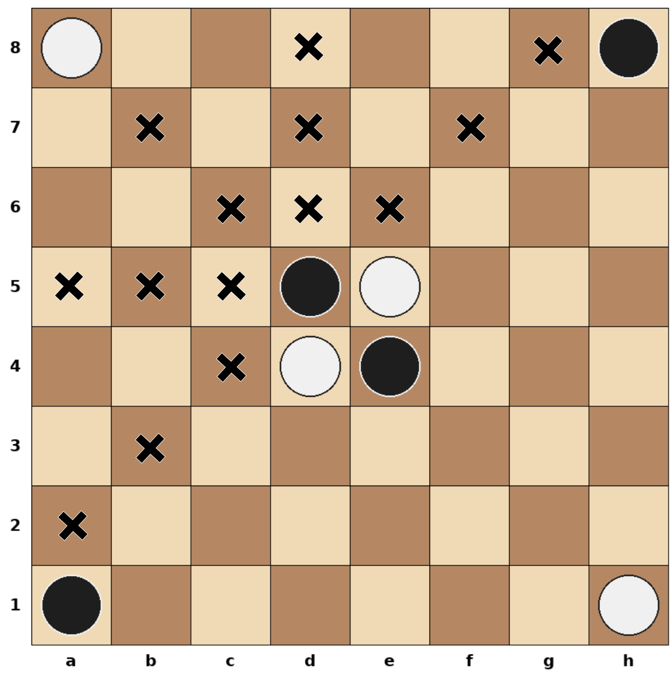


Figure 1.3: Possible moves (small black crosses) for the black piece located on square `d5`

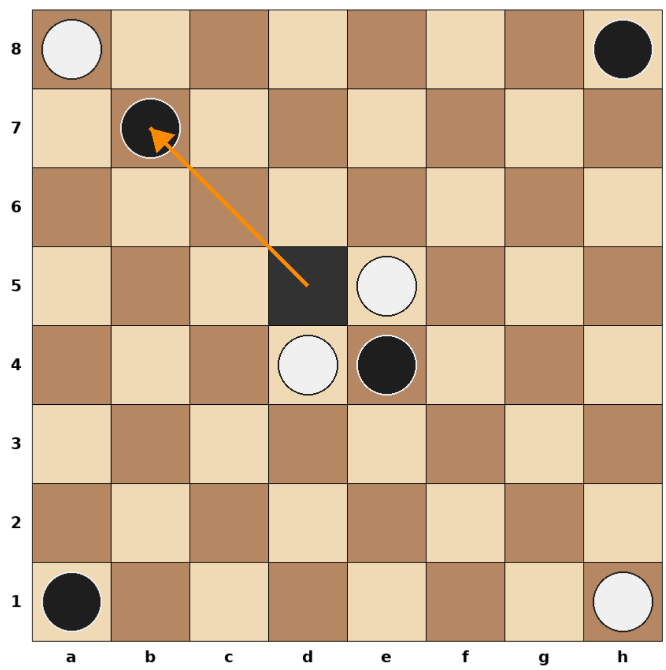


Figure 1.4: Black just moved from `d5` to `b7`. The starting square `d5` becomes inaccessible and impassable for the rest of the game

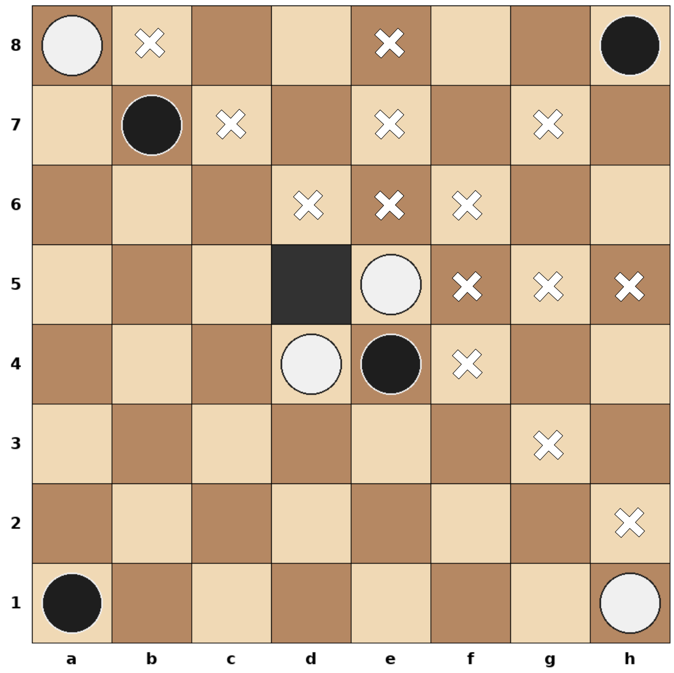


Figure 1.5: Possible moves (small white crosses) for the white piece located on square `e5`

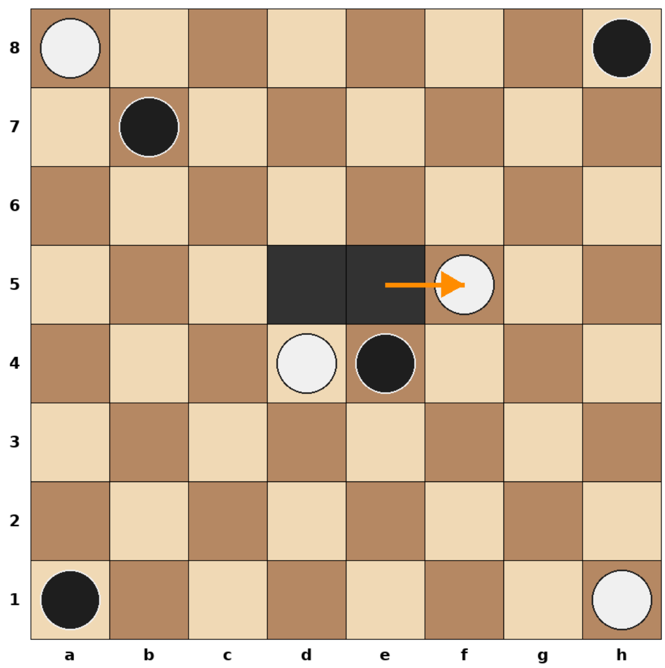


Figure 1.6: White just moved from `e5` to `f5`. The starting square `e5` becomes inaccessible and impassable for the rest of the game. The score is one point each (each player has moved once)

- Each player has four pieces.
- On their turn, the player chooses one of the pieces that can still move; if no piece can move, they pass their turn (we will denote by the move `a1:a1` to skip one's turn).
- They must move the chosen piece in one of the eight directions, as many squares as desired, but must not land on or be blocked by a piece or a hole.
- After moving the chosen piece, the starting square of the move becomes a hole and can no longer be crossed or landed on.
- After each move, the player earns one point.
- The game ends when both players can no longer move.
- The player with the most points wins the game.
- If both players have the same number of points, the game is declared a draw.

1.1.2 Interesting Characteristics of Yolah for Developing AIs

I chose the above rules for Yolah, on the one hand because I liked the dynamics of the Penguin game, but also because there are no cycles in the game, so there is no need for special rules to prevent a game from never ending. The number of moves available to each player is quite large at the beginning (but reasonable)[1], but it will decrease little by little with the appearance of holes[2]. This allows the AIs to look ahead a fairly large number of moves.

I also wanted to be able to reuse concepts used for the efficient implementation of chess, and the size of the board and the movement of the pieces (same way of moving as a Queen in chess) allow me to do that.

1.1.3 Game Example

To get an idea of how a Yolah game unfolds, we will have two artificial intelligences play against each other. The first AI will be based on Monte Carlo Tree Search and the second will be based on Minimax with a neural network. We will study both of these AIs later in the book. The second AI is stronger and you will see its zone isolation strategy in action!

The progression of the game is described in Figures 1.7, 1.8 and 1.9.

The white AI estimates that it is winning starting from move 10 (see Figure 1.7k). We can see at move 30 (see Figure 1.8k) that it has successfully isolated a zone where black can no longer access. At move 32 (see Figure 1.8m) it moves one of its pieces out of the isolated zone because the other piece will be able to collect all the points from

[1] 56 possible moves for the black player at the start of the game.

[2] The number of possible moves does not necessarily decrease after each move; there are configurations where the player has more than 56 available moves.

that zone. It is more useful to use the other piece to gather points elsewhere. Note that starting from move 47 (see Figure 1.9h onward), black has no more available moves and must therefore pass their turn.

The game is won by the white player 32 to 24, which is a very good score because the Yolah game seems to me to favor black.

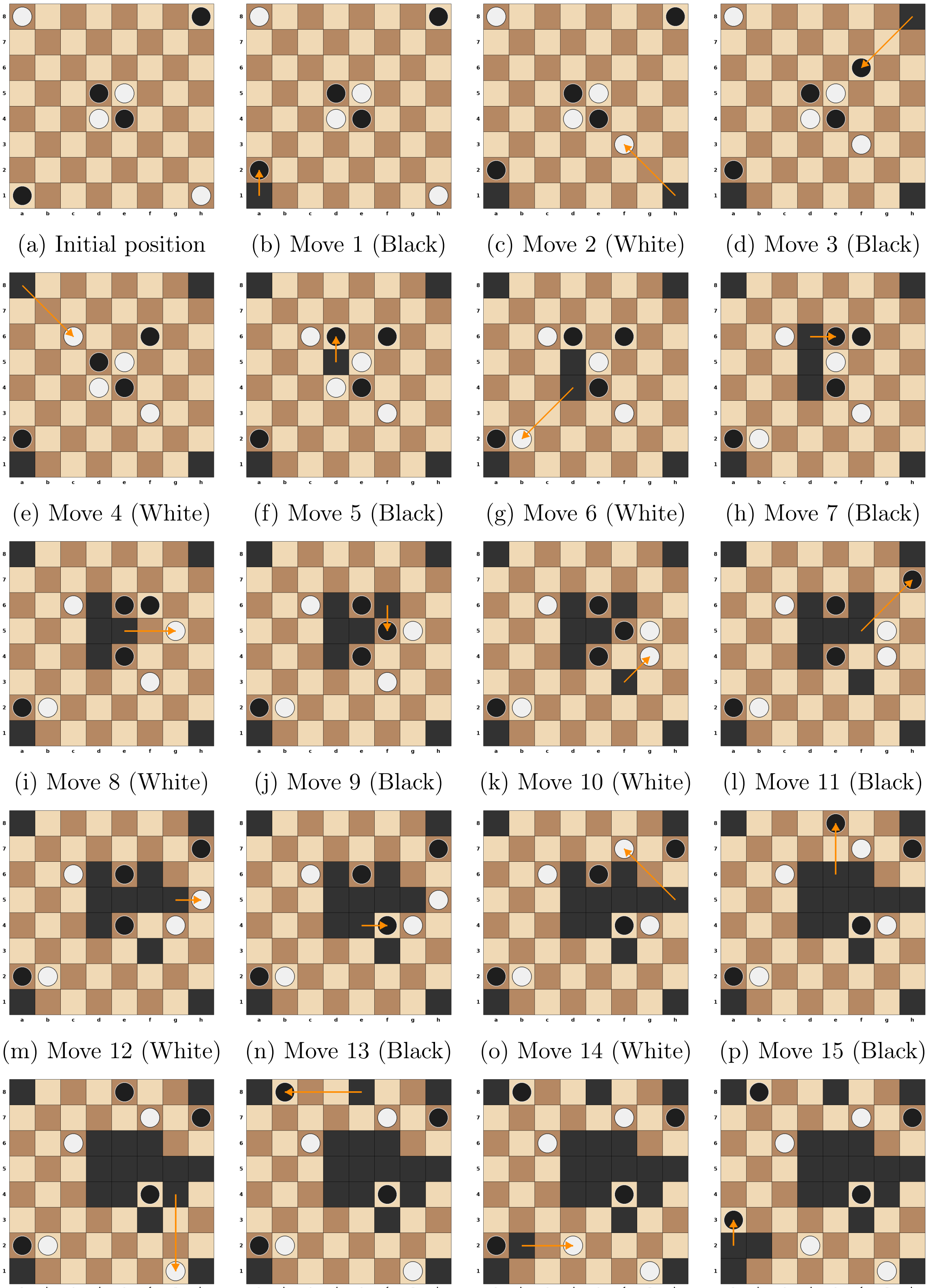
1.1.4 What's Next

In the next chapter, we will study the implementation of the Yolah game in [2]. This implementation is designed to be efficient because it will be important for the AIs to be able to play many games per second; their level of play will depend on it.

Chapter 3 describes the common interface for our different AIs. Chapter 4 presents a very simple AI based on Monte Carlo search. The next AI, described in Chapter **??**, is an evolution of the previous one and will allow, unlike the Monte Carlo AI, the development of a game tree. Note that these two AIs will not require heuristics provided by humans and only need the rules of the game. Chapter 6 presents an AI based on minimax tree search with heuristics provided by humans. The heuristic used by the AI is a linear combination of the heuristics provided by humans; the weights of each heuristic in this linear combination are learned using a genetic algorithm. Our last and strongest AI is presented in Chapter 7. A neural network is used instead of heuristics. This neural network is trained on a set of games played by the previous AI.

Chapter 8 then evaluates all these AIs by having them compete in a tournament. We will then conclude and propose different directions for creating other artificial players.

Happy reading!



(a) Initial position (b) Move 1 (Black) (c) Move 2 (White) (d) Move 3 (Black)

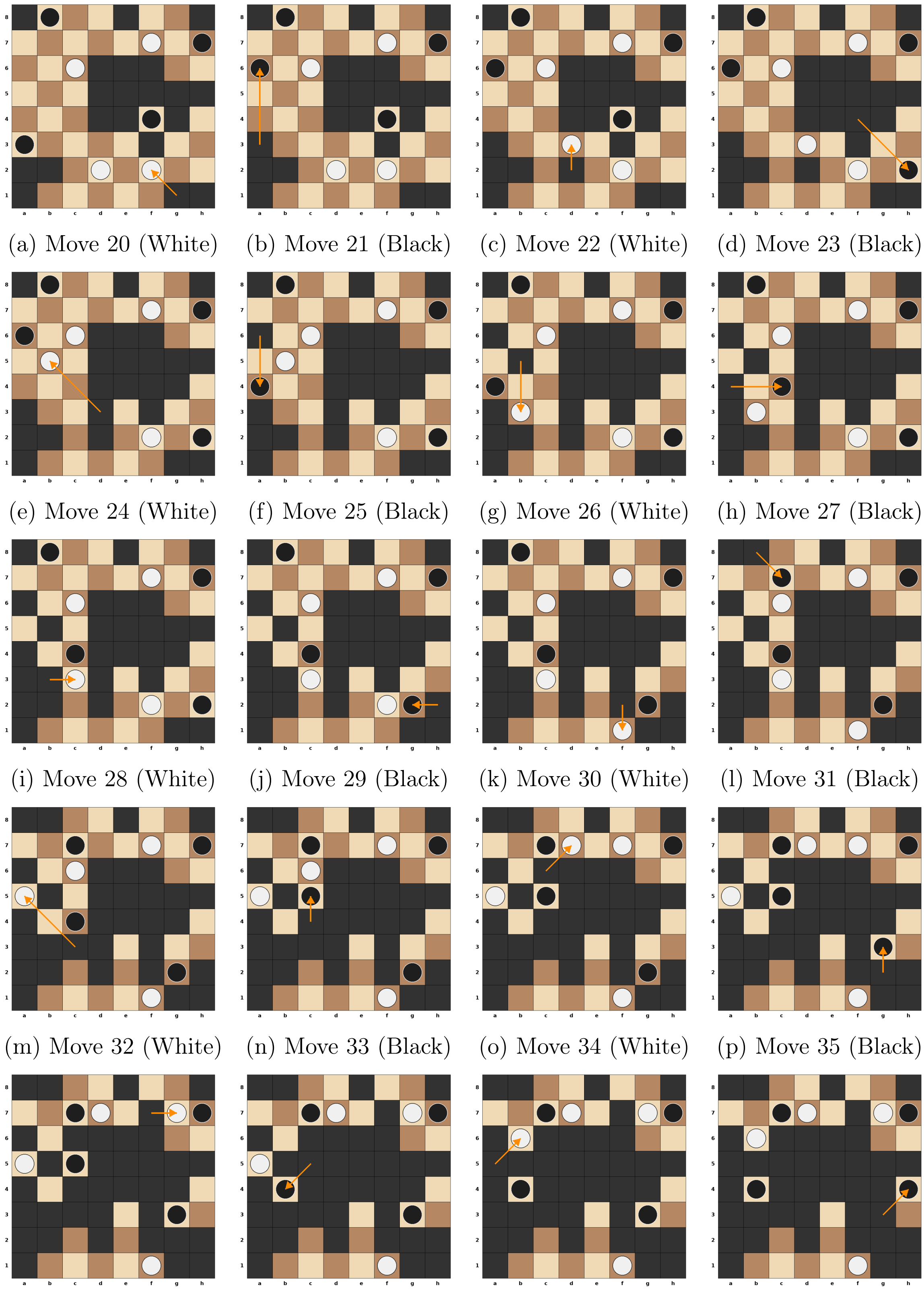
(e) Move 4 (White) (f) Move 5 (Black) (g) Move 6 (White) (h) Move 7 (Black)

(i) Move 8 (White) (j) Move 9 (Black) (k) Move 10 (White) (l) Move 11 (Black)

(m) Move 12 (White) (n) Move 13 (Black) (o) Move 14 (White) (p) Move 15 (Black)

(q) Move 16 (White) (r) Move 17 (Black) (s) Move 18 (White) (t) Move 19 (Black)

Figure 1.7: Game example between two AIs - moves 1 to 19



(a) Move 20 (White) (b) Move 21 (Black) (c) Move 22 (White) (d) Move 23 (Black)

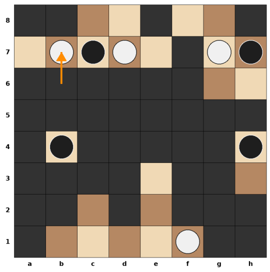
(e) Move 24 (White) (f) Move 25 (Black) (g) Move 26 (White) (h) Move 27 (Black)

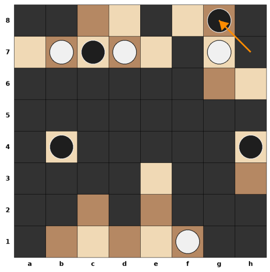
(i) Move 28 (White) (j) Move 29 (Black) (k) Move 30 (White) (l) Move 31 (Black)

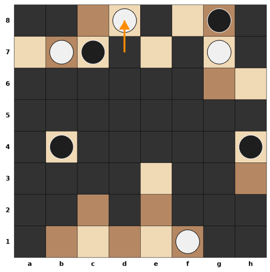
(m) Move 32 (White) (n) Move 33 (Black) (o) Move 34 (White) (p) Move 35 (Black)

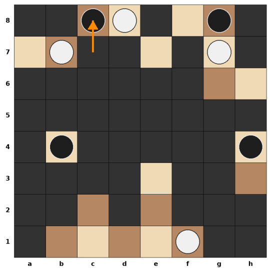
(q) Move 36 (White) (r) Move 37 (Black) (s) Move 38 (White) (t) Move 39 (Black)

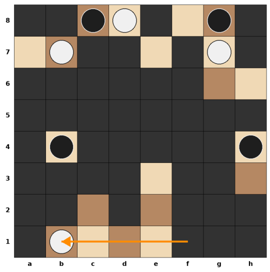
Figure 1.8: Game example between two AIs - moves 20 to 39

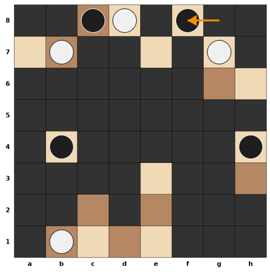

(a) Move 40 (White)

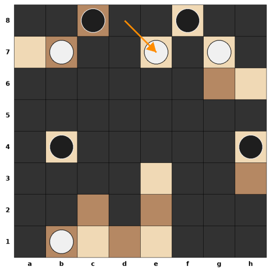

(b) Move 41 (Black)

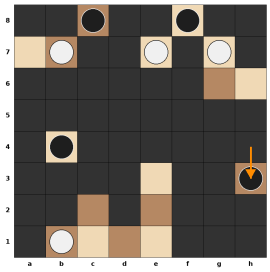

(c) Move 42 (White)

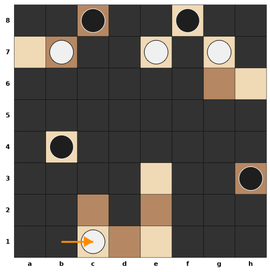

(d) Move 43 (Black)

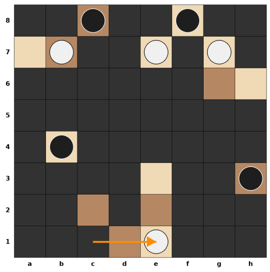

(e) Move 44 (White)

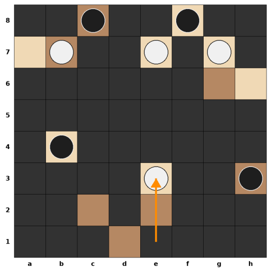

(f) Move 45 (Black)

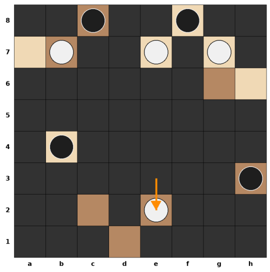

(g) Move 46 (White)

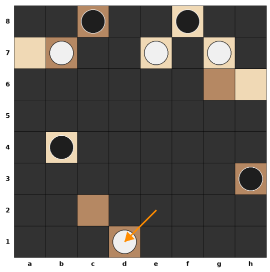

(h) Move 47 (Black)

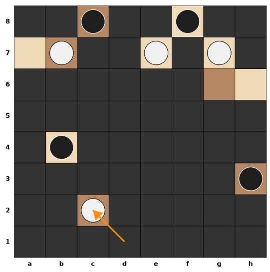

(i) Move 48 (White)

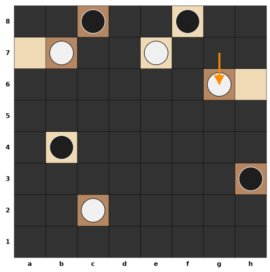

(j) Move 49 (White)

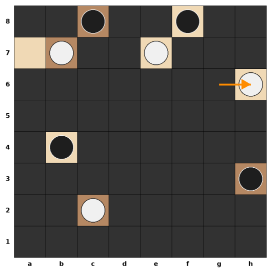

(k) Move 50 (White)


(l) Move 51 (White)


(m) Move 52 (White)


(n) Move 53 (White)


(o) Move 54 (White)


(p) Move 55 (White)

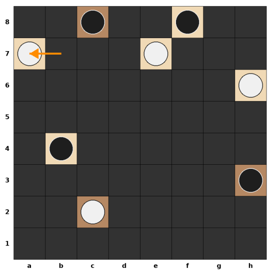

(q) Move 56 (White)

Figure 1.9: Game example between two AIs - moves 40 to 56. White wins 32 to 24

Chapter 2

Game Engine

Nous allons implémenter la gestion du jeu en C++ en essayant de créer une implémentation efficace pour pouvoir réaliser un maximum de parties à la seconde, ce qui sera important pour avoir des IA de bons niveaux. Nous testerons notre implémentation en générant un maximum de parties aléatoires en un temps donné à la fin de ce chapitre.

Nous allons représenter la position des noirs, des blancs et des cases détruites par des des entiers non signés sur 64 bits : `uint64_t`. Nous appelerons bitboard ces entiers. Nous prenons des `uint64_t` car le plateau de Yolah contient 64 cases, nous avons donc un bit par case. La table 2.1 donne la position de chaque cases du plateau dans le bitboard.

Table 2.1: Positions de chaque cases du plateau dans le bitboard

8	bit_{56}	bit_{57}	bit_{58}	bit_{59}	bit_{60}	bit_{61}	bit_{62}	bit_{63}
7	bit_{48}	bit_{49}	bit_{50}	bit_{51}	bit_{52}	bit_{53}	bit_{54}	bit_{55}
6	bit_{40}	bit_{41}	bit_{42}	bit_{43}	bit_{44}	bit_{45}	bit_{46}	bit_{47}
5	bit_{32}	bit_{33}	bit_{34}	bit_{35}	bit_{36}	bit_{37}	bit_{38}	bit_{39}
4	bit_{24}	bit_{25}	bit_{26}	bit_{27}	bit_{28}	bit_{29}	bit_{30}	bit_{31}
3	bit_{16}	bit_{17}	bit_{18}	bit_{19}	bit_{20}	bit_{21}	bit_{22}	bit_{23}
2	bit_{8}	bit_{9}	bit_{10}	bit_{11}	bit_{12}	bit_{13}	bit_{14}	bit_{15}
1	bit_{0}	bit_{1}	bit_{2}	bit_{3}	bit_{4}	bit_{5}	bit_{6}	bit_{7}
	a	b	c	d	e	f	g	h

Prenons comme exemple le plateau représentée dans la figure 2.1. Les positions des pièces noires sur le plateau :

seront représentées par le bitboard suivant :

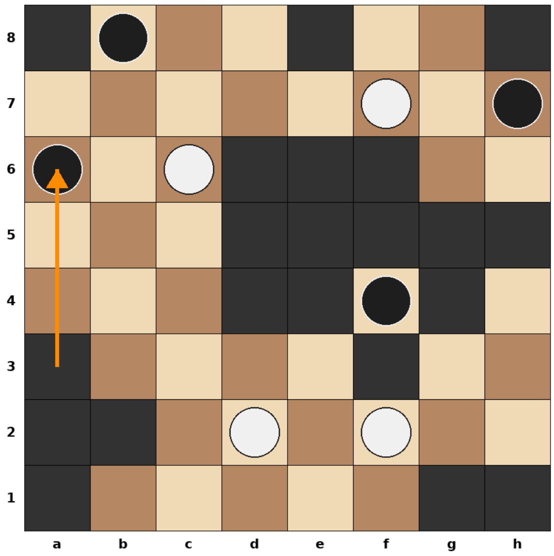


Figure 2.1: Configuration du plateau de jeu correspondant au coup 21 de la partie donnée en example dans le chapitre précédent (figure 1.8b)

Chapter 3

AI Players

Chapter 4

Monte Carlo Player

Chapter 5

MCTS Player

Chapter 6

Minmax Player

Chapter 7

Minmax with Neural Network Player

Chapter 8

AI Tournament

Chapter 9

Conclusion

Bibliography

[1] Anthropic. *Claude Sonnet 4.5*. https://www.anthropic.com/claude. Large Language Model. 2025.

[2] cppreference.com. *C++ Reference*. Accessed: 2025-01-28. 2025. URL: https://en.cppreference.com/.