Dou Shou Qi

Adversarial Search Methods for a Two-Player Board Game

Faculty of Engineering · University of Porto

Master in Data Science and Engineering

André Afonso

Nuno Vasconcelos

Contents

JUNGLE CHESS
PRESS SPACE TO START
HOW TO PLAY?
André Afonso
Nuno Vasconcelos
MECD06 - Artificial Intelligence
FEUP - 2022

Introduction to the game

Game Rules

1. **The Pawns**

   Each player has 8 pawns, numbered 1 to 8, according to their strength. All pieces can only move horizontally or vertically. Each piece can capture every weaker or equally strong opponent. The rat is the only exception to this rule, as it can capture the elephant.

2. **The game board**

   Three types of special squares are indicated: Lake, Den, Trap. All others are considered normal.

3. **Den squares**

   Each player possesses a Den. It is illegal to move your own pawn to your Den. As soon as an enemy Den is occupied by an adversary's pawn, the game ends.

4. **Lake squares**

   Each of the two lakes contains 6 squares. The lion and the tiger may jump over the lake. But only the rat can swim through them.

5. **Trap squares**

   Surrounding each Den are 3 trap squares. Each animal in a trap is considered to have 0 strength, and therefore it cannot defend itself.

6. **Objective**

   Move a piece into the opponent’s Den, or capture all their pawns.

Development environment

Development environent

Class diagram for the game

main.py

- Main pygame logic
- Different screens (stages)
- Allows user input

game.py

- Top layer
- Defines turns
- Evaluates winner
- Main screen draw functions

board.py

- Holds all the game logic
- Matrix representation of the board

piece.py

- Information about each piece of the board
- Helper function to draw image on screen

Demonstration

Adversarial Search

Evaluation functions developed

Evaluation functions developed

An iterative process was applied in the development of ever-improving evaluation functions for the game's AI

1. Simple piece count

A very simple function where the AI keeps a tally of the **total number of** Red and Black **pieces**

2. Piece strength evaluation

An improvement of the previous evaluation function where the **pieces' relative strength** is taken into account

3. Distance-based evaluation

Gives the AI an **incentive to advance** in the board regardless of capturing moves in the "near-future" (tree depth) – Manhattan distance

4. Strength + Distance eval.

Combines the previous two functions for decent human-like behaviour where **board advancement and captures** are taken into account

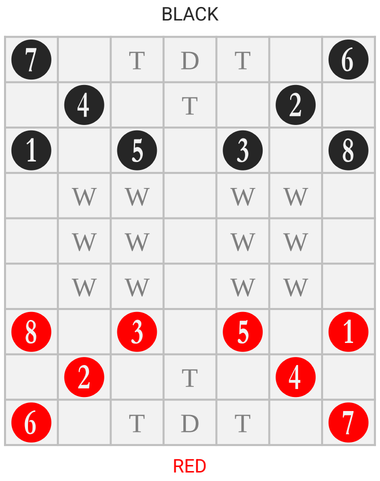
5. Position score matrices

Developed after dozens of Human-Human and Human-AI sessions and considering the different aspects of the game's strategy: **piece strength, development** and **cooperation/relation**

Evaluation functions developed

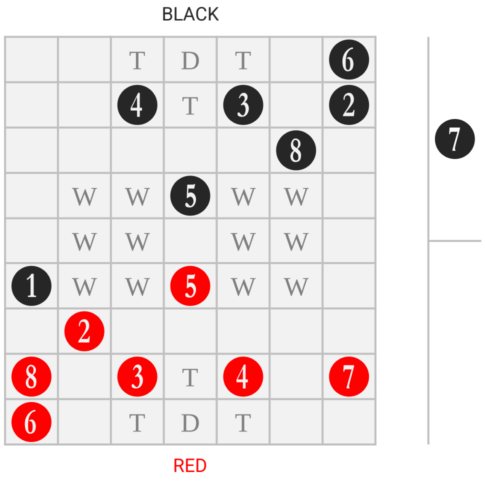
An iterative process was applied in the development of ever-improving evaluation functions for the game's AI

1. Simple piece count



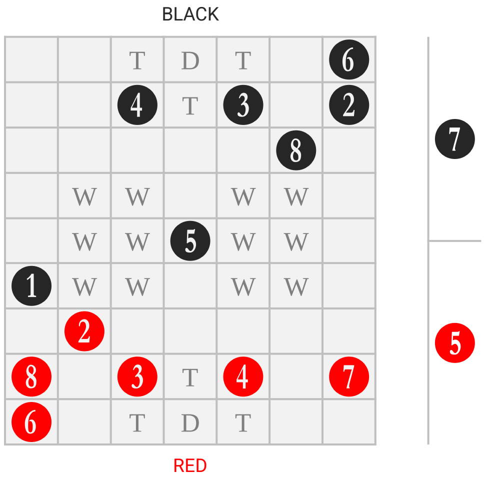
Initial Setup

8 vs 8



Black is missing its Lion (7)

7 vs 8



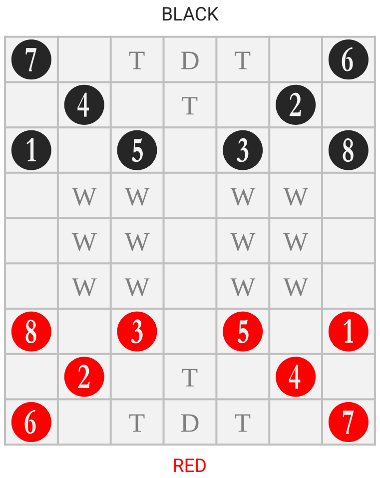
Black captured Red Puma (5)

7 vs 7

Evaluation functions developed

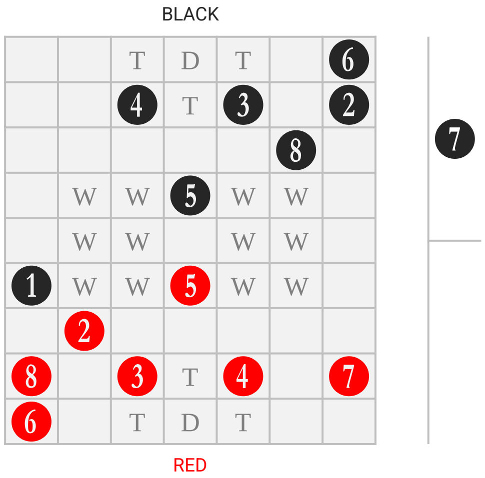
An iterative process was applied in the development of ever-improving evaluation functions for the game’s AI

2. Piece strength evaluation



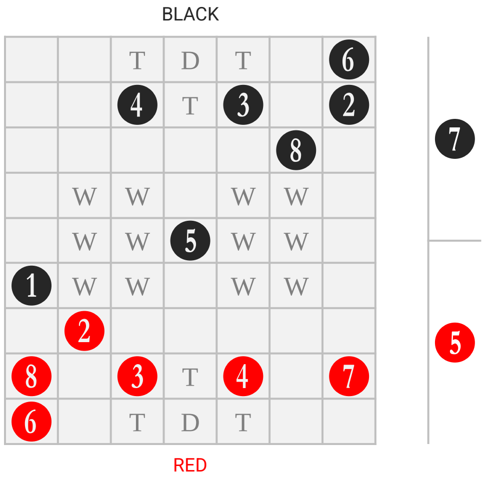
Initial Setup

36 vs 36



Black is missing its Lion (7)

29 vs 36



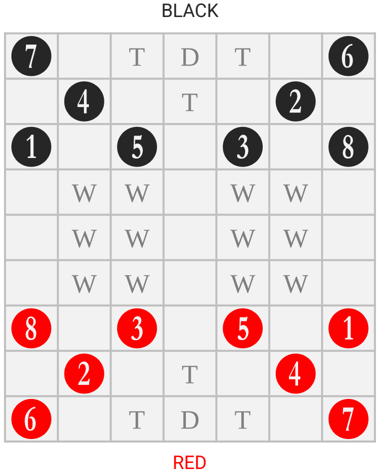
Black captured Red Puma (5)

29 vs 31

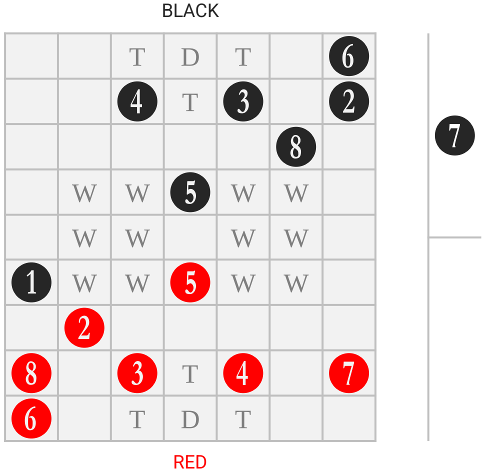
Evaluation functions developed

An iterative process was applied in the development of ever-improving evaluation functions for the game’s AI

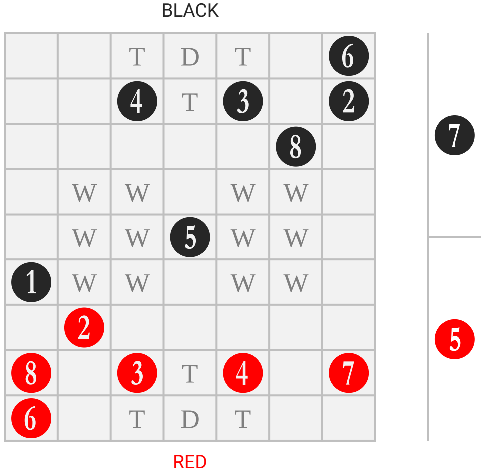
2. Piece strength evaluation



Initial Setup



Black is missing its Lion (7)



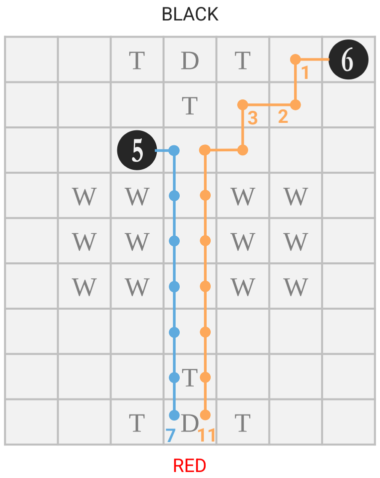
Black captured Red Puma (5)

i Due to the **rat’s special ability**, a value of **5** was considered, making the initial board value for each player **40**

Evaluation functions developed

An iterative process was applied in the development of ever-improving evaluation functions for the game's AI

3. Distance-based evaluation



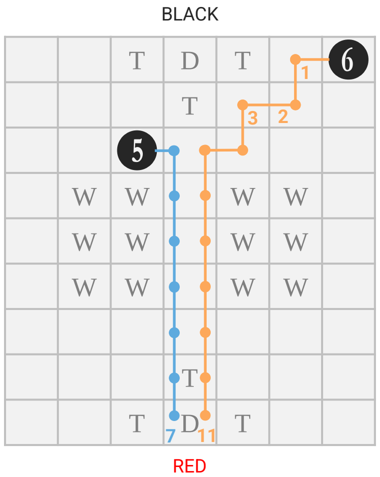
Manhattan Distance

- **I** At the beginning of the game each player has a combined distance of **72**
- **II** As the game progresses, both players try to **minimize their distance** to the opponents' den
- **III** This creates an interesting **"center-rush"** mechanic for both players

Evaluation functions developed

An iterative process was applied in the development of ever-improving evaluation functions for the game's AI

4. Strength + Distance eval.



Manhattan Distance

- **I** Combines the previous two functions tackling **board advancement** and **captures**
- **II** **Simulates** decent **human-like behaviour** and is a good proxy for low-difficulty levels
- **III** **Prioritizes capturing pieces** near the traps rather than attacking the enemy's den

Evaluation functions developed

An iterative process was applied in the development of ever-improving evaluation functions for the game's AI

5. Position score matrices

We considered 4 factors to **strategically evaluate the position** on the board:

1. **Material relationship**

   The **number of pieces** left for each player on the board

2. **Piece value**

   The **strength relationship** between each player's pieces.

3. **Development of the pieces**

   The **incentive to advance** in the board while keeping the den safe

4. **Cooperation / relationship between the pieces**

   The best placement of the pieces considering their **tactical role** in the game and their supporting / complementary roles

Evaluation functions developed

An iterative process was applied in the development of ever-improving evaluation functions for the game's AI

5. Position score matrices

BLACK

		T	0	T		
			T			
	0	0		0	0	
	0	0		0	0	
	0	0		0	0	
			T			
		T	inf	T		

RED

I Evaluating distance to the opponent's den by **incrementing the marginal utility** of the squares

Evaluation functions developed

An iterative process was applied in the development of ever-improving evaluation functions for the game's AI

5. Position score matrices

BLACK

10	10	T	0	T	10	10
11	11	11	T	11	11	11
12	12	12	12	12	12	12
13	0	0	13	0	0	13
14	0	0	14	0	0	14
15	0	0	15	0	0	15
16	16	16	16	16	16	16
17	17	17	T	17	17	17
18	18	T	Inf	T	18	18

RED

I

Evaluating distance to the opponent's den by **incrementing the marginal utility** of the squares

- Increase the square value **vertically**

Evaluation functions developed

An iterative process was applied in the development of ever-improving evaluation functions for the game's AI

5. Position score matrices

BLACK

10	10	T	0	T	10	10
11	11	11	T	11	11	11
12	12	12	12	12	12	12
13	0	0	13	0	0	13
14	0	0	14	0	0	14
15	0	0	15	0	0	15
16	16	16	16	16	16	16
17	17	17	T	17	17	17
18	18	T	Inf	T	18	18

RED

I

Evaluating distance to the opponent's den by **incrementing the marginal utility** of the squares

- Increase the square value **vertically**

Evaluation functions developed

An iterative process was applied in the development of ever-improving evaluation functions for the game’s AI

5. Position score matrices

BLACK

10	11	T	0	T	11	10
11	12	13	T	13	12	11
12	13	14	15	14	13	12
13	0	0	16	0	0	13
14	0	0	17	0	0	14
15	0	0	18	0	0	15
16	17	18	19	18	17	16
17	18	19	T	19	18	17
18	19	T	inf	T	19	18

RED

I Evaluating distance to the opponent’s den by **incrementing the marginal utility** of the squares

- Increase the square value **vertically**
- Increase the square value **inwards**

Evaluation functions developed

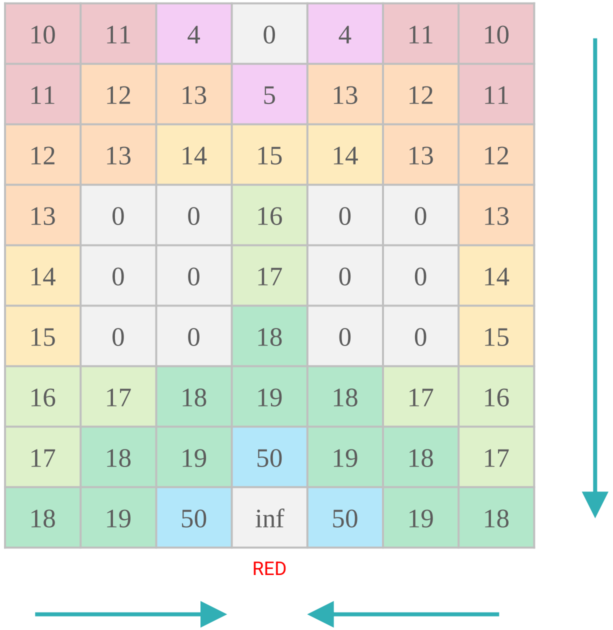
An iterative process was applied in the development of ever-improving evaluation functions for the game's AI

5. Position score matrices

BLACK

10	11	4	0	4	11	10
11	12	13	5	13	12	11
12	13	14	15	14	13	12
13	0	0	16	0	0	13
14	0	0	17	0	0	14
15	0	0	18	0	0	15
16	17	18	19	18	17	16
17	18	19	50	19	18	17
18	19	50	inf	50	19	18

RED



I

Evaluating distance to the opponent's den by **incrementing the marginal utility** of the squares

- Increase the square value **vertically**
- Increase the square value **inwards**
- **Penalize** own den; **incentivize attacking** opponent den

Evaluation functions developed

An iterative process was applied in the development of ever-improving evaluation functions for the game's AI

5. Position score matrices

BLACK

10	11	4	0	4	11	10
11	12	13	5	13	12	11
1	13	14	15	14	13	12
13	0	0	16	0	0	13
14	0	0	17	0	0	14
15	0	0	18	0	0	15
16	17	18	19	18	17	16
17	18	19	50	19	18	17
18	19	50	inf	50	19	18

RED

I. Evaluating distance to the opponent's den by **incrementing the marginal utility** of the squares

II. Accounting for the **non-centered starting position** of each piece

Evaluation functions developed

An iterative process was applied in the development of ever-improving evaluation functions for the game's AI

5. Position score matrices

BLACK

7	11	4	0	4	11	6
11	4	13	5	13	2	11
1	13	5	15	3	13	8
13	0	0	16	0	0	13
14	0	0	17	0	0	14
15	0	0	18	0	0	15
8	17	3	19	5	17	1
17	2	19	50	19	4	17
6	19	50	inf	50	19	7

RED

I Evaluating distance to the opponent's den by **incrementing the marginal utility** of the squares

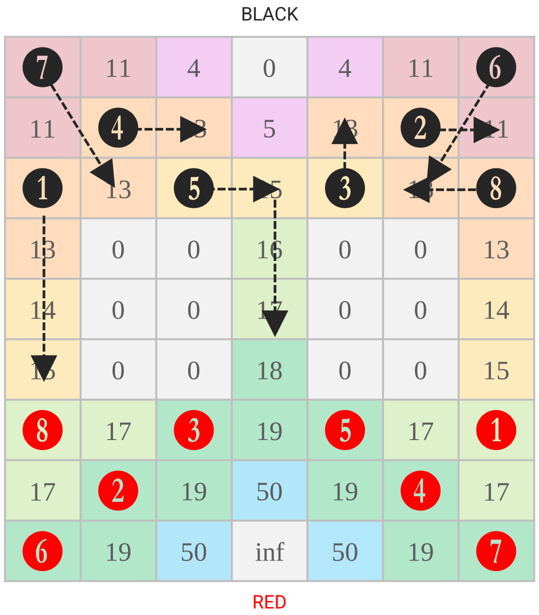
II Accounting for the **non-centered starting position** of each piece

III Considering each piece's **tactical role** in the game and their **supporting / complementary roles**

Evaluation functions developed

An iterative process was applied in the development of ever-improving evaluation functions for the game's AI

5. Position score matrices



I Evaluating distance to the opponent's den by **incrementing the marginal utility** of the squares

II Accounting for the **non-centered starting position** of each piece

III Considering each piece's **tactical role** in the game and their **supporting / complementary roles**

- **Rat**: attack the elephant, safe on water
- **Dog and Wolf**: weak pieces, guard the player's traps
- **Tiger and Lion**: attacking pieces
- **Cat**: weak piece, defend the elephant from mouse
- **Puma**: center-lane control and rush
- **Elephant**: development blocked by mouse, defend against lion attack

Evaluation functions developed

An iterative process was applied in the development of ever-improving evaluation functions for the game's AI

5. Position score matrices

BLACK

8	8	4	0	4	8	8
9	9	9	5	8	8	8
1	10	10	9	8	8	8
11	12	12	10	9	9	8
12	12	12	11	9	9	8
13	12	12	11	9	9	8
13	13	13	13	11	11	10
13	13	13	50	13	12	11
13	13	50	inf	50	13	11

RED

MOUSE DEVELOPMENT SCORES

BLACK

7	12	4	0	4	12	10
12	14	12	5	12	12	12
14	16	16	14	16	16	14
15	W	W	15	W	W	15
15	W	W	15	W	W	15
15	W	W	15	W	W	15
18	20	20	30	20	20	18
25	25	30	50	30	25	25
25	30	50	inf	50	30	25

RED

LION DEVELOPMENT SCORES

Piece		Value
Mouse	1	500
Cat	2	200
Dog	3	300
Wolf	4	400
Puma	5	500
Tiger	6	800
Lion	7	900
Elephant	8	1000

Adversarial Search

Search algorithms used and performance evaluation

Search algorithms used

Several algorithms were implemented in order to test their relative efficiency and computational performance

Standard function

Runs the selected algorithm up to a **predefined search depth**

Iterative Deepening function

Runs repeatedly with **increasing depth limits** until the goal is found or a **predefined time limit** is reached

Allows the search for "shallower" victories

1. Standard **Minimax**
2. **Alpha-Beta** pruning
3. **Alpha-Beta** pruning w/ **move-ordering**

Alpha-Beta pruning with move-ordering

Search algorithms used

GOAL: evaluate moves which can result in **faster pruning as early as possible**

Check for captures

Determine if there are immediate capturing moves for the current board position

Check for lion – tiger – elephant - mouse

Search for moves made by more valuable / stronger pieces currently on the board

Check for the remaining pieces

Only then resume the search for every other piece on the board

Performance evaluation

Analysing efficiency and computational performance for each algorithm

Analysing the impact of pruning and move-ordering

Depth 3: **Position scores**

	Minimax		Alphabeta		Move-ordering	
	Without capture	With capture	Without capture	With capture	Without capture	With capture
avg Processing time *per play (milliseconds)*	**1.524**	**1.569**	**1.046**	**515**	**632**	**458**
max Processing time *per play (milliseconds)*	3.282	2.940	2.013	733	1.668	632
			- 31,4%	**- 67,2%**	**- 58,5%**	**- 70,8%**
avg Searched boards	**5.929**	**6.221**	**1.311**	**816**	**909**	**516**
max Searched boards	12.240	11.461	3.239	1.024	1.802	651
			- 77,9%	**- 86,9%**	**- 84,7%**	**- 91,7%**

Performance evaluation

Analysing the impact of pruning and move-ordering

Iterative deepening, 1 sec timeout: **Position scores**

	Minimax		Alphabeta		Move-ordering	
	Without capture	With capture	Without capture	With capture	Without capture	With capture
avg Processing time *per play (milliseconds)*	2.796	2.084	4.367	3.340	3.032	4.623
max Processing time *per play (milliseconds)*	13.871	3.486	11.776	5.772	12.441	11.871
avg Searched boards	**11.173**	**8.395**	**4.644**	**3.090**	**1.677**	**1.562**
max Searched boards	50.530	14.886	24.569	7.218	10.824	3.122
			- 58,4%	**- 63,2%**	**- 85,0%**	**- 81,4%**
avg Depth	**3,21**	**3,13**	**3,76**	**3,92**	**3,33**	**3,87**
max Depth	4	4	4	4	5	4
			+ 17,1%	**+ 25,2%**	**+ 3,7%**	**+ 23,6%**

Determining AI difficulty levels

Framework to define the available AI difficulty levels

Reasoning behind the **parameter selection** for the **three** available **difficulty levels** (easy, medium, hard):

- **Avoid "blind guesses"** regarding the choices of depth level and evaluation function
- Determine the previous parameters based **on real Human *versus* AI plays** and outcomes
- Set the parameters according to the **win percentile of the Humans**
- Approx. **200 games** played with varying parameters

Easy

Around the **75th** percentile

Evaluation function: Strength + Distance

Search depth: 3 (or 1 second timeout)

Human win percentage: **79%**

Medium

Around the **50th** percentile

Evaluation function: Position score matrices

Search depth: 3 (or 1 second timeout)

Human win percentage: **56%**

Hard

Around the **25th** percentile

Evaluation function: Position score matrices

Search depth: 7 (or 15 seconds timeout)

Human win percentage: **22%**

Future work

Topics and study subjects
for further exploration

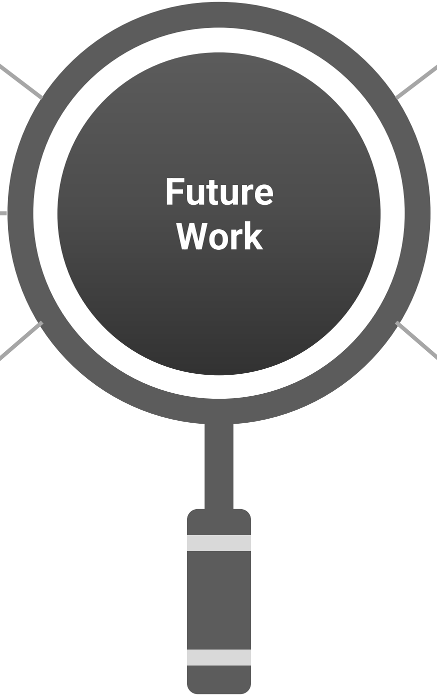
Future Work

Topics and study subjects for further exploration

01 Prevent previous moves
Insert method to prevent the same board position for both players in previous N moves

02 Enhance move-ordering
Explore other move-ordering criteria to further prune the search tree

03 Monte Carlo Tree-Search
Implement the MCTS heuristic for random sampling of the search space



04 GA for Position Scores
Develop a Genetic Algorithm (or other search heuristic) to optimize the parameters for the heuristic function

05 ML model for the endgame
Create a Machine Learning algorithm (Neural Network) for endgame scenarios with tablebases of board configurations

?

QUESTIONS