

# **The Tablut Challenge: Creating an AI Agent for an Asymmetric Board Game**

December 19, 2023

# Agenda

- Introduction to the Tablut Challenge
- Asymmetry in Strategy: White vs. Black Player
- Implementing the AI Agent: Tools & Technologies
- Managing Connection with a Java Server
- Player-Specific Heuristics: White Player
- Player-Specific Heuristics: Black Player
- Fine-tuning Heuristics using Genetic Algorithms
- Process of Creating Next Generation using Genetic Algorithm Mutation

# Introduction to the Tablut Challenge

---

Tablut is a board game similar to chess, with two asymmetric players aiming to win against each other.

---

The project involved creating an AI agent capable of being competitive against human and other AI players.

---

The AI agent was implemented using the Alpha-Beta cuts algorithm on the MiniMax algorithm, with enhancements like a time-stopping criteria and heuristics for evaluating the board state.

---

# Asymmetry in Strategy: White vs. Black Player

## White Player Strategy

The strategy of the white player is not the same as the black player.

The white player aims to protect and move their king to an exit square while the black player tries to capture the king.


## Black Player Strategy

The black player's goal is to capture the king, which is the win condition.


The black player needs to use specific strategies to prevent the white player from reaching the goal.

The game ends in a draw if the same state of the board is repeated twice.


# Implementing the AI Agent: Tools & Technologies



We used Python for the implementation, leveraging the AIMA library for the AI components.




We employed the MiniMax algorithm with Alpha-Beta cuts for the game strategy, enhancing it with a custom time-stopping criteria and singular extensions.




We used Numpy for board representation, aiming to improve efficiency.

# Managing Connection with a Java Server


The game was hosted on a Java server, requiring the AI agent to manage a stable connection.




The communication protocol with the server had to be carefully handled to ensure the proper flow of game data.



The time to compute a move was an important parameter, requiring the agent to adhere to the specified time limit.



The AI agent had to be designed to efficiently use the allocated time for move computation.



# Player-Specific Heuristics: White Player

## Heuristic Factors

Needed to evaluate each board state. In particular we consider:

- The number of white and black pieces on the board
- The proximity of the king to enemy's pieces
- The distance of the king from the center of the board
- The number of free ways to reach the king
- A matrix of position weights

## Weighted Linear Function

- The evaluation function is based on a weighted linear combination of the heuristic factors.
- Each factor has a specific weight based on its importance in evaluating the fitness of the board state.
- Example:  $w_1 * (\text{number of white pieces}) - w_2 * (\text{number of black pieces}) + w_3 * (\text{distance of king from center}) - w_4 * (n^\circ \text{ free-ways}) - w_5 * (\text{king surrounded}) + w_6 * (\text{position weights})$

# Player-Specific Heuristics: Black Player

## Heuristic Factors

Needed to evaluate each board state. In particular we consider:


- The number of white and black pieces on the board
- The proximity of black pieces to the king
- The availability of clear paths for the king to an exit
- Pawns around the king (encirclement)

## Weighted Linear Function

- The black player heuristics are designed to evaluate the fitness of the board state from the perspective of the black player.
- The evaluation function is based on a weighted linear combination of the heuristic factors.
- Example:  $w_1 * (\text{number of black pieces}) - w_2 * (\text{number of white pieces}) + w_3 * (\text{proximity of black pieces to the king}) + w_4 * (\text{n}^\circ \text{ free-ways to king})$



# Fine-tuning Heuristics using Genetic Algorithms



We used Genetic Algorithms to fine-tune the heuristic weights for both the White and Black players.

To do so, we create a population of weights that each player uses to play against each other. A selection process to choose the best weights per generation is also necessary.

The next generation of weights is produced by using the best fit elements from the previous generation, incorporating standard genetic algorithm operations such as crossover and mutation.

The goal was to discover the best set of parameters for both players' heuristics, maximizing the AI agent's performance either while playing with white and with black

# Process of Creating Next Generation using Genetic Algorithm

## Selection



The individuals from the current generation are selected for creating the next generation based on their fitness and suitability for reproduction.

## Crossover and Mutation



The selected individuals undergo crossover and mutation to create new combinations of parameters. This simulates the process of reproduction and genetic variation.

## Fitness Evaluation



The new combinations of parameters are evaluated for their fitness in the context of the game. The best-fit elements are chosen for the next generation.

# Thanks → everyone!

~ \$ \tLut team