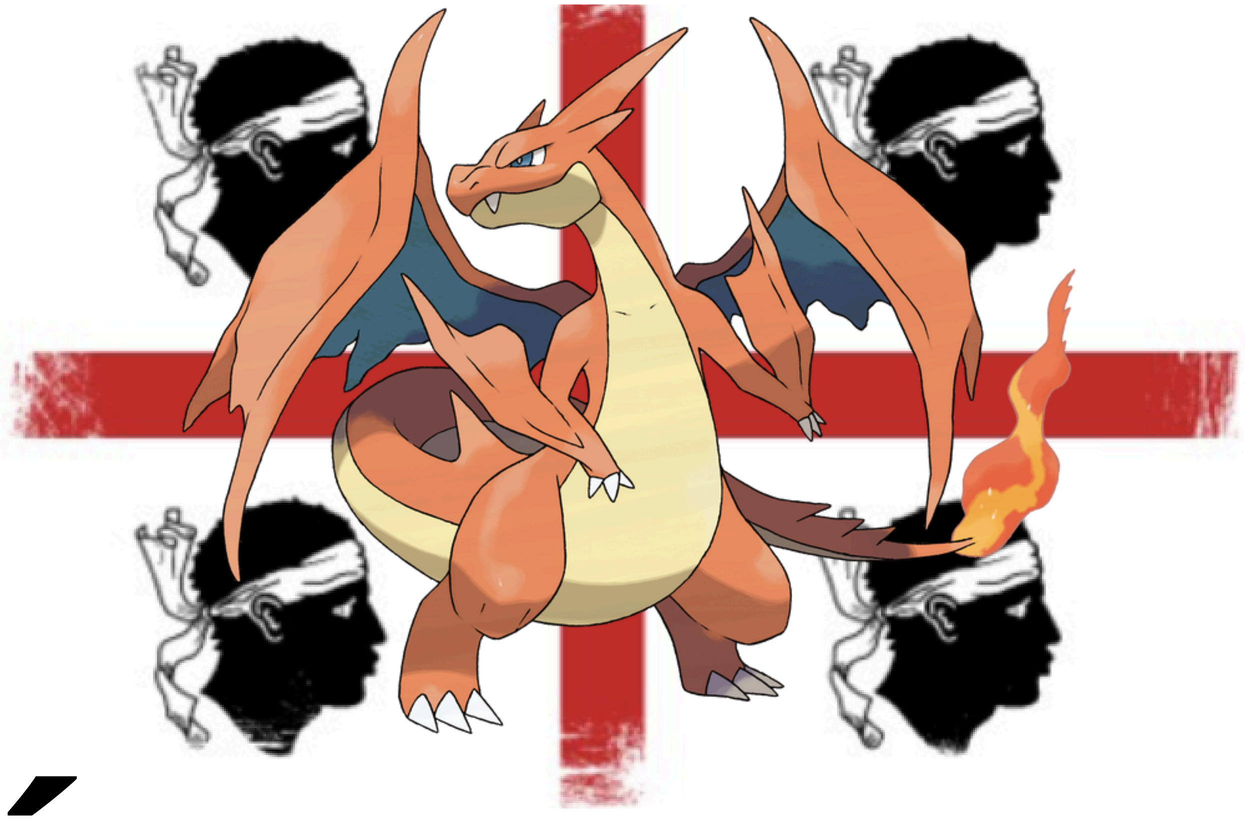


# POKÉMON BATTLES

## Charisardo team

Matilde Bisi, Su Qi Chen, Salvatore Mastrangelo, Antonio Napolitano, Vito Pampinella

# Context and aim

- Each player choses a move

**Team 0** active

party

Possible Moves:

0) ...
1) ...
2) ...
3) ...
4) switch with Pkm 1
5) switch with Pkm 2

- Compute pre-combat damages
- Decide actions order
- Compute post-combat damages
- Check if battle ended

**Team 1** active

party

Possible Moves:

0) ...
1) ...
2) ...
3) ...
4) switch with Pkm 1
5) switch with Pkm 2

**Our aim**:
Build an agent able to play with a random generated team against any random generated team

# Environment and task

**Performance measure:**

- Victory (battle)
- Pkm statistics (each turn)

**Environment**

- Multiagent (2 players)
- Stochastic
- Discrete
- Fully observable
- Sequential
- Known

**Sensors**

- Possible moves
- Pkm statistics
- Opponent possible moves
- Opponent statistics

**Actuators**

- 4 specific actions for each Pkm
- Switch Pkm

# Battle Policies

TREE SEARCH

**RandomPlayer**

Agent that selects actions randomly

**OneTurnLookahead**

Greedy heuristic that prioritizes damage output

variation

**TypeSelector**

Variation of One Turn Look ahead that utilizes a short series of if-else

**BreadthFirstSearch**

Basic tree search that traverses nodes in level order until it finds a state in which the current opponent is fainted, assuming that the opponent always skip turn

variation

**PrunedBFS**

Modification of Breadth First Search, assuming the opponent is One Turn Look ahead. Avoid actions that involve using a damaging move with a resisted type, or that involve switching to a Pkm with a subpar type match up.

assumed opponent

variation

**Minimax**

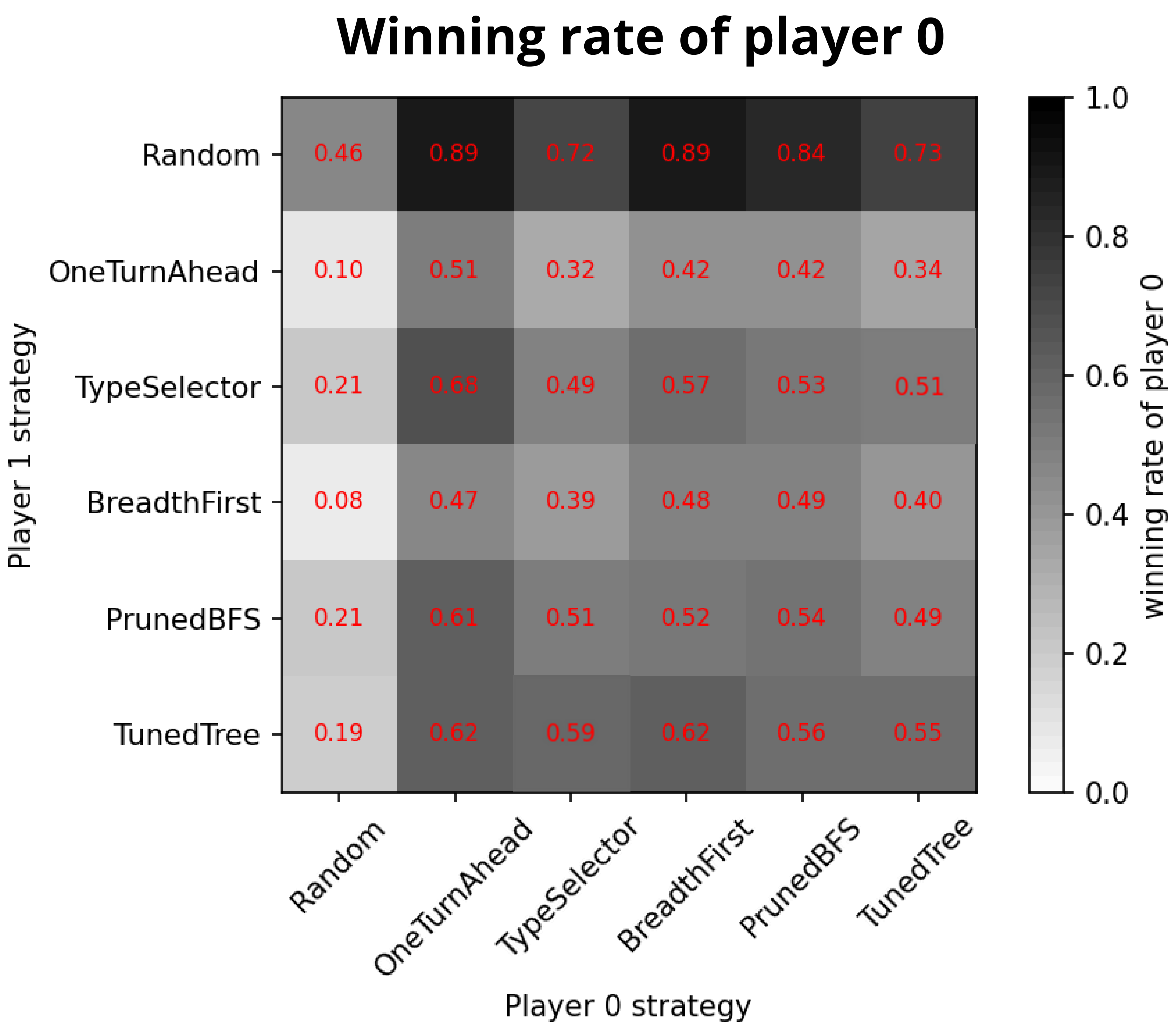
Tree search algorithm that deals with adversarial paradigms by assuming the opponent acts in their best interest

**TunedTreeTraversal**

Agent inspired on PrunedBFS. Assumes opponent is a TypeSelector. It traverses only to states after using moves recommended by a TypeSelector agent and non-damaging moves.

assumed opponent

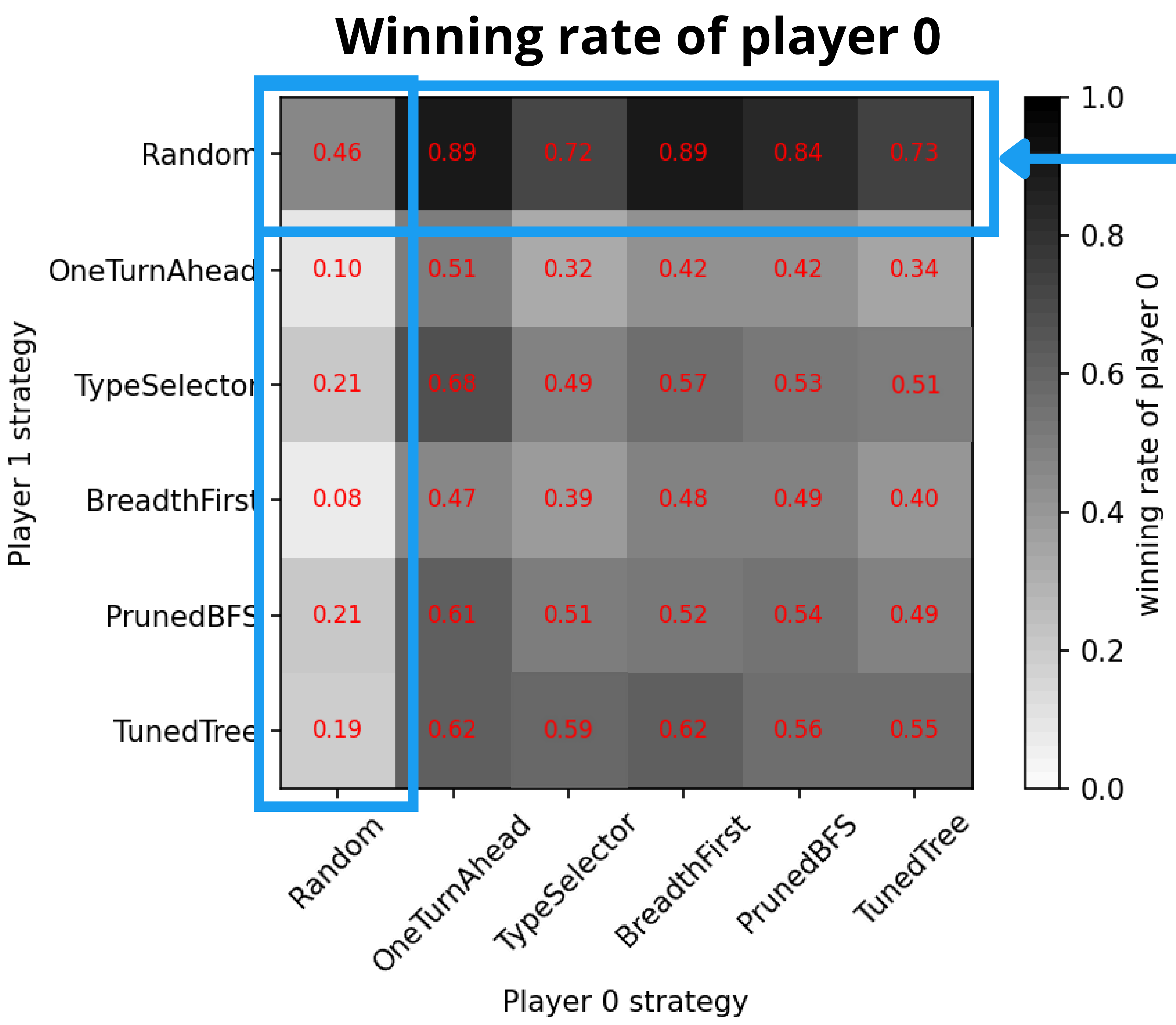
# Battle Policies analysis




Simulation of **100 battles** with random generated teams for each combination of battle policy

- Not very good results: Winning rate is between **0.4** and **0.6** in most cases

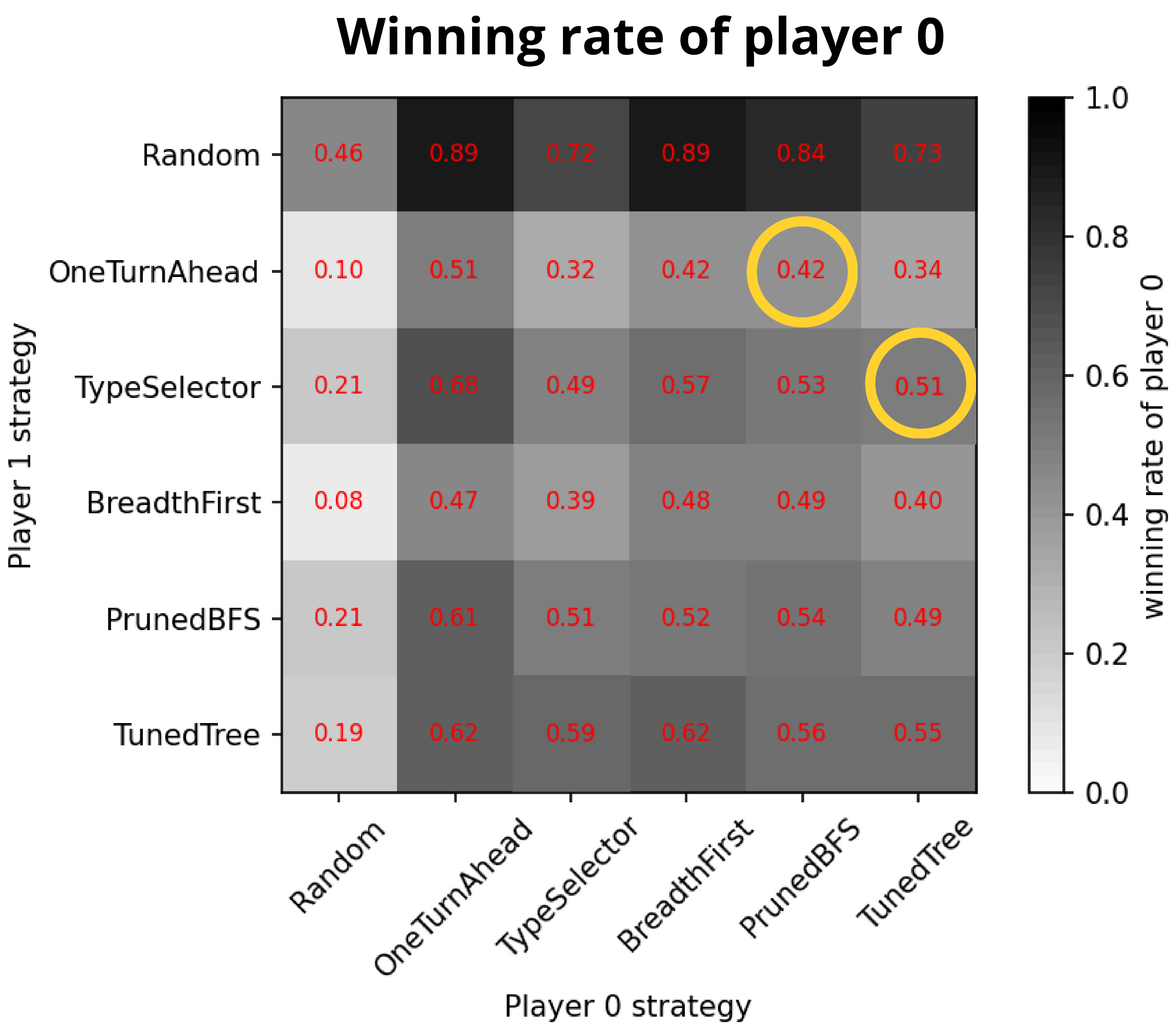
# Battle Policies analysis




Simulation of **100 battles** with random generated teams for each combination of battle policy

- Not very good results: Winning rate is between **0.4** and **0.6** in most cases
- **RandomPlayer** perform worse (as expected)

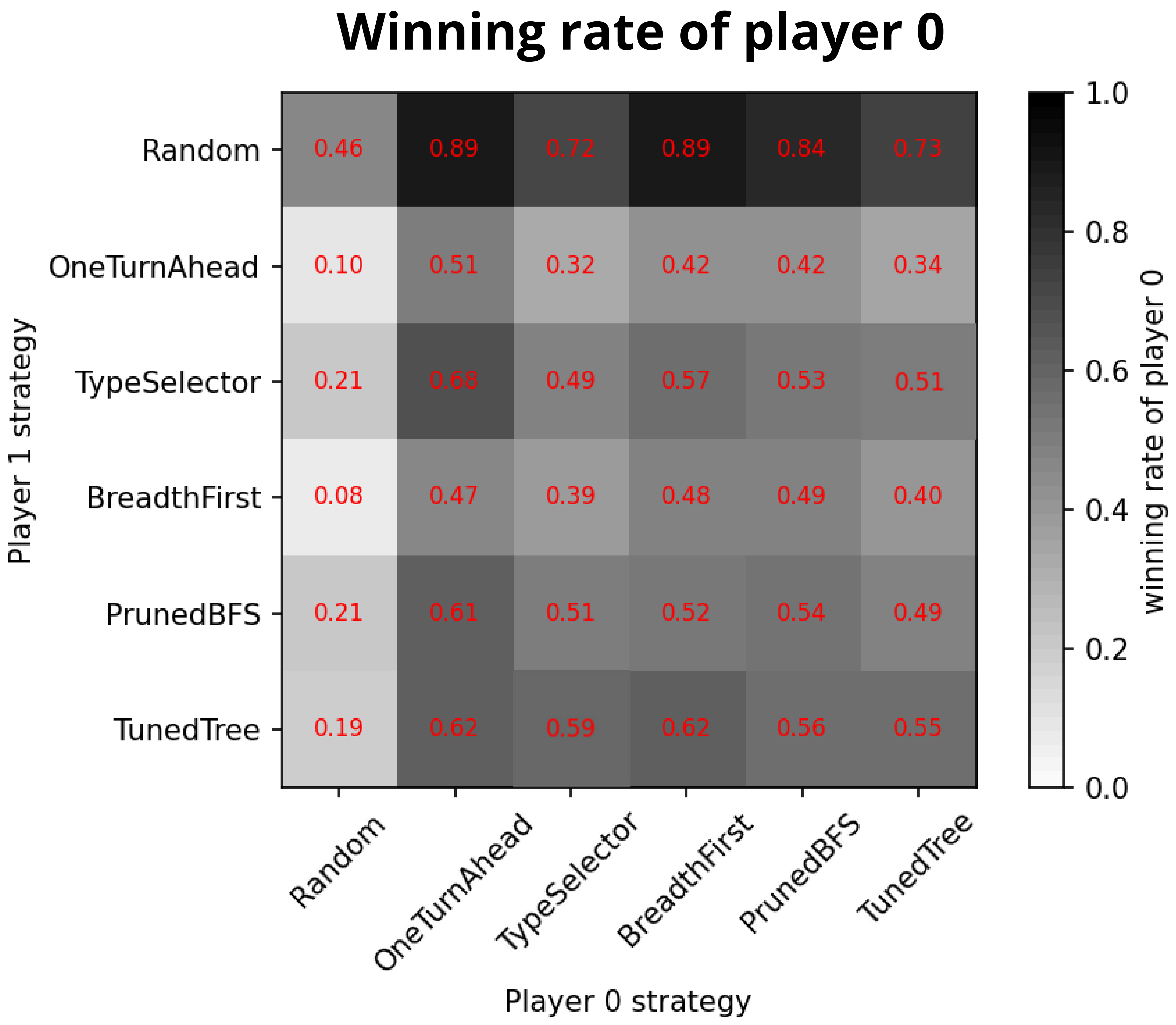
# Battle Policies analysis




Simulation of **100 battles** with random generated teams for each combination of battle policy

- Not very good results: Winning rate is between **0.4** and **0.6** in most cases
- **RandomPlayer** perform worse (as expected)
- Some results are different from what expected

# Battle Policies analysis



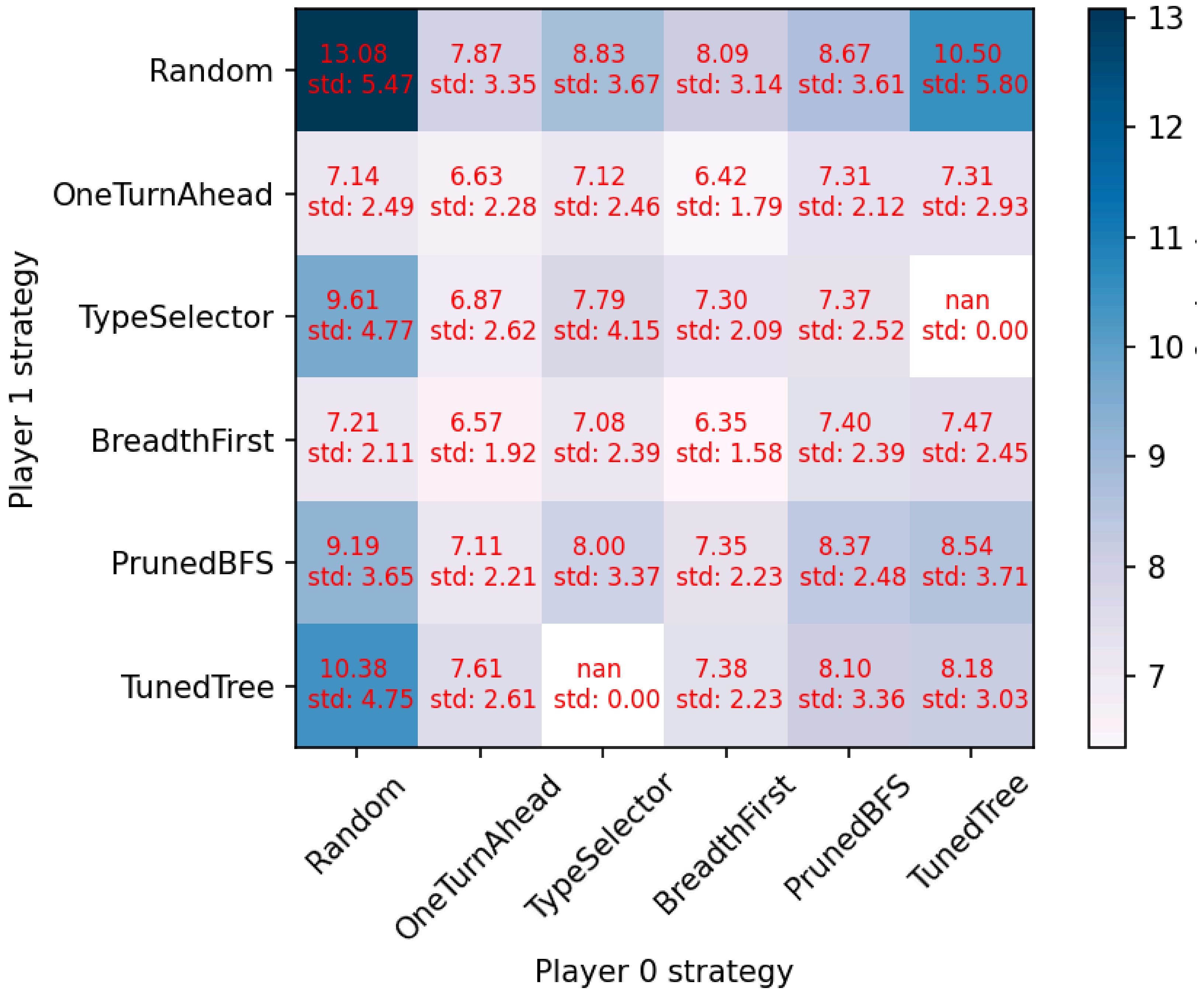

Simulation of **100 battles** with random generated teams for each combination of battle policy

- Not very good results: Winning rate is between **0.4** and **0.6** in most cases
- **RandomPlayer** perform worse (as expected)
- Some results are different from what expected

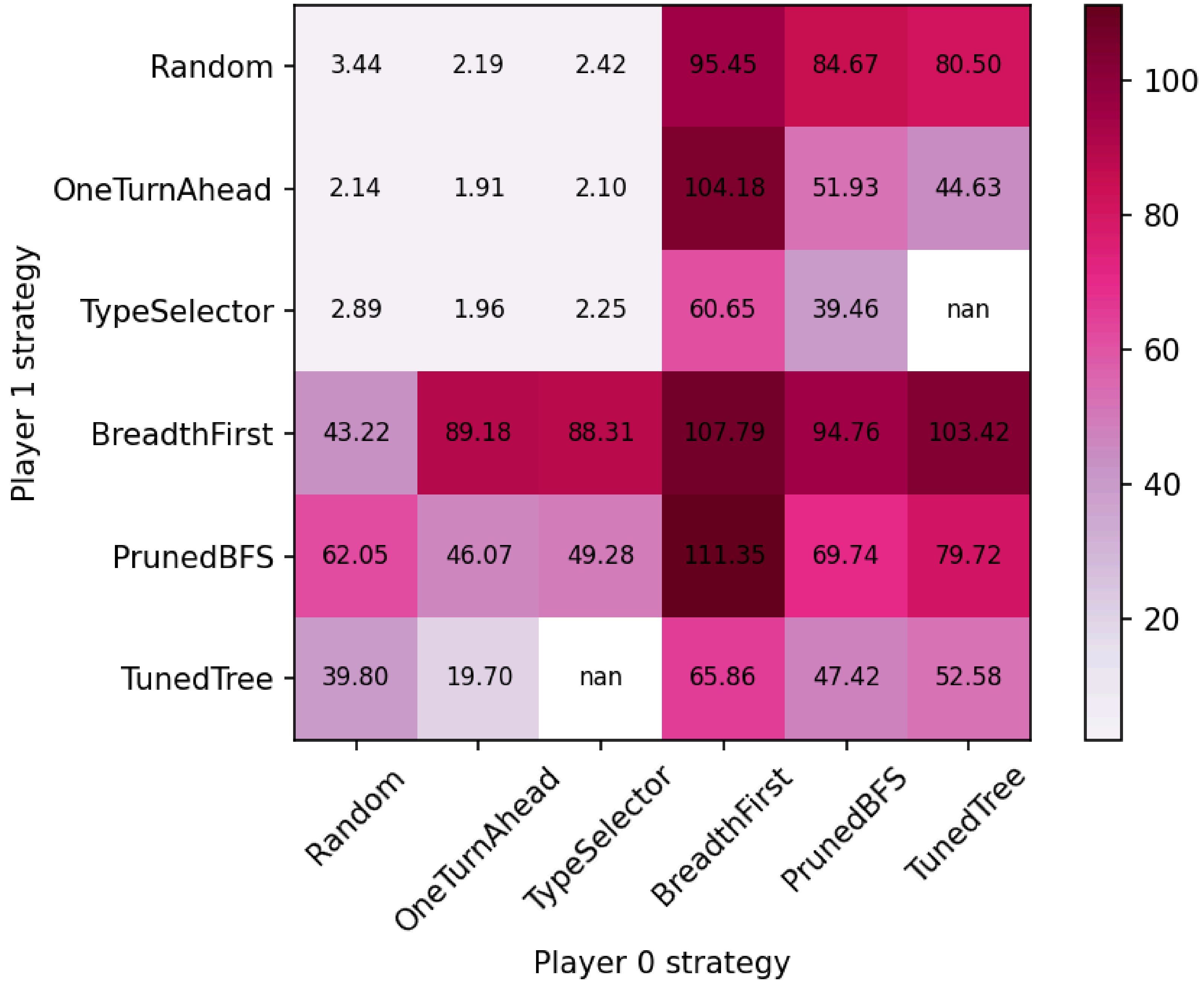
Anyway, accuracy is not high due to the small number of battles simulated

# Battle Policies analysis

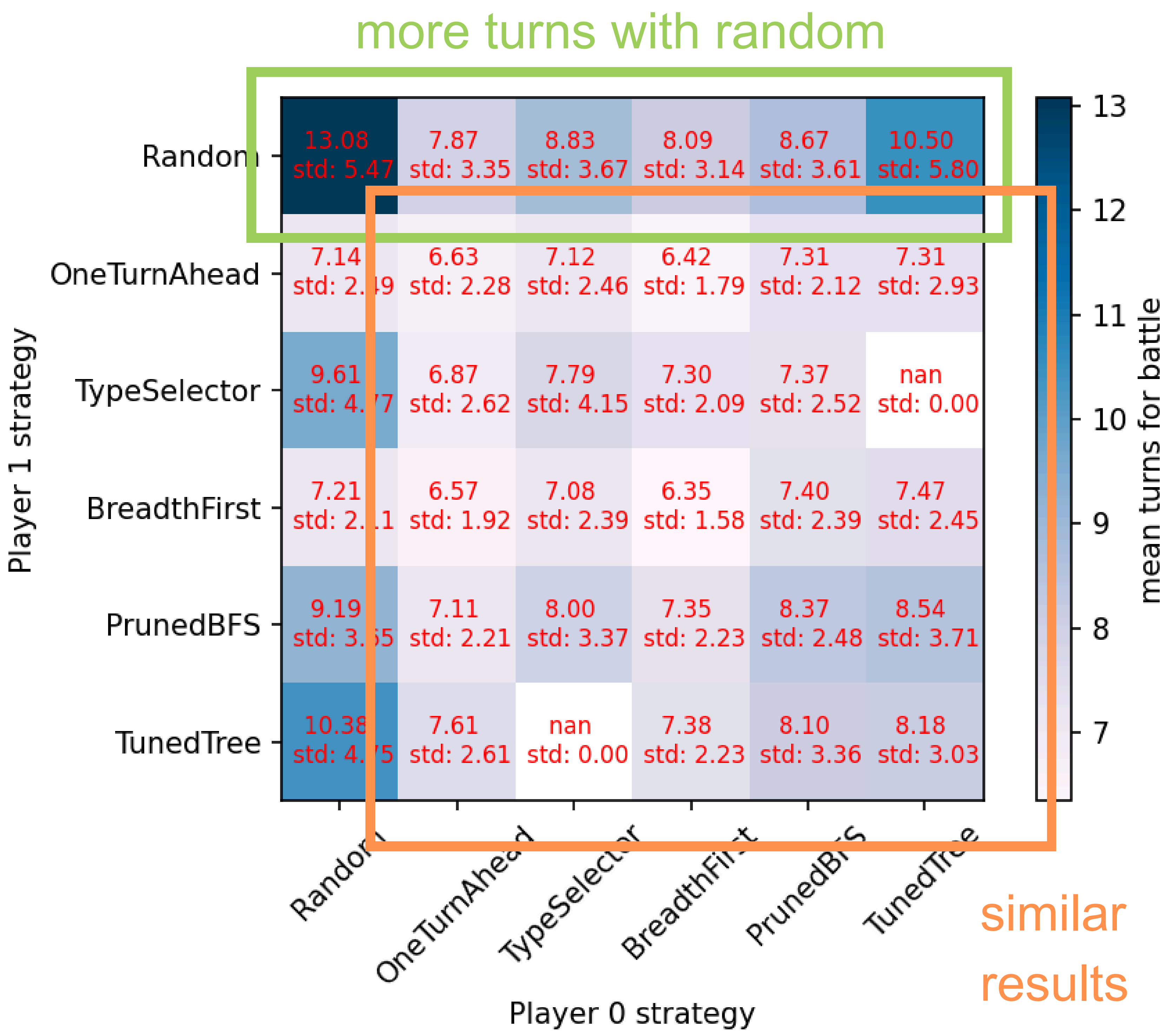
## Mean number of turns

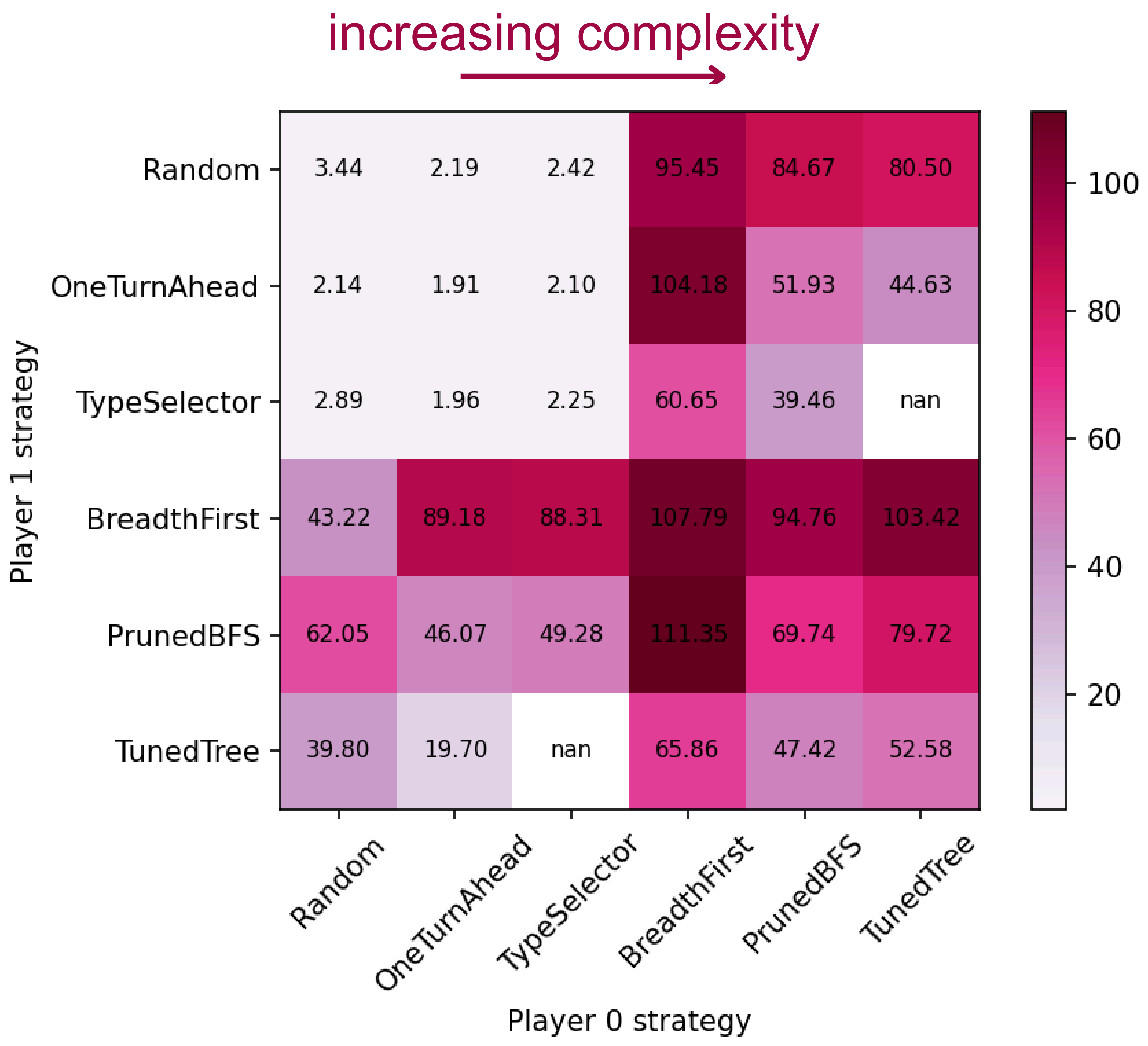



## Time to perform 100 battles [s]

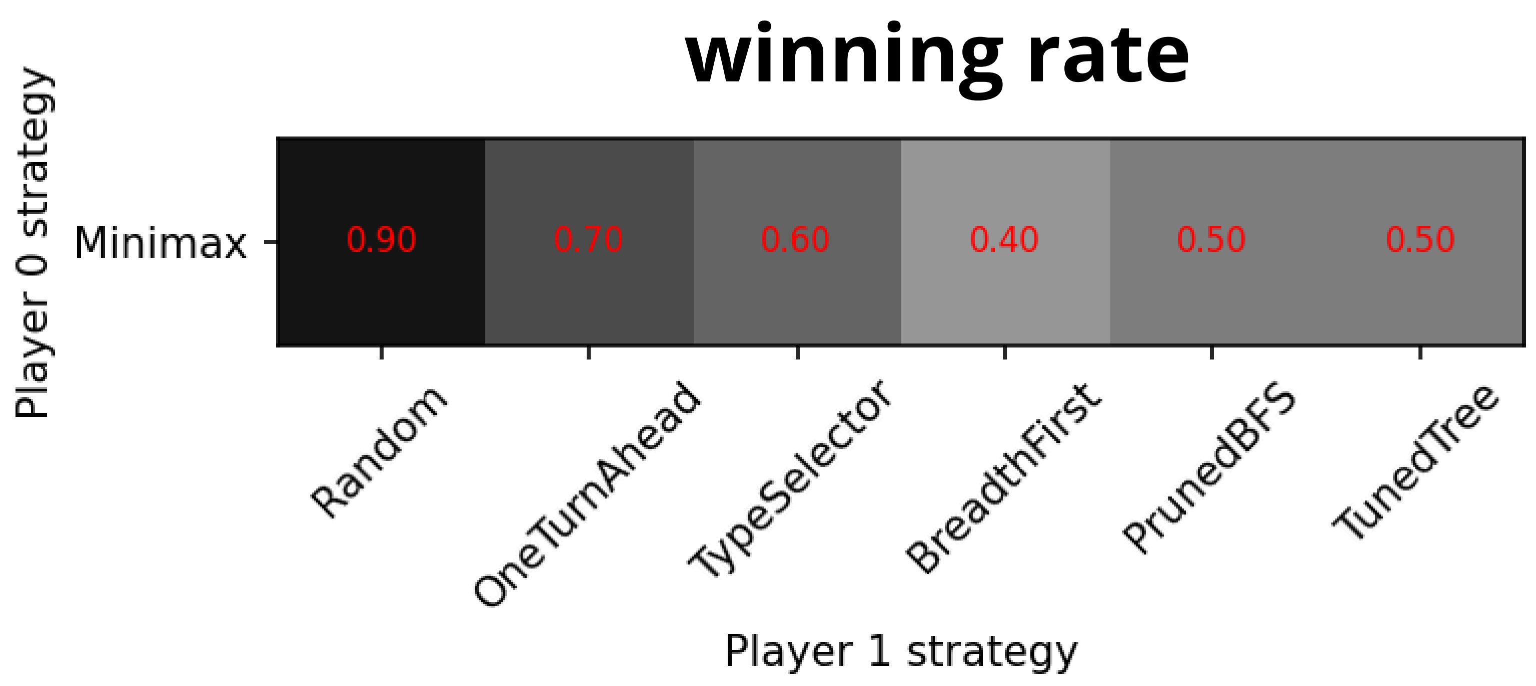


# Battle Policies analysis



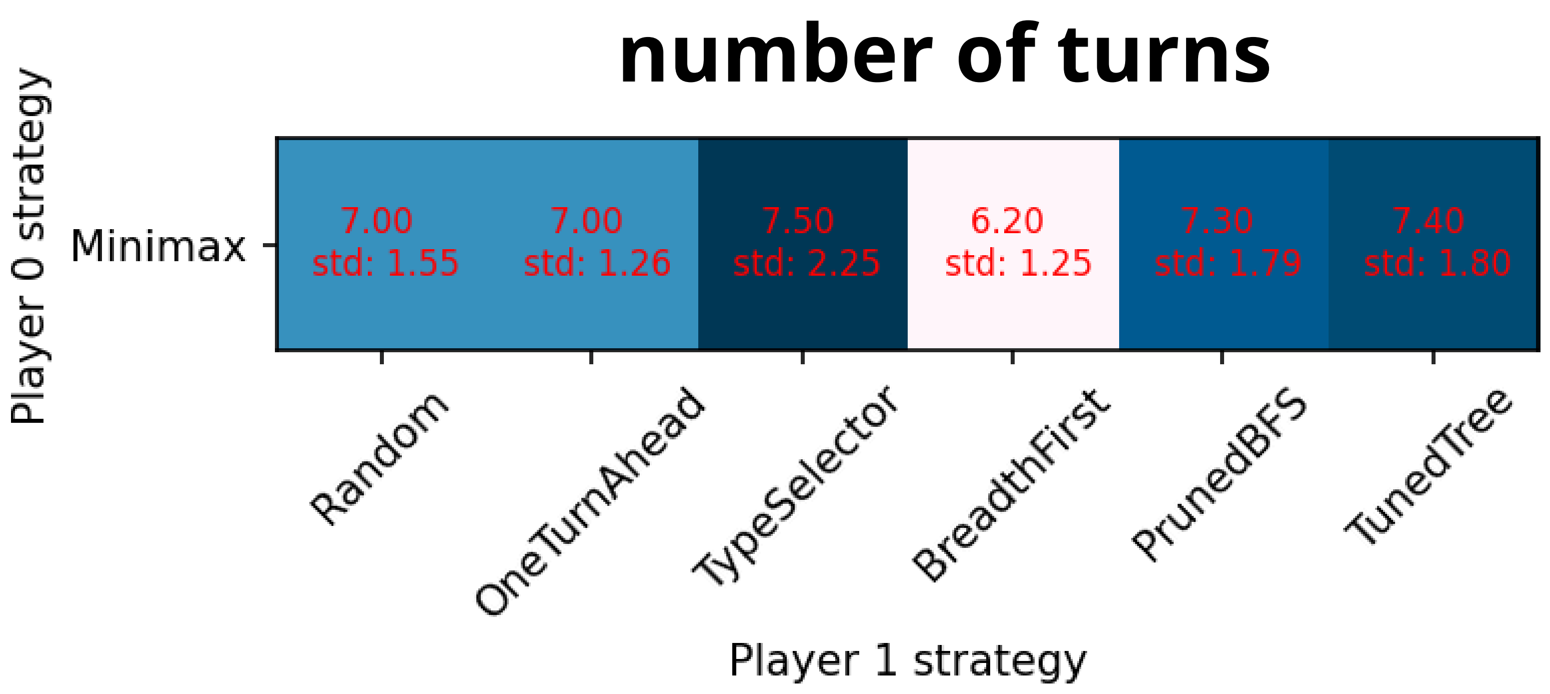



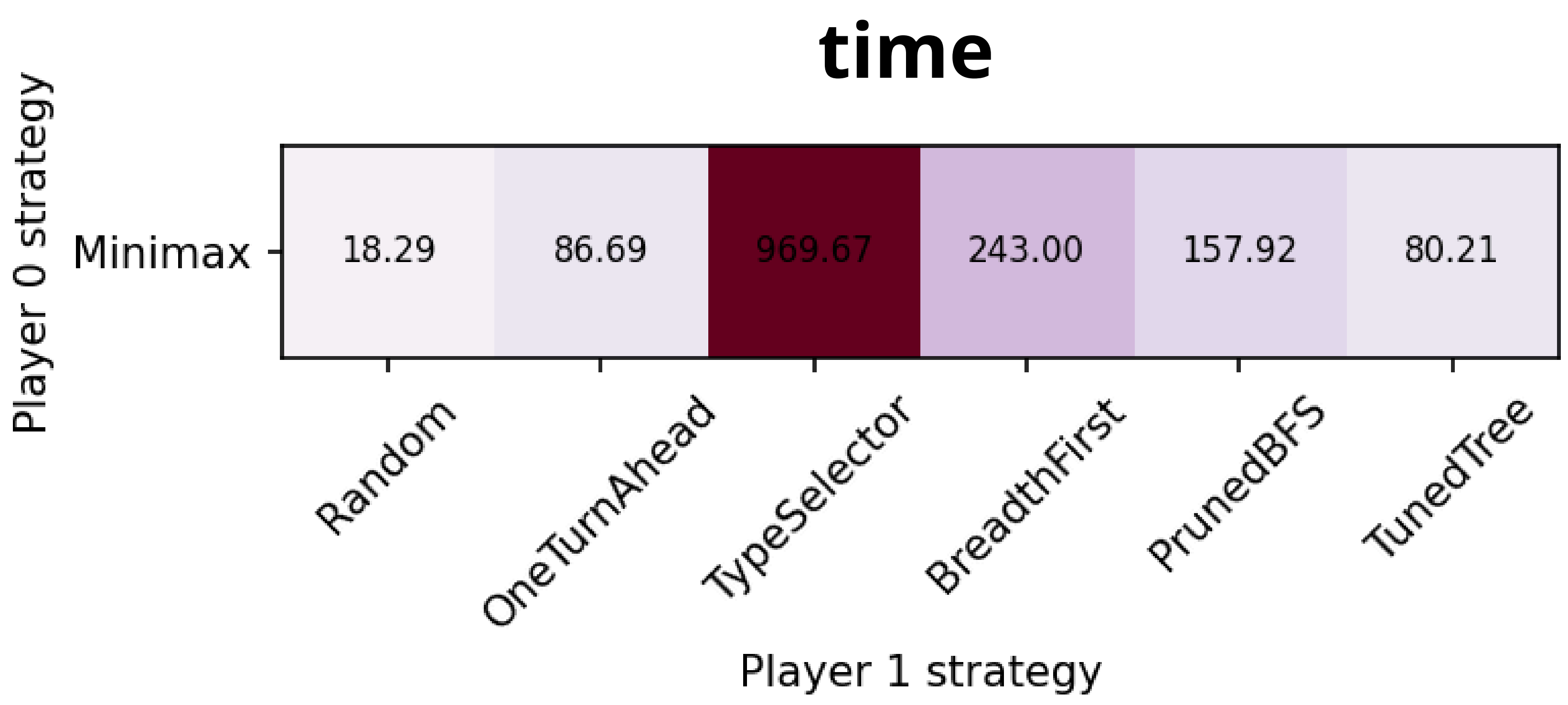
# Battle Policies analysis - Minimax




Simulation of **10 battles** due to the slowness of the policy

- Slower
- Limited advantage



# Our agents - Heuristical

**Calculate best move with a greedy approach**

`Best_Moves(source, target)`

Function that given two pokemons, returns a list of the best moves the source pokemon must do to kill the target

⟶

Use it to evaluate the best moves for our active pokemon on opponent pokemon and vice versa

CONSIDERATIONS:

Computation is faster but code has to be very complex for the algorithm to be better, and always remains an heuristic

# Our agents - Heuristical (algorithm)

**Calculate the probability of winning with an euristic algorithm**

If our pokemon can kill the other one with less moves, calculate the probability that all moves will strike

If our pokemon has less moves to the KO than the opponent's one, add to the calculation the probability that the opponent pokemon misses enough moves so that ours wins.

If our pokemon is slower, consider an ulterior error of the opponent

If both pokemon have the same number of moves, consider velocity

**Repeat calculation for non-active pokemons**

An active pokemon loses a turn for the switch, add a dummy move with 0 dmg and 100 accuracy →

Consider entry hazard damage, subtracting it to the HP of our pokemon before any calculations →

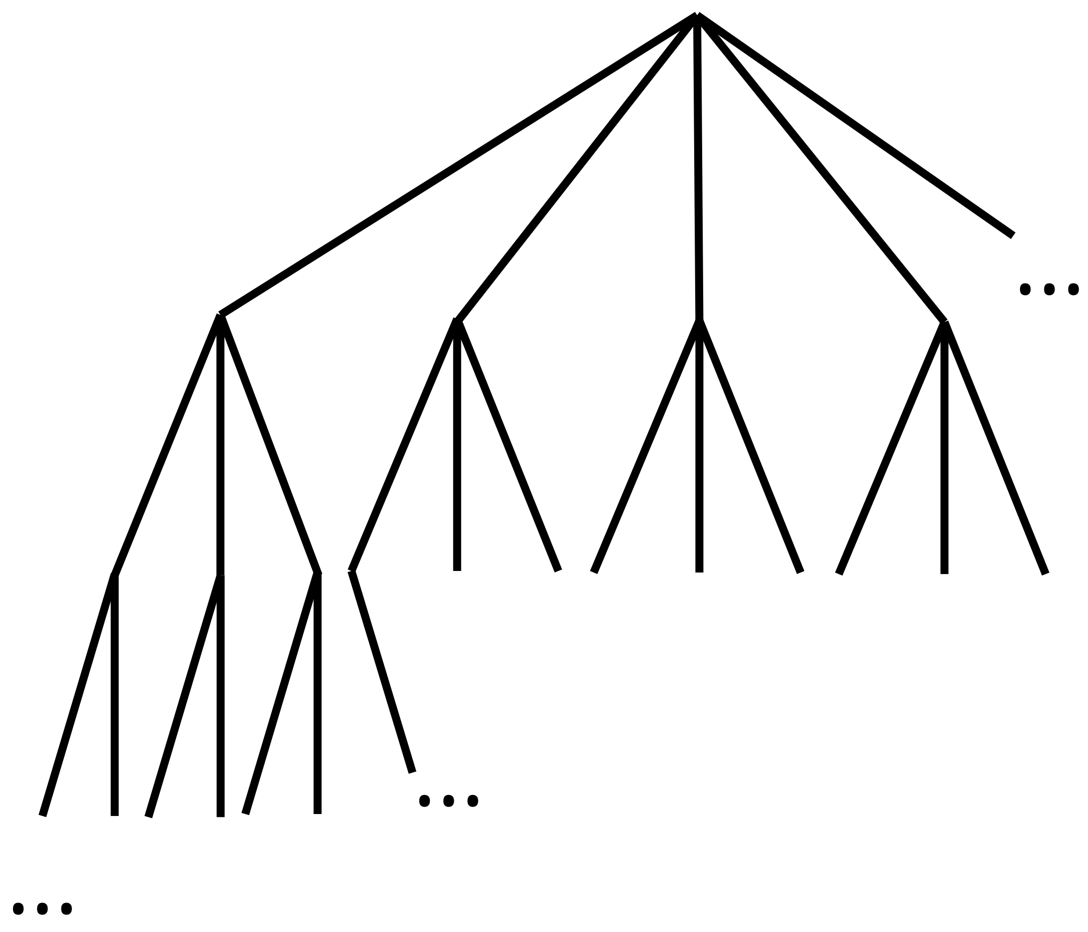
After the calculation on all three of our pokemons, we take the n-th root, where n is the respective number of moves, and choose the move based on the max value

CONSIDERATIONS:

Computation is faster but code has to be very complex for the algorithm to be better, and always remains an heuristic

# Our agents - Pruned Tree Search

- Each turn is **our move + opponent move** → 6*6 = 36 new nodes from each node

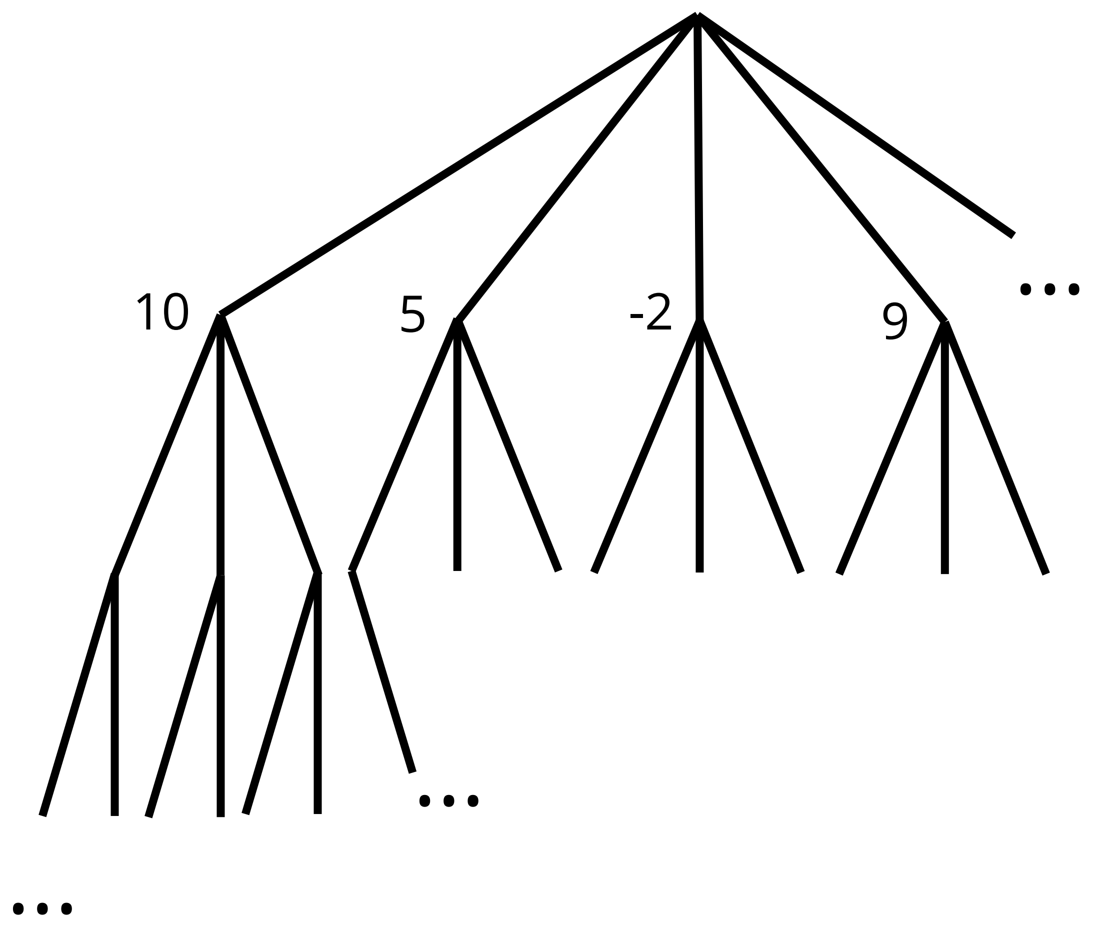


CONSIDERATIONS: Code is not too complex and approach can be arbitrarly accurate, but the computational load increases very fast with the accuracy

# Our agents - Pruned Tree Search

- Each turn is **our move + opponent move** → 6*6 = 36 new nodes from each node
- Each position has **points** based on:
  - Match up of our active Pkm against the opponent
  - Total HP left
  - Entry hazard
  - Statuses
  - ...

→ Evaluation based on the minimum number of points
between all possible positions determined by opponent moves

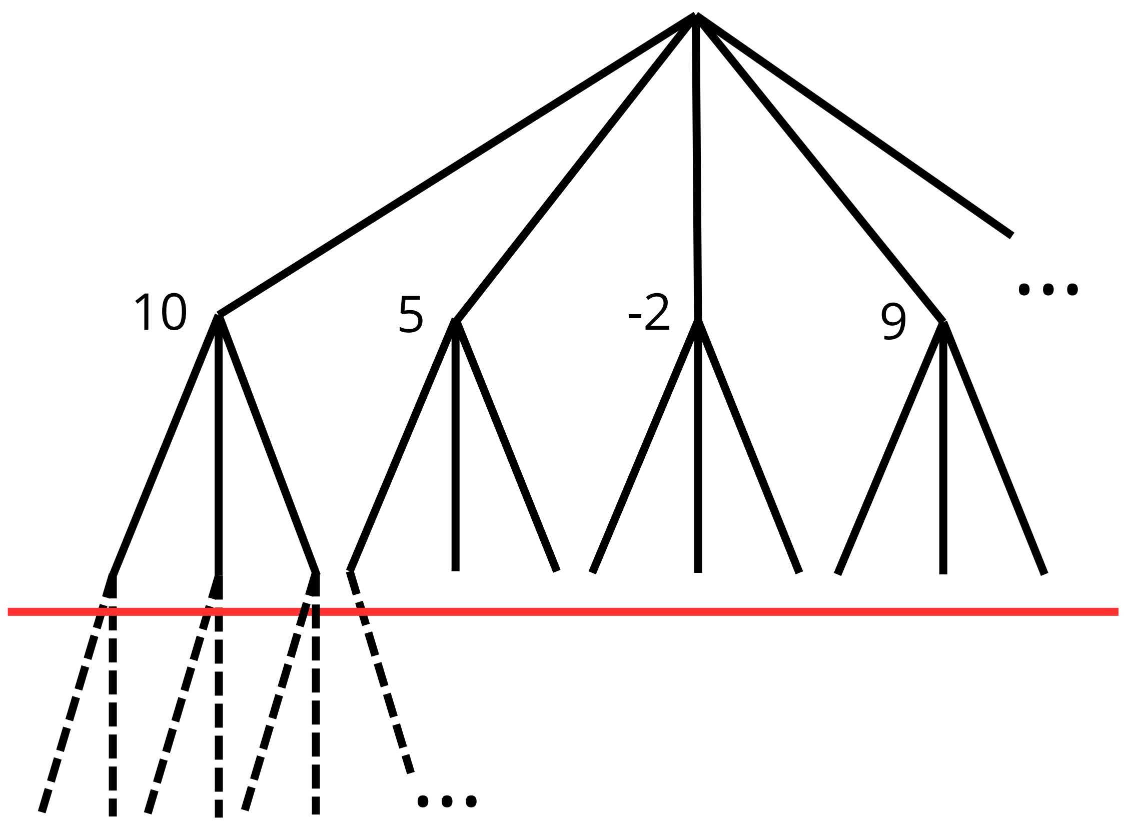



CONSIDERATIONS: Code is not too complex and approach can be arbitrarily accurate, but the computational load increases very fast with the accuracy

# Our agents - Pruned Tree Search

- Each turn is **our move + opponent move** → 6*6 = 36 new nodes from each node
- Each position has **points** based on:
  - Match up of our active Pkm against the opponent
  - Total HP left
  - Entry hazard
  - Statuses
  - ...

→ Evaluation based on the minimum number of points
between all possible positions determined by opponent moves



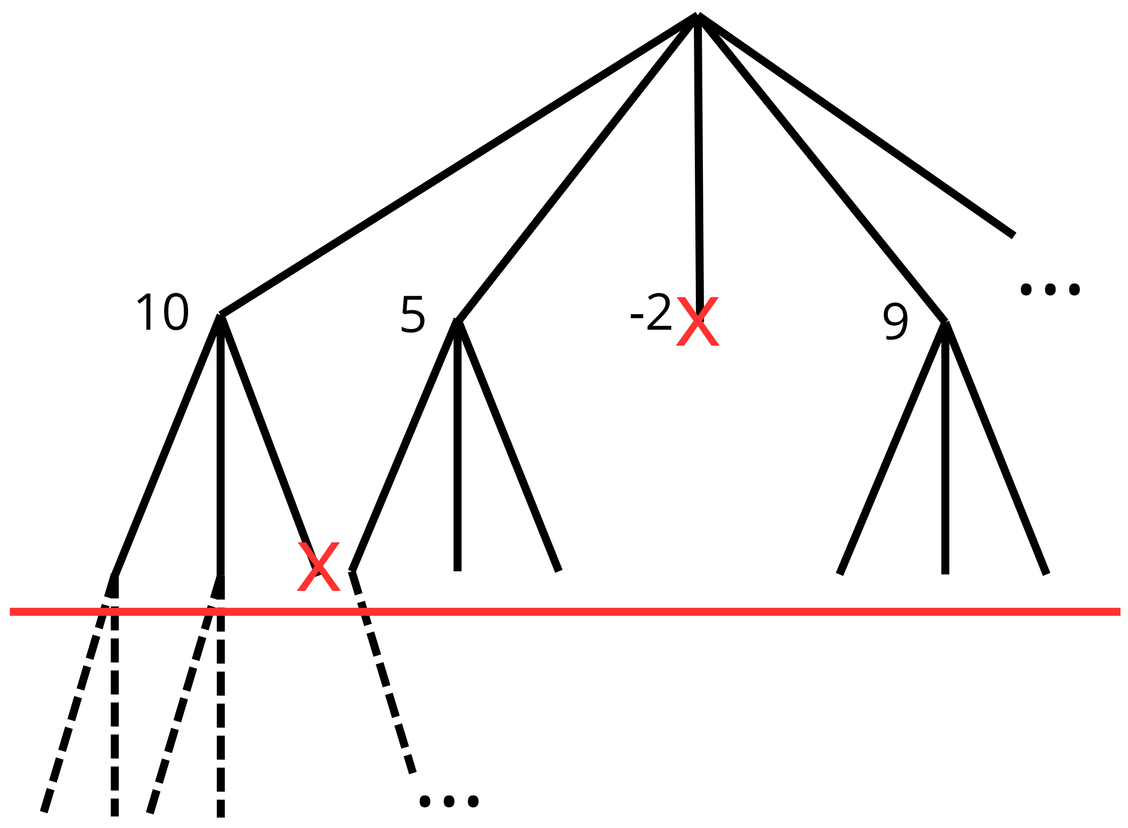

- Bredth first with **limited levels**

CONSIDERATIONS: Code is not too complex and approach can be arbitrarly accurate, but the computational load increases very fast with the accuracy

# Our agents - Pruned Tree Search

- Each turn is **our move + opponent move** → 6*6 = 36 new nodes from each node
- Each position has **points** based on:
    - Match up of our active Pkm against the opponent
    - Total HP left
    - Entry hazard
    - Statuses
    - ...

→ Evaluation based on the minimum number of points between all possible positions determined by opponent moves




- Bredth first with **limited levels**
- **Pruning** of too negative moves

CONSIDERATIONS: Code is not too complex and approach can be arbitrarily accurate, but the computational load increases very fast with the accuracy

# Our agents - Pruned Tree Search

- Each turn is **our move + opponent move** → 6*6 = 36 new nodes from each node
- Each position has **points** based on:
  - Match up of our active Pkm against the opponent
  - Total HP left
  - Entry hazard
  - Statuses
  - ...

→ Evaluation based on the minimum number of points between all possible positions determined by opponent moves

- Bredth first with **limited levels**
- **Pruning** of too negative moves

Key points:

- Find a good way to define utility of a position
- Find best number of levels (trade-off between cost and accuracy)

↓

We will try different possibilities

CONSIDERATIONS: Code is not too complex and approach can be arbitrarily accurate, but the computational load increases very fast with the accuracy

# Assessment

- Repeat initial analysis with our agents
- Battles between our agents and other competitors with different battle policies
- Evaluation on **winning rate** and **time/turns** to win
- Evaluation on how results vary with respect to some parameters (for example, number of levels in tree serch)