State predicting Nim Agent

Florentin-Cristian Udrea (S319029)

Impartial game theory

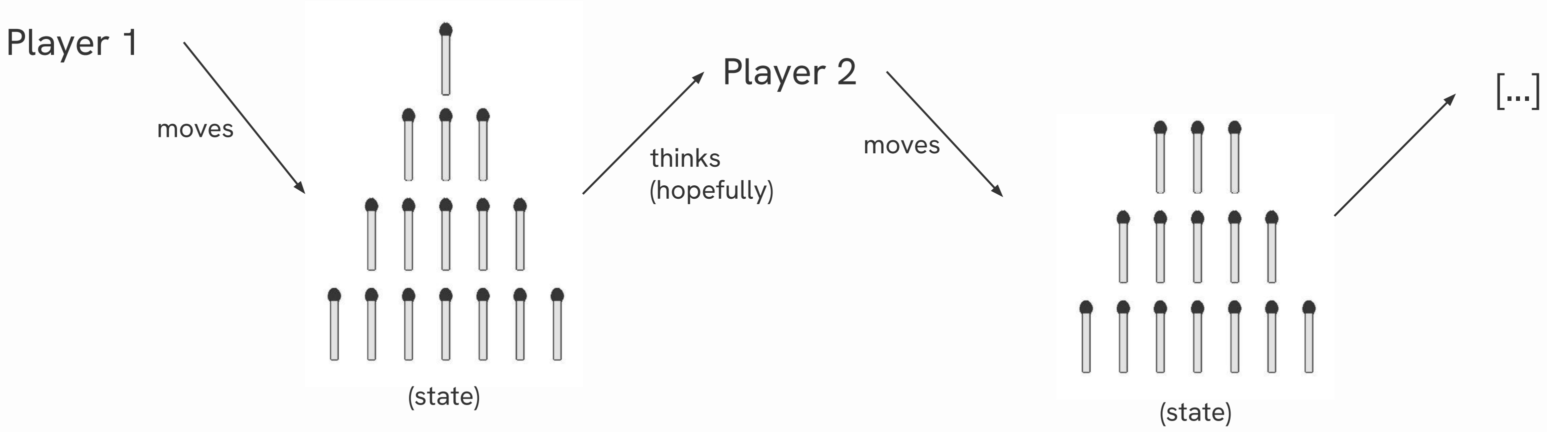
- Players taking turns
- No hidden information
- Players have the same legal moves from any given position

Impartial game theory

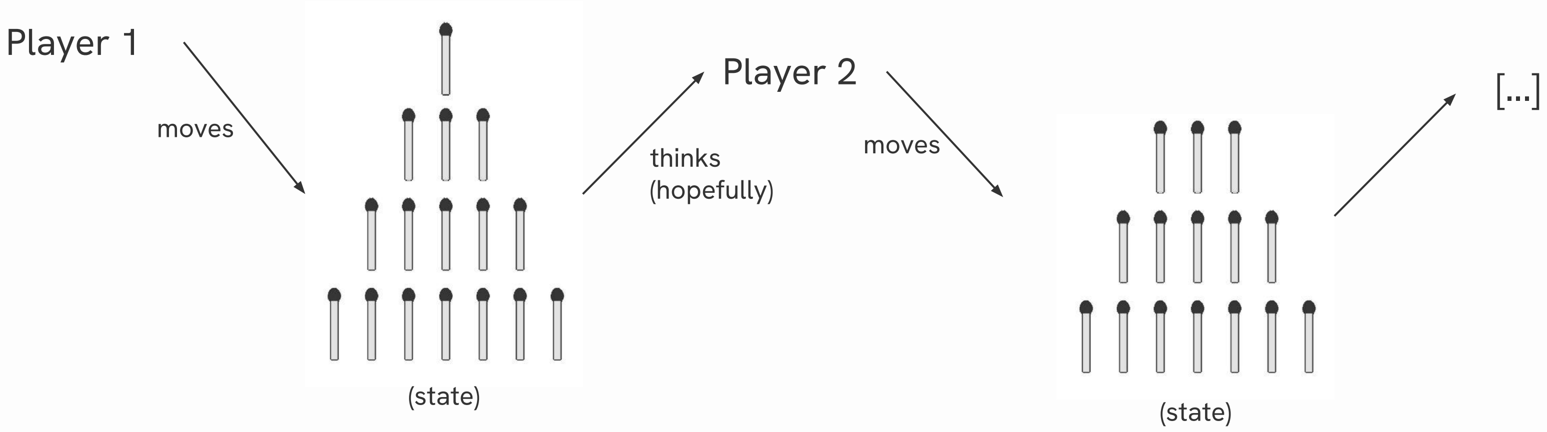
- Players taking turns
- No hidden information
- Players have the same legal moves from any given position

The position you are in defines if you're winning or losing

P-positions & N-positions

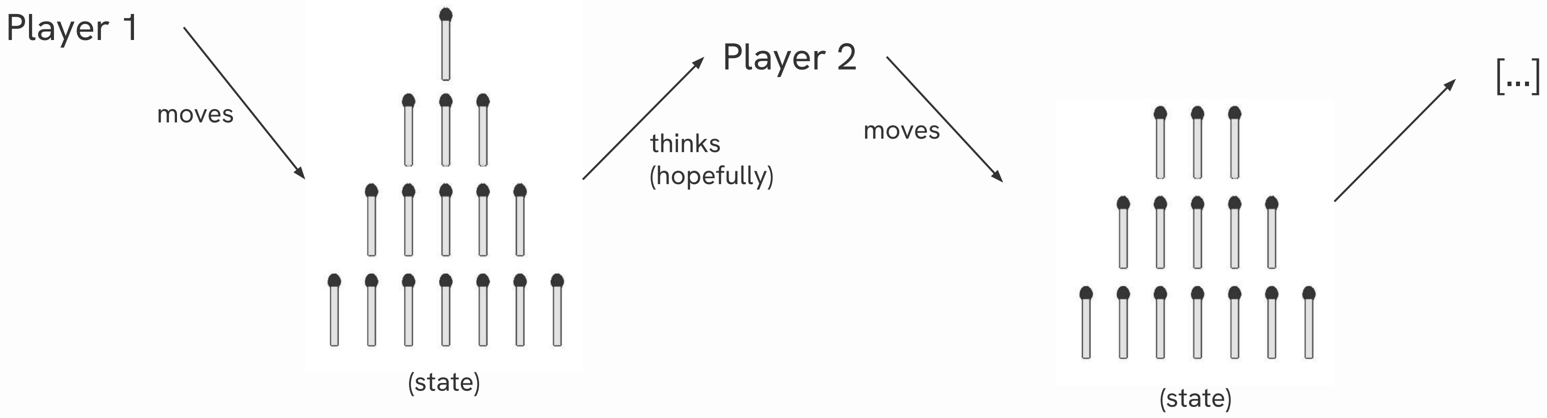


P-positions & N-positions



Every state can be defined as
a P-position or a N-position

P-positions & N-positions



N-position: the game is in a n-position if a win is secured for the **N**ext player to move

P-position: the game is in a n-position if a win is secured for the **P**revious player that moved

Idea & Strategy

Always choose the move that gets the next state in a P-position

Idea & Strategy

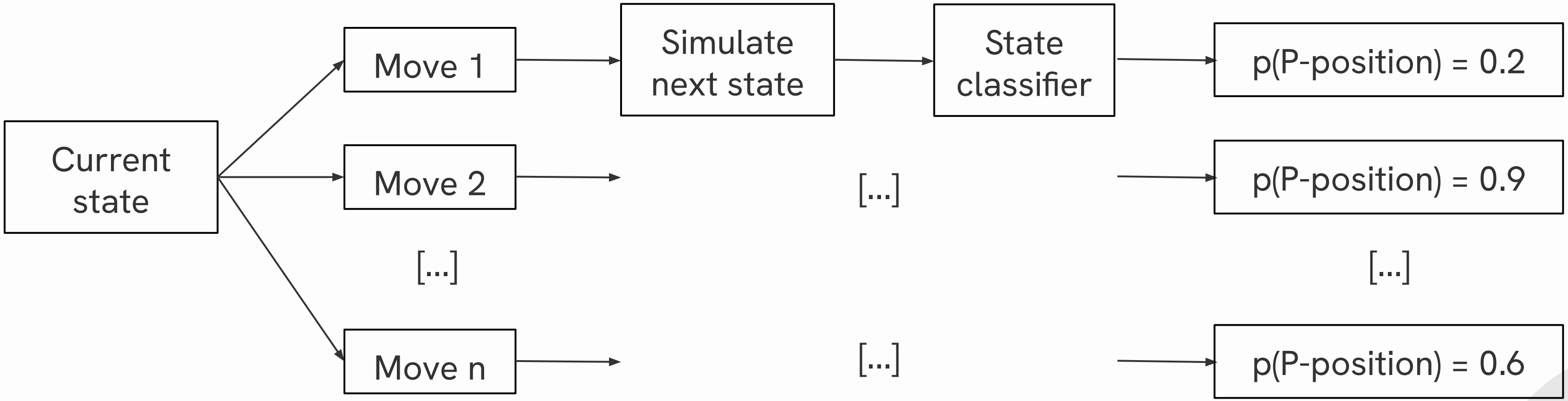
Always choose the move that gets the next state in a P-position

Use a binary classifier to predict if the new state is a P-position or N-position

Idea & Strategy

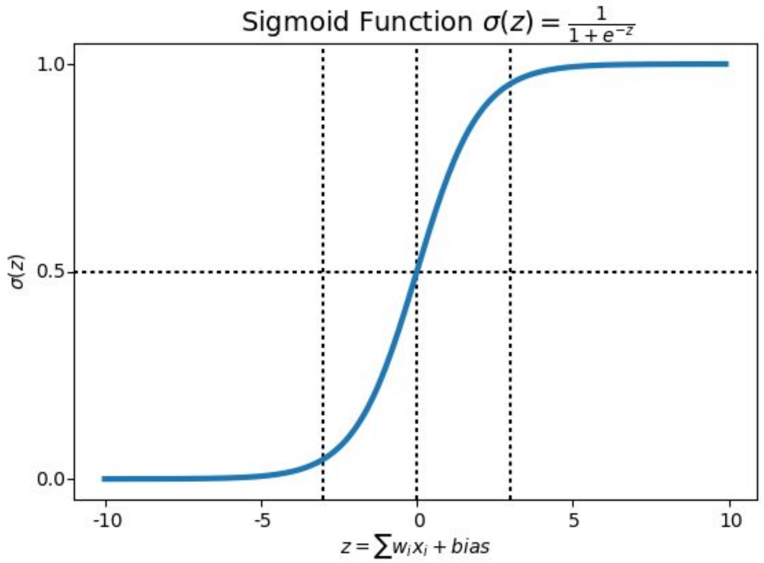
Always choose the move that gets the next state in a P-position

Use a binary classifier to predict if the new state is a P-position or N-position



Regression for Classification - Sigmoid

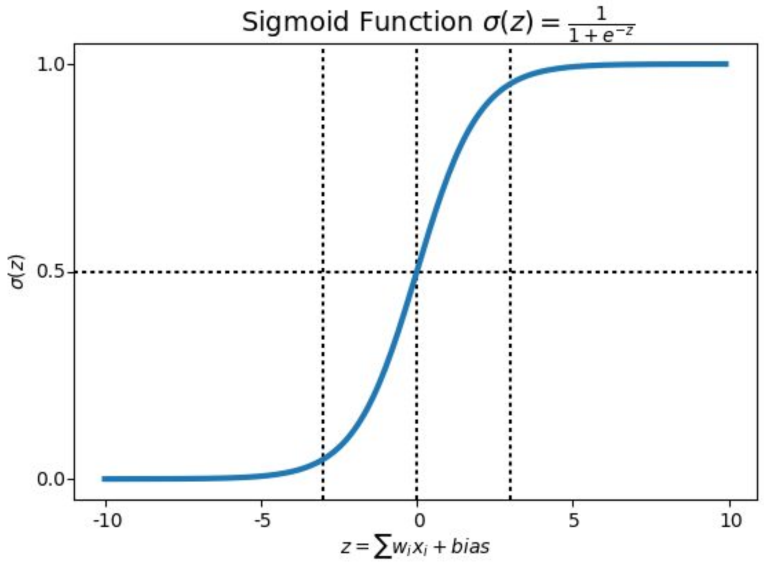
Linear regression, then put into the sigmoid function to get a (0,1) bounded result



Regression for Classification - Sigmoid

Linear regression, then put into the sigmoid function to get a (0,1) bounded result

Can be interpreted as binomial probability



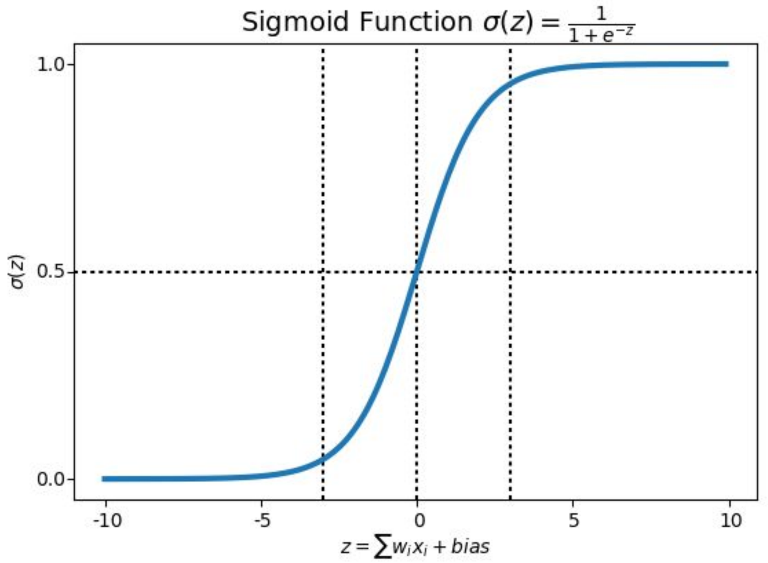
Regression for Classification - Sigmoid

Linear regression, then put into the sigmoid function to get a (0,1) bounded result

Can be interpreted as binomial probability

Mainly known as ***'Logistic classifier'***

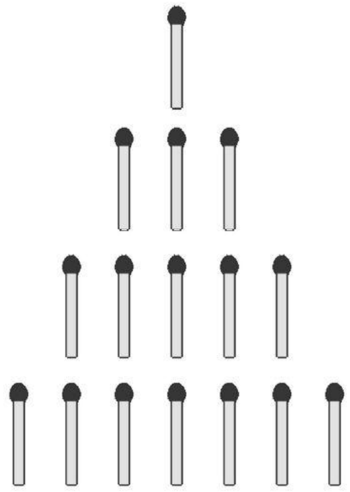
```
def get_logistic_score(features, weights):
  z = np.dot(features, weights)
  logistic_score = 1 / ( 1 + np.exp(-z) )
  return logistic_score
```



What are we predicting from ?

General, hand-extracted game features, such as:

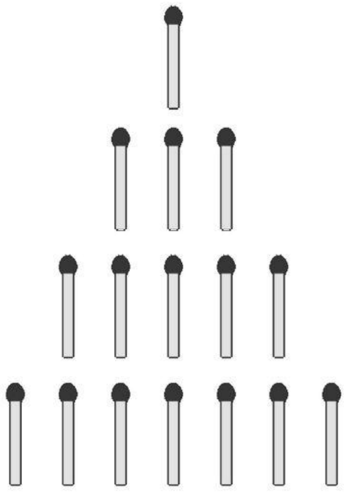
- Number of non-zero rows remaining



What are we predicting from ?

General, hand-extracted game features, such as:

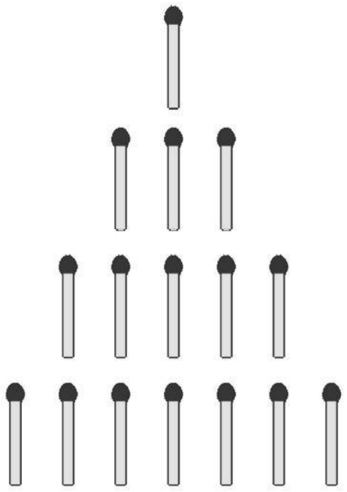
- Number of non-zero rows remaining
- Number of non-zero rows remaining having odd number of objects



What are we predicting from ?

General, hand-extracted game features, such as:

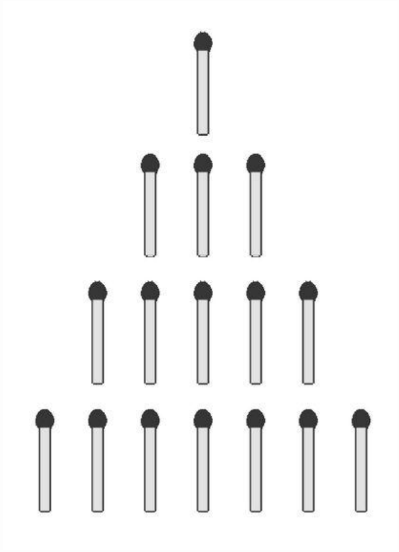
- Number of non-zero rows remaining
- Number of non-zero rows remaining having odd number of objects
- Maximum objects on a row



What are we predicting from ?

General, hand-extracted game features, such as:

- Number of non-zero rows remaining
- Number of non-zero rows remaining having odd number of objects
- Maximum objects on a row
- Mean value of non zero rows
- Etc.

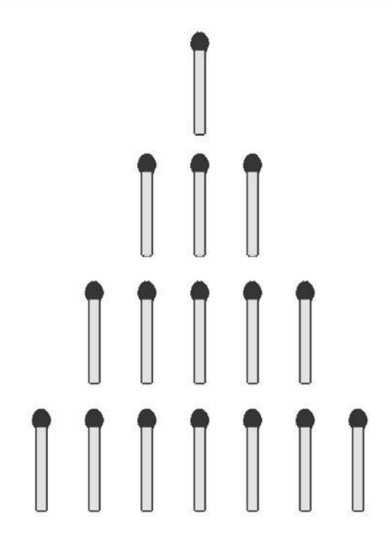


What are we predicting from ?

In order to grasp nonlinear relationships I added their polynomial combinations up to the **third degree**, for a total of 165 features

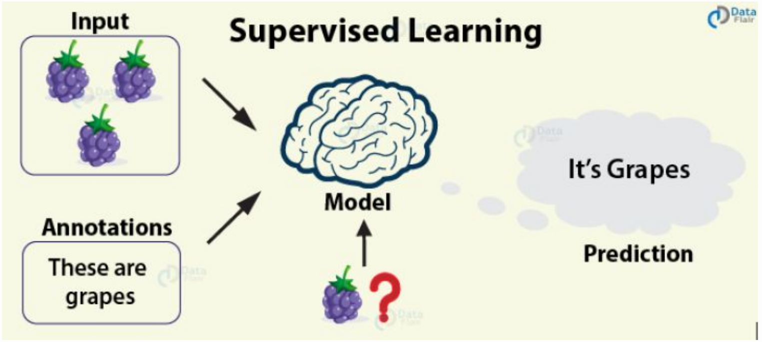
$$[x_1, \quad x_2] \longrightarrow [1, \quad x_1, \quad x_2, \quad x_1^2, \quad x_1 x_2, \quad x_2^2]$$

(*example:* only up to the second degree, having only two features)



No supervised learning

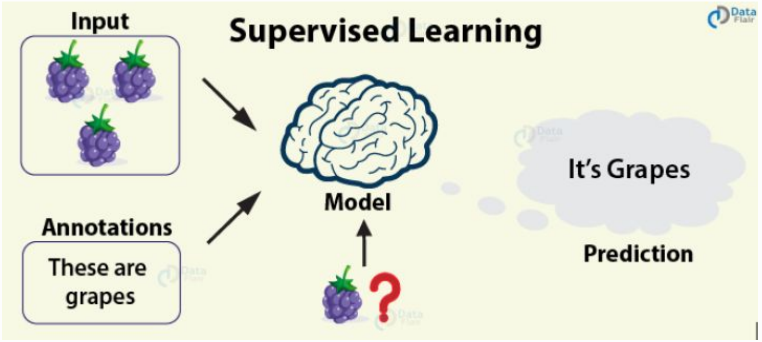
- Classification tasks use annotated data
- We have no data or annotations



No supervised learning

- Classification tasks use annotated data
- We have no data or annotations

Classification performance: Indirectly estimate using the average win-rate as a measure



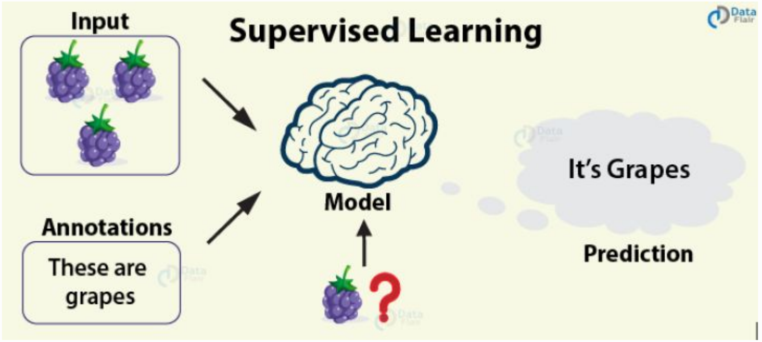
No supervised learning

- Classification tasks use annotated data
- We have no data or annotations

↓

Classification performance:
Indirectly estimate using the average win-rate as a measure

Optimization strategy:
Use evolutionary strategy to improve the weights of the regressor



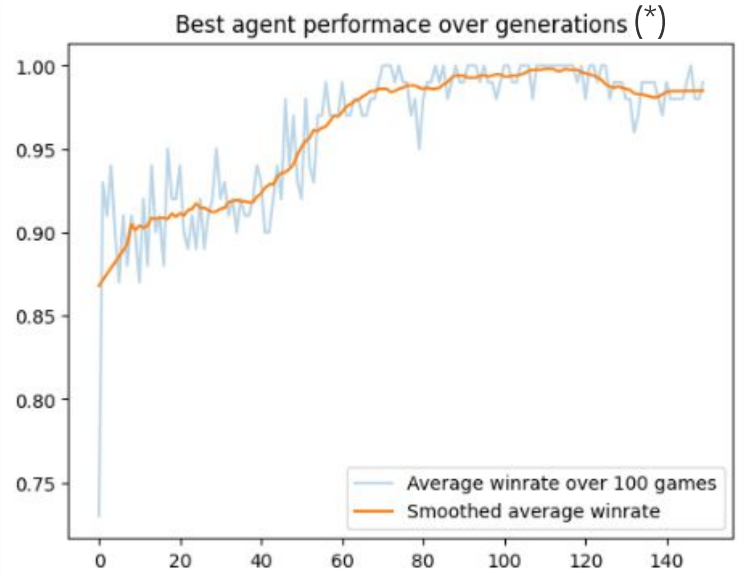
Implementation details

- ***Mutation***: one random weight gets changed by a gaussian sampled value
- ***Standard deviation***: gets lower linearly as we get to the final generations
  (assuming we explored enough and want to fine-tune)
- ***Cross-over***: one-cut crossover
- ***Selective pressure***: parents are selected randomly with probabilities
  proportional to their fitness (only *25% of the population*)

Even more details

- ***Population size*** = 30
- ***Mutation rate*** = 0.2
- ***Number of generations*** = 150
- ***Fitness matches*** = 100

Results



Final agent trained against *pure_random* in fitness

```
Fitness (vs. random) -> 0.893
Fitness (vs. gabriele) -> 1.0
Fitness (vs. optimal) -> 0.656
```

→ average winrate over 1000 games

Final agent trained against *optimal* in fitness (*)

```
Fitness (vs. random) -> 0.926
Fitness (vs. gabriele) -> 1.0
Fitness (vs. optimal) -> 0.969
```

→ average winrate over 1000 games

(*) made after lab deadline, so not on github yet

Issues

- Very slow training (3.5h for 150 generations, with multithreading)

Issues

- Very slow training (3.5h for 150 generations, with multithreading)
- Has to evaluate ALL possible moves, may be slow if Nim dimension is high

Issues

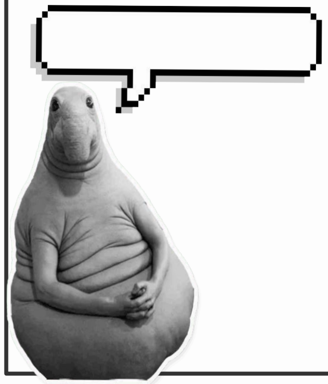
- Very slow training (3.5h for 150 generations, with multithreading)
- Has to evaluate ALL possible moves, may be slow if Nim dimension is high
- Result depends on the agent used in the fitness

Issues

- Very slow training (3.5h for 150 generations, with multithreading)
- Has to evaluate ALL possible moves, may be slow if Nim dimension is high
- Result depends on the agent used in the fitness
- Very dependent on the quality of the features

Thanks!

Florentin-Cristian Udrea (S319029)



GITHUB LINK