Group 23

Karl Meakin & Ben Maxwell

Our approach

Communicating with the engine

We used a parser generator to parse messages recieved from the engine. This was faster to setup and more robust than manually splitting and searching the string.

Selecting moves

Once a message is recieved from the game engine, update the agent's internal model of the state of the board, and consider which of the available moves to select next

We used Alpha-Beta search guided by heuristics:

1. Difference between player's score and opponent's score
2. Difference between number of stones player can capture and number of stones opponent can capture
3. Difference between player's chaining potential (moves that allow the player to have a second turn) and opponent's chaining potential
4. Difference in seeds on each side (avoid starvation)

These heuristics are used to evaluate the board at the depth limit, and to decide the order to search child nodes.

Unique aspects

Performance

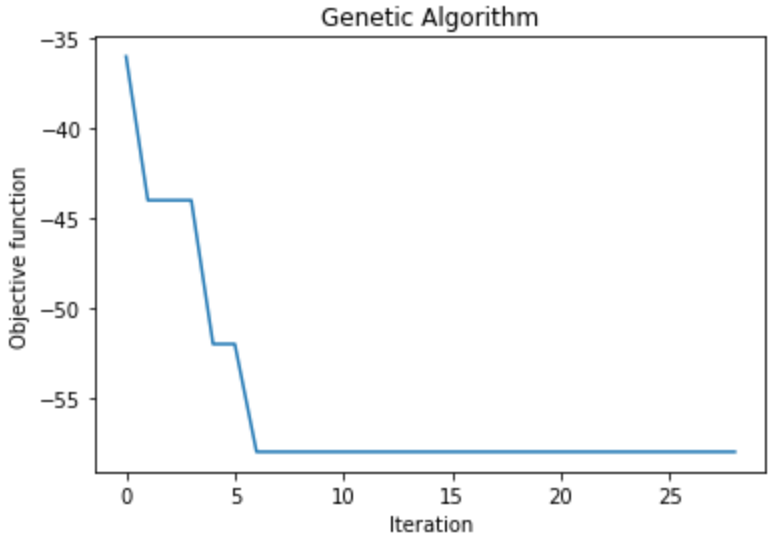
We chose to implement our bot in Rust. Rust is a systems programming language with performance competitve with C++ but with compile-time memory safety.

Because the size of the board is fixed at 7 pits per player, we can operate on fixed sized arrays on the stack, and avoid any dynamic memory allocation.

These factors allow us to acheive a higher performance, and explore the game tree to a deeper depth in the same amount of time, than the same algorithm implemented in Java or Python

Weights finding

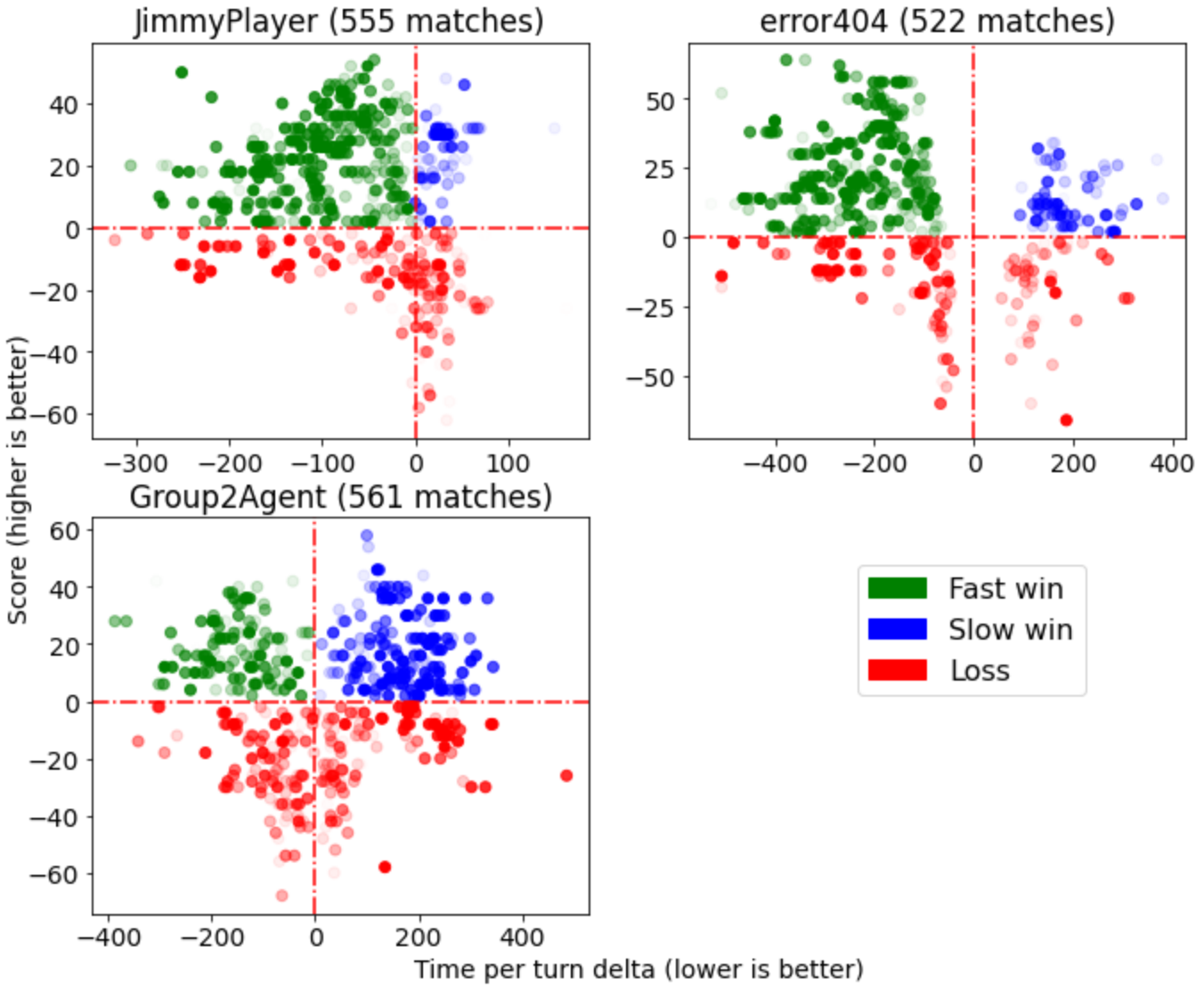
We used a genetic algorithm library to find weights for our heuristic to maximise win margin when playing against the provided test bots.




```
The best solution found:
[REDACTED]

Objective function:
-58.0
```

Performance



Performance