bound

Group 75
António Oliveira Santos – 202008004
Pedro Alexandre Ferreira e Silva – 202004985

The Game

Bound is a two-player strategy game where each player takes turns to move one of their 4 stones along an edge of the pentacle-shaped board to an empty space. The first player to get one of their opponent's stones surrounded on all three sides wins.

An anti stalemate rule is also enforced, where players can not repeat the sequence of the last X moves, X being a number agreed between the players.

References :

https://github.com/hoangchunghien/ai-isolation-game

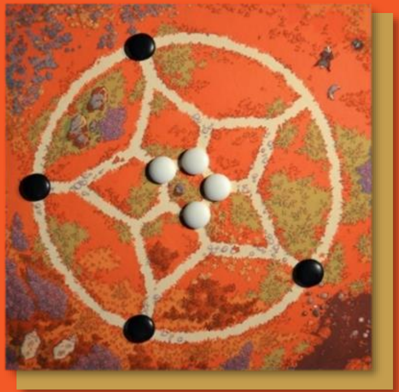
https://www.cs.lmu.edu/~ray/notes/asearch/

https://www.kickstarter.com/projects/turncoatgames/bound-a-print-at-home-abstract-strategy-game

https://en.wikipedia.org/wiki/Monte_Carlo_tree_search

Course Unit Slides

As a search problem...

State Representation	Board/Graph
Initial State	
Objective Test	If one of the pieces has no moves the opponent of the piece's owner is the winner
Problem Space	3^(n*4), n being the number of spaces on the board

As a search problem...

Operators

Move Piece

Precondition	Effect	Cost
Player´s piece Target is empty Not stalemate	Target is occupied by piece Previous place is empty	N/A

Evaluation functions

Evaluation Function

1. Σ available player movements – Σ available opponent movements

2. $\prod$ available player movements - $\prod$ available opponent movements

3. Σ of evaluation function 1 and 2

4. ($\prod$ available player movements - $\prod$ available opponent movements) + (# player pieces in middle)

From these, the one that was the most successful and followed a human like strategy was number 4.

It controls the middle of the board in order to limit the opponent´s pieces and can quickly escape incoming attacks!

Minimax

Find the best move for both players consecutively until a defined depth or an ending state is reached.

To achieve this alternate between maximizing and minimizing, depending on the player to make the move in each turn, that is, each player will make their very best move in a given, creating a more favourable position for them whilst disallowing the enemy to do so for themselves.

Alpha-beta cuts

In order to achieve a better running time for this algorithm we can cut certain moves from our move tree.

That is to say, if we have found a more stringent move, since each player is expected to play their very best move, we do not need to check every single one of the following, as the best one has already been found.

Minimax

Pseudo-code

```
function Minimax(state, maximizing, alpha beta, depth):

 If final_state or depth == 0:
  return evaluation(state)
 If maximizing:
  max_eval = +infinity
  For each available move:
  evaluation = minimax(new_state_after_move, minimizing, alpha, beta, depth-1)
   alpha = max(alpha, evaluation)
   If beta <= alpha:
    break
   return max_eval
 Else:
   min_eval = -infinity
  For each available move:
  evaluation = minimax(new_state_after_move, maximizing, alpha, beta , depth-1)
   alpha = min(alpha, evaluation)
   If beta <= alpha:
    break
   return min_eval
```

An implementation of negamax, a very similar algorithm where we evaluate one player's socre to be inverse to their opponent's, the same results are obtained with a simpler implementation, was also experimented with.

Monte Carlo Tree Search

Even if I don´t know all of the rules of the game, by imagining enough games I´m able come up with a good evaluation of a position!

In order to do so, we must check the possible moves of a given position, select a promising one and then see where that game will lead us, either a win or a loss.

Upper Confidence Bound

How do we select a promising move?

We use some sort of evaluation or policy that helps us dictate what a good move must be like. In our implementation, UCB is used. Upper Confidence Bound is an algorithm that takes into account a moves currently perceived value, the ammount of times it has been tested and a confidence constant controlled by us.

Monte Carlo Tree Search

Pseudo-code

```
function MCTS(state, interation_total):

 mcts = MCTS(state)

 While iteration > 0:
  node = mcts.select()
  leaf = mcts.expand(node)
  result = mcts.simulate(leaf)
  mcts.back_propagate(leaf, result)

 best_move = mcts.best_choice()
```

```
function select():

 leaves.map(update_ucb(leaf))
 return leaves.sort(leaf.ucb)[0]
```

```
function expand(node):

 moves = node.get_moves()
 move = moves[random]

 return move
```

```
function simulate(node):

 while not end:
  moves = node.get_moves()
  node = moves[random]
  end = node.is_final()

 if node.winner == player:
  return 1
 else:
  return -1
```

```
function back_propagate(node, result):

 if node == root: return
 node.update_value(result)
 node.update_visits()

 back_propagate(node.parente, result)
```

Results

Black-Red	Random	MCTS	Minimax 2	Minimax 4
Random	51.50%	20.00%	0.00%	N/A
MCTS	80.00%	80.00%	0.00%	0.00%
Minimax 2	100.00%	90.00%	73.33%	0.00%
Minimax 4	N/A	100.00%	100.00%	20.00%

Minimax using evaluation function 4

Winrate - Black vs Red

Red-Black	Random	MCTS	Minimax 2	Minimax 4
Random	51.90%	0.00%	0.00%	N/A
MCTS	75.00%	60.00%	0.00%	100.00%
Minimax 2	100.00%	90.00%	26.67%	10.00%
Minimax 4	N/A	100.00%	100.00%	60.00%

Winrate – Red vs Black

Minimax 4 vs Minimax 4 shows that a well structured strategy benefits from playing with the Red pieces, which are on the outside.
Even more so, Minimax 2 vs Minimax 2 shows us that this approach greatly benefits from playing the Red pieces.

Results

Black-Red	Minimax 2 (4)
Minimax 2 (1)	50%

Winrate - Black vs Red

Red-Black	Minimax 2 (4)
Minimax 2 (1)	46.67%

Winrate - Red vs Black

This result is not entirely expected, seeing as evaluation function 4 adapts a more human-like strategy where, as previously shown, playing with the Red pieces yields a significant advantage. It would be expected for the first instance to be more in favour of evaluation 4 and not even, and the second one more similar to an even result.

This could be due to the limited sample space, 30 runs each.

Conclusions

All of the goals that were set by the project's description were met and we are satisfied with the developed project. We have built an accurate recreation of Bound and implemented alongside it several different difficulty bots that explore different Artificial Intelligence algorithms.

In hindsight, we would have spent more time in the beginning structuring our approach more closely in order to achieve better running times.