Isolation

@author Pylypenko Maksym

Important

Player class is **A2Q1MyPlayer.java**
Approaches used: Minimax + Alpha-Beta pruning + Heuristic.

State Representation

Each state has an array of Pathways, each pathway has:

1. Player ID.
   a. If character - ascii value of a latter
   b. If number - ascii value + 48
2. Current position of a player.
   a. Has row and col.
3. Position of visited cells by the same player.
   a. Includes current position.

Heuristic Strategy

Limitations:

1. I'm not looking for a good end game yet, but rather a good debut developement.
2. Minimax is not efficient versus multiple players...
   a. detect the first alive enemy and ignore other players
   b. choose the most dangerous enemy and play as it were 1 vs 1.
   c. use very low depth…

   I used option "a", since it simple and good enough to compare heuristic to random players

.

Representation

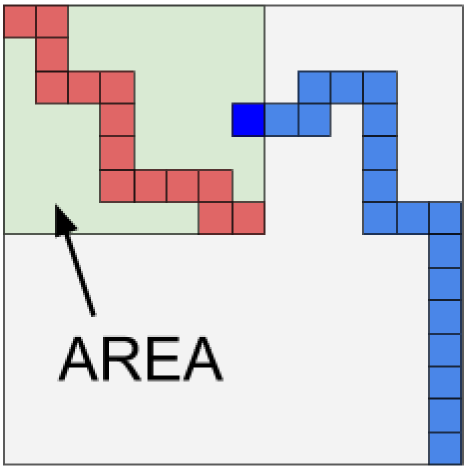
`Class State: double evaluate(int me)`

Implementation: calculating an area of a **captured square** and penalising enemy invasions.

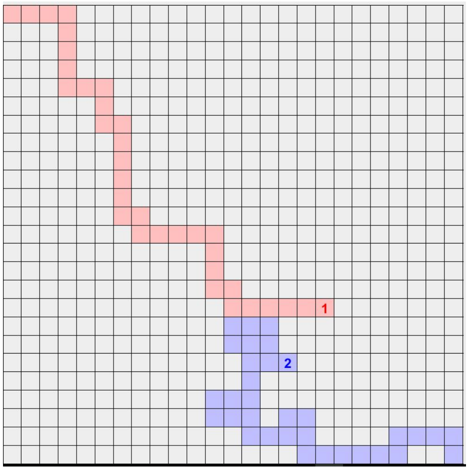
1. Find maximum and minimum rows.
2. Find maximum and minimum cols.
3. Find the Area `double score = ( maxRow - minRow ) * ( maxCol-minCol );`
4. For each enemy square in our territory `score-=5;`

Summary: definitely better than random, but requires a separate endgame algorithm that effectively fills the rest of the board. Generally ok versus defencive players, but will most likely lose to aggressive players.

Representation

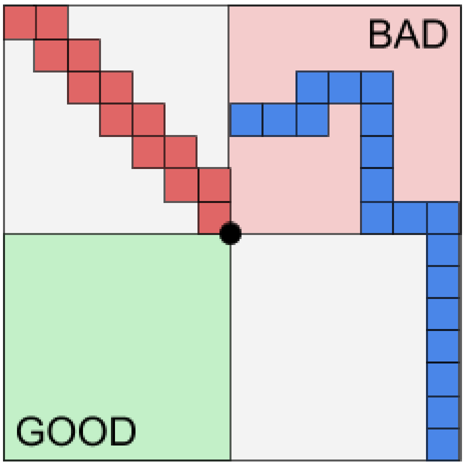


AI, Random



Alternative strategies…

not done yet!

1. Center control...	
Representation  	AI, Random

2. Capturing (rows*cols / number of players +1) is a guaranteed victory. Is there a way to abuse this to make a good heuristics ? If we look at these random 1 vs 4 games… It becomes clear that trying to play an “optimal” game puts us in a vulnerable position. Therefore, this strategy is awful for an early-mid game. However, it might be useful for a late stage...