Overview

We tend to think the AI we have created is a model-based reflex agent.

After simulating several different strategies on a virtual board, we found a very special yet simple pure defensive strategy that almost only cares about itself. So we just find all possible movements we can perform. Among all these child nodes, we give each of them a score by using our weighting functions. Then we perform the step which has the highest score.

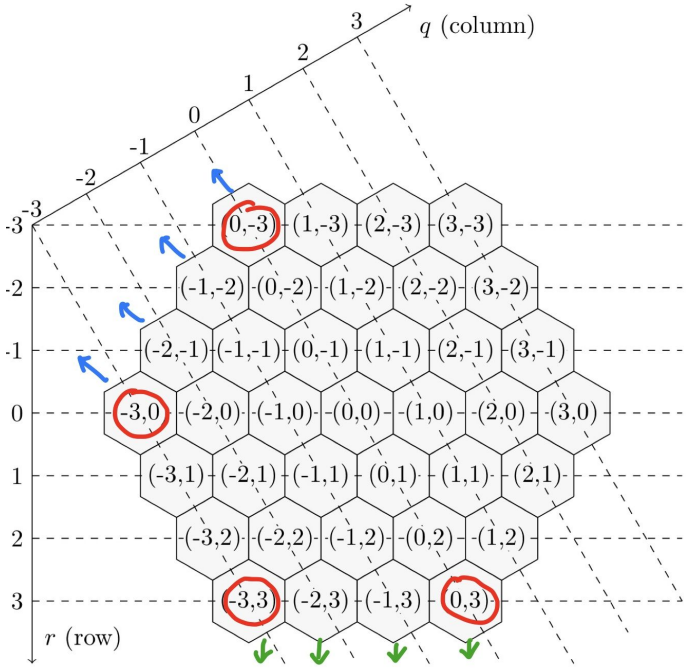
Due to the uniqueness of the strategy we are using, it wouldn't be necessary to use algorithms like Minimax or MaxN to predict general outcomes.

Strategy

The strategy we used is called "border guardian", because it has the following core mindsets:

1. Hold the Door

   During our experiments, we found that our spawning positions are closer to enemy destinations than any of the enemy pieces in general. Two of them wouldn't even need to move in order to occupy an enemy destination!
   Knowing that enemy pieces must stay on the destination border for at least a term before exiting, it would be wise to just ambush at enemy destinations and wait for the prey to come to us.
   For example, in picture 1, we occupied all four corner positions of enemy borders({(0, -3), (3, -3), (3, 0), (0, 3)}) with our pieces, and leave only two positions on each border for green pieces and blue pieces to go to. If they are smart enough, they would stay away from their destinations forever. If not, they would provide us with more red pieces.



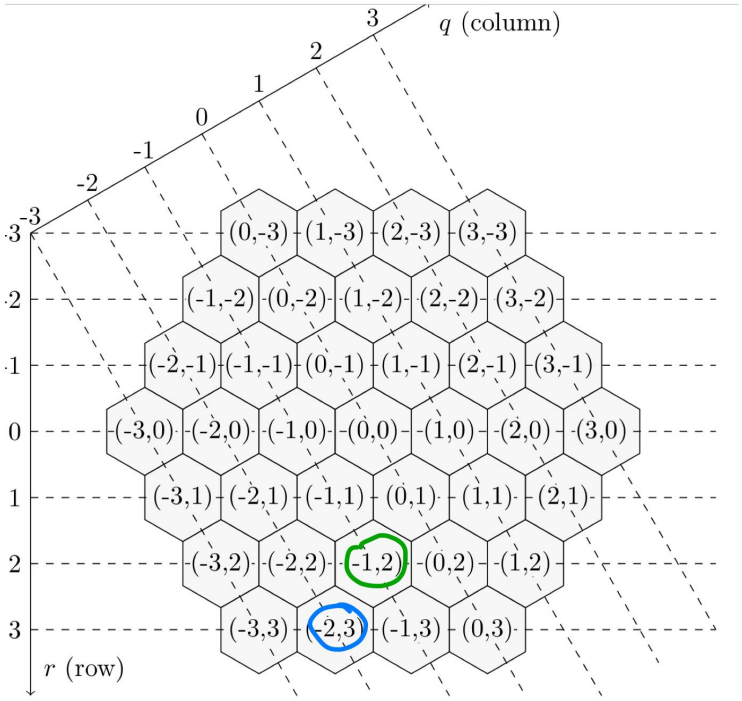
Picture 1

However, we'd have to rush to enemy borders to set up this giant trap, and sometimes it's impossible to take those positions quick and safely, that's why we also used a mindset of edge-walking and corner-taking.

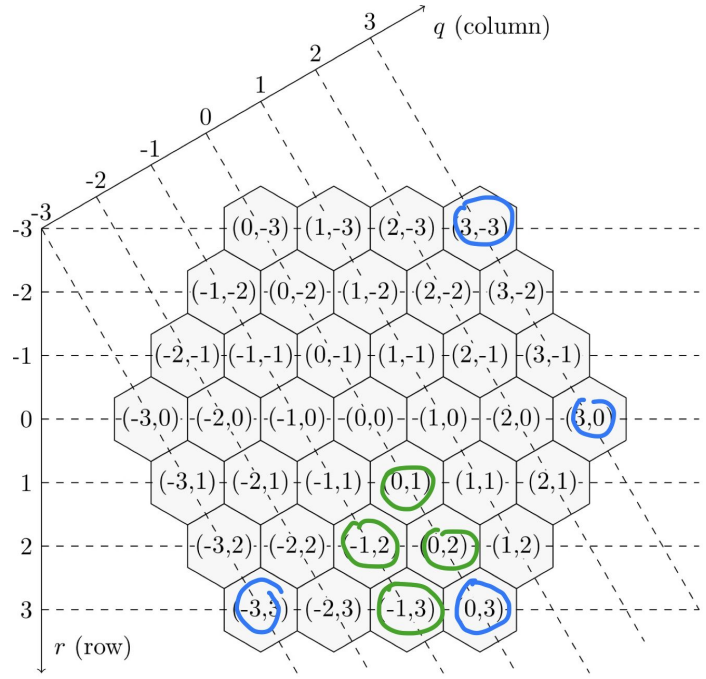
2. Edge-walking & Corner-taking

   We found that when a piece is walking on the border, it is less likely to be converted, because it has its back covered with the void outside of the board. This advantage becomes even more obvious when a piece is at corner. A piece at corner is basically unconquerable.

   As you can see in both picture 2 and 3, blue pieces can take green pieces, but the greens has nothing they can do to eat the blues.



Picture 2



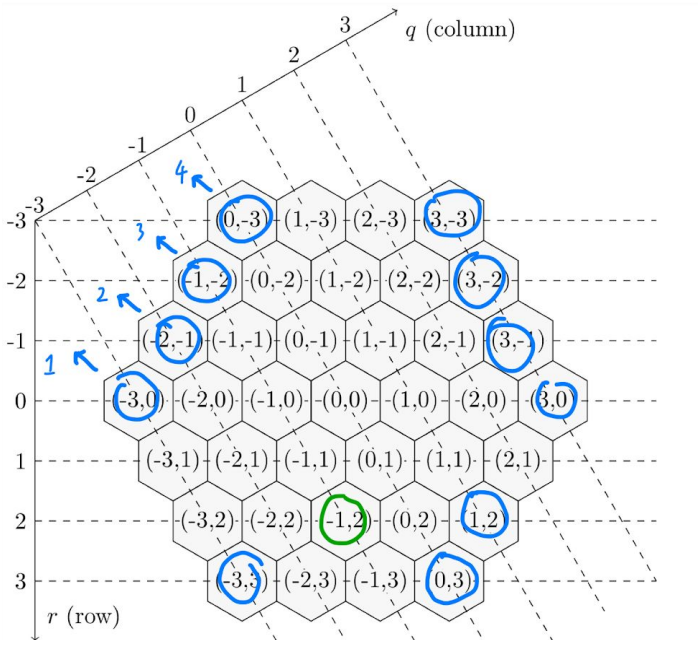
Picture 3

After trying out some aggressive board-center-walking strategies, we found walking on the edge and taking corners if possible is a must if we want to win.

3. Endgame Overlord

   If enough pieces fall into our traps throughout the game, and the piece difference balance is completely broken, we would move to our destination, because there would be no way for enemies to win or to stop us with the amount of pieces they have.

   We wouldn't get our pieces to exit once they get there, but to occupy all four of the positions first, then exit one by one, as shown in picture 4. In this way, there will be no comeback for enemies.



Picture 4

Evaluation Functions

To fully implement "the border guardian" in this game, we made 7 evaluation functions, i.e. eval(state) = w1*f1 + w2*f2 + w3*f3 + w4*f4 + w5*f5 + w6*f6 + w7*f7

[f1: number of guardians in position]

w1(strategic motivations) = 10

Returns how many pieces of ours would occupy the guard posts for this state.

[f2: number of edge pieces]

w2(strategic motivations) = 3

Returns how many pieces of ours are on edge.

Post guards also gains this weight.

[f3: total distance to all guard posts]

w3(strategic motivations) = -1

Traverse all posts, match them with nearest pieces, and record distances.

Returns the sum of all four shortest set of distances.

Hex distance function:

If |q1 - q2| < |r1 - r2|:

distance = |q1 - q2| + |r1 - r2 + q1 - q2|

If |q1 - q2| > |r1 - r2|:

distance = |r1 - r2| + |q1 - q2 + r1 - r2|

If |q1 - q2| = |r1 - r2|

distance = |q1 - q2| + |r1 - r2 + q1 - q2| = |r1 - r2| + |q1 - q2 + r1 - r2|

See picture 5 for example.

hex - distance function

① $|q_1 - q_2| < |r_1 - r_2|$

distance = $|q_1 - q_2| + |r_1 - r_2 + q_1 - q_2|$

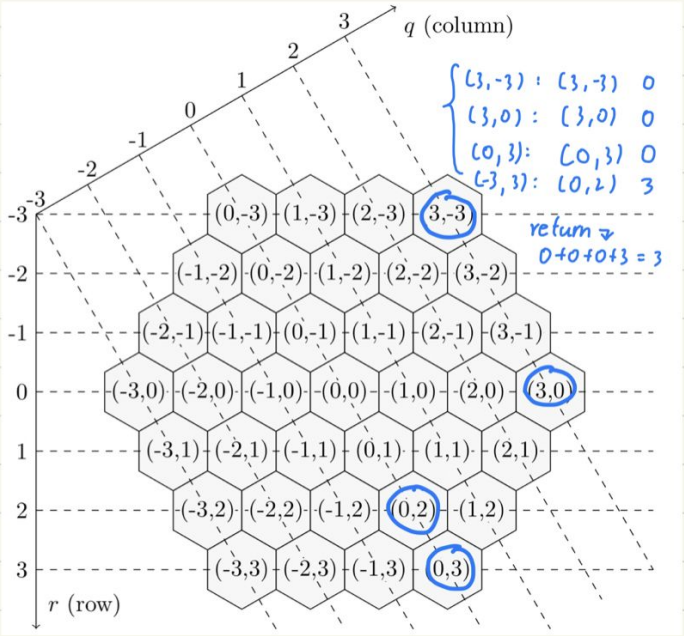
② $|r_1 - r_2| < |q_1 - q_2|$

distance = $|r_1 - r_2| + |q_1 - q_2 + r_1 - r_2|$

③ $|q_1 - q_2| = |r_1 - r_2|$

either is ok

e.g. distance from (0, 2) to (-3, 3)



$|q_1 - q_2| = 3$

$|r_1 - r_2| = 1$

⇓

use function ②

distance = $|r_1 - r_2| + |q_1 - q_2 + r_1 - r_2|$

$= 1 + |0 - (-3) + 2 - 3|$

$= 1 + 2$

$= 3$

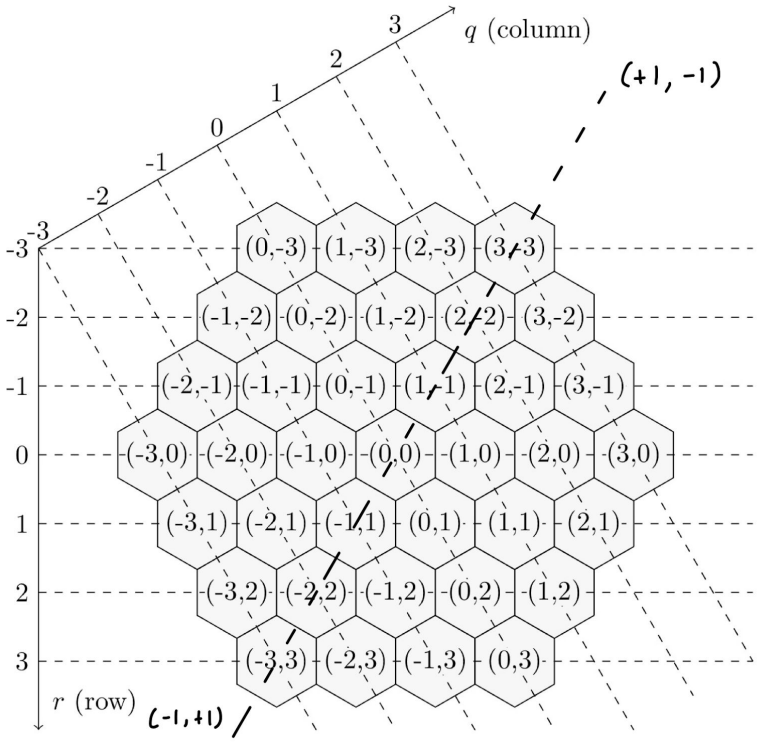
so distance from (0, 2) to (-3, 3) is 3

Picture 5

Why we use this function:

Due to the very nature of hex plane, we can't just use the manhattan distance function to calculate the distance between positions.

In Cartesian Plane, it takes two steps from (0, 0) to (-1, 1) because one is at diagonal position of another. In hex plane, however, it takes only one step.



But if we interpret the line through (1,-1) and (-1,1) as another dimension of the cartesian plane, and tweak the manhattan distance a little, we would be able to calculate the distance again.

[f4: how many pieces will be eaten]
w4(strategic motivations) = -100
Returns how many pieces risk being eaten for this state.

[f5: piece difference between us and enemies]
w5(strategic motivations) = 20
Returns 2 * our pieces - all enemy pieces.

[f6: distances to our destination]
w6(strategic motivations) = 5 or 0

If not endgame, ignore this weight function, returns 0.
If endgame, find the four pieces that are closest to the destinations, returns the sum of distances of them.

[f7: prey in range]
w7 = 300
When a guardian has an enemy piece nearby, we encourage it to stay in post and do nothing.