香港電腦奧林匹克競賽
Hong Kong Olympiad in Informatics

AI Competition

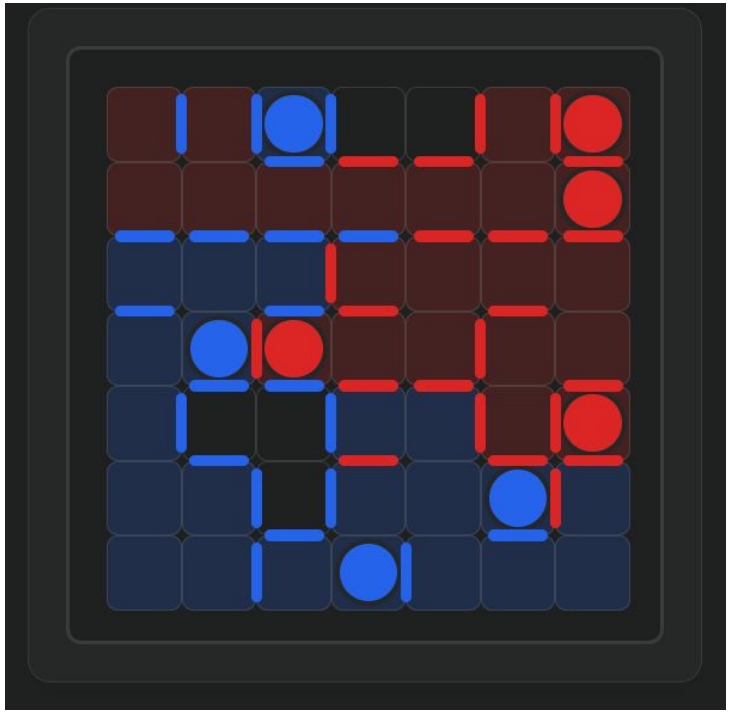
Sunny Cheung {__declspec}

2025-06-30

AI Competition

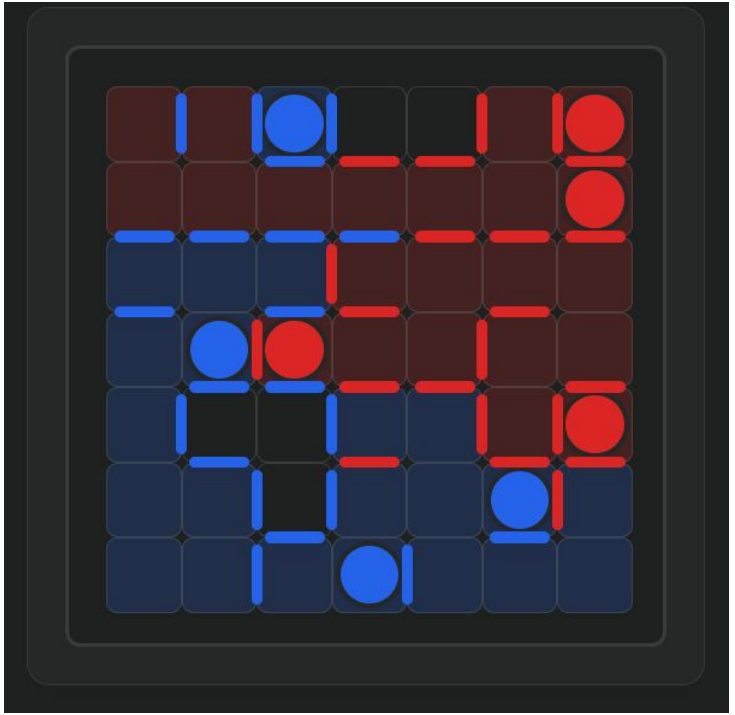
- Play as a team
- Write codes to play the game **Wall Go**
- Have fun playing against other teams

- Prize according to your team's rank.



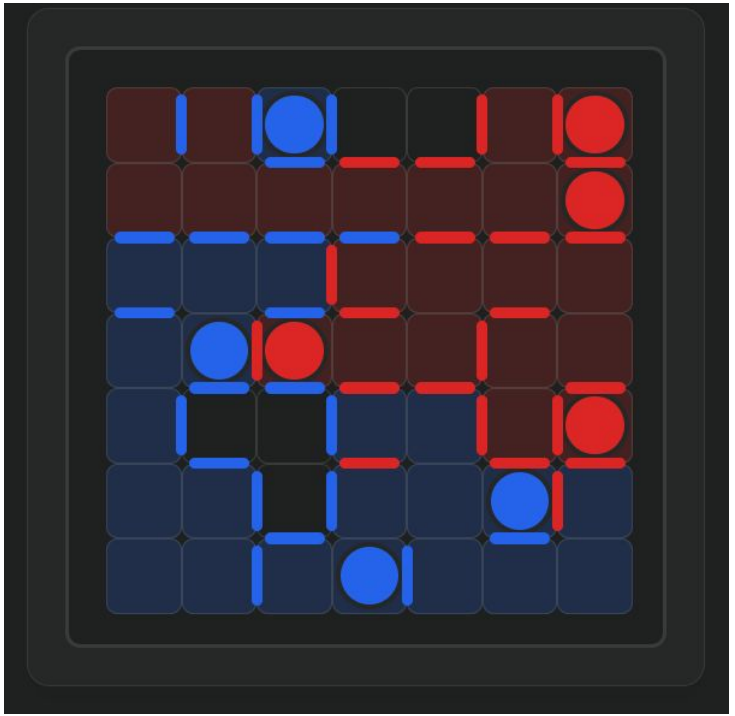
Wall Go

- The game is played in a 7x7 grid
- Build walls to create your own ‘territory’
- Objective: maximise your territory



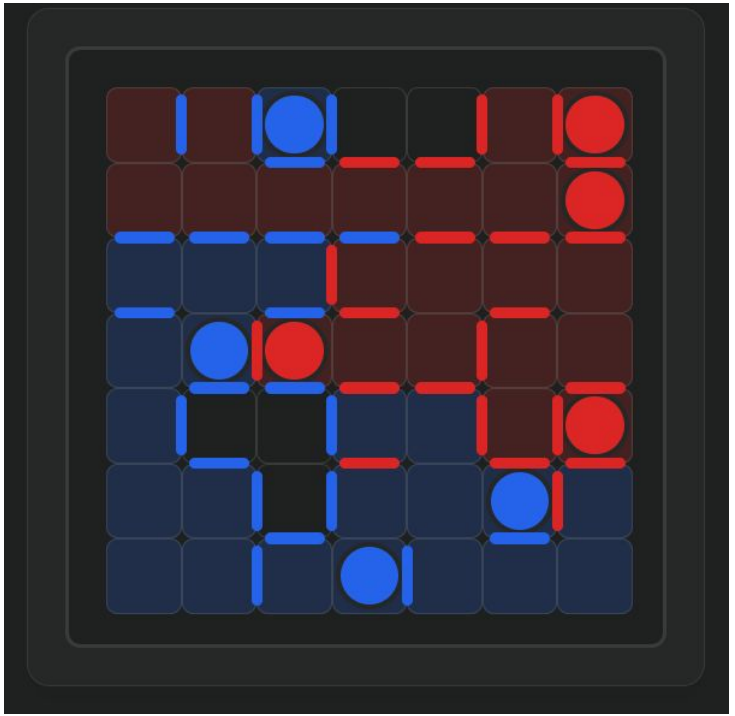
Wall Go

- Initially: place the pieces on the board
- Then turn-based: in each turn, the player moves a piece and place a wall
- All by a program (in elimination stage)



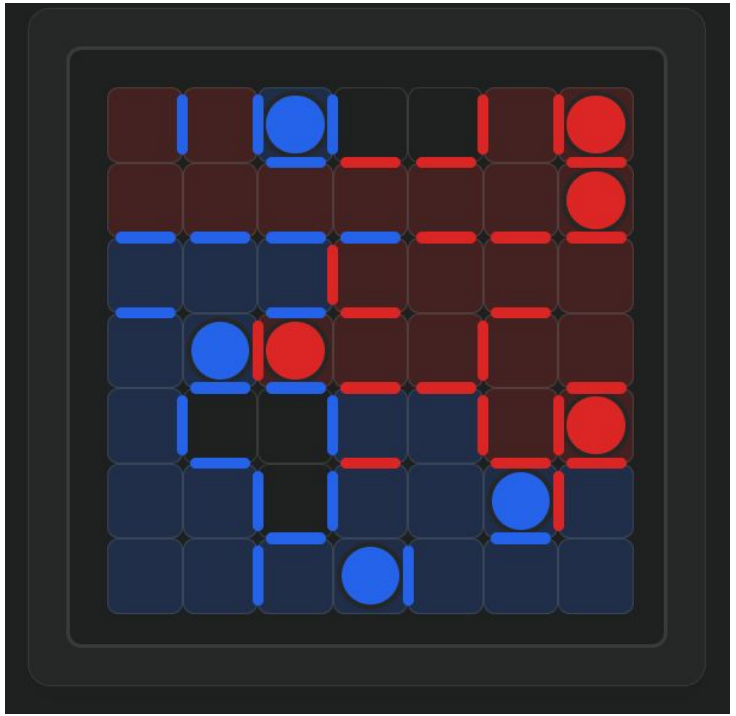
Wall Go - Piece Placement

- Two player game: each player has 4 pieces
  - Placement order (R=red, B=blue): RBBRRBBR



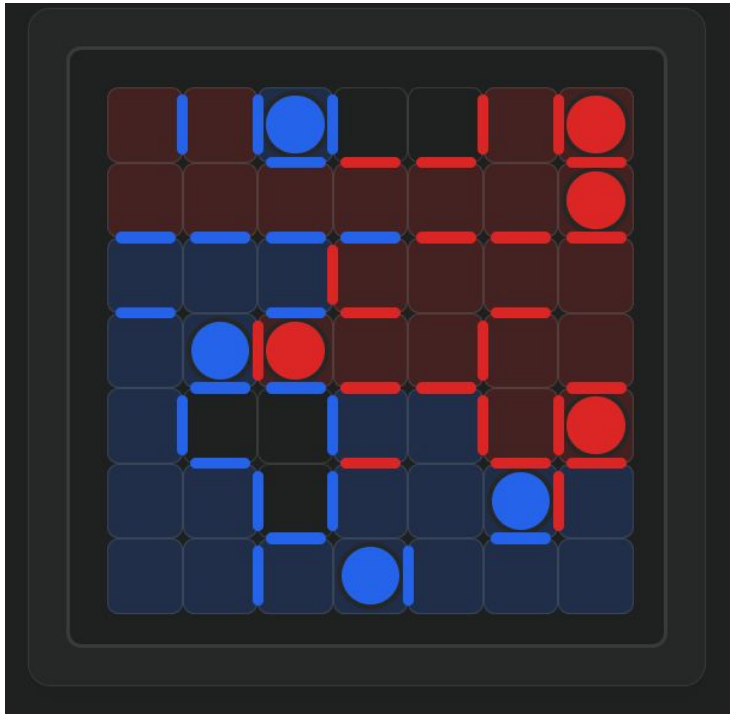
Wall Go - Piece Movement

- In each turn, select one piece
- Move the chosen piece 0, 1, 2 steps
  - Each direction: up/down/left/right
  - Cannot pass through existing walls or other pieces

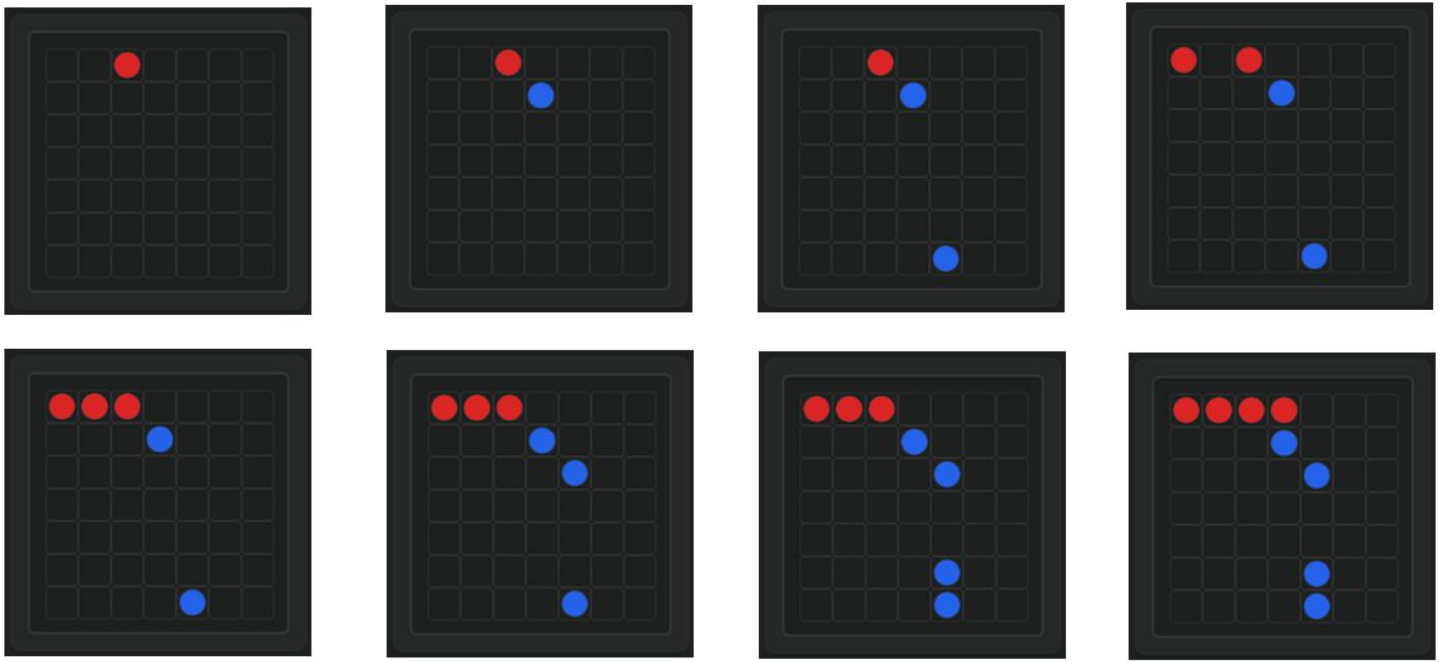


Wall Go - Wall Placement

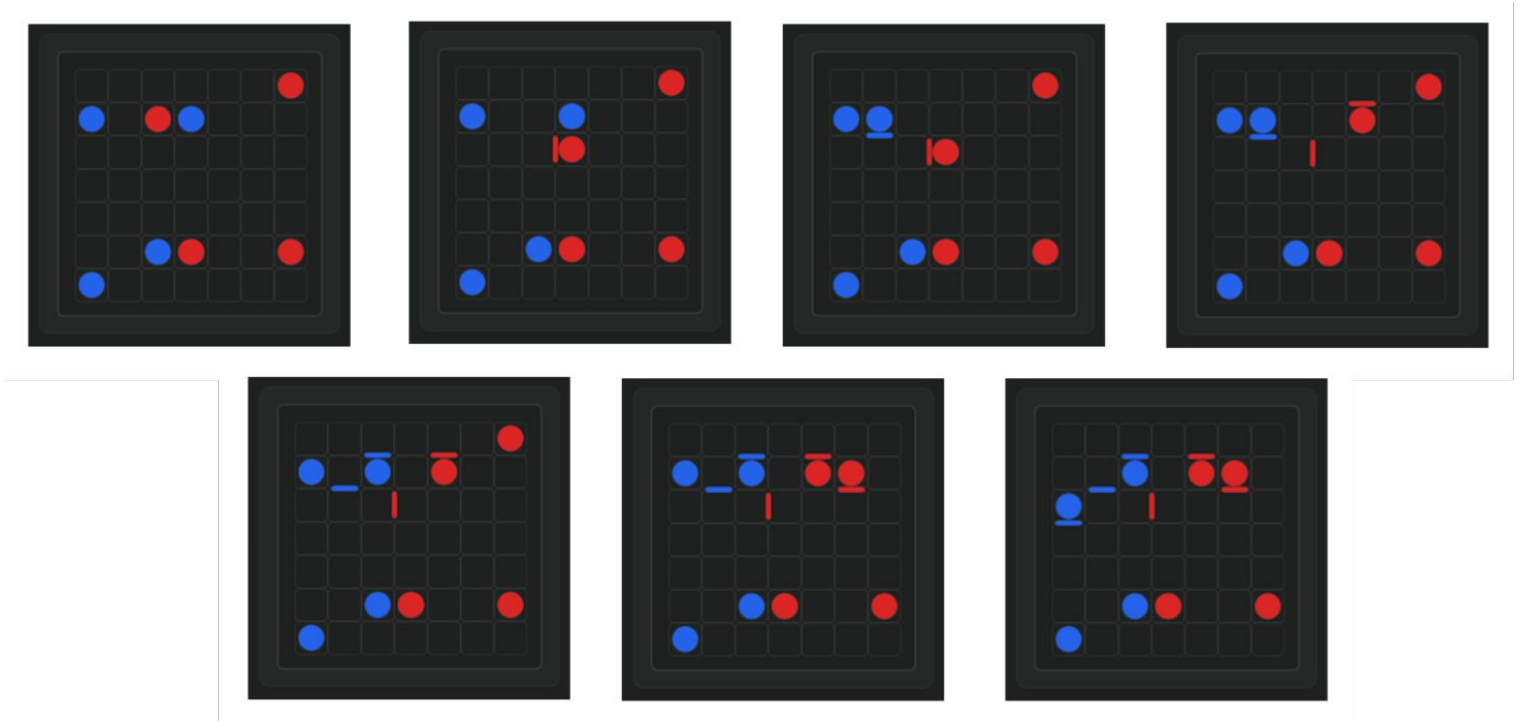
- In each turn, after moving a piece, **must** place a wall
  - Around the cell where the piece is after movement
  - The position must not have an existing wall



Example: placement

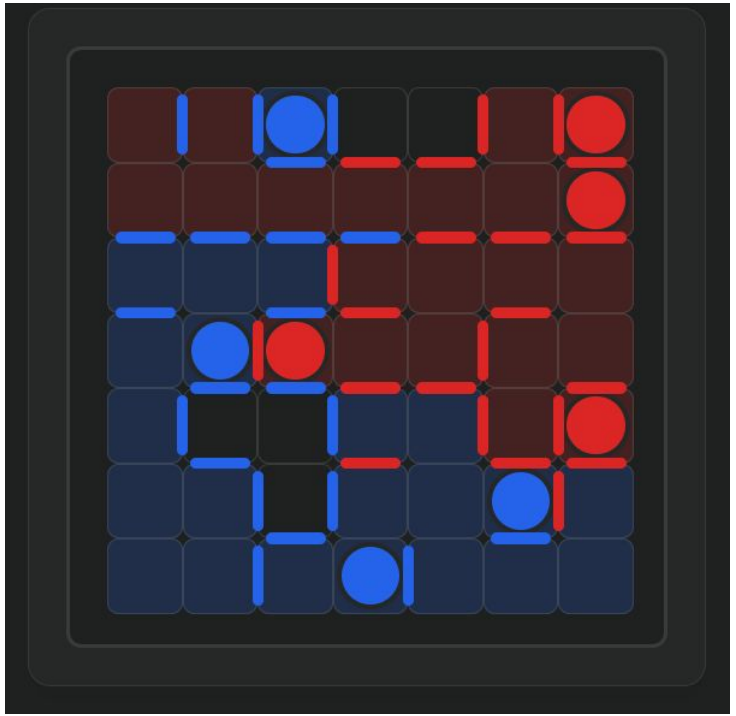


Example: move



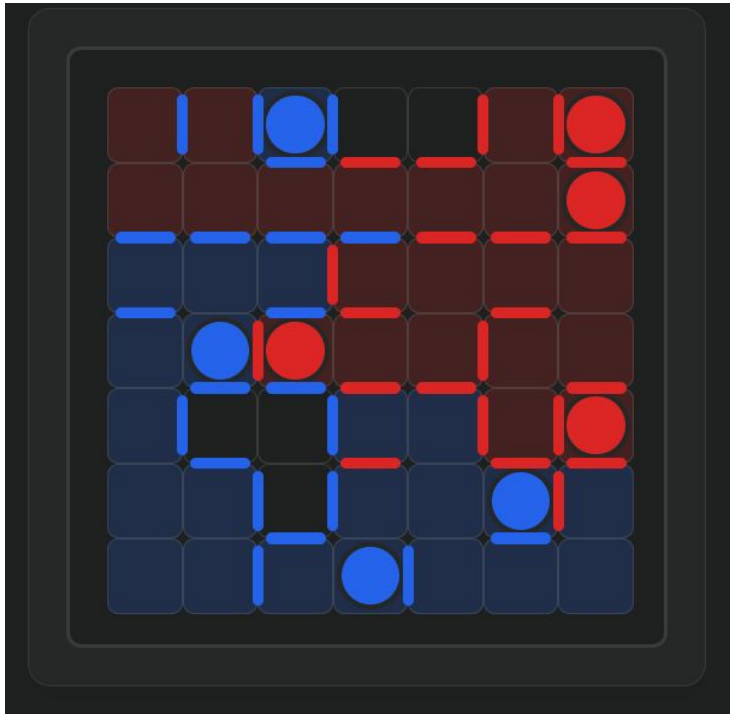
Wall Go - Game end

- Ending condition: when no two pieces of different colours reside in the same connected component
- Territory is calculated by number of cells in enclosed by walls and containing only that program's pieces
    - In the example game on the right, it is 22:22



Wall Go - Small details

- Walls can’t be placed on borders; borders count as permanent walls
- Once a wall is placed, it cannot be removed
- Tie breaker:
  - Single largest territory
  - If still tie, then the one who ends the game loses
- Each player **must** place a wall even if they don’t move the chosen piece



Tournament Format - Heat

- Contestants are divided into 12 teams, 3 people each
- In Heat, 3 teams form a group (4 groups in total)
- Each group play one game physically to decide group ranking
  - We'll play on https://schaoss.github.io/wall-go/
- Ranking will be decided by the sum of territory your team gained in 2 games

	Red	Blue
Game 1	Team 1, 4, 7, 10	Team 2, 5, 8, 11
Game 2	Team 2, 5, 8, 11	Team 3, 6, 9, 12
Game 3	Team 3, 6, 9, 12	Team 1, 4, 7, 10

Tournament Format - Double Elimination

Total 19 games (each loser of heat starts in loser bracket)

Heat Results

	Red Team	Blue Team	Red Cell	Blue Cell	Winning Team	Red Max Block	Blue Max Block
Group 123	1	2	22	19	1	8	13
	2	3	35	14	2	21	8
	3	1	19	27	1	9	16
Group 456	4	5	32	15	4	15	13
	5	6	34	14	5	10	10
	6	4	20	29	4	10	18
Group 789	7	8	20	29	8	9	29
	8	9	18	31	9	14	22
	9	7	35	8	9	19	3
Group ABC	10	11	21	28	11	9	11
	11	12	20	29	12	9	16
	12	10	21	28	10	10	28

	Team	Total Cell	Wins	Internal Rank
Group 123	1	49	2	2
	2	54	1	1
	3	33	0	3
Group 456	4	61	2	1
	5	49	1	2
	6	34	0	3
Group 789	7	28	0	3
	8	47	1	2
	9	66	2	1
Group ABC	10	49	1	2
	11	48	1	3
	12	50	1	1


1 A1 Team 2
8 D2 Team 10
4 D1 Team 12
5 A2 Team 1
2 B1 Team 4
7 C2 Team 8
3 C1 Team 9
6 B2 Team 5
11
12
Team 2
Team 12
Team 8
Team 5
16
Team 2
Team 8
18
Winner: Team 2
Team 2
Team 5
19
Loser of 18 (if necessary)
8 Team 9
9 A3 Team 3
7 Team 4
12 D3 Team 11
6 Team 1
10 B3 Team 6
5 Team 10
11 C3 Team 7
10
9
Team 9
Team 11
Team 6
Team 10
13
14
Team 12
Team 11
Team 5
Team 6
15
Team 12
Team 5
17
Team 8
Team 5

Implementation Details

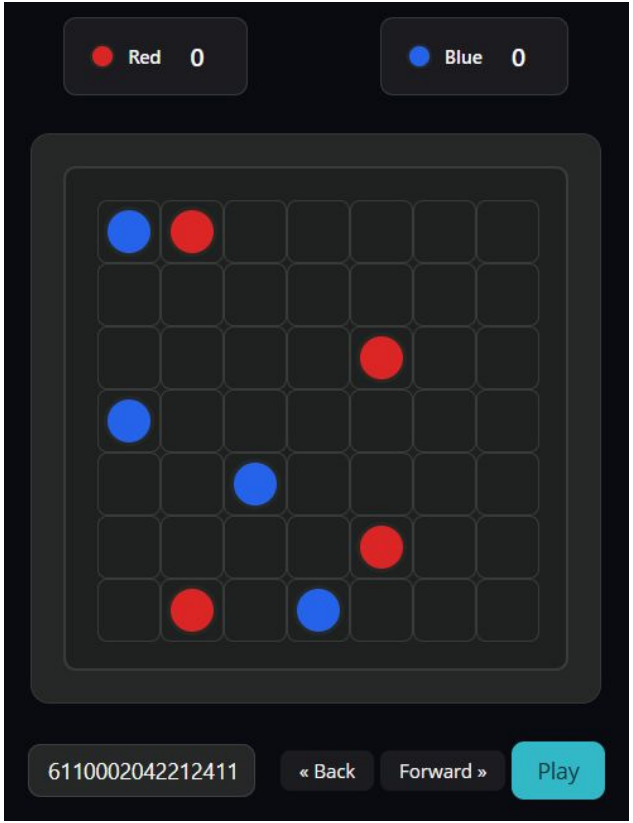
- You should spend the most time developing locally
- Implement `impl.cpp`, where you need to:
  - Inherit and implement the `Player` class
  - Provide two instances for playing the game, one for red, one for blue
- Auxiliary functions are declared at `types.h`
  - There is documentation in the code
- Do NOT manipulate the game state directly; just return the piece/move and the game engine will manipulate the game for you
- Find local compilation instructions in README.md
- Just submit your `impl.cpp` on the judge!
  - Your submission will be run against our bot

Visualisation

At the end of each game, `game.encode()` will be printed e.g.

```
5310522022212311201221220523061 3_2k12k5c20434ko0164bi0r44j40354pk0qi544
1a65a80104251214a201841211g4s41464am04o5hm02g5a214m4300c853802i4380sm5s
m1445q40s25h802m53603g4
```

- This represents the moves history and can be visualised
- Paste the string into the box in the visualiser and you can view the replay of the game
- The visualiser would also be automatically displaying the game with our bot
- Tip: use left/right arrow to go forward/back



Have fun!

香港電腦奧林匹克競賽
Hong Kong Olympiad in Informatics

Wall Go Sharing

Ka Hei Wai {wjx}

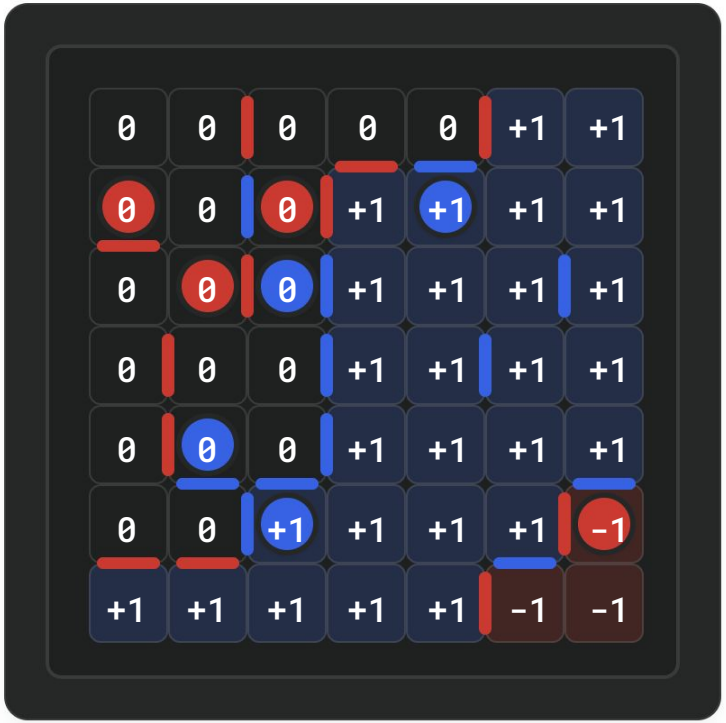
2025-06-30

How to define whether a move is good?

For turn based game, it's common to define a score for the each game state, and check which move go to the state with a highest score.

The simplest way is to directly use the game definition

- For enclosed area with only one type of player, grant score = number of cell.
- +score for your player, -score for opponent



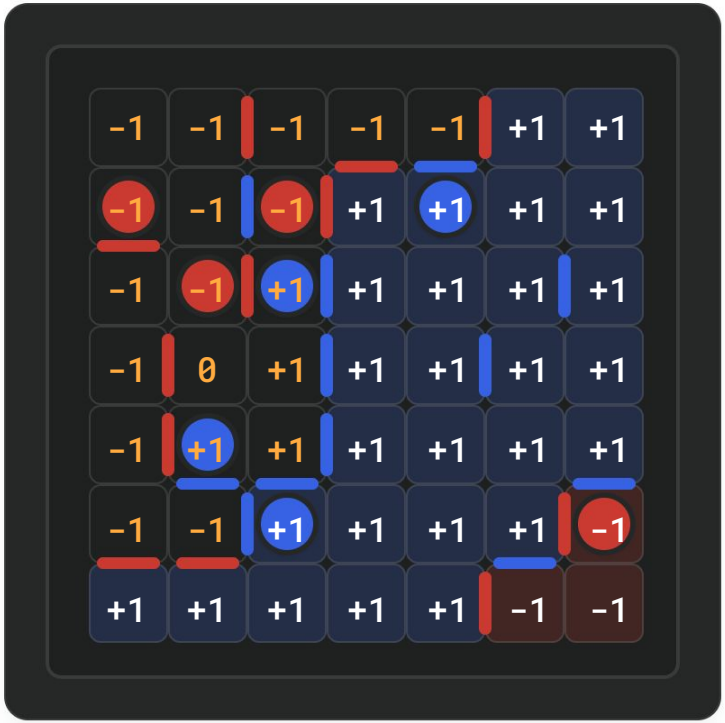
This is a game state of +24

- A still ongoing game, but we can give it a score (how good it is looking for us)

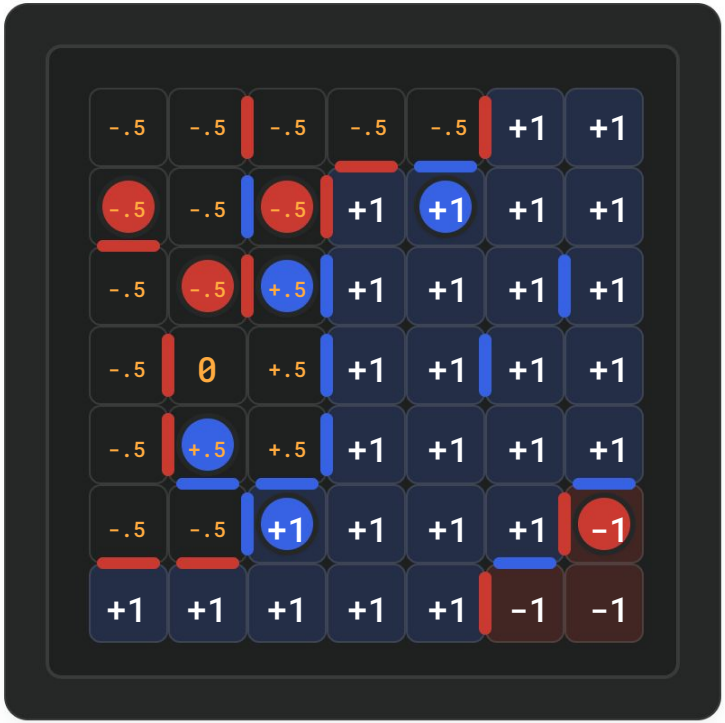
How to define whether a move is good?

- A problem going with this scoring format is that, we have basically no guidance in early stage of the game.
- It is also ignoring that, some cells, although not captured yet, is already very likely to be captured by you / opponent.

- We can place a guess on the cells' final ownership, based on current state.
  - e.g. assign the cell to the closest pieces in bfs distance, using floodfill



- This is a more accurate guess than the previous, taking in more context to consider.



- It is not convincing to give equal weighting to confirmed capture and guessed capture. We can give it another value (hyper-parameter to be tuned).

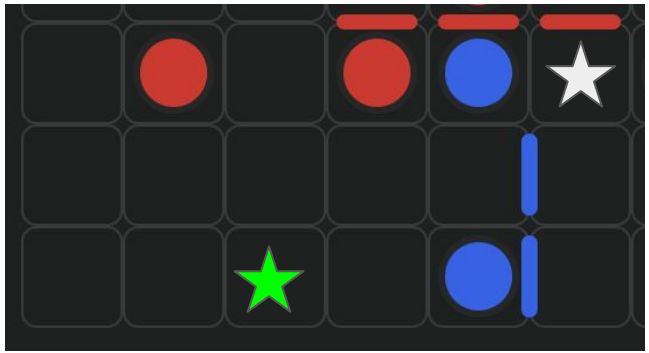
How to define whether a move is good?

- What is the score actually representing?
- How likely is the cell going to be yours. (from -1 to +1).

- How likely is the cell marked with White Star belong to Blue?
- Comparing to the cell marked with Green Star belong to Blue?

How to define whether a move is good?

- How about when there are multiple players?



- The score function can be something simple defined on the distance to red / blue:
  - **score(dist_to_nearest_red, dist_to_nearest_blue)**

How to define whether a move is good?

- **score(dist_red, dist_blue)**
- Suppose you are red.
- We want to give **more score** to smaller dist_to_nearest_red cell
- Give **less score** to smaller dist_to_nearest_blue cell

Something that works:

- Ethen: `(dist_blue - dist_red) / (max(dist_blue, dist_red)) * 0.9`
- Wjx: `1 - (dist_red) / (dist_red  + dist_blue) * 2`

- Just come up with many ideas and validate them by battling with others.

How to define whether a move is good?

- Good thing to do is, make all of the heuristics weighting easily configurable.
- Tune with your feeling, observe the behaviour on the game board.
- Decide how to adjust weightings to encourage / discourage behaviour.
- Human gradient descent

```
const float GURANTEED_SELF_WIN_SCORE = 0.9;
const float GURANTEED_OPPONENT_WIN_SCORE = -0.85;

const float SEMI_GURANTEED_SELF_WIN_SCORE = 0.8;
const float SEMI_GURANTEED_OPPONENT_WIN_SCORE = -0.75;

const int CONSIDER_BOUNDARY = 6;
const float NOT_GUARANTEED_WEIGHTING = 0.90;

const int TOP_K = 5;
```