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

AI Competition

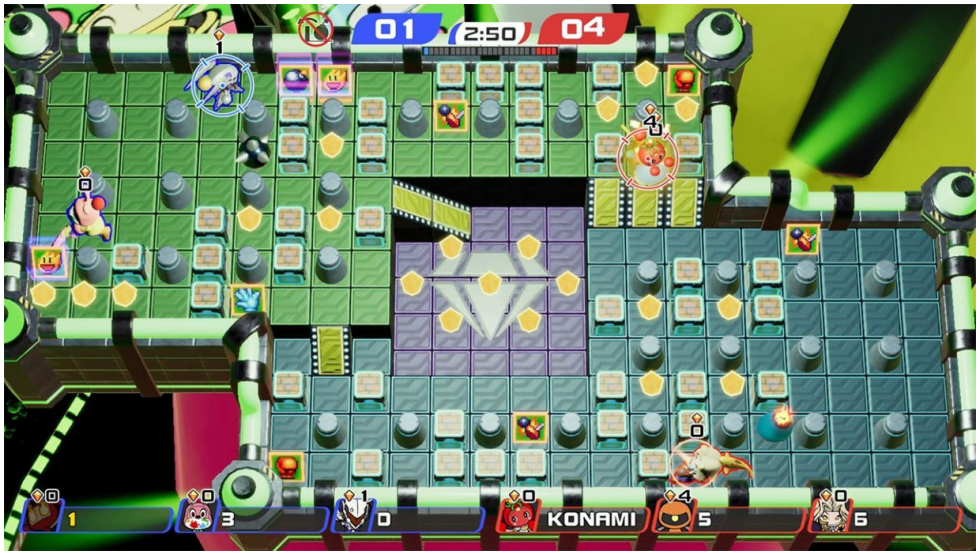
Dickson Wong {hywong1}
2026-06-29

AI Competition

- Play as teams of 2
- You will write code to play a game
- Teams will play against each other with their written code
- Coins will be given based on your team's performance

The Game: Bomberman

- Bomberman is a multiplayer PvP game where players try to eliminate each other
- Players can do so by moving around and placing bombs



Gameplay

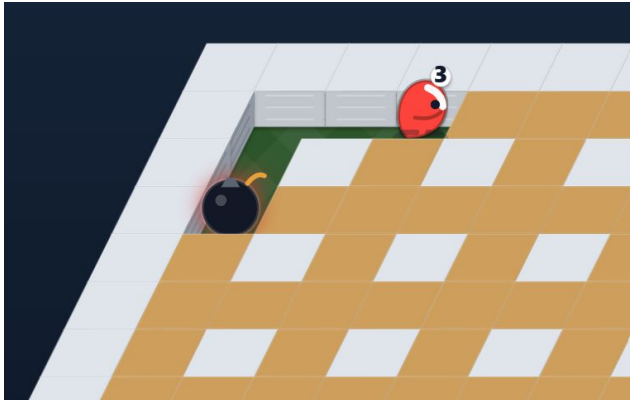
- Players take actions simultaneously at each tick
- The player can choose to do one of the following actions:
  1. Move to one of the orthogonally adjacent cells (up, down, left, right), if that cell is not obstructed
  2. Place down a bomb at the current cell

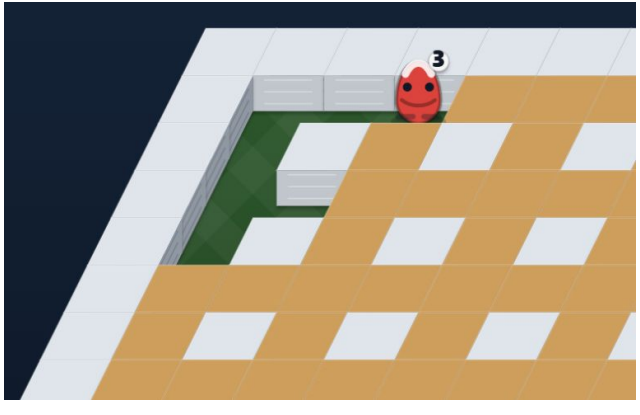
1 tick = 500ms

Your program must return the move in 300ms otherwise the server will take the IDLE action

Bomb Mechanics

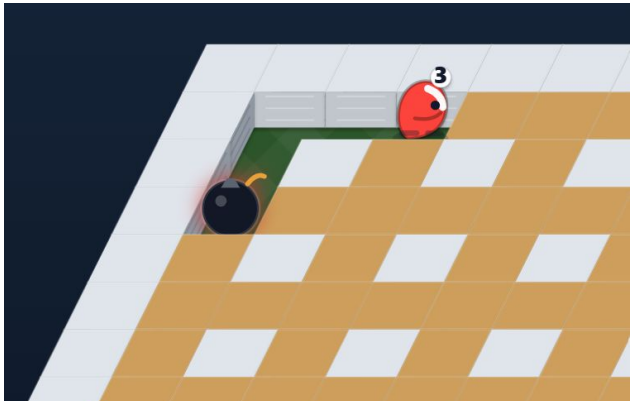
- Initially each player can only have at most 3 active bombs at any moment
- When a bomb is placed down, it will have 5 ticks before it explodes
- When the bomb explodes, it eliminates all players (including the player who placed it!) within 3 cells of the bomb in a cross shape
- The bomb will also destroy the first crate in each direction

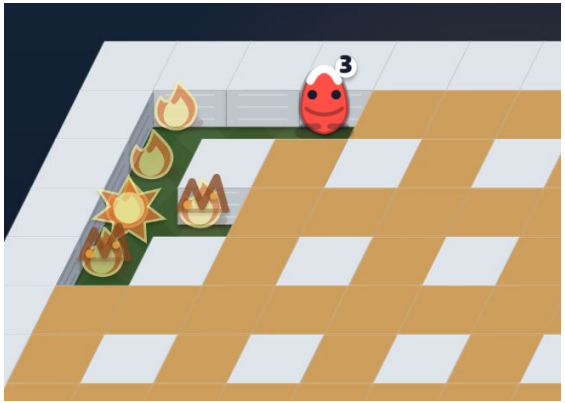


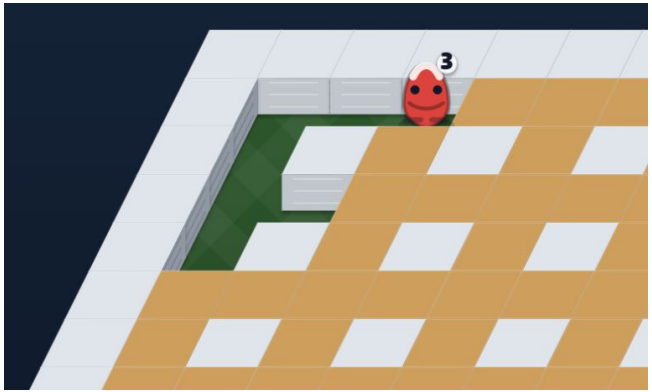


Bomb Mechanics

- Initially each player can only have at most 3 active bombs at any moment
- When a bomb is placed down, it will have 5 ticks before it explodes
- When the bomb explodes, it eliminates all players (including the player who placed it!) within 3 cells of the bomb in a cross shape
- The bomb will also destroy the first crate in each direction

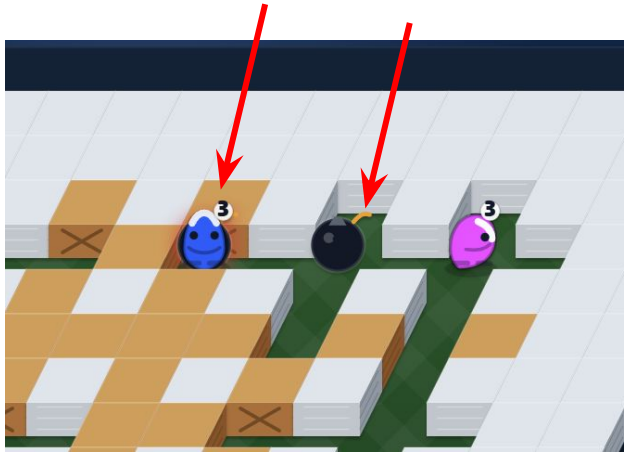


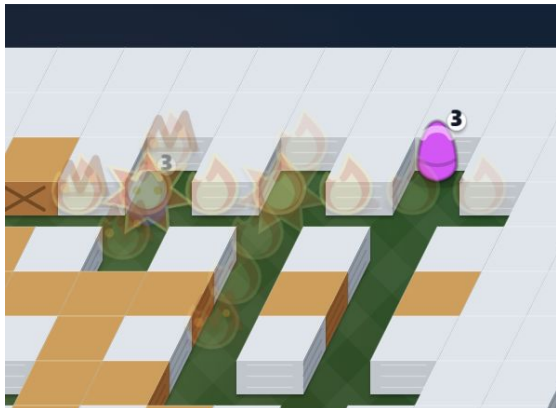


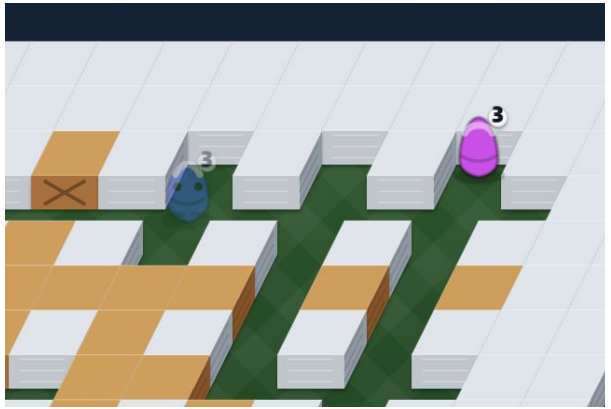


Bomb Mechanics

Explosions chain



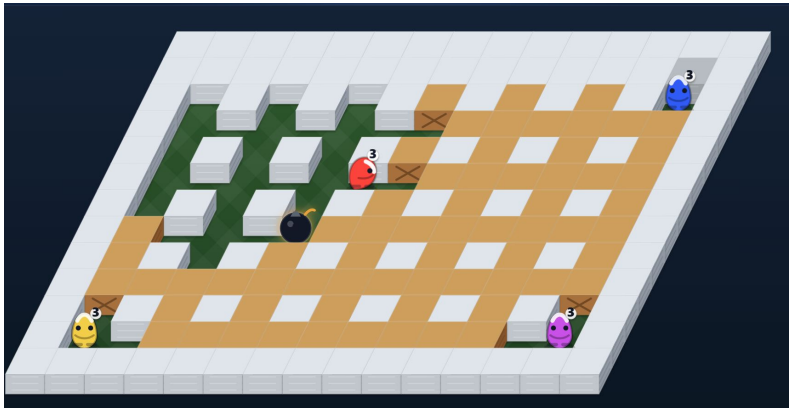




Hurry Up

To reduce the size of the map, after ~1 minute, walls will fall from the sky in a spiral pattern at a speed of 1 cell per tick. It will:

- replace cells with wall (whether or not it is empty or has a crate)
- Remove bombs (if any)
- and kill players standing under it.



Player collision

- Players are processed in ascending player ID.
- Two players targeting the same empty cell: lower ID moves; higher ID stays.
  - Therefore, players cannot directly swap cells.
  - A higher-ID player may enter a cell vacated earlier that tick by a lower-ID player.

`IsMoveValid()` alone cannot predict simultaneous conflicts; `Tick()` resolves them.

Tournament Format - Heat

- During the heat stage, games will be played to determine each team's Elo
- Each game will be played between 4 teams (selected out of 20 teams)
- Teams can submit their strategy code at any time, their last submitted strategy code will be used when they are up for the heat games
- The Elo leaderboard at the end of the heat stage will determine the seeding of teams
- The top 4 teams will get a bye to the group stage in the finals !!

Tournament Format - Group Stage

- The remaining 16 teams will form 4 groups
- Each group will play 1 game, and teams will be scored based on order of elimination in each game
- The top 2 teams will advance to the elimination stage

Tournament Format - Semifinals

- The 12 teams will be grouped into 3 groups of 4
- Each group will play 1 game, and teams will be scored based on order of elimination
- Top team of each group will advance to the final.
- Second team of each group will compete for another game for the final spot in the final

Tournament Format - Finals

- The 4 finalists will play one game to decide the winner

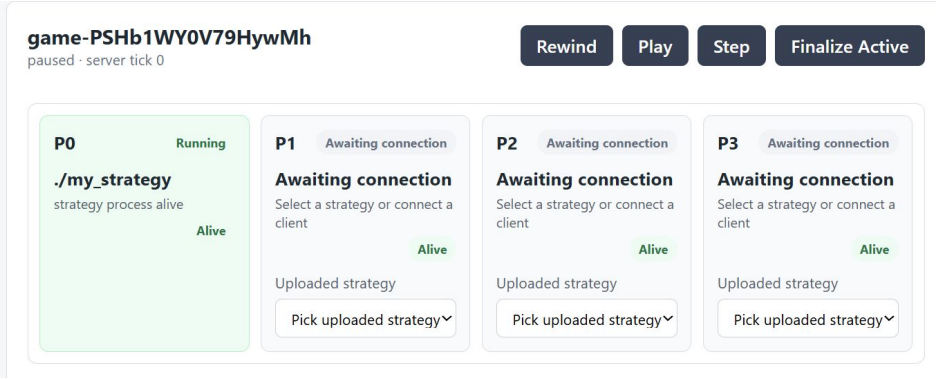
Implementation

- Clone https://github.com/hkoi/aicomp2026
  - https://github.com/hkoi/aicomp2026/archive/refs/heads/main.zip
- Implement `strategy.cpp`, where you should:
  - Implement `GetNextMove` in `strategy.cpp`, which would be called every tick
- It is NOT recommended to directly modify the `GameState` object yourself; the `ApplyUpdates` function will do it for you
- There are helper functions
  - `int FallingWallTick(int x, int y)`
  - `bool IsMoveValid(player_id, move)`
  - `GameUpdates Tick(moves_by_player_id)`

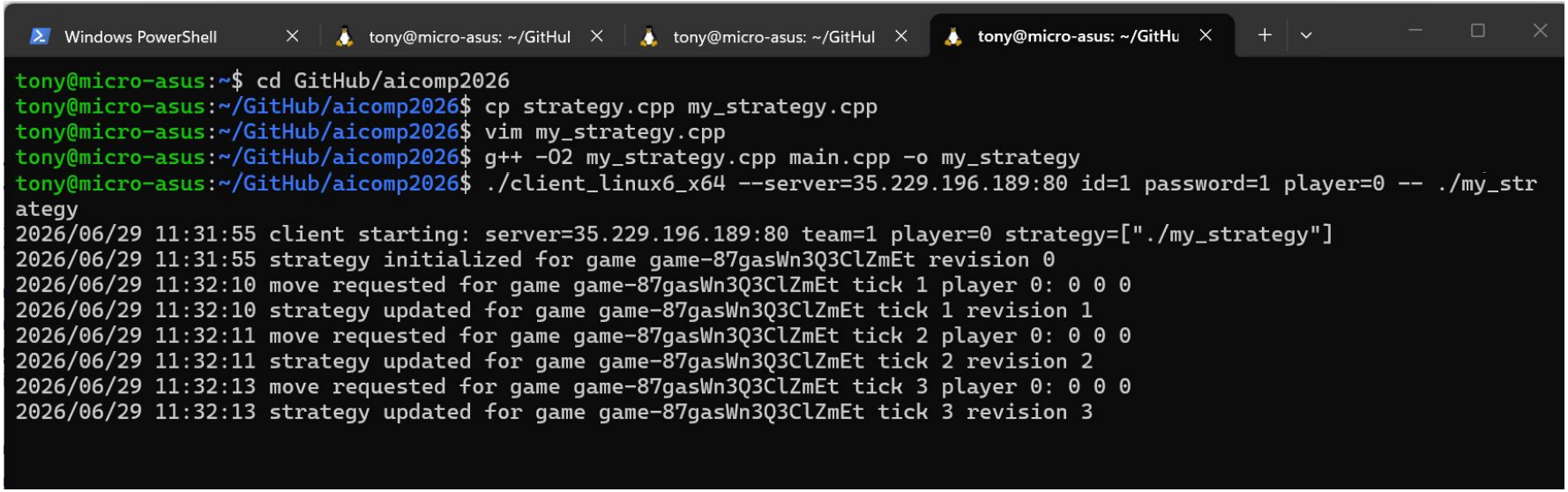
Implementation and Testing

Once you are done, connect your strategy to the tournament server

- g++ -O2 my_strategy.cpp main.cpp -o my_strategy
- `./client --server=<ip> id=<team id> password=<team password> player=[0..3] -- ./my_strategy`
- You should see that the strategy is now connected.
- Open multiple terminals and start multiple strategies with different player ids



WSL2 example



Solution Sharing

Strategy 1: Dodge Only

- Only dodge incoming bombs and the shrinking spiral of walls
- Pros: Doesn’t require much implementation (can be done with BFS)
- Cons: Requires other players to bomb crates for you to be able to dodge the spiral, which most of time leads to your player being trapped by a bomb

Strategy 2: Occupying center

- Path towards the center cell and place bombs to break crates if necessary
- Pros: The center cell is the last cell to fall in the spiral; securing that cell means victory as long as the player stays alive
- Cons: If the player cannot secure the center cell, they instantly lose as they have no way of killing their opponent

Strategy 2.1: Occupying center, improved

- Same philosophy as strategy 2, but places bombs to kill opponents occupying the center cell
- This either kills that opponent or frees up the center cell, alleviating the cons of the strategy 2

Strategy 3: Aggressive

- Greedily paths towards opponents, and places bombs to attempt to kill them
- Pros: Can effectively eliminate opponents in earlier stages of the game, when more crates are still present to limit movement
- Cons: Not very effective when the terrain has less crates left, as opponents will almost always have an escape path

Strategy 3.1: Aggressive with more lookahead

- Instead of killing the opponent, use bombs to restrict their movement and “pushes” them to a corner for a kill
- This would be effective even when the terrain is more open, but requires significantly more logic in implementation

Heuristics

- All of the strategies above are **heuristics**, a general rule-of-thumb that gives us a good enough move
- This is common for game strategies, where a “best move” is usually difficult to define with logic only
- Heuristics usually don’t require much algorithm knowledge – relies more on thinking about what aspects of the game are more/less important

How About Game State Trees?

- Some of you might be familiar with the concept of a **Game State Tree** in combinatorial games
- It models every possible game state, and searching through the tree with algorithms (e.g. α-β Tree Search) would give us the mathematically optimal move at each state
- Sounds good here, when the game is complicated and we don't have a good heuristic that solves the game...

However...

- This game (unlike Wall Go from last year) is not a combinatorial game
- Although we can still in theory use adaptations of game state trees and searching algorithms (e.g. DUCT for simultaneous games), we will still need a good heuristic function (in which we don't) to evaluate game states due to the large depth of the tree
- It could be useful in endgames where the game state tree would be relatively small and shallow