BH 2020 Postmortem

Team IDIOT

1 Introduction

by Houwang

Battlecode is an AI RTS programming competition where competitors design and program bots to compete against each other in a programming competition.

“Battlecode is a real-time strategy game, for which you will write an AI player. Your AI player will need to strategically manage a robot army and control how your robots work together to defeat the enemy team. As a contestant, you will learn to use artificial intelligence, pathfinding, distributed algorithms, and communications to make your player as competitive as possible.” https://battlecode.org/

Each year, teams worldwide compete to win prizes, or simply for the game itself.

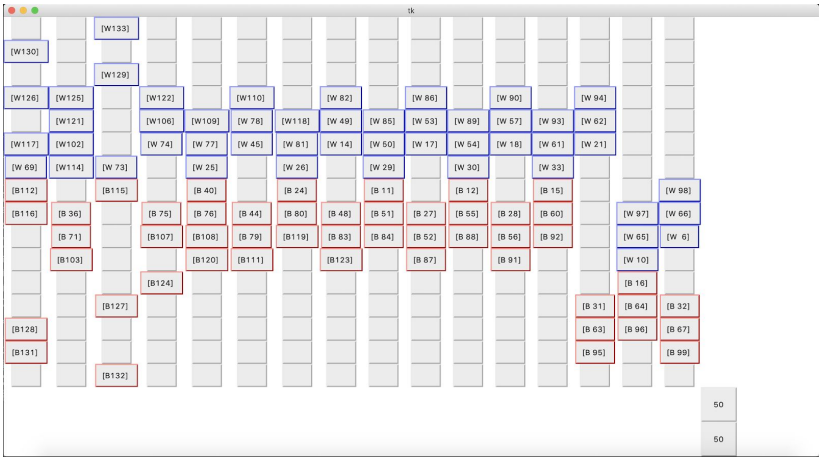
We are two highschool students from China who competed in Battlehack (A short version of Battlecode from the same organizer), finishing in top 4 - beaten by the final winner “Snakes and Ladders” in the semi-finals.

IDIOT: https://github.com/IDl0T
Houwang: https://github.com/Houwang123

2 Rules

by Houwang

http://bh2020.battlecode.org/specs.html
Battlehack 2020 took place on a 16 by 16 chess board, where the goal was to push pawns up high as possible. There are only pawns on the board, and each turn teams are allowed to spawn one pawn.



(Ongoing match example)
Pawns can either move forward or capture diagonally, as consistent with chess rules. This lasts 500 rounds, where in each round all units on the board will move.

When spawning pawns, the code is able to see the whole field. However, each individual pawn is only able to see bots in a 5*5 square centered around itself, and it can only use this as well as its own position to make decisions.
At the end of the match, the winner is determined as follows. Pawns in each role are counted, and the team with the pawn closest to the opponent's side of the board wins. If both teams have pawns equally pushed forward, the number of pawns on that row were counted and the team which owned more wins. Failing that, a coin flip decided ties.
An important part of this game was the ***turn queue.*** Turn order is dependent on each pawn and the spawning order - pawns spawned first will move first and irrespective of the team. This is a significant factor in pawn structure effectiveness. See **4 Strategies**

3 Preliminary Analysis

by IDIOT

We started with excluding unimportant specs. Bytecode limits, a method to measure resource usage, was not a cause of concern as the limit was set high. Macro[1], in practice, had limited effect. The only macro adopted by all top players is to maintain a balance of ally forces. i.e. Spawn more in weak columns. The spawning order, which significantly affects the result of a pawn trade, is difficult to control, so we ruled that out of consideration. Moreover, we found pawn advantages are much more important than positional advantages - since the game lasts 500 rounds, minor pawn advantages aggregate to an overwhelming advantage well enough to crush the enemy.

Therefore, the most diverse and influential strategy piece is **micro[2]**, and it should focus on obtaining a **pawn number advantage**. In other words, giving up a good position for pawn advantage is acceptable.

[1] Strategies in larger scale. Here, it means spawning.
[2] Strategies in smaller scale. e.g. combat&move logic for pawns

4 Strategies

by IDIOT

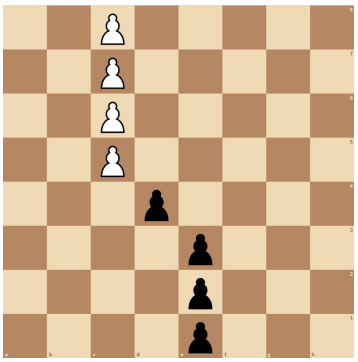


Fig. 1: Black lost in the trade

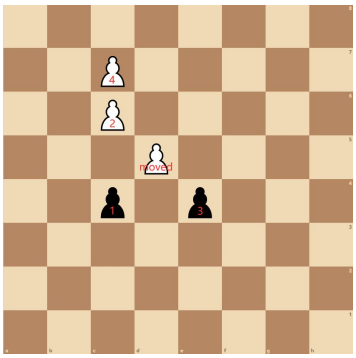


Fig. 2: White lost in the trade

4.1 Support Score

As stated in *3 Preliminary Analysis*, our micro should focus on trade & pawn advantage. Here we listed all causes of inefficient trade (lose more pawns than enemy does):

1. In a continuous exchange, the player with **fewer supporting pawns**[3] loses position, which means a future pawn loss if the player tries to push back. In *Fig. 1*, the remaining white pawn can hold position until a black pawn pushes forward, then it captures it.
2. In a continuous exchange, the player with a **bad move order** loses more pawns. *Fig. 2*
3. Non-continuous exchange. This can be totally avoided by not to suicide.

Therefore, we can infer the **supporting pawns** of both ally and enemy:

As seen in *Fig. 3*, every ally in the (+1) location contributes to the overall **support score** (a location in the back will only be active if the former one is active --- nested if). We speculate every enemy in (-3) location has 2 more backups (worst situation). Particularly, an ally in the (infer) location can reduce one or more (-3) to (-1).

[3] Pawns that can capture back immediately when an ally is captured

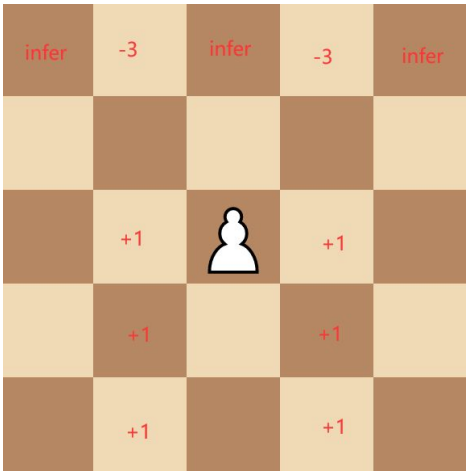


Fig. 3: The contribution to support score on different board locations. Locations not numbered are unimportant.

If score > 0, the pawn will move forward. However, it means a pawn will never advance in a stalemate. To address that, we introduce the **synced push**.

4.2 Synced Push

Firstly, we notice most of the time the board ends in something like *Fig. 4*. So the following statements are based on this particular stalemate situation. Pawns can know whether it's in this situation or not.

A **synced push**, as its name says, is that all pawns in the back move forward in the (almost) same time.

A synced push will immediately trigger a pawn trade. In the trade, the defender (Black) faces the risk stated in *4.1: the 2nd cause*. However, the attacker (White) has no such risk, since the supporting columns (even cols in *Fig. 4*) **don't need to move forward to catch up**.

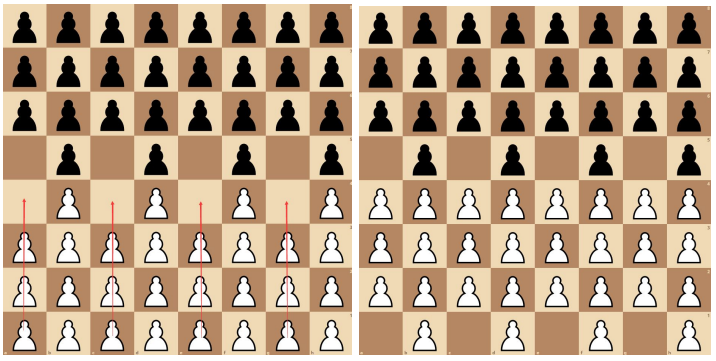


Fig. 4: A synced push.

How to sync? This is not obvious given that pawns cannot communicate and have no acknowledgement of the turn number. We set a timer in each pawn that it will start to push forward after some idle time. The key insight is: **let a pawn observe its "older[4]"**

[4] having more time standing still

ally and follow when it's pushing. For example, in *Fig. 5*, a white pawns watches 4 locations, and it will immediately set its "push timer" to 0 when it observes any ally pushing. Therefore, a synced push is achieved.

With **synced push**, we can hopefully gain pawn advantages and position at the same time. However, in practice, pushing too far forward could make your following-up pawns detach and incur a pawn loss. (*4.1: 1st cause*) This could explain our loss in the semi-final, where we got a considerable advantage early on but failed to maintain it.

Also, due to the 5x5 sensor range of pawns, the synced push can't be executed perfectly.

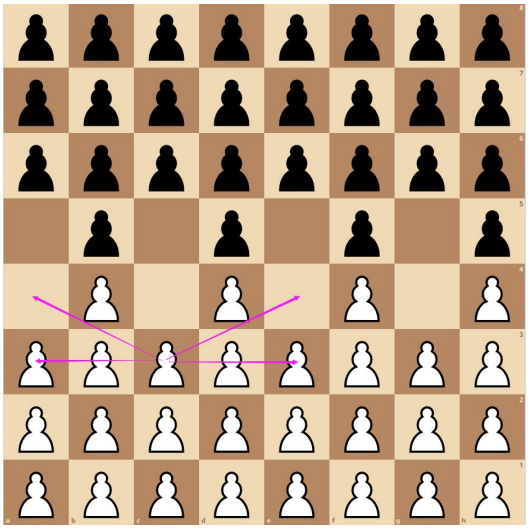


Fig. 5: A white pawn watches four locations.

4.3 Opening

A pawn can control at most 3 columns. Therefore, spawning 1 pawn in 3 columns guarantees the best position in the beginning.

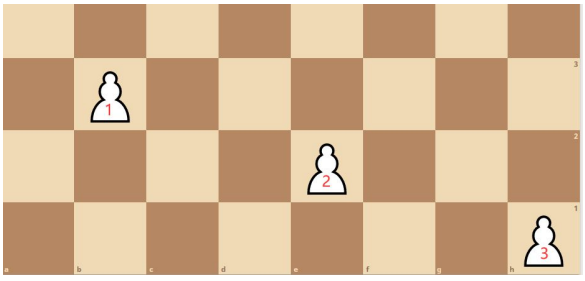


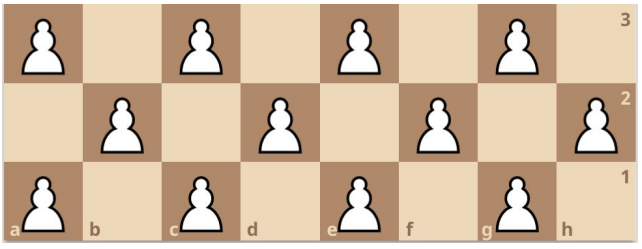
Fig. 6: Best opening

5 Experiments

by Houwang

5.1 Latticebot

Taking inspiration from former Battlecode matches, we chose to start with building a pawn lattice.



Such a structure allowed pawns to support each other when being captured and quickly beat the random *exampleplayer*. While the lattice structure later proved to not be the most efficient, *latticebot* acted as a useful benchmark with testing from *pressurebot* and *pillarbot*.

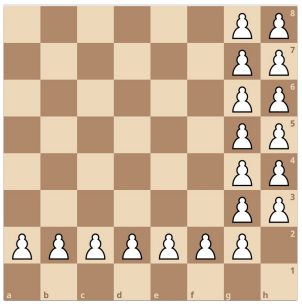
5.2 Pressurebot

Pressurebot was a version of *latticebot* with the same micro, that pushed pawns as high as possible, before attempting to synchronize pushes. It is one of the inspirations of *Syncbot*.

5.3 Pillarbot

Pillarbot was our first bot to largely use heuristics in determining pawn spawns. The intuition of this bot was as follows - attacking across the whole board will be less efficient than attacking along one column. Pillarbot utilized a pre-generated “defense lattice”, where we would try to build a wall (spawning heuristics determined) near our side of the board, before sending in all new pawns on the rightmost two columns.

This tactic successfully beat *pressurebot* and *latticebot* but quickly failed against other teams as it was shown it was always possible to defend a continuous push. A version of this tactic was later used with more success by team S&C and team Quantum mechanics for some time, though it was always vulnerable to aggressive whole court pushes.



5.4 Defensebot

Defencebot utilised the same spawning mechanism as *pillarbot*, but without two specific columns. This allowed it to never let an opponent reach our side of the board, but the lack of aggression caused us to fail to advance.

5.5 Controlbot

Controlbot was implemented as an aggressive *defencebot* which tried to move up aggressively, but rarely advance unless a pawn deemed our team to have an advantage. In scrimmages, this bot performed remarkably well, allowing us to move up to second on the leaderboard for some time.

5.6 Tradebot

Tradebot took inspiration from *ControlBot* and *Pillarbot*. It maintained aggression across the whole board, but tried to focus more on several columns, allowing us to create an advantage on one side of the board and maintain the other with well coded micro.

One significant change was made, which allowed controlbot to maintain a high position on the leaderboard.

Safe Micro - We greatly improved our micro for this iteration and made it more pawn efficient. See *4.1 Microheuristics.* This allowed us to gain a pawn advantage over weaker teams and improve the reliability.

5.7 Syncbot

At the time of Syncbot's conception, (2 days before the competition), tradebot was performing relatively well on the leaderboard (top 10) but was consistently losing to top tier teams.
Taking inspiration from *pressurebot,* syncbot utilised a synchronised advance every 25 turns which allowed us to gain position across the board while maintaining the pawn

count. This would overwhelm one sided pushes from tactics similar to *tradebot* and *pillarbot,* while the aggression would restrict the opponent from building enough forces to signal their own counterattack.
Syncbotv2 later placed top 4 in the final tournament.

Issues:

1. Aggression with lack of pawn buildup. See 4.2.
2. Theoretically, see 5.1, a focus on one side of the board would give a slight advantage. Syncbot failed to implement this in macro.

6 Final thoughts

Battlehack was a refreshing event during the middle of quarantine from COVID. There were few bugs in the engine, and we would like to congratulate Snakes and Ladders on their win! Also big thanks to Teh Devs for organizing the event.

Some suggestions:

- Make the board view symmetrical might greatly reduce the 1-1s[5]. It can be done by reversing the board in `get_board()`.
- Provide a clear list of restricted methods in restricted python. It is reusable for future contests.

7 Open source

Code is made available at
https://github.com/IDl0T/battlehack20

[5] For example, either team wins playing white.