CS1410 Final Writeup

Results:

My bot was able to beat RandBot > 95% of the time in all rooms, beat SafeBot 80% of the time in the small empty room, beat SafeBot about 60% of the time in a large empty room, beat SafeBot about 50% of the time in a large room with hyenas, beat AttackBot about 50% of the time in both small and large empty rooms, beat AttackBot 60% of the time in a large room with hyenas, beat TA1Bot about 40% of the time in a small empty room, beat TA1Bot about 10% of the time in a large empty room, and beat TA2Bot 40% of the time in both small and large empty rooms. My bot was not able to beat TA1Bot or TA2Bot at all in a large room with hyenas.

Backstory:

The very first thing that I tried was to use Alpha Beta pruning, using the Adversarial Search strategy. However, I realized that if I tried to transition and move the players to get the next board, it would actually move in the real game, which I didn't want (as Alpha Beta pruning requires trying many different moves and evaluating each of them before making an official move). I saw on Ed Discussion that a TA recommended someone make a deep copy of the GoT problem to then call move on, so that it won't move in the official game (and only make a move in the copy). I was really excited to then try this to improve my bot, but unfortunately even after I made a deep copy, it seemed to still be transitioning in the real game. I tried everything to debug and make it so that I could simulate moving and transitioning only in a copy and not the real thing, but nothing I tried worked.

Approach Development:

Thus, I decided to just stick with a heuristic that estimated the value of each possible move based on only the current board. How I achieved this was I looped through all possible actions, calculated a value for each action based off of certain conditions, then returned the action that gave the maximum value. The first conditions I checked for were basic edge cases that would cause the game to end, like hitting the opponent's or my own trail. If this was the case, I set the value equal to positive or negative infinity, respectively. The next thing that I considered was the distance of my player to the opponent. I calculated the distance of the opponent from my player using Euclidean distance. The closer the opponent is, the more I want to explore less and remain in my own territory. The farther the opponent is, the more I want to encourage exploring. The other factor I had originally considered was also the length of my temporary trail. The longer my temporary trail was, the more I wanted to encourage my player to go back to a permanent space to claim the area. Thus, I came up with 4 conditions: if the opponent is far from me and my temporary trail is long, then I should give a higher value to the action that would decrease the distance to the nearest permanent space; if the opponent is far from me and my temporary train is short, I can encourage further exploring and thus would give a higher value to the action that would increase the distance from the nearest permanent space; if the opponent is close to me and my temporary trail is long, I want to give a higher value to the action that would decrease the distance to the nearest permanent space; and lastly, if the opponent is close to me and my temporary trail is short, I can still encourage giving a higher value to the action that would decrease the distance to the nearest permanent space. However, a major downfall in my initial implementation of this was the way that I was calculating the distance to the nearest

permanent space. I was still using Euclidean distance, but the problem was with how I was keeping track of the permanent spaces. I did have enough time initially to figure out how to fill up the entire claimed space, so each time my bot claimed space, I was only keeping track of the spaces that were part of the temporary trail as permanent spaces, and nothing in between. I could see that this was causing my bot to exhibit weird behavior, so for the next submission, I looked at the function used to fill spaces in GoT problem code and used that logic to write my own function that would keep track of the permanent spaces. Another thing I did is I broke down my opponent distance conditions as well as my length of temporary trail conditions further into 3 sections each. If the opponent is extremely close to my player, with a distance of less than 1.5, I want to give an extremely high value to only staying within my claimed area, and not explore at all. Otherwise, if the opponent is still relatively close to me, I want to encourage exploring, but maybe only one or 2 at a time, rather than leaving a long trail. And lastly, if my opponent is really far from me, I only want to encourage exploring if my temporary trail is short; otherwise if my temporary trail is really long, I also want to encourage going back to a permanent space. Using these conditions, I assign each action a different value, and at the end the move that I take is the one which maximizes the value.

Once I saw that this heuristic was working fairly decently on all of the empty rooms, I just modified my code to account for hyenas. I handle these very similarly to how I handle the opponent. My strategy is to just avoid the hyenas when possible. Thus, I get the location of all hyenas, figure out the distance between the hyena that is closest to my player, and similarly assign values based on how close this is. I first used the exact same conditions as the opponent, but this seemed to not work well, so I adjusted the distance conditions a little for the hyenas, making the distance cutoffs bigger (aka even when the hyenas are slightly further away, my player should still go into defensive mode). This seemed to work better.

Shortcomings:
One of my biggest shortcomings is that I only consider the opponent's distance from my player, rather than also my opponent's distance from my temporary trail. Thus, sometimes my player will think it's ok to keep exploring, because my temporary trail isn't THAT long yet, and my player is quite a distance away from my opponent. However, the opponent could be really close to the start of my trail, and thus cut it off, making me lose. Another big shortcoming that I have is that my bot has no concept of the benefits of creating trails in a square-like manner, so instead of taking the shortest path to claim the most land, sometimes it will try to fill in all of the gaps inside the square before permanently claiming that area. Going along these lines, my bot also has no way to calculate or know that it can encircle the opponent or hyenas to end the game; its only function is to claim as much land as possible while avoiding the opponent and the hyenas. Additionally, sometimes when I observed my bot, even though the opponent was across the board from my player and thus the opponent's distance from my player was large, after claiming land for the first time, sometimes my bot would still remain in its own territory, so that seemed like a bug as well. Perhaps it was because of the way I was only considering distance to the nearest permanent space, and thus if all possible actions caused me to still remain in my permanent space, an arbitrary direction would have probably been chosen, which is a massive shortcoming. Also, my equations used to calculate value as well as my conditions were

somewhat arbitrary, just tuned with trial and error, so that may also be a reason why, and it is something I would like to fix further if I had more time. Lastly, a shortcoming I noticed only as I was doing this writeup, is that I never consider crashing into walls (I swear I thought I did, but looking back at my code now it seems that I don't???), which potentially is also causing some losses, but running my bot against the others haven't seemed to have any issues with running into walls, so perhaps my heuristic accounts for this implicitly, but perhaps it doesn't and this is another bug with my code.

Future Bots:
If I had more time, I would like to try to fix all of the above mentioned shortcomings to create a better and more accurate heuristic. Most of all though, I would like to really figure out how to do Alpha Beta pruning, or any method in adversarial search really, which this. It was really frustrating not being able to do this, especially when making a deep copy didn't work. Thus, if I had more time, it would have been nice to figure out how to do Alpha Beta pruning with cutoff, and thus that would have strengthened my bot greatly. If I had even more time, trying to figure out how to make a bot with Reinforcement Learning would have been interesting to do as well.