Zac Gross
Jonathan Gemmel
CSC 380
2/8/15

I am an undergrad.

Section 1: Youtube

Youtube Link

Section 2: Observations

- My first observation was that I could solve test case 1 by hand but it was trivial. I could not easily solve test case 2 by hand. So I ended up writing my own test case that I could fully solve. I wrote about this on the forum here.It was very helpful to fully understand a test case so I could reliably check to see if my algorithm was correct. Initially it was not and I would not have been able to tell otherwise.
- It also just helped me understand the problem more to do it by hand. I started seeing patterns that I didn't notice before like the similarity of branches by the leaves.
- I was initially unclear on the relationship between whose turn it was and whether I should be using min or max to pick the next turn. Again, doing the problem by hand helped me conceptually model the problem. I think I would have been able to correctly solve the problem without fully understanding this relationship but now I am confident in how the logic works.
- My program already takes a couple seconds to solve test case 2. I predict that the time required by my program increases very quickly as the branching factor increases. Test case 3 did not complete.
- I split the algorithm up between two steps: expanding the tree, and evaluating the tree. I chose to do it this way because it made more sense to me. I think you could combine the steps and expand the tree down to the leaves and then return back up with the evaluation of those nodes and it might save some time, but I didn't think it was worth it to complicate the process.
- It was beneficial to create a problem object to store all the data related to the current instance of the problem such as number of moves available and moves already used. It put all that data in one place and made it easy to track.
- Keeping the data of the current state separate from the node kept the code clean as well I think. I could keep the data of what was going in the state away from the node which was handling the tree logic.
- Most of the logic ended up being simpler than I thought. It was mostly just classic recursive methods on trees.

- I clocked my time and I did end up finishing in a bit over 3 hours. I took a break in the middle and also spent some time working out the hand done solution. I think starting from scratch I could definitely do it within the time limit.
- This problem was very light on data structures. I wrote my own node class but aside from that all I used was built in Python lists. This would be easy to port over to many other languages.

Section 3: Opinion Piece

What are the limitations of my dance battle engine?

Firstly, it can't do interactive play. I saw some people mention on the forums that they thought that was expected for the assignment. It isn't but I think it'd be a really cool function. I'd love to play a dance battle against my AI. I'm guessing it would win nearly every time but I think it'd be fun.

My engine was very susceptible to slowing down at large branching factors. The third test case did not finish. This is because it does an exhaustive search and does not have alpha-beta pruning. Setting a maximum depth and creating a heuristic to evaluate the nodes would majorly reduce cost on expanding the nodes. Adding alpha-beta pruning would greatly reduce the time to evaluate the nodes because it could skip over large branches once they are irrelevant.

As it currently is, even if it was given enough time to execute, it would use up a lot of memory because it expands the entire tree all at once. Again, the way to fix this is to set a maximum depth and create a heuristic.

If the rules change, such as allowing duplicate turns, I would have to heavily modify the engine. Portions such as tracking which turns have been used and generating child/successor nodes would have to be rewritten. It does not support changing to have Min play first.

There is no indication that the program is running or how close it is to completing when it is running. On test case 3 it just sits and looks like it is doing nothing. Ideally it would print out some indicator of what was going on and give a dump of the data if the program is stopped.

To enter a test case into my engine, it must be placed in the same folder and named a very specific way. It'd be nice to add in a feature to pass in the test case as a command line argument. That would make it easier to test many different cases and share input with other people.

The algorithm is pretty coupled with the specifics of the problem. I would not be able to easily apply this minmax algorithm to a very different problem. It would take some work to make a better class hierarchy and make the algorithm truly problem agnostic.

The engine is limited to two players. Three or more players would not work at all.