Checkers Player

Reuben Tanner *Blake Kelley*

"Hello Checkers"

Initially, one of the more challenging aspects of this assignment was getting the thing up and running. At first, we were running strictly on onyx which, after the correct libraries were provided, made debugging with anything more than print statements a massive pain. We quickly realized that this was going to be unacceptable if we really wanted to perform unit tests and ensure that the program was working as expected so we mounted the task of getting it running locally.

After spending quite a bit of time attempting to get the checkers program working on our local machines (Ubuntu 12.04) and receiving that magical email from you and Will Dietrich we finally got it up and running locally. Once we could actually run the GUI and run checkers, we quickly pushed it up to source control and started working on a method for debugging.

Debugging

Once the program was running locally, the task of being able to actually debug it began. After spending several hours trying to determine what an actual board looked like and how to get the program to into a step by step debuggable state to offer more insight into the program, we made a breakthrough. The proper solution only seemed to only have become clear after spending the time trying to hijack the pipe from the checkers server, write its output to a file, print it out to standard out and the other myriad of things that were

attempted. Once all that struggling had come to an end, a scan of the code in the main of the program showed an elegant solution that would solve our debugging woes; have the player play itself! After some commenting out and replacements combined with some cleverly placed #ifdefs, the player was up and running and playing itself in short order. All of this finally put us in a state where beginning to think about actually writing code made sense and seemed worthwhile (~10 hours to get here).

Pre-Development

Before actually beginning to code up our algorithm and heuristics, we felt it necessary to go over the existing code to understand HOW exactly this player performed its necessary operations to play checkers. Due to the horrendously ambiguous variable names, complete and total lack of formatting, spacing or naming conventions and downright bad design a complete run through of the code was required to make any sense of what was going on. Needless to say, there was a lot of stepping through line by line, renaming variables to be clear and descriptive, extracting constants in place of magic 0's, 1's and 2's floating around and the like. This whole process was very good, not just for this assignment but also for our future (or present) "bad code reading" skills in industry. Once a sufficient amount of time had been spent familiarizing ourselves with the code base, it was time to code up the algorithm.

Alpha-Beta Algorithm

Unfortunately, there isn't go to be much to say here because we just copied the code down that you had shown in class that worked. Although, to our credit, there were several times during the course of this project where we questioned our sanity and our ability to copy code down so a sufficient amount of time was spent watching youtube videos on the alpha-

beta search as well as minimax to figure out what was actually going on. The pseudocode from lecture5.ppt was compared, compared again and re-compared to what we already had and, surprisingly enough, we had no real big mistakes made in the algorithm itself.

Attempts (things you tried)

Ho ho ho, funny you should ask. Because there were attempts; lots of them. Being new to the whole heuristics based, game playing AI, we probably did not go about this the correct way in the first place because once alpha-beta was running we began to develop heuristics and I mean heuristic**s**. We started trying to do too many heuristics at once without fully ensuring that any of them were accurate through unit testing so there was quite a bit of time floundering with semi-working heuristics and trying to determine if we were having improvements or not. We also spent quite a bit of time tweaking our depth and there were many moments of confusion during this time. We had already decided that we were not going to attempt to do threads since neither of us had any threading knowledge but instead we were going to go for the hardcoded depth. This proved problematic at first for us because no matter how deep we set our depth in our player, the games would still finish in less than a second and we could not understand how or why. We tried everything we could think of to solve our depth mystery, even purposefully introducing segfaults in our code to attempt to bring the program down to see if our depths were being recognized. Eventually, one of our group members had the crackpot idea to swap the player arguments to checkers and that solved it. This prompted an investigation where we soon realized that how we were switching player turns was flawed when the player was player 1, we fixed that and we were able to play with both players no problem.

Other attempts: tracking pieces instead of iterating over the board (huge fail due to recursion), tracking the piece bounds to cut down on some search time (small fail due to not enough time to make it work).

Testing

In order to facilitate testing, we wrote a shell script (provided in the submission for future generations of checkers players) that ran a given number of trials against a given player for a given number of seconds and then provided a percentage of wins vs losses. Funny story about the test script: when we were in the early stages of testing we accidentally switched the wins vs losses so our losses were showing up as wins so we played it against rat, diff and depth5 and had 100% wins! We were ecstatic until we played random and only have 40-60% wins. Needless to say, profound confusion ensued and it even ended up as a draft email to you saying in essence: “I don’t think random is actually a random bot; I can beat diff, depth5 and rat 100% of the time but I can only beat random 50% of them time”. Once those words actually appeared, digitized on my screen, reality sunk in and I figured there must be something wrong with my script which prompted an investigation which showed that I had wins and losses switched *FACEPALM*. Once the bugs got worked out of the testing script, it proved to be an invaluable resource that saved a lot of time and provided quick results as to improvements or decreases in quality.

Another aspect of our testing involved unit tests for the heuristics. At one point during development, we stopped and thought “does this thing even do what we think it’s doing?” and thought, “well, we can keep guessing and making changes and seeing if our win percentage increases or we can do unit tests”. So, after a couple hours of writing the tests and working through them, ***many*** errors in the heuristics came to the surface and the games began to get won a lot more frequently.

Final Approach

Near the end of development, one of our group members woke up one morning and said "it's time to get done with this dang project" and wrote a checklist of the final things that needed to be done to ensure that the player was actually playing correctly. The checklist was as follows:

1. Unit test heuristics
2. Determine proper depths for different seconds
3. Recheck algorithm to ensure correctness
4. Figure out timing.

Once these things were completed, we still didn't beat depth5 but figured we had already spent enough time on it and couldn't afford any more (~40 hours).

Heuristics

The following heuristics were implemented from research on checkers heuristics, from reading checkers strategy and from watching the smarter bots play.

- hangOnWallsAndHomeRow - defensive heuristic that gave points to pieces that were on the homerow or on the walls.
- offensivePawns - offensive heuristic that pushed pawns towards kingship (BAD IDEA).
- middleKings - some checkers strategy that we read up on said that kings were more useful in the middle because they could do the most damage from there.
- jumpAvoidance - gave negative points to a piece if it moved directly into the path to be jumped.
- material difference - not the ratio.

Conclusion

Overall, this project was difficult and very discouraging at times. In fact, it made me cry; yes, you heard right, your project made a grown man cry….not really, I'm just kind of tired while writing this and it sounded funny. We learned a lot about game playing, about alpha-beta search, the importance of unit testing, the importance of getting a program into a debuggable/testable state, about not thrashing the code and just taking a break, about the effort it takes to get a good understanding of a crappy codebase and the importance of incremental development.