Final Report Package
CS 4850 Section 02
Spring 2023
SP-14 Green Chess AI
Tyler Ebersold, Michael Dirksen, Logan Lane
Professor Sharon Perry
10 April 2023
https://greenchessai.github.io/

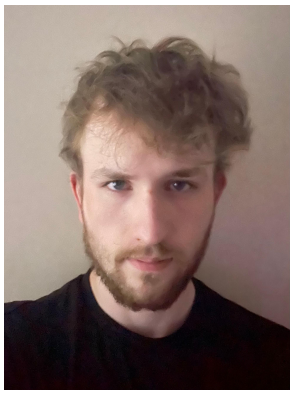


Table Of Contents

Overview:

Our project is to create a variation of the popular game Chess, using a special ruleset and modified piece behavior to provide a completely new playing experience. Additionally, the game will feature an AI opponent, intended to provide challenging yet fair games by using logic to weigh various moves against each other and select ones it finds to be better.

Project Team:

Roles	Name	Major Responsibilities	Contact
Team Leader	Tyler Ebersold	Organize Group and Responsibilities	404-232-5406
Team Members	Logan Lane	Developer and Documentation	678-221-1839
	Michael Dirksen	Developer and Documentation	404-333-3634
Instructor	Sharon Perry	Facilitate Project Process and Advise on planning and management.	Sperry46 on d2l

Final Deliverables:

- A Windows executable file containing our chess game, allowing you to play against an AI opponent
- A requirements document listing functional and non-functional requirements we had in mind while designing and creating the game
- A document describing the chess game rules
- Our project source code
- A project website with all of the above clearly organized and available

Milestone Events:

- First Milestone - By 2/28/23
    - Creating the board and the pieces along with the proper implementation of a graphical user interface.
- Second Milestone - By 3/15/23
    - Getting the pieces to move properly and allow them to be able to capture enemy pieces.
- Third Milestone - By 4/14/23

- Creating the AI that the player will compete against. The AI must be able to evaluate board positions, implement strategy, have a search algorithm to find the best move, and must follow the rules of the game (no illegal moves).

Collaboration and Communication Plan:

Our team will be using Discord as our primary method of communication and collaboration. Throughout the semester any internal meetings or work days will be conducted through Discord. Response times for team members are within 24 hours of communication.

Weekly communication will help us facilitate a successful project: Meeting times will be weekly either in class Monday and Wednesday from 11:00 - 12:15 or alternatively through Discord at varying intervals throughout the week.

Files and notes/resources will be shared through Discord, Google Docs, and GitHub. Team members have access to a shared Google account as well as a shared GitHub which will facilitate simultaneous workflow. Additionally, our work in Unity will be shared through version control software PlasticSCM.

The Project Leader, Tyler Ebersold, will be in charge of handling communication with the Project Manager Sharon Perry. However, all members of the Green Chess AI team will be able to communicate with either the Project Leader or Project Manager concerning deadlines and information pertaining to the success of the group.

Project Schedule and Task Planning:

Attached is the constructed Gantt Chart for our project. While originally very rough estimates, a lot of it turned out to be in line with what we were doing and some small tweaks were able to get it to where it needed to be - see the chart here.

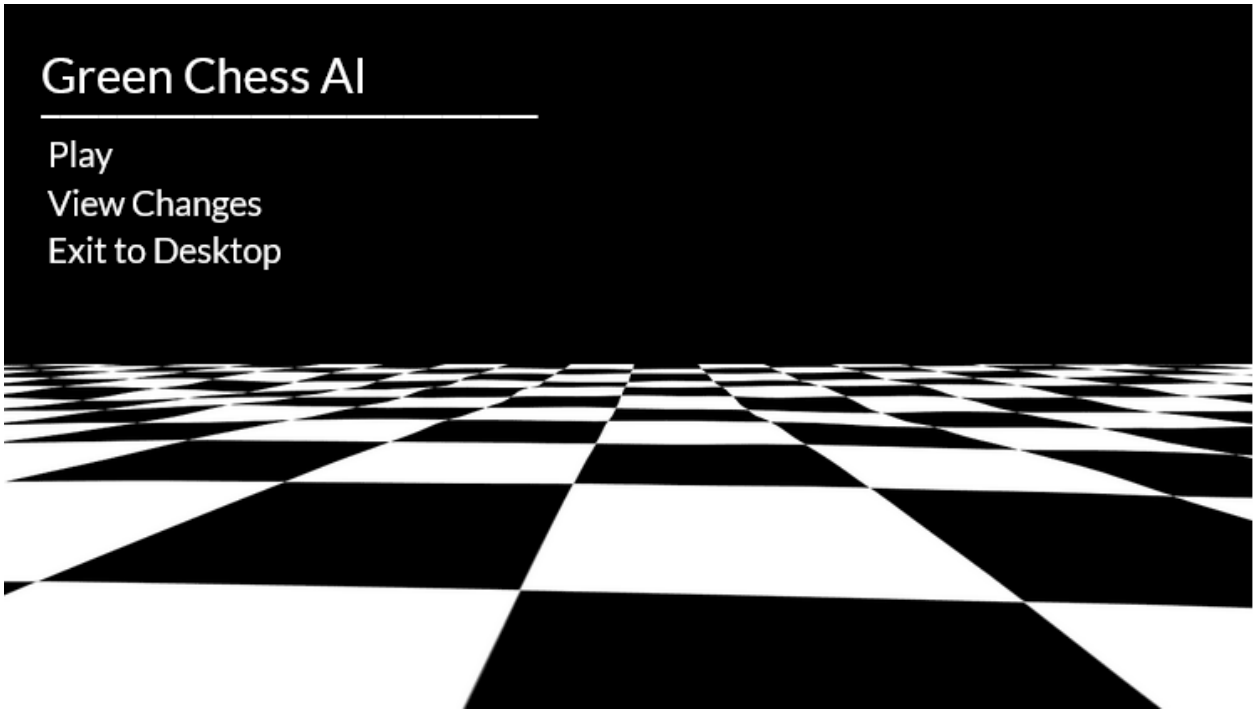
Deliverable	Task Name	% Complete	Work Hours	Current Status	Comments	Begin	Finish
Requirements	Define Requirements	100%	1	Complete			
	Review Requirements with SP	100%	1	Complete			
	Get Sign off on Requirements	100%	1	Complete			
	Complete Project Plan	100%	4	Complete		02/03/23	02/03/23
Project Design	Define Tech Required	0%	1				
	Develop Working Prototype	85%	100	Complete	Chess Board in Unity with ability to capture pieces	02/06/23	02/13/23
	Test Prototype	85%	6	In-Progress		02/14/23	02/20/23
	Website Beginnings	99%	24	Complete	Good Progress on website, few more things to add	02/20/23	04/01/23
Development	Review Prototype Design	100%	10	Complete		02/28/23	04/07/23
	Rework Requirements	100%	5	Complete		04/14/23	04/21/23
	Document Updated Design	100%	7	Complete		04/21/23	04/21/23
	Test Product	80%	9	In-Progress		04/21/23	04/28/23
Final Report	Presentation Preparation	100%	10	Complete	Presenting 3/13	04/28/23	05/06/23
	Poster Preparation	85%	10	Complete		04/28/23	05/06/23
	Final Report Submission	80%	20	In-Progress		02/14/23	04/20/23
Individual	WAR-1	100%	2	Complete		02/03/23	02/03/23
	WAR-2	100%	2	Complete		02/03/23	02/10/23
	WAR-3	100%	2	Complete		02/10/23	02/17/23
	WAR-4	100%	2	Complete		02/17/23	02/24/23
	WAR-5	100%	2	Complete		02/24/23	03/03/23
	WAR-6	100%	2	Complete		03/03/23	03/10/23
Milestones	Milestone 1 - Board Creation and Piece	100%	5			02/03/23	02/28/23
	Milestone 2 - Piece Movement and Capture	100%	10			02/03/23	03/15/23
	Milestone 3 - AI Completion and Testing	60%	15			02/03/23	4/14/23

Risk Assessment:

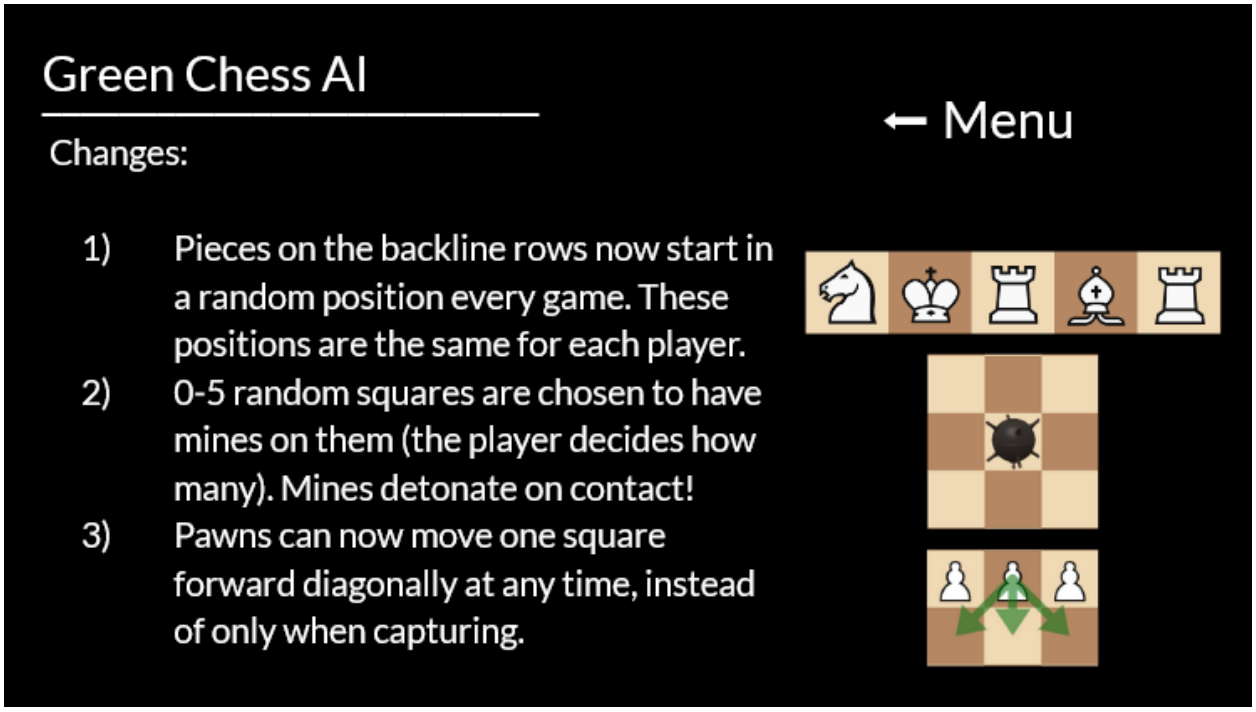
- Performance Issues: The algorithm for the AI can perform poorly if it is not optimized correctly. It needs to be made sure that the AI does not cause hardware to slow or consume too much processing power.
- Game Balance: The AI needs to provide a balanced gameplay experience for the user. If the AI becomes too strong and unbeatable, the user may find the experience unenjoyable. Additionally, if the AI is too weak and struggles to find good moves, the game will be boring.

UI Mockups:

Main Menu:



View Changes Screen:



Requirements:

1.0 Game Ruleset

1.1 Backline pieces are placed in random positions. The chosen positions are the same for both players.

1.2 Random squares not containing a piece will be selected to contain mines, which destroy any pieces that move onto it. These mines are shown briefly at the beginning of the game, but never again. After a mine detonates, it is gone forever.

1.3 Pawns can move diagonally forward as well as one square forward. On a pawn's first move, they may still move forward two squares, but cannot move two squares diagonally.

1.4 All other regular chess rules are in effect

2.0 Building the game

2.1 The game will be built in Unity and intended for Windows systems.

2.2 The project will use two-dimensional graphics

2.3 Develop game board and piece movement

2.3 Have a way for the AI to translate its decisions to in-game actions

3.0 Building the AI

3.1 Find a way to recognize valid, legal moves only

3.2 Utilize min-max algorithm to evaluate move effectiveness

3.3 Employ alpha-beta pruning to improve performance and eliminate unneeded moves

3.4 Make sure the AI never spends too long on a single move

4.0 User experience

4.1 The user will play vs our AI

4.2 Before the game begins, the user is allowed to choose the number of mines from a range of zero to five

4.3 The UI will be lightweight and clear. Information about the game, such as the active player's turn, check status, and captured pieces will be clearly displayed to the player.

Version Control Plan:

We used PlasticSCM for version control to help collaborate on the project. PlasticSCM is software built into Unity that allows for seamless integration into the Unity Hub as well as the engine itself, and makes simultaneous edits extremely easy. We were able to use PlasticSCM to share our code, keep track of changes made and rollback if needed, and generally keep our codebase organized. We also used a GitHub account to share our game's releases and all of our files through GitHub Pages, a free webhost.

Project Narrative / Technical Description:

Michael:
I have a lot of experience tweaking and modifying things that other people have already done, but I have never actually built something, especially a game from the ground up and I was very surprised by how much went into it. There is a lot of communication that needs to be done, and a lot of change in our experience was in the form of incremental tweaks, where you test something and it works maybe, but most of the time there's something that's a little off and you've got to go back and fix it – so you just take it one step at a time. Especially with three people simultaneously working on it everyone, is kind of – we're all working together, but everyone is doing their own like separate portion and so when we have to bring it all together there's little bits and pieces that don't exactly line up and so we have to make it line up.
We have to start with the rules of the game. The team that came before us didn't make chess, they made a game based off of chess, and we wanted to do the same thing except we wanted to make it more like chess. They had chess pieces but they moved completely differently and they had, like, "commanders" and so the game really wasn't the same. We liked the idea of keeping that chess game but making it play out differently, so little tweaks that would add a long – that would have a lasting effect in the long run there are a few things that we ended up doing and finalizing which was pawns being able to move diagonally, mines scattered across the board and then the randomization of the backline pieces. The mine thing was kind of a spur of the moment decision,

I was playing minesweeper and we were like making jokes about it and then we actually thought about it and it's like well if each game – if each new game has spaces that you just can't go on, that's going to change how everything plays out and that's going to change the decisions that the AI is going to make. So with that led the discussion about like, OK are we just gonna make the AI never step on a mine ever, or are we going to assign it like a very harsh negative weight so maybe it could still consider it if they could have no other option? Because we didn't want to lock up the game and have the AI in a stalemate type state but it was interesting to think about and talk about so we ended up including it. It's kind of a similar thing with the randomization of the backline pieces – it's not exactly the best game mechanic, and it can be frustrating at times, but it's a lot of fun to just play around with and see how you play differently and see how the AI plays differently. Some games we would get in and we just move our pawns, like, autopiloting, and then we realized "holy crap we've just boxed in our queen" or "we're about to get fools mated" or something, and sometimes the AI did catch that. Also, we ended up making one change when we actually got around to implementing our game and play testing and playing around with each other and playing with the AI, we found that pawns were super annoying and – they were unchecked completely by other pieces because they were able to move diagonally and so the game kind of was less about chess and more about like playing football trying to run your pawn down to the end zone and get the promotion before the other person did. Especially in late game scenarios there was no real way – like, if you just have a king and then one other backline piece, how do you stop your opponent's pawns? and you couldn't, really, and it was very frustrating. So we took out the ability for pawns to promote.
Another thing that came up with having to develop from scratch completely is that we all had to get used to each other and we had to get used to the game. Someone would do some work and they would push it or they would do like a pull request and we would say "what am I looking at – like, what is any of this? This isn't what I'm doing. I'm doing the same thing but differently – I'm working on the same concept but I'm building it completely differently, and you're doing that at the same time, so we had to –there were a few moments we had to sit down and come together and say,

"what are we doing? what's the plan here?" and it took a while to get used to. A specific example of this is in our source code, we have a file Chessman.cs that governs a piece's behavior. So we have someone coding the game in Unity that's using this Chessman.cs file and has these arrays for the board and keeping everything tracked, while someone else is working on the AI. Well we finish the AI, and we finish the game, and we go to combine them, and they are completely incompatible. The AI is using a custom class for the board, not an array, and its Piece class is completely different from Chessman – at the end of the day, its still only a few values, like color, type, etc. etc. But now we have to make this sort of bridge between the Chessman and the AI Piece class, and it's like, we should have thought of this before we started. We should have talked about this and had some communication. Because having two different chess piece classes that both can convert between each other, it's a bit silly. And we have a positions[,] array in the

code, as well as a Board class, that stores positions. But we were able to get everything working, so it turned out okay. Just a lot of time wasted and headaches involved.
Another issue that this caused was one of us would have like a preconception of something like, “oh we should always be the white player and the opponent – the AI – should always be the black player. Just for simplicity.” And then like a month down the line, someone who didn't implement it – the person who was coding something else – was like “OK, so can I play black”? and so we would look at the code and it's all hard coded and it's like – oh, it's kind of too late to have us play black now. So now the person's like, “well why you did you code it like that?” and you just have to shrug your shoulders and say like, “Oh well we could spend a long time changing everything and breaking everything and doing several fixes or we could just move on,” and sometimes we did have to go back and change everything and break everything. Tear it down to build it up stronger. But for something like that we just shrugged our shoulders and said hey lesson learned.
Testing is a big experience. You finish, or you get your program close to the finish line, and you're like alright let's do testing – I'm sure there's going to be a few issues we've got to iron out but it shouldn't take too long. You think you're right at the finish line, but there is so much that you have to do at that point forward – there are so many interactions that you didn’t anticipate or couldn’t anticipate that are going to come up, and it's not just logical bugs in the code, but it's also just like – we had a game where someone tried to block an attack on their king, so they moved in their queen to block it and – bam! The queen died to a mine explosion. So we got to this state that should be impossible in chess, where the white king is in check and it’s black’s turn. That should never happen – either it's checkmate, it's stalemate, or it's check and the player who is in check is taking their turn. So not only did our program not know what to do, it’s not like taking the king does anything, because you should never be in a position to take the king anyways, but we didn’t know what to do. We had to stop and say, “OK time out. The mines are supposed to be invisible so we have a few options here: we can either allow that so-called ‘impossible state’ and we can just make it legal to just take the king and that's the win, right, or we can have it so that you can't move on a mine if it would cause that state to happen.” We ended up making what we called the “fun” decision which is to allow that mine to be stepped on and your opponent to just take the king, because like, you didn't keep track of these mines. this doesn't have to be 1:1 regular chess, you lost. We would get situations like that more often than we thought. You know, you think these crazy interactions in your code is like one in a million ,but it actually happened all the time and so that's why comprehensive testing is so important. When we thought we were close to the finish line we were actually pretty far away.
Definitely the hardest part was not making the game or learning to collaborate but it's when we had our engine done, how do we do our AI? First off, we didn’t have much of a background in AI. I especially was completely clueless – Logan and Tyler knew a bit, so they told me the common, the standard way to do it is to use what's called the minimax algorithm. The way that minimax works – it’s a recursive algorithm, so it goes through a bunch of different moves and a bunch of different lines and it arrives at an end game board state. It has a bunch of internal

weights – each piece has a different weight, mines have a weight, and it uses these weights together to evaluate the board. If there's a lot of enemy piece activity near your king, that's worrying, and it's going to focus on how to get rid of that threat. If a move steps on a mine, it's probably not a good move, and it has a strong negative weight. So, the AI uses these weights and evaluates these end game board states and compares them with each other and then tries to say which one is the best for us. Then, it moves backwards from there, and says "OK, now that this is the best board state, how do we get there? what moves do I make to get there?" and you end up getting what's called a maximin value –a numerical representation of like, "assuming my opponent plays well, how good is this move?" and there are a lot of different factors that go into it there's where the pieces are there's where the mines are, there's what weights everything has, and it took a lot of us tweaking these values to get results we liked, but ultimately we feed it in an array that has all the information that it needs, every piece, every mine, and then it chugs the numbers and thinks about it for a bit and then it arrives at a few moves that it likes. Then it picks one and does it. Now the issue with this is, we implemented this code and we're like, "OK we're done. It works. It gives us a move." it actually started as a command line program that would just – we were testing, so we would feed it in an array – we didn't even have a game going, we just had like preset arrays and we fed the array and it would spit us out like a like a command line move like QxA2 or something – queen takes A2. So we got it working and we're like awesome OK let's put it in our game and so we put it in our game – which was a whole endeavor that I talked about earlier – and it takes forever, it was miserable. When you're in the command line you give it the command to run and you tab out and you go, "OK, I'll come back when you're done." When you're actually going to sit down and play this game it's completely boring. So I kind of felt like we were working at Google way back in the day. One of their founders said at a certain point early on all they cared about was speed. Speed is what's going to keep people invested, the longer people have to wait –every second they have to wait, they have an exponentially higher chance of leaving your site or stopping playing your game. So we did two things to help improve that, the first of which is we implemented Alpha-beta pruning. you have this gigantic tree of minimax values and different moves that this AI can take and what alpha beta pruning does is it kind of prevents the AI from going down a path of a horrible move. Because the minimax algorithm is recursive, it's going to go all the way to the end and then it's going to work its way backwards from there and alpha beta pruning kind of cuts that off and once you have a branch of a tree that is just a horrible move and has a horrible value it just prunes it off it says we don't need to do any comparisons over there we don't need to do anymore computations over there, it's done. And that saved a lot of time and that was very good, but our program was still too slow and we wanted to be able to – sorry, I wanted to be able to beat this AI. Logan and Tyler are a lot better at chess but I wanted to be able to beat this AI. So the other thing that we did is instead of trying to find The Best Move, why don't we just find a good move and go for that. Do we need to calculate every single possibility in the universe or when we find a good move we just say hey that's a good move let's do it? We're not trying to be the next deep blue we're not trying to be the next Stockfish, we just want to make a good and fun game and we

want to learn about AI. So sometimes it makes these horrible blunders where it doesn't look far ahead enough and sometimes it just plays way too conservatively and way too safely, especially with the queen. that was one thing we noticed it never wanted to give up its queen which we thought was interesting because I feel like queen sacrificing is a very AI thing to do and playing safe with your queen is a very human thing to do. Maybe we went a little too relaxed with trying to find just a good move. One issue though is we had this horrible bug where sometimes the AI just would not do anything – I didn't use the debugger to check, so I don't know if it was frozen or if it was just thinking and thinking and thinking for like hours, but I think it was a probably a logical error and it was probably like an infinite loop or something but sometimes we would just have to restart and we never did figure out what caused that.
But overall it was really interesting and really fun to make the program, we used unity just because it was like easy game dev for beginners. Looking back I probably wouldn't have, there was another group that just used like WinForms – I think that's C++, I'm not too sure – but it's just Microsoft standard windows GUI stuff, native windows, but when they did their presentation I was like, “Dang that looks good and that looks clean and that looks fast.” Unity is easy to use but everything felt very bloated like to do stuff you had to mess around in a GUI and you had to tie these scripts to these GUI objects and every time you changed something in the script you would go back to here and would take like a minute to reload them like a full minute and you would just sit there and the program was not usable during that time so looking back I don't know what I would have done because it was easy and it was helpful and there was a lot of tutorials online but maybe like a lightweight thing that we did in Python would have been more efficient, not sure. But the experience was invaluable just working with the team and learning all these different aspects of turning stuff in and documentation and teamwork. And we got it done!

Assumptions:

- The AI will assume that the player will make the best possible move each turn. Given each of the players moves, the AI must act accordingly.
- The AI will assume that there are no outside factors that will affect the game, the only possible moves are the legal moves on the board.

Challenges:

- Search Complexity: Creating an AI that can make legal and smart moves within a reasonable time frame. This was achieved using the minimax algorithm and alpha-beta pruning.
- Adaptability: Creating the AI to take into account factors such as the mines on the board, the random starting locations, and unpredictable moves from the user.
- Chess Logic: The overall logic and gameplay of chess, elements such as having the pieces move correctly, checkmate, and castling.

Test Plan:

1. Game Ruleset
    1.1. Backline pieces in randomized order.
    1.2. Mine positioning and explosiveness.
    1.3. Pawn movement diagonal.
2. Artificial Intelligence
    2.1. AI Movement, Valid moves, Captures
    2.2. Time spent per move
    2.3. AI winning game
3. User Interface
    3.1. New Game Button
    3.2. Mine Selection Slider
    3.3. Check Status shown
    3.4. Captured pieces shown

Test Report:

Testing was performed in three stages, testing the AI, testing the game engine, and testing the overall user experience. When testing the AI, we used a command line interface and fed the AI pre-set boards, observing its output, then immediately tweaking weights to see how it reacted. Eventually, we got the AI in a spot we wanted, so we moved it into the game engine and played game after game, sometimes with pre-set very unlikely scenarios. That way, we were able to test both the AI further but also the UX and any interesting interactions that might occur during gameplay.

Website / Source Code:

Our website can be found at https://greenchessai.github.io/. It is a hub for our project and contains all deliverables so far as well as a link to our GitHub where our source code can be found.

Appendices:

- Min-Max Algorithm: Recursive algorithm that evaluates a game backwards, assuming the opponent will play correctly, and finds the best way to reach that winning state.
- Alpha Beta Pruning: Search algorithm that looks to prune a tree in order to make a traversal algorithm more efficient. When poorly performing moves are found, that entire

branch of the tree will be trimmed off, meaning the min-max algorithm has much fewer moves to go through.