Spring 3-7-2019

Reversi Artificial Intelligence: A Project Management Analysis

Grant David Ross
Olivet Nazarene University, gross@olivet.edu

Reversi Artificial Intelligence:

A Project Management Analysis

Grant Ross

Olivet Nazarene University

Table of Contents

Abstract

This paper is meant to educate fellow computer science students of the importance of identifying, estimating and overcoming various challenges in software research through the lessons learned in the research for basic artificial intelligence (AI) conducted on the game Reversi. Teaching the various challenges and problems occurred along the way with the hopes that other students in the future can learn from and have an easier time producing better computer software research projects. Through educating other future students and taking the lessons learned in this project optimistically the next class of students will be able to create, manage and make better research in the future. Looking at the research conducted, and the data provided students and other novice programmers may be given new insight through the presentation of information found by researching AI algorithms. A new light will be shed on gathering, reading and presenting data for upcoming examination on software. For this the experience, lessons learned from research on basic AI will be used whose purpose conducted was to identify how AI preforms against other AI in the search to find the best algorithms and configurations for running a bot in a game or simulation. In the search for this the game Reversi was chosen for running these bots against each other. Open source code for the game and AI was used and then heavily modified to fit the needs of this research project. Data was gathered through many running's of the game with two AI bots playing each other on many different settings and used to search for trends in the data to find the best configuration for playing Reversi competitively. The program has many settings for the two bots such as search tree depth, evaluation method and search algorithm. The search tree depth was used for looking "X" amount of moves ahead where "X" ranged from 1-5 moves ahead meaning the search tree would be 2-6 deep. It no secret very quickly it became apparent the AI preformed much better the more it looked ahead versus looking for moves valuable in the short term. The evaluation method was much less clear off the bat as this had three configurations total score, sides and corners method where the bot would try to either score the most points possible in a single move, try to control the sides or walls of the board or try to control the corners of the board. The search tree algorithm was a way in which searching for nodes through the evaluation method was done behind the scenes however, between the two search algorithms Alpha-Beta Pruning and MiniMax there is no difference in game mechanics for the bots. However, one did take more computing power to calculate. The data from these bots playing each other was gathered, interpreted and analyzed to better understand AI and how it works. The programed developed for this project was the Advanced Programs Winner at Martin D. Walker School of Engineering Department of Computer Science in 2018 at Olivet Nazarene University.

An Introduction to Reversi

Reversi also known as Othello is a strategic board game meant for two players. The game is played on an 8x8 board with to-sided disks which can be one of two colors used for each player, typically this is black and white. Players will play their assigned color disk on their turn and will flip any disk of their opponent's color between the disk they just played and another one of their disks aligned in a straight line (horizontal, vertical or diagonal). The object of the game is to have the most disks turned to display your color when there aren't any more playable squares.

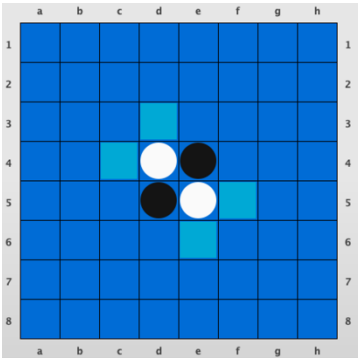

Figure 1

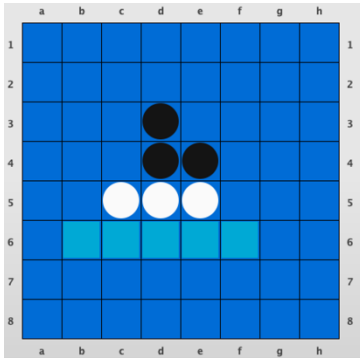

Figure 2

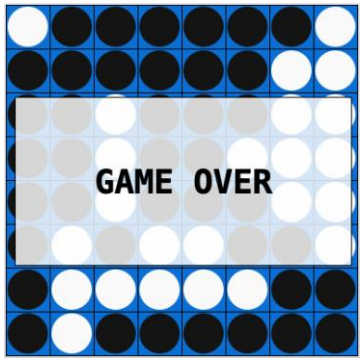

Figure 3

Gathering a basic fundamental understand of the game will be important in the continuation of this paper. The black player is always the first to move in Reversi and the game is setup with the two colors placed in the middle of the board in a 2x2 square with a diagonal pattern. Figure One displays the setup and the possible squares black could move to in a cyan color. Assuming black moved to D3 and white moved to C5 figure three shows the result and the next move possible by black. It can be helpful to think of a move as jumping the opponents color either horizontally, vertically or diagonally however, a disk is to be added to that given spot and no pieces will actually be moved, pieces are only added or flipped in this game. Figure 3 shows what a completed game could look like where the black player won with more disks flipped.

The Challenges of Editing Open Source

It would be quite a daunting task to program the entire game of Reversi, add AI and research the data from the games at the quality that was being aimed for. For this reason, open source code was searched for and used to minimize the workload of actually creating the game Reversi and instead focusing more on researching it. Editing and modifying open source code can be an issue for a couple reasons. The code used for this project was found on GitHub and needed to be modified quite a lot to fit the criteria of the project. It was quickly discovered that understanding someone else's code has many more challenges than expected. This may have been the first hurdle of the research project and was very overwhelming at first especially being

a software-oriented project. The three most useful things to keep in mind when editing others software are have a deep understanding of the software before coding starts, make small changes, and save different versions. When starting the software side of this project many hours were wasted trying to start coding the program right away, often times things were done incorrectly or could have been done in a much easier way and had to be re-done. Taking two or three hours to sit down fully understand how the code worked would have saved many hours on this project. A great way to gain a deeper understanding of code is to debug the program running many test cases slowly walking through the problem step by step to see how the operations work. Running these different tests shows how the program works and helps understand what the original developer had in mind when making the software. The next tip in working on someone else's code is to make very small changes, editing to much before testing has a much higher rate of things not working correctly and if things are broken it's hard to pin point what exactly broke the program. Typically, when individuals are working on their own code it's much easier to make changes that are more drastic as there is a greater understanding on how the program works. In general, it's smart to make small changes but this is specifically important to keep in mind when working on someone else's code. Lastly making many different versions was very important when making this program. Attempting to implement a major change that may take a couple hours to correctly code and apply may be disappointing when it breaks the program, at this point lots of time will be saved being able to re-launch a previous working version of code prior to breaking. Different versions also allow for trying and experimenting with different things due to the flexibility of always being able to return to pervious working versions.

Dealing with New Software and Java Libraries

Using the open source code provided there were some Java Libraries that were unfamiliar. Of these includes "javax.swing" which contains things like UIManager, GroupLayout and JPanel which are used in java to make graphical user interface (GUI). Never using JPanels before this was quite a learning curve to understand. To do this one would group the buttons, labels and panes they wanted into vertical and horizonal groups. These groups would list out the various things horizontally left to right and vertically top to bottom. Lots of time was wasted attempting to read the code, look at the program and understand why the layout looks the way it does. Instead when encountering a new library, it's a good idea to start with research on profession documentation published online to gather a better understand of what the library is and how it works. It took a while to get the hang of setting up the GUI correctly often times things need to be on the same level vertically or horizontally but on a different level on the other axis. Not doing research right away or being familiar with all of the java libraries caused more time to be put into the project and eventually led to research to understand and implement the libraries correctly.

Time Management

This project was planned out week by week over a couple months of worktime to ensure it would be on course to meet the deadline provided and to be the quality of work expected. Although the planned timeline was relatively close to what actually happened in reality, many more hours of work had to be used on a weekly basis to get the weekly update goal originally planned. With the various challenges that were unexpected such as implementing GitHub, modifying opensource code, and gathering the data of the games meant more hours had to be put into this project than estimated. For example, it was planned to import the project from github, organize through to code and play with the software a bit to get a better understanding in a three-hour timeframe however due to both computer technical difficulties and an unfamiliar understanding of Eclipse this process ended up taking nearly three times as long as expected. It is important to have good time estimations in time management because improper time estimations will lead to the inability to correctly know how much time needs to be deliciated toward a task with the time provided. This was a major challenge as nearly every time estimation ended up being half of the actual time it took to accomplish the goal at the quality expected. Looking back, it would be a better idea to explore the problem at hand and break it up into smaller pieces or sprints. Working while keeping an agile mindset would have helped the development process by helping the overall flow of the project go better by creating smaller goals that would have been easier to get an accurate time estimation on rather than trying to tackle and estimate a much larger problem. Thinking of each sprint that needs to be done, estimating the time required to do that sprint and adding all of the sprints together will generally be more accurate than taking a guess on a much larger problem. This was easily the hardest challenge of the entire project. Making sure the week to week goals were met while holding accountable to the goals set no matter how much time needed to be put in. It was very unmotivating to look at a large amount of work that needed to be done and often times instead of working toward that large goal it was easier to not do any work at all. A more agile process would have had a much smaller goals that would've been much more motivating to work toward. Weekly updates were required to be sent out to the professor who was overseeing the research, and this helped with accountability. Without this accountability it would be very easy to procrastinate the project and wait to do much of the work closer to the due date. Any project that is overwhelming should be broken down first by identifying a smaller subset of problems that can be solved then breaking the subsets into tasks or sprints. From there giving a time estimation on the time required to finish one sprint should be much easier than estimating the time of each larger goal. If this process was done correctly this project would have had much more time estimated on areas that were unclear and more time would have been devoted early in development saving time later for researching the data and trends obtained. Agile development capitalizes on flexibly and the ability to adjust to the needs of development which is specifically useful in time management. It was quite obvious that corners could be cut to save time by sacrificing quality however, since this project had a quality standard set as a goal these corners could not be cut at the expense of quality and instead had to be filled by spending more time to meet the criteria.

Program Overview

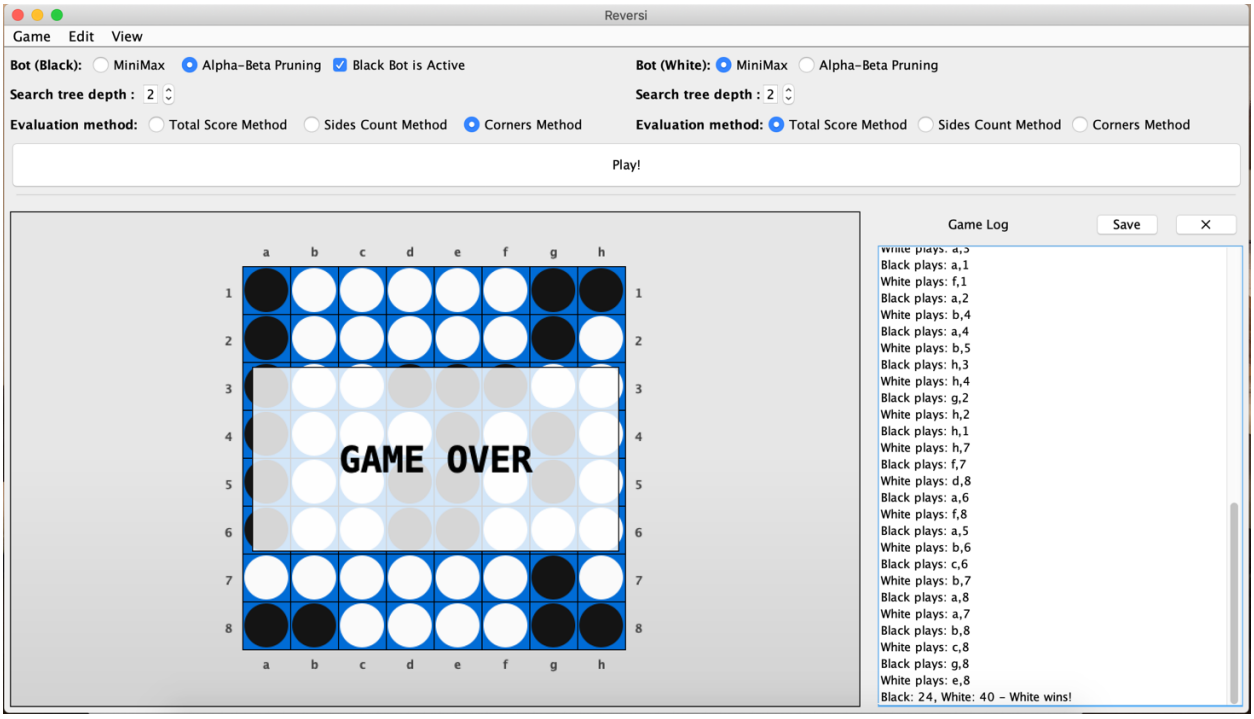

Figure 4

The figure above shows the software. Key things include the game log, save button, setting for both the black and white player. A play button to start the game and an area for the game to run and display. The black player bot contains a “Black Bot is Active” button this is used for activating and deactivating the bot essentially turning the game from a bot vs bot to human vs bot.

In a previous version of this program there was another button called “Mass Data” the purpose of this button was to run the game over and over with one bot rotating different settings as the other bot stays the same. These games were simulated and recorded in the game log with the settings used and the final score of the game. The purpose of this was to gather more data without having to setup and actually play game after game instead the script would do all these games itself. The game log showing the bot’s settings and the final score would be saved as a .txt document that could be used in research later. This was a very important feature recommended for when one needs to gather lots of data. When a task like this needs to be repeated over and over it’s smart to add automation to save time. Each bot contains fifteen different combinations of possibilities if we eliminate MiniMax and Alpha-Beta Pruning as they have no effect on the performance of the bot but instead are different ways of searching nodes. With 15 combinations possible for each bot there are 225 combinations of different bot settings to play through. It was much more productive to write a script to run the game over and over especially when some games would take a couple minuets to calculate the result depending on the depth of the search tree. Typically, a game where a search tree depth

is under 4 the game could be simulated in seconds however, when a search tree of 6 was used a single move would take longer than an entire game on a lower search tree depth.

```
939         switch (WhiteMethod) {
940             case 1:
941                 GameController.evalMethod1 = EvaluationMethod.VALID_MOVES_AND_TOTAL_SCORE;
942                 break;
943             case 2:
944                 GameController.evalMethod1 = EvaluationMethod.VALID_MOVES_AND_SIDES_COUNT;
945                 break;
946             case 3:
947                 GameController.evalMethod1 = EvaluationMethod.VALID_MOVES_AND_CORNERS;
948                 break;
949             case 4:
950                 GameController.evalMethod1 = EvaluationMethod.VALID_MOVES_AND_TOTAL_SCORE;
951                 WhiteMethod = 1;
952                 WhiteAlgo ++;
953                 break;
954         }
955 
956         //WHITE ALGO
957         switch (WhiteAlgo) {
958             case 1:
959                 GameController.algorithm1 = MiniMax.SearchAlgorithm.MINIMAX;
960                 break;
961             case 2:
962                 GameController.algorithm1 = MiniMax.SearchAlgorithm.ALPHA_BETA_PRUNING;
963                 break;
964             case 3:
965                 //end game
966                 break;
967         }
968 
969         //WHITE SPINNER
970             // switch statement with int data type
971             switch (WhiteSpinner) {
972                 case 1:
973                     GameController.bot2spinner = (int) 2;
974                     WhiteSpinner++;
975                     break;
976                 case 2:
977                     GameController.bot2spinner = (int) 3;
978                     WhiteSpinner++;
979                     break;
980                 case 3:
981                     GameController.bot2spinner = (int) 4;
982                     WhiteSpinner++;
983                     break;
984                 case 4:
985                     GameController.bot2spinner = (int) 5;
986                     WhiteSpinner++;
987                     break;
988                 case 5:
989                     GameController.bot2spinner = (int) 6;
990                     WhiteSpinner = 1;
991                     WhiteMethod++;
992                     break;
993             }
994 
```

Figure 5

This figure displays part of the code for running the program over and over while changing the settings. Each time the game was completed the search tree depth would increase, if it ever tried to increase past 6 it would reset to one and increase the evaluation method to a new setting.

This script was the easiest way to run through the various combinations of bots. The game log would mute all notifications except for the bot's settings and the final score. The data was then taken and moved into excel to be compiled and studied statistically.

Bot Strategies

The bots examined were composed of three evaluation methods total score, sides, and corners. Each method preformed differently, total looked into how many disks the bot could score in that move. The sides method looked into how the bot could have as many disks possible along the edge of the board. Lastly, the corners method looked to control all of the corners of the game. All of these are popular strategies among playing the game, typically having a corner results in having many points and controlling the edges will be a big upper hand however the object of the game is to score as much as possible. This made it challenging to guess how the bots would do playing against each other. The search tree depth was used to

look between one and five moves ahead in the game. A correlation could be drawn between an advanced player and a newer player where newer players are usually the ones who look to flip as many disks as they can, not seeing the bigger picture of the game and advanced players will look ahead to see greater possibilities. It was important throughout the process of this project to make hypothesis's and try to prove the correctness of such statement. Keeping the data and research in mind throughout the development is necessary when the end goal of the project is the data.

Reversi and its Algorithms

Importing the files of data into Excel allowed for the studying and graphing of the data so conclusions could be made for Reversi and the AI. Experiments were setup using base cases to test various theories and prove to look for the best methods of playing the game. One conclusion decided on before importing the data was looking more moves ahead as a more experienced player capable of planning moves would be thought to be better than a beginner player whom was looking in the short term not yet know what will help later in the game toward victory. This was quite obvious in the early stages of development however an uncertainty was playstyle, that is to say which method of playing will return the best results and be deemed the best way to competitively play Reversi based off the data provided.

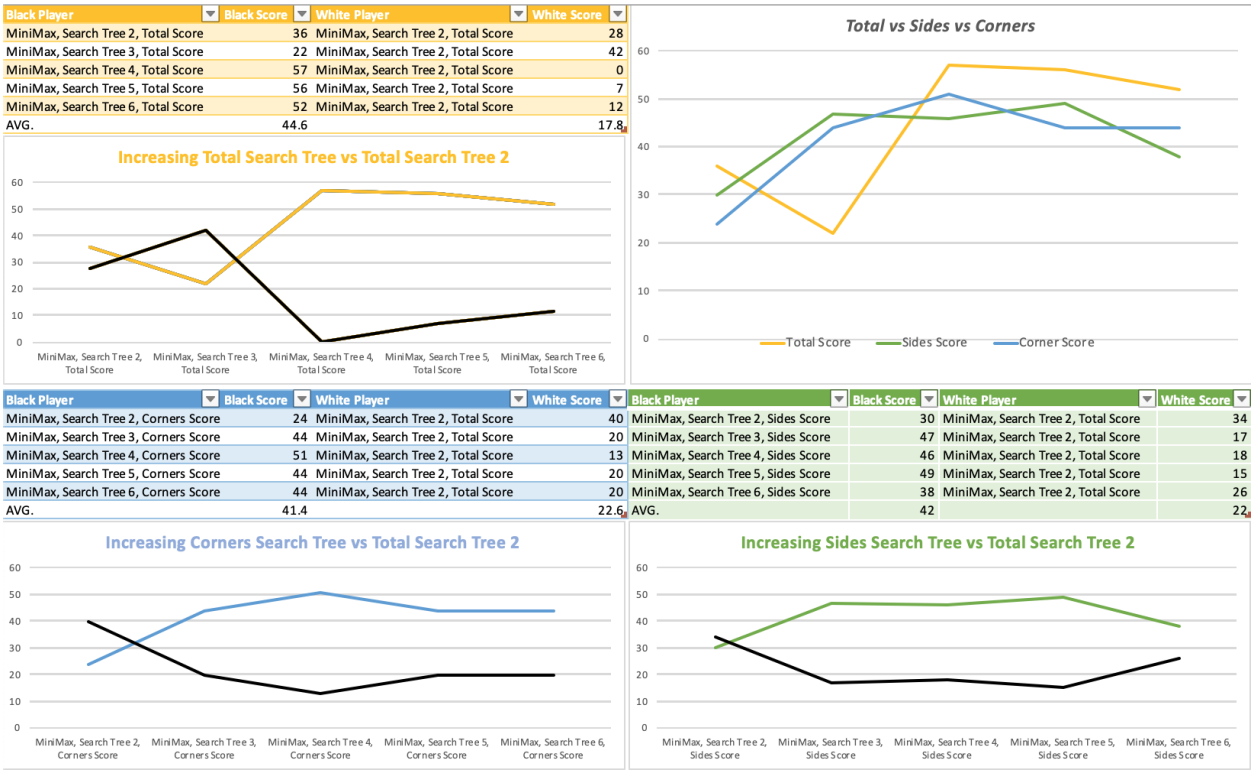

Figure 6

Figure 6 shows how a black player bot running each configuration did against a white bot playing looking one move ahead and a total score method. The purpose of the white bot was to be a base case for all of the configurations of every other possible bot to play against.

The other chart displays how many disks were scored against the base case to compare the methods to each other.

Looking at the graphs above it is easy to see that the base case (black line) was only able to win one game against each method. Not surprisingly this win was also only against bots looking one or two moves ahead of time anytime a bot looked ahead more than 2 moves it beat the base case. The number of disks scored were also put into a graph to see how the methods preformed against each other. Base cases were also created to run the sides and corners method, again only looking ahead one move like total score was to play against every other combination of bot. The data came out as the corner method averaged 33 disks per game, sides 25 disks, and total 21 disks per game. This corners bot not only won the most out of any other bot looking one move ahead it also averaged the most disks per game. One can draw conclusions from this that players who aren't too familiar with Reversi may have the best option of attempting to get the corners. This bot proved to be very good against humans too however if the strategy is known ahead of time it's much easier for the human to counter the bot's strategy.

Future Improvements

There are many areas this project can be expanded on by any future student looking to continue the research moving forward. One such area would be the implementation of self-learning AI that is to say an AI who knows how to play the game and through playing the game over and over will slowly self-learn how to play the game more effectively. It would be interesting to see how many times the bot would have to play to start frequently beating the other AI's and to see if it could beat the bots on the hardest settings or even a human player. The next big future improvement would be to look at every game setting possible and make a table of that data to study. Due computing restrictions the computer used could not always finish games that looked at a 5 moves ahead as 6-search tree depth vs a 6-search tree depth would take around 20 minuets to compute and eventually stop. The computer was trying to process thousands of moves ahead in the game and it would eventually crash the computer. The data from this method would be interesting to look for more trends game styles. The algorithm is also lacking a random factor, that is to say if there are three moves all equally weighted to occur the algorithm will always choose the first node. This is a problem because the outcome of two bot's playing on the same two settings will always be the same. If there was a randomness factor one would need to play the bots against each other on the same setting many times over to come up with a win and loss percentage vs having a set game that will re-occur every time. This change in randomness would make the game much different because one move that's different from the last game will lead to many more new moves and many more possibilities for more decisions to be made. With this randomness one may see that a total score bot looking one move ahead may beat a corner score bot looking four moves ahead 50% of the time however since this randomness was not put in the current program will either see the corner bot will win 100% of the time with no variation. Originally it was planned out to have some sort of self-learning AI however, the time estimation and other challenges that came up made this aspect not possible. If this program was to be released to the public, it

would be a good idea to update the graphics both on the GUI and on the board of the game. Much of the game looks rather plain and serves as function.

Conclusion

Although many things were learned on the technical side of things as well on reading and analyzing data. The major takeaways from this research project seemed to be more so on the various challenges of researching, creating, and publishing a large-scale project. The information provided may make it easier for future students to understand how to conduct a project dealing with statistics, open source and time management. It was interesting to be able to draw conclusions in the experiment such as playing Reversi and strategically trying to gain control of the corners proves to be a very effective way to play the game. Learning how to deal with the challenges that came about during the timeframe of this project such as editing open source, time management and staying agile in an independent project was very eye opening and I hope others seeing how the experiments where conducted and data was collected with the various challenges can help other students in the future with their software projects. The data reading and management was the easy part of the project the challenge was mostly in the creation on the software and the gathering of data that made this project so challenging.

Self-Lessons

Open source can be a bit intimidating to jump into it would be a good idea to first get familiar with the software obtained. Running through the program and debugging it a couple times with a few test cases would have helped me better understand what was going on and save lots of time down the road where time was spend trying to force things to work. Always make multiple versions for testing and modifying at first the entire project had to be restarted four or five times because it broke and I could not fix whatever broke. Sometimes this was due to messing with the files in Eclipse and sometimes this was due to seriously messing up feature and breaking the program. Lastly making small changes will always be a much smarter idea compared to making big changes to a program and not testing first. When working with new java libraries look online to find examples or professional documentation that will help guide in the right direction and allow for quicker learning. Time use and time estimations are the most important thing when doing a large project. This was by far the biggest takeaway from the project. Being able to develop in a more agile way would allow for better tracking of both of these things. Any future project programming in an agile way will be critical. The software was quite intense to write and could have been broken down into smaller bits making it much easier to grasp. At the end of the project the most important thing is the data so making sure to have plenty of accurate data is very important.

Sources

Reversi. (2019, February 23). Retrieved from https://en.wikipedia.org/wiki/Reversi

Working on someone else's code. (n.d.). Retrieved from https://softwareengineering.stackexchange.com/questions/149762/working-on-someone-elses-code

What Is Time Management?Working Smarter to Enhance Productivity. (n.d.). Retrieved from https://www.mindtools.com/pages/article/newHTE_00.htm

Cyberpython. (2010, December 10). Cyberpython/Reversi. Retrieved from https://github.com/cyberpython/Reversi