Optimization of Game Tactics

Zachary Mink and Ming Wong
Ithaca College
(Dated: May 8, 2011)

I. INTRODUCTION

In computer science, one way of optimizing a process is to run every possible configuration of the process, and keep track of which arrangement gave the most desirable results. If processing every configuration is not practical, one way of achieving an approximate, optimized solution is to "pick" many different configurations at random, and "choose" the best result out of those selected. For a computer science project, we decided to apply this method to find an optimized strategy for the computer game "StarCraft II."

"StarCraft II" is a real time strategy game. The basic objective of game is to build an army to defeat your opponent. To accomplish this task, a player must collect resources, construct buildings to create attacking units, and then build the attacking units themselves.

In the game "StarCraft II," one of the most important components to having a good strategy is to have a mostly set order of constructing buildings and creating units, generally called the "build order". Often, the only way to produce this order is to have memorized it from trial and error. To speed up the learning curve, our program intends to produce the optimal build order for a given strategy.

For simplicity, we decided to base the build order strategy on maximizing a player's "army value" at a chosen time. This army value is determined by the resource cost of each attacking unit, and gives an approximation of the strength of a player's army. Since army value is not the only factor contributing to the outcome of the game, our simplified program will only provide an insight into what makes a strong build order, not necessarily win the game for the user.

II. METHODS

We decided to make the entirety of our program a function in which the user inputs the time of gameplay that they would like their army value to be maximized, and the number of iterations the program will run for - increasing the accuracy of the output. The program is then governed by two main loops, one for running each iteration, or the number of games played, and the other for making decisions for every second of each simulated game.

Our program begins by initializing all of the information pertaining to each building and unit inside the iteration loop. This information is contained in large structures, detailing the name, number, resource cost, etc. of the unit or building in question.(See appendix A). Each structure contains parallel fields so that each type of information can be easily accessed through the use of cell arrays.

The main concept for the structure of our code was to have the "gameplay loop" either make a build choice, or nothing would be done every iteration. We used a time scale of one second for every iteration. The end time of the loop is determined by an input value - when the user wants their army value to be maximized.

After initializing all of our data, we began this "decision" loop by determining if a building or unit had been built during the previous iteration(see appendix B). If something had completed being built, all of the cell arrays containing the structs, and the structs themselves were updated.

Once we have a updated status, the program needs to determine which buildings and units can be built at that time. This begins by obtaining the amount of resources collected. Afterwards, a for loop determines which of the buildings and units can be built based on different requirements. If all of the "build requirements" are met for a certain unit or building, it is placed in what we called the "choice" cell array(see appendix C). At the end of this array, we also added the choice to do nothing.

After determining what can be built at that time, a choice to either build something or do nothing has to be made. To do this, we used a random number generator. We began by taking the choice array and assigning a numerical value, usually determined by the cumsum() function, to a cell of a vector representing each choice. !!!!!!!!!!!!!!!!!!!!!!!!. To decide which thing to build, we then compared a random generated number between zero and one to the vector created. If the number was less than or equal to the cell value, than that elements index was used to determine which choice to make(see appendix D).

After making a choice, the data structures are updated and the build choice is stored in a cell array that will be used later. Along with the build choice, the updated army value, and the time of making the choice will be stored in a vector. The loop will then run again for every "second" until the desired time. Once the desired time has been reached, the entire build order for that game will have been produced. At this point, the total army value will be compared to the last game's army value. Whichever army value is higher will have its respective build order stored, and the lower of the two's data will be erased and its vector will be reused(see appendix E). This process is then continued until the input number of iterations, creating an approximate maximized army value.

III. RESULTS

To test our program, we set the length of gameplay to be 5 minutes and the number of iterations to 10,000. The output is shown in Figure 1. With this information we played a game against the computer. We first played against a normal difficulty computer: Results

We then played against a hard difficulty computer with the same build order: Results

Compare output times to game output times

IV. CONCLUSIONS AND FUTURE WORK

Even though there is much that could be added to our program, it has already indicated that it could certainly be a helpful tool in learning to play StarCraft II.

Acknowledgments

Many thanks go to Dr. Doug Turnbull of the Ithaca College Computer Science Department