SOLVING WORDFEUD WITH ARTIFICIAL INTELLIGENCE

COURSE: DD143X

PROJECT SUPERVISOR: Michael Minock

Johan Wadenholt
860103-0235
johanwad@kth.se

Niklas Arnell
890628-0014
narnell@kth.se

STATEMENT OF COLLABORATION

We have been working together on every section in this report and throughout this course we have worked on each and every part collaboratively. The main parts have been the programming, testing, writing the report and doing the research. We have revised our code and the functions numerous of times together and both of us have complete understanding of all sections.

ABSTRACT

Background: With the introduction of Wordfeud on the mobile device market the general interest in word puzzle games has increased and hence the interest in programs designed to solve Wordfeud. Most Wordfeud solving programs designed today are in many ways limited since they do not look further ahead than what combination of letters that returns the highest score. This can be a huge limitation since these programs do not take in consideration what possibilities that might open up for the opponent.

Results: Wordfaid does sustain a positive win/loss-ratio while playing against the original program it is based on, Scrabaid. It does not always win by large amounts but all moves that Wordfaid makes have as much purpose of protecting itself against retaliation as playing high scoring words and hence not playing the riskiest words that could potentially result in huge wins.

Conclusions: Estimating your opponent's possible moves by randomizing his/her letters remaining in the bag several times and testing them on the board for each word that you could potentially play proves to be a valuable asset in the conquest for Wordfeud victory since it helps you avoid opening up big plays after your turn is over.

TABLE OF CONTENTS

INTRODUCTION

DESCRIPTION

Scrabble is a popular word puzzle game played by millions of people worldwide. The basic idea of Scrabble is to get as much points as possible by playing words consisting of a limited amount of tiles where each tile represents a letter. The board consists of a $15 \cdot 15$ grid of squares with some being multipliers which either multiplies the tile or the word/words it is connected to.

Each tile has a given amount of points based on how often it appears in the given dictionary. The sum of the points on the words tiles (and their multipliers) becomes the score of the word. The game starts with 100 tiles, where two of them are blanks giving the player the option of choosing which character it is supposed to represent. Every player has seven tiles on their hand until all tiles are placed on the board or neither of the player has the ability to form a word. The characters on the tiles are predefined based on the dictionary.

AIM AND OBJECTIVES

Our aim with creating a Scrabble solver is to create an AI which can compute different solutions and choose the best one based on relevant considerations. Our objective is to add as many features as possible to make the AI pick the best word possible.

RESEARCH QUESTIONS

During our research a few question have emerged. Our research is based on existing solvers and literature describing different techniques of playing scrabble. Different authors support different techniques which makes it up to us to evaluate these and investigate which techniques are best.

Are there any existing AI theories that can be applied on our study? Which combinations of factors will generate the most powerful solver? How many turns can we try to foresee within reasonable time?

DELIMITATION OF STUDY

Time is one factor which we will have to take into consideration. The number of operations a computer can perform during a period of time is limited which will lead to a trade-off between speed and accuracy.

METHODOLOGY

Our methodology to create the best Scrabble solver possible is divided into a few different steps. The first step is to collect information and choose which tactics to use while planning the solver, the second step is to program the solver and the methods to make it work, and the last step is to test the solver and evaluate its efficiency. We do however prepare for the fact that we might need to jump between the different steps as new information arises.

BACKGROUND

HISTORY OF SCRABBLE

The game was invented in the early 1940's by an out-of-work architect from New York named Alfred Mosher Butts. The game was named and trademarked "Scrabble" in 1948 by Butts and his partner James Brunot and remained under their development until 1972 when Selchow and Righter, a well-known game manufacturer bought the trademark. In 1989 HASBRO, the current owner of the trademark, bought the company after Selchow and Righter, at the time named COLECO, declared bankruptcy. Today, between one and two million sets of scrabble are sold each year in North America alone and is found in one out of three American homes.

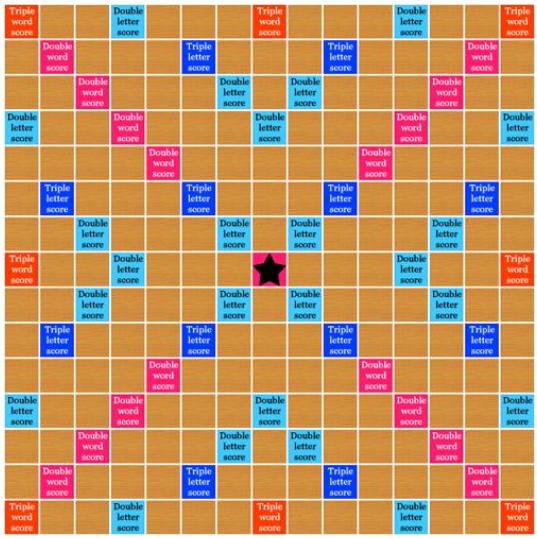

Figure 1: Traditional Scrabble board

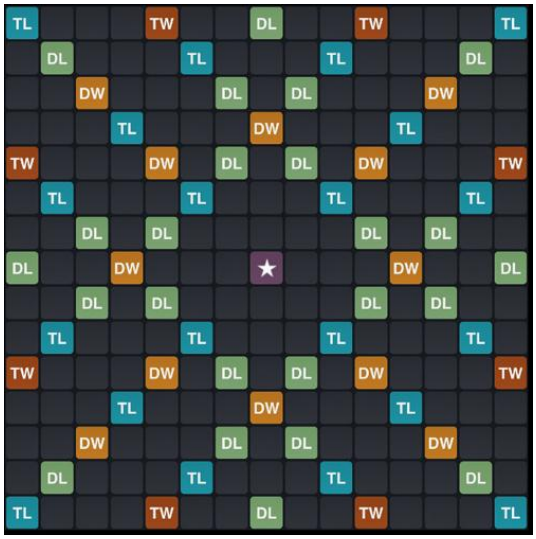

Figure 2: Wordfeud board

SCRABBLE TODAY

Since the introduction of Wordfeud, and similar applications on mobile devices, the general interest for word puzzle games has increased heavily. As a result of this there are also a lot of programs and applications that help players in different ways. However, something that most of these programs have in common is that they don't do any deeper analysis of the game. They are often limited to just analyzing how many points the player can possibly get during their next turn and not what move is most valuable for the player in the long run.

This can potentially lead to situations where if the player blindly follows the advice of the program they will set up even better possibilities for their opponents and with that the game might turn in the favor of your opponent which is obviously something you would want to avoid.

In these kinds of situations we think it could be interesting to do deeper analyses of the game and try to predict a couple of turns ahead by weighing in risks such as the one mentioned above.

ARE THERE ANY DIFFERENCES BETWEEN WORDFEUD AND SCRABBLE?

There are some minor differences between the two. Wordfeud has 104 tiles at start where Scrabble has 100. Some letters have slightly different values and the boards are set up in different ways with the multipliers placed differently.

All in all Wordfeud is almost the same thing as Scrabble with the same rules and goals but on a different game board.

SOLVING THE BOARD

As mentioned, the traditional Scrabble solvers only analyze the highest score a player can possibly achieve during his next turn. This might at times be the best solution but there are a lot of situations where this is not true. A negative aspect is for example the fact that you might provide your opponent with an advantageous opportunity. In order to find a better solution we need to go deeper, look ahead, and optimize with the information obtained from approximating the possible next steps.

Due to the nature of Scrabble we will only have partial information throughout the major part of the game. Since we do not know exactly which tiles our opponent has we will have to do extensive probability calculations on the tiles remaining in the bag. There are several ways to conduct these probability calculations and there are also several things you can analyze and it is not obvious which is best.

Since our program will need to perform a large amount of calculations in order to get proper results we understand that time will be an issue and hence it will be required to limit the amount of words analyzed and the amount of turns analyzed ahead with a time budget.

Due to the time constraints it will not be possible to analyze the game with an exact deterministic algorithm since it will timewise quickly grow beyond reason. We will instead need to use a “Monte Carlo” technique where we use random samples from the game in order to compute results. We deem that the pain threshold of how long someone is willing to wait for an answer to be somewhere around 100 seconds.

After running some initial tests with the program we came to the conclusion that with 100 seconds at hand we will for the major part of the game not be able to analyze further than one turn ahead. However, due to the fact that the probability calculations increase in value as the amount of tiles that we draw random samples from decreases it is fairly obvious that at the start of the game with 93 tiles

left in the bag, the random samples from two turns ahead combined with our time budget of 100 seconds will prove to be so random that they would not help us much.

Since we during the endgame no longer have the random element that comes with partial visibility in our calculations and rather have full visibility of the situation we will only need to determine our opponent's possible moves one time and because of that there will be quite a lot of time remaining in our budget and hence we can throughout this phase afford to look another step ahead.

After weighing all the options and considering the time constraints limiting us we have decided to use an existing open source scrabble solver written in Java as a base for our project and expanding it to include all the AI-elements we want.

The software we are using is called Scrabaid and was developed by Linus Chang in 2001 and has been left untouched ever since. The original program has no AI whatsoever and is limited to just returning the highest score available with the letters in your tray. However, even with this base software at our hands we still estimate that the programming phase will be quite extensive and be the major time sink.

CREATING THE SOLVER

The original software has the following three components:

- A word list generator that takes a word list of your choice and splits it into several different smaller word lists where each different word list contains words of a certain length.
- A hash generator that creates hash-strings for every possible combination of words using the generated word lists.
- A solver program that uses the aforementioned word lists and hash-tables to provide the user with a program that given a specific game board and player tray can return all the best words available in the word list composed of letters available on the board and in the player tray.

So how would one go about creating an AI with these elements at hand?
The first thing we implemented was a method to keep track of which tiles that still remained in the bag. This was done by just having a string representing a full bag and then iterating through the game board and player tray and removing the elements found there. The remainder would now be the tiles hidden to us, the opponent's tray plus the tiles left in the bag. This method was essential to later on be able to randomize letters for our probability analysis.

The next step was that for each match the program returned on the GUI we needed to analyze one step further to see what opportunities that might open up for the opposing player.
For each match returned from the program we did another analysis with the matched word placed on the board along with a randomized tray composed of tiles still remaining in the bag.

If you look at figure 3 or 4 below, you will see two instances of our Wordfaid.
As can be seen there are two values attached to each word in the list. The first value is the net gain the word will give you compared to the averaged score Wordfaid have estimated for your opponent and the second value is the actual score the word is worth.

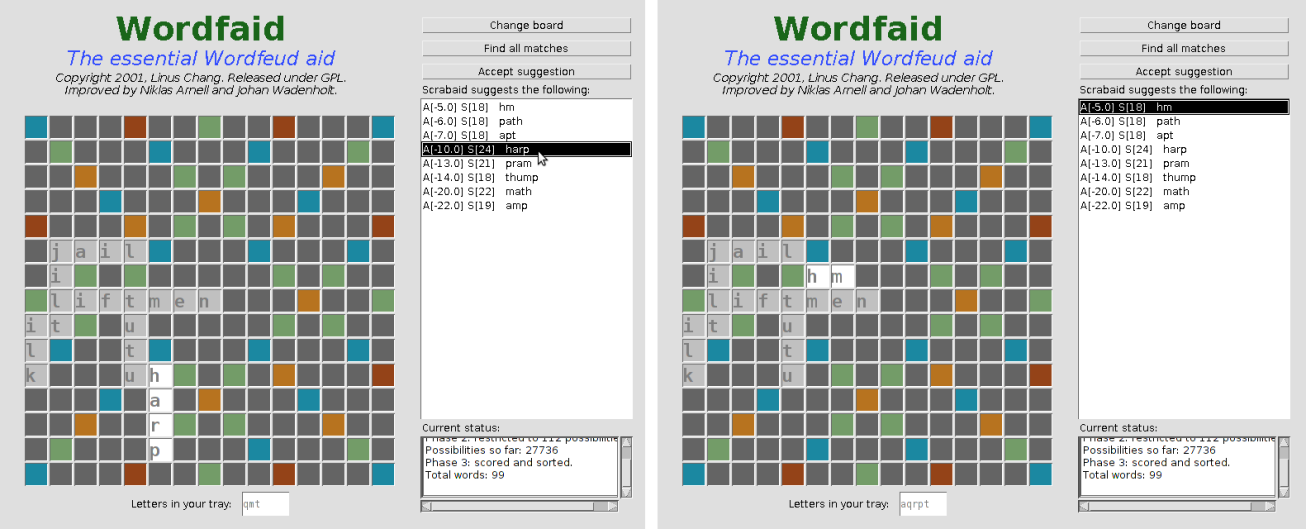

Figure 3 and Figure 4: Instances of the Wordfaid application.

Let us look a bit closer on the example shown in the figures above. If you look at figure 3 and figure 4 you will see that they are both instances of the same board but with different words currently chosen. A non-intelligent Wordfeud solver would here choose "harp" since it gives you 24 points and would at a first look be the obvious choice. Our solver does however notice that it would open up possibilities such as "harpy" for our opponent and because of that it downranks the word.

It analyzes this by, for each word in the list, placing that word on the board, randomizing seven tiles from the bag for our opponent, and look for all possible solutions on the board. It then keeps track of the highest score our opponent could possibly get with these tiles and after repeating this analysis for each word ten times we retrieve an average maximum score. Wordfaid will now subtract this average from the actual score of the word it played on the board and return a sum. If the sum is positive, odds are that you will be in a better position than your opponent after the next turn.

As seen in figure 3 and 4, unfortunately all the words in the list return a negative sum but Wordfaid does however suggest that we play "hm" on the board to minimize our losses. (Figure 4)

This also directly led us to our method of determining when it was the right time to change tiles. If all the possible matches got a score below a predetermined value we proposed a change of tiles. So which tiles do you want to trade in? We created another method, with similar function to the random “next-turn”-analysis mentioned above, where we for each possible unique combinations of letters of all sizes

from one to six tried filling out the missing letters with random ones from the bag and running the random analysis ten times with the newly created tray consisting of part random tiles and part old tiles to get an estimated possible score from the next turn. This will however consume quite a lot of time since we will get so many combinations,

$$\binom{7}{6}+\binom{7}{5}+\binom{7}{4}+\binom{7}{3}+\binom{7}{2}+\binom{7}{1}$$

which equals 126 to be exact, and we need to do the next step analysis for each instance.

It means that we will have 1260 instances of the original Scrabaid and we will for each of these 1260 instances look for all the best scores available with that unique tray and unfortunately this will take quite some time, around ten minutes on a fairly fast computer.

The time complexity is something that we have had to wrestle with during the whole programming phase. In order to get a really good analysis you have to examine a large amount of possible answers which in turn requires quite some computational power to handle. We have limited our program to only examine the top 20 words returned from the original Scrabaid. (20 words then examine each word 10 times = 200 instances.) We did mention earlier that we would like to examine our own turn after we have approximated our opponent's turn but it is here easily seen that for the major part of the game it is unrealistic since the time consumption will scale out of hands. To find all matches would then take closer to 10 minutes compared to the single minute it takes without the extra analysis. It is also not really that interesting to know what you can possibly play in your own upcoming turn until you close in on the endgame phase since the results returned will be far from exact since we only run it ten times.

The endgame phase begins when there are no tiles left in the bag. At this state of the game, instead of randomizing tiles from the bag and use probability to find their average score, we use the tiles left in the game (the ones currently on the opponent's rack) and calculates which score our tiles would have. If we can make an exit (place all our tiles on the board) we also gain the score of our opponent's tiles, and in the same way, if our opponent can make an exit based on our placed word, they gain the score of the tiles left in the rack. All of this is calculated by three cases:

Case 1: Your exit: (OUR WORD) + (OPPONENT'S TILES SCORE)

Case 2: Opponent exit: (OUR WORD) - (OPPONENT'S WORD + YOUR TILES SCORE)

Case 3: No-one exit: (OUR WORD) - (OPPONENT'S MAX SCORE)

RESULTS

Game Nr:	Wordfaid		Scrabaid
1	**411**		363
2	**501**		420
3	436		**448**
4	**529**		487
5	362		**432**
6	**426**		368
7	**425**		421
8	437		**472**
9	**392**		370
10	**415**		396
Total Wins:	**7**		3
Total Score:	**4334**		4177

Figure 4: Results from testing against the original Scrabaid.

After comparing our results to the regular Scrabble solver Scrabaid we could see that our algorithm were superior. The ten games we played with it resulted in us winning 70% of the games and finishing ahead with an average of almost 4% higher which is a notable amount. It was not hard to see that Wordfaid often had the upper hand due to the fact that it was always taking its opponent's possible moves into consideration and hence kept a healthy ratio between playing out high value words and protecting itself against retaliation from the opposing player.

WHY DID WORDFAID NOT WIN ALL GAMES?

Even though we would have wished that Wordfaid won all the games there is still the fact that there are random elements in Scrabble and Wordfeud that turns tides as they randomly favors different players. With that, the power of Wordfaid does not stretch further than playing the best possible moves based on the tiles at hand and hence if struck by a few rounds of bad luck it is hard to catch up since all the games are so close in score.

DISCUSSION

When we first started out we had a lot of ideas of how to optimize the program with several different methods such as: Are there certain letters that we would like to save? Do we want to make sure that we always have a certain amount of vowels left for our upcoming turn? Do we want to look out for multipliers? Little did we know that all these things that we had wanted to include as we started out would prove to be redundant to analyze individually since Wordfaid while calculating possible plays with random values now takes them all into consideration automatically.

One of the hardest parts of the project, which we struggled a lot with during the programming phase, was to limit our ambitions and goals to achieve a low running time. In our first discussions and plans we talked about making a 3-step analysis of the 50 most valuable words, which would have resulted in roughly 500 000 000 (50 words · 200 random racks for opponent ·100 random racks for AI · 500 average possibilities per word). This was, as we figured out early on, an impossible scenario. To solve this we had to lower the amount of comparisons in each step.

We did limit our program with a time budget of 100 seconds and the question one must ask oneself is: Is the time budget enough to provide decent results?
The easy answer here is that since we, as seen in our results, did win seven out of ten games against the original Scrabaid program and demonstrably it does therefore provide decent results. When the game starts out we do have 93 tiles in the bag that we want to randomize a tray from and the number of ways that we can randomize a tray out of those 93 tiles is close to ten billions. Does our, in comparison, small number of ten give us a good enough indication of our opponent's possible plays?

We would like to say that it does, even though it at most times fails to approximate the exact situation it does watch out for the extreme values that come from our opponent playing his tiles on multipliers. Since multipliers are the main way of getting high scores in Wordfeud they are also the most important thing to look out for while approximating upcoming turns and due to its design, Wordfaid does this fairly well.

Let us say that we had unlimited computational resources and could analyze every possibility for every steps ahead to the very end; How much better would Wordfaid be?
When the game starts out the first thing that one has to keep in mind here is that even if you have taken the average maximum scores out of ten billion possibilities and somewhere in your calculations have actually randomized the correct tray for your opponent it is still to you unknown which out of these ten billion possible trays that is the correct one. Of course this approximation would be a more accurate approximation of the game compared to only analyzing with ten random trays but how much better is it? Since it is all so random the main thing might not actually be to somewhere in your calculations have the correct random tray but rather that you make sure to block out certain parts of the board for your opponent. If we have a situation where you placing a word will open up the possibility for your opponent to play his word over a x3 word score square there will be a large amount of possible plays that he can make and still hit this x3 word score square. If we only hit one of these possible plays

Wordfaid will downrank the word we intended to play accordingly by comparing the opponent's possible score over the x3 to the value of the word we originally intended to play. Even though we did not calculate the right word for our opponent we still managed to avoid allowing him to outscore us massively during his next turn since we never opened up the possibility to play over the x3 word score square.

Another problem which we encountered was the issues of finding a good word list. The official word list of Wordfeud is not public, so we used a word list called ENABLE2K which we found out is used by many scrabble solvers. In the beginning we used Scrabaid's recommended word list, 2OF4BRIF, which was far from perfect. We compared these two word lists against each other in a real game and found out that ENABLE2K was superior every time as it, due to the larger amount of words, found words that 2OFBRIF did not have and hence returning far better options overall.

An issue still not resolved, which we felt forced to abandon, is the value of the blank tiles. These tiles can be used as any tile in the game and is supposed to be valued as 0. A solution to this, which we thought of in the beginning, was to use capital letters for these tiles to differentiate them from the regular ones. When starting to experiment with this, we found out that the larger part of how the program works in terms of patterns, hashes and other basics would have to be completely rewritten to serve this purpose. Wordfaid does however still calculate the score of the wildcards correctly while first placing them on the board.

One thing that has been both limited us and at the same time made Wordfaid possible is the Scrabaid software. If we would have had to create a solver from scratch the time needed would have far exceeded the time available to us and hence Wordfaid would without Scrabaid probably have stayed as a theoretical concept rather than an actual implementation. The apparent downside of using an already existing program not directly designed for our purpose was that we inherited the flaws and weaknesses of that program. This became painfully obvious to us while we were trying to manage the above mentioned blank tiles in a proper way and also since it was not designed to be run as many times as we wanted it to Wordfaid could probably have been way more efficient timewise with a different way of storing all information where we did not need to create as many instances of the board.

CONCLUSIONS

As shown in our results section Wordfaid did win seven out of ten times against the original software, Scrabaid, and this allows us to draw the conclusion that approximating the game a couple of turns ahead does prove to be a way better approach to winning the game rather than just trying to maximize your score every turn.

Another interesting thing we can conclude is the fact that you do not need a huge amount of randomized trays in order to get decent indications of how the game will play out over an upcoming couple of turns and the apparent question is now; why is it so?
As stated in our discussion we do at the start of the game only randomize ten trays out of the possible ten billion trays and yet the information given does provide valuable information. The reason for this is not that we somehow randomized the right trays out of all possibilities but it is rather the fact that we can block certain plays for our opponent. This is not done by finding out exactly which word the opponent can play but instead what parts of the board we should watch out for. If we are to open up the possibility for our opponent to play on a x3 word score square we don't have to find the exact word he intends to play but we will get a good enough indication if we randomize a single word that plays over the x3 word score square. The extreme value returned from this random instance will lower the value of the initial play we had in mind and hence warn us not to open up this play for our opponent and instead suggesting another option.

This is the reason why one can get good enough values even without randomizing the right trays for the opponent and that even with infinite amounts of computational power the results might not be that much better.

We have also noted the importance of handling the endgame phase properly. Since all the games were so close it was of the utmost importance that we analyzed two turns ahead in the end so that we could make sure that we were able to play out all our tiles and with that subtracting the values of our opponent's remaining tiles from his total.

The two main things that can be improved further on is the speed of the program and the way it handles blank tiles and if one were to improve these things, one would soon understand that the limitations of the original Scrabaid code would prove to be a major hindrance. One would probably be better off by creating a new program from scratch since in order to fix the two main limitations, which is the lack of proper blank tile handling and the way information is stored, one would have to rewrite the code entirely.

REFERENCES

1. National Scrabble Association, 2011. *History of Scrabble.* [online] Available at: <http://www.scrabble-assoc.com/info/history.html> [Accessed 10 Mars 2012].

2. Wordfeud, 2011. *Wordfeud help.* [online] Available at: <http://game01.wordfeud.com/wf/help/>[Accessed 11 Mars 2012]

3. Scrabaid, 2001. *Scrabaid - the essential scrabble solver.* [online] Available at: <http://scrabaid.sourceforge.net> [Accessed 15 Mars 2012]