Report

Assignment 3

Name: Yixi Rao

Lab: Fri15 L124_left 15:00-17:00 Alex Smith

Id: U6826541

Introduction

I designed my program to perform as an AI to play the Sushigo game, it will browse all the possible moves and finally to pick the best card, which will lead to winning.

Program design

To select the best card form hands, my choice or my core idea of this program is that finding all the possible situations of just one game and you will know which move will lead you to win. I divided the problem into four parts because each part can solve a single problem to let me get closer to the solution, and first three parts will act like a tool that I will use them together in the last part. The first part, I should generate all the possible moves of next state, the reason why I collect this is that all the game states of next state contain some very important information such as cards that a player has chosen. The second part, a tree is generated by the tool of part 1, the reasons why I create a tree are that it can enumerate all possible situations and it will build a fundamental structure of the whole program and lots of operation will be done on the tree. The third part, I have already built the tree, but searching all the nodes is a time-consuming work, so this is why I wrote the part 3 "prune the tree", which will omit some branches that we do not want when it tries to find the suitable value. The last part, I use the information or results came from all parts above, and I transform it into the card that I want from the suitable value that is the result of part 3.

On the Part 1 Function, in order to get the all possible moves of next state, my function `pickSushi` uses the case statement and where statement, because case statement can help me distinguish the different game states (finished, turn player), so I can do some specific operation respectively. And I use where statement, which can make my codes tidier and more understandable.

On the Part 2 function, I defined a group of function, which will change all the game states in the tree nodes to the number that stands for the winning or losing. The tree structure will be very useful because I will implement the minimax algorithm and the alpha-beta algorithm on it. Compared to the list, tree structure has the characteristic of the node and depth, which can arrange and organize the data of different state, not like the list or other data type, it is difficult to contain a different type of information. That is my main reason for choosing the tree structure as the basic skeleton.

On the part 3 function, to implement minimax algorithm and alpha-beta algorithm at the same time, I ingeniously made use of the lazy evaluation of Haskell to find the suitable value while omitting some branches, so the main idea of whole Part 3 is using lazy evaluation. In order to maximize and minimize, I defined the function `minimaList` to do it and the minimize function as well as the maximize are entirely symmetrical. I use where statement to contain three functions which can help me to omit the unnecessary branches and these three functions are related to each other. A Simplified description of these three functions functionality is that "assemble true value", "omit branches"," condition of omitting". And it will actually ignore some branches because of the feature of Haskell, which is when a value is evaluated, it will be used immediately without waiting rest of it. And it will work on my function of `decideOmit1` when the value is not satisfied with what I want, the branches attached to this value will be omitted. That is how my alpha-beta pruning works.

On the Part 4 function, to get the best card, I just need to interpret all the results came from Part 1, Part 2 and Part 3. And I implement this by writing three functions because it will make my code clearer and the reader will know what is happened.

Testing

On the part 1 function, I tested on it by putting some special states such as state contains Wasabi Nothing in cards and hands contain Nigiri, which will make sure whether it will return all the possible and correct cards after one player chooses one card, check the Nigiri whether can be attached to Wasabi or not. And I also use a lot of states which is the player 2 turn, which will check whether the

two hands will swap or not after player 2 turn. By the way, I write all possible states on paper in a simplified version, which will help me to collate it.

I tested the part 2 and part 3 together, this is correct and proper because if the function of part 3 alpha-beta algorithm is correct, then consequently, it will prove that I constructed the part 2 tree in a correct way. So, I tested my alpha-beta algorithm by sketching a tree first and map the algorithm to it to find the answer, on the process, I highlight which branches that I have omitted. After that, I decomposed my entire function into more small functions and added it to another file, then I put my tree on it, and passed the result to the next function until it is finished. The reason why I separate it is that it can help me to see which state of the function is wrong and I will easily to fix it. I also make sure whether it will prune some branches by writing the whole function steps on the paper followed by the Haskell rules.

For the Part 4 tests, I just gave the result of the test on Part 2 and 3 to the function to see whether it will return the card that I want. Besides, I wrote some initial game states, which is easy to predict what I should take to help me win the game, such as the cards just contain Nigiris. In addition, I designed a state of the opponent has already picked four dumplings, so it will check whether my function will prevent the opponent to collect five dumplings or not.

Beyond that, I wrote some tests on the AllTests.hs. For the part 2, I notice the property of the tree of its Node, which is all the nodes on each row or depth have their own value so we can extract the value by some helper function. This will be a decent material for testing. Not only can it test if the tree structure is built correct or not, but also can check the function that gets the score of a particular state. To test the alpha-beta pruning, I created a special tree form the lecture slides and the tree contains all the situations that the program will encounter such as some level one subtree will be omitted and some deeper depth branches will be omitted, and the proper value or path is in the last subtree of the level one depth. So, it will perfectly make sure my tree is built correctly. I simulate a whole Sushigo game by passing a sample state to the part 4 function and changing the hands manually based on the result of the function. So, it will make the test more dynamic, which will result in more convincing tests because it can show the processes of how it deals with a whole game.

I tested the entire program by entering the tournament, the result of it is I can win all the official bots almost all the time, which give me confidence, besides I played with my Ai either player 1 or player 2, to see whether it will make some weird moves or not, meanwhile I deliberately choose some cards to see whether it will make the wrong moves.

Reflection

My program is designed into four parts, and each part can solve a different but related problem, which will be propagated to the next part, however, I think the only thing can be changed in my program is the heuristic function, which I used is alpha-beta pruning. I can replace it by a smarter alpha-beta pruning algorithm or some other algorithms, which is changing the state of subtrees to both palyer1 and player2 have chosen their card in one turn rather than player 1 first and then player 2. The advantage of a smarter alpha-beta pruning is that it can reduce the branches of the tree, therefore, the pruning process will faster than the previous one so it can go through more possible moves. The disadvantage is that the minimax is hard to implement because we cannot minimize for the opponent moves.

I think I can write my heuristic function better, because actually my searching will only have time to search a limited depth of the tree, alternatively, I can develop a new algorithm, which is when it is your turn, you can sort the subtrees and place the maximum first. And vice versa for opponents (minimum), the reason why I think it is better is that it can be more efficient and just focus on the best move and never considering the worse moves.

Reference

Book of “Why Functional Programming Matters” John Hughes the University, Glasgow
http://worrydream.com/refs/Hughes-WhyFunctionalProgrammingMatters.pdf