Australian National University

ASSIGNMENT COVER SHEET

This coversheet should be filled electronically when possible and attached as the front page of your report in PDF format.

ANU College of Engineering and Computer Science
Australian National University
Canberra ACT 0200 Australia
www.anu.edu.au

Student ID For group assignments, list each student's ID	u6451847
Course Code	comp1100
Course Name	Programming as Problem Solving
Assignment number	3
Assignment Topic	ConnectX
Lecturer	Katya Lebedeva
Tutor	David O'Donohue
Tutorial (day and time)	Monday 15:00
Word count	1179
Due Date	Monday Week 12 (October 23), 9:00AM
Date Submitted	October 23
Extension Granted	

I declare that this work:

- ☑ upholds the principles of academic integrity, as defined in the University Academic Misconduct Rules;
- ☑ is original, except where collaboration (for example group work) has been authorised in writing by the course convener in the course outline and/or Wattle site;
- ☑ is produced for the purposes of this assessment task and has not been submitted for assessment in any other context, except where authorised in writing by the course convener;
- ☑ gives appropriate acknowledgement of the ideas, scholarship and intellectual property of others insofar as these have been used;
- ☑ in no part involves copying, cheating, collusion, fabrication, plagiarism or recycling.

Signatures

For group assignments, each student must sign.

Chucheng Qian

Assignment03- ConnectX

Name: Chucheng Qian ID: u6451847

1. Introduction

This report is written to explain my whole program which is the strategy for ConnectX game and some tests I used. The ConnectX game is a two-player and zero-sum connection game.It involves some strategies which help the player to win in a large percentage.

2. Reasons behind functions

First and foremost, I want to explain some reasons behind my main functions.

1.The function "heuristic".

This function is designed to make an evaluation of the situation of the current board.In the process of searching for a relatively most suitable column by looking ahead in a tree, I hope to use a function to evaluate the situation of some bottom nodes and then I can find a good node.

As this is a zero-sum game, the situation of the board comes from the total score I owned and the total score my opponent owned.When it is my turn, the best situation is the node which the differences between my score and the opponent's score are positive and largest, and for the opponent's turn, vice-versa.Therefore I the function like this.

2.The function "scoreList".

This function contains an overall function and two supported ones "maximise" and "minimise".

They all rely on one assumption that when it is my turn, I was trying to get the board which has the best situation among possible boards,and if it is my opponent's turn, he is trying to get the worst situation for me which is the best for him, in other words," minimizing my score".In the looking ahead process, I will get a minimax tree which has layers for me to maximise my scores which are nodes in the tree and for my opponent to minimise my scores.This is why I use minimax here to find a reasonable score for a possible move to help make decisions for my next move.

In addition, I use the alpha-beta pruning inside the "minimise" and "maximise" function.It is used to save processing time and improve my efficiency.I implement the alpha-beta pruning inside the minimax thus it can work along with minimax process and cut the branches which do not satisfy the requirements when I was searching downwards and selecting upwards through the tree.

3.The function "findIndex"

As I elaborated before, my function "scoreList" will output a score. This function is designed to use built-in function "map" to achieve a score list and then zip the index which represents the number of the available columns and the score list together as a tuple.Therefore i can use the function "findMax" to find the index of the largest score.

3.Efficiency of your program

The efficiency of this program comes from the use of Alpha-Beta pruning.It depends on alpha which represents the smallest and beta which represents the largest.If alpha is larger than beta for branch A when finishing searching one lower branch in branch A, it means other branches in A does not satisfy the situation and they can be ignored due to that whatever they are, they cannot influence the value of node A.The utilisation of this saves the time when searching the unnecessary branches and improves efficiency.

The amount of LookAhead my program can handle is from 3 to 5, and it depends on the complexity of the following situations.

4.Potential improvements

In the "makeMove" function, I set a mechanism that if the current board is empty, and I am the first player, the first discs will be put in the middle column.It is from the YouTube video from Numberphile which is illustrated on the assignment page and this is a solution for this game which can ensure one to win if the following moves are all considered thoroughly.

However, on one hand, if the grid has even number columns, the first move will not be put in the exact middle column and is similar to a random move.Therefore I would like to add a situation that if the grid has even number columns, it follows the regular evaluating way, and if it doesn't, the first move will select the middle column.Consequently, the program will be more precise when applied to different grids.

On the other hand, although it is a winning mechanism, my program may not get enough LookAhead to check through all possible final outcomes in the games, which means the program may fail to consider following moves thoroughly.Therefore, this winning mechanism may be a bit of useless.If I get enough time, I will check if the games work better when this mechanism is used by comparing to which is not being used for various grids.Then I can make a decision whether I will use it, or design another method to make the program faster and more practical.

5. Solving Problems

During the process that I was designing this game, some conceptual and technical problems show up.

One of the most important is that at first, I want to get all bottom nodes and then use minimax to search upwards.Then I found it is difficult as the depth of tree branches are different.

Then I try use minimax and alpha-beta pruning together to search one branch first and then the next in the same layer.If there is no need to search this branch, just ignore it and output the corresponding value.

6. Testing

The method i used to test my program is doctest and my program has passed these tests.

```
-- | ConnectX
--
-- >>> heuristic Board { board = [[Blue,Red,Empty,Empty],
[Blue,Red,Empty,Empty],[Blue,Blue,Red,Empty],[Red,Red,Blue,Empty],
[Red,Blue,Red,Empty]] ,blueScore = 6, redScore = 3,turn = BlueBot, dimension
= (5, 4), connect = 4} BlueBot
-- 3
--
-- >>> findIndex Board { board = [[Blue,Red],[Blue,Red],[Blue,Blue,Red],
[Red,Red,Blue],[Red,Blue,Red]] ,blueScore = 6, redScore = 3,turn = BlueBot,
dimension = (5, 4), connect = 4} 1 0
-- [(1,0),(2,0),(3,12),(4,12),(5,56)]
--
-- >>> findMax [(1,0),(2,0),(3,12),(4,88),(5,56)]
-- 4
--
-- >>> scoreList 1 BlueBot 0 (-1000,1000) Board { board = [[Blue,Red],
[Blue,Red],[Blue,Blue,Red],[Red,Red,Blue],[Red,Blue,Red]] ,blueScore = 6,
redScore = 3,turn = BlueBot, dimension = (5, 4), connect = 4}
-- 24
--
-- >>> makeMove Board { board = [[Blue,Red],[Blue,Red],[Blue,Blue,Red],
[Red,Red,Blue],[Red,Blue,Red]] ,blueScore = 6, redScore = 3,turn = BlueBot,
dimension = (5, 4), connect = 4} 1
-- 5
```

As the program contains several functions, I just want to test some important and comprehensive ones.

For function "heuristic", I want to test whether it outputs current situation for BlueBot which is the value that bluescore minus redscore.

For function "findIndex", I want to test whether it will output a correct list of tuples which contains the possible score and the index for it.

For functions "findMax" and "makeover", I want to test whether the Alpha-Beta pruning and minimax work correctly and whether the whole program will output the index which represents the best column to put in a blue discus.

7. Possible confusing part

A possible confusing part in my program may be the function "minimise" and "maximise".

Considering it is at the bottom of a tree branch, this branch has three nodes, and these tree nodes make up a list which is the possible input of these two functions.These function will recur the list and find the maximum or the minimum.In function "maximise", when the value of the node is larger than current alpha, it will be the new alpha and the function continues.For "minimise", it is same.

However, at the very beginning, i read the notes for alpha-beta pruning, I simply put a condition "alpha1>= beta1".Then I found it may be the same as the "case" I set when the current value of nodes larger than beta in "maximise" function.I retain these two "case" but am not very sure about that.

8. Conclusion

I designed this program depend on some assumptions and my understanding for Alpha-Beta pruning.The program may also fail when to meet some special case. I will keep working on it to find out how to write it more efficiently.