Australian National University

ASSIGNMENT COVER SHEET

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

+61 2 6125 5254

Student ID	U6611178		
Name	Danny Feng		
Course Code & Name	COMP1100 - Programming as Problem Solving		
Assignment Topic	Technical Report for Assignment 3 Backgammon Bot		
Lab Time	Monday 16:00 – 18:00		
Tutor	Jack Kelly <jack@jackkelly.name>		
Word count		Due Date	25 May 2018
Last Edited	25 May 2018	Extension Granted	

I declare that this work:

- ☐ upholds the principles of academic integrity, as defined in the ANU Policy: Code of Practice for Student Academic Integrity;
- ☐ 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.

Initials DF

For group assignments, each student must initial.

Technical Report for Assignment 2

Danny Feng

1. Introduction

This assignment is to implement A.I algorithm into the game Backgammon. In this report, I'm going to outlining not only the ideas, structure and solutions of my Backgammon Bot but also sharing my developing process includes the design decision and things I improved.

2. Reflection

Before I started doing it, I spent some time reading instructions, exploring the game rules and analyzing the given program function. I started the game one week after it released, and I successfully finished 1) my legal move bot that can pick a random position to move, 2) my greedy bot V1 that picks a best move based on position indexes from current state, 3) my greedy bot V2 that picks a best move based on scoring from all the current legal move, it's updated version 3 that fixed a move matching bug using a slightly different method, 4) my minimax bot V1 that inspired from lecturer's code and 5) my minimax bot V2 that looks into the future state without using lecturer's structure.

I used bottom-up design strategies when I was developing. When it came a problem, I broke it to several small function and then built an overall function that calls them. For example, function scoreBeenEaten, scoreEat, scoreGoodMove and scorePoint are to calculates the scores for different parts of the state. Then they are called by scoreState to give out a final score of the state they evaluate.

I also tried to do the expectiminimax with probability involved in the minimax. But since I started the assignment a little bit late, I haven't finished it. This is a lesson for me to start the assignment as earlier as possible in my future course.

with StateminimaxV2 calculates the score for each level and it is been called by rootV2 to generate a best move.

3. Heuristics

About my evaluation of a given board. I gave the priority to the (bPips – wPips), I wanted this value the greater the better. Pips is the total pieces distance from home. I found out that given a high weight of this result in a stronger bot that is eager to widen differences between its opponent. Besides, based on many tests I ran, I found out that if I make the bot prefer to move the piece to the point which only has one friendly piece on it then the win rate will increase. So

I made my heuristics check Just(White,1) and Just (Black,1) that focus on blot and block more. What's more, I also made my bot prefer to move its furthest piece to make my bot move all pieces back home faster.

The search method is a topic I really want to share. In my first greedy bot, I didn't use evaluation things at all! You might think that is not a good idea. But I will say, it is actually good for the greedy algorithm but bad for the later task like minimax. Firstly, it is efficient. My greedy bot V1 only check what properties that current state has. If there is a friendly piece alone on the board, it just picks a legal move that can go there. If not, it will check if there is an enemy piece alone, if not, it will just move the furthest piece. Compared to my later greedy bots that use a scoring idea that everybody has, it doesn't need to perform every legal move to evaluate all the possible states. Therefore, it only takes half of the memory as V2. Secondly, I used this version to join the tournament and the result is quite good.

```
*PlayAsWhite> greedyBotV1 (stateAfterDiceRoll (initialState White) (4,3))
[(24,4),(24,3)]
(0.01 secs, 1,445,160 bytes)
```

```
*PlayAsWhite> greedyBotV2 (stateAfterDiceRoll (initialState White) (4,3))
[(24,4),(24,3)]
(0.01 secs, 2,776,968 bytes)
```

4. A.I Performance

The pictures below are my A.I performance in the tournament. I uploaded my greedyBot V1 on 21st May 2018 and these were the result at that time. My tournament ID is 1980, it ranked 15 with a win ratio 0.771. I opted out after that to update my bot to a next intelligence level. To be honest, I don't think my greedy Bot V1 is not that strong. The reason of its success at that point of time was mainly that most of the students just uploaded their legal move bot.

Player Details: 1980

Back to Player Ranking and Reports

Student ID	Status	Errors	Points	Completed Matches	Running Matches	Score Difference	Win Ratio	Last Submission
1980	N/A	N/A			N/A	N/A	0.771	N/A

Opponent	Result	Score (player - opponent)
1001	W	3 - 0
1379	L	0 - 3
1440	W	3 - 0
1449	W	3 - 1
1600	L	0 - 3
1689	L	1 - 3
1805	W	3 - 0
1900	W	3 - 1
2203	W	3 - 0
2308	W	3 - 1
2309	W	3 - 0
2669	L	0 - 3
2715	W	3 - 0
2802	W	3 - 2
2841	L	1 - 3
2881	L	2 - 3
2939	L	0 - 3
3020	W	3 - 0
3289	W	3 - 1
3329	L	1 - 3
3692	W	3 - 0
3909	L	1 - 3
3983	W	3 - 2
4150	W	3 - 2
4183	L	1 - 3
4315	L	0 - 3
4388	L	2 - 3
4449	L	1 - 3
4761	W	3 - 1
4776	W	3 - 2
4983	W	3 - 0
5071	W	3 - 1
5091	W	3 - 0
7032	W	3 - 2
7890	W	3 - 0

Updated: 2018-05-21 11:33:09.215396

Player Ranking

All Matches

Match Count: 834

Matches Remaining: N/A

Statistics: N/A

Rank	ID	Status	Errors	Points	Matches	Running	Score Sum	Win Ratio	Last Submission
1	3909	N/A	N/A	52	35	N/A	N/A	1.486	N/A
2	5071	N/A	N/A	47	35	N/A	N/A	1.343	N/A
3	4449	N/A	N/A	44	35	N/A	N/A	1.257	N/A
4	4150	N/A	N/A	42	35	N/A	N/A	1.2	N/A
5	1689	N/A	N/A	40	35	N/A	N/A	1.143	N/A
6	4315	N/A	N/A	37	35	N/A	N/A	1.057	N/A
7	1379	N/A	N/A	36	35	N/A	N/A	1.029	N/A
8	1900	N/A	N/A	33	35	N/A	N/A	0.943	N/A
9	2939	N/A	N/A	32	35	N/A	N/A	0.914	N/A
10	2669	N/A	N/A	31	35	N/A	N/A	0.886	N/A
11	1449	N/A	N/A	30	35	N/A	N/A	0.857	N/A
12	2841	N/A	N/A	30	35	N/A	N/A	0.857	N/A
13	4776	N/A	N/A	29	35	N/A	N/A	0.829	N/A
14	1600	N/A	N/A	28	35	N/A	N/A	0.8	N/A
15	1980	N/A	N/A	27	35	N/A	N/A	0.771	N/A
16	3329	N/A	N/A	27	35	N/A	N/A	0.771	N/A
17	3289	N/A	N/A	26	35	N/A	N/A	0.743	N/A
18	2802	N/A	N/A	25	35	N/A	N/A	0.714	N/A
19	2715	N/A	N/A	23	35	N/A	N/A	0.657	N/A

5. Issues Encountered, Solutions, and Potential Improvement

I let greedy bot V1 V2 V3, minimax bot V1 V2 and legal move bot played against each other. The result is somehow confusing but interesting. When they fight against legal move bot, greedy V3 is always the strongest, followed by greedy V1. However, when greedy V1 and greedy V3 fight together, greedy V1 is often a little bit better than greedy V3. I think this is

because pips strategies will not become that strong when an opponent always like to blot and block. Besides, when fighting with legal move bot, my greedy bots performance are better than minimax bots. I think this is because the minimax algorithm always assumes that its opponent will always make a best move. But legal move does not.