Backgammon with Variable Luck

Abstract **–**

Context/Background

Creating successful backgammon AI opponents has been an interesting point in computer science for many years now. Due to the rules of the game and the inclusion of dice rolls, backgammon has a very high branching factor when constructing search trees of possible moves. This has led to many different approaches being tried in order to create AI opponents that can play at a higher level.

Aims

This project aims to make a backgammon game that is capable of being played in real time, and which the user can alter the difficulty of the game by changing the luck of the AI opponent. The system should also facilitate the ability for users to store their profiles and rankings, said rankings being altered by playing games against differing difficulty AI opponents.

Method

Users will interact with the game through a GUI, the game data will then be used to create a search tree of possible moves. An appropriate search algorithm will then use the tree to derive the best possible move to take, further into development this analysis will be used to influence the luck of dice rolls for the AI opponent, allowing the AI opponent to play at a varying level of difficulty. A state-transition evaluation function will dominate the AI strategy; I will experiment with this function with the aim of creating an AI opponent that appears to play in a human, like manner. By altering the depth of the search tree, possible strategies will be scored by the score potential of their sub trees.

Proposed Solution

Implement a backgammon game engine that will validate moves and facilitate user input through a GUI. The system should allow users to create profiles and store their preferences and game history. The AI will be implemented and focused around searching a tree of possible valid moves; I expect to try variations of the Mini-Max algorithm.

A good evaluation function will be essential to providing a good heuristic for possible moves, and should take into account various elements such as checker positions, baring off, taking checkers and blocking moves. Because of the very high branching factor of Backgammon, pruning will be essential to reduce the number of moves evaluated.

Keywords – Backgammon, variable luck, Mini-Max, Alpha-Beta pruning, AI, search tree, search algorithm, heuristic.

I. INTRODUCTION

Backgammon as a game

Backgammon is played on a 24-position (known as points) board, with two sides each having 12 points on each [10]. Each point is connected around the edge of the board, and players take turns to move their checkers in opposing directions as shown in Figure 1.

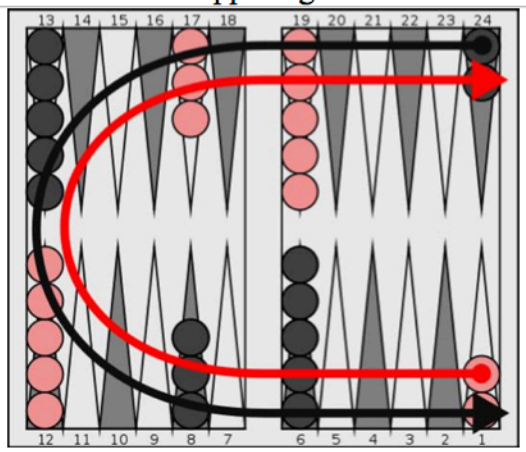

Figure 1. http://en.wikipedia.org/wiki/Backgammon#Rules The red player aims to move their checkers to positions 19-24 as this is their home board, whereas the black player aims to move their checkers to positions 6-1.

Players move by first rolling a pair of dice, the two values rolled can then be used to move their checkers. For example a roll of 5-4, could be used to move one checker forward by 9 places, or equally two different checkers by 5 and 4 places respectively. Additionally if a player rolls a double, this is interpreted as four moves, so rolling double 6 gives four moves of 6, which can be used in any legal combination.

A legal move is any move that uses all values of the dice rolled. Checkers can be moved to an empty position, a position currently occupied by one or more of the same colour checker or a position that contains exactly one checker of the opposing colour. In the case of the latter, the single checker of the opposing colour is placed on the bar, and the moving piece is placed at the position. The bar is placed in the middle of the board, between positions 18-19, and 6-7. If a player has any pieces on the bar, they must be moved off the bar before any other moves can be made. Pieces are moved off the bar by re-entering through the respective player's home board. A roll of 3 would allow the red/white player to enter a checker at position 3, or 22 for the black player.

Once all of a players' pieces reside in their respective home board, they may begin to bear off their checkers; the player who bears off all their checkers first wins the game. Checkers are beared off by rolling the exact value needed for that position, for example a 5 would be needed for bearing off from position 5 or 20. A greater dice roll can be used if no checkers reside in the exact position, so a dice roll of 5 could be used to bear off a checker in position 4 if and only if no checkers reside in positions 5 or 6.

Although there exists a doubling cube in Backgammon, I will not be including this in my project in order to reduce the complexity of the possible strategies.

Purpose of the Project

The purpose of my project is to investigate the performance of search tree algorithms with Backgammon, and to experiment with an evaluation function to aid a more competent level of gameplay by the AI. Another aim of my project is to include a variance in difficulty of gameplay by the AI through variable luck. Through analysis of the game search tree, I hope to be able to influence and alter the luck of the AI player.

Hauk, T., Buro, M., Schaeffer, J (2004) [2] looks at the performance of Minimax algorithms when applied to Backgammon, and investigates using a variation of the Minimax algorithm, Expectimax, to gain better results. Due to the branching factor of over 800 [1], a Minimax orientated implementation will need experimented with to achieve better results. I will focus on varying the depth of my project searches and improving the state evaluation function to improve the analysis of a move. Hopefully these two aspects will allow me to develop a better Backgammon AI.

Other approaches

There have been many approaches to developing better backgammon AI opponents, recently the use of artificial-neural-networks has been of significant interest. GNU-Backgammon [7] is one of the most well know variants, it uses a neural network to improve its gameplay versus a player. Neural networks have the ability to learn and recognize patterns by observation, creating links between artificial nodes, imitating a biological neural network.

Another example is TD-Gammon [8], a self-teaching neural network. By the use of reinforcement learning, TD-Gammon is able to learn based on the results of playing itself. As opposed to supervised learning, reinforcement learning uses a 'reward' signifying the outputs quality [9].

Variable luck

"In the short run, just about anyone can beat anyone given enough luck, and when you have dice, you have luck." [6]. Due to the real time requirement of my project, a more extensive analysis of possible moves is less possible. Because of this, at a given point a human player could find the game unchallenging, to combat this I will use variable luck to increase or decrease the apparent difficulty of the game, as required.

Aims and Deliverables

- The project will require a simple graphic user interface for the user to interact with, and to allow the user to change preferences.
- A basic game engine will be needed to parse moves inputted by the user, and for the AI to validate moves against legal/valid moves.
- An appropriate data-structure/combination of data-structures will be needed to allow the game to be stored efficiently, and allow analysis to be performed in an easy manner.
- The system will need to facilitate multiple users, with individual preferences and

game histories, allowing a ranking to be generated using their game history.

- A basic AI will be needed, then through development this should be increased to a more competent AI through experimenting with different search algorithms and methods, as well as developing a better evaluation function.
- Methods to implement variable luck to change the difficulty of the AI player.
- Various performance and efficiency measures will be needed to ensure the game is playable in real-time, i.e. the AI move should be made within twenty seconds.

II. Design

System architecture

The system will be developed in Java; I have chosen this language as it is the most suitable high level language that I am most familiar with. The system will be object orientated with multiple classes where applicable and multithreading may be used, if the warranted performance increases are required.

Java also has the benefit of being cross platform compatible through the use of .JAR's, and is much less OS dependent than other languages. Java also has a wide range of available libraries that will aid development.

Maintenance

Although good software-engineering principles are important, due to the timeframe of my project and the un-certainty of future development; my main focus will be on pursuing results, which may be at the cost of good software-engineering practices. However, when possible, good practices will be used with the aim of if the code is passed on/further developed it should be in a good enough state to allow this.

Testing & Evaluation

I plan to iteratively develop my project, with the aim of fulfilling the requirement of my project throughout the development. Testing will be done at the end of each requirement, and time for this has been allocated (see project plan for testing allocations). Also, throughout my project I will be submitting the game in various usable states for game testing versus human opponents, using the feedback to improve gameplay.

Requirements

ID	Description	Priority
1	Implement a basic board data structure and game engine	High
2	Implement a basic GUI that a user can use to play Backgammon	High
3	Implement a basic AI that will play a valid game	High
4	Implement a user profile system with game rankings	High
5	Improve the GUI to improve usability	Medium

6	Improve the AI, to play at a higher level	Medium
7	Implement of variable luck	Medium

1.Implement a basic board data structure and game engine

The way in which the board is represented in my system is very important. It has to allow easy analysis, and be broadly accessible without the need of a complex interface. Efficiency is also very important as the virtual board will eventually be used to create large tree structures in the later AI developments.

For these reasons I have chosen to use a 2D Java array, where columns will refer to board positions, and rows refer to colour. The use of an array will make representation through the GUI easier as individual positions can be addressed easily.

A game engine will validate moves through a set of validation clauses and if valid, will allow a move to be implemented. Due to the complexity of Backgammon the move validation part of the game engine will be essential to a successful development. The performance of the validation, will be imperative to allowing a real time analysis of the game.

2&5. Implement a basic GUI that a user can use to play Backgammon/ Improve the GUI to improve usability

Developing a graphical user interface will provide a more elegant way of interacting with the game. Although the same capabilities could be provided through a command line interface, a GUI will provide a more intuitive and less frustrating way of interacting with the game.

The GUI will facilitate not only the actual game play, but provide access to the user profile system, preferences and instructions on how to play the game as seen in Figure 2.

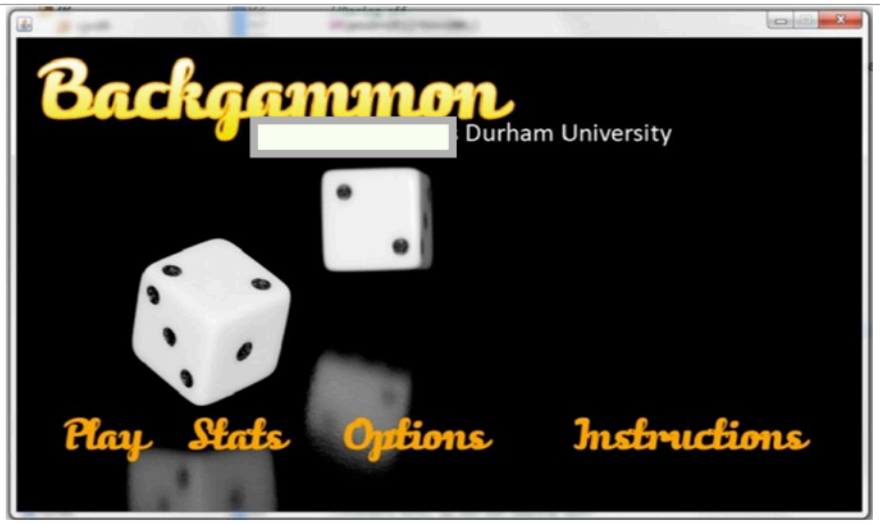

Figure 2. The prototype design, of the GUI main menu and the four main options of starting a new game, user stats/rankings, options (preferences) and game instructions.

The actual gameplay will be represented through a visual board as shown in Figure 3. To make the GUI as intuitive as possible, when a checker has been selected it is highlighted. The GUI will also need to inform the user if there are no more available moves.

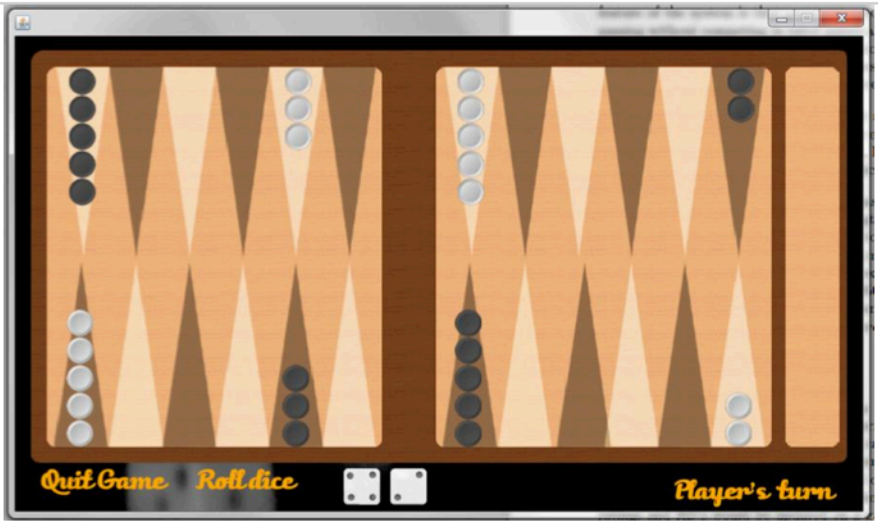

Figure 3. The board representation in the GUI. In the bottom right you can see whose turn it is to move as well the options to roll the dice, and quit the current game.

Due to my previous experience, I will be using Java's AWT platform to develop GUI for the system. AWT provides a wide range of tools to enable writing a capable GUI.

3&6. Implement a basic AI that will play a valid game/ Improve the AI, to play at a higher level

The proposed system is orientated mainly around developing an effective AI that plays backgammon. As such this is a very important requirement, as without an effective AI, development of the variable-luck-AI will be impossible. The requirements for the AI opponent is that it must play a valid move in less than 30 seconds (in order for the game to be playable in real-time). It also must pick what it considers to be a good move; this will be derived by the evaluation function.

The evaluation function will be progressive development across the development of all three levels of AI opponent (basic, improved and variable luck). The purpose of the evaluation function is to provide a heuristic for a single valid move (ie the use of a single dice roll). Hence the following scenarios should be considered:

- Sending an opposing checker to the bar
- Re-entering your own checker from the bar
- Fortifying a lone checker to avoid it being sent to the bar
- Bearing off a checker
- Moving alone

Although these scenarios alone will not provide a comprehensive heuristic as the following factors should also be considered:

- Position of a checker if alone, to any opposition checkers.
- The position of the checker being moved, checkers in the away board should be moved first
- Unnecessary moving of checkers in the home board.

From the above scenarios and factors I hope to be able to deliver a good heuristic, this heuristic will be used to judge a complete move (all dice rolls used).

Example values for evaluation function

Scenario/factor													
Sending an opposing checker to the bar							Value						
							+50						
Re-entering your own checker from the bar							+10						
Fortifying a lone checker							+40						
Bearing off a checker							+100						
Moving alone							-10						
Distance to opponent checkers (all values added)	Distance	1	2	3	4	5	6	7	8	9	10	11	12
	Value	-1	-1	-1	-2	-3	-4	-5	-4	-3	-2	-1	-1
The position of the checker being moved							+distance to home board						
Unnecessary moving of checkers in the home board.							-20						

Through development and experimentation I will change these values, as these values will alter the strategy of the AI opponent, for example setting the value of "sending an opposing checker to the bar" to +200 would almost certainly make that move the most valuable, regardless of any penalties such as moving alone and the distance to opponent checkers.

Although not ideal, the Mini-Max algorithm is a good basis for a solution. *"The Mini-Max algorithm is designed to determine the optimal strategy for MAX"* [3]. The Mini-Max algorithm models two players as max and min, playing on behalf of max, in the aim to find an optimal move. Under the premise that min will always play the move to their best advantage in order to minimise the potential gain of max. As shown in Figure 4 a search tree is generated of all the possible moves, each depth represents an alternate move by max or min. Max is first to move from the state A, the available moves are B1, B2 and B3. From these moves, min has the potential to move to C1, C2 and C3 from B1, C4, C5 and C6 from B2 and so on.

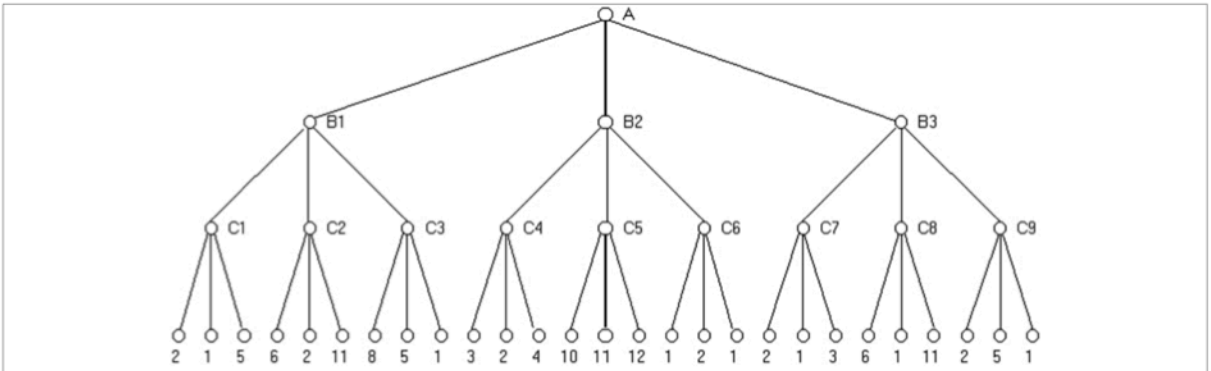

Figure 4. http://www.stanford.edu/~msirota/soco/minimax.jpg A search tree as generated by the Mini-Max Algorithm

As you can see from Figure 4 there are four levels (complete moves) of the tree (depth 4 or 4ply). The algorithm aims to build a path through the search tree by picking appropriate nodes; if the level is of max pick the highest value node (assumes max will pick the best move available), where is the level is min; pick the lowest value node (assuming min will pick the move optimal for themselves).

As mentioned above, the evaluation function will be used to provide a heuristic for the node value. Although ideally we would search for a terminal state (ie that of which max wins the game), this would for the most part of the game be far too deep in the search tree to consider. Therefor we must pick the best move for the prescribed depth.

For requirement 3; the basic AI, the AI will evaluate all moves currently available to max (the AI opponent) for the given dice roll. Even with a branching factor that can reach 4000 [3], all the potential moves can be calculated and evaluated in a very small time. The best move will then be picked from the generated (valid) moves. Although this will not consider the implications of the following move from min, this will provide a basic AI that is playable.

The reason to search to depths such as 4ply, is to form a basic strategy. If we only searched the immediate moves available and picked the best scoring node, this could lead to a situation that benefits the opponent hugely. It would be far better if we chose a lesser scoring node that does not give the opponent such a big benefit. Although we still have this problem when we cut-off the search, at a given depth, by re-evaluating the search tree on every AI, move we can avoid these situations.

Later in development, I aim to search to a deeper depth in the Mini-Max search tree, but with a complexity of $O(b^d)$ where b is the branching factor and d the depth, the full evaluation of 4 ply (depth 4) would result in $2.56 * 10^{10}$ nodes (using an average branching factor of 400). Even evaluating one million nodes a second would result in an evaluation taking over 7 million hours. Therefore it is imperative to implement pruning, for example the alpha-beta pruning algorithm is guaranteed to find the same solution as the Mini-Max algorithm, yet it has been proven to allow the increase of the search depth by a factor of $\frac{\log n}{\log \approx \frac{4}{3}}$ [4]. Yet this still would not yield a sufficient enough reduction to achieve 4 ply. Therefore, a probabilistic selection of dice rolls used to expand sub trees should be included, instead of all 21 dice rolls. Through experimenting with pruning and probabilistic dice-roll-selection, depths such as 4-ply, should be more achievable.

4. Implement a user profile system with game rankings

One feature of my system is to allow multiple users to register and create profiles within the game. Although it is possible to play the game without logging in, if the user is logged in their performance will be measured against the AI opponent, and a game ranking will be generated.

An appropriate solution is the Elo rating system [5]. Given an average rating, the Elo rating system will modify the current users rating based on their performance versus another players rating. If we fix the other players (AI opponent)'s score, we can still use this system to update a players rankings. The AI opponents score (although fixed) will be based on their difficulty. For example the scores of 800, 1000 and 1200 for; easy, medium and hard difficulty AI opponents, respectively.

After assigning a fixed k-factor , the Elo system works as:

$$R'_A = R_A + K(S_A - E_A)$$

Where E_A is calculated using:

$$E_A = \frac{1}{1 + 10(R_B - R_A)/400}$$

In this algorithm, R'_A is the new calculated rating of the player, and R_A is the existing rating. S_A is the actual result of the game, a win for the player = 1, a loss = 0. Finally R_B is the rating of the AI opponent as mentioned above.

Every time a user plays a game, the score is recorded and their rating updated. The user will be able to see their game history and ranking versus other players in the GUI menu (2&5).

7. Implement of variable luck

The implementation of variable luck will see the change of how dice rolls are performed. Until now a dice roll has been picked at random, and then the AI opponent has picked the best move from that. In short my implementation of variable luck will see all dice rolls evaluated to a determined depth, based on the best move available per that dice roll, the chances of that dice roll being picked (and said optimal move therefore also being picked) will be altered.

Due to the real-time constraints imposed on my system (moves to be played in less than 30 seconds by AI), there will be a reduction in the search depth per dice roll, compared to searching a single dice roll as before. I expect searching to 2-ply with all 21 dice rolls will be achievable. From the 2-ply dice roll trees, a single move will be picked per dice roll, this move will be picked by the combination of the evaluation heuristic, and the potential for min (calculated at depth 1). Once all dice rolls have a best move and a heuristic for that move, the weightings for that particular dice roll, will be changed accordingly (the better performance of the dice roll the more chance of it being picked).

This will be implemented through a large array. For example an array with 21 million rows with the initial natural weighting per dice roll will give each dice roll 1 million rows. Picking a row at random to select the dice roll will give the equal probability of 1/21. But, after all best moves are calculated, according to their respective scores the percentage of rows assigned to said dice roll will be altered as shown in Figure 5.

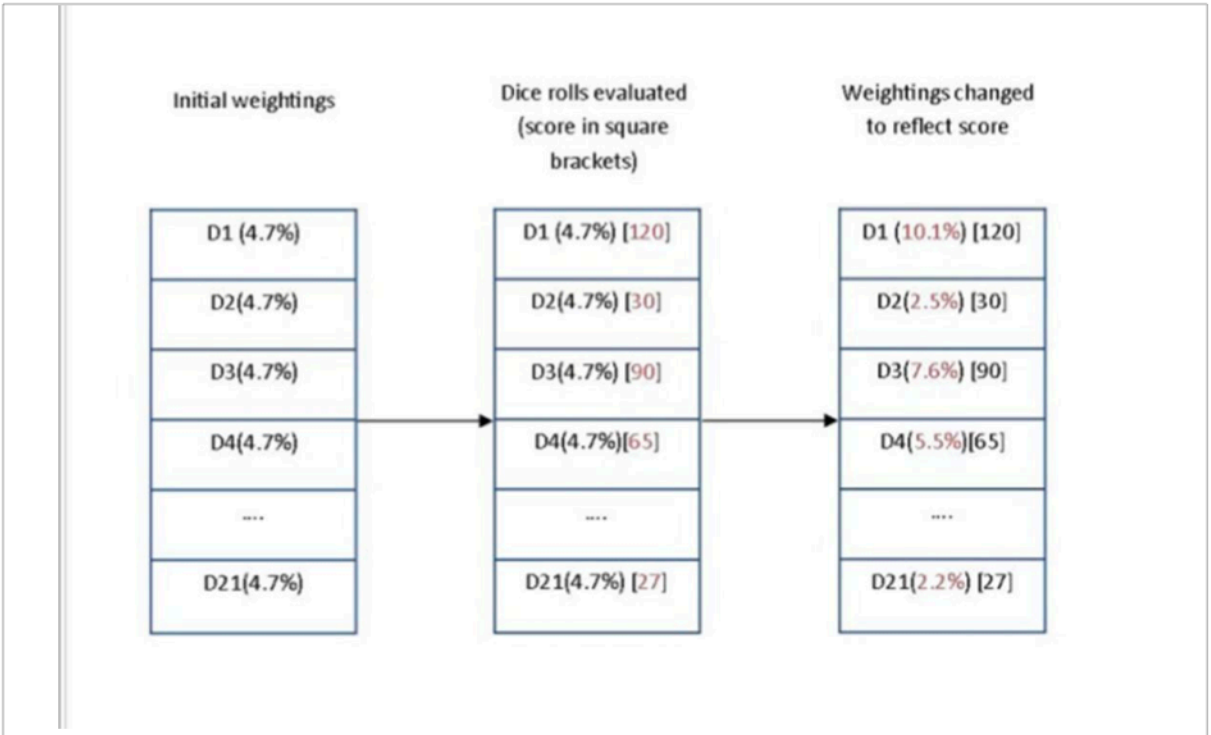

Figure 5. Shifting the bounds of the dice rolls probability after calculating the best move per dice roll (percentages only for illustration). As you can see D1 and D3 are more favorable, so their probabilities of being picked increase.

After the new percentages are calculated, the bounds in the array are changed, so for example in figure 5 D1 would now have 2,121,000 rows, over double it's initial allocation. The array will be shuffled to give a more random distribution of the rows, then a row will be selected at random, hence choosing the dice roll and move the AI will play.

The percentages will be calculated like so:

- Initial percentages for a dice roll will be the percentage of that dice rolls score, out of the total of all dice roll scores.
- If a hard difficulty is selected lower performing dice roll shares, will be decremented and higher performing dice roll shares incremented. If a lower difficulty is selected lower performing dice roll shares, will be incremented and higher performing dice roll shared decremented.

The variable luck will be implemented through changing in what manner the bounds are changed by, hence changing the difficulty of the game played by the AI opponent. Selecting an easier difficulty will make the AI more probable to roll worse dice rolls, where a harder difficulty will increase the probability of rolling good dice rolls. An example can be seen in figure 6.

1	2	3	4	5	6	7	8	9	10	TOTAL	
10	10	10	10	10	10	10	10	10	10	100	Original share
120	30	90	65	27	45	27	90	46	50	590	Score
0.20	0.05	0.15	0.11	0.05	0.08	0.05	0.15	0.08	0.08	1.00	Score performance (% of total score)
20.34	5.08	15.25	11.02	4.58	7.63	4.58	15.25	7.80	8.47	100.00	New share (no augmentation)
15.25	7.12	12.20	12.12	6.41	8.76	6.41	12.20	9.36	10.17	100.00	Example share (low difficulty)
24.41	4.07	17.54	12.67	3.20	5.95	3.23	17.54	5.46	5.93	100.00	Example share (high difficulty)

Figure 6. An example of how dice rolls probabilities can be changed in regards to how well they perform. Note that the example shares for low and high difficulties are only for illustration. Red indicates shares having decreased probability, green indicates increased probability.

REFERENCES

[1] Berliner H. J. "BKG : a program that plays backgammon" (1977)
[2] Hauk, T., Buro, M., Schaeffer, J. "*–minimax performance in backgammon. In: Proceedings of the Computers and Games Conference" (2004)
[3] Norvig P., Russell S., Artificial Intelligence: A Modern Approach (1995)
[4] Douglas McIllroy M., " (1982) "The Solution for the Branching Factor of the Alpha-Beta Pruning Algorithm and its Optimality
[5] Anderson , R. Formulating An Elo Rating for Major League Soccer (2011), last accessed 2nd January 2013 http://mlselo.f2f2s.com/MLS_Elo.pdf
[6] Simborg , P. Luck vs. Skill in Backgammon (2006), last accessed 23rd January 2013 http://www.bkgm.com/articles/Simborg/LuckVsSkill/index.html
[7] All about GNU Backgammon http://www.gnu.org/software/gnubg/manual/allabout.pdf
[8]Tesauro, G. TD-Gammon, a Self-Teaching Backgammon Program,
Achieves Master-Level Play (1994)
[9] Sutton, R. S., Barto, A. G., Reinforcement Learning: An Introduction (1998)
[10] http://www.backgammongames.co.uk/backgammon-rules.htm, last accessed 23rd January 2013