Approaching The Royal Game of Ur with Genetic Algorithms and ExpectiMax

Tang, Marco Kwan Ho (20306981) Tse, Wai Ho (20355528)
Zhao, Vincent Ruidong (20233835) Yap, Alistair Yun Hee (20306450)

Introduction

AI's for games are typically written using heuristics, such as maximizing surviving friendly pieces while minimizing enemy pieces for chess. However, heuristics are often difficult to formulate and also limit the potential for the innovation of new methods (or in the case of games, tactics). To address this, many have switched their attention to neural networks which have demonstrated great potential in recent years.

In order to evaluate the effectiveness of neural networks ourselves, we chose to create an AI to play the 'Royal Game of Ur' - a simple board game with complex underlying strategies.

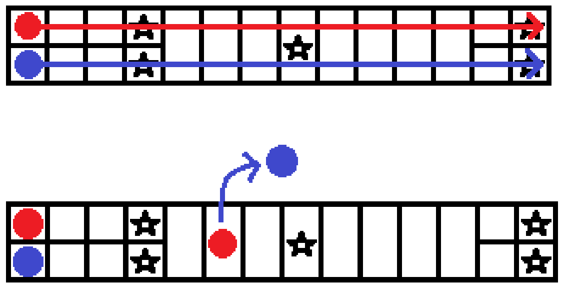


Figure 1. Diagram indicating basic rules of Ur

Problem Definition

There are many different and varying rulesets for the Royal Game of Ur, each with their unique quirks and strategies. For this project, we used the British Museum's simplified ruleset.

The objective is to bring all 7 of your pieces from one end of the board to the other before your opponent, similar to backgammon. Pieces all start off the board and follow a set path on the board, eventually ending off the board on the other end. The board is structured such that there are 3 zones: 2 safe-zones where each player's pieces cannot interfere with one another, and 1 'war-zone' in between, where displacements can take place (see figure 1).

Players take turns rolling 4 binary dice (equivalent to 4 coin tosses), with the resulting value determining the number of spaces the player must move one of their pieces (only 1 piece may be moved, and that piece must be moved the full distance). Each tile may only have 1 piece on it at a time: moving a piece onto a tile occupied by the enemy sends the enemy's piece back to the start, while moving onto a friendly-occupied tile is invalid. However, a piece cannot be displaced if it is on the flower tile, which also grants an extra turn to the player who lands on it.

The inputs into the AI's are the current state of the board (i.e. the position of each piece) and the result of the dice roll (which determines the number of places a player can move a piece). The output is the action to take, and subsequently the new game state. For example, if the current game state has one friendly piece close to the end and the number of moves allows the player to move it to the finish, one output action could be to move it to the finish.

We evaluate the success of the neural network based on the win rate against other neural networks in the training phase, as well as the win rate against our baseline expectimax AI.

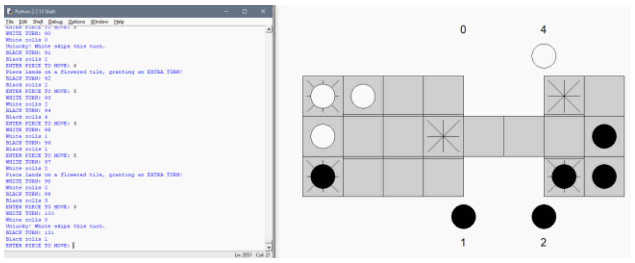
The baseline expectimax AI plays to a heuristic defined by ourselves. Compared to the neural networks, it has the advantage of having knowledge of the game mechanics and strategies, while the neural networks have to learn from ground up.

Infrastructure

We created a game engine in Python 2.7 and an optional graphical representation of the game using built-in graphics libraries. All necessary information for the AI's can be taken or derived from the engine's board state variable. In addition, functionality has been provided for a person to play, against one of the two AI's or another player.

For the sake of having a robust platform for the AI to play in, we allowed invalid moves to be made (e.g. moving a friendly piece onto a tile that is occupied by another friendly piece, or moving a piece that is at the end forward), but result in a pass.

In addition, pieces are not uniquely labelled, but identified by how far they have moved across the board. More specifically, the piece that has made the most progress is piece number 1, while the piece with the least is number 7. This is done to simplify the problem for AI's, which would otherwise require them to keep track of the positions of each unique piece.



```
ENTER PIECE TO MOVE: 5
WHITE TURN: 90
White rolls 0
Unlucky! White skips this turn.
BLACK TURN: 91
Black rolls 2
ENTER PIECE TO MOVE: 6
Piece lands on a flowered tile, granting an EXTRA TURN!
BLACK TURN: 92
Black rolls 2
ENTER PIECE TO MOVE: 5
WHITE TURN: 93
White rolls 2
BLACK TURN: 94
Black rolls 4
ENTER PIECE TO MOVE: 4
WHITE TURN: 95
White rolls 1
BLACK TURN: 96
Black rolls 1
ENTER PIECE TO MOVE: 5
WHITE TURN: 97
White rolls 2
Piece lands on a flowered tile, granting an EXTRA TURN!
WHITE TURN: 98
White rolls 2
BLACK TURN: 99
Black rolls 3
ENTER PIECE TO MOVE: 5
WHITE TURN: 100
White rolls 0
Unlucky! White skips this turn.
BLACK TURN: 101
Black rolls 1
ENTER PIECE TO MOVE: 
```

Figure 2. (Right) Visual representation of Ur using the game engine, (Left) Text-based representation and user input via shell

Approach

Ur can actually be modelled as a Markov Decision Process, since the next board state is dependent only on the current state. As a result, we decided that an expectimax approach would be effective in solving this problem.

The expectimax AI is built on basic heuristics which can be determined from the game's rules. Our heuristic can be summarized with a few lines:

- Get extra turns if possible

- Displace enemies if possible
- Avoid moving friendly pieces into the war-zone, where they are vulnerable to displacement
- Move friendly pieces off the war-zone if possible, to prevent them from being displaced

Aside from these, some other concepts are introduced to calculate the value of a board state for the AI:

- Number of pieces on board
- Distance travelled of each piece, the further the better

Using these heuristics, the expectimax AI is programmed to look several moves forward and make the move that would yield the best board state (on average).

As for the neural networks, we decided against the traditional method of backpropagation for training, since that would require using the same heuristic we used for the expectimax AI, and having 2 AI’s playing to the same heuristic would ultimately lead to uninteresting games. Instead, we decided to train neural networks from scratch, using a genetic algorithm approach.

We first created 100 networks, each with 18 inputs (14 for the positions of the 14 pieces, 3 for the positions of the flower tiles and 1 for the number of moves to take), 2 hidden layers with 12 neurons each, and 7 outputs corresponding to the piece to move. Each weight and bias of the networks was randomly generated to be a value between -5 and 5 with uniform distribution. These bounds were chosen to avoid math overflows when passing values through the sigmoid function to calculate activations of neurons.

After initialization, each network plays a single game against the other 99 networks in a round robin format. At the end of the round robin, the bottom half of the networks with the fewest wins are removed from the population, while the remaining 50 are bred to create new networks which inherit half of each parent’s weights and biases. 10 new networks are randomly generated and also added into the population to add more genetic diversity. Finally, the round robin, removal and repopulation process is repeated for a set number of generations. In our trials, we set the number of generations to 100. After the 100th generation, the best-performing 3 populations are chosen as the final networks that will be evaluated against the expectimax AI.

During the round robin, the winning network of each match is the AI that moves all of its pieces to the end first. In some cases, a stalemate may occur (be it from incompetence and repeated invalid move inputs, or from high level tactical turn passing). To account for this, a tie-break function is used to determine which network has made it farther in the game (i.e. more pieces at the finish, fewer pieces at the start).

Experiments

The training process of the networks took roughly 18 hours using a conventional computer. The best 3 networks were each put against the expectimax AI in a best of 5. The results were conclusively in the expectimax AI's favor, winning every single match 3 to 0. Aside from running time (in which expectimax took up to 5 seconds processing each move, while the neural networks took less than one tenth of a second) and memory usage, expectimax performed better than the neural networks in every way.

Upon closer inspection of the neural networks in the final generation, we discovered that the stage of 'development' of the networks were still 'infantile' at best - most networks were merely moving pieces out of the start and onto the first 4 tiles of the board, then getting stuck by making repeated invalid moves (since the tiles immediately after the start are all occupied, but the network repeatedly attempts to move more pieces on, resulting in an invalid move).

More testing revealed that none of the networks were even capable of moving a single piece to the finish when playing against a dummy opponent, and that the 'fitter' networks were exploiting the "fewer pieces at the start" clause of the tie-breaker to accumulate wins, rather than trying to reach the end.

Error Analysis

In hindsight, many different aspects of our methodology were flawed from the start, greatly limiting the success of our efforts on the neural network-based AI's.

Firstly, was the assumption that randomly generated networks would be capable of playing the game to even a smallest extent. As mentioned in the previous section, not a single network was capable of moving a single piece all the way to the end (which would be possible if the network had just chosen to repeatedly move piece '1'). When we discovered this, we attempted to implement a 'competence' test to every randomly generated network, which involves having the network move a single piece to the end. Further trials revealed that randomly generating a network that passed the competence test took an unfeasible amount of time.

A better approach to this issue would have been to generate networks randomly, and pass them through an initial bootstrap stage of training using a simplified heuristic and backpropagation, so that the networks would be able to learn the game rules and mechanics. Doing this before continuing onto the round robin stage would skip the hundreds or possibly thousands of generations required for networks to learn the game's rules through natural selection.

Secondly, was the assumption that our method of breeding would give the children 'traits' of its parents. 'Traits' in the context of neural networks and this problem are certain combinations of weights and biases that significantly influence the choice of the network.

However, arbitrarily taking half of each parent’s weights and biases does not guarantee that traits are passed on, and could very well destroy traits if only half of the relevant weights and biases of a particular trait are inherited.

Unfortunately, since the weights and biases of networks are indistinguishable from one another, the only solution to this issue would be to create more children and test each network.

Thirdly, was the platform in which the training and game was run. Although efforts were made to improve the robustness of the game engine for AI use (i.e. identifying pieces by progress and allowing for invalid inputs), an extra measure could have been implemented: to identify the most-progressed friendly piece not at the goal as number 1, the 2nd-most as number 2, and so on. In the case that there are some pieces at the goal already, numbers exceeding the remaining pieces not at the goal will refer to the piece that is furthest from the goal. This would remove the possibility of the AI making the invalid move of moving a piece at the goal forward.

Conclusion

In conclusion, the Royal Game of Ur is a prime platform for developing an AI. Due to its turn-based nature, the game can be modelled as a Markov Decision Process, simplifying the problem greatly.

We approached the problem using an expectimax AI and neural networks using genetic algorithms to varying success. The expectimax performed well against human players and randomized opponents, but suffered from lengthy processing times. The neural networks performed very poorly, primarily due to our flawed approach. Due to this, we are forced to postpone a conclusion on the effectiveness of neural networks over heuristic-based AI’s to a later study.

After identifying the errors in our approach, we proposed several changes to our approach, most notably introducing an initial training stage using backpropagation against simplified heuristic.