COMP30024 Artificial Intelligence Project Part B Report

Describe our approach:

1. What search algorithm have we chosen, and why?

In this project, the main search algorithm that we has chosen is the minimax search algorithm. The minimax algorithm is a recursive and a backtracking algorithm which is used in decision-making and game theory. The reason why we have used this algorithm is that this algorithm is highly efficient in case of a multi-stage game where a particular goal state is not pre-defined. It can analyse the state of the game board at each turn and figure out a local-optimal action to perform at this particular turn given the particular board state. Additionally, it provides an optimal move for the player when assuming the opponent player is also playing optimally, which means the system always considers the worst case in the game. If the opponent is not playing quite optimally, the program could get better result.

2. Have you made any modifications to an existing algorithm?

The algorithm structure has not been changed from what we have used in the Project Part A. For example, we still used the same notion of distance to think about optimality such as how far away a player is from reaching the winning side on the board. One difference is that we no longer considered the f(n) values as in the previous assignment, whereas the utility values were used in the minimax search algorithm. At each turn, we tried to assume that the opponent would be playing optimally by minimizing our utility values after our turn, while we would be trying to maximise utility provided with the opponent's behaviour. However, the concept of alpha-beta pruning is used to decrease the time complexity. Due to the board size which can be up to 15 and the state space, it is time-consuming to go through all states. In this case, we use alpha-beta pruning as the threshold in detecting state's utility. Specifically, if the current node's utility is smaller than alpha, then it is unnecessary to go through other nodes in this state, because the minimum utility value in this state will not exceed the current one.

Moreover, in our program, the cut-off strategy has been used instead of using the terminal state check. In this game, the board size is between 3 and 15, hence if the board size grows up to 15, the state space will go like 15^2+15^2*(15^2-1)+15^2*(15^2-1)*(15^2-2)+…, which will be an enormous size for the game to search to the terminal state each time. Thus, we choose to let the program search 4 steps in advance at each point, which means to carry out two whole minimax searches. As time complexity is specified in this assignment, this way will provide reasonable time usage and accuracy. As for the cut-off state test, we do not use other functions or other data structures which store the steps to check the state itself. In our design, each state stores the information of the whole board, and the game has a capture mechanism which means the number of red and blue pieces on the board might have a change based not on the placing at each step. In this case, it is hard to tell a cut-off state, and it is time-consuming. Therefore, we use a backtrack indexing for each turn. When the max_value or min_value function has been called, the index will increase by one; when the max_value function reaches the cut-off state number, the function will return the utility of the current state and decrease the index by one, which will guarantee the other states at the same level in the action list associated with the min-value function still have the chance to return their utility values; when all the needed states have been checked, the index

will decrease by one, which stands for the program going back to the previous state in the max step.

3. What are the features of your evaluation function, and what are their strategic motivations? If you have applied machine learning, how does this fit into your overall approach? What learning methodology have you followed, and why?

We did adopt a supervised machine learning approach in the development of our algorithm, and the methodology itself is associated with selecting the features to be incorporated in our approach.

The reason for using supervised learning:

1. It is more efficient to choose features.
2. It is more reliable to change weights because we could change the train sets.
3. It allows us to define each state's utility, which is easier to modify in later approaches.

As for the feature selection part:

In the collection feature state:

(1) The distance (the distance between the nearest pieces of this player and the current cell + the distance between the current cell and the end winning line) — mostly like the a* heuristic function — might be more beneficial to place the piece in the winning direction based on the connection set which starts from the start line

(2) The difference of chess pieces ("1" in the figures):: in the normal case, the difference will be 0 or 1, based on the turn of each player, so this feature is mainly focusing on the capture mechanism — in our sense, the more pieces your player has, the greater chance you will be winning the game

(3) Maximum difference of connected pieces ("2" in the figures):: we only consider the connected pieces on the winning direction of each player, which shows the effective effort of the player — if the longer the connection is along the right direction, the less steps needed for this player to put effort in winning this game

(4) The difference of the number of connected sets ("3" in the figures): the set of pieces with at least two pieces will be considered — the larger the number of connected sets is, greater the ability to build up a faster pass in the game

(5) Number of useful diamonds for red (i.e. harmful diamonds for blue): if the useful diamonds for the red player will be created based on the current steps, it will be a greater threat to the blue player's moves

In the feature selection stage:

- The "difference" is mainly used for decreasing the correlations between the blue result and the red result.
- The number of diamond is removed based on the logistic thinking.
- The difference of chess pieces is the result of the useful diamonds — if the program could think deeper, like 4 moves in advance, the effects of the number of diamonds will be replaced by the difference of chess pieces.
- The distance will be replaced by the maximum difference of connected pieces on the path direction — there are two reasons: the first one is that they are quite the same as both of them are detecting the distance to the winning line; the second one is that the space

complexity and the time complexity will increase more when using the distance as one of the features.

As for the rest of the three features in the selection part:
Firstly, we prefer to use the Spearman's correlation measurement to check the quality of each feature, because the feature and labels might not have strong linear correlations but strong nonlinear correlations. For example, the Pearson's correlation measurement will not be quite capable of capturing such phenomena in the features in out learning mechanism. If the correlation value is computed to be greater than 0.5 or less than -0.5, it means that the feature has the directive property in predicting the labels. In our test, all of these three features have high positive correlations with the labels, so these three features can be treated as good enough to be used to predict the labels.

	Differences of chess pieces	maximum difference of connected pieces	difference of number of connected sets
0	0.972282	0.961029	0.732432

Secondly, we use the Pearson's correlation measurement to check the pairwise correlations among all these three chosen features. Ideally, they should have low correlation scores, because we want each of them to influence the results of the labels independently. If the correlation of two particular features is high, it might influence the weight of predicting the labels. Unfortunately, according to our test during the machine learning procedure, they do display some high correlation figures, which might be somewhat significant and could not be easily ignored, so this part might demonstrate some limitations.

	corr1-2	corr1-3	corr2-3
0	0.887409	0.801199	0.541428

Thirdly, we are using the wrapper class to check which feature(s) may need to be deleted, because if the correlations between features are too high, there might be the feature(s) to influence the accuracy of the algorithm. As for the model, we are using the Stochastic Gradient Descent regressor. The gradient of the loss is estimated with each sample at a time, and the model is updated along the way with a decreasing strength schedule. This model provides more useful loss functions and reasonable penalty selections. We choose "huber" to be the loss function, which will modify the squared_error to focus less on getting outliers corrected by switching from the squared to the linear loss pasting a distance of epsilon. However, after some analysis and testing, the fact is that it might be available to ignore neither of the features, because they all have their own attribution to the accuracy.

	accuracy_123	accuracy_12	accuracy_23	accuracy_13
0	1.0	0.9	0.6	0.6

As for the predicting parameters:
The difference of chess pieces: 0.824

Maximum difference of connected pieces: 0.557
Difference of the number of connected sets: 0.200

```
array([0.82359546, 0.55724447, 0.19875641])
```

As for the train data set and how to allocate utility values:
We declare three cases: extremely worst case, extremely best case and the normal case.

- In the extremely worst case: (1) no red pieces (2) the length of blue pieces > the length of red pieces (-10<utility<0) — we make our model learn the worst position of the low utility scores
- In the extremely best case: (1) no blue pieces (2) the length of red pieces > the length of blue pieces (10<utility<20) — we make our model learn the best position of the high utility scores
- In the normal case: we define nine situations such that each one of these nine has its own utility value:

(1) Eat blue, but no influence on other two features — utility 3
(2) Eat blue, while increasing the difference of the number of connected sets — utility 7
(3) Eat blue, while increasing the maximum difference of connected pieces — utility 8
(4) Eat blue, while increasing the maximum difference of connected pieces and decreasing the difference of the number of connected sets — utility 9
(5) Increase the maximum difference of connected pieces, while decreasing the difference of the number of connected pieces — utility 7
(6) Increase the maximum difference of connected pieces, but no other influences — utility 6
(7) Increase the maximum difference of connected pieces, while increasing the difference of the number of connected pieces — not possible
(8) Increase the difference of the number of connected pieces — utility 3
(9) No influences on all of the three features — utility 1

How this supervised machine learning mechanism fits the overall approach:
According to the procedure of learning as discussed above, we selected the features to be considered in judging actions of the player at each turn. We dealt with how potential actions of a player could be quantified with some utility scores in order to determine if one particular action out of a list of possible actions would be desired so as to provide a higher utility score. Then, we turned back to our algorithm and the minimax search part to use these computed utility values. Our search algorithm, as discussed before, would try to instruct how the pieces of the player could act at each turn on the board that would lead to higher utility scores, based on the current state of the board to be incorporated as those features.

Performance evaluation

1. How have you judged your program's performance?

In our approach, because we have incorporated the alpha-beta pruning into our algorithm and in particular the searching part, the time complexity of our game-playing program can be guaranteed. To test this performance, we have varied the size of the board or altered the colours

of the two players to take into account different scenarios that may lead to complicated game states. Fortunately, our program's performance is relatively satisfactory because the games that we have played can be completed within the given time limit.

2. Have you compared multiple programs based on different approaches, and, if so, how have you selected which is the most effective?

During the process of our development, we used basically two game-playing agents to compete against each other to see if one or the other one would be a better solution. For one agent, we maintained the specification as above, and for the other agent, we tried to do some modifications such as to select different features and different weights. The best model, after several rounds of playing and learning, has been provided. We did not consider other machine learning approaches such as to use another loss function, so our method was to slightly change some features and values based on the an overall same mechanism.

According to the assignment specification, we also tested our optimal player with three different types of other players with increasing intelligence. Firstly, we considered an opponent who took actions randomly. Specifically, we just used some randomly generated index to choose one particular action out of various possible actions for the opponent at each of its turns. In this case, our agent won five out of five games against this opponent. Secondly, we considered an opponent which would act on a greedy manner such that, at each turn, it picked the action that resulted in maximum utility without considering our player's response and looking some rounds ahead. In this case, our agent also won five out of five games. Lastly, we considered an opponent with more intelligence by using some evaluation function to take actions. In particular, we considered the case of pure A* search without other advanced methodologies such as machine learning. Then, the opponent used this search method to take actions based on higher evaluation scores. Again, our agent won five out of five games against this opponent.

Other aspects

As for algorithmic optimisations:

We did not use special methods to cope with space complexity optimisations in our code development because we thought that space was not an issue out of control in our process. However, even if we did not encounter any space issues during our development, we still used some advanced data structures for the program because that could possibly even save more spaces. We used the alpha-beta pruning method to lead to time complexity optimisations. Moreover, we predefined the first move of our agent to reduce the time usage of the game total, because in our agent, the first move do not need to consider too much, as we noticed that the difference of pieces has the highest weight in predicting utility, therefore, if our agent plays in blue, the first step will always be "STEAL". If our agent plays in red, the first step will always be placed in the middle of start line, which is a good place to start the game.

As for specialize data structures:

In the development of our program, we used the data structure of two-dimensional array to store the board and especially the states on the board. We represented the board in this two-dimensional array with int1 and int2, instead of using, for example, a dictionary data structure, to store the board information such as with the key-value pairs representing coordinates and players respectively. Although we did not run out of space while testing, we used this advanced data structure because it could lead to more space efficiency in terms of its complexity.

As for alternative or enhanced algorithms:
We used some knowledge from the subject COMP30027 Machine Learning because it provides some detailed and enhanced instructions in terms of machine learning and feature selection. For instance, we adopted different correlation measurements to deal with nonlinear correlations between features. Moreover, we used some independence assessment to check the feasibility of potential features to be included in our learning model. In the end, we came up with necessary features, and we used accuracy calculation to test whether each of the features would have some attribution in explaining the model and making predictions. Then, only useful features have been selected by our program to proceed to the next step of allocating and determining utility scores that will be helpful in the minimax algorithm.