Concordia University
Department of Computer Science
and Software Engineering

Artificial Intelligence
COMP 472, Winter 2019
REPORT for Project 1

Raphaëlle Giraud[1] and Talar Kochakejian[2] and Tony Yuan[3] and
Michael Tarantino[4]
[1] 27514204 raphaelle.giraud@gmail.com
[2] 40018152 koushagjiantalar@gmail.com
[3]40029336 yuan.tony.1@gmail.com
[4] 26570070 taramic.42@gmail.com

Table of Contents

1. Introduction and technical details

The following report presents our findings resulting from the implementation of the game Double Card. The challenges encountered in designing the AI system are also discussed. Double Card has many similarities to Connect 4, with some key differences that increase the complexity of the game, such as an increased branching factor, due to each card having multiple possible rotations, as well as more complex late game rules.

The game is presented on an 8x12 board, where one player is assigned colours and the other dots. Players then take turns placing tiles that occupy two slots on the board, in which the tiles fall from the top in a similar fashion to connect 4. These tiles are domino shaped, with each domino having half of it being red and the other half being white. Each half also contains either a filled dot or hollow dot. The goal of the player assigned colours is to have 4 consecutive slots of the same colour while the player assigned dots is to have the same result but with dots. The consecutiveness can occur either horizontally, vertically or diagonally. Once all 24 cards have been used, players begin recycling from the board. Instead of new cards being added to the board, they are removed and placed back, while still conforming to the regular rules of card placement.

The decision making system used for the AI is a min max tree. The tree architecture allows for varying search depth, which can be increased or decreased as needed for time constraints. A simple informed heuristic was developed to score individual game states, tested against human players as well as against itself and refined based on the results of the test runs. A more detailed description of the heuristic development can be found in section 2.

2. Heuristic

For performing our min max algorithm, a heuristic with a good balance of speed as well as overall accuracy was required. There exists a proportional relation between calculation time of a heuristic, depth of the search and processing time. Differing emphasis on these individual parts had to be tested to strike the right balance between processing time and how informed the heuristic was.

In the process of encoding the game, an efficient method for checking for win condition needed to be developed, such that, at each play of a new card, the board would be checked for 4 consecutive colors and dots of the same type. This method focused on treating the new tile played as an origin point, and looking within a circular radius of 3 tiles around it, to determine if any sets of 4 or more exist. This concept was the motivation behind our heuristic as it allowed for an effective way to look for leaves of the min max tree that contain winning conditions.

The format of our code required us to make a node class, which contained heuristic function. A root node containing the current board state is created, and from that point, all possible legal moves for the current board were generated. This created the child node branches which could then have their child nodes be built using a recursive function.

Once the maximum depth value is reached, the heuristic function is applied on the resulting board. As stated previously, the game code developed contained an efficient method of checking for winning situations in the board. Given that the function would return a list of numbers, it seemed to be a perfect method to adapt into a heuristic function. It was modified to not only check for sets of 4 or more colors and dots of the same type, but also for open cells where cards could be played on the next turn.

For example, if a move by the AI would create a set of 3 with an open cell on either side, the board would be given a score depending on if the set was the AI's type or not. Different set sizes with different open cell amounts were given different scores.

The intention of this was to have the AI prefer to block its opponent from creating a long line of the same type of dots or colors, while simultaneously creating the longest sets possible for itself. Furthermore, when tested, this heuristic allowed the AI to choose a reasonable move in less than 3 seconds during regular play and in roughly 6 seconds during recycling play. Given the AI was not permitted to spend more than 6 seconds of processing for a single move, a more complex heuristic was not feasible. By the same reasoning, the depth of the search was limited to 2 levels.

We realized, only after the tournament, that the majority of the processing time was taken by the construction of the min max tree rather than the evaluation and search. For any regular move, there exists a total of 8 location in which the card may be played and a further 8 rotations for said card. Four of those rotations can be discounted as a horizontal card cannot ever originate in row H, as the rightmost cell would be placed out of bounds. From this information the branching factor can be determined.

$$loc*rot\text{-}inv = BF \tag{1}$$

This means, during regular play, there is a branching factor of 60. For a depth of 2, this results in 3600 individual game states to be evaluated. This value increases significantly during recycling play.

$$validRecyc*BF = recyclingBF \tag{2}$$

Given that a target card must be chosen, this would increase the branching factor by a factor ranging from 3 to 7, where 3 considers the situation, in which all valid targets are horizontal and 7 considers the situation, in which all valid cards are vertical. This indicates a branching factor between 180 and 420, which results in a range of 32400 to 176000 possible game states for a tree depth of 2.

As stated previously, a more complex heuristic would increase decision time of the AI past the limit of 6 seconds. This remained true even after the algorithms for generating legal moves and building the min max tree were optimized as much as possible. Simply put, the complexity of the game made us conclude that a more informed heuristic was unfeasible with the given requirements.

3. Results

On April 21st, we played against two teams in our COMP 472 class. Our AI would play against their AI.

Team name	Game 1	Game 2
Spaguetti	Lost honorably	Lost honorably
OVO	Win, opponent lost honorably	Lost honorably
We may be overthinking this, haha	Lost honorably	Lost honorably

Tournament results

Four of the fives games lost during the tournament were due to our AI completing a set of 4 for the opponent's chosen type during the AI's own turn. This is likely due to the AI prefering to play defensively when it can't immediately win. For example, let's assume the opponent is currently playing colors. When our AI sees a set of 3 anywhere on the board for the opponent and it can block it, it will immediately do so. However, there can be a situation where no matter how the AI plays, the opponent will win. The simplest example of this is having 3 red cards in a row horizontally followed by a blank space and then 3 white cards in a row. The AI knows that if it doesn't play there, it will lose. However, no matter how it chooses to play, the opponent will still win during the AI's turn.

Our AI's single win was when the opponent created a situation where it could win that turn, which it correctly identified. However, that only ever occurred once.

It is also possible that the AI played much too defensively. Since it prioritised blocking the opponent from getting sets of 3 and 4 or more, it would often ignore playing in a location that would allow it to win on it's next turn. The more aggressive opponents were therefore able to keep control of the board and set up a situation where they were guaranteed to win while our AI desperately tried to stop them.

It should also be noted that the AI would ignore sets of 2 and 1 of a single color or dot type. This may have had an effect on its, but it is difficult to tell given the complexity of the game.

When pitting a human against the AI, it would occasionally cause itself to lose on it's own turn, but this only happened approximately 10% of the time during tests. These difficulties and uncertainties will be further explained in section 4.

Another aspect of the heuristic worth note, is that it only considers and scores sets that originate from the last card played. This would mean that the AI would ignore the first card played as well as any sets of 3 or more that are created by it's placement. Therefore, the AI only considers a small part of the board and only the card played by the opponent. This is likely why the AI tends to play in a defensive manner. Even if by playing in a certain location would result in the AI winning on it's next turn, the heuristic won't score the game state originating from that card, but rather the score originating from the card its opponent will play.

During testing, an effect of this was quickly seen, in that the AI would often ignore a situation where it would instantly win on it's turn. This will be further elaborated on in section 4.

4. Difficulties

The project's difficulties were at first mostly regarding the implementation of all the game's rules and our unfamiliarity with Python.

Once we understood the game's rules and how it worked, we had to think about how we wanted to implement it, which functions we wanted to create and how to link them together.

We had some difficulties grasping the logic behind the game: when should we check if the space is available? How can we verify if the move is out of bounds? What are all the tests that we have to make in order to make sure that our project takes all the rules into consideration? Eventually, we were able to create an architecture that simplified implementation as much as possible, leaving only the game's rules to be implemented one at a time and rigorously tested.

The hardest part of D1 was recycling. We neglected to update the board when choosing a card to recycle before checking if the new location for the card was legal; we were simply updating once the play was done. Therefore, the game would allow a player put the card on top of an empty space thinking that the card, which was being recycled, still existed in its original location. Also, from the same logic, the game would prevent the player from placing a recycled card in the same location with a different rotation.

Further difficulties were encountered during the development of the AI. Namely, the development of an informed heuristic and its testing. When pitting a human against an AI, it would occasional make the human win on its own turn. Given the complexity of the game, it was difficult to determine whether this was due to an issue with the heuristic or if the human created a situation where the AI was forced to lose.

In addition to this, the time limit of 6 seconds meant that we were reluctant to try and increase the complexity of the heuristic as recycling already took roughly 6 seconds for the AI to evaluate. As such, we tried to make the simple heuristic as intelligent as possible.

One method of reducing processing time was to check every node for a winning state before generating its children. That way, if a board was in a winning state before maximum depth was reached, no new nodes would be created from that branch. This also had the effect of fixing an issue with the heuristic where the AI would ignore a move that would result in an instant win.

5. Responsibilities and contributions of each team member

All our commits are available on GitHub: Talark/polite-double-card and the report's history is available on Google docs: 472_P1_Report_27514204_40018152_40029336_26570070.

Name and student ID	Responsibilities and contributions
Raphaëlle Giraud, 27514204	Mostly worked on D1. Helped fixed some issues for D2. Bug testing for D1 and D2. Set up the group chat and the report's skeleton. Wrote the first part of the result section in the report.
Talar Kochakejian, 40018152	Mostly worked on D1. Set up the GitHub repository. Bug testing for D1 and D2. Wrote the "Read me" to run the project.
Tony Yuan, 40029336	Worked equally on both D1 and D2 (a noticeable contribution). Wrote the introduction of the report as well as part of the heuristic section. Bug testing for D1 and D2.
Michael Tarantino, 26570070	Team leader. Set up the team. Biggest contributor for D1 and D2. Implemented the project's logic and architecture. Helped in bug fixing. Wrote part of the heuristic and result sections in the report.