Timothée Guerin
260447866

# Comp 424 Project Report

## Method

My current method is getting the move that offer the highest probability of winning at the end of the game. For each possible move the player can do it calculate a value and get the highest. The algorithm separates the game in two phases. The first one goes from the 1st move to the 9th before the end. The second phase is during the last 8 moves.

However both phase work on the same principle. It calculate a potential game from the current move to the end of the game then depending on the outcomes of this game it update the score of this move.

The phase one is using the fact that we cannot as it the second phase calculate all the possible moves to the end of the game ($61! = 10^{66}$ possibilities at the first move, still $10! = 3628800$ at the 51st move) in 5 seconds. Then the phase one is only going to generate for each valid moves of the player a random game until the end and map result to this move. As this method is generating random path the larger the number of generated path is the better. Then this method is just going to iterate until 4.9 seconds have past then it will return the best move depending on all the random path outcomes.

The phase two is still using the end of the game to find the best move. However we now have only a few moves to take care of(Less than 8). Thus we have less than $8! = 40320$ ending possible which can be tried in less than 5 seconds. Then for each valid move it is going to reclusively try all ending this path can lead to. Then we are going to choose the move offering the biggest number of winning games.

Timothée Guerin
260447866

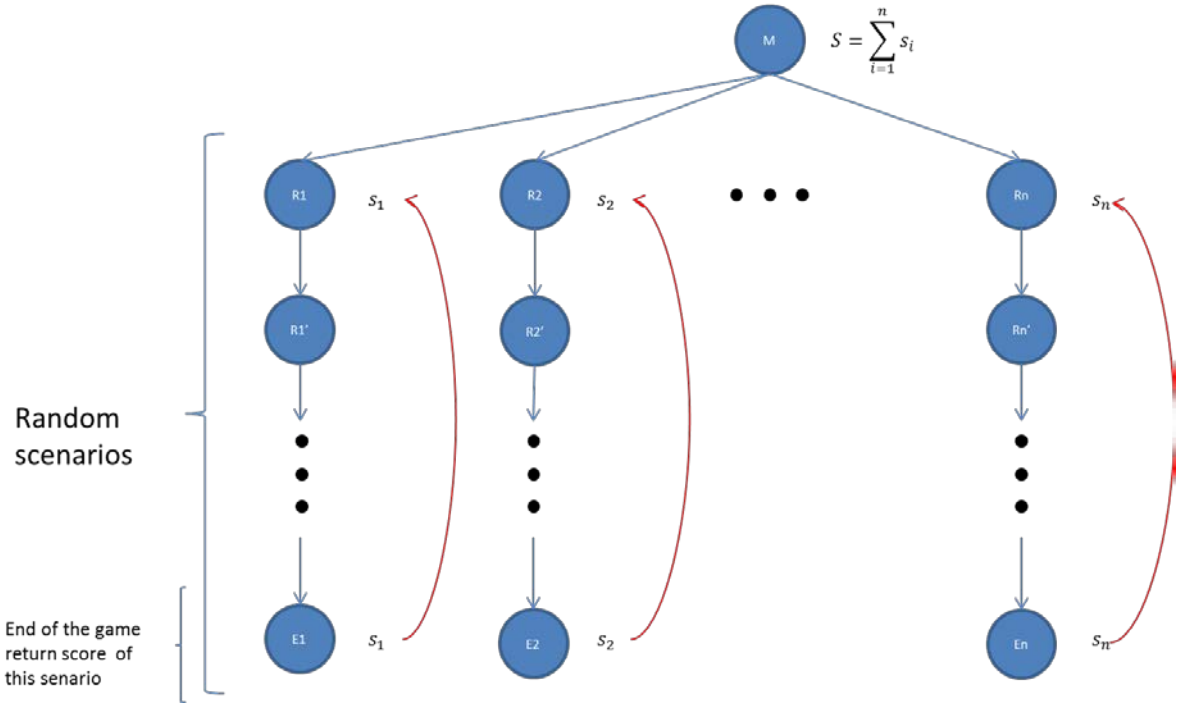
e.g.: We have two moves remaining M1W, M1B, M2W, M2B, and which lead to the following possibilities:

| Move | Next move | Scenario 1 ending | Scenario 2 ending | Scenario 3 ending |
|---|---|---|---|---|
| M1W | M2W | Win | Loose | Win |
| | M2B | Loose | Win | Win |
| M1B | M2W | Loose | Win | Win |
| | M2B | Loose | Win | Win |
| M2W | M1W | Loose | Loose | Win |
| | M1B | Loose | Loose | Win |
| M2B | M1W | Loose | Loose | Win |
| | M1B | Loose | Loose | Win |
| | | | | |
| Best | | M1W (1 win) | M1B (2 win) | M1W (2 win) |

This game is particular as we can have a turnaround at any time and in particular near the end (Two groups merging into one). Thus I found that going only a few move head is useless, everything can change the move after. It is why it use a method that go to the end of the game. Then it's trying a random path so we give an equals chance to all game scenario to happen. As doing "bad" move at the beginning can be better at the end. For example let's suppose the player is number one then we want an odd number of groups. Then we can choose between two move one that's going to let the number of groups to 3 and another one that is going create a $4^{th}$ group. Here on first thought we are going to choose the move 1 however as the game is so unstable(Can turnaround at any time) creating this $4^{th}$ group can be a better idea(A $5^{th}$ group is going to be created, it's going to merge with another)

Timothée Guerin
260447866

Phase 1 schema: for one move we have n different scenario randomly generated and get the sum of all the score of those.




## Pros and cons:

### Pros:

- Very simple
- Phase 1 is consuming few resources (One scenario at the time), recursion of less than 61.
- Even if the IA is using all the time possible to get the most accurate move it can still make good choices with a very fast turn time.

### Cons:

- This algorithm is time consuming as it exploit all the time it has to play.
- Phase 2 start depends on the computer performances. The more powerful it is the sooner we can start the phase 2(which is more precise)

- Working on probability: might me defeated by an opponent who can use it at his advantage.

## Other approaches:

I started a min-max approach going a few moves ahead (Depth increase when the number remaining moves decreases). However as explained before the game is so unstable in long run then choosing the move depending on the outcome of a close future is not suitable. This was not very efficient.

Then I modified this method by introducing the phase 2 of the current method. The IA was making the right choices (after bug correction) during the last move of the game. However it was still not working fine. The player was only winning 80-90% of the games against the random opponent. This was due to previously made move (Before entering phase 2); indeed the game was lost whatever the player was doing for the last 5 moves.

## Improvement

This IA doesn't exploit the other player turn waiting time. We could use it to get a more precise score and then choose a better choice. We can keep the same principle to find the best path but here as we don't know what the other player is going to do and we don't want to find the best move for him we are going to need to try all moves our player can do after each possible move of the other player. Then instead of an n size array to store the moves score we are going to have a $n^2$ matrix with the n moves possible for the opponent as row and the remaining $n-2$ moves for our player after. Then when it's our turn to play we just have to initialise the score array with the row corresponding on the opponent choice.

We could also do a learning algorithm that will save good move, for the beginning of the game for example.

The phase one is generating the scenarios randomly which give an advantage against a random opponent, we could find some heuristics that would put away useless move for a more human player.