IT3105 - Artificial Intelligence Programming

Quarto

Minimax intelligence using alpha-beta pruning

Marien Fressinaud
marienlf@stud.ntnu.no

18th September 2013

1. Introduction to my Quarto game

Before beginning, we have to get an overview of the game. The following diagram is a basic analysis of my game. The main class is Match which runs all logical functions (wait for a player, place pieces at specific positions, check if the game has been won, etc.). A Player has an Intelligence which can be Human, Random, Novice or Minimax.

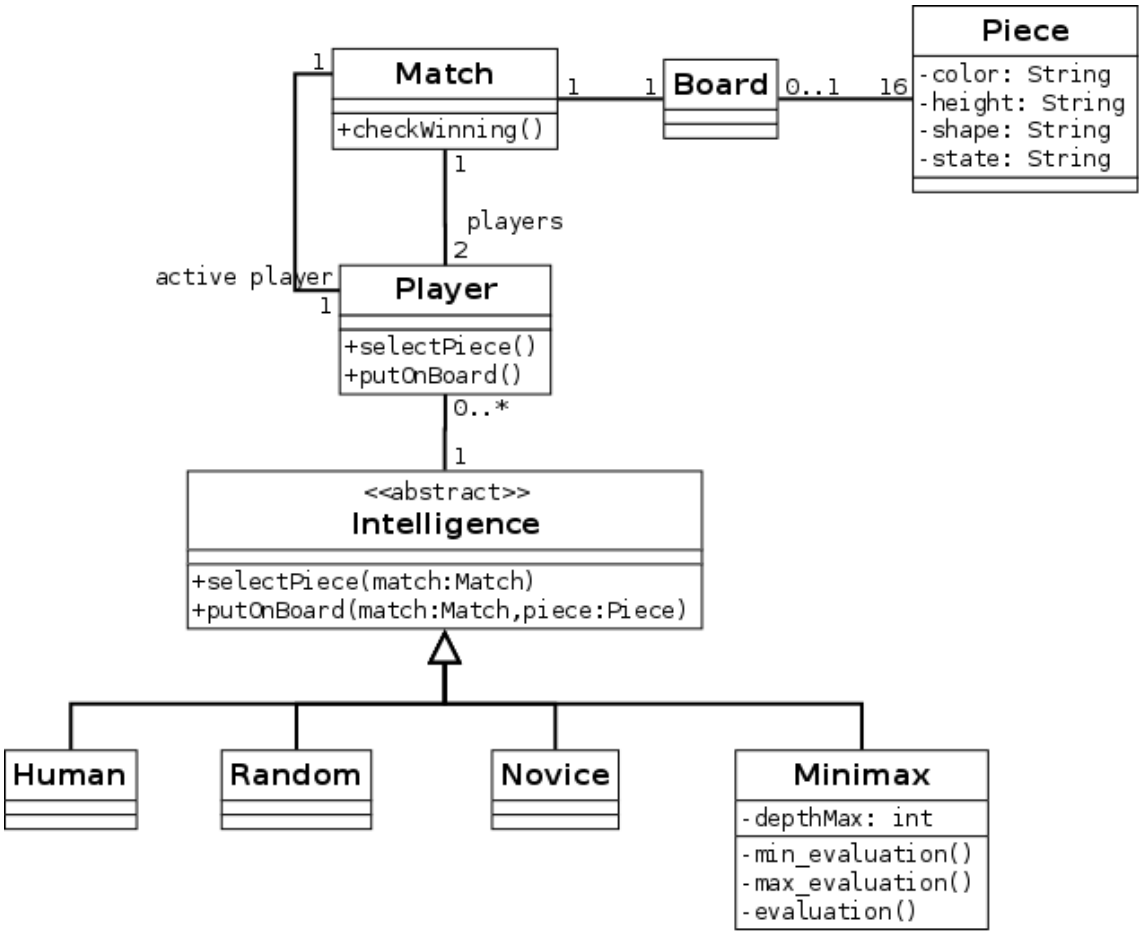


Illustration 1: Class diagram of my Quarto game

The Minimax intelligence is important to understand. For both `selectPiece()` and `putOnBoard()` functions, Minimax will evaluate the different possible states of the board.

- To put a piece on the board: we have just to maximize positions of this piece and keep the best one.
- To select a piece, it's a bit more complicated: algorithm must, for each available pieces, maximized its position on the board. Then, we have just to minimize these evaluations to select the less dangerous piece.

`min_evaluation()` and `max_evaluation()` functions are basically the same as in course slides. If current state is a terminal state (winning board, full board or max depth reached), I return an evaluation of last action. Otherwise I apply the corresponding algorithm (max or min), combined with alpha-beta pruning.

To optimize time cost without losing too much performance for Minimax, I applied a simple algorithm:

- If there are more than or equal 12 pieces on the board, I always use Minimax 1
- For more than or equal 10 pieces, I use Minimax 2 (except for Minimax 1 obviously)
- And finally, for more than or equal 8 pieces, I use Minimax 3

I don't allow utilization of Minimax 5 or more.

2. State-evaluation function

The most interesting function of Minimax algorithm is probably the state-evaluation one. The idea behind my function is pretty simple (too simple? I will explain that in the conclusion of results), I just check for the last played piece if its position, for each of its properties, is the better one. If it is, I increase the value of evaluation. But explain it more accurately.

Evaluation values

There are two specific states for a game:

- Winning state: this is the best (or worst, it depends) end for a game, so we give it the maximum value for an evaluation. Actually this value is 500, but this is arbitrary. The only important thing is to keep a value really greater than any "standard" evaluation.

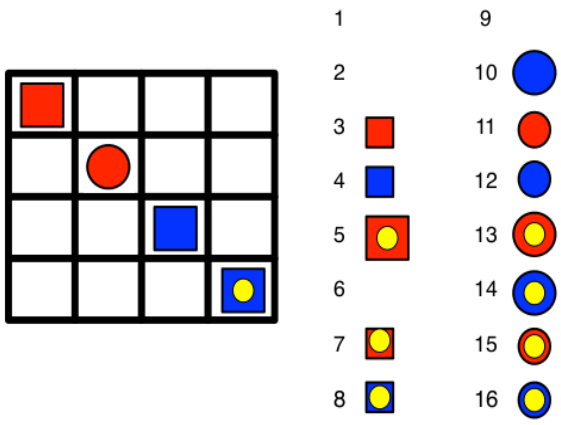


Illustration 2: The value returned will be 500 (4 tall pieces aligned)

- Draw state: that's not necessarly the best option, but a draw is a "good" end. Its evaluation value is 490.

Finally, a standard evaluation must return a value between 0 and 480. In this case, algorithm evaluates the last played position and check if this is the best one.

We will detail how we evaluate in the next two parts.

Property maximizations

The property maximization consists to check where is the best position for a piece property. For example, in the following illustration, the best position for properties are:

- Red and hollow: on 1st row or 1st column
- Tall: on the diagonal ([3,3] or [4,4]). This diagonal will also become a winning position for height property, I explain it in the next part.
- Circle: on the 2nd row or 2nd column

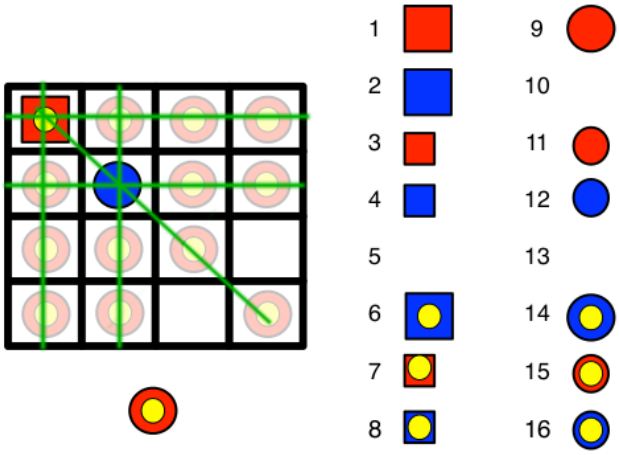


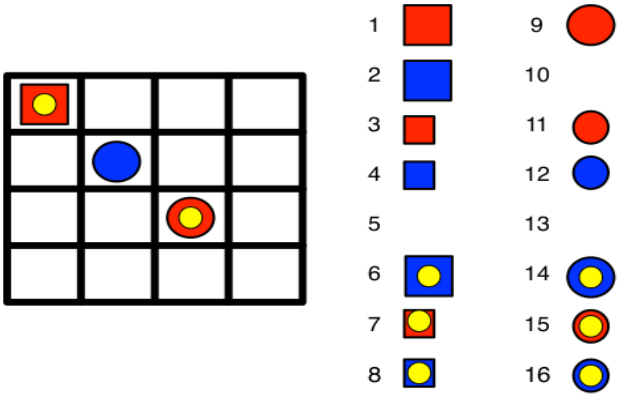
Illustration 3: The red, tall, circle and hollowed piece is selected

So, for each property which is maximized for the position we are currently evaluated, we increase value of our evaluation. In this example, positions [1,2] and [2,1] are the best because they maximize two different properties.

Increasing precision of evaluation

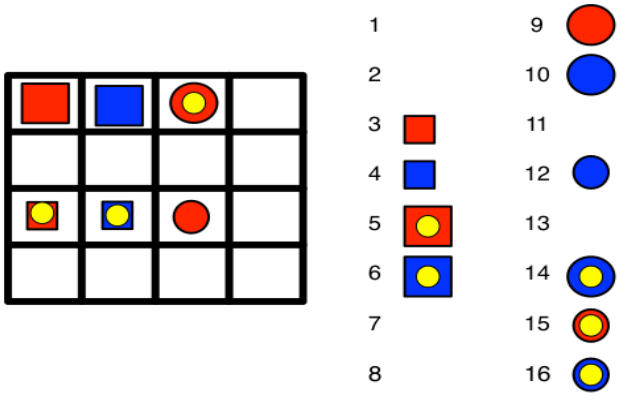
The first approach is correct but it's still a bit simple. It's the reason why I decided to improve it.

As I said in the previous part, if we put the piece in [3,3] or [4,4], the height property become a winning one: there are 3 tall pieces aligned, so it's surely a really good position to approach a won state.



So, to take this state in account, I just added the number of winning properties of board to the previous evaluation.

But wait a minute, it can't be enough! Consider the following state:



Here, we can see two winning properties: tall and short, the two opposite values of height property. So if we are currently putting a piece on the board (for example we are evaluating piece in [3,3]), this is a state we have to reject because we will cannot be able to select a piece for the other player without lose. In this situation, evaluation is decreased to zero.

3. Results

100 rounds Random vs. Novice

Random (wins)	7
Novice (wins)	93

This first result seems to be good enough. Random player wins sometimes, but it's still a good ratio.

20 rounds Novice vs. Minimax-3

Novice (wins)	8
Minimax 3 (wins)	12

This result is more controversial (like the next one). Here, we would expect a large difference since Minimax 3 can predict the game state with 3 moves ahead.

I do two suppositions:

- The evaluation function is not good enough and doesn't permit to estimate correctly a good game
- There is a bug during execution of Minimax algorithm

After rereading several times algorithm code, I expect the problem comes from the evaluation function.

However, I think my function is not so bad, so I have decided to keep it like that and assume the problem.

20 rounds Minimax-3 vs. Minimax-4

Minimax 3 (wins)	7
Minimax 4 (wins)	13

Same controversy as previously and same conclusion.

4. Tournament

Results

Minimax-2 ply, 200 games:

Losses / *Wins*	Sondre / Kristoffer	Sindre	Bart	**Marien**
Sondre / Kristoffer	x	192 (0)	111 (5)	**183 (2)**
Sindre	8 (0)	x	30 (0)	**36 (5)**
Bart	84 (5)	170 (0)	x	**128 (26)**
Marien	**15 (2)**	**159 (5)**	**46 (26)**	**x**

Minimax-3 ply, 20 games:

Losses / *Wins*	Sondre / Kristoffer	Sindre	Bart	**Marien**
Sondre / Kristoffer	x	4 (15)	15 (4)	**13 (5)**
Sindre	1 (15)	x	3 (5)	**10 (2)**
Bart	1 (4)	12 (5)	x	**18 (2)**
Marien	**2 (5)**	**8 (2)**	**0 (2)**	**x**

Experience from the tournament

The tournament was a great conclusion for this project. We have used the TCP server of Bart for that. In fact, we run this tournament twice, due to a lack of draws during the first one. In the two cases, results are mostly equivalent so this doesn't really matter.

It was interesting to compare our personal player agents to see if we have done a good work. Unfortunately, Sondre / Kristoffer group and Bart are definitively too strong for my Minimax agent.

An interesting result come from my plies against Sindre: we can see during the first game my agent beats his one 159-36 (5 draws). It's a really good score with a big gap and I'm glad about it. But if we look the second game, scores are closest and Sindre even beats me. That could be interesting to look further more decision process of my Minimax agent at

different levels. I guess the problem to improve my agent can be solve by understanding why Sindre beats me during the Minimax-3 ply.

Also, I note, during the Minimax-2 ply, how my agent fought against Bart agent with 26 draws. It's interesting since it's really more than all other battles. I can imagine our state-evaluation function are relatively equivalent, but mine has still some strong weaknesses.

To conclude, no matters who won: I knew my Minimax was not perfect. Even if I hoped a better score, it was my first real artificial intelligence experience and I'm pretty proud of its performance for a first tournament.

Now, a good challenge could be to improve my algorithm to win some more rounds against the best agents, but I think they already have some moves ahead.