Salomé Lahmar
16201438

AI for Games & Puzzles
Assignment 2 : Yavalath

1. Design of the program

My program consists of several objects interacting with each other :

- the Game, which handles the run of the game
- the Position, which represents the state of the board throughout the game
- the Player, which allows the interaction with a human player
- the AI, which will play against the Player choosing its moves according to an Alpha-Beta search, with or without an heuristic

2. Represent positions and print positions

The Position object is represented by an array of 121 elements, according to the suggestions made in the assignment.

The elements are simple characters :

- 'O's and 'X's for the pieces of both players
- blanks if the cell is playable, but empty at the moment
- carets if the cell is not playable

The printing function is the one suggested in the assignment.

3. Interpret user input

The Player has a `translate_to_cell` method, which takes a row and a column and translates them into the index of a cell in the Position's array.

Likewise, I implemented a `print_move` method in the AI object, which does the translation from cell to row and column, to display the AI's move in a way understandable to the user.

4. Finding patterns

The Position object has a `find_pattern` method. This method takes a pattern as argument, and browse the array to find rows, columns and diagonals that match this pattern.

In fact, it loops over the array with a step corresponding to the direction (1 for rows, 10 for columns and 11 for the diagonals) and aggregates the cells by concatenating them in a string per row, column or diagonal. Then, it checks the number of occurrences of the pattern in this string.

a) End of game

The end of game is detected by using the `find_pattern` method when :

- the pattern with the four same pieces is detected → the current player wins
- the pattern with the three same pieces is detected → the current player loses
- 61 moves were played without the second player taking the other's move as his own
- 62 moves were played with the second player taking the other's move as his own

b) Static evaluation function

The static evaluation function takes a list of patterns and their rate. For each pattern, it multiplies the number of occurrences of the pattern with its rate and then sums these values.

Choosing the right rate for each pattern can be difficult, because one sequence of pieces can contain several occurrences of one pattern or two different patterns. I chose the rates after testing several values.

5. Implement Alpha-Beta search with a Killer Heuristic and a History Heuristic option

I implemented the classic Negamax algorithm, with small changes. Instead of returning only the value of the move, the algorithm returns the cell as well.

The heuristic is an attribute of the AI and must be chosen at the beginning of the game.

For each heuristic, there is a dictionary containing the cells of the board along with how many times there was a cutoff for this move. This dictionary is modified everytime there is a cutoff. Before iterating over the children of the current position, the available moves at the current position are sorted in function of their value in the dictionary.

6. Experimentation

I experimented against my AIs by playing against them : for each move, each of the three different AIs (simple Alpha-Beta, Killer Heuristic, History Heuristic) returned the move it had chosen along with the number of static evaluations made to get to this result. Finally, the move chosen by the simple Alpha-Beta AI was played.

I played against my AIs at depths 3 and 4. (My computer wasn't powerful enough to run the AI on depth 5.) The results of this experimentation are presented below.

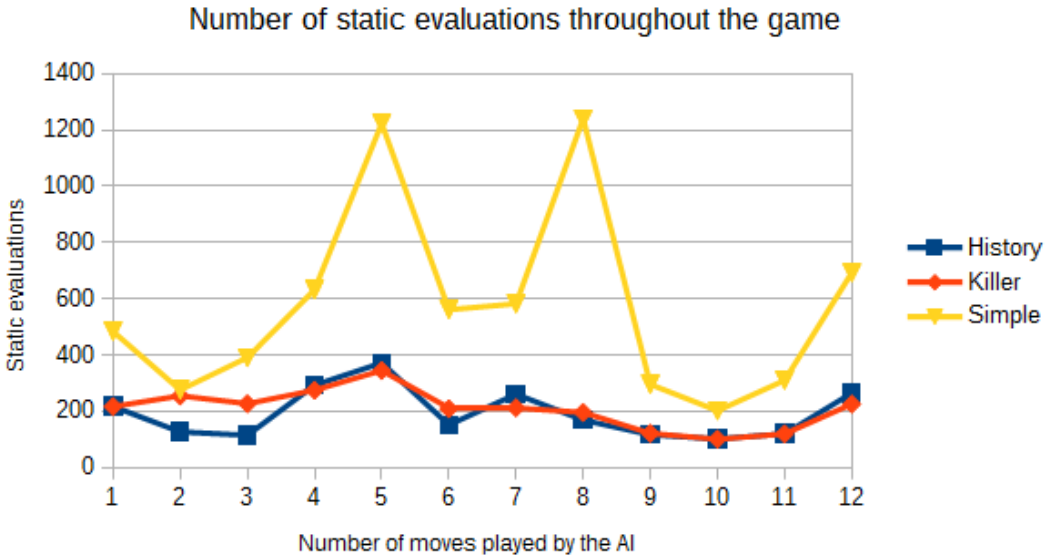
a) Depth = 3

	History		Killer		**Simple**	
	Moves	Static eval	Moves	Static eval	Moves	Static eval
Me	E4					
AI	E4	217	E4	217	**E4**	484
Me	E5					
AI	B4	127	B4	255	**B4**	276
Me	C4					
AI	E7	115	E7	227	**E7**	391
Me	C5					
AI	H4	293	H4	275	**H4**	632
Me	G5					
AI	F4	372	F4	345	**F4**	1223
Me	G4					
AI	C2	152	C2	209	**C2**	561
Me	D3					
AI	F1	261	F1	213	**F1**	582
Me	G2					
AI	G3	169	G3	196	**G3**	1240
Me	C7					
AI	C6	117	C6	122	**C6**	296
Me	H5					
AI	F5	102	F5	101	**F5**	202
Me	F3					
AI	I1	120	I1	120	**I1**	310
Me	H2					
AI	G1	265	G1	226	**G1**	693
Me	H1					

Figure 1: Experimentation against AI, depth = 3

In this game, the AI has won by making me playing three pieces in a row.

First, we can see that the AIs always agreed on the move to play. However, the number of static evaluations varies : the simple Alpha Beta is always the one which evaluates the most. Then, depending on the move, both of the AIs using the heuristics can be the quickest one.



On the graph above, we can see that the AI follow the same curve : their optima are reached at the same stages of the game.

b) *Depth = 4*

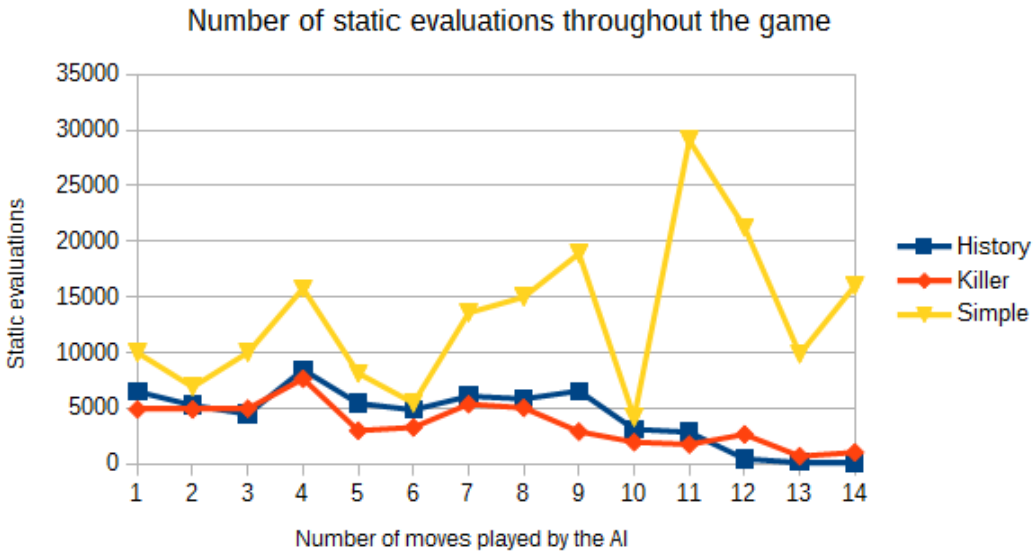
	History		Killer		**Simple**	
	Moves	Static eval	Moves	Static eval	**Moves**	Static eval
Me	E4					
AI	E4	6506	E4	4915	**E4**	9992
Me	E5					
AI	B1	5268	B1	4914	**B1**	6917
Me	C2					
AI	H4	4486	H4	4975	**H4**	10004
Me	F4					
AI	B4	8472	B4	7660	**B4**	15785
Me	C5					
AI	D5	5446	D5	3003	**D5**	8154
Me	C4					
AI	C3	4864	C3	3293	**C3**	5473
Me	B2					
AI	E7	6096	E7	5380	**E7**	13580
Me	G5					
AI	F2	5853	F2	5049	**F2**	14997
Me	H2					
AI	G3	6560	G3	2910	**G3**	18916
Me	H5					
AI	F5	3121	F5	1966	**F5**	4179
Me	E2					
AI	D2	2864	D2	1761	**D2**	29080
Me	G4					
AI	E3	470	D3	2661	**D3**	21260
Me	E3					
AI	D4	166	A3	700	**A3**	9880
Me	B3					
AI	D4	133	D4	1041	**D4**	16025

Figure 2: Experimentation against AI, depth = 4

In this game, the AI has won by placing four pieces in a row.

First, we can observe that the number of static evaluations has dramatically increased by incrementing the depth (multiplied by ten).

In this game, at the beginning, all of the AIs agreed on the move to play. Nevertheless, at the end, the one using the History Heuristic disagreed twice with the others. I observed the cells played and both of the choices were valid : they would make the AI win either on the next move or on the move after that.



Then, we can compare the static evaluations made by the two AIs using heuristics. On the graph above, the History heuristic is rather less efficient than the Killer heuristic at the beginning of the game, but becomes the more efficient at the end.

7. Conclusion

Depending on the stage of the game, using one of the heuristics can be more or less efficient than using the other. Anyway, using any of these heuristics improves significantly the speed of the program and can sometimes lead to a different choice of move.