University of Saskatchewan
Department of Computer Science
176 Thorvaldson Building
110 Science Place, Saskatoon, SK, S7N 5C9, Canada
Telephine: (306) 966-4886, Facimile: (306) 966-4884

Assignment 4

Games!

Date Due: 26 November 2018, 11:59pm **Total Marks: 60**

General Instructions

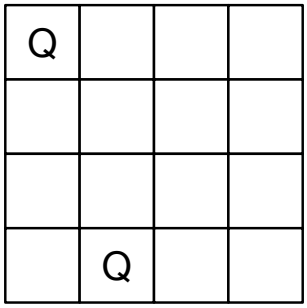
- **This assignment is individual work.** You may discuss questions and problems with anyone, but the work you hand in for this assignment must be your own work.
- Each question indicates what to hand in.
- Do not submit folders, or zip files, even if you think it will help.
- **Assignments must be submitted to Moodle.**

Version History

- **20/11/2018**: Corrected the statement about utility values for Variation 1a, when every column has a queen.
- **20/11/2018**: Fixed a few typos.
- **15/11/2018**: Clarified the meaning of the data in the table in Q1.
- **15/11/2018**: Reduced the number of files to hand-in. Please check the new requirements.
- **6/11/2018**: released to students

Overview

The N-Queens problem is to put N Queens on a board ($N \times N$) so that no queen attacks any other queen. (A queen can attack any square on the same row, same column, or same diagonal.) The following is a diagram of a game state with 2 queens on the board:



We will make a 2-player game of the N-Queens problem as follows: Players alternate putting a piece on the board, and can only place a queen on a square that is not being attacked, i.e., there is no queen already on the board in the same row or column or diagonal. In the N-Queens *game*, the objective is to be the last player to have a legal place to put a queen. We will work with two variations of this game, which differ in ~~2 aspects: how players can play, and~~ how the score is counted.

Variation 1a The players must place a queen on the left-most unoccupied column: Player 1 plays on Column 1, Player 2 on Column 2, Player 1 on Column 3, etc). For example, in the above diagram, it's Player 1's turn to move, and Player 1 must place a queen somewhere in column 3 (counting from the left, starting at 1). The winner is the last player to put a queen on the board. For example, if it's Player 1's turn to play in Column 5, but all squares in Column 5 are being attacked, then Player 1 loses, and Player 2 wins. The utility function would return 1 for a win for Player 1, or -1 for a win for Player 2. For example, Player 1 can place a queen in the above position, but Player 2 would be unable to play after that, so Player 1 would be the winner. There are no draws.
Note: If all N queens are on the board, then the last player to move is the winner: if N is even, then Player 2 wins, and the utility is ~~1~~ -1, and if odd, Player 1 wins, and the utility is ~~−1~~ 1.There are no draws.

Variation 1b This variation is identical to Variation 1a, except for the score of the game. The utility value is calculated by counting how many queens are on the board at the end of a game: if there are k queens on the board, and it's a win for Player 1, the utility is k (positive, odd number); if there are k queens on the board, and it's a win for Player 2, the utility is $-k$ (negative, even number). For example, the win for Player 1 (after playing one more queen on the board position above) counts for 3. There are no draws.
Note: If all N queens are on the board, then the last player to move is the winner: if N is even, then Player 2 wins, and the utility is $-N$, and if odd, Player 1 wins, and the utility is N.

Notes

- The purpose of this assignment is not to implement a real game, but to understand the game-tree search algorithms.
- These variations are games, but may not be interesting games to play as humans.
- Some of the variations of this game may not seem *fair* or *balanced*, and that's okay for our work.
- For $N < 3$, the game is quite boring.
- Draw out the game trees for $N \leq 4$ before you start coding. You'll need these trees to debug your code.

Programming

As in previous assignments, it's advisable to separate the game-specific code from the game-search code. The interface between your game-search code and your game-specific code is as follows:

- `is_terminal(state)`: returns a Boolean, True if the given state has no more legal moves.
- `utility(state)`: returns a numeric value. The state must be a terminal state.
- `actions(state)`: returns a list of actions (i.e., moves) that are legal in the given state. The function needs to examine the state to determine whose turn it is to move.
- `result(state, action)`: returns a new game state which is the result of taking the given action in the given state.
- Optional: `is_maxs_turn(state)`: Returns a Boolean value, True if it's Player 1's turn (Max) in the given state.
- Optional: `is_mins_turn(state)`: Returns a Boolean value, True if it's Player 2's turn (Min) in the given state.

Implementing the optional methods may not be necessary for all the questions, though it would be useful if you wanted to play the game interactively. We saw the use of these optional functions in Lecture 18.

None of the coding is difficult, though finding a bug may be difficult. As always, you should write enough tests for your methods and functions that you can debug them using a test script. Debugging a whole application is more of a waste of your time than writing a test script.

You'll be developing a code base throughout this assignment, and we only want to see it once to grade it. Each question will ask you to hand in a script that performs a task; this script can import the code you're developing, but don't hand everything in multiple times.

Execution instructions

The markers may or may not wish to run your program, to verify your results. To help them, you should provide brief instructions on what to do to get your program running. Include:

- Programming language used (including version, e.g., Java 8 or Python 3)
- A simple example of compiling and/or running the code from a UNIX shell would be best.

Keep it brief, and name it with the question number as the following example: `a4q1_EXECUTION.txt`. If your assignment uses third party libraries, they have to be included in your submission.

Clarification: You'll submit one file documenting the execution instructions for all your scripts.

Clarification: What to Hand in instructions

The first version of this assignment required separate hand in files for questions 1-7,9. This pushes against Moodle's very hard limit of 20 files maximum. To remedy this, the new requirements collect the requirements for these questions into 2 files:

- `A4_output.txt` will contain tables output from your scripts for questions 1-7,9. Please mark each Question's output clearly, so markers can easily identify which data belongs to which question.
- `A4_EXECUTION.txt` will contain execution instructions for all your scripts for questions 1-7,9. Please mark each Question's information clearly, so markers can easily identify which information belongs to which question.

You should not hand in separate output and execution instructions!

Question 1 (4 points):

Purpose: To implement Minimax Search for Variation 1a.

Degree of Difficulty: Moderate. Minimax is easy; the game will be the only tricky part.

Textbook: AIMA Chapter 5.1, 5.2

Implement Minimax search (with no bells or whistles) and apply it to Variation 1a, for game sizes $N \in \{1, \cdots, 10\}$. Specifically, write a loop to display the following table:

Size	Minimax Value	Best Opening Move	Time in Seconds
1	1	`('X', 0)`	3.72e-05
2	1	`('X', 0)`	6.52e-05
3	1	`('X', 1)`	0.0001642
4	-1	`('X', 0)`	0.000388
⋮	⋮	⋮	⋮

Clarification: In the above output, which is literally copy/paste from my computer, the Best Opening Move is represented by a pair `(WHO, ROW)`; in my implementation, I used the symbol `'X'` to represent player 1 (answering WHO), and for ROW, I give an integer value. There is no need to imitate my output, as long as it's clear.

Your output should go all the way up to 10 (or 11, if that amuses you). Your output should have the correct Minimax values, though there may be more than one opening move to achieve that value; it does not matter which of the equally good opening moves your script returns. Your times will vary, but reasonably you should expect your implementation to take only a few seconds at most. Try a few larger sizes, e.g. $N = 11, 12$, and observe the exponential nature of the search problem (again).

What to Hand In

- ~~A file named `a4q1.txt` containing your table, as above.~~
- **Revised:** Your table, and any accompanying explanation, in your file `A4_output.txt`, which is a single file that will contain your output for all questions. Be sure to mark clearly the question number as you add information to this file.
- A file named `a4q1.LANG` containing a script/app which, if executed, would produce the output submitted in `a4q1.txt` (except the timing information which will differ). Use the file extension appropriate for your choice of programming language, e.g., `.py` or `.java`.
- ~~A file named `a4q1_EXECUTION.txt` containing execution instructions for markers to run your script/app if they deem it necessary.~~
- **Revised:** Execution instructions for markers to run your script/app if they deem it necessary, clearly presented in a the file `A4_EXECUTION.txt`, which is a single file that will contain your execution instructions for all questions. Be sure to mark clearly the question number as you add information to this file.

Do not submit your entire codebase multiple times. Your script/app `a4q1.LANG` should run correctly, but it should not contain code for Minimax search, and should not contain the code for your implementation of Variation 1a (or any others).

Clarification: You'll be developing your code-base as you work on Questions 1-9. You will hand in your whole code base in Question 10. Your script for Question 1 will import modules/classes/libraries from (or include, or compile with) your code base, of course! But you don't need to hand in the game code or the search code in Question 1. Just hand in the script that creates the data above. When the marker runs your script, it should import or compile with the code you submit in Question 10.

Be sure to include your name, NSID, student number, and course number at the top of all documents.

Evaluation

Revised:

- 3 marks: Your file `A4_output.txt` contains the required table, with correct Minimax values, and plausible times, clearly marked as a4q1.
- 1 mark: Your file `a4q1.LANG` contains a script/app that produces the output in `A4_output.txt`.

Question 2 (4 points):

Purpose: To implement Minimax Search for Variation 1b.

Degree of Difficulty: Easy. Minimax is done already; you have to modify the utility function for the new variation.

Textbook: AIMA Chapter 5.1, 5.2

Implement Minimax search (with no bells or whistles) and apply it to Variation 1b, for game sizes $N \in \{1, \cdots, 10\}$. Specifically, write a loop to display the following table:

Size	Minimax Value	Best Opening Move	Time in Seconds
1	1	`('X', 0)`	5.70e-05
2	1	`('X', 0)`	4.91e-05
3	1	`('X', 1)`	8.61e-05
4	-2	`('X', 0)`	0.000281
⋮	⋮	⋮	⋮

Your output should go all the way up to 10 (or 11, if that amuses you). Your output should have the correct Minimax values, though there may be more than one opening move to achieve that value; it does not matter which of the equally good opening moves your script returns. Your times will vary, but reasonably you should expect your implementation to take only a few seconds at most. Try a few larger sizes, e.g. $N = 11, 12$, and observe the exponential nature of the search problem (again).

What to Hand In

- ~~A file named `a4q2.txt` containing your table, as above.~~
- **Revised:** Your table, and any accompanying explanation, in your file `A4_output.txt`, which is a single file that will contain your output for all questions. Be sure to mark clearly the question number as you add information to this file.
- A file named `a4q2.LANG` containing a script/app which, if executed, would produce the output submitted in `a4q2.txt` (except the timing information which will differ). Use the file extension appropriate for your choice of programming language, e.g., `.py` or `.java`.
- ~~A file named `a4q2_EXECUTION.txt` containing execution instructions for markers to run your script/app if they deem it necessary.~~
- **Revised:** Execution instructions for markers to run your script/app if they deem it necessary, clearly presented in a the file `A4_EXECUTION.txt`, which is a single file that will contain your execution instructions for all questions. Be sure to mark clearly the question number as you add information.

Do not submit your entire codebase multiple times. Your script/app `a4q2.LANG` should run correctly, but it should not contain code for Minimax search, and should not contain the code for your implementation of Variation 1a (or any others).

Be sure to include your name, NSID, student number, and course number at the top of all documents.

Evaluation

Revised:

- 3 marks: Your file `A4_output.txt` contains the required table, with correct Minimax values, and plausible times, clearly marked as a4q2.
- 1 mark: Your file `a4q2.LANG` contains a script/app that produces the output in `A4_output.txt`.

Question 3 (4 points):

Purpose: To implement Minimax Search with Alpha-Beta Pruning for Variation 1a.

Degree of Difficulty: Moderate. Your implementation of Variation 1a is done; you have to implement Alpha-Beta pruning.

Textbook: AIMA Chapter 5.3

Implement Minimax Search with Alpha-Beta Pruning and apply it to Variation 1a, for game sizes $N \in \{1, \cdots, 10\}$. Specifically, write a loop to display the following table:

Size	Minimax Value	Best Opening Move	Time in Seconds
1	1	`('X', 0)`	3.60e-05
2	1	`('X', 0)`	3.98e-05
3	1	`('X', 1)`	8.51e-05
4	-1	`('X', 0)`	0.000322
⋮	⋮	⋮	⋮

Your output should have the the same Minimax values as Question 2, though there may be more than one opening move to achieve that value; it does not matter which of the equally good opening moves your script returns. Your times will vary, but reasonably you should expect your implementation to take less time than for Question 2, especially for larger values of N. Try a few larger sizes, e.g. $N = 11, 12$, and observe the exponential nature of the search problem (again).

What to Hand In

- ~~A file named `a4q3.txt` containing your table, as above.~~
- **Revised:** Your table, and any accompanying explanation, in your file `A4_output.txt`, which is a single file that will contain your output for all questions. Be sure to mark clearly the question number as you add information to this file.
- A file named `a4q3.LANG` containing a script/app which, if executed, would produce the output submitted in `a4q3.txt` (except the timing information which will differ). Use the file extension appropriate for your choice of programming language, e.g., `.py` or `.java`.
- ~~A file named `a4q3_EXECUTION.txt` containing execution instructions for markers to run your script/app if they deem it necessary.~~
- **Revised:** Execution instructions for markers to run your script/app if they deem it necessary, clearly presented in a the file `A4_EXECUTION.txt`, which is a single file that will contain your execution instructions for all questions. Be sure to mark clearly the question number as you add information.

Do not submit your entire codebase multiple times. Your script/app `a4q3.LANG` should run correctly, but it should not contain code for Minimax search, and should not contain the code for your implementation of Variation 1a (or any others).

Be sure to include your name, NSID, student number, and course number at the top of all documents.

Evaluation

Revised:

- 3 marks: Your file `A4_output.txt` contains the required table, with correct Minimax values, and plausible times, clearly marked as a4q3.
- 1 mark: Your file `a4q3.LANG` contains a script/app that produces the output in `A4_output.txt`.

Question 4 (4 points):

Purpose: To implement Minimax Search with Alpha-Beta Pruning for Variation 1b.

Degree of Difficulty: Moderate. Your implementation of Variation 1b is done; you have to implement Alpha-Beta pruning.

Textbook: AIMA Chapter 5.3

Implement Minimax Search with Alpha-Beta Pruning and apply it to Variation 1b, for game sizes $N \in \{1, \cdots, 10\}$. Specifically, write a loop to display the following table:

Size	Minimax Value	Best Opening Move	Time in Seconds
1	1	`('X', 0)`	3.60e-05
2	1	`('X', 0)`	3.98e-05
3	1	`('X', 1)`	8.51e-05
4	-2	`('X', 0)`	0.000322
⋮	⋮	⋮	⋮

Your output should have the the same Minimax values as Question 3, though there may be more than one opening move to achieve that value; it does not matter which of the equally good opening moves your script returns. Your times will vary, but reasonably you should expect your implementation to take less time than for Question 2, especially for larger values of N. Try a few larger sizes, e.g. $N = 11, 12$, and observe the exponential nature of the search problem (again).

What to Hand In

- ~~A file named `a4q4.txt` containing your table, as above.~~
- **Revised:** Your table, and any accompanying explanation, in your file `A4_output.txt`, which is a single file that will contain your output for all questions. Be sure to mark clearly the question number as you add information to this file.
- A file named `a4q4.LANG` containing a script/app which, if executed, would produce the output submitted in `a4q4.txt` (except the timing information which will differ). Use the file extension appropriate for your choice of programming language, e.g., `.py` or `.java`.
- ~~A file named `a4q4_EXECUTION.txt` containing execution instructions for markers to run your script/app if they deem it necessary.~~
- **Revised:** Execution instructions for markers to run your script/app if they deem it necessary, clearly presented in a the file `A4_EXECUTION.txt`, which is a single file that will contain your execution instructions for all questions. Be sure to mark clearly the question number as you add information.

Do not submit your entire codebase multiple times. Your script/app `a4q4.LANG` should run correctly, but it should not contain code for Minimax search, and should not contain the code for your implementation of Variation 1a (or any others).

Be sure to include your name, NSID, student number, and course number at the top of all documents.

Evaluation

Revised:

- 3 marks: Your file `A4_output.txt` contains the required table, with correct Minimax values, and plausible times, clearly marked as a4q4.
- 1 mark: Your file `a4q4.LANG` contains a script/app that produces the output in `A4_output.txt`.

Question 5 (4 points):

Purpose: To implement a depth-cutoff and a heuristic evaluation function for Minimax Search, to allow play in Variation 1a larger than $N = 10$.

Degree of Difficulty: Easy. You have to modify your search algorithm, using the interface for imperfect real-time decisions.

Textbook Chapter: AIMA 5.4

Implement a depth-cutoff and heuristic evaluation function for Variation 1a. Apply your implementation with a cut-off at depth 4, for game sizes $N \in \{1, \cdots, 20\}$. Specifically, write a loop to display the following table:

Size	Minimax Value	Best Opening Move	Time in Seconds
1	1	`('X', 0)`	3.60e-05
2	1	`('X', 0)`	8.92e-05
3	1	`('X', 1)`	0.000143
4	-1	`('X', 0)`	0.000280
⋮	⋮	⋮	⋮

Since the cut-off is at depth 4, the first 4 rows in your output should have correct minimax values; after that, the minimax values will reflect your evaluation function, and probably won't be optimal. Your times will vary, but reasonably you should expect your implementation to take less time than for Question 2, especially for larger values of N. In particular, you should expect to run all examples in a few seconds at most.

Additional functions for your interface. The textbook recommends that you

1. Replace your test for a terminal state with a function that determines whether to cut off search.
2. Replace the utility function with a function that estimates utility for any given state.

I recommend otherwise! Add two new methods/functions to your Variations, as follows:

- `cutoff_test(state, depth)`: returns a Boolean, True if the search should be terminated at this state and depth.
- `eval(state)`: returns an estimate of the Minimax value of the given state.

By introducing these two new functions into the interface, your search algorithms need to be modified as well. Use the cutoff test and the evaluation function in combination, but only check for the cut-off after you've determined that the state is not an actual terminal state. This arrangement adds a few lines to the search algorithms, but helps keep your utility and estimated minimax functions independent, which is good for development.

The heuristic evaluation function. Such a function might be difficult to invent for this game. You are welcome to try. You should remember that your estimate should return a value in the range of your utility function; a confident estimate should be close to the true minimax value, and you can express uncertainty by producing estimates that are between the extremes, perhaps using fractional (floating point) estimates within the range. **Alternatively, your evaluation function can return a random utility value. This is easy to start with, in any case.**

What to Hand In

- ~~A file named `a4q5.txt` containing your table, as above.~~
- **Revised:** Your table, and any accompanying explanation, in your file `A4_output.txt`, which is a single file that will contain your output for all questions. Be sure to mark clearly the question number as you add information to this file.
- A file named `a4q5.LANG` containing a script/app which, if executed, would produce the output submitted in `a4q5.txt` (except the timing information which will differ). Use the file extension appropriate for your choice of programming language, e.g., `.py` or `.java`.
- ~~A file named `a4q5_EXECUTION.txt` containing execution instructions for markers to run your script/app if they deem it necessary.~~
- **Revised:** Execution instructions for markers to run your script/app if they deem it necessary, clearly presented in a the file `A4_EXECUTION.txt`, which is a single file that will contain your execution instructions for all questions. Be sure to mark clearly the question number as you add information to this file.

Do not submit your entire codebase multiple times. Your script/app `a4q5.LANG` should run correctly, but it should not contain code for Minimax search, and should not contain the code for your implementation of Variation 1a (or any others).

Be sure to include your name, NSID, student number, and course number at the top of all documents.

Evaluation

Revised:

- 3 marks: Your file `A4_output.txt` contains the required table, with correct Minimax values, and plausible times, clearly marked as a4q5.
- 1 mark: Your file `a4q5.LANG` contains a script/app that produces the output in `A4_output.txt`.

Question 6 (4 points):

Purpose: To implement a depth-cutoff and a heuristic evaluation function for Minimax Search, to allow play in Variation 1b larger than $N = 10$.

Degree of Difficulty: Easy. You have to implement a cutoff and evaluation function for Variation 1b. Your changes to Minimax from Question 5 should already be finished.

Textbook Chapter: AIMA 5.4

Implement a depth-cutoff and heuristic evaluation function for Variation 1b. Apply your implementation with a cut-off at depth 4, for game sizes $N \in \{1, \cdots, 20\}$. Specifically, write a loop to display the following table:

Size	Minimax Value	Best Opening Move	Time in Seconds
1	1	`('X', 0)`	4.70e-05
2	1	`('X', 0)`	3.89e-05
3	1	`('X', 1)`	8.30e-05
4	-2	`('X', 0)`	0.000338
⋮	⋮	⋮	⋮

Since the cut-off is at depth 4, the first 4 rows in your output should have correct minimax values; after that, the minimax values will reflect your evaluation function, and probably won't be optimal. Your times will vary, but reasonably you should expect your implementation to take less time than for Question 2, especially for larger values of N. In particular, you should expect to run all examples in a few seconds at most.

The notes about the interface, and the evaluation function, from Question 5, apply here as well. If you've adopted a good design, you may only need to over-ride the evaluation function from Question 5.

What to Hand In

- ~~A file named `a4q6.txt` containing your table, as above.~~
- **Revised:** Your table, and any accompanying explanation, in your file `A4_output.txt`, which is a single file that will contain your output for all questions. Be sure to mark clearly the question number as you add information to this file.
- A file named `a4q6.LANG` containing a script/app which, if executed, would produce the output submitted in `a4q6.txt` (except the timing information which will differ). Use the file extension appropriate for your choice of programming language, e.g., `.py` or `.java`.
- ~~A file named `a4q6_EXECUTION.txt` containing execution instructions for markers to run your script/app if they deem it necessary.~~
- **Revised:** Execution instructions for markers to run your script/app if they deem it necessary, clearly presented in a the file `A4_EXECUTION.txt`, which is a single file that will contain your execution instructions for all questions. Be sure to mark clearly the question number as you add information to this file.

Do not submit your entire codebase multiple times. Your script/app `a4q6.LANG` should run correctly, but it should not contain code for Minimax search, and should not contain the code for your implementation of Variation 1a (or any others).

Be sure to include your name, NSID, student number, and course number at the top of all documents.

Evaluation

Revised:

- 3 marks: Your file `A4_output.txt` contains the required table, with correct Minimax values, and plausible times, clearly marked as a4q6.
- 1 mark: Your file `a4q6.LANG` contains a script/app that produces the output in `A4_output.txt`.

Question 7 (4 points):

Purpose: To implement a depth-cutoff and a heuristic evaluation function for Minimax Search with Alpha-Beta pruning, to allow play in games larger than $N = 10$.

Degree of Difficulty: Easy. You'll need to adapt your Alpha-Beta search algorithm similar to the way you adapted Minimax in Question 5.

Textbook Chapter: AIMA 5.4

Adapt your Alpha-Beta search algorithm from Q4 similar to the way you adapted Minimax in Question 5. Apply your implementation with a cut-off at depth 4, for game sizes $N \in \{1, \cdots, 20\}$. Specifically, write a loop to display the following tables:

Variation 1a

Size	Minimax Value	Best Opening Move	Time in Seconds
1	1	`('X', 0)`	3.41e-05
2	1	`('X', 0)`	3.81e-05
3	1	`('X', 1)`	0.000106
4	-1	`('X', 0)`	0.000440
⋮	⋮	⋮	⋮

Variation 1b

Size	Minimax Value	Best Opening Move	Time in Seconds
1	1	`('X', 0)`	4.70e-05
2	1	`('X', 0)`	3.89e-05
3	1	`('X', 1)`	8.30e-05
4	-2	`('X', 0)`	0.000338
⋮	⋮	⋮	⋮

Since the cut-off is at depth 4, the first 4 rows in your output should have correct minimax values; after that, the minimax values will reflect your evaluation function, and probably won't be optimal. Your times will vary, but reasonably you should expect your implementation to take less time than for Question 2, especially for larger values of N. In particular, you should expect to run all examples in a few seconds at most.

The notes about the interface, and the evaluation function, from Question 5, apply here as well. If you've adopted a good design, you may only need to over-ride the evaluation function from Question 5.

What to Hand In

- ~~A file named `a4q7.txt` containing your table, as above.~~
- **Revised:** Your table, and any accompanying explanation, in your file `A4_output.txt`, which is a single file that will contain your output for all questions. Be sure to mark clearly the question number as you add information to this file.
- A file named `a4q7.LANG` containing a script/app which, if executed, would produce the output submitted in `a4q7.txt` (except the timing information which will differ). Use the file extension appropriate for your choice of programming language, e.g., `.py` or `.java`.
- ~~A file named `a4q7_EXECUTION.txt` containing execution instructions for markers to run your script/app if they deem it necessary.~~

- **Revised:** Execution instructions for markers to run your script/app if they deem it necessary, clearly presented in a the file `A4_EXECUTION.txt`, which is a single file that will contain your execution instructions for all questions. Be sure to mark clearly the question number as you add information to this file.

Do not submit your entire codebase multiple times. Your script/app `a4q7.LANG` should run correctly, but it should not contain code for Minimax search, and should not contain the code for your implementation of Variation 1a (or any others).

Be sure to include your name, NSID, student number, and course number at the top of all documents.

Evaluation

Revised:

- 3 marks: Your file `A4_output.txt` contains the required table, with correct Minimax values, and plausible times, clearly marked as a4q7.
- 1 mark: Your file `a4q7.LANG` contains a script/app that produces the output in `A4_output.txt`.

Question 8 (10 points):

Purpose: To compare the output gathered in the previous questions.

Degree of Difficulty: Easy. You'll need to understand why the output is the way it is.

Textbook Chapter: AIMA 5.1-4

Answer the following questions with a brief comment for each.

1. Compare the Minimax values for A4Q1 and A4Q2. The minimax value used a different scale (-1,1) versus a count of the number of queens, but did this change which player won the game?
2. Compare the runtimes for A4Q1 and A4Q2. Which variation seemed to be faster? Explain why.
3. Compare the Minimax values for A4Q1 and A4Q3. Were they the same?
4. Compare the runtimes for A4Q1 and A4Q3. How much faster was the search, for games with larger N?
5. Compare the Minimax values for A4Q2 and A4Q4. Were they the same?
6. Compare the runtimes for A4Q2 and A4Q4. How much faster was the search, for games with larger N?
7. Compare the Minimax values for A4Q1 and A4Q5. Were they the same? Explain.
8. Compare the runtimes for A4Q1 and A4Q5. How much faster was the search, for games with larger N?
9. Compare the Minimax values for A4Q5 and A4Q6 (Minimax with cut-off, two variations) with the results from A4Q7 (Alpha-Beta with cut-off, two variations). Were they the same? Explain.
10. Compare the runtimes for A4Q5 and A4Q6 (Minimax with cut-off, two variations) with the results from A4Q7 (Alpha-Beta with cut-off, two variations). How much faster was the search, for games with larger N?

None of these questions is difficult. This is more about requiring you to consider what you've done, than it is about understanding anything deep or complex.

What to Hand In

- A file named `a4q8.txt` containing your comments to the above questions.

Be sure to include your name, NSID, student number, and course number at the top of all documents.

Evaluation

- 10 marks: Your file `a4q8.txt` contains plausible responses to the above questions.

Question 9 (6 points):

Purpose: To implement an interactive script/application that plays N-Queens.

Degree of Difficulty: Moderate. You'll need to return actions, not just values, and you'll need to distinguish between the scripts that plan moves for a single player (Q1-6), and scripts that take turns making moves in a real game.

Textbook Chapter: AIMA 5.1-4

This is the fun part. Implement an interactive script/application that plays N-Queens. Make use of your implementations from the previous questions.

Make sure you understand the difference between what the search algorithms are doing (planning a single choice), and what you need to do here (allow two planning agents to take turns making moves). You might start by implementing human vs. computer (or computer vs. human), so that you can figure things out, then replace the human player with the computer.

You should allow each player to use its own search algorithm, so you could play (plain) Minimax against Alpha-Beta, using equal depth cutoffs, or different depth cutoffs. You can model a *weak* player using a shallower cutoff, and a stronger player using a deeper cutoff. Play around with the cutoffs, so that the first few moves of the game do not take longer than a minute or so (shorter is better for experimentation though).

Try the following scenarios:

1. Try $N = 10$, equal cut-off value 4. Compare the outcome of your games with the known Minimax values (as you determined in Q1-Q4).
2. Try $N = 10$, equal cut-off value 5. Compare the outcome of your games with the known Minimax values (as you determined in Q1-Q4). Were these the same as when the cutoff was 4?
3. Try $N = 10$, Player 1 cut-off 5, Player 2 cut-off 3. Compare the outcome of your games with the known Minimax values (as you determined in Q1-Q4). Did the deeper cutoff result in better decisions?
4. Try $N = 10$, Player 1 cut-off 3, Player 2 cut-off 5. Compare the outcome of your games with the known Minimax values (as you determined in Q1-Q4). Did the deeper cutoff result in better decisions?
5. Repeat the above scenarios with $N = 20$.

If you are using a random evaluation function, be sure to run each scenario a number of times (you don't want to draw a conclusion about random effects based on one example)! If you are using a deterministic evaluation function, then there is no point to repeat the scenarios more than once.

Report your results in a tabular format, e.g.:

N	Cut-off P1	Cut-off P2	Win/Loss P1
10	4	4	7W, 3L
10	3	5	2W, 8L
⋮	⋮	⋮	⋮

What to Hand In

- ~~A file named `a4q9.txt` containing your table, as above.~~
- **Revised:** Your table, and any accompanying explanation, in your file `A4_output.txt`, which is a single file that will contain your output for all questions. Be sure to mark clearly the question number as you add information to this file.
- A file named `a4q9.LANG` containing a script/app which, if executed, would produce the output submitted in `a4q9.txt` (except the timing information which will differ). Use the file extension appropriate for your choice of programming language, e.g., `.py` or `.java`.
- ~~A file named `a4q9_EXECUTION.txt` containing execution instructions for markers to run your script/app if they deem it necessary.~~
- **Revised:** Execution instructions for markers to run your script/app if they deem it necessary, clearly presented in a the file `A4_EXECUTION.txt`, which is a single file that will contain your execution instructions for all questions. Be sure to mark clearly the question number as you add information to this file.

Do not submit your entire codebase multiple times. Your script/app `a4q9.LANG` should run correctly, but it should not contain code for Minimax search, and should not contain the code for your implementation of Variation 1a (or any others).

Be sure to include your name, NSID, student number, and course number at the top of all documents.

Evaluation

Revised:

- 3 marks: Your file `A4_output.txt` contains the required table, with correct Minimax values, and plausible times, clearly marked as a4q9.
- 1 mark: Your file `a4q9.LANG` contains a script/app that produces the output in `A4_output.txt`.

Question 10 (16 points):

Purpose: To obtain grades for your implementations.

Degree of Difficulty: Easy. Just make sure your code is well-documented, well-organized, and hand it in. Nothing new here!

Textbook Chapter: AIMA 5.1-4

Hand in your code. All the scripts from the previous questions should be based on this code, by importing or compiling as necessary. All the scripts from previous questions must run using these files.

Your code should be documented internally so that a reader can make sense of it.

What to Hand In

- Files named `a4q10.LANG` containing the code for your variations, and your search algorithms. If you have more than one file to submit (a good idea), you can name them appropriately, using `a4q10` as a prefix.

Do not submit your entire codebase multiple times. This is the only place where your variations and search algorithms are submitted.

Be sure to include your name, NSID, student number, and course number at the top of all documents.

Evaluation

- 2 marks: Your implementation of Variation 1a includes functions/methods for the interfaces described throughout this assignment.
- 2 marks: Your implementation of Variation 1a is well-documented.
- 2 marks: Your implementation of Variation 1b includes functions/methods for the interfaces described throughout this assignment.
- 2 marks: Your implementation of Variation 1b is well-documented.
- 2 marks: Your implementation of Minimax Search makes use of the interface described through the assignment.
- 2 marks: Your implementation of Minimax Search is well-documented.
- 2 marks: Your implementation of Minimax Search with Alpha-Beta Pruning makes use of the interface described through the assignment.
- 2 marks: Your implementation of Minimax Search with Alpha-Beta Pruning is well-documented.

Work for the ambitious

We worked with two variations of the N-Queens game. There is a third, which I removed from the assignment.

In Variation 2, the players are **not** restricted in their choice of column, but can play a queen in any row-column position, so long as it is not currently attacked by any other queen already on the board. For example, in the above diagram, it's Player 1's turn to move, and Player 1 can place a queen in either of the two unoccupied columns. The winner is the last player to put a queen on the board. The utility function would return 1 for a win for Player 1, or -1 for a win for Player 2. There are no draws.

Implement this variation, and try out your search algorithms on it. The branching factor is much higher, and moves are more strategic. This variation is a good opportunity to employ a *transposition table*, which stores board positions and utility values so that if the board position appears more than once in a search, the full search is not repeated. Use a hash table!