CS 6360: Advanced AI
Assignment 2

Due: 03/03/2017, at midnight

General Instructions:

If anything is ambiguous or unclear:

1. **Discuss possible interpretations with other students, your TA, and instructor**
2. **Make assumptions, state them explicitly, and then use them to work out the problem**
3. **Use Piazza for discussions among yourselves, and also for questions, also for questions and clarifications you need from the instructor and the TA. Piazza levels the playing field because the responses to questions asked are of interest to all, and shared by all. If you have very specific questions, you may email the TA**

Remember that after general discussions with others, you are required to work out the problems by yourself. All submitted work must be your own work. Please refer to the Honor code for clarifications.

Playing the Game of Hex

Programming Assignment. (150 points)

For this assignment you will implement the Game of Hex, a strategy board game for two players played on a hexagonal grid, theoretically of any size and several possible shapes, in your case you will develop the game for an 8×8 grid.[1] The game was invented by Piet Hein in 1942 while a student at Niels Bohr's Institute for Theoretical Physics, and subsequently and independently by John Nash in 1948 while a mathematics graduate student at Princeton.

In the game, one player plays white pieces, while the other plays black (in the figure it is red and blue), with play alternating between players and placement only allowed on unoccupied hexagons. Alternate sides of the board are designated white and black (or red and blue in the figure), and the goal of the game is to complete a unbroken chain of pieces between one player's two sides. The game cannot end in a draw since no chain can be completely blocked except by a complete chain of the opposite color.

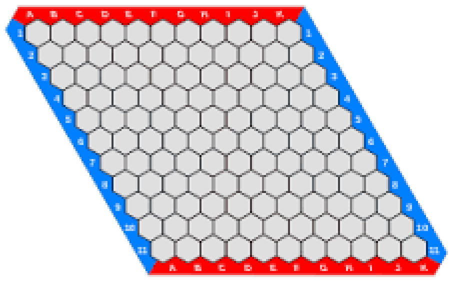

Figure 1: Standard 11x11 Hex Board

[1] See Wikipedia page: https://en.wikipedia.org/wiki/Hex_(board_game)

Previously, Hayward et al. [2005][2] gave an algorithm strong enough to solve any 7×7 board game state in Hex, i.e., they solved all 49 7×7 openings (one stone states) with search trees totaling 14.2 million nodes. Rasmussen et al. [2007][3] verified these results more efficiently, requiring fewer than 0.5 million nodes, but were unable to solve any 8×8 states.

You can develop your program in a language of your choice (e.g., C++, Java, Python, etc.), but your game have the capability of playing in two modes: (1) your program versus a human, where the human enters their moves by typing in the position to place the next coin; and (2) your program should be able to play against another one you have written. You will implement two versions of your program:

1. The first one implements the **minimax algorithm with alpha-beta pruning** using a heuristic evaluation function.
2. The second one is primarily implements a **Monte Carlo Tree search algorithm**, i.e., the **UCT algorithm** discussed in class for two player games. In this version of the program, you may employ strategies and patterns to implement cutoffs on inferior and dead end board configurations.

   For both versions of the program, be clear how you apply termination conditions and the game winner. (Remember you should be able to decide win/loss in certain configurations even before a full chain is completed).

For your alpha-beta search algorithm, you should develop your program so you can report the following:

1. The evaluation function and its justification in terms of the max-min strategy ((e.g., either try to maximize opponent's cost to win or try to minimize self-cost to win). Your evaluation function can be pattern based, and you can detect and cut off below dead end and (justified) inferior configurations.
2. Initial threshold of the search depth, and how you applied a selective search expansion function, if you did.
3. The following statistics: (1) number of nodes evaluated and (2) max size of the search tree (you should update the search tree when applying pruning).
4. Report of at least 5 games played by humans against the program. (it may be helpful to ask friends that are willing to play against your program, so you can observe the results and tune your evaluation functions.

For your MCTS algorithm, you should develop your program so you can report the following:

1. Report the search parameters, for example: (a) threshold for the expansion depth, (b) the simulation method employed, (c) the evaluation function used for selection steps (e.g., UCB1 or its variations), (d) your expansion policy (whether you expand multiple nodes or one node at each iteration.) and (e) the policy for implement for picking the most promising move for each iteration (examples, max number of simulations, max number

[2] Hayward, R., Björnsson, Y., Johanson, M., Kan, M., Po, N., & Van Rijswijck, J. (2005). Solving 7× 7 Hex with domination, fill-in, and virtual connections. *Theoretical Computer Science*, *349*(2), 123-139.

[3] Rasmussen, R., Maire, F., & Hayward, R. (2007, December). A template matching table for speeding-up game-tree searches for Hex. In Australasian Joint Conference on Artificial Intelligence (pp. 283-292). Springer Berlin Heidelberg.

of wins, highest win rate, highest win rate and highest number of simulations associated with the move, and its variations). You should try different approaches and study their consequences by comparing them against each other.

2. Report statistics, such as: (a) the number of simulations performed per move (to study efficiency of different approaches) and (b) the max size of the search tree for all moves (as an estimate of memory used).
3. Play against this configuration yourself (or get friends to play against your program) so that you can evaluate, learn, and refine you approach to playing.

Remember, a key part of this assignment is to play the AI (knowledge-heavy) versus the MCTS (knowledge-light), and carefully evaluate the pros and cons of each. Play them against each other using a time limit per move. You can vary this time limit at different stages of the games, but the amount of time per move at various stages should be predetermined before you start the competition. Don't just look at wins and losses but try to evaluate critical moves to explain the differences in approach. Try to optimize each algorithm before competition. You will report the win rates for the competition, giving each an equal number of times to play white and black.

A thorough discussion of your approaches, how you refined them as you implemented and played the games, and your discussion of the final results obtained is a very important component of this assignment.

Since your implementation will involve playing against opponents, it is important to print out the board configuration, after every move that the program makes. When you play AI against MCTS, you need not print out the board configurations after every turn.

Also, remember, you can use references from papers and web sources on approaches and strategies that have been developed for playing Hex. But you **cannot use code developed by others as part of your implementation**. This includes all sources – the web, papers and reports, and of course, other students code. (Doing so will be considered a violation of the honor code).

You will also provide the TA with instructions on how to run your program, as a separate document from the Project Report.

Suggestions from class notes and Blackboard:

1) Board representations (http://www.redblobgames.com/grids/hexagons/)
2) Paper references and papers uploaded to Blackboard regarding various approaches to the game of Hex.

Submission: You will submit your code (source and executable), you documentation, and your report that includes the results of your experiments as zipped file on Blackboard.