**CS 165A – Artificial Intelligence, Fall 2018**

**Machine Problem 2**
(100 points)

**Due *Wednesday, December 5, 2018, 11:59pm***

---

**Notes:**

- Make sure to re-read the "Policy on Academic Integrity" on the course syllabus.
- Any updates or corrections will be posted on the Announcements page in web page of the course, so check there occasionally.
- You have to work individually for this assignment.
- Each student must turn in their report and code electronically.
- Responsible TA for Machine Problem 2: Yanju Chen yanju@cs.ucsb.edu
- Visit **http://cs.ucsb.edu/~cs165a** for tournament status and related updates.

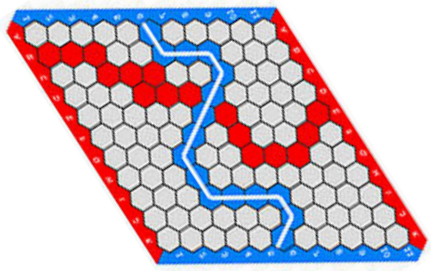
---

# 1. Problem Definition

You are going to implement a program that plays Hex (a strategy board game) with a human opponent. Your program should be able to search for the best move so that it ends up winning over a human opponent (or at least make the match very challenging). Towards this end, you should explore the **Minimax** algorithm and other adversarial search algorithm improvements, like alpha-beta pruning, in order to quickly calculate the program's next move and eventually win.

# 2. Hex

Hex is a strategy board game for two players played on a hexagonal grid, theoretically of any size and several possible shapes, but traditionally as an 11×11 rhombus. Players alternate placing markers or stones on unoccupied spaces in an attempt to link their opposite sides of the board in an unbroken chain.



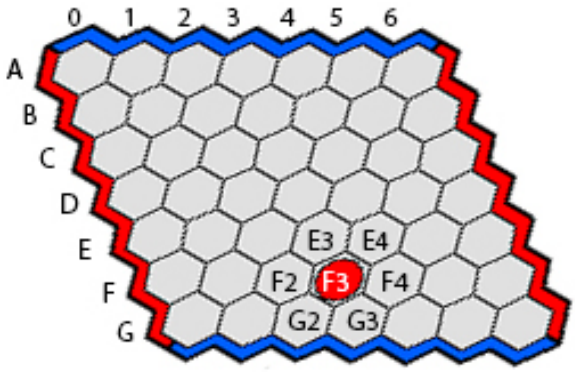
You can also watch this video for more information on playing Hex: https://www.youtube.com/watch?v=2MNalT1g3m8

To play Hex for free, visit http://www.lutanho.net/play/hex.html

# 3. Technical Details

Your program should be able to process the following two optional command line arguments:

- `-p <ai_color>`: This option is to specify the AI's color (either RED or BLUE). If the option is not specified, the AI will be playing RED and the human player will be playing BLUE. In both cases, the RED players move first.
- `-s <board_size>`: This option is to specify the board size. The game board in this problem is always a square. We will allow sizes from 1×1 to 26×26, but usually test on some more common settings such as 7×7 and 11×11. If s is not specified, the default value should be 7×7.




**The Default Game Settings:**

In this machine problem, we have the following default game settings for better evaluation of all the programs:

1) Every row is represented by a single letter in upper case: A, B, …, Z;
2) Every column is represented by a single integer: 0, 1, …, 25;
3) To locate a position, use the form <letter><integer>: D1, C2, A25, etc.;
4) RED always plays first;
5) RED always tries to link the side with letter indexing, and BLUE always tries to link the side with integer indexing, as shown by the game board above.

Once the execution begins, your program should be able to interact with the human player through a command line interface:

1) For AI's turn, your program should output the position indicating the choice of the AI's move in a line.
2) For human player's turn, your program should wait for human player's input of a position.
3) AI and human player take turns during the game play.
4) Your AI should not take more than 30 seconds to decide on its new move;
5) The human players are allowed as much time as they need to decide their moves.

For example, a valid game log of the above 7×7 game board (AI as RED) can be:

```
D3
E4
D2
D1
```

where D3 and D2 are outputs of AI, and E4 and D1 are human player inputs. Notice that your program only needs to output AI's position calculated by your algorithm, and the human player's moves are read form the screen/STDIN. **You don't need to print the human player's moves.**

You will also be provided a '*HexReferee*' program that will allow for two programs to compete with each other! Technically, each program will have no knowledge that it plays with another program, they will both think that they compete with a human. The referee script will be redirecting the moves of each program to the command line interface of the other program to simulate these human players. Using the referee, you should be able to test your program with its own self as the opponent. The referee will also enable having a *tournament* to find which implementation performs the best, given 30 seconds per move (more information on the tournament section).

Note: In order to render your program compatible with the referee you need to have an executable binary and abide by the output rules. Your program must be compatible with the referee for grading as well.

**Executable:**
Please also provide an executable wrapper script that just executes your code. Name this script "HexPlayer.sh". For C/C++ you might have the following simple script:

```
#!/bin/bash
./HexPlayer $@
```

for Python:

```
#!/bin/bash
python ./HexPlayer.py $@
```

and for Java:

```
#!/bin/bash
java ./HexPlayer $@
```

The above scripts will just execute your code by simply running the wrapper scripts like this:

```
./HexPlayer.sh -p RED -s 11
./HexPlayer.sh -p BLUE -s 13
```

The first script initializes an 11×11 game board with AI as RED playing first. The second script initializes a 13×13 game board with AI as BLUE and human playing first. These scripts also allow command line arguments which will be directly passed to your code. (Notice: Python 3 is also available on CSIL. Type `python3` in command line to execute it.)

**Output:**
In every turn, your program must either:

- print the position of the AI's choice of move in a line, or
- wait and read the human player's move from screen/STDIN

**Do not print anything if it's the human player's turn**. At the end, when either player wins or there are no valid moves left, your program should output a message indicating the game status (whatever message you prefer) and exit. You can check the outputs of the provided programs to see how your output should look like in order to be compatible with what the referee script expects. A valid game log is displayed in the above game settings section.

**Implementation Restrictions:** You are not allowed to use any third party frameworks or non-native libraries for your implementation. You can use standard and native libraries that are installed on CSIL for every user (the definition of standard or native libraries includes any piece of code that does not require installation or downloading). Also keep in mind that your submission can't exceed 10MBs.

# 4. Instructions on What and How to Submit

Use the CSIL **`turnin`** command to submit the necessary files and directories. Note that all directory and file names are case sensitive. For this assignment you need to issue the following command:

```
% turnin mp2@cs165a mp2
```

Make sure your output format is checked by the validation program before you turn in your work. Once it's successfully turned in, you will see a confirmation message on screen like:

```
*** TURNIN OF mp2 TO cs165a COMPLETE! ***
```

otherwise, please check your network or other possible issues.

Once you are finished developing your code, copy all necessary source files (including header files, source code files, the pdf report, makefile, etc.) into a directory named "mp2". The grader will compile and execute your programs. So, your code must compile without referencing any external files or libraries that are not included in your submission. Standard header files, such as iostream.h, or other native libraries and modules are fine to use as long as they are available in CSIL.

You may use C/C++, Java, or Python (both 2 and 3 are available on CSIL) for this programming assignment. You may use any OS/compiler version for development, but your final code must compile and run on the CSIL machines. **So make sure you compile and run the final version of your code on CSIL machines before turning it in.**

# 5. Detailed Submission Guidelines

- C/C++: Include a "Makefile" which builds the program by default (e.g. typing "make" in the mp2 directory should build the HexPlayer executable program), your source code files (.cpp, .c, .hpp), your header files ".h" if you have any, and any other files that are needed to compile and run your code successfully.
- Java: Include your source code file ".java", class files ".class", an executable wrapper script named "HexPlayer.sh" that executes your java code, and any other files that are needed to build and run your code successfully.
- Python: Include your source code files ".py", an executable wrapper script named "HexPlayer.sh" that executes your python code, and any other files that are needed to run your code successfully.

In all cases, regardless of language, the grader should be able to just run "make" (if code is in C/C++) and then execute the "HexPlayer.sh" executable.

Note: Put all the necessary files in a directory named mp2 and submit the directory as a whole. When the grader extracts your submission all your submitted files should be in mp2/.

# 6. Report Preparation

As for the report, it should be between 1 and 3 pages in length (no more than 3 pages) with at least 11pt font size and should consist of at least the following sections:

- **Architecture:** Brief explanation of your code architecture (describe the classes and basic functionality)
- **Search:** How you search for the best move (which algorithm you use, what heuristics, optimizations, etc)
- **Challenges:** The challenges you faced and how you solved them
- **Weaknesses:** Weaknesses in your method (you cannot say the method is without flaws) and suggest ways to overcome them.

Note that the most important part is the Search section. There should be only one pdf report which includes all the above (no additional readme file).

# 7. Grading

You will be graded based on your program's ability to win other AI programs! The main weight of the grade (60pts) will be based on your program's performance so you should make your search as optimal as possible. To evaluate the gameplay performance of your program, we will have it compete with our own implementations of 3 different baselines. If your program wins all of them, then you should expect to get the full 60pts. The three

baselines are provided to you so that you can test your own program before the final submission. Note that the grading platform is the Linux machines in CSIL. **Make sure that your program does not not exceed the 30 seconds per move rule on these machines (it might be running faster in your laptops).**

Grade Breakdown (**100pts**):

- **20pts** Complete Report
- **10pts** Makefile and execution specifications
- **10pts** Correct program specifications: command line arguments, no segfaults or exceptions, no logic bugs during gameplay
- **60pts** Gameplay performance (based on the number of games your program wins against our baselines)
  - **30pts** if your program wins at least 4 out of 7 games against baseline#1 (random move, 7×7)
  - **20pts** gameplay performance against baseline#2 (depth 2 minimax, 7×7)
    - **10pts** if your program wins at least 1 out of 7 games as RED player
    - **10pts** if your program wins at least 1 out of 7 games as BLUE player
  - ***n* pts (up to 10pts)** you get $n$ points where $n$ is the number of games your program wins out of 10 games against baseline#3 (optimized search, 11×11)
  - There's a time limit of 30 seconds per move in all the games. Programs failed to decide a move within time limit will lose the game immediately. Any invalid output of your program that does not abide by the output rules will also fail your game.

**An important notice about your algorithm:**

Even though Hex has winning strategies for some game settings (e.g., 7×7 game board), you are **NOT** allowed to use any of these winning strategies. However, you can learn or implement your heuristic functions based on these winning strategies, and your implementation should be based on search algorithms.

*Plagiarism Warning: We are going to use software to check plagiarism. You are expected to finish the project by yourself without using any code from others.*

# 8. Tournaments!

Participation in the Tournaments is optional but can gain additional credit (and fame!). There will be two kinds of tournaments, a pre-submission tournament and a post-submission tournament.

- The pre-submission tournament will have a defending champion. Every time you submit your code, if your submission contains the text file TOURNAMENT, your program will be tested versus the current defending champion. If you win, you become the defending champion. If you do not win, the defending champion will remain the same. To become the defending champion, you also need to beat our simple AI programs which form the baseline, so you can't become the defending champion just by implementing random moves. The defending champion of all the tournament participants before every Tuesday and Friday 10pm will get 3pts extra credit for MP2 (plus the FAME of being the defending champion). Matches will be performed in batches on a regular basis depending on the load, so do not flood the turnin submission system or you will get banned from the tournament (only submit once you have the results of your previous submission, otherwise only your latest submission will be used).
- The post-submission tournament will be a tournament between all final submissions that contain the text file TOURNAMENT (if you have multiple submissions, the latest one with TOURNAMENT file will be used). The program that wins the tournament gets 50% extra credit. The two runner-ups get 30% and 20% (for second and third position). The post-submission tournament will be held after the deadline.

**Hex match setup:** A single match consists of 9 games (3 sets). Every set consists of 3 games. There are one set with board size of 7, another set with board size of 11, and the other with board size of 13. In every set, the challenger AI will play RED in 2 games, and the defending champion will play RED in 1 game.

**Winning a match:** The winner of a match is the player that has at least 5 wins (best out of 9). In case of a tie in a game during the tournament the defending champion gets the win. If you tie with our programs during grading it will count as your win.

**Information updates:** visit **http://cs.ucsb.edu/~cs165a** for more update information.