**CS 4701 Pre-Proposal**

**I. The Team**

*Team name*: password123
*The members:* Benjamin Jaeger, bgj9, Brian Wojcik, bmw75, Areeb Malik, amm385
*Language:* Python
*Team web page:* http://cs4701.wikispaces.com/password123
*Team repository:* https://github.com/amm385/password123

**II. Problem statement and motivation**

The mission here is to build an ingenious bot that, with a superior AI algorithm, will crush all other CS4701 Chinese Checkers playing bots and make them wish they had never been compiled. Our algorithm will direct our pieces from our starting location to the starting location of our opponent's before our opponent accomplishes the same.

**III. I/O Specification**

The input/output is specified by the procedures outlined in the project description. This will consist of communication with an external game server. Our program will initially take in data from the server regarding the game layout. Before each move, our program will input the opponent's most recent move along with a notification that we are to move next. After calculation, we will output to the server our move and if necessary, input any errors that take place. We will also output our own internal game analysis and metrics to terminal.

**IV. Background Reading**

[1] Paula Ulfhake, *A Chinese Checkers-playing program*, Lund University, 2000. hem.passagen.se/baolan/release/china.pdf
[2] Chinese Checkers. Ashish Gupta. Northwestern University. 10 Sept. 2011. http://www.cs.northwestern.edu/~agupta/_projects/chinese_checkers/web/

**V. General Approach**

We will implement the Minimax Algorithm to search for our next move based on the current board configuration. This general algorithm requires a *position evaluation function* to assign a value to each possible move loosely based on the pieces' total distance to the finish. The goal is to maximize our position function and to minimize our opponent's. In other words, Minimax considers both offensive and defensive strategies.

*Before start of game:*

At the start of the game, prior to either side making a move, we will analyze the board and create optimal paths for our pieces to move over. To do so we will create a custom *position evaluation function* specific to the board's layout. Board locations that are "closer" to the opponents starting location will yield a higher value. The board could contain a number of obstacles, so we will use the Shortest Path Algorithm to determine which locations are actually "closer" and which locations should be avoided.

*During game:*

The game rules specify that the total allowed calculation time for each turn is ten

minutes. Our algorithm will look as many turns into the future as time allows for. We will use various optimizations to look even farther. For example, in the first few rounds, our opponent's moves do not need to be considered - their initial moves have minimal effect on our game play. Throughout the whole game, we will use Alpha-Beta Pruning to discount moves that are not worthwhile. Near the end of the game, an important strategy in checkers is to not leave a few marbles behind (A.K.A. stragglers) while moving the rest across the board. The stragglers will not have other marbles to jump over and hence will have a longer traversal time. Minimax is unable to anticipate this late game phenomenon since it can only look at a limited amount of turns into the future. To solve this problem, we will tweak the *position evaluation function* to dis-incentivize stragglers.

*Analysis of game progress:*

We will keep certain metrics in order to understand how our strategies are playing off in a given game. For instance, a simple metric such as estimated number of moves until completion can be used to determine game progress. We can take this number and compare it against what we would expect / how we performed against a test bot to determine how well strategies are doing. We will also keep track of our performance relative to our opponent in a similar manner.

Another possible metric would be a rate-of-play factor, which would incorporate the average number of jumps per move in a given time period. This would help to ensure that we're not lagging in speed at any time. Knowing our rate will be crucial to evaluating how current strategies are performing.

We are likely to develop more metrics once we start experimenting with our algorithms and determining variables that may need monitoring and adjustment.

*Adaptation to game progress:*

We will use the above metrics to adjust our game play in order to optimize our algorithms. For instance, if our progress to success is significantly slower than expected, we will alter our strategies and attempt to either (a) slow down our opponent (possibly at the sacrifice of our own speed) or (b) speed up our bot (possibly at the sacrifice of our defense). We will likely use many future metrics to adjust our strategies in the future as well.

**VI. Schedule**

General Schedule:

9/13: Preliminary proposal planned detailing approach to project and planned methods of solving the problem statement above.

9/16: Alterations to proposals made; Proposal finalized.

10/4: Working bot created, using minimax tree to make best offensive move. Time Permitted, incorporate defensive moves and blocking.

11/1: Bot is now fully functional and makes both offensive and defensive moves. Altering/ tweaking how the bot performs in circumstances will occur at this time. What the bot looks for and optimizes at each move will be experimented with at this time. Scripting 2 bots to play against each other with slightly different priorities will help determine optimal strategy and what to stress/focus on.

11/15: Optimization will occur at this time if close to calculation time limit, including pruning

unnecessary move calculations.
11/29: Final presentations with demo.

If Progress is quick and ahead of schedule:
Create versions of bot that prioritize different play styles, i.e. focusing on blocking rather than making long chains. Collecting and Analyzing game outcome data from these bots will help find optimal play style.

If Progress is slow and behind schedule:
Sacrifice pruning abilities and focus on getting a working bot, reducing the number of steps looked ahead.

Primitive Personal Schedule through Working bot:

Brian: Learn Python, storing game board information, path-finding.
Areeb: Position Evaluation Function.
Ben: Minimax backend and structure.