Nikita Kiriy (nk327)
Juan Pablo Sarmiento (jps327)

09/19/2011
CS 4701

AI Proposal

Our goal is to make an adaptive AI player that can win a game against another "smart" AI. Our project will be split into two stages: **stage one** is to construct an AI player with an understanding of the rules and the board before the game starts, that can beat a random player by heuristic move choices and projections; **stage two** is to make a bot that can also analyze an opponent's strategy in real time and adapt to it.

At the start of the game, our bot will take as an input the board layout and any obstacles that may have been placed. At each turn, it will take as an input the position of all pieces, and of any neutral gray "special" pieces that may have been placed. The output at each turn will be the next move to perform.

The class textbook will be used for high level techniques and design. However, in order to create a smart AI in this game, we need to be able to handle very large branching factors. Therefore, we will also be using external resources, such as reading papers on AI systems developed for games with large branching factors, such as Go. There is a large amount of literature available related to making computer Go players, and some of the more abstract strategies developed for playing these games can be adapted for Chinese checkers. For instance, alpha-beta pruning is used to cut down the search tree for Go players; the tree is effectively reduced to 2 or 3 branches and then searched to a large depth. Other helpful external resources would be to find papers on programs that have been built to play the traditional Chinese Checkers game.

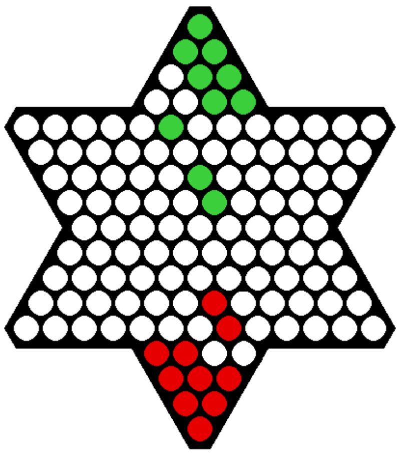
Our approach to building this AI will follow the stages we described above. For the first stage (basic rules and heuristic move choices) we plan to have the algorithm learn from a data set of recorded games. Once this has been complete, we will move on to the next stage of our AI: one that can make predictions of what the opponent's next moves could be and adapt to these forecasts. This could be done by running the move decision algorithm on the opponent's position or using a neural net to model enemy behavior, but more research needs to be done to plan this stage out.

We've identified several components to the system that can be independently developed and tested. First, we need to create a time distribution decision system: given that we have a limited time of 10 minutes we need to make sure that our AI always knows how much time it has left, and how much time to spend in planning and in decision making each turn. We also need a learning system that can train off of previous games. A game system can also be developed independently - this would include the entire game logic and rules of the game, in order to ensure that our AI can play the game correctly. Finally, we also need a shortest-path finder system that can take obstacles into account (in case we are given a board where certain locations cannot be used) - this will be an essential component to calculating the costs of taking different paths and can serve as a heuristic.

We've mentioned how we plan to train our AI using previously recorded games, so it is important to determine where this data would come from. One source would come from having friends play against our bot, for which we would need to create the interface that will allow a human to play against the AI. A way to make sure that the training set for the AI is made up of good games is to use AI-human games where the AI loses--this way the algorithm can continue improving. And posting the program online may help collect a sufficiently large data set for training.

In order to verify the progress of our AI and it's problem-solving capabilities, we will need to come up with some metrics in order to measure it's performance. First we will need to estimate the average time needed per move and the overhead planning time, our goal will be to minimize these times so that we can do as many moves possible in the 10 minutes that are given to us. Next, we need to ensure that the memory usage does not become so large that it causes the program to crash. Finally, although the current description of the game says that the idea of a victory margin has been "temporarily" abandoned, we will still be measuring the victory margin and working to improve this. The reason is that a victory margin can still be used as a metric to see how well our AI bot was doing up until the point that its 10 minutes ran out.

We will also develop test cases on which we can run our AI to measure it's performance. We will be using the concept of a "n-move checkmate." As in, we will give the AI a prearranged board with a winning strategy composed of one or two moves. We can use this "toy" problem to make sure that our AI determines the winning moves. The advantages of this testing strategy is that we can then backtrack, and begin giving the AI a prearranged board composed of more moves in order to make the test cases harder. With each new prearranged board we can always check that our AI determines the winning moves and use our previously mentioned metrics to evaluate it's performance.


Easy test case, red player has a two move win.

The schedule we will be following in order to make sure that all stages of our AI are completed on time:

Component	**Date**
Program that can move pieces.	Sep 25th
Game logic, something that can barely play	Oct 4th (1st code review)
Something that can play better Have interface for friends to play against out bot	Oct 18th
Stage 1 complete, working on adaptive AI	Nov 1st (2nd code review)
Make AI better (can win games now).	Nov 15th
Present examples of our (adaptive) AI playing.	Nov 29th (final presentation)