**Results:**

In the end, I was only able to create a mini-max and mini-max-beta-alpha algorithm for a tic-tac-toe AI program and a mini-max algorithm for a Checkers AI program. The mini-max and mini-max-beta-alpha algorithm for the Tic-Tac-Toe AI program was unbeatable, but not optimal (see **Philosophical Lessons Learned** below).

*Results that demonstrated comparable human intelligence:*

Within the Tic-Tac-Toe and Checkers programs, I created a method that played an AI program (that uses the mini-max algorithms) versus another AI program that uses the same algorithms. In the Checkers game, I noticed an interesting observation when I changed the depth to which the AI programs can see into the future. If the depth was >=5, the AI program that moved second would always make sure to keep its last row occupied with its pieces to prevent the first player from getting a King. This type of habit is consistent to what some expert checkers players do. When I changed the depth to <=3, both AI players played extremely defensive, making sure to keep all of their pieces as bunched together as possible. They took very few chances to kill the opposing player’s pieces if it had to sacrifice its own killing piece in the next turn. This once again, demonstrated another habit that is consistent with expert checkers players as well as human psychology. If a human player is attempting to win a 2 player game, but does not have the ability to predict the next possible moves; than it seems intuitive for the human players to play defensive. The increase in depth to >=5 allowed the AI to see future opportunities at certain positions that a lower depth could not provide; therefore, allowing the AI to take more chances.

*Results that demonstrated inferior AI intelligence to human intelligence:*

The Checkers AI program was able to play “intelligently” for the majority of the game until only a few remaining pieces were left for either side. At that point, my AI program would not make any intelligent moves with its King pieces. I believe this problem occurred because my depth was too low and/or my heuristic evaluation function was too simplistic. This always occurred in situations in the game where all of the AI pieces were more than 4 rows from an opposing player’s piece or the promotion row. I was only able to go up to a depth of 8 without making my program run slow. For all depths from 1 to 8, the end results for an AI vs AI checkers games always ended in a situation where one AI player clearly had an advantage in terms of the number of pawns and kings it had; however, it would not make the obvious move to promote its pawns in order to win the game. The other AI player had 1 or 2 King pieces left and no pawns left; however, it would not make obvious moves to kill the opposing player’s pawn pieces.

**Lessons Learned:**

My AI program for checkers was very basic. I only used mini-max algorithm when additional methods (evaluator functions, quiescence, end-game database, utilizing existing proving start moves of checkers grandmasters). The reason I did not consider these additional methods was because of the mistakes I made that caused me to run out of time. These mistakes and lessons I learned are below. My AI program also did not consider chain kills > 2 since when I tried to implement my program with > 2 chain kills, there was not enough stack memory within the IDE to make it past the first MiniMax turn.

I attempted to create an unbeatable Checkers program from the start with the assumption that I knew everything I needed to know regarding how to program this 2 player game. However, the mistake I made was that I rushed to quickly into the programming of the project. Because of this, I wasted too much time trying to correct and restructure my code when my structure of the code was not optimal or theoretically correct. I assumed certain aspects of the project would be more complicated than it actually was and I also made assumptions that certain difficult aspects of the project would be easy. For example, I assumed I needed a database to store intermediate possibilities; however, the cache memory for Netbeans was enough to make the AI program run efficiently for up to 8 moves on a 12 X 12 board checkers game for MiniMax and up to 12 moves for Alpha-Beta modification. I also assumed the implementation of the Mini-Max algorithm would take up most of my time, but it turned out to be that the algorithm for generating all possibilities for the next states of a checker’s game board was the most time consuming.

I had to completely disregard my Checkers AI program and restart from scratch. I believed if I had created my AI tic-tac-toe program first, then creating my Checkers AI program would have been exponentially quicker. The main structure for my Tic-Tac-Toe program was entirely similar to my Checkers AI program. Since Tic-Tac-Toe was a simpler and smaller game than Checkers, it made it easier for me to create the structure of my program than trying to visualize it from a Checkers stand point.

**Theoretical Lessons Learned:**

One of the main aspects of the Artificial Intelligence lecture that I continue to have thoughts on has been the topic of Human Intelligence vs AI intelligence. In the first few AI assignments that discussed Human Intelligence vs AI intelligence, I discussed differences between how AI approaches problems and how a Human approaches problem. In addition, I also questioned whether human max potential is the limit to what AI programs can achieve. I questioned if it is possible if there are missing components that allows AI potential to exceed human potential, and that human beings are just not capable of understanding what these missing components are and how to implement these missing components. After analyzing the results of my AI programs for checkers and Tic-Tac-Toe, I believe one of the missing components to lowering the perceived gap between Human potential and AI potential is self-learning and the impact of psychology.

I chose to base my final project on 2 player state games because I believed this topic would provide more insight into my thoughts on this matter. In particular, the various 2 player state games that either have or not have an unbeatable AI program were basic and simplistic to discuss about. From the 5 two player games (Checkers, GO, Chess, Tic-Tac-Toe, Connect 4) that I analyzed for this final project, only 3 (Tic-Tac-Toe, Connect 4, Checkers) out of the 5 two player games have been claimed to be unbeatable. Tic-Tac-Toe and Connect 4 unbeatable programs were able to be designed by algorithms in polynomial time. Checkers unbeatable program required exponential time to solve. Regarding my thoughts about how I believe self-learning and psychology are important topics in my discussion of Human intelligence vs AI intelligence, I discus a specific example of a situation that I observed during a Tic-Tac-Toe game between myself and an AI.

In this example, if the AI were to go first, the heuristic values that it assigned for all 9 positions on the Tic-Tac-Toe board were equal. It assigned equal heuristic values because it assumed that its opponent was also unbeatable due to the mini-max algorithm. In a game between a human vs. human, the first move on a tic-tac-toe board that tend to provide victories are at the corner edges of the board. If the second player does not place his piece at the middle square and if the first player places his first piece at a corner, then the second player is certain to lose if the first player is unbeatable. If an AI were to go second, its move will always be placed at the middle square since the mini-max algorithm will score this middle square the highest on the second turn. “Naïve” human tic-tac-toe players tend to not realize that the necessary move to make on the second turn is in the middle square. I realized that my unbeatable AI program was not optimal in terms of how it could maximize its win rate versus “naïve” human players. It gives up opportunities to win easier because it assumed its opponent is unbeatable. As a result, my AI program won’t differentiate the difference between a corner edge and a middle edge.

The example discussed above about the Tic-Tac-Toe problem comes back to how we can bridge the gap (where the gap is referring to how a player is able to make obvious choices based on past experiences or psychological analysis of opponents) between AI players and human players. I believe, as mentioned briefly in the first paragraph, that a self-learning function is the key that helps make up the link between AI and a human. In assignment 4, I discussed how AI uses probability to base its decisions on past statistics. A large part of the reason why we cannot come up with an unbeatable program for GO and Chess is because these types of problems would be considered NP problems in my opinion. Since no one has been able to prove whether P = NP, there actually could exist an unbeatable algorithm for GO and Chess but we have not been able to find an NP problem that could reduce to an unbeatable algorithm that is solvable by a theoretical Turing machine. The next time I decide to create an AI program for a two player zero sum game, I would like to consider how I can apply a self-learning modification to my program as another aspect to how the AI will make decisions.