**Pentago AI: Final Report**

**Team Egg and Cheese (Croissants Optional)**

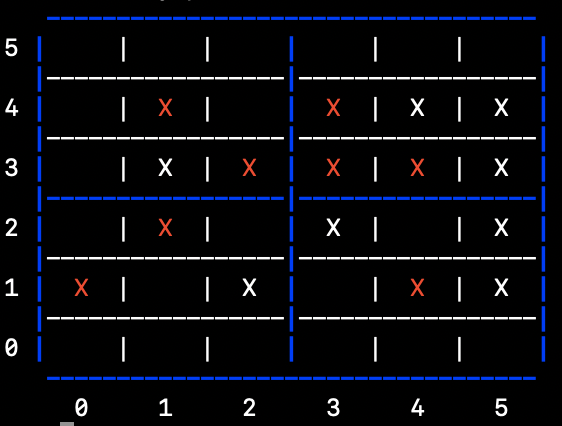
Karson Daecher (kbd45), John DeMoully (jjd268), Richard Greenbaum (rrg73), Max Melamed (mbm259)

## **Keywords**

Minimax, Adversarial Search, Alpha-Beta Pruning, Gameplay Bot(s), Bot vs. Bot, Gameplay Analysis, Pentago

## **Section 1: Project plan**

For our practicum project, we set out to create a program that plays the board game Pentago. Like chess and Go, Pentago is a two player, deterministic, perfect knowledge, zero sum game. However, the smaller board leads to simpler game play and a more manageable search tree. The game board is a 6x6 square broken into four smaller 3x3 quadrants. The object of the game is to get 5 marbles in a row before the opponent does, and a turn consists of placing a marble in an empty hole and then twisting one of the 3x3 blocks by 90 degrees in either direction. A picture of a real Pentago board is shown below next to the representation of the same board by our program.

**Project Objectives**

Goal 1: Our most basic goal was to create a program that users would enjoy playing against. This requires the program to be able to play with a basic level of competence; users should be able to tell that they are playing against a bot that does more than choose a random move.

Goal 2: Our more challenging but primary goal was to create a program that is very difficult to beat. Success by this metric could be seen by a near perfect win rate against new players, a very high win rate against players with some experience, and no losses before 7 moves have been played by each side.

Goal 3: After doing some research and stumbling upon the Perfect Pentago website (<https://perfect-pentago.net/>), we added one final stretch goal. This website, built by DeepMind researcher and Stanford PhD Geoffrey Irving, has an interactive explorer for perfect Pentago play that allows the user to set up any Pentago position on a game board and shows, for all possible moves from that position, whether perfect play from then on would result in a win, loss, or tie. This website also shows that Pentago is a first player win game, meaning that the player that goes first can always win with optimal play no matter what the other player does. Thus our stretch goal was to create a bot that plays perfect Pentago, meaning that it would always win when moving first. This could be tested by loading positions reached in games against our bot into the interactive Pentago explorer and seeing if the opponent has any moves that would lead them to a win or a tie. If so, we know that our bot must have deviated from the optimal move sequence that would guarantee a winning outcome.

## **Section 2: System Overview**

**Program Execution**

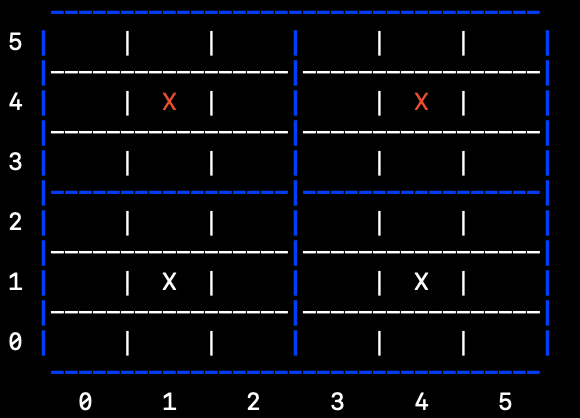
Our program is written in python and is run in terminal using the command “python \_\_main\_\_.py”. The file \_\_main\_\_.py controls the interaction with the user as well as the game flow. The file calls upon functions in GamePlay.py which handle move making and board manipulations. Upon running the main file, a paragraph is shown with the rules of the game and the user is asked if they want to go first as well as the search depth they would like the bot to use. Then the game begins.

**Evaluation Function**

The core component of our bot’s decision making is an evaluation function which takes in a game board and returns a number representing how good that position is for the bot. Positive numbers indicate a strong position for the bot (bad for the player) and negative positions indicate a bad position for the bot (good for the player). The evaluation function is based on counting marble sequences for both competitors. It loops through every row, column, and game winning diagonal (a diagonal with 5 or more spaces) and counts the number of sequences of length 2, 3, 4, and 5 for the player and for the bot. It then creates a score for the player and for the bot by multiplying the number of sequences of each type by a constant defined for each sequence length, with larger sequence lengths corresponding to a larger point value. Finally, the function returns the bot score - player score.

In addition to this baseline calculation, the evaluation takes into account a few nuanced metrics discussed below.

* Center marble bonus: The evaluation function gives a bonus for each marble that is in the center of its tile. These locations are particularly valuable because tile rotations do not affect their position. This bonus leads our bot to choose the center spots first.
* Strong vs weak sequence: The evaluation function gives a higher score to sequences of 2 and 3 that are contained on the same tile than to sequences of 2 and 3 that are spread across multiple tiles. This is because these ‘strong’ sequences cannot be split up by tile rotations.
* Impossible Sequences: This is the most powerful add-on to our evaluation function. It does not give credit for sequences in rows 1 and 4 or columns 1 and 4 unless the player/bot controls two adjacent center spots. Similarly it does not give credit for sequences in either of the two primary diagonals if the player/bot does not control two diagonal center spots. This is because such sequences cannot result in a win without control of the two proper center spots. To see why, observe the common opening position below:



It is white’s move. Without this add-on, our bot would view moves (2,2) and (2,1) equally, as they both result in a strong sequence of length 2. However clearly the move (2,1) should be chosen, as a sequence across the main diagonal cannot result in a win without control of the (4,4) spot.

* Center Control: Having marbles in the center of the board is better than having marbles on the edges because they have a greater influence in creating and blocking sequences. Thus our function gives a small bonus for every marble that is not on the board’s perimeter.

**Minimax Algorithm**

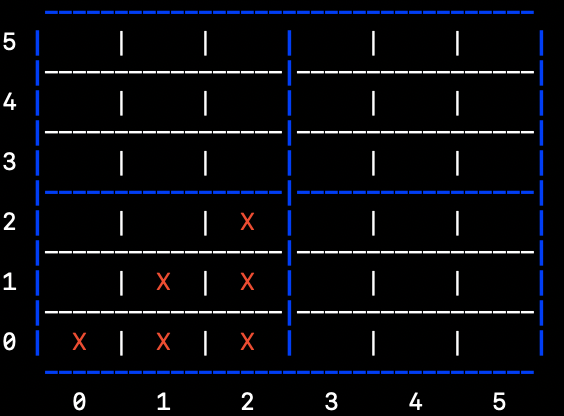
To make our decisions as to which move are the most advantageous considering our evaluation function outlined above, we utilized the minimax algorithm. As Pentago is a zero sum game, in this case, our bot is trying to maximize the evaluation function on its turns while utilizing the same evaluation function to minimize the opposing players ensuing move opportunities.

The flow of the minimax algorithm relies on the ability to forecast gameplay for each combination of possible moves up to a certain depth. This means that we are looking at each possible marble position, each possible quadrant to rotate as well as a direction to rotate it for both our bot as well as the opposing player. The actual structure of our algorithm is very similar to that outlined in chapter 5 of the 4th edition of *Artificial Intelligence: A Modern Approach* by Russell & Norvig[[1]](#footnote-0). Minimax is a very standard algorithm for two player adversarial games and was further improved with the addition of both Alpha-Beta and symmetric pruning.

**Pruning**

Pruning is a key component of our system’s performance. Our algorithm recognizes 288 possible opening moves (36 spots \* 4 tiles \* 2 rotation directions). This means that a minimax search to depth 2 requires evaluating ~83,000 board positions and a search to depth 3 requires evaluating ~24 million board positions. Without pruning, a depth 2 search and evaluation takes several seconds and depth 3 in unattainable in a reasonable amount of time. To cut down on the number of positions our bot needs to analyze, we use two types of pruning: symmetric pruning and alpha beta pruning.

Symmetric Pruning: While the number of possible moves at each depth seems high, the number of unique moves is far lower, especially in the beginning of the game when the board is relatively empty. As stated above, our algorithm recognizes 288 opening moves. However, **there are only 6 unique opening move outcomes!** All other moves are identical to one of these 6 outcomes. The 6 unique opening move outcomes are shown below.



For symmetric pruning, we keep track of all of the boards that have already been evaluated at a certain depth. Before evaluating a new move, we generate the board it would lead to as well as all of the boards that are identical to it. This involves rotations of the board, flipping the board over its horizontal and vertical axes, and flipping the board over its two primary diagonals. If any of these boards are equal to a board that has already been checked, we stop evaluation and continue on to the next move. However, it turns out that the array manipulations required to generate the symmetric boards for each move takes more time than it saves through pruning. For this reason we commented out the code that creates the board symmetries and only check if the board directly created by the current move has already been evaluated.

Alpha Beta Pruning: Alpha beta pruning is a different type of pruning that stops the evaluation of a subtree when it has been found that this subtree cannot influence the final decision. This presents the potential for huge increases in efficiency, as it allows for entire subtrees to be cut off from further exploration. As stated in the Wikipedia[[2]](#footnote-1) article on Alpha Beta Pruning, the expected number of leaf nodes evaluated for alpha beta pruning when the best moves are searched first is O(bd/2) where b is the branching factor and d is the depth of the search. Substituting in 288 for b and 3 for d, we would expect the number of nodes evaluated with optimal alpha beta pruning to be reduced from ~24 million to under 5,000. Thus to maximize the benefit of alpha beta pruning, we try to get our search to start with the best moves by first evaluating all of the moves at depth 1. Then we sort the moves by decreasing depth 1 evaluation evaluation and then begin a minimax algorithm with pruning at depth 3. After implementing this boosted version of alpha beta pruning, our bot was able to make decisions at depth 3 after just 5-15 seconds depending on the board position.

**“Perfect” Pentago**

We luckily found that the solved Pentago website built by Dr. Irving had a public API that would allow us to test our bot automatically against someone playing the “perfect” moves every time. Of course, since Pentago is a first-player win game, it would make no sense to have our bot go second, as the perfect Pentago would win every time. So we set out to build a system in which our bot would make a move first, get an optimal move back from the perfect Pentago backend, and proceed accordingly. This proved to be much more difficult than originally expected, but we ended up achieving our goal. The system is set up as follows: our bot makes a move, thus generating a new board. The new board is converted into the proper format required by the API. This format, according to Dr. Irving himself, was “a mixed radix number with radix 2^16 for quadrants and 3 within each quadrant.” While this took a while to figure out, we eventually were able to properly format our board and get a perfect move back. In order to ensure unique and meaningful results, we randomly selected the “perfect” player’s move from all equally valuable moves. Since the perfect Pentago backend only evaluates “win,” “lose,” or “tie,” we made no judgements on the quality of a move beyond that. The way the system is set up, and the way the “perfect” API works, our bot would be guaranteed to lose if it made even one slip-up, since every move made by the “perfect” player would lead to a win for it. We were hopeful that, depending on the depth, we could avoid mistakes at least some of the time.

**Section 3: Evaluating Our Success**

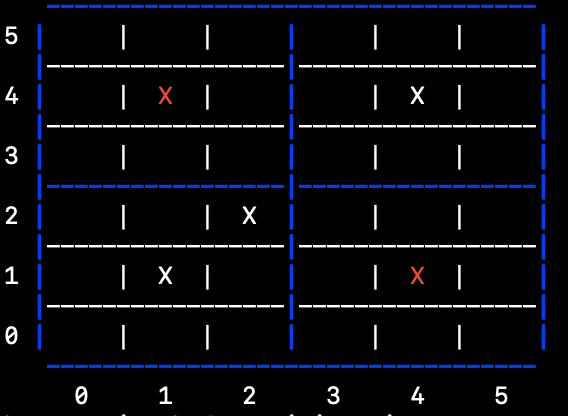
**Performance Analysis at each Depth**

Depth 1: Our bot at depth 1 plays a reasonably strong Pentago game that is a great match for beginning players. It moves myopically in a way that will increase its own score by expanding upon its sequences but pays no attention to blocking sequences of its opponent. Since it checks every possible move (that is not pruned), it will always play a winning move if one is available. From watching beginning players play against the bot, it seems like there are two types of strategies that most beginners chose.

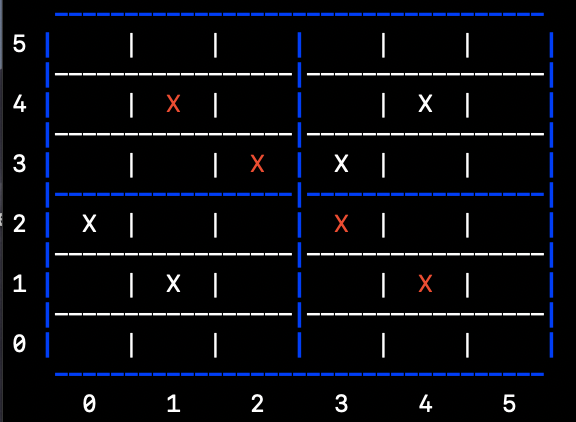
* The first strategy is to try to control two center spots from the start and then try to connect those two center spots in a sequence of 5. If the player goes first (which they usually chose to do), they will win in 5 moves because the bot will do the same thing and lose the race. Since depth 1 only looks one move into the future, the bot will not realize that the player will succeed before it does and it will continue trying to build its own sequence until the player wins.
* The second strategy is for the player to try to scatter their marbles across the board to gain board coverage. While this may seem reasonable at first, it unfortunately is a very poor strategy in Pentago. Players who tried this strategy against the bot at depth 1 usually lost fairly quickly.

Depth 2: Our bot at depth 2 is very strong, and it is a significant improvement in play from depth 1. Since it checks every possible board two moves into the future (that is not pruned), it will always stop the player from winning if possible. Thus the only way to beat our bot at depth 2 is to put it in a situation where the player will win regardless of the bot’s next move. This is very difficult for beginning players to do, although they occasionally stumble upon such a position.

Depth 3: Our bot at depth 3 plays the strongest Pentago that we were able to achieve, as shown by its win rate against human competitors as well as against the Perfect Pentago bot. This makes sense, as its analysis extends further into the future and it is thus able to make the most informed move decisions. However, the extra depth level can actually make the bot play worse in some cases. This is because having the bottom level of the search tree be on the bot’s move (a maximizing level) can lead to decisions that are woefully optimistic. An example is shown in the position below.



Our bot is red, and it is our bot’s move. The most important thing to consider in this position is stopping white from placing a marble in the (3,3) spot. This would create a run of 4 with open spots on either side[[3]](#footnote-2), guaranteeing a loss for our bot unless it can win on the next move (which it clearly cannot in this case). The bot at depth 2 responds properly by placing a marble in spot (3,3) and rotating the bottom left tile. However, the bot at depth 3 does not see the opponent’s run of 4 as being so bad, because the depth 3 analysis extends one level further where the bot would be able to place a marble on spot (3,2) and rotate the bottom left tile, creating its own run of 4 and breaking up the opponent’s run of 4. This position would be as follows.

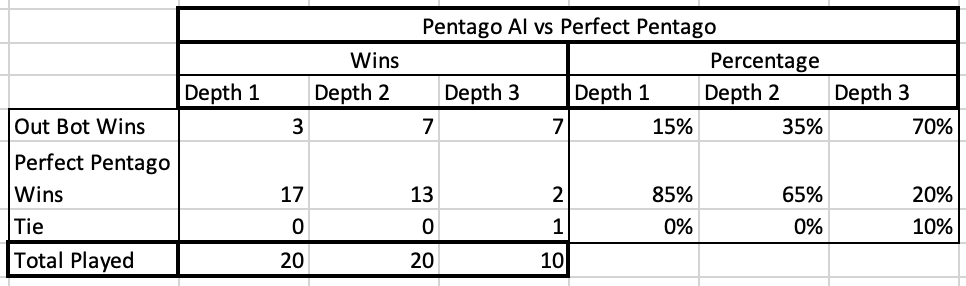


Since the analysis ends here, our bot (red) believes that it has put itself in a good position. It has a run of 4 compared to two runs of 2 for the player. However, if the analysis had extended one level further, the bot would have realized that the player will win on the next move. One of the several winning moves for white from this position is to place a marble in spot (2,0) and rotate the bottom left tile to the right.

It is also worth noting that at times, our bot at depth 3 will pass up a winning move if it knows that it will have another winning move on its next turn no matter how the opponent responds. This occurs if the guaranteed win in 3 turns is found by the minimax algorithm before the win in one turn. Since the bot sees both outcomes as equal, it does not overwrite the first winning combination.

**Playing Against “Perfect” Pentago**

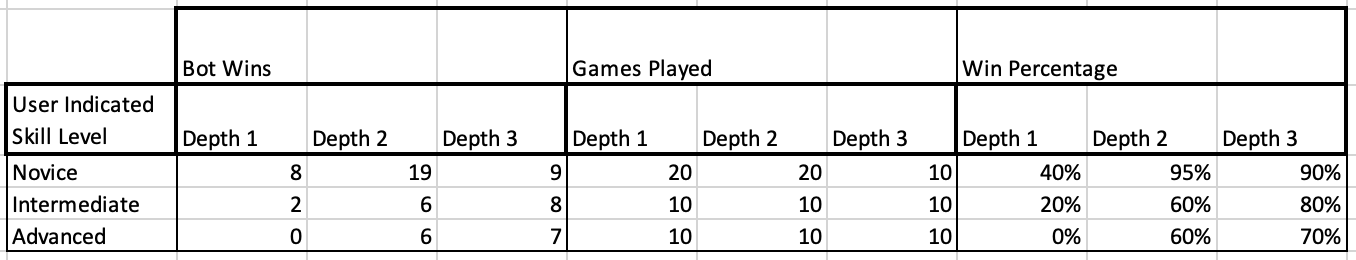
With the knowledge that the player going first always has a winning strategy available to them, we were hopeful that our bot would, at least some of the time[[4]](#footnote-3), be able to beat the “solved” version of Pentago when our bot played first. And, as seen in the table below, we found that we were able to beat the Perfect Pentago bot, especially when we ran our bot at minimax of higher depths. At depth 1, predictably, our bot played pretty poorly against the Perfect Bot. However, we were still able to beat the Perfect Pentago bot 15% of the time. At depth 2, our bot did significantly better, with closer to a 50% win average, but still lost to the Perfect Pentago bot a majority of the time. At depth 3, however, we were able to beat the Perfect bot 70% of the time, and notably, this was the only time in all of our testing that we saw a tie between players, because the board filled up. We only did 10 runs at depth 3 though, due to the the length of time it takes to run our bot at depth 3.



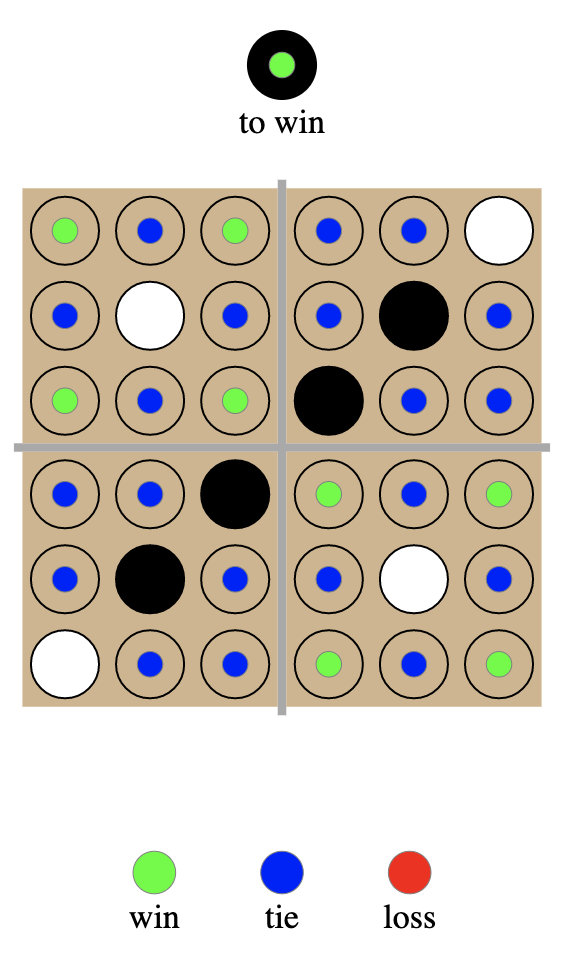
**Did we meet our goals?**

Goal 1: We were incredibly successful in our goal of creating a playable version of Pentago that was “fun” for users to play against. We created a terminal based application that allows users to play against bots of varying difficulties (search depths). Because of the fact that our bot was able to beat opponents regularly, we see that it is a worthy adversary that people can enjoy playing a game against.

Goal 2: It is also clear that we were successful in our secondary goal of creating a skilled bot that was very difficult to beat. When searching to depth 2, we saw that our bot beat opponents of all skill levels at least half of the time and defeated almost every “beginner” opponent. We also found on depth 2 that our bot was never bested in less than 7 moves per side (14 total), which was an additional part of our goal. See table below for quantitative metrics on performance statistics. Note that all novice players were bumped up to intermediate after their fourth game and bumped up to advanced upon their request (though these classifications were only for our data recording purposes).



Goal 3: Unfortunately we were not able to meet our stretch goal of playing perfect Pentago. When we played games against our bot (meaning that we were making stronger adversarial moves than random selection) and loaded each position reached into the Pentago interactive explorer, we always eventually reached a position where our bot chose a move that diverged from the perfect play sequence that would guarantee a win. The first few times this happened, we were able to identify a small bit of logic that could be added to our evaluation function to make our bot smarter and correct the mistake. For example, this is what led us to add a small bonus for control of the center and to ignore sequences that could not lead to a win. While these improvements did make our bot stronger and allow it to dive further into a perfect game before diverging, we eventually reached one particular position that led us to believe that our project could not achieve perfect Pentago play without significant changes to our system. The position is shown below.



Our bot is black, and it is black’s turn. It needs to choose one of the green spots to continue its winning sequence. However, none of the green spots would ever be chosen by our bot, as its evaluation function is based on creating sequences of highest value. Thus it would always choose moves such as (2,1) or (1,2) which create two strong sequences of 2 before selecting a spot which creates no runs for itself and does not block an immediate large run from the opponent. Selecting a green spot would require an understanding of the game that is deeper than our bot could easily be provided within the context of the system we created.

1. <https://www.cs.cornell.edu/courses/cs4700/2019fa/Textbook/> [↑](#footnote-ref-0)
2. <https://en.wikipedia.org/wiki/Alpha%E2%80%93beta_pruning> [↑](#footnote-ref-1)
3. Any such open ended run of 4 is a guaranteed win in the following move. There is nothing the opponent can do to stop it, as they cannot block the sequence of 5 in both adjacent open spots. [↑](#footnote-ref-2)
4. We say sometimes and not all the time because there is a degree of randomness built in to which “perfect” move is chosen [↑](#footnote-ref-3)