1. Authors: Liu, Grace; Vitzthum, Nathan
2. Agent name: Gerald Flortinstein. Our agent is an old man who has a provocative persona. He is very confident in his skills and always believes that he will win. His confidence can be daunting to the other player and which also factors into his high success rate (making the opponent less confident in themselves).
3. Our agent's "twin" differs as it has a different persona. Rather than a provocative persona, our twin has a nice and sweet persona.
4. If we are pruning (a parameter passed into the minimax function) and it is the maximizing players turn, for every state, the search first finds the max value between the current alpha value of the node and the best score seen thus far, and sets this to be equal to the new alpha value. If the beta is less than or equal to the alpha value, we break out of the for loop, effectively pruning the rest of the children. If it is the minimizing player's turn, the beta value will be updated based on the maximum between the current beta value and the best score seen, and if beta is less than or equal to the alpha value, the search is pruned. The cut offs are detected in the minimax function, and pruning logic occurs when the pruning flag is set to "True". Our agent does use child ordering by static evaluation if "order_moves" is set to "True", and orders the moves based on the static evaluation of the resulting states, which makes the search more guided and informed, leading to more effective pruning, and increasing likelihood of alpha-beta cutoffs. Our agent does measure the number of static evaluations performed with and without alpha beta pruning in our test_move_ordering function which is discussed in the extra credit portion of the report.
5. Our agent is called Gerald Flortinstein and has a persona of provocative. It is not modeled after a real person, we just wanted our agent to have a very egotistical persona. Even when he is losing, or if the game is neutral, his remarks always are attacking the opponent. This is a part of his tactic as he hopes these remarks will scare his opponent, possibly causing them to play a wrong move due to fear.
6. Our dialog uses if then statements based on the static evaluation of the game to return responses that correlate to the current state. For instance, if the current state's evaluation is heavily in our agent's favor, our agent will return a random utterance from a bank of options that are boastful/arrogant and demeaning to the opponent. However, if our agent is losing the game, the responses come from a bank of options that are dismissive of the opponent's skill or show disbelief in what is happening. We also have a bank of responses that are for neutral game states where our agent says something provocative to the opponent. Similarly, we have banks of responses for our twin that are much kinder and more encouraging to the opposing agent. Even when losing, the twin is complimenting the other agent's ability much unlike our main agent.
7. Our agent has two features in which it is able to keep our utterances responsive to the opponent's remarks. If our opponent asks "What's your take on the game so far?" Our agent will respond with a game summary, and will give a summary of what moves has been played and who played the moves. Our agent will also predict who will win the game based on the static evaluation of the current game state (compares the static evaluation value to k*20). On the other hand, if our opponent asks "Tell me how you did that," our agent will respond by giving a detailed description of the agent's thought

process behind making that move. This includes the time our agent spent thinking about what move to make, how many static evaluations and how many alpha-beta cutoffs he made. If the opponent says neither of these remarks, our agent will respond based on the static evaluation function of the current state of the game.

8. To develop our dialog capabilities, we first decided on what process we wanted to implement for choosing utterances, which we decided on using if-then statements based on game conditions and the current utterance of the opposing player. To decide on how to judge the current state of the game, we tried a few options and checked the results. We wanted to have a set of responses for a clear winning position and a clear losing position for our agent. This led us to using a static evaluation value of +/- 20k, where k is the number of X's or O's in a row needed to win. With our evaluation function this seemed to judge who was winning the game very well for our utterance implementation. Also for our extra credit, we added responses for when the opposing player asked "Tell me how you did that" and "What's your take on the game so far". A further explanation of this implementation can be found in the next question.
9. Extra Credit:

Part 1A: Implementing reordering of search using static evaluation

Testing state:

```
+--------------------+
| -  .  .  .  .  .  -|
| .  O  .  .  .  X  .|
| .  .  .  .  .  .  .|
| .  .  .  X  .  .  .|
| .  .  .  .  .  .  .|
| .  X  .  .  .  O  .|
| -  .  .  .  .  .  -|
+--------------------+
```

It is O's turn to move.

Results without move ordering:
Time: 1.0001 seconds
Static Evaluations: 33744
Alpha-Beta Cutoffs: 1097

Results with move ordering:
Time: 1.0001 seconds
Static Evaluations: 33924
Alpha-Beta Cutoffs: 641

Testing state:

```
+--------------------+
| -  O  X  .  X  .  -|
| .  O  O  X  O  X  .|
```

```
| X  X  O  O  X  .  . |
| .  O  X  X  O  O  . |
| .  X  O  O  X  .  . |
| .  O  X  .  O  X  . |
| -  .  .  .  .  .  - |
+---------------------+
```
It is X's turn to move.

Results without move ordering:
Time: 0.5139 seconds
Static Evaluations: 18278
Alpha-Beta Cutoffs: 1304

Results with move ordering:
Time: 0.1556 seconds
Static Evaluations: 4527
Alpha-Beta Cutoffs: 599

The results above are from our test_move_ordering method that we used to see the benefits of ordering the possible moves at each state we arrive at in minimax. As you can see, the benefits were far greater in the later stages of the game when the potential scores for the possible moves had a greater range of possibilities compared to the early game state. Early in the game almost all possible moves have such a low score that ordering the moves has a negligible effect on the efficiency of the code. However, later in the game as shown above, ordering the moves is far faster, taking less than a third of the time to get through the max_ply we set for the test compared to the non-ordering test and it had to evaluate far fewer states because it more quickly found the winning moves.

Part 2B: Responding to opponent's remark of "Tell me how you did that"
If our opponent asks "Tell me how you did that," our agent will respond by giving a detailed description of the agent's thought process behind making that move. This works by calling our function called "explain_last_move." If there was no previous move, our agent will respond by saying we do not have information about the last move. Otherwise, the agent will report the time our agent spent thinking about what move to make, how many static evaluations and how many alpha-beta cutoffs he made. These values are stored during the search, and incremented when it occurs. For example, if an alpha-beta cut off occurs, the alpha_beta_cutoffs variable will increment, and our function will use this variable when reporting how we decided our previous move.

Part 2C: Responding to opponent's remark of "What's your take on the game so far?"
If our opponent asks "What's your take on the game so far?" our agent will respond with a game summary and win prediction. This is done so by calling our "generate_game_summery" function. This function will loop through every state in the game_history, and for each state, mention which player made a move, and what the move was. The function will also give a prediction of

which player will win given the current state of the game. Our agent does this by using the static evaluation of the current game state. If the static evaluation of the current board state is greater than k*20 (which is what we set the heuristic threshold to be in our program), it will predict X to win. If the static evaluation of the current board state is less than k*-20, it will predict the O player to win. If neither conditions match, the agent will report back that it thinks the game will end in a draw.