# CS 5804 Mini-Project
# Group 12: AI Checkers

Alina Bhatti, Liam Davis-Wallace, Alex Marrero, Sarah Ramboyong & John Swecker

# Problem Statement & Analysis

- Checkers is a well known and common 2-player board game that is played around the world
- It has been relatively overlooked in the AI game world

**We will create an interactive checkers AI that is competitive with strong human players, equipped with it's own strategies, to complete a challenging and thoughtful game of checkers.**

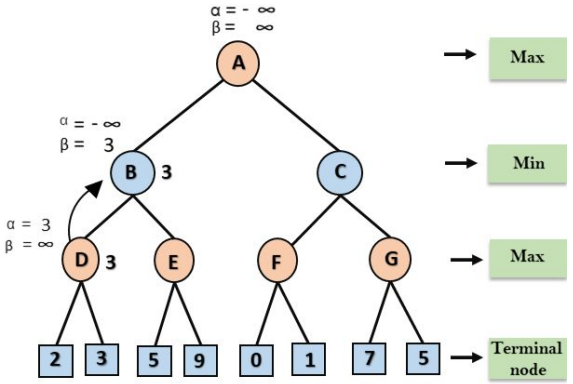
- Target audience: Checker players & board game enthusiasts

# Use-Case Scenarios

- Primary actor: Checkers player
- Preconditions
  - User is able to interact with a graphical user interface to input moves and view opponent moves
- Main flow:
  - Human player launches the AI-integrated checkers game
  - Human player selects the desired difficulty level of the AI opponent
  - Program populates the board with tiles and game pieces
  - Human player makes a move*
  - AI opponent makes a move*
  - Repeat * process until one player wins the game
  - Game displays who won and offers an option to play again
- Post conditions:
  - Human player experiences a challenging and engaging gameplay experience
  - AI player performs according to its skill level

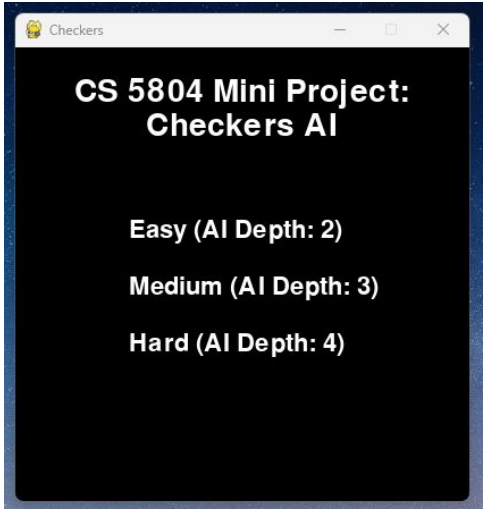
# AI Algorithm & Models

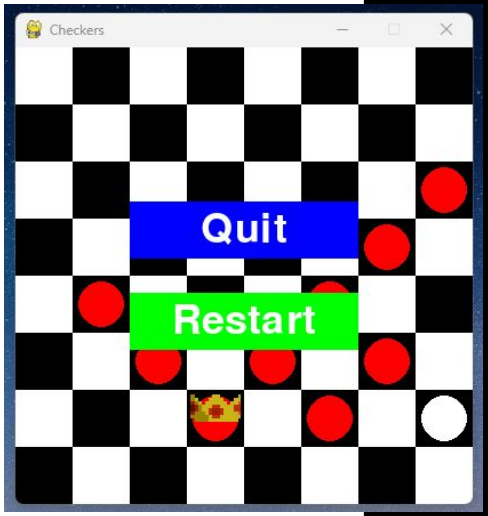
- Min-Maxing with Alpha Beta Pruning
  - Used Alpha Beta pruning to prune poor moves which will save computation time
  - Used heuristic evaluation function with more weights on more prominent features (ex. red/black kings has more weights compared to the regular red/black pieces)
  - More specifically, recursive functionality decrementing depth each call, where user chooses the depth of the tree search based on difficulty
  - As the depth increases in the tree, it means that there are more possibilities/boards of the next piece's move to consider which means it would be a higher time complexity and it would take longer to generate moves from the AI side.



# Results

- Functional checkers adversary
  - Three difficulty levels
- Agent decision making based on utility of each move from minimax search
  - Algorithm cost minimized by alpha-beta pruning
- Game ends when one player has no legal moves left
- User can restart a new game or quit and close the GUI




Demonstration

# Lessons Learned

- Lessons:
  - Heuristic Design Approach
  - Accuracy vs time tradeoffs
  - User interface integration
  - Team collaboration
- Future Work and Improvements/Implementations:
  - Implement multiple jumps at once
  - Potentially include a hints or move recommendation system for the user