Algorithm

Interface (GUI)

- Using java swing components and imported pictures/videos from the internet, assemble a title page that contains all the necessary information
    - Title
    - Instructions page (How to play)
    - Singleplayer
    - Two player
    - Load a game
    - Settings (Adjust frame size, graphics, etc.)
    - Set difficulty of AI
- By clicking on a JButton, the current panel is cleared and a new panel is added to the frame
- Panels are stored in a JPanel [] array, each button will reference a different panel in the array

List of Classes

- Point class: Point
    - Represents an piece on the gameboard
    - Contains variables like position
- Parent class: gameBoard
    - This class is the parent class that contains methods that are re-used in each gamemode
    - *Variables*
        - 2D array of **point** objects representing the game board
        - More variables to be added during execution phase
    - *Methods*
        - Contructing the game board
            - Two constructor methods
            - One constructor makes a blank grid (for new game)
            - One constructor loads a game state from a file (loading a saved game)
        - Clearning the game board
        - Dropping a piece (when button is clicked) and determining the position of that piece
        - Checkwin algorithm (used by both PvE and PvP)
        - More methods to be added during execution phase
- Subclasses: singlePlayer, multiPlayer

        - Both classes extend gameBoard
        - In singlePlayer, the minimax algorithm is added after a player move
            - Methods in the parent class are still used
        - In multiPlayer, drop piece and checkwin methods are called after each move
- GUI class: MainMenu
    - Class for the GUI
    - Contructs the window and all the components for the title frame
- Settings class
    - Class for the settings pane
- Other classes: Other classes may be added if more frames are decided to be implemented

Game Mechanics

- Each piece is represented by an object - Point
    - Every "point" object has attributes, such as colour and position
- Represent the board using a 2D 6 x 7 array of Points
- On the GUI, each position on the board is organized with a grad layout (6 x 7)
- Use a Board object to represent the 6x7 grid of pieces
    - Will contain methods that can clear, access and modify the point object at a specific location

- *Placing a Piece*
    - The grid layout will consist of 6 buttons that the user can click (one for each column)
    - Once the button is clicked, depending on the user's colour, a component that is in the shape of a circle with the user's colour will cover the JButton on the gridlayout
        - A for loop iterates through the positions of the column, the place is put in the position where the first empty space is found
    - The point 2D array will also be updated to match the GUI board's state, as well as the gamestate object

- *Timer*
    - Using java swing's timer object, create a timer that initializes when the user presses play
    - Users will be able to set a time limit
    - If the time limit is surpassed and no players have won, a draw is reached

- *Max Number of Moves*
    - Each time a player makes a move, a counter is increased

    - Users will be able to set a move limit
    - If the counter reaches the move limit and no players have won, the game is a draw

- *Save and load a game*
    - If the user presses the save button, the state of the board will be written to a text file
    - 1 means red piece, 0 means no piece, -1 means yellow piece
    - When user presses load game, the secondary constructor in class *gameBoard* is called and the game begins with the loaded state

Singleplayer (AI)

- *Minimax Algorithm*
- https://www.youtube.com/watch?v=y7AKtWGOPAE
- After the user places a piece, every possible config (each column) where the piece can be put is trialed
    - *Scoring Mechanism (Weights will have to be adjusted during monitoring and controlling phase)*
        - If the piece can be put in the center column, + 4
        - If the piece can be put so that it forms a line with the other and only piece (line of 2), + 2
        - If the piece can be put so that it forms a line with two other pieces, + 5
        - If the piece can be put so that is forms a line of four, + 1000000
        - If the piece can be put so that the opponent can form a line of two, - 2
        - If the piece can be put so that the opponent can form a **winnable** line of three, -100
- Using the scoring mechanism and a specified depth level, the minimax algorithm will trail every single possibility for the user and the AI (branch out like a tree diagram)
- At the final depth level, it will select the move the AI makes that has the maximum point value
- It will then move up a depth level and choose the move the user makes that has the least point value (accounting for the best move the user can make)
- It will keep doing selecting max and min paths until it reaches its current state (current move)
- Based on which paths the algorithm selected, the AI will follow those paths
- Eventually, the AI will always make the best move AND account for all the possible future BEST moves the user can make
- Depending on the difficulty, the minimax algorithm's depth level might be reduced OR a chance element is added (Ex. 5% of making a bad move instead of making the best move)

Multiplayer

- The place piece method is called each time until two times the move limit is reached
- The check win method is called after each move
- *After Each Move*
    - After a piece is placed, a **checking** algorithm checks all the possible 4-chain configurations that contain the position of that piece
    - If a 4-piece chain is found, the game ends and the winner is announced (depending on who placed the last piece)
    - *Check Win Algorithm (Using recursion)*
        - After a piece is placed, the piece's position is recorded as variables
        - For loop to iterate through all 8 pieces adjacent to the placed piece
        - If a piece immediately adjacent to the placed piece is the same colour (2 in a row), the method will return 1 + recursive call (next point), else return 0 + recursive call (next point)
        - Call the method again, but pass the adjacent point through the method
            - Counter will only increase if the adjacent piece is consistently in the same condition
        - *Exit condition*
            - After the recursive method is called, it should return an integer
            - If integer > 4, the user wins
            - Else the game resumes and the next player plays a piece
        - *Edge Condition*
            - Use try catch validation inside the for loop to break when a point that is index out bounds is referenced