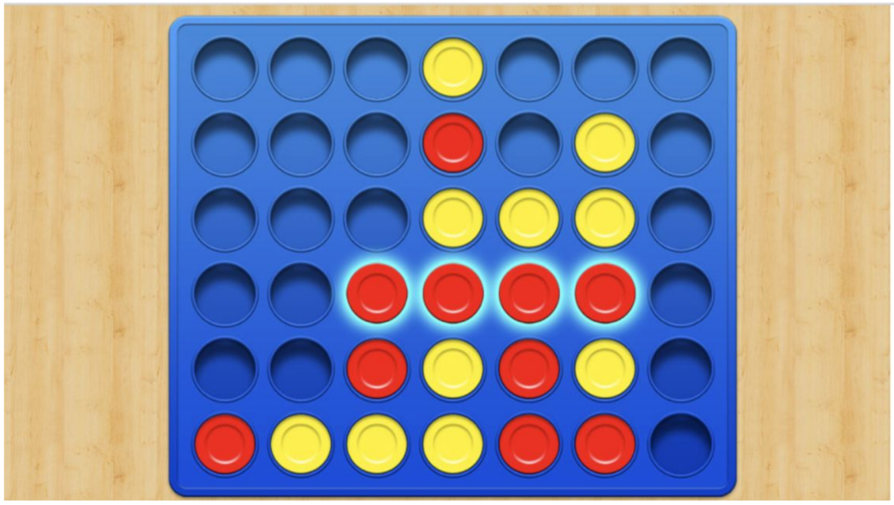
<u>Unit 4 Functional Specifications - (Anika, Arnav and Albert)</u>

Advanced Connect 4 is a program that is based off of the original game connect 4. However, this game is more advanced and built with a computer AI that can play against the player. The base game of connect 4 is played on a board with 7 columns and 6 rows for a total of 42 squares. There are two players, who use tokens of blue and red. The players alternate placing tokens into columns. The game implements gravity such that the tokens fall to the bottom of each column if there are no tokens already in said column, if there are, the token will fall on top of the tallest tokens.

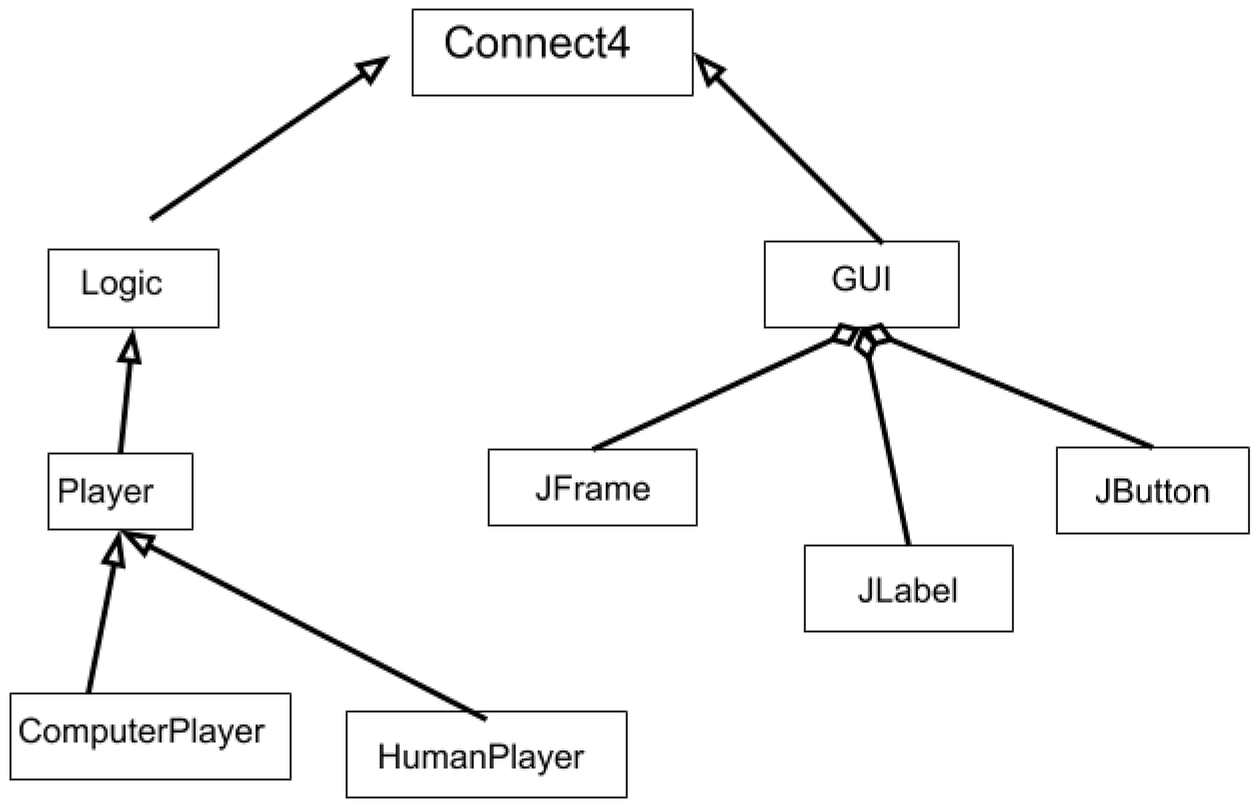

Winning Position for Red, they have 4 in a row

However, this game is more complex than the original connect 4 as players start out with 100 tokens, and each player needs to bid on the start of their turn. The player that bids more gets the privilege of going first and the tokens that both players bid are burned. If the game board is filled up or both players have 0 tokens, the game ends in a draw. Whoever gets 4 tokens horizontally, vertically or diagonally first wins.

This game may sound simple, however, the connect 4 portion of the board has **4,531,985,219,092** possible permutations and the bidding portion will expand this to over **1.7*10^170** possible paths the game can be taken. Thus, the AI could not possibly plan out every path the game could take, instead, it uses set rules to play the game.

This program will have two game modes, Player v. Player and Player vs. Computer AI.

Object Oriented Design



This picture shows the class diagram for Advanced Connect 4. This program involves 6 classes.

Connect4 Class
The connect 4 class will be the base class for the entire project. The purpose of this class will be to hold most of the methods for actually running the game and it will be the class in which we test the entire program.

GUI Class
The GUI is a java application that gives the player who is playing against the computer or the other player something to visually look at. The program can also be run without the GUI and it will be played in the console. The GUI is the class that checks where the player clicks in order to make the move. In addition, it will also display how many coins each player has and the current state of the board. We decided to implement the JFrame, JLabel, and JButton classes and these classes have the "has-a" association relationship, also known as aggregation, with the GUI class. These three classes will represent the frame, labels, and buttons for the Connect4 game.

Player Class
The player class will have the basic functionality of a player in Connect4. With this class, we will have basic methods such as move() and insert() to implement the moves of the user and Computer AI. The ComputerPlayer class and HumanPlayer class will then extend from this class which will help make the code more efficient as we will not have to rewrite the existing code.

ComputerPlayer Class
The computerPlayer class is the AI that plays against the player. There will be methods in the class that check the status of the board. Using hard rules that we program directly into the ComputerPlayer, we will choose the best move to make considering the state of the board and the amount of coins each player has.

HumanPlayer class
The HumanPlayer class will extend the base Player class for the game. This class will have the user control the moves for the game. Whether it be bidding a specific amount of money or inserting a coin into the Connect4 grid, the class represents the user's moves in the game.

Logic class
The logic class holds the rules and the comparator for bidding. Whoever, bids the highest wins the privilege of making the move during that turn and the comparator will hold that information. In addition, this class makes sure that all moves are valid(ie one player is not trying to force additional tokens down a filled column. This will also make sure that the game ends and that the correct person is rewarded with a win when there is a four in a row(horizontal, vertical or diagonal).

<u>Structural Design</u>

Data	Interface => class
Connect4 coins columns	Stack => Stack< Int, Int>
Track score for each player	*Map* => HashMap<Int, Array>
Maximum number of coins	final int MAX_NUM_COINS
Connect4 grid	*Array* => Array<Int, Stacks>
Tokens(Played on the board)	int tokens
Maximum number of tokens on board	Final int MAX_NUM_TOKENS

Coins(For bidding on a move)	int coins

The Connect4 grid will be implemented as a 2-d array in which we will have a constant variable that represents the maximum number of tokens allowed in the grid. We have chosen to hold the current state of the board in an array of stacks, where each stack corresponds to one column(7 in total). The coin objects in each column will be represented as a stack since each coin will be pushed onto the stack when the player chooses to place a coin in a column that already has coin objects in it. This is the easiest way to store the data, unless we use a 2-d array, which does not put the coin on the next available space.

Each player in the game will start off with a set amount of money that they can bid to play a turn. Not only will we need a constant variable for the maximum number of Connect4 tokens, but the game will also require us to have a constant that represents the maximum number of coins.

The Advanced Connect 4 project also needs to track the score for the player that is playing. The game will not only save the winner so you know how many games you have won, but it will also include the saved state of the board for viewing. To implement this, we decided that a hashmap will be the simplest data structure to use. The hashmap will store the round number and the current state of the board.