Comprehensive Exercise - Connect Game

CSC-116 (601)

Christine Weld
clweld@ncsu.edu

Table of Contents

Table of Contents 1
Introduction 2
Software Requirements 3
User Interface 3
Error Handling 4
Software Design 5
UML Diagram 5
Implementation 6
ConnectGame.java Implementation 6
Main Method 6
Helper Methods 6
Class Constants 6
Methods 6
Board.java Implementation 8
Class Constants 8
Instance Fields 8
Constructor 8
Methods 9
Player.java Implementation 10
Class Constants 10
Instance Fields 10
Constructor 10
Methods 10
Testing 11
System Testing 11
Team Reflection 12

Introduction

The ConnectGame project will design and develop an application for playing Connect Four (or Five or Six), a game that has been popular for generations. The game consists of a grid game board on which players take turns placing their pieces. When the player drops their piece in a column, the piece will drop to the bottom-most row that is not occupied by another game piece. A player wins the game by arranging four game pieces such that they are connected across a row, down a column, or diagonally. This version of the game can be expanded to require four, five, or six tiles to be connected with the corresponding grid sizes 8x8, 10x10, or 12x12.

Compile Source Code: You will use the following commands to compile the `ConnectGame` program from the `ConnectGame` directory:

```
$ mkdir bin
$ javac -d bin -cp bin src/*
```

Software Requirements

In this version of the game, the users will choose if the game should be played as Connect 4, Connect 5, or Connect 6. The corresponding board size width and length will be double the winning number of connected pieces (ie. if Connect 4 pieces to win, then the game board is 8 x 8, if Connect 5, then board is 10 x 10, if Connect 6, then board is 12 x 12).

How to win:

- The winner will be declared when the winning number of their own pieces are connected vertically, horizontally, or diagonally.
- If every space on the board is filled without a winner, then the game is a draw.

After each game:

- When a game is complete, the players are asked if they would like to play again.
- If there is a new game, the player that started first initially will then start second, and vice versa.
- The program will keep track of and display the current wins, losses, and ties for each player

User Interface

To start the game, the user must enter three valid command line arguments

1. Winning number of connected pieces (4, 5, or 6)
2. First player's name
3. Second player's name

Program Execution: You will use the following command to run the `ConnectGame` program from the `ConnectGame` directory:

```
$ java -cp bin ConnectGame 4 Player1 Player2
```

Gameplay:

- Each player will be assigned their own pieces (Player1 is X, Player2 is O) that will be displayed on the board as they take turns choosing where to place each piece.
- The player will select a column to place their piece, and the piece will "drop" to the first available spot at the bottom of the column.
- If there are already one or more pieces in a column, and another piece is added, then the new piece will be on top of the previous piece.
- After a player chooses a column to place their piece, the game board is displayed again with the current state of pieces in play and a current count of maximum connected pieces for each player.
- Players will take alternate turns until either one wins or the game ends in a draw.

After each game:

- The winner is declared and a current count of wins, losses, and draws for each player is displayed.
- The player will be prompted to enter 'y' or 'yes' to start a new game or enter 'n' or 'no' to stop playing.
- Either player can end the game early by entering 'q' or 'quit' at their turn.

Error Handling

Command line arguments:
If no first, second, and third command line arguments are entered, or invalid first argument (not an integer 4, 5, or 6), then there should be a usage message and the program should immediately exit.

```
$ java -cp bin ConnectGame
Usage: java -cp bin ConnectGame {4 - 6} player1 player2
```

```
$ java -cp bin ConnectGame X Player1 Player2
Usage: java -cp bin ConnectGame {4 - 6} player1 player2
```

User input:

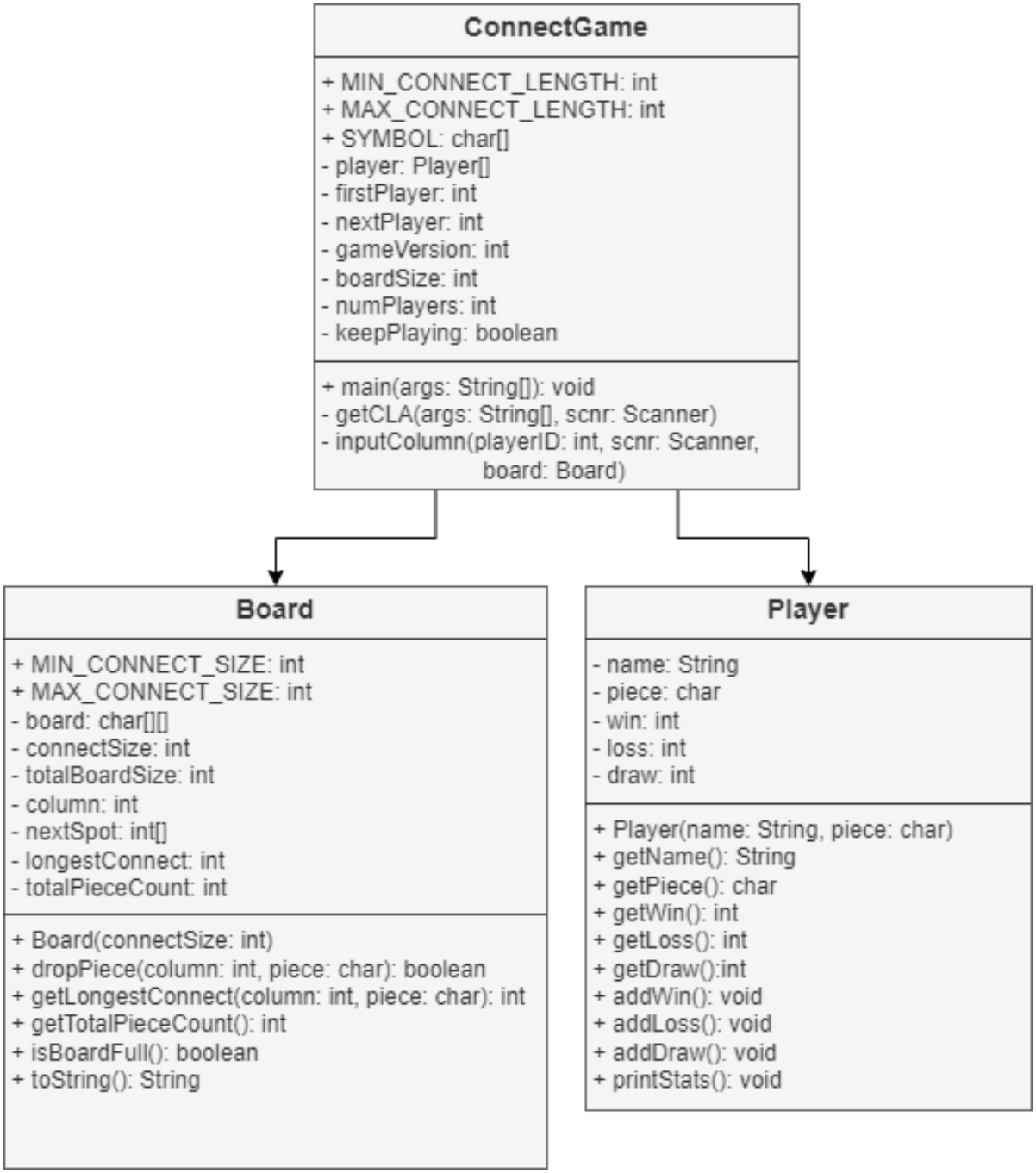
- If a player enters a column that is already full, they are prompted that the column cannot fit more pieces and they should select another column.
- If the player enters a non-existent column number (less than 1, or greater than game board size) or invalid input (string, float, etc), they are prompted that it is not a valid column and they should select another column.
- If the player enters an invalid response to the play again prompt after each game, they are prompted again for a valid response.

Software Design

This program is built on three classes:

1. The `ConnectGame` class is the control program of the game
2. The `Board` class represents the game board
3. The `Player` class represents each player

UML Diagram



Implementation

ConnectGame.java Implementation

Main Method

The ConnectGame program plays Connect Game, which has three versions for how many connected pieces a player needs to win: Connect Four, Connect Five, or Connect Six. The two players take turns entering an integer into the console for the board column in which to place their piece in the game board.

After each turn, the program checks if the piece placement meets a win or draw condition. Then the game board is updated with the new piece and the board is displayed in the console. When the game ends in a win or draw, the console displays a message declaring the winner, and displays each player's current win, loss, and draw stats.

After the game ends, the program prompts the player if they want to play again. If so, the game board is created anew, but with the same player definitions (name, piece, stats). The first game is started with the first player, the next game started by the second player, and vice versa while consecutive games are played. The players can quit the program by entering 'q' at the column input prompt, or by entering 'n' after the game ends.

Helper Methods

The game version (4, 5 , or 6), the name of the first player, and the name of the second player are defined in the command line and handled by the `getCLA` method. If invalid or too few command line arguments are entered, a usage message is printed and the program exits.

The validation for the player's column input is handled by the `inputColumn` method. The method checks if the input is an integer, if it's in the valid range of the board size, and if the column has space for another piece.

The board is displayed by method `displayBoard` to avoid duplicating the printout code.

Class Constants

Declare and initialize the following public static class constants:

- `MIN_CONNECT_LENGTH = 4` : an integer of minimum allowable number of connected pieces to win.
- `MAX_CONNECT_LENGTH = 6` : an integer of maximum allowable number of connected pieces to win.
- `SYMBOL[] = { 'X', 'O' }` : a character array of token symbols for the players.

Methods

Complete the following methods:

- `public static void main ( String[] args)` : The main method prompts players with a welcome message and assigns symbol characters to each player. While `keepPlaying`, it constructs a new `Board` with `gameVersion`. During gameplay, it calls `inputColumn` for player input, `board.dropPiece` to place pieces on board, and `board.toString()` to display updated game board. After each player's turn, it checks for a win condition (one player has the desired number of

connected pieces) or a draw condition (the entire board is filled with pieces without either player connecting the desired number). After each game is over, it displays player stats with `player.printStats()`, and prompts if players would like to play again ('y' or 'n'). It reassigns `firstPlayer` to alternate the starting player for the next game.

- `public static void getCLA( String[] args, Scanner scnr )` : This method reads in command line arguments and assigns them to instance fields (`args[0]` for `gameVersion`, `args[1]` and `args[2]` for `player`), calculates and assigns `boardSize`, or exits program if arguments are invalid (less than 3 arguments, first argument is not an integer, or an invalid integer for valid game version 4, 5, ot 6).
- `public static int inputColumn( int playerID , Scanner scnr, Board board )` : This method prompts the player to enter a column number to place their piece and returns the index of the selected column. It checks for invalid input (not an integer from `1` to `boardSize)`and reprompts until the input is valid. It allows the player to enter 'q' to quit the program. It throws IllegalArgumentException if playerID is invalid (not from `0` to `numPlayers`). After a valid piece is placed, the board will print to the command line with the player's stats (name piece, and longest connect)
- `public static void displayBoard( Board board, Scanner scnr )` : This method formats and prints out the board. This is called at the beginning of the game and again after each play.

Board.java Implementation

The Board class represents the game board and placement of each player's pieces. The board length and width dimensions will be set as twice the connect win condition. The board will need to track the longest connection of each player's pieces by searching the rows, columns, and diagonals. If a selected column is full, the player will need to select a new column. If the game continues until the board is full, then the game will end with a draw.

Class Constants

Declare and initialize the following public static class constants:

- `MIN_CONNECT_SIZE = 4` : an integer for minimum allowable connection size
- `MAX_CONNECT_SIZE = 6` : an integer for maximum allowable connection size

Instance Fields

Declare the following instance fields:

- `board` : a two dimensional character array to represent the board grid. The row and column dimensions are defined by `connectSize * 2`. `connectSize` is validated by checking against `MIN_CONNECT_SIZE` and `MAX_CONNECT_SIZE`. Each element is initialized with the '|' or '_' character for the following appearance:

```
|_|_|_|_|_|_|_|_|
|_|_|_|_|_|_|_|_|
|_|_|_|_|_|_|_|_|
|_|_|_|_|_|_|_|_|
|_|_|_|_|_|_|_|_|
|_|_|_|_|_|_|_|_|
|_|_|_|_|_|_|_|_|
|_|_|_|_|_|_|_|_|
```

- `connectSize` : an integer for the connect win condition, determines board size
- `totalBoardSize` : an integer for the total possible number of pieces that can be played, is equal to board size squared
- `column` : an integer of player input for the column to drop the player's piece
- `nextSpot` : an integer array for where the next piece in a column should be placed
- `longestConnect` : an integer for the longest connection of a player's pieces
- `totalPieceCount` : an integer for how many total pieces have been played

Constructor

- `public Board(int connectSize)` : This is the constructor of the class. It accepts `connectSize` as a parameter to create the assignments for the instance fields. This should throw an `IllegalArgumentException` with the message "Invalid connect size" if outside the `MIN_CONNECT_SIZE` and `MAX_CONNECT_SIZE` constants. The `nextSpot` array should be initialized and have all elements set to the last row in the `board` array. The `totalPieceCount` should be set to `0` and `totalBoardSize` should be calculated using the parameter `connectSize`.

Methods

Complete the following methods:

- `public boolean dropPiece(int column, char piece)` : This method adds a new piece to the board. If the user selects a column not available on the board, the method should return false. This should use the `nextSpot` array to determine the first available row in a column for their piece to be placed. If the column is full, the method should also return false. Once the piece is added to the board array, the `nextSpot` array should be updated as well.
- `public int getLongestConnect(char piece)` : This method should return an integer of the longest connected series of characters that match the player's piece parameter. This will look at the 2D array `board` in the vertical, horizontal, forward slash, and backward slash directions to see if there is a new longest connect length compared to the current longest length.
- `public int getTotalPieceCount()` : This method will return the integer of total pieces played
- `public boolean isBoardFull()`: This method will return true if the total pieces played is equal to the `totalBoardSize`
- `public String toString()` : This method will convert the board array to a string.

Player.java Implementation

The player class represents each player's own data such as name, piece symbol, wins, losses, and draws. The values for the instance fields will be assigned by the ConnectGame main method.

Class Constants

There are public static class constants for this class.

Instance Fields

Declare the following private instance fields:

- `name` : a String for the player's name.
- `piece` : a character symbol to represent the player's pieces on the board
- `win` : an integer to track the total wins a player has earned
- `loss` : an integer to track a total losses a player has earned
- `draw` : an integer to track the total draws a player has earned

Constructor

- `public Player(String name, char piece)` : This is the constructor of the class. The instance fields `name` and `piece` are initialized from the passed parameters. The `win`, `loss`, and `draw` fields are initialized to `0`.

Methods

Complete the following methods:

- `public String getName()` : this will return a string containing the player's name.
- `public char getPiece()` : this will return the player's piece character for the board
- `public int getWins()` : this will return the integer value of a player's win total
- `public int getLosses()` : this will return the integer value of the player's loss total
- `public int getDraws()` : this will return the integer value of the player's draw total
- `public void addWin()` : this will increment the player's win count
- `public void addLoss()` : this will increment the player's loss count
- `public void addDraw()` : this will increment the player's draw count
- `public void printStats()` : this will print a formatted string containing the player's name and win, loss, and draw counts

Testing

System Testing

System testing of Connect Game is described in the file `SystemTestPlan_CE.pdf` found in the `project_docs` directory.

The tests were executed from the `/bin` directory, using `.txt` files stored in the `test_files` directory. The `.txt` files, named after each test case, were accessed in the command line using the `<` operator, and represent the console input expected from a user. The console output from the program was directed to a `.txt` file of the same test case name prepended with `output_`, by using the `>` operator in the command line.

For example from the `/bin` directory:

```
$ java ConnectGame 4 Player1 Player2 <../test_files/testStartNewGameYes.txt
>../test_files/output_testStartNewGameYes.txt
```

There were no `test.java` Junit test files used to test `ConnectGame.java`