SFWR ENG 2AA4
Team Static Void
Group 7
Assignment 3
Dr. Alan Wassyng
Monday, April 6th, 2015

Group Information

Group 7
Team Static Void

Curtis Milo	1305877
Colin Gillespie	1330655
Zachary Bazen	1200979
Alexandra Rahman	1305735
Radhika Sharma	1150430
Jade Orkin-Fenster	1335823

Requirements: Assignment 1

1. Enable the user to set up a 7-by-6 Connect Four frame (7 columns, 6 rows) similar to the one shown above. The frame should include two different coloured discs (blue and red), one on each side of the frame similar to the above picture. At the start of a game there should be no discs already entered in the frame. Note: We will always use blue and red for the colour of the discs.

2. The order of play (blue first or red first) shall be determined randomly.

3. The user shall be able to choose to start a new game, or enter discs to represent the current state of a game by placing different coloured discs in the frame. This latter option shall be achieved by allowing the user to select a colour and then click on the position for that disc. When all the page 2 discs the user wants to place have been placed in the frame (the end of this phase being indicated in some way by the user), the system should analyze whether the current state is possible or not (a disc may not be supported by another disc underneath it), and all the errors shall be highlighted in some way on the screen. Errors include an unbalanced number of blue versus red discs, depending on which colour had the first move. The final state of this frame cannot include any winning connected “line” of 4 discs of the same colour.

Requirements: Assignment 2

1. The Connect Four frame is the same as in Assignment 1. All the rules of the game are as specified in Assignment 1.

2. This version requires you to be able to make moves, in turn. The moves have to be legal moves. The application has to recognize when the game has been won, or if the game cannot be won. The result of the game must be displayed at all times. Suggested results are: “Blue wins”; “Red wins”, “Game drawn”; and “Game in progress”. Each move must be checked to show that it is legal. (In the gravity game this is usually quite simple.)

3. The order of play (blue first or red first) shall be determined randomly.

4. The user shall be able to choose to start a new game, store an existing unfinished game, and restart a stored game.

Requirements: Assignment 3

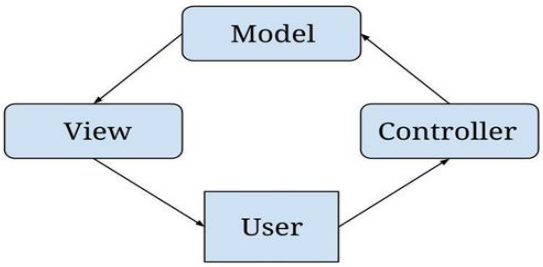
1. Being able to choose between two modes of operation:
    a. 2 player Connect Four, in which 2 people can play against each other;
    b. Or 1 player against the computer.

2. In 2 player mode, the behavior is the same as in assignment 2.

3. For playing against the computer:
    a. All the “rules” of the 2 player game should be the same for the game played against the computer.
    b. Specifically, the same mechanism should be used to decide who plays first, once it has been decided whether the computer plays blue or red.
    c. The user shall be able to choose to start a new game, store an existing unfinished game, and re-start a stored game.
    d. You will need an algorithm to determine the best next move for the computer. For this assignment, this does not have to be the optimum move, just try and make it give you a reasonably good move. It obviously has to be a legal move.

Application Decomposition

The game was partitioned into three different parts based off the MVC Pattern (Model, View, and Controller). This resulted in 4 main classes, Player, GameLogic, Grid, and Controller



1. Model
    a. Player
    b. GameLogic

2. View
    a. Grid

3. Controller
    a. Controller

Controller: The Controller class converts the users' clicks to information that the module can understand.

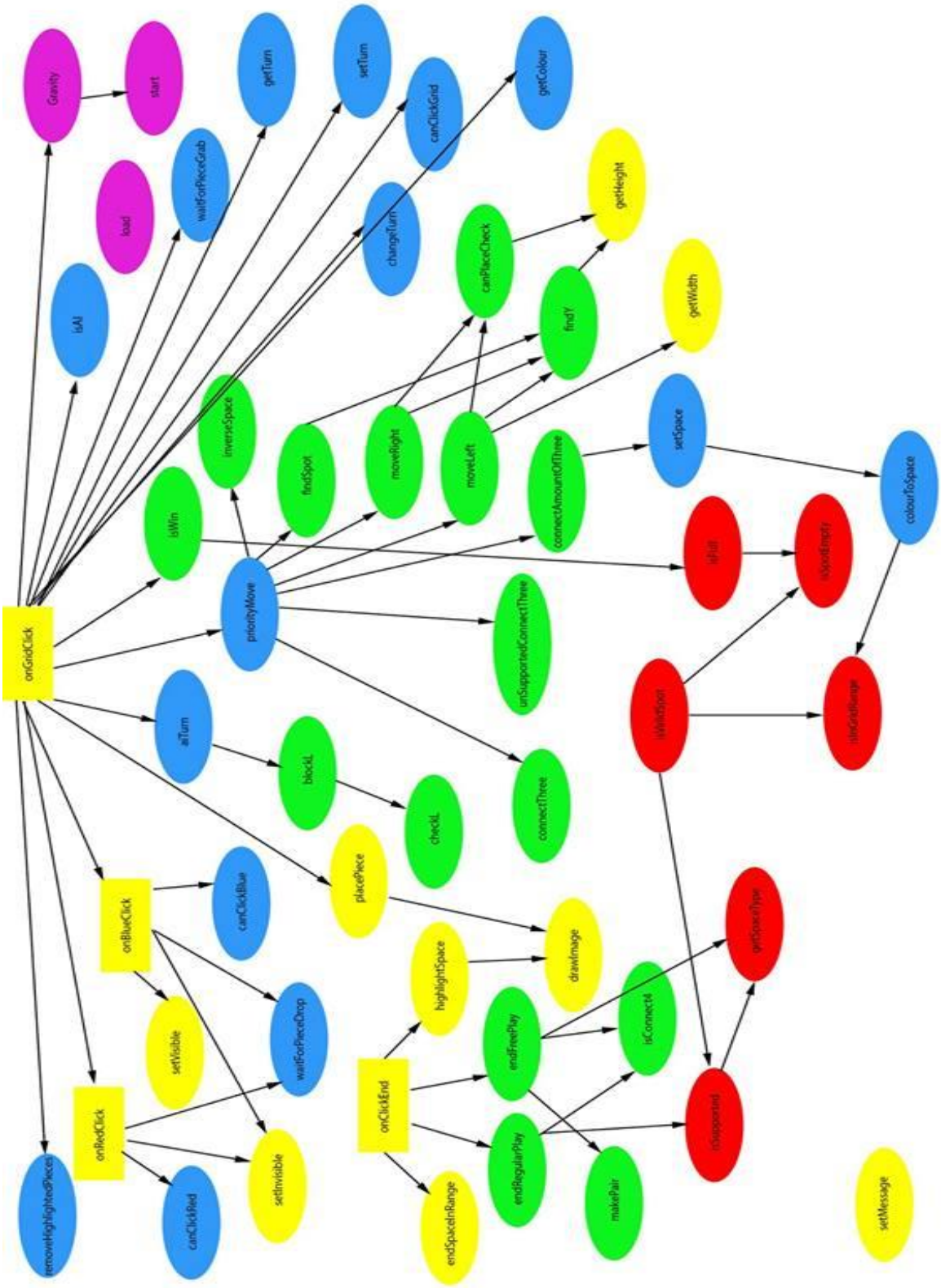
View: The Grid class displays the internal data to the user in the form of a Connect 4 board.

Model: The Player and Game Logic class work together to simulate the game. Both of these classes take commands from the controller class and output the subsequent data on the grid.

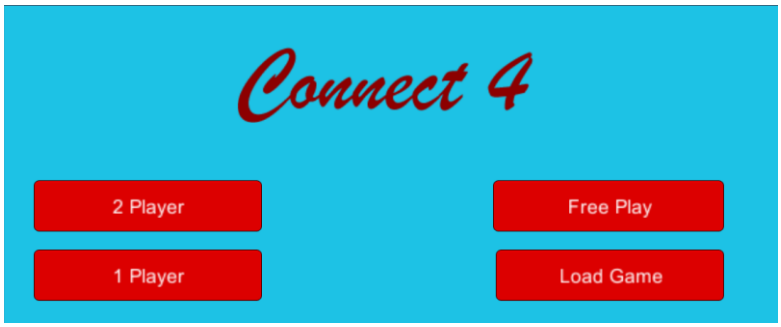
These classes work together to provide functionality to the Game module while still being efficient. The users' clicks are interpreted by the controller, and converted to data can be understood by the module. The information is then sent the Model which is broken down into two classes, Player and GameLogic. The GameLogic class is used to ensure that the rules of the game are being followed (i.e. illegal moves are not being made) as well as checking for wins or draws. The Player class ensures that the turns are alternated between players. This information is then sent to the grid which converts the modules interpretation of the data back to a visual form, which the user can easily understand.

This approach was used in conjunction with the principle of separation of concerns. It was known that this project would require certain elements to be handled in separate classes. Not doing so would result in confusion. Using the MVC pattern allowed for any changes to the game to be made without difficulty. Because of class independence, changes could be made faster than if the classes depended on each other. For example, if all a sudden the grid size changed to a 7x7 grid in the specifications of the project, the only change needed would be to one value in the grid class.

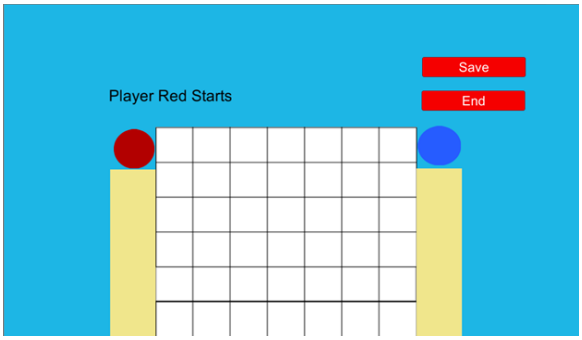
Uses Relationship



Description of User View:
Start Screen



The first screen that the users are shown is the start screen. This has the title of the game along with four buttons for the user to press. The first button labeled '2 Player' will begin a game in the 'Regular Mode' of the game where a random player will start and they will begin to take turns placing pieces; The button labelled 'Free Play' will set the mode to be a free play mode where the person will be able to place pieces down in whatever way they wish. When the "Load Game' button is clicked, a previously saved game will be loaded onto the play board, and the user has the option to continue playing. Lastly, the '1 Player' button allows the user to play the game with the AI as the opponent.



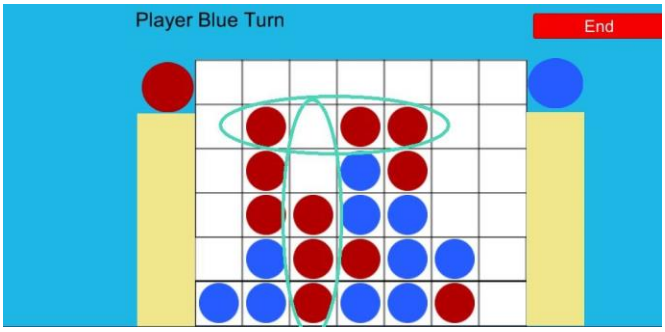
Whichever button is pressed, the game will initially display the above scene. If the user is in regular mode, then a message will display whose turn it is and what piece they have been assigned. The user will then be able to click on the appropriate piece. When the user clicks on a piece, that piece will become invisible, indicating that the piece can be placed wherever there is a valid spot. If the user is in free mode then the game will start with one of the pieces missing so that the player will be able to place pieces anywhere there is not already a piece by clicking on a square.

There is a message box that will display messages that will explain to the user any errors as to why the software cannot perform a certain task or will tell the user what can be done next, for example if the user clicks on the wrong colour it will display "Cannot click this piece".

When the user presses the end button if they are in a regular game it will simply end the application, however if the user is in free play it will calculate if the current pieces on the board make a valid gam

A.I Description

1. When the AI has the first move, it will always place the piece at (3, 0).
2. When the player has the first move and places his piece at (3, 0), the random function LRmove is called. LRmove will either output a 0 or a 1.
    - 0: AI places its piece to the right of the player's dropped piece.
    - 1: AI places its piece to the left of the player's dropped piece.
3. If the player does not place his piece at (3, 0), the AI will place his own piece at (3, 0) and step 2 is bypassed.
4. For every one of the AI's moves a random function LTRmove is called where the output is 0, 1, or 2.
    - 0: AI places its piece to the left of the player's dropped piece.
    - 1: AI places its piece on top of the player's dropped piece
    - 2: AI places its piece to the right of the player's dropped piece.
5. When the AI must place a piece to the right or left of the most recently dropped piece, and the player's piece is placed in column 0 or column 6, then the board wraps around. For example, if the player's piece is dropped in column 0 and the random function outputs a 0, the AI will place its piece in the column 6, and vice versa for column 6.
6. If the AI cannot place a piece in the designated column then it will check the next column to the right or left, according to the LTRmove or LRmove until it is possible to place its piece.
7. If the most recent piece drop fills a row and the AI is to place a piece on top of the most recently dropped piece then the random function LRmove is called and the AI's piece is placed accordingly.
8. When a connect3 is present, the AI will place its piece to complete the connect3. This will either block the opponent's connect4 or finish the AI's connect4. This is given highest priority.
    a. If there are two or more connect3s present then the AI's connect3 takes precedence.
9. When the special case unsupported connect3 is detected, as shown in Figure 1, then the AI will place its piece in the column that creates the connect3. In the figure's case the AI will place its piece at (2, 3), this will block the connect3 but support a connect4 (this is an example of what the algorithm is going to prevent).
10. When the AI's placed piece will support a connect4 for the opponent, then this move is not performed. This is given third highest priority
    a. If it supports a connect4 and the LTRmove returned a 0 or a 2, then step 6 is performed.
    b. If it supports a connect4 and the LTRmove returned a 1, then LRmove is called and the AI's piece is placed accordingly.



MIS

Space

Public Enum

Space{Red, Blue, Empty}

- enumeration for the types of spaces

GameState

Public Enum

GameState{Win, Ongoing, Draw}

- enumeration for the states the game can be in
- Refer back to requirement #2, Assignment 2. Allows the application to recognize whether the game has been won, results in a draw, or is ongoing.

GameLogic

Public Methods

GameState isWin(int x, int y)

- Method for checking which state the game is in. Can be of GameState Win, Draw, or Ongoing
- Calls method isConnect4
- Refer to requirement #2 for assignment 2. Allows the application to detect when the game has been won.

<table>
<tr><th colspan="3">isWin</th></tr>
<tr><td>isConnect4 = True</td><td colspan="2">endState.Win</td></tr>
<tr><td rowspan="2">isConnect4 = False</td><td>isFull = True</td><td>endState.Draw</td></tr>
<tr><td>isFull = False</td><td>endState.Ongoing</td></tr>
</table>

Black Box Testing

Test 1
Input: isConnect4 = True
Output: endState.Win
Expected Output: endState.Win
Result: Pass

Test 2
Input: isConnect4 = False & isFull = True
Output: endState.Draw
Expected Output: endState.Draw
Result: Pass

Test 3
Input: isConnect4 = False & isFull = False
Output: endState.Ongoing
Expected Output: endState.Ongoing
Result: Pass

White Box Testing

Test 1
Input: Grid object with (3,3), (4,2), (5,1), and (6,0) set to Space.Red ; (Grid, 5, 1)
Output: GameState.Win
Expected Output: GameState.Win
Result: Pass

Test 2
Input: Grid object with all spaces set to Space.Red ; (Grid, 5, 1)
Output: GameState.Draw
Expected Output: GameState.Draw
Result: Pass

Test 3
Input: Grid object with (3,3), (4,2), (5,1), and (6,0) set to Space.Red ; (Grid, 0, 0)
Output: GameState.Ongoing
Expected Output: GameState.Ongoing
Result: Pass

bool isConnect4(int x, int y)

- Checks to see if there are four pieces of the same colour beside one another that creates a straight line in any direction for the specified coordinates.
- Calls method getSpaceType.
- Refer back to requirement #3 of assignment 1 . Allows the user to check for whether the game has reached the state of four pieces in a row (diagonally, horizontally, or vertically) that are the same enumeration type of Colour.

isConnect4		
getSpaceType = Empty	return false	
getSpaceType = full	isConnect4 = vertical	return true
	isConnect4 = horizontal	
	isConnect4 = left diagonal	
	isConnect4 = right diagonal	
	isConnect4 ! = vertical	return false
	isConnect4 != horizontal	
	isConnect4 != left diagonal	
	isConnect4 != right diagonal	

Black Box Testing

Test 1
Input: Vertical connect four with reds at the bottom in free play mode.
Output: Connect4 is invalid
Expected Output: Connect4 is invalid
Result: Pass

Test 2
Input: Horizontal connect four with blues at the bottom in free play mode.
Output: Connect4 is invalid
Expected Output: Connect4 is invalid
Result: Pass

Test 3
Input: Diagonal connect four with reds at the bottom in free play mode.
Output: Connect4 is invalid
Expected Output: Connect4 is invalid
Result: Pass

Test 4
Input: Vertical connection of five pieces with reds at the bottom in free play mode.
Output: Invalid move
Expected Output: Invalid move
Result: Pass

Test 5
Input: Horizontal connection of five pieces with reds at the bottom in free play mode.
Output: Invalid move
Expected Output: Invalid move
Result: Pass

Test 6
Input: Diagonal connection of five pieces with reds at the bottom in free play mode.
Output: Invalid move
Expected Output: Invalid move
Result: Pass

Test 7
Input: Vertical connection of three pieces with reds at the bottom in free play mode.
Output: Invalid move
Expected Output: Invalid move
Result: Pass

Test 8
Input: Horizontal connection of three pieces with reds at the bottom in free play mode.
Output: invalid move
Expected Output: Invalid move
Result: Pass

Test 9
Input: Diagonal connection of three pieces with reds at the bottom in free play mode.
Output: Invalid move
Expected Output: Invalid move
Result: Pass

White Box Testing

Test 1
Input: Grid object with (1,0), (2,0), (3,0), and (4,0) set to Space.Red ; (Grid, 2, 0)
Output: True
Expected Output: True
Result: Pass
Horizontal connect four

Test 2
Input: Grid object with (1,0), (2,0), (3,0), and (4,0) set to Space.Red ; (Grid, 0, 0)
Output: False
Expected Output: False
Result: Pass
Empty space

Test 3
Input: Grid object with (0,0), (1,1), (2,2), and (3,3) set to Space.Red ; (Grid, 2, 2)
Output: True
Expected Output: True
Result: Pass
Upward-to-the-right diagonal connect four

Test 4
Input: Grid object with (3,3), (4,2), (5,1), and (6,0) set to Space.Red ; (Grid, 5, 1)
Output: True
Expected Output: True
Result: Pass
Downward-to-the-right diagonal connect four

Test 5
Input: Grid object with (0,0), (0,1), (0,2), and (0,3) set to Space.Red ; (Grid, 0, 2)
Output: True
Expected Output: True
Result: Pass
Vertical connect four

ArrayList endFreePlay()

- Checks to see if it is possible to play a valid game from that of the free play mode in it's current state of the board.
- Checks to see if there are any possible connect 4s already on the board or to be on the board in one turn.
- Returns an ArrayList containing the index positions of any possible errors with the current state of the board.
- Refer back to requirement #3 of assignment 1. Allows players to end the free play mode

endFreePlay	
Valid end = isConnect4	return true
Valid end = isSupported Everywhere	return true
Valid end = disks in range	return true
Valid end != isConnect4	return false
Valid end != isSupported Everywhere	return false
Valid end != disks in range	return false

Black Box Testing

Test 1
Input: Two connected red pieces on the bottom and two connected blue pieces on the bottom.
Output: endFreePlay is valid
Expected Output: endFreePlay is valid
Result: Pass

Test 2
Input: Two connected red pieces on the bottom and one blue pieces on the bottom.
Output: endFreePlay is valid
Expected Output: endFreePlay is valid
Result: Pass

Test 3
Input: One blue piece at position (0, 1).
Output: endFreePlay is valid
Expected Output: endFreePlay is valid
Result: Pass

Test 4
Input: Two connected red pieces on the bottom and no blue pieces.
Output: endFreePlay is invalid
Expected Output: endFreePlay is invalid
Result: Pass

Test 5
Input: Three connected red pieces on the bottom and no blue pieces.
Output: endFreePlay is invalid
Expected Output: endFreePlay is invalid
Result: Pass

Test 6
Input: Four connected blue pieces on the bottom and three red pieces on top of the blue pieces.
Output: endFreePlay is invalid
Expected Output: endFreePlay is invalid
Result: Pass

White Box Testing

Test 1
Input: Grid object with all spaces set to Space.Empty
Output: [ ]
Expected Output: [ ]
Result: Pass
No errors to find

Test 2
Input: Grid object with (0,0), (0,1), (0,2), and (0,3) set to Space.Red
Output: [[0,0], [0,1], [0,2], [0,3], [0,4]]
Expected Output: [[0,0], [0,1], [0,2], [0,3]]
Result: Fail
See "Fail" note below

Test 3
Input: Grid object with (0,0), (0,1), and (0,2) set to Space.Red
Output: [[0,0], [0,1], [0,2], [0,3]]
Expected Output: [[0,0], [0,1], [0,2], [0,3]]
Result: Pass
Finds spots where adding a piece in the next turn will result in a connect four win

Test 4
Input: Grid object with (0,5) set to Space.Red
Output: [[0,5]]
Expected Output: [[0,5]]
Result: Pass
Unsupported piece is found

Test 5
Input: Grid object with (0,0), (0,1), (1,0), and (1,1) set to Space.Red
Output: [ ]
Expected Output: [ ]
Result: Pass
This method does NOT check for balanced pieces (onClickEnd handles this)

Test 6
Input: Grid object with (1,0), (2,0), and (3,0) set to Space.Red
Output: [[0,0], [4,0]]
Expected Output: [[0,0], [4,0]]
Result: Pass
Finds spots where adding a piece in the next turn will result in a connect four win

Test 7
Input: Grid object with (1,0), (2,0), (3,0), and (4,0) set to Space.Red
Output: [[0,0], [1,0], [2,0], [3,0], [4,0], [5,0]]
Expected Output: [[1,0], [2,0], [3,0], [4,0]]
Result: Fail
Function highlights both a connect four that has been found and spots where adding a piece in the next turn results in a connect four. If a connect four is already present, other information is unnecessary

ArrayList endRegularPlay (Grid g)

- Returns an array of the grid location coordinates of the winning four pieces
- Refer to requirement 3 of assignment 1, this method allows for the checking if the game has been won or not

endRegularPlay	
isConnect4 == True	Return winningFour
isConnect4 == False	errorPoint

Black Box Testing

Test 1
Input: isConnect4 == True
Output: winningFour
Expected Output: winningFour
Result: Pass

Test 2
Input: isConnect4 == False
Output: errorPoint
Expected Output: errorPoint
Result: Pass

White Box Testing

Test 1
Input: Grid object with all spaces set to Space.Empty
Output: [ ]
Expected Output: [ ]
Result: Pass
There are no winning four connected pieces, so nothing to return

Test 2
Input: Grid object with (0,0), (0,1), (0,2), and (0,3) set to Space.Red
Output: [[0,0], [0,1], [0,2], [0,3]]
Expected Output: [[0,0], [0,1], [0,2], [0,3]]
Result: Pass
Finds the winning four connected pieces

Test 3
Input: Grid object with all spaces set to Space.Red
Output: an array of all pair arrays
Expected Output: an array of all pair arrays
Result: Pass
See isConnect4. This method must be called as soon as the winning piece is placed, because it does not only return four locations - it will return all possible connect-fours that it finds

int[] connectThree(Grid g, Space c)

- Checks to see if there are three pieces in succession, if there are three pieces of one colour that are in succession, the method returns the position of where the possible fourth piece will be. If there is no succession of three pieces of one colour on the grid, the method returns null.
- refer to requirement 3.4, this allows the AI to determine the next best move

connectThree		
g.isValidSpot(i,j)&& g.getSpaceType(i,j) == Space.Empty	isConnect = True	Return pair
	isConnect = False	Iterate through board
g.isValidSpot(i,j)&& g.getSpaceType(i,j) !== Space.Empty		Iterate through board

Black Box Testing

Test 1
Input: g.isValidSpot(3,3) && g.getSpaceType(3,3) == Space.Empty && isConnect = True
Output: Returns pair (3,3)
Expected Output: Return pair (3,3)
Result: Pass

Test 2
Input: g.isValidSpot(3,3)&& g.getSpaceType(3,3) == Space.Empty && isConnect = False
Output: Iterates through board
Expected Output: Iterates through board
Result: Pass

Test 3
Input: g.isValidSpot(3,3) && g.getSpaceType(3,3) !== Space.Empty
Output: Iterates through board
Expected Output: Iterates through board
Result: Pass

int connectAmountOfThree(Grid g, Space c)

- Returns the number of times that three pieces of one colour are in succession.
- If there is no succession of three pieces of one colour on the grid, the method returns null.
- refer to requirement 3.4, this allows the AI to determine the next best move

connectAmountOfThree		
g.isValidSpot(i,j) && g.getSpaceType(i,j) == Space.Empty	isConnect = True	Count ++
	isConnect = False	Iterate through board
g.isValidSpot(i,j) && g.getSpaceType(i,j) !== Space.Empty		Iterate through board

Black Box Testing

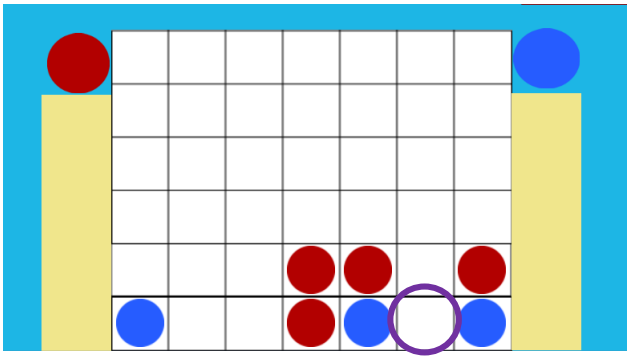
Test 1
Input: g.isValidSpot(3,3) && g.getSpaceType(3,3) == Space.Empty && isConnect = True
Output: Count ++
Expected Output: Count ++
Result: Pass

Test 2
Input: g.isValidSpot(3,3) && g.getSpaceType(3,3) == Space.Empty && isConnect = False
Output: Iterate through board
Expected Output: Iterate through board
Result: Pass

Test 3
Input: g.isValidSpot(3,3) && g.getSpaceType(3,3) !== Space.Empty
Output: Iterate through board
Expected Output: Iterate through board
Result: Pass

void unSupportedConnectThree(Grid g, ArrayList dontPlace, Space c)

- In the following diagram the player is playing with red pieces and the A.I is playing with blue pieces.
- This method will check if the player could potentially have four in a row, if a piece from the A.I is placed in the indicated spot (spot indicated with purple circle on figure)
- If there is no succession of three pieces of one colour on the grid, the method returns null.
- refer to requirement 3.4, this allows the AI to determine the next best move



unSupportedConnectThree		
g.getSpaceType(i,j) == Space.Empty	isConnect = True	Return pair
	isConnect = False	Iterate through board
g.getSpaceType(i,j) !== Space.Empty		Iterate through board

Black Box Testing

Test 1
Input: g.getSpaceType(3,3) == Space.Empty && isConnect = True
Output: Returns pair (3,3)
Expected Output: Returns pair (3,3)
Result: Pass

Test 2
Input: g.getSpaceType(3,3) == Space.Empty && isConnect = False
Output: Iterate through board
Expected Output: Iterate through board
Result: Pass

Test 3
Input: g.getSpaceType(3,3) !== Space.Empty
Output: Iterate through board
Expected Output: Iterate through board
Result: Pass

int[] blockL(Grid g, Space c)

- Checks for all “L” blocks within the board
- refer to requirement 3.4, this allows the AI to determine the next best move

blockL	
checkL(g,i,j,c) = True	Return pair
checkL(g,i,j,c) = False	Iterate through board

Black Box Testing

Test 1
Input: checkL(g,3,3, c) = True
Output: Return pair (3,3)
Expected Output: Return pair (3,3)
Result: Pass

Test 2
Input: checkL(g,3,3,c) = False
Output: Iterate through board
Expected Output: Iterate through board
Result: Pass

int[] findSpot(Grid g, Arraylist dontPlace, int x_pos)

- Follows a series of checks to determine where the AI should place the piece if there are no moves that will advance the AI's connected pieces, or block the opponent's connected pieces
- refer to requirement 3.4, this allows the AI to determine the next best move

findSpot		
canPlaceCheck(g,dontPlace,x) = True	Return pair	
canPlaceCheck(g,dontPlace,x) = False	rand>0	moveRight(g,dontPlace,x,0)
	rand!>0	moveLeft(g,dontPlace,x,0)

Black Box Testing

Test 1
Input: canPlaceCheck(g,dontPlace,x) = True
Output: Returns pair
Expected Output: Returns pair
Result: Pass

Test 2
Input: canPlaceCheck(g,dontPlace,x) = False && rand>0
Output: moveRight(g,dontPlace,x,0)
Expected Output: moveRight(g,dontPlace,x,0)
Result: Pass

Test 3
Input: canPlaceCheck(g,dontPlace,x) = False && rand!>0
Output: moveLeft(g,dontPlace,x,0)
Expected Output: moveLeft(g,dontPlace,x,0)
Result: Pass

int[] moveRight(Grid g, Arraylist dontPlace, int x_pos)

- Places a piece of the A.I's color to the right of the players last placed piece
- refer to requirement 3.4, this allows the AI to determine the next best move

int[] moveLeft(Grid g, Arraylist dontPlace, int x_pos)

- Places a piece of the A.I's color to the left of the players last placed piece
- refer to requirement 3.4, this allows the AI to determine the next best move

bool canPlaceCheck(Grid g, Arraylist dontPlace, int x_pos)

- Finds available spots where the A.I can place pieces without advancing the opponent's existing connect4's
- refer to requirement 3.4, this allows the AI to determine the next best move

canPlaceCheck		
pair[0] == x_Pos	Return False	
pair[0 !]== x_Pos	g.getSpaceType (x_Pos, g.getHeight()-1) != Space.Empty	Return False
	g.getSpaceType (x_Pos, g.getHeight()-1) = Space.Empty	Return True

Black Box Testing

Test 1
Input: pair[0] == x_Pos
Output: False
Expected Output: False
Result: Pass

Test 2
Input: pair[0 !]== x_Pos && g.getSpaceType (x_Pos, g.getHeight()-1) != Space.Empty
Output: False
Expected Output: False
Result: Pass

Test 3
Input: pair[0 !]== x_Pos && g.getSpaceType (x_Pos, g.getHeight()-1) = Space.Empty
Output: True
Expected Output: True
Result: Pass

int findY(Grid g, int x)

- Finds the height of the given x position on the given Grid
- refer to requirement 3.4, this allows the AI to determine the next best move

findY	
x < 0 || x > g.getWidth ()-1 || g.getSpaceType(x, y) != Space.Empty	return -1
x < 0 || x > g.getWidth ()-1 || g.getSpaceType(x, y) = Space.Empty	Iterate through board till g.isValidSpot(x,y), return y

Black Box Testing

Test 1
Input: x < 0 || x > g.getWidth ()-1 || g.getSpaceType(3,4) != Space.Empty
Output: -1
Expected Output: -1
Result: Pass

Test 2
Input: x < 0 || x > g.getWidth ()-1 || g.getSpaceType(3,4) = Space.Empty
Output: 4
Expected Output: 4
Result: Pass

Space inverseSpace(Space c)

- Returns the inverse of the given space c
- refer to requirement 3.4, this allows the AI to determine the next best move

inverseSpace		
c == Space.Red	return Space.Blue	
c !== Space.Red	c == Space.Blue	return Space.Red
	c !== Space.Blue	return Space.Empty

Black Box Testing

Test 1
Input: c == Space.Red
Output: returns Space.Blue
Expected Output: returns Space.Blue
Result: Pass
Test 2
Input: c !== Space.Red && c == Space.Blue
Output: returns Space.Red
Expected Output: returns Space.Red
Result: Pass

Test 3
Input: c !== Space.Red && c !== Space.Blue
Output: returns Space.Empty
Expected Output: returns Space.Empty
Result: Pass

Grid

Public Constructor

Grid ()

- Initializes the internal representation of connectFourBoard by initializing all entries of type enumeration Space.Empty
- Refer back to requirement #1, Assignment 1. Allows the application to set-up a new 7 by 6 playing board and initialize the entire board with enumeration Space.Empty.

Public Methods

int getWidth()

- Returns the BOARD_WIDTH field, which is the field in the x-component on the board.
- Refer back to requirement #1, Assignment 1, requires the board to have a width of 7. Allows the user retrieve the value of the constant BOARD_WIDTH.

int getHeight()

- Returns the BOARD_HEIGHT field, which is the field in the y-component on the board
- Refer back to requirement #1, Assignment 1, requires the board to have a height of 7. Allows the user retrieve the value of the constant BOARD_HEIGHT.

bool isFull()

- checks to see if all spots on the board are filled with pieces
- Refer back to requirement #2, Assignment 2. Allows the user to check whether all indices of the connectFourBoard field are not of enumeration space type Empty.

isFull	
For all pieces in top row, isSpotEmpty = True	False
For all pieces in top row, isSpotEmpty = False	True

Black Box Testing

Test 1
Input: For all pieces in top row, isSpotEmpty = True
Output: False
Expected Output: False
Result: Pass

Test 2
Input: For all pieces in top row, isSpotEmpty = False
Output: True
Expected Output: True
Result: Pass

White Box Testing

Test 1
Input: Grid object with all spaces set to Space.Empty
Output: False
Expected Output: False
Result: Pass

Test 2
Input: Grid object with all spaces set to Space.Red
Output: True
Expected Output: True
Result: Pass

Test 3
Input: Grid object with only top row set to Space.Red and rest set to Space.Empty
Output: True
Expected Output: False
Result: Fail
As an optimisation, the method only checks the top row of the grid on the assumption that during gameplay, it will be impossible to fill the top row without filling all other spaces on the grid. This input must be avoided through proper implementation of the game

bool isValidSpot(int x, int y)

- Checks to see if a piece can be placed in the spot at the coordinates specified.
- Refer back to requirement #3, assignment 1. Allows the program to check whether the current move is valid. Checks to see if the spot is empty and supported, if both are untrue, then the current state is not valid.

isValidSpot		
Spot empty	Is supported	True
	Is not supported	False
Spot not empty	False	

Black Box Testing

Test 1
Input: In play board and spot empty and is supported
Expected Output: True
Output: True
Result: Pass

Test 2
Input: In play board and spot empty and is not supported
Expected Output: False
Output: False
Result: Pass

Test 3
Input: In play board and spot not empty
Expected Output: False
Output: False
Result: Pass

Test 4
Input: Out of play board
Expected Output: False
Output: False
Result: Pass

White Box Testing

Test 1
Input: Grid object with all spaces set to Space.Empty, (0,0)
Expected Output: True
Output: True
Result: Pass
isSpotEmpty = True, isSupported = True

Test 2
Input: Grid object with all spaces set to Space.Empty, (0,5)
Expected Output: False
Output: False
Result: Pass
isSpotEmpty = True, isSupported = False

Test 3
Input: Grid object with all spaces set to Space.Red, (0,5)
Expected Output: False
Output: False
Result: Pass
isSpotEmpty = False, isSupported = True

bool isSupported(int x, int y)

- "Supported" means there are no empty spaces below the selected space within the grid.
- Checks that all the spaces below the space where the user is placing the piece, are full
- If the piece is in the bottom row, the piece is automatically supported
- Refer back to requirement #3 of assignment 1. Allows the program to check whether the current state is possible. If the current piece is not supported then the current state is not possible.

isSupported			
Is a bottom spot	True		
Is not a bottom spot	Spot below is empty	False	
	Spot below is filled	All spots below are supported	True
		Some spots below are not supported	False

Black Box Testing

Test 1
Input: Is a bottom spot
Output: True
Expected Output: True
Result: Pass

Test 2
Input:Is not a bottom spot and spot below is empty
Output:False
Expected Output: False
Result: Pass

Test 3
Input: Is not a bottom spot and spot below and all spots below are supported
Output: True
Expected Output:True
Result: Pass

Test 4
Input:Is not a bottom spot and spot below and some spots below are not supported
Output: False
Expected Output: False
Result: Pass

White Box Testing

Test 1
Input: (5,0)
Output: True
Expected Output: True
Result: Pass
This coordinate is on the bottom row so it is supported

Test 2
Input: (100,0)
Output: True
Expected Output: False
Result: Fail
This method does not check if the coordinate is in range, and thus, locations that are out of bounds are evaluated as supported. This input must be avoided in the implementation of the software (e.g. only call this method through isValidSpot)

Test 3
Input: (100,5)
Output: IndexOutOfBounds Runtime error
Expected Output: IndexOutOfBounds Runtime error
Result: Pass
Space doesn't exist (board isn't this large)

Test 4
Input: Grid object with all spaces set to Space.Empty, (1,1)
Output: False
Expected Output: False
Result: Pass
(1,1) does not have a piece at (1,0) to support it

Test 5
Input: Grid object with all spaces set to Space.Red, (6,5)
Output: True
Expected Output: True
Result: Pass
All pieces have pieces below them to support them

Test 6
Input: Grid object with all spaces set to Space.Red except bottom row set to Space.Empty, (6,5)
Output: False
Expected Output: False
Result: Pass
(6,5) is not supported because although there are pieces at (6,4), (6,3), (6,2), and (6,1), there is none at (6,0)

void setSpace(int x, int y, Colour colour)

- Sets the selected spot to the desired Colour enum.
- Refer back to requirement #3, Assignment 1. Allows the players to place their pieces when the move is valid.

White Box Testing

Test 1
Input: (6,5, Space.Red)
Output: connectFourBoard [x, y] = Space.Red
Expected Output: connectFourBoard [x, y] = Space.Red
Result: Pass
Set a space to Red

Test 2
Input: (6,5, Space.Empty)
Output: connectFourBoard [x, y] = Space.Empty
Expected Output: connectFourBoard [x, y] = Space.Empty
Result: Pass
Set a space to Empty

Test 3
Input: (6,100, Space.Empty)
Output: IndexOutOfBounds Runtime error
Expected Output: IndexOutOfBounds Runtime error
Result: Pass
Set a space that doesn't exist (board isn't this large)

Space getSpaceType(int x, int y)

- Checks to see if the player has clicked within the playing area.
- If the player has clicked within the playing area, returns the Space enum at the specified coordinates.
- Refer back to requirement #3. Allows the connect four method to know what types of pieces are where to check for connect four.

getSpaceType	
In play board	Return red or Return Blue
Out of play board	IndexOutOfBounds Exception

Black Box Testing

Test 1
Input: In play board and piece is red
Output: Red
Expected Output: Red
Result: Pass

Test 2
Input: In play board and piece is blue
Output: Blue
Expected Output: Blue
Result: Pass

Test 3
Input: Out of play board
Output: IndexOutOfBounds exception
Expected Output: IndexOutOfBounds exception
Result: Pass

White Box Testing

Test 1
Input: Grid object with all spaces set to Space.Empty, (6,5)
Output: Space.Empty
Expected Output: Space.Empty
Result: Pass

Test 2
Input: Grid object with all spaces set to Space.Red, (6,5)
Output: Space.Red
Expected Output: Space.Red
Result: Pass

Test 3
Input: Grid object with all spaces set to Space.Blue, (6,5)
Output: Space.Blue
Expected Output: Space.Blue
Result: Pass

Test 4
Input: Grid object with all spaces set to Space.Empty, (100,100)
Output: IndexOutOfBounds Runtime error
Expected Output: IndexOutOfBounds Runtime error
Result: Pass

bool endSpaceInRange()

- Only valid in free play mode.
- checks if the number of blue pieces and number of red pieces have a difference of either 1 or 0
- Refer back to requirement #3 of assignment 1. Checks to see that after free play mode, there is a difference of either 1 or 0 between the number of red and blue pieces

endSpaceInRange	
Red – Blue = 1 | 0	return true
Red - Blue ! = 1 | 0	return false

Black Box Testing

Test 1
Input: Two red pieces on top on two other red pieces and one blue piece.
Output: endSpaceInRange is invalid
Expected Output: endSpaceInRange is invalid
Result: Pass

Test 2
Input: Two connected red pieces and one blue piece.
Output: endSpaceInRange is valid
Expected Output: endSpaceInRange is valid
Result: Pass

Test 3
Input: Two blue pieces on top on two other blue pieces, a disconnected blue piece and one red piece.
Output: endSpaceInRange is invalid
Expected Output: endSpaceInRange is invalid
Result: Pass

Test 4
Input: Two connected blue pieces and one red piece.
Output: endSpaceInRange is valid
Expected Output: endSpaceInRange is valid
Result: Pass

Test 5
Input: No pieces on the board.
Output: endSpaceInRange is valid
Expected Output: endSpaceInRange is valid
Result: Pass

White Box Testing

Test 1
Input: Grid object with all spaces set to Space.Empty
Output: True
Expected Output: True
Result: Pass
With no Red or Blue pieces, they must be balanced

Test 2
Input: Grid object with all spaces set to Space.Red
Output: False
Expected Output: False
Result: Pass
Excess of Red pieces over Blue pieces

Test 3
Input: Grid object with the spaces in 3 rows set to Space.Red and 3 rows set to Space.Blue
Output: True
Expected Output: True
Result: Pass
Exact balance where the number of Red pieces equals the number of Blue pieces

Test 4
Input: Grid object with 3 rows and 1 space set to Space.Red, the rest set to Space.Blue
Output: True
Expected Output: True
Result: Pass
The number of Red pieces is equal to the number of Blue pieces plus one. This should be considered balanced

Controller

Public Constructors

void Start()

- Initializes the game.
- Determines which game mode will be played (Regular or Free Play).
- Determines which player starts at random.
- Refer back to requirement #3, assignment 1. This is a method, which determines which mode the player is in and initializes the game.

Public Methods

void autoClick()

- Determines which player's turn it is to place a piece on the Grid

autoClick	
players [turn].getColour () == Space.Red	onRedClick()
players [turn].getColour () !== Space.Red	onBlueClick()

Black Box Testing

Test 1
Input: players [turn].getColour () == Space.Red
Output: onRedClick()
Expected Output: onRedClick()
Result: Pass

Test 2
Input: players [turn].getColour () !== Space.Red
Output: onBlueClick()
Expected Output: onBlueClick()
Result: Pass

void onRedClick()

- In Regular Mode:
    - If it is currently red's turn, the current state is waitForPieceDrop and the red piece at the top of the screen is no longer visible.
    - If it is currently red's turn, and the blue piece at the top of the screen is selected an error message is displayed.
- In Free Play Mode
    - Sets freePlayColour to Colour.Red.
    - The red piece at the top of the screen becomes no longer visible.
    - The blue piece at the top of the screen stays visible.
- Refer back to requirement #3 of assignment 1. This method enables the red player to do any of the possible moves they would like if they are playing with the red pieces.

onRedClick		
Free_Play	Colour = Red	
Regular	Is Player Red's turn	Colour = Red
	Is not Player Red's turn	Error message

Black Box Testing

Test 1
Input: Free play
Output: Clicker is red
Expected Output: Clicker is red
Result: Pass

Test 2
Input: Regular and is player red's turn
Output: wait for piece drop
Expected Output: wait for piece drop
Result: Pass
Test 3
Input: Regular and is not Player red's turn
Output: Error message
Expected Output: Error message
Result: Pass

void onBlueClick()

- In Regular Mode:
    - If it is currently blue's turn, the current state is waiForPieceDrop and the blue piece at the top of the screen is no longer visible.
    - If it is currently blue's turn, and the red piece at the top of the screen is selected an error message is displayed.
- In Free Play Mode
    - Sets freePlayColour to Colour.Blue.
    - The blue piece at the top of the screen becomes no longer visible.
    - The red piece at the top of the screen stays visible.
- Refer back to requirement #3. This method enables the player to do any of the possible moves they would like if they are playing with the red pieces.

onBlueClick		
Free_Play	Colour = Blue	
Regular	Is Player Blue's turn	Colour = Blue
	Is not Player Blue's turn	Error message

Black Box Testing

Test 1
Input: Free_Play
Output: Clicker is blue
Expected Output: Clicker is blue
Result: Pass

Test 2
Input: Regular and is player blue's turn
Output: wait for piece drop
Expected Output: wait for piece drop
Result: Pass

Test 3
Input: Regular and is not player blue's turn
Output: Error message
Expected Output: Error message
Result: Pass

void onGridClick()

- In Regular Mode:
  - If the selected spot is valid, sets the player's pieceColour on the grid in that spot (both in the grid object and in the GUI).
  - Sets the pieceColour's ball in the top of the in the corner to visible .
  - Changes the player's state to waitForPieceGrab.
  - Swaps control to the other player.
- In Free Play Mode:
  - If the selected spot is valid, sets the player's pieceColour on the grid in that spot (both in the grid object and in the GUI).
- Refer back to requirement #3 of assignment 1. This method enables the player to click on the grid.

onGridClick			
Free_Play	Place piece		
Regular	Player can click grid	Is valid spot	Place piece
		Is not valid spot	Error message
	Player cannot click grid	Error message	

Black Box Testing

Test 1
Input: Free Play
Output: Place piece
Expected Output: Place piece
Result: Pass

Test 2
Input: Regular and player can click grid and valid spot
Output: Place piece
Expected Output: Place piece
Result: Pass

Test 3
Input: Regular and player can click grid and not valid spot
Output: Error message
Expected Output: Error message
Result: Pass

Test 4
Input: Regular and player cannot click grid
Output: Error message
Expected Output: Error message
Result: Pass

void turnForAI()

- Checks if it is the AI's turn to place a piece. If it is the AI's turn to place a piece, sets the AI's piece to invisible and waits 3 seconds before placing it's piece
- refer to requirement 3.4, this allows the AI to determine when to make a move

turnForAI	
players [turn].isAI () = True	AI's piece is set to invisible and is placed on the grid 3 seconds later
players [turn].isAI () = False	No action taken

Black Box Testing

Test 1
Input: players [turn].isAI () = True
Output: AI's piece is set to invisible and is placed on the grid 3 seconds later
Expected Output: AI's piece is set to invisible and is placed on the grid 3 seconds later
Result: Pass

Test 2
Input: players [turn].isAI () = False
Output: No action taken
Expected Output: No action taken
Result: Pass

void onClickEnd()

- In Regular Mode:
  - A button that allows the user to return to the start screen.
- In Free Play Mode:
  - Check if there are any errors with the current grid setup. If there are no errors, the user is sent back to the start screen. If there are errors, the incorrect pieces are highlighted.
- Refer to requirement #3 of assignment 1. This method is only used for the free play mode, and checks whether or not the current state of the grid is valid.

onClickEnd		
Regular Mode	End game	
Free_Play Mode	If |Blue pieces – Red pieces| = 0 or 1	End game
	If |Blue pieces – Red pieces| = greater than 1	Error message

Black Box Testing

Test 1
Input: Regular mode
Output: End Game
Expected Output: End Game
Result: Pass

Test 2
Input: Free play mode and |Blue pieces – Red pieces| = 0 or 1
Output: End Game
Expected Output: End Game
Result: Pass

Test 3
Input: Free play mode and |Blue pieces – Red pieces| = greater than 1
Output: Error message
Expected Output: Error message
Result: Pass

void changeTurn()

- Changes the player's turn (if turn is 1, change to 0 and vice versa).
- Refer back to requirement #3 of assignment 1 . This is a method that alternates player turns.

void changeTurn		
Turn	Equals 0	Turn =1
	Does not equal 0	Turn = 0

Black Box Testing

Test 1
Input: Turn is 0
Output: increment turn
Expected Output: increment turn
Result: Pass

Test 2
Input: turn is not 0
Output: turn equals zero
Expected Output: turn equals 0
Result: Pass

void placePiece(int x, int y, Colour c)

- Draws a piece at the selected coordinates of the grid on the GUI screen.

placePiece		
mode = Free_Play	c = Blue	piece = BlueChip
	c = Red	piece = RedChip
mode = Regular	c = Blue	piece = BlueChip
	c = Red	piece = RedChip

Black Box Testing

Test 1
Input: mode = Free_Play & c = Blue
Output: piece = BlueChip
Expected Output: piece = BlueChip
Result: Pass

Test 2
Input: mode = Free_Play & c = Red
Output: piece = RedChip
Expected Output: piece = RedChip
Result: Pass

Test 3
Input: mode = Regular & c = Blue
Output: piece = BlueChip
Expected Output: piece = BlueChip
Result: Pass

Test 4
Input: mode = Regular & c = Red
Output: piece = RedChip
Expected Output: piece = RedChip
Result: Pass

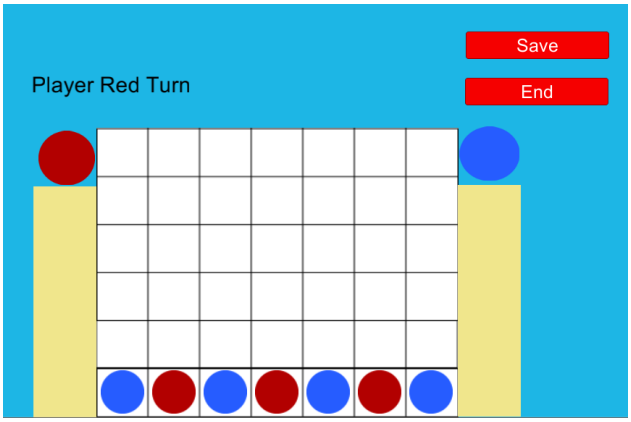
void saveGame()

- This method is an event driven stimulus of the save() method
- Waits for the corresponding button to be clicked, calls stringOfBoard(), and then applies the save() method on the result from stringOfBoard().

Black Box Testing

Test 1

Input:



Output:

Blue,Red,Blue,Red,Blue,Red,Blue
Empty,Empty,Empty,Empty,Empty,Empty,Empty
Empty,Empty,Empty,Empty,Empty,Empty,Empty
Empty,Empty,Empty,Empty,Empty,Empty,Empty
Empty,Empty,Empty,Empty,Empty,Empty,Empty
Empty,Empty,Empty,Empty,Empty,Empty,Empty
1
1

Expected Output:

Blue,Red,Blue,Red,Blue,Red,Blue
Empty,Empty,Empty,Empty,Empty,Empty,Empty
Empty,Empty,Empty,Empty,Empty,Empty,Empty
Empty,Empty,Empty,Empty,Empty,Empty,Empty
Empty,Empty,Empty,Empty,Empty,Empty,Empty
Empty,Empty,Empty,Empty,Empty,Empty,Empty
1
1

Result: Pass

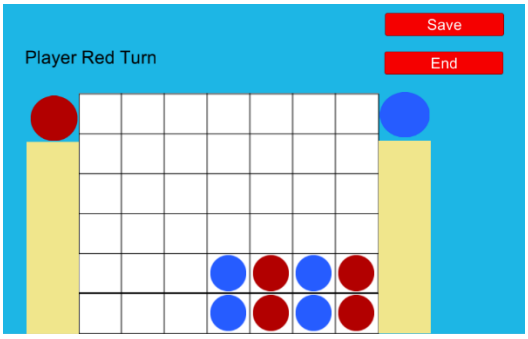
void loadGame()

- Retrieves the data of the saved game from the text file.
- Uses the load() method to read the data in the “Saved_Game” text file and then calls stringToSetSpace() on the data and creates a new Grid object corresponding to the data

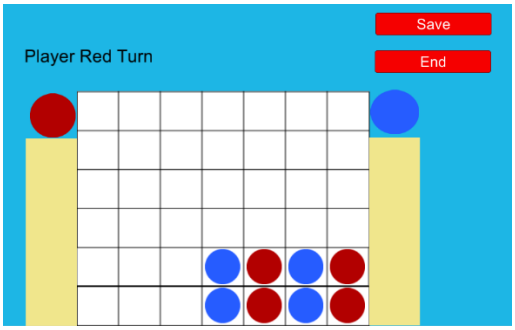
Black Box Testing

Test 1

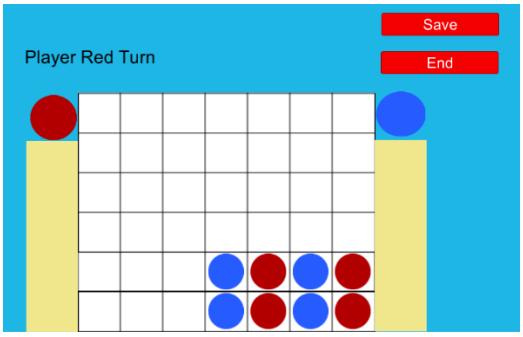
Input:



Output:



Expected Output:



Result: Pass

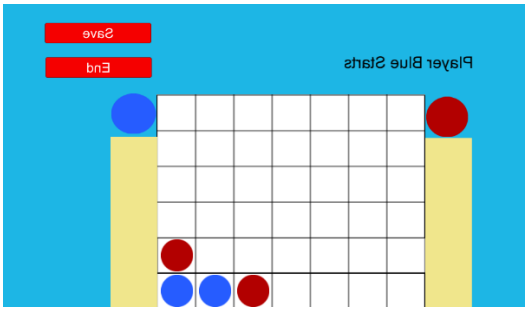
string stringOfBoard()

- Takes the current state of the grid class and converts it to an array of strings where each spot in the array is stored as either Empty, Red, or Blue.

Black Box Testing

Test 1

Input:



Output:

Empty,Empty,Empty,Empty,Red,Blue,Blue
Empty,Empty,Empty,Empty,Empty,Empty,Red
Empty,Empty,Empty,Empty,Empty,Empty,Empty
Empty,Empty,Empty,Empty,Empty,Empty,Empty
Empty,Empty,Empty,Empty,Empty,Empty,Empty
Empty,Empty,Empty,Empty,Empty,Empty,Empty
0
1

Expected Output:

Empty,Empty,Empty,Empty,Red,Blue,Blue
Empty,Empty,Empty,Empty,Empty,Empty,Red
Empty,Empty,Empty,Empty,Empty,Empty,Empty
Empty,Empty,Empty,Empty,Empty,Empty,Empty
Empty,Empty,Empty,Empty,Empty,Empty,Empty
Empty,Empty,Empty,Empty,Empty,Empty,Empty
0
1

Result: Pass

White Box Testing

Test 1

Input: Grid object with all spaces set to Space.Empty, Red's turn, Free Play mode

Output:

"Empty,Empty,Empty,Empty,Empty,Empty,Empty
Empty,Empty,Empty,Empty,Empty,Empty,Empty
Empty,Empty,Empty,Empty,Empty,Empty,Empty
Empty,Empty,Empty,Empty,Empty,Empty,Empty
Empty,Empty,Empty,Empty,Empty,Empty,Empty
Empty,Empty,Empty,Empty,Empty,Empty,Empty
1
0
-1

Expected Output:

"Empty,Empty,Empty,Empty,Empty,Empty,Empty
Empty,Empty,Empty,Empty,Empty,Empty,Empty
Empty,Empty,Empty,Empty,Empty,Empty,Empty
Empty,Empty,Empty,Empty,Empty,Empty,Empty
Empty,Empty,Empty,Empty,Empty,Empty,Empty
Empty,Empty,Empty,Empty,Empty,Empty,Empty
1
0
-1"

Result: Pass

Test 2

Input: Grid object with all spaces set to Space.Empty, Blue's turn, Regular mode, Player 0 is AI

Output:

"Empty,Empty,Empty,Empty,Empty,Empty,Empty
Empty,Empty,Empty,Empty,Empty,Empty,Empty
Empty,Empty,Empty,Empty,Empty,Empty,Empty
Empty,Empty,Empty,Empty,Empty,Empty,Empty
Empty,Empty,Empty,Empty,Empty,Empty,Empty
Empty,Empty,Empty,Empty,Empty,Empty,Empty
0
1
0"

Expected Output:

"Empty,Empty,Empty,Empty,Empty,Empty,Empty
Empty,Empty,Empty,Empty,Empty,Empty,Empty
Empty,Empty,Empty,Empty,Empty,Empty,Empty
Empty,Empty,Empty,Empty,Empty,Empty,Empty
Empty,Empty,Empty,Empty,Empty,Empty,Empty
Empty,Empty,Empty,Empty,Empty,Empty,Empty
0
1
0"

Result: Pass

Test 3
Input: Grid object with all spaces set to Space.Empty, Blue's turn, Regular mode, Player 0 is AI

Output:

"Empty,Empty,Empty,Empty,Empty,Empty,Empty
Empty,Empty,Empty,Empty,Empty,Empty,Empty
Empty,Empty,Empty,Empty,Empty,Empty,Empty
Empty,Empty,Empty,Empty,Empty,Empty,Empty
Empty,Empty,Empty,Empty,Empty,Empty,Empty
Empty,Empty,Empty,Empty,Empty,Empty,Empty
0
1
0"

Expected Output:

"Empty,Empty,Empty,Empty,Empty,Empty,Empty
Empty,Empty,Empty,Empty,Empty,Empty,Empty
Empty,Empty,Empty,Empty,Empty,Empty,Empty
Empty,Empty,Empty,Empty,Empty,Empty,Empty
Empty,Empty,Empty,Empty,Empty,Empty,Empty
Empty,Empty,Empty,Empty,Empty,Empty,Empty
0
1
0"

Result: Pass

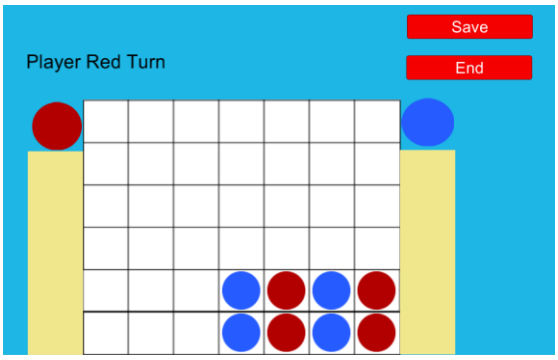
Space stringToSetSpace(string str)

- A helper method for loadGame().
- Checks the inputs and sets the position according to the inputs to the correct Space value.

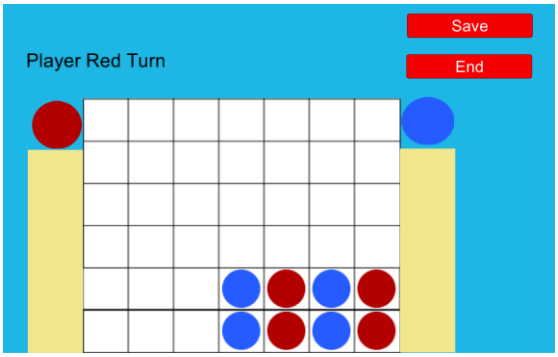
Black Box Testing

Test 1

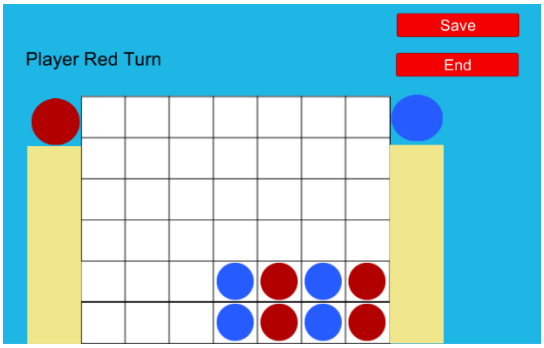
Input:



Output:



Expected Output:



Result: Pass

White Box Testing

Test 1
Input: "Red"
Output: Space.Red
Expected Output: Space.Red
Result: Pass

Test 1
Input: "Blue"
Output: Space.Blue
Expected Output: Space.Blue
Result: Pass

Test 1
Input: "test"
Output: Space.Empty
Expected Output: Space.Empty
Result: Pass

Player

Public Constructors

Player(Space pieceColour)

- Constructor to set the player's piece colour

Player (Space pieceColour, bool isAI)

- Creates a player object of Type Player.
- Player's piece colour is defined by pieceColour.
- bool isAI (optional: default false) determines whether or not the player is an AI or not.
- Refer back to requirement #3 of assignment 1. This is a constructor assigning the piece colour to the player.

Public Methods

int getTurn()

void setTurn(int turn)

bool canClickRed()

- Returns True if the player is able to click on the red piece at the top of the screen.
- Refer back to requirement #3. This is a method that determines if one is allowed to click the red game piece. Allows a player to start or continue playing the game.

bool canClickRed()				
Conditions	WaitingForPieceClick & Colour.Red	WaitingForPieceClick & Colour.Blue	WatingForGridClick & Colour.Red	WatingForGridClick & Colour.Blue
Return Value	True	False	False	False

Black Box Testing

Test 1
Input: State =Turn.WaitingForPieceClick, pieceColour =Colour.Red
Output: True
Expected Output: True
Result: Pass

Test 2
Input: State =Turn.WaitingForPieceClick, pieceColour =Colour.Blue
Output: False
Expected Output: False
Result: Pass

Test 3
Input: State =Turn.WatingForGridClick, pieceColour =Colour.Red
Output: False
Expected Output: False
Result: Pass

Test 4
Input: State =Turn.WatingForGridClick, pieceColour =Colour.Blue
Output: False
Expected Output: False
Result: Pass

bool canClickBlue()

- Returns true if we are able to click on the blue piece at the top of the screen.
- Refer back to requirement #3. This is a method that determines if one is allowed to click the blue game piece. Allows a player to start or continue playing the game.

bool canClickBlue()				
Conditions	WaitingForPieceClick & Colour.Red	WaitingForPieceClick & Colour.Blue	WatingForGridClick & Colour.Red	WatingForGridClick & Colour.Blue
Return Value	False	True	False	False

Black Box Testing

Test 1
Input: State =Turn.WaitingForPieceClick, & pieceColour =Colour.Red
Output: False
Expected Output: False
Result: Pass

Test 2
Input: State =Turn.WaitingForPieceClick, & pieceColour =Colour.Blue
Output: True
Expected Output: True
Result: Pass

Test 3
Input: State =Turn.WatingForGridClick, & pieceColour =Colour.Red
Output: False
Expected Output: False
Result: Pass

Test 4
Input: State =Turn.WatingForGridClick, pieceColour =Colour.Blue
Output: False
Expected Output: False
Result: Pass

bool canClickGrid()

- Returns true if the player has already clicked on a piece at the top of the screen, and is ready to place a piece on the Grid.
- Refer to requirement #3, Assignment 1. Allows a player to start or continue playing the game.

bool canClickGrid()		
Conditions	State =Turn.WaitingForPieceClick	State =Turn.WatingForGridClick
Return Value	False	True

Black Box Testing

Test 1
Input: State =Turn.WaitingForPieceClick
Output: False
Expected Output: False
Result: Pass

Test 2
Input: State =Turn.WatingForGridClick
Output: True
Expected Output: True
Result: Pass

bool isAI()

- Returns true if the players is an AI.
- Refer back to requirement #3 and #6.2. This is a constructor, which creates an artificial intelligent player.

bool isAI()		
Conditions	type=PlayerType.AI	type=PlayerType.Human
Return Value	True	False

Black Box Testing

Test 1
Input: type=PlayerType.AI
Output: True
Expected Output: True
Result: Pass

Test 2
Input: type=PlayerType.Human
Ouput: False
Expected Output: False
Result: Pass

Space getColour()

- Returns the color of the current player's piece

int[] aiTurn(Grid g)

- Determines where the AI wants to place a piece.
- Returns priorityMove(g)
- refer to requirement 3.4, this allows the AI to determine when to make a move
  .

void setAI(bool isAI)

- Sets the type field to AI
- refer to requirement 3.4, this allows the AI to determine when to make a move

setAI	
isAI = True	type =PlayerType.AI
isAI = False	No action taken

Black Box Testing

Test 1
Input: isAI = True
Output: type set to PlayerType.AI
Expected Output: type set to PlayerType.AI
Result: Pass

void waitForPieceDrop()

- Changes the current state to waiting for the human player. To get out of this state, a mouse click on the grid must be received.
- Refer back to requirement #3. This is a method, which analyzes the current state.

void waitForPieceDrop()	
Conditions	None
Return Value	State =Turn.WatingForGridClick

Black Box Testing

Test 1
Input: State =Turn.WatingForGridClick
Output: State =Turn.WatingForGridClick
Expected Output: Turn.WatingForGridClick
Result: Pass

Test 2
Input: State =Turn.WatingForPieceClick
Output: State =Turn.WatingForGridClick
Expected Output: Turn.WatingForGridClick
Result: Pass

void waitForPieceGrab()

- Changes the state of the player, to waiting for mouse click on the piece at the top of the screen. To leave this state a mouse click on a piece at the top of the screen must be received.
- Refer back to requirement #3. This is a method, which analyzes the current state.

void waitForPieceGrab()	
Conditions	None
Return Value	State =Turn.WatingForPieceClick

Black Box Testing

Test 1
Input: State =Turn.WatingForGridClick
Output: State =Turn.WatingForPieceClick
Expected Output: Turn.WatingForPieceClick
Result: Pass

Test 2
Input: State =Turn.WatingForPieceClick
Output: State =Turn.WatingForPieceClick
Expected Output: Turn.WatingForPieceClick
Result: Pass

MID

GameLogic

Private Constants

int BOARD_WIDTH
- constant for board width, set to integer 7

int BOARD_HEIGHT
- constant for board height, set to integer 6

int[] makePair(int x, int y)
- Creates an array of x and y values
- Used for error system

Private Methods

bool checkL(Grid g, int i , int j, Space c)
- checks for right or left “L” formations for the spot specified on the grid, where I and j are positions on the board
- refer to requirement 3.4, this allows the AI to determine the next best move

checkL		
(i>0) && (i<g.getWidth()-2) && (j>1) && (g.getSpaceType(i,j-2) == c) && (g.getSpaceType(i,j-1) == c)&& (g.getSpaceType(i,j) == Space.Empty)&& (g.getSpaceType(i+1,j) == c)&& (g.getSpaceType(i+2,j) == c)) = True		True
(i>0)&& (i<g.getWidth()-2) && (j>1) && (g.getSpaceType(i,j-2) == c) && (g.getSpaceType(i,j-1) == c) && (g.getSpaceType(i,j) == Space.Empty) && (g.getSpaceType(i+1,j) == c)&& (g.getSpaceType(i+2,j) == c)) = False	if((i>1)&& (i<g.getWidth()-1) && (j>1) && (g.getSpaceType(i,j-2) == c) && (g.getSpaceType(i,j-1) == c) && (g.getSpaceType(i,j) == Space.Empty) && (g.getSpaceType(i-1,j) == c)&& (g.getSpaceType(i-2,j) == c)) = True	True
	if((i>1)&& (i<g.getWidth()-1) && (j>1) && (g.getSpaceType(i,j-2) == c) && (g.getSpaceType(i,j-1) == c) && (g.getSpaceType(i,j) == Space.Empty) && (g.getSpaceType(i-1,j) ==c)&& (g.getSpaceType(i-2,j) == c)) = False	False

Black Box Testing

Test 1
Input:

(i>0) &&
(i<g.getWidth()-2) &&
(j>1) &&
(g.getSpaceType(i,j-2) == c) && (g.getSpaceType(i,j-1) == c)&&
(g.getSpaceType(i,j) == Space.Empty)&&
(g.getSpaceType(i+1,j) == c)&&
(g.getSpaceType(i+2,j) == c)) = True

Output: True
Expected Output: True
Result: Pass

Test 2
Input:

(i>0) &&
(i<g.getWidth()-2) &&
(j>1) &&
(g.getSpaceType(i,j-2) == c) && (g.getSpaceType(i,j-1) == c)&&
(g.getSpaceType(i,j) == Space.Empty)&&
(g.getSpaceType(i+1,j) == c)&&
(g.getSpaceType(i+2,j) == c)) = False

&&

if((i>1)&&
(i<g.getWidth()-1) &&
(j>1) &&
(g.getSpaceType(i,j-2) == c) &&
(g.getSpaceType(i,j-1) == c) &&
(g.getSpaceType(i,j) == Space.Empty) &&
(g.getSpaceType(i-1,j) == c)&&
(g.getSpaceType(i-2,j) == c)) = True

Output: True
Expected Output: True
Result: Pass

Test 3
Input:

```
(i>0) &&
(i<g.getWidth()-2) &&
(j>1) &&
(g.getSpaceType(i,j-2) == c) && (g.getSpaceType(i,j-1) == c)&&
(g.getSpaceType(i,j) == Space.Empty)&&
(g.getSpaceType(i+1,j) == c)&&
(g.getSpaceType(i+2,j) == c)) = False

 &&

if((i>1)&&
(i<g.getWidth()-1) &&
(j>1) &&
(g.getSpaceType(i,j-2) == c) &&
(g.getSpaceType(i,j-1) == c) &&
(g.getSpaceType(i,j) == Space.Empty) &&
(g.getSpaceType(i-1,j) == c)&&
(g.getSpaceType(i-2,j) == c)) = False
```

Output: False
Expected Output: False
Result: Pass

int[] moveRight(Grid g,ArrayList dontPlace,int x_Pos,int reps)

- Finds a valid spot on the right of the last place piece placed by the opponent, in which the AI can place a piece
- refer to requirement 3.4, this allows the AI to determine the next best move

moveRight		
canPlaceCheck(g,dontPlace,x_Pos) = True	return pair	
canPlaceCheck(g,dontPlace,x_Pos) = False	reps >= g.getWidth()	return pair
	reps !>= g.getWidth()	return moveRight(g,dontPlace,++x_Pos,++reps)

Black Box Testing

Test 1
Input: canPlaceCheck(g,dontPlace,x_Pos) = True
Output: returns pair
Expected Output: returns pair
Result: Pass

Test 2
Input: canPlaceCheck(g,dontPlace,x_Pos) = False && reps >= g.getWidth()
Output: returns pair
Expected Output: returns pair
Result: Pass

Test 3
Input: canPlaceCheck(g,dontPlace,x_Pos) = False && reps !>= g.getWidth()
Output: returns moveRight(g,dontPlace,++x_Pos,++reps)
Expected Output: returns moveRight(g,dontPlace,++x_Pos,++reps)
Result: Pass

int[] moveLeft(Grid g,ArrayList dontPlace,int x_Pos,int reps)

- Finds a valid spot on the left of the last place piece placed by the opponent, in which the AI can place a piece
- refer to requirement 3.4, this allows the AI to determine the next best move

moveLeft		
canPlaceCheck(g,dontPlace,x_Pos) = True	return pair	
canPlaceCheck(g,dontPlace,x_Pos) = False	reps >= g.getWidth()	return pair
	reps !>= g.getWidth()	return moveLeft(g,dontPlace,++x_Pos,++reps)

Black Box Testing

Test 1
Input: canPlaceCheck(g,dontPlace,x_Pos) = True
Output: returns pair
Expected Output: returns pair
Result: Pass

Test 2
Input: canPlaceCheck(g,dontPlace,x_Pos) = False && reps >= g.getWidth()
Output: returns pair
Expected Output: returns pair
Result: Pass

Test 3
Input: canPlaceCheck(g,dontPlace,x_Pos) = False && reps !>= g.getWidth()
Output: returns moveLeft(g,dontPlace,++x_Pos,++reps)
Expected Output: returns moveLeft(g,dontPlace,++x_Pos,++reps)
Result: Pass

int pieceRotate(Grid g,int x)

- If the AI is to place a piece on the right of the last placed piece, however the last placed piece is in the right most column, this method allows the grid to wrap around, and the piece to be in the left most column
- If the AI is to place a piece on the right of the last placed piece, and the desired column is already full of pieces, this method find the next available column to place the piece
- refer to requirement 3.4, this allows the AI to determine the next best move

pieceRotate		
x >= g.getWidth ()	return 0	
x !>= g.getWidth ()	x < 0	return g.getWidth()-1
	x >= 0	return x

Black Box Testing

Test 1
Input: x >= g.getWidth ()
Output: 0
Expected Output: 0
Result: Pass

Test 2
Input: x !>= g.getWidth () && x < 0
Output: returns g.getWidth()-1
Expected Output: returns g.getWidth()-1
Result: Pass

Test 3
Input: x !>= g.getWidth () && x >= 0
Output: returns x
Expected Output: returns x
Result: Pass

Grid

Private Constants

int BOARD_WIDTH

- The maximum width of the board that is being used (constant value of 7, stored as an integer, for this project)

int BOARD_HEIGHT

- The maximum height of the board that is being used (constant value of 6, stored as an integer, for this project).

Private Fields

Space[,] connectFourBoard

- For every index within the double array, there exist three possible enumeration values (Red, Blue, Empty}
- The internal representation of the connect four playing board.

Private Methods

bool isInGridRange(int x, int y)

- This method checks to see if the given coordinates are within the grid space
- Refer to requirement # 3 of assignment 1. This method allows for the game to detect whether a move is valid or not

isInGridRange	
(x < 0 || x > BOARD_WIDTH - 1) || (y < 0 || y > BOARD_HEIGHT - 1)	False
! (x < 0 || x > BOARD_WIDTH - 1) || (y < 0 || y > BOARD_HEIGHT - 1)	True

Black Box Testing

Test 1
Input: x = 8 & y = 9
Output: False
Expected Output: False
Result: Pass

Test 2
Input: x = 3 & y = 4
Output: True
Expected Output: True
Result: Pass

Test 3
Input: x = 3 & y = 9
Output: False
Expected Output: False
Result: Pass

Test 4
Input: x = 8 & y = 4
Output: False
Expected Output: False
Result: Pass

White Box Testing

Test 1
Input: (-1,5)
Output: False
Expected Output: False
Result: Pass

Test 2
Input: (100,5)
Output: False
Expected Output: False
Result: Pass

Test 3
Input: (5,-1)
Output: False
Expected Output: False
Result: Pass

Test 4
Input: (-5,100)
Output: False
Expected Output: False
Result: Pass

Test 5
Input: (6,5)
Output: True
Expected Output: True
Result: Pass

bool isSpotEmpty(int x, int y)

- Checks to see if the player has clicked within the playing grid area or not.
- If the player has clicked within the playing area, the method returns true (if spot is empty) or false (if spot is not empty).
- Refer back to requirement #3. Allows the program to check whether the current state is possible. If there is a piece in the selected spot, then the current state is not possible.

isSpotEmpty		
Is in Grid range	Empty Spot	True
	Full Spot	False
Is not in Grid range	False	

Black Box Testing

Test 1
Input: In play board and empty spot
Expected Output: True
Output: True
Result: Pass

Test 2
Input: In play board and full spot
Expected Output: False
Output: False
Result: Pass

Test 3
Input: Out of play board
Expected Output: False
Output: False
Result: Pass

White Box Testing

Test 1
Input: Grid object with all spaces set to Space.Empty, (i,j) in double for-loop
Expected Output: True
Output: True
Result: Pass

Test 2
Input: Grid object with all spaces set to Space.Red, (i,j) in double for-loop
Expected Output: False
Output: False
Result: Pass

Test 3
Input: Grid object with all spaces set to Space.Blue, (i,j) in double for-loop
Expected Output: False
Output: False
Result: Pass

Test 4
Input: Grid object with all spaces set to Space.Empty, (100,100)
Expected Output: False
Output: False
Result: Pass

Controller

Private Enumeration

Mode{Regular, Free_Play}

- Refer back to requirement #3. This is a constructor for the mode.

Private Constants

int GRID_OFFSET_X

- The grid on the x-axis becomes active 291 units to the right.

int GRID_OFFSET_Y

- The grid on the y-axis becomes active 0 units up.

int SPACE_WIDTH

- Each space in which a piece can be placed is 71 units wide.

int SPACE_HEIGHT

- Each space in which a piece can be placed is 66 units tall.

Int CIRCLE_RADIUS

- The radius of each piece is 35 units

int yAboveGrid = 425

- The correct y-coordinate for pieces to be placed directly above the grid (so that they can be dropped down)

Private Fields

bool PlaceAI = false

- Initializes AI to false

bool finishedGame = false

- Initializes finishedGame to false

Grid grid

- The grid which has 6 rows and 7 columns.

Mode mode

- The mode in which you can play, either Regular or Free_Play.

Space freePlayColour

- The colour of the piece you are using in the Free_Play mode.

Player[] players

- An array of the players so that a player can be dynamically chosen.

int turn

- Used to determine who starts.

Private Methods

void setMode()

- Sets the mode to either FreePlay or Regular depending on what the user chooses

setMode	
gameMode == 0	mode = Mode.Free_Play
gameMode !== 0	mode = Mode.Regular

Black Box Testing

Test 1
Input: gameMode == 0
Output: mode = Mode.Free_Play
Expected Output: mode = Mode.Free_Play
Result: Pass

Test 1
Input: gameMode !== 0
Output: mode = Mode.Regular
Expected Output: mode = Mode.Regular
Result: Pass

checkWin(int x, int y)

- Checks whether there is a win or a draw on the Grid.
- If there is neither a win or draw, the players continue playing

checkWin		
!(gameState == GameState.Ongoing)	gameState == GameState.Win	Display win message
	gameState !== GameState.Win	Display draw message
(gameState == GameState.Ongoing)	Players continue playing	

Black Box Testing

Test 1
Input: !(gameState == GameState.Ongoing) && gameState == GameState.Win
Output: Display win message
Expected Output: Display win message
Result: Pass

Test 1
Input: !(gameState == GameState.Ongoing) && gameState !== GameState.Win
Output: Display draw message
Expected Output: Display draw message
Result: Pass

Test 1
Input: (gameState == GameState.Ongoing)
Output: Players continue playing
Expected Output: players continue playing
Result: Pass

highlightSpace(int x, int y)

- Only works in free play mode.
- Checks the position of the incorrect piece.
- Highlights the pieces that are not valid for a proper ending. The highlight is a yellow shadow behind the incorrect piece.

Black Box Testing

Test 1
Input: Three red pieces connected at the bottom of the board.
Output: The piece needed to create a connect four is highlighted.
Expected Output: The piece needed to create a connect four is highlighted.
Result: Pass

Test 2
Input: A blue piece at position (0, 1)
Output: Highlights the blue piece.
Expected Output: Highlights the blue piece.
Result: Pass

Test 3
Input: Four connected red pieces are placed at the bottom of the screen followed by a blue with three connected blue pieces on top followed by a red piece.
Output: Highlights the four connected red pieces.
Expected Output: Highlights the four connected red pieces.
Result: Pass

Test 4
Input: A red piece is placed at position (0, 2) and a blue piece at position (0, 3).
Output: Highlights both the red and the blue pieces.
Expected Output: Highlights both the red and the blue pieces.
Result: Pass

setInvisible(GameObject button)

- Only valid in Free_Play mode.
- Makes the red piece or blue piece at the top of the screen invisible, when a player is putting the pieces of the same colour on the board.

Black Box Testing

Test 1
Input: Red pieces are being placed on the board.
Output: Red button on the top left corner is invisible.
Expected Output: Red button on the top left corner is invisible.
Result: Pass

Test 2
Input: Blue pieces are being placed on the board.
Output: Blue button on the top right corner is invisible.
Expected Output: Blue button on the top right corner is invisible.
Result: Pass

setVisible(GameObject button)

- Only valid in Free_Play mode.
- Make the red piece or blue piece at the top of the screen visible, when a player is not putting the pieces of that colour of the board.

Black Box Testing

Test 1
Input: Red pieces are being placed on the board.
Output: Blue button on the top left corner is visible.
Expected Output: Blue button on the top left corner is visible.
Result: Pass

Test 2
Input: Blue pieces are being placed on the board.
Output: Red button on the top right corner is visible.
Expected Output: Red button on the top right corner is visible.
Result: Pass

void drawImage(int xpos, int ypos, GameObject sprite)

- Only valid in Free_Play mode.
- Puts a piece on the board depending on where the user has clicked on the board.

Black Box Testing

Test 1
Input: Red piece at position (0, 0)
Output: A red piece appears at position (0, 0).
Expected Output: A red piece appears at position (0, 0).
Result: Pass

Test 2
Input: Blue piece at position (0, 6).
Output: A blue piece appears at position (0, 6).
Expected Output: A blue piece appears at position (0, 6).
Result: Pass

Test 3
Input: No piece are on the board.
Output: No piece on the board.
Expected Output: No piece on the board.
Result: Pass

Test 4
Input: A red piece is placed at position (0, 2) and a blue piece at position (0, 3). A highlight appears on both the red piece and the blue piece. The user clicks on the board.
Output: Both the red and the blue pieces are not highlighted.
Expected Output: Both the red and the blue pieces are not highlighted..
Result: Pass

Test 5
Input: Four connected red pieces are placed at the bottom of the screen followed by a blue with three connected blue pieces on top followed by a red piece. The four connected red pieces are highlighted. The user clicks on the board.
Output: The four connected red pieces are not highlighted.
Expected Output: The four connected red pieces are not highlighted.
Result: Pass

removeHighlightPieces()

- Only valid in Free_Play mode.
- Removes the highlight behind the incorrect pieces when the player clicks on the board.

Black Box Testing

Test 1
Input: Three red pieces connected at the bottom of the board and the space beside it is highlighted. The user clicks on the board.
Output: The piece needed to create a connect four is not highlighted.
Expected Output: The piece needed to create a connect four is not highlighted.
Result: Pass

Test 2
Input: A blue piece at position (0, 1) is highlighted and the user clicks on the board.
Output: The blue piece is not highlighted.
Expected Output: The blue piece is not highlighted.
Result: Pass

void setMessage(string message)

- Prints out the message to the user

Black Box Testing

Test 1
Input: string “Hello World”
Output: Screen prints out “Hello World”
Expected Output: Screen prints out “Hello World”
Result: Pass

void save(string data)

- Writes data to the text file “Saved_Game.txt”

Black Box Testing

Test 1

Input:

Output:

Empty,Empty,Empty,Empty,Empty,Empty,Blue
Empty,Empty,Empty,Empty,Empty,Empty,Red
Empty,Empty,Empty,Empty,Empty,Empty,Blue
Empty,Empty,Empty,Empty,Empty,Empty,Red
Empty,Empty,Empty,Empty,Empty,Empty,Blue
Empty,Empty,Empty,Empty,Empty,Empty,Red
0
1

Expected Output:

Empty,Empty,Empty,Empty,Empty,Empty,Blue
Empty,Empty,Empty,Empty,Empty,Empty,Red
Empty,Empty,Empty,Empty,Empty,Empty,Blue
Empty,Empty,Empty,Empty,Empty,Empty,Red
Empty,Empty,Empty,Empty,Empty,Empty,Blue
Empty,Empty,Empty,Empty,Empty,Empty,Red
0
1

Result: Pass

White Box Testing

Test 1
Input: ("testing 123")
Output: "testing 123" in file Saved_Game.txt
Expected Output: "testing 123" in file Saved_Game.txt
Result: Pass

string load()

- Reads data from the text file “Saved_Game.txt”

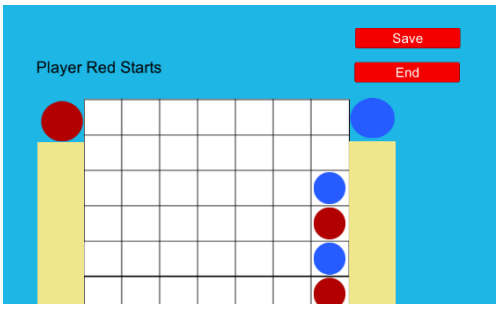
Black Box Testing

Test 1

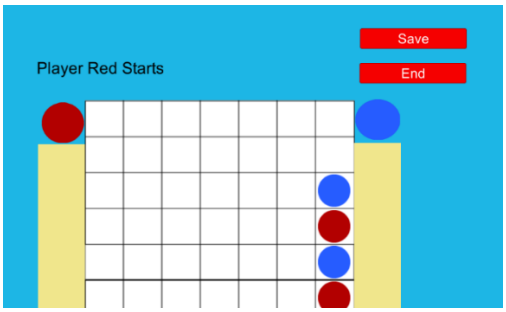
Input:

Empty,Empty,Empty,Empty,Empty,Empty,Red
Empty,Empty,Empty,Empty,Empty,Empty,Blue
Empty,Empty,Empty,Empty,Empty,Empty,Red
Empty,Empty,Empty,Empty,Empty,Empty,Blue
Empty,Empty,Empty,Empty,Empty,Empty,Empty
Empty,Empty,Empty,Empty,Empty,Empty,Empty
1
1

Output:



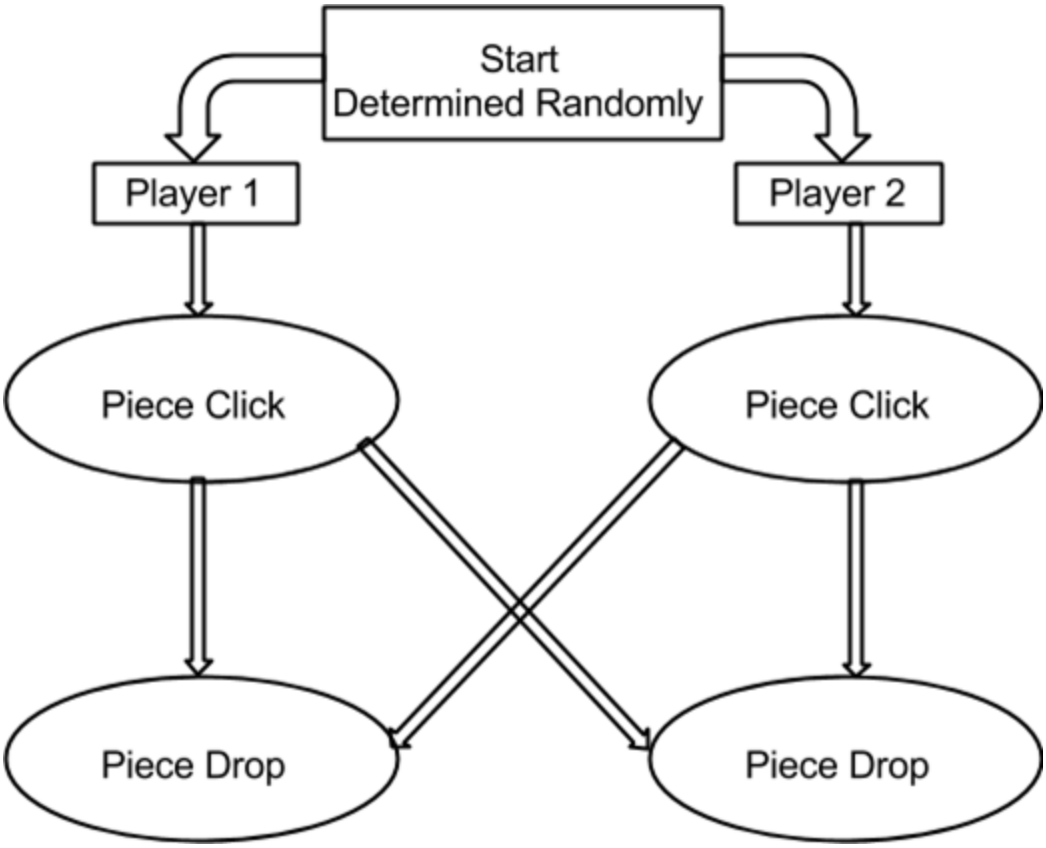
Expected Output:



Result: Pass

White Box Testing

Test 1
Input: "testing 123" in Saved_Game.txt
Output: "testing 123"
Expected Output: "testing 123"
Result: Pass


Start
Determined Randomly
Player 1
Player 2
Piece Click
Piece Click
Piece Drop
Piece Drop

Player

Private Enumerations

PlayerType{Human,AI}

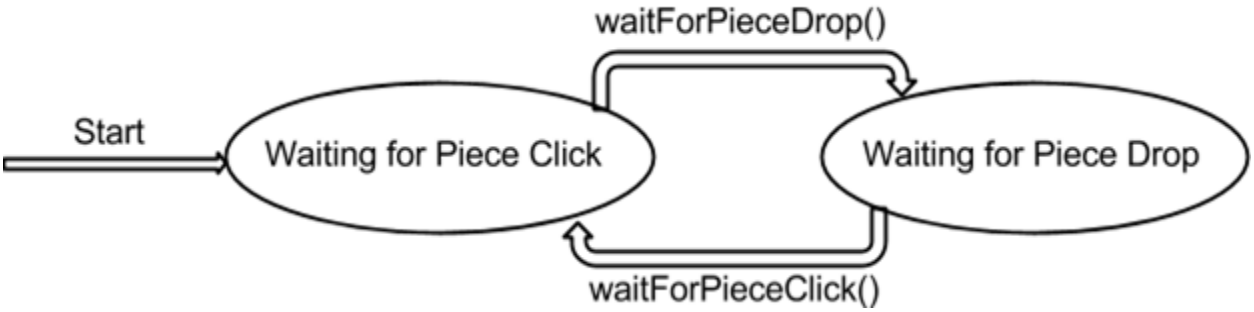
Turn{WaitingForPieceClick,WatingForGridClick}

Private Fields

PlayerType type
- Holds if the player is a Human or an AI.

Turn state
- There are two states; waiting to pick up a piece and waiting to drop one.

Space pieceColour
- The colour of the players piece.



Private Methods

Int[] priorityMove(grid g)
- Series of checks for AI to determine where to place next piece

priorityMove				
Case 1	connectThreeUs > 0	GameLogic.connectThree (g, this.pieceColour)		
	connectThreeUs =< 0	GameLogic.findSpot (g,cannotPlace,x_Pos)		
Case 2	connectAmountOpponent >= 1	GameLogic.connectThree (g, GameLogic.inverseSpace (this.pieceColour)		
	connectAmountOpponent < 1	GameLogic.findSpot (g,cannotPlace,x_Pos)		
Case 3	turn < startMoves.Length / 2	startMoves [turn, 0] <g.getWidth()/2	g.isValidSpot (startMoves [turn, 0], startMoves [turn, 1]) = True	return pair
			g.isValidSpot (startMoves [turn, 0], startMoves [turn, 1]) = False	GameLogic.moveLeft (g, new ArrayList (), --startMoves [turn, 0])
		startMoves [turn, 0] >=.getWidth()/2	g.isValidSpot (startMoves [turn, 0], startMoves [turn, 1]) = true	return pair
			g.isValidSpot (startMoves [turn, 0], startMoves [turn, 1]) = false	GameLogic.moveRight (g, new ArrayList (), ++startMoves [turn, 0])
	turn >= startMoves.Length / 2	No action taken		
Case 4	checkL != null	return checkL		
	checkL = null	No action taken		
Case 5	GameLogic.findSpot (g,cannotPlace,x_Pos)			

Gravity

- A script that describes the behavior of player piece objects when they are instantiated on the screen.
- In Free Play mode:
    - piece position is translated downward from original position until it reaches the position indicated by the player's mouse click
- In Regular mode:
    - piece remains in original position

Private Fields

int GRID_OFFSET_X
- The grid on the x-axis becomes active 291 units to the right.

int GRID_OFFSET_Y
- The grid on the y-axis becomes active 0 units up.

int SPACE_WIDTH
- Each space in which a piece can be placed is 71 units wide.

int SPACE_HEIGHT
- Each space in which a piece can be placed is 66 units tall.

int CIRCLE_RADIUS
- The radius of the piece with offset is described by this field, radius of pieces are 35.

float speed
- The rate at which the piece moves due to gravity.

Vector2 target
- The position to which the piece should travel by gravity.

int gameMode
- The mode that the game is in.

Gravity	
Free_Play	Player piece does not move
Regular	Player piece moves downward and stops in space designated by mouse click
	Player cannot click grid

Internal Review/Evaluation of the Design

- **Good points**

    - The Grid
        - Combined the Colour (in Grid.cs) and Space (in Controller.cs) private Fields into Space field
        - Modularized the whole file to have less comparisons
        - Implementation for assignment 3. For example in Player.cs the set up checks to see if the player is an AI.

    - The Controller
        - Allowed the user to save and load single games using a text file which the implementation of is very
        - Flexible
        - Reliably saves one game

    - The GameLogic
        - Because first move can result in a win or lose, randomizing who starts ensures fairness.
        - AI is smart enough to beat player a few times,
        - Most moves of the AI are predicted and not randomized
        - AI is able to block when the opponent has 3 pieces in a row

    - The View
        - More appealing graphical user interface.
        - The error messages are clear and concise.

- **Needs Improvement**

    - The Controller
        - Allow the user to remove a placed piece in the Free_Play mode if the piece is invalid.
        - Add a Reset and/or Undo option for putting pieces down in each game mode.
        - Doesn't allow multiple save states

    - The View
        - Change the colouring of when pieces are highlighted.
        - Make the graphical user interface even more appealing by adding a new "Connect Four" logo at the title. Maybe a change of colours (or shading of colours) is in order.

    - The GameLogic
        - AI can be tricked easily when using patterns