

1810705 - Software Engineering

by Kristian Leigh Cole

FILE	12521_KRISTIAN_LEIGH_COLE_1810705_- _SOFTWARE_ENGINEERING_339836_726525503.DOCX (798.8K)		
TIME SUBMITTED	25-MAR-2021 02:45PM (UTC+0000)	WORD COUNT	5000
SUBMISSION ID	147940632	CHARACTER COUNT	25327

Contents

1.0 User Story Map	3
2.0 Application_Design	4
2.1 Application_Logo	4
2.2 Application_Wireframe.....	4
2.3 Application–Pick_A_Counter	4
2.4 Application–Choose_Name_&_Colour	5
2.5 Application–Choose_Column.....	5
2.5 Application–Drop_Counter	6
2.6 Application–Winning_Line	6
2.7 Application–Row_Of_Three_Destroyed	7
2.8 Application–Opponent_Counter_Captured.....	7
2.9 Application–Game_Won	8
3.0 Application_Evaluation	8
4.0 Class_Responsibility_Collaborator_(CRC)_Card_Design_Approach	9
4.1 Class_Responsibility_Collaborator_Cards_&_Evaluation	10
4.2 Class_Responsibility_Collaborator_Evaluation.....	11
5.0 Software_Testing_Evaluation	11
6.0 White_Box_Testing	12
6.1 Scenario_One	12
6.2 Scenario_Two.....	12
6.3 Scenario_Three	13
7.0 Black_Box_Testing	13
7.1 Scenario_One	13
7.2 Scenario_Two.....	14
7.3 Scenario Three	14
7.4 Black_Box_Tests.....	15
7.5 White Box Tests	17
8.0 Connect4_Functions	20
8.0.0 connect_ Variables.....	20
8.0.1 chooseCounter	20
8.0.2 changePlayerNameColour	20
8.0.3 changePlayerName	20
8.0.4 dropColumn	20
8.0.5 uploadImage	21

8.0.6 fourintarow	21
8.0.7 threeOrCapture.....	22
9.0 Connect4_Unit_Tests.....	23
9.0.1 chooseCounter_Test.....	23
9.0.2 changePlayerNameColour_Test.....	24
9.0.3 changePlayerName_Test	24
9.0.4 dropColumn_Test	24
9.0.5 uploadImage_Test.....	24
9.0.6 fourintarow_Test	24
9.0.7 threeOrCapture_Test.....	24
References	25

1.0 User Story Map

Primary Goal: Allow Users to drop counters into a 6x7 board, to play a custom game of connect 4

USER FLOW

Player one chooses counter X or O. Player two has counter type left over by default.

Present user an X or O sprite to choose from

As a user I want to choose my counter to drop into the game to line up on a grid

As a user I want to be able to turn the sound of the game on or off

As a user I want to play an opponent to win a game

As a player I want to choose my own image icon which can be uploaded or chosen from a list

Choose Column to drop counter into

Present each player the complete board.

As a user I want to see a grid where I can line up token in lines of four and three

As a user when it is my turn to play I want to be able to change to my player colour to know whose turn it is

Allow each player to choose a column at the top of the grid

As a user I want to choose which column to drop my counter into to line up counters

As a user I want to have a choice of six columns to drop counters into

Show each player the board after moving down the grid to its final place on the board

As a player I want to see the counter move down the board when inputted

1

Check where counter has landed in the grid

If the counter dropped by either player forms a horizontal, vertical or vertical line with a length of 4 the player wins the game

As a user I want to see the board flash red when I win the game to know I have won the game

As a user I want to be asked if I want to play another game once the game has been won

If the counter dropped by either player forms a horizontal, vertical or vertical line with a length of 3 the player wins the game.

As a user I want the board to flash green when I destroy 3 counters of another player

If player has not won on turn, next turn continues

any gaps left by previous player are filled by counters above the gaps

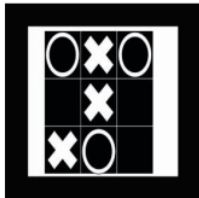
As a user I want all gaps to be filled when a row of three has been won during game

Any gaps filled by counters above that form a line of three are destroyed vertically do not destroy the game

2.0 Application_Design

The benefits of the user stories highlighting the initial requirements of the users. This is a collection of prioritised requirements that lays out what the application should do to function. The Agile method of development performs best leading on from this design method as the initial release will be Minimum Viable Product with future iterations.

2.1 Application_Logo



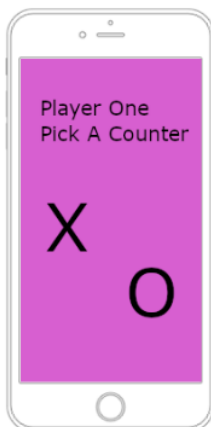
The application logo shows the concept of the applications purpose, which is Noughts and Crosses. The user stories describe a requirement for an application that can be used on a computational device.

2.2 Application_Wireframe



The application wireframe of a mobile device shows the web app link on the desktop. As this is a web application and will more than likely be displayed on a mobile device rather than a desktop computer this documents accompanying wireframes will reflect the choice. The benefits of wireframe modelling show the users what the application will look like on chosen device. This also give the developers a chance to look at the UI requirements and get a general feel of what works for the user and what does not. There are standards to adhere to in UI design such as accessibility, Input Controls, Navigational Components, Informational Components and Containers [2].

2.3 Application-Pick_A_Counter



The user map requests:

- Each player is shown an X or a O to choose from.
- The user task requires a sprite is chosen.
- Once the counter type is chosen both players are aware of which counter they are looking out for in the game.

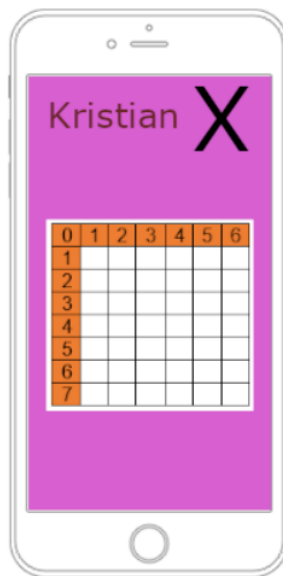
2.4 Application–Choose_Name_&_Colour



The user stories request the ability to:

- Rename the Player 1
- change the colour of the name
- upload an image to replace the player name.
- The ability to turn the sound on and off in the game.

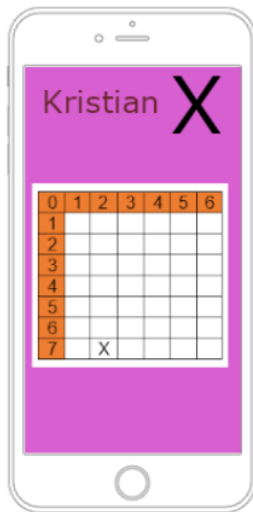
2.5 Application–Choose_Column



The user stories request

- The ability to click on a column to drop the chosen counter into.
- To see the playing board on screen.
- The grid is set at six columns by seven rows.
- The colour of the board will change to the colour chosen at the start of the game or by the default colour allocated to current playing username when it is their turn.

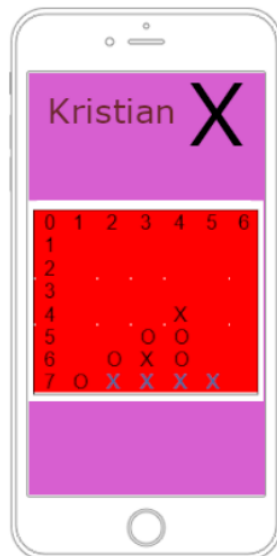
2.5 Application–Drop_Counter



The user requires that:

- The counter is shown moving down the board.
- The counter resting at the lowest possible point on the board where space is available.

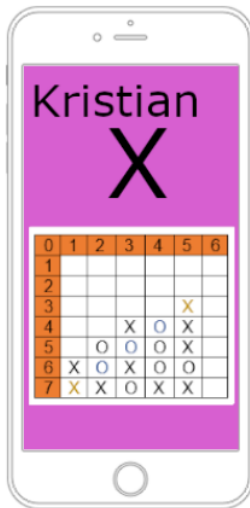
2.6 Application–Winning_Line



The rules of the game are that:

- If a player gets four in a row the player wins the game.
- The grid flashes red when the player has won to inform them of the win.

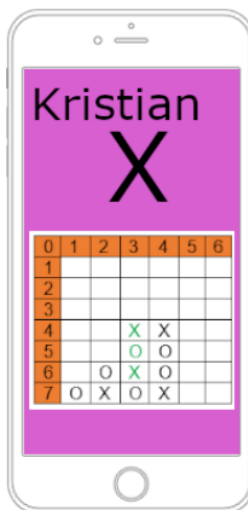
2.7 Application–Row_Of_Three_Destroyed



The rules of the game are that:

- If three counters are in a row and bookended by an opponents counters the three counters are destroyed.
- The counters above then fall into the gaps provide by the destroyed counters.

2.8 Application–Opponent_Counter_Captured



The rules of the game are that:

If a players single counter is bookended by another players counter then it is converted to the original players counter, it is captured.

2.9 Application–Game_Won



The user requires that:

- When the game is won by either player, the winner is asked if they want to play the game again.

3.0 Application_Evaluation

The design of the Connect four application initially already had a functioning vision as the game already existed, this design would build on that vision in customization. The initial user stories were very loose in their requirements but was made easier when put into some priority. For example, each of the user stories in the user map (see 1.0 User Story Map) were first put aside into basic groupings with overarching description that gave an overview of what the user required. A main flow is set out first, which is a very brief description of what needs to happen:

1. Player One Chooses a counter either X or O. Player Two had counter left over by default.

The above main flow item is then broken down into user tasks. The main flow item is expanded upon in the user tasks:

2. Present User an X or O sprite to choose from
3. Add Sprite counter type next to player name

At this point the original user stories are lined up in order of priority beneath each of the user tasks:

4. **Present User an X or O sprite to choose from**

4.1 – As a user I want to choose an X or O counter to drop into the game and line up on the grid.

4.2 – As a user I want to be able to turn the sound on or off.



4.3 – As a user I want to play an opponent to win a game.

5. Add Sprite counter type next to player name.

5.1 – As a user I want to see which counter token I have chosen to know which line to look out for.

5.2 – As a player I want to choose my own player name to differentiate between players.

5.3 – As a player I want to choose the colour of my player name.

5.4 – As a player I want to choose my own image icon which can be uploaded or chosen from a list.

The strengths of this method of designing an application:

- The developers get an idea of what they are going to create.
- Functionality valuable to users.
- Allows for future planning and solutions for problems can be mapped out earlier on.
- Initial user interface problems are weeded out as a vision is seen and analysed by multiple parties.
- Requirements are of manageable size and scope.
- The entire application lifecycle can be seen from start to finish.
- Developers can work out what is to be expected of them and of the user.
- The users are permitted to overthink their initial requirements and build on them.
- A Minimum viable product can be released with scope for iterations.
- Prioritized user stories can be placed in future iterations of the applications release.
- Testing plans can be written at earlier stages.

The weaknesses of this method of designing an application:

- **Vague requirements.**
- Smaller projects can be mapped out where larger scalable projects will find difficulty. For example, a project where there are a higher level of tasks for the user and perhaps the user does not have the bigger picture in mind.)
- The danger of focusing on less valuable requirements.
- By the nature of the user story being less flexible than a solid requirement it can be open to scope creep.
- User story misinterpretation.

4.0 Class_Responsibility_Collaborator _(CRC)_Card_Design_Approach

Class Name	
Responsibilities	Collaborators

The Class Responsibility Collaborator card approach is used to map out the theoretical structure of the classes that will be implemented in the application. At its most basic it can be hand written on card, headed by a class name (Class title), responsibilities (class members) and collaborators (Classes the class will rely on to function) [3]. The cards represent what the final classes will be responsible

for when the application is functional. This method can also be used to map our database tables if the application were to stretch that far.

4.1 Class_Responsibility_Collaborator_Cards_&_Evaluation

Player	
Player_Name Player_Colour Sprite Counter Email Won/Lost Computer	Game

The Player class contains the responsibilities for each Player within the game:

1. Player_Name – [string] the user stories referred to any player being able to change their name.
2. Player_Colour – [string] The user stories referred to any player being able to change the colour of their name. A Hex colour code will be entered.
3. Sprite – [int] the user stories had request of being able to upload a sprite image in place of a player name. Also restricting the importing of images over a certain size.
4. Counter – [char] The counter for the player, either an X or a O.
5. Email – [string] Perhaps not so important in MVP but as part of as future iteration where a user could email their results to themselves.
6. Won/Lost – [bool] The outcome of the current game
7. Computer – [bool] if there are no players available the second player will be automated.

The single collaborator in the class is the Game class. Each Player generates a single Player class with all its attributes, on entry to the application and instantiates a Game object by which to join other players across the world.

Game	
Any_Waiting_Players Player_1_Score Player_2_Score Winner Column_Members 4_In_A_Row 3_In_A_Row Captured Date	

The Game class is an object created from the Player class and is used to mediate games between players:

1. Any_Waiting_Players – [array] holds an array of five players who are waiting to play. The waiting time could be seconds based on levels of world wide players. If zero then the player will face an automated opponent.
2. Player_1/Player_2_Score – [int] holds scores of both players.
3. Winner – [string] holds winning players name/sprite to display back to the winning player.

4. Column_Members - [array] a multidimensional array of the 6 columns by 7 rows that hold either null values, X's or O's. The application will be updating this array per user turn and reshuffling based on gaps from destroyed counters.
5. 4_In_A_Row – [bool] holds a true or false as to whether 4 in a row has been achieved. This is used to highlight the winner of the game and end current game.
6. 3_In_A_Row – [bool] holds a true or false as to whether 3 in a row has been achieved. This is used to highlight whether 3 in a row has occurred and to trigger the function which will reshuffled the board and continue the game.
7. Captured – [bool] hold a true or false as to whether counters have been captured. This will trigger a function to reshuffle the board based on captured counters.
8. Date - [date] holds the current date.

4.2 Class_Responsibility_Collaborator_Evaluation

The CRC cards produced for the application contain sufficient detail to enable the code to be implement and tested. A player class object has the responsibility of holding the player name, counter type and link to the Game class. The Game class has the responsibility of holding two players within a single game.

The CRC cards reflect the acceptance criteria of the user stories:

1. A player can choose a counter, the CRC class has a variable entitled Counter.
2. A player wants to change the colour of the player name
3. A player wants to upload an image, the CRC class has a variable which can hold integer binary data.
4. A player want to see who has won, the CRC card has Winner variable state.

In terms of the actual game:

1. A player wants to know who the winner of the game is, CRC class member Winner.
2. A player wants to know where X and O has been dropped, the CRC member array Column Members can hold the player counter information.
3. A player wants to know when 4 in a row has occurred, CRCA card hold 4 In A Row state.
4. A player wants to know when 3 in a row has occurred, CRCA card hold 3 In A Row state.
5. A player wants to know when an opponent's counter has been captured, CRC holds the Captured state.

5.0 Software_Testing_Evaluation

There are two types of testing involved in software design, Black and White box testing. Black Box Testing is a software testing method in which the internal structure of the code, the class and function design is unknown to the testing body. This type of testing is carried out by a software tester and treats the software how a general user would.

White Box testing is the opposite whereby all the internal code and class design is known so can be targeted within the scope of testing as a software developer.

The importance of testing an application determines whether the application is fully functional and if there are any issues that require rework they can be caught before release and remedied.

6.0 White_Box_Testing

White box testing or clear box testing is where the software developer runs path, loop and condition testing on the underlying code within software. The software developer should have full knowledge of the internal structure of the code and design and has the ability to test every aspect of the software at a low level. This type of testing is suitable for algorithm testing and is the most time-consuming part of the development. At this stage of testing the logic of the software is targeted and how the inner workings of the application function under arbitrary use. This type of testing relies on a design document rather than the user stories or user story maps [4].

6.1 Scenario_One

A non-functional test within a piece of software requires an input and an output. The following code highlights the functions use in Javascript without Typescript as an example:

```
Function add(input)
{
    //Assign input to local variable
    var in = parseInt(input);
    //Set output to an integer type
    var out = 0;
    //Add input to a number
    out = in + 14;
    return out;
}
```

This function works perfectly if a number is passed as the argument, the expectation is a number would be applied every time, this would fail to run correctly if a string object had been passed to the function. This happens when textboxes are the target inputs where free text is the input. This type of test is a white box test because underlying code functionality is being targeted as expected outputs with reasonable inputs.

6.2 Scenario_Two

Imagine a function that calls itself recursively until an expected result is reached such as a counter level in a for loop.

```
Var counter = 0;
Function numTest()
{
```

```

While(counter < 1000)
{
    If(counter < 1000)
    {
        Counter--;
    }
    Else
    {
        numTest();
    }
}
}

```

The example code will never reach the required counter number as there is an error in the counter increment. This code would run infinitely and the software would hang. A white box test of this function would pick this issue up early because the expected result is a counter increment and release from the while loop.

6.3 Scenario_Three

The testing of each statement, object, and function on an individual basis. This would require unit testing where every object used is tested for type relevance such as inputting a string into an integer variable. Every statement is tested for input and output and every function is tested for expected results and non-expected results.

7.0 Black_Box_Testing

Black box testing or outer software testing is carried out by software testers with little to no knowledge of the implementation of the code. This type of testing is purely functional as in can the software perform the tasks it has been designed to carry out. This testing usually relies on a requirements document or user stories and does not require the tester has any awareness of the underlying code classes or functions. This type of testing looks at the behaviour of the software and is high level analysis and takes much less time to carry out over White Box Testing.

7.1 Scenario_One

Functional testing in black box testing is a test to see if the function fits the requirement.

An example of this is a login screen:

- The user enters a username
- The user enters a password.
- It is the wrong the password but the submission of the details allows the user access to the system.

Obviously, the this is not how a login screen functions and various black box testing would reveal that the correct functionality is that a login screen does not allow the user access to the system without the correct credentials.

7.2 Scenario_Two

Boundary testing of software is the testing of the range of values input into a piece of software. Imagine a system where you are required to enter a quantity of a free item for free delivery:

- The input box has no limits as to how many numbers of items the user requests.
- Even though the user cannot write text into the box it does accept numbers.
- Most systems would limit the input either on the input box or at code level.
- The user writes a million into the input and is expecting to receive a million free items.

Obviously, this amount cannot be met and is a limitation of the system that doesn't attach itself to stock data or is limited to one item per user. The white box test on this boundary would be to input multiple ranges of numbers expecting that only a low quantity is expected.

7.3 Scenario Three

Decision table testing is a white box testing method where there are chains of decisions triggered from an initial input or UI item interaction.

Imagine the user is filling in an email input box which also requires two tick boxes to be populated that covers GDPR usage of user data and an option to receive future emails from third party vendors.

Condition	Email entered	GDPR Ticked	Third Party Ticked
Enter Email	yes	no	yes
Choose GDPR option set	yes	yes	yes
Tick Third Party Email	yes	no	no

If the program allows the email to be entered, the GDPR is left blank and the exposure to third party email is ticked yes the email may be shared with third party without GDPR type being chosen. The white box testing should cover the chain of events which leads to the GDPR option set being unchecked. More complex chains will have larger tables that cover the intricacies of a longer chain. The allows paths to be mapped out theoretically then applied in practice to functions in practice.

7.4 Black_Box_Tests

Test Title	Preconditions (Setup)	Running The Test	Test Inputs	Expected Outputs	Description Of Test
Load Game Mobile	Web browser installed on operating system	Open web browser and navigate to game page/Or click on Icon	Link in browser	Game starts in browser	Clicking on icon opened game.
Load Game Computer	"	"	"	"	"
Load Game Tablet	"	"	"	"	"
Load Game Laptop	"	"	"	"	"
Change Player Name	Text Box on screen.	Click on textbox, changed player name.	Entered string into textbox	Name displayed in the textbox	Entering text into textbox, name to be displayed on screen
Upload Image	Upload Button.	Click upload button, choose image file to upload. JPEG, PNG, JPG or BMP file to upload	Chose image file from disk folder.	Image file is uploaded, player name is removed and replaced by image.	File size limits prevent large files being uploaded. Sprite size of 15pixels by 15 pixels accepted.
Upload random file	Button present to press.	Click on upload button and choose image file to upload. TXT, CSV, DOC, XML file to upload	"	Random file is attempted upload	Non image files are not accepted.
Change Name Colour	Colour palette	Random colour chosen from the palette on screen.	Colour is chosen from palette	Player name changes colour	The colour chosen made the player name colour change
Choose Counter X	X Counter is on screen	X/O counters are present on screen and X is chosen	X is chosen from the screen	X is shown next to players name	The X counter is chosen and displayed next to the players name

Choose Counter O	O Counter is on screen to press	O and X counters are present on screen and O is chosen	O is chosen from the screen	O is shown next to players name	The O counter is chosen and displayed next to the players name
Drop counter O	Grid is shown on screen	O counter is moved to a column on the grid to drop.	X is dropped at any column	X counter moved down grid and rests at bottom of grid or on top of previously dropped counters.	The X counter is dropped at the top of the screen and moves down the grid.
Three In A Row	Three counters are bookended by opponent's counter	After dropping counter at the tail end of three opponent's counters.	counter is on either side of three opponent's counters	O counters are destroyed	The O counters are destroyed when X bookends three counters
Four In A Row	Four counters are in a row	A counter dropped in line with three same counters	counter dropped into grid	counter lines up with 3 other opponent's counters	counter lines up with 3 same counters and game is won
Capture Counter	One counter is bookended by an opponent's counter	Counter dropped in line a single opponent's counter	counter dropped into grid on opponent's counter	counter captured	counters capture opponent's counter by bookending one opponent's counter
Game Starts Over	Player wins or loses game	Game offers restart	Tick button on screen	Button is pressed and game restarts	Player chooses to start turn again.

7.5 White Box Tests

Test Title	Preconditions (Setup)	Running The Test	Test Inputs	Expected Outputs	Description Of Test
Player Name Change Function special characters	Player name string object length capped to 10 characters	changePlayerName() function triggered on change of playername string object	A word inputted by the user	Playername object populated with inputted text	The player inputs a line of text, alphanumerical permitted, the playerone one object is populated.
Player Name Change Function string	"	"	"	"	"
Player Name Change Function integers	"	"	"	"	"
Upload Image Function – image file	Image integer object	uploadImage() function triggered on button click	Image file type chosen from file store	File type is either jpeg, jpg, bmp, png	Image file is uploaded into game and stored in byte array object
Upload Image Function – text file	Image integer object	uploadImage() function triggered on button click	Text file type chosen from file store	File type is not either jpeg, jpg, bmp, png	file is not uploaded into game
Upload Image Function – random file	Image integer object	uploadImage() function triggered on button click	Random non text or image file chosen	File type is not either jpeg, jpg, bmp, png	"
Change Name Colour Function – random colour	Playernamecolour string object	changePlayerColour() function triggered on pallet click	Hex colour integer passed to object	Hex colour object applied	Player name colour changes to hex colour chosen.
Choose Counter Function X	Counter char object	chooseCounter() function triggered on counter X sprite click	Character X	Character is applied to counter char object	Counter is type X chosen by the user.

Choose Counter Function O	Counter char object	chooseCounter() function triggered on counter O sprite click	Character O	Character is applied to counter char object	Counter is type O chosen by the user.
Column Choice function – Column 1	Grid array object	dropColumn() function is triggered on grid column 1 choice	Number 1 is passed to the function for column 1	Column 1 is chosen	Choose column number 1
Column Choice function – Column 2	"	dropColumn() function is triggered on grid column 2 choice	Number 2 is passed to the function for column 2	Column 2 is chosen	Choose column number 2
Column Choice function – Column 3	"	dropColumn() function is triggered on grid column 3 choice	Number 3 is passed to function for column 3	Column 3 is chosen	Choose column number 3
Column Choice function – Column 4	"	dropColumn() function is triggered on grid column 4 choice	Number 4 is passed to function for column 4	Column 4 is chosen	Choose column number 4
Column Choice function – Column 5	"	dropColumn() function is triggered on grid column 5 choice	Number 5 is passed to the function for column 5	Column 5 is chosen	Choose column number 5
Column Choice function – Column 6	"	dropColumn() function is triggered on grid column 6 choice	Number 6 is passed to the function for column 6	Column 6 is chosen	Choose column number 6
Three In A Row X/O	Iterate grid array object	threeOrCapture function triggered	Counter is passed as argument	Three counters bookended by an opponents counter	Destroyed counter combination

Four In A Row XO	"		fourcountersinaRow function triggered	Counter passed as argument	Four counters in a row	Winning combination
Capture X/O	"		captureCounter triggered	Counter passed as argument	counter on grid bookended by opponent counter	Capture combination

8.0 Connect4_Functions

8.0.0 connect_ Variables

```
private string playerName = "";
private byte[] imgSprite = null;
private int playerNameColour = 0;
private char counter = '-';
private bool threeinaRow = false;
private bool fourinaRow = false;
char[,] grid = new char[,]
{
    { '-', '-', '-', '-', '-', '-', '-' },
    { '-', '-', '-', '-', '-', '-', '-' },
    { '-', '-', '-', '-', '-', '-', '-' },
    { '-', '-', '-', '-', '-', '-', '-' },
    { '-', '-', '-', '-', '-', '-', '-' },
    { '-', '-', '-', '-', '-', '-', '-' },
    { '-', '-', '-', '-', '-', '-', '-' },
    { '-', '-', '-', '-', '-', '-', '-' }
};
```

8.0.1 chooseCounter

```
public char chooseCounter(char counterType)
{
    counter = counterType;
    return counter;
}
```

8.0.2 changePlayerNameColour

```
public int changePlayerNameColour(string hexCol)
{
    int hex = Int32.Parse(hexCol,
System.Globalization.NumberStyles.HexNumber);
    playerNameColour = hex;
    return playerNameColour;
}
```

8.0.3 changePlayerName

```
public string changePlayerName(string newName)
{
    playerName = newName;
    return playerName;
}
```

8.0.4 dropColumn

```
public bool dropColumn(int colChoice)
{
    int row = 0;
    int i = 0;
    bool success = false;
    for (i = 0; i < 7; i++)
    {
        if (grid[i, colChoice] == '-')
        {
            row = i;

```

```

        }
        else
        {
            success = true;
            break;
        }
    }
    grid[i, row] = counter;
    return success;
}

```

8.0.5 uploadImage

```

public bool uploadImage(string path)
{
    bool success = false;
    if (File.Exists(path))
    {
        if (Path.GetExtension(path).Contains(".jpeg") ||
            Path.GetExtension(path).Contains(".jpg") ||
            Path.GetExtension(path).Contains(".png") ||
            Path.GetExtension(path).Contains(".bmp"))
        {
            FileStream fs = new FileStream(path,
                FileMode.Open,
                FileAccess.Read);
            BinaryReader br = new BinaryReader(fs);
            long numBytes = new FileInfo(path).Length;
            imgSprite = br.ReadBytes((int)numBytes);
            success = true;
        }
    }
    else
    {
        success = false;
    }
    return success;
}

```

8.0.6 fourintarow

```

public char fourcountersinaRow(char currCounter)
{
    int curNumCount = 0;
    char lastCounter = ' ';

    //Horizontal Check
    for (int r = 0; r < 7; r++)
    {
        for (int c = 0; c <= 6; c++)
        {
            if (grid[r, c] == currCounter && lastCounter ==
currCounter)
            {
                curNumCount++;
                if (curNumCount == 4) { break; }
            }
            else
            {
                curNumCount = 0;
            }
        }
        if (curNumCount == 4) { fourinaRow = true; break; }
    }
}

```

```

    }

    //Reset check
    lastCounter = ' ';
    //Vertical Check
    for (int c = 0; c < 6; c++)
    {
        for (int r = 0; r <= 7; r++)
        {
            if (grid[r, c] == currCounter && lastCounter ==
currCounter)
            {
                curNumCount++;
                if (curNumCount == 4) { break; }
            }
            else
            {
                curNumCount = 0;
            }
        }
        if (curNumCount == 4) { fourinaRow = true; break; }
    }

    //Diagonal Check
    int rowCount = 1;
    for (int c = 0; c < 6; c++)
    {
        if (grid[rowCount, c] == currCounter && lastCounter ==
currCounter)
        {
            curNumCount++;
            if (curNumCount == 4) { break; }
        }
        rowCount++;
        if (curNumCount == 4) { fourinaRow = true; break; }
    }

    return currCounter;
}

```

8.0.7 threeOrCapture

```

public bool threeOrCapture(char currCounter, int choice)
{
    int curNumCount = 0;
    char lastCounter = ' ';
    char comparisonCounter = ' ';
    bool success = false;
    switch(currCounter)
    {
        case 'X': comparisonCounter = 'O'; break;
        case 'O': comparisonCounter = 'X'; break;
        default: success = false;
    }

    //Horizontal Check
    for (int r = 0; r < 7; r++)
    {
        for (int c = 0; c <= 6; c++)
        {
            if (grid[r, c] == currCounter &&

```

```

        lastCounter == currCounter &&
        grid[r,c-1] == comparisonCounter &&
        grid[r, c+choice] == comparisonCounter
    )
    {
        curNumCount++;
        if (curNumCount == choice) { break; }
    }
    else
    {
        curNumCount = 0;
    }
}
if (curNumCount == choice) { inaRow = true; break; }
}

//Reset check
lastCounter = ' ';
//Vertical Check
for (int c = 0; c < 6; c++)
{
    for (int r = 0; r <= 7; r++)
    {
        if (grid[r, c] == currCounter &&
            lastCounter == currCounter &&
            grid[r - 1, c] == comparisonCounter &&
            grid[r + choice, c] == comparisonCounter)
        {
            curNumCount++;
            if (curNumCount == choice) { break; }
        }
        else
        {
            curNumCount = 0;
        }
    }
    if (curNumCount == choice) { inaRow = true; break; }
}

//Diagonal Check
int rowCount = 1;
for (int c = 0; c < 6; c++)
{
    if (grid[rowCount, c] == currCounter &&
        lastCounter == currCounter &&
        grid[rowCount, c - 1] == comparisonCounter &&
        grid[rowCount, c + choice] == comparisonCounter)
    {
        curNumCount++;
        if (curNumCount == choice) { break; }
    }
    rowCount++;
    if (curNumCount == choice) { inaRow = true; break; }
}
return success;
}

```

9.0 Connect4_Unit_Tests

9.0.1 chooseCounter_Test

[TestMethod]

```

public void chooseCounterXTest()
{
    char xcounte = con.chooseCounter('Y');
    Assert.AreEqual(xcounte, 'X');
}
public void chooseCounterOTest()
{
    char xcounte = con.chooseCounter('Y');
    Assert.AreEqual(xcounte, 'O');
}

```

9.0.2 changePlayerNameColour_Test

```

[TestMethod]
public void changePlayerNameColourTest()
{
    int col = con.changePlayerNameColour("#808080");
    Assert.AreEqual(col, "#808080");
}

```

9.0.3 changePlayerName_Test

```

[TestMethod]
public string changePlayerNameTest()
{
    string str = con.changePlayerName("Kristian@1");
    Assert.Equals(str, "Kristian@1");
}

```

9.0.4 dropColumn_Test

```

[TestMethod]
public bool dropColumnTest()
{
    bool dropNum = con.dropColumn(39);
    Assert.IsTrue(dropNum);
}

```

9.0.5 uploadImage_Test

```

public bool uploadImageTest()
{
    bool upl = con.uploadImage("C:\\IMG\\text.txt");
    Assert.IsFalse(upl);
}

```

9.0.6 fourintarow_Test

```

[TestMethod]
public void fourcountersinaRowTest()
{
    char count = con.fourcountersinaRow('P');
    Assert.AreNotEqual(count, 'X');
}

```

9.0.7 threeOrCapture_Test

```

[TestMethod]
public bool threeOrCaptureTest()
{
    bool chk = con.threeOrCapture('X', 3);
    Assert.AreEqual(chk, true);
}

```

References

- [1] M. Dacey, "BSC User Stories," Trinity College, Swansea, 2021.
- [2] Unknown, "User Interface Design Basics," U.S. General Services Administration Technology Transformation Services, 16 03 2021. [Online]. Available: <https://www.usability.gov/what-and-why/user-interface-design.html>. [Accessed 16 03 2021].
- [3] S. W. Ambler, "The Object Primer 3rd Edition: Agile Model Driven Development with UML 2," Cambridge University Press, 2004, p. Chapter 8 .
- [4] M. Dacey, "Software Testing," Swansea, 2021.

FINAL GRADE

74/100

GENERAL COMMENTS

Instructor

Eval design (10): 10

User Stories (20): 18

CRC (20): 14

SW Testing (30): 17

Coding (20): 15

Feedback:

=====

Liked the demo app presentation leading up to the user stories.

Nice evaluation of both the user stories and CRCs. The CRCs you provided are a little more complicated than are actually required. For example, a responsibility could just read "Manage player details". Typically, to work out the attributes and methods we would revert to UML class diagrams.

The software testing discussion was a little confused, essentially with white box testing we have access to the code and can design tests with full knowledge of how the code works. With black box tests we treat the system/code as a whole, so we test that the given inputs produce the correct outputs for given scenarios.

Good effort with the code, not complete and the code could have been improved in parts.



Comment 1

....so that



Comment 2

As a I want....so that



Comment 3

You need expand upon this



Comment 4

I think there probably should be some collaborators in here



Comment 5

Actually, it means that is not testing some function/feature of the software e.g. performance, security, etc.



Comment 6

No, non-functional tests are typically black box tests e.g. we load a server by trying multiple simultaneous connections to it. We are not interested in how the server works, just does it work under the load we are applying,

PAGE 15

PAGE 16

PAGE 17

PAGE 18

PAGE 19

PAGE 20

PAGE 21

PAGE 22



Comment 7

Is this an input validation check?

I don't think you need this!!

PAGE 23

PAGE 24

PAGE 25
