

Software Engineering Design

by Edward Patch

FILE	11299_EDWARD_PATCH_SOFTWARE_ENGINEERING_DESIGN_339836_890677482.PDF (1.25M)		
TIME SUBMITTED	21-MAR-2021 02:29PM (UTC+0000)	WORD COUNT	6262
SUBMISSION ID	147476340	CHARACTER COUNT	32881

Software Engineering Design

1st Given Edward Patch
Software Engineer Student (of HND Year 2)
Software Principles and Testing
University of Wales Trinity St. Davids (of Mike Dacey)
Swansea, Wales
Student ID: 1801492

CONTENTS		
I	Introduction	2
II	Planning — Tools and Methods	2
II-A	GitHub	2
II-B	xUnit Testing	2
II-C	Jupyter Notebook	2
III	Planning — User Interface Wireframe	2
IV	Planning — Unified Modelling Language	2
V	Planning — User Story	2
VI	Planning — Class Responsibility Collaborator (CRC) Cards	3
VII	Planning — User Sequence Diagram	3
VIII	Terminology	3
VIII-A	Blocking Strategy	3
IX	Planning — Jupyter Notebook	3
IX-A	Player Logic	3
IX-B	Grid Logic	4
X	Planning — Win Check Logic	4
X-A	Vertical Logic — Plan A	4
X-B	Horizontal Logic — Plan A	4
X-C	Horizontal Logic — Plan B	5
X-D	TR Diagonal Logic — Plan A	5
X-E	TL Diagonal Logic — Plan A	5
XI	End-Theory — Win Check Logic	5
XI-A	Vertical Logic	5
XI-B	Horizontal Logic	5
XI-C	TR Diagonal Logic	6
XI-D	TL Diagonal Logic	6
XII	Planning — Blocking Strategy	6
XII-A	Horizontal Strategy — Plan A	6
XII-B	Horizontal Strategy — Plan B	6
XII-C	Vertical Strategy — Plan A	6
XII-D	TR Diagonal Strategy — Plan A	7
XII-E	TL Diagonal Strategy — Plan A	7
XIII	End-Theory — Blocking Strategy	7
XIII-A	Vertical Strategy	7
XIII-B	Horizontal Strategy	7
XIII-C	TR Diagonal Strategy	8
XIII-D	TL Diagonal Strategy	8
XIV	Planning — Test Plan	8
XV	Implementation — Classes/Objects	9
XVI	Evaluation — xUnit Tests	9
XVI-A	Player Logic - xUTPlayer	9
XVI-B	Grid Logic - xUTGrid	10
XVI-C	GameStatus Logic - xUTGameStatus . .	11
XVI-D	Block Move Theory — xUTBlock-MoveTheory	13
XVII	Evaluation — Test Results	14
XVIII	List of Abbreviations	14
XIX	Conclusion	14
XX	Appendix	14
XX-A	Blocking Strategy Transitions	14
XX-B	User Interface Wireframe	15
XX-C	User Story Diagram	16
XX-D	UML Diagram	17
XX-E	CRC Diagram	18
XX-F	User Sequence Diagram	19
XXI	References	21

Abstract—Creation, testing and implementation methods are required for a Connect 4 two player game. The aim of the game is to reach four in a row to win the game.

Index Terms—xUnit, Connect Four, C#, Console Application

I. INTRODUCTION

Planning, implementation and evaluation of a Connect 4 game with a default of 6 x 7 (w x h) grid size.

Rules of the game uses the traditional connect four rules with a twist. Read the Milton Bradley Company [1] instruction manual to understand the traditional rules of Connect 4.

The twist of the traditional rules enables players to eliminate their opponent's pieces. Refer to sub-section XX-A Blocking Strategy Transitions, page 14.

Links are access to the project repository and project files.

Repository - [Connect Four UT](#)

Link - <https://github.com/ShinkuKira21/Connect-Four-UT>
Connect 4 C# Game - [Connect Four C# Game](#)

Link - <https://github.com/ShinkuKira21/Connect-Four-UT/tree/main/C4-Theory>

Connect 4 Planning - [Jupyter Notebooks](#)

Link - <https://github.com/ShinkuKira21/Connect-Four-UT/tree/main/C4-JupyterNB>

II. PLANNING — TOOLS AND METHODS

A. GitHub

GitHub cloud enables collaboration, source control and project management. Collaboration and source control is possible with GitHub's branch management and tracking system. GitHub has many tools, such as the native Git Bash or GitHub Desktop.

GitHub is built-up around the framework of Git, which widens the functionality. Git offers version control and code sharing. GitHub provides a cloud where source code can be uploaded and retrieved via the framework, Git.

Cameron McKenzie, TechTarget [2] mentions the commonly used Git commands "For most developers, Git commands, such as commit, reset, push, revert, checkout, fetch and merge are self-evident in their purpose and thus easy to understand."

Automated tasks and workspaces are available for GitHub projects creating an ideal workstation for an individual, team or organisation. GitHub suits Open-source, freeware and closed source software by giving an option to create public/private repositories.

However, larger projects that are private requires GitHub's subscriptions found on [GitHub's Pricing](#) page. GitHub's Pricing Link - <https://github.com/pricing>

B. xUnit Testing

xUnit Testing .NET Core enables a method to test new or existing code logic. xUnit Testing written in CSharp (C#) programming language offers an Assert class library, which provides tools to test expected data with existing data. A few Assert methods used in this project with the description described by Microsoft [3],

AssertClass(Microsoft.VisualStudio.TestTools.UnitTesting.

List of Datatypes

- string - The string data type holds a string of characters.
- char - The character (char) data type holds one character.
- int - The integer (int) data type holds whole numbers only. An int can only hold from -2,147,483,648 to 2,147,483,647 based on the Microsoft's [4] Integral numeric types (C# reference) documentation.
- bool - The boolean (bool) data type stores True or False values.

Assert Functions

Note: All return paths of Assert class functions are in the Test Explorer of Visual Studio. All functions return boolean values.

Method : Definition

- AreEqual(string, string) : Compares two string arguments.
- AreEqual(char, char) : Compares two char arguments.
- AreEqual(int, int) : Compares two int arguments.
- IsTrue(bool) : Checks if bool argument is true.
- IsFalse(Booleen) : Checks if bool argument is false.

Private functions can be created in a xUnit class to help testing procedures by setting up an expected environment.

C. Jupyter Notebook

[Jupyter Lab](#) offers a service called [Jupyter Notebook](#). [IPython](#) Python plugin is utilised to help generate any diagrams in Jupyter Notebook. Jupyter Notebook, formerly known as the IPython Notebook. Jupyter Notebook is used in this particular case to create theory code and planning documentation like Unified Modelling Language (UML), Class Responsibility Collaborator (CRC) and User Sequence diagram. To gain access to these notebooks by clicking any external link mentioned above the evidence figures. To browse planning documentation, visit the [C4-JupyterNB](#) section of the Connect-Four-UT repository. Link: <https://github.com/ShinkuKira21/Connect-Four-UT/tree/main/C4-JupyterNB>

III. PLANNING — USER INTERFACE WIREFRAME

[User Interface Wireframe](#) of the Connect 4 project. Refer to sub-section XX-B User Interface Wireframe, page 15, to see evidence of the User Interface Wireframe.

IV. PLANNING — UNIFIED MODELLING LANGUAGE

[UML Class Diagram](#) of the Connect 4 project. Refer to sub-section XX-D UML Diagram, page 17, to see evidence of the UML diagram.

V. PLANNING — USER STORY

[User Story Diagram](#) of the Connect 4 project. Refer to sub-section XX-C User Story Table, page 16, to see evidence of the User Story table. The table is based off ProductPlan [5] User Story example. A user story describes the role of a object and what communication each object requires. These are often found in Agile Software Development.

VI. PLANNING — CLASS RESPONSIBILITY COLLABORATOR (CRC) CARDS

[CRC Cards Diagram](#) of the Connect 4 project. Refer to sub-section XX-E CRC Diagram, page 18, to see evidence of the CRC diagram.

VII. PLANNING — USER SEQUENCE DIAGRAM

[User Sequence Diagram](#) of the Connect 4 project. Refer to sub-section XX-F User Sequence Diagram, page 19, to see the User Sequence diagram evidence.

VIII. TERMINOLOGY

Note: Only valid if no other context is added.

The X-axis represents the vertical axis. The Y-axis represents the horizontal axis. i variable represents the X-axis incrementation loop, and j variable represents the Y-axis incrementation loop. $n++$ is equivalent to $n = n + 1$.

Note: The following terminology is found when describing diagonal logic.

TL stands for Top-Left. *TR* stands for Top-Right. *BL* stands for Bottom-Left. *BR* stands for Bottom-Right.

A. Blocking Strategy

Referring to section XII Planning - Blocking Strategy, page 6. Mentioned memory cells explain what will happen during the process. Memory cells mentioned on their own will first set to the grid icon. The arrow represents how the memory will move from one to another. Refer to Section XX-A Blocking Strategy Transitions, page 14, to see an example. From the referenced example:

3,0 - 2,1 - 1,2 memory cells will set to the grid icon.
1,1 to 2,1 - 1,1 counter will move to 2,1, and 1,1 will set to the grid icon.

IX. PLANNING — JUPYTER NOTEBOOK

Jupyter notebook plans are available at [C4-JupyterNB](#). Link: <https://github.com/ShinkuKira21/Connect-Four-UT/tree/main/C4-JupyterNB>

A. Player Logic

The text-based code figure below displays how the players should be set up. Click on the following link, [Player Plan Design](#), to look at the Players Logic Planning written in Markdown and Python.

```
* UI Logic *
Enter Player 1s Nick: James
Enter Player 2s Nick: Emily

* xUT Logic *

Player 1 - Choose starter icon
(X/O): O
xUnit Testing Logic
SetPlayerName(string p1Nick, string
p2Nick)
# selector to choose the playerIndex
names = GetPlayerName(int
playerIndex)

# p1Icon is defaulted to X
SetPlayerIcon(char p1Icon = 'X')

# Internal logic will determine
player twos piece
icons = GetPlayerIcon(int
playerIndex)
```

Fig. 1. Text-Base Plan of Player Logic Jupyter Notebook Plan.

B. Grid Logic

The text-based code figure below displays the theory of the expected grid layout. Click on the following link, [Grid Plan Design](#), to look at the Grid Logic Design written in Markdown.

```
## UI Logic

'''
  0 1 2 3 4 5 6
0 - - - - -
1 - - - - -
2 - - - - -
3 - - - - -
4 - - - - -
5 - - - - -
6 - - - - -
7 - - - - -

Enter Y: 3

  0 1 2 3 4 5 6
0 - - - - -
1 - - - - -
2 - - - - -
3 - - - - -
4 - - - - -
5 - - - - -
6 - - - - -
7 - - - X - -

Enter Y: 6

  0 1 2 3 4 5 6
0 - - - - -
1 - - - - -
2 - - - - -
3 - - - - -
4 - - - - -
5 - - - - -
6 - - - - -
7 - - - X - O
'''
```

Fig. 2. Text-Base Plan of Grid Logic Jupyter Notebook Plan.

X. PLANNING — WIN CHECK LOGIC

Note: Yellow cells represent X counter and green cells represents green counter. Refer to section XI Win Check for the detailed analyst of the following **Win Check Logic** plans. The **Affected Memory Cells** outputs the expected winner of the game.

A. Vertical Logic — Plan A

Image 3 demonstrates the X-axis increments by one ($i++$). Refer to sub-section XI-A Vertical Logic, page 5.

Connect 4					
Realistic		1	2	3	4
	1	x	-	-	-
	2	x	-	-	-
	3	x	-	o	-
	4	x	-	o	o
Memory Ref		1	2	3	4
	1	0,0	0,1	0,2	0,3
	2	1,0	1,1	1,2	1,3
	3	2,0	2,1	2,2	2,3
	4	3,0	3,1	3,2	3,3
Affected Memory Cells					
End Game X Wins					

Fig. 3. Plan of vertical win logic - Plan A.

B. Horizontal Logic — Plan A

Image 4 demonstrates the Y-axis increments by one ($j++$). Refer to sub-section XI-B Horizontal Logic, page 5.

Connect 4					
Realistic		1	2	3	4
	1	-	-	-	-
	2	x	x	x	-
	3	o	o	o	o
	4	x	o	x	o
Memory Ref		1	2	3	4
	1	0,0	0,1	0,2	0,3
	2	1,0	1,1	1,2	1,3
	3	2,0	2,1	2,2	2,3
	4	3,0	3,1	3,2	3,3
Affected Memory Cells					
End Game O Wins					

Fig. 4. Plan of horizontal win logic - Plan A.

C. Horizontal Logic — Plan B

Image 5 demonstrates the similar theory of **Plan A**. This example displays the Win Check should be prioritised before the Blocking Strategy check.

Connect 4					
		1	2	3	4
Realistic	1	x	o	-	-
	2	o	x	-	-
	3	x	o	x	-
	4	o	o	o	o
Memory Ref	1	0,0	0,1	0,2	0,3
	2	1,0	1,1	1,2	1,3
	3	2,0	2,1	2,2	2,3
	4	3,0	3,1	3,2	3,3
Affected Memory Cells					
End Game O Wins					

Fig. 5. Plan of horizontal win logic - Plan B.

D. TR Diagonal Logic — Plan A

Image 6 demonstrates the X-axis starts at three and decrements by one ($i = 3; i--$), and Y-axis starts at zero and increments by one ($j++$). Refer to sub-section XI-C TR Diagonal Logic, Page 6.

Connect 4					
		1	2	3	4
Realistic	1	-	-	-	x
	2	-	-	x	o
	3	o	x	x	x
	4	x	o	o	o
Memory Ref	1	0,0	0,1	0,2	0,3
	2	1,0	1,1	1,2	1,3
	3	2,0	2,1	2,2	2,3
	4	3,0	3,1	3,2	3,3
Affected Memory Cells					
End Game X Wins					

Fig. 6. Plan of top right diagonal win logic - Plan A.

E. TL Diagonal Logic — Plan A

Note: This logic only works for even-numbered grid sizes such as 4x4.

Evidence below demonstrates that both X-axis and Y-axis increments by one ($i++$ and $j++$). Refer to sub-section XI-D TL Diagonal Logic, page 6.

Connect 4					
		1	2	3	4
Realistic	1	o	-	-	-
	2	o	o	-	-
	3	x	x	o	x
	4	x	x	o	o
Memory Ref	1	0,0	0,1	0,2	0,3
	2	1,0	1,1	1,2	1,3
	3	2,0	2,1	2,2	2,3
	4	3,0	3,1	3,2	3,3
Affected Memory Cells					
End Game O Wins					

Fig. 7. Plan of top left diagonal win logic - Plan A.

XI. END-THEORY — WIN CHECK LOGIC

A. Vertical Logic

After examining the fig 3 Vertical Win Logic - Plan A, page 4, the following vertical win logic is planned.

```
for loop i til less than y
  for loop j til less than x
    if counter is equal to 4
      end game with winners name.
    if grid[j][i] equal to
      previousCounter
      counter++
    else
      counter = 0
```

B. Horizontal Logic

After examining the fig 4 Horizontal Win Logic - Plan A, page 4, the following horizontal win logic theory is planned.

```
for loop i til less than x
  for loop j til less than y
    if counter is equal to 4
      end game with winners name.
    if grid[i][j] equal to
      previousCounter
      counter++
    else
      counter = 0
```

C. TR Diagonal Logic

The evidence of fig 6 TR Diagonal Win Logic - Plan A, page 5, helped create the following win check logic.

```
for i = x loop til i is equal to 0
  for loop j til less than y
    if counter is equal to 4
      end game with winners name.
    if grid[i][j] equal to
      previousCounter
      counter++
    else
      counter = 0
```

D. TL Diagonal Logic

The following logic has been created based on the Top-Left Diagonal logic fig 7 TL Diagonal Win Logic - Plan A, page 5. This theory should work on any size grids.

```
for loop i til less than x - 3
  for loop j til less than y - 3
    if grid[i][j] == previousCounter and
      grid[i][j] equal to
      grid[i+1][j+1] and grid[i][j]
      equal to grid[i+2][j+2] and
      grid[i][j] equal to
      grid[i+3][j+3]
      end game with winners name.
```

XII. PLANNING — BLOCKING STRATEGY

A. Horizontal Strategy — Plan A

Evidence of the horizontal strategy demonstrates that the logic of fig 4 is similar. However, instead of checking for identical, the check should check for the opposite player, clearing the affected memory cells. Refer to sub-section XIII-B Horizontal Strategy, page 7.

Connect 4					
Realistic		1	2	3	4
	1	-	-	-	-
	2	-	-	-	-
	3	-	-	-	-
	4	x	x	x	o
Memory Ref		1	2	3	4
	1	0,0	0,1	0,2	0,3
	2	1,0	1,1	1,2	1,3
	3	2,0	2,1	2,2	2,3
	4	3,0	3,1	3,2	3,3
Affected Memory Cells					
3,0 3,1 3,2					
Reset Cells to Blank					

Fig. 8. Plan of horizontal replacement strategy - Plan A.

B. Horizontal Strategy — Plan B

This example demonstrates how the 2,0 and 2,1 counters will move down, after the 3,0 - 3,1 and 3,2, the grid icon replaces the mentioned slots. Refer to sub-section XIII-B Horizontal Strategy, page 7.

Connect 4					
Realistic		1	2	3	4
	1	-	-	-	-
	2	-	-	-	-
	3	o	x	-	-
	4	x	x	x	o
Memory Ref		1	2	3	4
	1	0,0	0,1	0,2	0,3
	2	1,0	1,1	1,2	1,3
	3	2,0	2,1	2,2	2,3
	4	3,0	3,1	3,2	3,3
Affected Memory Cells					
3,0 3,1 3,2					
2,0 -> 3,0 2,1					
Reset Cells to Blank					

Fig. 9. Plan of horizontal replacement strategy - Plan B.

C. Vertical Strategy — Plan A

The vertical strategy indicates that the counter on 0,0 has to move down to 3,0, after setting 1,0 - 2,0 and 3,0 memory cells to grid icon. Refer to sub-section XIII-A Vertical Strategy, page 7.

Connect 4					
Realistic		1	2	3	4
	1	o	-	-	-
	2	x	-	-	-
	3	x	-	-	-
	4	x	o	o	-
Memory Ref		1	2	3	4
	1	0,0	0,1	0,2	0,3
	2	1,0	1,1	1,2	1,3
	3	2,0	2,1	2,2	2,3
	4	3,0	3,1	3,2	3,3
Affected Memory Cells					
1,0 2,0 3,0					
0,0 -> 3,0					
Reset Cells to Blank					

Fig. 10. Plan of vertical replacement strategy - Plan A.

D. TR Diagonal Strategy — Plan A

Plan A changes the logic to the previous theories. This is due to the logic checking if the first three are a particular counter then checking if there is a block. Refer to sub-section XIII-C TR Diagonal Strategy, page 6.

Connect 4					
		1	2	3	4
Realistic	1	-	-	-	x
	2	-	-	x	x
	3	-	o	o	x
	4	x	x	o	o
		1	2	3	4
Memory Ref	1	0,0	0,1	0,2	0,3
	2	1,0	1,1	1,2	1,3
	3	2,0	2,1	2,2	2,3
	4	3,0	3,1	3,2	3,3
		Affected Memory Cells			
		3,0	1,2	0,3	
		Reset Cells to Blank			
		-	-	-	-

Fig. 11. Plan of top right diagonal replacement strategy - Plan A.

E. TL Diagonal Strategy — Plan A

This top-left diagonal strategy example shows that 0,0 - 1,1 and 2,2 memory cells will change to grid icon. On turn, causes the memory cell 0,1 counter to move to 1,1. Refer to sub-section XIII-D, page 8.

Connect 4					
		1	2	3	4
Realistic	1	x	o	-	-
	2	o	x	-	-
	3	x	o	x	-
	4	o	x	o	o
		1	2	3	4
Memory Ref	1	0,0	0,1	0,2	0,3
	2	1,0	1,1	1,2	1,3
	3	2,0	2,1	2,2	2,3
	4	3,0	3,1	3,2	3,3
		Affected Memory Cells			
		0,0	1,1	2,2	
		0,1 -> 1,1			

Fig. 12. Plan of top left diagonal replacement strategy - Plan A.

XIII. END-THEORY — BLOCKING STRATEGY

A. Vertical Strategy

Code block is created from the fig 10 Vertical Strategy - Plan A, page 6.

```

for loop i til less than y
  for loop j til less than x
    if counter is equal to 3
      if grid[j][i] equal to
        previousCounter
        counter++
      else if grid[j][i] is not equal to
        gridIcon
        remove blocked counter cells and
        drop pieces in existing slots.
    else
      counter = 0

```

B. Horizontal Strategy

Logic shown below is derived from the horizontal strategies found at fig 8 Horizontal Strategy - Plan A, page 6 and fig 9 Horizontal Strategy - Plan B, page 6.

```

for loop i til less than x
  for loop j til less than y
    if counter is equal to 3
      if grid[i][j] equal to
        previousCounter
        counter++
      else if grid[i][j] is not equal to
        gridIcon
        remove blocked counter cells and
        drop pieces in existing slots.
    else
      counter = 0

```

C. TR Diagonal Strategy

Reference to fig 11 TR Diagonal Replacement - Plan A, page 7.

The standard win check logic of fig 6 End Theory - Win Check, TR Diagonal, page 5, checking the third difference could work but would fail if the opposite counter is placed anywhere other than the end position. An attempt of the code is demonstrated below:

```
for loop i til less than x
for loop j is less than y
  if i is less than x and j - 3 is
    more than or equal to 0
    if board[i][j] equal to
      board[i+1][j-1] and
      board[i][j] equal to
      board[i+2][j-2] and
      board[i][j] is not equal to
      board[i+3][j-3] and
      board[i+3][j-3] is equal to
      GridIcon
      board[i][j] = board[i-1][j]
      board[i][j+1] = board[i-1][j+1]
      board[i][j+2] = board[i-1][j+2]

return true
```

D. TL Diagonal Strategy

Reference to fig 12 TL Diagonal Replacement - Plan A, page 7.

```
for loop i til less than x
for loop j is less than y
  if i is less than x and j - 3 is
    more than or equal to 0
    if board[i][j] equal to
      board[i+1][j+1] and
      board[i][j] equal to
      board[i+2][j+2] and
      board[i][j] is not equal to
      board[i+3][j+3] and
      board[i+3][j+3] is equal to
      GridIcon
      board[i][j] = board[i-1][j]
      board[i+1][j+1] = board[i-1][j]
      board[i+2][j+2] = board[i-1][j]

return true
```

XIV. PLANNING — TEST PLAN

Note: Group labels follows: A - Player, B - Grid, C - GameStatus and D - Block Move Strategy.

TABLE I
TABLE OF TEST PLANNING OF THE XUNIT TESTS.

Group	Test	What to test?	Expected Result
A	Player Constructor Test	Test if player name and player icons are set properly.	Names / Icon 0: James / O 1: Mary / X
A	Edit Player Test	Test if names and icons are not identical to the first set of names and icons.	Names/Icon 0: James / O 1: Mary / X After Edition Is not James or O. Is not Mary or X.
A	Player Constructor Icon Constraint	Test player class constructor constraints to see if it accepts P instead of X or O.	P should not be set. Default of X should be set.
A	Player Edit Icon Constraint	Test player class edit method to see if it accepts P instead of X or O.	P should not be set. Default of X should be set.
B	Grid Constructor Test	Test if grid is identical to the expected grid.	Refer to sub-section IX-B Grid Logic, page 4
B	Grid Make Move Test	Test if players can drop counters on the grid.	The counters should drop down to the relevant places.
B	Grid Validation	Test two different class instances, with the same moves to see if identical. Another test is carried out to test if the grids match the static grid.	These three grids should match
C	Vertical Win Check	Test if vertical win check works on 6w x 9h grid.	Test top half and bottom half of the win check should count as a win.
C	Horizontal Win Check	Test if horizontal win check works on 6w x 7h grid.	Left half and right half of the win check should count as a win.
C	Diagonal Top Left Win Check	Test if top-left diagonal win check works on 6w x 7h grid.	Top, bottom part of diagonals should count as a win, including intersection.
C	Diagonal Top Right Win Check	Test if top-right diagonal win check works on 6w x 7h grid.	Top, bottom part of diagonals should count as a win, including intersection.
D	Vertical Block Strategy	Test vertical block strategy.	Vertical column should reset to grid icon.
D	Horizontal Block Strategy	Test horizontal block strategy.	Three matching counters should reset to grid icon. Pieces on top should fall down.
D	Diagonal Block Strategy	Test diagonal block strategy.	Three matching counters should reset to grid icon. Pieces above diagonal should fall down.

XV. IMPLEMENTATION — CLASSES/OBJECTS

After comparing XX-D UML Diagram, page 17, XX-E CRC Diagram, page 18 and XX-F User Sequence Diagram, page 19 design documentation, three dominant classes selected are as followed:-

- **GameStatus** - Manages the status of the game. The class includes whether the game is being played or not and determines the winner of the game. The GameStatus will contain win check functionality found under XI End Theory - Win Check, page 5. A relationship to the Grid class is necessary to check the current Grid for current GameStatus updates.
As the GameStatus class requires access to the winner name and icon relationship, the class would have to communicate to the Players class.
- **Grid** - The main element of the game would have to be the Grid object. The class takes care of displaying, setting, and the block move functionality. Blocking Strategy XIII, page XIII is included in the Grid class. The Grid class handles players pieces. The Grid class needs a relationship with the Players class to correctly ensure the correct pieces store on the Connect 4 grid. Extra functionality includes the grid icon variable to enable players to change the Grid icon from default.
- **Players** - Players class will handle player names and icons. Functionality includes setters and getters of player names and icons.

XVI. EVALUATION — XUNIT TESTS

Note: *T_n* - Test number.

End evaluation of tests created from the table I Planning - Test Plan, Page 8.

A. Player Logic - *xUTPlayer*

Test 1 — Player Test

The Player Test will test the Players class constructor to ensure the Constructor and getter methods works. Tests if the names array matches the return value of the Players class `GetPlayerName` getter function and if 'O' character is equal to `GetPlayerIcon(0)` getter function.

Code:

```
public void PlayerTest ()
{
    play = new Players(names, 'O');

    for(int i = 0; i < 2; i++)
        Assert.Equal(names[i],
            play.GetPlayerName(i));

    Assert.Equal('O',
        play.GetPlayerIcon(0));
}
```

Outcome (T1): The test is successful.

Tweaks: No tweaking required.

Test 2 — Player Edit Test

The Player Edit Test will test the Players class's setters. Tests if `SetPlayerName` for player one and player two will not be equal with the names array. `SetPlayerIcon` must match 'X' character.

Code:

```
public void PlayerEditTest ()
{
    play = new Players(names, 'O');

    // inverted names
    play.SetPlayerName("Mary", 0);
    play.SetPlayerName("John", 1);
    play.SetPlayerIcon('X');

    for (int i = 0; i < 2; i++)
        Assert.NotEqual(names[i],
            play.GetPlayerName(i));

    Assert.Equal('X',
        play.GetPlayerIcon(0));
}
```

Outcome (T1): The test is successful.

Tweaks: No tweaking required.

Test 3 — Player Constructor Icon Constraint

The Player Constructor Icon Constraint checks the icon constraint. The only characters should be 'X' or 'O'. The test will check if 'P' is not equal to `GetPlayerIcon(0)` Players getter and is equal to 'X'.

Code:

```
public void
    PlayerConstructorIconConstraint ()
{
    // X will be selected as default
    // if O is not chose
    play = new Players(names, 'P');

    Assert.NotEqual('P',
        play.GetPlayerIcon(0));
    Assert.Equal('X',
        play.GetPlayerIcon(0));
}
```

Outcome (T1): The test is successful.

Tweaks: No tweaking required.

Test 4 — Player 2 Icon Test

The Player 2 Icon Test checks if the icon of player two is opposite to the player one icon. A player cannot have identical pieces. Otherwise, it makes it hard to understand what is happening within the game.

Code:

```
public void Player2IconTest ()
{
    play = new Players(names, 'O');
    Assert.Equal('X',
        play.GetPlayerIcon(1));

    play.SetPlayerIcon('X');
    Assert.Equal('O',
        play.GetPlayerIcon(1));
}
```

Outcome (T1): The test is successful.

Tweaks: No tweaking required.

B. Grid Logic - xUTGrid

Test 1 — Grid Setup Test

The Grid Setup Test checks the ideal grid. The ideal grid is size 4wx4h with the default grid icon, '-' is equal to the generated grid from the Grid class.

Code:

```
public void GridSetupTest ()
{
    play = new Players(names, 'O');
    grid = new Grid(ref play, 4,
        gridIcon: '-');

    Assert.Equal(IdealGrid(),
        grid.OutputGrid());
}
```

Outcome (T1): The test is unsuccessful. **Reason:** Spacing between characters were inconsistent, character(s) "\t".

Tweaks: Spacing between characters were changes to four empty white spaces to make the spacing more consistent.

Outcome (T2): The test is unsuccessful. **Reason:** Vertical spacing did not exist for the heading line.

Tweaks: Character(s) "\n" added after heading line loop to create vertical space between heading and playing field.

Outcome (T3): The test is successful.

Tweaks: No tweaking required.

Test 2 — Default Grid Test

The Default Grid Test will check if the default grid IX-B Grid Logic, page 4, is identical to the actual grid scripted.

Code:

```
public void DefaultGridTest ()
{
    play = new Players(names, 'X');
    grid = new Grid(ref play);

    Assert.Equal(DefaultGrid(),
        grid.OutputGrid());
}
```

Outcome (T1): The test is successful.

Tweaks: No tweaking required.

Test 3 — Grid Move Test

The Grid Move Test checks if a player can make a move. This is done by checking if Ideal Grid is not identical to the generated grid from the Grid class.

Code:

```
public void GridMoveTest ()
{
    play = new Players(names, 'X');
    grid = new Grid(ref play, 4,
        gridIcon: '-');

    grid.MakeMove(0, 2);
    grid.MakeMove(1, 2);

    Assert.NotEqual(IdealGrid(),
        grid.OutputGrid());
}
```

Outcome (T1): The test is successful.

Tweaks: No tweaking required.

Test 4 — Grid Validation 1

The Grid Validation 1 checks if moves that were committed to a copy of the generated grid by Grid class is identical to the same moves committed on main generated grid by Grid class. This helps reduce any inconsistencies within the algorithm of the grid code used.

Code:

```
public void GridValidation1()
{
    play = new Players(names, 'X');
    Grid copyGrid = new Grid(ref
        play, 4);
    grid = new Grid(ref play, 4);

    copyGrid.MakeMove(0, 2);
    copyGrid.MakeMove(1, 2);
    copyGrid.MakeMove(0, 2);
    copyGrid.MakeMove(1, 2);
    copyGrid.MakeMove(0, 2);

    grid.MakeMove(0, 2);
    grid.MakeMove(1, 2);
    grid.MakeMove(0, 2);
    grid.MakeMove(1, 2);
    grid.MakeMove(0, 2);

    Assert.Equal(
        copyGrid.OutputGrid(),
        grid.OutputGrid());

    Assert.Equal(
        ValidationGridNoOverlap(),
        grid.OutputGrid());
}
```

Outcome (TI): The test is successful.

Tweaks: No tweaks required.

C. GameStatus Logic - xUTGameStatus

Note: GameStatus false indicates the game is ended, where if GameStatus is true indicates the game is resuming.

Test 1 — Vertical Move Win Check

The Vertical Move Win Check checks for the following:-

- Check if GameStatus is false and winner is Mary.
- Check if GameStatus is false and winner is John.
- Check if GameStatus is true.

To justify this check, it will check if the vertical win check within the GameStatus class works correctly.

Code:

```
public void VerticalMoveWinCheck()
{
    play = new Players(names, 'X');
    grid = new Grid(ref play);
    gs = new GameStatus(ref play, ref
        grid);

    VerticalMove(1, 0);

    // if winner is Mary
    Assert.False(gs.GetGameStatus());
    // this updates the winner
    (if false gameOver)
    Assert.Equal(1, gs.GetWinner());

    grid.ClearGrid();

    // if winner is John
    VerticalMove();
    Assert.False(gs.GetGameStatus());
    Assert.Equal(0, gs.GetWinner());

    grid.ClearGrid();

    // No winner, Game Continuing
    VerticalMove(bWinGame: false);
    Assert.True(gs.GetGameStatus());
}
```

Outcome (TI): The test is successful.

Tweaks: No tweaking required.

Test 2 — Horizontal Move Win Check

The Vertical Move Win Check checks for the following:-

- Check if GameStatus is false and winner is Mary.
- Check if GameStatus is false and winner is John.
- Check if GameStatus is true.

To justify this check, it will check if the horizontal win check within the GameStatus class works correctly.

Code:

```
public void HorizontalMoveWinCheck ()
{
    play = new Players(names, 'X');
    grid = new Grid(ref play);
    gs = new GameStatus(ref play, ref
        grid);

    HorizontalMove(1, 0);

    // if winner is Mary
    Assert.False(gs.GetGameStatus());
    // this updates the winner
    (if false gameOver)
    Assert.Equal(1, gs.GetWinner());

    grid.ClearGrid();

    // if winner is John
    HorizontalMove();
    Assert.False(gs.GetGameStatus());
    Assert.Equal(0, gs.GetWinner());

    grid.ClearGrid();

    // No winner, Game Continuing
    HorizontalMove(bWinGame: false);
    Assert.True(gs.GetGameStatus());
}
```

Outcome (T1): The test is successful.

Tweaks: No tweaks required.

Test 3 — Diagonal TL BR Move Win Check

The Diagonal TL BR Move Win Check tests following:-

- Check if GameStatus is false and winner is Mary.
- Check if GameStatus is false and winner is John.
- Check if GameStatus is true.

To justify this check, it will check if the diagonal top-left to bottom-right win check within the GameStatus class works correctly.

Code:

```
public void
    DiagonalTLBRMoveWinCheck ()
{
    play = new Players(names, 'X');
    grid = new Grid(ref play);
    gs = new GameStatus(ref play, ref
        grid);

    DiagonalTLBRMove(1, 0);

    // if winner is Mary
    Assert.False(gs.GetGameStatus());
    // this updates the winner
    (if false gameOver)
    Assert.Equal(1, gs.GetWinner());

    grid.ClearGrid();

    // if winner is John
    DiagonalTLBRMove();
    Assert.False(gs.GetGameStatus());
    Assert.Equal(0, gs.GetWinner());

    grid.ClearGrid();

    // No winner, Game Continuing
    DiagonalTLBRMove(bWinGame: false);
    Assert.True(gs.GetGameStatus());
}
```

Outcome (T1): The test is successful.

Tweaks: No tweaking required.

Test 4 — Diagonal TR BL Move Win Check

The Diagonal TR BL Move Win Check tests following:-

- Check if GameStatus is false and winner is Mary.
- Check if GameStatus is false and winner is John.
- Check if GameStatus is true.

To justify this check, it will check if the diagonal top-right to bottom-left win check within the GameStatus class works correctly.

Code:

```
public void
    DiagonalTRBLMoveWinCheck()
{
    play = new Players(names, 'X');
    grid = new Grid(ref play);
    gs = new GameStatus(ref play, ref
        grid);

    DiagonalTRBLMove(1, 0);

    // if winner is Mary
    Assert.False(gs.GetGameStatus());
    // this updates the winner
    (if false gameOver)
    Assert.Equal(1, gs.GetWinner());

    grid.ClearGrid();

    // if winner is John
    DiagonalTRBLMove();
    Assert.False(gs.GetGameStatus());
    Assert.Equal(0, gs.GetWinner());

    grid.ClearGrid();

    // No winner, Game Continuing
    DiagonalTRBLMove(bWinGame: false);
    Assert.True(gs.GetGameStatus());
}
```

Outcome (T1): The test is successful.

Tweaks: No tweaks required.

D. Block Move Theory — xUTBlockMoveTheory

The Block Move Theory testing has two problems. If pieces are higher than one row then it won't drop. Another example, if XOOX then it won't eliminate, but if XXXO then it will.

Another problem that exists, is that if XOX is already in place, then the extra X that is placed on the end, XOOX, then it will eliminate. This should not happen, as it's not blocking a win at this point.

Test 1 — Vertical Elimination

The Vertical Elimination ensures that the sub-section XII-C Vertical Strategy - Plan A, page 6 is implemented properly. The check checks if EliminateMove returns true.

Code:

```
public void VerticalElimination()
{
    play = new Players(names, 'X');
    grid = new Grid(ref play);
    board = grid.GetGrid();
    VerticalMove();

    Assert.True(EliminateMove(board));
}
```

Outcome (T1): The test is successful.

Tweaks: No tweaking required.

Test 2 — Horizontal Elimination

The Horizontal Elimination checks if the implementation for the planning of sub-section XII-A Horizontal Strategy - Plan A, page 6 works in the test code correctly.

Code:

```
public void HorizontalElimination()
{
    play = new Players(names, 'X');
    grid = new Grid(ref play);
    board = grid.GetGrid();
    HorizontalMove();

    Assert.True(EliminateMove(board));
}
```

Outcome (T1): The test is successful.

Tweaks: No tweaking required.

Test 3 — Diagonal TL BR Elimination

The Diagonal TL BR Elimination checks if the implemented logic is identical to the plan of sub-section XII-E TL Diagonal Strategy - Plan A, page XII-E.

Code:

```
public void DiagonalTLBRElimination()
{
    play = new Players(names, 'X');
    grid = new Grid(ref play);
    board = grid.GetGrid();
    DiagonalTLBRMove();

    Assert.True(EliminateMove(board));
}
```

Outcome (T1): The test is successful.

Tweaks: No tweaks required.

Test 4 — Diagonal TR BL Elimination The Diagonal TR BL Elimination checks if the implemented logic is identical to the plan of sub-section XII-D TR Diagonal Strategy - Plan A, page XII-D.

Code:

```
public void DiagonalTRBLElimination()
{
    play = new Players(names, 'X');
    grid = new Grid(ref play);
    board = grid.GetGrid();
    DiagonalTRBLMove();

    Assert.True(EliminateMove(board));
}
```

Outcome (T1): The test is successful.

Tweaks: No tweaks required.

XVII. EVALUATION — TEST RESULTS

TABLE II
TABLE OF xUNIT SUMMARY.

Group	xUnit Test Name	Actual Result	Alterations
A	Player Test	Passed.	N/A
A	Player Edit Test	Passed.	N/A
A	Player Constructor Icon Constraint	Passed.	N/A
A	Player 2 Icon Test	Passed.	N/A
B	Grid Setup Test	T1: Failed. T2: Failed. T3: Passed.	Added four white spaces for consistent spacing. Added new line after heading loop.
B	Default Grid Test	Passed.	N/A
B	Grid Move Test	Passed.	N/A
B	Grid Validation 1	Passed.	N/A
C	Vertical Move Win Check	Passed.	N/A
C	Horizontal Move Win Check	Passed.	N/A
C	Diagonal Top Left Move Win Check	Passed.	N/A
C	Diagonal Top Right Move Win Check	Passed.	N/A
D	Vertical Elimination	Passed.	N/A
D	Horizontal Elimination	Passed.	N/A
D	Top Left Diagonal Elimination	Passed.	N/A
D	Top Right Diagonal Elimination	Passed.	N/A

XVIII. LIST OF ABBREVIATIONS

- C# - CSharp Programming Language.
- UML - Unified Modelling Language.
- CRC - Class Responsibility Collaborator.
- (i) - Loop variable for X-Axis.
- (j) - Loop variable for Y-Axis.
- (i++) - i incrementation by 1.
- (j++) - j incrementation by 1.
- TR - Top-Right.
- TL - Top-Left.
- BR - Bottom-Right.
- BL - Bottom-Left.
- Tn - Test Number.

XIX. CONCLUSION

In conclusion, the planning of Connect 4 strategy covers the player, grid and functionality of movement and blocking techniques. A test plan and evaluation table establish what features need to be tested and tested via xUnit tests.

UML, CRC cards and user sequence diagrams created in Jupyter Notebook with IPlantUML based on the Connect 4 game. The C# Connect 4 game and planning documentation project is on GitHub at <https://github.com/ShinkuKira21/Connect-Four-UT>.

After compiling the block strategy logic, three unique problems were found. In sub-section XVI-D Block Move Theory — xUTBlockMoveTheory, page 13, these problems were addressed.

XX. APPENDIX

A. Blocking Strategy Transitions

Transition — S-A

Connect 4					
		1	2	3	4
Realistic	1	-	-	x	-
	2	-	x	x	o
	3	-	x	o	o
	4	x	o	o	x
Memory Ref	1	0,0	0,1	0,2	0,3
	2	1,0	1,1	1,2	1,3
	3	2,0	2,1	2,2	2,3
	4	3,0	3,1	3,2	3,3
Affected Memory Cells					
Game Continues					

Fig. 13. Transition Step A

Transition — S-B

Connect 4					
		1	2	3	4
Realistic	1	-	-	x	o
	2	-	x	x	o
	3	-	x	o	o
	4	x	o	o	x
Memory Ref	1	0,0	0,1	0,2	0,3
	2	1,0	1,1	1,2	1,3
	3	2,0	2,1	2,2	2,3
	4	3,0	3,1	3,2	3,3
Affected Memory Cells					
Move Block Detected					

Fig. 14. Transition Step B

Transition — S-C

Connect 4						
Realistic		1	2	3	4	
	1	-	-	x	o	
	2	-	x	-	o	
	3	-	-	o	o	
	4	-	o	o	x	
Memory Ref		1	2	3	4	
	1	0,0	0,1	0,2	0,3	
	2	1,0	1,1	1,2	1,3	
	3	2,0	2,1	2,2	2,3	
	4	3,0	3,1	3,2	3,3	
Affected Memory Cells						
		3,0	2,1	1,2		
		1,1 -> 2,1		0,2 -> 1,2		

Fig. 15. Transition Step C

Transition — S-D

Connect 4						
Realistic		1	2	3	4	
	1	-	-	-	o	
	2	-	-	x	o	
	3	-	x	o	o	
	4	-	o	o	x	
Memory Ref		1	2	3	4	
	1	0,0	0,1	0,2	0,3	
	2	1,0	1,1	1,2	1,3	
	3	2,0	2,1	2,2	2,3	
	4	3,0	3,1	3,2	3,3	
Affected Memory Cells						
Move Blocked						

Fig. 16. Transition Step D

B. User Interface Wireframe

Page 1



Fig. 17. Figure of User Interface Wireframe created by Edward Patch with Moqups [6].

C. User Story Diagram

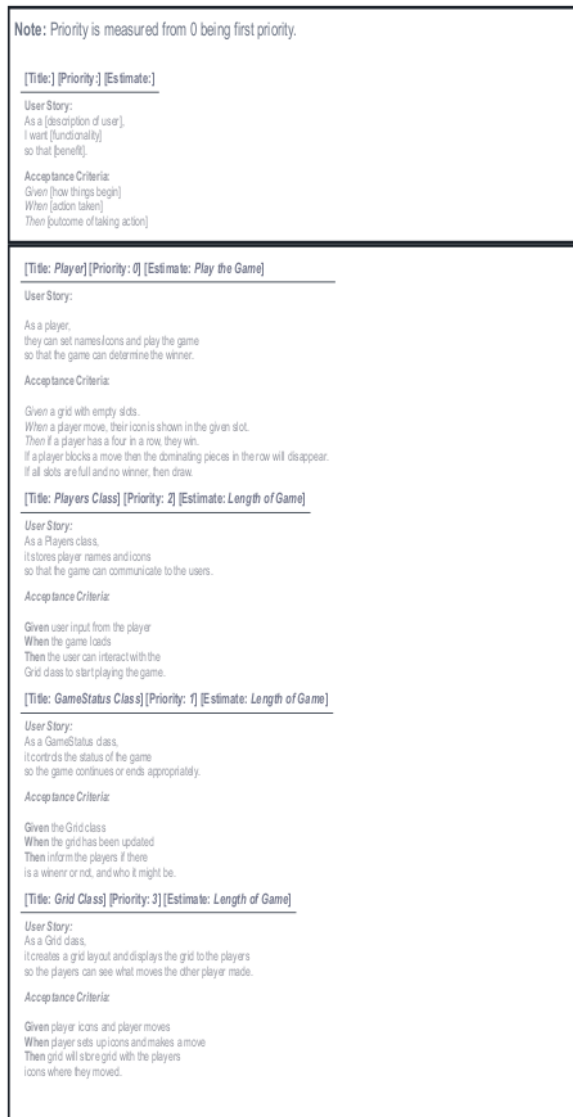


Fig. 18. Figure of [UML Diagram](#) created by Edward Patch.

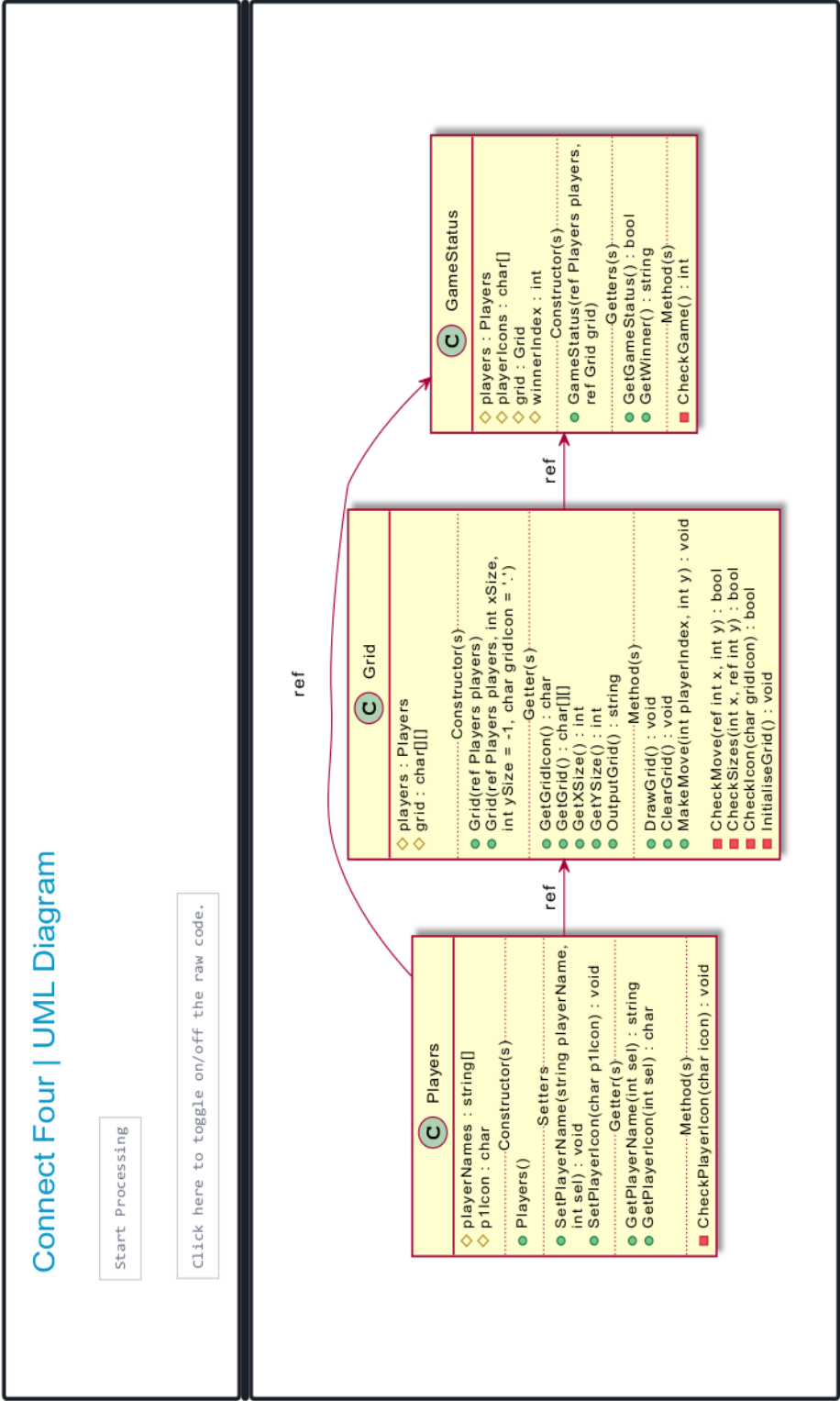


Fig. 19. Figure of UML Diagram created by Edward Patch.

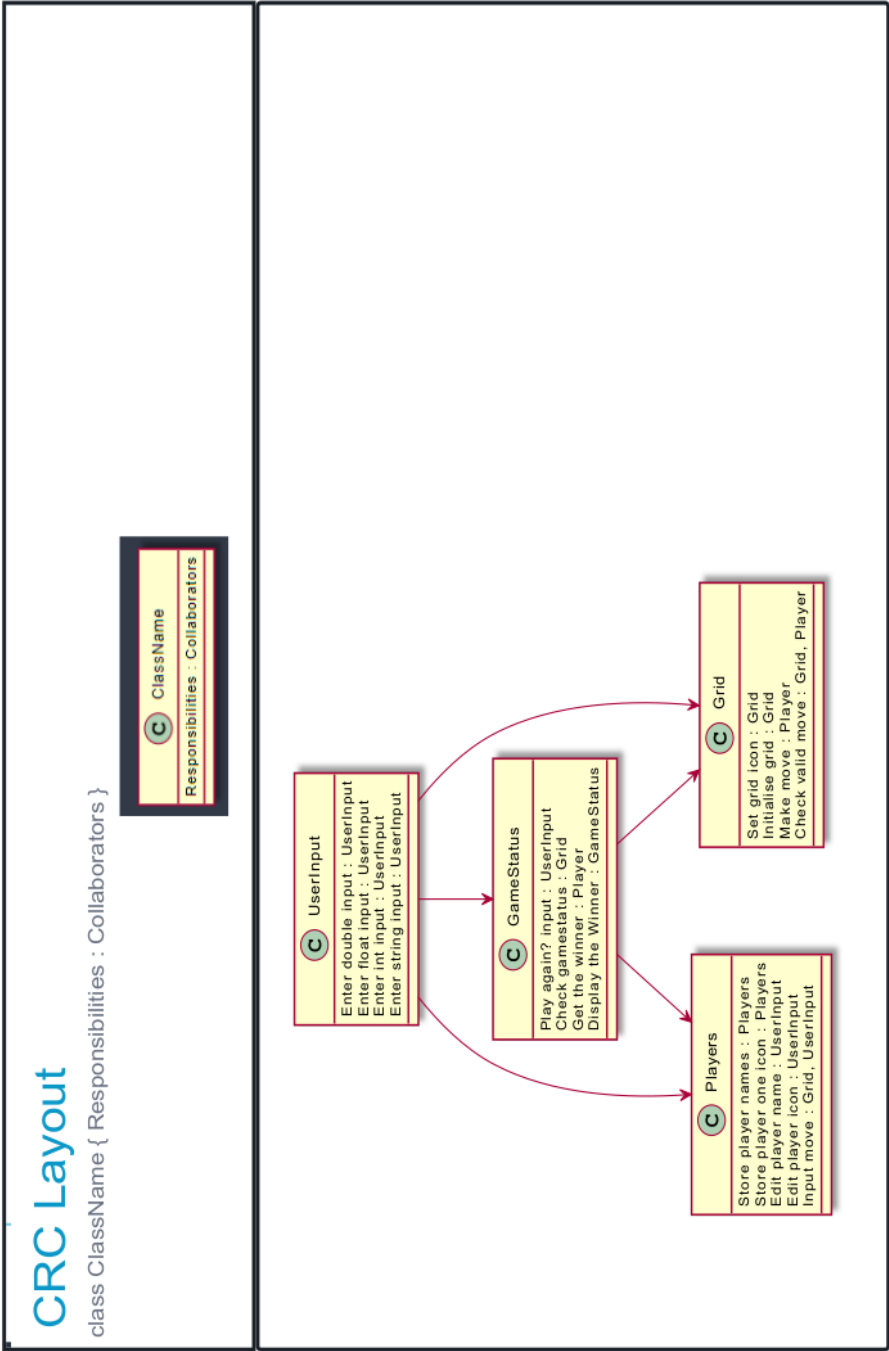


Fig. 20. Figure of CRC Diagram created by Edward Patch.

F. User Sequence Diagram

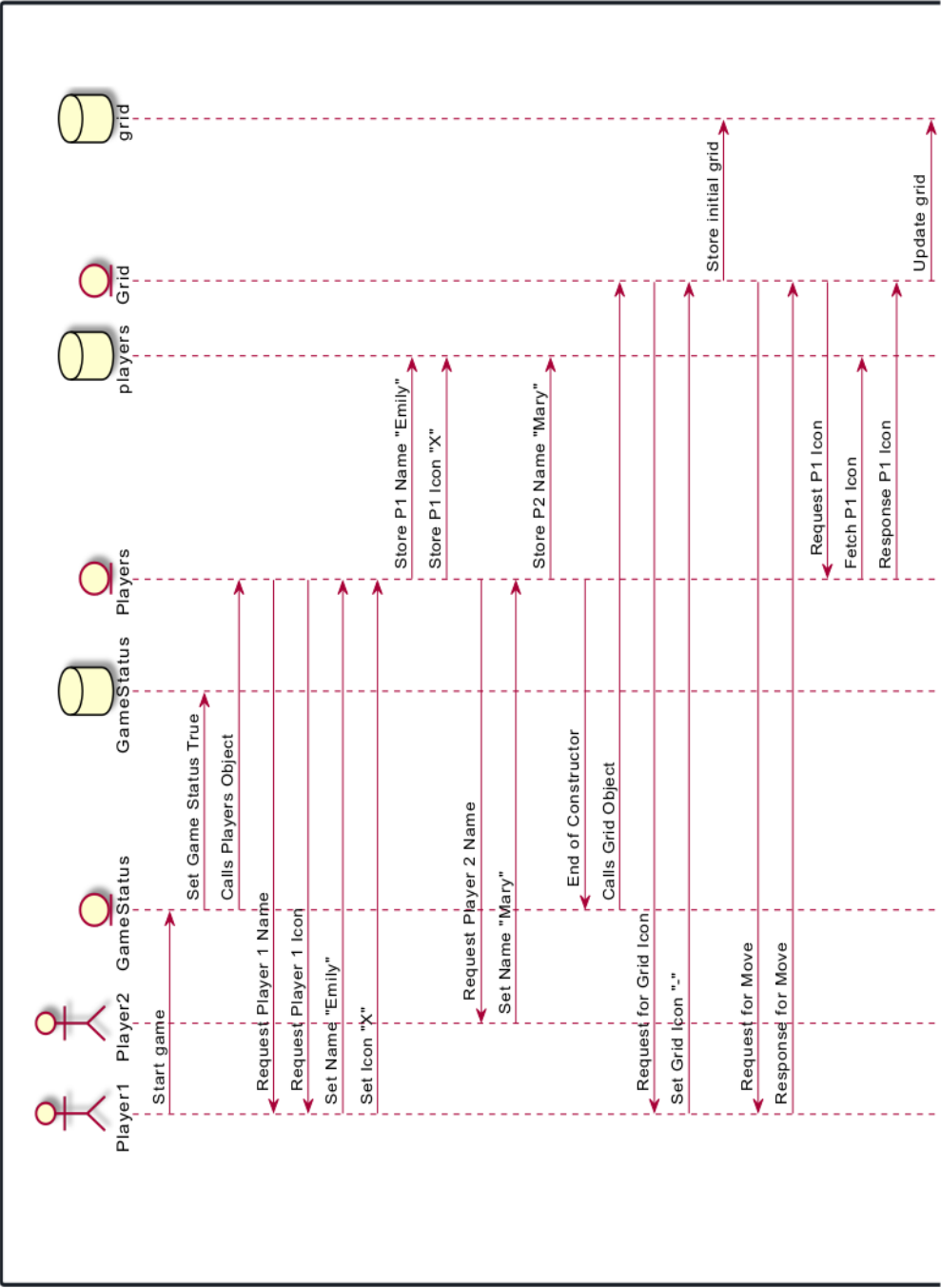


Fig. 21. Figure of User Sequence Diagram created by Edward Patch.

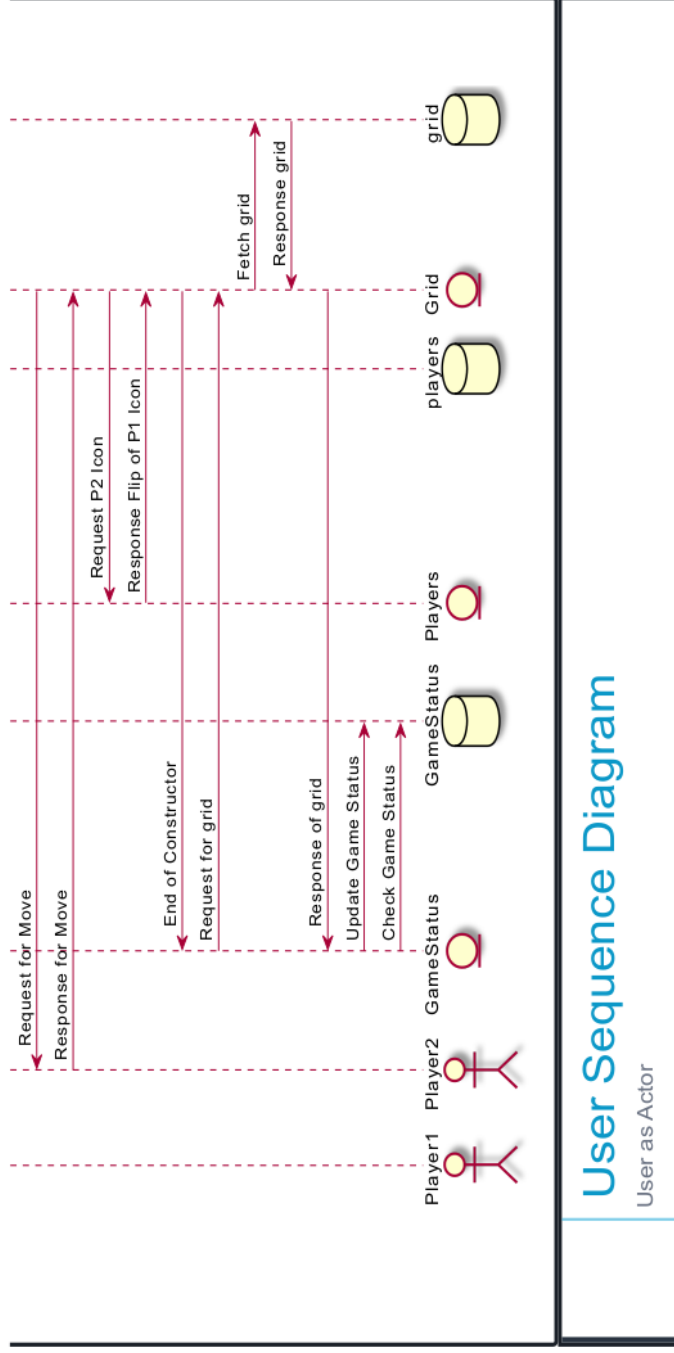


Fig. 22. Figure of [User Sequence Diagram](#) created by Edward Patch.

XXI. REFERENCES

- [1] Milton Bradley Company, "Connect Four." [Online]. Available: <https://www.fgbradleys.com/rules/Connect%20Four.pdf>
- [2] C. McKenzie and TechTarget, "Git vs. GitHub: What is the difference between them?" Nov. 2018. [Online]. Available: <https://www.theserverside.com/video/Git-vs-GitHub-What-is-the-difference-between-them>
- [3] Microsoft, "Assert Class (Microsoft.VisualStudio.TestTools.UnitTesting)." [Online]. Available: <https://docs.microsoft.com/en-us/dotnet/api/microsoft.visualstudio.testtools.unittesting.assert>
- [4] BillWagner, pkulikov, and Microsoft, "Integral numeric types - C# reference," Oct. 2019. [Online]. Available: <https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/builtin-types/integral-numeric-types>
- [5] Product Plan, "User Story Examples in Product Development | Definition and Template." [Online]. Available: <https://www.productplan.com/glossary/user-story/>
- [6] Moqups, "Online Mockup, Wireframe & UI Prototyping Tool · Moqups." [Online]. Available: <https://moqups.com>

Software Engineering Design

GRADEMARK REPORT

FINAL GRADE

80/100

GENERAL COMMENTS

Instructor

Eval design (10): 8

User Stories (20): 16

CRC (20): 17

SW Testing (30): 22

Coding (20): 17

Feedback:

=====

I would liked to have seen a user story map showing how the stories are related to the whole design.

The CRC cards are not really CRC cards, this more of a UML class diagram.

The assignment asked for more evaluation of the design and testing, where you have focused more on the implementation. I like this work, so I am juggling the marks around a a bit to reflect that.

The code needed comments.

The method CheckGame() in

the GameStatus class could have been cleaner (subdivided).

Arguably, display the grid in the Grid class is a responsibility that should be given to a different class...mobe OutputGrid() and DrawGrid() to a new class GridDisplay or equivalent.

Not all features implemented.

PAGE 1

PAGE 2

PAGE 3

PAGE 4

PAGE 5

PAGE 6



Comment 1

You can use similar code to TR code above, essentially you are traversing a straight line with a different gradient

PAGE 7

PAGE 8

PAGE 9

PAGE 10

PAGE 11

PAGE 12

PAGE 13

PAGE 14

PAGE 15

PAGE 16

PAGE 17

PAGE 18

PAGE 19

PAGE 20
