

yr3-sem2-sep-practice

by Jericho Uzzell

Submission date: 02-May-2023 08:43AM (UTC+0100)

Submission ID: 204227303

File name: 12014_Jericho_Uzzell_yr3-sem2-sep-practice_490085_1346695263.docx (641.73K)

Word count: 4017

Character count: 29368

University of Wales Trinity Saint David

School of Computing

Software Engineering Principles and
Practice

ACCA6027

Jericho Uzzell

1702201

20 April 2023



Prifysgol Cymru

Y Drindod Dewi Sant

University of Wales

Trinity Saint David

Context menu

Context menu	2
1. Introduction	3
2. Connect four	3
2.1. What is connect four?	3
2.1.1. Additional rules	4
2.2. User stories	6
2.3.1. What are user stories?	6
2.3.1. User stories	7
2.3. CRC Cards	8
2.3.1. What is a CRC Card?	8
2.3.2. Identified responsibilities	8
2.3.3. Cards	8
2.4. Software testing	9
2.4.1. Purposed software	9
2.4.2. Software result	9
2.4.3. Black box testing	12
2.4.4. White box testing	13
2.5. Version control	14
2.6. Evaluation of design	15
3. References	16
4. Figures	17
5. Appendix	18
5.1. Javascript	18
5.2. HTML	27
5.3. CSS	28
5.4. Version control Git commits	30

1. Introduction

This report will cover software engineering practices by creating a game of connect four, setting rules and parameters and then creating software from these rules. Connect Four is a classic game which normally involves two players, playing on a horizontal grid dropping designated coloured counters within the grid. User stories will be created prior to developing this application, outlining from the persona of an individual what is expected from the system.

CRC Cards will also be created as part of this report outlining the software classes overview and requirements. Once the application has been created, it will be shown within the software section where this report will discuss black-and-white box testing which will be used to test the application produced.

Lastly, this report will state its findings on version control practices and how it was implemented during this report. Using tools like Git and GitHub, it will be used to store the code of the game, which will enable other developers to review edits and suggest edits. With version control, the entire life span of code can be reviewed within seconds, visibly seeing who changed what and when.

2. Connect four

2.1. What is connect four?

According to [3], connect four is a game that is played most commonly by two players, who play on a vertical grid. The players have objects known as counters which are coloured a different colour, most commonly yellow and red counters says [3]. Players take turns to play, and the counters are placed into a free spot in a horizontal space, each opponent can block counters - commonly, there are five horizontal columns with six vertical rows states [3]. To win a normal game of connect four, you must get a vertical, horizontal or diagonal line of four counters from the same player without it being blocked by another opponent's counter.



Figure 2 - Shows the game of connect four box

According to [1], they state that Howard Wexler created connect four back in 1969 when they were in their early thirties. Why did Howard Wexler create connect four? According to [2], Howard first thought that he wanted to invent something, he thought of games and games do not require too elaborate tools or machinery. [2] states that Howard took his time and noticed most games like chess etc, are all horizontal so, can a game be created vertically? After some time, connect four was invented, Howard stated that it is so simple that he does not know why he never thought of it sooner, states [2].



Figure 1 - Howard Wexler in his home studio - by Bud Glick

2.1.1. Additional rules

Within this report, additional rules will be in place. If there is an opponent's counter between another, for example, red has three counters that have a yellow counter that is horizontal or vertical within it; then the counters in-between will be destroyed. If this space is in the vertical axis then then the free space will move all counters down.

Once a counter has been destroyed, the counters that were used can no longer be used to destroy further counters. The below examples show only show two yellow counters, but note that this is for an example whereas the real designs will require three opponents counters between.

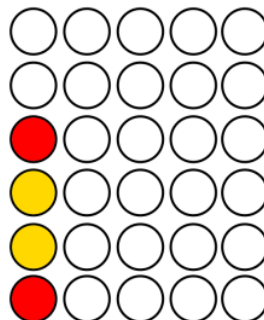


Figure 3 - Shows red counters between yellow counters

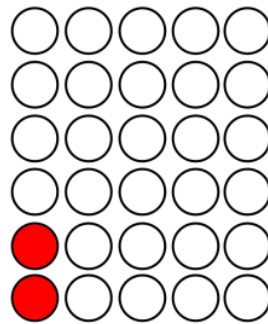


Figure 4 - Shows the counters moving down after destroying the opponent's counters

The below example will show what will happen when an enemy's counter has been destroyed horizontally.

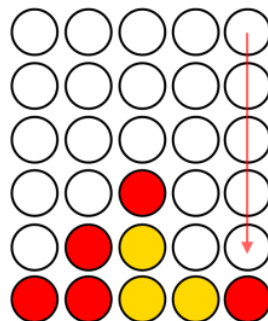


Figure 5 - Shows a red counter being dropped between two yellow counters

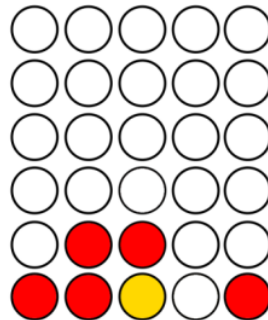


Figure 6 - Shows what happened after the counters are destroyed - counters move down

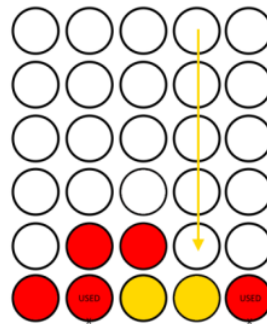


Figure 7 - Shows what happens when another counter is dropped down into the same area - the two red counters cannot destroy them

2.2. User stories

2.3.1. What are user stories?

According to [4], a user story is generally an informal way of explaining features from a perspective of a user when using the software. [4] states that stories keep the focus on the person using the software, from a customer, administrator or another perspective. [4] continues and states that stories are only done when the task outlined can be completed by the user, developers should listen to feedback, note that stories may take time to complete, that the developer should note the personas it is from and subtasks or tasks should be outlined.

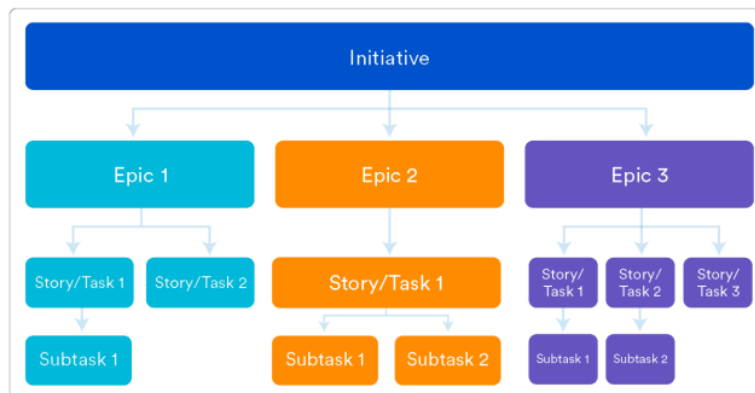


Figure 14 - Epics and Initiatives

[5] states to write a user there is a template but, there are a couple of things to remember, who is it for, what the system expected and why is it important (which is optional). Simply put, it will be who, what and why.

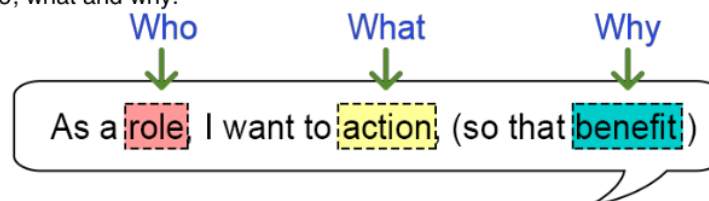


Figure 15 - Three steps to user story

2.3.1. User stories

From following the above examples, the below have been created in reference to the game connect four.

User stories		
As a <type>	I would like to <task>	so that I can <goal>
Player	Player connect four	Play against another player
Player	Select where my counter goes	I can win
Player	Alter my name	I am identifiable
Player	Reset the board	Rematch my opponent
Player	Keep track of who has won	
Player	Clear the board after the round is over	Play again
Player	Change the rules	How many counters are required to win
Player	Alter the size of the board	Have different ways of playing
Player	Alter how many counters are required to be in a row before they are destroyed	
Player	Have the game to run smoothly	I can play
Software Developer	Have the code nested appropriately	I can understand the code faster
Software Developer	Have comments commented appropriately near code	Revist old code if needed

2.3. CRC Cards

2.3.1. What is a CRC Card?

Class, responsibilities, and collaborates also known as CRC are known for object-oriented design according to [6]. The intention behind these is, the most salient classes within the object-oriented system are listed with outlines of what the responsibilities are of each class and its collaborator (collaborators mean the class relies on other classes to work) according to [6].

2.3.2. Identified responsibilities

From reviewing the above, the following have been identified, application, player and board. The application will deal with initialising the board and player, and it will interact with the board when the player does an action such as placing a counter down a column.

The player class will house statistics of wins, who is currently playing, and whether the player won after selecting a column and it will display all information from within itself. Lastly, the board will generate a grid defined by parameters that the user can alter, the board will check whether columns are full, if the board is full, if someone has won and lastly if a counter has met the criteria to be destroyed.

2.3.3. Cards

Below are the CRC cards that have complied with their responsibilities and collaborators listed.

Application

Responsibilities	Collaborators
• Handle user input and game moves • Retrieve game state and board status	• Board object • Player Object • User input (number & mouse click)

Player

Responsibilities	Collaborators
• Show players counters • Track whose turn it is • Has player won after turn • List winners	• Board object

Board

Responsibilities	Collaborators
• Create board • Check if there is space on board and if the column is	• Player object

- | | |
|---|-------|
| <ul style="list-style-type: none">• Display where the counter ends up• Check if special rules have been successful | valid |
|---|-------|

2.4. Software testing

2.4.1. Purposed software

The website that will be created will be done via a web application, utilising HTML 5, Javascript, JQuery and CSS. The plan is to create this website with object-orientated programming being used. Once done, testing will be commenced on this site which will be detailed below.

The design is to be basic and only a concept and, it is to be expected that all features are fulfilled that have been outlined within the user stories and CRC cards.

2.4.2. Software result

As part of this report, software was created and tested using methods such as white box and black box testing.



Figure 8 - shows the starting state of connect four



Figure 9 - shows what happens after player one takes a go



Figure 10 - shows what happens if two opponents' counters are between one another



Figure 11 - shows what happens prior to an opponent's counter being placed



Figure 12 - shows what happens after an opponent's counter is destroyed, counters go darker colour to show they have been used

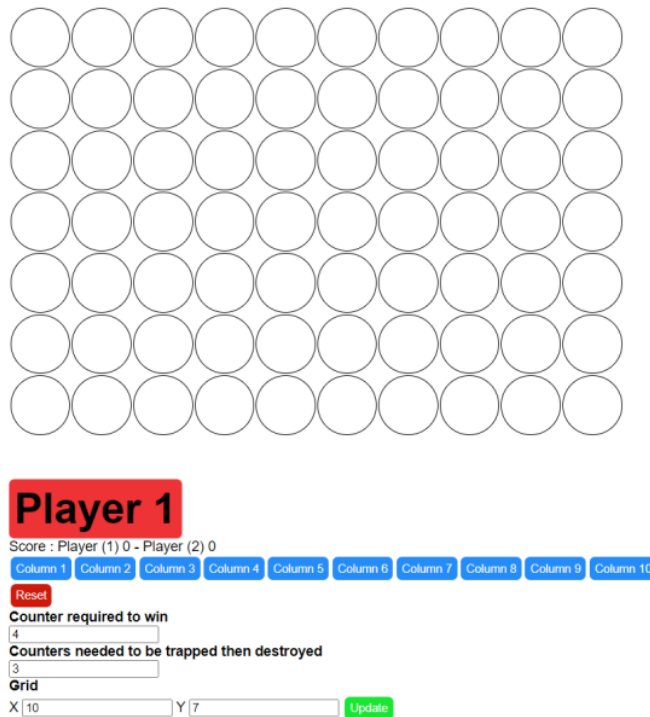


Figure 13 - shows that the grid can be altered

2.4.3. Black box testing

According to [7], black box testing, also known as behavioural testing mainly focuses on requirements and specifications for running applications based on their software - the focus is that the tester would not know what the internal code is like or the structure of key elements. [7] continues and states that there are several types of black box testing, such as equivalence class testing, boundary value testing and decision table testing.

Owing to the fact you should not know what is going on code-wise to be compliant with black box testing, testing the above application may test features such as talking to other users via a chat option, does it repeat the expected behaviour, and whether is it slow. Other tests could be if you were versing the computer, does the computer take too long to make a move, does the computer slow down if it cannot make a decision where the place the counter. These expected outputs and repeatable inputs should be tested.

2.4.4. White box testing

[8] states that white box testing, which is also known as clear, glass box or structural testing evaluates the code and internal structure of an application. [8] continues and says that knowing the internal code of the software, you can perform tests where you can check if they follow your expected operation.

Simply put, all paths within a module must have been exercised at least once, all logic decisions verified and, all loops executed at their set parameters and final parameters says [8].

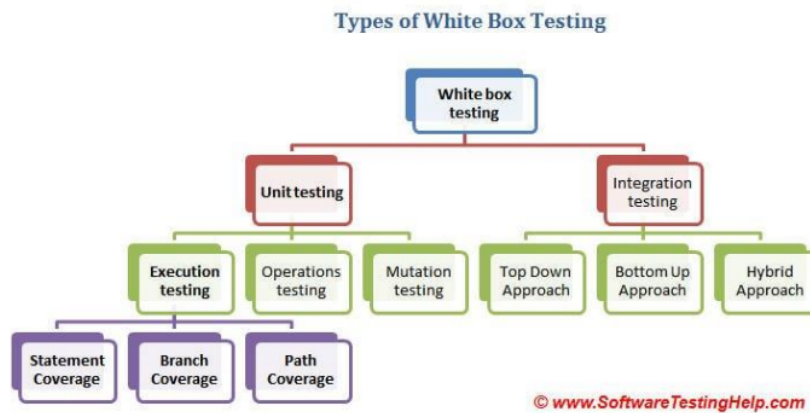


Figure 16 - shows the white box testing structure

Examples of when you may do white box testing could be when dealing with different types of variable types as if you have an if statement and you read data from a database and, the data you read is marked as true but the code in the if statement does not treat it as such, this may result in irregularities.

2.5. Version control

[9] states that version control, which is also known as source control assists with the management of source code and versions of the system which assists developers to manage source code over time. [9] states that version control systems are used in software development environments with Git being the most popular tool. The ability to revert, review, pull changes, push changes, merge commits, stash commits and branch away from the main repository to push later assists a lot with development.

Further to this, having pull requests from developers who may not directly have permission to push to the main branch can be set up, this is beneficial for open-sourced software. An example of what was done with the website produced for this report can be seen below.

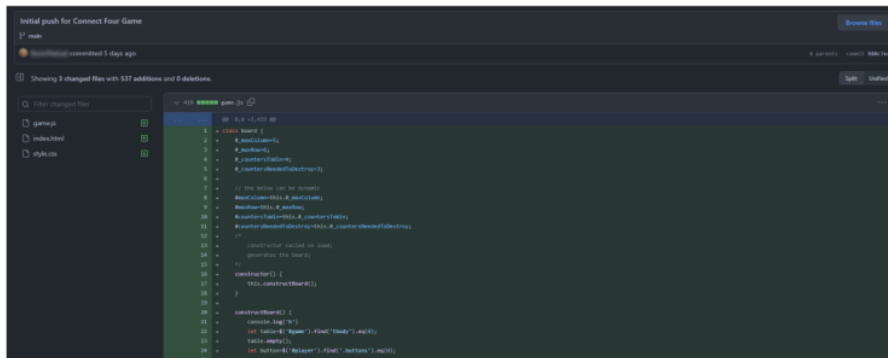


Figure 17 - GitHub showing the initial push of the website

With this in mind, the above push message does not correctly conform to the suggested messages. According to [10], you can name your git commits by the following types:

- Feat - Feature introduced
- Fix - Bug fix
- Chore - Updating dependencies (not fixing/adding features)
- Refactor - Neither fixes a bug nor adds a feature
- Docs - Updates README or other markdown files
- Style - Edits white-space, missing text like semi-colons
- Test - New or correcting previous tests
- Perf - Performance improvements
- CI - Continuous integration related
- Build - Affects build system or external dependencies
- Revert - Reverts a previous commit

[10] states that you can form this sentence for your commit to tracking potential changes in the future, simply put "Type(targeting file): description". With all this in mind, version control is a very beneficial tool that all software developers and engineers should utilise - having not version control method means you cannot track the cycle of code, meaning bugs or other concerns can come up.

It is beneficial as developers can be given feedback on their code if it caused a bug or, praise for code they have done.

2.6. Evaluation of design

Creating user stories and CRC cards is a new thing for me. The fact that you can simplify the processes required, what the user expects from the application, why and what processes there are to test your application. Regarding source control, that is something I currently do a lot of, aside from the commit messages I fully comply with these practices and support them as going between computers and working with a group of coders makes developing simpler.

A weakness that can be noted regarding CRC cards is that on a smaller application, it can be a waste of time - it is only useful for large scaled applications with a wide array of users. It was beneficial to have a CRC card for the board as it outlined links required to the player and what it was responsible for. Most of these were discovered when planning the code out by writing it down on paper which took some time - if you upscale this it can make a project far longer.

Lastly, covering the testing phase of this report, the theory and hypotheses were discussed with the potential area where to investigate - further to this small tests verifying the code actually worked were performed. However, no real testing was done in this area.

3. References

- [1] R.Stellabotte, "Toy Story: Catching Up with Howard Wexler, Inventor of the Classic Game Connect 4", *fordham*, 11 July 2018, [Online.], Available: <https://news.fordham.edu/fordham-magazine/toy-story-catching-up-with-howard-wexler-inventor-of-the-classic-game-connect-4/#:~:text=Toy%20Story%3A%20Catching%20Up%20with,the%20Classic%20Game%20Connect%204>, Accessed 20 April 2023
- [2] howardwexlertoys, "Inventing Connect Four", *howardwexlertoys*, 2023, [Online.], Available: <https://www.howardwexlertoys.com/bio/inventing-connect-four/>, Accessed: 20 April 2023
- [3] hasbro, "Connect 4 Game Instructions, Rules & Activity Guide - Hasbro", *hasbro*, 2023, [Online.], Available: <https://instructions.hasbro.com/en-my/instruction/connect-4-game>, Accessed 20 April 2023
- [4] B.Rehkopf, "User stories with examples and a template", *atlassian*, 2023, [Online.], Available: <https://www.atlassian.com/agile/project-management/user-stories>, Accessed: 21 April 2023
- [5] visual-paradigm, "What is User Story?", *visual-paradigm*, 2023, [Online.], Available: <https://www.visual-paradigm.com/guide/agile-software-development/what-is-user-story/>, Accessed 22 April 2023
- [6] agilealliance, "What are CRC Cards?", *agilealliance*, 2023, [Online.], Available: <https://www.agilealliance.org/glossary/crc-cards/>, Accessed 17 April 2023
- [7] T.Hamilton, "What is BLACK Box Testing? Techniques, Types & Example", *guru99*, 8 April 2023, [Online.], Available: <https://www.guru99.com/black-box-testing.html>, Accessed 25 April 2023
- [8] softwaretestinghelp, "White Box Testing: A Complete Guide With Techniques, Examples, & Tools", *softwaretestinghelp*, 26 March 2023, [Online.], Available: <https://www.softwaretestinghelp.com/white-box-testing-techniques-with-example/>, Accessed 24 April 2023
- [9] atlassian, "What is version control?", *atlassian*, 2023, [Online.], Available: <https://www.atlassian.com/git/tutorials/what-is-version-control>, Accessed 26 April 2023
- [10] N.Pina, "How to Write Better Git Commit Messages – A Step-By-Step Guide", *freecodecamp*, 4 January 2022, [Online.], Available: <https://www.freecodecamp.org/news/how-to-write-better-git-commit-messages/>, Accessed 27 April 2023

4. Figures

[Figure 1] - R.Stellabotte, "Toy Story: Catching Up with Howard Wexler, Inventor of the Classic Game Connect 4", *fordham*, 11 July 2018, [Online.], Available: <https://news.fordham.edu/fordham-magazine/toy-story-catching-up-with-howard-wexler-inventor-of-the-classic-game-connect-4/#:~:text=Toy%20Story%3A%20Catching%20Up%20with,the%20Classic%20Game%20Connect%204>, Accessed 20 April 2023

[Figure 2] - howardwexlertoys, "Inventing Connect Four", *howardwexlertoys*, 2023, [Online.], Available: <https://www.howardwexlertoys.com/bio/inventing-connect-four/>, Accessed: 20 April 2023

[Figure 3] - Shows red counters between yellow counters

[Figure 4] - Shows the counters moving down after destroying the opponent's counters

[Figure 5] - Shows a red counter being dropped between two yellow counters

[Figure 6] - Shows what happened after the counters are destroyed - counters move down

[Figure 7] - Shows what happens when another counter is dropped down into the same area - the two red counters cannot destroy them

[Figure 8] - Shows the starting state of connect four

[Figure 9] - Shows what happens after player one takes a go

[Figure 10] - Shows what happens if two opponents counters are between another

[Figure 11] - Shows what happens prior to an opponents counter being placed

[Figure 12] - Shows what happens after an opponents counter is destroyed, counters go darker colour to show they have been used

[Figure 13] - Show that the grid can be altered

[Figure 14] - B.Rehkopf, "User stories with examples and a template", *atlassian*, 2023, [Online.], Available: <https://www.atlassian.com/agile/project-management/user-stories>, Accessed: 21 April 2023

[Figure 15] - visual-paradigm, "What is User Story?", *visual-paradigm*, 2023, [Online.], Available: <https://www.visual-paradigm.com/guide/agile-software-development/what-is-user-story/>, Accessed 22 April 2023

[Figure 16] - softwaretestinghelp, "White Box Testing: A Complete Guide With Techniques, Examples, & Tools", *softwaretestinghelp*, 26 March 2023, [Online.], Available: <https://www.softwaretestinghelp.com/white-box-testing-techniques-with-example/>, Accessed 24 April 2023

5. Appendix

5.1. Javascript

```
class board {
  #_maxColumn=5;
  #_maxRow=6;
  #_countersToWin=4;
  #_countersNeededToDestroy=3;

  // the below can be dynamic
  #maxColumn=this.#_maxColumn;
  #maxRow=this.#_maxRow;
  #countersToWin=this.#_countersToWin;
  #countersNeededToDestroy=this.#_countersNeededToDestroy;
  /*
    constructor called on load;
    generates the board;
  */
  constructor() {
    this.constructBoard();
  }

  constructBoard() {
    console.log('h')
    let table=$('#game').find('tbody').eq(0);
    table.empty();
    let button=$('#player').find('.buttons').eq(0);
    button.empty();
    let out="";
    let column=0;
    let row=0;
    for(row=0;row<this.#maxRow;row++) {
      out="";
      for(column=0;column<this.#maxColumn;column++) {
        out+='<td></td>';
      }
      table.append('<tr>'+out+'</tr>');
    }
    for(column=0;column<this.#maxColumn;column++) {
      button.append('<button      class="counter"      column="'+column+'>Column
'+(column+1)+'</button>');
    }
    $('#counters_to_win').val(this.#countersToWin);
    $('#counters_to_destroy').val(this.#countersNeededToDestroy);
    $('#grid_x').val(this.#maxColumn);
    $('#grid_y').val(this.#maxRow);
  }
}
```

```
    $('#other_buttons').show();
}

reset() {
    this.#maxColumn=this.#_maxColumn;
    this.#maxRow=this.#_maxRow;
    this.#countersToWin=this.#_countersToWin;
    this.#countersNeededToDestroy=this.#_countersNeededToDestroy;
}
/*
    updatebaord called by user to update board
*/
updateBoard() {
    let grid_x=parseInt($('#grid_x').val());
    let grid_y=parseInt($('#grid_y').val());
    let towin=parseInt($('#counters_to_win').val());
    let todestroy=parseInt($('#counters_to_destroy').val());
    if(grid_x<0||grid_x>15||typeof(grid_y)!='number') return 2;
    if(grid_y<0||grid_y>15||typeof(grid_y)!='number') return 3;
    if(towin<0||towin>15||typeof(towin)!='number') return 4;
    if(todestroy<0||todestroy>15||typeof(todestroy)!='number') return 5;

    this.#maxColumn=parseInt(grid_x);
    this.#maxRow=parseInt(grid_y);
    this.#countersToWin=parseInt(towin);
    this.#countersNeededToDestroy=parseInt(todestroy);
    console.log(this.#maxColumn)
    return true;
}

/*
    freespaces called at every round checking if there are free places;
    returns bool;
*/
spacesFree() {
    let table=$('#game').find('tbody').eq(0);
    let col=null;

    for(col=0;col<this.#maxColumn;col++) {
        if(table.find('tr').eq(0).find('td').eq(col).attr('class')==")
        table.find('tr').eq(0).find('td').eq(col).attr('class')==undefined) return true;
        }
        return false;
    }
}
/*
    !PRIVATE!
    validzone called when player presses a column - to place counter;
```

```
        returns bool;
    */
    #validZone(column=null) {
        if(column==null) {
            console.error('Column was null!');
            return false;
        }
        if(typeof(column)!='number') {
            console.error('Column was not a number!');
            return false;
        }
        if(column>this.#maxColumn) {
            console.error('Max column is '+this.#maxColumn+'!');
            return false;
        }
        let table=$( '#game' ).find('tbody').eq(0);
        if(table.find('tr').eq(0).find('td').eq(column).length==0) {
            console.error('Could not find column '+column+'!');
            return false;
        }
        table=table.find('tr').eq(0).find('td').eq(column);

        // console.log(table.attr('class'));

        if(table.attr('class')=='' || table.attr('class')==undefined) return true;
        console.error('Column full!');
        return false;
    }

    /*
    !PRIVATE!
    dropcounter called when validzone is true
    */
    #dropCounter(column=null,counter=null) {
        let table=$( '#game' ).find('tbody').eq(0);
        let row=0;
        let tmpTable=null;
        let tmprow=null;
        let force_row=null;

        for(row=0;row<this.#maxRow;row++) {
            tmprow=row+1;
            tmpTable=table.find('tr').eq(row).find('td').eq(column);
            if(tmpTable.attr('class')=='' || tmpTable.attr('class')==undefined) {
                //current column free - check if next one is aswell - if so loop;
                if(tmprow<=this.#maxRow) {
```

```
        if(table.find('tr').eq(tmprow).find('td').eq(column).attr('class')!="&&
table.find('tr').eq(tmprow).find('td').eq(column).attr('class')!=undefined) {
            force_row=tmprow - 1;
        }
    }
}

if(force_row==null) force_row=this.#maxRow - 1;

table.find('tr').eq(force_row).find('td').eq(column).addClass(counter);
// console.log(force_row);
// console.log(column);
// console.log(counter);
return true;
}

/*
!PRIVATE!
move animates
*/
move(column,ply) {
    if(!this.#validZone(column)) return false;
    this.#dropCounter(column,(ply.whos_turn()==0?'red':'yellow'));
}

connectFour(ply) {
    let current_ply=ply.whos_turn();
    let colour=(current_ply==0?'red':'yellow');
    let table=$('#game').find('tbody').eq(0);
    let tmptable=null;

    let row=null;
    let column=null;
    let valid_hor=0;
    let valid_vert=0;
    // let valid_diag=0;

    // console.warn(current_ply)
    // console.warn(colour)

    /* checks horizontal */
    for(row=0;row<this.#maxRow;row++) {
        valid_hor=0;
        tmptable=table.find('tr').eq(row);
        for(column=0;column<this.#maxColumn;column++) {
            // console.log(tmptable.find('td').eq(column).hasClass(colour));
```

```
        if(tmptable.find('td').eq(column).hasClass(colour)) valid_hor++;
        else valid_hor=0;

        if(valid_hor>=this.#countersToWin) return true;
    }
}

for(column=0;column<this.#maxColumn;column++) {
    valid_vert=0;
    // console.log('-----');
    for(row=0;row<this.#maxRow;row++) {
        tmptable=table.find('tr').eq(row).find('td').eq(column);
        // console.log('col='+column+' row='+row+' class='+tmptable.hasClass(colour));
        // console.log(tmptable.find('td').eq(row).hasClass(colour));
        if(tmptable.hasClass(colour)) valid_vert++;
        else valid_vert=0;

        if(valid_vert>=this.#countersToWin) return true;
    }
}
return false;
}

specialConnectFour(ply) {
    let current_ply=ply.whos_turn();
    let colour=(current_ply==0?'red':'yellow');
    let table=$('#game').find('tbody').eq(0);
    let tmptable=null;

    let row=null;
    let column=null;
    let my_counter_hor=false;
    let valid_hor=0;
    let hor_local_stores={};
    let hor_local_stores_mine={};
    let valid_vert=0;
    let my_counter_vert=false;
    let vert_local_stores={};
    let vert_local_stores_mine={};

    let re_gig_board=false;

    /* checks horizontal */
    for(row=0;row<this.#maxRow;row++) {
        valid_hor=0;
        hor_local_stores={};
        tmptable=table.find('tr').eq(row);
```

```
my_counter_hor=false;

for(column=0;column<this.#maxColumn;column++) {
  // console.log(tmptable.find('td').eq(column).hasClass(colour));
  if(tmptable.find('td').eq(column).hasClass(colour)                                &&
tmptable.find('td').eq(column).attr('cannot_destroy_others')==undefined) {
    valid_hor=0;
    hor_local_stores_mine[column]=tmptable.find('td').eq(column);
    my_counter_hor=true;
  }else if(my_counter_hor && (tmptable.find('td').eq(column).attr('class')!=" &&
!tmptable.find('td').eq(column).hasClass(colour)                                &&
tmptable.find('td').eq(column).attr('cannot_destroy_others')!=undefined)) {
    hor_local_stores[column]=tmptable.find('td').eq(column);
    valid_hor++;
  }else valid_hor=0;

  if(valid_hor>=this.#countersNeededToDestroy) {
    if(tmptable.find('td').eq(column + 1).hasClass(colour)                                &&
tmptable.find('td').eq(column + 1).attr('cannot_destroy_others')==undefined) {
      hor_local_stores_mine[column]=tmptable.find('td').eq(column + 1);
      $.each(hor_local_stores,function(k,i){
        $(this).removeClass('red yellow');
      });
      $.each(hor_local_stores_mine,function(k,i){
        $(this).attr('cannot_destroy_others','true')
      });
      re_gig_board=true;
      break;
    }
  }
}
if(re_gig_board) {
  console.log('regig break')
  break;
}
}

if(!re_gig_board) {
  for(column=0;column<this.#maxColumn;column++) {
    valid_vert=0;
    vert_local_stores={};
    my_counter_vert=false;

    // console.log('-----');
    for(row=0;row<this.#maxRow;row++) {
      tmptable=table.find('tr').eq(row).find('td').eq(column);
      // console.log('col='+column+' row='+row+' class='+tmptable.hasClass(colour));
```

```
        // console.log(tmptable.find('td').eq(row).hasClass(colour));
        if(tmptable.hasClass(colour)                                &&
tmptable.attr('cannot_destroy_others')==undefined) {
            vert_local_stores_mine[row]=tmptable;
            valid_vert=0;
            my_counter_vert=true;
        }else if(my_counter_vert) {
            vert_local_stores[row]=tmptable;
            valid_vert++;
        }else valid_vert=0;

        if(valid_vert>=this.#countersNeededToDestroy) {
            if(table.find('tr').eq(row + 1).find('td').eq(column).hasClass(colour) &&
table.find('tr').eq(row + 1).find('td').eq(column).attr('cannot_destroy_others')==undefined) {
                vert_local_stores_mine[row + 1]=table.find('tr').eq(row +
1).find('td').eq(column);
                $.each(vert_local_stores,function(k,i){
                    $(this).removeClass('red yellow');
                });
                $.each(vert_local_stores_mine,function(k,i){
                    $(this).attr('cannot_destroy_others','true')
                });
                re_gig_board=true;
                break;
            }
        }
    }
    if(re_gig_board) {
        console.log('regig break')
        break;
    }
}

// console.log('regiging '+re_gig_board)
if(!re_gig_board) return false;

let tmptable_=null;

for(row=this.#maxRow-1;row>=0;row--) {
    tmptable=table.find('tr').eq(row);
    // tmptable.find('td').removeClass('red yellow').addClass('dev')

    // console.log('row='+row);

    for(column=0;column<this.#maxColumn;column++) {
```

```
        // console.log('>'+tmptable.find('td').eq(column).attr('class')+' '+column+'
'+this.#maxColumn)
        if((tmptable.find('td').eq(column).attr('class')==undefined) ||
tmptable.find('td').eq(column).attr('class')==") && row>0) {
            tmptable_=table.find('tr').eq(row - 1);
            // console.log('--');
            // console.log(tmptable_.find('td').eq(column));
            // console.log(tmptable_.find('td').eq(column).attr('class'))
            if((tmptable_.find('td').eq(column).attr('class')!=undefined) &&
tmptable_.find('td').eq(column).attr('class')!=")) {
                // console.log('moving')

tmptable.find('td').eq(column).attr('class',tmptable_.find('td').eq(column).attr('class'));
            if(tmptable_.find('td').eq(column).attr('cannot_destroy_others')!=undefined)
tmptable.find('td').eq(column).attr('cannot_destroy_others','cannot_destroy_others');

                tmptable_.find('td').eq(column).attr('class','');
                tmptable_.find('td').eq(column).removeAttr('cannot_destroy_others');
            }
        }
    }
    // if(row==0) break;
}

}

}

class player {
    #current_player=0;
    #player1_wins=0;
    #player2_wins=0;

    constructor() {
        this.details();
    }

    whos_turn() {
        return this.#current_player;
    }
    winners() {
        return this.#player1_wins+ ' '+this.#player2_wins;
    }

    details() {
        $('#player').find('.scores').text('Score : Player (1) '+this.#player1_wins+' - Player (2)
'+this.#player2_wins);
    }
}
```

```
    $('#player').find('.current_player').text('Player      '+ (this.#current_player      +
1)).attr('player',this.#current_player);
  }

  reset() {
    this.#current_player=0;
    this.#player1_wins=0;
    this.#player2_wins=0;
    this.details();
  }

  play(oop,column) {
    if(!oop.spacesFree()) {
      return false;
    }
    this.details();
    oop.move(column,this);

    //check if a connect four was made
    // console.log(oop.connectFour(this));
    // console.log('regiging board');
    if(oop.specialConnectFour(this)) {
      console.log('regiging board');
    }

    if(oop.connectFour(this)) {
      if(this.#current_player==0) this.#player1_wins=this.#player1_wins+1;
      else this.#player2_wins=this.#player2_wins+1;
      // console.log('YOU WON')
      oop.constructBoard();
    }else{
      this.#current_player=(this.#current_player==0?1:0);
    }
    this.details();
    return true;
  }
}

const boards = new board();
const players = new player();

// $('#game table tbody').find('tr').eq(5).find('td').eq(0).addClass('red')
// $('#game table tbody').find('tr').eq(5).find('td').eq(1).addClass('yellow')
// $('#game table tbody').find('tr').eq(4).find('td').eq(1).addClass('red')
```

```
$(document).on('click','button.counter',function() {  
    let column=$(this).attr('column');  
    if(column==undefined||column==null) return;  
    players.play(boards,parseInt(column))  
});
```

```
$(document).on('click','button.reset',function() {  
    boards.reset();  
    boards.constructBoard();  
    players.reset();  
});
```

```
$(document).on('click','button.update',function() {  
    boards.updateBoard();  
    boards.constructBoard();  
    players.reset();  
    // boards = new board();  
    // players = new player();  
});
```

5.2. HTML

```
<!DOCTYPE html>  
<html>  
  <head>  
    <meta charset="utf-8">  
    <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">  
    <meta http-equiv="X-UA-Compatible" content="IE=edge">  
    <meta name="viewport" content="width=device-width, initial-scale=1">  
    <meta name="description" content="Connect four rebarded.">  
    <meta name="author" content="Created by Jericho Uzzell">  
  
    <title>Connect 4</title>  
    <link rel="stylesheet" href="style.css"></link>  
  </head>  
  <body>  
    <div id="game">  
      <table>  
        <tbody></tbody>  
      </table>  
    </div>  
    <div id="player">  
      <h3 class="current_player"></h3>  
      <p class="scores"></p>  
      <p class="buttons"></p>  
      <div class="other_buttons" style="display: none;">  
        <button class="reset">Reset</button>  
        <h4>Counter required to win</h4>  
        <input id="counters_to_win" type="input" />  
      </div>  
    </div>  
  </body>  
</html>
```

```
<h4>Counters needed to be trapped then destroyed</h4>
<input id="counters_to_destroy" type="input" />
<h4>Grid</h4>
X <input id="grid_x" type="input" />
Y <input id="grid_y" type="input" />
<button class="update">Update</button>
</div>
</div>
</body>
<script src="https://code.jquery.com/jquery-3.6.4.min.js" integrity="sha256-
oP6HI9z1XaZNBxJURtCoUT5SUnxFr8s3BzRI+cbzUq8="
crossorigin="anonymous"></script>
<script src="game.js"></script>
</html>
```

5.3. CSS

```
:root{
  --counter-reds:rgb(240, 52, 52);
  --counter-destroyed-reds:rgb(100, 1, 1);
  --counter-yellows:rgb(228, 228, 93);
  --counter-destroyed-yellows:rgb(90, 90, 0);
  --counter-dev:rgb(50, 15, 206);
  --cell-size:2em;
}
#game table tbody tr td{
  border:1px solid #000;
  height:var(--cell-size);
  min-height:var(--cell-size);
  max-height:var(--cell-size);
  min-width:var(--cell-size);
  max-width:var(--cell-size);
  width:var(--cell-size);
  font-size: var(--cell-size);
  text-align: center;
  border-radius: 70%;
}
#game table tbody tr td.red{
  background-color: var(--counter-reds);
}
#game table tbody tr td.red[cannot_destroy_others="true"]{
  background-color: var(--counter-destroyed-reds);
}
#game table tbody tr td.yellow{
  background-color: var(--counter-yellows);
}
#game table tbody tr td.yellow[cannot_destroy_others="true"]{
  background-color: var(--counter-destroyed-reds);
}
}
```

```
#game table tbody tr td.dev{
  background-color: var(--counter-dev);
}

#player{
  margin-top: 3em;
}
button {
  border:0px;
  padding:6px;
  border-radius:6px;
  margin:2px;
  cursor: pointer;
}
button.counter{
  background-color: rgb(19, 142, 250);
  color:#fff;
}
button.reset{
  background-color: rgb(206, 27, 4);
  color:#fff;
}
button.update{
  background-color: rgb(8, 228, 8);
  color:#fff;
  cursor: pointer;
}
h1,h2,h3,h4,h5,p{
  margin:0px;
}

#player .current_player[player="0"] {
  background-color: var(--counter-reds);
}
#player .current_player[player="1"] {
  background-color: var(--counter-yellows);
}
#player .current_player {
  font-size: 3.0em;
  max-width: max-content;
  border-radius: 6px;
  padding:6px;
}

*{
  font-family: arial;
}
```

Jericho Uzzell (1702201) – Software Engineering Principles and Practice - Sem 2 - 20/04/23
- UWTSD

Jericho Uzzell (1702201) – Software Engineering Principles and Practice - Sem 2 - 20/04/23
- UWTSD

5.4. Version control Git commits

```
cd /uni_projects/sep
git init -b sep
nano test.txt
git add .
git commit -m "Init"
--
git push origin sep
--
```

```
Jericho@Jericho-PC MINGW64 ~
$ cd F:/Server/Uni/test
```

```
Jericho@Jericho-PC MINGW64 /f/Server/Uni/test
$ git init -b sep2
Initialized empty Git repository in F:/Server/Uni/test/.git/
```

```
Jericho@Jericho-PC MINGW64 /f/Server/Uni/test (sep2)
$ git add .
```

```
Jericho@Jericho-PC MINGW64 /f/Server/Uni/test (sep2)
$ nano test.txt
```

```
Jericho@Jericho-PC MINGW64 /f/Server/Uni/test (sep2)
$ git commit -m "Init"
On branch sep2
```

```
Jericho@Jericho-PC MINGW64 /f/Server/Uni/test (sep2)
$ git commit -m "Init"
[sep2 (root-commit) 143f8c7] Init
1 file changed, 0 insertions(+), 0 deletions(-)
create mode 100644 test.txt
```

FINAL GRADE

67 / 100

GENERAL COMMENTS

Instructor

Eval of Design (20):13

User Stories (20): 13

CRC (20):12

SW Testing (20):11

VC (20):13

Coding: 5

Feedback:

=====

You needed to provide more evaluation of the design process and some discussion of the user stories/CRC cards you produced.

Needed a bit more analysis of whit/black box testing and the benefits/drawbacks.

You also produced some code which I will give you some additional marks for creating.

Version control benefits discussed quite well and basic GIT operation demonstrated.

You were not asked to develop any code but you have so that is a bonus. Nice to see you converting design into code. I will give you some extra marks for this.

PAGE 2

PAGE 3

PAGE 4

PAGE 5

PAGE 6

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

PAGE 21

PAGE 22

PAGE 23

PAGE 24

PAGE 25

PAGE 26

PAGE 27

