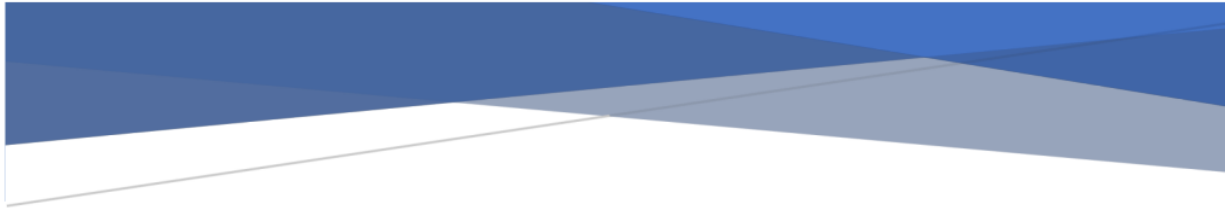


SEPT Assignment

by Rhys David Whatty

FILE	17862_RHYS_DAVID_WHATTY_SEPT_ASSIGNMENT_339836_1640708663.D OCX (197.11K)		
TIME SUBMITTED	21-APR-2021 03:04PM (UTC+0100)	WORD COUNT	2991
SUBMISSION ID	150016562	CHARACTER COUNT	11920



SOFTWARE ENGINEERING DESIGN

ACCA5030 – Software Engineering Principles and
Testing

Rhys Whatty
1900045

Contents

Design Evaluation..... 2

User Feedback input 4

 User Stories 4

 User Story Map 5

 Discussion..... 6

CRC Cards 7

 Discussion..... 8

Software Testing 9

 Black Box and White Box Testing 9

 Testing the code..... 10

Appendices..... 13

 Appendix A: Design Discussion 14

 Appendix B: Software Code 15

Bibliography 13

Introduction

When developing software for a user, it is important to consider what it is the user wants out of the software, and to ensure that when what you have created is completed, that the project does meet all of these requirements. To ensure this, planning is done before beginning the project into what the user requirements are and being able to visualise them in a way that makes the development process easier.

Prior to beginning this project, a group discussion was held in which was discussed the main points that the code needed to achieve to function. The points from this discussion can be found in Appendix A. Using the information, the rest of the planning was completed, and the code was produced.

Design Evaluation

Based on the design that was produced, there are some notable strengths that will come to the code. Using a while command with a Boolean variable means that the game can run in an infinite loop until the Boolean value is changed, which can be done once a win condition has been met. The win conditions will be checked on each tile of the board at the end of every round, and while this may not be an efficient method, it greatly reduces the chance of the code not detecting a win when there has been a 4-in-a-row. The way the win condition is designed to check means it can be similarly replicated to check for an example of a trapped row of three.

User Feedback input

User Stories

- As a player, I want to be able to view the rules of the game so that new players can understand how to play the game.
- As a player, I want to be able to play it as a 2-player game so that I can play with my friends.
- As a player, I want the game to alternate players each turn so that it is fair.
- As a player, I want the game to be able to detect for a win condition of a 4 in a row in any direction so that the game ends at the right time.
- As a player, I want the game to be let players trap other player's tokens and delete them so that there is an element of added difficulty.
- As a player, I want the game to continue until a winner is declared so that the game does not end prematurely without a victor.
- As a player, I want to be able to see a board on-screen so that I can tell what tokens have been placed where.



User Story Map

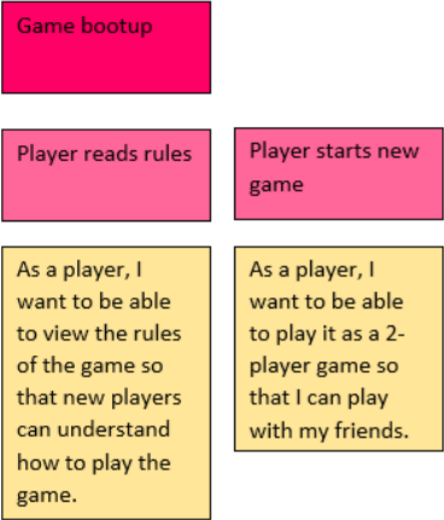


Fig.1

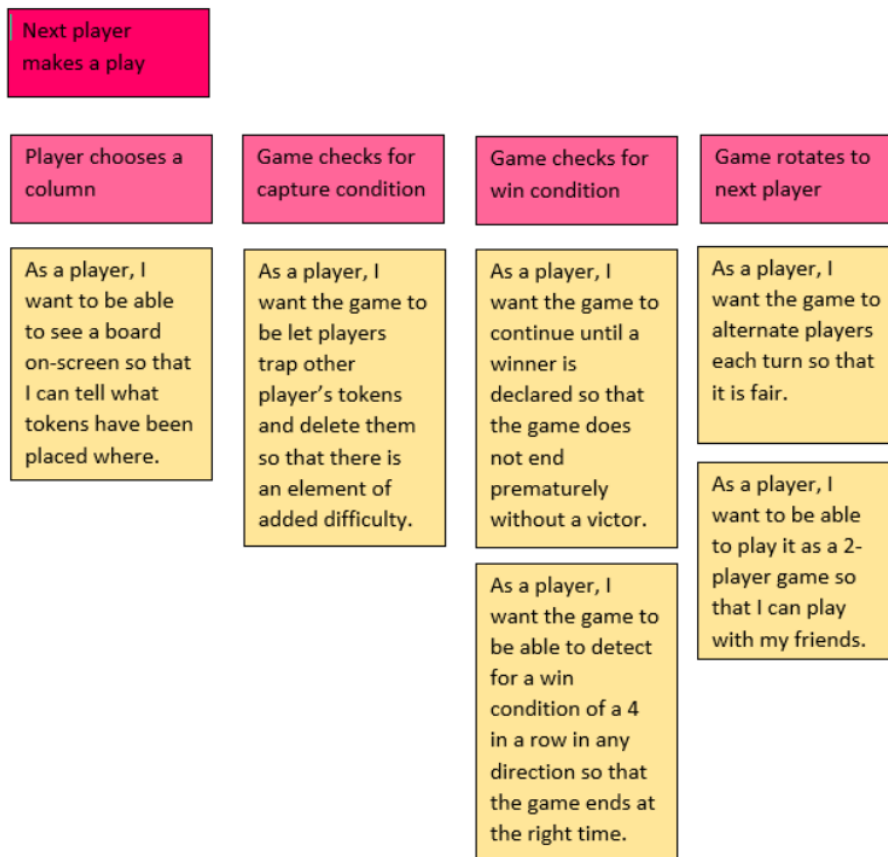


Fig.2

Discussion

The user stories, as shown on the story map, each play a role how the game will play out, with each major step of the game featuring at least one requirement from the user stories. The stories sufficiently cover the gameplay of the Connect-4 game, including the added rules from the classic play, and also provide reasoning behind each of the requirements from the perspective of the software user.

CRC Cards

Class:Player

Responsibilities:

- Play a token
- Aim to get a 4-in-a-row

Collaborators:

- Board

Player:P1

Responsibilities:

- Play first

Collaborators:

- Token:P1Token

Player:P2

Responsibilities:

- Play second

Collaborators:

- Token:P2Token

Class:Board

Responsibilities:

- Store tokens
- Check for/declare win condition
- Check for/declare capture condition

Collaborators:

- Token
- Player

Class:Token
Responsibilities: • Play on board
Collaborators: • Board

Token:P1Token
Responsibilities: • Trap P2 tokens
Collaborators: • Player:P1

Token:P2Token
Responsibilities: • Trap P1 tokens
Collaborators: • Player:P2Tokens

Discussion

The CRC cards take advantage of having a parent class for Tokens and Players, and then for each have classes for the first player and the second player. This means that the two players can exist to combat one another while not duplicating the base code for what their role is due to them sharing the parent class for the player and the token. Most of the functions of the game are controlled by the board as that will be where the state of the game will be stored, meaning it will be easier to check for any conditions that would affect the game. Overall, they well reflect the criteria that was mentioned in the User stories and sufficiently allow the code to be easily implemented and tested.

Software Testing

Black Box and White Box Testing

When testing any form of code, it will be split into one of two types of testing, white box testing and black box testing. The difference is who tests the software, with white box testing being performed by people linked to the creation of the code, while black box testing is performed by people who were not involved in the creation of the code. Both types of tests are beneficial in their own way, as they take different approaches to checking the functionality of the software, with White box testing more closely linked to the inner workings of the software and monitoring the actual code directly, while Black box testing focuses on the behaviour of the software as a whole. [1]

In gaming, black box testing is often performed in the form of an Open Beta version of a game, in which the development team will release an unfinished version of the game prior to its official release to select players (often at the cost of a pre-order of the game) to allow them to search for any playability issues to iron out before the game releases generally to the public.

Testing the code

Below is a test table with which I tested the software that was created (see Appendix B).

Test Number/title	Description	Pre-conditions	Inputs	Expected Outcome	Actual Outcome	Works?
1 – Display rules	Game will display the rules from the home screen when prompted	None	Entering “2” when prompted	Code displays Connect-4 rules in text form	Code prints out the rules of the game	Yes
2 – Start new game	Game will begin with a fresh board	None	Entering “1” when prompted	Code generates a fresh board	Fresh board is generated for start of play	Yes
3 – Player 1 plays a token	Game registers the play of player 1	Game begun, player 1’s turn	Player enters a column number	Code places an X in the correct column	Code places the X at the highest available point in the selected column	Yes
4 – Player 2 plays a token	Game registers the play of player 2	Game begun, player 2’s turn	Player enters a column number	Code places an O in the correct column	Code places the O at the highest available point in the selected column	Yes
5 – Player rotation is functional	Game rotates between player 1 and 2 at the end of each turn	A turn is completed	Player enters their play	Game rotates back to the other player	Game switches from P1 to P2, and P2 to P1	Yes
6 – Player wins by horizontal win	Player 1 or 2 win by obtaining a horizontal 4 in a row	A piece is played, minimum 7 plays made	Player enters their play	Game detects a horizontal 4 in a row and declares the winner	Game detects the 4 in a row for scenarios of a player 1 win and a player 2 win	Yes
7 – Player wins by vertical win	Player 1 or 2 win by obtaining a vertical 4 in a row	A piece is played, minimum 7 plays made	Player enters their play	Game detects a vertical 4 in a row and declares the winner	Game detects the 4 in a row for scenarios of a player 1 win and a player 2 win	Yes

8 – Player wins by diagonal win (in both directions)	Player 1 or 2 win by obtaining a diagonal 4 in a row	A piece is played, minimum 7 plays made	Player enters their play	Game detects a diagonal 4 in a row and declares the winner	Game detects the 4 in a row for all scenarios of a player 1 win and a player 2 win	Yes
9 – Player attempts an out of bounds play	Player 1 or 2 attempts to enter a number that is greater or less than the confines of the board	Game begun	Player enters their play	Game will that the number does not match the confines of the board	Game flags up an error and asks the player to re-enter a valid column	Yes
10 – Capture a row of three	Player captures another players row of three	Game begun, minimum 5 plays made	Player makes their play	Game detects 3 tokens are trapped between 2 of the other players and removes them	Code does not pick up on the captured pattern	No

Despite extensive testing of the code, a solution was unable to be discovered for the final unresolved test, and as such has not been fixed.

Conclusion

Completing the discussion process in which the team plans out how the code will work based on the requirements for the user requirements is both important and useful, as it allows for the team to ensure that the system has been built to the user's needs and demands, and also can help with the development process by having the project laid out in the order of execution, meaning the structure is easier to put together.

Testing a product is always important to ensure everything is working properly, and to be able to fix any issues that arise, either prior to release of the project or after it has gone out to the consumer. Utilising both black box and white box testing are useful things to do as each type of testing may bring up different types of issues.

Bibliography

- [1] T. Sheakh, "A Comparative Study of Software Testing Techniques Viz. White Box Testing Black Box Testing and Grey Box Testing," *International Journal of Allied Practice, Research and Review*, vol. II, no. 5, p. 8, 2015.

Appendices

Appendix A: Design Discussion

- Game starts with option to display rules
- Starts by drawing blank table
- Check player input to ensure its within boundaries of the board
- At the end of each play, check for a play that results in a trapped row or a victory
- Place code in a Boolean While command
- Ensure all instances of a win are covered
- Checks for a win status on every tile of the board

Appendix B: Software Code

```
#include <iostream>
#include <sstream>
#include <cstdlib>
using namespace std;

int table[7][6];
int choice, player;
int i = 0;
bool gameover, matchover = false;

void check(int x)
{
    if (table[x - 1][i] != 0 && i < 6)
    {
        i++;
        check(x);
    }
    else if (player == 1 && i < 6)
    {
        table[x - 1][i] = 1;
        i = 0;
    }
    else if (player == 2 && i < 6)
    {
        table[x - 1][i] = 2;
        i = 0;
    }
    else
    {
        cout << "Wrong" << endl;
        i = 0;
        player++;
    }
}

void draw()
{
    for (int j = 0; j < 9; j++)
    {
        if (j < 2)
        {
            cout << "-";
        }
        else if (j > 7)
        {
            cout << j - 1 << "--" << endl;
        }
        else
        {
            cout << j - 1 << "----";
        }
    }
    for (int j = 0; j < 6; j++)
    {
        for (int k = 0; k < 7; k++)
        {
            if (table[k][j] != 0)
            {
                if (table[k][j] == 1)
                {
                    cout << "| X |";
                }
            }
        }
    }
}
```

```

        }
        else
        {
            cout << " | 0 | ";
        }
    }
    else
    {
        cout << " |   | ";
    }
}
cout << endl;
}
for (int j = 0; j < 35; j++)
{
    cout << "=";
}
cout << endl;
}

void win_check()
{
    for (int j = 0; j < 6; j++)
    {
        for (int k = 0; k < 7; k++)
        {
            if (table[k][j] == 1 && table[k + 1][j + 1] == 1 && table[k + 2][j + 2] == 1 && table[k + 3][j + 3] == 1)
            {
                matchover = true;
                cout << "Game Over, Player 1 wins!" << endl;
            }

            if (table[k][j] == 1 && table[k + 1][j - 1] == 1 && table[k + 2][j - 2] == 1 && table[k + 3][j - 3] == 1)
            {
                matchover = true;
                cout << "Game Over, Player 1 wins!" << endl;
            }

            if (table[k][j] == 2 && table[k - 1][j - 1] == 2 && table[k - 2][j - 2] == 2 && table[k - 3][j - 3] == 2)
            {
                matchover = true;
                cout << "Game Over, Player 2 wins!" << endl;
            }

            if (table[k][j] == 2 && table[k - 1][j - 1] == 2 && table[k - 2][j - 2] == 2 && table[k - 3][j - 3] == 2)
            {
                matchover = true;
                cout << "Game Over, Player 2 wins!" << endl;
            }

            if (table[k][j] == 1 && table[k][j - 1] == 1 && table[k][j - 2] == 1 && table[k][j - 3] == 1)
            {
                matchover = true;
                cout << "Game Over, Player 1 wins!" << endl;
            }
        }
    }
}

```

```

        if (table[k][j] == 1 && table[k - 1][j] == 1 && table[k - 2][j]
== 1 && table[k - 3][j] == 1)
        {
            matchover = true;
            cout << "Game Over, Player 1 wins!" << endl;
        }

        if (table[k][j] == 2 && table[k][j - 1] == 2 && table[k][j - 2]
== 2 && table[k][j - 3] == 2)
        {
            matchover = true;
            cout << "Game Over, Player 2 wins!" << endl;
        }

        if (table[k][j] == 2 && table[k - 1][j] == 2 && table[k - 2][j]
== 2 && table[k - 3][j] == 2)
        {
            matchover = true;
            cout << "Game Over, Player 2 wins!" << endl;
        }
    }
}

void trap_Check()
{
    for (int j = 0; j < 6; j++)
    {
        for (int k = 0; k < 7; k++)
        {
            if (table[k][j] == 1 && table[k + 1][j + 1] == 2 && table[k +
2][j + 2] == 2 && table[k + 3][j + 3] == 2 && table[k + 4][j + 4] == 1)
            {
                for (int l = j; l < 6; l++)
                {
                    table[k][j] == table[k][j + 1];
                }

                if (table[k][j] == 1 && table[k + 1][j - 1] == 2 && table[k +
2][j - 2] == 2 && table[k + 3][j - 3] == 2 && table[k + 4][j - 4] == 1)
                {
                    for (int l = j; l < 6; l++)
                    {
                        table[k][j] == table[k][j + 1];
                    }
                }

                if (table[k][j] == 1 && table[k][j - 1] == 2 && table[k][j - 2]
== 2 && table[k][j - 3] == 2 && table[k][j - 4] == 1)
                {
                    for (int l = j; l < 6; l++)
                    {
                        table[k][j] == table[k][j + 1];
                    }
                }

                if (table[k][j] == 1 && table[k - 1][j] == 2 && table[k - 2][j]
== 2 && table[k - 3][j] == 2 && table[k - 4][j] == 1)
                {
                    for (int l = j; l < 6; l++)
                    {

```

```

        table[k][j] == table[k][j + 1];
    }
}

    if (table[k][j] == 2 && table[k + 1][j + 1] == 1 && table[k +
2][j + 2] == 1 && table[k + 3][j + 3] == 1 && table[k + 4][j + 4] == 2)
    {
        for (int l = j; l < 6; l++)
        {
            table[k][j] == table[k][j + 1];
        }
    }

    if (table[k][j] == 2 && table[k + 1][j - 1] == 1 && table[k +
2][j - 2] == 1 && table[k + 3][j - 3] == 1 && table[k + 4][j - 4] == 2)
    {
        for (int l = j; l < 6; l++)
        {
            table[k][j] == table[k][j + 1];
        }
    }

    if (table[k][j] == 2 && table[k][j - 1] == 1 && table[k][j - 2]
== 1 && table[k][j - 3] == 1 && table[k][j - 4] == 2)
    {
        for (int l = j; l < 6; l++)
        {
            table[k][j] == table[k][j + 1];
        }
    }

    if (table[k][j] == 2 && table[k - 1][j] == 1 && table[k - 2][j]
== 1 && table[k - 3][j] == 1 && table[k - 4][j] == 2)
    {
        for (int l = j; l < 6; l++)
        {
            table[k][j] == table[k][j + 1];
        }
    }
}

}

void playerChoice()
{
    player = 1;
    while (matchover == false)
    {
        cout << "Player " << player << ": ";
        cin >> choice;
        if (choice > 0 && choice < 8)
        {
            check(choice);
            draw();

            if (player == 1)
            {
                player++;
            }
            else
            {

```

```

        player--;
    }
}
else
{
    cout << "Invalid play" << endl;
}
trap_Check();
win_check();
}
}

int main(void)
{
    while (!gameover)
    {
        int option;
        cout << "Choose an option:" << endl << "1. Start Game" << endl << "2.
Display rules" << endl << "3. Exit Program" << endl;
        cin >> (option);

        switch (option)
        {
            case 1:
                cout << "New Game starting" << endl;
                draw();
                playerChoice();
                return 0;
                break;

            case 2:
                cout << "Rules: " << endl << "2 Players will take turns in
placing a counter into the board." << endl << "The counter will fall down to the
lowest available free space underneath it." << endl;
                cout << "The end goal of the game is to line four of your
counters in a row, in any direction; vertically, horizontally or diagonally." << endl;
                cout << "A player can trap another players icon. " << endl << "If
a player places a counter on either side of a row of three of another players counter,
then the three counters of the other player is destroyed." << endl;
                cout << endl << endl << endl;
                break;

            case 3:
                gameover = true;
                break;

            default:
                cout << "Error, invalid entry" << endl;
                break;
        }
    }

    cout << "Closing Program...";
    cin.get();
}

```

SEPT Assignment

GRADEMARK REPORT

FINAL GRADE

53/100

GENERAL COMMENTS

Instructor

Eval design (10): 5

User Stories (20): 13

CRC (20): 9

SW Testing (30): 15

Coding (20): 11

Feedback:

=====

The evaluation is very brief and the user stories & CRCs needed more thought. There seems to be a fundamental misunderstanding of CRCs.

The test provided could have been a little more formal. No evaluation of software testing provided.

The code could have been cleaner, needed comments and is a little over complicated in parts. However, you have had a go and this is not a bad attempt.

PAGE 1

PAGE 2

PAGE 3

PAGE 4

PAGE 5



Comment 1

Remember if you can split a story then do so, because stories are meant to be self contained units of work. Some of these stories can be split up.

PAGE 6

PAGE 7

PAGE 8

PAGE 9



Comment 2

Not sure what this is implying?

What you need is a class called Token.

The player 1 and player 2 tokens are instances (objects) created from this common class.



Comment 3

Not sure what this is implying

PAGE 10

PAGE 11

PAGE 12

PAGE 13

PAGE 14

PAGE 15

PAGE 16

PAGE 17

PAGE 18

PAGE 19

PAGE 20
