

Software principles 1602835

jake russell

by Jake Andrew Russell

FILE	12019_JAKE_ANDREW_RUSSELL_SOFTWARE_PRINCIPLES_1602835_JAKE_RUSSELL_302868_2131412330.DOCX (97.86K)		
TIME SUBMITTED	13-FEB-2021 03:28PM (UTC+0000)	WORD COUNT	1850
SUBMISSION ID	144755179	CHARACTER COUNT	8735

Software Principles and Practice
Jake Russell
1602835

Table of Contents

Design Evaluation and User stories..... 3

Class-Responsibility-Collaboration (CRC) Cards..... 4

Software Testing Evaluation 5

Version Control 5

Appendix 6

 Code 6

Design Evaluation and User stories

The design for this Connect 4 game works well as the default rules of connect 4, however the author was unable to successfully implement the amendments of the rules presented in the brief, such as the capturing of counters or the destruction of counters. A greater use of classes could have possibly been implemented as currently there is only one class being used. However, the author believes that due to their only being one object (being the player) involved for this application, that it is the only appropriate section of code to be implemented as a class.

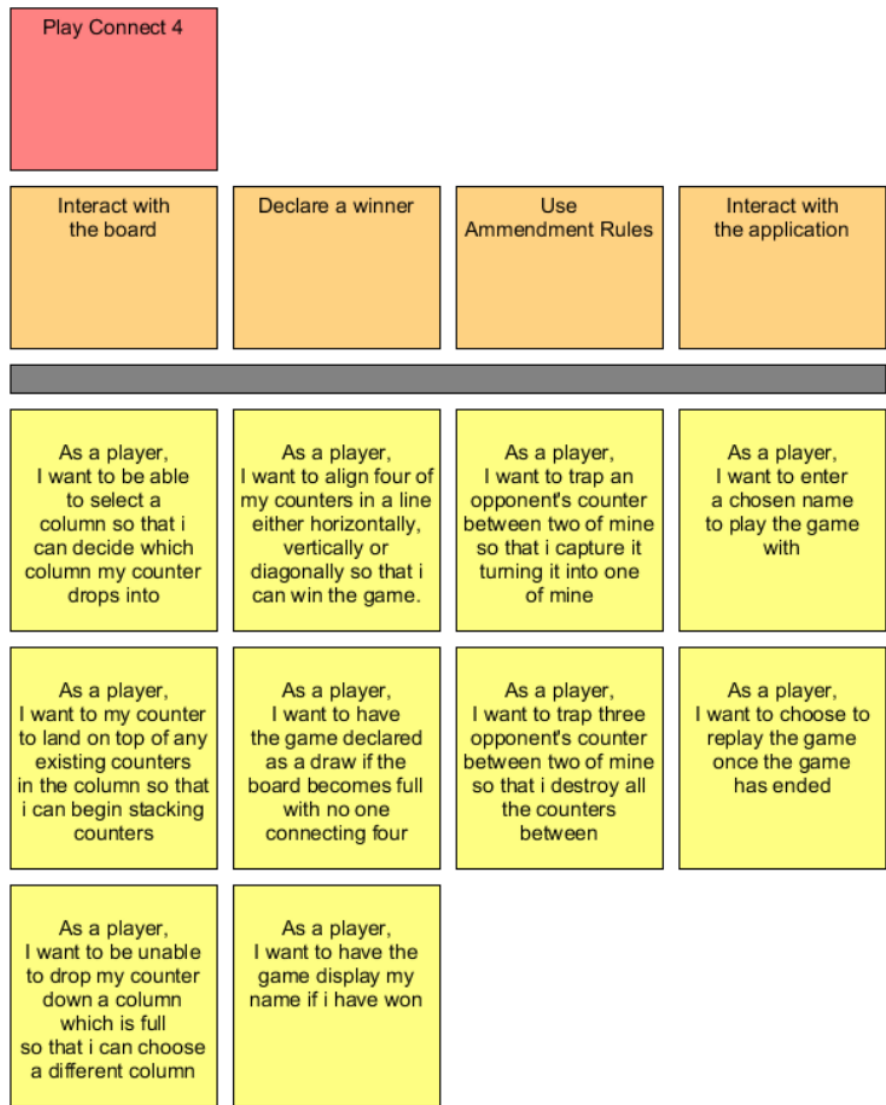


Figure 1: User story map

The above image (figure 1) is a User story map developed for this application. The first row is the user activities, the two being to play Connect 4 and to play with the amendment rules. The user tasks for the “Play connect 4” activity are as follows: to be able to interact with the board, to have a winner be found and declared, to be able to play with new amendment rules, and to interact with the application. For the first user story of the “Interact with board” task, the user wishes to be able to select a column when they are a player in order to allow them to specify what column their counter will drop down. The benefit of the story is that the players have complete freedom to choose which column to drop their counters down as that is the rules of connect 4. The author believes the developed design has a reasonably efficient method of doing this by asking the user to enter from 1 to 7, 1 being the first column and seven being the last, the program will then drop their counter down the specified column. The second story for this task has the user wanting their counters to land on top of any existing counters in the column they choose so that they can start stacking counters and the author believes the design made is very successful at carrying this out. The third story is that as a player the user wants to be unable to drop their counter in a column that is full, that way they can choose a different column instead, the author once again believes the design succeeds in carrying out this want as the design will check whether the number of tokens in a column is at its max amount, and if so gives the user the opportunity to select a different column instead.

Class-Responsibility-Collaboration (CRC) Cards

CRC cards are cards that are divided into 3 sections, the first being where the name of the class is, the second to the bottom left is the class’s responsibilities, in other words, what the class does/holds, and the third section to the bottom right where the collaborators are wrote, the collaborators being other classes that this class uses in order to carry out its responsibilities. This approach is effective at relaying information on how classes are to be implemented within the program to programmers. Below is a CRC made for class used with this program:

Player		
Knows Player Name Knows Player Symbol Gets Drop Location Checks Column Room Checks for winner Outputs the winner		Program

Figure 2: Player Class CRC card

The author believes that there is sufficient detail to enable most of the code to be implemented and tested apart from maybe being unclear that this class also drops the counters down a column after getting the location and checking how many counters are in the column currently. Relating it to the

user stories, it allows the user to choose a column to drop a counter down, it will check for and declare a winner, and through checking the room of a column, allow the counters to land on top of other counters or prevent the user from dropping a counter into a full column.

Software Testing Evaluation

Software testing is an integral part of software design, with the world moving further and further into a digital age, our reliance software and technology to work as intended becomes more critical. Bugs and anomalies can create disastrous results depending on the software's purpose, and even if it is not a disastrous scenario, quite often software is sold to customers, therefore customers will have an expectation for the software to work as advertised, and if it does not, that can cause problems for the customer and the company's involved in the transaction and creation alike.

Two methods of software testing are Black and White box testing. Black box testing is where the software in question is being tested by a tester unaware of the internal structure, design, or implementation of the code, this is typically done by independent software testers who get hired by a company to serve this purpose. On the other hand White box is where the tester does know about the internal structure, design, and implementation. Typically done by the software developers. The term testable code refers to code that is written so it is independently verifiable, meaning it is code that carries out specific behaviours which the tester wants to verify, this makes it code that allows automated testing to be quick and easy. If a test fail using testable code, it should be able to say what exactly why it failed so a correction can be made immediately.

Version Control

"Version control software is a key tool in the software development process", the author agrees with this quote, because using version control software host a multitude of advantages. Version control allows a user to have more backups of their software, say if they accidentally deleted a file, version control would allow them to recover it. Version control also allows a user to keep a large history of their changes, say a programmer was working on a project but they get pulled of it for several months, when returning they can use version control to see every change made and why they made them to allow them to return with ease. Version control becomes especially helpful when collaboration is involved as every programmer can see what each other does and the changes they each make.

Appendix

Code

```
using System;

namespace Connect4
{
    class player //struct to hold player data
    {
        public string playerName;
        public char playerSymbol;

        public int DropLocation(char[,] board) //method to find the drop location a
        user wishes to drop their counter
        {
            int dropPlace = 0;
            Console.WriteLine(playerName + "'s turn");
            while (dropPlace < 1 || dropPlace > 7)
            {
                Console.WriteLine("Please enter a number from 1 to 7, 1 being the
first column and 7 being the last: ");
                dropPlace = Convert.ToInt32(Console.ReadLine());
            }
            while (board[1, dropPlace] == 'X' || board[1, dropPlace] == 'O')
            {
                Console.WriteLine("That cannot place a new counter on column, please
choose a new column: ");
                dropPlace = Convert.ToInt32(Console.ReadLine());
            }

            return dropPlace;
        }

        public void CheckBelow(char[,] board, int dropPlace) //method to check how
        many counters are below the drop location and place the counter on top
        {
            int length = 6;
            int turn = 0;
            while (turn != 1)
            {
                if (board[length, dropPlace] != 'X' && board[length, dropPlace] !=
'0')
                {
                    board[length, dropPlace] = playerSymbol;
                    turn = 1;
                }
                else
                {
                    length--;
                }
            }
        }

        public int CheckFour(char[,] board) //method to check whether a player has
        connected 4
        {
            char XO;
        }
    }
}
```

```

int win;
XO = playerSymbol;
win = 0;

for (int i = 8; i >= 1; i--)
{
    for (int j = 9; j > -1; j--)
    {
        if (board[i, j] == XO &&
            board[i - 1, j - 1] == XO &&
            board[i - 2, j - 2] == XO &&
            board[i - 3, j - 3] == XO)
        {
            win = 1;
        }

        if (board[i, j] == XO &&
            board[i, j - 1] == XO &&
            board[i, j - 2] == XO &&
            board[i, j - 3] == XO)
        {
            win = 1;
        }

        if (board[i, j] == XO &&
            board[i - 1, j + 1] == XO &&
            board[i - 2, j + 2] == XO &&
            board[i - 3, j + 3] == XO)
        {
            win = 1;
        }

        if (board[i, j] == XO &&
            board[i - 1, j] == XO &&
            board[i - 2, j] == XO &&
            board[i - 3, j] == XO)
        {
            win = 1;
        }

        if (board[i, j] == XO &&
            board[i, j + 1] == XO &&
            board[i, j + 2] == XO &&
            board[i, j + 3] == XO)
        {
            win = 1;
        }
    }
}

return win;
}

public void PlayerWins() //function to output that a player has one
{
    Console.WriteLine(playerName + " has won!");
}

}

class Program
{

```

```

static void Main(string[] args)
{
    player playerOne = new player();
    player playerTwo = new player();
    char[,] board = new char[9, 10];
    int dropPlace, win, complete, replay;
    Console.WriteLine("Welcome to connect 4");
    Console.WriteLine("Player one, please enter your name");
    playerOne.playerName = Console.ReadLine();
    playerOne.playerSymbol = 'X';
    Console.WriteLine("Player two, please enter your name");
    playerTwo.playerName = Console.ReadLine();
    playerTwo.playerSymbol = 'O';
    complete = 0;
    win = 0;
    replay = 0;
    CreateBoard(board);
    while (replay != 2) //while loop that plays until a user sepcifies that
they no longer want to play
    {
        dropPlace = playerOne.DropLocation(board);
        playerOne.CheckBelow(board, dropPlace);
        CreateBoard(board);
        win = playerOne.CheckFour(board);
        if (win == 1) //if player one has won, tell the users then asks
whether they wish to replay
        {
            playerOne.PlayerWins();
            replay = Restart(board);
            if (replay == 2)
            {
                break;
            }
        }

        dropPlace = playerTwo.DropLocation(board);
        playerTwo.CheckBelow(board, dropPlace);
        CreateBoard(board);
        win = playerTwo.CheckFour(board);
        if (win == 1) //if player two has won, tell the users then asks
whether they wish to replay
        {
            playerTwo.PlayerWins();
            replay = Restart(board);
            if (replay == 2)
            {
                break;
            }
        }

        complete = CompleteBoard(board);
        if (complete == 7) //if both players have filled the board with no
winner, declar a draw and asks whether they wisjh to replay
        {
            Console.WriteLine("It's a draw");
            replay = Restart(board);
            if (replay == 2)
            {
                break;
            }
        }
    }
}

```

```

        Console.ReadLine();
    }

    static void CreateBoard(char[,] board) //function to generate the board in its
current state
    {
        int rows = 6, columns = 7, i, j;

        for (i = 1; i <= rows; i++)
        {
            Console.Write("|");
            for (j = 1; j <= columns; j++)
            {
                if (board[i, j] != 'X' && board[i, j] != 'O')
                    board[i, j] = '*';
                Console.Write(board[i, j]);
            }
            Console.Write("| \n");
        }
    }

    static int Restart(char[,] board) //function to create a fresh board and
return whether the users wish to play again
    {
        int restart;
        Console.WriteLine("Would you like to play again? Enter (1) for Yes and (2)
for No");
        restart = Convert.ToInt32(Console.ReadLine());
        if (restart == 1)
        {
            for(int i =1; i <= 6; i++)
            {
                for(int j = 1; j <= 7; j++)
                {
                    board[i, j] = '*';
                }
            }
        }
        else
        {
            Console.WriteLine("Until next time");
        }
        return restart;
    }

    static int CompleteBoard(char[,] board) //function to check whether the
players have filled the board with counters
    {
        int complete;
        complete = 0;
        for(int i = 1; i <= 7; i++)
        {
            if(board[1,i]!='*')
            {
                complete++;
            }
        }
        return complete;
    }
}

```

FINAL GRADE

52/100

GENERAL COMMENTS

Instructor

Evaluation of design(10): 4

User Stories + Map (Benefits) (20): 9

CRC cards (20): 9

Software testing evaluation (20): 9

Version Control (10): 5

Coding (20): 16

Comments:

The user story mapping section is bit brief and the examples shown is bit short. Likewise, CRC card discussion needed more evaluation of the benefits or drawbacks of the approach.

Software testing evaluation should have been a bit more detailed. The report is a bit rushed.

Quite liked the code but wanted to see some evidence of testing.

PAGE 1

PAGE 2

PAGE 3

PAGE 4

PAGE 5

PAGE 6

PAGE 7

PAGE 8

PAGE 9