

Software Testing Assignment

Timothy Day

by Timothy James Day

Submission date: 28-Apr-2023 09:34AM (UTC+0100)

Submission ID: 203821069

File name:

42289_Timothy_James_Day_Software_Testing_Assignment_Timothy_Day_490085_308703278.docx (296.54K)

Word count: 2574

Character count: 12761

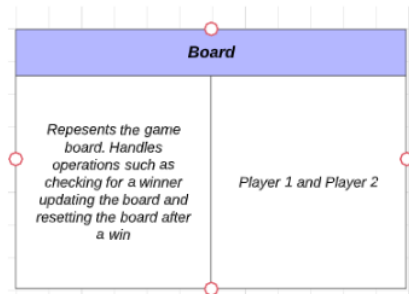
Software Engineering Principles and Testing

By Timothy Day

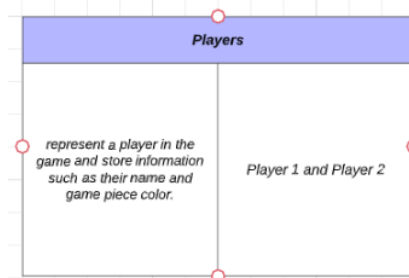
In this assignment I will be creating different diagrams including flow charts and CRC cards to describe the process to a connect 4 game and a connect 4 game with the change in rules set. And evaluate each diagram. Also I have explored using a variety of testing of the program. Including black and white box testing.

I created CRC cards for an ordinary game of connect 4 and a game of connect 4 with the rule changes giving an overview of different classes within the games.

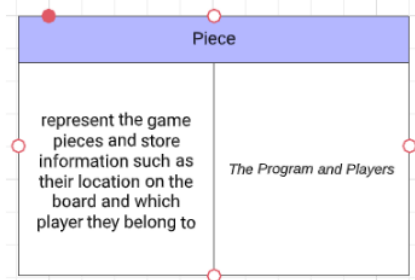
The first CRC class is the board that handles operations such as checking for a winner at the end of the round. Updating the board after each turn and resetting the board after a player has won. And that each player is a collaborator for the class because the players are the only collaborators to interact with the board.



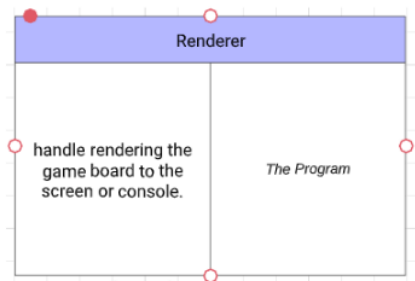
The second CRC class is the player which represents each player and stores player information relevant to the game. Such as Player Names and Game piece colours. The collaborators for this class are the player as this class requires user input.



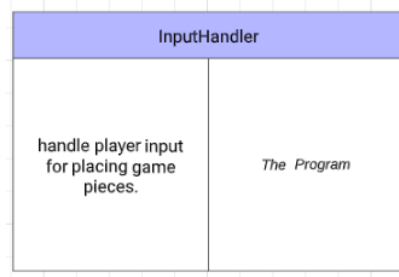
The third CRC class is the Game piece. Which represents the game pieces and stores information such as the pieces locations on the board and to which piece belongs to which player which is signified by different colours. The collaborators are the program that keeps track of piece locations on the board and the players which interact with the pieces.



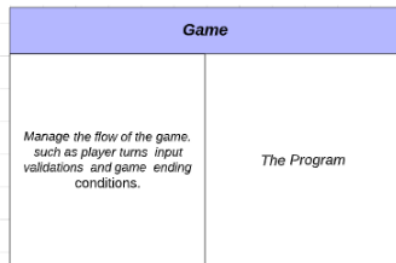
The fourth CRC class is the Game Renderer which handles rendering the game board on the screen. The collaborator is the program as the program will produce the user interface (UI) to allow users to interact with the game.



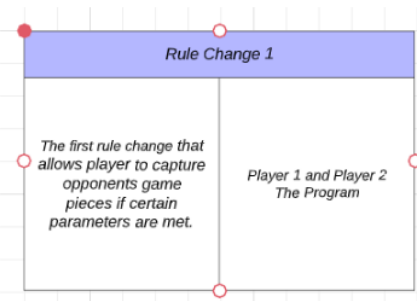
The fifth CRC class is InputHandler which handles each users input and places the game piece where the user wants to place them. The collaborator is the Program as these are programable inputs controlled by a user input via a mouse click, key bind or an controller input.



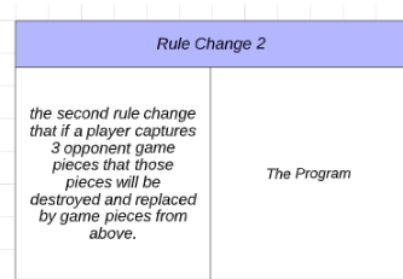
The sixth CRC class is the Game. The game class manages the flow of the game such as player turns, validating inputs and understanding and knowing game ending conditions. The collaborator is the program as the program runs the game for the users.



The seventh CRC class is the first rule change that allows player to capture opponents game pieces if certain parameters are met. The collaborators are the players as the game will prompt the player if the rule is able to be conducted. The program is also a collaborator as the program will keep track of each move the players make.

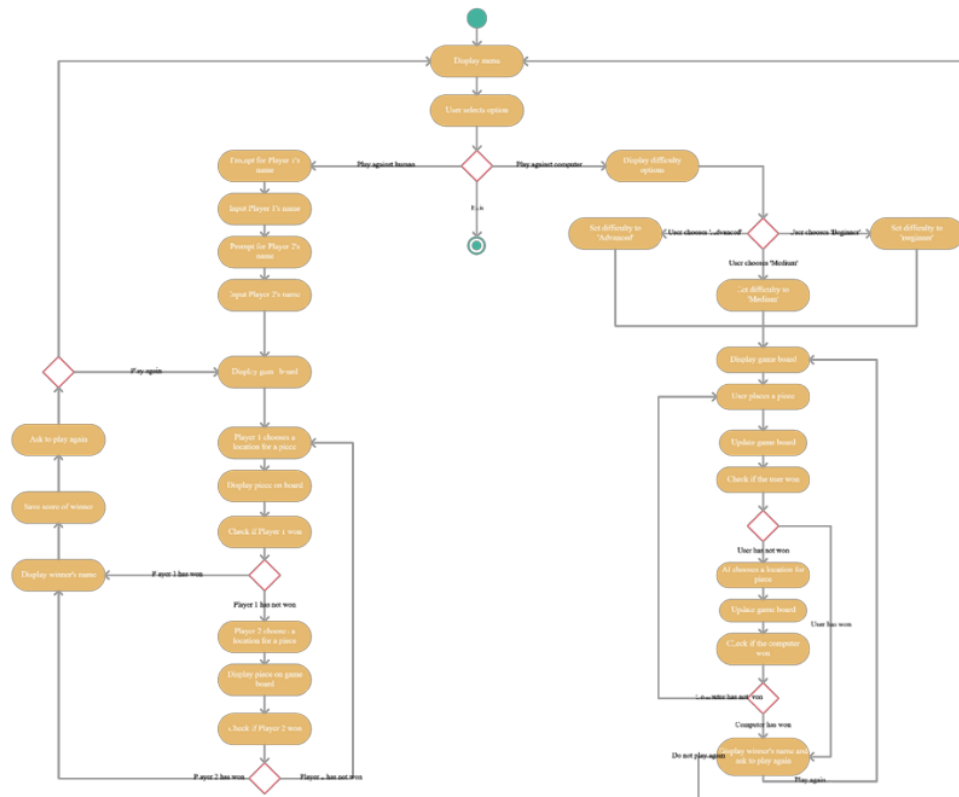


The eighth CRC class is the second rule change that if a player captures 3 opponent game pieces that those pieces will be destroyed and replaced by game pieces from above. The collaborator is the program as the program keeps track of each piece and then if three are captured, they are destroyed and replaced by pieces above them.



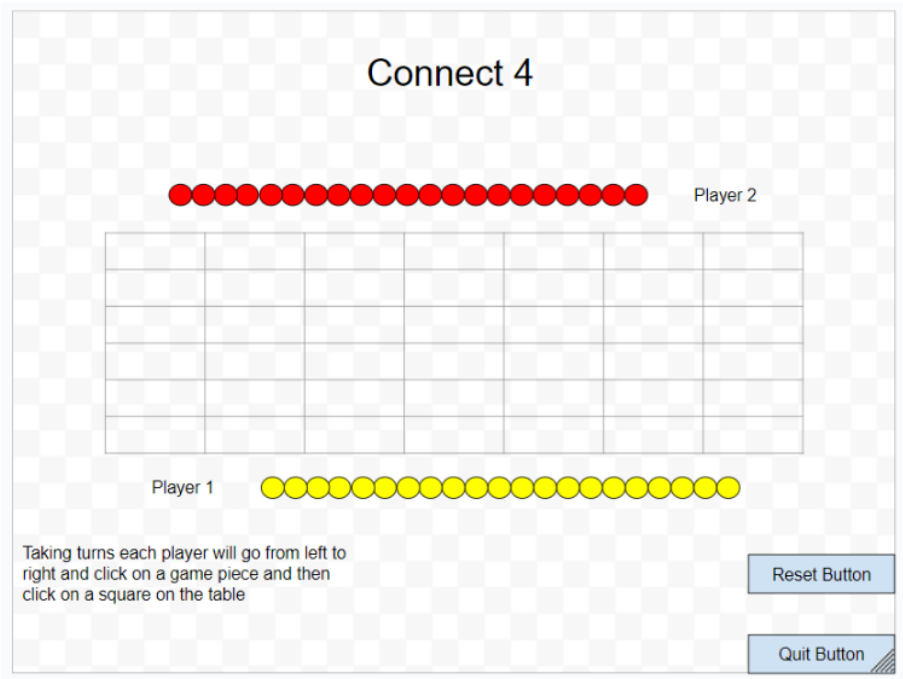
In conclusion these CRC cards just scratch the surface of each aspect of the game and can go even more in depth with each aspect.

Next to understand the flow of the game I created a flowchart that runs through a typical game of connect 4.



For a brief overview of this flow chart. The user can select between playing against another user or the computer (CPU) if playing against a human the user can input user data such as name for both player one and two. Next the game will generate a board and player one will go first then player two will go after and repeat till a winner is declared. Then the program will then display the winners name and ask if the users want to play again. If the user chooses to play against the CPU then the program will ask the user to set the difficulty of the game. Next the game will generate a board and the user will go first in placing a game piece then the it will be the CPUs turn and then repeat till a winner is declared. Then the program will display the winners name and ask the user if they want to play again. This flowchart gives a good understanding of how an ordinary game of connect 4 will proceed within a program. With the rule changes given in the assignment I would add extra details to the flowchart showing the change in the rules. Now a player can capture another player's game piece if that game piece is surrounded vertically, horizontally and diagonally if the program tracks the movement of each game piece in a game then the game will give a prompt to the player that they can capture their opponents piece. And the new second rule is that if a player captures three opponent pieces those pieces will be destroyed and replaced by the game pieces above the destroyed ones. This rule can be programmed into the game and game can execute this when the certain parameters have been met.

Below is a simple wire frame diagram of connect 4 showing each player, the board, each game piece for both players and buttons to reset the game and to quit the game. Also included is a little instruction in how to play by click on each game piece and then to click on with board square the player wants to place the piece.



There are multiple way we could test out the program. Manual testing means that we play the program and collect data points on what works and what doesn't work.

Gameplay testing involves playing the game to ensure that the program functions as expected. This could include testing moves, win conditions and other gameplay elements. User interface testing involves testing the games graphical user interface to ensure that it is easy to use, visually appealing, and responsive. This could include testing the layout, colour scheme and the responsiveness of the interface.

Compatibility testing involves testing the game on different devices, browsers and operating systems to ensure that it functions correctly across different platforms. Multiplayer testing involves testing the games multiplayer functionality to ensure that it works and that players can connect locally and online without issues.

Edge case testing involves testing the game in scenarios that are unlikely to occur during normal gameplay but could still occur. This involves players making invalid moves or where the game board is full, and no more moves can be made. A big downside to manual testing is that it is time consuming.

Automated testing allows you to test the game's functionality without manual intervention; this could include testing game logic and testing GUI. The downside to unit testing is limited test coverage. Automated testing is only as good as the tests that have been created. If the tests have not been designed to cover all possible scenarios, there may be bugs or issues that are missed. Costs as automated testing can be expensive to set up and maintain.

Unit testing involves testing individual parts of the game's code such as functions or modules to ensure they function as expected. The downside to unit testing is limited scope. Unit testing focuses on individual code modules or functions and may not be able to test the interactions between these modules. As a result, it may not detect issues that arise when multiple modules interact with each other.

Integration testing involves testing how different parts of the game's code interact with each other. This could involve testing games between AI and humans or checking how the GUI interacts with the game's logic. The downside to integration testing is complexity. Integration testing involves testing the interaction between different components or systems, which can be complex and time-consuming. The complexity increases with the number of components or systems being integrated, making it difficult to test comprehensively.

User acceptance testing involves getting feedback from real users to ensure the game meets their expectations. Invite beta testers to play the game and provide feedback. The downside to UAT is that it is resource intensive. UAT requires dedicated resources, including personnel, equipment, and time, which can be costly for organizations. UAT can also delay the release of software if issues are identified during testing.

White box testing would involve testing the game's internal workings and code to ensure that it functions correctly.

A code walkthrough is a manual review of the code by a team of developers to ensure that the code is well-organized, easy to read, and follows best practices.

Code coverage analysis involves using specialized software tools to measure the percentage of code that has been executed during testing. This can help ensure that all parts of the code have been tested and that there are no untested areas.

Path testing involves analysing the game's code to identify all possible paths through the code and testing each path to ensure that it functions correctly.

Boundary value analysis involves testing the game's code at the boundaries of its input values to ensure that it can handle edge cases and unexpected inputs.

Mutation testing involves modifying the game's code to introduce faults and errors and then running tests to ensure that these faults are caught and handled correctly.

By using a combination of these white box testing methods, we can ensure that the game's code is well-organized, easy to read, and free of bugs and errors, which will help ensure that it functions correctly and provides an enjoyable experience for players.

The downsides to white box testing are.

Requires technical expertise: White box testing requires testers to have a deep understanding of the code and its internal workings. This means that testers need to have technical expertise and programming skills, which can be a limitation if the testing team lacks the necessary skills.

Limited scope: White box testing focuses on testing the internal workings of the code, such as logic, data flow, and control flow. It may not cover all possible scenarios or use cases, leading to issues being missed.

Time-consuming: White box testing can be time-consuming, as it requires testers to analyze and understand the code before writing test cases. This can be a significant investment of time, especially for complex codebases.

Risk of over-testing: With white box testing, there is a risk of testing too much and not focusing on the most critical aspects of the code. This can lead to tests that do not effectively prioritize critical areas of the code or overlook critical scenarios.

Risk of under-testing: On the other hand, there is also a risk of under-testing with white box testing. Testers may focus too much on the internal workings of the code and overlook testing the code's external behaviour, such as its user interface and user experience.

Overall, white box testing is a valuable testing method that can help identify issues and improve code quality. However, it should be complemented with other testing methods, such as black box testing and user acceptance testing, to ensure that the software is thoroughly tested from both an internal and external perspective.

Black box testing would involve testing the game's functionality from a user's perspective, without any knowledge of its internal workings or code.

Functional testing involves testing the game's functionality to ensure that it meets its requirements and specifications. This could include testing the game's basic gameplay mechanics, such as how moves are made and how the game is won or lost.

GUI testing involves testing the game's graphical user interface to ensure that it is easy to use and visually appealing. This could include testing the layout of the game board, the color scheme, and the responsiveness of the interface.

Usability testing involves testing the game to ensure that it is easy to use and understand. This could involve testing the game's instructions, the ease of making moves, and the overall flow of the game.

Compatibility testing involves testing the game to ensure that it functions correctly on different devices and operating systems. This could include testing the game on different browsers, mobile devices, and operating systems to ensure that it works correctly across different platforms.

Performance testing involves testing the game's performance under different loads and conditions to ensure that it can handle many players and moves without crashing or slowing down.

By using a combination of these black box testing methods, you can help ensure that the game is easy to use, visually appealing, and functions correctly from a user's perspective, which will help provide an enjoyable experience for players.

The downsides to black box testing are.

Limited visibility: Black box testing is based on testing the software from an end-user's perspective, which means that testers have limited visibility into the internal workings of the code. This can make it difficult to identify the root cause of issues that are discovered during testing.

Limited coverage: Black box testing focuses on testing the external behaviour of the software, such as its user interface and user experience. This means that it may not cover all possible scenarios or use cases, leading to issues being missed.

Limited testing of complex scenarios: Black box testing may not effectively test complex scenarios that require knowledge of the internal workings of the code. This can result in critical issues being missed.

Repetitive testing: Black box testing can be repetitive and time-consuming, as it requires testers to manually test the software for each use case or scenario. This can result in a significant investment of time, especially for large or complex software products.

Lack of technical expertise: Black box testing does not require testers to have a deep understanding of the code and its internal workings. This can limit the ability of testers to identify technical issues that require specialized knowledge.

Overall, black box testing is an important testing method that provides valuable feedback from an end-user's perspective. However, it should be complemented with other testing methods, such as white box testing and integration testing, to ensure that the software is thoroughly tested from both an internal and external perspective.

In conclusion there many different ways we can test the game and I think the best course of action is to select a few testing methods from both white and black box testing principles. Even with the downsides of each. Black box and white box testing are core staples of the industry.

In this section I am going to conduct hypothetical test that could be used on the program.

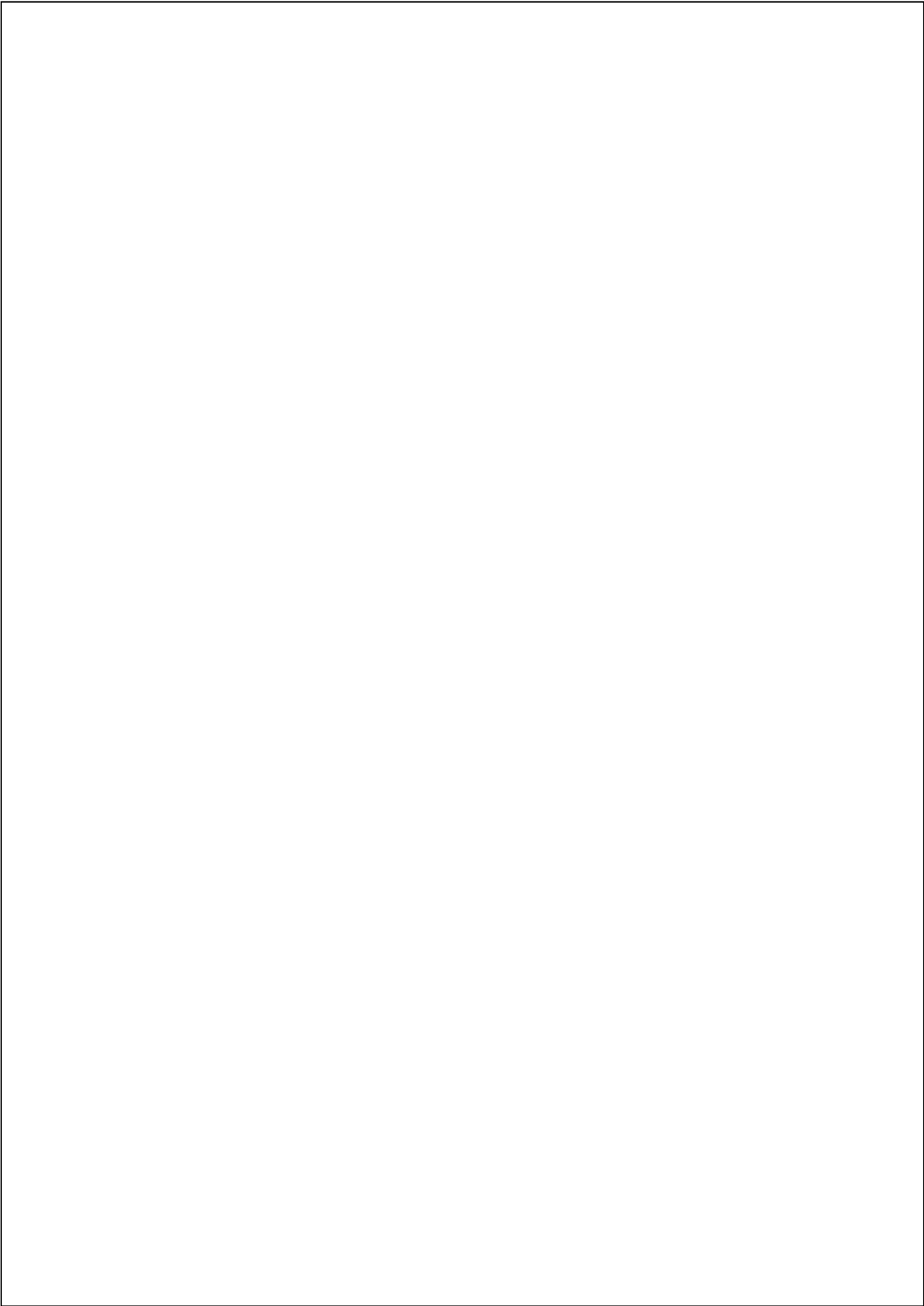
Test 1:

Run the game and check over the user interface if all seems as expected proceed to playing a game against the AI. Test out multiple moves the user could conduct in the first play. Try moving multiple game pieces to see if you can go in any order. Basically, try to break the game logic and hopefully the game will work as intended. Make sure that all possible events could take place within game do. Test valid and invalid moves. Winning, losing and tie conditions. Test out performance such as having multiple instances of the game open at once.

Test 2:

Repeat the same testing above but include a second user. This allows us to test out the multiplayer aspect of the game.

In a real-world scenario, I would be able to test out the game further and more in-depth. Holding multiple beta phases that would allow for user feedback and the implementation of that feedback.



Software Testing Assignment Timothy Day

GRADEMARK REPORT

FINAL GRADE

47 / 100

GENERAL COMMENTS

Instructor

Eval of Design (20): 10

User Stories (20): 0

CRC (20): 12

SW Testing (20): 20

Testing (20): 5

Feedback:

=====

The design process is not properly evaluated, you have missed the discussion of user stories and the Agile team work approach. You have provided some flow charts.

CRC 's are intended to focus on classes in OOP and they were not as detailed as I would have liked.

Very detailed discussion of SW testing. Only two tests were identified for the game software, they lacked a little detail For example, the test setup and correct result.

PAGE 1

PAGE 2

PAGE 3

PAGE 4

PAGE 5

PAGE 6

