**City University of Hong Kong**

CS3343 Software Engineering Practice

2023/24 Semester A

Test Report

Project Title: Connect Four Game

Group 27

|  |  |
| --- | --- |
| **Name** | **Student Number** |
| Fong Tsz Wai | 57156110 |
| JI Xin | 40147182 |
| Ka Lam Mark Alexander LEE | 56759365 |
| LIN Xiaoyang | 57126676 |
| SONG Rui | 57126559 |

Table of Contents

[1. Introduction 2](#_Toc437368962)

[2. Module Organization 3](#_Toc564991969)

[3. Testing Strategy 5](#_Toc1649906482)

[4. Test Case Development 12](#_Toc995181349)

[5. Refactor 15](#_Toc234074573)

[6. Coverage Analysis 19](#_Toc578938195)

# Introduction

This testing report aims to provide a comprehensive overview of the testing process and results for the Connect 4 software. This testing effort ensures the quality, reliability and functionality of our software program. By conducting a range of tests including functional testing and performance testing, we aim to identify and address any defects or issues that may impact the users’ experience while enjoying the game.

This report outlines the module organization, testing strategy applied and the results obtained during the testing phase. It also includes an analysis of the identified defects and improvements to them. The insights gained from this testing process will assist the team in refining the software and ensuring its quality.

The following sections of this report provide detailed information on the testing process and test results.

# Module Organization

In this project, the Iterative Model is adopted. The first iteration focuses on the basic functions of Connect 4, while the second iteration deals with the additional features, AI (Artificial Intelligence).

*1st Iteration*

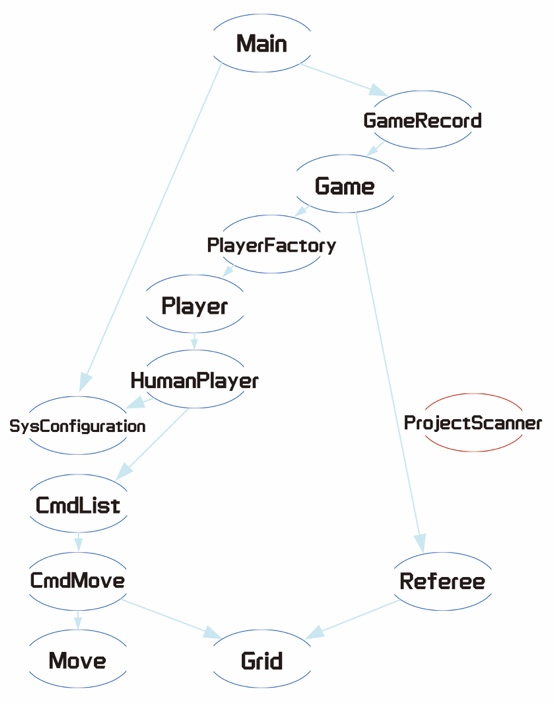


Figure 2.1 Diagram for test cases in 1st iteration

The diagram in Figure 2.1 describes our system's concrete structure, providing the basic functions for the player to interact with. It consists of class Main, GameRecord, Game, PlayerFactory, Player, HumanPlayer, SysConfiguration, CmdList, CmdMove, Move, Grid, Referee and ProjectScanner.

*2nd iteration*

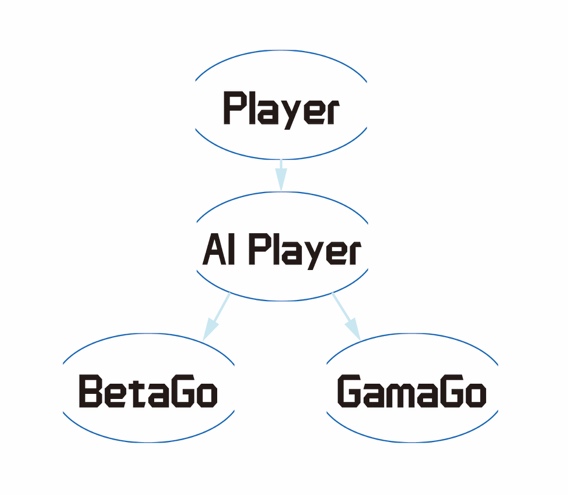


Figure 2.2 Diagram for test cases in 2nd iteration

In Figure 2.2, AI features are added. Two subclasses are implemented under AI Player interface. BetaGo and GamaGo represent two AI opponents of different difficulty. While providing challenging connect 4 games, human players can also ask for suggestions (In-game suggestion chances will be limited by configuration).

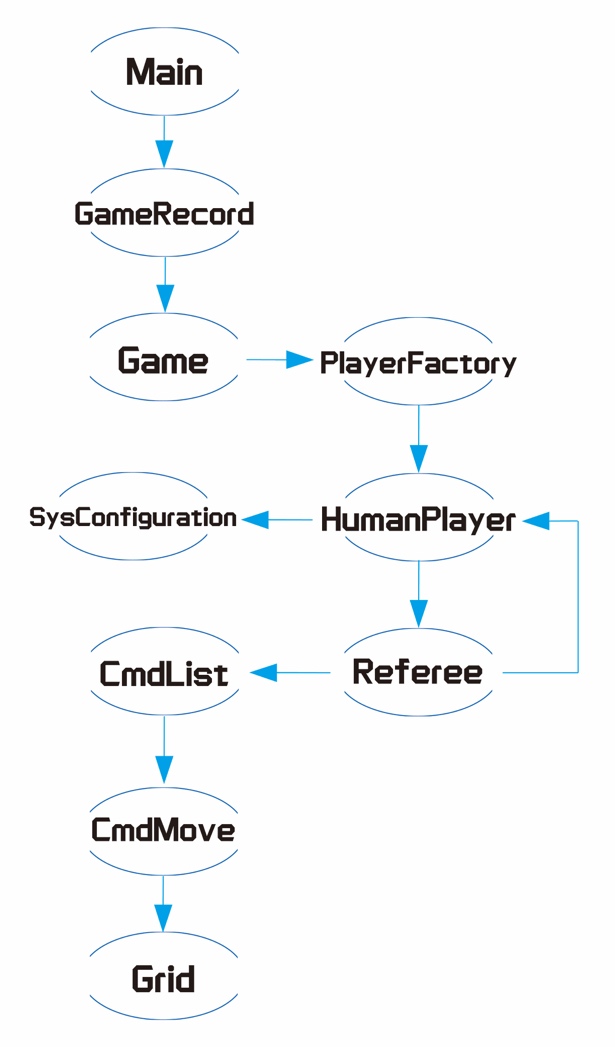
As shown in Figure 2.3, the control flow graph for a new game is demonstrated. A new Game variable will be created and stored inside the GameRecord. During the initiation of human player, the program will call SysConfiguration class to check the configuration ettings. In the game, players will be prompted to input actions. After being tested by the Referee class, if the input is improper, players will be asked to input again. Otherwise, the CmdList class will be invoked to create or review actions of CmdMove, thereby affecting the state of the Grid. *{Main -> GameRecord.addGame() -> new Game() -> PlayerFactory.createPlayer() -> SysConfiguration.getSysConfig() -> HumanPlayer.makeMove() -> Referee.isValidMove() -> CmdList.createCmdMove() -> new CmdMove() -> Grid.update()}*

Figure 2.3 Control flow graph for GameRecord.addGame()

More detailed control flow testing will be demonstrated in the following sections.

# Testing Strategy

In our testing strategy, we have adopted the Bottom-up approach, considering the crucial role of the bottom-level structures. This approach allows us to prioritize the testing of critical units, including game board (Grid), and player actions (CmdMove). These units form the foundation of the entire game. Thus, ensuring the functionality of the concreate components in the early stages would convince us in the future development.

Unit tests were carried out at the beginning of the testing process. The following classes are covered:

* Grid (Storing the state of the game board)
* Move (Storing the detail of movement by players)
* SysConfiguration (Read or write a local txt file which storing game settings)
* ProjectScanner (Handle human player’s input)

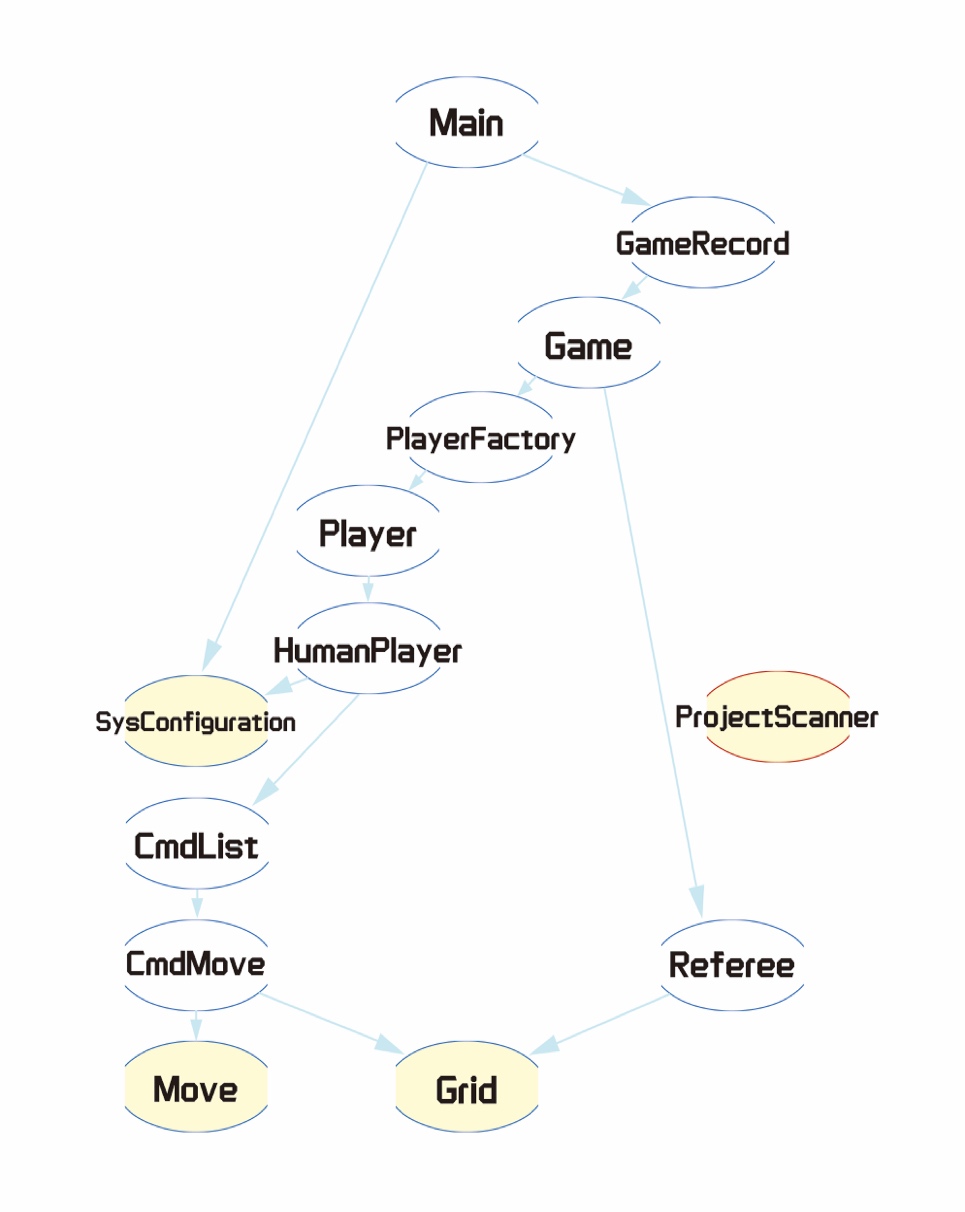


Figure 3.1 Unit tests

After unit tests, integration testing will be conducted in 6 stages.

1st stage:

* Referee + Grid (Check the game state of Grid, e.g. any winner or full column)
* CmdMove + Grid + Move (Responsible for updating current state of Grid)

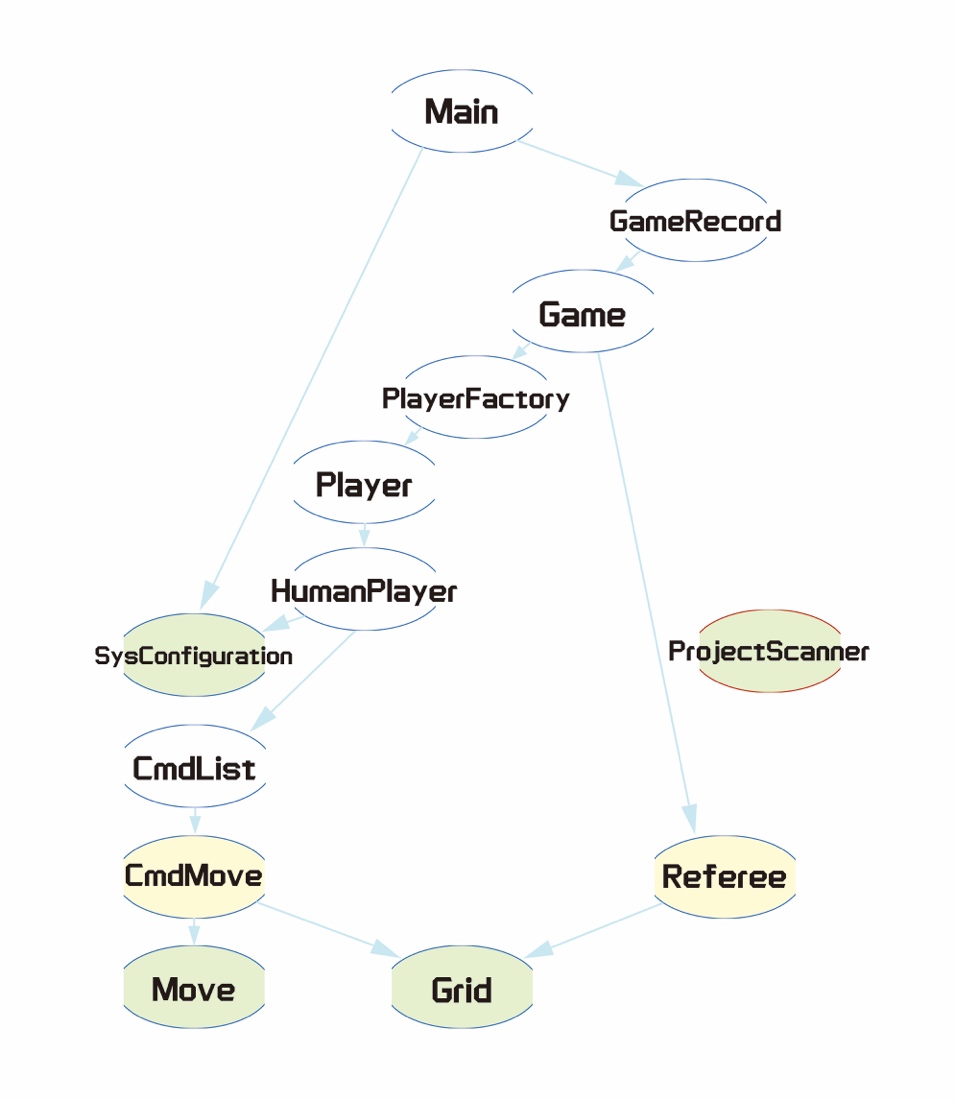


Figure 3.2 Integral tests in stage 1

2nd stage:

* CmdList + CmdMove + Move + Grid (Storing all the CmdMoves in a game and check if it is able for undo/redo action)

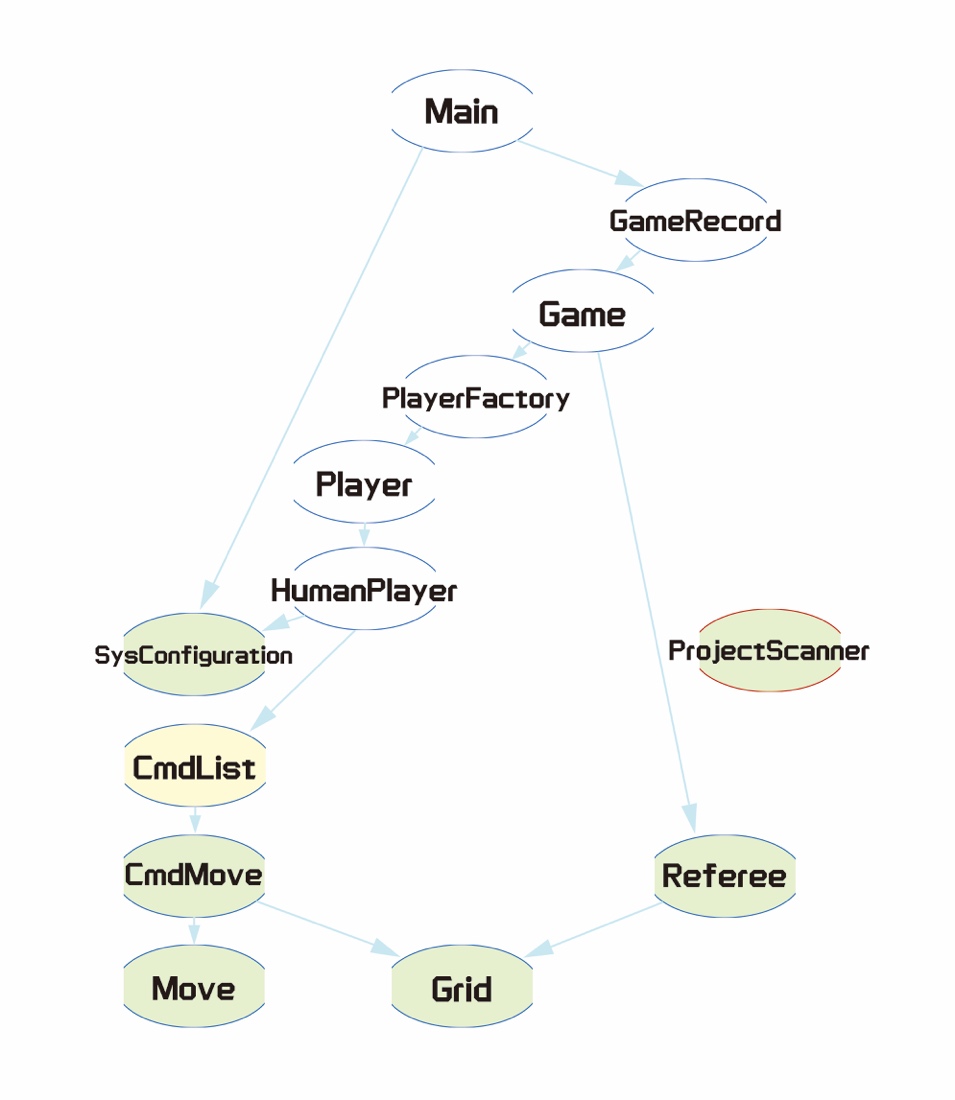


Figure 3.3 Integral tests in stage 2

3rd stage:

* HumanPlayer [subclass of Player] + SysConfiguration + CmdList + CmdMove + Move + Grid (Calling CmdList to make move while it is there turn)

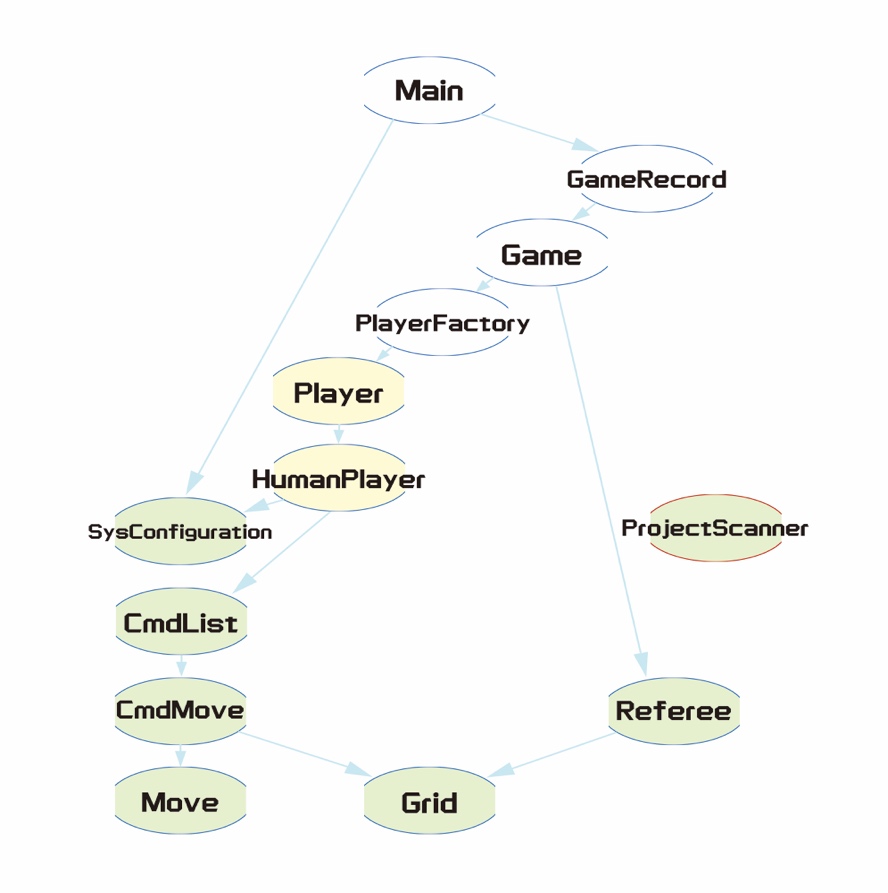


Figure 3.4 Integral tests in stage 3

4th stage:

* PlayerFactory + HumanPlayer + SysConfiguration + CmdList + CmdMove + Move + Grid (Create Player class)

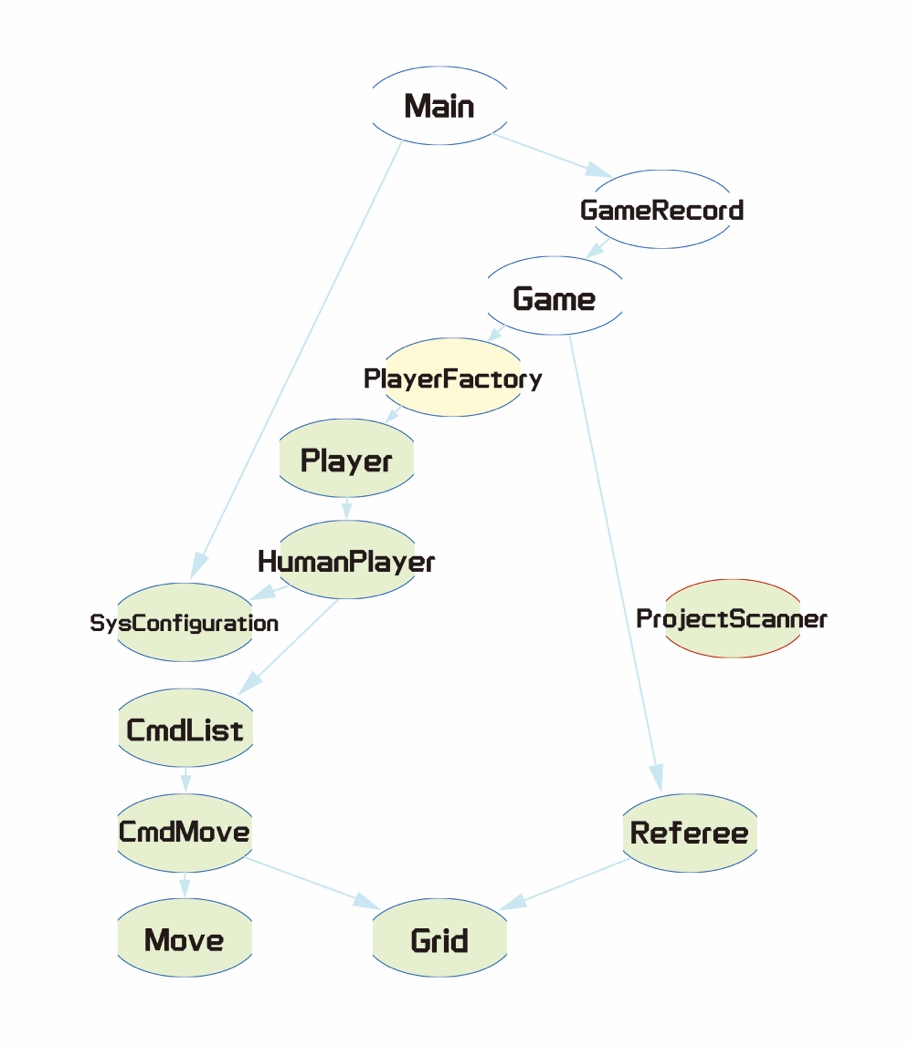


Figure 3.5 Integral tests in stage 4

5th stage:

* Game + PlayerFactory + HumanPlayer + SysConfiguration + CmdList + CmdMove + Move + Grid (Handling game process)

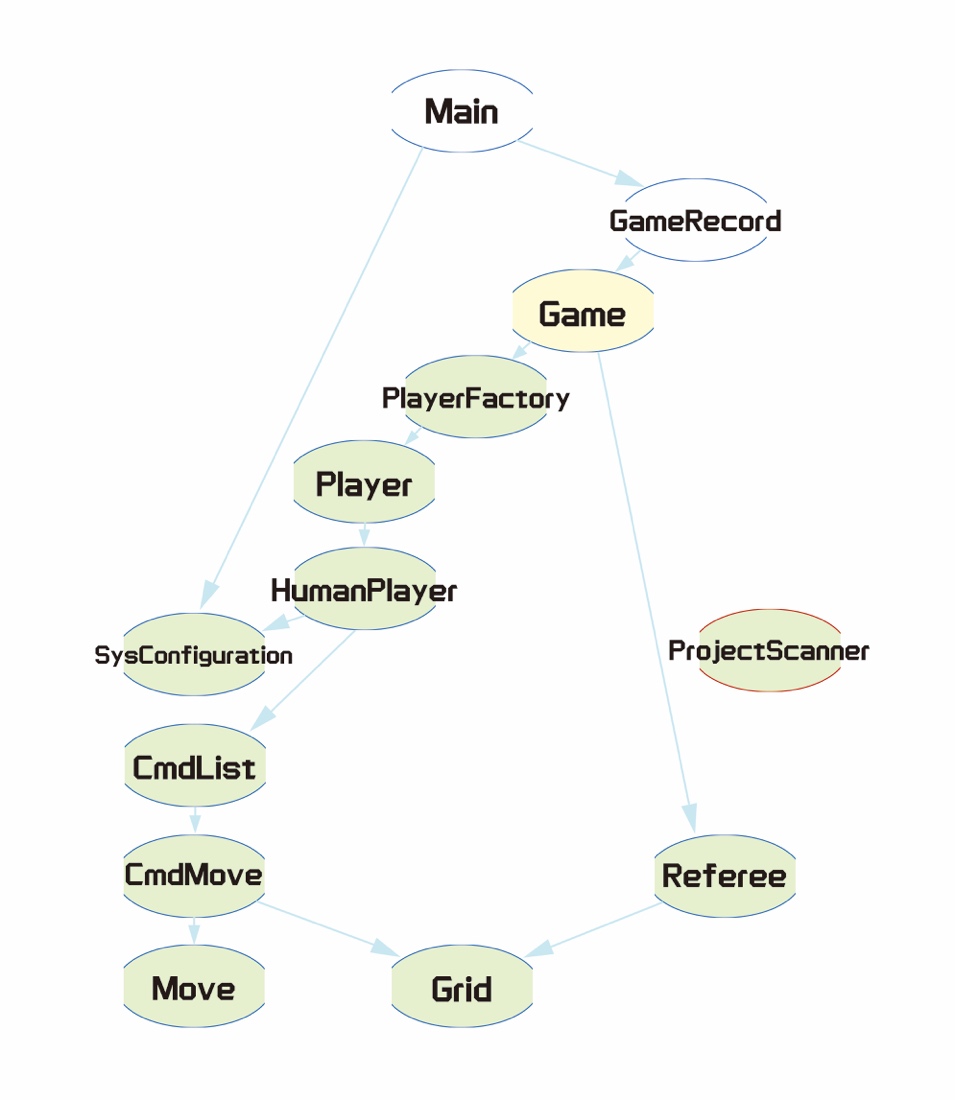


Figure 3.6 Integral tests in stage 5

6th stage:

* GameRecord + Game + PlayerFactory + HumanPlayer + SysConfiguration + CmdList + CmdMove + Move + Grid (Manage all games)

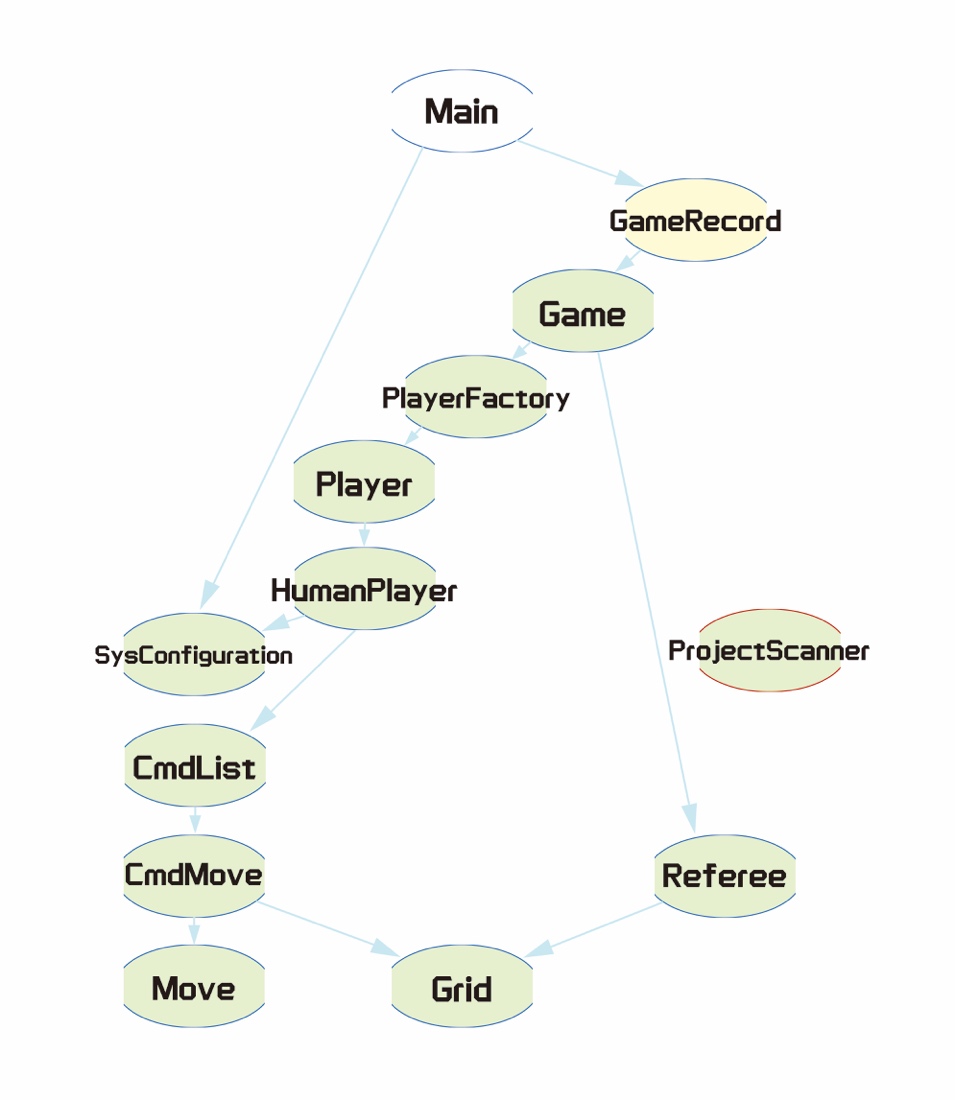


Figure 3.7 Integral tests in stage 6

After all integral tests are passed, we will conduct system tests to ensure the whole program is functional as expected from the user perspective.

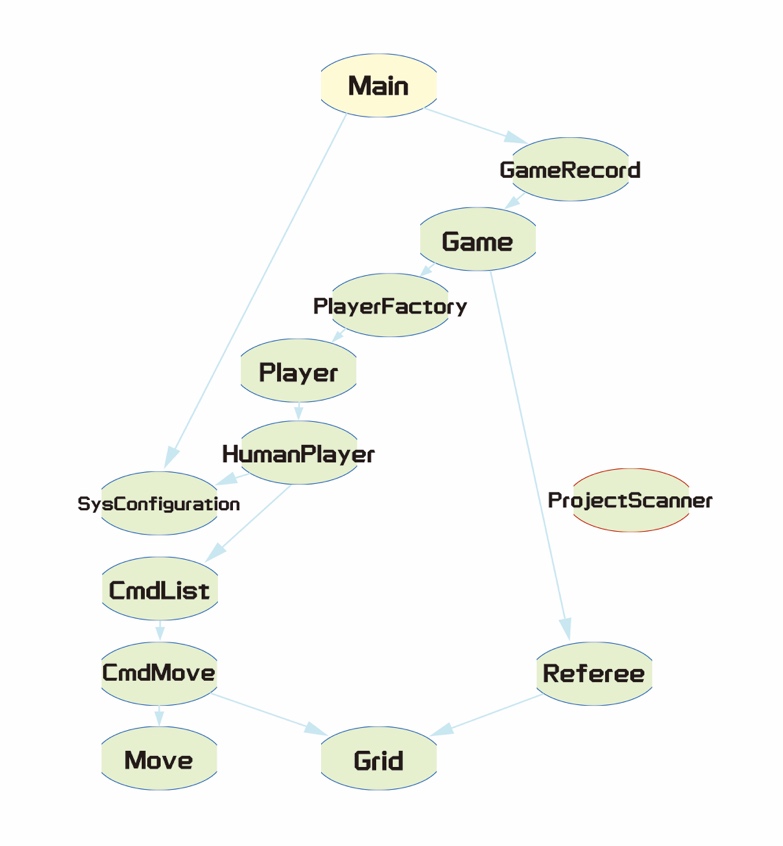


Figure 3.8 System tests

As we have already completed the basic functionality and ensured its smooth operation through testing in the first iteration, our testing process is greatly simplified in the 2nd iteration development. We only need to add test cases for the HumanPlayer class, which requests AI suggestions, and the PlayerFactory class responsible for player creation.

# Test Case Development

In our project, we have chosen to prioritize C/DC (Condition/Decision Coverage) as the primary consideration for test case development. C/DC coverage ensures that each condition and decision point within the system is thoroughly tested, covering all possible combinations and outcomes during the game.

By focusing on C/DC coverage, we aim to validate the correctness and reliability of the system. This approach allows us to identify potential flaws, inconsistencies, and vulnerabilities in the decision-making process, ensuring that the system behaves as intended under various conditions that happened in connect 4 games.

Below are some test cases for the Referee Class:

Test function: *isValidMove(Grid grid, int x)*

Usage : Check if the column x of grid is available

|  |
| --- |
| Program Source: |
| Truth table for the test cases T1-T5 from top to bottom:   |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | | grid | x | x < 0 | x > 6 | grid.getGrid()[0][x] != ‘ ‘ | x < 0 || x > 6 | Return | |  | -10 | true | false | false | true | [Input is out of range 0-6. Please try again.]  false | |  | 8 | false | true | false | true | [Input is out of range 0-6. Please try again.]  false | |  | 3 | false | false | true | false | [Column 3 is already full. Please try again.]  false | |  | 3 | false | false | false | false | true | |

Test function: *isFullGrid(Grid grid)*

Usage : Check if the grid is fully filled

|  |
| --- |
| Program Source: |
| Truth table for the test cases T1-T2 from top to bottom:   |  |  |  | | --- | --- | --- | | grid | grid.getGrid()[0][i] == ‘ ‘ | Return | |  | false | false | |  | true | true | |

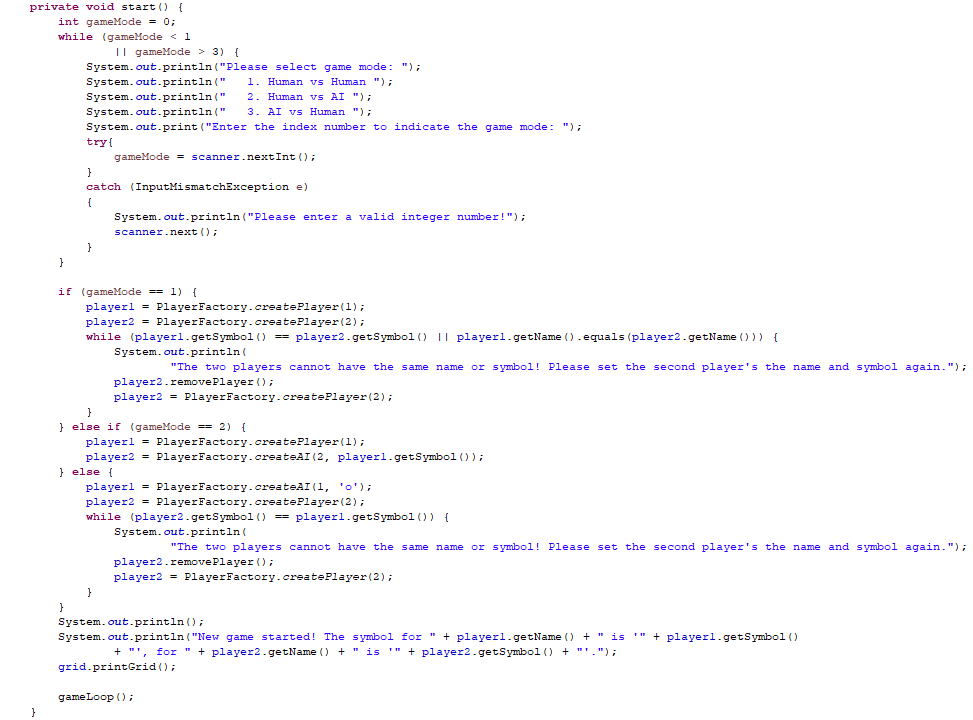
# Refactor

In the testing process, we have discovered server code smell issues. To improve the overall quality of the codebase, refactoring is required.

Some refactoring cases will be demonstrated in the following part, and analysis of the related refactoring processes will be discussed.

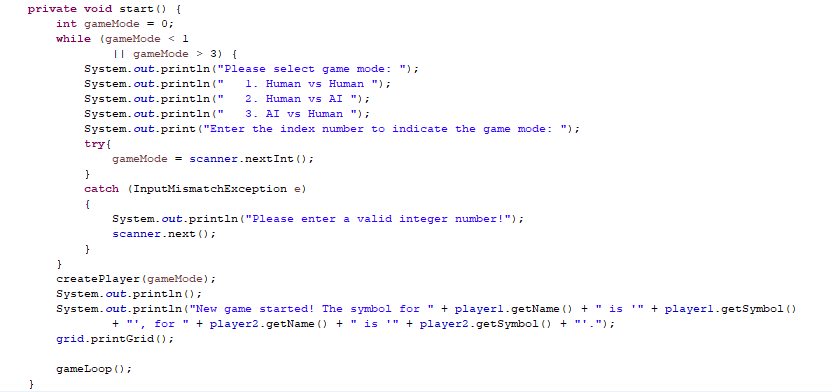
*Refactored code 1*

Before refactoring: Long method which is not easy to read or modify.

Figure 5.1 Game.start() before refactoring

Apply extract method

*After refactoring:*

Figure 5.2 Game.start() after refactoring

By extracting a portion of code from Game.start() into a separate method with a descriptive name *{Game.createPlayers(int gameMode)},* it improves code readability and understandability. Also, this makes the code more self-explanatory, as the extracted method’s name serves as a high-level summary of its functionality. This not only improves maintainability but also decreases the chances of introducing bugs when making changes.

*Refactored code 2*

*Before refactoring:* Complex calling between classes



Figure 5.3 create new game in Main() before refactoring

*After refactoring:*

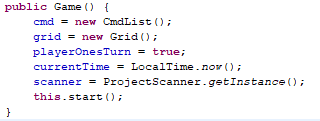


Figure 5.4 constructor of Game() after refactoring



Figure 5.5 GameRecord.addGame() after refactoring



Figure 5.6 create new game in Main() after refactoring

By improving the calling relationships between the classes, the code becomes more comprehensible and modifiable, reducing the intricacy caused by intertwined dependencies. This simplification facilitates smoother testing, debugging, and future enhancements, ultimately resulting in a more streamlined and modular system design.

# Coverage Analysis

Our team has diligently crafted our test cases, achieving 100% under Eclipse’ Branch Coverage. However, there are a couple of exceptional scenarios in the statement coverage that were not executed. These include the handling of InterruptedException when invoking java.lang.Thread in GamaGo and the handling of CloneNotSupportedException when utilizing the clone function in the Grid, which implements the Cloneable interface.

A screenshot of a graph

Description automatically generatedFigure 6.1 Statement Coverage in Eclipse

A screenshot of a computer

Description automatically generatedFigure 6.2 Branch Coverage in Eclipse

Overall, the test coverage analysis demonstrates that the project has achieved a commendable level of coverage across various criteria. The branch coverage of 100% indicates a thorough examination of the codebase, ensuring that most of the code paths have been tested. Moreover, the achieved test coverage provides confidence in the quality and reliability of the system, while the identified areas for improvement pave the way for future enhancements in testing practices.