**CONNECT 4**

**COMP 474**

**Prepared by:** Jarrod X.Banks, Asmina Akram, Syeda Nowreen,

Sudeshna Bhattacharya, and Michael Flatley

INTRODUCTION:

We have worked on a game named as “Connect Four”. It is a puzzle/logic game. Here players try to make 4 of a same color disks horizontally, vertically or diagonally. For developing this game we have followed agile methodologies and build it within 2 sprints (15 days each sprint) timeframe. We have used java as our programming language and implemented Graphical User Interface (GUI) to build the interface.

GAME OVERVIEW:  
 **Game Title:**  Connect 4  
**Genre:** Puzzle/Logic  
**Audience:** Ages 5+ can play the game  
  
**System Configuration/Game Requirement:**   
 **RAM:** 1.5 GB

**Processor:** Intel Celeron or higher

**Java Libraries:** LWJGL GUI library   
  
**Game Rules:**

* Each player puts a disc in a row alternately.
* The disc falls down straight and will only be stopped by another piece.
* The aim is to be the first of the two players to connect four discs of the same color vertically, horizontally or diagonally.
* If each cell of the grid is filled and no player has already connected four discs, the game ends drawn, so no player wins.

**Strategy to win:**

* One strategy to win this game is to prepare two chances to connect four discs at once, therefore setting up a double bind.
* This will consequently lead to the victory, since the player can only avoid one of the two chances.
* A perfect solution for a Connect4 game with 7 rows is already mathematically proven.
* Assuming a perfect play and best defense by both players the beginning player will always win by starting with the middle row.
* The decision to start at any other row will ultimately lead to a draw, if both players play perfect.

**Design Technique:**

The design technique that we undertook to develop our game is **Agile Methodology.**

Agile software development is a set of principles for [software development](https://en.wikipedia.org/wiki/Software_development) in which requirements and solutions evolve through collaboration between self-organizing, [cross-functional teams](https://en.wikipedia.org/wiki/Cross-functional_team). It promotes adaptive planning, evolutionary development, early delivery, and continuous improvement, and it encourages rapid and flexible response to change.

**Sprint Cycles:**

In our projectwe came across 2 Sprint Cycles in which we set different goals, some of which we were able to achieve and some we couldn’t. This project was very helpful in understanding the proper application of Sprint.

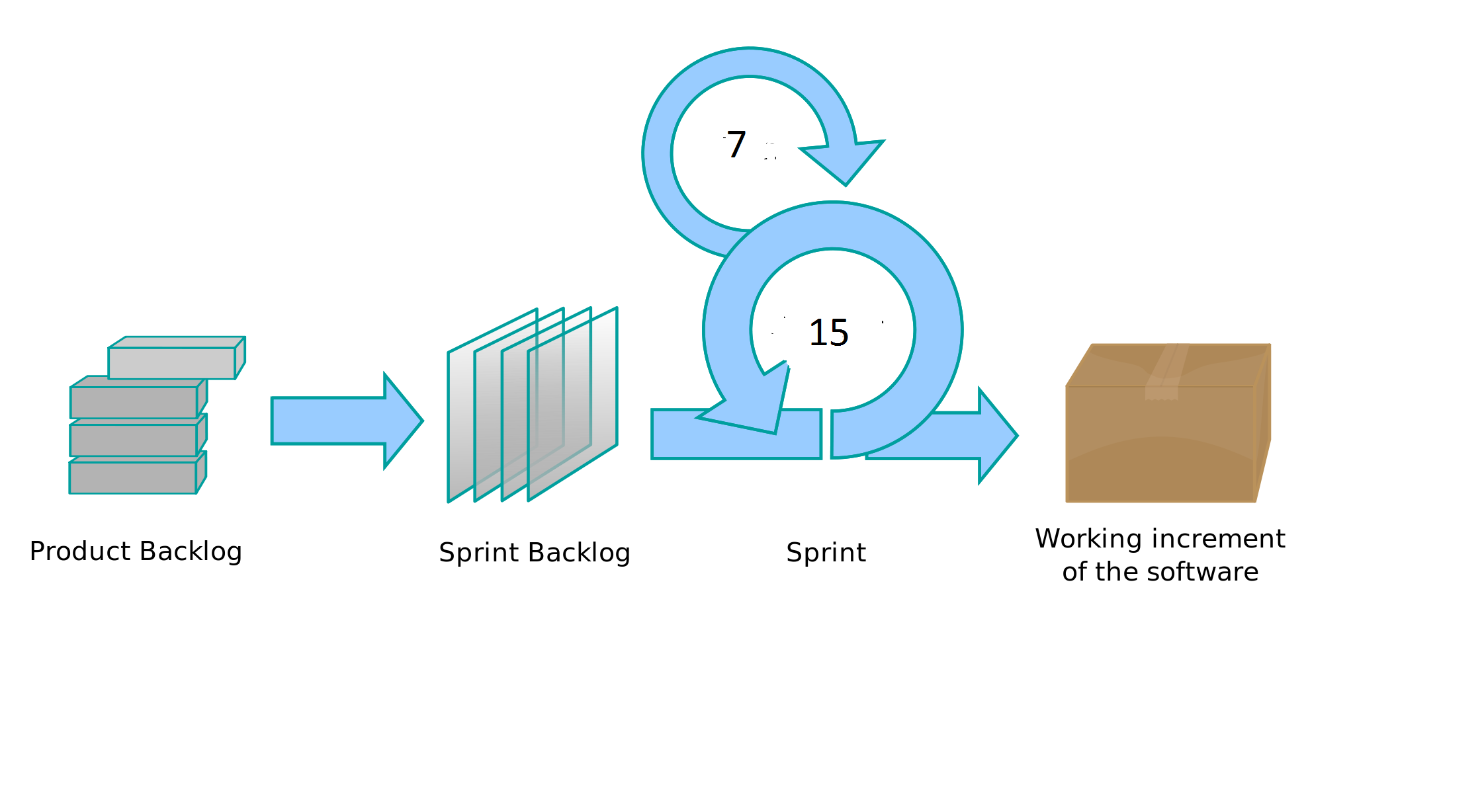


Fig: Our approach towards the module

**SPRINT CYCLE:**

We encountered 2 Sprint Cycles throughout our process of development of this game:

SPRINT 1:

Week 1 (7 April - 14 April)

* We came up with the basic game structure with some basic features necessary to play the game on console.
* At this point 2 players with all the basic necessities in working condition could play the game.
* We did not encounter any bugs at this point of time, so we planned for our next week’s priorities.

Week 2 (15 April - 21 April)

* At this stage we started working on the additional features along with concept of how to prepare the grid for the game.
* We even started building our grid using GUI .
* Successfully connected backend with the frontend.
* Started the documentation.

SPRINT 2:

Week 1 (21 April - 27 April)

* We were implementing mainly GUI and were trying to make our board look pretty by adding the button functionality to it so that the action is done when the user clicks the button.
* We created a reset button to our view, and added methods to help us move from main menu to the game. But for some unknown reason, our display started flickering.
* Code cleanup was done.
* We created a variable for the board so that, when the win has been calculated the board will pause for two seconds and will close the display.

Week2 (28 April - 5 May)

* We updated the push buttons for the push game mode.
* Added the necessary inline commenting.
* We faced memory leakage problem where the loop was running forever without any break point, the issue was fixed.
* Added Artificial Intelligence to our game level mode; with the easy level but the display flickering issue was not fixed.
* We then updated a button on the menu for our easy push AI mode.
* We faced another issue, where our game wasn’t closing even after hitting the X button on the upper right corner. We fixed this problem by adding an if statement in our main menu which made the display close.
* Our AI became little smart, to defeat the user.
* The test classes were added in our last week of the project development phase. Few of the classes were very complex to test and we could test them by seeing the display output screen.
* We tried to do clean up the redundant code and edited few methods.
* Finally, we fixed our Memory leak issue by updating the board when the left mouse button is clicked.
* Added a brief javadoc file for all the classes.
* Made our software product documentation with the product presentation slides.

**Project Description**

**How to run the project/game:**

1. Fetch/download the application from GitHub.
2. GitHubRepository: <https://github.com/8anksy/ConnectFour_Comp474Spring16.git>
3. Import it as project on eclipse/ IntelliJ/ or any IDE
4. All the libraries can be found on libs folder.
5. Run Main.java and enjoy the game!

**Features:**

**Regular Game:**  Here players try to match 4 same color pieces horizontally, vertically or diagonally. And they will drop the pieces from top and it will fall according to gravity.

**Push Game:** Here players try to match 4 same color pieces horizontally, vertically or diagonally but player can push pieces from the bottom.

**TwoPlayer Mode:** 2 players can play the game. **One Player Mode (Easy):** 1 player can play the game with our easy AI robot.

**Reset:** At any point players can reset the game and start from beginning.

**Artificial intelligence:** Our AI will first try to block the other player from winning. It will look for 2 same pieces of other player in a row then it will look in column. It will also try to win the game.

**REQUIREMENTS:**

|  |  |  |  |
| --- | --- | --- | --- |
| Functional Requirement | Status | Done by | Sprint |
| 1. Starting the game | Done | J Banks | 1 |
| 2. Place a checker in the board | Done | S Nowreen | 1 |
| 3.Find winner by checking horizontally, vertically and diagonally. | Done | S Nowreen | 1 |
| 4. Add Reset option. | Done | S Nowreen | 1 |
| 5.Switching Between Players | Done | S Nowreen | 1 |
| 6. Illegal Moves | Done | S Nowreen | 1 |
| 7. Board display | Done | M Flatley | 1 |
| 8. Menu Display | Done | M Flatley | 1 |

|  |  |  |  |
| --- | --- | --- | --- |
| 9. User Manual | Done | A Akram | 2 |
| 10.Testing user services | Done | J Banks | 2 |
| 11. Documentation | Done | A Akram, S Bhattacharyya, S Nowreen | 2 |
| 12.Presentation Slide | Done | A Akram, S Bhattacharyya | 2 |
| 14. There will be the following buttons:   * 1 player regular * 2 player regular * 1 player push * 2 player push * Exit | Done | M Flatley | 2 |
| 15. Ending the game | Done | M Flatley | 2 |
| 16. Push play style main menu buttons | Done | M Flatley | 2 |

|  |  |  |  |
| --- | --- | --- | --- |
| 17. Dropping pieces in the game (UI) | Done | M Flatley | 2 |
| 18. Easy mode AI | Done | S Nowreen | 2 |
| 19. Hard mode AI | Not Done |  | 2 |

**Non-Functional Requirement:**

|  |  |
| --- | --- |
| **Non Functional Requirement** | **Status** |
| 1.The average latency needs to be less than 50 millisecond for swapping move from one player to another. | Done |
| **2.** The system shall be stable and shall not go down, should have average downtime less than 0.02% | Not Done |
| **3.** The system shall calculate the winning move in less than 50 milliseconds. | Done |
| **4.** 100% documentation needs to be finished regarding how to play the game. | Done |
| **5.** Unit test coverage should be more than 80% | Not Done |

**TECHNICAL BARRIERS:**

BUG1:

Memory leakage and Bug and Fixation  
Date: April 27 2016 (second sprint)

Our memory Leakage was happening because we are always keep checking WinCheck in while (true) loop. So, it keeps running forever. There was no break point. It even starts checking for winner before the board implements.

|  |  |
| --- | --- |
| while (true) { | |
|  |  |
|  | WinCheck winCheck = new WinCheck(board); |
|  | char result = winCheck.getWinner(board); |
|  | if (result == 'D') { |
|  | System.out.println("It is a draw!"); |
|  | //board.isFinished=true; |
|  | break; |
|  | }  .  .  .  board.printBoard();  isRed =! isRed;   |  | | --- | | } |   Fix: The board has to be created first. After each player's turn win check has to be checked.  BUG 2:  Functionality Error:  For the following case Red should win. But it is asking for the Black’s turn. |
|  |  |

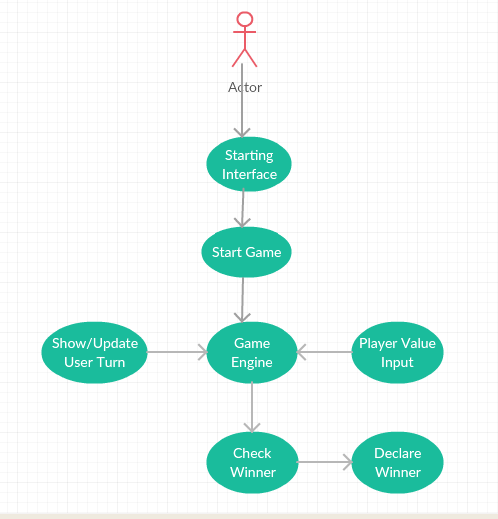
Fix: The way checkers are getting stored in the board was not doing it properly based on raw column array. It was storing in column raw array instead of raw column array.   
  
Date: May 1   
  
BUG 3: Reset Button is not working.  
Fix: The OnClick method was misplayed

BUG 4: Memory leakage. Memory is leaking from several places. Solving this creating screen flickering when the game starts.   
(Fixed)

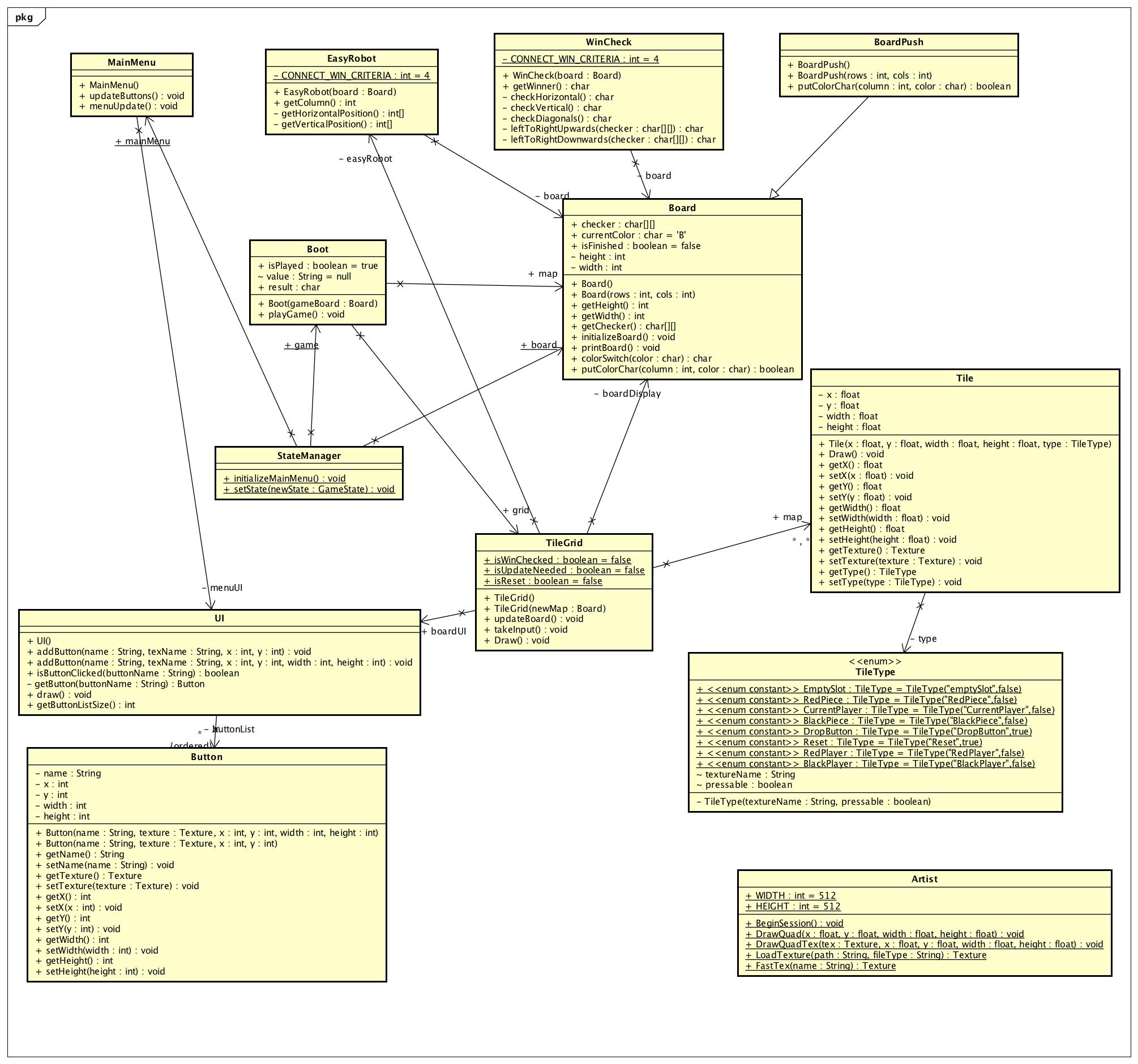
BUG 5: Game is not starting with a single touch on touchpad. Click is working fine. (Not Fixed)

BUG 6: Sometimes the display flickers (find appropriate word). May be because of library compatibility. (Fixed)

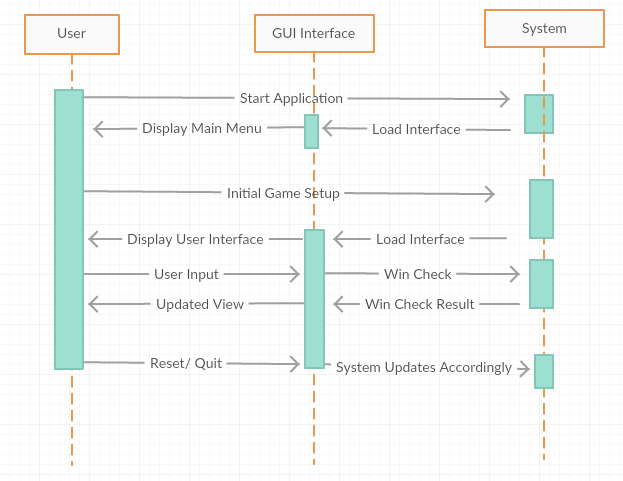
**USE CASE DIAGRAM:**



**CLASS DIAGRAM:**



**SEQUENCE DIAGRAM:**



**TESTING:**

**Acceptance Test:**

|  |  |  |
| --- | --- | --- |
| Task or Requirement | Passed | Failed |
| 1. Does the game starts on user click? | yes |  |
| 2. Does reset button works? | Yes |  |
| 3. Does the player switch after one player’s move? | Yes |  |
| 4. For 1 player game AI robot puts its pieces? | Yes |  |
| 5. Declare a winner after 4 of a kind? (Vertically, horizontally or diagonally? | Yes |  |
| 6. Push Game push pieces from bottom? | Yes |  |
| 7. Does it draw when no one wins? | Yes |  |

|  |  |  |
| --- | --- | --- |
| 8. Does all the menu buttons works? | Yes |  |
| 9. All the push button in the bottom works? | Yes |  |
| 10. Does clicking outside push button will not consider as a move? | Yes |  |

According to our acceptance test plan,   
Test Passes: 10 (100% pass rate)

Test Failed: 0 (0% Fail rate)

Testing Framework:

We tried to automate as many tests as possible using the JUnit 4 framework. The integration in the Eclipse IDE allowed us to test each class as a separate component. One difficulty we ran into while testing was deciding whether or not a test was necessary as component fell between the GUI and backend. Which part of that class should we test and which part can be manually tested visually? We were not able to achieve 80% test coverage due to time constraints and higher priority tasks.

**CODING STYLE:**   
  
We have coded in Java and used Eclipse as IDE. We tried to follow the standard coding style such as:

* Comments
* Indentation
* Line Breaks
* Naming convention
* Operator-operand spacing
* Alignment

We also used Javadoc for auto generated code documentation.

**SCREENSHOTS:**

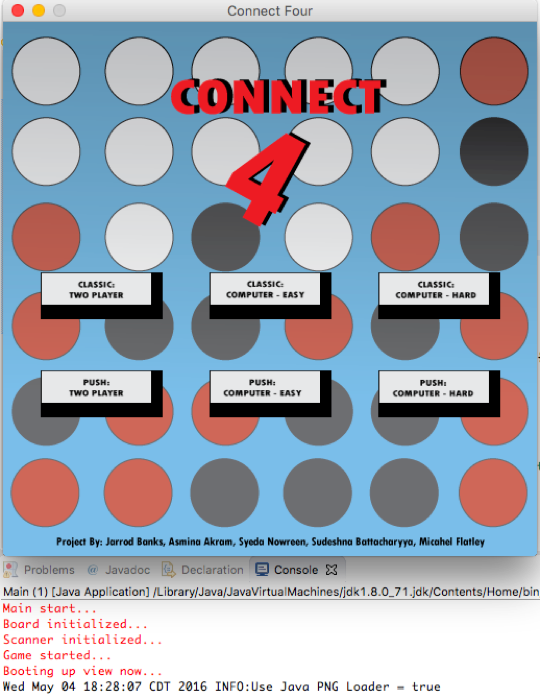


Figure (a): Menu

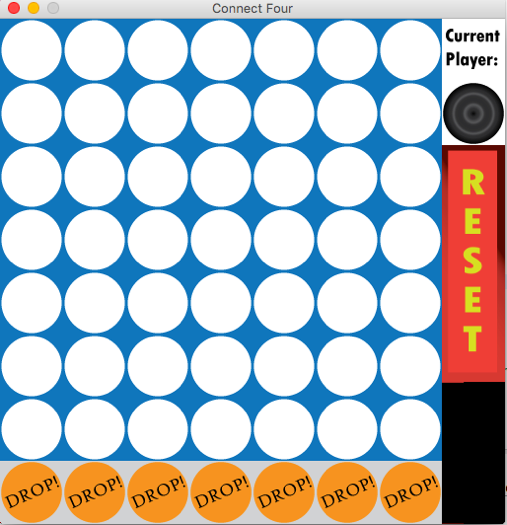


Figure (b): Grid

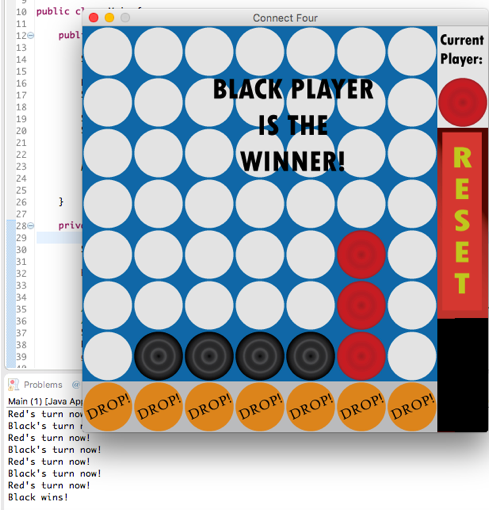


Figure (c): Winner Alert

**Remarks:**

* Adding new feature to this app is becoming expensive and time consuming because the app was not properly designed. Technical debt is very high for this app.
* Our design is not perfect. Because our system is tightly couples with excessive dependencies.
* AI is not that smart. It only checks vertically and horizontally not diagonally.

**Future Plans:**

* Better AI for the game.
* Going back to main menu from the play screen.
* User choosing grid size.
* Add timer to the game.
* User will select his/her preferred color.
* After game ends it should show start again/ return to main menu option.