Design Document

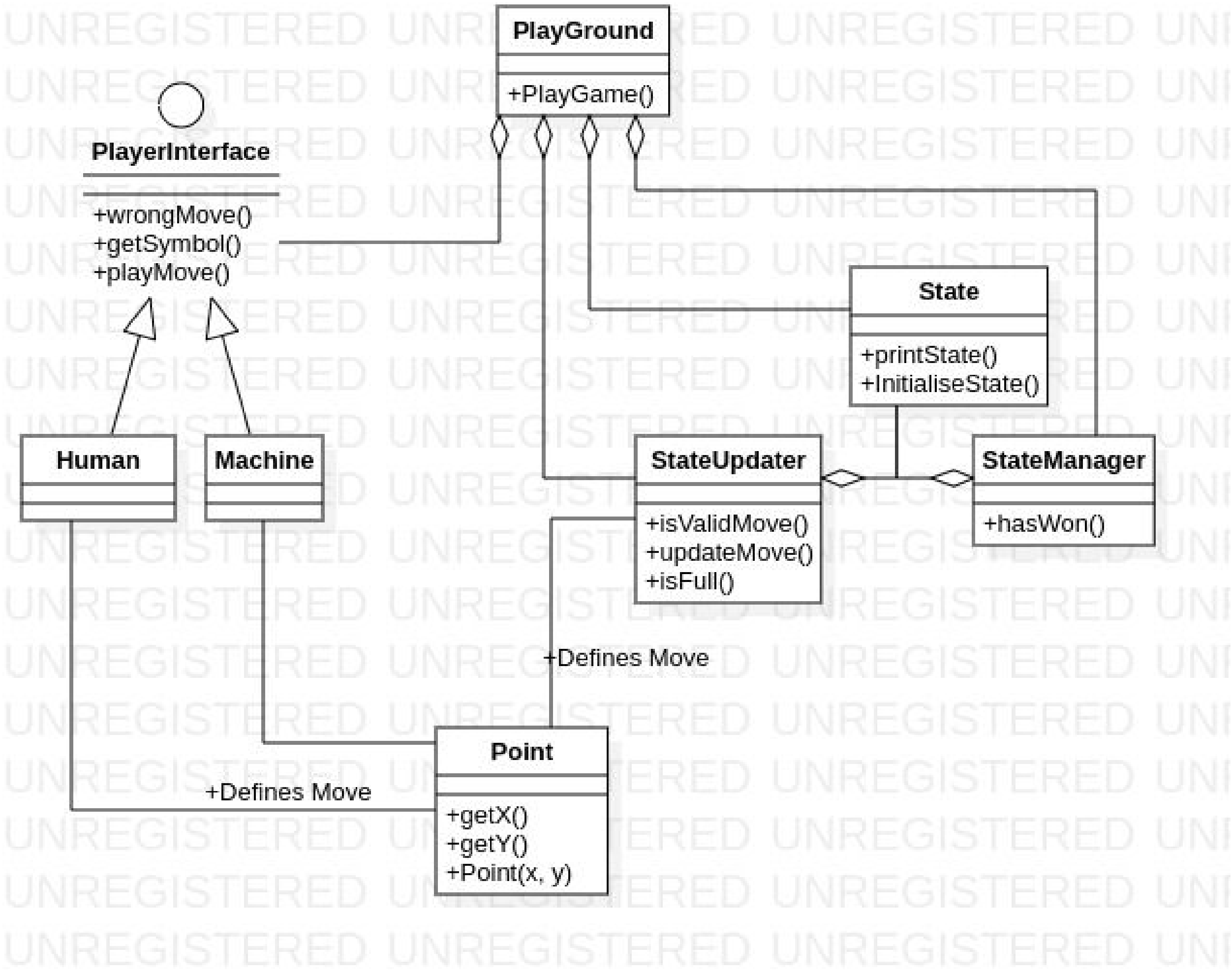
#138935 Karunam Goyal.

SECTION I: **First Iteration Game Project Design**

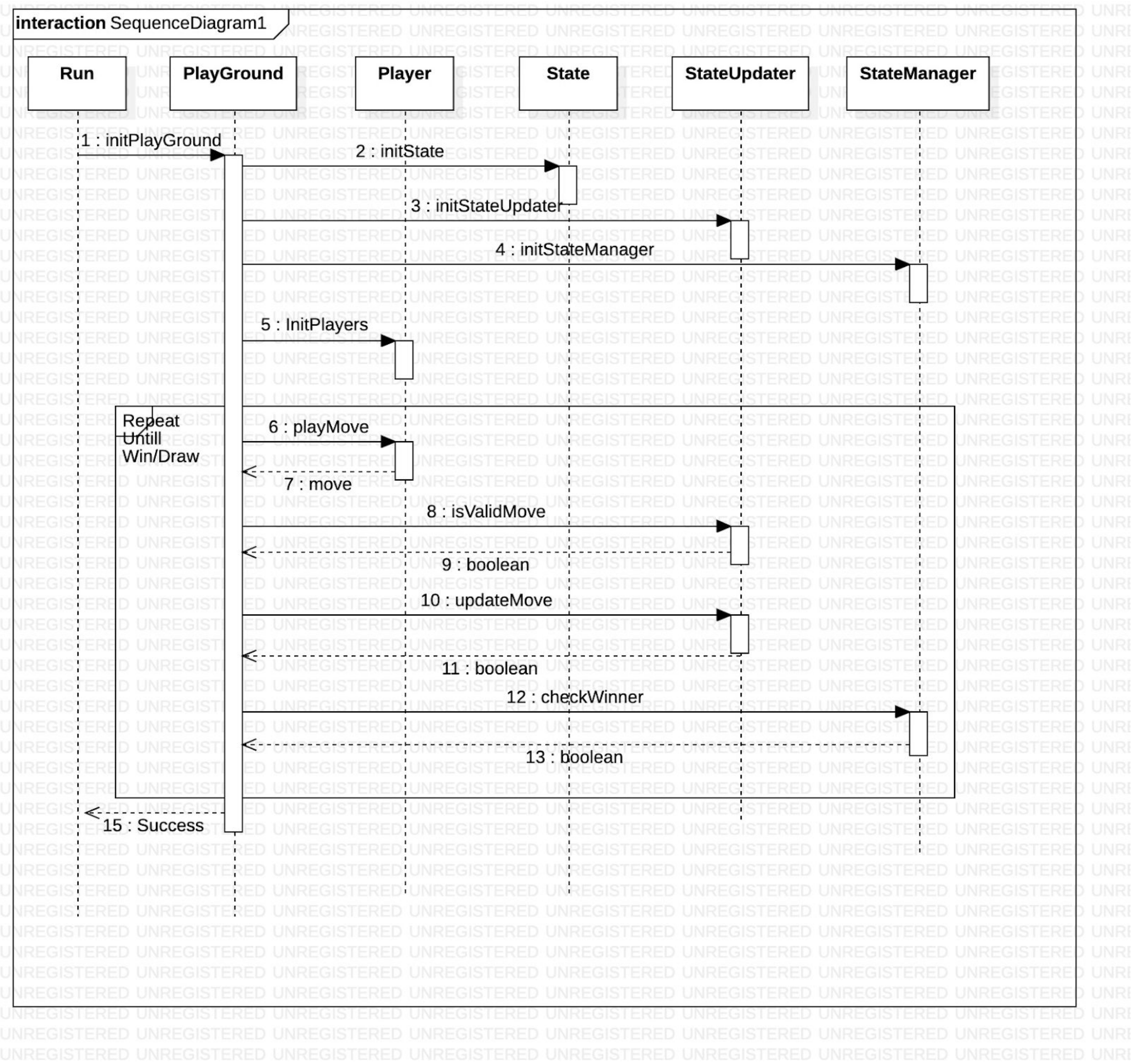
[Till Phase III or Phase IV Submitted Upto 29th January, 2020]

PART I :

Class Diagram



Sequence Diagram



PART II : **Design Desicions and assumptions**

Design was to make classes independent and group similar functions together. Whole idea was to create a sandbox-like structure where players will come and play on a board represented by a playground file and each class represents a task happening on the playground like managing the state of the board, etc. There are comments in the java files to tell what the purpose of that class is. You can find some detail at the end of this file or Readme.md

While making a back move we assumed that the player will go back only once because the player on the other side already made his move. Tried to deduce mathematical equations to map various levels and make moves.

PART III : **Feature Specific Design/Choices and Conventions/Assumptions**

GameDesign v2.0 - Requirement I

Made **Run.sh** and **Compile.sh** so that its easy for user

Completed - 1. Tic-Tac-Toe consists of 3x3 Square Cells

Basic 2-D Array of dynamic size and matching using loop so that can be extended to NxN keeping future requirement in mind. Tried to make State Updater and Manager separately which update and check winner respectively

Completed - 2. Game Between Two Humans

Two Instances of humans.

Completed - 3. Game Between Human and Machine

Machine uses Random Point for current implementation but can be changed to min-max algorithm easily without affecting any other section in the project structure.

Completed - 4. Winning Criteria - 3 Cells in Row/Column/Diagonal are in the Same State.

Used loop instead of manually checking making it extendable for future releases.

Completed - 5. Announce Winning Player

Printing the Winner if any and finishing the game after asking if he wishes to continue.

GameDesign v2.0 - Requirement II

Completed - 6. Enhanced Tic-Tac-Toe Game Consist of 9x9 Squares...

Already incorporated in previous design just entered n = 9

Completed - 7. Enhanced Tic-Tac-Toe will continue to expand in depth levels...

Used 3d Array- and Mathematical Equation to map Different levels with previous level

Completed - 8. Extend Game to 4x4 Board

Changed n = 4; user can enter whatever board size he wants

Completed - 9. Human Players are Biased...

Added a revert Move method

Partially Completed - 10. Storing and Retrieving Game State

You have to go back various moves Not implemented properly

Completed - 11. Store Players Game Statistics: Leaderboard

Designed Leaderboard class which will manage the score during the game and LeaderBoard Whole which will measure All the Games

GameDesign v3.0 - Requirement III

Completed - 12. Super Tic-Tac-Toe Game Extends Enhanced Tic-Tac-Toe Game...

Designed Hex Class and used 2*n-1 x 4*n-3 Size array and Mapped valid indices accordingly

Completed - 13. Design Winning and Losing Criterias On All Edges...

With checkWinner in StateManager.

Not Completed - 14. Incorporate Irregular shaped Hexagonal Boards

GameDesign v4.0 - Requirement IV

Completed - 15. Incorporate Biased Game Board

Added Revert Move Option

Not Completed - 16.Incorporate Connect Four Game In Design

Completed - 17. Discover Newer Abstract Types

Hex and TicTacToe had many similar functions so moved them to abstract class from which they were inheriting

Completed - 18. Refactor and Reuse Code In Both Games

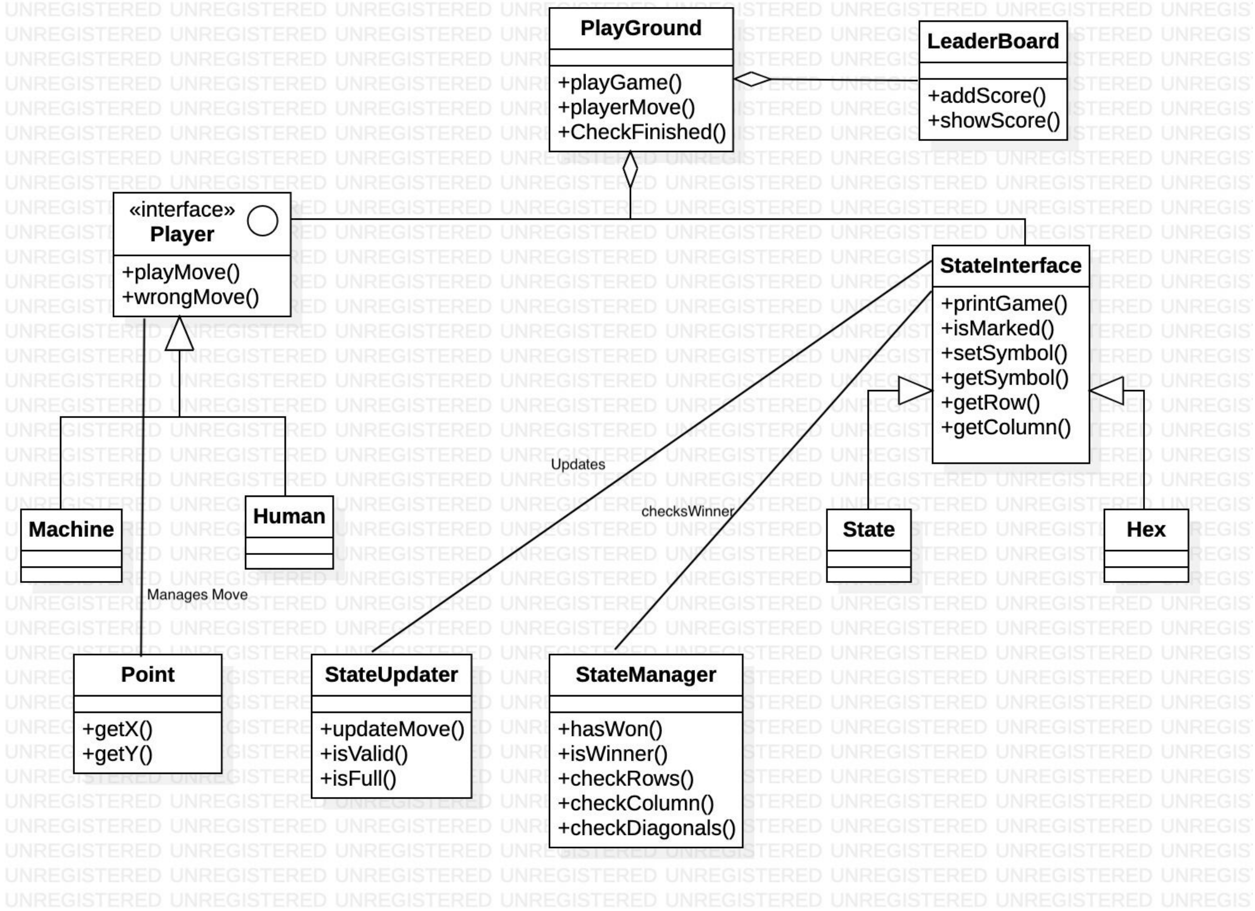
Refactored the code and one can see The Implementation in the github repository many methods are implemented in the parent class of TicTacToe and Hex Except Initialization and checkWinner in the state manager. Other than that everything was same

SECTION II: Second Iteration[Refactoring/Redesign] Game Project Design

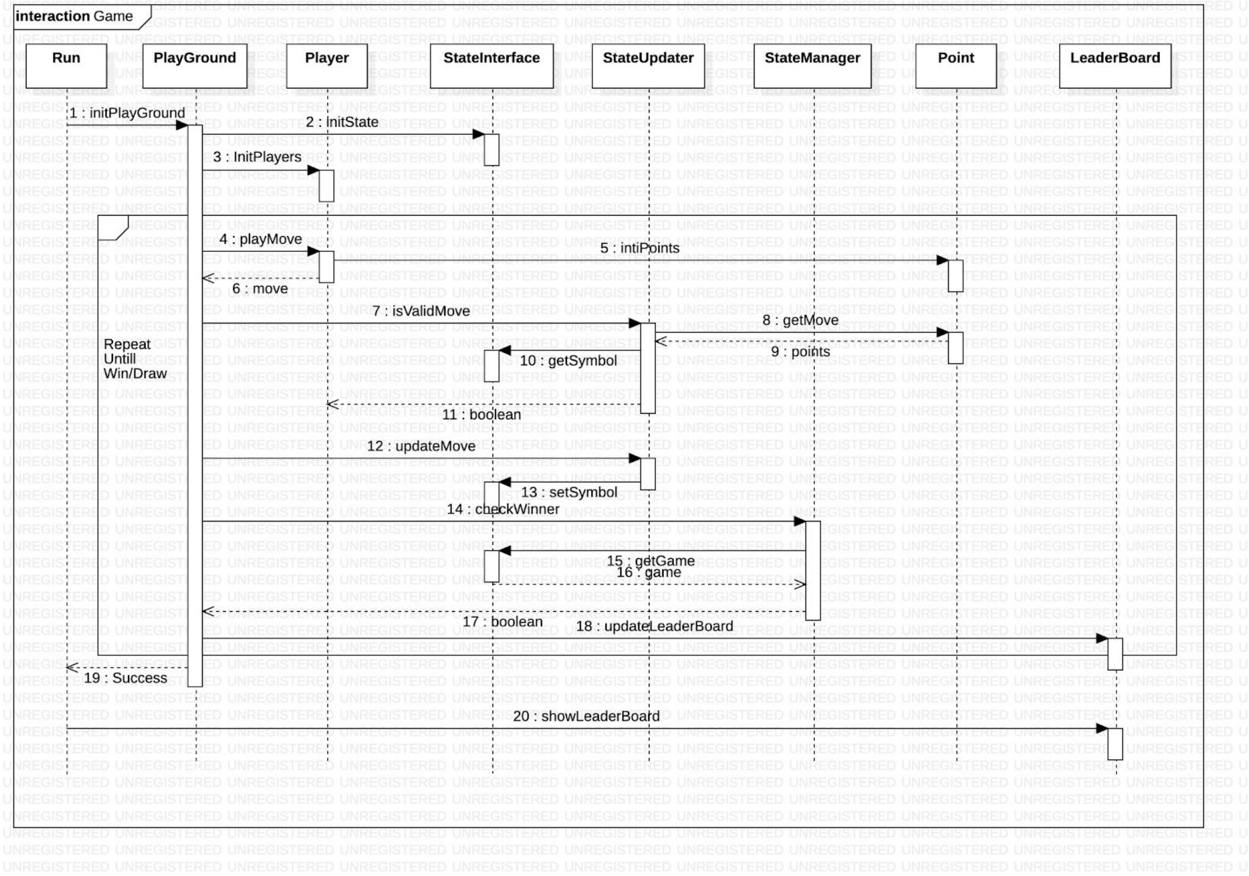
[Till Phase III or Phase IV Submitted Upto 03rd February, 2020]

PART I :

Class Diagram



Sequence Diagram



PART II : **Design Desicions and assumptions**

Difference from the previous design is reusing code more and doing some more encapsulation making it more maintainable in the long run. Some methods were similar so made them more general and thus used abstract class instead of interface for TicTacToe and Hex. The readability of code has increased. Tried to just make it cleaner. But the basic idea was the same as the previous design.

Design was to make classes independent and group similar functions together. Whole idea was to create a sandbox-like structure where players will come and play on a board represented by a playground file and each class represents a task happening on the playground like managing the state of the board, etc. There are comments in the java files to tell what the purpose of that class is. You can find some detail at the end of this file or Readme.md

While making a back move we assumed that the player will go back only once because the player on the other side already made his move.

```
Instance of a Game Being Played at a Moment
|''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''|
| |''''''''| |'''''''''|         |''''''''''''''|  |'''''''''''| |
| |Player1 | | Player2 |         |StateInterface|  |LeaderBoard| |
| |        | |         |         |              |  |           | |
| '''''''''' '''''''''''         ''''''''''''''''  ''''''''''''' |
|                                                                |
|                |''''''''''''''| |'''''''''''''|                |
|                | State Updater| | StateManager|                |
|                |              | |             |                |
|                '''''''''''''''' '''''''''''''''                |
|                           PlayGround                           |
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
```

PART III: Feature Specific Design/Choices, Conventions and Assumptions

GameDesign v2.0 - Requirement I

Completed - 1. Tic-Tac-Toe consists of 3x3 Square Cells

Basic 2-D Array of dynamic size and matching using loop so that can be extended to NxN keeping future requirement in mind. Tried to make a State Updater and Manager separately which updates and checks the winner respectively. Same as Previous design.

Completed - 2. Game Between Two Humans

Two Instances of humans. Same as Previous design.

Completed - 3. Game Between Human and Machine

Machine uses Random Point for current implementation but can be changed to min-max algorithm easily without affecting any other section in the project structure. Same as Previous design.

Completed - 4. Winning Criteria - 3 Cells in Row/Column/Diagonal are in the Same State. Used loop instead of manually checking making it extendable for future releases. Same as Previous design.

Completed - 5. Announce Winning Player

Printing the Winner if any and finishing the game after asking if he wishes to continue. Same as Previous class.

GameDesign v2.0 - Requirement II

Completed - 6. Enhanced Tic-Tac-Toe Game Consist of 9x9 Squares...

Already incorporated in the previous design just entered n = 9. Same as Previous class.

Completed - 7. Enhanced Tic-Tac-Toe will continue to expand in depth levels...

Now used State in State and made it more abstract rather than using 3d array and now symbol became a property of the class after refactoring.

Completed - 8. Extend Game to 4x4 Board

Humans can Enter the n value whatever he wants.

Completed - 9. Human Players are Biased...

Revert method to nullify the move Same as Previous design.

Partially Completed - 10. Storing and Retrieving Game State

Retrieve the state after various back moves not implemented correctly.

Completed - 11. Store Players Game Statistics: Leaderboard

LeaderBoard and LeaderBoardWhole Implementation Same as Previous Design

GameDesign v3.0 - Requirement III

Completed - 12. Super Tic-Tac-Toe Game Extends Enhanced Tic-Tac-Toe Game…

Implemented it similar to TicTactoe game. All the methods are the same except for initialization.

Completed - 13. Design Winning and Losing Criterias On All Edges…

Same as Previous design.

Not Completed - 14. Incorporate Irregular shaped Hexagonal Boards

GameDesign v4.0 - Requirement IV

Completed - 15. Incorporate Biased Game Board

Same as Previous Design

Not Completed - 16.Incorporate Connect Four Game In Design

Completed - 17. Discover Newer Abstract Types

The Code is now Refactored one can see the change in class diagrams

Completed - 18. Refactor and Reuse Code In Both Games

Refactored

SECTION III: GameDesign Project Feature and Test Cases Implementation Status and Description

Implemented Most Test Cases in MachineTest.java StateTest.java HexTest.java

Can be Run Through **RunTest.sh** After Compiling through **CompileTest.sh**

GameDesign v1.0 - Requirement I

Completed - 1. Tic-Tac-Toe consists of 3x3 Square Cells

TestCases

Test1 : Test Generation of 3x3 Board

StateTest.java

Test2 : Test Initialisation of 3x3 Board

StateTest.java

Test3 : Test Cell Generation

StateTest.java

Completed - 2. Game Between Two Humans

TestCases: StateTest.java

Partially Completed - 3. Game Between Human and Machine
Machine Points Validity Checked using CheckValid in MachineTest.java
Completed - 4. Winning Criteria - 3 Cells in Row/Column/Diagonal are in Same State.
TestCases:
Test1 : StateTest.java
CheckRows Columns Diagonals
Completed - 5. Announce Winning Player
StateTest.java

GameDesign v2.0 - Requirement II
Completed - 6. Enhanced Tic-Tac-Toe Game Consist of 9x9 Squares...
TestCases:
Test1 : StateTest.java
Completed - 7. Enhanced Tic-Tac-Toe will continue to expand in depth levels...
TestCases:
Test1 : StateTest.java
Completed - 8. Extend Game to 4x4 Board
TestCases:
Test1 :StateTest.java
Completed - 9. Human Player is Biased...
Test1 : StateTest.java
Not Completed - 10. Storing and Retrieving Game State

Partially Completed - 11. Store Players Game Statistics: Leaderboard
TestCases:
Test1 : HexTest.java and StateTest.java

GameDesign v3.0 - Requirement III
Completed - 12. Super Tic-Tac-Toe Game Extends Enhanced Tic-Tac-Toe Game...
TestCases:
Test1 : HexTest.java
Completed - 13. Design Winning and Losing Criterias On All Edges...
TestCases:
Test1 : HexTest.java
Not Completed - 14. Incorporate Irregular shaped Hexagonal Boards

GameDesign v4.0 - Requirement IV
Completed - 15. Incorporate Biased Game Board
TestCases:
Test1 : HexTest.java
Not Completed - 16. Incorporate Connect Four Game In Design
Not Possible - 17. Discover Newer Abstract Types

All the test cases are based on refactored code- 18. Refactor and Reuse Code In Both Games

SECTION III: How To Run

Clone https://github.com/karunamgoyal/FKApplyDesign.git

git checkout dev

Then Run Shell Script It Will Automatically Compile All the Files

./Compile.sh

Then To Run the Game Type

./Run

To Run Test First Compile Test with

./CompileTest.sh

then

./RunTest.sh

In your terminal

Enter Coordinates of the box and top left corner being 0 0 (Basic Coordinate Convention for Matrix)

Directory Structure of Java Files

```
+FKApplyDesign
  -+src
   -+Players
    - Player.java
    - Machine.java
    - Human.java
   -+State
    - LeaderBoard.java
    - LeaderBoardWhole.java
    - Hex.java
    - StateInterface.java
    - State.java
    - StateManager.java
    - StateUpdater.java
    - Point.java
   - PlayGround.java
   - Run.java
  - Compile.sh
  - Run.sh
```

Directory Structure of Class Files

```
+FKApplyDesign
  -+src
   -+Players
    - Player.class
    - Machine.class
    - Human.class
   -+State
    - LeaderBoard.java
    - LeaderBoardWhole.java
    - Hex.java
    - StateInterface.java
    - State.class
    - StateManager.class
    - StateUpdater.class
    - Point.class
   -+Playground
    - PlayGround.class
   - Run.class
  - Run.sh
```

Directory Structure of TestFiles

```
+FKApplyDesign
  -+src
   - MachineTest.java
   - HexTest.java
   - StateTest.java
  - RunTest.sh
  - CompileTest.sh
  -junit.jar
  -hamcrest.jar
```

Maintainability

CodeBase is Managed over multiple directories so it is easy for it to manage and maintain the Code for extension and later versions.

Playground Class Which Act As A Box Where Game is Played

LeaderBoard Maintains LeaderBoard

Player is an Interface act as a Base for Human and Machine Player

Human and Machine Class Plays their Moves and they have a symbol they play

StateInterface Act as an Abstract class for TicTacToe and Hex

StateUpdater as the name Suggests Contains a State Variable and Updates the state and checks its value

StateManager manages the State and Checks if Someone Has Won and returns accordingly

PlayGround Contains the Objects of Players, State, Manager, Updater and the game is played and it act as a box

Run Runs The PlayGround

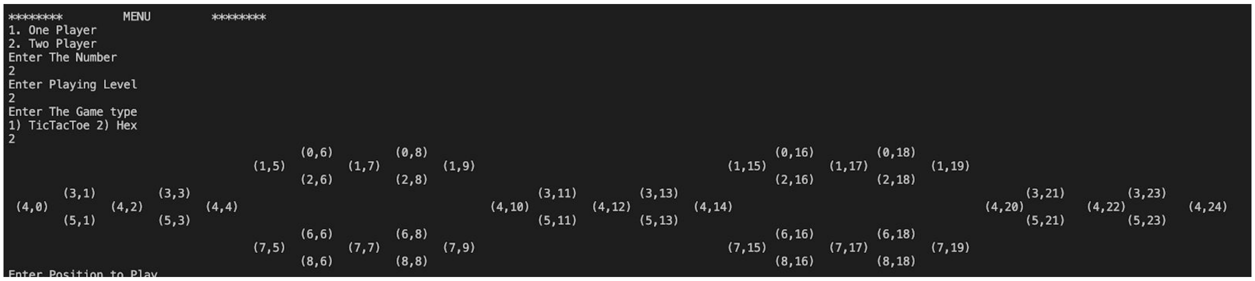
Functionality

- Multiple Players
- M X M Row For both TicTacToe and Hex
- Players can go to multiple Levels
- Scoring based on level
- Bias in the move - Revert move

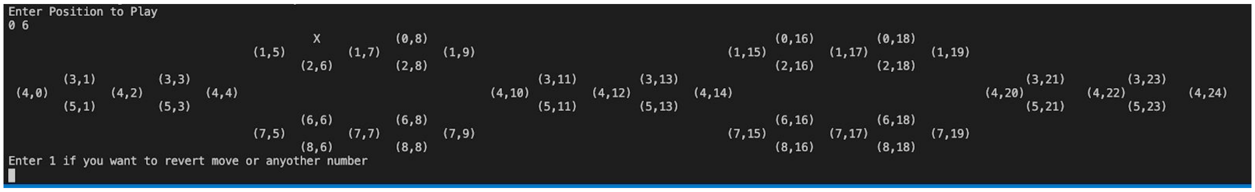
Test

```
Added Test using Junit and testing most critical classes and also
including corner cases
```

Sample Inputs

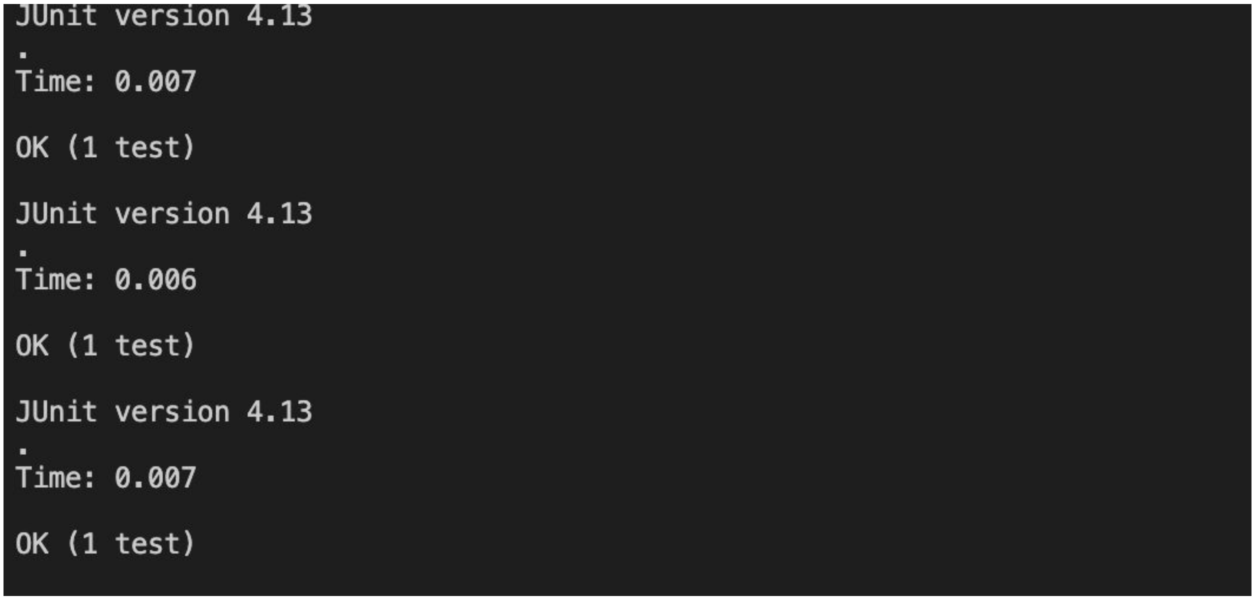


```
********        MENU        ********
1. One Player
2. Two Player
Enter The Number
2
Enter Playing Level
2
Enter The Game type
1) TicTacToe 2) Hex
2
                                           (0,6)         (0,8)                                                   (0,16)         (0,18)
                                    (1,5)         (1,7)         (1,9)                                     (1,15)         (1,17)         (1,19)
                                           (2,6)         (2,8)                                                   (2,16)         (2,18)
        (3,1)         (3,3)                                                   (3,11)         (3,13)                                                   (3,21)         (3,23)
 (4,0)         (4,2)         (4,4)                                     (4,10)         (4,12)         (4,14)                                     (4,20)         (4,22)         (4,24)
        (5,1)         (5,3)                                                   (5,11)         (5,13)                                                   (5,21)         (5,23)
                                           (6,6)         (6,8)                                                   (6,16)         (6,18)
                                    (7,5)         (7,7)         (7,9)                                     (7,15)         (7,17)         (7,19)
                                           (8,6)         (8,8)                                                   (8,16)         (8,18)
Enter Position to Play
```



```
Enter Position to Play
0 6
                                             X           (0,8)                                                   (0,16)         (0,18)
                                    (1,5)         (1,7)         (1,9)                                     (1,15)         (1,17)         (1,19)
                                           (2,6)         (2,8)                                                   (2,16)         (2,18)
        (3,1)         (3,3)                                                   (3,11)         (3,13)                                                   (3,21)         (3,23)
 (4,0)         (4,2)         (4,4)                                     (4,10)         (4,12)         (4,14)                                     (4,20)         (4,22)         (4,24)
        (5,1)         (5,3)                                                   (5,11)         (5,13)                                                   (5,21)         (5,23)
                                           (6,6)         (6,8)                                                   (6,16)         (6,18)
                                    (7,5)         (7,7)         (7,9)                                     (7,15)         (7,17)         (7,19)
                                           (8,6)         (8,8)                                                   (8,16)         (8,18)
Enter 1 if you want to revert move or anyother number
```

Test Results are OK Passed



```
JUnit version 4.13
.
Time: 0.007

OK (1 test)

JUnit version 4.13
.
Time: 0.006

OK (1 test)

JUnit version 4.13
.
Time: 0.007

OK (1 test)
```

Why I Think This Design is better

```
This Design is Good because I've Tried To Do Loose Coupling and use
Composition rather than inheriting the things.
This can be further extended to play on multiple systems after adding
locking mechanisms or similar Techniques
```