School of Information Technologies
Faculty of Engineering & IT

ASSIGNMENT/PROJECT COVERSHEET - INDIVIDUAL ASSESSMENT

Unit of Study: COMP9220 Object Oriented Design

Assignment name: Renju / Gomoku Game Assignment2

Tutorial time: 2015/4/24

Tutor name: Masa Takatsuka

DECLARATION

I declare that I have read and understood the *University of Sydney Academic Dishonesty and Plagiarism in Coursework Policy*, and except where specifically acknowledged, the work contained in this assignment/project is my own work, and has not been copied from other sources or been previously submitted for award or assessment.

I understand that failure to comply with the the *Academic Dishonesty and Plagiarism in Coursework Policy*, can lead to severe penalties as outlined under Chapter 8 of the *University of Sydney By-Law 1999* (as amended). These penalties may be imposed in cases where any significant portion of my submitted work has been copied without proper acknowledgement from other sources, including published works, the internet, existing programs, the work of other students, or work previously submitted for other awards or assessments.

I realise that I may be asked to identify those portions of the work contributed by me and required to demonstrate my knowledge of the relevant material by answering oral questions or by undertaking supplementary work, either written or in the laboratory, in order to arrive at the final assessment mark.

Student ID: 440563495

Student name: Hui Chen

Signed chen hui **Date** 2015/4/24

SIT Building, J12
The University of Sydney
NSW 2006 Australia

T +61 2 9351 3423
F +61 2 9351 3838
E sit.info@sydney.edu.au
sydney.edu.au/it

ABN 15 211 513 464
CRICOS 00026A

- Task 1

1. Refined Analysis

The original design analysis contain only a table, i add a few descriptions to illustrate the design idea behind more clear.

Gomoku game could be thought of consisting A play strategy and rules of determining the game results.
So, that strategy can be modelling to a Play class, It holds player information and necessary operations for handling the game rounds. It includes the count for each round game, this count of attribute was in the Check before, it would be better to increment the counter in the Play, because the objective of Check is to determine the status of the game and not necessary to interact with every round of the game, it just need to rely on the objects pass by the 'Play'. The Coordinate do not need to be a attribute of Play, since only purpose of this object is to pass to the Board instance. The getPlayer method is to retrieve the current player and pass the checkResult method.

The disk board in the game is modelling to Board Class, which holds a actual data structure of board. The setBoard method return boolean instead of void at the before, because the setBoard method have the coordinate check, if not passing the check, it would promote player and continue the current round, so it need to return a flag indicating the current round need to continue.

Rules can be modelling to a Check class. It holds the rules to determine the status of a game. The playerr is changed to winner, that is to retrieve and stored the winner in the checkResult method. The matrix do not need to store in the Check, the reason is the 'matrix' parameter of checkResult is passed by Gomoku every time of calling that method.

Those two class are the attributes in the Gomoku game class, encapsulate all the operations needed for the game.

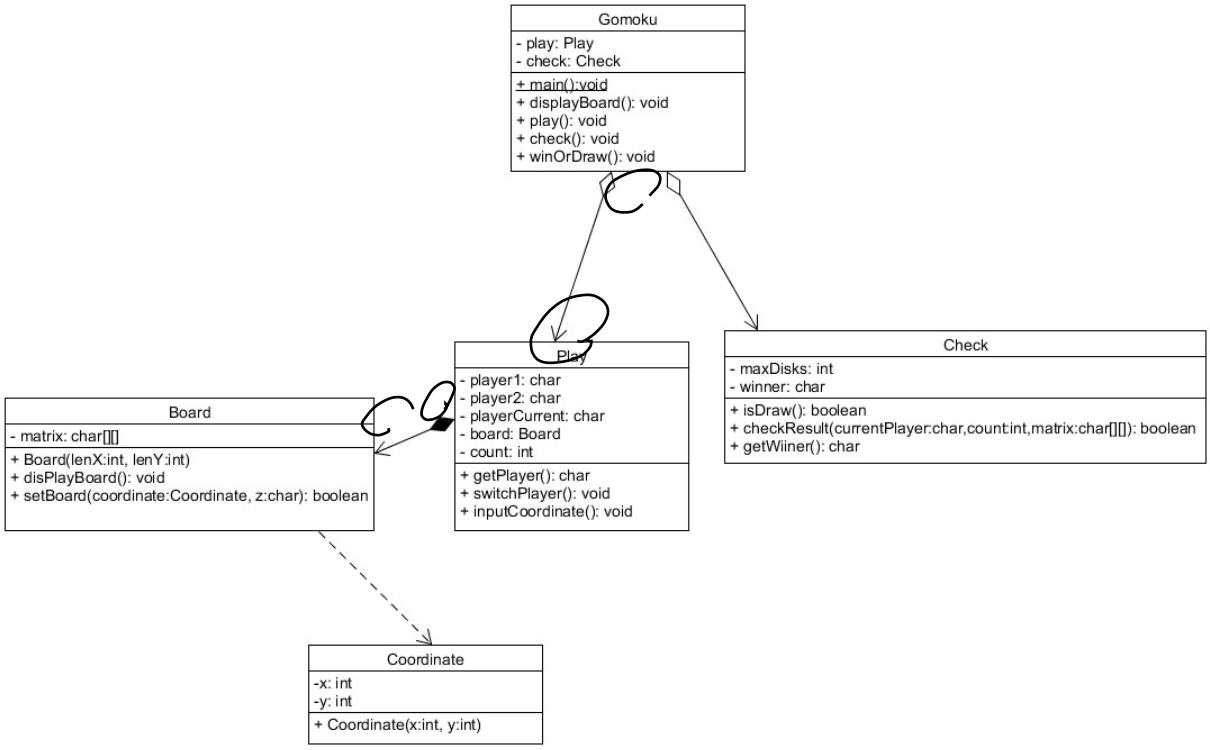
The following table lists the attributes and functions of classes based on the specifications.

No.	Specification	Analysis	Objects	Functions
1	When the software starts, it displays the empty 15 x 15 grids. At the each grid, either Black(X) or White stone (O) can be placed	Game board is a "board: char[15][15]". Two chars to fill the board: 'X', 'O'	board: Board in the Play Class	displayBoard(): void play(): void, SetBoard(coordinate:Coordinate, z:char): boolean in the Board class

2	Each player takes 32 disks and chooses one color to use throughout the game.	'X' and 'O' also present two player.	player1: char, player2: char, in the Play Class	
3	Black place a disk first followed by white placing a disk.	Player input coordinate (x:int, y:int) to player the game, if there was one disk in that place or the coordinate is out of the board, promote player type again	a Coordinate to be created to pass to setBoard method at each time	inputCoordinate(): void getPlayer(): char in the Play class
4	Players take turns in putting their disks on the board	When one player inputs a valid coordinate and the game does not stop, the system will switch player so that another player could input coordinate.	currentPlayer: char in the Play Class	switchPlayer(): void in the Play class
5	The game ends when one of player archived an unbroken row of five stones in the same color.	When one player inputs a coordinate, the system will check if he wins. max Disks is 64, and count in the Play instance gets to 64 and no one wins, then the game is draw.	A Check object has check operations for determining the result of a game. attributes: maxDisks: int winner: char	checkResult(currentPlayer:char, maxtrix:char[][]): boolean getWinner(): char, isDraw(): boolean in the Check class

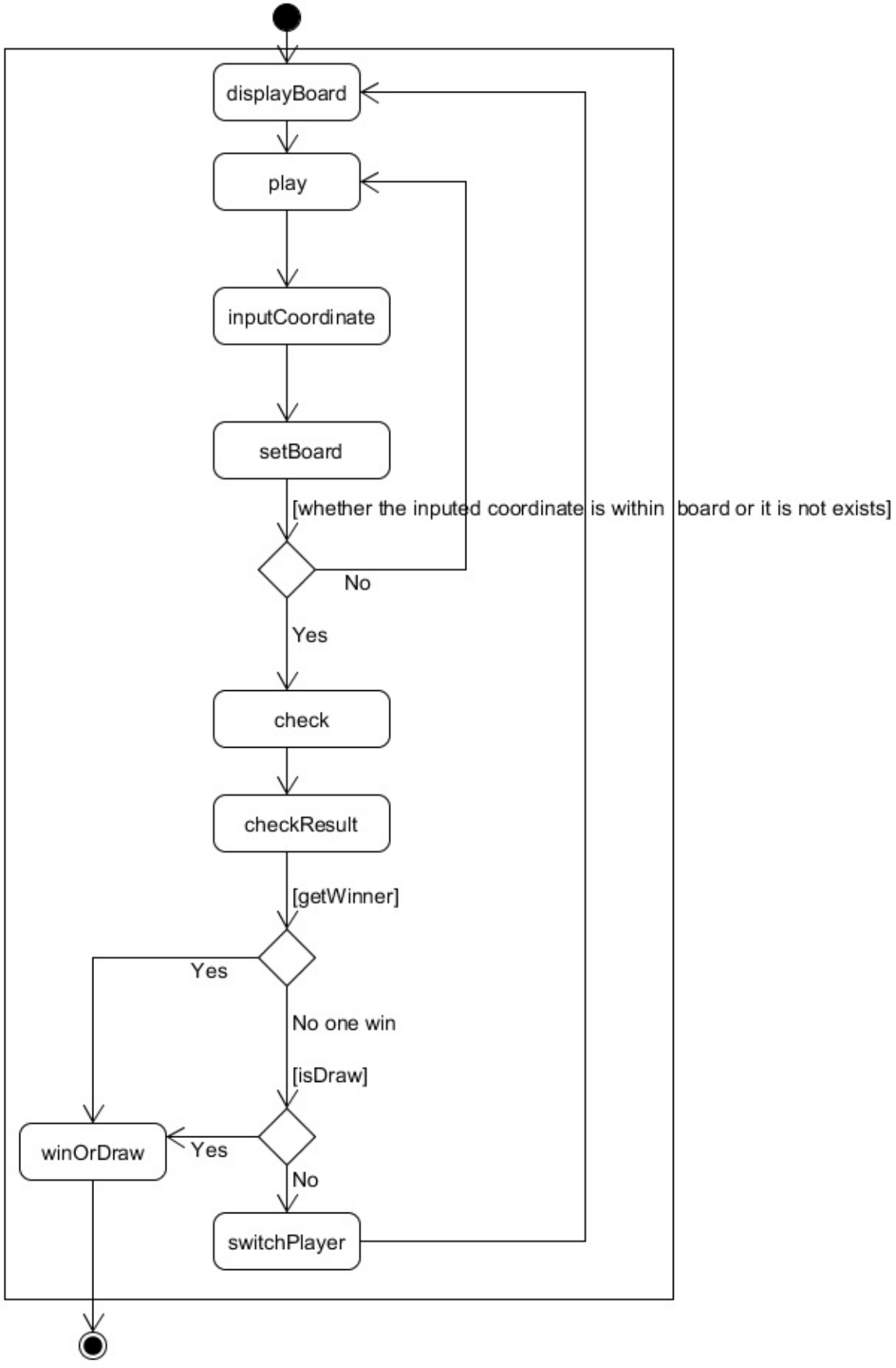
2. Refined class diagram

The class diagram is modified to correct some relationship between classes, ensure write the class correctly. The Gomoku class would contain Play and Check private instance, the relation between them is aggregation. Board instance is contained in the Play in order to encapsulate the operations of manipulate the game board. And also, the coordinate object used in the Play instance is to pass to the Board instance, there is a dependency between Board and Coordinate.



3. Refined activity diagram

The modification of the activity diagram contains the function call at each stage, instead of some notes on the diagram, make more clear on how to
write the program logic. The operation between play and Check valid step should contain the inputCoordinate method of Play class and setBoard method of Board class. And also Check valid step logic is placed inside the setBoard method. Afterwards, the check method of Gomoku is called, what's inside is calling the checkResult of Check class. Inside of checkResult, the getWinner is to determine the winner if exists, if not then the isDraw is called to see if the game could continue. If those checks are return 'Yes', the game would go to end and print out the results. Otherwise it calls switchPlayer to swap the player for next round.


displayBoard
play
inputCoordinate
setBoard
[whether the inputed coordinate is within board or it is not exists]
No
Yes
check
checkResult
[getWinner]
Yes
No one win
[isDraw]
winOrDraw
Yes
No
switchPlayer

- Task 2

1. **Improved Analysis**:

The main idea of this improvement design using inheritance and polymorphism is to abstract the the Operations in the game to Interface and abstract class.

At first, 'Check' class could be abstracted to a abstract class called 'CheckRule'. It contains a check result attribute and one abstract method 'check', which takes required objects as parameter to apply the game rules. Its sub class is 'GomokuPlayCheck'. It is almost the same as the previous 'Check' class, despite the a 'check' method takes over the previous 'checkResult' method, and its parameter changes to Map instead of those 3 parameters before, and would set the 'checkResult' as a message would return to the front and a flag indicating a game is finished. 'isDraw' and 'winOrDraw' method which previous in 'Gomoku' are set to private, they acts as helper functions of 'check' method. The ‘getWiiner’ method is eliminated because ‘wiiner’ attribute only use in this class. The constructor of 'GomokuPlayCheck' need a maximum number of disks passing in.

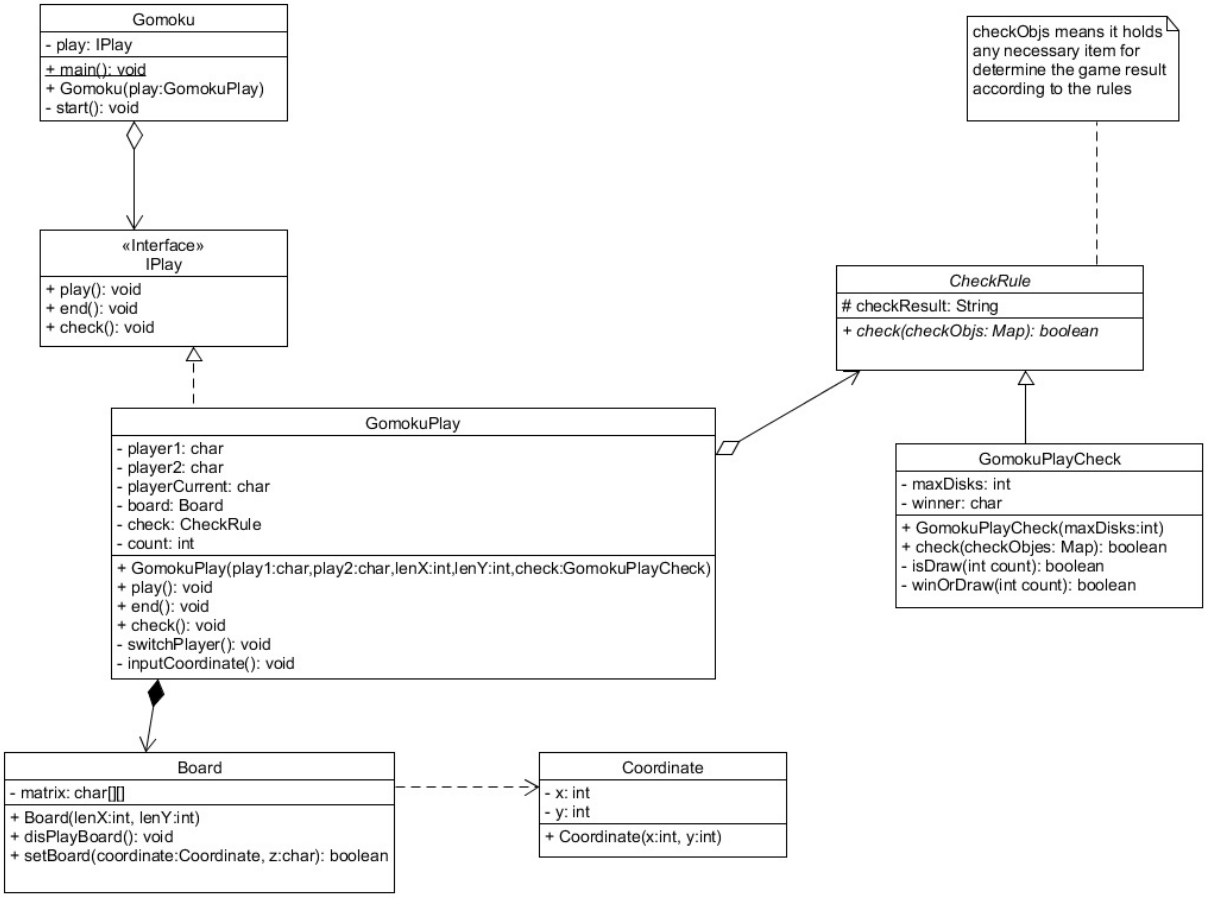
Secondly, 'Play' class could be abstracted to a Interface called 'IPlay', which only contain the methods signature needed to handling the game strategies, defining the operations involved. In this case, a play method is defined to
how to handle the player operations; a check method is defined to determine the status of the game; a end method is defined to do the operations after the game going to the end. The real implementation class is 'GomokuPlay'. It is
almost the same as previous 'Play' class, despite that it has 'CheckRule' interface as an attribute which would be used in the 'check' method. The reason of not having 'CheckRule' abstract class in 'Gomoku' class is that, if we do so, 'GomokuPlay' or even 'IPlay' would have a method for returning the board object in order to pass to the ‘CheckRule’, such as getting the internal states of the game playing, but internal states should only be manipulated by 'GomokuPlay' without any visibility to outside. The 'switchPlayer' and 'inputCoordinate' methods are set to private, they only acts as helper functions of 'play' and 'check' methods. The ‘getPlayer’ method is eliminated, because ‘playCurrent’ attribute only manipulate in this class, do not need to return to outside. The constructor would need the colour character of 2 players, the horizontal and vertical length of board to initialise a 'board' object and a real implementation class of 'CheckRule'.

What's more, 'Gomoku' could be simplified to just one 'IPlay' interface as an attribute, the constructor takes the real implementation class to initialise 'IPlay', and a private 'start' method to call the methods defined in 'IPlay' interface.

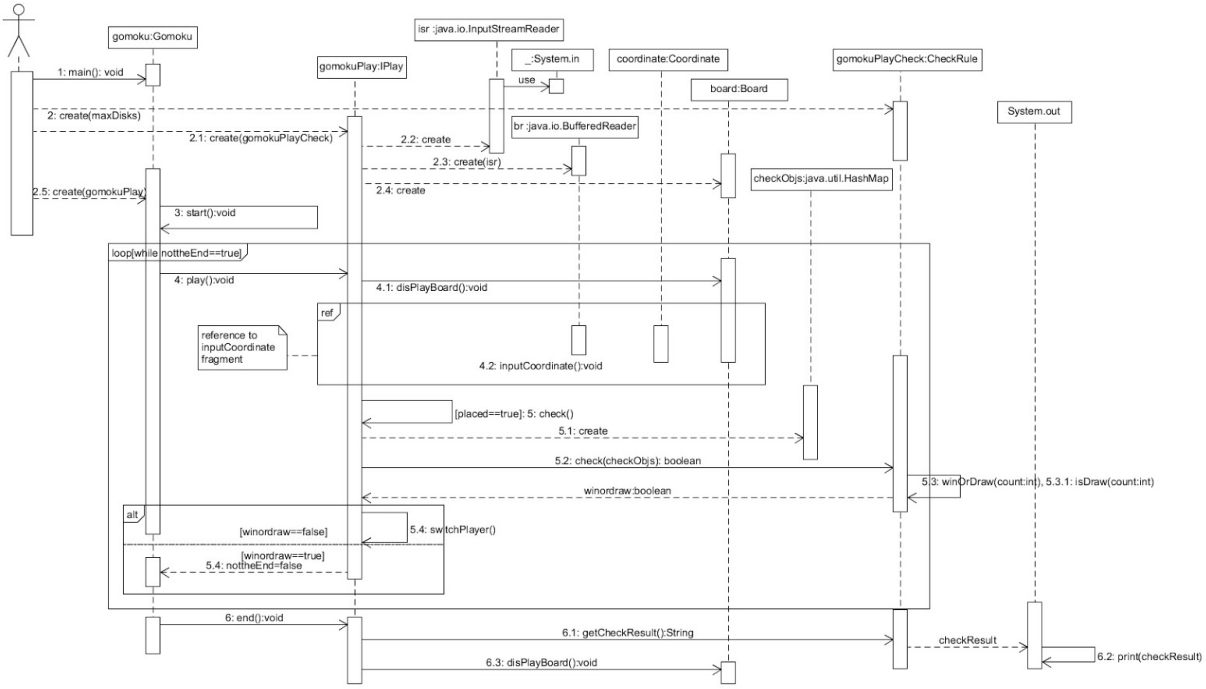
GomokuPlay provides the different implementation of game play strategies, that contributes to polymorphism.

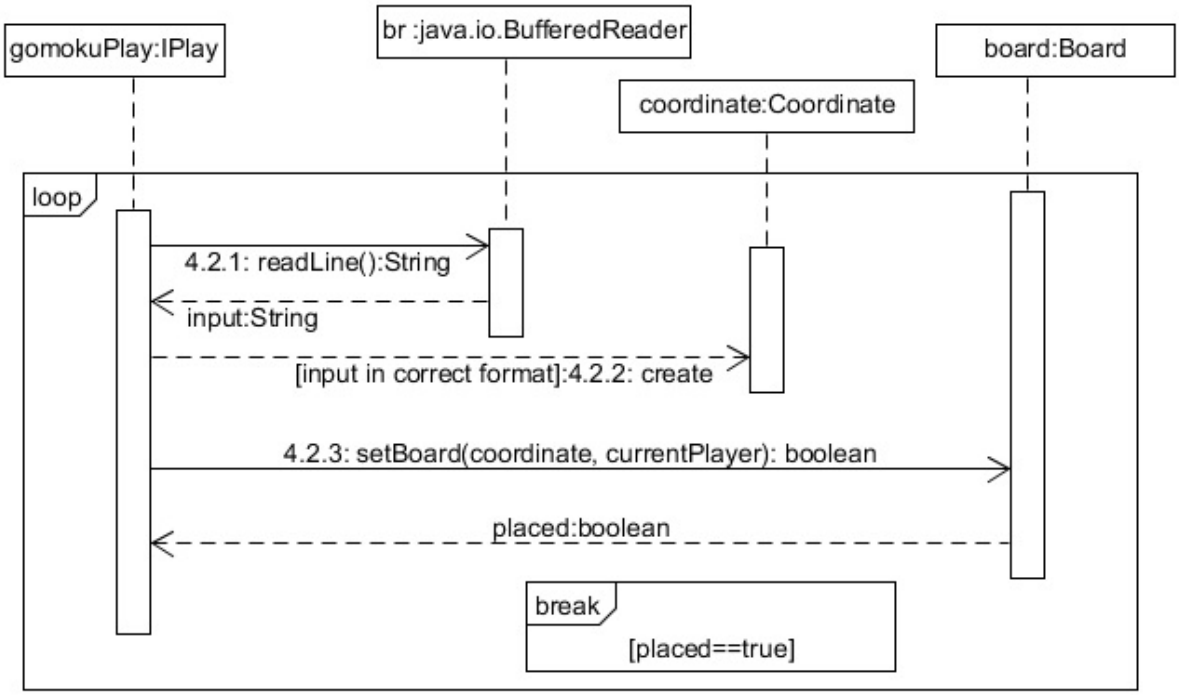
GomokuPlayCheck inherits 'CheckRule' gives the different implementation of abstract methods of parent class,
it also contributes to polymorphism, could make other new implementation of abstracted methods to apply different rules.

2. Improved class diagram



3. Improved sequence diagram


gomoku:Gomoku
isr :java.io.InputStreamReader
gomokuPlay:IPlay
_:System.in
coordinate:Coordinate
gomokuPlayCheck:CheckRule
board:Board
System.out
br :java.io.BufferedReader
checkObjs:java.util.HashMap
1: main(): void
use
2: create(maxDisks)
2.1: create(gomokuPlayCheck)
2.2: create
2.3: create(isr)
2.4: create
2.5: create(gomokuPlay)
3: start():void
loop[while nottheEnd==true]
4: play():void
4.1: disPlayBoard():void
ref
reference to inputCoordinate fragment
4.2: inputCoordinate():void
[placed==true]: 5: check()
5.1: create
5.2: check(checkObjs): boolean
5.3: winOrDraw(count:int), 5.3.1: isDraw(count:int)
winordraw:boolean
alt
[winordraw==false]
5.4: switchPlayer()
[winordraw==true]
5.4: nottheEnd=false
6: end():void
6.1: getCheckResult():String
checkResult
6.2: print(checkResult)
6.3: disPlayBoard():void


gomokuPlay:IPlay
br :java.io.BufferedReader
coordinate:Coordinate
board:Board
loop
4.2.1: readLine():String
input:String
[input in correct format]:4.2.2: create
4.2.3: setBoard(coordinate, currentPlayer): boolean
placed:boolean
break
[placed==true]