CS3343 Software Engineering
Connect Four

Leung Kin Chung	54028180
Ma Kei Tat	53967959
Pau Kwok Lun	54115681
Tong Pang Tat	54115693

Connect Four

- Two-players
- Suspended grid
- Discs

Objectives

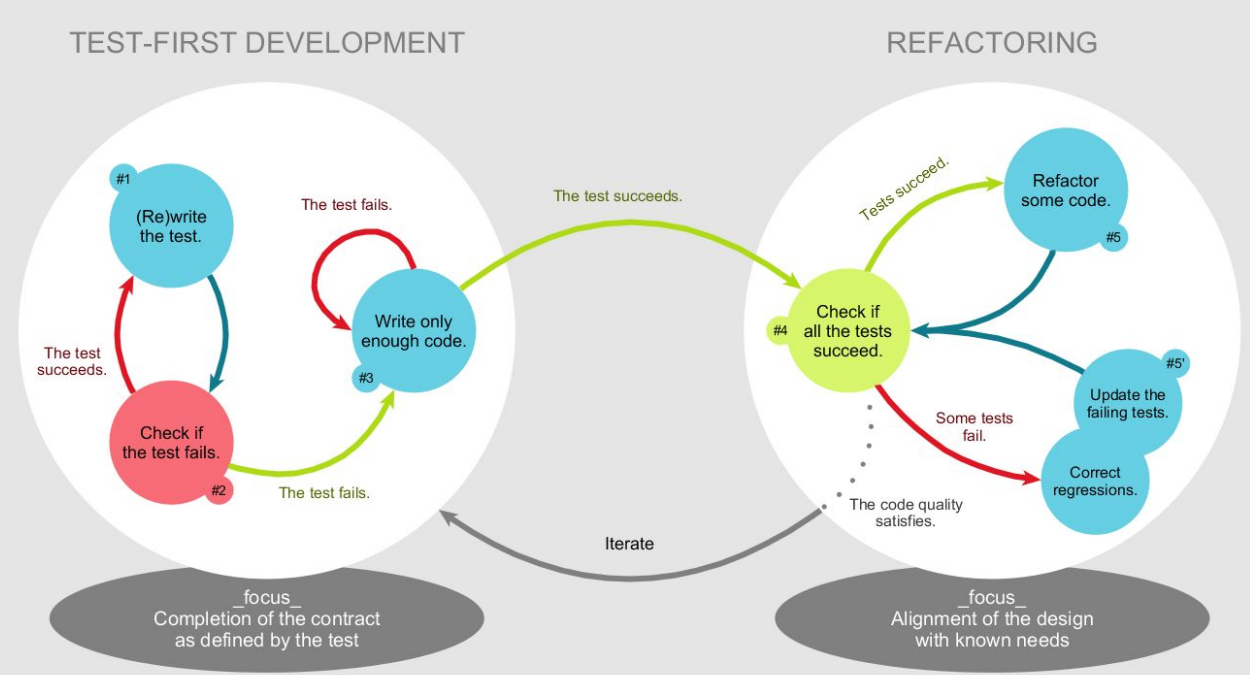
- Entertainment and brain training
- Discovering algorithm
- Project management

Project Team Orgranization

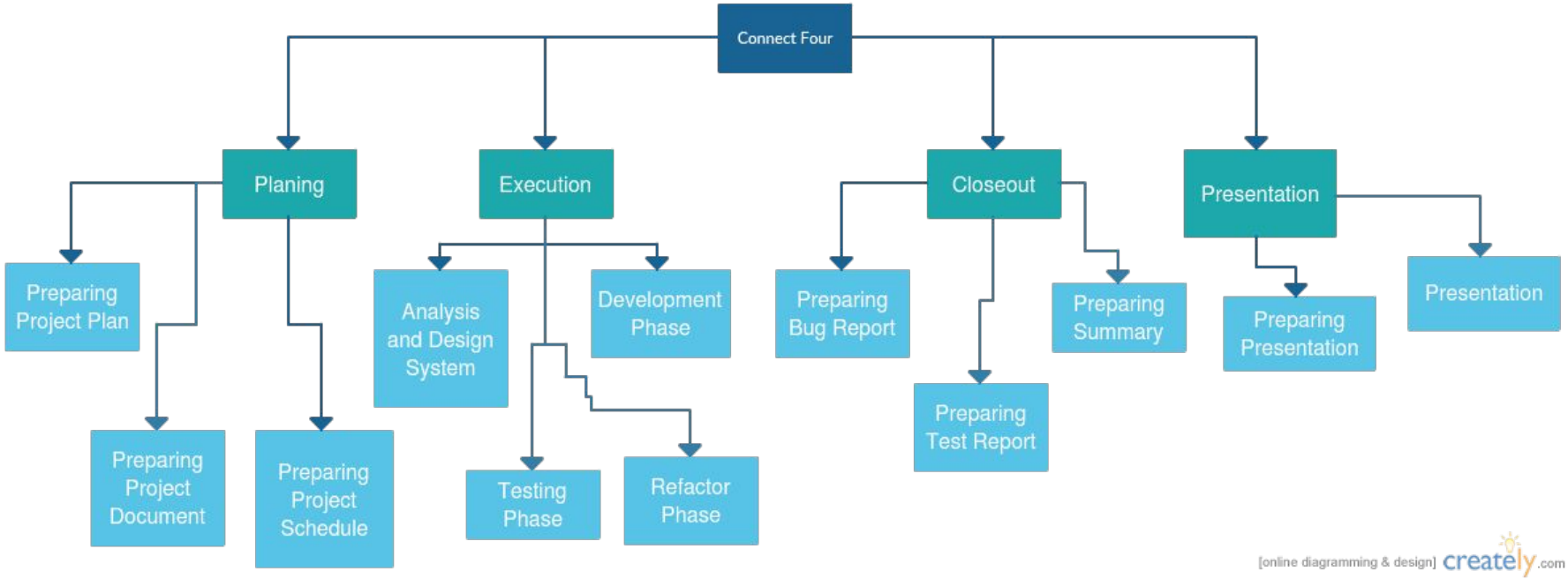
Member	Role
Tong Pang Tat	Project Manager
Ma Kei Tat	Analyst Programmer
Leung Kin Chung	Programmer
Pau Kwok Lun	Tester

Software Development Methodology

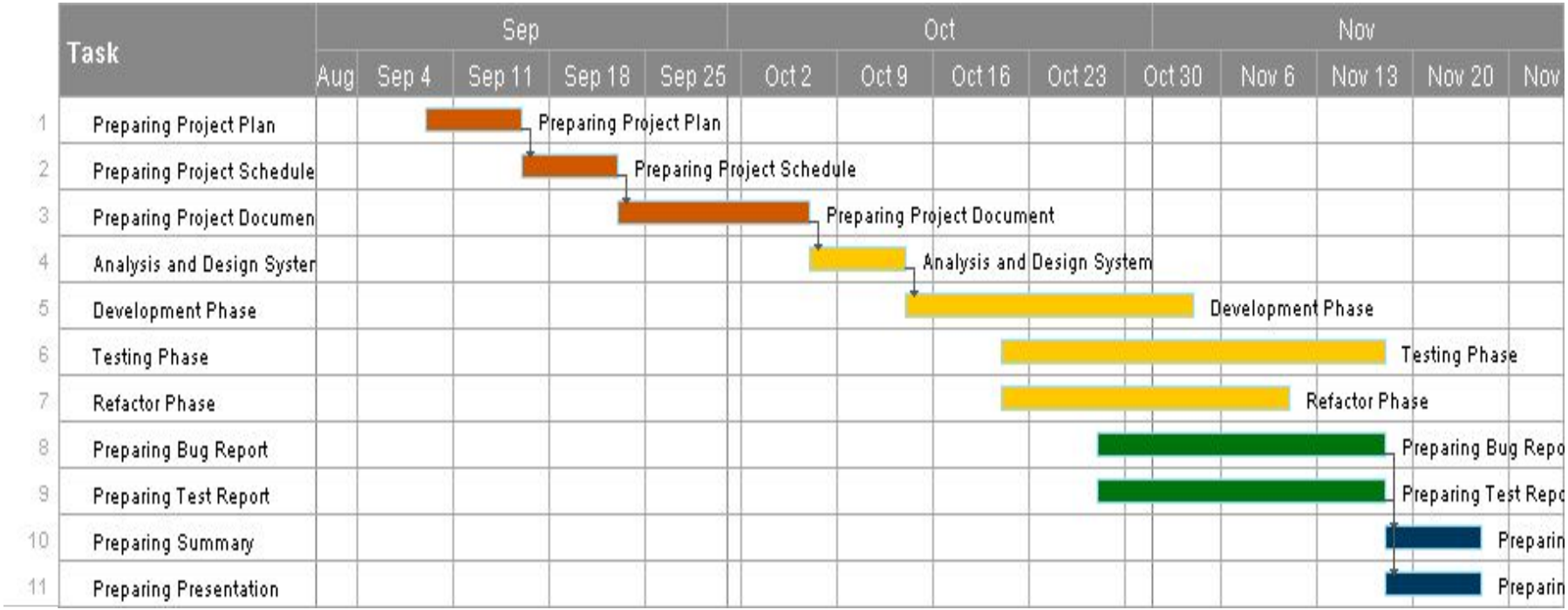
- Test-Driven Development (TDD)
  - Test case
  - Program code
  - Test case
  - Refactoring
  - Repeat



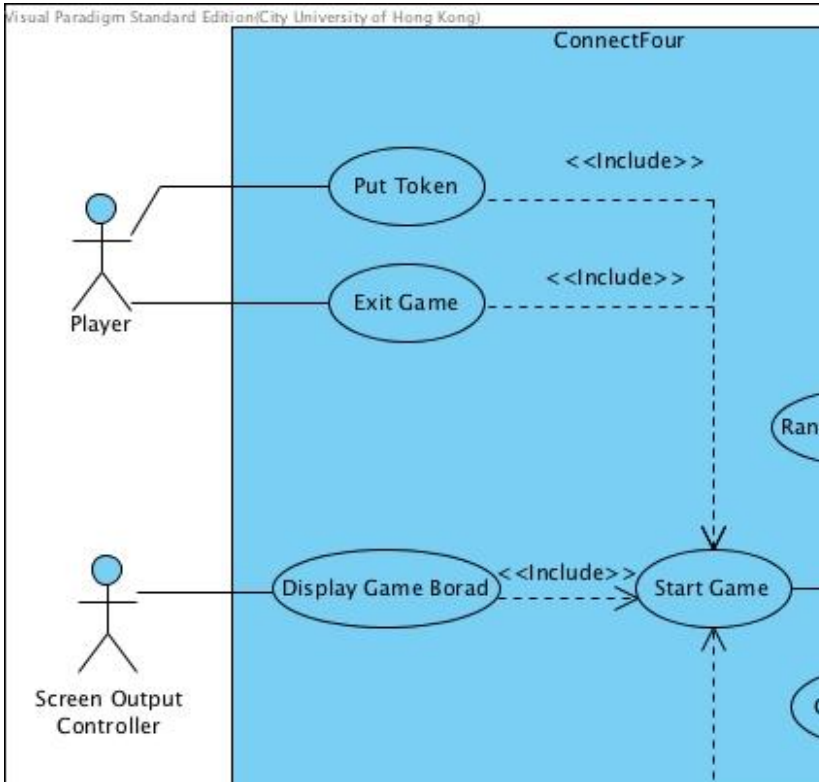
Work Breakdown Structure (WBS)

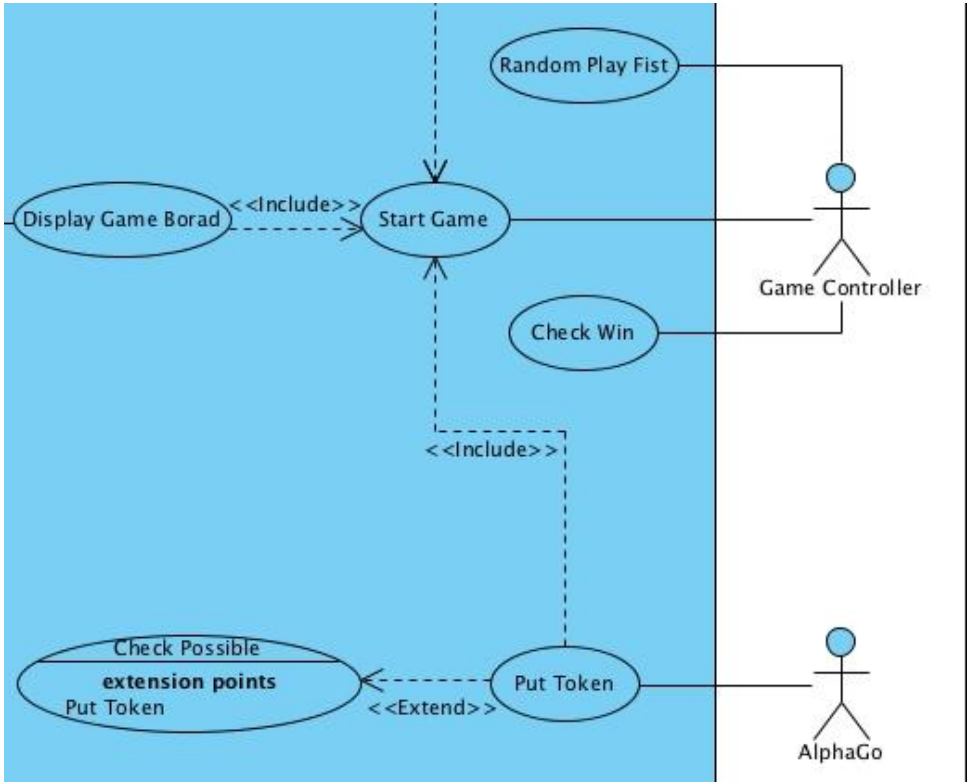


Project Schedule

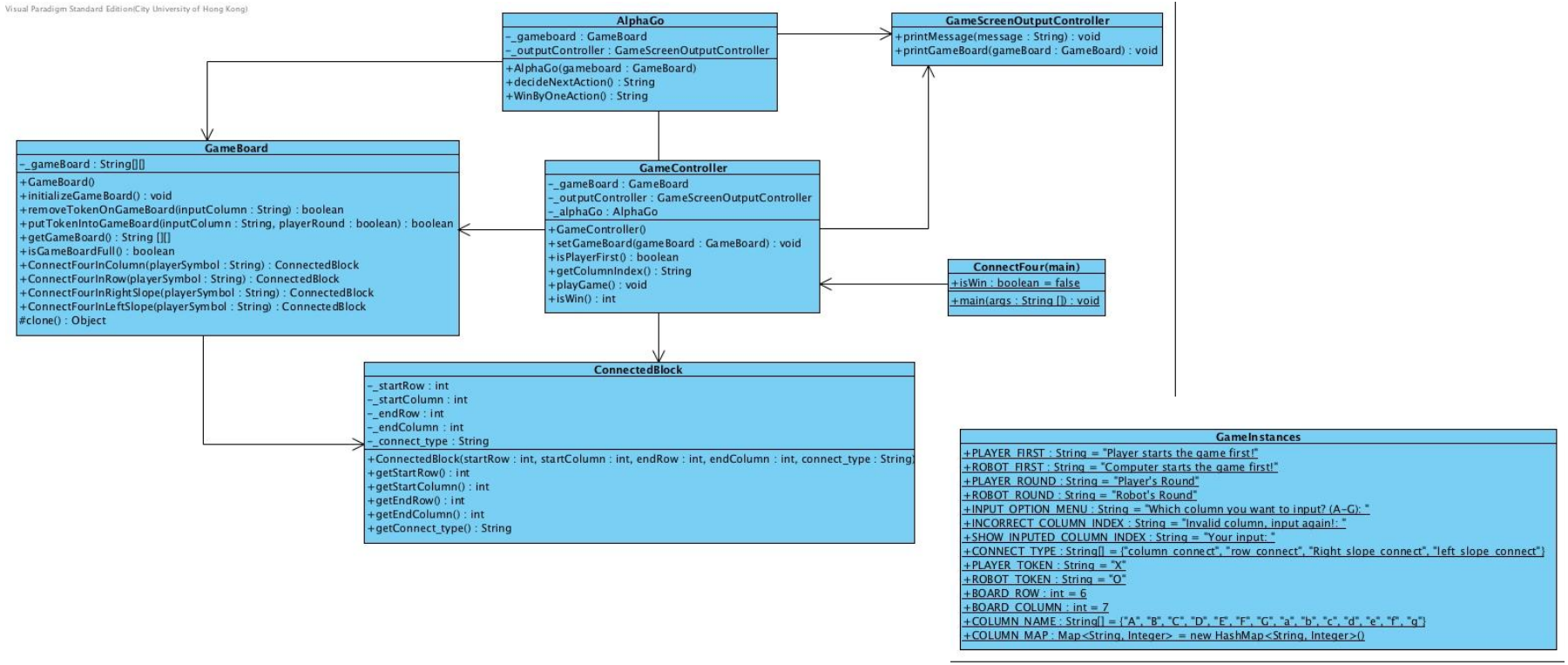


Use Case Diagram





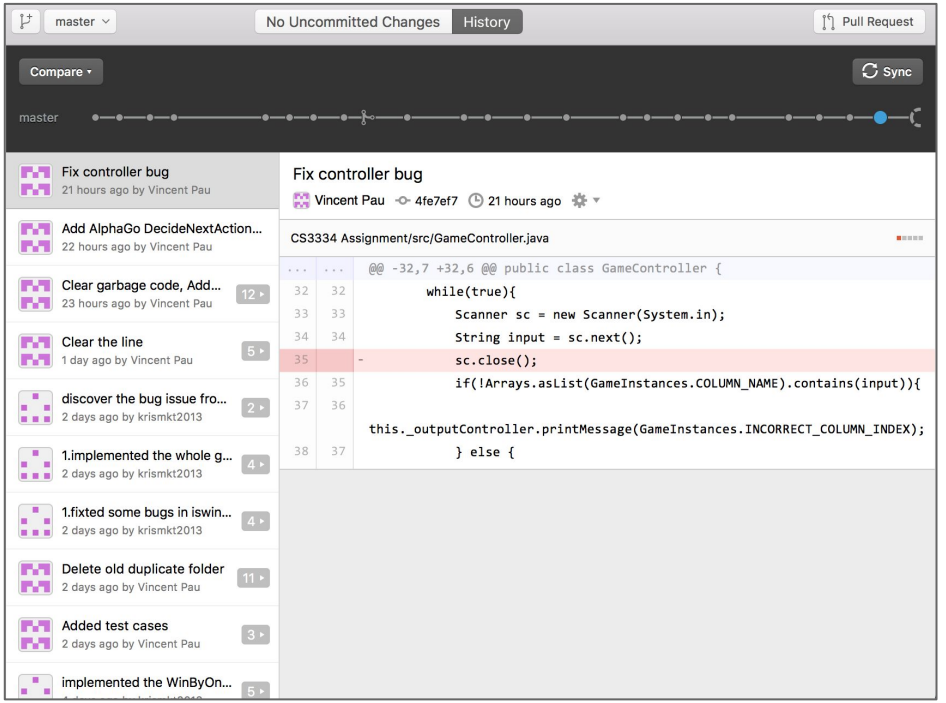
Class Diagram



Version Control

- GitHub
- Software configuration management

```
CS3343 — -bash — 80×31
Vincent-de-MacBook-Pro:GitHub Vincent$ cd CS3343/
Vincent-de-MacBook-Pro:CS3343 Vincent$ git status
On branch master
Your branch is up-to-date with 'origin/master'.
nothing to commit, working directory clean
Vincent-de-MacBook-Pro:CS3343 Vincent$ git pull
Already up-to-date.
Vincent-de-MacBook-Pro:CS3343 Vincent$ git push
warning: push.default is unset; its implicit value has changed in
Git 2.0 from 'matching' to 'simple'. To squelch this message
and maintain the traditional behavior, use:

  git config --global push.default matching

To squelch this message and adopt the new behavior now, use:

  git config --global push.default simple

When push.default is set to 'matching', git will push local branches
to the remote branches that already exist with the same name.

Since Git 2.0, Git defaults to the more conservative 'simple'
behavior, which only pushes the current branch to the corresponding
remote branch that 'git pull' uses to update the current branch.

See 'git help config' and search for 'push.default' for further information.
(the 'simple' mode was introduced in Git 1.7.11. Use the similar mode
'current' instead of 'simple' if you sometimes use older versions of Git)

Everything up-to-date
Vincent-de-MacBook-Pro:CS3343 Vincent$
```



Progam flow

How the game AI work

Stratagy：

1. Any action can make me win in this round ?

2. Any action can make the human player win in this round?

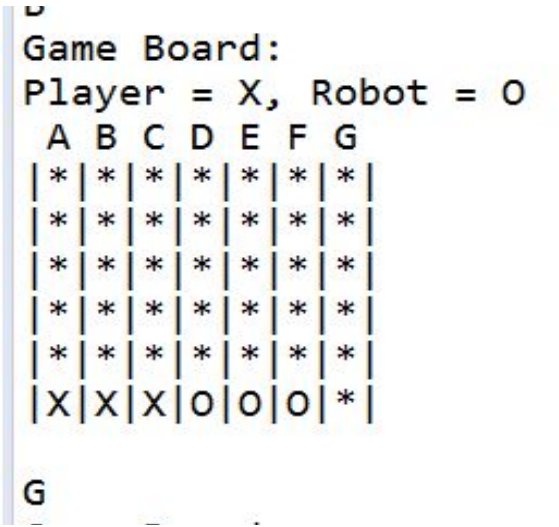
3. Any action can make me sure win in next round?

4. Any action can make the human player sure win in next round?

5. Any action can may make me win in coming 3 rounds

Program flow

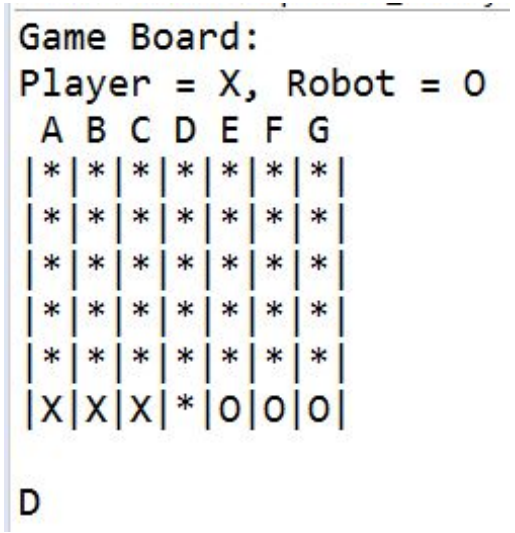
1. Any action can make me win in this round ?




```java
public String WinByOneAction(GameBoard gb) throws CloneNotSupportedException{
    String[] checkingColumns = { "A", "B", "C", "D", "E", "F", "G" };
    GameBoard gameBoard = (GameBoard) gb.clone();
    GameController controller = new GameController();
    controller.setGameBoard(gameBoard);

    for(int i=0; i<checkingColumns.length; i++){

        boolean inputResult = gameBoard.putTokenIntoGameBoard(checkingColumns[i], false);
        //this._outputController.printGameBoard(gameBoard);

        if(inputResult){
            int actionResult = controller.isWin();
            if(actionResult > 0){
                gameBoard.removeTokenOnGameBoard(checkingColumns[i]);
                return checkingColumns[i];
            }
            else{
                gameBoard.removeTokenOnGameBoard(checkingColumns[i]);
            }
        }

    }
    return null;
}
```

Program flow

2. Any action can make the human player win in this round?




```java
public String BlockPlayerWinByOneAction() throws CloneNotSupportedException{

    String[] checkingColumns = { "A", "B", "C", "D", "E", "F", "G" };
    GameBoard gameBoard = (GameBoard) this._gameboard.clone();
    GameController controller = new GameController();
    controller.setGameBoard(gameBoard);

    for(int i=0; i<checkingColumns.length; i++){

        boolean inputResult = gameBoard.putTokenIntoGameBoard(checkingColumns[i], true);
        //this._outputController.printGameBoard(gameBoard);

        if(inputResult){
            int actionResult = controller.isWin();
            if(actionResult > 0){
                gameBoard.removeTokenOnGameBoard(checkingColumns[i]);
                return checkingColumns[i];
            }
            else{
                gameBoard.removeTokenOnGameBoard(checkingColumns[i]);
            }
        }

    }
    return null;
```

Program flow

3. Any action can make me sure win in next round?

```
Game Board:
Player = X, Robot = O
 A B C D E F G
|*|*|*|*|*|*|*|
|*|*|*|*|*|*|*|
|*|*|*|*|*|*|*|
|*|*|O|*|O|*|*|
|*|*|O|O|X|X|*|
|*|*|X|O|X|O|*|

D
Game Board:
```

```
public String MakeOneActionToHaveTwoDifferentWinningWays(GameBoard gb) throws CloneNotSupportedException{
    String[] checkingColumns = { "A", "B", "C", "D", "E", "F", "G" };
    GameBoard gameBoard = (GameBoard) gb.clone();
    GameController controller = new GameController();
    controller.setGameBoard(gameBoard);

    for(int i=0; i<checkingColumns.length; i++){

        boolean inputResult = gameBoard.putTokenIntoGameBoard(checkingColumns[i], false);
        int ways = 0;

        if(inputResult){
            for(int j=0; j<checkingColumns.length; j++){
                inputResult = gameBoard.putTokenIntoGameBoard(checkingColumns[j], false);

                if(inputResult){

                    int actionResult = controller.isWin();
                    if(actionResult > 0){

                        ways++;

                    }
                    gameBoard.removeTokenOnGameBoard(checkingColumns[j]);

                }

            }
            if(ways > 1){
```

Program flow

4. Any action can make the human player sure win in next round?

```
ublic String BlockPlayerMakeOneActionToHaveTwoDifferentWinningWays() throws CloneNotSupportedException{
    String[] checkingColumns = { "A", "B", "C", "D", "E", "F", "G" };
    GameBoard gameBoard = (GameBoard) this._gameboard.clone();
    GameController controller = new GameController();
    controller.setGameBoard(gameBoard);

    for(int i=0; i<checkingColumns.length; i++){

        boolean inputResult = gameBoard.putTokenIntoGameBoard(checkingColumns[i], true);
        int ways = 0;

        if(inputResult){
            for(int j=0; j<checkingColumns.length; j++){
                inputResult = gameBoard.putTokenIntoGameBoard(checkingColumns[j], true);

                if(inputResult){

                    int actionResult = controller.isWin();
                    if(actionResult > 0){

                        ways++;

                    }
```

Line: 169

Program flow

5. Any action can may make me win in coming 3 rounds

```
Game Board:
Player = X, Robot = O
 A B C D E F G
|*|*|*|*|*|*|*|
|*|*|*|*|*|*|*|
|*|*|*|*|*|*|*|
|*|*|*|*|*|*|*|
|*|*|*|*|*|*|*|
|*|O|*|*|*|X|*|
```

```
public List posibleSeconTokenInput(GameBoard gb) throws CloneNotSupportedException{
    String[] checkingColumns = { "A", "B", "C", "D", "E", "F", "G" };
    List bestActions = new ArrayList();
    List normalActions = new ArrayList();
    List suggestedActions = new ArrayList();
    GameBoard gameBoard = (GameBoard) gb.clone();
    GameController controller = new GameController();
    controller.setGameBoard(gameBoard);

    for(int i=0; i<checkingColumns.length; i++){
        boolean inputResult = gameBoard.putTokenIntoGameBoard(checkingColumns[i], false);
        if(inputResult){
            String bestResult =this.MakeOneActionToHaveTwoDifferentWinningWays(gameBoard);
            if(!(bestResult == null)){
                System.out.println("i :"+i+" best");
                this._outputController.printGameBoard(gameBoard);

                bestActions.add(checkingColumns[i]);
            }
            else{
                List normalResult = this.MakeOneActionToHaveOnetWinningWay(gameBoard);

                System.out.println("i :"+i+" normal outter");
                this._outputController.printGameBoard(gameBoard);

                if(!normalResult.isEmpty()){
                    normalActions.add(checkingColumns[i]);
                }
            }
```

Program flow

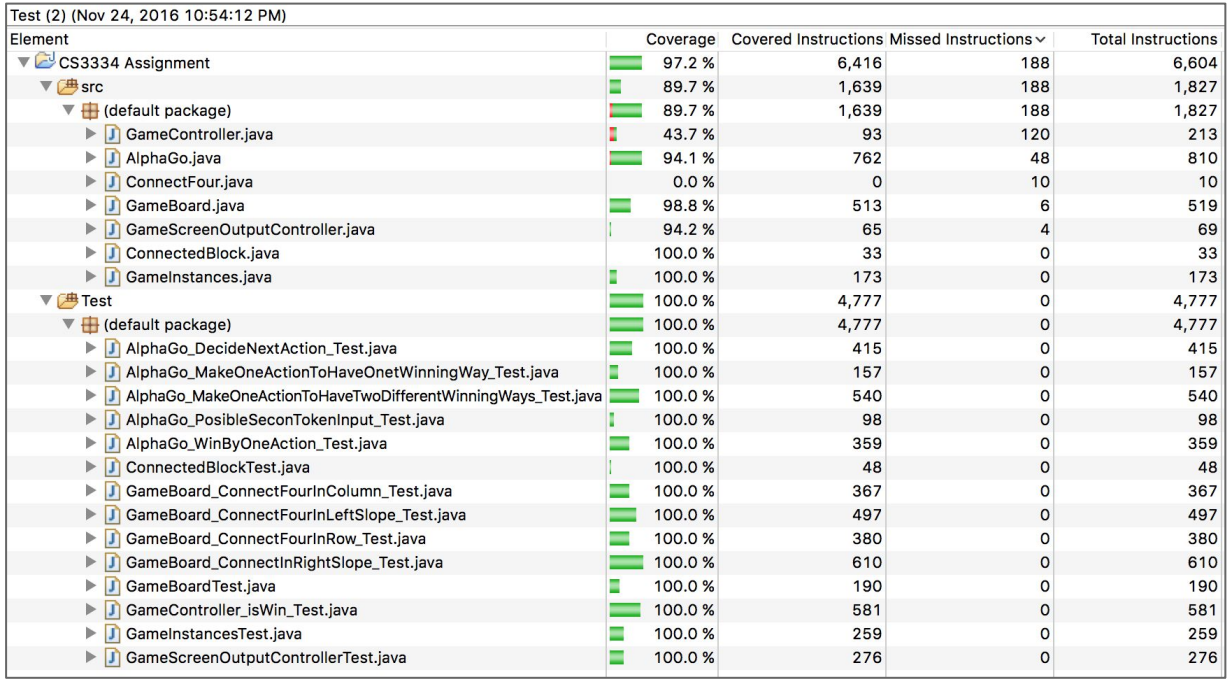
if the above conditions are also no(false)

then put a robot token randomly

```
Player = X, Robot = O
 A B C D E F G
|*|*|*|*|*|*|*|
|*|*|*|*|*|*|*|
|*|*|*|*|*|*|*|
|*|*|*|*|*|*|*|
|*|*|*|*|*|*|*|
|*|*|*|*|*|*|*|

Game Board:
Player = X, Robot = O
 A B C D E F G
|*|*|*|*|*|*|*|
|*|*|*|*|*|*|*|
|*|*|*|*|*|*|*|
|*|*|*|*|*|*|*|
|*|*|*|*|*|*|*|
|*|*|*|*|*|*|O|

Which column you want to input? (A-G):
```

Unit Testing

- JUnit 4
- Statement Coverage
- Branch Coverage
- Loop Coverage
- Condition Coverage
- Decision Coverage
- Path Coverage



Test (2) (Nov 24, 2016 10:54:12 PM)

Element	Coverage	Covered Instructions	Missed Instructions	Total Instructions
CS3334 Assignment	97.2 %	6,416	188	6,604
src	89.7 %	1,639	188	1,827
(default package)	89.7 %	1,639	188	1,827
GameController.java	43.7 %	93	120	213
AlphaGo.java	94.1 %	762	48	810
ConnectFour.java	0.0 %	0	10	10
GameBoard.java	98.8 %	513	6	519
GameScreenOutputController.java	94.2 %	65	4	69
ConnectedBlock.java	100.0 %	33	0	33
GameInstances.java	100.0 %	173	0	173
Test	100.0 %	4,777	0	4,777
(default package)	100.0 %	4,777	0	4,777
AlphaGo_DecideNextAction_Test.java	100.0 %	415	0	415
AlphaGo_MakeOneActionToHaveOnetWinningWay_Test.java	100.0 %	157	0	157
AlphaGo_MakeOneActionToHaveTwoDifferentWinningWays_Test.java	100.0 %	540	0	540
AlphaGo_PosibleSeconTokenInput_Test.java	100.0 %	98	0	98
AlphaGo_WinByOneAction_Test.java	100.0 %	359	0	359
ConnectedBlockTest.java	100.0 %	48	0	48
GameBoard_ConnectFourInColumn_Test.java	100.0 %	367	0	367
GameBoard_ConnectFourInLeftSlope_Test.java	100.0 %	497	0	497
GameBoard_ConnectFourInRow_Test.java	100.0 %	380	0	380
GameBoard_ConnectInRightSlope_Test.java	100.0 %	610	0	610
GameBoardTest.java	100.0 %	190	0	190
GameController_isWin_Test.java	100.0 %	581	0	581
GameInstancesTest.java	100.0 %	259	0	259
GameScreenOutputControllerTest.java	100.0 %	276	0	276

Statement Coverage

- Statement has been executed at least once.

```
public void printGameBoard(GameBoard gameBoard) {
    // String[][] gameBoard, String playToken, String robotToken
    String[][] gameBoardInstance = gameBoard.getGameBoard();

    System.out.println("Game Board:");
    System.out.println("Player = " + GameInstances.PLAYER_TOKEN + ", Robot = " + GameInstances.ROBOT_TOKEN);
    System.out.println(" A B C D E F G");
    for (String[] row : gameBoardInstance) {
        System.out.print("|");
        for (String field : row) {
            System.out.print(field + "|");
        }
        System.out.println();
    }
    System.out.println();
}
```

Branch Coverage

- Each branch has been executed at least once.

```
public String WinByOneAction(GameBoard gb) throws CloneNotSupportedException {
    String[] checkingColumns = { "A", "B", "C", "D", "E", "F", "G" };
    GameBoard gameBoard = (GameBoard) gb.clone();
    GameController controller = new GameController();
    controller.setGameBoard(gameBoard);

    for(int i=0; i<checkingColumns.length; i++){
        boolean inputResult = gameBoard.putTokenIntoGameBoard(checkingColumns[i], false);
        //this._outputController.printGameBoard(gameBoard);

        if(inputResult){
            int actionResult = controller.isWin();
            if(actionResult > 0){
                gameBoard.removeTokenOnGameBoard(checkingColumns[i]);
                return checkingColumns[i];
            } else {
                gameBoard.removeTokenOnGameBoard(checkingColumns[i]);
            }
        }
    }
    return null;
}
```

Loop Coverage

- Each loop should be tested with 0, 1, and >1 times

```
public int isWin(){
    int result = 0;
    String[] tokens = {GameInstances.PLAYER_TOKEN, GameInstances.ROBOT_TOKEN};
    ConnectedBlock cb;
    for(int i = 0; i<tokens.length; i++){
        result = 0;
        cb =  this._gameBoard.ConnectFourInColumn(tokens[i]);
        if(!(cb == null)){
            result++;
        }

        cb =  this._gameBoard.ConnectFourInRow(tokens[i]);
        if(!(cb == null)){
            result++;
        }

        cb =  this._gameBoard.ConnectFourInRightSlope(tokens[i]);
        if(!(cb == null)){
            result++;
        }

        cb =  this._gameBoard.ConnectFourInLeftSlope(tokens[i]);
        if(!(cb == null)){
            result++;
        }

        if(result>0){
            return result;
        }
    }
    return 0;
}
```

Condition Coverage

- Each condition has been evaluated to be true and false.

```
public int isWin(){
    int result = 0;
    String[] tokens = {GameInstances.PLAYER_TOKEN, GameInstances.ROBOT_TOKEN};
    ConnectedBlock cb;
    for(int i = 0; i<tokens.length; i++){
        result = 0;
        cb =  this._gameBoard.ConnectFourInColumn(tokens[i]);
        if(!(cb == null)){
            result++;
        }

        cb =  this._gameBoard.ConnectFourInRow(tokens[i]);
        if(!(cb == null)){
            result++;
        }

        cb =  this._gameBoard.ConnectFourInRightSlope(tokens[i]);
        if(!(cb == null)){
            result++;
        }

        cb =  this._gameBoard.ConnectFourInLeftSlope(tokens[i]);
        if(!(cb == null)){
            result++;
        }

        if(result>0){
            return result;
        }
    }
    return 0;
}
```

Decision Coverage

- Each decision has been evaluated to be true and false.

```
while((innerRow < GameInstances.BOARD_ROW) && (innerColumn < GameInstances.BOARD_COLUMN)){
    //System.out.println("inner_Row: " + innerRow +" inner_column: "+innerColumn);
    if((this._gameBoard[innerRow][innerColumn]).equals(playerSymbol)){
        if(startColumn < 0 ){
            startRow = innerRow;
            startColumn = innerColumn;
        }
        endRow = innerRow;
        endColumn = innerColumn;
    } else {
        startRow = -1;
        startColumn= -1;
        endRow = -1;
        endColumn = -1;
    }
    //System.out.println("Star_Row: "+startRow +" Start_Column: "+ startColumn+" End_Row: "

    int difference = endRow - startRow;
    difference = (difference > 0) ? difference : -difference;
    if(difference >= 3){
        ConnectedBlock cb = new ConnectedBlock(startRow, startColumn, endRow, endColumn,
                GameInstances.CONNECT_TYPE[2]);
        return cb;
    }
    innerRow++;
    innerColumn++;
}
```

Program segment in AlphaGo.ConnectFourInRightSlope()

Path Coverage

- Each path has been executed at least once.

```
while((innerRow < GameInstances.BOARD_ROW) && (innerColumn < GameInstances.BOARD_COLUMN)){
    //System.out.println("inner_Row: " + innerRow +" inner_column: "+innerColumn);
    if((this._gameBoard[innerRow][innerColumn]).equals(playerSymbol)){
        if(startColumn < 0 ){
            startRow = innerRow;
            startColumn = innerColumn;
        }
        endRow = innerRow;
        endColumn = innerColumn;
    } else {
        startRow = -1;
        startColumn= -1;
        endRow = -1;
        endColumn = -1;
    }
    //System.out.println("Star_Row: "+startRow +" Start_Column: "+ startColumn+" End_Row: "

    int difference = endRow - startRow;
    difference = (difference > 0) ? difference : -difference;
    if(difference >= 3){
        ConnectedBlock cb = new ConnectedBlock(startRow, startColumn, endRow, endColumn,
                GameInstances.CONNECT_TYPE[2]);
        return cb;
    }
    innerRow++;
    innerColumn++;
}
```

Program segment in AlphaGo.ConnectFourInRightSlope()

Bug Report

- Scanner is close, throw NoSuchElementException
- Fixed

```
Game Board:
Player = X, Robot = O
 A B C D E F G
|*|*|*|*|*|*|*|
|*|*|*|*|*|*|*|
|*|*|*|*|*|*|*|
|*|*|*|*|*|*|*|
|*|*|*|*|*|*|*|
|*|*|*|X|O|*|*|

Which column you want to input? (A-G):
Exception in thread "main" java.util.NoSuchElementException
        at java.util.Scanner.throwFor(Scanner.java:862)
        at java.util.Scanner.next(Scanner.java:1371)
        at GameController.getColumnIndex(GameController.java:34)
        at GameController.playGame(GameController.java:58)
        at ConnectFour.main(ConnectFour.java:5)
```

```
public String getColumnIndex(){
    while(true){
        Scanner sc = new Scanner(System.in);
        String input = sc.next();
        if(!Arrays.asList(GameInstances.COLUMN_NAME).contains(input)){
            this._outputController.printMessage(GameInstances.INCORRECT_COLUMN_INDEX);
        } else {
            sc.close();
            return input;
        }
    }
}
```

Bug Report

Tokens cannot be removed correctly, after trials

```
Game Board:
Player = X, Robot = O
 A B C D E F G
|*|*|*|*|*|*|*|
|*|*|*|*|*|*|*|
|*|*|*|*|*|*|*|
|*|*|*|*|*|*|*|
|*|*|*|*|*|*|O|
|X|*|*|*|*|*|O|

Which column you want to input? (A-G):
```

```
Game Board:
Player = X, Robot = O
 A B C D E F G
|*|*|*|*|*|*|O|
|*|*|*|*|*|*|O|
|*|*|*|*|*|*|O|
|*|*|*|*|*|*|X|
|*|*|*|*|*|*|O|
|X|*|*|*|*|O|O|

Which column you want to input? (A-G):
```

Refactoring

Before Refactoring

- Part of code extracted from decideNextMove() function
- Used for checking different conditions
- Repeated checking for several times
- Duplication of codes is found
- Same code structure is repeated

```
try {

    action = this.winByOneAction();
    if(!(action == null)){
        return action;
    }

    action = this.blockPlayerWinByOneAction();
    if(!(action == null)){
        return action;
    }

    action = this.makeTwoWinWays();
    if(!(action == null)){
        return action;
    }

    action = this.blockPlayerMakeOneActionToHaveTwoDifferentWinningWays();
    if(!(action == null)){
        return action;
    }
```

Refactoring

After Refractoring

- Applied “Substitute Algorithm” method
- Replaced the repeated stucture
- A clearer algorithm is used
- Different conditions are first stored into an array
- Search through the array
- Obtain the corresponding action

```
try {

    String[] conditions = new String[]{
            this.winByOneAction(),
            this.blockPlayerWinByOneAction(),
            this.makeTwoWinWays(),
            this.blockPlayerMakeOneActionToHaveTwoDifferentWinningWays(),
            possibleMoves[rand.nextInt(possibleMoves.length)]};

    for(int i=0; i<conditions.length-1; i++){
        if( conditions[i] != null ){
            return conditions[i];
        }
    }
```

Refactoring

Before refactoring

- Part of code extracted from winnerIsFound() function
- Used for checking connection of four tokens
- Repeated checking for direction
- Duplication of codes is found
- Same code structure is repeated

```
ConnectedBlock cb;
for(int i = 0; i<tokens.length; i++){
    result = 0;
    cb =  this._gameBoard.connectFourInColumn(tokens[i]);
    if(!(cb == null)){
        result++;
    }

    cb =  this._gameBoard.ConnectFourInRow(tokens[i]);
    if(!(cb == null)){
        result++;
    }

    cb =  this._gameBoard.ConnectFourInRightSlope(tokens[i]);
    if(!(cb == null)){
        result++;
    }

    cb =  this._gameBoard.ConnectFourInLeftSlope(tokens[i]);
    if(!(cb == null)){
        result++;
    }

    if(result>0){
        return result;
    }
}
```

Refactoring

After refactoring

- Applied “Substitute Algorithm” method
- Replaced the repeated stucture
- Add the returned value to the arraylist
- Search through the arraylist
- If connection is found, result will be incremented by one and returned

```
ArrayList<ConnectedBlock> connectedBlockList = new ArrayList<>();

for(int i = 0; i<tokens.length; i++){
    result = 0;
    connectedBlockList.add(this._gameBoard.connectFourInColumn(tokens[i]));
    connectedBlockList.add(this._gameBoard.ConnectFourInRow(tokens[i]));
    connectedBlockList.add(this._gameBoard.ConnectFourInRightSlope(tokens[i]));
    connectedBlockList.add(this._gameBoard.ConnectFourInLeftSlope(tokens[i]));

    for(ConnectedBlock cb : connectedBlockList){
        if(cb != null)
            result++;
    }
    if(result>0){
        return result;
    }
}
```

Refactoring

Before refactoring

- The variable - “input” refered to getColumnIndex() and alphaGo.decideNextAction() at different condition
- The use of the variable is unclear
- This may confuse other teammates and make the code less comprehensive
- Potential risk of causing bug may arise

```
String input;
if(playerRound){// playerRound = true -> means th
    outputController.printMessage(GameInstances.I
    input = this.getColumnIndex();
    outputController.printMessage(GameInstances.S
    gameBoard.putTokenIntoGameBoard(input, true);
    playerRound = false;
}
else{// playerRound = false -> means that there i

    input = this._alphaGo.decideNextAction();
    System.out.println(input);

    System.out.println("before put token to board
    this._outputController.printGameBoard(gameBoa

    gameBoard.putTokenIntoGameBoard(input, false)
    playerRound = true;
}
```

Refactoring

After refactoring

- The original variable is divided into two variables with more specific and meaningful name
- The function of those variables become more clear

```
String playerMove;
String AI_Move;
if(playerRound){// playerRound = true -> means that there is the human player round
    outputController.printMessage(GameInstances.INPUT_OPTION_MENU);
    playerMove = this.getColumnIndex();
    outputController.printMessage(GameInstances.SHOW_INPUTED_COLUMN_INDEX + playerMove +"\n");
    gameBoard.putTokenIntoGameBoard(playerMove, true);
    playerRound = false;
}
else{// playerRound = false -> means that there is the robot round

    AI_Move = this._alphaGo.decideNextAction();
    System.out.println(AI_Move);
```

- Reduce time for other teammates to remember and trace the value refered by the variable

DEMO

Q & A