BACS2063 Data Structures and Algorithms

ASSIGNMENT 202005

Student Name	:	Cheong Yin Lam
Student ID	:	20WMR09435
Programme	:	RSF2
Tutorial Group	:	G5
Team Name	:	Connect Four (Cheong Yin Lam, Joan Hau, Lee Ling, Lim Ming Yeu, Tan Yong Kang)

Declaration

- I confirm that I have read and complied with all the terms and conditions of Tunku Abdul Rahman University College's plagiarism policy.
- I declare that this assignment is free from all forms of plagiarism and for all intents and purposes is my own properly derived work.

_____

Student's signature

21/8/2020

_____

Date

BACS2063 Data Structures & Algorithms - Assignment Rubrics				Name: Cheong Yin Lam		
Programme: RSF2				Group/Team: G5 / Connect Four		
No.	**Aspect**	**Developing 0 - 4 marks**	**Approaching 5-7 marks**	**Ideal 8-10 marks**	**Score**	**Remarks**
1	**Idea (based on overview and user guide)**	The idea has a low degree of creativity and novelty among all students submissions.	The idea has a moderate degree of creativity and novelty among all students submissions.	The idea has a high degree of creativity and novelty among all students submissions.		
2	**Entity class diagram (Team Mark)**	Reflects poor ability to analyze the entities required in the application.	Reflects moderate ability to analyze the entities required in the application.	Reflects excellent ability to analyze the entities required in the application.		
3	**Implementation of entity class(es)**	Entity class(es) is implemented incorrectly.	Entity class(es) is implemented correctly but is inconsistent with the entity class diagram.	Entity class(es) is implemented correctly and is consistent with the entity class diagram.		
4	**Collection abstract data type (ADT) specification**	ADT specification is missing 2 or more of the elements: conciseness, correctness, completeness.	ADT specification is missing 1 of the elements: conciseness, correctness, completeness.	ADT specification is concise, correct and complete.		
5	**Implementation of collection ADT**	Reflects poor ablity to synthesize to create new collection ADT.	Reflects moderate ablity to synthesize to create new collection ADT.	Reflects excellent ablity to synthesize to create new collection ADT.		
6	**Selection of collection ADTs for the client program**	ADT is not appropriate and weak justification is provided.	ADT is appropriate and some justification is provided.	ADT is appropriate and sound justification is provided.		
7	**Implementation of client program**	Reflects poor ablity to synthesize to create new solutions using appropriate ADTs **with use of standard Java naming convention**.	Reflects moderate ablity to synthesize to create new solutions using appropriate ADTs **with use of standard Java naming convention**.	Reflects excellent ablity to synthesize to create new solutions using appropriate ADTs **with use of standard Java naming convention**.		
8	**Overall solution (Team Mark)**	Overall solution demonstrates low ability to apply knowledge and skills of data structures and algorithms.	Overall solution demonstrates moderate ability to apply knowledge and skills of data structures and algorithms.	Overall solution demonstrates high ability to apply knowledge and skills of data structures and algorithms.		
9	**Team skills - contribution to integration of modules.**	Student demonstrates poor collaborative skills in integrating the application.	Student demonstrates moderate in collaborative skills in integrating the application.	Student demonstrates excellence in collaborative skills in integrating the application.		
10	**Team skills - peer assessment**	Student demonstrates poor levels in interactive communications, relationships and collaborative skills in managing relationships in the team.	Student demonstrates moderate ability in interactive communications, relationships and collaborative skills in managing relationships in the team.	Student demonstrates excellence in interactive communications, relationships and collaborative skills in managing relationships in the team.		
				Total / 100	**0**	

CLO2 **Cognitive Skills** relates to thinking or intellectual capabilities and the ability to apply knowledge and skills. The capacity to develop levels of intellectual skills progressively begins from understanding, critical/creative

CLO3 **Team Skills** include: **Relationship-building:** Ability to build strong relationship, interact and work effectively with people to attain the same objectives.
Respect: Ability to identify and respect the attitudes, behaviours and beliefs of others.

BACS2063 Data Structures and Algorithms - Team Skills Peer Assessment				
Relationship-building:	Ability to build strong relationship, interact and work effectively with people to attain the same objectives.			
Respect:	Ability to identify and respect the attitudes, behaviours and beliefs of others.			
Instructions: Indicate marks and justification for all team members' including yourself.				
Name	**Relationship (5 marks)**	**Respect (5 marks)**	**Total marks (10 marks)**	**Justification for the marks awarded (Required)**
Cheong Yin Lam	5	5	10	Involved in discussion about the game and received opinions from others. Communicate with each other to know who is pushing the latest version of their module and the main program to prevent file crashing.
Joan Hau	5	5	10	Involved in discussion about the game and received opinions from others. Communicate with each other to know who is pushing the latest version of their module and the main program to prevent file crashing.
Lee Ling	5	5	10	Involved in discussion about the game and received opinions from others. Communicate with each other to know who is pushing the latest version of their module and the main program to prevent file crashing.
Lim Ming Yeu	5	5	10	Involved in discussion about the game and received opinions from others. Communicate with each other to know who is pushing the latest version of their module and the main program to prevent file crashing.
Tan Yong Kang	5	5	10	Involved in discussion about the game and received opinions from others. Communicate with each other to know who is pushing the latest version of their module and the main program to prevent file crashing.

Table of Contents

1. Introduction to the Game

Connect Four is a game that allows two players to be involved in a round of the game to insert the tokens and compete who can get the four connected tokens first. In our Connect Four game design, the program will allow the players to select if they want to play the classic mode or the tournament mode. The program will also allow the players to choose the connect number to determine the winner which is connect 3 (simple mode), connect 4 (medium mode) or connect 5 (hard mode). Besides, the program will allow few players to play with each other by taking turns to play. Each round will only consist of two players. In the **classic mode**, the winner in the specific round will be able to continue to play with the other player in the next round. In the **tournament mode**, the winner is able to play with others until the final winner exists.

Moreover, the program will set a prompt for the number of players after the players choose the game mode. There are only allowed 4 or 8 players to participate in the **tournament mode** without less than or more than. For the **classic mode**, there is no restriction for the number of players. Then the program will let each player enter their name and a favorite alphabet without repeating from other players to represent their player's name and character of the token in the game. The character of the token is only able to be same when there is an upper case and a lower case. After that, when the game is started, the program will display the two players' names who are involved in the specific round with their token character. The two players will take turns to enter the column number to insert the token into the board. When the same token is connected as the selected connect number, the player who belongs to the token will be considered as the winner. Moreover, there are four ways to detect the connected token which is horizontal connected, vertical connected, positive diagonal connected and negative diagonal connected. In addition, when the board is fully inserted with the tokens, the system will display a sentence to tell the players there is no winner and the system will let the two players play again.

In each round, if the winner exists, the system will display the name of the winners and their token used. There will also display a tree diagram for players to know who is winning and the winner will battle with which player (another winner) during the **tournament mode**. At the end of the game in both game mode, the program will display the ranking report with the player's score. The score is dependent on the time taken in each round and the number of tokens

used. When all the players have taken turn to play, the system will display a ranking report at the end as a closure.

Last but not least, the players are able to view their ranking in the ranking board which can be entered from the main menu. The ranking board will display the highest scores from the previous players. It will store the scores in the specific ranking board based on the number of players in the game.

2. User Guide

At the beginning, the main menu will be displayed as the users can start playing the game, entering the ranking board for viewing the previous players' ranking, or exit the game. Users are only allowed to enter the integer of the selection that is given in the menu. When users decide to start the game, the program will prompt the users to select the game mode which is classic mode or tournament mode. Guidance about each mode is provided in below and the raking board also:

When users are selected the **classic mode**:

The system will prompt for the players' numbers and there is no restriction on the number of players except must be more than one player. Then, the system will prompt the players for getting their players' name and their character of the token which must be an alphabet. Players are able to enter their nickname as the player's name and the favorite alphabet as their character of the token. However, the same player's name or character of the token is not allowed except the character of the token is different cases which means there are only allowed the players to have same alphabet when there is an uppercase alphabet and a lowercase alphabet. After the players' name and the character of the token have been entered, the system will let players to select the connect number at once which is simple mode (connect 3), medium mode (connect 4) or hard mode (connect 5) to determine how many tokens of the player connected together are considered as the winner.

After the connect number of the game is selected by the players, the first-round game will be started and the first two players will be assigned and displayed by the program. During

the game, the two players who are involved in the round are required to take turns to enter the valid column number for inserting their token into the board. Alphabet and the symbol are not allowed to represent the column number. Furthermore, the program will automatically be detected if the winner exists. When the winner exists, the name of the winner and the token used by the players will be displayed and the next round will be started when still have player haven't taken turns. The winner will be able to play with the next player in the following round. At the end of the game, after all players have taken turns, the ranking report of the players will be displayed.

When users are selected the **tournament mode**:

The system will prompt for the players' numbers and there must be 4 or 8 players to participate in the game together. Number of players more than or less than are not allowed. Then, the system will prompt the players for getting their players' name and their character of the token which must be an alphabet. Players are able to enter their nickname as the player's name and the favorite alphabet as their character of the token. However, the same player's name or character of the token is not allowed except the character of the token is different cases which means there are only allowed the players to have same alphabet when there is an uppercase alphabet and a lowercase alphabet. After the players' name and the character of the token have been entered, the system will let players to select the connect number at once which is simple mode (connect 3), medium mode (connect 4) or hard mode (connect 5) to determine how many tokens of the player connected together are considered as the winner.

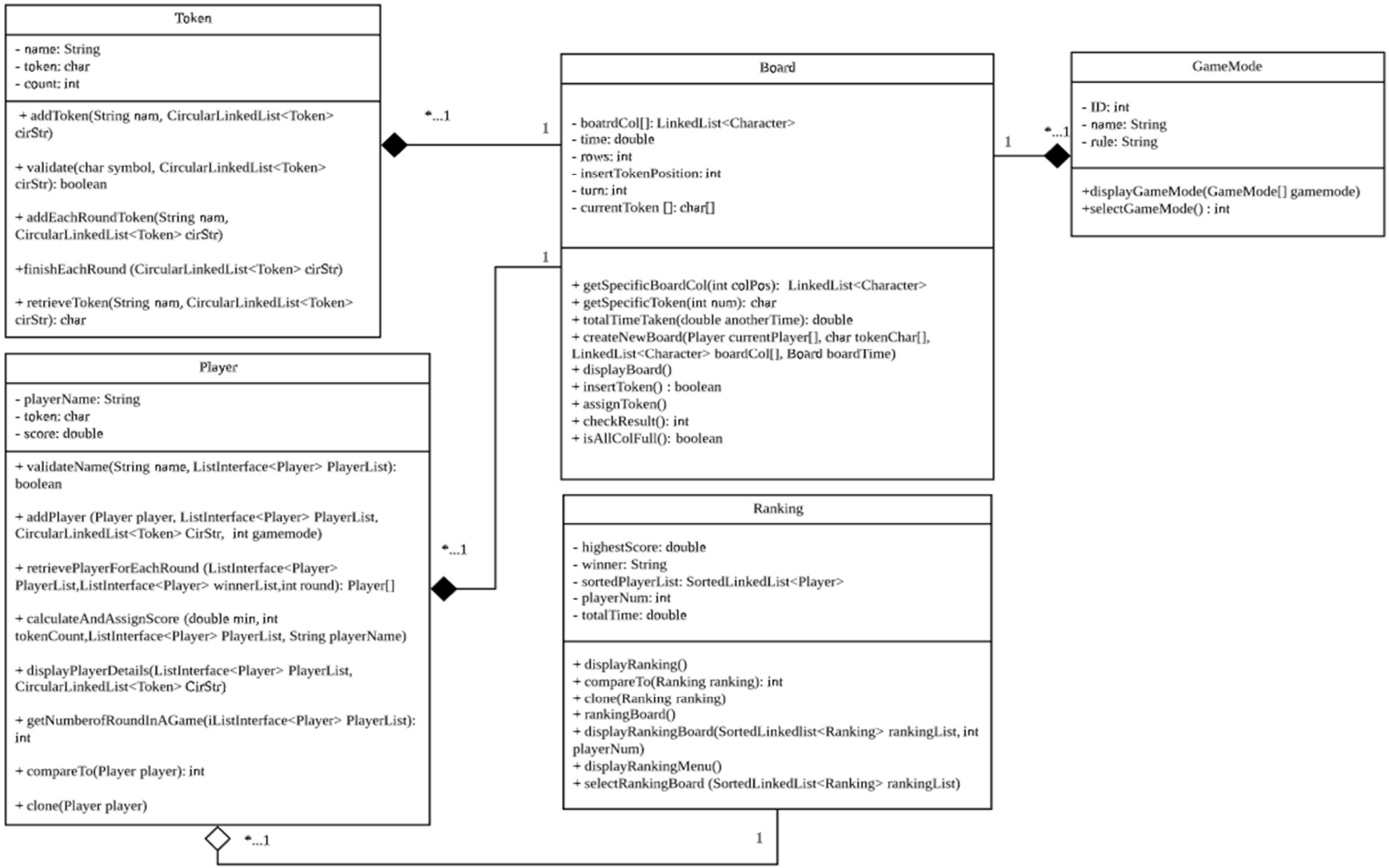
After the connect number of the game is selected by the players, the first-round game will be started and the tree diagram will be displayed for the players to know who is the opponent of them. Then, the players in the rightest side of the tree diagram will be the first-round players and these two players will be assigned and displayed by the program to let all the players know who are the current players. Furthermore, during the game, the two players who are involved in the round are required to take turns to enter the valid column number for inserting their token into the board. Alphabet and the symbol are not allowed to represent the column number. Moreover, the program will automatically be detected if the winner exists. When the winner exists, the name of the winner and the token used by the players will be displayed and also the tree diagram which is to let the players know who will be able to battle

with another player. Then, the next round will proceed until the top of the tree diagram is filled with the final winner name. At the end of the game, after the final winner is existing, the ranking report of the players will be displayed.

Ranking board in Main Menu**:**

Besides, players are able to view the ranking in the ranking board which is entered from the main menu. Then, they are required to enter the ranking board based on the number of players in that game they are involved. For example, 2 players are involved in the game so can enter the 2 Player Ranking Board to view the ranking records. If there are more than 5 players, the players can enter the 'More than 5 Players Ranking Board' for viewing the ranking records.

3. Entity Class Diagram



4. Implementation of Entity Class(es)

The entity class that I in charge of this game is the game board, the class name I declare is called Board. Below is the source code in my board class:
(The print screen of the source code below has been omitted the getters and setters)

```
public class Board {
    Scanner scan = new Scanner(System.in);

    double time;
    int rows;

    LinkedList<Character> boardCol[];

    int insertTokenPosition = 0;
    int turn = 1;

    char currentToken[];


    public LinkedList<Character> getSpecificBoardCol(int colPos){
        return boardCol[colPos];
    }

    public char getSpecificToken(int num){
        return currentToken[num];
    }

    //----- cal total time----//
    public double totalTimeTaken(double anotherTime){  //return min
      double timeTaken;

      if(time > anotherTime){
           timeTaken = time - anotherTime;
      }else{
          timeTaken = anotherTime - time;
      }
      return (timeTaken / 60000 );  // (/1000/60)

     }


}
```

5. ADT Specification

Linked List ADT

Linked List ADT is a linear collection of the data structures which is arranged with an ordered sequence.

boolean add (T newEntry)
Description: Add a *newEntry* to the top of the list.
Parameter: a *newEntry*.
Precondition: None.
Postcondition: The *newEntry* is added to the list.
Return: true if the *newEntry* is added successfully else false.

boolean add (int newPosition, T newEntry)
Description: Add a *newEntry* to the specific position (*newPosition*) of the list. Position 0 is indicated as the first entry of the list.
Parameter: a *newPosition* and a *newEntry*.
Precondition: the *newPosition* must be between 0 and the total elements of the list - 1.
Postcondition: The *newEntry* is added to the specific position of the list.
Return: true if the *newEntry* is added successfully else false.

T remove (int givenPosition)
Description: To remove the element from the list based on the *givenPosition*.
Parameter: a *givenPosition*
Precondition: the *givenPosition* must be between 0 and the total elements of the list - 1.
Postcondition: The element of the *givenPosition* is removed from the list.
Return: the removed element when removing the element from the list is success else null.

void clear ()
Description: To clear all elements of the list.
Parameter: None
Precondition: None
Postcondition: All the elements of the list are removed.
Return: None

boolean replace (int givenPosition, T newEntry)
Description: To replace the *newEntry* to the list based on the *givenPosition*.
Parameter: a *givenPosition* and a *newEntry*.
Precondition: the *givenPosition* must be between 0 and the total elements of the list – 1.
Postcondition: The element at the *givenPosition* of the list is replaced with the *newEntry*.
Return: true if replacement successfully else false

T getEntry (int givenPosition)
Description: To retrieve element's data from the list based on the *givenPosition*.
Parameter: a *givenPosition*.
Precondition: the *givenPosition* must be between 0 and the total elements of the list – 1.
Postcondition: The list will remain unchanged.
Return: the element's data if the *givenPosition* is valid else null.

boolean contains (T anEntry)
Description: To check if the list contains the entry (*anEntry*).
Parameter: an *anEntry*.
Precondition: None
Postcondition: The list will remain unchanged.
Return: true if the list contains at least one *anEntry* else false.

boolean equals (T anEntry, T nodeData)
Description: To check the *nodeData* is equal to *anEntry*.
Parameter: an *anEntry* and a *nodeData*.
Precondition: None
Postcondition: None.
Return: true if the *nodeData* and the *anEntry* is equals else false.

int getLength ()
Description: To retrieve the length of the list.
Parameter: None.
Precondition: None
Postcondition: The list will remain unchanged.
Return: the length of the list.

boolean isEmpty ()
Description: To check if the list is empty.
Parameter: None.
Precondition: None
Postcondition: The list will remain unchanged.
Return: true if the list is empty else false.

boolean isFull ()
Description: To determine if the list is full.
Parameter: None.
Precondition: None
Postcondition: The list will remain unchanged.
Return: true if the list is full else false.

6. ADT Implementation

The ADT I used to store the data of the board in this game (Connect Four) is linked list ADT which implements the list interface. The source code of the list interface and linked list class is shown below:

Java interface: list interface

```
public interface ListInterface<T> {

  public boolean add(T newEntry);

  public boolean add(int newPosition, T newEntry);

  public T remove(int givenPosition);

  public void clear();

  public boolean replace(int givenPosition, T newEntry);

  public T getEntry(int givenPosition);

  public boolean contains(T anEntry);

  public boolean equals(T anEntry, T nodeData);

  public int getLength();

  public boolean isEmpty();

  public boolean isFull();
}
```

Implementation class: Linked List

```
public class LinkedList<T> implements ListInterface<T> {

    private Node node; // based on col
    private int length; // represent row

    public LinkedList() {
        clear();
    }

    @Override
    public boolean add(T newEntry) {
        Node newNode = new Node(newEntry);

       if (isEmpty()) {
         node = newNode;
       } else {
           Node tempNode = node;
           for(int i = 0 ; i < length; i++ ){
              if(tempNode.next != null){
                  tempNode = tempNode.next;
              }
           }
           tempNode.next = newNode;

       }
       length++;
       return true;
    }
```

```
@Override
public boolean add(int newPosition, T newEntry) {

    if ((newPosition >= 0) && (newPosition < length)) {
      Node newNode = new Node(newEntry);

      if (isEmpty() || (newPosition == 1)) {
        newNode.next = node;
        node = newNode;
      } else {
        Node tempNode = node;
        for (int i = 1; i < newPosition - 1; i++) {
          tempNode = tempNode.next;
        }

        newNode.next = tempNode.next;
        tempNode.next = newNode;
      }

      length++;

    } else {
      return false;
    }

    return true;
}
```

```
@Override
public T remove(int givenPosition) {
    T removeValue;

    if ((givenPosition >= 0) && (givenPosition < length)) {

      if (givenPosition == 1) {
        removeValue = (T) node.data;
        node = node.next;
      } else {

        Node tempNode = node;
        for (int i = 1; i < givenPosition - 1; i++) {
          tempNode = tempNode.next;
        }

        removeValue = (T) tempNode.next.data;
        tempNode.next = tempNode.next.next;
      }
      length--;

    }else{
        return null;
    }

    return removeValue;
}
  @Override
  public final void clear() {
      node = null;
      length = 0;
  }


  @Override
  public boolean replace(int givenPosition, T newEntry) {

      if ((givenPosition >= 0) && (givenPosition < length)) {
        Node currentNode = node;
        for (int i = 0; i < givenPosition ; i++) {
          currentNode = currentNode.next;
        }
        currentNode.data = newEntry;
      } else {
        return false;
      }

      return true;
  }
```

```
@Override
public T getEntry(int givenPosition) {

    if ((givenPosition >= 0) && (givenPosition < length)) {
      Node currentNode = node;
      for (int i = 0; i < givenPosition; i++) {
        currentNode = currentNode.next;
      }
      return (T) currentNode.data;
    }

    return null;
}


@Override
public boolean contains(T anEntry) {

    Node tempNode = node;

    while (tempNode != null) {
        if (equals(anEntry, (T)tempNode.data)){
            return true;
        }else{
            tempNode = tempNode.next;
        }
    }

        return false;
}

@Override
public boolean equals(T anEntry, T nodeData) {
    return anEntry.equals(nodeData);
}


@Override
public int getLength() {
    return length;
}
```

```
    @Override
    public boolean isEmpty() {
        return length == 0;
    }


    @Override
    public boolean isFull() {
      return false;
    }

      //---------- Node ----------//
       public class Node<T> {

          private T data;
          private Node next;

          private Node(T data) {
            this.data = data;
            next = null;
          }

          private Node(T data, Node next) {
            this.data = data;
            this.next = next;
          }

      }
}
```

7. Selection of Collection ADTs

The ADT I use in my client program is the linked list collection ADT which I use to store the character of each row in the specific column of the game board. In this game design, the players will select the columns to insert their token and the token will be automatically assigned to the bottom of the columns if there is no any token. If there are some tokens inserted in the column, the latest insert token will place on the top of them but not the top of the column. Therefore, linked list ADT is suitable to use by my board class in storing the token of the rows that contains in each game board's columns because the element in the linked list is in ordered arrangement and can directly point the node of the new element to the previous element so that the new element will be at the top of the previous element. Figure 7.1 has shown that the pointer direction of the list which is the top element will point to the element which is under them. Besides, the Figure 7.1 also showed that each list represents the column of the board and each element data in the list represents the rows in the column.

Besides, by using this linked list ADT, I can replace the original character of the specific row in the specific column to the player's token character while the player enters the column number for inserting their token. In addition, I am using this ADT to retrieve the column's elements for displaying the board and comparing the data to check if the winner is existing. This ADT is useful while applying in the program which is able to store the character of the token into the specific column with assigning the token on the top of the previous token easily. The linked list also will assign the element of the list in an ordered arrangement.

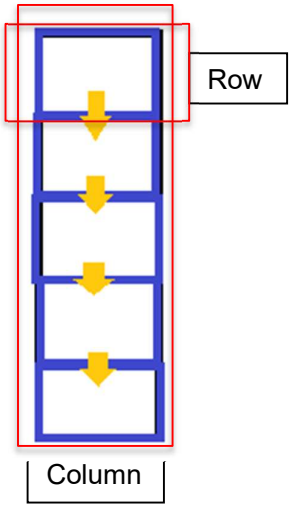


Figure 7.1 The pointer direction of the linked list

8. Implementation of Client Program

My client program is called *BoardConnectFour*. Below are the screenshots of **the functions' code** in the BoardConnectFour.java and also the **methods of the functions called** in the main program (BoardMain.java) to show how I called the functions from the main.

```
Player currentPlayer[];
Board board;


public void createNewBoard(Player currentplayer[], char tokenChar[],
        LinkedList<Character> boardCol[], Board boardTime){

    this.currentPlayer = currentplayer;

    board = boardTime;

    for(int j = 0; j < boardCol.length; j++){
        boardCol[j] = new LinkedList();
        for(int i = 0; i < board.getRows(); i++){
            boardCol[j].add('-');
        }
    }
    board.setBoardCol(boardCol);
}
```

Figure 8.1.1 Create New Board Function with Assign the Checkerboard

```
BoardConnectFour mainBoard;
Board board;
LinkedList<Character> boardCol[] = new LinkedList[cols];
```

Figure 8.1.2 Variables Declaration in Main

```
    if(gamemode==1)
currentPlayer = play.retrievePlayerForEachRound(PlayerList, winnerList,round);
    else currentPlayer=tour.retrievePlayerForEachRound(numOfPlayer);
currentToken[0] = token.retrieveToken(currentPlayer[0].getName(), CirStr);
currentToken[1] = token.retrieveToken(currentPlayer[1].getName(), CirStr);

System.out.println(currentPlayer[0].getName() + " >> "+ currentPlayer[1].getName());

mainBoard = new BoardConnectFour();
board = new Board(currentToken, rows);

mainBoard.createNewBoard(currentPlayer, currentToken,boardCol, board);
```

Figure 8.1.3 Variables Initialization and createNewBoard Function called in Main

```
public void displayBoard(){

    // display column number
    System.out.print("\n");
    for(int j = 0; j < board.getBoardCol().length; j++){
        System.out.print((j+1) +" ");
    }

    //display board
    for(int i = 0; i < board.getRows(); i++){
        System.out.print("\n");
        for(int j = 0; j < board.getBoardCol().length; j++){
            System.out.print(board.getSpecificBoardCol(j).getEntry(i)+" ");

        }
    }

}
```

Figure 8.2.1 Display Board Function

```
mainBoard.displayBoard();
```

Figure 8.2.2 Display Board Function Called in Main

```
public boolean insertToken(){

    if(board.getTurn() % 2 == 1){
        System.out.print("\n\n"+currentPlayer[0].getName() +"'s turn: ");
    }else{
        System.out.print("\n\n"+currentPlayer[1].getName() +"'s turn: ");
    }

    String selectCol = scan.nextLine();

     if(selectCol.compareTo("") == 0){
        System.out.println("Pls enter a column number ...");
        return false;
    }
    char selectedCol = selectCol.charAt(0);

    if(Character.isDigit(selectedCol)){
        board.setInsertTokenPosition(Character.getNumericValue(selectedCol));
    }else{
        System.out.println("Invalid NUMBER ! \n\nPls try again...");
        return false;
    }
    //check selected column for insert token
    if(board.getInsertTokenPosition() <= 0 ||
            board.getInsertTokenPosition()> board.getBoardCol().length){
        System.out.println("Invalid column number! Pls try again...");
        return false;
    }else{
        // check if col full
        int isColFull = board.getRows();
        for(int i = 0; i < board.getRows(); i++){
            Character entry = board.getSpecificBoardCol(
                board.getInsertTokenPosition()-1).getEntry(board.getRows()-1- i);

            if(entry == '-'){
                break;
            }else{
                isColFull--;
            }
        }
        if(isColFull == 0){
            System.out.println("Col " +board.getInsertTokenPosition()+
                    " is full ! Pls proceed to another row ! ");
            return false;
        }
    }
   return true;
}
```

Figure 8.3.1 Insert Token Function

```
do{
    insertSuccess = mainBoard.insertToken();

}while(!insertSuccess);
```

Figure 8.3.2 Insert Token Function Called in Main

```
public void assignToken(){

    boolean assignSuccess;
    int replaceRow = board.getRows();
    Character entry;
    int turn = board.getTurn();
    do{
        replaceRow--;
        entry = board.getSpecificBoardCol(board.getInsertTokenPosition()-1).getEntry(replaceRow);

    }while(entry != '-');


    if(turn % 2 == 1){
        assignSuccess = board.getSpecificBoardCol(
            board.getInsertTokenPosition()-1).replace(replaceRow, board.getSpecificToken(0));

    }else{
        assignSuccess = board.getSpecificBoardCol(
            board.getInsertTokenPosition()-1).replace(replaceRow, board.getSpecificToken(1));
    }

    if(assignSuccess){
        turn++;
        board.setTurn(turn);
    }
}
```

Figure 8.4.1 Assign Token Function

```
mainBoard.assignToken();
```

Figure 8.4.2 Assign Token Function called in Main After Token Insert Successfully

```
public int checkResult(int connectNum){
    Character entry;
    switch (connectNum) {
            case 3:
                //check for horizontal win
                for(int i = board.getRows() - 1 ; i > 0; i--){
                    for(int j = 0; j < board.getBoardCol().length - 2; j++){
                        entry = board.getSpecificBoardCol(j).getEntry(i);

                        if( !(entry.equals('-'))
                                && board.getSpecificBoardCol(j+1).getEntry(i).equals(entry)
                                && board.getSpecificBoardCol(j+2).getEntry(i).equals(entry)){
                            return board.getTurn();
                        }
                    }
                }
                //check for vertical win
                for(int j = 0; j < board.getBoardCol().length; j++){
                    for(int i = 0; i < board.getRows() - 2; i++){
                        entry = board.getSpecificBoardCol(j).getEntry(i);

                        if( !(entry.equals('-'))
                                && board.getSpecificBoardCol(j).getEntry(i+1).equals(entry)
                                && board.getSpecificBoardCol(j).getEntry(i+2).equals(entry)){
                            return board.getTurn();
                        }
                    }
```

Figure 8.5.1 Check Result Function (Connect 3)

```
                //check for diagonal win (+ve slope)
                for(int i = board.getRows() - 1 ; i > 2; i--){
                    for(int j = 0; j < board.getBoardCol().length - 2; j++){
                        entry =  board.getSpecificBoardCol(j).getEntry(i);

                        if( !(entry.equals('-'))
                                &&  board.getSpecificBoardCol(j+1).getEntry(i-1).equals(entry)
                                &&  board.getSpecificBoardCol(j+2).getEntry(i-2).equals(entry)){
                            return board.getTurn();
                        }
                    }
                }

                //check for diagonal win (-ve slope)
                for(int i = board.getRows() - 1 ; i > 0; i--){
                    for(int j = board.getBoardCol().length - 1; j > 1; j--){
                        entry = board.getSpecificBoardCol(j).getEntry(i);

                        if( !(entry.equals('-'))
                                && board.getSpecificBoardCol(j-1).getEntry(i-1).equals(entry)
                                && board.getSpecificBoardCol(j-2).getEntry(i-2).equals(entry)){
                            return board.getTurn();
                        }
                    }
                }
                break;
```

Figure 8.5.2 Check Result Function (Connect 3)

```
case 5:
    //check for horizontal win
     for(int i = board.getRows() - 1 ; i > 0; i--){
        for(int j = 0; j < board.getBoardCol().length - 4; j++){
            entry = board.getSpecificBoardCol(j).getEntry(i);

            if( !(entry.equals('-')) && board.getSpecificBoardCol(j+1).getEntry(i).equals(entry)
                    && board.getSpecificBoardCol(j+2).getEntry(i).equals(entry)
                    && board.getSpecificBoardCol(j+3).getEntry(i).equals(entry)
                    && board.getSpecificBoardCol(j+4).getEntry(i).equals(entry)){
                return board.getTurn();
            }
        }
    }
    //check for vertical win
      for(int j = 0; j < board.getBoardCol().length; j++){
        for(int i = 0; i < board.getRows() - 4; i++){
            entry = board.getSpecificBoardCol(j).getEntry(i);

            if( !(entry.equals('-')) && board.getSpecificBoardCol(j).getEntry(i+1).equals(entry)
                    && board.getSpecificBoardCol(j).getEntry(i+2).equals(entry)
                    && board.getSpecificBoardCol(j).getEntry(i+3).equals(entry)
                    && board.getSpecificBoardCol(j).getEntry(i+4).equals(entry)){
                return board.getTurn();
            }
        }
    }
```

Figure 8.5.3 Check Result Function (Connect 5)

```
    //check for diagonal win (+ve slope)
    for(int i = board.getRows() - 1 ; i > 3; i--){
        for(int j = 0; j < board.getBoardCol().length - 4; j++){
            entry = board.getSpecificBoardCol(j).getEntry(i);

            if( !(entry.equals('-')) && board.getSpecificBoardCol(j+1).getEntry(i-1).equals(entry)
                    && board.getSpecificBoardCol(j+2).getEntry(i-2).equals(entry)
                    && board.getSpecificBoardCol(j+3).getEntry(i-3).equals(entry)
                    && board.getSpecificBoardCol(j+4).getEntry(i-4).equals(entry)){
                return board.getTurn();
            }
        }
    }
    //check for diagonal win (-ve slope)
     for(int i = board.getRows() - 1 ; i > 3; i--){
        for(int j = board.getBoardCol().length - 1; j > 3; j--){
            entry = board.getSpecificBoardCol(j).getEntry(i);

            if( !(entry.equals('-')) && board.getSpecificBoardCol(j-1).getEntry(i-1).equals(entry)
                    && board.getSpecificBoardCol(j-2).getEntry(i-2).equals(entry)
                    && board.getSpecificBoardCol(j-3).getEntry(i-3).equals(entry)
                    && board.getSpecificBoardCol(j-4).equals(entry)){
                return board.getTurn();
            }
        }
    }
    break;
```

Figure 8.5.4 Check Result Function (Connect 5)

```
default: // connect 4

    //check for horizontal win
    for(int i = board.getRows() - 1 ; i >= 0; i--){
        for(int j = 0; j < board.getBoardCol().length - 3; j++){
            entry = board.getSpecificBoardCol(j).getEntry(i);

            if( !(entry.equals('-')) && board.getSpecificBoardCol(j+1).getEntry(i).equals(entry)
                    && board.getSpecificBoardCol(j+2).getEntry(i).equals(entry)
                    && board.getSpecificBoardCol(j+3).getEntry(i).equals(entry)){
                return board.getTurn();
            }
        }
    }

    //check for vertical win
    for(int j = 0; j < board.getBoardCol().length; j++){
        for(int i = 0; i <= board.getRows()  - 4; i++){
            entry = board.getSpecificBoardCol(j).getEntry(i);

            if( !(entry.equals('-')) && board.getSpecificBoardCol(j).getEntry(i+1).equals(entry)
                    && board.getSpecificBoardCol(j).getEntry(i+2).equals(entry)
                    && board.getSpecificBoardCol(j).getEntry(i+3).equals(entry)){
                return board.getTurn();
            }
        }
    }
```

Figure 8.5.5 Check Result Function (Default - Connect 4)

```
        //check for diagonal win (+ve slope)
        for(int i = board.getRows()  - 1 ; i > 2; i--){
            for(int j = 0; j < board.getBoardCol().length - 3; j++){
                entry = board.getSpecificBoardCol(j).getEntry(i);

                if( !(entry.equals('-')) && board.getSpecificBoardCol(j+1).getEntry(i-1).equals(entry)
                        && board.getSpecificBoardCol(j+2).getEntry(i-2).equals(entry)
                        && board.getSpecificBoardCol(j+3).getEntry(i-3).equals(entry)){
                    return board.getTurn();
                }
            }
        }
        //check for diagonal win (-ve slope)
        for(int i = board.getRows() - 1 ; i > 2; i--){
            for(int j = board.getBoardCol().length - 1; j > 2; j--){
                entry = board.getSpecificBoardCol(j).getEntry(i);

                if( !(entry.equals('-')) && board.getSpecificBoardCol(j-1).getEntry(i-1).equals(entry)
                        && board.getSpecificBoardCol(j-2).getEntry(i-2).equals(entry)
                        && board.getSpecificBoardCol(j-3).getEntry(i-3).equals(entry)){
                    return board.getTurn();
                }
            }
        }
        break;
}
return 0;
```

Figure 8.5.6 Check Result Function (Connect 4)

```
checkResult = mainBoard.checkResult(connectNum);
```

Figure 8.5.7 Check Result Function Called in Main

```
//---- check if all col full ---//
public boolean isAllColFull(){

    for(int j = 0; j < board.getBoardCol().length; j++){
        for(int i = 0; i < board.getRows(); i++){
            if( board.getSpecificBoardCol(j).getEntry(i) == '-'){
                return false;
            }
        }
    }
      return true;
}
```

Figure 8.6.1 Check is All Column in the Board is Full

```
if(mainBoard.isAllColFull()){
    System.out.println("\nTHERE IS NO WINNER !! This round will PLAY AGAIN !");
    insertSuccess = false;
}else{
```

Figure 8.6.2 isAllColFull Function called in Main

9. Game Source Code

All of the source code of the Connect Four Game is available to view and get at:

https://github.com/yongkang0827/Connect4