CSCI 230 Program Four   Assigned Monday Sept 7   Due Friday Sept 11 by midnight

In this program we will implement a Queue ADT to accumulate movement commands issued by one or more players in a networked simulation or game. In these applications a server computer maintains a master copy of the state of the simulation environment or game board. It receives movement commands from each user and updates it model of the environment or game board. Changes to the environment or board are then broadcast to each user so everyone sees the current state.

You will have the opportunity to work in teams of two to implement the following:

1) Class to represent one player's movement turn in the map or maze. Private instance variables include a unique integer player id number and current movement direction selected from one of five enumerated values. An enumeration type lets you declare variables whose values can be only one of several specified values. The public methods of Move are as follows. You will need to conform to these public method signatures for compatibility with the other modules of the program.

public class Move

{

  public enum Direction { MOVE_NONE, MOVE_LEFT, MOVE_RIGHT, MOVE_DOWN, MOVE_UP };

  public Move(): default constructor assigns player number to 0 and direction to MOVE_NONE.

  public void setPlayer(int N): Sets player id number to given value assumed to be >= 0.

  public void setDirection(Direction D): Sets move direction to given enumerated value.

  public int getPlayer(): return unique player number >= 0.

  public Direction getDirection(): get movement direction.

  public static String directionToString(Direction d): return String "NONE", "LEFT", "RIGHT", "DOWN", or "UP" as corresponds to the given movement direction.

}

2) Represent each player by writing the following class:

Player(): default constructor initializes position X, position Y to 0, score to 0, and symbol to 'A'.

public Player(char sym): constructor initializes position X, position Y to 0, score to 0, and symbol to the given char value.

public void setXY(int x, int y): set x-y coordinates of player in the game board. X-gives the horizontal position 0 to board size – 1 and Y-gives the vertical position 0 to board size – 1.

public void setScore(int s): Set player score to given integer value.

public int getX(): Return x-coordinate as an integer.

public int getY(): Return y-coordinate as an integer.

public char getSymbol(): Return char symbol that is used to display this player in the game board.

public int getScore(): Return score of player.

3) Define an interface to represent a queue of player moves.

```
public interface MoveQueueADT

{

  public boolean isEmpty(): return true if queue is empty; else, false.

  public boolean isFull(): return true if full; else, false.

  public boolean insert(Move M): if queue is not full, append given Move to end of queue and

            return true; else, return false, leaving queue unchanged.

  public Move remove(): if queue is not empty, remove and return first Move from queue; else, null.

}
```

4) Define class DLMoveQueue which implements MoveQueueADT. It will use a doubly-linked list of DNode objects, each of which contains a Move and DNode previous and DNode next pointers. The DLMoveQueue will implement a doubly-linked list of DNodes, where the insert method always adds the new DNode to the end of the list and the remove method always takes out the first DNode.

You will be provided with Java classes that read the starting board configuration and list of player moves from text files.

You will need to complete the movePlayer(Move M) method of class GameMap.

@param M Given move that identifies the one player who makes the move and the Direction of movement: MOVE_LEFT: decrease x-position by one, MOVE_RIGHT: increase x-position by one, MOVE_DOWN: increase y-position by one, MOVE_UP: decrease y-position by one. If new player position will be a valid board position x and y are between [0, BOARD_SIZE-1] and the new player position is NOT a FENCE_TILE, then allow the player to occupy that new map square. If the new square contains a SCORE_TOKEN then increase that player's score by the value of the score token. Enact a valid move by setting the player's old map square to the BLANK_TILE and the new player's square to that player's symbol. Otherwise, if the move is not valid, then make no change to the board. Return true if move is valid.

Submit the following files – the top of each source file must include comments that name both members of the team and commentary on what each person contributed to that source file.

Move.java, Player.java, MoveQueueADT.java, DNode.java, DLMoveQueue.java, Maze.txt, Moves.txt

Maze.txt is your original starting game board. Moves.txt is your original game play moves that ends with all score $ tokens cleared from the board.

Please ZIP all files together. Both team members must submit the same ZIP file into OAKS.

Grading Criteria

10	Move class
10	Player class
5	MoveQueueADT interface (contains no method bodies)
10	DNode – double-linked list node contains Move and prev/next pointers.
45	DLMoveQueue
5	Maze.txt – an original starting board containing at least 4 score tokens.
5	Moves.txt – list of player moves (at least 16 moves) that ends in all score tokens cleared.
10	Comments for all source files that describe the contribution of each team member.