CS 1027A - Assignment 3 - The Floor Is Lava!

● Graded

Student

Mohammed Ali Abdul-nabi

Total Points

20 / 20 pts

Autograder Score

15.0 / 15.0

Passed Tests

[--------------- TEST 00 (Path) --------------] (0.5/0.5)
[--------------- TEST 01 (Path) --------------] (0.5/0.5)
[--------------- TEST 02 (Path) --------------] (0.5/0.5)
[--------------- TEST 03 (Path) --------------] (0.5/0.5)
[--------------- TEST 04 (Path) --------------] (0.5/0.5)
[--------------- TEST 05 (Path) --------------] (0.5/0.5)
[--------------- TEST 06 (Path) --------------] (0.5/0.5)
[--------------- TEST 07 (Path) --------------] (0.5/0.5)
[--------------- TEST 08 (Path) --------------] (0.5/0.5)
[--------------- TEST 09 (Path) --------------] (0.5/0.5)
[--------------- TEST 00 (Stack) --------------] (0.5/0.5)
[--------------- TEST 01 (Stack) --------------] (0.5/0.5)
[--------------- TEST 02 (Stack) --------------] (0.5/0.5)
[--------------- TEST 03 (Stack) --------------] (0.5/0.5)
[--------------- TEST 04 (Stack) --------------] (0.5/0.5)
[--------------- TEST 05 (Stack) --------------] (0.5/0.5)
[--------------- TEST 06 (Stack) --------------] (0.5/0.5)
[--------------- TEST 07 (Stack) --------------] (0.5/0.5)
[--------------- TEST 08 (Stack) --------------] (0.5/0.5)
[--------------- TEST 09 (Stack) --------------] (0.5/0.5)

Question 2

Code Logic

1 / 1 pt

✓ **– 0 pts** Correct - Meaningful variable names, private instance variables used

– 0.5 pts Click here to replace this description.

– 1 pt Wrong - No meaningful logic

Question 3

Code Formatting/Readability

2 / 2 pts

✓ **– 0 pts** Correct

– 0.5 pts Click here to replace this description.

– 1 pt Click here to replace this description.

– 1.5 pts Click here to replace this description.

– 2 pts No proper code formatting. Code not readable

Question 4

Comments

2 / 2 pts

✓ **– 0 pts** Correct - Comments are proper and relevant

– 0.5 pts Click here to replace this description.

– 1 pt Click here to replace this description.

– 2 pts Wrong - Comments are NOT proper and relevant or/and no comments included.

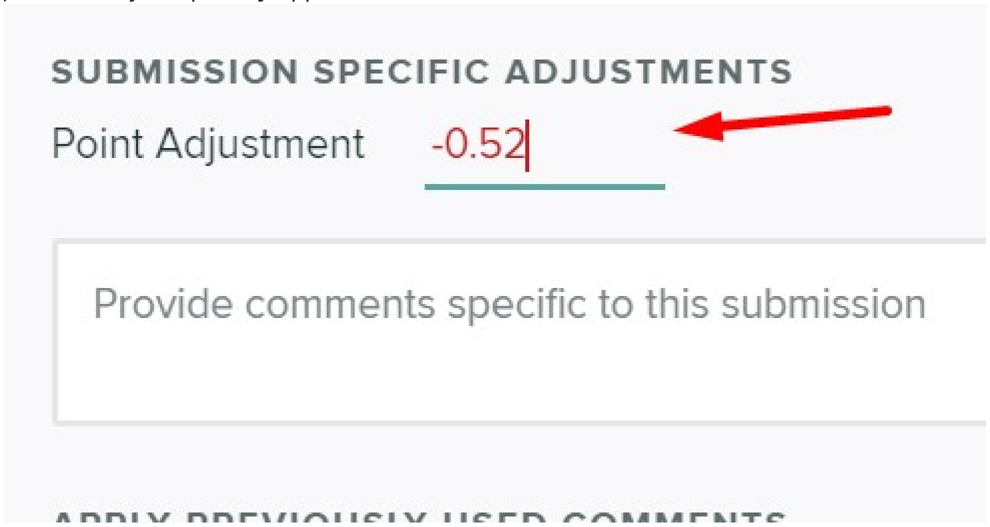
Question 5

Penalties **0** / 0 pts

5.1 *Late Submissions* -2/day **0** / 0 pts

✓ **– 0 pts** Click here if not late.

– 0 pts **@TAs: DO NOT ADD YOUR OWN RUBRICS HERE** *Please enter the deduction in the* **Point Adjustment** *field below if late penalty applies.*



SUBMISSION SPECIFIC ADJUSTMENTS

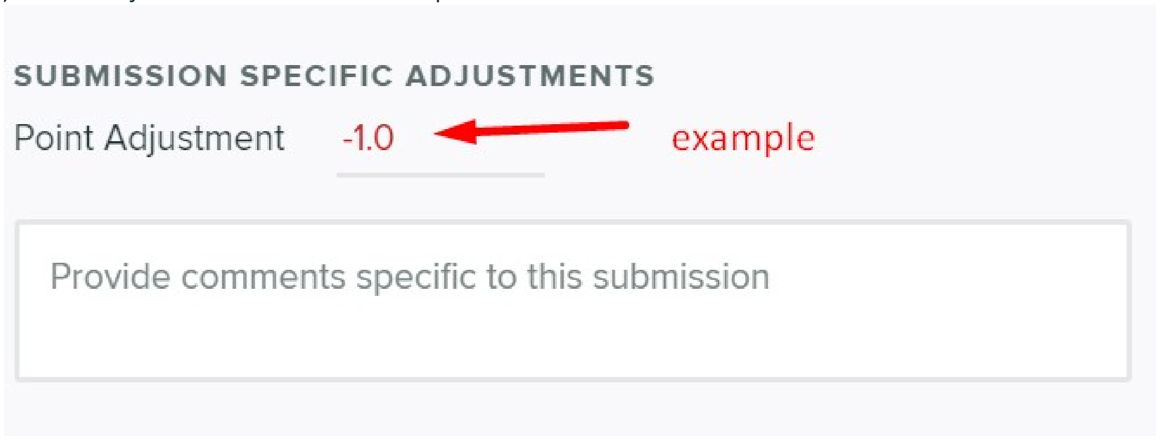
Point Adjustment -0.52

Provide comments specific to this submission

5.2 Incorrect submission (doesn't compile, package line, .class file, etc.) -5 **0** / 0 pts

✓ **– 0 pts** Click here if no submission error

– 5 pts **@TAs: DO NOT ADD YOUR OWN RUBRICS HERE** *Please enter the deduction in the* **Point Adjustment** *field below if submission incorrect* Example:



SUBMISSION SPECIFIC ADJUSTMENTS

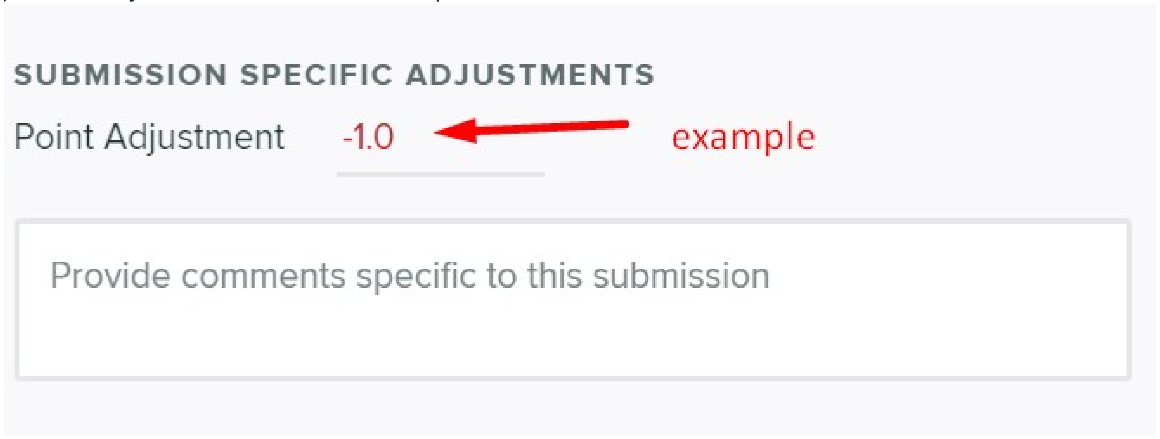
Point Adjustment -1.0

Provide comments specific to this submission

5.3 Incorrect instance variables or methods -2 **0** / 0 pts

✓ **– 0 pts** Click here if no submission error

– 2 pts **@TAs: DO NOT ADD YOUR OWN RUBRICS HERE** *Please enter the deduction in the* **Point Adjustment** *field below if submission incorrect* Example:


SUBMISSION SPECIFIC ADJUSTMENTS

Point Adjustment -1.0

Autograder Results

[--------------- TEST 00 (Path) --------------] (0.5/0.5)

[--------------- TEST 01 (Path) --------------] (0.5/0.5)

[--------------- TEST 02 (Path) --------------] (0.5/0.5)

[--------------- TEST 03 (Path) --------------] (0.5/0.5)

[--------------- TEST 04 (Path) --------------] (0.5/0.5)

[--------------- TEST 05 (Path) --------------] (0.5/0.5)

[--------------- TEST 06 (Path) --------------] (0.5/0.5)

[--------------- TEST 07 (Path) --------------] (0.5/0.5)

[--------------- TEST 08 (Path) --------------] (0.5/0.5)

[--------------- TEST 09 (Path) --------------] (0.5/0.5)

[--------------- TEST 00 (Stack) --------------] (0.5/0.5)

[--------------- TEST 01 (Stack) --------------] (0.5/0.5)

[--------------- TEST 02 (Stack) --------------] (0.5/0.5)

[--------------- TEST 03 (Stack) --------------] (0.5/0.5)

[--------------- TEST 04 (Stack) --------------] (0.5/0.5)

[--------------- TEST 05 (Stack) --------------] (0.5/0.5)

[--------------- TEST 06 (Stack) --------------] (0.5/0.5)

[--------------- TEST 07 (Stack) --------------] (0.5/0.5)

[--------------- TEST 08 (Stack) --------------] (0.5/0.5)

[--------------- TEST 09 (Stack) --------------] (0.5/0.5)

Submitted Files

ArrayStack.java Download

```java

public class ArrayStack <T> implements StackADT<T> {
    private T[] array;
    private int top;

    public ArrayStack(){                                //Constructor initialize data.
        this.array = (T[]) new Object[10];
        top = -1;
    }


    public void push(T element) {                       //Add element to top of stack.
        expandCapacity();                               //Expand capacity
        this.array[this.top+1] = element;               //Add element on "top" location.
        top += 1;                                       //Iterate top.

    }


    public T pop() throws StackException {
        if (isEmpty()) {
            throw new StackException("Stack is empty");
        }
        shrinkCapacity();               //Shrink Capacity if needed
        T topStack = array[top];        //For the return statement we return the top of stack.
        array[top] = null;              //Not necessary but delete it.
        top --;                         //Shrink top.

        return topStack;
    }


    public T peek() throws StackException {             //Returns the top of the ArrayStack.
        if (isEmpty()) {                                //Checks if its empty first.
            throw new StackException("Stack is empty");     //Throws exception if it is.
        }
        return array[top];
    }

    public boolean isEmpty() {                          //Checks if ArrayStack is empty
        return top == -1;
    }

    public int size() {                                 //Returns size of the ArrayStack
        return top+1;
    }

    public void clear() {                                    //Clears all data in ArrayStack
        while (!isEmpty()){                                  //Pop when it's not empty.
            pop();
        }
        top = -1;                                            //When its empty we reset top
        this.array = (T[]) new Object[10];                   //and initialCapacity of ArrayStack.
    }

    public int getCapacity(){                                //Get total capacity.
        return array.length;
    }

    public int getTop(){                                     //Gettor for private top.
        return top;
    }


    public String toString() {                               //Return the ArrayStack in a string.
        if (isEmpty()) {                                     //Check is empty first and return.
            return "Empty stack.";
        }

        StringBuilder out = new StringBuilder("Stack: ");   //String Builder new String.
        for (int i = top; i >= 0; i--) {                     //Iterate through the ArrayStack.
            out.append(array[i]);                            //Add onto array each time.
            if (i > 0) {                                     //Adds the comma and space on all except last.
                out.append(", ");
            }
        }
        out.append(".");                                     //Add dot at end.
        return out.toString();
    }


    private void expandCapacity( ) {
        //Create a new array, as it to generic data type and then set it to the old array capacity + 10
        if ((double) size() / (double) array.length >= .75) {   //Checks if we are at 75% usage.
            int newCapacity = array.length + 10;
            T[] expandedArray = (T[]) new Object[newCapacity];
            for (int i = 0; i <= top; i++) {        //Duplicates the items in the array.
                expandedArray[i] = array[i];
            }
        this.array = expandedArray;                          //Makes the old array the expandArray.
        }
    }


    private void shrinkCapacity() {
        //Create a new array, as it to generic data type and then set it to the old array capacity - 10
        if ((double) size() / (double) array.length <= 0.25 && array.length >= 20) {    //Checks if we are at 25% usage and above 20 slots.
            int newCapacity = array.length - 10;
            T[] shrunkArray = (T[]) new Object[newCapacity];
            for (int i = 0; i <= top; i++) {       //Duplicates the items in the array.
                shrunkArray[i] = array[i];
            }
            this.array = shrunkArray;                              //Makes the old array the shrunkArray.
        }
    }
}

```

MineEscape.java Download

```java

public class MineEscape {
    private Map map;
    //Counter for how much gold we have picked up along the way.
    public int numGold;
    //Array Storing number of keys.
    private int[] numkeys;

    public MineEscape(String filename)  {      //Initialize variables.
        try {
            this.map = new Map(filename);
            numGold = 0;
            numkeys = new int[3];              //Follows RGB so numKeys[0] = red,numKeys[1] = green, numKeys[2] = blue.

        } catch (Exception e) {               //Print exception thrown.
            System.out.println(e.getMessage());
        }
    }

    private MapCell findNextCell(MapCell cell) { //For loop for each to check all sides in priority.

        for (int i = 0; i < 4; i++) {
            MapCell neighbor = cell.getNeighbour(i);
            if (neighbor != null && !neighbor.isMarked()) {        //Make sure neighbour is not null or marked.
                if (neighbor.isExit()) {                          //Check if it's an exit.
                    return neighbor;
                }
            }
        }

        for (int i = 0; i < 4; i++) {
            MapCell neighbor = cell.getNeighbour(i);
            if (neighbor != null && !neighbor.isMarked()) {           //Make sure neighbour is not null or marked.
                // Check for key cells
                if (neighbor.isKeyCell() || neighbor.isGoldCell()) {    //Check if it is a key or gold.
                    if (neighbor.isBlue()) {                         //Check color of they key
                        numkeys[2] = numkeys[2] + 1;                   //Add key to corresponding cell in array.
                    }
                    if (neighbor.isGreen()) {
                        numkeys[1] = numkeys[1] + 1;
                    }
                    if (neighbor.isRed()) {
                        numkeys[0] = numkeys[0] + 1;
                    }
                    return neighbor;
                }
            }
        }

        for (int i = 0; i < 4; i++) {
            MapCell neighbor = cell.getNeighbour(i);
            if (neighbor != null && !neighbor.isMarked()) {            //Make sure neighbour is not null or marked.
                if (neighbor.isLockCell()) {                           //Check if it is a lock cell.
                    //Check if we have enough keys to open it and return the first one we have enough for.
                    if ((neighbor.isBlue() && numkeys[2] >= 1) || (neighbor.isRed() && numkeys[0] >= 1) || (neighbor.isGreen() && numkeys[1] >= 1)) {
                        return neighbor;
                    }
                }
            }
        }

        for (int i = 0; i < 4; i++) {
            MapCell neighbor = cell.getNeighbour(i);

            if (neighbor != null && !neighbor.isMarked()) {            //Make sure neighbour is not null or marked.
                if (neighbor.isFloor()) {                              //If it's a floor cell return it
                    cell = neighbor;
                    return cell;
                }
            }
        }
        return null;                              //If non of these is possible than we are stuck and must backtrack.
    }


    public String findEscapePath(){
        ArrayStack<MapCell> s = new ArrayStack<>();     //Stores the Mapcell objects.
        s.push(map.getStart());                         //Add starting position to Stack
        boolean running = true;                         //Program is running
        map.getStart().markInStack();                   //Mark the start of the map as part of path

        //Path string that we store the way out of the map, to be retuned at the end.
        StringBuilder escapeRoute = new StringBuilder("Path: " + map.getStart().getID() + " ");

        while (!s.isEmpty() && running){                //When the program is running and the stack isn't empty.
            MapCell curr = s.peek();

            if (curr.isExit()){                                     //Stop the program when on the exit cell.
                running = false;
                break;
            }

            if(curr.isGoldCell()){                          //If the current cell is a gold cell
                numGold++;                                  //Increase our gold count
                curr.changeToFloor();                       //Change the  cell from gold to floor
            }

            if(curr.isKeyCell()){                           //Change the key cell to a floor cell after pick up
                curr.changeToFloor();
            }

            for(int i = 0; i <= 3; i++){                    //Checking adjecent cells for lava.
                if(curr.getNeighbour(i) != null) {          //Make sure neighbour isn't null
                    if (curr.getNeighbour(i).isLava()) {    //If Neighbour is lava, gold is deleted
                        numGold = 0;
                    }
                }
            }

            MapCell next = findNextCell(curr);              //Find the next cell to go to from current position

            if(next == null){                               //If findNextCell returns null
                curr = s.pop();                             //Backtrack and mark outOfStack
                curr.markOutStack();                        //Repeats until another option is found
            }

            else{                                           //If it doesn't return null
                escapeRoute.append(next.getID() + " ");             //Add cell ID to escapePath
                s.push(next);                               //Add to path arraystack
                next.markInStack();                         //Mark it as in the Stack (path)
                if (next.isLockCell()){                     //If the cell is a lockCell
                    if(next.isRed() && numkeys[0] >= 1){    //Check each for which color and if we have a key for it
                        next.changeToFloor();               //If we have the key change the lockCell to a floor
                        numkeys[0] = numkeys[0] -1 ;}       //Reduce the number of corresponding keys by one.
                    if(next.isGreen() && numkeys[1] >= 1){
                        next.changeToFloor();
                        numkeys[1] = numkeys[1] -1 ;}
                    if(next.isBlue() && numkeys[2] >= 1){
                        next.changeToFloor();
                        numkeys[2] = numkeys[2] - 1;}
                }
            }

        }
        if (!running){                                      //When the program stops running
            escapeRoute.append(numGold + "G");              //Add amount of gold to end of escapeRoute string
            return escapeRoute.toString();
        }
        else {                                          //If there is not an option for nextCell than NoSolution
            return "No solution found";
        }
    }

    public static void main (String[] args) throws Exception {
        if (args.length != 1) {
            System.out.print("Map file not given in the arguments.");
        }
        else {
            MineEscape search = new MineEscape(args[0]);
            String result = search.findEscapePath();
            System.out.println(result);
        }
    }
}



```