CSE 8A Lecture 19

• Reading for next class: review quiz!

• REMINDERS:
  – LAST LAB tomorrow: PSA10
  – PSA10 due on Friday
  – CAPES! And a supplemental survey in lab tomorrow…
  – Confirm your exam change requests
1. How many times does the body of this loop execute?

```java
int index = 10;
while (index>=0)
{
    System.out.println("CSE8A ROCKS!");
    index = index - 1;
}
```

A. 11   B. 10   C. 0   D. Infinite
2. What does the following print

```java
int x = 10;
if(x<9)
{
    x = x + 3;
}
if(x>10)
{
    x = x + 1;
}
System.out.println(x);
```

A. 14  B. 13  C. 11  D. 10
3. What is the decimal number 21 in binary?

A. 10101
B. 10110
C. 11111
D. 00021
4. /**Method to copy part of the passed source sound into this sound at the given
start index
* @param source the source sound to copy from
* @param sourceStart the starting index to copy from in the source (the copy
* will include this)
* @param sourceStop the ending index (the copy won’t include this)
* @param targetStart the index to start copying into

public void splice (Sound source, int sourceStart, int sourceStop, int targetStart)
{
    //loop copying from source to target
    for(int sourceIndex = sourceStart, targetIndex = targetStart;
        sourceIndex < sourceStop && sourceIndex < this.getLength();
        sourceIndex++, targetIndex++)
    {
        this.setSampleValueAt(targetIndex,
            source.getSampleValueAt(sourceIndex));
    }
}

There seems like one might be missing. Why is there no parameter:
targetStop -- the index you want to stop copying into in the target sound?

A. we always copy up to the end of the target (calling object) sound, so we don't need a targetStop
B. The targetStop index is implicitly defined based on the targetStart and the (sourceStop-sourceStart)
C. targetStop is the same as sourceStop, so we don't have to pass it separately
D. This is an error, there should be a targetStop parameter
PSA – 10: Turtle path planning

• Basically consists on finding the way from the start to the goal locations.
PSA – 10: Turtle path planning

• We will use a **Map** (grid of cells)
  
  – Way of accessing this grid is similar to access pixels in a Picture: top-left is (0,0); bottom-right is (width-1, height-1);

• We actually have two matrices
  
  – Occupancy
  
  – Cost
PSA – 10: Turtle path planning

• How to get to the red Cell?
PSA – 10: Turtle path planning

• How to get to the red Cell?
PSA – 10: Turtle path planning

• What are the red numbers?

A) COST
B) STRAIGHT-LINE DISTANCE

A: Yes
B: No
PSA – 10: Turtle path planning

• Important ideas:
  - COST: how many “steps” (we count one cell up, down, left, right as one “step” left)
  - PATH: array of cells to follow (Cell[] path = new Cell[n])
  - NEIGHBORS: We consider 4 neighbors (up, down, left, right)
PSA – 10: Turtle path planning

• Basic idea:
  – We start in the “goal” cell: that cell has \( \text{cost} = \text{zero} \).
  – Every “free” neighbor has cost+1
  – The neighbor of that neighbor has cost+1+1
  – …
  – Propagate the wave until we find the \( \text{cost} \) from the start location.
PSA – 10: Turtle path planning

• How to get to the red Cell?
PSA – 10: Turtle path planning

• Steps:

1. Load map from a .txt file (all 0s and 1s). Initialize occupancyMap (0: free; 1: occupied)

2. Compute costMap (int[][][] costMap) given startCell and endCell
   • This method is given
   • Once we run it, we already know the length of the path!

• What is the total length of the path from the start to the end cell? (including both the start and the end cell)?

A) costMap.getWidth() * costMap.getHeight()

B) getCostMapValueAt(endCell) + 1

C) getCostMapValueAt(startCell) + 1
PSA – 10: Turtle path planning

- Steps (1, 2 just need to be run. 3, 4 you need to do)

  1. Load map from a .txt file (all 0s and 1s). Initialize occupancyMap (0: free; 1: occupied)

  2. Compute costMap (int[][] costMap) given startCell and endCell
     - This method is given
     - Once we run it, we already know the length of the path!

  3. Find the path (fill the Cell[] path) looping on the values of costMap.
     - Start in startCell, until we reach the endCell

  4. Make the turtle move from cell to cell to traverse all cells in the path in order.
PSA – 10: Turtle path planning

//Initialize the map here

int n1, n2;

Map map1 = new Map("map.txt");

map1.repaint();

...

Cell startC = new Cell(n1,n2);
Cell endC = new Cell(n1,n2);

Turtle lostTurtle = null;

Cell[] mazePath;
PSA – 10: Turtle path planning

CAREFUL! (typical issues so far)

1. Repaint

2. Checking “valid” cell values.
CAREFUL! (typical issues so far)

1. Repaint

```
//Initialize the map here
int n1, n2;
Map map1 = new Map("map.txt");
map1.repaint();
...
Cell startC = new Cell(n1, n2);
Cell endC = new Cell(n1, n2);

Turtle lostTurtle = null;
Cell[] mazePath;
```
CAREFUL! (typical issues so far)

2. Checking “valid” cell values.

Will these two pieces of code do the same thing?

A. Yes, always

B. Yes, but only for some cells

C. No, never

if (startC.getX() >= 0 && startC.getX() < map1.getMapWidth() && startC.getY() >= 0 && startC.getY() < map1.getMapHeight() && map1.isFree(startC))

if (map1.isFree(startC) && startC.getX() >= 0 && startC.getX() < map1.getMapWidth() && startC.getY() >= 0 && startC.getY() < map1.getMapHeight() )
CAREFUL! (typical issues so far)

2. Checking “valid” cell values.

Don’t call any method with a cell as a parameter (fillCell, isFree, …) if you are not sure that cell is “valid”.

1) if ( startC.getX() >= 0 && startC.getX() < map1.getMapWidth() && startC.getY() >= 0 && startC.getY() < map1.getMapHeight() && map1.isFree(startC) )

2) if ( map1.isFree(startC) && startC.getX() >= 0 && startC.getX() < map1.getMapWidth() && startC.getY() >= 0 && startC.getY() < map1.getMapHeight() )
PSA – 10: Turtle path planning

Important Class Map methods:

1. No return value
2. Name of class

A. public Map(String mapFileName) {...

B. public boolean isFree(Cell aCell) {...

C. public int getMapWidth() { return (this.widthInCells); }

D. public int getMapHeight() { return (this.heightInCells); }

Consider the methods above. Which is a constructor? A-D above and…

E. More than one OR none of these
PSA – 10: Turtle path planning

Important Class Map methods:

A. public Map(String mapFileName) { ...  

B. public boolean isFree(Cell aCell) { ...  

C. public int getMapWidth() { return (this.widthInCells); }  

D. public int getMapHeight() { return (this.heightInCells); }  

Consider the methods above. Which is a getter? A-D above and…  

E. More than one OR none of these
PSA – 10: Turtle path planning

Important Class Map methods:

A. `public Map(String mapFileName){ ...}

B. `public boolean isFree(Cell aCell){ ...`

C. `public int getMapWidth(){ return (this.widthInCells); }`

D. `public int getMapHeight(){ return (this.heightInCells); }`

Consider the methods above. Which is a setter? A-D above and…

E. More than one OR none of these
PSA – 10: Turtle path planning

Important Class Map methods:

A. public Map(String mapFileName) { ... 
B. public boolean isFree(Cell aCell) { ... 
C. public int getMapWidth() { return (this.widthInCells); } 
D. public int getMapHeight() { return (this.heightInCells); } 

What are the names of two of the instance variables in the Map class?
Important Class Map methods:

```java
public Map(String mapFileName) { ... }

public boolean isFree(Cell aCell) { ... }

public int getMapWidth() { return (this.widthInCells); }

public int getMapHeight() { return (this.heightInCells); }
```

Assume you have a file with a map named “map.txt”. Which line of code declares a new variable of type Map named myMap, but does not create a new Map object?

A. `Map myMap;`
B. `Map myMap();`
C. `Map myMap("map.txt");`
D. `Map myMap = new Map();`
E. `Map myMap = new Map("map.txt");`
PSA – 10: Turtle path planning

Important Class Map methods:

```java
public Map(String mapFileName) {
    ...
}

public boolean isFree(Cell aCell) {
    ...
}

public int getMapWidth() {
    return (this.widthInCells);
}

public int getMapHeight() {
    return (this.heightInCells);
}
```

Assume you have a file with a map named “map.txt”. Assuming you have already declared a variable of type Map, as in the previous slide. Which line of code creates a new Map object and puts its reference in myMap?

A. Map myMap(“map.txt”);
B. myMap(“map.txt”);
C. Map myMap = new Map(“map.txt”);
D. myMap = new Map(“map.txt”);
E. new Map(“map.txt”);
PSA – 10: Turtle path planning

Important Class Map methods:

/* Constructor that loads a map from a file (mapFileName)  **/
public Map(String mapFileName){  ...  }

/* This method returns a boolean that tells us if the given cell is
free or occupied  */
public boolean isFree(Cell aCell){  ...  }

/* This method returns the numbers of cells in the map in the x
direction (width of the map in cells)  */
public int getMapWidth(){      return (this.widthInCells);  }

/* This method returns the numbers of cells in the map in the y
direction (height of the map in cells)  */
public int getMapHeight(){      return (this.heightInCells);  }
PSA – 10: Turtle path planning

Important Class Map methods:

/* This method paints (fills) the cell in the map with coordinates (xCell, yCell), with the given color (cellColor). */
private void fillCell(int xCell, int yCell, Color cellColor) { ...}

/* This method draws the border of the cell in the map with coordinates (xCell, yCell), in the given color (squareColor) */
public void drawBorderOfCell(int xCell, int yCell, Color squareColor) {
    drawBorderOfCell( xCell, yCell, Color.BLACK);
}

True or false: the Method in blue is an example of method overloading.
A. True   B. False
Important Class Map methods:

/* This method draws the border of the cell in the map with coordinates (xCell, yCell), in the given color (squareColor) */

public void drawBorderOfCell(int xCell, int yCell, Color squareColor) {
    ...}

/* ?? */

public void drawBorderOfCell(int xCell, int yCell) {
    drawBorderOfCell(xCell, yCell, Color.BLACK);
}

What is the purpose of having the method in blue?
A. There is no point, you should remove it
B. It causes an error, you should remove it
C. It is a convenience, but not necessary
D. For a map to have black cells by default, is must be present
public void paintMap() {
    for (int i = 0; i < occupancyMap.length; ++i) {
        for (int j = 0; j < occupancyMap[i].length; ++j) {
            if (occupancyMap[i][j] == -1) {
                // This cell has not been initialized, -1
                // is the default value. It's OUT of the map.
                // Set to GRAY
                fillCell(i, j, Color.GRAY);
            } else if (occupancyMap[i][j] == 1) {
                // This cell has an obstacle. Set to BLACK.
                fillCell(i, j, Color.BLACK);
            } else {
                drawBorderOfCell(i, j); // What color is painted??
            }
        }
    }
}
Referring to the code on the previous slide:

Where is occupancyMap declared?
A. It is never declared, but in Java variables don’t have to be declared before they are used.
B. It is declared in another method in the class Map
C. It is declared in another method in another class (not Map)
D. It is declared outside of all methods but inside the class Map
E. It is declared outside of all methods, but inside another class (not Map)
Referring to the code on the previous slide:

Where does the default value -1 come from in the 2D array occupancyMap?

A. It is set to this value in the constructor of the Map class
B. -1 is the default value for 2D arrays of ints, so this happens when the array is instantiated
C. It is set to this value in the for-loop header
PSA – 10: Turtle path planning

What color will be drawn if the value of the cell in occupancyMap is 0?

A. You cannot tell from the code we’ve seen so far
B. Black border, empty inside
C. Black border, black inside
D. Green border, empty inside
E. Red border, red inside

B. Black border, empty inside
Important Class Map methods:

```java
public boolean computeCostMap(Cell startCell, Cell endCell) {
    // implementation...
}

public int getCostMapValueAt(Cell aCell) {
    return costMap[aCell.getX()][aCell.getY()];
}
```
PSA – 10: Turtle path planning

Write the code for this method (part of the Map class). It should print the neighbors of the given cell that are free

```java
public void listFreeNeighborsOf(Cell oneCell) {
    // create Cell objects to represent the neighbors
    ...
    // check if they are valid
    ...
    // check if they are free
    ...
    System.out.println( );
    ...
}
```

Useful methods of the Cell class:
- int getX(), int getY(), String toString()

Useful methods of the Map class:
- int getMapWidth()
- int getMapHeight()
- boolean isFree(Cell aCell)
public void listFreeNeighborsOf(Cell oneCell) {
    // create Cell objects to represent the neighbors

    // check if they are valid

    // check if they are free

    ... System.out.println();
    ...
}

Useful methods of the Cell class:
int getX(), int getY(), String toString()

Useful methods of the Map class:
int getMapWidth()
int getMapHeight()
boolean isFree(Cell aCell)
• **PLAY VIDEOS**

• Lab tomorrow: Related to PSA10!

• Keep studying for the final! Again review reading quiz on thursday