Design Snake Game

shashipk11

Let us see how to design a basic Snake Game that provides the following functionalities:

- Snake can move in a given direction and when it eats the food, the length of snake increases.
- When the snake crosses itself, the game will be over.
- Food will be generated at a given interval.

***Asked In**: Amazon, Microsoft, and many more interviews.*

This question is asked in interviews to Judge the Object-Oriented Design skill of a candidate. So, first of all, we should think about the classes.

The main classes will be:

1. Snake
2. Cell
3. Board
4. Game

The class Game represents the body of our program. It stores information about the snake and the board. The Cell class represents the one point of display/board. It contains the row no, column no and the information about it i.e. it is empty or there is food on it or is it a part of snake body?

Java

```
// To represent a cell of display board

public class Cell {

    private final int row, col;
    private CellType cellType;

    public Cell(int row, int col)
    {
        this.row = row;
        this.col = col;
    }

    public CellType getCellType() { return cellType; }

    public void setCellType(CellType cellType)
    {
        this.cellType = cellType;
    }

    public int getRow() { return row; }

    public int getCol() { return col; }
}
```

Java

```
// Enum for different cell types

public enum CellType {

    EMPTY,
    FOOD,
    SNAKE_NODE;
}
```

Now, the Snake class, contains the body and head. We have used Linked List to store the body because we can add a cell in O(1).

Grow method will be called when it eats the food. Other methods are self-explanatory.

Java

```
// To represent a snake
```

```java
import java.util.LinkedList;

public class Snake {

    private LinkedList<Cell> snakePartList
        = new LinkedList<>();
    private Cell head;

    public Snake(Cell initPos)
    {
        head = initPos;
        snakePartList.add(head);
        head.setCellType(CellType.SNAKE_NODE);
    }

    public void grow() { snakePartList.add(head); }

    public void move(Cell nextCell)
    {
        System.out.println("Snake is moving to "
                           + nextCell.getRow() + " "
                           + nextCell.getCol());
        Cell tail = snakePartList.removeLast();
        tail.setCellType(CellType.EMPTY);

        head = nextCell;
        head.setCellType(CellType.SNAKE_NODE);
        snakePartList.addFirst(head);
    }

    public boolean checkCrash(Cell nextCell)
    {
        System.out.println("Going to check for Crash");
        for (Cell cell : snakePartList) {
            if (cell == nextCell) {
                return true;
            }
        }

        return false;
    }

    public LinkedList<Cell> getSnakePartList()
    {
        return snakePartList;
    }

    public void
    setSnakePartList(LinkedList<Cell> snakePartList)
    {
        this.snakePartList = snakePartList;
    }

    public Cell getHead() { return head; }
```

```
    public void setHead(Cell head) { this.head = head; }
}
```

The Board class represents the display. It is a matrix of Cells. It has a method generate Food which generates the
the food at a random position.

Java

```
public class Board {

    final int ROW_COUNT, COL_COUNT;
    private Cell[][] cells;

    public Board(int rowCount, int columnCount)
    {
        ROW_COUNT = rowCount;
        COL_COUNT = columnCount;

        cells = new Cell[ROW_COUNT][COL_COUNT];
        for (int row = 0; row < ROW_COUNT; row++) {
            for (int column = 0; column < COL_COUNT;
                 column++) {
                cells[row][column] = new Cell(row, column);
            }
        }
    }

    public Cell[][] getCells() { return cells; }

    public void setCells(Cell[][] cells)
    {
        this.cells = cells;
    }

    public void generateFood()
    {
        System.out.println("Going to generate food");
        int row = 0, column = 0;
        while (true) {
            row = (int)(Math.random() * ROW_COUNT);
            column = (int)(Math.random() * COL_COUNT);
            if (cells[row][column].getCellType()
                != CellType.SNAKE_NODE)
                break;
        }
        cells[row][column].setCellType(CellType.FOOD);
        System.out.println("Food is generated at: " + row
                           + " " + column);
    }
}
```

The main class (Game) which keeps the instance of Snake and Board. It's method "update" needs to be called at a fixed interval (with the help of user input).

Java

```
// To represent Snake Game
public class Game {

    public static final int DIRECTION_NONE
        = 0,
        DIRECTION_RIGHT = 1, DIRECTION_LEFT = -1,
        DIRECTION_UP = 2, DIRECTION_DOWN = -2;
    private Snake snake;
    private Board board;
    private int direction;
    private boolean gameOver;

    public Game(Snake snake, Board board)
    {
        this.snake = snake;
        this.board = board;
    }

    public Snake getSnake() { return snake; }

    public void setSnake(Snake snake)
    {
        this.snake = snake;
    }

    public Board getBoard() { return board; }

    public void setBoard(Board board)
    {
        this.board = board;
    }

    public boolean isGameOver() { return gameOver; }

    public void setGameOver(boolean gameOver)
    {
        this.gameOver = gameOver;
    }

    public int getDirection() { return direction; }

    public void setDirection(int direction)
    {
        this.direction = direction;
    }

    // We need to update the game at regular intervals,
    // and accept user input from the Keyboard.
    public void update()
```

```
{
    System.out.println("Going to update the game");
    if (!gameOver) {
        if (direction != DIRECTION_NONE) {
            Cell nextCell
                = getNextCell(snake.getHead());

            if (snake.checkCrash(nextCell)) {
                setDirection(DIRECTION_NONE);
                gameOver = true;
            }
            else {
                snake.move(nextCell);
                if (nextCell.getCellType()
                    == CellType.FOOD) {
                    snake.grow();
                    board.generateFood();
                }
            }
        }
    }
}

private Cell getNextCell(Cell currentPosition)
{
    System.out.println("Going to find next cell");
    int row = currentPosition.getRow();
    int col = currentPosition.getCol();

    if (direction == DIRECTION_RIGHT) {
        col++;
    }
    else if (direction == DIRECTION_LEFT) {
        col--;
    }
    else if (direction == DIRECTION_UP) {
        row--;
    }
    else if (direction == DIRECTION_DOWN) {
        row++;
    }

    Cell nextCell = board.getCells()[row][col];

    return nextCell;
}

public static void main(String[] args)
{

    System.out.println("Going to start game");

    Cell initPos = new Cell(0, 0);
    Snake initSnake = new Snake(initPos);
    Board board = new Board(10, 10);
    Game newGame = new Game(initSnake, board);
```

```
        newGame.gameOver = false;
        newGame.direction = DIRECTION_RIGHT;

        // We need to update the game at regular intervals,
        // and accept user input from the Keyboard.

        // here I have just called the different methods
        // to show the functionality
        for (int i = 0; i < 5; i++) {
            if (i == 2)
                newGame.board.generateFood();
            newGame.update();
            if (i == 3)
                newGame.direction = DIRECTION_RIGHT;
            if (newGame.gameOver == true)
                break;
        }
    }
}
```

Code Explanation:

1. The code in this class represents a snake game.
2. The Snake object stores the information about the snake and the Board object stores the information about the board.
3. The direction variable keeps track of which direction the player is moving in (left, right, up, or down).
4. The isGameOver() method checks to see if there is a game over condition.
5. If there is a game over condition, then setGameOver() sets the gameOver flag to true so that it will stop playing when there is a game over.
6. The getDirection() method returns an integer value that indicates which direction the player is currently moving in (0 for left, 1 for right, 2 for up, and -2 for down).
7. The code is responsible for managing the game play of the Snake Game.
8. The class has a number of properties and methods that are relevant to game play.
9. The first property is the snake object which references the actual Snake Game character.
10. The snake object has a number of properties including direction, board and gameOver.
11. The next property is the Board object which references the playing surface on which the Snake Game takes place.
12. The Board object also has a direction property which indicates where in space the player is located relative to the playing surface.
13. The last two properties are used to keep track of whether or not the game is currently over.
14. gameOver will be set to true if the player loses, whileisGameOver will be set to false if
15. The code starts by printing out “Going to update the game.”
16. This is a message that will be displayed every time the code executes.
17. Next, the code checks to see if gameOver has been set to true.
18. If it hasn’t, then the code sets direction to DIRECTION_NONE and sets gameOver to true.

19. The next part of the code determines which cell in the snake's head is being used as a reference point.
20. The getNextCell() method uses row and col variables to determine this information.
21. Then, it returns the Cell object for use in other parts of the program.
22. The next section of code updates various aspects of the game based on user input from keyboard keys.
23. First, it checks whether any key presses have been made.
24. If so, it uses those key presses as inputs for moving or growing cells in the snake's body.
25. Finally, it updates various elements onscreen based on what was done with those cells (e.g., generating food).
26. The code updates the game at regular intervals and accepts user input from the Keyboard.
27. If the user inputs a direction other than DIRECTION_NONE, then the code sets the direction to that chosen input.
28. If the user inputs a cell that is not on the snake's path, then the code moves the snake to that cell and checks if it crashes into anything along its way.
29. If it does, then set Direction to DIRECTION_NONE.
30. Otherwise, if the cell is a food item, then the code will grow the snake and call generateFood() on board .
31. The code starts by creating a new instance of the Snake class.
32. This object will represent the player's snake in the game.
33. The initPos variable stores the location of this snake at any given time.
34. Next, a new Board object is created and initialized with the size of the playing area (10×10).
35. This board will be used to track where the snake has moved and what obstacles it has encountered along its way.
36. A new Game object is then created, which contains information about the game itself as well as our snake instance.
37. The gameOver property is set to false so that we can keep track of whether or not there is currently a game being played.
38. The direction property is set to DIRECTION_RIGHT so that users can control their snake's movement using their keyboard input.
39. The code then periodically updates both the Game object and Board objects based on user input received from the Keyboard class.
40. Whenever a keystroke is detected, an event handler for that particular key is called.
41. In this case, we simply print out "Press left arrow to move left" onscreen whenever Left Arrow keystrokes are detected by our code.
42. The code creates a new instance of the Snake game, initializes it to the given position (0, 0), and creates a new Board object.
43. The code then creates a new Game object and sets its properties to match those of the Snake object.
44. Next, the code declares two variables: initSnake and board.

45. initSnake is an instance of the Snake class, while board is an instance of the Board class.
46. The next line of code sets up a timer that will update the game at regular intervals.
47. This will allow us to react to user input from the keyboard.
48. Finally, the code declares two variables: gameOver and direction.
49. gameOver is set to false so that the game can continue even if it reaches its end condition.

Reference:
http://massivetechinterview.blogspot.com/2015/10/snake-game-design.html

Last Updated : 24 Apr, 2023

Similar Reads

1. Design a Chess Game
2. Design a Logistics System
3. Design an online book reader system
4. Design a Hit Counter
5. Types of Models in Object Oriented Modeling and Design
6. Design Goals and Principles of Object Oriented Programming
7. Design GeeksforGeeks T-Shirt Challenge
8. Software Design Patterns
9. Last Minute Notes - Compiler Design
10. Ant Design

Previous Next

Article Contributed By :

shashipk11

shashipk11

Vote for difficulty

Current difficulty : Medium

Easy | Normal | Medium | Hard | Expert

Improved By : shubham_singh, Akanksha_Rai, secret_giggle, ankit_kumar_, satyaprasanna

Article Tags : Object-Oriented-Design, Technical Scripter 2018, Technical Scripter

Report Issue

GeeksforGeeks
A-143, 9th Floor, Sovereign Corporate Tower, Sector-136, Noida, Uttar Pradesh - 201305

feedback@geeksforgeeks.org

C++

PHP

GoLang

SQL

R Language

Android Tutorial

Linked List

Stack

Queue

Tree

Graph

Algorithms

Sorting

Searching

Greedy

Dynamic Programming

Pattern Searching

Recursion

Backtracking

Web Development

HTML

CSS

JavaScript

Bootstrap

ReactJS

AngularJS

NodeJS

Computer Science

GATE CS Notes

Operating Systems

Computer Network

Database Management System

Software Engineering

Digital Logic Design

Engineering Maths

Python

Python Programming Examples

Django Tutorial

Python Projects

Python Tkinter

OpenCV Python Tutorial

Python Interview Question

Data Science & ML

Data Science With Python

Data Science For Beginner

Machine Learning Tutorial

Maths For Machine Learning

Pandas Tutorial

NumPy Tutorial

NLP Tutorial

Deep Learning Tutorial

DevOps

Git

AWS

Docker

Kubernetes

Azure

GCP

Competitive Programming

Top DSA for CP

System Design

What is System Design

Top 50 Tree Problems

Top 50 Graph Problems

Top 50 Array Problems

Top 50 String Problems

Top 50 DP Problems

Top 15 Websites for CP

Interview Corner

Company Wise Preparation

Preparation for SDE

Experienced Interviews

Internship Interviews

Competitive Programming

Aptitude Preparation

Commerce

Accountancy

Business Studies

Economics

Management

Income Tax

Finance

SSC/ BANKING

SSC CGL Syllabus

SBI PO Syllabus

SBI Clerk Syllabus

IBPS PO Syllabus

IBPS Clerk Syllabus

Aptitude Questions

SSC CGL Practice Papers

Monolithic and Distributed SD

Scalability in SD

Databases in SD

High Level Design or HLD

Low Level Design or LLD

Top SD Interview Questions

GfG School

CBSE Notes for Class 8

CBSE Notes for Class 9

CBSE Notes for Class 10

CBSE Notes for Class 11

CBSE Notes for Class 12

English Grammar

UPSC

Polity Notes

Geography Notes

History Notes

Science and Technology Notes

Economics Notes

Important Topics in Ethics

UPSC Previous Year Papers

Write & Earn

Write an Article

Improve an Article

Pick Topics to Write

Write Interview Experience

Internships

Video Internship