Name: - Udbhavsinh Solanki

Id: - 0873798

Explanation of the game and code: -

I have made the Game of life utilizing kotlin language. Our Game of life is a zero-player game, implying that its advancement is dictated by its underlying state, requiring no additionally input. One connects with the Game of Life by making an underlying design and seeing how it develops. A cell C is spoken to by a 1 when alive, or 0 when dead, in a m-by-m (or m×m) square exhibit of cells. We ascertain N - the entirety of live cells in C's eight-area neighborhood, at that point cell C is alive or dead in the cutting edge on some given principles: - Any live cell with less than two live neighbors kicks the bucket, as though brought about by under-populace. Any live cell with a few carry on with neighbors' lives on to the people to come. Any live cell with in excess of three live neighbors passes on, as though by over-populace. Any dead cell with precisely three live neighbors turns into a live cell, as though by multiplication. I have chiefly utilized courotines and useful programming. I have made a few kotlin documents. And stored that code files using .kt extension.

Let us take a tour on the coding sites: -

```
package Amey_kathiwadi

val empty get() = Maze7(3, 1)

val blinker
  get() = """
      ...........
      ...x.......
      ...x.......
      ...x.......
      ...........
  """.toMaze()

val beacon
  get() = """
      ...........
      ...xx......
      ...xx......
      .....xx....
      .....xx....
      ...........
  """.toMaze()

val glider
  get() = """
      ...x..............
      ....x.............
      ..xxx.............
      ..................
      ..................
      ..................
      ..................
      ..................
      ..................
  """.toMaze()


fun main() {
  var m = """
      ...x.....
      ....x....
      ..xxx....
      .........
      .........
  """.toMaze()

  repeat(5) {
    println()
    println(m.renderToString())
    m = m.nextGeneration(EvolutionCell::conwayLaws)
  }
}
```

```
package Amey_kathiwadi

enum class CellState {
  DEAD,
  ALIVE
}

interface EvolutionCell {
  val neighbours: Int
  val state: CellState
}

interface MazeWorld {
  fun nextGeneration(evolutionRule: EvolutionCell.() -> CellState): MazeWorld
}
```

```
package Amey_kathiwadi


private val emptyCell = listOf("&")
private val aliveCells = listOf("*", "?", "+", "#", "@")

fun Maze7.renderToString() = buildString {
  for (y in 0 until height) {
    for (x in 0 until width) {
      append(
        when (get(x, y)) {
          CellState.ALIVE -> aliveCells
          CellState.DEAD -> emptyCell
        }.random()
      )
    }
    append("\n")
  }
}
```

	python	php
Popularity	Very popular	Very popular
Frameworks	A lot of frameworks	A few frameworks
Learning	Easy to learn	Harder to learn

```
package Amey_kathiwadi

import Amey_kathiwadi.CellState.ALIVE
import Amey_kathiwadi.CellState.DEAD


fun EvolutionCell.conwayLaws() = when (state) {
  ALIVE -> when (neighbours) {
    2, 3 -> ALIVE  // living on
    else -> DEAD   // underpopulation or overpopulation
  }

  DEAD -> when (neighbours) {
    3 -> ALIVE     // reproduction
    else -> DEAD
  }
}
```

```
package Amey_kathıwadı

import kotlin.math.min


class Maze7(val width: Int, val height: Int) : MazeWorld {
  private val state = Array(height) { Array(width) { CellState.DEAD } }

  operator fun get(x: Int, y: Int) = state[y][x]
  operator fun set(x: Int, y: Int, value: CellState) {
    state[y][x] = value
  }

  override fun nextGeneration(evolutionRule: EvolutionCell.() -> CellState): Maze7 {
    val copy = Maze7(width, height)
    for (y in 0 until height) {
      for (x in 0 until width) {

        val cell = object : EvolutionCell {
          override val neighbours by lazy {
            countAliveNeighbors(x, y)
          }

          override val state by lazy { get(x, y) }
        }

        copy[x, y] = cell.evolutionRule()
      }
    }
    return copy
  }
}
```

```
package Amey_kathiwadi

import kotlin.random.Random

fun randomMaze(width: Int, height: Int, p: Double = 0.3) = Maze7(width, height).apply {
  for (y in 0 until height) {
    for (x in 0 until width) {
      if (Random.nextDouble() <= p) {
        this[x, y] = CellState.ALIVE
      }
    }
  }
}

|
```