Carlos Soliz

Documentación Actividad n°14

Año: 2024
Profesores: York Elias MANSILLA MUÑOZ y Jorge Fabián SILES GUZMÁN

Materia: Proyecto de Implementación de Sitios Web Dinámicos
Grupo: Carlos Soliz

Carlos Soliz

Documentación del Código: Laberinto en Processing

Variables Globales

- **cols, rows**: Número de columnas y filas en el laberinto.
- **w**: Ancho de cada celda del laberinto.
- **grid**: Matriz de booleanos que representa el laberinto. `true` representa una pared y `false` un camino.
- **turtle**: Posición de la tortuga, representada por un objeto `PVector`.
- **food**: Posición de la comida, también representada por un objeto `PVector`.
- **gameOver**: Variable booleana que indica si el juego ha terminado.

Funciones Principales

`setup()`

- Inicializa la ventana con dimensiones 400x400.
- Calcula las dimensiones del laberinto (número de columnas y filas) basadas en el ancho de las celdas.
- Llama a las funciones `initializeMaze()`, `spawnTurtle()` y `spawnFood()` para crear el laberinto, colocar la tortuga y generar la comida respectivamente.
- Establece la tasa de fotogramas a 10 FPS.

`draw()`

- Dibuja el fondo de la ventana y controla la lógica principal del juego.
- Si el juego no ha terminado (`!gameOver`), verifica colisiones, dibuja el laberinto, la tortuga y la comida.
- Si el juego ha terminado, muestra un mensaje de "Juego Terminado".

Funciones de Inicialización

`initializeMaze()`

- Inicializa el laberinto utilizando una matriz estática predefinida para que el diseño del laberinto sea siempre el mismo.
- La matriz `staticMaze` contiene `true` para paredes y `false` para caminos. Asigna esta matriz al grid del juego.

java

- `boolean[][] staticMaze = {`
- `  {true, true, true, true, true, true, true, true, true, true},`
- `  {true, false, false, false, true, false, false, true, false, true},`

```
  {true, false, true, false, true, false, true, true, false, true},
  {true, false, true, false, false, false, false, false, false, true},
  {true, false, true, true, true, false, true, true, true, true},
  {true, false, false, false, true, false, false, false, false, true},
  {true, true, true, false, true, true, true, false, true, true},
  {true, false, false, false, false, false, true, false, false, true},
  {true, false, true, true, true, false, true, true, false, true},
  {true, true, true, true, true, true, true, true, true, true}
};
```

Funciones para la Tortuga

`spawnTurtle()`

- Coloca la tortuga en una celda aleatoria que no sea una pared. Utiliza un bucle `do-while` para evitar que la tortuga aparezca en una celda ocupada por una pared.

java

```
void spawnTurtle() {
  int i, j;
  do {
    i = floor(random(cols));
    j = floor(random(rows));
  } while (grid[i][j]); // Repite hasta que se encuentre una celda vacía
  turtle = new PVector(i * w + w / 2, j * w + w / 2); // Posición centrada en la celda
}
```

`drawTurtle()`

- Dibuja la tortuga en la pantalla. Utiliza un círculo rojo que se posiciona según las coordenadas actuales de la tortuga.

java

```
void drawTurtle() {
  fill(255, 0, 0); // Color rojo
  noStroke();
  ellipse(turtle.x, turtle.y, w, w); // Dibuja un círculo que representa la tortuga
}
```

`keyPressed()`

- Controla el movimiento de la tortuga con las teclas de dirección (arriba, abajo, izquierda, derecha). Cada vez que se presiona una tecla, la función `moveTurtle()` se llama con los desplazamientos correspondientes.

`moveTurtle(float dx, float dy)`

- Mueve la tortuga según el desplazamiento dado (`dx` y `dy`).
- Verifica si la nueva posición es válida (que no sea un muro) antes de actualizar la posición de la tortuga.

java

```
void moveTurtle(float dx, float dy) {
  float newX = turtle.x + dx;
  float newY = turtle.y + dy;

  int i = floor(newX / w);
  int j = floor(newY / w);
  if (i >= 0 && i < cols && j >= 0 && j < rows && !grid[i][j]) {
    turtle.x = newX;
    turtle.y = newY;
  }
}
```

Funciones para la Comida

`spawnFood()`

- Genera la comida en una celda aleatoria que no sea una pared. También utiliza un bucle `do-while` para asegurarse de que la comida no caiga en una pared.

java

```
void spawnFood() {
  int i, j;
  do {
    i = floor(random(cols));
    j = floor(random(rows));
  } while (grid[i][j]); // Repite hasta que encuentre una celda vacía
  food = new PVector(i * w, j * w); // Posiciona la comida en una celda vacía
}
```

drawFood()

- Dibuja la comida en la pantalla. Se representa con un círculo verde en el centro de la celda asignada.

java

```
void drawFood() {
  fill(0, 255, 0); // Color verde
  noStroke();
  ellipse(food.x + w / 2, food.y + w / 2, w / 2, w / 2); // Dibuja un círculo para la comida
}
```

Funciones de Verificación

checkCollision()

- Verifica si la tortuga ha chocado con una pared. Si la tortuga toca una pared, el juego termina.
- También verifica si la tortuga ha encontrado la comida. Si la tortuga toca la comida, el juego también termina y se muestra un mensaje.

java

```
void checkCollision() {
  int i = floor(turtle.x / w);
  int j = floor(turtle.y / w);
  if (grid[i][j]) {
```

```
    gameOver = true; // Termina el juego si la tortuga toca una pared
  }

  // Verifica si la tortuga ha encontrado la comida
  if (dist(turtle.x, turtle.y, food.x + w / 2, food.y + w / 2) < w / 2) {
    gameOver = true; // Termina el juego si la tortuga encuentra la comida
    println("¡Comida encontrada!");
  }
}
```