MANUAL DE SISTEMA

Le secret de l'estanque

ALGORITMIA Y PROGRAMACIÓN II

2024

Inicializando y declarando

```
// Variables principales
int menu = 0;
boolean estado = false;
boolean gano = false;
```

Estas variables son necesarias para cada parte del código que a continuación veremos con detalle.

```
// Matriz
int[][] matriz;
int M = 10;
int N = 10;
float t; // Tamaño x cuadr
```

Estas nos servirán para la creación y visualización de la matriz (en donde se encontrara el juego).

```
// CONDICIONES
int tapcontador;
boolean limiteAlcanzado =
int tiempoLimite = 30;
int tiempoRestante;
int tiempoInicio;

int vidas = 3;
```

Estas variables seran especificamente para los contadores de tiempo, el contador de movimiento y el de vida.

```
// BOTONES
int nivel = 1;
float Mx, Nx, MNy;
float tx1, tx2, ty;
int contador = 10;
float nx, ny;
float h, sep;
float bx, by, btx, bty;
float vx, vy;
```

Inicializamos y algunas tambien declaramos las variables de cada boton, el contador y el nivel.

```
// IMAGENES
PImage fondo;
PImage imagen;
PImage flor;
PImage flor2;
PImage victoria;
PImage derrota;
```

Declaramos las variables de cada parte del juego.

```
float prop_x;
float prop_y;
```

Inicializamos las variables que nos daran las proporciones adecuadas.

Inicializamos las variables que nos daran las imagenes que necesitamos.

```
void setup() {
  fullScreen();
  fondo = loadImage("Iterativo.png");
  fondo.resize(width, height);
  flor = loadImage("loto1.png");
  flor2 = loadImage("loto3.png");
  derrota = loadImage("perdio.png");
```

Inicializamos las proporciones y apartir de estas, tambien inicializamos los botones que necesitaremos mas adelante, de manera que nos aseguremos que cada ususario cuente con todo el juego en su lugar.

```
  prop_x = width / 2160.0;
  prop_y = height / 1440.0;

  t = 65 * min(prop_x, prop_y);

  Mx = 775 * prop_x;
  Nx = 1100 * prop_x;
  MNy = 1085 * prop_y;

  tx1 = 1258 * prop_x;
  tx2 = 793 * prop_x;
  ty = 963 * prop_y;

  nx = 755 * prop_x;
  ny = 400 * prop_y;
  h = 95 * prop_y;
  sep = 100 * prop_y;


 bx = 833 * prop_x;
 by = 1228 * prop_y;
 btx = 495 * prop_x;
 bty = 130 * prop_y;

 vx = 105 * prop_x;
 vy = 25 * prop_y;
}
```

Dibujando

```
void draw() {
  switch (menu) {
    case 0:
      principal();
      break;
    case 1:
      level1();
      if (gano) {
        victoria = loadImage("gano.png");
        image(victoria, 0, 0, width, height);
      }
      if (vidas == 0 || tiempoRestante == 0 || tapcontador == 0) {
        image(derrota, 0, 0, width, height);
      }
      break;
    case 2:
      level2();
      if (gano) {
        victoria = loadImage("1.png");
        image(victoria, 0, 0, width, height);
      }
      if (vidas == 0 || tiempoRestante == 0 || tapcontador == 0) {
        image(derrota, 0, 0, width, height);
      }
      break;
    case 3:
      level3();
      if (gano) {
        victoria = loadImage("1.png");
        image(victoria, 0, 0, width, height);
      }
      if (vidas == 0 || tiempoRestante == 0 || tapcontador == 0) {
        image(derrota, 0, 0, width, height);
      }
      break;
    default:
      background(0);
      fill(255);
      textSize(100 * min(prop_x, prop_y));
      textAlign(CENTER, CENTER);
      text("GAME OVER", width / 2, height / 2);
  }
```

En cada nivel definimos cuando el jugador gana o pierde. Para que asi aparezca su imagen correspondiente.

Métodos importantes

```
void principal() {
  image(fondo, 0, 0, width, height);
  fill(0, 0, 0);
  textSize(48);
  textAlign(LEFT, UP);
  text(str(contador), 1045 * prop_x, 985 * prop_y, 190, 78);
}
```

Organizamos lo primero que vera el usuario al entrar a nuestro juego.

Tenemos un metodo para cada nivel en donde colocamos el fondo de el nivel correpondiente y llamamos a los metodos adecuados para que este funcione.

```
void level1() {
  imagen = loadImage("Nivel_1.png");
  image(imagen, 0, 0, width, height);
  if (estado) {
    mostrarMatriz();
    verificarTiempo();
    mostrarInfo();
  }
}
```

```
void iniciarJuego(int f) {
  matriz = new int[M][N];
  for (int i = 0; i < M; i++) {
    for (int j = 0; j < N; j++) {
      matriz[i][j] = (int)random(f);
    }
  }
  tapcontador = (N < 15 && M < 15) ? 20 : 15;
  tiempoRestante = tiempoLimite;
  tiempoInicio = millis();
  vidas = 3;
  gano = false;
  estado = true;
}
```

Al iniciar se crea aletoriamente la matriz para garantizar el funcionamiento del juego.

Con nuestra matriz mxn si en dado caso m o n es mayor que 15 la cantidad de movimiento eds 20, si es menor se limita a 15. Por otra parte, inicializamos el tiempo, las vidas y si gano.

```
void mostrarMatrizRecursivo(int i, int j, float offsetX, float offsetY) {
  if (i >= M) return;

  float x = offsetX + (i * t);
  float y = offsetY + (j * t);

  if (matriz[i][j] == 1) {
    fill(90, 166, 15);
    rect(x, y, t, t);
    float florSize = t * 0.9;
    image(flor, x + (t - florSize) / 2, y + (t - florSize) / 2, florSize, florSize);
  } else if (matriz[i][j] == 2) {
    fill(90, 166, 15);
    rect(x, y, t, t);
    float florSize = t * 0.9;
    image(flor2, x + (t - florSize) / 2, y + (t - florSize) / 2, florSize, florSize);
  } else {
    fill(225, 240, 201);
    rect(x, y, t, t);
  }
```

Nos encargamos de mostrar graficamente cada parte de la matriz en este caso aqui se coloca cada flor en las casillas seleccionadas aleatoriamente.

```
void mostrarMatriz() {
  float offsetX = width * 0.25;
  float offsetY = height * 0.2;
  mostrarMatrizRecursivo(0, 0, offsetX, offsetY);
}
```

se encarga de dibujar cada celda de la matriz en la pantalla. Utiliza recursión para recorrer todos los elementos de la matriz.

```
void verificarVictoria() {
  gano = true;
  for (int i = 0; i < M; i++) {
    for (int j = 0; j < N; j++) {
      if (matriz[i][j] > 0) {
        gano = false;
        return;
      }
    }
  }
}
```

En este metodo determinar el estado del juego y proporcionar retroalimentación al jugador sobre si ha ganado o no.

Este metodo crucial para implementar un sistema de gestión del tiempo en tu juego o aplicación. Permite controlar cuándo debe terminar la partida debido al agotamiento del tiempo

```
void verificarTiempo() {
  tiempoRestante = tiempoLimite - (millis() - tiempoInicio) / 1000;
  if (tiempoRestante <= 0) {
    estado = false;
    tiempoRestante = 0;
  }
}
```

```
void mostrarInfo() {
  fill(255);
  textSize(45);
  textAlign(RIGHT);
  text(tiempoRestante, 2050 * prop_x, 100 * prop_y);
  textAlign(LEFT);
  text("Vidas: " + vidas, 650 * prop_x, 100 * prop_y);
  textAlign(CENTER);
  text(tapcontador, 1695*prop_x, 100 * prop_y);
}
```

En este metodo se muestra toda la informacion importante en cada nivel como lo es el tiempo, los movimientos y las vidas que le resta al ususario

```
void explotarBurbujaRecursivo(int x, int y, int valorInicial) {
  if (x < 0 || x >= M || y < 0 || y >= N || matriz[x][y] != valorInicial) return;

  matriz[x][y]--;

  explotarBurbujaRecursivo(x - 1, y, valorInicial); // Arriba
  explotarBurbujaRecursivo(x + 1, y, valorInicial); // Abajo
  explotarBurbujaRecursivo(x, y - 1, valorInicial); // Izquierda
  explotarBurbujaRecursivo(x, y + 1, valorInicial); // Derecha
}
```

Para finalizar, este metodo eliminar elementos adyacentes en una matriz que comparten un valor específico, comenzando desde una posición inicial (x, y). Utiliza recursión para explorar y decrementar el valor de las celdas adyacentes (arriba, abajo, izquierda y derecha) que coinciden con valorInicial. La función termina su ejecución si se encuentra fuera de los límites de la matriz o si la celda actual no tiene el mismo valor que el inicial.

MousePressed

Este controla eventos que se ejecuta cuando el usuario hace clic con el mouse. Su propósito principal es gestionar la interacción del usuario con el menú del juego y la lógica del juego en sí. A continuación, desglosaremos su funcionamiento en detalle.

```
case 0:
  if ((mouseX > bx && mouseX < (bx + btx)) &&
      (mouseY > by && mouseY < (by + bty))) {
    menu = nivel;
    int f = 2;
    if (nivel == 3) {
      f = 3;
    }
    iniciarJuego(f);
    println("Iniciando programa...");
  }
```

Iniciar juego si se hace clic en el botón correspondiente

Cambia al nivel que seleccion el usuario e inicia el juego.

```
    (mouseY > ty && mouseY < (ty + 100 * prop_y)) && contador < 20) {
  contador++;
  println("Contador: " + contador);
}
if ((mouseX > tx2 && mouseX < (tx2 + 110 * prop_x)) &&
    (mouseY > ty && mouseY < (ty + 100 * prop_y)) && contador > 10) {
  contador--;
  println("Contador: " + contador);
```

Si se preciona al boton se incrementa la cantidad del contador.

Si se preciona al boton se disminuye la cantidad del contador.

```
if ((mouseX > Mx && mouseX < (Mx + 280 * prop_x)) &&
    (mouseY > MNy && mouseY < (MNy + 80 * prop_y))) {
  M = contador;
  println("M: " + M);
}
if ((mouseX > Nx && mouseX < (Nx + 280 * prop_x)) &&
    (mouseY > MNy && mouseY < (MNy + 80 * prop_y))) {
  N = contador;
  println("N: " + N);
}
```

Si se precionan lod botones correspondientes se capturan m y n en base a el contador.

En esta parte se maneja la seleccion de los niveles dependiendo de el boton que escoja.

```
for (int i = 0; i < 3; i++) {
  if (mouseX > nx && mouseX < (nx + 650 * prop_x) &&
      mouseY > (ny + i * sep) && mouseY < (ny + i * sep + h)) {
    nivel = i + 1;
    println("Nivel " + nivel + "...");
    break;
  }
}
```

```
if ((mouseX > vx && mouseX < (vx + 300 * prop_x)) &&
    (mouseY > vy && mouseY < (vy + 100 * prop_y))) {
  menu = 0;
  println("Regresando al menú principal...");
}
```

Este es el boton de volver al menu principal.

```
if ((mouseX > vx && mouseX < (vx + 300 * prop_x)) &&
    (mouseY > vy && mouseY < (vy + 100 * prop_y))) {
  menu = 0;
  println("Regresando al menú principal...");
}
if (estado) {
  float offsetX = width * 0.25;
  float offsetY = height * 0.2;

  int x = int((mouseX - offsetX) / t);
  int y = int((mouseY - offsetY) / t);

  if (x >= 0 && x < M && y >= 0 && y < N && matriz[x][y] > 0) {
    int valorInicial = matriz[x][y];
    explotarBurbujaRecursivo(x, y, valorInicial);
    tapcontador--;
    verificarVictoria();
  } else {
    if (gano || vidas <= 0) {
      estado = false;
    } else {
      vidas--;
    }
  }
}
```

En esta parte del código, se verifica si el juego está activo (estado es verdadero) y se calculan los desplazamientos para centrar la matriz en la pantalla. Luego, se determinan los índices x y de la celda seleccionada por el usuario. Si la celda está dentro de los límites y contiene un valor positivo, se llama aexplotarBurbujaRecursivo para eliminar burbujas adyacentes, se decrementa el contador de toques (tapcontador), y se verifica si el jugador ha ganado. Si la selección es inválida y el jugador no ha ganado ni ha perdido todas sus vidas, se decrementa el número de vidas restantes; si ha ganado o se han agotado las vidas, se establece estado` como falso, indicando que el juego ha terminado.

Contacto

Si tienes preguntas sobre el funcionamiento del software o requieres asistencia técnica, puedes ponerte en contacto con las creadoras del sistema o el equipo de soporte de Almacenes de La Prosperidad a través de los siguientes medios:

Creadoras del Software

Este software fue desarrollado por:

KATHERIN BARRERA, VALERIA FLOREZ Y DAVID CASALLAS.

Correo Electrónico de las Creadoras:

Para consultas técnicas o información sobre el desarrollo, puedes escribir a:

- lkatherin@uninorte.edu.co
- florezvaleria@uninorte.edu.co
- decasallas@uninorte.edu.co