



Universidad Veracruzana
FACULTAD DE INGENIERÍA ELÉCTRICA Y ELECTRÓNICA

É P H I M E R O

Proyecto Final GC
Lara Xocuis Martha Denisse
S22002213

December 08, 2024

Contents

1 INTRODUCCIÓN	1
1.1 Objetivo	1
1.2 Detalles Importantes	2
1.2.1 TOMOE: Personaje Principal	2
1.2.2 Historia	3
2 REQUISITOS	4
2.1 Funcionales	4
2.2 No funcionales	4
3 DISEÑO	5
3.1 Arquitectura	5
3.1.1 Godot 4.3	5
3.1.2 C y freeglut	8
3.2 Minijuegos	9
3.2.1 Minijuego 1: Réplica Puzzle de Undertale	9
3.2.2 Minijuego 2: Ordenar Signos Matemáticos	10
3.2.3 Minijuego 3: Preguntas Capsiosas	11
3.3 Diagramas	12
3.4 Detalles Visuales y Sonoros	12
4 Implementación	13
4.1 Réplica Puzzle de Undertale	13
4.2 Ordenar signos matemáticos	18
5 Resultados	24
6 Manual de usuario	24
6.1 ¿CÓMO JUGAR?	24
7 Conclusiones	24
8 Referencias	25

1 INTRODUCCIÓN

'ÉPHIMERO' es un videojuego en 2D que explora los conceptos de **aceptación, incertidumbre y lucha interna entre el protagonista y lo incontrolable**. El proyecto utiliza las habilidades aprendidas en la experiencia educativa "graficación por computadora" haciendo uso de **C y OpenGL**, complementados con las herramientas del motor **Godot 4.3**. También se integra el uso de estructuras de datos como *pilas, colas y árboles* para desarrollar la lógica de minijuegos y la interacción con el jugador. Dichos minijuegos se enriquecieron con transformaciones geométricas de freeglut y texturizado de objetos en 2D con el uso de **stb_image.h**

En el motor de godot se implementaron animaciones manuales y originales en **Aseprite**, así como también el uso de recursos libres para fondos o elementos del entorno.

Otra cosa que tiene Éphimero es su música; con una selección de temas como "Burning Hill" de Mitski, "The Color of Depression" de Nier Automata o "Waterfall" de Undertale y, de igual forma, **temas originales creados específicamente para este proyecto**.

1.1 Objetivo

El objetivo del juego es sumergir al jugador en una **travesía filosófica** donde el protagonista debe **confrontar su propia lucha interna**: la muerte de su mejor amigo en la guerra.

Mientras el protagonista explora un mundo misterioso se enfrenta a desafíos que lo invitan a reflexionar sobre su propia vida y el significado de aceptar lo incontrolable. En paralelo, el diseño del juego busca mostrar la aplicabilidad de los conceptos de programación y diseño de videojuegos en un entorno interactivo, contribuyendo al aprendizaje y dominio de herramientas como OpenGL, C y Godot.

1.2 Detalles Importantes

1.2.1 TOMOE: Personaje Principal

Tomoe Miyahara es el personaje principal de 'éphimero', inspirado por la figura histórica del soldado **Shoichi Yokoi**, quien fue un militar del ejército imperial japonés durante la Segunda Guerra Mundial.



Figure 1: El soldado japonés Shoichi Yokoi, en 1941 y 1972

Tomoe representa la resiliencia y la determinación, recordando al jugador la importancia de no rendirse, sin importar las adversidades. El nombre "Tomoe Miyahara" fue elegido por su sonoridad, evocando una sensación de fuerza y delicadeza al mismo tiempo. Al diseñar este personaje, se buscó no solo hacer honor a una figura histórica, sino también transmitir un mensaje poderoso y emotivo sobre la perseverancia en tiempos de crisis.



Figure 2: Diseño de Tomoe en C con freeglut

1.2.2 Historia

Es el año 1941 y estás en medio de la **Segunda Guerra Mundial**. Tú, Tomoe Miyahara, un soldado que lucha con valentía junto a su mejor amigo, quien es casi como un hermano para ti, te cubre la espalda dentro del campo de batalla. Sin embargo, un ataque inesperado de las fuerzas estadounidenses le arrebata la vida a tu amigo frente a tus ojos. El dolor y el shock te dejan totalmente paralizado, y, al no poder soportar más el sentimiento, decides dejar el campo de batalla y huir.

Después de un largo camino, te pierdes en un bosque. El vacío de tu corazón es indescriptible pero, es allí, en la soledad de la naturaleza, donde encuentras una cueva extraña. Dentro de esa cueva algo sumamente inusual te llama la atención: una máquina expendedora de refrescos. Impulsado por la curiosidad decides prenderla y usarla, pero caes al suelo por un desmayo repentino.

Al despertar sales de la cueva, el mundo ha cambiado, el bosque es más brillante y el ambiente es más vivo. Las personas a tu alrededor no parecen las mismas y algo se siente diferente. **La guerra había terminado.**

Decides regresar a lo que creías que era tu hogar, pero ya no es tuya, una familia desconocida ha tomado tu lugar. Totalmente confundido te vuelves a perder hasta llegar a un parque, donde te topas a una persona que te deja sin aliento: un hombre con un rostro idéntico a tu amigo fallecido, y a su lado una mujer que, al verte, no puede evitar su asombro. Ella te reconoce, pues tu apariencia es igual a la de su hijo, quien murió en la guerra.

Caes en un impacto emocional brutal, ¿habías muerto en esa guerra? ¿por qué tu amigo sigue vivo?, y te das cuenta: **habías viajado en el tiempo**, en un mundo paralelo donde lo que ocurrió no fue lo que viviste con anterioridad, en este mundo, tú fuiste quien murió y tu amigo siguió adelante con su vida, como tú hubieras deseado.

La familia de tu amigo te recibe como un extraño y te encuentras viviendo una vida que no es la tuya, pero por momentos parece tan real. Pasas un día entero con ellos, compartes risas y conversaciones, pero tu corazón sabe que no perteneces ahí. La tristeza te vuelve a consumir, pero sacas fuerza y decides regresar a la cueva para volver a tu realidad original.

Vuelves a 1941, al caos. Las lágrimas nublan tu vista pero hay algo en ti que es diferente, sigues luchando. Sientes la necesidad de continuar desde adentro, no solo por ti, sino por él: tu amigo, quien hubiera querido que siguieras adelante. Sales corriendo a la batalla con la esperanza de sobrevivir y honrar su memoria.

Porque, al final, lo que importa es salir adelante, aunque el destino sea incierto y aunque la guerra sea imparable.

2 REQUISITOS

2.1 Funcionales

- El jugador podrá moverse en el mundo para explorar e interactuar con objetos para seguir con la historia y tener coherencia en la narrativa
- El jugador debe completar minijuegos para pasar a otros niveles y seguir con la historia.

2.2 No funcionales

- El juego es portable y funcional con Linux Windows
- El diseño es apto para funcionar en máquinas que cumplen los recursos mínimos
- Minijuegos de C ejecutados desde Godot

3 DISEÑO

3.1 Arquitectura

3.1.1 Godot 4.3

Se optó por utilizar el motor de godot por su facilidad de manejo de nodos, implementación de animaciones.

- **TOMOE:** escena del personaje principal donde se incluyen las animaciones (con Aseprite)



Figure 3: Creando animaciones en Aseprite



Figure 4: Creando animaciones en Aseprite

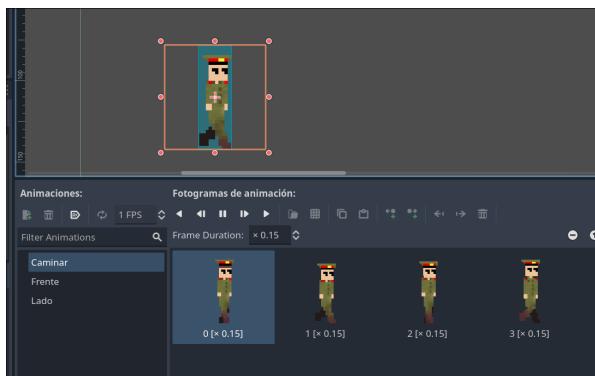


Figure 5: Creando animaciones en Aseprite

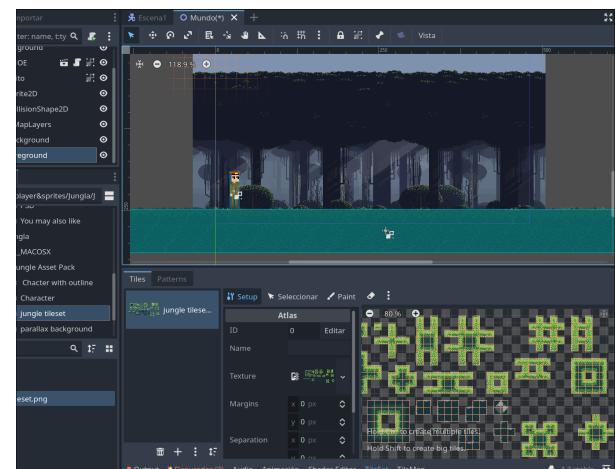


Figure 6: Poniendo nodo en Godot

- **MUNDO1:** escena de inicio donde el personaje se encuentra en el bosque

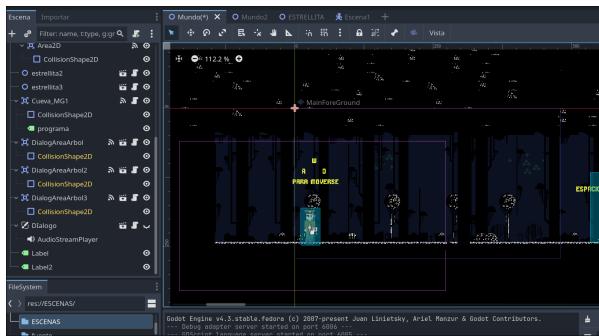


Figure 7: Creando tilemap en Godot

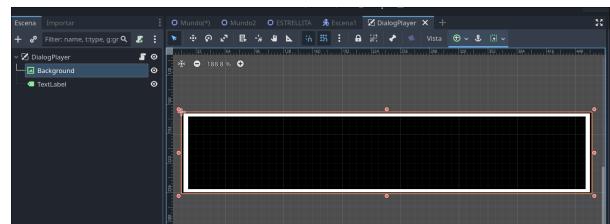


Figure 8: Nodo canvas para el dialogo

EL DIÁLOGO SE MANEJA POR UN .json y ese se carga a un script

El diálogo se maneja por **señales** y varios scripts para ello, si el jugador entra al área del collisionshape2d y hay una entrada por teclado, la señal hace que funcionen los algoritmos de los scripts e imprima los diálogos de acuerdo al .json.

```

1  {
2    "arbol": [
3      "Era frio",
4      "Y no pude pensar en nada más"
5    ],
6    "arbol2": [
7      "Fue como un pantallazo",
8      "Uzumaki se fue frente mío"
9    ],
10   "arbol3": [
11     "...",
12     "Creo que no querré volver"
13   ],
14   "CUEVA1": [
15     "Llegué aquí sin pensar",
16     "La guerra no ha terminado...",
17     "Tal vez necesito un refugio"
18   ],
19   "CUEVA2": [
20     "No creo que sea por aquí"
21   ]
22 }
```

Figure 9: .json para los dialogos



Figure 10: CollisionShape para interactuar

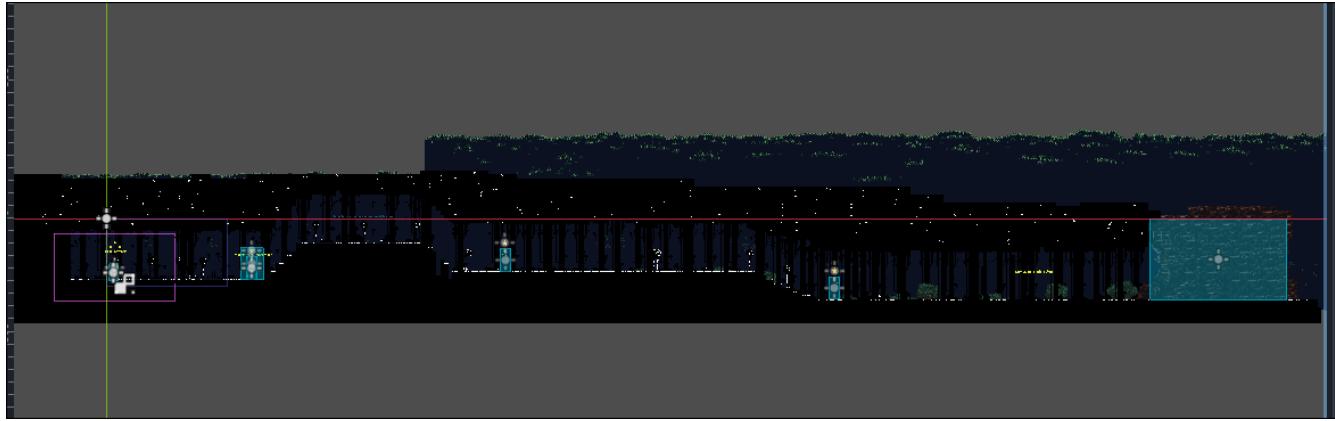


Figure 11: Todo el tilemap creado

Se utilizaron assets libres por derechos de autor :)

- **MUNDO2:** escena de la cueva

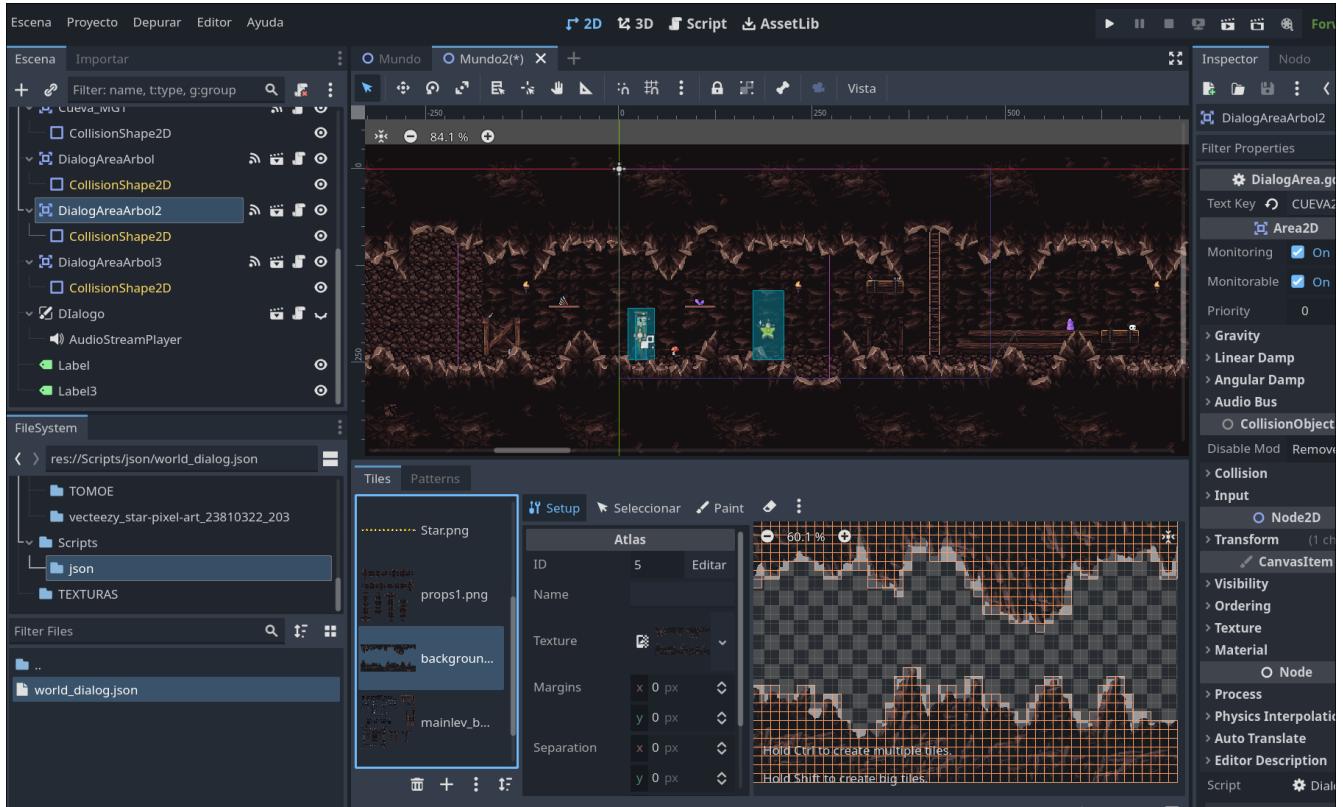


Figure 12: Todo el tilemap creado

3.1.2 C y freeglut

- Usado para la creación de minijuegos
- Cada minijuego se abría en Godot dependiendo el mundo y objetivo, se hace mediante otra pantalla

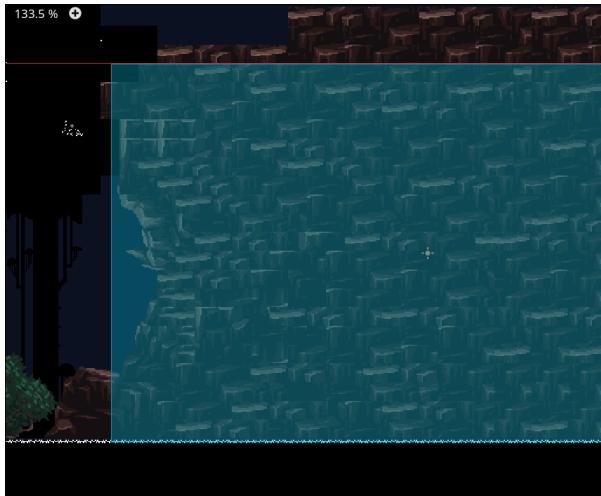


Figure 13: Area para compilar el programa

```
22
23 func open_c_program():
24     # convierte la ruta de Godot a una ruta absoluta del sistema de ar-
25     var texture_path = ProjectSettings.globalize_path("res://TEXTURAS/
26     "
27     # ruta del programa
28     var command = "/home/denissexocuis/Documentos/folder-master/master
29     "
30     # ruta de la textura
31     var arguments = [texture_path] # El programa C debe esperar este
32     "
33     # ejecuta el programa en c
34     var output = []
35     var result = OS.execute(command, arguments, output, false, false)
36     "
37     if result == 0:
```

Figure 14: Script para compilar



Figure 15: Área para compilar el programa

3.2 Minijuegos

3.2.1 Minijuego 1: Réplica Puzzle de Undertale

El primer minijuego que el personaje tiene que realizar para avanzar es un puzzle recreado del minijuego de Undertale. Se usa una pila con la secuencia de teclas para la resolución y una cola para la entrada por teclado de dicha resolución.

Ganar es muy facil, debe haber un espacio vacio entre el triangulo amarillo y blanco, si no lo hay, se reinicia el juego.

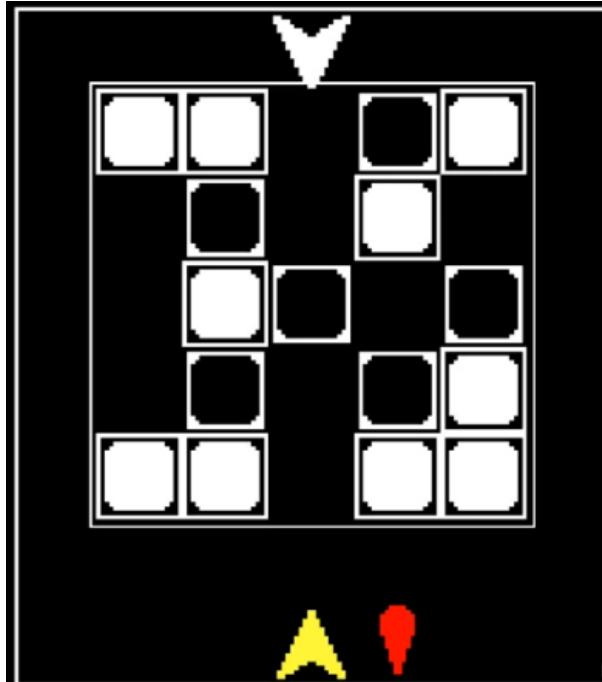


Figure 16: Puzzle de Undertale en Hotland

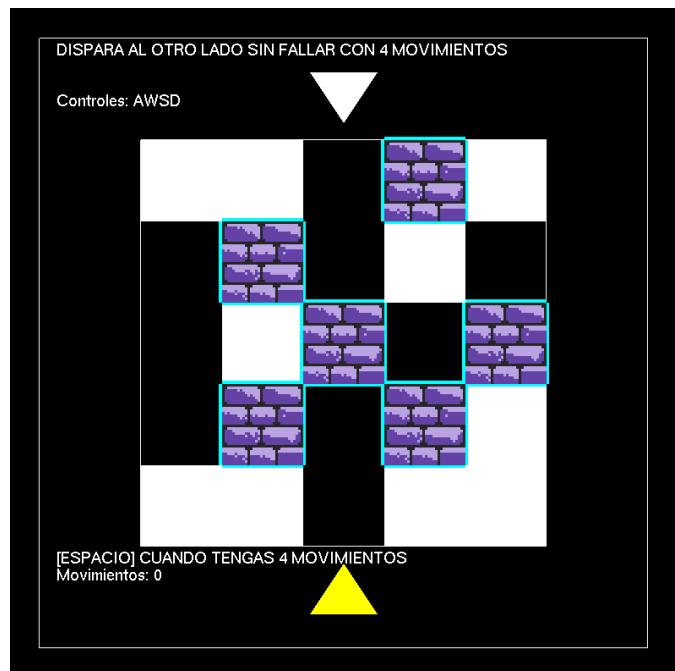


Figure 17: Puzzle recreado

3.2.2 Minijuego 2: Ordenar Signos Matemáticos

El segundo minijuego es sobre ordenar signos de menor a mayor de forma jerárquica. Se usa una cola que tiene los signos y una pila para que, de acuerdo a la selección del jugador, se llene. Se verifica si la pila está en el orden correcto.

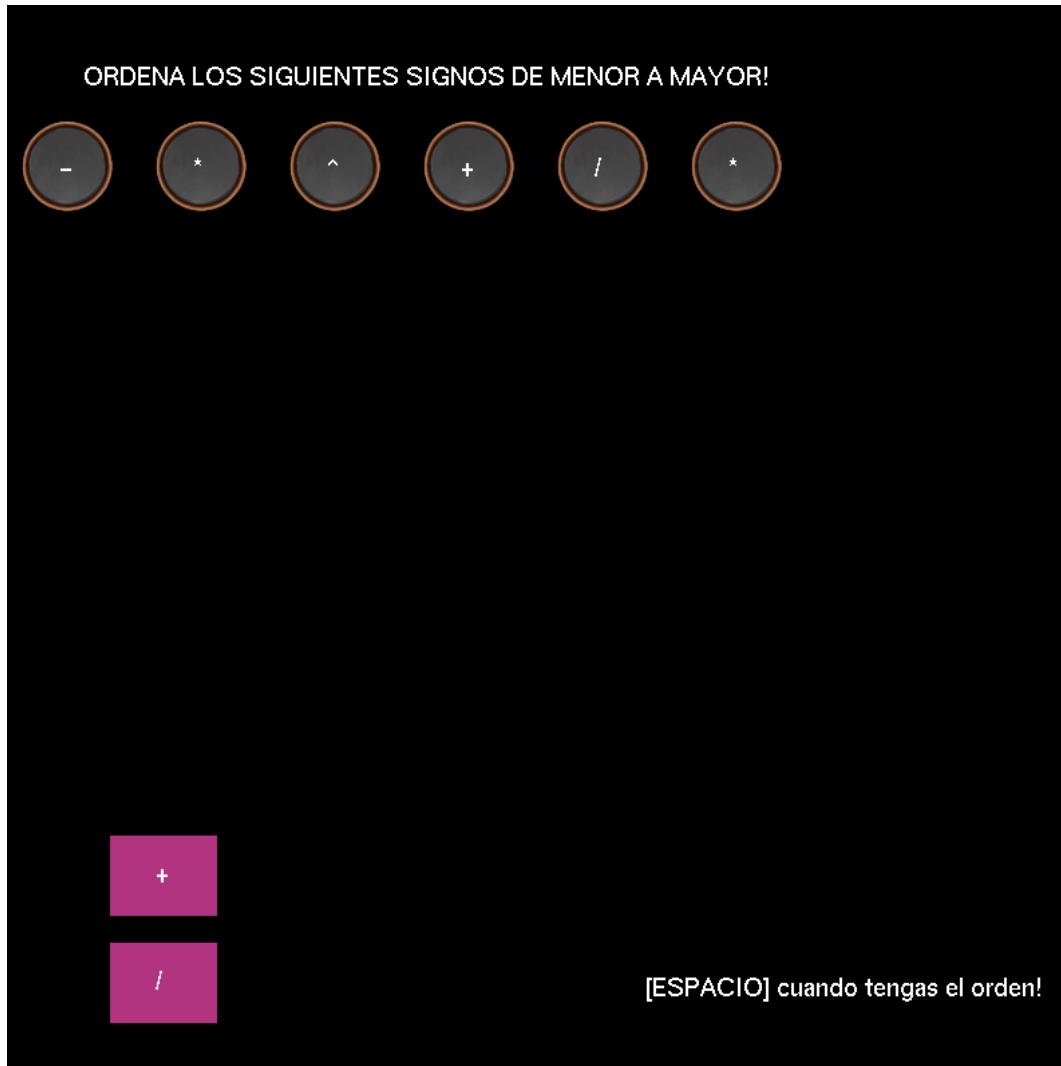


Figure 18: Ordenar signos

3.2.3 Minijuego 3: Preguntas Capsiosas

El tercer minijuego es sobre preguntas capsiosas, se usa un arbol binario para las preguntas y entrada por teclado de dichas preguntas. Es un minijuego con una lógica sencilla pero lo complejo de este son el nivel de las preguntas ya que tambien hay preguntas muy específicas sobre la segunda guerra mundial.

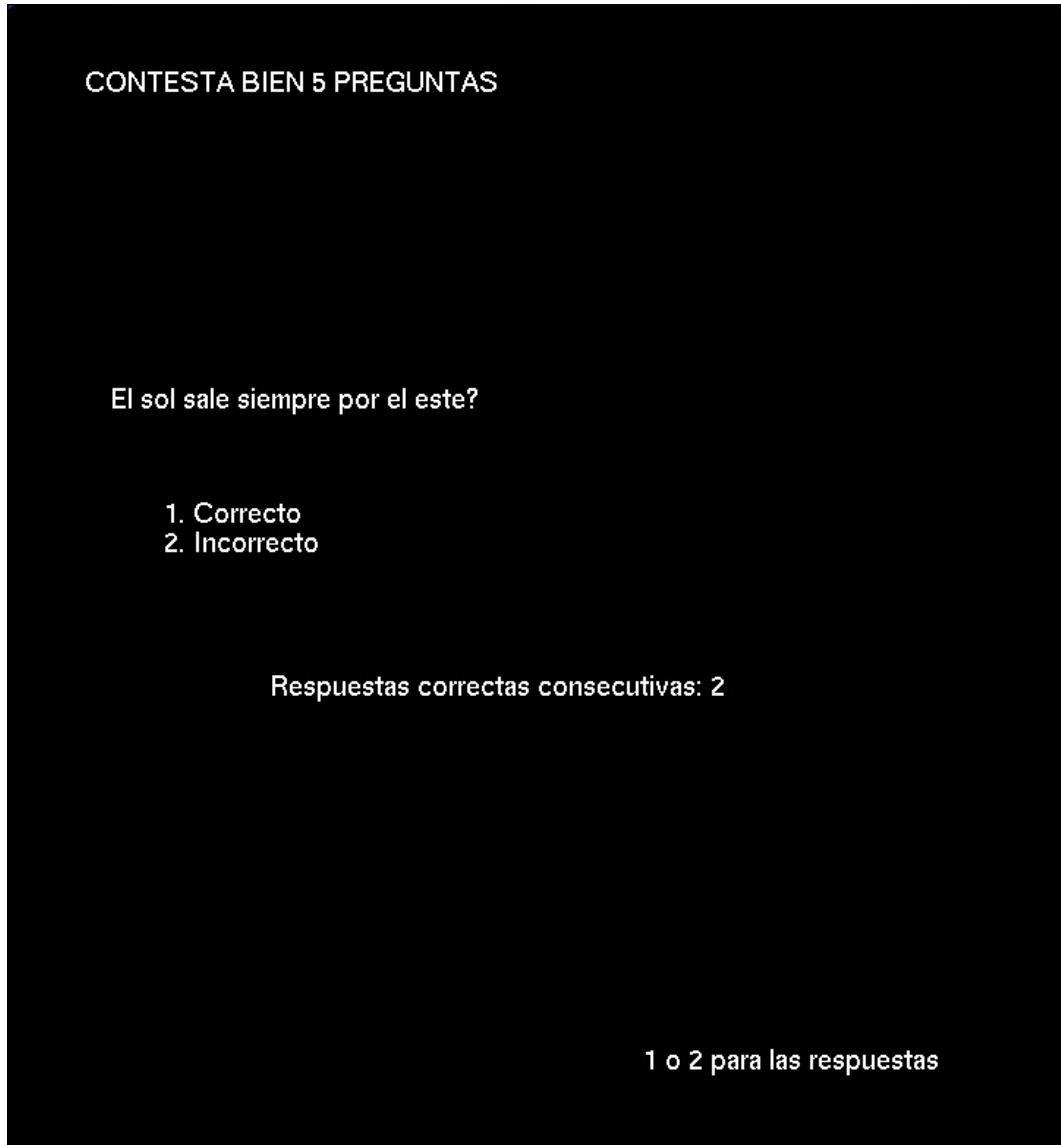


Figure 19: Ordenar signos

3.3 Diagramas

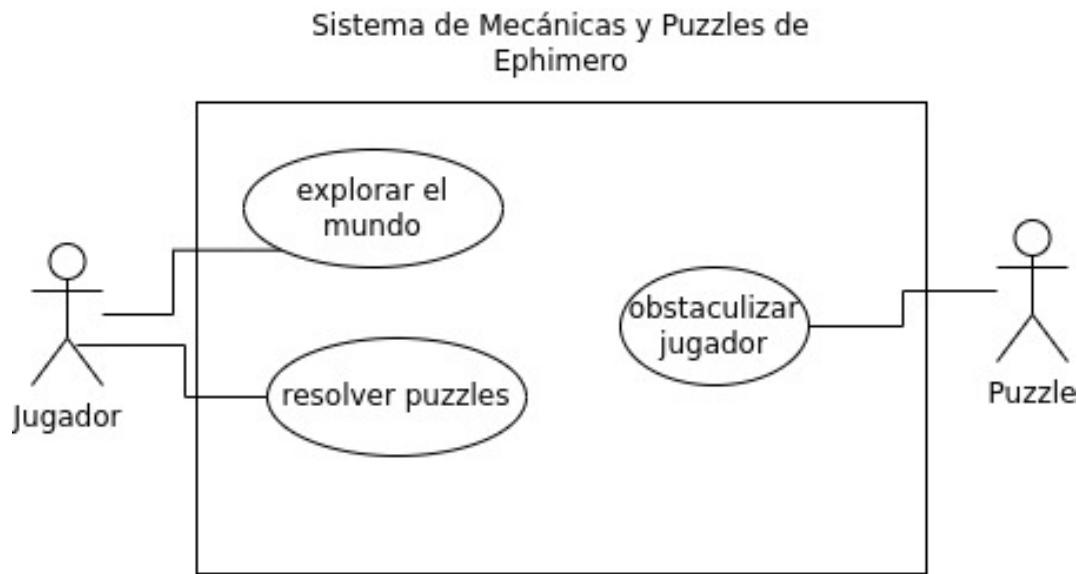


Figure 20: Ordenar signos

3.4 Detalles Visuales y Sonoros

Música incluida en el juego:

- Burning Hill - Mitski
- The Color of Depression - Nier Automata
- Waterfall - Undertale [HASTA LA ESCENA 3]
- éphimero - Canción Original de 'Éphimero' by Schleske Alan

4 Implementación

Se usó `GL/freeglut.h`, `math.h` y `stb_image.h` para ambos juegos.

4.1 Réplica Puzzle de Undertale

Se definen constantes como el número de cuadros (`CUADROS_No`) y macros para redondear números (`ROUND`), así como `stb_image.h` para cargar texturas de manera sencilla.

Se define una estructura `CUADRO` que representa un "cubo" para dibujar cada cubo que haga (se hace un arreglo de estructuras), también se ocupa una `COLA` para la entrada de teclado y una `PILA` para almacenar la secuencia de movimientos para resolver el puzzle.

```

9  #include <stdio.h>
10 #include <GL/freeglut.h>
11 #include <math.h>
12 #define CUADROS_No 15
13 /* para redondear a dos decimales porque tuve problemas
14 /* con la comparación de posiciones TvT
15 #define ROUND(u) (round((u) * 100.0) / 100.0)
16
17 //! TEXTURA uv
18 #define STB_IMAGE_IMPLEMENTATION
19 #include "stb_image.h"
20 /* se usó stb_image.h en vez de SOIL por su facilidad de compilación ;)
21
22 //? es movable? TRUE, FALSE
23 enum _bool{FALSE, TRUE};
24 //? variables para la textura
25 unsigned int textureid;
26 // const char *path = "GAME1.png";
27 int move = 0, exito = 2;
28
29 //! estructura para arreglo de estructuras, es para cada cubito
30 typedef struct
31 {
32     float x, y, size;
33     enum _bool bbol;
34 }CUADRO;
35
36 CUADRO cuadros[CUADROS_No];
37
38 //! Cola para verificar la entrada del teclado del jugador
39 typedef struct nodocola
40 {
41     char letra;
42     struct nodocola *next;
43 }NODITO_COLA;
44
45 typedef struct cola
46 {
47     NODITO_COLA *frente;
48     NODITO_COLA *final;
49 }COLA;
50
51 /* crear un nodocola
52 NODITO_COLA *crear_nodo(char w)
53 {
54     NODITO_COLA *nodo;
55     nodo = (NODITO_COLA *)malloc(sizeof(NODITO_COLA));
56     nodo->letra = w;
57     nodo->next = NULL;
58     return nodo;
59 }
```

Funciones de o con Colas

- Crear nodo
- Insertar
- Vaciar cola

```

93  /// PILA para manejar el orden de la resolución del puzzle
94  //? resolución: w a s a xD
95  typedef struct nodopila
96  {
97  |     char letra;
98  |     struct nodopila *next;
99  }PILA;
100
101 /* pop
102 PILA *pop(PILA **topo)
103 {
104     if(!*topo) return NULL;
105     PILA *aux = (*topo);
106     (*topo) = (*topo)->next;
107     return aux;
108 }
109
110 /*push
111 void push(char letra, PILA **topo)
112 {
113     PILA *nodito = malloc(sizeof(PILA));
114     if(!nodito) return;
115     nodito->letra = letra;
116     nodito->next=(*topo);
117     (*topo) = nodito;
118 }
119
120 void cargarResolucion(PILA **topo)
121 {
122     push('a', &(*topo));
123     push('s', &(*topo));
124     push('a', &(*topo));
125     push('w', &(*topo));
126 }
127
128 static void init()
129 {
130     glClearColor(0.0, 0.0, 0.0, 0.0);
131     gluPerspective(60.0,1.0,4.0,100.0);
132     glShadeModel(GL_SMOOTH);
133     glDepthFunc(GL_LESS);
134     glEnable(GL_DEPTH_TEST);
135 }
```

Funciones de o con Pilas

- Push
- Pop
- Cargar resolución (secuencia de movimientos para resolver el puzzle)
- rellenarPila: vuelve a llenar la pila con la nueva resolución, la vacía y luego vuelve a llenar

La función **CargarTextura** carga la textura desde un archivo usando el **stb_image.h** y usando parámetros de textura del propio OpenGL.

```

138 void cargarTextura(const char *path)
139 {
140     int width, height, nrChannels;
141
142     unsigned char *data = stbi_load(path, &width, &height,
143     &nrChannels, 0);
144
145     glGenTextures(1, &textureid);
146     glBindTexture(GL_TEXTURE_2D, textureid);
147
148     glTexImage2D(GL_TEXTURE_2D, 0, (nrChannels == 4 ? GL_RGBA : GL_RGB), width,
149     height, 0, (nrChannels == 4 ? GL_RGBA : GL_RGB), GL_UNSIGNED_BYTE, data);
150
151     glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
152     glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
153     glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
154     glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
155
156     stbi_image_free(data);
157 }
158
159 void lineas()
160 {
161     //! lineas
162     glBegin(GL_LINE_STRIP);
163         glVertex3f(0.9, 0.9, 0.0);
164         glVertex3f(-0.9, 0.9, 0.0);
165         glVertex3f(-0.9, -0.9, 0.0);
166         glVertex3f(0.9, -0.9, 0.0);
167         glVertex3f(0.9, 0.9, 0.0);
168     glEnd();
169
170     glBegin(GL_LINE_STRIP);
171         glVertex3f(0.6, 0.6, 0.0);
172         glVertex3f(-0.6, 0.6, 0.0);
173         glVertex3f(-0.6, -0.6, 0.0);
174         glVertex3f(0.6, -0.6, 0.0);
175         glVertex3f(0.6, 0.6, 0.0);
176     glEnd();
177 }
```

Funciones de dibujo

- Lineas: dibuja las lineas blancas del margen
- drawCuadros: dibuja los cuadros que se definen en el arreglo (depende si es movable o no)
- InicializarC : inicializa los datos de los cuadros del arreglo
- texto : muestra el texto en pantalla, también agrego texto dinámico dependiendo de variables de control
- draw: renderiza la ventana con freeglut

```
367
368 void keyboard(unsigned char key, int x, int y)
369 {
370     /* las había puesto como variables globales pero tuve problemas y guardaba basura :(
371     static COLA head = {NULL, NULL};
372     static PILA *tope = NULL;
373
374     /* cargar la resolución [PILA] si es null desde el principio
375     if(!tope) cargarResolucion(&tope);
376     move++; /* aumentar cantidad de movimientos para imprimir
377     //? si se pasan los movimientos, checar si todos los movimientos coinciden con la resolución de
378     la PILA
379     if(move>4)
380     {
381         while(head.frente)
382         {
383             /* hacer pop en la pila si es el orden correcto
384             if(tope->letra == head.frente->letra)
385             {
386                 pop(&tope);
387                 /* avanzar
388                 NODITO_COLA *temp = head.frente;
389                 head.frente = head.frente->next;
390                 free(temp);
391                 continue;
392             }
393             head.frente = head.frente->next;
394         }
395         /* si se eliminó todo, es porque completó el puzzle
396         if(!tope)
397         {
398             exito = 1;
399             move = 4;
400             glutPostRedisplay();
401             /* cerrar la ventana automáticamente
402             glutTimerFunc(3000, exitP, 0);
403             return;
404         } /* sino, el orden está mal y es necesario reiniciar
405         else
406         {
407             move = 0;
408             exito = 0;
409             vaciarCola(&head);
410             llenarPila(&tope);
411             inicializarC();
412             drawCuadros();
413             glutPostRedisplay();
414         }
415
416
417     }
418     /*? sino, ir encolando la entrada del usuario
```

```

417 }
418 //? sino, ir encolando la entrada del usuario
419 else insertarCola(&head, key);
420
421 //? Lógica de movimiento dependiendo la letra
422 switch (key)
423 {
424     case 'd':
425         //! Lógica por colisión en x
426         for(int i=0; i <= CUADROS_No; i++)
427         {
428             /* se mueve?
429             if(cuadros[i].bbol)
430             {
431                 int move=1;
432                 //? verificar si hay una colisión con cuadros estáticos
433                 for(int j=0; j <= CUADROS_No; j++)
434                 {
435                     /* es cuadro estático?
436                     if(!cuadros[j].bbol)
437                     {
438                         /* hay colisión?
439                         if(ROUND(cuadros[i].x + 0.24) == ROUND(cuadros[j].x) &&
440                             ROUND(cuadros[i].y) == ROUND(cuadros[j].y))
441                         {
442                             move=0;
443                             break;
444                         }
445                     }
446                 }
447
448                 //? verificar si hay una colisión con cuadros NO estáticos
449                 for(int j=0; j <= CUADROS_No; j++)
450                 {
451                     if(cuadros[i].bbol)
452                     {
453                         /* hay colisión?
454                         if(ROUND(cuadros[i].x + 0.24) == ROUND(cuadros[j].x) &&
455                             ROUND(cuadros[i].y) == ROUND(cuadros[j].y))
456                         {
457                             move=0;
458                             break;
459                         }
460                     }
461                 }
462
463                 //? verificar si pasa del margen de líneas
464                 if(ROUND(cuadros[i].x + 0.24) == 0.72) move = 0;
465                 if(move) cuadros[i].x += 0.24;
466             }
467             break;
468         case 'a':
469             //! Lógica por colisión en x

```

main* ⓘ ① 0 △ 0 ⌂ 0 Live Share ▶ Compile & Run ⓘ Compile ⓘ Debug ✓

Funciones lógicas o manejo de usuario

- keyboard: una función sumamente importante ya que ahí manejo la inicialización de la cola y pila, además de que valido el orden del puzzle después de 4 movimientos DE ACUERDO a las teclas que se presionan. Cada tecla tiene su propia lógica de movimiento, ahí se verifican colisiones con cuadros dinámicos o estáticos, al igual que el propio margen del juego (líneas blancas)

4.2 Ordenar signos matemáticos

Tenemos un define para el tamaño máximo de la pila, es para controlar cuantos signos pueden meterse en la pila. El enum es para manejar el estado del texto, es para mostrar un texto de victoria o vencida, también se almacenan coordenadas del mouse.

Funciones de o con Pilas

- pop
- push
- pilaLlena : para ver si se alcanzó el límite
- vaciarPila : eliminar todos los signos de la pila



```
6
7 #include <stdio.h>
8 #include <GL/freeglut.h>
9 #include <math.h>
10 #define TAMANIO_PILA 8
11 //! TEXTURA vv
12 #define STB_IMAGE_IMPLEMENTATION
13 #include "stb_image.h"
14 /* se usó stb_image.h en vez de SOIL por su facilidad de compilación ;)
15
16 //? variables para la textura
17 unsigned int textureid;
18 // const char *path = "button.png";
19 float mouseX, mouseY;
20
21 enum _bool{FALSE, TRUE, MIDDLE}BOOL;
22
23 enum _bool BOOL = MIDDLE;
24 typedef char TipoData;
25
26 /**
27 * *el jugador recibe numeros aleatorios (guardados en una cola)
28 * * debe ordenarlos de menor a mayor (guardado en una pila)
29 */
30
31 //! -----PILA-----
32 > typedef struct nodopila ...
33 }PILA;
34
35 /* pop
36 > void *pop(PILA **tope) ...
37
38 /*push
39 > void push(PILA **tope, TipoData x) ...
40
41 PILA *tope = NULL;
42
43 /* pila llena
44 > int pilaLlena() ...
45
46 /* vaciar pila
47 > void vaciarPila() ...
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
```

main* ① ② ③ ④ ⑤ ⑥ Live Share ▶ Compile & Run ⚙ Compile ⚙ Debug ✓

Funciones de o con Colas

- Crear nodo
- Insertar
- Vaciar cola
- eliminarDeCola: buscar un valor y eliminarlo
- CargarCola: llenar la cola con operadores matemáticos

```
85 //! -----COLA-----
86 typedef struct nodocola
87 {
88     TipoData x;
89     struct nodocola *suivant;
90 }NODITO_COLA;
91
92 > typedef struct cola...
93 }COLA;
94
95 COLA cola_head = {NULL,NULL};
96
97 /* crear un nodocola
98 > NODITO_COLA *crear_nodo(TipoData x) ...
99 */
100
101 /* insertar
102 > void insertarCola(TipoData x) ...
103 */
104
105 /* vaciar cola
106 void vaciarCola()
107 {
108     NODITO_COLA *temp;
109     while(cola_head.avant)
110     {
111         temp = cola_head.avant;
112         cola_head.avant = cola_head.avant->suivant;
113         free(temp);
114     }
115     cola_head.avant = cola_head.final = NULL;
116 }
117
118 /* eliminar por dato
119 void eliminarDeCola(TipoData x)
120 {
121     NODITO_COLA *temp = cola_head.avant,
122             *prev = NULL;
123
124     while(temp)
125     {
126         if(temp->x == x)
127         {
128             if(!prev) /* primer nodo
129             {
130                 cola_head.avant = temp->suivant;
131                 free(temp);
132             }
133             else
134             {
135                 prev->suivant = temp->suivant;
136                 free(temp);
137             }
138         }
139     }
140 }
```

Funciones de dibujo

- drawPila: dibujar la pila en la ventana con rectángulos
- drawCola: dibujar la cola con textura
- drawWINorFAIL: mostrar mensajes de victoria o vencida dependiendo la variable del enum
- text: dibuja las instrucciones
- draw: renderiza la escena

```
247
248 void drawPila()
249 {
250     PILA *actuel = tope;
251     float x = -0.8,
252          y = -0.8;
253
254     while(actuel)
255     {
256         glColor3f(0.7,0.2,0.5);
257
258         glBegin(GL_QUADS);
259             glVertex2f(x, y + 0.05);
260             glVertex2f(x + 0.2, y + 0.05);
261             glVertex2f(x + 0.2, y - 0.1);
262             glVertex2f(x, y - 0.1);
263     glEnd();
264
265     /* dibujar valores de la pila
266     glColor3f(1.0, 1.0, 1.0);
267     glRasterPos2f(x + 0.085, y - 0.036);
268     glutBitmapCharacter(GLUT_BITMAP_HELVETICA_18, actuel->x);
269
270     actuel = actuel->suivant;
271     /* desplaza un poco para el otro nodo, es para el espacio entre noditos
272     y += 0.2;
273 }
274 }
275
276 void drawCola()
277 {
278     NODITO_COLA *actuel = cola_head.avant;
279     /* posicion en la pantalla
280     float x = -0.98,
281           y = 0.7;
282
283     while(actuel)
284     {
285         glColor3f(1,1,1);
286         glBindTexture(GL_TEXTURE_2D, textureid);
287         glEnable(GL_TEXTURE_2D);
288
289         glBegin(GL_QUADS);
```

Funciones lógicas o manejo de usuario

- exitP: cerrar la ventana
- VALOR: asignarle un valor numérico al operador, para comparar si está ordenada la pila
- verificarRespuesta: verifica si los signos ingresados en la pila están en orden ascendente
- seleccionMouse: aquí detecta el click del mouse y mueve los signos de la cola a la pila, después vuelve a dibujar
- mouse: convierte las coordenadas del mouse a coordenadas de OpenGL
- keyboard: detecta la pulsación de la tecla espacio y verifica si la pila está ordenada
- reiniciarJuego: vacía la pila y la cola, recarga la cola y redibuja.

```
315
316 void drawWINorFAIL(int bool)
317 {
318     if(bool)
319     {
320         /* posición del texto
321         glRasterPos2f(0,0);
322         const char *mensaje = "MUY BIEN!";
323         for (const char *c = mensaje; *c != '\0'; c++)
324         {
325             glutBitmapCharacter(GLUT_BITMAP_HELVETICA_18, *c);
326         }
327         /* cerrar la ventana automáticamente
328         glutTimerFunc(3000, exitP, 0);
329
330         glutPostRedisplay();
331     }
332     else
333     {
334         /* posición del texto
335         glRasterPos2f(0,0);
336         const char *mensaje = "HAZLO DE NUEVO!";
337         for (const char *c = mensaje; *c != '\0'; c++)
338         {
339             glutBitmapCharacter(GLUT_BITMAP_HELVETICA_18, *c);
340         }
341         glutPostRedisplay();
342     }
343 }

344
345 // asignarle valor a los signos para saber si está ordenado
346 int VALOR(char signo)
347 {
348     switch (signo)
349     {
350         case '^': return 3;
351         case '/': case '*': return 2;
352         case '+': case '-': return 1;
353     }
354 }
355
356 int verificarRespuesta()
```

The screenshot shows a dark-themed code editor with a C++ file open. The code contains functions for drawing text ('drawWINorFAIL'), handling operator precedence ('VALOR'), and checking user input ('verificarRespuesta'). The interface includes standard file operations (New, Open, Save, etc.) and a toolbar with icons for live share, compile/run, and debug.

```
380 void seleccionMouse(float x, float y)
381 {
382     float ancho = 0.2, alto = 0.1, margen = 0.04; int i=0;
383
384     /* para ver si el click está adentro del boton dibujado
385     NODITO_COLA *actuel = cola_head.avant;
386     float x_inicio = -0.99,
387           y_final = 0.75f;
388
389     while(actuel)
390     {
391         float COLA_X = x_inicio + i * (ancho + margen),
392             COLA_Y = y_final;
393
394         if (x >= COLA_X && x <= COLA_X + ancho && y >= COLA_Y -
395             alto && y <= COLA_Y)
396         {
397             /* extraer el num y ponerlo en la pila
398             int num = actuel->x;
399             push(&tope, num);
400             eliminarDeCola(num);
401             /* dibujar pila
402             /* dibujar pila
403             drawPila();
404             /* volver a dibujar cola porque se eliminó
405             drawCola();
406             glutPostRedisplay();
407
408             break;
409         }
410         actuel = actuel->suivant; i++;
411     }
412 }
```

```
14
15  /* manejar la entrada por mouse
16  void mouse(int button, int estado, int x, int y)
17  {
18      if (button == GLUT_LEFT_BUTTON && estado == GLUT_DOWN)
19      {
20          //? convertir coordenadas de mouse a coordenadas de opengl
21          mouseX = (x / (float)glutGet(GLUT_WINDOW_WIDTH)) * 2.0 -
22                      1.0;
23          mouseY = 1.0 - (y / (float)glutGet(GLUT_WINDOW_HEIGHT)) *
24                      2.0;
25
26          if (mouseX >= -1.0 && mouseX <= 1.0 && mouseY >= -1.0 &&
27              mouseY <= 1.0)
28              seleccionMouse(mouseX, mouseY);
29      }
30  }
```

5 Resultados

SE ADJUNTA VIDEO DE PRUEBA DEL VIDEOJUEGO: CLICK AQUI

6 Manual de usuario

6.1 ¿CÓMO JUGAR?

El usuario tiene 4 controles:

- w: saltar
- a: ir a la izquierda
- d: ir a la derecha
- ESPACIO: interactuar con objetos
- mouse: para opciones del menú

7 Conclusiones

Desafortunadamente cada versión que sacan de godot sale algo nuevo o cambian las formas de implementar cosas, por lo tanto, manejar y compilar los programas con texturas si fue un reto; aún así, me encuentro satisfecha de poder haber logrado que dichos programas en C se ejecuten en Godot como ventanas externas, también me encuentro satisfecha sobre como manejé opciones, texturas, algoritmos y animaciones.

Una dificultad que tuve era tomarme el tiempo de hacer los scripts en el propio motor ya que, como mencioné, algunas cosas habían cambiado, por lo que me tomaba más tiempo diseñar el juego en el motor que hacer un minijuego en C.

Sin embargo, se logró hacer una combinación de ambos de forma exitosa.

8 Referencias

Godot 4.3 y página oficial: Godot Engine. (2024). Godot Engine 4.3. Godot Engine.Godot Engine

LibreAseprite: LibreSprite. (n.d.). LibreSprite.Github

OpenGL SuperBible: Wright, R., Lipchak, B., & Haemel, N. (2014). OpenGL SuperBible: Comprehensive Tutorial and Reference (4th ed.). Addison-Wesley.

Itch.io: Itch.io. (n.d.). Itch.io. Itch.io