Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

**PGE: Parcial I**

* **Asignatura:** Programación Genérica y Eventos
* **Alumnos:** Gonzalo Rementeria y Manuel Nicolais
* **Tema:** Parcial I – Programación Genérica y Eventos
* **Fecha de entrega:** 13/09/24.
* **Software:** Godot Engine

**Instrucciones:**

1. **Identificación de Problemas de Usabilidad:**
   * Seleccionar una aplicación de software con la que interactúen regularmente (puede ser un software de código abierto o uno de uso académico).
   * Identificar las características específicas que hacen que el software sea difícil de usar o poco amigable para el usuario.
   * Documentar estas características y justificar por qué representan un problema desde la perspectiva del usuario.
2. **Propuesta de Mejora:**
   * Basándose en los problemas identificados, cada equipo desarrollará un prototipo funcional en C++ que incluya mejoras a través de:
     + **Callbacks:** Implementación de funciones de retorno para mejorar la interacción dinámica en la interfaz.
     + **Bucles:** Uso de bucles eficientes para gestionar las interacciones continuas y el flujo de la interfaz.
     + **Eventos Paint:** Modificación de la apariencia de la interfaz en tiempo real según las interacciones del usuario.
     + **Funciones de Conversión:** Integración de funciones que faciliten la adaptación de formatos o tipos de datos según las necesidades del usuario.
3. **Desarrollo del Prototipo:**
   * Crear un prototipo en C++ que implemente las soluciones propuestas. Este prototipo no tiene que ser un programa completo, pero sí debe demostrar claramente cómo las mejoras propuestas resuelven los problemas de usabilidad identificados.
4. **Documentación:**
   * Cada equipo deberá entregar un informe que incluya:
     + Una descripción del software analizado y los problemas de usabilidad encontrados.
     + Una explicación detallada de las soluciones propuestas y cómo se implementaron en el prototipo.
     + El código fuente del prototipo en C++ con comentarios que expliquen cada sección relevante.
     + Capturas de pantalla o videos que muestren el prototipo en funcionamiento, resaltando las mejoras implementadas.

**Introducción:**

En el presente informe, en el contexto de la materia, Programación Genética y Eventos se detallarán diversas funcionalidades o características de la interfaz grafica del programa de software libre seleccionado, en este caso **Godot Engine**, motor utilizado para el desarrollo de videojuegos 2D y 3D de código abierto.

Icono

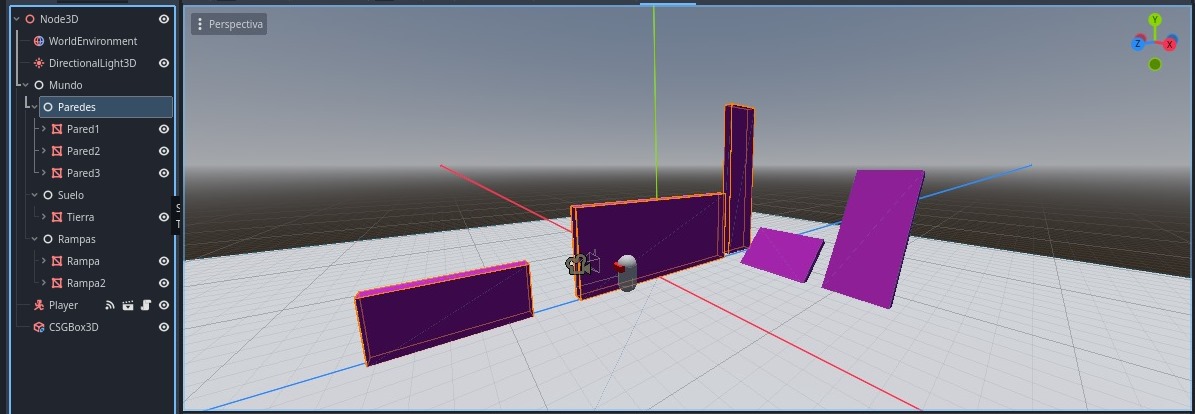
Descripción generada automáticamente

**Desarrollo:**

En esta sección del informe se detallarán las que se consideran características poco amigables o que representen algún tipo de problema para el usuario con respecto a la interfaz de usuario o atributos del motor. Se debe tener en cuenta que estas características están siendo analizadas utilizando la última versión de Godot hasta la fecha en Windows, la cual es **Godot 4.2**

1. **Contornos en los objetos que pertenecen a un nodo especifico:**

El motor Godot funciona puramente utilizando un sistema de nodos, con hijos y padres asignados a estos, el problema que podemos apreciar es cuando en una escena hacemos uso de varios objetos. En el momento de organizar los objetos por nodos para mantener un orden en el proyecto, el usuario no puede apreciar cuando selecciona, por ejemplo, un Nodo Vacío, la selección de objetos que pertenezcan a este nodo. Podría ser mas intuitivo para el usuario poder apreciar todos los objetos que pertenecen a ese nodo de forma de un contorno alrededor de estos como se puede ver en la imagen.



**Ilustración 1 – ejemplo del cambio a aplicar**

Actualmente en Godot 4.2 lo que se ve en la **ilustración 1** no existe, específicamente ablando, Cuando seleccionamos el Nodo “Paredes” desde el los objetos, se puede ver como en la Escena los tres objetos que pertenecen a ese nodo se ven resaltados por un recuadro naranja(Pared1, Pared2, Pared3).

**Implementación en C++**

La forma en la que se implemento este cambio utilizando los conceptos antes mencionados es la siguiente.

**Clase Nodo:**

Esta clase representa un nodo dentro del sistema. Cada nodo tiene un nombre y un estado (seleccionado o no).

*// Clase Nodo que representa cada nodo que puede ser seleccionado o deseleccionado*

class Nodo {

public:

    string nombre; *//string que identifica cada Nodo*

    bool seleccionado; *// booleano que indica si el nodo está seleccionado o no*

*// Constructor por defecto*

    Nodo() : nombre(""), seleccionado(false) {}

*// Constructor que recibe un nombre para inicializar el nodo.*

    Nodo(string n) : nombre(n), seleccionado(false) {}

*// Método para seleccionar el nodo*

*//Cambia el estado del nodo a seleccionado e imprime un mensaje.*

    void seleccionar() {

        seleccionado = true;

        separador();

        cout << "Nodo " << nombre << " seleccionado.\n";

        cout << "Se remarcaron los objetos de este nodo.\n";

    }

*// Método para deseleccionar el nodo*

*//Cambia el estado del nodo a no seleccionado e imprime un mensaje.*

    void deseleccionar() {

        seleccionado = false;

        separador();

        cout << "Nodo " << nombre << " deseleccionado.\n";

        cout << "Se desmarcaron los objetos de este nodo.\n";

    }

};

**Función convertirEventoAManejador**

Esta función toma un evento (como "seleccionar" o "deseleccionar") y el nodo afectado, y devuelve una función de callback que se encargará de ejecutar la acción correspondiente sobre el nodo.

*// Función de conversión de eventos a manejadores, que ahora recibe el nombre del nodo*

function<void()> convertirEventoAManejador(const string& evento, Nodo& nodo) {

    if (evento == "seleccionar" || evento == "1") {

        return [&]() { nodo.seleccionar(); separador(); }; *// Callback para seleccionar el nodo*

    }

    if (evento == "deseleccionar" || evento == "2") {

        return [&]() { nodo.deseleccionar(); separador();}; *// Callback para deseleccionar el nodo*

    }

    return []() { cout << "Evento no reconocido.\n"; }; *// Evento no válido*

}

**Función DespachadorEventos**

Crea un mapa de nodos, y permite al usuario seleccionar o deseleccionar nodos

*// Función para el despachador de eventos que permite seleccionar entre varios nodos*

void despachadorEventos() {

*// Creamos un mapa de nodos donde la clave es el nombre del nodo*

    map<string, Nodo> nodos = {

        {"Nodo1", Nodo("Nodo1")},

        {"Nodo2", Nodo("Nodo2")},

        {"Nodo3", Nodo("Nodo3")}

    };

*// Bucle principal que escucha eventos simulados del usuario*

    while (true) {

        string evento, nombreNodo;

        cout << "Ingrese el evento: \n";

        cout << "1. Seleccionar\n2. Deseleccionar\n3. Salir\n";

        cin >> evento;

*// Si el evento es "salir", se termina el programa*

        if (evento == "salir" || evento == "3") {

            break;

        }

*// Solicita el nombre del nodo para aplicar el evento*

        cout << "Ingrese el nombre del nodo ('Nodo1', 'Nodo2', 'Nodo3'): ";

        cin >> nombreNodo;

*// Verifica si el nodo existe en el mapa*

        if (nodos.find(nombreNodo) != nodos.end()) {

*// Conversión del evento a un manejador para el nodo especificado*

            function<void()> manejadorEvento = convertirEventoAManejador(evento, nodos[nombreNodo]);

*// Ejecuta el manejador correspondiente (seleccionar o deseleccionar)*

            manejadorEvento();

        } else {

            cout << "Nodo no encontrado.\n";

        }

    }

}

**Función main**

Solo ejecuta la función despachadorEventos.

int main() {

*// Ejecutamos el bucle despachador de eventos*

    despachadorEventos();

    return 0;

}

En la implementación de este código se utilizaron los conceptos vistos en la unidad 1 de la materia Programación Generica y Eventos junto con otras funciones proporcionadas por el entorno de trabajo, los cuales por mencionarlos son:

Bucle despachador de eventos: Captura eventos del usuario y los mapea a manejadores apropiados.

Callbacks: Las acciones sobre los nodos (selección y deselección) son ejecutadas como callbacks.

Conversión de eventos: Mapea eventos del usuario a manejadores de eventos específicos del nodo seleccionado.

Mapeo de nodos: Los nodos se gestionan mediante un mapa que asocia los nombres de los nodos con sus instancias.

**Conclusión:**

Como conclusión a este prototipo de implementación a desarrollar en Godot se puede tener en cuenta que este cambio puede ayudar mucho a los usuarios o desarrolladores de videojuegos con este software a tener una mejor noción de los objetos que son parte de un Nodo en la escena principal, con el fin de evitar errores o problemas a largo plazo con la utilización de varios objetos y ayudar a la organización de el proyecto de los usuarios.

**Codigo en Github:** <https://github.com/ManuNicolais/PGE/tree/main/Parcial%201>