UNIVERSIDAD AUTÓNOMA DE CHIHUAHUA

FACULTAD DE INGENIERÍA

CONCEPTOS AVANZADOS DE PROGRAMACIÓN

INGENIERÍA EN CIENCIAS DE LA COMPUTACIÓN

Profesor: Raymundo Cornejo García

Integrantes:

Javier Andres Tarango Fierro 329904

Ximena Romero Chavez 329898

Juan Daniel Villegas Terrazas 329545

Ricardo Corral Sánchez 329606

Luis Enrique Fernández Reza 329865

29/05/2019

INTRODUCCIÓN

Nuestro proyecto consiste en un juego de rol en el cual se desarrolla una historia intergaláctica.

El personaje principal que es nombrado por el usuario es un viajero espacial que luego de haber regresado de una misión de exploración, se percata de que su hogar, el planeta tierra ha sido invadido por una raza alienígena que ha capturado a la mayoría de sus habitantes incluyendo a su familia.

Nuestro personaje tendrá como misión adentrarse en el espacio con el fin de encontrar y salvar a su familia. Tendrá que enfrentar diferentes adversidades para lograr su objetivo y salir victorioso, pero no solo se trata de decidir. El jugador tendrá que pensar las consecuencias de sus decisiones las cuales pueden afectar el final y el transcurso de la historia.

OBJETIVO

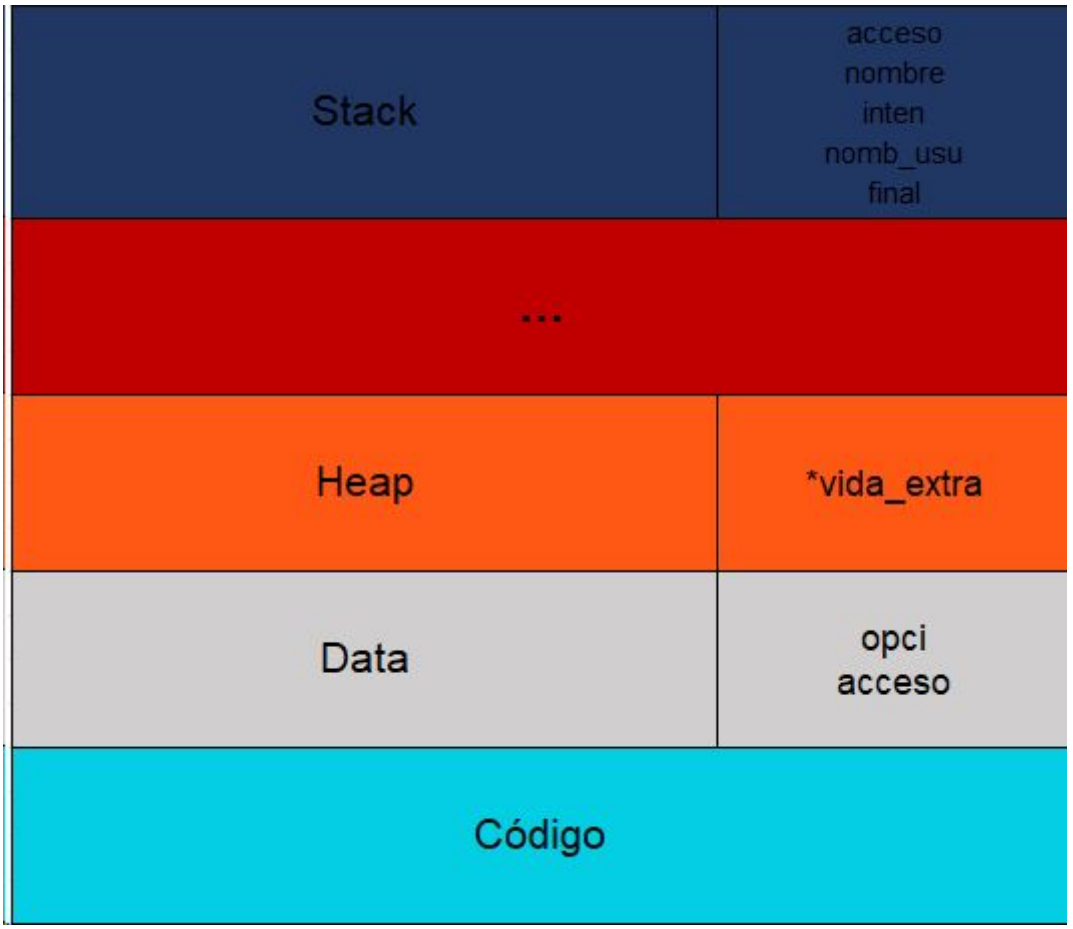
- ➢ Crear un juego de rol que ponga a prueba la astucia y audacia del jugador.
- ➢ Garantizar un momento de diversión y entretenimiento.
- ➢ Hacer reflexionar al jugador sobre cómo una decisión puede hacer la diferencia entre el éxito y fracaso.

Es importante conocer que existen consecuencias al tomar una decisión y que esto puede cambiar mucho el transcurso de una historia.

No sólo es cuestión de decidir, nosotros queremos mostrar que así la más pequeña decisión puede cambiarlo todo, hasta el fin del juego.

DESCRIPCIÓN (TEMAS DE CLASE QUE FUERON IMPLEMENTADOS)

Esquema de memoria



Ejecución de programas

Utilizamos el manejo de memoria dinámica asignando un espacio de memoria a cierto objeto que encuentras durante la historia. También para obtener vida o perderla.

Subprogramas

Los subprogramas son rutinas, procedimientos o conjuntos de instrucciones que realizan una labor específica, tienen

- Definición de subprograma: describe la interfaz de la abstracción del subprograma y sus acciones

```
void nombre() {
        string nombre;
        cout << "Escribe tu nombre por favor: ";
        cin >> nombre;
        introduc(nombre);
}
```

- Encabezado de subprograma: es la primera parte de la definición. Especifica la unidad sintáctica como subprograma, la lista de parámetros y el nombre (si no es anónimo).

```
void nombre();
void salir();
void introduc(string nomb_usu);
int ValidarEntrada();
```

- Perfil de parámetros: contiene el orden, número, y tipo de parámetros

```
(string nomb_usu)
```

- Protocolo de un subprograma: Es su perfil más el tipo de dato de retorno

```
void introduc(string nomb_usu)
```

Se usaron subprogramas para estructurar la historia. Cada escenario distinto está en un distinto subprograma y entre estos se van llamando y pasándole parámetros. Un ejemplo es el siguiente:

```
/**
*@brief Esta función es la segunda opcion del menu, es para finalizar el juego.
*@param no recibe parametros.
*@return regresa un void, es decir, no devuelve nungun valor.
*/
void salir(){
    system("cls");
    system("color 0A");
    cout<<"Gracias por jugar Odyssey Space\n";
}
```

Templates

Es un tipo de subprograma llamado genérico, es un solo subprograma a diferencia de la sobrecarga de funciones y las operaciones pueden realizarse sobre varios tipos de datos.

Creamos un subprograma genérico que nos permitiera trabajar con una cadena de texto y con números enteros, esto para mostrar al jugador la puntuación y desplegar en pantalla los finales alternativos.

```
template <typename anfibio>
void fin(anfibio final) {
    carac.puntos += 5;
        cout << final << endl;
}
```

Memoria dinámica

Utilizamos la memoria dinámica para crear un espacio para un objeto, este le servirá al jugador a obtener vida extra. Tenía que ser creado dinámicamente por que dependiendo de las decisiones del jugador este elemento iba o no a existir.

```
vida_extra = new int(35);
```

Manejo de eventos y excepciones

Cuando nuestros programas son notificados de ciertos eventos han sido detectados por software o hardware y nosotros reaccionamos y realizamos las acciones preventivas a estos errores.

Utilizamos la programación defensiva para capturar excepciones y validar las entradas para que no se introduzcan datos distintos a los solicitados.

```
        catch (const char* ErrorUno) {
                cout << ErrorUno << endl;
                inten++;
        }
        catch (int ErrorDos) {
                cout << "Seleccione la opcion 1 o 2" << endl;
                inten++;
        }
        catch(std::out_of_range){
            inten++;
    cout<<"Seleccione la opcion 1 o 2" << endl;
```

Aserciones

Assert compara la expresión y si es falsa aborta la ejecución del programa. Existen las: precondiciones que se refiere a las sentencias que se deben cumplir (verdaderas) antes de ejecutar una función, o método; y las postcondiciones que se refiere a las sentencias que se deben cumplir (verdaderas) después de ejecutar una función, o método.

Las aserciones las utilizamos para detener el programa en caso de haber un evento que no se puede cambiar, en nuestro caso lo utilizamos después de crear el objeto de vida extra de manera dinámica, por si no se asignaba de manera correcta se detenga y también que al ingresar más de 5 valores incorrectos para las decisiones

se detuviera como castigo y también que el usuario no pueda poner valores incorrectos de manera infinita.

```
acceso = ValidarEntrada();
if (acceso == 1){
    vida_extra = new int(35);
    assert(vida_extra != NULL);
```

```
do {
assert(inten!=5);
        cin >> eleccionCadena;
        int validacion = isValidInt(eleccionCadena);
```

Apuntadores

Existen situaciones donde los subprogramas se deben llamar indirectamente. Generalmente ocurre cuando no sabemos cuál programa llamar en tiempo de ejecución. En C y C++ es mediante apuntadores.

Usamos apuntador de funciones porque aún no sabíamos la primera decisión del usuario, así que este apuntador nos ayudó a llamar a la función según el número de entrada de la consola.

```
void (*fun[])()={nombre,salir};
```

```
acceso = ValidarEntrada();
fun[acceso-1]();
```

Anidamiento

El anidamiento se refiere al proceso de juntar o escribir estructuras/funciones dentro de un closure, es decir poner un for dentro de otro for o una función dentro de otra, existen diferentes tipos de anidamientos.

Lo utilizamos en la estructura, esta contiene dos funciones fundamentales que te ayuda a administrar las características del personaje, la de “quitar_vid” y “aum_vid”, estas dos funciones anidadas nos ayudan a interactuar con la vida. También utilizamos un anidamiento de if para la verificación del valor de entrada de la consola.

```
*@brief Contiene la vida y los puntos del jugador
*/
struct Personaje{
    int vida = 100;
    int puntos = 0;
    void quitar_vid(int cant) { vida -= cant;}
    void aum_vid(int cant) { vida += cant;}
}carac;
```

```
try {
        if (validacion != 0) {
                //string::size_type sz;
                eleccionEntero = stoi(eleccionCadena); //&sz
                if (eleccionEntero >= 1 && eleccionEntero <= 2) {
                        if (eleccionEntero == 1) {
                                return 1;
                        }
                        else {
                                return 2;
                        }
                }
                else {
                        throw 1;
                }
        }
        else {
                throw "Ingrese un entero";
        }
```

Concurrencia

La concurrencia te ayuda a: agrupar código similar, separar código sin relación a la tarea principal y a realizar programas más sencillos de entender y probar.

Nosotros lo utilizamos para generar un hilo con el reloj, esto con la finalidad de que el jugador pueda saber el tiempo que tardó en finalizar el juego.

```
void reloj(clock_t t)
{
    t = clock();
    return;
}
```

```
system( cls );
thread tiempo (reloj, t);
tiempo.join();
```

Bibliografía

cplusplus.com - The C++ Resources Network. (s.f.). Recuperado 30 mayo, 2019, de http://www.cplusplus.com/