TP1

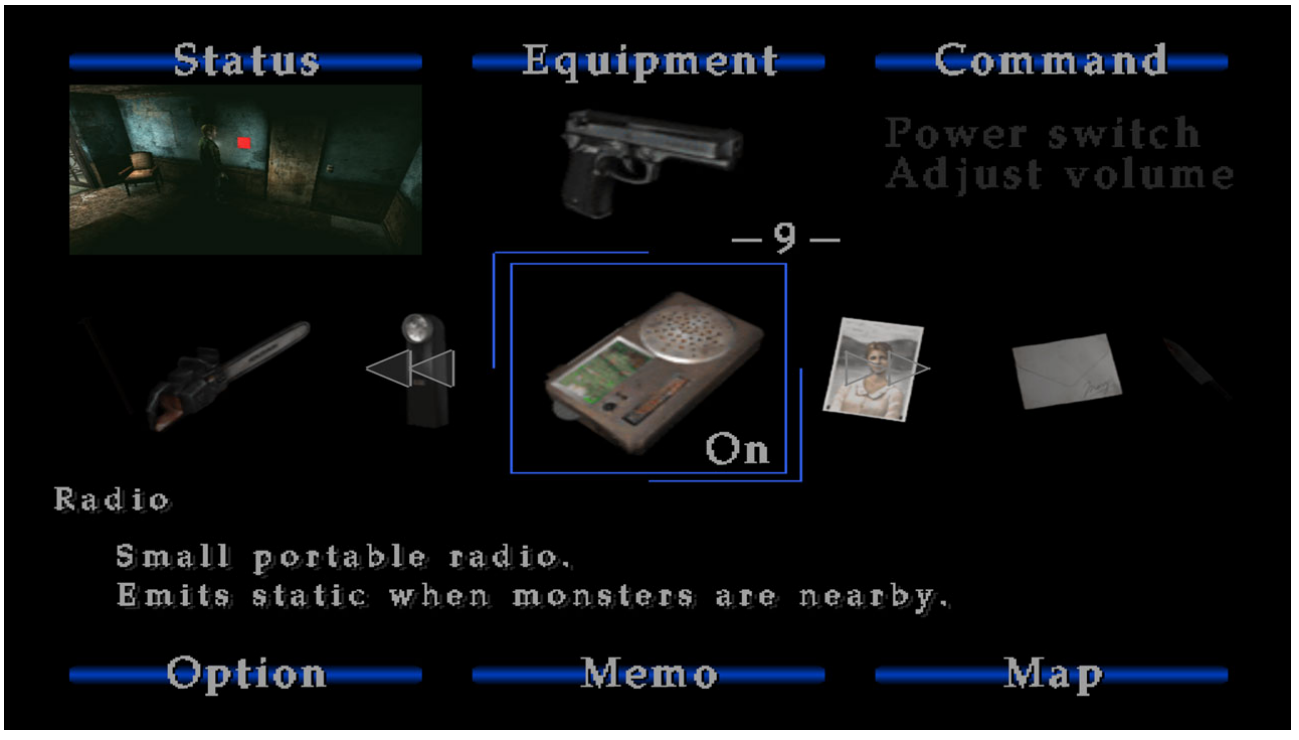
EL INVENTARIO DE JAMES

Introducción

James Sunderland es un oficinista estadounidense que un día recibe una extraña carta cuyo remitente era su esposa, Mary Sheperd-Sunderland. Esta misma decía que lo esperaba en "Su lugar especial" dentro del extraño pueblo de Silent Hill, ¿El problema? **La esposa de James lleva muerta tres años**. Con este tétrico contexto se nos introduce en la trama del famoso juego ***Silent Hill 2***.

James explorando Silent Hill luego de su llegada al pueblo

Por suerte para este trabajo no nos vamos a poner en los zapatos de James, sino en los de un programador en la empresa Konami. Esta se encarga del próximo remake del juego, ya que el original salió en de 2001. Nuestra responsabilidad va a ser la de desarrollar el inventario de James, controlado por el jugador, junto con una serie de funcionalidades.



Ejemplo del inventario In-game del jugador en la versión de PS2

Desde Konami nos comentan que el TDA que quieren utilizar para la tarea es un **Vector Dinámico**, ya que les da la posibilidad de expandir el tamaño del inventario en tiempo de ejecución (Es decir, mientras el jugador se encuentra corriendo el programa). No obstante, también debemos ser capaces de ofrecer una serie de funcionalidades que luego los programadores de otras áreas van a aprovechar para volver funcional nuestro TDA dentro del juego. Por eso es que para este trabajo **no debemos ofrecer experiencia de usuario amplia**, ya que justamente eso es trabajo de otra persona. Únicamente un breve pedido de instrucciones a través de un menú mediante el cual mostrarle a nuestros empleadores que el inventario funciona correctamente usando prints por consola.

Funcionalidades a implementar

Se debe implementar el **TDA Vector**, cumpliendo las firmas brindadas por la cátedra. Los métodos públicos del vector **NO DEBEN SER ALTERADOS**, aunque se pueden expandir los métodos privados.

Ademas, la lista completa de las funcionalidades que nos llegó del **Inventario** es la siguiente:

- **Alta:**

  El inventario debe ser capaz de agregar un ítem recogido por el jugador.

- **Baja:**

  El inventario debe ser capaz de eliminar ítems que el jugador decida usar.

- **Consulta:**

  El inventario debe ser capaz de listar su contenido.

- **Límite de ítems:**

  El inventario no debe permitir agregar un ítem si se excedieron los 15 elementos.

- **Cargado y guardado:**

  El inventario debe ser capaz de cargar datos sobre el inventario del jugador así como también generar un archivo *save file* con la información de este al cerrar el programa, *en una ruta especifica* **(*) Este punto se desarrolla más adelante.**

Creación y lectura de Save files

Como sucede en muchísimos otros juegos de un jugador, debemos ser capaces de ofrecer la posibilidad de guardar la partida. Los juegos hacen esto a través de archivos de guardado (Save files), los cuales tienen información sobre la partida (Items en el inventario, posición geográfica en el mapa, salud actual, arma equipada, etc). Por eso, nuestra responsabilidad como desarrolladores del inventario es ser capaces de tanto cargar como guardar los ítems que el jugador haya recolectado mientras jugaba. Esto lo vamos a hacer a través de archivos **.csv** los cuales deberán tener el formato indicado por Konami. El formato mencionado es:

Nombre_Item,Tipo_Item
Nombre_Item,Tipo_Item
Nombre_Item,Tipo_Item
[...]

Los tipos confirmados para esta nueva entrega son:

1. *Curativos* (Botiquín, Bebida curativa, etc)
2. *Munición* (Vienen en cajas de 30 y son de un tipo de arma específico)
3. *Puzzle* (Son llaves, herramientas o elementos que permiten progresar al jugador)

Por ejemplo:

LlaveEscaleraTerraza,PUZZLE
MunicionRevolver,MUNICION
Botiquin,CURATIVO
Destornillador,PUZZLE

Al iniciarse, el inventario **debe cargarse con los ítems especificados en su save file**, y *se le debe especificar la ruta deseada*. Puede ser ingresada por el usuario o simplemente una ruta constante declarada previamente.

Items

Los ítems son parte del proceso de desarrollo de otro equipo, que tuvo la amabilidad de pasarnos los archivos asociados a esta **clase.** Nosotros debemos incorporar objetos de tipo **ítem** dentro de nuestro inventario, los cuales se **construyen** con la información leída del save file (O cargada por el tester al correr nuestro programa).

La interfaz de la clase **Ítem** es:

1. *Item(string nombre, string tipo)*: Constructor.
2. *void listarInformacion()*: Muestra por pantalla la información del ítem.
3. *operator<<()*: Utilizar este método (archivo << item) para cargar la información al archivo .csv.
4. o*perator==(string nombre):* Devuelve true si el nombre coincide.

También declara constantes que se recomiendan utilizar:

1. const string *TIPO_PUZZLE* = **"PUZZLE"**
2. const string *TIPO_MUNICION* = **"MUNICION"**
3. const string *TIPO_CURATIVO* = **"CURATIVO"**

Ejemplo de ejecución del programa

```
//Consulta:
"Acción sobre el inventario: " CONSULTA

"1: LlaveEscalera, Tipo: PUZZLE
 2: Botiquín, Tipo: CURATIVO
 3: MuniciónRevolver, Tipo: MUNICION"

//Baja:
"Acción sobre el inventario:" BAJA

"Indique el ítem a eliminar: " Botiquín
"Acción sobre el inventario:" CONSULTA

"1: LlaveEscalera, Tipo: PUZZLE
 2: MunicionRevolver, Tipo: MUNICION"

"Indique el ítem a eliminar: " Botiquín
"Error de inventario! No se dispone de ese ítem"

//Alta:
"Acción sobre el inventario: " ALTA

"Nombre del ítem: " Botiquín
"Tipo del ítem " CURATIVO

"Acción sobre el inventario:" CONSULTA

"1: LlaveEscalera, Tipo: PUZZLE
 2: MunicionRevolver, Tipo: MUNICION
 3: Botiquín, Tipo: CURATIVO"
```

El anterior es un ejemplo de la ejecución esperada del programa. Servirá como una prueba del correcto funcionamiento del trabajo una vez terminadas todas las funcionalidades.

Tests y primitivas

Para este trabajo practico, contaran con un repositorio donde encontraran:

- La clase **Ítem**, con la cual deberán implementar su Inventario.
- La firma del Vector (Vector.hpp). Como se mencionó anteriormente, los métodos públicos no deben ser alterados.
- Los tests ofrecidos por la cátedra, que probaran su TDA Vector. De no pasar todas las pruebas, **la entrega queda desaprobada inmediatamente**. En la branch “template” hay tests adaptados para un TDA Template. Los alumnos que adapten el vector para ejecutar esta variante recibiran **2** puntos extra.

Para ir al repositorio, hace clic aquí.

Se recomienda clonar o descargar el repositorio y comenzar a trabajar a partir de ahí.

Aclaraciones y criterios de evaluación

Para que el trabajo sea aceptado por nuestros superiores en Konami, este tiene que cumplir las siguientes condiciones:

- Debe implementar el TDA de manera **totalmente original y propia sin el uso de bibliotecas como STD <Vector>**
- No debe perder memoria al correrlo con Valgrind
- Debe correr los tests ofrecidos por la cátedra y también respetar las firmas con las que se realizaron estos
- No se deben subir archivos de configuración de los IDEs (solo subir .cpp, .h / .hpp y CMakeLists.txt).
- El formato de separación de líneas debe ser LF (Revisar configuración del IDE)

Los criterios de evaluación por parte nuestra serán:

- Compilación: sin warnings ni errores.
- Funcionalidad.
- Eficiencia espacial.
- Eficiencia temporal.
- Buenas prácticas de programación (nombres descriptivos, indentación, etc.).
- Modularización.
- Precondiciones y postcondiciones.
- Uso de memoria dinámica.

Normas de entrega

Se deberá subir al campus un único archivo comprimido **(.zip)** en la sección TPs.

Este archivo deberá tener un nombre formado de la siguiente manera:

Padron_Apellido_TP1

La fecha de entrega vence el **Domingo 1/10 23:59**.

Puntaje: **30** puntos.

Música para ponerse en clima con la temática: https://www.youtube.com/watch?v=oBktZDfm0uY&ab_channel=MelodicMusicExtension