UNIVERSIDAD DE SAN CARLOS DE GUATEMALA

FACULTAD DE INGENIERÍA

ORGANIZACIÓN DE LENGUAJES Y COMPILADORES 2

PRIMER SEMESTRE 2013

Proyecto 1

Objetivos:

Objetivo General

- Aplicar conceptos generales de compiladores en una aplicación práctica.

Objetivos específicos

- Reafirmar los conceptos básicos de Compiladores.
- Hacer uso de la biblioteca log4j para logging de errores.
- Construir una aplicación que tenga como objetivo el aprendizaje de conceptos básicos de programación orientada a los niños.

Definición del problema

Una empresa independiente le contrata a usted para que desarrolle un videojuego para lanzarlo al mercado, la empresa desea que la principal característica del videojuego sea que el mismo sea completamente personalizable, de tal manera que los usuarios finales puedan definir la forma del tablero, los atributos del jugador y de los enemigos, y sus patrones de movimiento. El objetivo del juego es que el jugador atraviese una serie de laberintos infestados por enemigos, y logre llegar a la casilla de salida para ganar el laberinto, al llegar al último laberinto el jugador habrá ganado el juego.

El software contará con una interfaz, la cual debe ser amigable para el usuario y de fácil uso, en la cual se cargarán archivos de configuración del juego y este los interpretará para mostrar al usuario la pantalla del juego, con el tablero y configuraciones indicadas en el archivo.

Interfaz

La aplicación deberá mostrar un menú para ingresar los archivos de personaje, mapa y enemigos, luego el usuario deberá seleccionar la opción de iniciar, y se mostrará el mapa y personaje, junto a sus enemigos, la aplicación permitirá que el personaje se mueva a través del mapa y peleé contra los enemigos, el flujo será entonces:

- El usuario selecciona uno o más archivos de nivel
- El usuario selecciona uno o más archivos de enemigos
- El usuario selecciona un archivo de personaje
- El usuario elige la opción de iniciar juego
- Se mostrará gráficamente el mapa, enemigos y personaje permitiendo que se muevan a través del mapa.
- Cuando el jugador encuentra la salida del último nivel cargado se considerará ganador y deberá mostrarse un mensaje en pantalla.

Editor

Se debe contar con un editor de código para la edición de archivos propios de la aplicación, el cual contará con una opción de revisión que permitirá mostrar los errores del archivo abierto, además contará con las siguientes opciones:

- Abrir: abre un archivo de cualquiera de los tres lenguajes descritos anteriormente, reconociendo automáticamente qué tipo de archivo es, de acuerdo a las extensiones indicadas posteriormente.
- Guardar: guarda el archivo que se está trabajando, si el archivo no se ha guardado anteriormente, entonces funciona de manera similar a guardar como.
- Guardar Como: guarda el archivo como un nuevo archivo con el nombre especificado, se debe poder indicar qué tipo de archivo se desea almacenar, si de personaje, de nivel o de enemigo.
- Revisar errores: revisa errores de sintaxis y semánticos en el archivo actual.

NOTA: No se permite el uso de ninguna aplicación externa para la edición del archivo, debe ser posible editar cualquiera de los tres tipos de archivo desde la aplicación.

Creación de nueva sesión de juego

Se debe proporcionar una interfaz sencilla que permita agregar niveles, enemigos y al personaje, las indicaciones de estos archivos, así como el lenguaje de cada uno, se encuentran más adelante, el diseño de la interfaz quedará a discreción del estudiante, sin embargo debe contar con las opciones mencionadas.

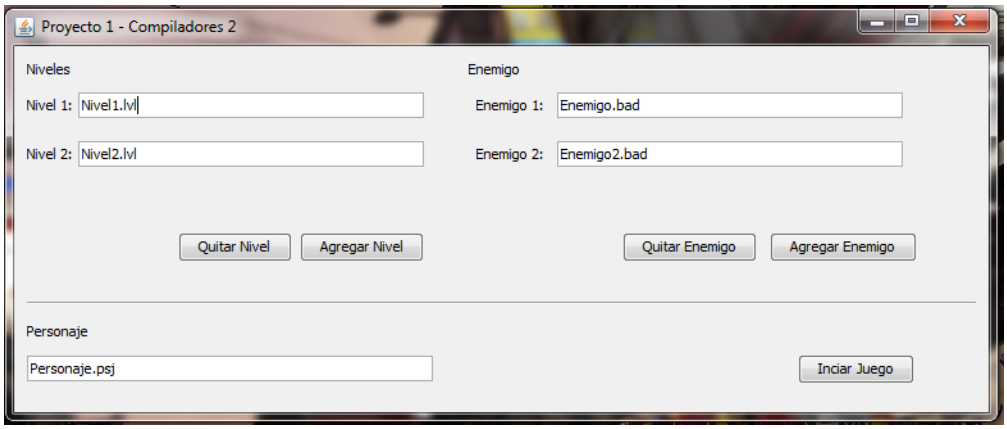


Imagen de muestra del menú de inicio

El juego

La aplicación deberá proveer una plataforma que ejecutará el juego con una perspectiva top down, o vista de águila, en el cual se presentará el laberinto, el personaje y los enemigos. El jugador tendrá una serie de atributos, los cuales serán definidos más adelante, así como la estructura del nivel. Los enemigos, sus atributos y su patrón de comportamiento también serán definidos a través de una serie de archivos, los cuales comenzarán a ejecutarse desde el momento en que inicie el juego.

Si se ingresan varios archivos de nivel, éstos deberán ejecutarse de manera lineal, es decir, el primer archivo será el nivel uno, cuando el jugador pase este nivel, deberá cargarse el segundo archivo, el cual será el nivel dos, y así sucesivamente, en caso de que el jugador pase el último nivel ingresado deberá mostrarse un mensaje de felicitaciones, con el nombre del personaje, si es que este tuviese nombre (Ver sección, Archivo del jugador).

Archivo de nivel

El archivo de nivel especificará una pantalla o nivel que el jugador deberá cruzar para poder terminar el juego, el archivo de nivel especificará el inicio del jugador, de los enemigos, salidas y otros espacios especiales. El archivo de nivel tendrá una extensión .lvl.

Estructura de un nivel

Un nivel será una cuadrícula de n x m cuadros, cada cuadro representará un espacio que tendrá un efecto sobre el personaje o enemigo, los tipos de espacio no están pre definidos, estos espacios tendrán que ser definidos por el usuario dentro del archivo de nivel.

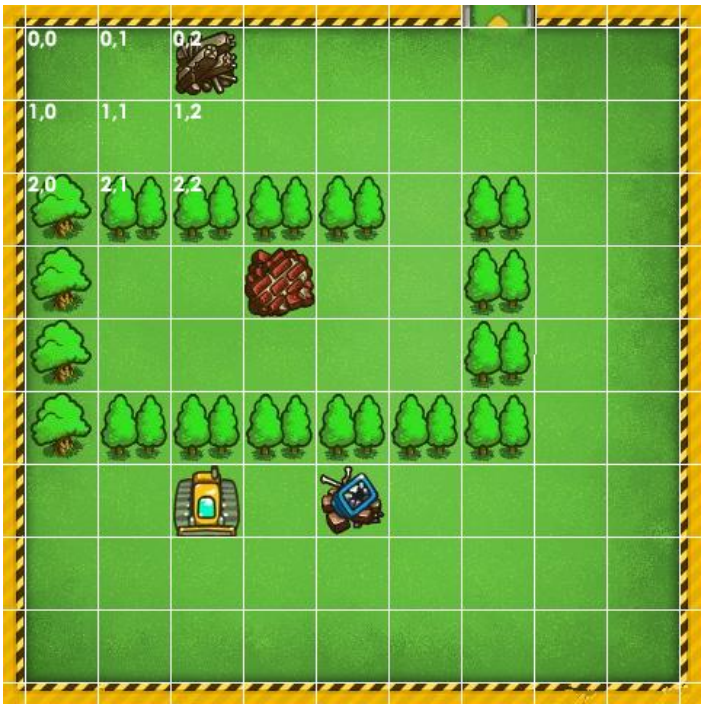


Imagen 1: Estructura de un nivel de 9 x 9

Restricciones de la estructura de un nivel

Un nivel debe contar con las siguientes casillas:

- Un sólo lugar para que inicie el jugador, si se define más de uno es considerado un error de semántica.
- N lugares para que inicien los enemigos, todos los campos de inicio de enemigos se llenarán, independientemente de cuantos archivos de enemigo se definiesen, si se define más de un archivo de enemigos, el algoritmo de llenado será Round Robin, es decir, la primer casilla de enemigo tendrá un enemigo del primer archivo, la segunda del segundo y así sucesivamente, al acabarse los archivos de enemigos y aún haber casillas para inicio de enemigos se volverá a crear un enemigo del primer archivo.
- Al menos una casilla de mejora, en la que se le aumentará algún poder al jugador, este puede ser más vida o más magia.
- Al menos una casilla de salida, cuando el jugador toca esta casilla habrá pasado el nivel.

Estructura del archivo de nivel

```
<Importar>
</Importar>
<Espacios>
   //declaración de espacios posibles
</Espacios>
<Nivel>
 //declaración de los espacios que conforman el nivel
```

```
</Nivel>
```

La estructura del archivo de nivel tendrá tres áreas:

- El área de <Importar>, en esta área se podrán indicar otros archivos de extensión .lvl para cargar los espacios definidos en ese archivo y poder utilizarlos en el archivo actual. Esta área es opcional
- El área de <Espacios>, donde se definirán los posibles espacios que puedan utilizarse en el archivo, y su efecto sobre el personaje. Esta área es opcional, pero si no se importó ningún archivo y no se define ningún espacio, entonces no se podrá crear el nivel.
- El área de <Nivel>, en esta área se definirá el tamaño del nivel, así mismo se especificará que tipo de espacio es cada cuadro de la cuadrícula. El archivo de nivel principal debe contener esta sección, sin embargo existe un archivo de nivel especial que define únicamente espacios, este archivo se especifica en la sección "Archivo de nivel especial".

Definición del lenguaje del archivo de nivel

Sección Importar

En la sección importar se colocarán los archivos de otros niveles a importar con la siguiente sintaxis:

```
<archivo> Direccion/del/archivo </archivo>
```

Al importar un archivo estarán disponibles los espacios definidos en dicho archivo para utilizarlos en el archivo actual. Por ejemplo:

```
<Importar>
<archivo> /home/Pablo/niveles/espacios.lvl </archivo>
</Importar>
```

NOTA: es posible importar más de un archivo.

Sección Espacios

La sección espacios sirve para definir los tipos de casilla que se podrán utilizar en el nivel actual, para esto se utilizará la siguiente sintaxis:

```
<Espacio>
 <Extiende> NombreDelEspacioPadre </Extiende>
 <Nombre>NombreDelEspacio </Nombre>
 <Pasable> verdadero/falso </Pasable>
 <Especial> Atributo que aumenta </Especial>
 <Imagen>/direccion/de/la/imagen </Imagen>
 <Inicio> verdadero/falso </Inicio>
 <Enemigo> verdadero/falso </Enemigo>
```

```
 <Fin> verdadero/falso  </Fin>
</Espacio>
```

Donde cada etiqueta representa:

- Extiende: indica el nombre de otro tipo de espacio que será el padre del tipo actual, reemplazando los valores por defecto por los del tipo padre. Valor por defecto, nulo.
- Nombre: El nombre del tipo de espacio que se está definiendo, este será utilizado para indicar que casillas del mapa son de este tipo.
- Pasable: Indica si el tipo de espacio será transitable, es decir si el jugador y los enemigos podrán pasar sobre él. Valor por defecto, verdadero.
- Espacial: Indica si el espacio aumenta o disminuye alguna de las capacidades del jugador, en la siguiente sección se indican los tipos de atributos que puede aumentar. Valor por defecto, nulo.
- Imagen: Indica la dirección de la imagen a utilizar para las casillas de este tipo.
- Inicio: Booleano que indicará si la casilla es o no el inicio del nivel, cada nivel puede tener solamente una casilla de tipo inicio. Valor por defecto, falso.
- Enemigo: Indica si la casilla es inicio de algún enemigo. Puede existir cualquier número de casillas de enemigo, inclusive no existir casillas de enemigos para algún nivel. Valor por defecto, falso.
- Fin: Indica si la casilla es el final del nivel, cada nivel puede tener solamente una casilla de tipo fin. Valor por defecto, falso.

NOTA: todos los atributos de la etiqueta <Espacio>, excepto el nombre e imagen, son opcionales, en cuyo caso deberá utilizarse su valor por defecto.

Etiqueta especial

La etiqueta especial indica que la casilla tiene algún efecto sobre el jugador cuando éste pasa sobre la casilla, estos efectos pueden ser:

- <dano>número</dano>: le hace una cantidad de daño al jugador
- <cura> número </cura>: le cura una cantidad de vida al jugador
- <magia> número <magia>: aumenta su poder mágico
- <invencibilidad></invencibilidad>: hace al jugador invencible por cierto tiempo, el tiempo queda a discreción del estudiante.

Ejemplos de sección de espacios

```
<Espacios>
<Espacio>
 <Nombre>normal </Nombre>
 <Pasable> verdadero </Pasable>
 <Especial> </Especial>
 <Imagen>/home/Pablo/grama.png </Imagen>
 <Inicio> falso </Inicio>
```

```
 <Enemigo> falso </Enemigo>
 <Fin>falso</Fin>
</Espacio>

<Espacio>
 <Nombre>FuenteVida </Nombre>
 <Pasable> verdadero </Pasable>
 <Especial> <cura>10</cura> </Especial>
 <Imagen>/home/Pablo/fuente.png</Imagen>
 <Inicio> verdadero </Inicio>
 <Enemigo> falso </Enemigo>
 <Fin>falso</Fin>
</Espacio>
</Espacios>
```

Sección <Nivel>

En la sección nivel se definirá la estructura del nivel, la sintaxis de esta sección es:

```
<Nivel>
 <Nombre>NombreDelNivel</Nombre>
 <Tamano>
  <X>número</X>
  <Y>número</Y>
 </Tamano>
 <Estructura>
  <Casilla>
   <X>número</X>
   <Y>número</Y>
   <Tipo>NombreDeEspacio</Tipo>
  </Casilla>
  <Casilla>
   <X>número</X>
   <Y>número</Y>
   <Tipo>NombreDeEspacio</Tipo>
  </Casilla>
  .
  .
  .
 <Estructura>
</Nivel>
```

Donde:

- **Nombre**: indica el nombre del nivel
- **Tamaño:** sus etiquetas hijo indican el tamaño de casillas en X y Y que conforman el nivel.
- **Estructura:**

- **X**: indica la posición X de la casilla en la cuadrícula, para referencia ver la imagen 1.
- **Y**: indica la posición Y de la casilla en la cuadrícula, para referencia ver la imagen 1.
- **Tipo**: indica el tipo de la casilla, el tipo debe ser un tipo definido anteriormente en la sección espacios o importado para poder ser utilizado en el nivel.

NOTA: El archivo debe contener definición para todos los espacios, de lo contrario deberá mostrarse un error semántico. Por ejemplo si es de 10 * 10 deberá contar con 100 definiciones de casilla.

Archivo de nivel especial

Se puede dar el caso de que un archivo de nivel no cuente con área de <Nivel>, en este caso el archivo debe tener la estructura mostrada en el ejemplo siguiente:

```
[<Importar>
//definición de imports
</Importar>]
<Espacios>
//definición de espacios
</Espacios>
<Nivel></Nivel> //sección vacía
```

NOTA: Los corchetes [] indican opcionalidad.

Archivo de Jugador

El archivo de jugador especificará las características del jugador como la vida, capacidad de movimiento, magia, entre otros especificados a continuación. El archivo de jugador tendrá una extensión .psj.

El jugador

El personaje será la representación del jugador dentro del juego, el objetivo es llevar al personaje desde la casilla de inicio hasta la casilla del final, el jugador cuenta con las siguientes características predefinidas.

- Moverse: El personaje podrá moverse utilizando las flechas direccionales, el movimiento se realizará por casillas. Las acciones de cada tecla se definen a continuación:
  - Flecha arriba ↑: moverá al personaje una casilla hacia el frente (dirección en la que esté observando actualmente).
  - Flecha abajo ↓: moverá al personaje una casilla hacia atrás (dirección contraria a la que esté observando actualmente).
  - Flecha derecha →: girará al personaje 90 grados a favor de las agujas del reloj a partir de la dirección actual.

    - Flecha izquierda ⟵: girará al personaje 90 grados en contra de las agujas del reloj a partir de la dirección actual.
- Atacar: El personaje podrá atacar cuando el usuario presione la tecla "A", al realizar esto el daño del personaje se le restará de los puntos de vida del enemigo que tenga al frente.
- Magia: El personaje podrá utilizar un ataque mágico cuando el usuario presione la tecla "S", al realizar este ataque, se lanzará una bala mágica en línea recta desde donde está viendo el personaje, esta bala mágica deberá continuar viajando hasta que salga del mapa, choque contra una casilla impasable, o golpee a un enemigo. En el caso de golpear a un enemigo se deberá restar el poder mágico de la vida del enemigo.

Definición del lenguaje del archivo jugador

El archivo del jugador definirá únicamente los atributos del personaje que será controlado por el jugador, la sintaxis general del archivo es:

```
Iniciar Personaje
 Valor:Atributo
 Valor2:Atributo2
 .
 .
 .
Finalizar Personaje
```

Los atributos del personaje pueden ser:

- Nombre: indica el nombre del personaje, éste se mostrará en el mensaje de felicitación en caso se logren pasar todos los niveles.
- Vida: indica la vida del personaje, la vida inicial del personaje debe ser mayor a cero, y cuando esta alcanza un valor menor o igual a cero, el personaje pierde el juego.
- Fuerza: indica cuánta vida pierden los enemigos al ser golpeados por el ataque (A) del personaje.
- Magia: indica con cuanto poder mágico cuenta el personaje, cada ataque mágico (S) gasta una cantidad de magia.
- Fuerza_Magica: indica cuánta vida pierden los enemigos al ser golpeados por el ataque mágico (S) del personaje.
- Costo_Magico: indica cuánta magia pierde el personaje cada vez que realiza un ataque mágico.

Ejemplo de un archivo de jugador:

```
Iniciar Personaje
 Thral:Nombre
 100:Vida
 25:Fuerza
 100:Magia
```

```
 50:Fuerza_Magica
 20:Costo_Magico
Fin Personaje
```

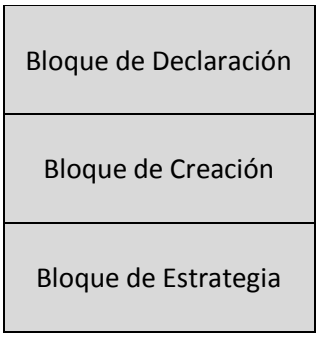
NOTA: Todos los atributos son obligatorios. Debe existir un indicador en la pantalla que muestre la vida y poder mágico del personaje.

Archivo de Enemigo

El archivo de enemigo especificará los atributos de un enemigo, así como su patrón de movimiento. Los atributos de un enemigo son: su vida y su capacidad de ataque. El archivo de enemigo tendrá una extensión .bad.

Definición del lenguaje del archivo de enemigo

Un archivo de enemigo debe seguir el siguiente esquema de lenguaje:



Comentarios

Para tener mejor comprensión de la programación se pueden utilizar comentarios, de la siguiente manera.

Tipo de comentario	Inicio	Fin	Ejemplo
Comentario de Línea	---	---	--- Este es un comentario de una línea
Comentario de Bloque	---*	*---	---* Este Es un comentario De varias líneas *---

El fin de los comentarios es documentar las actividades en lenguaje.

Inicio de Bloque

Los bloques iniciarán con una etiqueta representativa seguida de dos puntos, de la siguiente manera:

Etiqueta	Bloque que anuncia
BD:	Bloque de declaración
DC:	Bloque de creación
DE:	Bloque de estrategia

Un ejemplo se muestra en el siguiente archivo *Ejemplo.bad*:

```
BD:
        --- Aquí van las instrucciones de declaración
CD:
        --- Aquí van las instrucciones de creación
DE:
        --- Aquí van las instrucciones de estrategia
```

Ejemplo1.bad

Bloque de Declaración

En esta sección se declarán las variables a ser utilizadas dentro de los métodos y funciones, la forma de declaración es la siguiente:

Comandos Utilizables:

addf(nombre_var, tipo_var, valor_inicial)

Esta instrucción crea una variable nueva, con un identificador y tipo único, es decir no pueden existir dos variables con el mismo nombre y el mismo tipo, pero si pueden existir dos o más variables con el mismo nombre pero con diferente tipo.

Por Ejemplo:

```
BD:
        Addf(variable1,Integer,0)
        Addf(variable1,boolean,false)

        ……..
```

Definición parámetros:

nombre_var:	cadena de texto que identificará de forma única a la variable dentro de la tabla de símbolos.
tipo_var:	cadena de texto que identificará el tipo de la variable
Valor_inicial:	valor opcional que representa el valor que tendrá inicialmente esta variable (tomar en cuenta que este puede o no venir).

Tipos Aceptados:

String	Cadena de texto de longitud máxima 256
Boolean	Valor entero que representa 1=verdadero y 0= falso, también se pueden utilizar las palabras reservadas *true* y *false* que equivalen al 1 y 0 respectivamente
Integer	Valor entero del -16599999 al 16599999
Long	Valor entero del -165999990000 al 165999990000
Char	Cadena de longitud 1 (un solo carácter)
Float	Valor numérico con punto decimal

NOTA: De no asignar un valor inicial se deberá hacer uso del comando setf (se explicará posteriormente) para asignarle un valor a esta variable. La variable no podrá ser utilizada si no ha sido inicializada previamente.

Bloque de Creación

En este lugar se procederá a crear los diversos objetos para un enemigo, los cuales serán pintados o utilizados en el tablero de juego.

Comandos Utilizables:

crearImagen(id_imagen, nombre, ruta)

Cargará una serie de imágenes al sistema, haciendo un repositorio de imágenes las cuales podrán ser utilizadas en los distintos escenarios y personajes del juego.

Definición parámetros:

Id_imagen:	valor numérico que identificará de forma única a la imagen a cargar.
Nombre:	nombre para esta imagen, el mismo será desplegado en la parametrización, deberá delimitarse por comillas dobles. Ej.: “mi Nombre”.
Ruta:	ruta física o path de la imagen (en dónde será guardada localmente en disco), deberá delimitarse por comillas dobles. Ej.: “mi Ruta”

crearPotencia(id_potencia, nivel)

Designa los diferentes tipos de potencia que hay, lo que representa que tan agresivo será el enemigo y la frecuencia con que realizará el ataque.

Definición de parámetros:

id_potencia: Número correlativo que representará el nivel de potencia.

Nivel: cadena que representa los distintos niveles de la siguiente forma:

Tipos de Niveles aceptados:

Tinny	pasivo, no genera mayor daño al jugador
Low	bajo, genera un daño leve
Medium	normal, genera un daño considerable
Hight	alto, genera un daño mayor
Heavy	muerte (también denominado kill), mata instantáneamente al jugador.

crearArma(id_arma, nombre , max, min, no_municiones)

Crea un catálogo de armas disponibles para que pueda utilizarlas el enemigo,

Definición de parámetros:

id_arma:	identificador numérico único del arma.
nombre:	cadena de texto que representa el nombre a mostrar en la interfaz para esta arma
max:	valor máximo de municiones o eventos de disparo que posee, al llegar a este número de municiones se deberá mostrar en pantalla un mensaje.
min:	número menor de municiones o eventos de disparo que posee, después de este número el arma queda inutilizable.
no_municiones:	número de municiones o detonaciones disponibles que puede realizar esta arma (conforme se utilice este número debe disminuir hasta llegar a 0 o al número mínimo definido *min*).

NOTA: Se deberá implementar un warning (mensaje de advertencia), anunciando si el nivel del arma está bajo (esto para que el contrincante tenga una ventaja visual sobre el enemigo ya que este es un buen momento para que el enemigo sea atacado)

crearEnemigo(id_enemigo, nombre, id_potencia, id_imagen)

Crea un nuevo enemigo, en el tablero pueden existir *n* número de enemigos con *n>=1*.

Definición de parámetros:

id_enemigo:	identificador numérico único del enemigo.
nombre:	cadena de texto que representa el nombre del enemigo a mostrar en la interfaz.
id_potencia:	potencia asignada a este enemigo
id_imagen:	identificador único de la imagen que se le asignará a este enemigo.

crearEstrategia(id_estrategia, nombre, punteo)

Una estrategia es un movimiento que seguirá el enemigo dentro del tablero, con movimiento nos referimos a todas las acciones que este realizará de forma autónoma, una estrategia puede ser de evasión, de ataque, de huida, etc.

Definición de parámetros:

id_estrategia:	identificador único de la estrategia
nombre:	nombre de la estrategia (evasión, ataque, huida, etc. Estas son definidas por el usuario)
punteo:	cada estrategia tendrá un punteo, este será la suma en milisegundos del tiempo que dure la vida de la estrategia hasta que el enemigo sea eliminado, esto se mostrará en un reporte para que el usuario pueda elegir la mejor estrategia, la mejor estrategia será la que tenga mayor tiempo de vida dentro del juego.

Bloque de Estrategia

En este lugar se procederá a tramar la lógica de juego del enemigo, es decir el comportamiento que tendrán los enemigos para accionar contra los jugadores de forma automática.

Este bloque debe estar ordenado lógicamente, es decir: para que una estrategia tenga resultados (incremente o decremento su punteo) se debe generar las sentencias correctas y optimas para que el enemigo cumpla su función en contra del jugador.

A continuación definimos las distintas opciones disponibles.

Condiciones, Bucles y Controles:

Condiciones

Las condiciones son comparaciones que se realizan entre pares de variables o valores, esta evaluación en global da como resultado un valor booleano. Las condiciones a evaluar son:

Nombre	Símbolo	Evento	Observaciones
Igualación	=	Compara dos valores o variables si son iguales retorna 1, de lo contrario 0	
Diferenciación	¡=	Evalúa que dos valores o variables sean diferentes retorna 1 en caso de diferencia y 0 en caso contrario	
Mayor que	>	Evalúa que un valor o variable sea mayor a otro en comparación	Se puede combinar con el símbolo = Ej.: >=
Menor que	<	Evalúa que un valor o variable sea menor a otro en comparación	Se puede combinar con el símbolo = Ej.: <=
Es nulo	¡!	Evalúa si una variable fue declarada pero no le fue asignado un valor inicial o una asignación posterior.	

Nexos Lógicos

Se utilizan para unir una pareja de condición, los nexos lógicos son los siguientes:

Nombre	Símbolo	Evento	Observaciones
AND	&&	Compara que dos condiciones sean iguales a 1 si éste es el caso retorna 1, de lo contrario retorna 0	
OR	||	Evalúa que al menos una de dos condiciones sea igual a 1 si este es el caso retorna 1, de lo contrario retorna 0	
XOR	&|	Compara que dos condiciones sean iguales en este caso retorna 1, de lo contrario retorna 0	1=1 retorna 1 0 = 0 retorna 1 0 En los demás casos
NOT	¡	Cambia el valor de la condición a su valor negado.	

Bucle Básico (While)

mientras <condición>
// Instrucciones
fin_mientras

Descripción: Realizará las instrucciones dentro del bucle mientras la condición especificada se cumpla. Si esta condición nunca se cumple o se desea terminar el bucle de manera temprana se utilizará la instrucción stop.

Definición:

<condición> Condición que tiene como resultado un valor booleano.

// Instrucciones Serie de instrucciones de estrategia a ejecutar.

Instrucciones de stop:

Esta instrucción detiene la ejecución del bucle (se sale del bucle) de forma inmediata y sin corroborar que se cumpla la condición de salida, las instrucciones de stop son:

Detener: Detiene el bucle y se sale inmediatamente.

Pausar(valor_var): Detiene el bucle temporalmente hasta que se cumpla el tiempo en *valor_var* expresado en milisegundos, este tiempo también puede ser una variable previamente definida.

Bucle Limitado (For)

para nombre_var1=valor_inicial,***aumentar***(nombre_var1)=valor_final ***hacer***
// Instrucciones
fin_para

para nombre_var1=valor_inicial,***disminuir***(nombre_var1)=valor_final ***hacer***
// Instrucciones
fin_para

Descripción: Realiza un bucle arrancando desde un valor inicial hasta completar un valor final, o hasta encontrar una instrucción de stop.

Definición:

Nombre_var1: nombre de la variable de control sobre el bucle, la cual deberá ser previamente declarada.

valor_inicial: Valor en donde iniciará el bucle, este valor podrá ser una variable previamente declarada o un simple valor numérico, en ambos casos deberá ser de tipo entero

valor_inicial: Valor en donde finaliza el bucle, este valor podrá ser una variable previamente declarada o un simple valor numérico, en ambos casos deberá ser de tipo entero

Aumentar(nombre_var1): define que la variable declarada será incrementada en 1 en cada iteración del bucle.

Disminuir(nombre_var1): define que la variable declarada será decrementada en 1 en cada iteración del bucle.

// Instrucciones Serie de instrucciones de estrategia a ejecutar.

Control

si <condición> **entonces**
// Instrucciones
otro_caso <condición>
// Instrucciones
otro_caso
// Instrucciones
fin_si

Descripción: Evalúa si una condición cumple o no, permitiendo que se ejecute una u otra parte de la estrategia, funciona a manera de filtro ya que al final solo un bloque podrá ser ejecutado.

Definición:

<Condición>	Condición que tiene como resultado un valor booleano.
// Instrucciones	Serie de instrucciones de estrategia a ejecutar.
Otro_caso <condición>	Si no se cumplió el caso anterior evaluará una nueva <condición>, esta parte es opcional, es decir puede no venir.
Otro_caso	En caso de no cumplir con ninguno de los casos anteriores, se procederá a ejecutar este bloque, al igual que la instrucción anterior esta instrucción es opcional, es decir podría no venir.

Comandos Utilizables

setf(nombre_var, valor)

Le asigna un valor a una variable previamente definida (si la variable ya cuenta con un valor, este se reemplazará por el nuevo valor) y esta podrá ser utilizada en las siguientes instrucciones.

Definición de parámetros:

nombre_var:	Nombre de la variable previamente declarada sobre la cual se realiza la asignación del nuevo valor.
Valor:	Nuevo valor que será asignado a la variable.

NOTA: Se deberá validar que la variable exista y que el valor sea del tipo correcto para que el casteo sea satisfactorio. Siguiendo la siguiente definición:

Tipo Variable	Tipo Valor	Resultado del Casteo
Entero	Entero	Entero
Entero	String	Error
Entero	Booleano	Entero

Entero	Float	Float
Entero	Char	Entero ASCII
String	<Cualquier otro Tipo>	String
Booleano	<Cualquier otro Tipo>	Booleano
Float	Entero	Float
Float	Float	Float
Float	String	Error
Float	Booleano	Error
Char	Entero	ASCII
Char	Char	Char
Char	<Cualquier otro Tipo>	Error

Las instrucciones que se pueden utilizar se dividen en Funciones y Procedimientos.

Funciones

Son utilizadas para obtener datos específicos del tablero y para realizar operaciones de casteo sobre variables, las funciones a implementar son:

func boolean getMuniciones(id_enemigo)

Descripción: Devuelve el numero de municiones que tiene actualmente el enemigo.

Resultado: Valor Entero

func boolean getLibre(id_enemigo, posicion_x, posicion_y)

Descripción: Devuelve si la casilla en el tablero delimitada por las coordinadas (posicion_x, posicion_y) está libre para un enemigo en particular. Se debe verificar que esta casilla sea inicio para un enemigo.

Resultado: Valor booleano

func boolean bordeTablero(id_enemigo)

Descripción: Devuelve si el enemigo se encuentra en el borde del tablero, revisando sus cuatro puntos cardinales (norte, sur, este, oeste)

Resultado: Valor booleano

func getVal(nombre_var)

Descripción: Devuelve el valor actual que tiene la variable previamente declarada.

Resultado: Valor del tipo de variable declarada.

func string getf(nombre_var)

Descripción: Devuelve el valor actual en string que tiene la variable previamente declarada.

Resultado: Cadena de texto

func boolean getBool(valor)

Descripción: Castea el contenido de Valor a su respectivo valor booleano, por ejemplo getBool("true") retornara true

Resultado: Valor booleano

func string getStr(valor)

Descripción: Castea el contenido de Valor a su respectivo valor String, por ejemplo getStr(true) retornara "true"; getStr(1) retornara "1"

Resultado: Cadena de Texto

func integer getInt(valor)

Descripción: Castea el contenido de Valor a su respectivo valor entero, por ejemplo getInt("11") retornara 11; getInt(true) retornara 1, getInt(11.01) retorna la parte entera 11

Resultado: Valor Entero

func integer aumentar(nombre_var)

Descripción: Le suma una unidad a nombre_var, la variable debe estar previamente declarada y ser de tipo entero.

Resultado: Valor entero

func integer disminuir(nombre_var)

Descripción: Le resta una unidad a nombre_var, la variable debe estar previamente declarada y ser de tipo entero.

Resultado: Valor entero

func integer ArmaPropia(id_enemigo,id_arma)

Descripción: Revisa que el arma le haya sido asignada al enemigo.

Resultado: Valor booleano

Procedimientos

Son las respectivas acciones que realizará el enemigo sobre el tablero, estas son la lógica del juego como tal, su combinación con los controles, bucles y funciones; crearán un entorno de juego autónomo y de alta realidad virtual. Los procedimientos utilizables son:

proc asignarArma(id_enemigo, id_arma)

Descripción: le asigna un arma al enemigo, el enemigo puede tener *n* número de armas con *n>=0*

Definición de Parámetros:

id_enemigo: identificador numérico del enemigo previamente creado.

id_arma: identificador número del arma asignada previamente creada.

NOTA: Este evento debe verse como el cambio de arma de un enemigo, por ejemplo si el enemigo ya no tiene municiones puede cambiar su arma, si necesita un arma con más potencia puede cambiarla, etc.

proc asignarPaso(id_estrategia, id_enemigo, posicion_x, posicion_Y, tipo_movimiento)

Descripción: Asignará un evento de dar un paso a la estrategia, este es un paso lento como una caminata, lo que realizara será un avance de paso a paso hasta que logre llegar a la coordenada x, y (con esta instrucción no podrá saltar hasta la posición, deberá llegar paso a paso).

Definición de Parámetros:

id_estrategia: identificador numérico de la estrategia a la que se le asignará el evento, la estrategia debe estar previamente creada.

id_enemigo: identificador numérico del enemigo previamente creado.

posicion_x: valor numérico que representa la coordenada en x a donde se deberá mover el enemigo

posicion_y: valor numérico que representa la coordenada en y a donde se deberá mover el enemigo

tipo_movimiento: cadena de texto que identifica el tipo de movimiento que realizara el enemigo:

Tipos de Movimiento:

lineal: primero caminará sobre el eje x hasta llegar al valor posicion_x y luego sobre el eje y para llegar a la posicion_y para completar la coordenada (posicion_x,posicion_y).

circular: dará un paso en x y luego paso en y sucesivamente hasta llegar a la coordenada (posicion_x,posicion_y).

NOTA: al realizar este movimiento se tendrá un resultado el cual deberá ser sumado o restado al punteo de la estrategia involucrada de la siguiente manera:

Resultado del Movimiento	Punteo
Choque con otro Enemigo	-1
Choque con el Jugador	+1
Coordenada fuera de tablero	-2
Coordenada dentro del tablero	+1

proc asignarSalto(id_estrategia,id_enemigo, posicion_x, posicion_y)

Descripción: Asignará un evento de salto a la estrategia, este será un paso rápido, lo cual hará desaparecer al enemigo de su posición actual y reaparecer en la nueva posición solicitada.

Definición de Parámetros:

id_estrategia: identificador numérico de la estrategia a la que se le asignara el evento, la estrategia debe estar previamente creada.

id_enemigo: identificador numérico del enemigo previamente creado.

posicion_x: valor numérico que representa la coordenada en x a donde se deberá mover el enemigo

posicion_y: valor numérico que representa la coordenada en y a donde se deberá mover el enemigo

NOTA: al realizar este movimiento se tendrá un resultado el cual deberá ser sumado o restado al punteo de la estrategia involucrada de la siguiente manera:

Resultado del Movimiento	Punteo
Choque con otro Enemigo	-1
Choque con el Jugador	+1
Coordenada fuera de tablero	-2
Coordenada dentro del tablero	+1

proc avanzar(id_estrategia,id_enemigo, no_pasos)

Descripción: El enemigo caminará un número de pasos determinado según la última orientación en donde se quedó.

Definición de Parámetros:

id_estrategia: identificador numérico de la estrategia a la que se le asignará el evento, la estrategia debe estar previamente creada.

id_enemigo: identificador numérico del enemigo previamente creado.

no_pasos: número de pasos que se avanzarán al frente, según la orientación actual.

***NOTA*:** Avanzar siempre llevará al enemigo hacia el frente.

Al realizar este movimiento se tendrá un resultado el cual deberá ser sumado o restado al punteo de la estrategia involucrada de la siguiente manera:

Resultado del Movimiento	Punteo
Choque con otro Enemigo	-1
Choque con el Jugador	+1
Coordenada fuera de tablero	-2
Coordenada dentro del tablero	+1

proc girar(id_estrategia,id_enemigo, orientación)

Descripción: realizará un giro a favor o en contra de las manecillas del reloj, y colocará la orientación actual del enemigo.

Definición de Parámetros:

id_estrategia: identificador numérico de la estrategia a la que se le asignará el evento, la estrategia debe estar previamente creada.

id_enemigo: identificador numérico del enemigo previamente creado.

orientación: valor numérico con signo que representa la orientación en la que girará el enemigo de la siguiente forma:

Tipos de Orientación:

-1: girará en contra de las manecillas del reloj.

+1: girará a favor de las manecillas del reloj (también es válido usar 1, el signo + estará implícito).

proc asignarHabilidad(id_estrategia,id_enemigo, id_arma, no_detonaciones)

Descripción: Evento que realizará el ataque del enemigo, dependiendo del arma que sea seleccionada esta será la reacción que realizará.

Definición de Parámetros:

id_estrategia: identificador numérico de la estrategia a la cual se le asignará el evento, la estrategia debe estar previamente creada.

id_enemigo: identificador numérico del enemigo previamente creado.

id_arma: identificador numérico del arma a utilizar en este evento, el arma debe estar previamente creada y aún debe tener municiones, de lo contrario no podrá ser utilizada. También es importante que esta arma haya sido asignada a este enemigo, de lo contrario no podrá utilizarla.

no_detonaciones: valor numérico que representa el número de disparos que realizará, hay que tomar en cuenta el número mínimo y máximo de municiones que tiene el arma, así como actualizar las municiones disponibles que le quedarán al arma luego del evento.

NOTA: al realizar este movimiento se tendrá un resultado el cual deberá ser sumado o restado al punteo de la estrategia involucrada de la siguiente manera:

Resultado del Movimiento	Punteo
Por cada disparo acertado al Jugador	+10
Por cada disparo fallido	-2
Por cada disparo acertado a otro enemigo	-5

Notas Importantes

Todos los eventos de una estrategia pueden repetirse un número ilimitado de veces dentro de una estrategia, sin orden alguno es decir: cualquier evento puede aparecer primero, esto para lograr una interactividad total e independiente del enemigo.

Eventos Implícitos y Cronometrados:

Cada cierto tiempo se deberán actualizar los estatus del enemigo y sus componentes esto implicará utilizar los siguientes métodos.

proc ActualizarPunteo() Actualizará en pantalla el punteo del enemigo.

proc ActualizarVidas() Actualizará en pantalla el número de vidas que aún tiene el enemigo.

proc ActualizarTiempoTranscurrido() Actualizará el tiempo en pantalla desde que se inició el juego hasta la actualidad.

proc ActualizarArmas() Actualizará el estatus de las armas, el número de municiones disponibles, las armas que ya no se pueden utilizar, etc.

proc ActualizarMuniciones() Actualizará el número de municiones para el arma que actualmente tiene en habilidad el enemigo.

Todas estas funciones (Eventos implícitos) podrán ser llamadas también a petición del usuario presionando un botón específico para cada función.

Manejo de errores

Se debe mostrar un reporte de los errores que se encuentren durante las fases de compilación del proyecto, todos los errores deben tener una descripción concisa de porque se generó el error, así como su posición. Por ejemplo:

Línea	Columna	Tipo	Descripción
1	30	Léxico	No se esperaba símbolo $
24	6	Semántico	El nivel no cuenta con un inicio
67	2	Sintáctico	Se esperaba <

Consideraciones

- Cuando el compilador encuentre un error (ya sea léxico, sintáctico o semántico) este deberá continuar analizando, pero no deberá ejecutar, la finalidad es mostrar al usuario la mayor cantidad de errores posible que se encuentren en su archivo.

Lenguaje e IDE a utilizar

- El lenguaje a utilizar será Java.
- El IDE a utilizar queda a discreción del estudiante.
- Las herramientas de generación de scanner y parser a utilizar son JFlex o JLex y CUP.

Restricciones y datos importantes

- La calificación del proyecto es personal, y deberá realizarse el día y horario acordados.
- Copias totales o parciales de proyectos tendrán una nota de cero puntos y serán reportados al catedrático titular de su sección, así como a la escuela de ciencias y sistemas.
- Se deberán utilizar los lenguajes y herramientas indicados, en caso contrario no se tendrá derecho a calificación.
- Gramáticas o código copiado de internet tendrá una nota de cero puntos y serán reportados al catedrático titular de su sección, así como a la escuela de ciencias y sistemas.

Entregables

- Código fuente del proyecto, incluyendo los archivos de JFlex y CUP para la creación de los analizadores.
- Manual Técnico de la aplicación.

- Manual de Usuario de la aplicación.
- Juego funcional.

NOTA: La forma de entrega será en un CD debidamente identificado con los datos del curso y del estudiante.

Fecha de entrega

Jueves 4 de abril del 2013 **No hay prórroga**, se debe entregar todo de acuerdo a la sección entregables, la entrega será presencial. El lugar y hora serán acordados con sus auxiliares.

Ejemplo completo

Archivo de nivel

A continuación se presenta un ejemplo de un archivo de nivel completo y su resultado esperado.

```
<Espacios>
 <Espacio>
  <Nombre>normal </Nombre>
  <Pasable> verdadero </Pasable>
  <Especial> </Especial>
  <Imagen>/home/Pablo/grama.png </Imagen>
  <Inicio> falso </Inicio>
  <Enemigo> falso </Enemigo>
  <Fin>falso</Fin>
 </Espacio>

 <Espacio>
  <Nombre>fuenteVida </Nombre>
  <Pasable> verdadero </Pasable>
  <Especial> <cura>20</cura> </Especial>
  <Imagen>/home/Pablo/rio_blanca.png</Imagen>
  <Inicio> falso </Inicio>
  <Enemigo> falso </Enemigo>
  <Fin>falso</Fin>
 </Espacio>

<Espacio>
  <Nombre>inicio </Nombre>
  <Pasable> verdadero </Pasable>
  <Especial></Especial>
  <Imagen>/home/Pablo/letrero.png</Imagen>
  <Inicio> verdadero </Inicio>
  <Enemigo> falso </Enemigo>
  <Fin>falso</Fin>
 </Espacio>

<Espacio>
  <Nombre>fuenteMagica </Nombre>
  <Pasable> verdadero </Pasable>
  <Especial> <magia>100</magia> </Especial>
  <Imagen>/home/Pablo/rio_azul.png</Imagen>
  <Inicio> falso </Inicio>
  <Enemigo> falso </Enemigo>
  <Fin>falso</Fin>
 </Espacio>

<Espacio>
  <Nombre>lava </Nombre>
  <Pasable> verdadero </Pasable>
  <Especial> <dano>100</dano> </Especial>
  <Imagen>/home/Pablo/ lava.png</Imagen>
  <Inicio> falso </Inicio>
```

```
  <Enemigo> falso </Enemigo>
  <Fin>falso</Fin>
 </Espacio>

<Espacio>
  <Nombre>piedra </Nombre>
  <Pasable> falso </Pasable>
  <Especial></Especial>
  <Imagen>/home/Pablo/ piedra.png</Imagen>
  <Inicio> falso </Inicio>
  <Enemigo> falso </Enemigo>
  <Fin>falso</Fin>
 </Espacio>

<Espacio>
  <Nombre>salida </Nombre>
  <Pasable> verdadero </Pasable>
  <Especial></Especial>
  <Imagen>/home/Pablo/ cueva.png</Imagen>
  <Inicio> falso </Inicio>
  <Enemigo> falso </Enemigo>
  <Fin>verdadero</Fin>
 </Espacio>

<Espacio>
  <Nombre>enemigo </Nombre>
  <Pasable> verdadero </Pasable>
  <Especial></Especial>
  <Imagen>/home/Pablo/ enemigo.png</Imagen>
  <Inicio> falso </Inicio>
  <Enemigo> verdadero</Enemigo>
  <Fin>fin</Fin>
 </Espacio>
</Espacios>
<Nivel>
 <Nombre>GreenHills</Nombre>
 <Tamano>
  <X>5</X>
  <Y>5</Y>
 </Tamano>
 <Estructura>
  <Casilla>
    <X>0</X>
    <Y>0</Y>
    <Tipo>inicio</Tipo>
  </Casilla>
  <Casilla>
    <X>0</X>
    <Y>1</Y>
    <Tipo>normal</Tipo>
  </Casilla>
  <Casilla>
    <X>0</X>
```

```
  <Y>2</Y>
  <Tipo> normal </Tipo>
</Casilla>
<Casilla>
  <X>0</X>
  <Y>3</Y>
  <Tipo> normal </Tipo>
</Casilla>
<Casilla>
  <X>0</X>
  <Y>4</Y>
  <Tipo>fuenteVida</Tipo>
</Casilla>
<Casilla>
  <X>1</X>
  <Y>0</Y>
  <Tipo>piedra</Tipo>
</Casilla>
<Casilla>
  <X>1</X>
  <Y>1</Y>
  <Tipo>lava</Tipo>
</Casilla>
<Casilla>
  <X>1</X>
  <Y>2</Y>
  <Tipo> piedra</Tipo>
</Casilla>
<Casilla>
  <X>1</X>
  <Y>3</Y>
  <Tipo> piedra </Tipo>
</Casilla>
<Casilla>
  <X>1</X>
  <Y>4</Y>
  <Tipo>normal</Tipo>
</Casilla>
<Casilla>
  <X>2</X>
  <Y>0</Y>
  <Tipo> salida </Tipo>
</Casilla>
<Casilla>
  <X>2</X>
  <Y>1</Y>
  <Tipo>normal</Tipo>
</Casilla>
<Casilla>
  <X>2</X>
  <Y>2</Y>
  <Tipo> piedra </Tipo>
</Casilla>
```

```
<Casilla>
  <X>2</X>
  <Y>3</Y>
  <Tipo> normal </Tipo>
</Casilla>
<Casilla>
  <X>2</X>
  <Y>4</Y>
  <Tipo>normal</Tipo>
</Casilla>
<Casilla>
  <X>3</X>
  <Y>0</Y>
  <Tipo>piedra</Tipo>
</Casilla>
<Casilla>
  <X>3</X>
  <Y>1</Y>
  <Tipo>normal</Tipo>
</Casilla>
<Casilla>
  <X>3</X>
  <Y>2</Y>
  <Tipo>piedra</Tipo>
</Casilla>
<Casilla>
  <X>3</X>
  <Y>3</Y>
  <Tipo>normal</Tipo>
</Casilla>
<Casilla>
  <X>3</X>
  <Y>4</Y>
  <Tipo>piedra</Tipo>
</Casilla>
<Casilla>
  <X>4</X>
  <Y>0</Y>
  <Tipo>fuenteMagia</Tipo>
</Casilla>
<Casilla>
  <X>4</X>
  <Y>1</Y>
  <Tipo>enemigo</Tipo>
</Casilla>
<Casilla>
  <X>4</X>
  <Y>2</Y>
  <Tipo>normal</Tipo>
</Casilla>
<Casilla>
  <X>4</X>
  <Y>3</Y>
```

```
    <Tipo>normal</Tipo>
  </Casilla>
  <Casilla>
    <X>4</X>
    <Y>4</Y>
    <Tipo>enemigo</Tipo>
  </Casilla>
 <Estructura>
</Nivel>
```

Archivo de definición de enemigo

```
BD:
        addf(Continuar,boolean,false)
        addf(Saltar,boolean)
        addf(EnemigoActual,integer,0)
        addf(ContadorEnemigos,integer)
        addf(NumeroEnemigos,integer,0)
        addf(tempEnemigos,integer,0)
        addf(idArmaActual,integer)
        addf(x,integer,0)
        addf(y,integer,0)
        addf(widthTablero,integer,getInt("5"))
        addf(heightTablero,integer,5)
        addf(esquinaActual,integer,0)
        addf(modoAtaque,integer,0)
        addf(myControl, char,'a')
        addf(valorDecimal,float,11.01)
BC:
        crearImagen(1,"myImagen1","/home/Usuario/Desktop/imagen1.jpg")
        crearImagen(2,"myImagen2","/home/Usuario/Desktop/imagen2.jpg")
        crearImagen(3,"myImagen3","/home/Usuario/Desktop/imagen3.jpg")
        crearImagen(4,"myImagen4","/home/Usuario/Desktop/imagen4.jpg")
        crearImagen(5,"myImagen5","/home/Usuario/Desktop/imagen5.jpg")
        --- DEFINICION: crearPotencia(id_potencia,nivel)
        crearPotencia(1,Tinny)
        crearPotencia(2,Low)
        crearPotencia(3,Medium)
        crearPotencia(4,Hight)
        crearPotencia(5,Heavy)
        --- DEFINICION: crearArma(id_arma,nombre,tipo,max,min,no_municiones)
        crearArma(1,"WaterCase",Agua,10,0,10)
        crearArma(2,"FireCase",Fuego,100,0,90)
        crearArma(3,"Congelador",Hielo,5,1,5)
        crearArma(4,"EmpatyWar",Hipnotica,3,0,3)
        --- DEFINICION: crearEnemigo(id_enemigo,nombre,id_potencia, id_imagen)
        crearEnemigo(1,"General",5,1)
        crearEnemigo(2,"Sargento",3,2)
        crearEnemigo(3,"Especialista",2,3)
        crearEnemigo(4,"Cabo",1,5)
        --- DEFINICION: crearEstrategia(id_estrategia,nombre,punteo)
        crearEstrategia(1,"Evasion",0)
```

```
	crearEstrategia(2,"Ataque",0)
BE: --- Bloque de Estrategia
	--- DEFINICION: setf(nombre_var,valor)
	setf(Saltar,false)
	setf(NumeroEnemigos,4)
	asignarArma(1,1)
	asignarArma(1,2)
	asignarArma(1,3)
	asignarArma(1,4)
	asignarArma(2,1)
	asignarArma(2,3)
	asignarArma(3,1)
	asignarArma(3,2)
	asignarArma(4,2)
	--- Bucle Basico
	mientras getVal(Continuar)
		setf(EnemigoActual,ContadorEnemigos)
		--- Logica del Juego
		mientras getMuniciones(getVal(EnemigoActual))>=0
			--- Iniciamos Atacando
			--- Segun la posicion elegimos un salto o un giro, y tambien elegimos un tipo de arma
			setf(modoAtaque,1) --- Estrategia de Ataque
			si getLibre(getVal(EnemigoActual),x,y) entonces
				setf(idArmaActual,1)
				setf(x,getVal(x))
				setf(y,getVal(y))
				asignarPaso(getVal(modoAtaque),getVal(EnemigoActual),getVal(x),getVal(y),lineal)
			otro_caso getLibre(getVal(EnemigoActual),x+1,y)
				setf(idArmaActual,2)
				setf(x,getVal(x)+1)
				setf(y,getVal(y))
				asignarPaso(getVal(modoAtaque),getVal(EnemigoActual),getVal(x),getVal(y),circular)
			otro_caso getlibre(getVal(EnemigoActual),x,y+1)
				setf(idArmaActual,3)
				setf(x,getVal(x))
				setf(y,getVal(y)+1)
				asignarPaso(getVal(modoAtaque),getVal(EnemigoActual),getVal(x),getVal(y),lineal)
			otro_caso getlibre(getVal(EnemigoActual),x-1,y)
				setf(idArmaActual,4)
				setf(x,getVal(x)-1)
				setf(y,getVal(y))
				asignarPaso(getVal(modoAtaque),getVal(EnemigoActual),getVal(x),getVal(y),circular)
			otro_caso getlibre(getVal(EnemigoActual),x,y-1)
				setf(idArmaActual,3)
				setf(x,getVal(x))
				setf(y,getVal(y)-1)
				asignarPaso(getVal(modoAtaque),getVal(EnemigoActual),getVal(x),getVal(y),lineal)
			otro_caso
				setf(modoAtaque,2) --- Estrategia de Evasion
				si getVal(esquinaActual)=0 entonces

	asignarSalto(getVal(modoAtaque),getVal(EnemigoActual),getVal(widthTablero),getVal(heightTablero))
					setf(esquinaActual,3)
```

```
                        otro_caso getVal(esquinaActual)=1
                                asignarSalto(getVal(modoAtaque),getVal(EnemigoActual),0,getVal(heightTablero))
                                setf(esquinaActual,2)
                        otro_caso getVal(esquinaActual)=2
                                asignarSalto(getVal(modoAtaque),getVal(EnemigoActual),getVal(widthTablero),0)
                                setf(esquinaActual,1)
                        otro_caso getVal(esquinaActual)=3
                                asignarSalto(getVal(modoAtaque),getVal(EnemigoActual),0,0)
                                setf(esquinaActual,0)
                        otro_caso
                        fin_si
                        girar(1,getVal(EnemigoActual),-1)
                fin_si
                --- Realiza el Disparo (detonacion) del arma elegida, verificando que el arma le
                --- haya sido asignada al enemigo en cuestion
                si ArmaPropia(getVal(EnemigoActual),getVal(idArmaActual)) entonces
                        asignarHabilidad(getVal(modoAtaque),getVal(EnemigoActual),getVal(idArmaActual),2)
                otro_caso
                   --- Deberia reportar el error, el enemigo no puede utilizar esta arma
                        pausar(getInt(100))
                fin_si
                avanzar(1,getVal(EnemigoActual),1)
        fin_mientras
        --- Cambia turno al siguiente Enemigo
        si getVal(ContadorEnemigos) >= getVal(NumeroEnemigos) entonces
                --- Regresa al Primer enemigo
                setf(ContadorEnemigos,1)
        otro_caso
                --- Cambia al siguiente
                setf(ContadorEnemigos,ContadorEnemigos+1)
        fin_si
        --- Reviza que almenos un enemigo tenga municiones
        setf(Continuar,getBool(getStr(false)))
        --- Bucle Limitado
        para tempEnemigos=1,aumentar(tempEnemigos)=getVal(NumeroEnemigos) hacer
                si (getMuniciones(getVal(tempEnemigos))>0 entonces
                        --- Encuentra al menos un enemigo con municiones para continuar la estrategia
                        setf(Continuar,getBool("true")
                        Detener
                fin_si
        fin_para
        ActualizarArmas()
        ActualizarMuniciones()
fin_mientras
ActualizarPunteo()
ActualizarVidas()
```

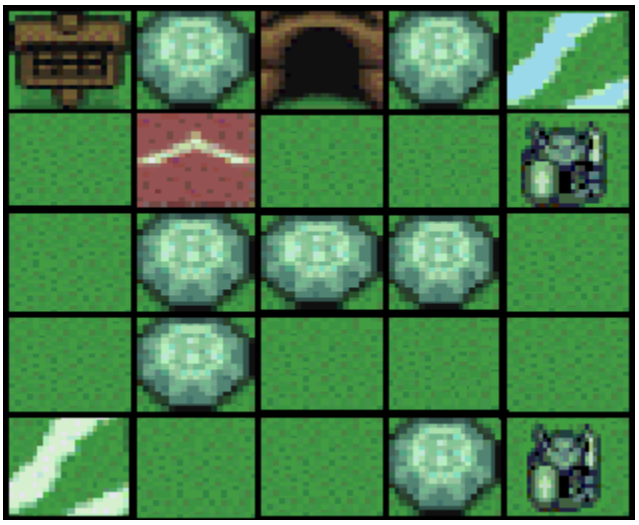


Imagen 2: Estructura de un nivel de 5*5, resultado del archivo anterior.