*Instituto Tecnológico de Costa Rica*

*Unidad de Computación*

Proyecto número uno

League-TEC

Aarón Líos Cubillo

Mario Andrés Rojas Varela

Sebastián Josué Sandí Blanco

Programación Orientada a Objetos

Ing. Jonathan Solís

Sede San Carlos

Septiembre de 2024

Índice

[Restricciones 1](#_Toc1896717810)

[Descripción del problema 2](#_Toc181960123)

[Análisis del problema 2](#_Toc823245310)

[Entendimiento del problema 3](#_Toc517076936)

[Entradas 6](#_Toc1234307117)

[Arenas 6](#_Toc1109637880)

[Personajes 6](#_Toc982196142)

[Entradas de menú 6](#_Toc1142202793)

[Salidas 6](#_Toc1505429462)

[Solución del problema 7](#_Toc288284386)

[Bitácora 9](#_Toc120747969)

[Análisis de resultados 10](#_Toc788331147)

[Conclusiones 11](#_Toc2002918506)

[Recomendaciones 11](#_Toc338844503)

## **Restricciones**

* El videojuego debe de ser programado con Java.
* El videojuego va a estar diseñado para computadoras con sistema operativo Windows 10 en adelante.
* Debe de poder ejecutarse en computadoras de gama baja.
* Se debe de contar con una interfaz gráfica.

## **Descripción del problema**

Se tiene que desarrollar un videojuego del tipo MOBA usando Java como lenguaje de programación. El videojuego debe de tener una interfaz gráfica desarrollada con herramientas compatibles con Java. Debe de tener un sistema de ataque o jugadas por turnos, con un ganador y un perdedor, el ganador será el jugador que derribe las torres del rival. Cada jugador debe de escoger una “arena” o campo de batalla propio, tres personajes con los cuales tendrá que jugar y enfrentar al equipo del rival, y la posición de sus torres dentro de su lado del tablero.

## **Análisis del problema**

**Arenas de Juego:** Cada arena debe tener al menos una torre por equipo (no se define ningún máximo de torres) y debe permitir un mínimo de tres personajes por equipo. Además, existen cuatro tipos de arenas con características elementales (fuego, agua, tierra y aire) que afectan las características de los personajes en función de su elemento, otorgando bonificaciones si el tipo del personaje es igual al de la arena. El terreno de juego debe de tener un tamaño mínimo de 10 columnas por 10 filas, y debe de estar dividido en dos mitades, una para cada jugador.

**Personajes:** Los personajes deben de poder ser creados como objetos personalizables, con atributos específicos como puntos de vida, maná, daño y velocidad de movimiento (la cual define cuántas casillas puede avanzar por turno). Sus movimientos están limitados a direcciones simples (arriba, abajo y adelante). Cada personaje tendrá sus habilidades y estas costarán maná. Los personajes podrán subir de nivel, pero solo tras derrotar a un personaje rival o destruir una torre, lo que va a incrementar sus estadísticas en un 25% (valor asignado por el proyecto, pero se podría modificar a fin de mejorar la experiencia de juego). Y tras morir, un personaje tiene que esperar un turno para poder reaparecer en alguna de las torres de su equipo.

**Torres:** Las torres son elementos clave en el juego, es aquí donde los personajes reviven, funcionando como una base o zona segura para cada uno de los equipos. Son el objetivo de cada equipo, pues destruirlas es la única forma de ganar la partida.

**Turnos:** Cada personaje tiene su turno de juego (a menos que esté muerto), donde podrán avanzar, lanzar habilidades o luchar contra otro personaje o torre. El personaje que se mueve primero es escogido aleatoriamente.

**Resultado:** Será un marcador o mensaje que informará cual jugador ganó la partida, solo aparece cuando un jugador destruye todas las torres del rival.

**Movimiento:** los personajes deben de poder desplazarse hacia arriba, abajo o adelante, más nunca podrán retroceder. Cada personaje tendrá tres acciones a realizar: moverse, pasar turno o utilizar una habilidad.

**Partida:** Primero se debe de permitir la selección de personajes, donde cada jugador va a escoger sus tres personajes, y los personajes seleccionados por un jugador no pueden ser seleccionados por el rival.

**Estadísticas:** cada jugador debe de tener sus estadísticas de juego, como sus partidas jugadas, sus partidas ganadas, y sus partidas perdidas. También con información de sus partidas pasadas como cuantos de sus personajes han muerto, cuántas torres ha perdido, cuantos personajes ha matado y cuantas torres ha destruido. Y para finalizar, su rendimiento o porcentaje de victorias.

**Documentación interna:** Se debe explicar qué hace cada parte del código por medio de comentarios. Y las funciones deben de especificar las variables que reciben, las que devuelven, y sus restricciones de uso.

## **Entendimiento del problema**

El proyecto tiene como objetivo desarrollar en el lenguaje Java un juego MBA, donde se va a requerir de la programación orientada a objetos, ya que dicho juego va a contar con diferentes clases, como serían las arenas (el escenario donde se desarrollarán batallas), personajes o torres. Cada una de estas clases con sus respectivos atributos o métodos que van a ser requeridos para la jugabilidad.

A continuación, se muestra el código y una representación (con UML) de las clases a utilizar en el proyecto:

* Arena

Código:

@startuml

class Arena {

-TipoArena tipoArena

-int filas

-int columnas

-List <Torre> torres

-List <Personajes> personajesEquipo1

-List <Personajes> personajesEquipo2

+void agregarTorre ()

+void agregarPersonaje ()

+void establecerDimensiones ()

}

enum TipoArena {

Arena de fuego

Arena de agua

Arena de tierra

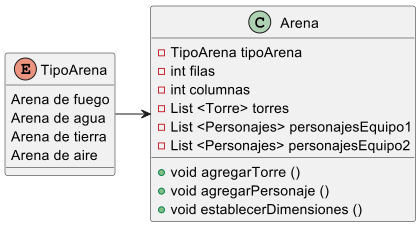
Arena de aire

}

TipoArena -> Arena

@enduml

Representación:



* Personajes

Código:

@startuml

class Personajes {

-String nombre

-int vida

-int mana

-int ataque

-String elementoVital

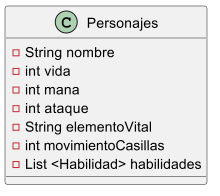
-int movimientoCasillas

-List <Habilidad> habilidades

}

@enduml

Representación:



* Habilidades

Código:

@startuml

class Habilidad {

-String nombre

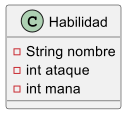
-int ataque

-int mana

}

@enduml

Representación:



* Torres

Código:

@startuml

class Torre {

-int vida

-int ubicacionFila

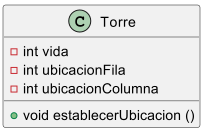
-int ubicacionColumna

+void establecerUbicacion ()

}

@enduml

Representación:



## **Entradas**

El juego va a requerir de distintos datos de entrada que van a ser ingresados por el usuario. Estos datos se dividen según el tipo de clase o funcionalidad que se requiera (arenas, personajes, torres, personalización, etc).

### **Arenas**

* Diseño de las arenas del juego.
* Extensión de la arena (columnas y filas).
* Colocación de torres y personajes en las arenas.

### **Personajes**

* Creación, edición y eliminación de personajes
* Puntos de salud, puntos de daño, maná, tipo de elemento y habilidades.
* Movimiento del personaje (número de casillas que se puede desplazar por turno).
* Acciones que puede realizar el personaje: atacar, movilizarse, activar habilidades.

### **Entradas de menú**

* Seleccionar opciones del menú: crear arena, crear personajes, empezar partida, poner el juego en pausa, rendirse o salir del juego.
* Creación de usuarios.

## **Salidas**

El juego también va a generar distintos datos de salida que serán mostrados al usuario como resultado de las interacciones o el desarrollo de las partidas. Estos datos se dividen según el tipo de clase o funcionalidad que sea necesario (arenas, personajes, torres, resultados de batalla, etc.).

* Aumento de estadísticas: Los personajes suben de nivel o habilidad al derrotar una torre o a otro enemigo.
* Reducción del maná: Luego de usar algún método relacionado con el maná se debe hacer una reducción de este para indicar que se gastó.
* Regeneración del maná: El maná se regenera un 25% por cada turno.
* Detección de ganador/perdedor: El programa debe ser capaz de determinar e indicar quién es el ganador de la partida.
* Mostrar visualización de las arenas.
* Estadísticas calculadas según las partidas jugadas.

## **Solución del problema**

Para desarrollar el juego y resolver los problemas previamente mencionados, se propone estructurar el proyecto en diversas clases, cada una con sus atributos y métodos correspondientes, lo que va a permitir una organización clara y eficiente del código.

**Arenas**: En el caso de las arenas, se creó una clase para representar todos los tipos de arenas en el juego, los atributos que tendrían estas arenas son:

* El elemento correspondiente al tipo de arena (string).
* El número de columnas de la arena (int).
* El número de filas de la arena (int).
* Las torres colocadas en la arena (class torre)
* Los personajes colocados en la arena (class personaje)

Los métodos que las arenas requieren van enfocados en su construcción y también las bonificaciones que puedan obtener los personajes. Estos métodos serían:

* Construcción de la arena: Representar y construir la arena en pantalla en base a las filas y columnas correspondientes.
* Bonificación de personajes: Comprobar si el elemento de alguno de los personajes desplegados es igual al de la arena, para otorgarle ciertas bonificaciones.

**Personajes**: Para manejar los personajes se creó una clase con distintos atributos enfocados a su información y también a sus habilidades, estos atributos serían:

* Puntos de vida (int).
* Nivel (int).
* Maná (int).
* Daño del personaje (int).
* Elemento vital del personaje (string).
* Defensa del personaje (int).
* Rango de movimiento (int).
* Estado: vivo o muerto (bool).

Por otro lado, los personajes reciben una gran variedad de alteraciones durante las partidas, por lo que los métodos que necesitarán son:

* Bonificación de arena: Si el elemento de la arena coincide con la del personaje se aumentará su daño un 10%.
* Aumento del nivel: Al destruir torres o a otro personaje sube de nivel y sus estadísticas aumentarán un 25%.
* Disminuir maná: Al usar habilidades se disminuye el maná del que dispone el personaje.
* Reaparecer: Si el estado del personaje es “muerto” (false), tiene que esperar un turno para reaparecer. Al completarse este turno el personaje va a reaparecer en la arena.
* Moverse por la arena: Desplazar al personaje un número x de filas o columnas.

**Habilidades**: Para organizar la distribución de habilidades de los personajes, se creó una clase para las mismas, donde los atributos que requiere serían:

* Costo de habilidad (int).
* Personaje al que pertenece (clase Personaje).
* Clasificación de la habilidad (string).

Los métodos de cada habilidad están enfocados en el impacto que generan a los personajes que están en la arena, los siguientes métodos varían según la clasificación de la habilidad, estos serían:

* Infligir daño: Causar un efecto de daño en los personajes enemigos.
* Curación: Aumentar o regenerar vida en los aliados.
* Alteración a personajes: Cualquier tipo de alteración a las estadísticas de un personaje dentro de la arena.

**Torres**: Las torres desplegadas por la arena cuentan con distintas estadísticas representadas con atributos, como serían:

* Puntos de vida (int): Deben tener el valor de 10000 al iniciar.
* Equipo (int/string): el equipo al que va a pertenecer la torre.
* Columna (int): columna en la que se va a colocar dentro de la arena.
* Fila (int): fila en la que se va a colocar dentro de la arena.
* Siguiente torre (Clase Torre): De ser necesario la clase torre tendría un puntero a la siguiente torre en función de su equipo.

En el caso de las torres, los métodos necesarios son:

* Disminuir la vida de la torre: Al recibir daño la torre tiene que bajar sus puntos de salud según el daño indicado por el personaje que la atacó.
* Cambiar estado de la torre: Si la vida de la torre llega a cero, entonces debe cambiar su estado a destruida.
* Colocación: Representación y construcción visual de la torre en la arena según su fila y columna.

**Jugador**: El juego cuenta con una clase jugador que administre todos los datos necesarios de cada usuario. Los atributos que requiere la clase Jugador serían:

* Nombre (string).
* Personajes (clase Personaje).
* Torres (clase Torres).
* Estadísticas (clase Estadísticas)
* Turno (bool).

Los métodos que requiere la clase Jugador son:

* Actualizar estadísticas.
* Seleccionar personajes y torres.

**Partida**: El juego cuenta con una clase para administrar las partidas que hay en el juego, sus atributos:

* Ganador (string).
* Estadísticas (clase Estadísticas).
* Participantes (clase Jugador).

Los métodos que requiere la clase Partida son:

* Asignar los turnos en que se jugará la partida.
* Asignar la arena en que se va a jugar la partida.
* Asignar los jugadores y personajes que va a tener la partida.

## **Bitácora**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Actividades | Mario Rojas | Sebastián Sandí | Aarón Líos | Total |
| Paso 1 | 1 hora | 1 hora | 1 hora | 3 horas |
| Paso 2 | 2 horas | 1 hora | 1 hora | 4 horas |
| Paso 3 | 1 hora | 1 hora | 1 hora | 3 horas |
| Diseño de aplicación | 2 horas | 2 horas | 2 horas | 6 horas |
| Investigación de funciones | 1 hora | 1 hora | 1 hora | 3 horas |
| Programación | 16 horas | 16 horas | 7 horas | 39 horas |
| Documentación interna | 2 hora | 2 hora | 5 hora | 9 horas |
| Pruebas | 6 horas | 6 horas | 6 horas | 18 horas |
| Elaboración documentación externa | 1 hora | 1 hora | 6 horas | 8 horas |
| Total | 32 horas | 31 horas | 30 horas | 93 horas |

## **Análisis de resultados**

|  |  |  |
| --- | --- | --- |
| Tarea | Estado | Observaciones |
| Menú de inicio | 100% |  |
| Nueva partida | 100% |  |
| Estadísticas de juego | 100% |  |
| Creación de usuarios | 100% |  |
| Creación de personajes | 100% |  |
| Selección de personajes | 100% |  |
| Posicionamiento de torres | 100% |  |
| Seleccionar arena | 100% |  |
| Movimiento de los personajes | 100% |  |
| Habilidades de los personajes | 100% |  |
| Combate | 100% |  |
| Subir de nivel | 100% |  |
| Ganar la partida | 100% |  |

## **Conclusiones**

Las herramientas que brinda Java y sus librerías para el diseño gráfico de aplicaciones son una gran herramienta para el desarrollo de videojuegos y aplicaciones. Pues la orientación a objetos del lenguaje facilita la creación y desarrollo de los elementos que componen cualquier videojuego. Muy pocos lenguajes poseen la facilidad con la cual se manejan las clases en java, pues con Java es muy fácil crear un archivo para la edición de un elemento del juego, heredándole elementos o variables globales almacenadas en otros archivos o paquetes del mismo proyecto. Los paquetes fueron otro elemento importante durante el desarrollo del juego, pues permiten el separar los archivos según su uso o funcionalidad dentro del videojuego, y, además, es posible acceder a estos desde cualquier otro archivo o carpeta del proyecto. Y la mejor de todas las ventajas que ofrece Java, es la facilidad con la cual es posible poder llamar cualquier función desde un punto central del proyecto, pudiendo almacenar los datos de una forma ordenada y pudiendo acceder a ellos con facilidad.

## **Recomendaciones**

Es posible que los paquetes se puedan aprovechar todavía más, separando y segmentando el código en función de su uso dentro del videojuego final.

Implementar más getters y setters.

Emplear más funciones para re escalar las imágenes dentro del juego y así evitar cualquier problema de compatibilidad.