Mage Arena

Entrega Final

Listado de integrantes

- Christian Gallo – A00098992

Enunciado del Proyecto

En el presente Proyecto, se pretende realizar un videojuego llamado Mage Arena con un estilo retro y con un estile de arte en pixel art. La temática del juego se presenta en una arena de batalla en la cual un mago debe luchar en contra de enemigos con sus poderosos hechizos hasta eliminarlos a todos en el menor tiempo posible.

Contextualizando un poco más el proyecto, el objetivo principal del juego es eliminar a un número determinado de enemigos

El juego tendrá un menú principal que permita iniciar una nueva partida, cargar una anterior, consultar los puntajes alcanzados históricamente y salir del juego.

En cada partida, se creará una arena de batalla, la cual es una ventana de 1280 por 720 pixeles con un color de fondo claro que permita contrastar a los diferentes elementos que se van a visualizar en esta ventana. Estos elementos son el player, mobs y los hechizos. El player o personaje principal que es el mago, tendrá movimiento dentro de la arena, horizontal y vertical, y lanzará hechizos, ambos controlados por el usuario. Los hechizos serán bolas de fuego volarán desde la ubicación del mago en la arena, hacia la dirección del puntero del mouse. Esto ocurrirá cuando el usuario presione el clic izquierdo, intentando de esta manera quitarles vida a los enemigos del mago cuando los hechizos los toquen. Los enemigos o mobs del mago, llamados Slimes son gelatinas que se desplazan dentro de la arena de forma aleatoria y en caso de tocar al player, este perderá puntos de vida, donde si los puntos de vida llegan a cero, se perderá el juego.

Requerimientos funcionales

Terminología usada:

- Una entidad se define como todo objeto que se encuentra dentro de la arena, estos pueden ser el player, mobs y hechizos.
- Una colisión entre dos objetos se define cuando la coordenada de uno de los objetos es la misma que la del otro objeto.
- Los visuals se define como la forma en la que se visualizará, imágenes, fotos, etc.
- Pixel art se define como la técnica de arte digital en la cual las imágenes son realizadas mediante el dibujo de cada pixel individualmente.

El videojuego debe contar con las siguientes funcionalidades:

1. Al iniciar el programa, se accederá a la ventana menú de inicio y contendrá los siguientes elementos:
    1.1. Un texto que contenga “MAGE ARENA” en la parte superior central de la ventana
    1.2. Un botón para empezar una nueva partida especificado en el Req 6, con el nombre buttNewGame
    1.3. Un botón par cargar una partida empezada
    1.4. Un botón para acceder a la ventana de puntajes especificado en el Req 4
2. Acceder a una ventaja cuando se gane el juego que contenga:
    2.1. Un texto que diga “YOU WON” en la parte superior central de la ventana
    2.2. El puntaje alcanzado en la partida
    2.3. La duración de la partida
3. Acceder a una ventana cuando se pierda el juego que contenga
    3.1. Un texto que diga “YOU LOST” en la parte superior central de la ventana
    3.2. El puntaje alcanzado en la partida
    3.3. La duración de la partida

4. Acceder a una ventaja que visualice todos los puntajes alcanzados ordenadas por:
    4.1. Por partidas ganadas primero, luego por puntaje alcanzado de manera descendiente, y finalmente por duración de la partida de manera ascendente
    4.2. Por partidas ganadas, luego por fecha de creación de manera descendiente y luego por puntaje alcanzado de manera descendiente
5. Acceder a una ventana de pausa en la partida expuesta en el Req 6.
    5.1. Con un texto que diga “PAUSE” en la parte superior central de la ventana
    5.2. Con un botón para guardar la partida, es decir, serializar todos los objetos en la partida.
    5.3. Con un botón para salir de la partida, abriendo de esta manera la ventana de menú.
6. Tener partidas de juego
    6.1. Tendrá un puntaje
    6.2. Tendrá una duración de la partida que comienza desde la creación de la partida hasta su fin que puede ser porque el player haya muerte, especificado en el Req 6.4.1.1.1
    6.3. Una arena en la cual se pueda movilizar las diferentes entidades: PLAYER, MOBS y hechizos
        6.3.1. Constará de una ventana de 1280 por 720 pixeles
            6.3.1.1. Los bordes de esta ventana delimitarán la arena por lo que ninguna entidad puede salirse por fuera de la ventana.
            6.3.1.2. Existirá una colisión entre las entidades y el borde de la arena cuando la coordenada de entidad sea igual que uno de los bordes de la arena
                6.3.1.2.1. En el caso de existir una colisión con un hechizo, este objeto será eliminado
                6.3.1.2.2. En el caso de existir una colisión con el player, este no podrá

moverse en la dirección hacia el borde de la arena’

6.3.1.2.3. En caso de existir una colisión con un mob, este no podrá moverse en la dirección hacia el borde de la arena.

6.3.2. Debe tener una visual en Pixel Art que contraste con las visuals de las diferentes entidades dentro de la arena.

6.3.3. Se visualizará en la parte superior derecha de la arena una barra que visualice el nivel o puntos de vida del player.

6.4. Un personaje principal, PLAYER, con el cual se pueda jugar.

6.4.1. Características generales

6.4.1.1. El player tiene un nivel de vida representado como 100 puntos máximo, que podrá disminuir si este recibe daño por parte de los mobs.

6.4.1.1.1. En caso de que el player llegue a una vida de 0 puntos, este morirá y se terminará la partida y se mostrará la ventana de partida perdida

6.4.1.2. Los visuals del player serán en Pixel Art

6.4.2. Con mecánicas de movimientos

6.4.2.1. Con la tecla W, se moverá el personaje hacia arriba

6.4.2.2. Con la tecla S, se moverá el personaje hacia abajo

6.4.2.3. Con la tecla A, se moverá el personaje hacia la izquierda

6.4.2.4. Con la tecla D, se moverá el personaje hacia la derecha

6.4.2.5. Con unas visuales en Pixel Art con animación

6.4.2.6.

6.4.3. Con mecánicas de combate

6.4.3.1. Donde al presionar Clic Izquierdo, se lance un hechizo hacia la dirección del Puntero.

6.4.3.1.1. Se moverá en la arena en línea recta hasta que exista una colisión con el limite de la arena y se elimine el objeto.

6.4.3.2. Los hechizos causen daño a los enemigos al existir una colisión entre las dos entidades.

6.4.3.2.1. El daño causado será de 20 puntos.

6.4.3.3. El player poseerá una barra de vida que irá disminuyendo a medida que reciba daño de los enemigos

6.5. Un conjunto de enemigos del PLAYER

6.5.1. Un enemigo llamado Slime

6.5.1.1. Que realicen daño al player al existir una colisión con esta entidad

6.5.1.2. El movimiento de los slimes será de manera aleatoria alrededor de la arena

6.5.1.2.1. Se escogerá una coordenada aleatoria dentro de la arena que se encuentre alrededor de la coordenada del slime en un máximo 10 pixeles.

6.5.1.3. Debe tener una visual en Pixel Art.

6.5.1.4. El lugar de creación y aparición del slime dentro de la arena, será en una coordenada aleatoria.

6.5.1.5. En todo momento de una partida existirán 3 slimes.

6.5.1.5.1. Cuando un slimes sea eliminado por el player, se creará un nuevo slime.

6.5.2. La partida se ganará cuando se eliminen a 20 slimes en el juego y se abrirá la ventana de victoria

7. Un sistema de autenticación y registro para iniciar la aplicación
    - 7.1.1. Para la autenticación o inicio de la aplicación
        - 7.1.1.1. Se solicitará un nombre de usuario ya creado y registrado en la base de datos
        - 7.1.1.2. Se solicitara una contraseña que sea acorde al usuario proporcionado.
    - 7.1.2. Para el registro de un nuevo usuario:
        - 7.1.2.1. Se solicitará un nombre de usuario
        - 7.1.2.2. Se solicitará una contraseña
8. Almacenar logs de las sesiones de juego de los usuarios
    - 8.1.1. Cada vez que inicie una persona a la aplicación, se almacenará la siguiente información:
        - 8.1.1.1. El tiempo que estuvo abierta la aplicación
        - 8.1.1.2. El usuario con la que estuvo la sesión abierta
        - 8.1.1.3. La fecha y hora del sistema con la que se inició la sesión
    - 8.1.2. Se deberá poder buscar los registros
        - 8.1.2.1. dados una fecha (NOT FOUND EXCEPTION)
        - 8.1.2.2. Dado un nombre de usuario
        - 8.1.2.3. Dado una duración

Justificación

Una de las principales motivaciones que tuve para iniciar en el mundo de la programación fue la creación de videojuegos. Considero que, al implementar este proyecto, será posible aplicar todos los conceptos aprendidos en el curso de APO 2 y APO 1 mediante el paradigma orientado a objetos y así mismo me encuentro en la capacidad para desarrollar todo el proyecto por mi propia cuenta.

Requerimientos no funcionales

La aplicación deberá tener las siguientes restricciones o condiciones de modelación:

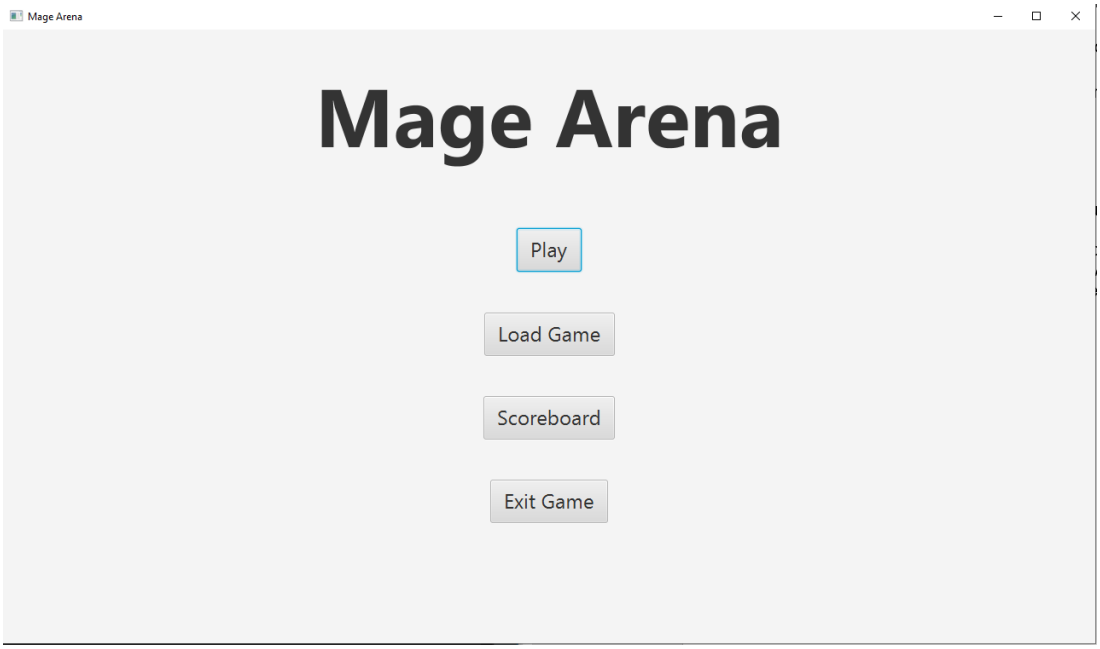
1. Persistencia mediante serialización de clases
    a. de las partidas enunciado en el Req 6
    b. de los usuarios enunciado en el Req 7
    c. de los logs enunciado en el Req 8
2. Persistencia mediante archivos de texto (escritura/lectura de archivos)
    a. de los puntajes históricos alcanzados en las partidas enunciado en el Req 6.1
3. Uso de estructuras de datos
    a. Las partidas guardas serán modeladas con listas enlazadas dobles
    b. Los puntajes históricos serán modelados con listas enlazadas dobles
    c. La base de datos de usuarios será modelada mediante arboles binarios de búsqueda (ABB)
        i. El criterio de orden del ABB estará dado por el orden lexicográfico del username del usuario.
        ii. Las búsquedas se modelarán con un método recursivo utilizando un recorrido en inorden
        iii. La adición de usuarios a la estructura de datos se modelará con un método recursivo
    d. La base de datos de logs será modelada mediante arboles binarios de búsqueda (ABB)
        i. El criterio de orden del ABB estará dado por la duración de la sesión.
        ii. Las búsquedas se modelarán con un método recursivo utilizando un recorrido en preorden
        iii. La adición de registros o logs a la estructura de datos se modelará con un método recursivo
4. Uso de métodos de ordenamiento
    a. Para el ordenamiento de los puntajes históricos se realizarán de la siguiente manera:
        i. Por puntaje alcanzado de manera descendiente y duración de la partida de manera ascendente por

medio de un ordenamiento de objetos mediante el algoritmo de Bubble sort
        ii. Por Fecha de creación de manera descendiente y luego por Puntaje alcanzado de manera descendiente por medio de un ordenamiento de objetos mediante el algoritmo de Selection sort
5. Implementación de concurrencia
    a. La verificación del estado de cada enemigo será realizado por su propio hilo. Existirán siempre 3 enemigos en cualquier instante dado.
    b. La verificación constante del estado de la vida del personaje será realizada por un hilo.
6. Uso de primitivas de gráficos
    a. La barra de vida del personaje será visualizada con primitivas de gráficos
    b. El puntaje actual de la partida será visualizado con primitivas de gráficos
    c. El tiempo de la partida será visualizado con primitivas de gráficos.
7. Uso de excepciones
    a. Personalizadas
        i. La excepción AccessDeniedException cuando se intente acceder a la base de datos y el usuario no tenga los permisos para visualizarlo
        ii. La excepción SaveNotFoundException cuando se intente buscar una partida guardada y no se encuentre
        iii. La excepción PlayerNotFoundException cuando se intente buscar a un player determinado y no se encuentre.
        iv. La excepción UserAlreadyExist cuando se intente registrar un nuevo usuario y este ya exista.
    b. Propias de Java
        i. La excepción FileNotFoundException cuando no se encuentren las imágenes .PNG de los sprites del juego.

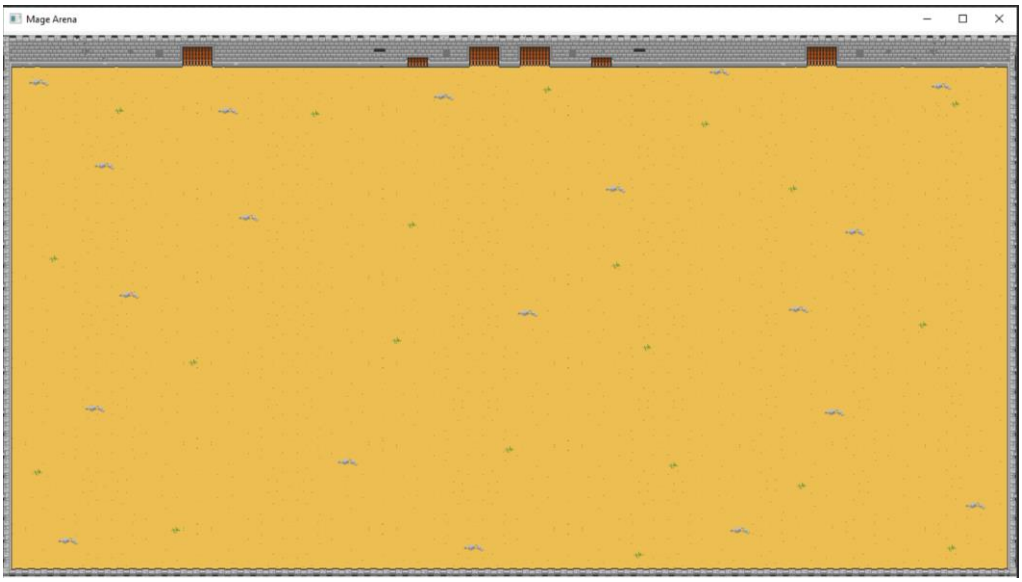
ii. La excepción IOException cuando ocurra algún problema durante el proceso de cargado de los archivos .fxml y/o en el proceso de deserialización
iii. La excepción NullPointerException cuando no se encuentre alguna entidad en la arena.
iv. La excepción Exception cuando se esté inicializando la aplicación.

Sketches

El siguiente sketch es será el menú principal del juego en donde se podrá acceder a las diferentes funcionalidades que tenga el juego como es iniciar una partida nueva al presionar el botón “Play”, cargar partidas guardadas al presionar el botón “Load Game”, consultar los puntajes históricos al presionar el botón “Scoreboard” y finalmente salir de la aplicación al presionar el botón “Exit Game”.

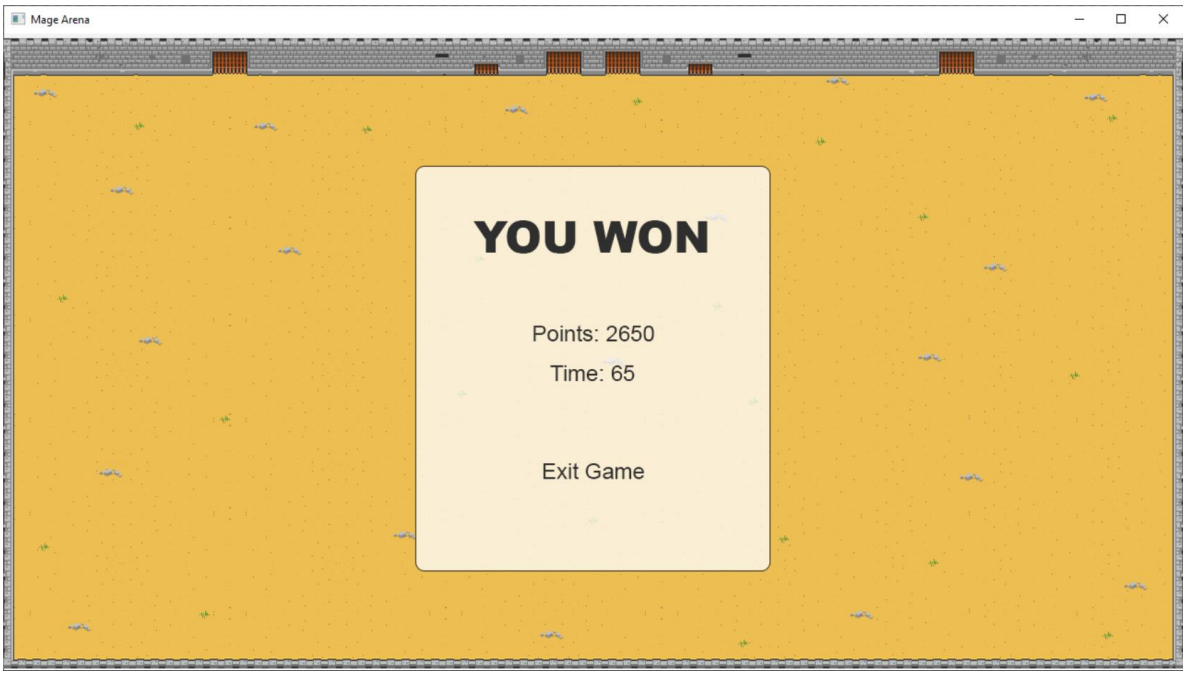


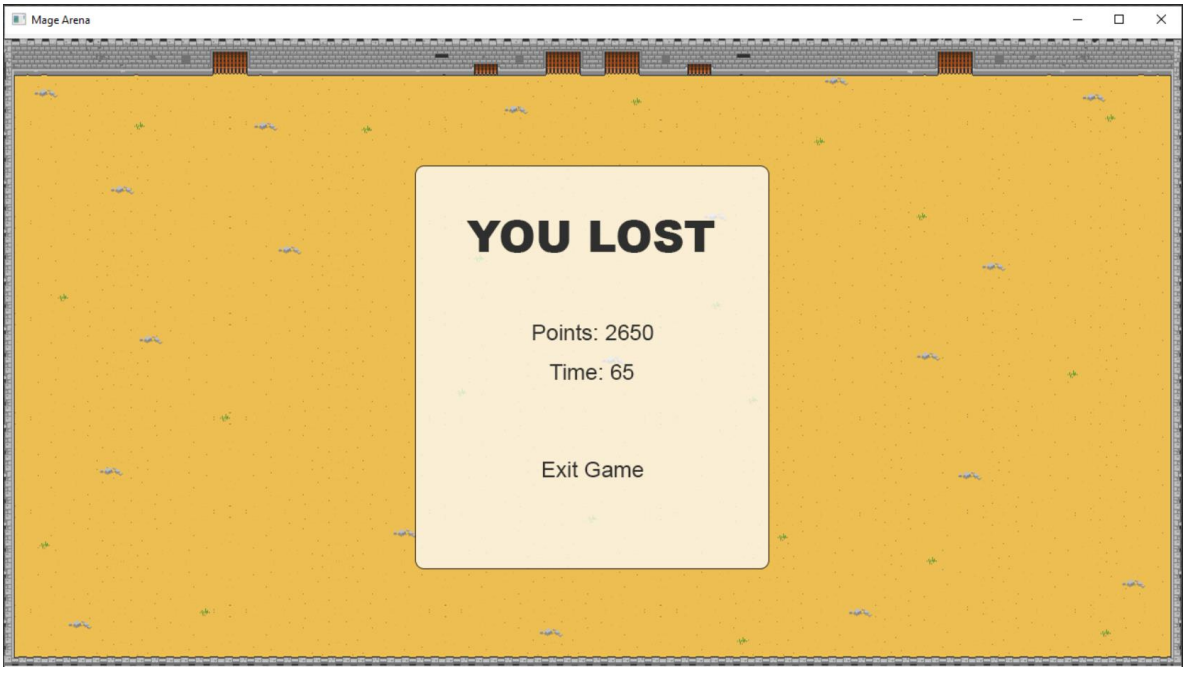
Esta será la ventana que se visualizará al crear una nueva partida, en esta se puede ver la arena en la cual se podrá jugar.



Esta ventana se desplegará cuando el usuario o jugador presione la tecla “ESC” para pausar el juego. Así mismo, podrá guardar la partida y salir de la partida para volver al menú principal.

Las siguientes dos ventanas aparecerán cuando se gane o pierda la partida. En estas se podrán consultar el puntaje alcanzado en la partida y el tiempo de duración de la partida. También, es posible devolverse al menú principal al presionar el botón “Menu”.





Esta será la ventana para abrir partidas guardadas anteriormente y al hacer clic sobre una de estas se abrirá la ventana de la partida para reanudar el juego. También, es posible devolverse al menú principal al presionar el botón “Menu”.

Esta será la ventana para consultar todos los puntajes alcanzados históricamente, se pueden presionar los botones “Score 1” o “Score 2” para elegir un criterio de orden. También, es posible devolverse al menú principal al presionar el botón “Menu”.

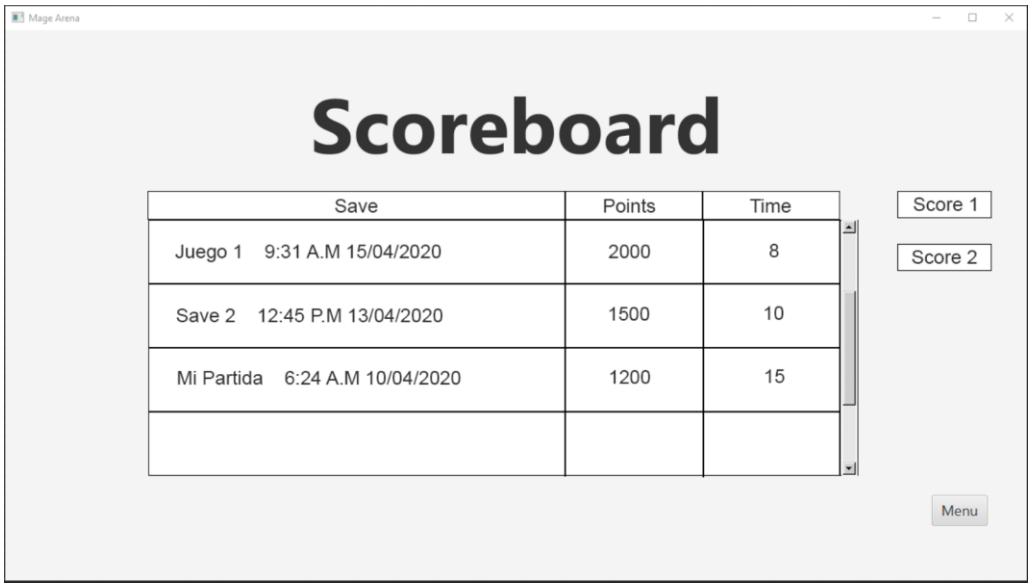


Diagrama de clases

Path: docs/Class Diagram/Class_Diagram.jpg

Diseño de casos de pruebas

- ## User

Configuración de los Escenarios

Nombre	Clase	Escenario

Diseño de Casos de prueba

Objetivo: Verificar que el constructor y los métodos getter de la clase User funciona de manera correcta, es decir, que se le asignen de manera correcta los atributos del objeto creado.

Clase	Método	Escenario	Valores de Entrada	Resultado
User	User(string username, string password)	Ninguno	username = "Chris", password "admin"	Un objeto de la clase User (left = null, right = null, parent = null, username = "Chris", password = "admin")

- ## Log

Configuración de los Escenarios

Nombre	Clase	Escenario
randomUser	User	Un objeto de la clase User (left = null, right = null, parent = null, username = "Chris", password = "admin")
actualDate	GregorianCalendar	Un objeto de la clase GregorianCalendar con la fecha actual del sistema.

Diseño de Casos de prueba

Objetivo: Verificar que el constructor y los métodos getter de la clase Log funciona de manera correcta, es decir, que se le asignen de manera correcta los atributos del objeto creado.

Clase	Método	Escenario	Valores de Entrada	Resultado
Log	Log(double sesssionTime, GregorianCalendar date, User user)	randomUser, actualDate	sessionTime = 10	Un objeto de la clase Log (sessionTime = 10, parent = null, left = null, right = null, date = actualDate, user = randomUser)

- ## Score

Configuración de los Escenarios

Nombre	Clase	Escenario
randomUser	User	Un objeto de la clase User (left = null, right = null, parent = null, username = "Chris", password = "admin")

Diseño de Casos de prueba

Objetivo: Verificar que el constructor y los métodos getter de la clase Score funciona de manera correcta, es decir, que se le asignen de manera correcta los atributos del objeto creado.

Clase	Método	Escenario	Valores de Entrada	Resultado
Score	Score(GregorianCalendar date, int score, double matchDuration)	actualDate	score = 10.2, matchDuration = 15.1	Un objeto de la clase Score (user = randomUser, score = 10.2, matchDuration = 15.1)

- ## Slime

Configuración de los Escenarios

Nombre	Clase	Escenario
slimeSprite	AnimatedImage	un objeto de la clase AnimatedImage
slimeMovement	SlimeMovement	Un objeto de la clase SlimeMovement

Diseño de Casos de prueba

Objetivo: Verificar que el constructor de la clase Slime funciona correctamente, es decir, se agregan correctamente los atributos del objeto.

Clase	Método	Escenario	Valores de Entrada	Resultado
Slime	Slime(AnimatedImage sprite, double posX, double posY, double width, double height, Movement movement, double health, double damage)	slimeSprite, slimeMovement	sprite=slimeSprite, posX = 500, posY = 500, width=100, height=100, movement=SlimeMovement, health=100, damage=20	Un objeto de la clase Slime con los atributos sprite=slimeSprite, posX = 500, posY = 500, width=100, height=100, movement=SlimeMovement, health=100, damage=20

- # SlimeFactory

Configuración de los Escenarios

Nombre	Clase	Escenario
slimeSprite	AnimatedImage	un objeto de la clase AnimatedImage
slimeMovement	SlimeMovement	Un objeto de la clase SlimeMovement

Diseño de Casos de prueba

Objetivo: Verificar que el constructor de la clase SlimeFactory funciona correctamente

Clase	Método	Escenario	Valores de Entrada	Resultado
Slime	SlimeFactory	ningunoi	ninguno	Un objeto de la clase SlimeFactory

Objetivo: Verificar que el método createMob de la clase Slime funciona correctamente, es decir, que crea Slimes

Clase	Método	Escenario	Valores de Entrada	Resultado
Slime	createMob(): Mob	slimeSprite, slimeMovement	sprite=slimeSprite, posX = 500, posY = 500, width=100, height=100, movement=SlimeMovement, health=100, damage=20	Un objeto de la clase Slime con los atributos sprite=slimeSprite, posX = 500, posY = 500, width=100, height=100, movement=SlimeMovement, health=100, damage=20

- # Fireball

Configuración de los Escenarios

Nombre	Clase	Escenario
fireballAttackObject	fireballAttack	un Objeto de la clase FireballAttack

Diseño de Casos de prueba

Objetivo: Verificar que el constructor de la clase FireballAttack y sus getters funciona correctamente

Clase	Método	Escenario	Valores de Entrada	Resultado
Fireball	Fireball()	ninguno	ninguno	Un objeto de la clase Fireball

Objetivo: Verificar que el método attack(Entity entity) : void funciona correctamente, es decir, modifica correctamente el atributo de Health the la entidad.

Clase	Método	Escenario	Valores de Entrada	Resultado

- # FireballAttack

Configuración de los Escenarios

Nombre	Clase	Escenario
fireballAttackObject	fireballAttack	un Objeto de la clase FireballAttack

Diseño de Casos de prueba

Objetivo: Verificar que el constructor de la clase FireballAttack y sus getters funciona correctamente

Clase	Método	Escenario	Valores de Entrada	Resultado
FireballAttack	FireballAttack()	ninguno	ninguno	Un objeto de la clase FireballAttacj=k

Objetivo: Verificar que el método attack(Entity entity) : void funciona correctamente, es decir, modifica correctamente el atributo de Health the la entidad.

Clase	Método	Escenario	Valores de Entrada	Resultado
FireballAttack	attack(Entity entity) : void	FireballAttackObject	ninguno	Modifica el atributo health de la entidad que se le pasa, si es que lo tiene, segun la cantidad de health del objeto que invoca el método.

- **SlimeAttack**

Configuración de los Escenarios

Nombre	Clase	Escenario
slimeAttackObject	slimeAttack	un Objeto de la clase slimeAttack

Diseño de Casos de prueba

Objetivo: Verificar que el constructor de la clase SlimeAttack y sus getters funciona correctamente

Clase	Método	Escenario	Valores de Entrada	Resultado
SlimeAttack	SlimeAttack()	ninguno	ninguno	Un objeto de la clase SlimeAttack

Objetivo: Verificar que el método attack(Entity entity) : void funciona correctamente, es decir, modifica correctamente el atributo de Health the la entidad.

Clase	Método	Escenario	Valores de Entrada	Resultado
SlimeAttack	attack(Entity entity) : void	slimeAttackObject	ninguno	Modifica el atributo health de la entidad que se le pasa, si es que lo tiene, segun la cantidad de health del objeto que invoca el método.

- **ActivatePerkAttack**

Configuración de los Escenarios

Nombre	Clase	Escenario
activatePerkAttackObject	ActivatePerkAttack	un Objeto de la clase ActivatePerkAttack

Diseño de Casos de prueba

Objetivo: Verificar que el constructor de la clase ActivatePerkAttack y sus getters funciona correctamente

Clase	Método	Escenario	Valores de Entrada	Resultado
ActivatePerkAttack	ActivatePerkAttack()	ninguno	ninguno	Un objeto de la clase ActivatePerkAttack

Objetivo: Verificar que el método attack(Entity entity) : void funciona correctamente, es decir, modifica correctamente el atributo de Health the la entidad.

Clase	Método	Escenario	Valores de Entrada	Resultado
ActivatePerkAttack	attack(Entity entity) : void	activatePerkAttackObject	ninguno	Modifica el atributo health de la entidad que se le pasa, si es que lo tiene, segun la cantidad de health del objeto que invoca el método.

- # Health

Configuración de los Escenarios

Nombre	Clase	Escenario
healthSprite	AnimatedImage	Un objeto de la clase AnimatedImage con los sprites de Health
noMovement	NoMovement	un objeto de la clase NoMovement
activatePerkAttack	ActivatePerkAtt	un objeto de la clase ActivatePerkAttack

Diseño de Casos de prueba

Objetivo: Verificar que el constructor de la clase Health y sus getters funciona correctamente

Clase	Método	Escenario	Valores de Entrada	Resultado
Health	Health()	healthSprite noMovement activatePerkAttack	sprite=healthSprite, posX = 500, posY = 500, width=100, height=100, movement=noMovement, attack=activatePerkAttack	Un objeto de la clase Health con los atributos sprite=healthSprite, posX = 500, posY = 500, width=100, height=100, movement=noMovement, attack=activatePerkAttack

- # Armor

Configuración de los Escenarios

Nombre	Clase	Escenario
armorSprite	AnimatedImage	Un objeto de la clase AnimatedImage con los sprites de Armor
noMovement	NoMovement	un objeto de la clase NoMovement
activatePerkAttack	ActivatePerkAttack	un objeto de la clase ActivatePerkAttack

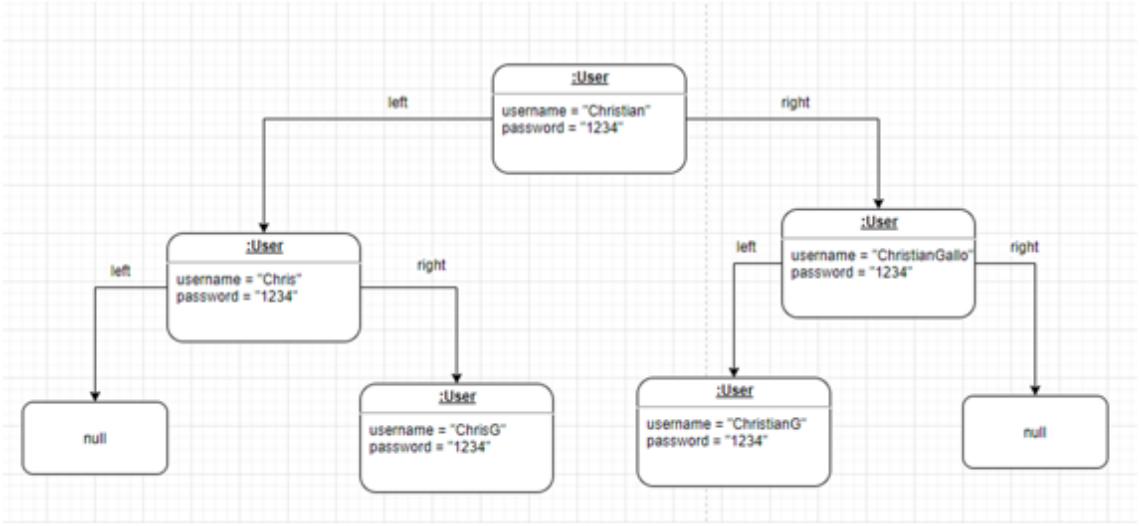
Diseño de Casos de prueba

Objetivo: Verificar que el constructor de la clase Armor y sus getters funciona correctamente

Clase	Método	Escenario	Valores de Entrada	Resultado
Armor	Armor()	armorSprite noMovement activatePerkAttack	sprite=armorSprite, posX = 500, posY = 500, width=100, height=100, movement=noMovement, attack=activatePerkAttack	Un objeto de la clase Health con los atributos sprite=healthSprite, posX = 500, posY = 500, width=100, height=100, movement=noMovement, attack=activatePerkAttack

- **GameManager**

userABB



Configuración de los Escenarios

Nombre	Clase	Escenario
serializedGameManager	GameManager	Crea un archivo con un objeto serializado de la clase GameManager
randomPlayer	GameManager	Un objeto Player
randomLog	Log	un objeto de la clase Log
randomUser	User	Un objeto de la clase User (left = null, right = null, parent = null, username = "Chris", password = "admin")
userABB	GameManager	

Diseño de Casos de prueba

Objetivo: Verificar que el constructor y los métodos getter de la clase GameManager funciona de manera correcta, es decir, que se le asignen de manera correcta los atributos del objeto creado. Adicionalmente, verificar que se mantenga la persistencia del modelo en caso de existir la serialización.

Clase	Método	Escenario	Valores de Entrada	Resultado
GameManager	GameManager()	ninguno	ninguno	Un objeto de la clase GameManager (saves = null, match = null)
GameManager	GameManager()	serializedGameManager	ninguno	Un objeto de la clase GameManager con los atributos del objeto serializado
GameManager	loadGameManager(): GameManager	serializedGameManager	ninguno	Retorna un objeto GameManager de los archivos serializados
GameManager	loadGameManager(): GameManager	ninguno	ninguno	Retorna null al no haber archivos serializados

Objetivo: Verificar que el método newMatch(): void funciona correctamente, es decir, crea un nuevo match

Clase	Método	Escenario	Valores de Entrada	Resultado
GameManager	newMatch(): void	Ninguno	Ninguno	Se le asigna correctamente a la asociación match un objeto de la clase Player

Objetivo: Verificar que el método loadMatch(Player save) : void funciona correctamente, es decir, carga correctamente la partida guardada.

Clase	Método	Escenario	Valores de Entrada	Resultado
GameManager	loadMatch(Player save) : void	randomPlayer	Ninguno	Asigna a la asociación match el objeto Player = randomPlayer
GameManager	loadMatch(Player save) : void	Ninguno	save = null	Se lanza la excepción SaveNotFoundException

Objetivo: Verificar que se pueden agregar objetos de la clase Log al ABB de logs cumpliento la propiedad de orden por medio del método addLog(double sesssionTime, GregorianCalendar date, User user): void

Clase	Método	Escenario	Valores de Entrada	Resultado
GameManager	addLog(double sesssionTime, GregorianCalendar date, User user): void	randomUser, actualDate	sessionTime = 10	Un objeto de la clase Log (sessionTime = 10, parent = null, left = null, right = null, date = actualDate, user = randomUser) agregado correctamente al ABB de logs, es decir, cumple con la propiedad de orden basado en la duración de la sesión

Objetivo: Verificar que se agregan objetos de la clase user a al ABB de users cumpliento la propiedad de orden por medio del método addUser(string username, string password) : void

Clase	Método	Escenario	Valores de Entrada	Resultado
GameManager	addUser(string username, string password) : void	Ninguno	username = "Chris", password "admin"	Un objeto de la clase User (left = null, right = null, parent = null, username = "Chris", password = "admin")

Objetivo: Verificar que se agregan objetos de la clase Player a la lista enlazada de saves al final de esta por medio del método saveMatch(Player match) : void

Clase	Método	Escenario	Valores de Entrada	Resultado
GameManager	saveMatch(Player match) : void	Ninguno	username = "Chris", password "admin"	Un objeto de la clase User (left = null, right = null, parent = null, username = "Chris", password = "admin")