Pontificia Universidad Católica de Chile
Escuela de Ingeniería
Departamento de Ciencia de la Computación

IIC2233 Programación Avanzada (2025-1)

Tarea 2

Entrega

- **Tarea y** `README.md`
  - **Fecha y hora oficial (sin atraso): ACTUALIZAR**
  - **Fecha y hora máxima (2 días de atraso): ACTUALIZAR**
  - **Lugar:** Repositorio personal de GitHub — Carpeta: `Tareas/T2/`.
    El código debe estar en la rama (*branch*) por defecto del repositorio: `main`.
  - **Pauta de corrección**: **ACTUALIZAR**.
  - **Bases generales de tareas (descuentos)**: **en este enlace**.
- **Ejecución de tarea**: La tarea será ejecutada **únicamente** desde la terminal del computador. Además, durante el proceso de corrección, se cambiará el nombre de la carpeta "`T2/`" por otro nombre y se ubicará la terminal dentro de dicha carpeta antes de ejecutar la tarea. **Los *paths* relativos utilizados en la tarea deben ser coherentes con esta instrucción.**

Objetivos

- Aplicar conceptos de programación orientada a objetos (POO) para modelar y resolver un problema complejo.
- Utilizar *properties*, clases abstractas y polimorfismo como herramientas de modelación. **ACTUALIZAR**
- Procesar *input* del usuario de forma robusta, manejando potenciales errores de formato.
- Trabajar con archivos de texto para leer y procesar datos.
- Familiarizarse con el proceso de entrega de tareas y uso de buenas prácticas de programación.

Índice

TO-DO

Acá pueden anotar sus ideas o cosas que estén pendientes. Cuando se publique el enunciado esta parte se ocultará. Para agregar notas y comentarios pueden hacerlo utilizando su nombre de la siguiente forma:

paca *for Frodo!*
Claudio *Se ha metido un mutante en la tarea! D:*
Luc *Comentario de ejemplo*
Victor *Comentario de ejemplo*
Maira *Comentario de ejemplo*
Enzo *Hola a todos!!!!*

- Los **parametros** se anotan con `ESTE_FORMATO`
- Nombres de archivos y entidades `con_este_formato`
- Enlaces van **en este formato**

1. Resumen ejecutivo

En esta tarea se solicita que desarrolles un programa en **Python** que simule un juego solitario, es decir, de un solo jugador. Se trata de un juego de combate en que el usuario compite contra la computadora para derrotar a su enemigo. La ejecución e interacción se debe realizar en su totalidad a través de la terminal del sistema, y la entrega debe realizarse en tu repositorio personal.

2. *DCConquista de Yggdrasil*

Con el pasar de los meses, tu dedicación al arte del bonsái y al desarrollo de *DCCortaRamas* te llevó a perfeccionar la simulación de poda y equilibrio arbóreo. Sin embargo, algo inesperado ocurrió. Uno de los árboles digitales dentro de tu simulador creció más allá de los límites establecidos. Sus raíces se expandieron por el código, sus ramas se multiplicaron en fractales imposibles y, antes de que pudieras detenerlo, había evolucionado en algo más...

Claudio *Necesitamos una imagen... si alguien le quedan créditos gratis en DALL · E se agradece propongan ideas xd*

Nació Yggdrasil.exe, el árbol corrupto

Poseía un código tan complejo que parecía tener voluntad propia. Su simetría perfecta emanaba un aura de poder, pero su crecimiento descontrolado generaba errores en la simulación. Pronto, comenzó a consumir otros bonsáis digitales, absorbiendo sus datos y reescribiendo las reglas del simulador a su favor. Cada intento de detenerlo solo lo hacía más fuerte. La simulación, que antes era un espacio de aprendizaje y armonía, ahora estaba bajo la sombra de un árbol monstruoso que amenazaba con tomar el control total. Tu única opción era enfrentarlo.

Programaste un nuevo modo de juego: una batalla contra la inteligencia artificial de Yggdrasil.exe. Para ello, debías seleccionar uno de tus bonsáis entrenados, cultivado con paciencia y estrategia, y prepararlo para la lucha. Cada árbol tenía habilidades únicas: el Ficus Centella, maestro de la regeneración; el Ciprés del Trueno, que desataba tormentas digitales; y el Sauce Espectral, que se desvanecía en el código como un fantasma, entre otros. Pero incluso con estos poderosos aliados, la batalla no sería fácil.

El enfrentamiento se desarrollaría en tiempo real, con raíces que se entrelazaban para restringir el crecimiento enemigo, ramas que chocaban como espadas en un duelo de precisión y hojas afiladas que caían como cuchillas. La simetría perfecta otorgaba ventajas en velocidad y resistencia, mientras que los árboles asimétricos podían apostar por un estilo caótico, pero impredecible.

Yggdrasil.exe no se contendría. Su código evolucionaba en tiempo real, adaptándose a cada uno de tus movimientos. Si no lo derrotabas, absorbería por completo la simulación y con ella, el mundo digital del bonsái desaparecería.

La última batalla estaba cerca. Tú y tu bonsái debían enfrentar a Yggdrasil.exe en un combate final. Si ganabas, podrías restaurar el equilibrio del sistema. Si perdías... bueno, tendrías que empezar a buscar otro pasatiempo. El destino del universo digital del bonsái está en tus manos.

3. Flujo del programa

En este juego, asumirás el rol de un jardinero que administra un árbol guerrero y debe tomar decisiones estratégicas para fortalecerlo y derrotar al árbol del jardinero enemigo. Al comenzar, deberás elegir la dificultad del juego, la cual debes entregar como un **argumento** por la terminal. Luego, deberás seleccionar tu árbol base, el cual podrás mejorar adquiriendo ítems Claudio *incluir referencia a la seccion de la entidad Item* en la tienda. Para ello, contarás con una cantidad `DINERO_INICIAL` Claudio *revisa cómo usar parámetros. En la*

sección TODO hay un ejemplo[1], que aumentará al **pasar ronda** Claudio *Vamos a cambiar "transcurrir tiempo"por "pasar ronda"*.

Claudio *Esto va antes de todo lo que se dijo, mejor dejarlo al inicio para que el flujo quede tal cual se espera al ejecutar el programa*

La interacción con el programa se realiza únicamente a través de la terminal paca *terminal?*, mediante un sistema de menús anidados. paca *'Primero' viene iniciar el programa en la terminal, seria precisa en 'En el primer Menu'* Al iniciar el programa desde la terminal, accederás al primer menú: **Selección de árbol** , donde deberás elegir un árbol entre los disponibles. Luego, ingresarás al menú de Inicio Claudio *Incluir referencia*, desde donde podrás administrar tu partida con diferentes opciones;

la **Tienda** Claudio *referencia* donde podrás comprar **Ítems** Claudio *referencia*como pájaros, frutas y flores, que modificarán los atributos de tu árbol.

paca *Aqui me confundo entre acciones y acciones de menú, creo que el flujo tiene que ser bien general y entenderse bien como para armar un diagrama, y en cada sección explicar el detalle.*

Al **Atacar**, se iniciará una ronda de combate en la que tu árbol infligirá daño al árbol enemigo y recibirá daño. Tras cada enfrentamiento, si ninguno de los dos árboles ha sido derrotado, volverás al **Menú de Inicio** para continuar la partida. Si la rama principal de tu árbol es destruida, perderás la partida; en ese caso, se deberá mostrar un mensaje indicando tu derrota. Por otro lado, si logras destruir la rama principal Maira *aquí quiero poner un pie de página para dar un poco de contexto con lo de la rama* del árbol enemigo, se imprimirá un mensaje de victoria. En ambos casos, la partida concluirá y el programa se cerrará automáticamente.

Claudio *Vamos a hacer que se muestre un mensaje final y se cierre el juego*

Con la opción **Mostrar información**, podrás ver el estado actual de tu árbol, incluyendo sus ramas vivas y los ítems. También podrás acceder a un resumen de estadísticas que mostrará tu estado actual en la partida. Además, puedes **Salir**, para terminar la partida.

Maira *Intenté otra redacción de el flujo para que sea mas secuencial , intente recolectar sus comentarios para hacerlo, ¿esta bien como, el largo? o debe ser aún más general?*

Maira *quiero hacer las referencias pero algunas secciones no me aparecen como referenciables*

Propuesta flujo 2.0

En este juego, asumirás el rol de un jardinero que administra un árbol guerrero y debe tomar decisiones estratégicas para fortalecerlo y derrotar al árbol del jardinero enemigo. Al comenzar, seleccionarás la dificultad del juego, la cual se proporciona como un **argumento** desde la terminal. La interacción con el programa se realiza exclusivamente a través de la terminal, mediante un sistema de menús anidados.

Una vez establecida la dificultad, ingresarás al primer menú: **??**, donde deberás elegir entre los árboles disponibles. Cada **árbol** está compuesto por **ramas** que poseen atributos relacionados a la vida, la resistencia y ataque. Además, las ramas pueden ser equipadas con ítems, que modifican sus estadísticas o les otorgan efectos especiales.

Después de elegir tu árbol, ingresarás al **??**, donde podrás administrar la partida mediante las siguientes opciones: **Tienda**, **Atacar**, **Imprimir árbol**, **Pasar el tiempo** y **Salir**. Comenzarás con una cantidad

[1] A lo largo del enunciado encontrarás varias palabras escritas en `ESTE_FORMATO` Claudio *Lo mismo, formato de parámetro*. Estas palabras corresponden a parámetros y puedes encontrar los detalles sobre ellos en la sección `parametros.py` Claudio *Incluir referencia cruzada*.

inicial de dinero `DINERO_INICIAL`[2], que podrás utilizar para comprar ítems en la **??**. La tienda es el único lugar donde puedes adquirir ítems de forma voluntaria.

El juego se desarrolla en rondas, y en cada una podrás atacar o pasar la ronda. Si decides pasar ronda, recibirás una `GANANCIA_PASIVA`, que incrementará tu dinero, se activarán los efectos de ramas e ítems que dependan del tiempo y, además, existirá una `PROB_ITEM` de obtener un ítem de baja calidad en alguna de tus ramas, así como una `PROB_ITEM_MALICIOSO` de que aparezca un ítem perjudicial.

Si decides **atacar**, se iniciará una **ronda de ataque** entre los árboles. Tras cada enfrentamiento, si ambos árboles sobreviven, regresarás al **??** para continuar la partida. Si la **rama principal** de tu árbol es destruida, perderás la partida y se mostrará un mensaje de derrota. Por otro lado, si logras destruir la rama principal del árbol enemigo, se mostrará un mensaje de victoria. En ambos casos, el juego finalizará automáticamente.

4. Menús

4.1. Ejecución

Tu programa deberá ejecutarse a través de un archivo llamado `main.py`, el cual debe aceptar argumentos por línea de comandos para indicar la dificultad del juego. El formato de la ejecución es

Para manejar argumentos desde la línea de comandos, puedes investigar sobre `sys.argv` en internet.

Es importante considerar que los menús sean a prueba de cualquier tipo de error. Esto significa que, si se ingresa una opción inválida, el programa debe informar sobre el error y volver a mostrar las opciones del menú hasta que se ingrese una opción válida.

El programa debe contener 3 menús: Selección inicial, Menú Principal y Tienda y las interacciones entre ellos serán explicadas a continuación. Además, todos los menús deben incluir una manera de ir al Menú principal. **Puedes definir el diseño estético de los menús**, pero estos deben cumplir todos los requisitos mínimos establecidos.

paca *Antes de esto, explicar: 'Tu programa debe contener 3 menús: Menu de Inicio, Menu Principal y Menú de Tienda. Las interacciones entre ellos serán explicados a continuación.'*

4.2. Selección de árbol

paca *Aclarar que sólo se puede elegir 1 árbol a la vez.*

Al ejecutar tu programa, se mostrará en consola el menú de inicio del juego, donde el usuario podrá seleccionar entre las siguientes opciones:

Opciones del Selección de árbol

- Selección de árboles disponibles: Se mostrarán todas las opciones de árboles cargados desde archivos, según la dificultad seleccionada. Por ejemplo, si el archivo contiene los árboles; **Roble Fortachón**, **Secuoya Rascacielos** y **Espino Espinoso**, el menú deberá mostrar cada árbol junto con su cantidad de ramas. Una vez seleccionado un árbol, se imprimirá un mensaje de confirmación.

[2] A lo largo del enunciado encontrarás varias palabras escritas en `ESTE_FORMATO`. Estas palabras corresponden a parámetros y puedes encontrar los detalles sobre ellos en la sección `parametros.py`

```
----------------------------
    SELECCIÓN INICIAL
----------------------------

[1] Roble Fortachón : 8 Ramas

[2] Secuoya Rascacielos : 12 Ramas

[3] Espino Espinoso : 5 Ramas

Indique su opción:
```

Figura 1: Ejemplo de selección de árbol

Maira *¿incluyo ejemplos de los mensajes de confirmación y esas cosas? o no? :O*

```
** Has seleccionado a Secuoya Rascacielos. ¡Buena elección! **
```

Figura 2: Ejemplo de mensaje de confirmación para la elección de Secuoya Rascacielos

Consideraciones

1. El menú solo debe mostrar las opciones de árboles que estén disponibles en los archivos cargados.
2. No será posible acceder al menú principal sin haber seleccionado previamente un árbol, además no será posible seleccionar más de un árbol.
3. Una vez seleccionado un árbol, se deberá redirigir automáticamente al menú principal, y no será posible regresar a la selección del árbol.

paca *hay que darle una repasada a esta redacción después*

4.3. Menú principal

Tras seleccionar el árbol, se accederá automáticamente al menú principal, el cual incluye las siguientes opciones

paca *Asumo que no se puede llegar a este menú si no has elegido un árbol primero.*

Opciones del Menú Principal

paca *Menu de Inicio y Menu Principal siento que se parecen un poco en nombre... pero es sólo maña mía*

Maira *si puede ser confuso, ¿Selección de árbol será mejor?*

Claudio *La solución es entender a la selección inicial no como un menu propiamente tal. Porque el jugador nunca vuelve a esa sección, una vez iniciado el juego no puede "re-elegir" arbol. Es una sección previa al menú principal. Ponle de nombre "Seleccion inicial", como un paso previo antes de llegar al verdadero menu.*

paca *Hay que poner una referencia de como es todo el menú, y de ahi explicar los componentes. Como en las otras tareas, pueden sacar ideas, donde estén todas las opciones ordenadas y dsps las explican una a una.*

```
----------------------------
       MENÚ PRINCIPAL
----------------------------
Dinero disponible: 2$
Ronda Actual: 1

[1] Mostrar información del árbol

[2] Comprar ítem

[3] Pasar ronda

[4] Comenzar combate

[5] Salir del juego

Indique su opción:
```

Figura 3: Ejemplo del Menú principal

- Al seleccionar la opción **Comenzar combate**, se iniciará una batalla entre el árbol del usuario y el árbol enemigo. Las reglas y mecánicas específicas del combate están detalladas en la sección Combate
- Al seleccionar la opción **Comprar ítem**, se desplegará un sub-menú que permite adquirir diferentes ítems para mejorar las ramas de tu árbol. este menú se describe en la sección Menú Ítems
- Al seleccionar la opción **Pasar ronda** se inicia una ronda dentro del juego sin combate y se activan diferentes efectos. En primer lugar, la GANANCIA PASIVA, que debe sumarse al dinero actual. Además, se aplican los efectos de ramas e ítems que dependen del tiempo. También existe la posibilidad de que aparezcan ítems de baja calidad en el inventario o ítems maliciosos.

  Maira *describo el tema de aparición aleatoria de ítems y el parámetro sobre eso aquí o se hace en entidades?*

  Claudio *Todas esas mecánicas deben ir en el flujo/mecánicas del juego, en la sección entidades va solo el desglose de los tipos que existen*

  Maira *lo puse en la propuesta flujo 2.0 en el parrafo 4 está bien ponerlo ahí? aquí no menciono nada de esos parámetros?*

- Al seleccionar la opción **Mostrar información del árbol** el programa debe preguntarle al usuario si quiere obtener la información completa el árbol , o un resumen. la **Información completa del árbol** debe mostrar una descripción del árbol, que incluya sus ramas, las estadísticas individuales de cada una y los ítems equipados en estas. la opción Resumen …
- Al seleccionar la opción **Salir del juego** se imprimirá un mensaje de despedida y se cerrará el programa. Un ejemplo de mensaje podría ser:

4.4. Tienda

Se mostrarán todas las opciones de ítems cargadas desde el archivo (). Por ejemplo, si el archivo contiene los ítems: Item1, Item2, Item3, el menú deberá desplegar todos estos ítems junto a sus atributos de mejoras

para la rama correspondiente. Una vez seleccionado un ítem, se imprimirá un mensaje de confirmación. Un ejemplo de mensaje podría ser:

Si se selecciona la opción **Volver al menú principal**, el programa regresará al menú principal con el árbol equipado con los nuevos ítems adquiridos.

Consideraciones:

Al escoger un ítem, este no desaparecerá del catálogo, por lo que podrá ser seleccionado nuevamente siempre que el usuario tenga suficiente dinero. El ítem podrá ser adquirido tantas veces se desee.

5. Entidades

En esta sección se detallarán las distintas entidades que deberás implementar para modelar *DCConquista de Yggdrasil*. Para ello, tendrás que utilizar conceptos clave de la **Programación Orientada a Objetos** como *herencia*, *multiherencia* y *clases abstractas*. Ten en cuenta que cada uno de estos conceptos debe ser incluido en la tarea **al menos una vez**, por lo que deberás descubrir dónde implementarlos **correctamente** según lo propuesto en el enunciado.

Las tres entidades principales de *DCConquista de Yggdrasil* son **??**, **??** e **??**. Debes incluir **como mínimo** las características nombradas a continuación, pero siéntete libre de añadir nuevos atributos y métodos si lo estimas necesario.

5.1. Árbol

El `Árbol` es tu centro de operaciones en *DCConquista de Yggdrasil*. Es a través de él que podrás comandar los Ataques desde tus ramas hacia las del **Árbol Enemigo**. Para lograr tu objetivo, tu árbol debe resistir a los ataques enemigos y destruir su rama principal, por lo que la estrategia es clave para alcanzar la victoria.

A continuación se presentan las características de un árbol: Enzo *En la sección de Archivos se definen más atributos para los árboles, pero no están en el documento de requerimientos. Los añado??*

- **Rama Principal**: Contiene la rama principal (tronco) de tu árbol.
- **Dinero**: Contiene el total en int de dinero que puedes usar en el Menú Tienda. Claudio *Ojo aquí, el dinero no debería ser un atributo del árbol. Es más bien de la «partida de juego»* Enzo *Especifico que no es dinero del árbol pero que lo almacena? o lo quito? Si lo quito cómo se manejaría el dinero?*
- **Nombre**: Contiene el nombre de tu árbol.

Además, debe poder realizar las siguientes acciones:

- **Cargar Rama**: Enzo *Al final, como los árboles no pueden crecer, Árbol no tendrá este método no?* Claudio *En efecto no es obligatorio, pero quizás mencionarlo después de este listado como un .ºye quizas te parezca buena idea tener un metodo que se encargue de meter ramas"para cuando les toque construir el árbol a partir de los archivos* paca *Me complica cómo se evaluaría algo así de libre elección* Claudio *En ese caso, para no dejarlo tan "libre" podemos pedirles que, en alguna parte, creen un método `cargar_arbol` que se encargue de construir la entidad árbol a partir de la información del archivo. ya luego ahí tienen la libertad de si quieren hacerlo recursivamente, rama a rama, todo el árbol de una, etc... pero ese método que se encargar de trabajar el archivo y outputear un árbol debe estar*
- **Cargar Ítem**: Debe buscar una de sus ramas y cargarle un `Ítem`.
- **Atacar**: Recibe una `rama` del árbol rival, y le aplica los puntos de daño calculados del árbol (la forma de obtener los puntos de daño se detalla en la sección **??**)

- **Pasar Ronda**: Tu ~~bonsái todopoderoso~~ árbol debe poder actualizar su estado una vez se pase a la siguiente ronda. Esto implica que se aplicarán los efectos de todas las `ramas` y sus `ítems`, pero también implica la aparición aleatoria de estos en tu inventario y de ~~plagas~~ ítems maliciosos en el `Árbol`.
- **Resumir Árbol**: Tu árbol también debe poder mostrar el estado de cada una de sus ramas. Para esto se debe sobrescribir el método `__str__` de la clase y hacer uso de las presentaciones de las ramas. Enzo *Podría especificar que este resumen del árbol debe representar su estructura, y dar un ejemplo de cómo podría ser ¿?* Claudio *Acuérdate que hay dos métodos. Este resumen solo muestra la informacion estadistica de todo el arbol, tipo, cuantas ramas tiene, cuanto es el ataque total , la vida total, etc. SIn mostrar cómo es realmente el arbol. Luego debe haber* ***otro*** *método (segun yo este otro metodo debe ser el que sobrescriba str y repr), y en ese otro metodo debe mostrar realmente todo el arbol, imprimirlo de una forma bonita mostrando todas las ramas* Enzo *Ok!! Cambiar este para que sea un resumen más estadístico del árbol, y especificaré el otro*

5.2. Rama

Las `ramas` son la unidad principal de tu árbol en *DCConquista de Yggdrasil*, quienes darán todo de sí por mantenerlo en pie para así vencer al malvado **Árbol Enemigo**.

Las `ramas` poseen diversas estadísticas que dictaminan qué tan poderosos serán los ataques de tu `árbol` y cómo este resistirá a las arremetidas de su rival. A continuación, se presentarán las diferentes características de las ramas:

- **Nombre**: Contiene el nombre de la rama.
- **Salud Máxima**: Corresponde a un entero positivo que representa cantidad máxima de puntos de vida que puede tener la rama. Enzo *No definí un rango de valores. Viendo la sección de archivos de las ramas, estas no tienen un rango de vida*
- **Salud**: Corresponde a un int entre `0` y `SALUD_MÁXIMA` que representa la cantidad de vida actual de la rama.
- **Daño Base**: Corresponde a un int entre `10` y `50` que representa la cantidad de puntos de daño que inflinge la rama.
- **Defensa**: Corresponde a un float entre `-0.5` y `0.5` que representa la resistencia de la rama al ser atacada. Enzo *Cambiado !*
- **Ítem**: Si lo hay, corresponde al objeto de la clase `ÍTEM` equipado en la rama.
- **Subramas**: Corresponde a todas las ramas que derivan de la rama actual. Esto incluye tanto a sus ramas hijas como las que surgen de ellas.

Además, las ramas pueden realizar diferentes acciones, en las que pueden aplicarse efectos según su **tipo** (Ver sección **??**). Estas acciones se detallan a continuación:

- **Agregar Subrama**: Recibe un objeto `rama` que agregará a sus subramas.
- **Cargar Ítem**: Permite a la rama equiparse un objeto `ítem`, y que apliquen así sus efectos sobre ella.
- **Retirar Ítem**: Permite a la rama descartar el `ítem` que tenía equipado, perdiendo sus efectos.
- **Pasar Ronda**: Actualiza el estado de la rama al pasar de ronda, ejerciéndose los efectos que apliquen sobre ella.
- **Atacar**: Retorna los puntos de daño que ejerce la rama. Esto incluye la influencia que pueda tener el efecto de la rama en su daño.

- **Recibir Ataque**: Recibe los puntos de daño ejercidos por el **Árbol Enemigo**, que son aplicados a la `SALUD` de la rama en función de su defensa y los posibles efectos que esta tenga.

5.2.1. Tipos de Ramas

Existen distintos tipos de `ramas`, cada uno con un efecto especial a la hora de combatir en la *DCConquista de Yggdrasil*.

Estos efectos difieren en cuándo son aplicados y cómo influyen en las estadísticas de la rama y las de sus demás compañeras. Los tipos de ramas y sus efectos se detallan a continuación:

Rama	**Efecto**
Ficus	Cada vez que transcurre una ronda, esta rama y sus hijas recuperan 10 % de la vida faltante.
Celery	Por cada ronda que esta rama no ataque ni forme parte de un ataque, gana 1 % de daño.
Hyedrid	En cada ronda en que esta rama no tenga un ítem equipado, su ataque es un 5 % mayor.
Paalm	Cada vez que esta rama recibe un ataque, su defensa aumenta en 0.02
Alovelis	Por cada ronda en que esta rama no reciba daño, regenera el 5 % de la salud de todas sus ramas hijas
Pine	En cada ronda, si tiene un ítem equipado, su defensa aumenta 0.03; de lo contrario, aumenta 0.01.
Cactoos	Cada vez que esta rama recibe un ataque, gana 1 % de daño

Cuadro 1: Tipos de ramas y sus efectos.

5.3. Ítem

Los ítems en *DCConquista de Yggdrasil* juegan un papel fundamental para potenciar las ramas de tu árbol y asegurar la victoria. Estos podrán ser adquiridos a través de la **??** a cambio de dinero, y sirven para fortalecer las estadísticas de las ramas en el combate.

5.3.1. Tipos de Ítems

6. Combates

Como el objetivo del juego es derrotar al **Árbol Enemigo**, el combate es una de las partes centrales del programa. Para cada ataque, el **jugador** debe tomar la mejor decisión para atacar a su rival para poder destruir su **rama principal**. Al momento de combatir el **jugador** debe elegir desde que rama atacar, esto ya que el daño realizado se calcula en base a la suma total del ataque de cada rama con sus subramas, dividido por la cantidad de estas utilizadas. Luc *aca no sabia si poner un diagrama igual de ejemplo, para que se entienda mejor*. Claudio *full si, diagrama ayuda a entender mucho mejor*

El ataque siempre será recibido por la rama más alta, es decir, la que tenga el mayor nivel de profundidad dentro del **Árbol**. En caso de tener 2 o mas subramas al mismo nivel de profundidad, se debe calcular de forma aleatoria cual recibe el ataque.

Para recibir daño, primero se debera calcular cuanto sera el daño a recibir, para luego restarlo a la vida. Luc *no sabia si ponerle altiro vida - (daño) o que se asuma la formula es solo pal daño y luego deben restarla a la vida ¿?,* Claudio *mejor ponlo separado, «asi se calcula el ataque», eso es como "fase 1", y despues «asi se -aplica- el daño», que seria fase 2. Dividirlo en dos fases facilita entender que hay efectos que aplican a solo una de las dos* Luc *cambiado*
Para recibir daño:

$$\text{daño\_a\_recibir} = \text{round}\left(\text{ataque\_enemigo} \times (1 - \text{defensa\_rama})\right)$$

$$\text{vida} = (\textit{vida} - \text{daño\_a\_recibir})$$

Para realizar daño al enemigo:

$$\text{daño\_a\_realizar} = \text{round}\left(\text{ataque\_propio} \times (1 - \text{defensa\_rama\_enemiga})\right)$$

$$\text{vida\_rama\_enemigo} = (\text{vida\_rama\_enemigo} - \text{daño\_a\_realizar})$$

En caso de que el daño recibido por una rama sea mayor a su salud, el daño sobrante debe ser pasado a la siguiente rama, donde se debe recalcular el daño recibido por la nueva rama en base a la fórmula. Ademas, si una rama es destruida, todas sus subramas tambien se verán destruidas. Luc *nota personal: añadir diagrama*

Por último, el enemigo ataca al azar, es decir, se debe calcular de forma aleatoria con rama el **Árbol enemigo** atacara.

7. Archivos

Para el desarrollo de *DCConquista de Yggdrasil* será necesaria la lectura y carga de archivos. Estos contendrán información sumamente relevante para la ejecución correcta de tu programa.

7.1. Árboles por dificultad

Archivos de texto `arboles_faciles.txt`, `arboles_medios.txt`, `arboles_dificiles.txt` ubicados en la carpeta `data/`, que definen los parámetros de los arboles que el jugador y el enemigo podrán seleccionar según la dificultad seleccionada:

```
{
    Roble Guerrer;
    Ficus;
    {
        Hyedrid;
        Paalm;
        Alovelis
    };
    {
        Pine;
        Cactoos;
        Cactoos;
        {
            Alovelis;
            Hyedrid;
            Hyedrid
        };
        Celery
    };
    Celery
};
{
    Sauce Veloz;
    Ficus;
    {
        Hyedrid;
        Paalm;
        Alovelis
    }
}
```

Figura 4: Ejemplo de archivo `arboles_faciles.txt`

7.2. Ramas Disponibles

Archivo `ramas.txt` que define los tipos de ramas modificables:

Atributo	**Tipo**
`nombre_rama`	String
`puntos_de_vida`	Integer
`resistencia`	Float
`daño_base`	Integer
`item`	Item|None
`efecto`	Callable aplicado al atacar/defender

Cuadro 2: Especificación de ramas

```
Ficus,1600,0.85,45
Celery,1200,0.75,35
Hyedrid,1000,0.65,75
Paalm,2000,0.90,40
Alovelis,1800,0.80,25
Pine,900,0.60,60
Cactoos,800,0.50,55
```

Figura 5: Ejemplo de archivo `ramas.txt`

7.3. Ítems de la Tienda

Archivo `items.txt` con efectos modificables para ramas:

Atributo	**Tipo**
`nombre_item`	String
`ataque`	Fórmula aplicada (e.g. `"ataque + 2"`)
`defensa`	Fórmula aplicada (e.g. `"defensa + 2"`)
`vida_maxima`	Fórmula aplicada (e.g. `"vida_maxima + 2"`)
`precio`	Integer

Cuadro 3: Especificación de ítems

```
Espinas;Ataque;ataque*1.5;defensa*0;80
Savias;Mixto;salud+20;ataque*0.8;50
Cortezas;Defensa;defensa*1.3;velocidad*0.5;60
```

Figura 6: Ejemplo de ítems con efectos duales

7.4. Ítems malignos

Archivo `items_malignos.txt` con efectos modificables para ramas:

Atributo	**Tipo**
`nombre_item`	String
`ataque`	Fórmula aplicada (e.g. `"ataque + 2"`)
`defensa`	Fórmula aplicada (e.g. `"defensa + 2"`)
`vida_maxima`	Fórmula aplicada (e.g. `"vida_maxima + 2"`)

Cuadro 4: Especificación de ítems

```
Espinas;Ataque;ataque*1.5;defensa*0;80
Savias;Mixto;salud+20;ataque*0.8;50
Cortezas;Defensa;defensa*1.3;velocidad*0.5;60
```

Figura 7: Ejemplo de ítems con efectos duales

7.5. parametros.py

Para esta tarea, se requiere la creación de un archivo `parametros.py` donde deberás **completar todos los parámetros mencionados a lo largo del enunciado**. Dichos parámetros se presentarán en `ESTE_FORMATO` y en ese color. Además, es fundamental incluir cualquier valor constante necesario en tu tarea, así como cualquier tipo de *path* utilizado.

Un parámetro siempre tiene que estar nombrado de acuerdo a la función que cumplen, por lo tanto, asegúrate de que sean descriptivos y reconocibles. Ejemplos de parametrización:

```
CINCO = 5  # mal parámetro
DINERO_MAXIMO = 3000  # buen parámetro
PROBABILIDAD_TORMENTA = 0.2  # buen parámetro
```

Si necesitas agregar algún parámetro que varíe de acuerdo a otros parámetros, una correcta parametrización sería la siguiente:

```
PI = 3.14
RADIO_CIRCUNFERENCIA = 3
AREA_CIRCUNFERENCIA = PI * (RADIO_CIRCUNFERENCIA ** 2)
```

Dentro del archivo **parametros.py**, es obligatorio que hagas uso de todos los parámetros almacenados y los importes correctamente. Cualquier información no relacionada con parámetros almacenada en este archivo resultará en una penalización en tu nota. Recuerda que no se permite el **hard-coding**[3], ya que esta práctica se considera incorrecta y su uso conllevará una reducción en tu calificación.

8. `.gitignore`

Para esta tarea **deberás utilizar un `.gitignore`** para ignorar los archivos indicados, este deberá estar dentro de tu carpeta `Tareas/T2/`.

Los elementos que no debes subir y **debes ignorar mediante el archivo `.gitignore`** para esta tarea son:

- El enunciado.
- La carpeta `data/`

Recuerda **no ignorar archivos vitales de tu tarea como los que tú creas o modificas, o tu tarea no podrá ser revisada**.

Es importante que hagan un correcto uso del archivo `.gitignore`, es decir, los archivos **deben** no subirse al repositorio debido al uso correcto del archivo `.gitignore` y no debido a otros medios.

9. Importante: Corrección de la tarea

En el **ACTUALIZAR** se encuentra la distribución de puntajes. En esta señalará con color **amarillo** cada ítem que será evaluado a nivel funcional y de código, es decir, aparte de que funcione, se revisará que el código esté bien confeccionado. Todo aquel que no esté pintado de amarillo será evaluado si y sólo si se puede probar con la ejecución de su tarea.

[3] *Hard-coding* es la práctica de ingresar valores directamente en el código fuente del programa en lugar de parametrizar desde fuentes externas.

Importante: Todo ítem corregido por el cuerpo docente será evaluado únicamente de forma ternaria: cumple totalmente el ítem, cumple parcialmente o no cumple con lo mínimo esperado. Finalmente, todos los descuentos serán asignados manualmente por el cuerpo docente respetando lo expuesto en **el documento de bases generales**.

Para terminar, si durante la realización de tu tarea se te presenta algún problema o situación que pueda afectar tu rendimiento, no dudes en contactar al ayudante de Bienestar de tu sección. El correo está en el **siguiente enlace**.

10. Restricciones y alcances

- Esta tarea es **estrictamente individual**, y está regida por el **Código de honor de Ingeniería**.
- Tu programa debe ser desarrollado en Python 3.11.X con X mayor o igual a 7.
- Tu programa debe estar compuesto por uno o más archivos de extensión `.py` que estén correctamente ordenados por carpeta. **No se revisará archivos en otra extensión como `.ipynb`**.
- Toda el código entregado debe estar contenido en la carpeta y rama (*branch*) indicadas al inicio del enunciado. Ante cualquier problema relacionado a esto, es decir, una carpte distinta a `T2` o una rama distinta a `main`, se recomienda preguntar en las ***issues* del foro**.
- Si no se encuentra especificado en el enunciado, supón que el uso de cualquier librería Python está prohibido. Pregunta en la *issue* especial del **foro** si es que es posible utilizar alguna librería en particular.
- Debes adjuntar un **único archivo markdown**, llamado `README.md`, **conciso y claro**, donde describas los alcances de tu programa, cómo correrlo, las librerías usadas, los supuestos hechos, y las referencias a código externo. El no incluir este archivo, incluir un readme vacío o el subir más de un archivo `.md`, conllevará un **descuento** en tu nota.
- Esta tarea se debe desarrollar **exclusivamente** con los contenidos liberados al momento de publicar el enunciado. No se permitirá utilizar contenidos que se vean posterior a la publicación de esta evaluación.
- Se encuentra estrictamente prohibido citar código que haya sido publicado **después de la liberación del enunciado**. En otras palabras, solo se permite citar contenido que ya exista previo a la publicación del enunciado. Además, se encuentra estrictamente prohibido el uso de herramientas generadoras de código para el apoyo de la evaluación.
- Cualquier aspecto no especificado queda a tu criterio, siempre que no pase por sobre otro que sí sea especificado por enunciado.

Las tareas que no cumplan con las restricciones del enunciado obtendrán la calificación mínima (1,0).