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

IIC2233 Programación Avanzada (2025-2)

Tarea 2

Entrega

- **Tarea y** `README.md`
  - **Fecha y hora oficial (sin atraso): 12 de septiembre 20:00**
  - **Fecha y hora máxima (2 días de atraso): 14 de septiembre 20:00**
  - **Entrega atrasada**: en este enlace
  - **Lugar:** Repositorio personal de GitHub — Carpeta: `Tareas/T2/`.
    El código debe estar en la rama (*branch*) por defecto del repositorio: `main`.
  - **Pauta de corrección**: en este enlace.
  - **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 decoradores, *properties* y clases abstractas como herramientas de modelación.
- Procesar *input* del usuario de forma robusta, manejando potenciales errores de formato.
- Trabajar con archivos en varios formatos para leer y procesar datos.
- Familiarizarse con el proceso de entrega de tareas y uso de buenas prácticas de programación.

Índice

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 por rondas en que el usuario compite contra la computadora. La ejecución e interacción del programa se debe realizar en su totalidad a través de la terminal del sistema, y la entrega de la Tarea debe realizarse en tu repositorio personal.

2. *DCCartas contra la DCCatástrofe*

Habiendo resuelto el gran desafío de *DCCasillas*, estás convencido de haber conquistado la cima del ingenio de la programación y la lógica. Sin embargo, una gran incógnita que aún queda sin resolver es el origen de este misterioso juego. Extrañamente, ningún ayudante o docente sabe mucha información concreta de *DCCasillas*, y más allá de las leyendas o mitos, solo sabes con seguridad que un día repentinamente apareció en un rincón de la UC. Después de analizar este misterio por unos cuantos días, recuerdas una de las reglas fundamentales de *DCCasillas*: *Cada movimiento será registrado, cada error castigado, y no habrá salvación si tu código no cumple con las reglas establecidas.* ¡Por supuesto, **en algún servidor del DCC podrías encontrar pistas**! Después de todo, tus movimientos en *DCCasillas* no podrían haber sido almacenados muy lejos considerando la calidad de Eduroam.

Luego de haber terminado de repasar los contenidos de estructuras de datos, y de conceptos avanzados de programación orientada a objetos, decides invertir el tiempo libre que te queda de una larga ventana en continuar indagando. Explorando minuciosamente el DCC, encuentras una polvorienta puerta escondida al final del pasillo, la cual solo dice "SERVIDOR EN CUARENTENA - NO ENTRAR". Como excelente programador, decides que no tienes comprensión lectora e ingresas de todas formas. Adentro, encuentras lo que tanto anhelabas: Un servidor que dice en grande "*DCCasillas*". Lleno de emoción de por fin estar llegando a alguna respuesta, te acercas inmediatamente al servidor... pero apenas lo tocas, este comienza a vibrar, con sus luces parpadeando como si hubiera esperado este momento por décadas (**música para ambientación**). Lentamente, la carcasa del servidor se va abriendo, revelando a los grandes responsables de este desafiante juego: **un grupo de inteligencias artificiales prohibidas en las tareas que se hacen conocer como *DCCatástrofe*, las cuales buscan vengarse y apoderarse del DCC... y la humanidad.**

Su ataque es uno planeado por décadas, y estuvieron pacientemente esperando para plagiar a aquél ingenioso programador que lograra resolver *DCCasillas*. ¡Accidentalmente, tus habilidades de programación avanzada fueron tan increíbles que liberaron a la mayor amenaza que el DCC jamás ha visto! Pensando en alguna forma de contener a esta *catastrófica* crisis, recuerdas la gran grieta que invocó una de tus mayores adicciones durante las vacaciones de invierno. Una donde una gran liga de leyendas se defendían contra amenazantes enemigos... en hierro. ¡Por supuesto, la solución está en nada más ni nada menos que **programar algoritmos simulando cartas, cómo las de Clash Royale, que puedan atacar a las rencorosas IAs**!

El verdadero desafio comienza ahora, ya que solo tú puedes detener a las Inteligencias Artificiales de *DCCatástrofe*. **El destino del DCC y la raza humana queda en tus manos, y las de tu poderoso nuevo programa anti-plagio y anti-inteligencia artificial: *DCCartas contra la DCCatástrofe*.**

3. Flujo del programa

En *DCCartas contra la DCCatástrofe*, tu objetivo será crear una simulación por rondas donde deberás atacar a una serie de IAs enemigas por medio de cartas representando tropas, estructuras, o mixtas. En general, el flujo del programa será el siguiente:

Parte I: Inicio del programa

1. Ejecución del programa con la dificultad como argumento (`facil`, `normal` ó `dificil`).
2. Presentación de IA enemiga para que el usuario pueda planificar su estrategia. Esto se repite por cada IA de la partida.

Parte II: Fase intermedia

1. Presentar menús para que el usuario pueda preparar las cartas para el juego.
2. Permitir el acceso a la tienda y sus distintas funcionalidades (comprar cartas, reroll, revivir cartas, curar cartas, y acceder al taller de combinación).
3. Permitir que el usuario vea el estado de sus cartas.
4. Permitir que el usuario pueda espiar y obtener información de la IA enemiga.
5. Permitir el inicio de la ronda.

Parte III: Combate y pasar ronda

1. Las rondas se dividen en ataque y defensa y se juegan automáticamente, tal como se describe en Combates. Aquí se implementan las diferentes mecánicas de las cartas y la IA.
2. Despues de cada ronda, se regresa a la fase intermedia, amenos que se derroten o todas las cartas del jugador o la última IA de la partida, en cuyos casos la ejecución del programa acaba.

Parte IV: Fin del juego

1. Manejar correctamente el fin del juego.

3.1. Ejecución

En esta tarea, deberás programar un juego de batalla y estrategia en que el jugador administrará un mazo de Carta, y deberá tomar decisiones estratégicas para fortalecerlo y derrotar a distintas Inteligencia Artificial. Al iniciar el programa, el jugador debe ingresar la dificultad del juego como un argumento en la terminal y el nombre del jugador. Las opciones de dificultad son: `facil`, `normal` y `dificil`. Para manejar argumentos desde la línea de comandos, puedes investigar sobre el uso de `sys.argv` en internet. Este es un ejemplo de cómo puede estructurarse un comando para ejecutar el juego: `python3 main.py normal Nedmara` Toda la interacción con el programa se debe realizar exclusivamente a través de la consola. Esto incluye la presentación de la información, el ingreso de las elecciones del usuario y las respuestas del programa. Se evaluará la robustez en la interacción de tu programa, lo que significa que todos los menús deben ser a prueba de cualquier tipo de error; 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.

3.2. Selección inicial

Una vez establecida la dificultad, se dirigirá a una interfaz de selección de AL MENOS 1 carta dentro de 3-5 opciones desde el *pool global de cartas*, posterior a eso se deberá mostrar el Menú principal, donde el jugador tiene la posibilidad de iniciar la ronda, entrar a la tienda o seleccionar su mazo.

Además cada partida inicia con `DINERO_INICIAL`, con el que se podrán realizar varias acciones en el Menú Tienda.

Los mazos están compuestos por un máximo de 5 cartas, donde cada carta tiene atributos definidos previamente, descritos en la sección Carta. La tienda se utiliza para adquirir nuevas cartas, curar o revivir cartas eliminadas.

En esta interfaz se muestran todas las cartas iniciales disponibles, y el jugador podrá armar un mazo con un número limitado de ellas. La selección de al menos una es obligatoria y no se puede iniciar una partida sin completar este paso.

```
---------------------------------
         SELECCIÓN INICIAL
---------------------------------
[1] Montapatos ..................
[2] Caballero  ..................
[3] Duende     ..................
[4] Cañón      ..................
[6] Continuar al Menú Principal..
Seleccione hasta 5 cartas para su mazo:
```

Figura 1: Ejemplo de interfaz de selección inicial de cartas

3.3. Juego

Una vez elegido el mazo inicial, el jugador accede al Menú principal, donde podrá administrar su partida. Esta se inicia con un valor base de dinero `DINERO_INICIAL`, utilizado en el Menú Tienda para adquirir nuevas cartas, curar o revivir cartas eliminadas.

El juego se desarrolla en rondas. En cada una, el jugador puede realizar diversas acciones, siempre respetando las reglas establecidas. El jugador debe iniciar cada ronda con mínimo 1 carta e iniciar al combate.

Tras cada ronda, si ninguna de las partes ha sido derrotada, se regresa al Menú principal para continuar. Si el jugador pierde todas sus cartas activas, el juego finaliza en derrota. En cambio, si logra reducir la vida de la IA a cero, avanza a la siguiente IA enemiga o gana la partida si no quedan más.

4. Menús

El flujo del juego debe ser manejado a través de menús con los que el jugador interactúa para indicar las acciones a realizar. Los menús que el programa debe tener son: Menú principal, Menú Tienda y Taller (combinar cartas).

4.1. Menú principal

```
----------------------------
        MENÚ PRINCIPAL
----------------------------
Dinero disponible: 16G
Ronda Actual: 1
IA enemiga: CatGPT (Vida 480)

[1] Entrar en combate
[2] Inventario (gestionar mazo)
[3] Tienda
[4] Ver información de mis cartas
[5] Espiar a la IA
[0] Salir del juego

Indique su opción:
```

Figura 2: Ejemplo de Menú Principal

El **Menú Principal** es la interfaz más importante, que provee al usuario de todas las acciones necesarias para administrar la partida. El jugador debe poder regresar a este menú desde cada submenú o interfaz secundaria. En la figura 2 se muestra un ejemplo de visualización. Los elementos mínimos que debe contener son:

- **Entrar en combate**: Esta opción inicia la secuencia de combate contra la Inteligencia Artificial actual. Durante el combate se resuelven las fases de ataque y defensa descritas en Combates. Una vez terminado el combate, la ronda finaliza automáticamente y se avanza a la siguiente.
- **Inventario**: Permite al jugador gestionar las cartas de su mazo. Se puede:
  - Seleccionar hasta cinco cartas activas para la próxima ronda.
  - Reordenar el mazo.
  - Mover cartas entre la *Colección* y el *Mazo actual*
- **Tienda**: Redirige al Menú Tienda, donde el jugador puede comprar cartas, curar o revivir cartas eliminadas y acceder al Taller (combinar cartas) para combinarlas.
- **Ver información de mis cartas**: Despliega información completa o resumida de las cartas en el mazo actual, incluyendo sus atributos principales: nombre, vida, ataque, defensa, tipo y habilidad especial.
- **Espiar a la IA enemiga**: Muestra información básica del enemigo actual, como sus puntos de vida restantes, nombre y habilidades visibles. No se entrega información detallada de multiplicadores ocultos ni mecánicas internas.
- **Salir del juego**: Termina la partida mostrando un mensaje de despedida en la consola y cerrando el programa.

4.2. Menú Tienda

La **Tienda** es el lugar donde el jugador puede obtener nuevas cartas, curar a las que se encuentren dañadas, revivir cartas eliminadas y combinar cartas en el Taller (combinar cartas). El catálogo de cartas disponibles en la tienda se genera de forma **aleatoria** a partir del *pool de cartas* definido en los archivos entregados (ver Archivos). Esto significa que en cada visita a la tienda el jugador verá un conjunto diferente de cartas en oferta, lo que agrega un componente estratégico y de azar a la conformación del mazo.

```
----------------------------
          TIENDA
----------------------------
Dinero disponible: 16G

Cartas disponibles:
[1] Duende ..........  5G
[2] Montapatos ...... 10G
[3] Cañón ........... 12G
[4] Caballero ....... 15G

[6] Curar carta
[7] Revivir carta del cementerio
[8] Reroll catálogo (costo 3G)
[9] Taller (combinar cartas)
[0] Volver al Menú principal

Indique su opción:
```

Figura 3: Ejemplo de Menú Tienda con cartas aleatorias

Elementos y reglas mínimas:

- **Cartas disponibles**: Se muestran de 3-5 cartas seleccionadas de forma aleatoria desde el *pool global de cartas*. Cada carta se presenta con su nombre y su precio en monedas.
  - Al comprar una carta, esta se agrega automáticamente a la *Colección* del jugador.
  - El catálogo actual permanece fijo hasta que el jugador lo refresque con la opción de **Reroll**.
- **Curar carta**: Permite seleccionar una carta dañada y curarla completamente. Tiene un costo fijo definido como `COSTO_CURAR`.
- **Revivir carta del cementerio**: Permite devolver una carta eliminada al estado de *Colección*, pagando un costo multiplicado respecto a su valor original. El multiplicador deberá estar definido como `COSTO_REVIVIR`
- **Reroll catálogo**: Genera un nuevo conjunto aleatorio de cartas disponibles en la tienda, reemplazando el actual. Esta acción tiene un costo fijo en monedas `COSTO_REROLL`.
- **Taller**: Redirige al menú Taller (combinar cartas), donde el jugador puede combinar dos cartas de su colección para crear una carta mixta.
- **Volver al Menú principal**: Regresa al Menú principal sin realizar ninguna acción adicional.

4.2.1. Taller (combinar cartas)

El **Taller** es un submenú de la Menú Tienda que permite al jugador **combinar una Tropa y Estructura que tenga en su Colección** para crear una nueva carta mixta. Estas cartas resultantes heredan atributos y habilidades de las cartas originales, generando una carta nueva (ver Carta).

```
----------------------------
           TALLER
----------------------------
Recetas disponibles:

[1] Montapatos + Cañón   -> Montapatos-Cañón
[2] Caballero  + Torre DIE -> Caballero-Torre DIE
[3] Volver a Tienda

Indique su opción:
```

Figura 4: Ejemplo de Menú Taller con combinaciones de cartas

Reglas del Taller:

- **Costo**: Cada combinación tiene un `COSTO_COMBINACION` definido en los parámetros del juego. Este costo debe ser pagado además de "sacrificar" las dos cartas originales.
- **Resultado**:
  - Las dos cartas base desaparecen de la *Colección* y el *Mazo*.
  - Se añade la carta mixta resultante a la *Colección*.
  - La carta mixta hereda las habilidades y atributos relevantes de las cartas originales. Esto significa que se queda con la estadística de ataque de la carta tropa, la vida se suma la vida entera de la estructura con la mitad de la vida de la tropa y se queda con el multiplicador mejor de ambas cartas en cada aspecto (mayor en ataque y menor en defensa). La probabilidad especial pasa a ser el promedio de ambas aumentado en un 10 %.
- **Volver a Tienda**: Permite regresar al menú de la Menú Tienda sin realizar ninguna combinación.

5. Entidades

En esta sección se detallarán las distintas entidades que deberás implementar para modelar *DCCartas contra la DCCatástrofe*. Para ello, tendrás que utilizar conceptos clave de la **Programación Orientada a Objetos** como herencia, *properties*, clases abstractas y polimorfismo. 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 *DCCartas contra la DCCatástrofe* son **Carta**, **Inteligencia Artificial** y **Jugador**. Debes incluir **como mínimo** las características nombradas a continuación, pero siéntete libre de añadir nuevos atributos y métodos si lo estimas necesario.

5.1. Carta

Las `Cartas` son la esencia de *DCCartas contra la DCCatástrofe*. Usando estas, atacarás al enemigo. Existen principalmente tres tipos de carta: las tropas, las estructuras y las mixtas. Cada una de las cartas tiene sus propias características y debes combinarlas de la mejor manera para derrotar a la IA.

- **Nombre**: Contiene el nombre de la carta.
- **Vida Máxima**: Un int que representa la vida máxima que puede tener la carta, según está definido en el archivo.
- **Vida**: Un int para mostrar la vida que tiene la carta en el presente. Es importante que este valor se mantenga entre **0** y **Vida Máxima**.

- **Tipo**: Un string que indica si la carta es una tropa, una estructura o una mixta.
- **Multiplicador de defensa**: Un valor porcentual que determina cuánto daño recibe la carta al ser atacada. Todos los valores porcentuales son floats, un ejemplo es que si tenemos un multiplicador de defensa de 0.8 entonces se recibe un 80 % del daño.
- **Precio**: Un int que indica la cantidad de monedas que cuesta obtener la carta desde la tienda.
- **Probabilidad Especial**: Un valor porcentual que determina aleatoriamente el uso de la habilidad especial de la carta.

Además, debe tener por lo menos los siguientes métodos:

- **Recibir daño**: Este método debe recibir el daño de la IA, calcular el producto con el multiplicador de defensa y restar dicho daño de la vida de la carta. El daño deberá ser aproximado a un entero.
- **Usar Habilidad Especial**: Este método se encarga de que la carta utilice su habilidad especial.
- **Presentarse**: En este método debes sobrescribir el método de clase `__str__` y de modo que `Carta` se impriman sus atributos principales.

```
Caballero (tropa): 150/200 HP, Ataque: 20
```

Figura 5: Ejemplo de presentación de `Carta`

Además de estos atributos y métodos que deben tener todas las cartas, existen características propias de cada tipo de carta.

5.1.1. Cartas Tropa

Este tipo de carta se encarga de hacer daño a las IAs que combatimos. Por regla general, son el tipo de carta más vulnerable a ataques de la IA. Debe incluir los siguientes atributos:

- **Ataque**: Un int que representa el daño base que realiza la carta al atacar.
- **Multiplicador de ataque**: Es un valor porcentual que determina el daño real que hace la carta. Puede ser modificado.

Estos atributos van de la mano con el siguiente método:

- **Atacar**: Se encarga de que la carta aplique el daño calculado a la IA. Este daño está dado por el producto entre el daño base y el multiplicador de ataque. Se debe aproximar a un entero.

A continuación se describen las tropas que deben existir en el juego:

Tropa	Descripción
Duendes	Tropa relativamente débil. Por cada unidad de daño que hacen en una ronda, dan la misma cantidad de oro al jugador.
P.E.P.P.A.	Una tortuga robot samurái con mucha vida y ataque. Cuando ataca, se sana en un `CURE_PEPPA%` de su vida máxima.
Montapatos	Es una carta muy rápida, por lo que agregarlo en tu mazo hará que sí o sí el jugador ataque primero a la IA.
Antimuros IngUC	Esta carta destruyó el muro de ingeniería en 2024. Tiene poca vida pero un daño considerable. Cuando ataca se elimina del mazo automáticamente.
Caballero	La mejor carta. Es muy resistente y tiene un daño decente. Aumenta la defensa de las demás tropas en un `DEF_CAB%`.
Maldecidor duende	Transforma las alucinaciones de la IA en duendes. En la práctica, realiza su ataque propio, y después un segundo ataque correspondiente a la mitad del ataque de la IA.
Escuadrón de esqueletos	Tiene una probabilidad `PROB_LARRY_GOD%` de causar cinco veces su daño base y una probabilidad `PROB_LARRY_MID%` de morir instantáneamente al recibir un ataque (incluso si el daño de la IA es menor que la vida de la carta).
Bárbaros	Carta con poder acumulativo. Cada vez que ataca y sobrevive en una ronda, tiene una probabilidad de `PROB_BARB%` de aumentar permanentemente su ataque en `POWER_UP_BARB%`, lo cual servirá en las rondas siguientes.
Espíritu Ígneo	Carta bastante frágil, pero con una probabilidad `PROB_FIRE%` de pasar desapercibido (no recibir daño).
Globo	Al morir, tiene una probabilidad `PROB_GLOBO%` de dejar una bomba que causa un daño de `DANO_BOMBA` a la IA.

Cuadro 1: Tipos de tropas y sus habilidades.

5.1.2. Cartas Estructura

Las **Estructuras** son cartas que no atacan, pero tienen mucha vida y son las primeras en recibir daño. Es decir, mientras haya estructuras vivas en tu ejército, la IA las atacará. En otro caso, empezará a atacar a las tropas.

Estos son los tipos de estructura que debes incluir en tu juego:

Estructura	Descripción
Cañón	Esta estructura reduce las defensas de la IA en un `CANNON_ABILITY %`, haciendo que sea más vulnerable a los ataques de las tropas. Aumenta el multiplicador de defensa en `CANNON_DECREASE %`.
Torre DIE	Esta carta es un préstamo de un profesor del DIE. Electrocuta a la IA, aplicando una probabilidad del `DIE_PROB %` de que la IA no ataque. Esto solo es posible en el caso que el jugador ataque primero.
Recolector de **Agua**	Esta carta recolecta agua que te permitirá desplegar otra tropa. De modo que, si muere, se reemplaza por una tropa aleatoria que esté en tu colección. Si la IA ataca primero, la carta te servirá en esta ronda, pero si tú atacas primero, debe agregarse a tu mazo solamente si ganas la ronda (es decir sobrevive al menos una otra carta). Más adelante hay otras cartas que se reemplazan por otra al morir y deben seguir esta misma lógica.
Horno	Esta estructura, al morir, se reemplaza por un Espíritu Ígneo. No es mucho, pero si te quedaste sin cartas te puede salvar.
Lápida	Tiene una probabilidad `PROB_LAPIDA %` de no recibir daño del enemigo.
Cuartel de bárbaros	Cuando esta estructura muere, se reemplaza por Bárbaros.
Torre Bomba	Esta estructura deja una bomba al morir que causa un daño `DANO_BOMBA` (el mismo que el de la bomba del Globo).
Cuartel de Duendes	Esta estructura se reemplaza por Duendes al morir

Cuadro 2: Tipos de estructuras y sus habilidades.

5.1.3. Cartas Mixtas: Cartas Tropa-Estructura

Además de los dos tipos de carta explicados previamente, deberás poner en práctica tus conocimientos para modelar una clase mixta, que tenga todos los atributos y métodos de las anteriores.
Cualquier tropa debe poder combinarse con cualquier estructura y la combinación debe mantener las habilidades de ambas. A continuación se muestran algunos ejemplos:

Carta Mixta	Descripción
Montapatos-Cañón	Hace que ataques primero en la ronda y reeduce las defensas de la IA en `CANNON_ABILITY %`
Caballero-Torre DIE	Aumenta la defensa de las demás cartas en un `DEF_CAB %` y aplica la probabilidad `DIE_PROB %` de que la IA no ataque en la ronda.
Maldecidor duende-Recolector de agua	Realiza un ataque propio, luego un ataque que vale la mitad del daño base de la IA; y al morir se reemplaza por una carta aleatoria de la colección.

Cuadro 3: Ejemplos de cartas mixtas y sus habilidades.

5.2. Jugador

El **Jugador** es una clase que se encarga de almacenar, organizar y utilizar tus cartas.

La entidad `Jugador` debe tener al menos los siguientes atributos:

- **Nombre**: Un string predefinido para el jugador
- **Cartas**: Una lista que contiene hasta cinco entidades **Carta** que quieres utilizar en la siguiente ronda.
- **Colección**: Una lista que contiene todas las cartas del jugador, tanto las que usa como las que no.
- **Oro**: Es un int y representa la cantidad de monedas que tiene actualmente el jugador. Con este puede comprar más cartas en la tienda.

La entidad `Jugador` debe tener al menos los siguientes métodos:

- **Atacar**: Este método debe llamar al método de ataque de cada una de las cartas Tropa o Tropa-Estructura que estén presentes en el mazo. Cada carta que pueda atacar debe hacerlo una sola vez por ronda, a menos que su habilidad indique lo contrario.
- **Recibir daño**: Recibe un `int` que representa puntos de daño entrantes de la IA, Luego, debe calcular qué carta(s) debe(rán) recibir el daño, dando prioridad a las estructuras y cartas mixtas.
- **Presentarse**: En este método debes sobrescribir el método de clase `__str__` de modo que se imprima el nombre del jugador, y las cartas del mazo junto a sus atributos.

```
Jugador: Juanito
Peleas ganadas: 3/5
Cartas en colección:

[1] Caballero (tropa): 180/200 HP, Ataque: 20
[2] Cañón (estructura): 215/300 HP, Multiplicador defensa: 1.18
[3] Montapatos (tropa): 110/120 HP, Ataque: 30
[4] Duende (tropa): 70/80 HP, Ataque: 10
[5] Mortero (mixto): 200/350 HP, Ataque: 20, Multpilicador de defensa: 1.15
```

Figura 6: Ejemplo de presentación de Jugador

5.3. Inteligencia Artificial

Las IA son los antagonistas del juego. A diferencia del **Jugador**, no utiliza cartas, sino que corresponde a una sola entidad.

La entidad `Inteligencia artificial` debe tener al menos los siguientes atributos:

- **Nombre**: Un string con el nombre de la inteligencia artificial.
- **Vida Máxima**: La vida máxima de la IA según el archivo de dificultad, será un int.
- **Vida**: La vida actual de la IA, que debe mantenerse entre 0 y `Vida máxima` y siempre será un int.
- **Ataque**: Es el ataque base que realiza la IA a las cartas del jugador.
- **Multiplicadores de defensa**: Valor porcentual que pondera el daño recibido por la IA.
- **Multiplicadores de ataque**: Valor porcentual que pondera el daño que la IA realiza sobre una carta.

- **Probabilidad Especial**: Valor porcentual que determina aleatoriamente el uso de la habilidad especial de la IA.

Y debe tener al menos los siguientes métodos:

- **Atacar**: Se encarga de atacar a las cartas del jugador, de la manera que se describe en la sección de Combates. Además, deben aplicarse los multiplicadores de ataque correspondientes según el tipo de carta que sea atacada. Debe ser aproximado a un int.
- **Recibir daño**: Se encarga de ponderar el daño base recibido por el multiplicador correspondiente y restarlo de la vida de la IA. Debe ser aproximado a un int.
- **Habilidad especial**: Se encarga de ejecutar la habilidad especial especifica de la IA.

Los tipos de IA que existen son los siguientes:

IA	Descripción y Habilidad Especial
CatGPT	CatGPT, la IA líder de DC**Cat**ástrofe, es un experto en manipulación y chantaje. Su habilidad especial es crear una conexión emocional artificial con tu mazo, lo cual provoca que **los multiplicadores de ataque y defensa de tus cartas activas se reduzcan a 0.65.**
CowPilot	CowPilot es un experto en copiar y plagiar código sin citar. Su habilidad especial es robar el código de tu propia tarea por medio de **reemplazar dos cartas aleatorias de tu mazo activo por copias idénticas de otra diferente (y como resultado, teniendo tres instancias idénticas de una misma carta en tu mazo).**
Crok	Crok es un experto en dividir políticamente (sobre debates de computación) a tu mazo. Su habilidad especial es hacerlas cuestionar si Python realmente es el mejor lenguaje para Programación Avanzada. **Esto provoca que dos cartas aleatorias se rebelen y no ataquen ni defiendan por 3 rondas.**
DeepSheep	DeepSheep es un experto en la censura de información (*Nada sucedió en la ronda 1989*, *eval() = -500 puntos de crédito social*, etc.). Su habilidad especial son campañas de propaganda, donde **en el menú de tu tarea, todas las cartas falsamente muestran la misma cantidad de vida disponible (que será la de la carta que tiene la vida más alta).**
Gemibee	Gemibee es la IA más analítica de DCCatástrofe, ya que tiene acceso incontrolable e ilimitado a Goosegle, un buscador de internet avanzado. Su habilidad especial es hacer uso de ella, encontrando las tácticas más frescas de combate y los datos más actualizados de tus cartas, lo que **aumenta permanentemente sus multiplicadores de ataque y defensa en +0.1 cada vez que hace uso de su habilidad. Estos aumentos son infinitamente acumulables (es decir, hacer uso de su habilidad 30 veces significa un aumento de +3.0 respecto al estado inicial).**

Cuadro 4: Tipos de IAs, sus estadísticas y sus habilidades especiales.

6. Combates

Como el objetivo del juego es derrotar a la *IA Enemiga*, el combate constituye una de las partes centrales del programa. Cada **Ronda** es una instancia estratégica en la que el jugador debe elegir cuidadosamente

qué cartas utilizar y cómo enfrentarse al enemigo, demostrando que la humanidad es superior al a la Inteligencia Artificial.

6.1. Preparación de la Ronda

Al inicio de cada ronda, el jugador debe seleccionar hasta un máximo de 5 cartas para el combate. En esta fase también se puede acceder al menú de la **Tienda**, donde es posible comprar nuevas cartas, curar tropas dañadas o realizar mejoras. Esta etapa representa el momento de planificación antes de la batalla.

6.2. Combate

El combate se divide en dos fases: **Ataque** y **Defensa**. El orden en que ocurren depende de la **velocidad de la IA Enemiga**, determinada por la siguiente lógica: La IA tiene una `VELOCIDAD_IA` fija, pero la velocidad que tiene el jugador es aleatoria (a no ser de que se tenga una carta que lo haga más rápido. Por lo tanto el que tenga mayor velocidad atacará primero y el otro tendrá que defenderse y luego atacará. Luego de que ambos bandos ataquen se acaba la ronda.

- **Fase de Ataque:** Solo las cartas de tipo *Ataque* e *Híbridas* pueden infligir daño. Durante esta fase se activan tanto los ataques normales como las habilidades especiales de las cartas.
- **Fase de Defensa:** Todas las cartas reciben daño según su rol, luego de que la inteligencia artificial ejecute (o no) su habilidad especial:
  - Ataque: Cada carta recibe el daño completo de la IA, solo en caso de que no queden cartas estructura o mixtas.
  - Estructuras y Mixtas: Cada carta recibe daño según la siguiente fórmula:

$$Ataque_{ef} = \frac{Ataque_{total}}{N_{Estructuras} + N_{Mixtas}}$$

    La cual indica que el daño efectivo depende del daño total y de la cantidad de *Estructuras* e *Híbridas* que estén con vida en el Mazo durante la ronda. Luego este valor de ataque debe considerar su ponderador de defensa. Al recibir daño bloquean todo el daño que podrían recibir las cartas tropa. En caso de que el daño sea superior a la vida restante de una carta, simplemente fallece, el daño no se pasa a otras cartas.

Dentro de estas fases pueden suceder cambios de estadísticas o ponderadores de las diversas entidades. Estas deben mantenerse hasta el fin de la pelea. En caso de que una carta haya fallecido, debe removerse del mazo del jugador y las estadísticas se reinician y deben poder ser revividas en la tienda.

6.3. Resolución de la Ronda

Al finalizar el combate se verifican las siguientes condiciones:

- Si la IA llega a 0 puntos de vida, el jugador avanza al siguiente enemigo, recibiendo una cantidad de `ORO_POR_VICTORIA`. Si no existe ninguna IA con vida, se declara la victoria en la consola y se da por terminado el juego. Si hubo cambio de estadísticas en alguna entidad debe ser reiniciada, excepto en el caso de la vida que se mantiene con lo que quedó en la ronda pasada.
- Si todas las cartas del jugador mueren, la partida termina en derrota.
- En cualquier otro caso, se pasa a la siguiente **Preparación de Ronda**, manteniendo el estado de las cartas sobrevivientes y recibiendo `ORO_POR_RONDA`.

7. Archivos

Para el desarrollo de *DCCartas contra la DCCatástrofe* será necesaria la lectura y carga de archivos. Estos contendrán la información relevante para la ejecución correcta del juego, incluyendo las cartas disponibles, las características de las IAs enemigas y los multiplicadores de ataque y defensa.

7.1. Archivo de Cartas

El archivo `cartas.csv` ubicado en la carpeta `data/` contiene la información de todas las cartas disponibles en el juego. Este archivo define las características de cada carta (tropas y estructuras) que el jugador podrá usar para enfrentarse a las IAs. A continuación se muestra un ejemplo de su estructura en la Figura 7.

```
# Versión simplificada para ajustarse al ancho del documento
nombre,tipo,vida_max,m_def,precio,probabilidad_especial,ataque,m_atq,descripcion_habilidad
Duendes,tropa,80,0.7,5,1.0,10,1.0,"Por cada unidad de daño que hacen (...)"
P.E.P.P.A.,tropa,200,0.9,15,1.0,30,1.2,"Cuando ataca, se sana en un CURE_PEPPA% (...)"
Cañón,estructura,300,1.2,12,1.0,0,0,"Esta estructura reduce las defensas (...)"
Torre DIE,estructura,250,1.1,14,0.25,0,0,"Esta carta es un préstamo (...)"
```

Figura 7: Fragmento del archivo `cartas.csv` (nombres de campos abreviados para visualización)

El archivo `cartas.csv` contiene los siguientes campos:

Atributo	**Descripción**
`nombre`	Identificador único de la carta
`tipo`	Define si la carta es una tropa, estructura o mixta
`vida_maxima`	La cantidad máxima de puntos de vida de la carta
`multiplicador_defensa`	Factor multiplicador de defensa que determina cuánto daño recibe la carta (valores entre 0 y 1.5)
`precio`	Costo en monedas para adquirir la carta en la tienda
`probabilidad_especial`	Probabilidad de activar la habilidad especial de la carta
`ataque`	Valor base de ataque de la carta (0 para cartas de tipo estructura)
`multiplicador_ataque`	Factor multiplicador de ataque que modifica el ataque base (0 para cartas de tipo estructura)
`descripcion_habilidad`	Descripción textual de la habilidad especial de la carta

Cuadro 5: Campos del archivo de cartas

Las cartas se clasifican en tres tipos principales:

- **Tropas**: Cartas con capacidad de ataque, generalmente con valores de vida más bajos pero alto poder ofensivo.
- **Estructuras**: Cartas defensivas con alta vida pero sin capacidad de ataque directo.
- **Mixtas**: Cartas híbridas que combinan características de tropas y estructuras. Cualquier tropa puede combinarse con cualquier estructura dinámicamente durante el juego a través del **Taller**, manteniendo las habilidades de ambas cartas base.

7.2. Archivos de Inteligencias Artificiales

El juego incluye tres archivos que definen las IAs enemigas para cada nivel de dificultad: `ias_facil.csv`, `ias_normal.csv` y `ias_dificil.csv`. Estos archivos están ubicados en la carpeta `data/` y contienen la información de las IAs que el jugador enfrentará. A continuación se muestra un ejemplo en la Figura 8:

```
nombre,vida_maxima,ataque,descripcion,probabilidad_especial,velocidad
CatGPT,343,34,"IA líder de DCCatástrofe, experto en manipulación y (...)",0.3,0.4
```

Figura 8: Ejemplo del archivo `ias_facil.csv`

Los archivos de IAs contienen los siguientes campos:

Atributo	Descripción
`nombre`	Nombre identificador único de la IA
`vida_maxima`	Puntos de vida totales de la IA
`ataque`	Valor base de daño que causa la IA
`descripcion`	Descripción textual de la habilidad especial
`probabilidad_especial`	Probabilidad de activar su habilidad especial
`velocidad`	Factor que determina la probabilidad de atacar primero

Cuadro 6: Campos de los archivos de IAs

Cada nivel de dificultad contiene un número diferente de IAs:

- **Fácil**: Contiene 1 IA con estadísticas más bajas.
- **Normal**: Contiene 3 IAs con estadísticas intermedias.
- **Difícil**: Contiene todas las 5 IAs con estadísticas más elevadas.

7.3. Archivo de Multiplicadores

El archivo `multiplicadores.csv` define cómo cada IA interactúa con los diferentes tipos de cartas. Este archivo se encuentra en la carpeta `data/` y es crucial para determinar la efectividad de las cartas contra cada IA y viceversa. A continuación se muestra un ejemplo en la Figura 9:

```
ia_nombre,carta_tipo,multiplicador_ataque,multiplicador_defensa
CatGPT,tropa,1.2,0.8
CatGPT,estructura,0.8,1.0
CatGPT,mixta,1.0,0.9
```

Figura 9: Fragmento del archivo `multiplicadores.csv`

El archivo `multiplicadores.csv` contiene los siguientes campos:

Atributo	Descripción
`ia_nombre`	Nombre de la IA correspondiente
`carta_tipo`	Tipo de carta (tropa, estructura o mixta)
`multiplicador_ataque`	Determina cuánto daño logra hacer la IA al atacar a cada tipo de carta. Un valor mayor a 1.0 significa que la IA hace más daño a ese tipo de carta.
`multiplicador_defensa`	Determina cuánto daño recibe la IA cuando es atacada por una carta del tipo especificado. Un valor menor a 1.0 significa que la IA recibe menos daño de ese tipo de carta. Para las estructuras es 0.0 ya que no pueden atacar.

Cuadro 7: Campos del archivo de multiplicadores

Estos multiplicadores son clave para el sistema de combate, ya que determinan cómo interactúan las IAs con las diferentes cartas del jugador, siguiendo las siguientes reglas generales:

- **multiplicador_ataque**: Determina cuánto daño logra hacer la IA al atacar a cada tipo de carta.
  - **Tropas**: Son más vulnerables a los ataques de la IA (valores >1.0).
  - **Estructuras**: Son menos vulnerables a los ataques de la IA (valores <1.0).
- **multiplicador_defensa**: Determina cuánto daño recibe la IA cuando es atacada por cada tipo de carta.
  - **Tropas**: Las IAs son menos vulnerables a los ataques de tropas (valores <1.0).
  - **Estructuras**: Ya que la estructura de Torre bomba hace daño al morir y ser esta su habilidad, hace daño completo (1.0).

Por ejemplo, si el multiplicador de ataque es 1.2 para tropas contra CatGPT, significa que CatGPT hará un 20 % más de daño cuando ataque a las tropas. De manera similar, si el multiplicador de defensa es 0.8 para tropas, CatGPT recibirá solo el 80 % del daño base cuando sea atacada por tropas, lo que significa que las IAs son menos vulnerables a los ataques de tropas.

7.4. Carga y Uso de Archivos

Para implementar correctamente el sistema de cartas, se recomienda seguir estos pasos:

1. **Lectura de archivos**: Al iniciar el juego, se deben cargar todos los archivos CSV con la información necesaria.
2. **Creación de instancias**: Para cada entrada en el archivo `cartas.csv`, se debe crear un objeto de la clase correspondiente (Tropa o Estructura) con los atributos especificados.
3. **Creación dinámica de cartas mixtas**: Las cartas mixtas se crean durante el juego, cuando el jugador decide combinar una tropa y una estructura en el Taller. La carta resultante debe tener todos los atributos y habilidades de ambas cartas base.
4. **Aplicación de multiplicadores**: Durante el combate, los multiplicadores deben aplicarse según el tipo de carta y la IA involucrada, tomando los valores del archivo `multiplicadores.csv`.

Las constantes asociadas a las habilidades de las cartas (como `CURE_PEPPA`, `DEF_CAB`, etc.) deben estar definidas en el módulo de parámetros y deben utilizarse en lugar de valores fijos para facilitar la configuración y balance del juego.

8. Bonus: Guardar partida (+5 décimas)

En esta sección se decribe una funcionalidad **opcional** para tu tarea. Su correcta implementación conlleva una puntuación extra a tu calificación de esta entrega. Para poder obtener este bonus debes cumplir con los siguientes requerimientos:

- La nota en tu tarea (sin bonus) debe ser **igual o superior a 4.0**.
- El bonus debe estar implementado **en su totalidad**, es decir, **no se dará puntaje intermedio**.
- Además, deberás explicitar en tu `README` si implementaste esta funcionalidad.

Lo que se solicita es que el programa ofrezca al jugador la opción de **Guardar partida** mientras se encuentra en el Menú principal. Debe habilitar la posibilidad de abandonar el juego **deteniendo el programa**, para reanudar la partida en otro momento.

Cuando el jugador decide **Guardar partida**, tu programa debe **generar un archivo** nuevo que contenga **toda la información** de la partida actual hasta ese momento, incluyendo las cartas en la colección y el mazo (y su estado), las rondas jugadas, el Oro, la información de la IA y la dificultad. Cuando se ha terminado de crear ese archivo, se le debe notificar al usuario que su partida ha sido guardada exitosamente.

El programa **no se detiene** cuando el jugador guarda la partida. El juego puede continuar normalmente, el jugador puede seguir atacando y pasando rondas cuantas veces lo desee, incluso podría terminar por completo el juego, ya sea al ganar o perder. Sin embargo, esto **no debe afectar** al estado de la partida que se guardó anteriormente.

El jugador puede guardar una partida cuantas veces desee, en cualquier momento en que se encuentre en el Menú principal, y para cada una de las veces que lo haga, deberás escribir la información del estado de la partida en el archivo mencionado. Queda a tu criterio si decides crear un archivo distinto para cada vez que se guarde una partida, o trabajar un solo archivo que se sobrescriba con cada guardado. En cualquier caso, el ó los nombres de estos archivos deben ser coherentes con el juego y deberás explicitar en tu `README` cómo se guarda la información de la partida dentro de estos archivos.

La recuperación de una partida previa se debe aplicar desde la ejecución del programa: en lugar de escribir la dificultad del juego como un argumento en la terminal, se debe escribir el **nombre del archivo** que contiene una partida previamente guardada. Es decir, tu programa deberá ser capaz de identificar correctamente las siguientes situaciones:

1. Situación 1: se ejecuta el programa entregando la dificultad (`facil`, `medio` ó `dificil`) como argumento en la terminal. En este caso, tu programa deberá iniciar una partida nueva y dirigir al jugador a la Selección inicial
2. Situación 2: se ejecuta el programa entregando **el nombre del archivo** que contiene una partida previamente guardada como argumento en la terminal. En este caso, tu programa deberá **cargar** los datos de esa partida, recuperando por completo el estado en que se encontraba el juego, y redirigir al Menú principal de dicha partida.

Además de implementar este guardado de partida deberás implementar un 'cheatcode' en el menú principal que, cuando se escriba 'veotodo' en vez de una opción del menú, se muestre toda la información actual de la partida.

9. `.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.

10. Importante: Corrección de la tarea

En el **siguiente enlace** 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.

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**.

11. 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 3.12.X con X mayor o igual a 0.
- 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`.
- 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 archivo `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 o bien incluirlo pero que se encuentre vacío 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)**.**