# TP3 - Simulador de batallas de Gunplas

## Introduccion

La empresa Banzai Mamco nos contrató para realizar un producto ultra secreto. En conmemoración del lanzamiento del New Gundam Breaker este 22 de Junio, la empresa quiere implementar una versión reducida del juego que pueda ayudar a atraer gente a comprar la versión completa del juego. En esta ocasión particular, intentan atraer puntualmente a programadores, asi que creyeron que sería una buena idea hacer un simulador de combates donde cada uno pudiese crear sus propias inteligencias artificiales y competir entre ellas.

El CEO de la empresa no quiere prestarnos ni una versión completa del juego (podrían regalarnos una, ¿no?) antes de la fecha de salida para evitar posibles filtraciones. Dijo que deberíamos conformarnos con lo que vemos en los trailers del juego:

- Trailer 1
- Trailer 2

En compensación nos ofreció auspiciar un torneo el mismo dia del lanzamiento, con premios para las mejores inteligencias artificiales creadas. De cualquier forma, aun no aclaró cuáles serían.

## Consigna

El objetivo del trabajo práctico es implementar:

- Un simulador de combates entre modelos Gundam (*Gunpla*)
- Una o más inteligencias artificiales que utilicen el simulador

## Reglas del juego

- Un *Gunpla* contiene obligatoriamente un esqueleto.
- Un *Gunpla* puede contener como máximo una parte de cada tipo.
- Un *Gunpla* puede equiparse adicionalmente armas hasta un máximo del número de slots de armas disponibles en el esqueleto (las armas disponibles en las partes no cuentan).
- Un esqueleto debe tener un valor de energía mayor a 0.
- Un esqueleto debe tener un valor de movilidad mayor o igual a 100.
- Una parte puede tener una cantidad mayor o igual a 0 de armas adosadas.
- Las partes y armas pueden tener valores negativos de velocidad, armadura, escudo o energía.
- Las partes y las armas tienen pesos mayores o iguales a 0.
- Las armas tienen uno de los 3 tipos de munición:

    - `FISICA`: la reducción de daño se calcula según la fórmula de reducción de daño físico (ver la sección de fórmulas más adelante).
    - `LASER`: la reducción de daño se calcula según la fórmula de reducción de daño láser.
    - `HADRON`: no tiene reducción de daño.
- Las armas son de uno de los 2 tipos:
    - `MELEE`: Luego de terminar el ataque, el oponente puede contraatacar con un arma.
    - `RANGO`: Luego de terminar el ataque, el oponente no puede contraatacar.
- Las armas pueden combinar sus ataques si:
    - Son del mismo tipo.
    - Son de la misma clase.
    - Tienen el mismo tipo de munición.
    - No está descargada.
- Luego de usar un arma, se debe esperar `Tiempo de recarga` turnos para volver a usarla (Vuelva a estar cargada).

# Ciclo de juego

Cada jugador es un piloto que controlará un *Gunpla*. Se separa a los pilotos en equipos que compiten entre sí; el equipo que logra ser el único con *Gunplas* activos es el ganador de la partida.

- A cada piloto se le da para elegir un esqueleto de una lista de esqueletos disponibles. El mismo esqueleto puede ser elegido por más de un piloto.
- Se generan partes y armas de forma aleatoria y se separan en distintas pilas según su tipo.
    - Las partes se generan ya con sus respectivas armas.
- Se dispone a los pilotos en una ronda, en orden aleatorio.
- Mientras haya partes o armas para elegir:
    - A cada piloto se le ofrecen las partes en el tope de cada pila.
    - El piloto elige un arma o parte, quitándose la misma de la pila correspondiente.
    - Se pasa las pilas al siguiente piloto en la ronda.
- Cada piloto equipa su *Gunpla* con algunas de las partes elegidas (las partes no utilizadas se descartan).
- Se ordena a los pilotos según la velocidad de sus *Gunpla* y se los encola en la cola de turnos.
- Mientras haya dos o más equipos con *Gunplas* activos:
    - Se le da el turno al siguiente piloto.
    - El piloto elige a quien atacar.
    - El piloto elige el arma con la que atacar.
    - El piloto ataca al *Gunpla* del enemigo con su *Gunpla* (aplicando el algoritmo de combinación de armas y cálculo de daño explicado más adelante).

- Si el daño fue nulo:
  - Se encola un turno extra del enemigo.
- Si la energía restante del *Gunpla* enemigo es negativa y de valor absoluto mayor al 5% de la energía máxima del mismo:
  - Se encola un turno extra del jugador actual.
- Si el arma elegida fue de tipo `MELEE` y el oponente no fue destruido:
  - El oponente contraataca: elige un arma y la utiliza como un ataque normal. No aplican reglas de turnos extra o combinación de armas.
- Se encola un turno del jugador actual.

## Clases a implementar

Para el trabajo es necesario implementar mínimamente las siguientes clases con los métodos detallados. El resto del diseño queda a criterio de los alumnos. los nombres de las clases no necesitan ser respetados, pero sí las firmas de los métodos (es decir, los nombres y argumentos que reciben).

La inteligencia sólamente puede acceder a los métodos abajo descriptos.

### Clase `Gunpla`

Representa un *Gunpla*. Un *Gunpla* esta compuesto de un `Esqueleto`, un conjunto de partes y un conjunto de armas.

| Atributo | Método | Descripción |
|---|---|---|
| Peso | `get_peso()` | Devuelve el peso total del *Gunpla*. Un *Gunpla* pesa lo que pesa la sumatoria de sus partes y armas |
| Armadura | `get_armadura()` | Devuelve la armadura total del *Gunpla*. Un *Gunpla* tiene tanta armadura como la sumatoria de la armadura de sus partes y armas |
| Escudo | `get_escudo()` | Devuelve el escudo total del *Gunpla*. Un *Gunpla* tiene tanto escudo como la sumatoria del escudo de sus partes y armas |
| Velocidad | `get_velocidad()` | Devuelve la velocidad total del *Gunpla*. Un *Gunpla* tiene tanta velocidad como la sumatoria de las velocidades de sus partes y esqueleto |

| Atributo | Método | Descripción |
|---|---|---|
| Energía | `get_energia()` | Devuelve la energía total del *Gunpla*. Un *Gunpla* tiene tanta energía como la sumatoria de la energía de sus partes, armas y esqueleto |
| Energía restante | `get_energia_restante()` | Devuelve la energía que le resta al *Gunpla* |
| Movilidad | `get_movilidad()` | Devuelve la movilidad de un *Gunpla*. Se calcula según la fórmula descripta en la seccion de fórmulas |
| Armamento | `get_armamento()` | Devuelve una lista con todas las armas adosadas al *Gunpla* (Se incluyen las armas disponibles en las partes) |

## Clase `Esqueleto`

Representa el esqueleto interno del *Gunpla*.

| Atributo | Método | Descripción |
|---|---|---|
| Velocidad | `get_velocidad()` | Devuelve la velocidad del esqueleto. Es un valor fijo |
| Energía | `get_energia()` | Devuelve la energía del esqueleto. Es un valor fijo |
| Movilidad | `get_movilidad()` | Devuelve la movilidad del esqueleto. Es un valor fijo |
| Slots | `get_cantidad_slots()` | Devuelve la cantidad de slots (ranuras) para armas que tiene el esqueleto. Es un valor fijo |

## Clase `Parte`

Representa una parte de un *Gunpla*.

| Atributo | Método | Descripción |
|---|---|---|
| Peso | `get_peso()` | Devuelve el peso total de la parte. Una parte pesa lo que pesa la sumatoria de sus armas más el peso base de la parte |
| Armadura | `get_armadura()` | Devuelve la armadura total de la parte. Una parte tiene tanta armadura como la sumatoria de la armadura de sus armas más la armadura base de la parte |

| Atributo | Método | Descripción |
|---|---|---|
| Escudo | `get_escudo()` | Devuelve el escudo total de la parte. Una parte tiene tanto escudo como la sumatoria del escudo de sus armas más el escudo base de la parte |
| Velocidad | `get_velocidad()` | Devuelve la velocidad de la parte. Es un valor fijo |
| Energía | `get_energia()` | Devuelve la energía total de la parte. La parte tiene tanta energía como la sumatoria de la energía de sus armas más la energía base de la parte |
| Armamento | `get_armamento()` | Devuelve una lista con todas las armas adosadas a la parte |
| Tipo de parte | `get_tipo_parte()` | Devuelve una cadena que representa el tipo de parte. Ej: `"Backpack"` |

## Clase `Arma`

Representa un arma.

| Atributo | Método | Descripción |
|---|---|---|
| Peso | `get_peso()` | Devuelve el peso del arma. Es un valor fijo |
| Armadura | `get_armadura()` | Devuelve la armadura del arma. Es un valor fijo |
| Escudo | `get_escudo()` | Devuelve el escudo del arma. Es un valor fijo |
| Velocidad | `get_velocidad()` | Devuelve la velocidad del arma. Es un valor fijo |
| Energía | `get_energia()` | Devuelve la energía del arma. Es un valor fijo |
| Tipo de munición | `get_tipo_municion()` | Devuelve el tipo de munición del arma: `"FISICA"\|"LASER"\|"HADRON"` |
| Tipo de arma | `get_tipo()` | Devuelve el tipo del arma: `"MELEE"\|"RANGO"` |
| Clase de arma | `get_clase()` | Devuelve la clase del arma, la cual es un string. Ejemplo `"GN Blade"` |
| Daño | `get_daño()` | Devuelve el daño de un ataque del arma. Es un valor fijo |

| Atributo | Método | Descripción |
|---|---|---|
| Hits | `get_hits()` | Devuelve la cantidad de veces que puede atacar un arma en un turno. Es un valor fijo |
| Precisión | `get_precision()` | Devuelve la precisión del arma |
| Tiempo de recarga | `get_tiempo_recarga()` | Devuelve la cantidad de turnos que tarda un arma en estar lista |
| Disponible | `esta_lista()` | Devuelve si el arma es capaz de ser utilizada en este turno o no |
| Tipo de parte | `get_tipo_parte()` | Devuelve el tipo de parte de un arma. **Siempre** es `"Arma"` |

## Clase `Piloto`

Inteligencia artificial para controlar un *Gunpla*.

| Método | Descripción |
|---|---|
| `__init__()` | Crea un piloto y no recibe ningun parámetro |
| `set_gunpla(gunpla)` | Asigna un *Gunpla* a un piloto |
| `get_gunpla()` | Devuelve el *Gunpla* asociado al piloto |
| `elegir_esqueleto(lista_esqueletos)` | Dada una lista con esqueletos, devuelve el índice del esqueleto a utilizar |
| `elegir_parte(partes_disponibles)` | Dado un diccionario: {`tipo_parte`: `parte`}, devuelve el tipo de parte que quiere elegir. Este metodo se utiliza para ir eligiendo de a una las partes que se van a reservar para cada piloto, de entre las cuales va a poder elegir para armar su modelo |
| `elegir_combinacion(partes_reservadas)` | Dada una lista con partes previamente reservadas, devuelve una lista con las partes a utilizar para construir el *Gunpla*. Este metodo se utiliza para elegir las partes que se van a utilizar en el modelo de entre las que se |

| Método | Descripción |
| --- | --- |
| | reservaron previamente para cada piloto. |
| `elegir_oponente(oponentes)` | Devuelve el índice del *Gunpla* al cual se decide atacar de la lista de oponentes pasada |
| `elegir_arma(oponente)` | Devuelve el arma con la cual se decide atacar al oponente |

**Aclaracion**: Los metodos de elegir no modifican ni agregan partes o armas. El almacenamiento de las partes elegidas asi como la combinacion de las partes elegidas en un *Gunpla* funcionan por fuera del piloto.

# Fórmulas

## Movilidad

Siendo `base` la movilidad del esqueleto, `peso` el peso del *Gunpla* y `velocidad` la velocidad del *Gunpla*:

```
movilidad = (base - peso / 2 + velocidad * 3) / base
```

La `movilidad` tiene un límite superior de 1 e inferior de 0.

## Reducción de daño físico

Siendo `daño` el daño recibido y `armadura` la armadura del *Gunpla*:

```
daño reducido = daño - armadura
```

El `daño reducido` tiene un límite inferior de 0.

## Reducción de daño láser

Siendo `daño` el daño recibido y `escudo` el escudo del *Gunpla*:

```
daño reducido = daño - daño * escudo
```

El `daño reducido` no tiene límite. Si es negativo, implica que aumenta la energía del *Gunpla*.

## Algoritmo de cálculo de daño

Atacante:

- Se usa `Hit` veces el arma. Cada uso genera `Daño` con un `Precisión%` de probabilidad (sino, el daño es 0).
- Con un `(25 * Precisión)%` de probabilidad el daño se multiplica por 1,5.
- Con una probabilidad, puede combinar su ataque con el de otra arma combinable, aplicando el mismo algoritmo del atacante de forma recursiva:
    - Si el arma es `MELEE`, la probabilidad es del 40%.
    - Si el arma es `RANGO`, la probabilidad es del 25%.
    - Si es un contraataque, la probabilidad es siempre nula.

Defensor:

- Con un `(80 * Movilidad)%` de probabilidad, evade el daño completamente.
- Reduce el daño según el tipo.
- Absorbe el daño restante.

# Sugerencias de implementacion

## Implementacion minima de los metodos elegir del piloto

Los metodos "elegir" se pueden implementar todos con

```
return random.choice(<secuencia de donde elegir>)
```

para implementar una inteligencia simple para probar.

## Determinacion de los valores de cada atributo

Se recomienda inicializar los valores de los atributos de las partes esqueletos y armas usando:

```
random.randint(inicio, fin)
```

de forma que la simulacion quede balanceada.

# Criterios de aprobación

A continuación se describen los criterios y lineamientos que deben respetarse en el desarrollo del trabajo.

# Informe

El informe debe consistir en una descripción del **diseño** del programa.

Debe recordarse que la etapa de diseño es *anterior a la implementación*, por lo tanto debe describirse, utilizando texto y/o diagramas, cómo se va a estructurar el código para cumplir con las especificaciones de la consigna.

Algunas preguntas que deberían responderse:

- A grandes rasgos, ¿cuales son las estrategias utilizadas para la IA? ¿Qué estructuras de datos se utilizan?
- ¿Qué estructuras de datos se utilizaron en el diseño del simulador?

# Código

Además de satisfacer las especificaciones de la consigna, el código entregado debe cumplir los siguientes requerimientos:

- El código debe ser claro y legible.
- El código debe estructurarse en funciones y, cuando corresponda, módulos. Las funciones deben definirse de la manera más genérica posible.
- Todas las funciones deben estar adecuadamente documentadas, y donde sea necesario el código debe estar acompañado de comentarios.

# Entrega

La entrega del trabajo consiste en:

- El informe y código fuente impresos. Para el código fuente utilizar una tipografía `monoespacio`.
- El informe en formato *PDF*.
- Una versión digital de todos archivos `.py` de código, separados del informe. En el caso de ser más de un archivo, comprimidos en un `.zip`.
- Una versión digital con solo el codigo de los pilotos en formato `.py` comprimidos en un `.zip`. Opcionalmente se puede agregar al `zip` un archivo `.txt` con el nombre de fantasia asociado a cada inteligencia a utilizar durante el torneo

El informe y código fuente impreso deben entregarse en clase. Los dos últimos (PDF y código fuente) deben enviarse utilizando el formulario de entregas.

Este trabajo práctico se desarrolla en forma **grupal**. El plazo de entrega vence el **Viernes 22 de Junio de 2018**.