

Materia:	Programación I		
Nivel:	1º Cuatrimestre		
Tipo de Examen:	Segundo Parcial		
Apellido <sup>(1)</sup> :		Fecha:	
Nombre/s <sup>(1)</sup> :		Docente a cargo <sup>(2)</sup> :	Falcone, Samaniego
División <sup>(1)</sup> :		Nota <sup>(2)</sup> :	
DNI <sup>(1)</sup> :		Firma <sup>(2)</sup> :	

(1) Campos a completar solo por el estudiante en caso de imprimir este enunciado en papel.

(2) Campos a completar solo por el docente en caso de imprimir este enunciado en papel.

## Dragon Ball Z Trading Card Game [TCG]



## Menú de opciones



## Ranking



### Sobre el juego:

La partida consta de una batalla de cartas entre el jugador y la computadora, cada participante tendrá un mazo de cartas con cartas aleatorias. Cada carta posee 4 stats: **HP**, **Attack**, **Defense** y un **Bonus** numérico según la cantidad de estrellas que tenga la esfera del dragon en la esquina inferior izquierda de la carta (la suma de los stats de todo el mazo hacen los stats del participante, el total de HP, Attack y Defense). Cada vez que se juega una mano se medirán los ataques de las cartas del jugador y de la computadora siendo ganadora la carta que más ataque tenga (Sumando el bonus a dicho ataque). La partida termina cuando se cumpla **al menos uno** de los siguientes indicadores:

- Uno de los participantes se queda sin cartas y su HP es inferior al del oponente.
- El HP de un participante llega a 0
- El tiempo de partida llega a 0 (A partir de allí gana quien mas HP tenga)

### Referencia:



## Sobre los comodines

El participante contará con 2 comodines (de 1 solo uso):



Cada comodín será de un solo uso, luego de eso el comodín/buff no se podrá volver a activar en la misma partida. (El botón debe dejar de ser visible o simplemente no hacer nada al darle click)

**HEAL:** Se recuperará toda la vida perdida al valor que tenía al iniciar la partida.

**SHIELD:** Se dará un “escudo” que protegerá al participante cuando pierda la jugada, haciendo que el daño vaya al enemigo en vez de sí mismo. Si en una mano el jugador hubiera perdido (menos atk que el enemigo) el “SHIELD” lo salva y hace que el enemigo pierda todos los stats de su propia carta (sumándole el bonus), para que se entienda mejor: el clásico “espejito rebotin”.

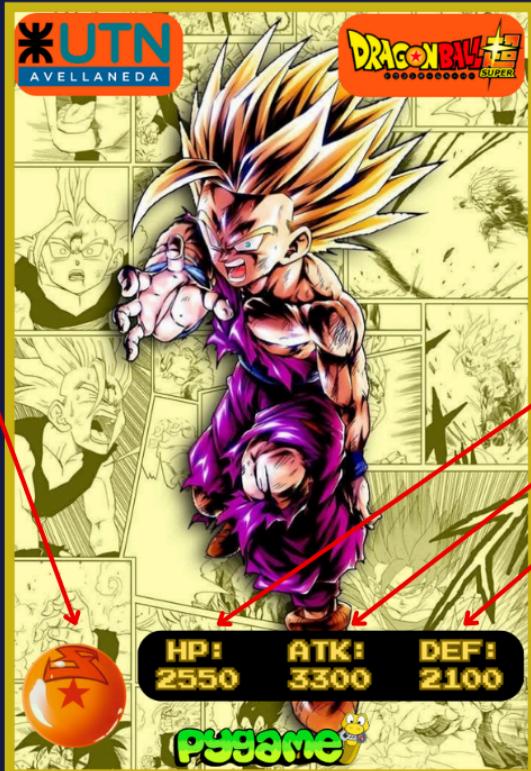
## Sobre las cartas:

### Bonus

Según cantidad de estrellas

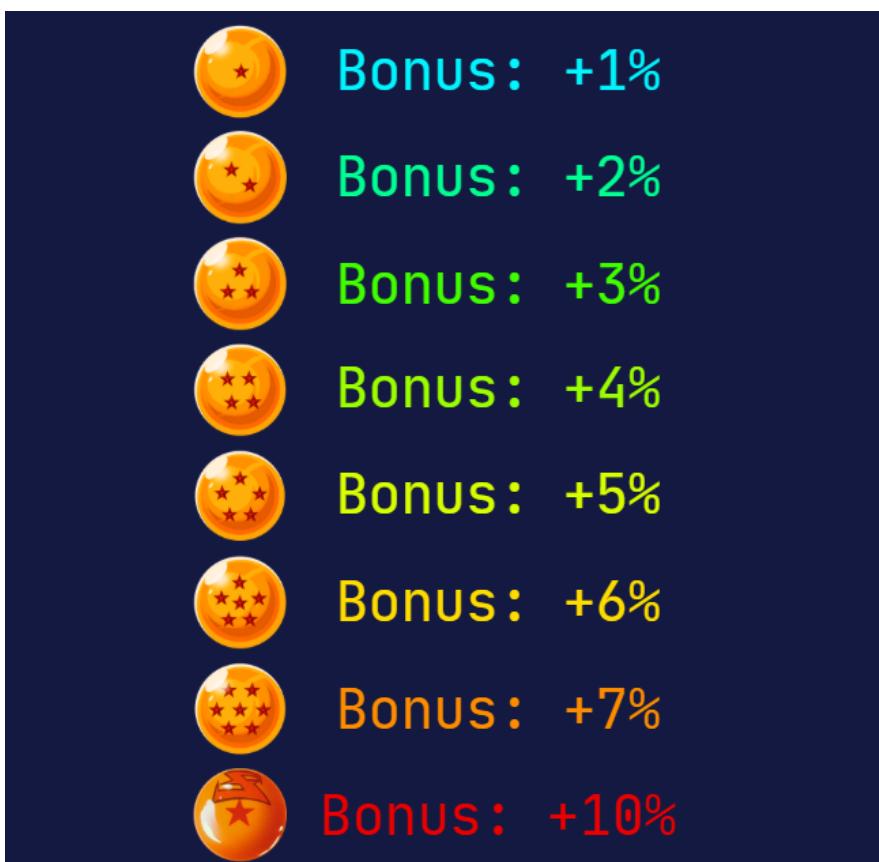
- 1★: ATK + 1%
- 2★: ATK + 2%
- 3★: ATK + 3%
- 4★: ATK + 4%
- 5★: ATK + 5%
- 6★: ATK + 6%
- 7★: ATK + 7%
- Z★: ATK + 10%

En este ejemplo:  
Z★: Al ataque se le suma el 10% para realizar las comparaciones de daño



Vida  
Ataque  
Defensa

## Sobre los bonus de las cartas:



Estos bonus se aplican al **ataque** al momento de realizar la comparación de daño entre la carta del jugador y la del oponente. También se aplica al momento de realizar el cálculo para restar vida (HP), ataque (ATK) y defensa (DEF) al participante que perdió la jugada.

## Sobre las pantallas

El juego debe tener las siguientes pantallas:

- Opciones (Para activar o desactivar el sonido)
- Ranking
- Menú Principal
- Ingreso de nombre (Al finalizar la partida, para luego ver el nombre y puntaje en el ranking)
- Pantalla de la partida (Donde se realizará el combate de cartas)

## Sobre las configuraciones

El juego debe leer datos de setup y de los mazos de un archivo JSON. Podés ayudarte con un módulo de variables para las rutas de imágenes o sonidos pero el grueso de configuraciones debe venir de un archivo JSON.

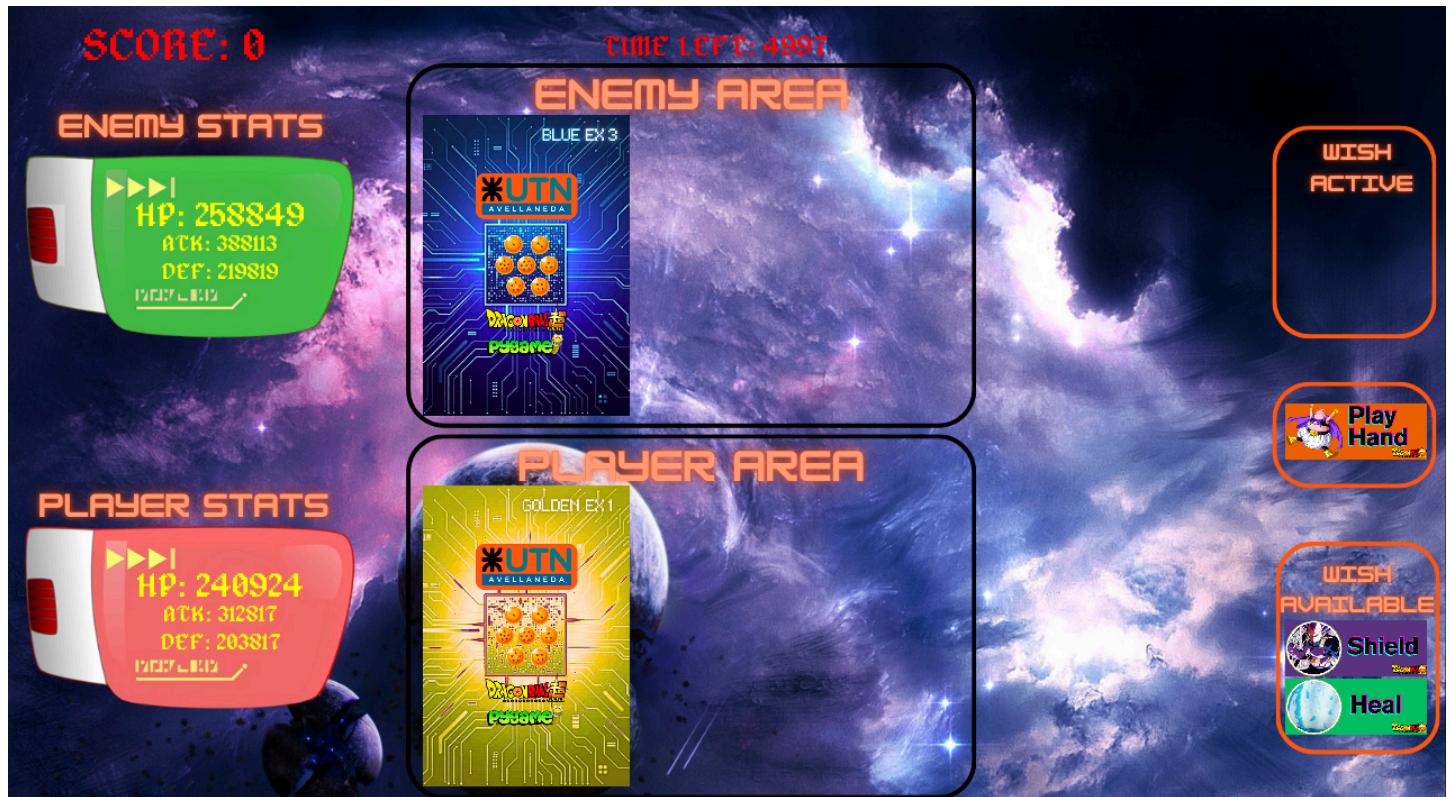
Ejemplo de JSON (Cantidad de cartas por mazo que recibe cada jugador)

```
(ɔ) configs.json > ...
...
1  {
2      "nivel_1": {
3          "cantidades": {
4              "green_deck_expansion_1": 10,
5              "green_deck_expansion_2": 10,
6              "green_deck_expansion_3": 10,
7              "blue_deck_expansion_1": 5,
8              "blue_deck_expansion_2": 5,
9              "blue_deck_expansion_3": 5,
10             "red_deck_expansion_1": 5,
11             "red_deck_expansion_2": 5,
12             "red_deck_expansion_3": 5,
13             "purple_deck_expansion_1": 5,
14             "purple_deck_expansion_2": 5,
15             "silver_deck_expansion_1": 5,
16             "silver_deck_expansion_2": 5,
17             "silver_deck_expansion_3": 5,
18             "golden_deck_expansion_1": 2,
19             "black_deck_expansion_1": 2,
20             "platinum_deck_expansion_1": 2
21         }
22     }
23 }
24 }
```

**Aclaración:** Pueden agregar más configuraciones al JSON si así lo desea

## Modalidad de batalla

Al momento de iniciar el nivel, cada player tendrá una cantidad de HP, ATK y DEF según la sumatoria de stats de las cartas de su mazo (No se debe aplicar el bonus de cada carta en este paso)



Al momento de jugar cada mano (una mano es considerada cuando una carta de cada participante está dada vuelta en sus respectivas áreas de juego) se deben comparar los ataques de cada carta (en este punto hay que sumarle las bonificaciones a los ataques según la cantidad de estrellas que tenga la esfera de la parte inferior izquierda explicado en la página anterior). El participante con el ataque más bajo será el perdedor de la mano. La carta perdedora descontará sus stats (sumándole el bonus a los 3 stats) al jugador perdedor. Como dicha cantidad será un número flotante, te recomiendo que lo parsees a entero para que en el visor de stats se pueda ver siempre un número entero.



En este ejemplo, la carta perdedora (del enemigo) tiene de stats:

HP: 2400 (+ 2% = 2448)

ATK: 3300 (+ 2% = 3366)

DEF: 1900 (+ 2% = 1938)

**Con lo cual, al momento de restarle los stats que posee la carta perdedora a los stats del enemigo, se hará con ese bonus (+2%) aplicado (parseado a entero).**

**TIP:** Para que exista una posibilidad de que uno de los dos participantes llegue con el HP a 0, se puede crear una función que elija un valor de forma aleatoria (al momento de restar stats al perdedor) el cual indique si ese ataque es un “golpe crítico”, haciendo que el daño a quitar del participante perdedor sea X2 o X3 (doble o triple). Podés ayudarte con el **módulo random** usando **random.choice()**

## Respecto a los puntos

Cada mano que el jugador gane, sumará puntos. Tienen la libertad de elegir la cantidad de puntos que el jugador puede ganar por mano ganada (incluyendo la mano donde se usa el comodín Shield).

**Pueden elegir entre las modalidades listadas a continuación:**

- *Un puntaje fijo por mano ganada*
- *Un cálculo entre el ataque de la carta ganadora - la defensa de la carta perdedora (para variar aun más el puntaje)*
- *Alguna modalidad elegida por el/la alumno/a*
- *Se puede sumar un puntaje bonus según el tiempo que queda de partida (En caso de no ser 0)*

## Finalización del juego

La partida termina cuando se cumpla **al menos uno** de los siguientes indicadores:

- Uno de los participantes se queda sin cartas y su HP es inferior al del oponente.
- El HP de un participante llega a 0
- El tiempo de partida llega a 0 (A partir de allí gana quien mas HP tenga)

En cualquiera de los 3 casos, se redireccionará a la pantalla de escritura de nombre (donde debe verse el puntaje obtenido en la última partida) y luego de dar click a continuar, se redireccionará a la pantalla de ranking, donde tendrá que verse el puntaje y nombre del jugador (en caso de estar entre los mejores 10 puntajes).

**⚠ IMPORTANTE ⚠**: El código debe estar subido a un **repositorio de Github** el cual respete el siguiente formato de nombre:

**TPFinal\_Div317\_Apellido\_Nombre** (respetando las mayúsculas, por ejemplo:  
TPFinal\_Div317\_Argento\_Pepe)

**Opcional: Grabar un video gameplay donde se muestre todas las funcionalidades del juego (no más de 5 minutos de video) y subirlo a YouTube.**

## Condiciones de Aprobación No Directa (Nota 4):

Para lograr nota 4, deberán desarrollar la lógica del juego en consola. La interacción con el usuario debe ser prolífica.

Contenidos que se deben aplicar obligatoriamente:

- Manejo avanzado de TDA: listas, diccionarios, sets y tuplas.
- Manejo de strings.

- Lectura y escritura de archivos de texto (txt, **csv, json**).
- Funciones: teniendo en cuenta el paradigma funcional.

Pueden aplicar cualquier concepto visto hasta el primer parcial. Por ejemplo ordenamiento, funciones recursivas, etc. Tiene que estar estéticamente agradable.

### Objetivos de Aprobación No Directa (Calificación de 4 a 5 puntos):

Que el estudiante:

- 1) Aplique técnicas de programación que involucre el adecuado manejo de listas, diccionarios, sets y tuplas.
- 2) Logre leer y escribir archivos de distintos tipos, con las técnicas vistas en clase.
- 3) Aplique de manera eficiente funciones, teniendo en cuenta el paradigma funcional y los principios DRY (Don't Repeat Yourself).
- 4) Logre desarrollar el juego en consola, haciendo uso de las distintas técnicas de programación provistas en la cátedra.
- 5)

**Nota: para resolver la lógica del juego, podrán aplicar cualquier concepto visto en la primera parte de la materia.**

### Objetivos de Aprobación Directa (Calificación de 6 a 10 puntos):

Que el estudiante:

- 1) Aplique técnicas de programación relacionadas con la biblioteca Pygame:
  - Configuraciones
  - Posicionamiento
  - Manipulacion de imagenes
  - Movimientos
  - Sonidos
  - Eventos
  - Colisiones
- 2) Logre aplicar los conceptos de paradigma funcional, haciendo uso de técnicas de modularización, para la reutilización de código y organización del mismo.
- 3) Logre desarrollar un juego en pygame estéticamente prolijo y funcional, sin descuidar los objetivos establecidos para la aprobación no directa.

### Dinámica del juego [Dragon Ball Trading Card Game](#)

Material descargable: [Link](#)

Pueden descargar material para la interfaz gratuitamente desde los siguientes enlaces:

- **Material:**

- **Iconografía:**
  - [FlatIcon](#)
  - [PngWing](#)
  - [FreePick](#)
- **Edición de imágenes:**
  - [Canva](#)
  - [Photopea](#)
- **Edición de audio gratuita**
  - [Audacity](#)