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

IIC2233 Programación Avanzada (2019-2)

Tarea 01

Entrega

- **Avance de tarea**
  - **Fecha y hora:** miércoles 4 de septiembre de 2019, 20:00
  - **Lugar:** GitHub — Carpeta: `Tareas/T01/`
- **Tarea**
  - **Fecha y hora:** sábado 14 de septiembre de 2019, 20:00
  - **Lugar:** Repositorio personal de GitHub — Carpeta: `Tareas/T01/`
- `README.md`
  - **Fecha y hora:** lunes 16 de septiembre de 2019, 20:00
  - **Lugar:** Repositorio personal de GitHub — Carpeta: `Tareas/T01/`

Objetivos

- Aplicar conceptos de programación orientada a objetos (POO) para modelar y resolver un problema.
- Utilizar correctamente *properties*, clases abstractas, polimorfismo y multiherencia.
- Comunicar diseños orientados a objetos a través de documentación externa.
- Procesar *input* del usuario de forma robusta, manejando potenciales errores de formato.

Índice

1. *Initial P*

Tu programa de **LegoSweeper** fue un éxito: Enzo logró subir las notas a tiempo y mantener su cargo de **Líder Supremo**. Sin embargo, esto no significa la derrota total del **Dr. Pinto**, quien propone un nuevo desafío para quedarse con el puesto de jefe: una carrera de autos a máxima velocidad.

Debido a restricciones de tiempo ~~y de presupuesto~~, no fue posible conseguir autos de verdad para la competencia, por lo que te piden a tí, un programador experto en POO, que crees un programa que permita simular la carrera en su totalidad y así saber quien es el justo ganador.

Figura 1: Cualquier parecido con *Initial D* es mera coincidencia.

2. Flujo del juego

Tu misión es crear un programa que permita a un usuario entrar y competir en la simulación de una carrera. La ejecución e interacción del juego será mediante consola, por lo que en ésta aparecerán todas las instrucciones para el jugador.

El objetivo del jugador de *Initial P* es competir y ganar carreras. Cada vez que el jugador gane una competencia, obtendrá dinero y experiencia. El dinero permitirá comprar mejoras para sus vehículos, o comprar uno nuevo; mientras que la experiencia aumentará las habilidades del piloto.

Al iniciar el programa se abrirá el Menú de sesión, el cual permitirá al jugador comenzar o reanudar una partida. Una vez iniciada la sesión se procederá a mostrar el Menú principal, el cual permitirá comprar vehículos e iniciar una carrera. Para cada opción, se debe mostrar el menú correspondiente: Menú de compra de vehículos o Menú de preparación de carrera.

Durante una competencia, se medirá el tiempo que demora cada competidor en dar una vuelta. Como recompensa parcial, el menor tiempo por vuelta obtendrá dinero y experiencia. Después de cada vuelta, se le dará al jugador la oportunidad de ir a los *Pits*[1], donde podrá reparar y mejorar su vehículo.

Una vez completadas todas las vueltas correspondientes a una carrera, se elegirá al ganador en base a sus tiempos y se premiará al que quede en el primer lugar. Posterior a esto, se volverá al menú de inicio donde el jugador podrá iniciar una nueva competencia. El juego no termina a menos de que el jugador seleccione la opción de salir, en el Menú principal.

[1] Los *Pits* son lugares donde los vehículos se detienen a realizar reparaciones, cambiar neumáticos o cargar combustibles en las carreras.

3. Menús

La interacción por consola se realice mediante **menús**. Cada menú muestra opciones disponibles al usuario, para luego recibir y procesar su decisión.

Todos los menús deben ser **a prueba de errores de usuario**, tener la opción de **volver atrás** o salir del juego cuando corresponda, y ser **intuitivos** de utilizar. El formato de los menús queda a tu criterio. Puedes crear más submenús adicionales a los siguientes, siempre y cuando se cumpla con las funcionalidades básicas. Como mínimo deberás implementar los siguientes menús:

3.1. Menú de sesión

Este es el menú en el cual se inicia la sesión del jugador, donde al usuario se le dará la opción de crear una nueva partida o cargar una partida existente. Detalles de la creación y cargado de sesiones se encuentran en Manejo de las partidas.

3.2. Menú principal

Después de iniciar sesión, se mostrará este menú. Se le presenta al jugador las opciones de: comprar nuevos vehículos; iniciar una carrera; guardar la partida; o salir del programa.

3.3. Menú de compra de vehículos

En este menú se debe mostrar el dinero actual del jugador y los vehículos disponibles para ser comprados junto a sus respectivos precios. Debe tener la opción de comprar uno de los vehículos o de regresar al menú de inicio. Un usuario puede tener múltiples vehículos del mismo tipo. Al comprar un vehículo se le debe asignar un nombre que debe ser único entre las posesiones del usuario.

A continuación, un ejemplo de cómo se podría mostrar esta información:[2]

```
Dinero actual: $5.260
Vehículos disponibles para comprar:
1) Automóvil        $550
2) Troncomóvil      $900
3) Motocicleta      $370
4) Bicicleta        $1.050

Ingrese el número del vehículo que desea comprar (ingrese 0 para regresar):
```

3.4. Menú de preparación de carrera

Si el jugador decide iniciar una carrera, primero deberá seleccionar la `Pista` donde desea participar y posteriormente elegir entre sus vehículos el que desea usar durante la corrida. Una vez seleccionados, se procederá a mostrar el Menú de carrera.

[2] Los ejemplos son sólo de referencia; puedes hacer los menús como más te acomode.

3.5. Menú de carrera

Una vez iniciada la carrera y cada vez que se complete una vuelta se debe mostrar este menú. Al completarse una vuelta, se le debe ofrecer al jugador la opción entrar a los *Pits* o no. Si el jugador decide entrar a los *Pits*, se debe mostrar Menú de los *Pits*, en caso contrario, se continúa la carrera.

3.6. Menú de los *Pits*

Aquí, se le deberán presentar el dinero actual que tiene el usuario, las piezas del vehículo que se pueden mejorar (junto a sus respectivos costos) y la opción de volver a la carrera. Queda a tu discreción si el jugador puede elegir **una** única mejora del menú de *Pits* por cada vuelta que recorre, o todas las que le permite su dinero por cada pasada por los *Pits*.

A continuación, un menú de ejemplo para un vehículo sin motor:

```
Dinero actual: $200
Partes a mejorar:
1) Chasis           $100
2) Carrocería       $50
3) Ruedas           $120
4) Zapatillas       $270

Ingrese el número de la parte a mejorar (ingrese 0 para regresar):
```

4. Flujo de la carrera

Una carrera consiste en dar un número fijo de vueltas a una pista en el menor tiempo posible. La velocidad de los competidores se basa en el tipo de vehículo, las mejoras de sus componentes, las habilidades y personalidad del piloto. Estas estadísticas también se verán afectadas por las diferentes pistas, variando así la velocidad final del vehículo.

En el recorrido también participarán contrincantes, que serán otros pilotos generados por tu programa a partir del archivo `contrincantes.csv`. La cantidad de contrincantes y quienes participarán en la carrera vienen definidos en el archivo `pistas.csv`[3].

En cada vuelta, tanto el piloto como sus contrincantes se podrán encontrar con objetos u obstáculos que podrán aumentar o disminuir su velocidad. Los obstáculos también dañarán al vehículo, afectando su `chasis` (explicado en `Vehiculo`). En cada vuelta existe la posibilidad de que el vehículo tenga un accidente[4], la que está dada por Accidentes durante las vueltas, por lo que en el caso de que el vehículo tenga un accidente, su `chasis` llegara a cero.

Si el `chasis` se destruye completamente[5], el jugador se deberá retirar de la carrera, volviendo al Menú principal. Las mejoras compradas durante las carreras se mantienen en todo momento, ya sea que ganes, pierdas una carrera o cierres la partida. Además, cada vez que se inicie una carrera, el vehículo se recuperará de todos sus daños obtenidos en las carreras anteriores.

Al finalizar cada vuelta, se le debe notificar al usuario:

[3] Del listado de contrincantes que aparece en la entrada del archivo, deberás elegir aleatoriamente la cantidad indicada.

[4] Para determinar si el evento ocurre, es recomendable que usen la función `random.random` para obtener un número aleatorio entre 0 y 1, el cual deberás comprobar si es mayor o menor que la probabilidad del evento.

[5] Es decir, su valor llega a ser 0.

- El número de la vuelta actual y la cantidad de vueltas restantes.
- Si su velocidad se vio afectada por algún objeto u obstáculo en la pista.
- Los participantes que siguen en la competencia ordenados según su posición en la carrera, junto al tiempo que les tomó dar una vuelta y el tiempo total acumulado.[6]
- Los competidores que se retiraron.

A continuación, un ejemplo de cómo se podría mostrar la información anterior:[7]

```
Vuelta: 5 de 7

Tu velocidad no se vio afectada durante la vuelta.

Orden de los competidores:
N° | Nombre             | Tiempo vuelta     | Tiempo acumulado
1  | Enzini             | 100.3 seg         | 345.7 seg
2  | Gioconcha          | 50.2 seg          | 402.3 seg
3  | Dr Pinto           | 130.4 seg         | 576.1 seg

Competidores descalificados:
- Wielandt el anti chef
```

En el caso de que el jugador quede en primer lugar al terminar una vuelta, se le entregará una cantidad de dinero definida según la fórmula Dinero por vuelta.

Antes de comenzar la siguiente vuelta, se deberá mostrar el Menú de carrera. En el caso de que se seleccione entrar a los *Pits*, se procederá a reparar el vehículo de forma automática, lo cual añadirá un tiempo de reparación dado por la fórmula Tiempo en los *Pits*, el cual será agregado al tiempo de la siguiente vuelta.

También se deberá presentar la opción de mejorar alguna de las partes del vehículo, por lo que se deberá mostrar el Menú de los *Pits*. Cada parte mejorable tiene un valor numérico asociado, que representa el nivel en el que se encuentra. Mejorar una parte del vehículo tendrá un costo monetario asociado según la característica a mejorar, lo cual aumentará la característica una cantidad definida de veces.

Una vez terminadas las reparaciones y mejoras en los *Pits*, se procederá a simular otra vuelta de la carrera. En el caso de que el jugador decida no entrar a los *Pits*, comenzará otra vuelta de forma inmediata sin ninguna mejora o reparación a su vehículo.

Este proceso se repetirá hasta que el jugador complete las vueltas necesarias para que termine la carrera o hasta que ya no pueda continuar en la carrera debido a los daños sufridos.

Al finalizar la carrera, se debe mostrar el orden final de llegada de los competidores; quien se haya demorado menos tiempo en completar todas las vueltas se declara el ganador y, si es el usuario, ganará dinero y experiencia.

5. Entidades

Para que estos ambiciosos participantes logren llevar a cabo la carrera, necesitan tener cómo competir y un lugar para desarrollarla. Es por esto que a continuación se presentan todas las entidades que debes modelar e implementar en tu programa.

[6] La forma en que muestras el orden de llegada queda a tu criterio.
[7] Para lograrlo puedes hacer uso `printf-style String Formatting`.

5.1. `Vehiculo`

Los vehículos son utilizados por los pilotos y son la principal herramienta para obtener la victoria. Dependiendo de su peso, sus componentes y de las mejoras que puedan adquirir, será más probable o menos probable completar cada vuelta en primer lugar. Las partes de un vehículo son:

- `chasis`: es el esqueleto del vehículo, la durabilidad del chasis está directamente relacionada a la vida útil del vehículo.
- `carrocería`: es el escudo del vehículo, te permitirá defenderte de los choques y ataques de tus viles contrincantes. De esta pieza depende el atributo de defensa.
- `ruedas`: son la parte que está en contacto directo con el suelo, lo que proporciona una tracción.
- `motor`: de estar presente en el vehículo, le proporciona la potencia necesaria para moverse.
- `zapatillas`: a falta de un motor, el vehículo viene con zapatillas integradas, las que permitirán al piloto poner todo su esfuerzo para moverse.

Al momento de comprar un vehículo, a cada una de estas partes y al peso del vehículo se le asignará un valor aleatorio uniforme[8] entre `MIN`[9] y `MAX` que son valores entregados dentro de un diccionario en el archivo `parametros.py`.

Por ejemplo, el `chasis` de una bicicleta tendrá una durabilidad entre `MIN` y `MAX`, estando estos valores en el diccionario `CHASIS` que está dentro del diccionario `BICICLETA`, y todo esto contenido en el archivo `parametros.py`.

Además si se quiere aplicar una mejora dentro de los *Pits*, cada parte tendrá un `COSTO` y un `EFECTO` asociado. El `EFECTO` será la cantidad de veces en que aumentarán la variables afectadas por las partes respectivas. Las constantes están indicadas en `parametros.py`.

Por ejemplo, si un vehículo tiene unas `ruedas` con 5 de tracción y el valor de `EFECTO` es 4, entonces la tracción final del vehículo será 20 (ya que es 5×4), y si se vuelve a mejorar la tracción quedará con un valor de 80 (20×4). Este proceso se puede repetir indefinidamente, mientras el jugador tenga el dinero suficiente para pagar el `COSTO` asociado cada vez que adquiere una mejora.

Para que cada piloto desarrolle su propia estrategia, existen diferentes tipos de vehículos que, dependiendo de la pista y su piloto, pueden funcionar mejor o peor durante la carrera. A continuación se presentan los distintos tipos de vehículos:

Automóvil: el automóvil, tal cual como se conoce, está compuesto por un `chasis`, una `carrocería`, cuatro `ruedas` y un `motor`. Se destaca por su robustez para enfrentar caminos rugosos y por alcanzar altas velocidades.

Troncomóvil: el troncomóvil es un vehículo muy especial, inspirado en Los Picapiedras. Posee lo mismo que un automóvil a excepción del `motor`, el cual fue sustituido por un par de `zapatillas`, que facilita el andar del piloto.

Bicicleta: la mejor amiga para algunos es la bicicleta, vehículo compuesto por un `chasis`, una `carrocería` y dos `ruedas`. Su potencia solo dependerá de qué tan bueno es su piloto y las `zapatillas` que traiga el vehículo.

Motocicleta: no puede faltar la motocicleta, que está compuesta por un `chasis`, una `carrocería`, dos `ruedas` y un `motor`. Perfecta para viajar solo y a grandes velocidades.

[8] La función `randint` de la librería `random` te podría ser útil.

[9] Las palabras escritas en `ESTE_FORMATO` son parámetros que tendrás que escribir e importar desde el archivo `parametros.py`.

5.2. `Pista`

Son los mapas o rutas que podrás elegir al comenzar una carrera. Cada pista tiene sus características, las cuales afectarán (de buena o mala forma) el tiempo por cada vuelta. Para esta carrera, las pistas existentes son:

Pista Helada: en esta pista la velocidad de tu vehículo disminuye, puesto que al tener mucho hielo, es más fácil resbalarse. Además, a los pilotos no les agrada mucho el frío, pero pueden soportarlo mejor mientras mayor sea su contextura. El efecto del frío en los pilotos es continuo, por lo que a mayor tiempo en la pista, mayor es la pérdida de velocidad. La cantidad de hielo en la pista viene dada en el archivo `pistas.csv`. En cuanto al efecto del frío en el conductor, viene dado por Efecto de la Hipotermia. Respecto a cómo la cantidad de hielo afecta en el cálculo en la velocidad, se especifica en Velocidad Recomendada.

Pista Rocosa: las rocas son las peores enemigas de los vehículos, es por esto que la gran presencia de piedras gigantes en esta pista va disminuyendo la durabilidad del chasis del vehículo, y tendrás que estar más tiempo en los *Pits* arreglándolo. Por otra parte, al haber rocas, la velocidad de tu vehículo se verá afectada. Al igual que en la pista anterior, la cantidad de rocas es entregada en el archivo `pistas.csv`. El cálculo para determinar como afectan las rocas al vehículo esta dado en Daño al vehículo. En cuanto al efecto en la velocidad, puedes encontrarlo en Velocidad Recomendada.

Pista Suprema: esta pista es una combinación malvada entre la Pista Helada y la Rocosa, los efectos de ésta también son una combinación de los efectos producidos en las pistas anteriores. Todos los datos de esta pista vienen dados en el archivo `pistas.csv`. Los cálculos para esta pista los encuentras en Efecto de la Hipotermia, Daño al vehículo y en Velocidad Recomendada.

5.3. `Piloto`

Los pilotos son quienes competirán en las carreras, con el eventual objetivo de quedarse con el tan preciado título de Jefe Supremo. Cada uno tiene las siguientes características que influirán en su desempeño:

Nivel de experiencia: el nivel de experiencia corresponde a un `int` mayor o igual a 0, el cual le ayudará a manejarse mejor en las pistas más difíciles. Este valor comienza siendo 0 para cualquier piloto nuevo.

Contextura: la contextura corresponde a un valor `int` aleatorio uniforme entre dos valores, siendo los rangos específicos para cada tipo de piloto. Esta característica influirá en el desempeño del piloto en las pistas de hielo al proporcionar resistencia al frío y así evitar la hipotermia.

Equilibrio: tomará un valor `int` aleatorio uniforme entre dos valores específicos para cada tipo de piloto. Indicará que tan adepto es para manejar vehículos de dos ruedas, mientras que los vehículos de cuatro ruedas no se verán afectados por esta característica.

Personalidad: corresponde a un `str`, que puede ser `'osado'` o `'precavido'`. La personalidad de los pilotos tendrá efecto en velocidad de conducción, la dificultad de control del vehículo y en la cantidad de experiencia ganada al final de la carrera.

Además, cada piloto de *Initial P* puede pertenecer a alguno de los siguientes equipos[10], lo cual afectará las características iniciales de todos los pilotos:

[10] Los equipos de los pilotos son permanentes, por lo que no se debe dar la posibilidad de cambiarlos.

Tareos: el área de tareas escuchó el rumor de la competencia y es por eso que su jefa, *la Gioconcha*, ha decidido unirse a la carrera. Aprovechando que *Enzini* y *Dr. Pinto* están ocupados con su rivalidad, ella buscará convertirse en la Líder Suprema. Los miembros de su equipo tienen una contextura que varía entre `26` y `45`, mientras que su equilibrio varía entre `36` y `55`. Por último, su personalidad es del tipo `'precavido'`.

Híbridos: el actual jefe supremo, *Enzini* está dispuesto a todo para mantener su cargo, por lo que junto a los híbridos decide enfrentarse a sus contrincantes. Sus integrantes se caracterizaran por tener contextura entre `35` y `54`, y un equilibrio que se encuentra entre `20` y `34`. Al ser un equipo con variadas habilidades, su personalidad tiene una probabilidad igual (50:50) de ser `'precavido'` u `'osado'`.

Docencios: a pesar de que su plan de los legos falló, el *Dr. Pinto* no se dará por vencido tan fácilmente, así que reunió a todos los docencios en su equipo para tratar de ganar la competencia. Este equipo se caracteriza por tener una contextura entre `44` y `60`, un equilibrio que varía entre `4` y `10` y, por supuesto, tendrán siempre una personalidad del tipo `'osado'`.

6. Archivos

Para lograr recrear las carreras deberás hacer uso de los siguientes archivos en formato CSV[11], los cuales vendrán separados por comas (`,`): `pistas.csv`, `contrincantes.csv`, `pilotos.csv` y `vehiculos.csv`. Considera que el contenido y el orden de las columnas de estos archivos puede variar, por lo que tu programa deberá adaptarse a dichos cambios. Además, estos archivos tendrán *encoding* UTF-8 y se espera que se mantenga a medida que se ejecute el programa[12].

Se te entregará un archivo `parametros.py` el cual deberás rellenar e **importar** correctamente durante la ejecución de tu programa.

6.1. Archivos estáticos

Para facilitar el testeo del programa, se te entregarán los archivos descritos a continuación. El contenido de estos archivos no debe cambiar durante la ejecución de tu programa. Su propósito es sólo ser usados en modo lectura.

6.1.1. `pistas.csv`

Contiene la información sobre todas las pistas disponibles.

[11] Se recomienda **NO** abrir los archivos `.csv` con Excel, ya que podrían desconfigurar los archivos. Para visualizarlos, se recomienda abrirlos con tu editor de texto.

[12] Para esto, recuerda usar el argumento `encoding='utf-8'` de la función `open` de Python

Nombre	Tipo de dato	Descripción
Nombre	`str`	Indica el nombre de la pista.
Tipo	`str`	Indica el tipo de la pista. Tiene tres valores posibles: `'pista hielo'`, `'pista rocosa'` y `'pista suprema'`.
Hielo	`int`	Indica la cantidad de hielo de la pista. Si no es una pista de hielo, el valor de este atributo no afecta ni debe ser leído. Puede que se omita o esté presente.
Rocas	`int`	Indica la cantidad de rocas de la pista. Si no es una pista rocosa, el valor de este atributo no afecta ni debe ser leído. Puede que se omita o esté presente.
Dificultad	`int`	Indica el nivel de dificultad de la pista.
NúmeroVueltas	`int`	Indica la cantidad de vueltas necesarias para terminar la carrera.
Contrincantes	`str`	Una lista con los nombres de los contrincantes, separados por punto y coma (`;`).
LargoPista	`int`	Indica el largo de una vuelta a la pista (distancia).

Ejemplo de una fila del archivo:

```
Pradera Nevada,pista hielo,32,,6,3,john;paul;lucia fernández;salmonella,1032
```

En este ejemplo, se omitió la variable `Rocas`. Considera que las siguientes líneas son equivalentes a la primera en cuanto a la información relevante para el programa.

```
Pradera Nevada,pista hielo,32,-,6,3,john;paul;lucia fernández;salmonella,1032
```

```
Pradera Nevada,pista hielo,32,2,6,3,john;paul;lucia fernández;salmonella,1032
```

En los tres casos, al ser una pista de hielo, la cantidad de rocas es irrelevante y no debe ser considerada.

6.1.2. `contrincantes.csv`

Contiene la información de todos los contrincantes de la carrera.

Nombre	Tipo de dato	Descripción
Nombre	`str`	Indica el nombre del piloto.
Nivel	`str`	Indica el nivel del piloto. Podrá tomar los valores de: `'principiante'`, `'experto'` y `'beato'`. No tiene un efecto real en el juego pero se debe informar al jugador contra quiénes compite.
Personalidad	`str`	Indica el valor de la personalidad del piloto. Tiene dos valores posibles: `'osado'` y `'precavido'`.
Contextura	`int`	Indica la contextura del piloto. Proporciona resistencia al frío.
Equilibrio	`int`	Indica el equilibrio del piloto, necesario para conducir vehículos de dos ruedas.
Experiencia	`int`	Indica la experiencia actual del piloto.
Equipo	`str`	Indica el equipo del piloto. Determina las estadísticas base de éste. Tiene tres valores posibles: `'Tareos'`, `'Híbridos'` y `'Docencios'`.

Ejemplo de una fila del archivo:

```
Raúl Álvarez,beato,precavido,14,15,99,Tareos
```

6.2. Archivos modificables

Éstos son archivos que deberás crear y que contendrán la información de las partidas guardadas, y por lo tanto su contenido puede cambiar durante la ejecución de tu programa. Cada vez que se cree una nueva partida o se compre un vehículo se deberá crear una nueva fila en el archivo correspondiente. Mientras que al guardar, deberán sobrescribir los valores correspondientes.

6.2.1. `pilotos.csv`

Contiene la información de todos los pilotos. A diferencia de los competidores, éstos no tienen nivel.

Nombre	Tipo de dato	Descripción
Nombre	`str`	Indica el nombre del piloto. Es el mismo nombre del jugador. Cada nombre es único entre los registrados.
Dinero	`int`	Indica el dinero que tiene el jugador.
Personalidad	`str`	Indica el valor de la personalidad del piloto. Tiene dos valores posibles: `'osado'` y `'precavido'`.
Contextura	`int`	Indica la contextura del piloto. Proporciona resistencia al frío.
Equilibrio	`int`	Indica el equilibrio del piloto, necesario para conducir vehículos de dos ruedas.
Experiencia	`int`	Indica la experiencia actual del piloto.
Equipo	`str`	Indica el equipo del piloto. Determina las estadísticas base de éste. Tiene tres valores posibles: `'Tareos'`, `'Híbridos'` y `'Docencios'`.

Ejemplo de una fila del archivo:

```
John Pendleton,2,osado,40,26,1,Híbridos
```

6.2.2. `vehículos.csv`

Contiene la información sobre todos los vehículos creados y quiénes son sus dueños. Un piloto no puede tener dos vehículos con el mismo nombre.

Nombre	Tipo de dato	Descripción
Nombre	`str`	Indica el nombre del vehículo. Es distinto a la categoría de éste.
Dueño	`str`	El nombre del dueño del vehículo.
Categoría	`str`	Indica la categoría del vehículo. Tiene cuatro valores posibles: `'automóvil'`, `'motocicleta'`, `'troncomóvil'` y `'bicicleta'`.
Chasis	`int`	Indica la durabilidad del chasis del vehículo.
Carrocería	`int`	Indica la defensa que proporciona la carrocería al vehículo.
Ruedas	`int`	Indica la tracción de las ruedas del vehículo.
Motor o Zapatillas	`int`	Indica la potencia del motor o las zapatillas.
Peso	`int`	Indica el peso del vehículo.

Ejemplo de una fila del archivo:

```
The Big Boss,Enzini,automóvil,64,48,16,32
```

6.2.3. `parametros.py`

La información de los parámetros[13] y números mencionados anteriormente, al igual que los *paths* de los archivos que utilice tu programa, se deben encontrar en este archivo, en donde cada línea de código almacena una constante con su valor respectivo. Es importante recalcar que debes **importar** este archivo **como un módulo** y así usar los valores almacenados.

Para distinguir si algo merece ser parametrizado o no, siempre es bueno pensar si es en sí mismo una "entidad"[14] o no, por ejemplo:

```
1 ONCE = 11  # mal parámetro
2 MIN_EQUILIBRIO_TAREO = 11  # buen parámetro
```

Es recomendable que los parámetros estén bien modelados. Esto consiste (entre otras cosas) en que, de haber un valor que realmente dependa de uno o más parámetros distintos (por ejemplo: el área de una esfera que dependa de otros parámetros), éste no dependa de valores arbitrarios.

En el caso presentado en el ejemplo anterior, lo correcto sería:

```
1 # En caso de que se cambie el valor de RADIO o PI,
2 # entonces también cambia el valor de AREA_ESFERA
3 RADIO = 7
4 PI = 3,1415
5 AREA_ESFERA = 4 * PI * RADIO ** 2
```

Si bien el correcto modelado de los parámetros no tendrá puntaje, bonificación o descuento asociado en esta tarea, se recomienda fuertemente el tratar de parametrizar lo mejor posible y siguiendo las reglas anteriores, ya que podría ser un ítem evaluado en tareas posteriores. Recibirás un *feedback* adecuado sobre esta parte en tu corrección, con fines formativos. Por último, cabe destacar que **todos** los parámetros escritos dentro de `parametros.py` deben ser utilizados de alguna forma dentro de tu programa.

Importante: Cualquier valor que debería ser un parametro que se encuentre *hardcodeado* dentro de tu programa, conllevará un descuento en tu nota.

[13] Los parámetros indicados en el enunciado son los elementos escritos en `ESTE_FORMATO`.
[14] No confundir con "objeto".

6.3. Manejo de las partidas

A cada línea del archivo `pilotos.csv` la llamaremos una **entrada**. Cada entrada del archivo `pilotos.csv` se corresponde a una partida guardada. El nombre del piloto será el nombre que el jugador ingrese. Los nombres de las distintas entradas deberán ser únicos.

6.3.1. Nueva partida

Al momento de querer crear una nueva partida, el jugador deberá indicar un nombre de usuario válido y luego elegir un equipo entre Tareos, Híbridos o Docencios. El nombre de usuario se considera válido si: no es utilizado por otros jugadores y sólo contiene caracteres alfanuméricos[15] y espacios. Luego, se creará una nueva entrada en el archivo `pilotos.csv` conteniendo la información inicial del piloto recién creado. Sus características base serán valores aleatorios en los rangos especificados en `Piloto` según el equipo seleccionado.

Después, el usuario deberá seleccionar un vehículo inicial, posteriormente se le deberá asignar un nombre[16] y finalmente, al igual que el piloto, se deberá almacenar como una entrada en el archivo `vehiculos.csv`, donde las características del vehículo serán valores aleatorios que dependerán de los valores indicados en `parametros.py` (como se indicó en la sección `Vehiculo`).

Estas modificaciones a los archivos deberán hacerse al momento de la creación del piloto y vehículo.

6.3.2. Guardar partida

Al guardar una partida, deberás actualizar la información del archivo `pilotos.csv` con cualquier cambio que se haya realizado al piloto durante la partida (como un incremento de experiencia, por ejemplo). También aplicar cualquier cambio en algún vehículo en el archivo `vehículos.csv`. Deberás respetar el uso de mayúsculas y minúsculas.

6.3.3. Cargar partida

Al elegir cargar una nueva partida, el usuario deberá ingresar su nombre válido. Con éste, tu programa comprobará si existe la entrada correspondiente en `pilotos.csv`. En caso de que no exista la entrada, se le deberá informar al usuario, pedir un nuevo nombre y dar la opción de volver al menú anterior.

7. Fórmulas

A continuación se presentan las distintas fórmulas que permiten el funcionamiento del juego. Considera que deberás usar la función **piso**[17] cuando aparezca.

[15] Para comprobarlo puedes usar el método `str.isalnum()`. Notese que los espacios son caracteres no alfanuméricos.

[16] Debe cumplir las mismas características que los nombres de usuario: ser único (entre los vehículos de usuario) y estar compuesto por caracteres alfanuméricos y espacios.

[17] $\lfloor x \rfloor$ es la función **piso** de x. Puedes usarla con `math.floor` o usando una división parte entera (`//`).

7.1. Cálculo de la velocidad

Estas son las fórmulas que influyen en el cálculo de la velocidad del vehículo durante una vuelta a la pista. La velocidad (`velocidad_real`) se calcula de la siguiente forma:

Velocidad Real: es la velocidad de un vehículo durante una vuelta. Considera que los valores de `dificultad_control` e `hipotermia` son **menores** o **iguales** a cero.

$$\texttt{velocidad\_real} = \max(\texttt{VELOCIDAD\_MINIMA},\ \texttt{velocidad\_intencional} + \texttt{dificultad\_control} + \texttt{hipotermia})$$

Donde:

Velocidad Intencional: es la velocidad a la que el piloto tiene la intención de ir, dada su personalidad.

Si la personalidad del piloto es `'osado'`:

$$\texttt{velocidad\_intencional} = \texttt{EFECTO\_OSADO} \times \texttt{velocidad\_recomendada}$$

Si la personalidad del piloto es `'precavido'`:

$$\texttt{velocidad\_intencional} = \texttt{EFECTO\_PRECAVIDO} \times \texttt{velocidad\_recomendada}$$

Velocidad Recomendada: dadas las condiciones del terreno, las características de tu vehículo y experiencia como conductor, existe una velocidad recomendada a la que deberías ir. Excederla te pondrá en peligro de sufrir un accidente. Si la pista es demasiado difícil y no eres lo suficientemente hábil, es mejor que bajes un poco la velocidad y te apegues a tus posibilidades. La `velocidad_base` es la potencia del motor o zapatillas de tu vehículo.

$$\begin{aligned}\texttt{velocidad\_recomendada} = \ & \texttt{velocidad\_base} + (\texttt{traccion\_ruedas} - \texttt{hielo\_pista}) \\ & + (\texttt{defensa\_carroceria} - \texttt{rocas\_pista}) \\ & + (\texttt{experiencia\_piloto} - \texttt{dificultad\_pista})\end{aligned}$$

Efecto de la Hipotermia: si te da hipotermia, pensarás más lento y reaccionarás más lento. Aunque no quieras, inconscientemente reducirás la velocidad. Ten en cuenta que mientras más tiempo pase, la hipotermia será cada vez peor.

$$\texttt{hipotermia} = \min(0,\ \texttt{numero\_vuelta} \times (\texttt{contextura\_piloto} - \texttt{hielo\_pista}))$$

Dificultad de control del vehículo: mientras más pesado sea el vehículo, más fácil será de controlar si tu equilibrio no es muy bueno. Esto te hará ir más lento.

Si el vehículo tiene 2 ruedas y el piloto es `'osado'`:

$$\texttt{dificultad\_control} = \text{mín}\left(0,\ \texttt{equilibrio} - \left\lfloor \frac{\texttt{PESO\_MEDIO}}{\texttt{peso\_vehículo}} \right\rfloor\right)$$

Si el vehículo tiene 2 ruedas y el piloto es `'precavido'`:

$$\texttt{dificultad\_control} = \text{mín}\left(0,\ \texttt{equilibrio} \times \texttt{EQUILIBRIO\_PRECAVIDO} - \left\lfloor \frac{\texttt{PESO\_MEDIO}}{\texttt{peso\_vehículo}} \right\rfloor\right)$$

Si el vehículo tiene 4 ruedas:

$$\texttt{dificultad\_control} = 0$$

7.2. Sucesos durante la carrera

Daño al vehículo: si estamos en una pista rocosa nuestro vehículo estará recibiendo daño si no tiene una defensa lo suficientemente alta.

$$\texttt{daño\_recibido\_cada\_vuelta} = \text{máx}\left(0,\ \texttt{defensa\_carroceria} - \texttt{rocas\_pista}\right)$$

Tiempo en los *Pits*: mientras más daño reciba tu vehículo, más tiempo deberás permanecer en los *Pits* para que éste se repare.

$$\begin{aligned}\texttt{tiempo\_pits} = \texttt{TIEMPO\_MINIMO\_PITS} + (&\texttt{durabilidad\_inicial\_chasis} \\ &- \texttt{durabilidad\_actual\_chasis}) \times \texttt{VELOCIDAD\_PITS}\end{aligned}$$

Dinero por vuelta: cada vuelta que sobrevivas es una nueva puerta que se abre ante ti. Conseguirás dinero por cada una.

$$\texttt{dinero\_vuelta\_x} = \texttt{numero\_vuelta} \times \texttt{dificultad\_pista}$$

Accidentes durante las vueltas: si vas más rápido de lo que deberías o si la durabilidad del chasis de tu vehículo llega a cero, puede que éste explote. Sí, porque sí.

$$\begin{aligned}\texttt{probabilidad\_accidente} = &\frac{\texttt{velocidad\_real} - \texttt{velocidad\_recomendada}}{\texttt{velocidad\_recomendada}} \\ &+ \left\lfloor \frac{\texttt{durabilidad\_maxima\_chasis} - \texttt{durabilidad\_actual\_chasis}}{\texttt{durabilidad\_maxima\_chasis}} \right\rfloor\end{aligned}$$

Tiempo por Vuelta: sirve para calcular el tiempo que se tardará en dar una vuelta.

$$\texttt{tiempo\_vuelta} = \left\lceil \frac{\texttt{largo\_pista}}{\texttt{velocidad\_real}} \right\rceil$$

7.3. Ganador de la carrera

Las carreras son un mundo peligroso y que solo recompensa a los mejores. Solo el ganador recibirá puntos de experiencia y un premio en dinero.

Dinero por ganar: es el dinero que se obtiene al ganar una competencia.

$$\texttt{dinero\_ganador} = \texttt{numero\_vueltas\_total} \times (\texttt{dificultad\_pista} + \texttt{hielo\_pista} + \texttt{rocas\_pista})$$

Ventaja: la ventaja con el último jugador es la diferencia entre los tiempos en los que llegan a la meta. esta se calcula como:

$$\texttt{ventaja\_con\_ultimo\_lugar} = \texttt{tiempo\_ultimo\_lugar} - \texttt{tiempo\_primer\_lugar}$$

Experiencia por ganar: es la experiencia que gana el jugador cuando gana una carrera, el valor dependerá de la personalidad de este.

Si el piloto es `'precavido'`:

$$\texttt{experiencia\_recibida} = (\texttt{ventaja\_con\_ultimo\_lugar} + \texttt{dificultad\_pista}) \times \texttt{BONIFICACION\_PRECAVIDO}$$

Si el piloto es `'osado'`:

$$\texttt{experiencia\_recibida} = (\texttt{ventaja\_con\_ultimo\_lugar} + \texttt{dificultad\_pista}) \times \texttt{DESBONIFICACION\_OSADO}$$

8. *Bonus*:

En esta tarea habrá una serie de *bonus* que podrás obtener. Cabe recalcar que necesitas cumplir los siguientes requerimientos para poder obtener *bonus*:

1. La nota en tu tarea (sin *bonus*) debe ser **igual o superior a 4.0**[18]
2. El *bonus* debe estar implementado **en su totalidad**, es decir, **no se dará puntaje intermedio**.

Finalmente, la cantidad máxima de décimas de *bonus* que se podrá obtener serán **8 décimas**.

[18] Esta nota es sin considerar posibles descuentos.

8.1. Buenas Prácticas (5 décimas)

Para obtener este *bonus* tu código deberá cumplir con una serie de requerimientos que son considerados como buenas prácticas al momento de programar. Más específicamente deberás cumplir los siguientes requerimientos:

1. **Clases Menú**:

   Deberás modelar y utilizar **todos los menús de tu programa como clases**, más específicamente, deberás crear una clase `Menu` de la cual todos los otros menús hereden. También, cada opción de cada menú debe estar modelado como un método que es llamado cuando esta opción es elegida. Por ejemplo:

```python
class Menu:
    def __init__(self, ...):
        pass

    def recibir_input(self, ...):
        pass

class MenuInicio(Menu):
    def __init__(self, ...):
        pass

    def recibir_input(self, ...):
        pass

    def cargar_partida(self, ...):
        pass

    def crear_partida(self, ...):
        pass
```

   Cabe recalcar que la clase `Menu` no tiene por que tener ningún método asociado a alguna elección, **pero si puede tener los métodos que consideren necesarios**. Finalmente, **todos los menús y sub menús deben ser clases distintas**.

2. **Archivo `funciones.py`**:

   Deberás agregar un módulo llamado `formulas.py` a tu programa. En este módulo deberás definir **todas las fórmulas matemáticas** utilizadas para los cálculos indicados en la sección Fórmulas. Por ejemplo:

```python
def calcular_hipotermia(...):
    # ...
    if contextura_piloto < cantidad_hielo_pista:
        return (contextura_piloto - cantidad_hielo_pista) * turno_actual
    return 0
```

   Como dentro de este archivo tienen que estar todas las fórmulas, también se requiere que se utilice este módulo **cada vez que se quiera hacer uso de una fórmula**.

8.2. Power-Ups (3 décimas)

Para obtener este *bonus* deberás implementar una variedad de *power-ups*, que sólo el jugador obtendrá al final de cada vuelta y de manera aleatoria. Éste podrá elegir entre utilizarlo durante la siguiente vuelta, o bien guardarlo para después. Solo se puede llevar un *power-up* a la vez, por lo que si el jugador completa una vuelta guardando uno, no podrá conseguir otro. Los *power-ups* a implementar son los siguientes:

Caparazón Este confiable ítem dañará a un contrincante al azar, quitándole `DMG_CAPARAZON` de durabilidad a su `chasis`.

Estrella Llegada del espacio, la estrella te otorgará invencibilidad durante una vuelta, evitando que tu vehículo reciba daño alguno.

Relámpago Directamente desde las nubes, una lluvia de relámpagos ataca a todos tus oponentes, reduciendo su velocidad base en `SPD_RELAMPAGO` durante una vuelta.

9. Diagrama de clases y avance de tarea

En conjunto con el programa, se tendrá que realizar un diagrama de clases modelando las entidades necesarias para realizar el juego. Este diagrama se entregará en dos ocasiones:

Una versión preliminar, que corresponderá al avance de esta tarea. A partir de los diagramas entregados, se les brindará un *feedback* general de cómo va la modelación de sus programas y además, les permitirá optar por **hasta 2 décimas** adicionales en la nota final de su tarea.

Luego de esto, junto a la entrega final, deberán entregar una versión final de su diagrama que **represente fielmente** la modelación del problema de su programa.

En ambos casos, el diagrama deberá:

- Entregarse en **formato PDF o de imagen**.
- Contener todas las clases junto con sus atributos y métodos.
- Contener todas las relaciones existentes entre las clases (agregación, composición y herencia).
- No es necesario indicar la cardinalidad ni la visibilidad (público o privado) de los métodos o atributos.

Para realizar el diagrama de clases te recomendamos utilizar **draw.io**, **lucidchart** o aplicaciones similares.

Sería conveniente que adjunten a su diagrama un documento con una explicación general de su modelación. Esto con el fin de ayudar la corrección del ayudante a reconocer su razonamiento.

Tanto el diagrama (en formato PDF o de imagen) como la explicación de su modelación (en formato **Markdown**) deben ubicarse en la misma carpeta de entrega de la tarea.

10. `.gitignore`

A partir de esta tarea **se evaluará el correcto uso del archivo `.gitignore`**, para más información con respecto a los descuentos asociados a errores en uso pueden revisar la **guía de descuentos**.

Deberás ignorar los archivos indicados en Archivos estáticos con un `.gitignore` que deberá estar dentro de la carpeta `T01/`. Puedes encontrar un ejemplo de `.gitignore` en el siguiente ***link***.

11. Descuentos

A partir de esta tarea los siguientes descuentos aplicarán los siguientes cambios, para que los tengas en consideración al momento de escribir tu tarea.

Modularización (5 décimas): se descontarán 5 décimas si uno o más archivos exceden las **400** líneas de código. En este calculo se consideran los saltos de línea y los comentarios.

Malas prácticas (5 décimas): si se encuentran malas prácticas en el código de la tarea, como el uso de **variables globales**, será penalizado. Cualquier duda con este descuento, haga una *issue*.

Uso de `.gitignore` (hasta 5 décimas): se descuentan hasta 5 décimas por no usar correctamente el archivo `.gitignore`.

12. Importante: Corrección de la tarea

Para esta tarea, el carácter funcional del juego será el pilar de la corrección, es decir, **sólo se corrigen tareas que se puedan ejecutar**. Por lo tanto, se recomienda hacer periódicamente pruebas de ejecución de su tarea y `push` en sus repositorios.

Cuando se publique la distribución de puntajes, se señalará con color amarillo cada ítem que será evaluado a nivel de código, todo aquel que no esté pintado de amarillo significa que será evaluado si y sólo si se puede probar con la ejecución de su tarea. En tu `README` deberás señalar el archivo y la línea donde se encuentran definidas las funciones o clases relacionados a esos ítems.

13. 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 3.7.
- Tu programa debe estar compuesto por uno o más archivos de extensión `.py`.
- 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. **Tendrás hasta 48 horas después del plazo de entrega** de la tarea para subir el `README` a tu repositorio.
- Tu tarea podría sufrir los descuentos descritos en la **guía de descuentos**.
- Entregas con atraso de más de 24 horas tendrán calificación mínima (1,0).
- Cualquier aspecto no especificado queda a tu criterio, siempre que no pase por sobre otro.

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