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

IIC2233 Programación Avanzada (2019-1)

Tarea 01

Entrega

- **Avance de tarea**
  - **Fecha y hora:** miércoles 27 de marzo de 2019, 20:00
  - **Lugar:** GitHub — Carpeta: `Tareas/T01/`
- **Tarea y diagrama de clases final**
  - **Fecha y hora:** domingo 7 de abril de 2019, 20:00
  - **Lugar:** GitHub — Carpeta: `Tareas/T01/`
- `README.md`
  - **Fecha y hora:** lunes 8 de abril de 2019, 20:00
  - **Lugar:** GitHub — Carpeta: `Tareas/T01/`

Objetivos

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

Índice

1. Introducción a DCCivilization

Después de meses de ocultarse en las sombras, debilitado por haber tomado muchas ayudantías el semestre anterior, el malvado **Doctor Valdivieso** busca dominar el mundo una vez más. Sin embargo, no posee el poder suficiente para atacar al mundo directamente, por lo que se vio forzado a crear una pequeña civilización y enviar a su ejército de ayudantes a destruirte.

Frente a esta situación, el **Gran Polea** te pide a ti que lo ayudes a crear una estrategia perfecta para derrotar al malvado doctor. Para esto, decides crear un juego simple que simule una civilización cuyo objetivo es crecer, generar recursos y construir un ejército con el fin de destruir a las civilizaciones enemigas.

Para crear tu juego, decides usar lo que aprendiste en Programación Avanzada, en específico Programación Orientada a Objetos, para modelar los distintos ámbitos de tu civilización y asegurar tu victoria contra el mal.

2. DCCivilization

En este programa, un usuario puede jugar DCCivilazation interactuando con la consola. El jugador principal se enfrentará a otras civilizaciones manipuladas por la máquina.

DCCivilization es un juego **por turnos** en el que controlas una civilización. Cada civilización tiene recursos, personas y edificios, lo que les permitirá atacar a otras civilizaciones, defenderse de ataques enemigos e investigar sobre nuevas tecnologías para ganar el juego.

En cada turno puedes realizar distintas acciones, como recolectar recursos, crear personas, construir edificios, atacar otras civilizaciones o simplemente terminar el turno y pasar al siguiente. Luego de finalizar tu turno, se ejecutarán las acciones de las otras civilizaciones en el juego, que se entregarán en un archivo.

Es importante destacar que las acciones del jugador **siempre se ejecutan primero**, y después se ejecutan las acciones del resto de las civilizaciones de acuerdo al orden que se indicará en el archivo.

3. Civilizaciones

Cuando comience un juego nuevo, el jugador tendrá la posibilidad de elegir su civilización para jugar entre tres posibilidades: **DCC**, **La Comarca** y **Cobreloa**. Cada civilización tiene un líder distinto, pero todas comparten el tipo de personas y edificios que pueden crear. Cada una tiene un número máximo de personas que puede contener, definido por una constante entregada en un archivo (Subsección 6.1).

3.1. Recursos

Existen tres tipos de recursos en este vasto universo: **Oro**, **Madera** y **Piedra**. Estos recursos pueden ser extraídos por algunas personas y se acumulan para ser utilizados en la creación de más personas y construcción de edificios. Cada tipo de recurso tiene un uso específico.

3.2. Personas

Son las unidades básicas de toda civilización. Toda persona posee *HP*[1], definida por un número aleatorio uniforme entre `MIN_HP_PERSONA` y `MAX_HP_PERSONA` (Subsubsección 6.1.5)[2]. Pueden ser de tres especializaciones diferentes:

- **Trabajador**. El trabajador es la unidad que se encarga de la recolección de recursos y de la construcción de edificios. Cada uno de ellos puede hacer cualquiera de estas dos tareas pero **no pueden trabajar en más de una a la vez**. En otras palabras, solo volverán a estar disponibles para realizar otra actividad una vez que terminen la última que les fue asignada.

  Cuando un trabajador se asigna a recolectar un recurso, la cantidad obtenida de este recurso es un valor aleatorio uniforme entre `MIN_CANTIDAD_RECURSO` y `MAX_CANTIDAD_RECURSO` (Subsubsección 6.1.5). Dicha cantidad extraída será añadida a los recursos de la civilización al **inicio del siguiente turno**, y el trabajador quedará libre para ser asignado a otra actividad.

  Las civilizaciones comienzan con 3 trabajadores y ninguna otra persona.

- **Soldado**. Los soldados son entrenados específicamente en el cuartel. Nadie sabe de dónde proviene su sorprendente fuerza, pero lo que sí es seguro es que fueron entrenados para morir en sangre y honor por su civilización. Gracias a sus habilidades, los soldados son las unidades que tienen la capacidad de atacar a otras civilizaciones.

  El poder de fuerza de cada soldado es asignado al momento en que son creados, el que es un valor aleatorio uniforme entre `MIN_ATAQUE_SOLDADO` y `MAX_ATAQUE_SOLDADO` (Subsubsección 6.1.5).

- **Ayudante**. Llega un punto en el que toda civilización ve la luz y no puede resistir su sed de conocimiento, lo que eventualmente da paso a la construcción del DCCowork (un tipo de edificio). Es allí donde nacen estos seres iluminados: los ayudantes. Estos individuos poseen una característica intrínseca denominada IQ la cual les permite generar nuevos avances tecnológicos para la civilización. Son las únicas personas con esta capacidad.

  El IQ de cada ayudante se asigna al momento de su creación, y es un valor aleatorio uniforme entre `MIN_IQ_AYUDANTE` y `MAX_IQ_AYUDANTE` (Subsubsección 6.1.5).

3.3. Edificios

Al igual que todo gran imperio, es necesario tener edificios para que entrenen a sus habitantes. Los edificios no se construyen por arte de magia, sino que necesitan de **recursos**, **trabajadores** y **cantidad de turnos** para completarse. Todo edificio posee *HP*, la que puede ser reducida al sufrir ataques. Algunos edificios también pueden crear personas de un tipo en específico a cambio de recursos.

En el momento de construir un edificio, el juego debe asignar automáticamente los trabajadores necesarios, por la cantidad de turnos en que éste demora en construirse.

La cantidad de recursos y trabajadores necesarios para construir un edificio, el *HP* inicial de un edificio, y la cantidad recursos necesarios para utilizar un edificio en el juego, dependen del tipo de edificio. Todas esas cantidades serán entregados en un archivo (Subsubsección 6.1.2). En el caso de que no hayan recursos o trabajadores suficientes para construir el edificio, se le deberá informar al usuario de esto y no permitir la construcción.

A continuación se explicará cómo funciona cada edificio.

[1] Del ingles *Health Points*, representa la vitalidad de cada persona o edificio.
[2] *Hint:* La función `randint` de la librería `random` te podría ser útil.

- **Centro Urbano**

  Este es el primer edificio con el que empiezas tu civilización. En este edificio se crean trabajadores. Solo puede existir un Centro Urbano por civilización, y su destrucción significa la eliminación de la civilización del juego.

- **Cuartel**

  Este edificio permite crear soldados para poder atacar y defenderte de otras civilizaciones. Las civilizaciones no parten con un Cuartel y solo puedes tener uno construido a la vez.

- **DCCowork**

  Los ayudantes, dándose cuenta que no contaban con un espacio propio para trabajar como se debe, se adueñaron de un edificio y hablaron con su respectivo líder para que se convierta en su santuario de investigación, el DCCowork.

  Es aquí donde se crean a los ayudantes, que a su vez generan avances tecnológicos. Las civilizaciones no parten con un DCCowork y solo puedes tener uno construido a la vez.

- **Murallas**

  Para proteger a tu civilización de compañía indeseada, puedes ordenar a tus trabajadores a construir estos edificios alrededor de tu ciudad. Este es un edificio que no es capaz de crear unidades (personas), y puedes construir tantos como quieras. Este tipo de edificio provee defensa ante ataques enemigos. Esto se explica en detalle en la Subsección 4.1.

3.4. Líderes

Cada civilización cuenta con un líder, el cual le otorga ciertos beneficios que las otras civilizaciones no poseen.

1. El DCC tiene de líder al **Doctor Valdivieso**, quien posee el poder de sobreexplotar a sus ayudantes, lo cual se traduce en una bonificación de 50 % de los puntos de progreso científico generados por estas unidades (Subsección 4.2). Además, se generan 2 ayudantes inmediatamente tras completar la construcción del DCCowork.
2. La Comarca tiene como líder al **Gran Polea** quien posee el poder de **reparación de murallas**, lo que causa que en cada turno su muralla recupere un cuarto de su *HP* máxima. Además, la cantidad de piedra y madera extraída por sus trabajadores aumenta en un 10 % y un 25 % respectivamente.
3. Cobreloa, cuyo líder es **Cruz**, posee una bonificación a la cantidad de recursos mineros (oro y piedra) recolectado por turno de un 50 %. Además, el poder del fútbol le otorga bonificaciones a sus puntajes de ataque y defensa totales. (ver Subsección 4.1 para más detalles).

4. Funcionalidades

Para poder modelar de manera correcta a las distintas civilizaciones, así como sus acciones y evolución a lo largo del tiempo, es necesario que tu programa implemente las siguientes funcionalidades:

4.1. Ataque y Defensa

Una civilización es capaz de atacar a otra en un turno cualquiera. Al momento de atacar a una civilización rival, el **poder total de ataque** P_A con el cual lo hace depende de: la fuerza total de sus soldados F_S (definido como la suma de la fuerza de todos sus soldados); y de la civilización específica que decidió atacar. Se utilizan las siguientes fórmulas para calcular P_A:

- **DCC**: La gran cantidad de ayudantes esclavizados permite reforzar el ataque mientras más tecnología posea la civilización.

  $$P_A = F_S + C_{\text{tec}}$$

  Donde C_{tec} es la cantidad de puntos de tecnología de la civilización.

- **La Comarca**: Su conocimiento avanzado sobre el procesamiento de piedras les permite realizar ataques más "duros".

  $$P_A = F_S + \frac{C_{\text{piedra}}}{2}$$

  Donde C_{piedra} es la cantidad de piedra que posee la civilización.

- **Cobreloa**: Su espíritu futbolístico aumenta su fuerza, independiente de qué tan bien posicionados estén.

  $$P_A = F_S \times 1{,}15$$

Por su lado, la civilización que recibe el ataque tiene un **poder total de defensa** P_D para defenderse. Las civilizaciones tienen un **poder base de defensa** B_D común que se calcula bajo la siguiente fórmula:

$$B_D = HP_S + HP_T + HP_E$$

Donde HP_S es la suma total de HP de todos los **soldados defensores** en la civilización; HP_T es la suma total de HP de todos los **trabajadores** en la civilización; y HP_E es la suma total de HP de los edificios en la civilización, sin contar el DCCowork, que no participa en la defensa de la civilización.

Para calcular el poder total de defensa P_D, se utilizan las siguientes fórmulas:

- **DCC**: Los bajos presupuestos del DCC han obligado a los ayudantes a trabajar de soldados **'Part-Time'**, aportando a la defensa total.

  $$P_D = B_D + 100 \times C_{\text{ayud}}$$

  Donde C_{ayud} es la cantidad de ayudantes de la civilización.

- **La Comarca**: La gran cantidad de árboles talados que deja esta civilización actúa como barricada ante los ataques.

  $$P_D = B_D + \frac{C_{\text{madera}}}{2}$$

  Donde C_{madera} es la cantidad de madera que posee la civilización.

- **Cobreloa**: Su incesante esfuerzo por defender posiciones los lleva a aumentar su resistencia ante los ataques.

  $$P_D = B_D \times 1,1$$

Cabe destacar que el valor numérico final de tanto P_A como P_D debe ser entero, por lo que se deberá aproximar al entero más cercano en caso de que se obtenga un número decimal.

Si el poder de ataque total del atacante es mayor o igual al poder total de defensa del oponente ($P_A \geq P_D$), entonces la civilización atacada pierde su centro urbano y es eliminada (ya no participa en el juego). Tras esto, una cantidad aleatoria de cada recurso y de los ayudantes de esta última pasan a ser de la civilización vencedora. Esta cantidad sigue una distribución **uniforme** entre 0 y la cantidad total del recurso o número de ayudantes con que cuente la civilización perdedora. Los ayudantes obtenidos serán elegidos al azar entre los que tenía la civilización destruida.

En caso de que el poder total de ataque sea menor que el poder total de defensa ($P_A < P_D$), se deberán repartir los daños del ataque sobre la civilización atacada en el siguiente orden:

- Primero se daña a la(s) muralla(s), restando de su *HP* el poder de ataque del rival. Si el *HP* de una muralla llega a 0, entonces ésta es destruida. El orden de aplicación de daño sobre las murallas debe ser comenzando por las murallas con menor *HP*.
- Si todas las murallas son destruidas, entonces se comienza a dañar a cada soldado, restando de su *HP* el daño que **sobró** del ataque tras la caída de las murallas. Si un *HP* llega a 0, entonces el soldado muere (RIP). Esto se debe hacer comenzando por los soldados que poseen menor *HP* hasta los que se encuentran en mejor estado.
- Si todos los soldados caen, entonces se procederá a dañar al cuartel.
- Posterior a la caída de éste, la última línea defensiva corresponde a los trabajadores, donde se debe utilizar la misma mecánica de los soldados, comenzando por los que están más debilitados.
- Finalmente, se daña con el poder de ataque restante al centro urbano, y en caso de que caiga, la civilización atacada es eliminada y se procede a pasar ayudantes y recursos al ganador como se indicó anteriormente.

A modo de ejemplo, si una civilización tiene un poder de ataque de 20, primero se aplicará daño a la muralla de menor *HP*. Supongamos tal muralla tiene *HP* de 2, entonces, ésta es destruída ya que hay suficiente daño para reducir su *HP* a 0. Luego de esto, restan 18 puntos de poder de ataque restantes para repartir como daño. Se aplicaría primero sobre cualquier muralla restante. De no quedar murallas, entonces se comienza a dañar a los soldados, y así sucesivamente, siguiendo el orden explicado anteriormente.

Es importante recalcar que cuando una civilización ataca, ésta **no recibe ningún daño** (esto solo ocurre cuando una civilización se defiende). Sin embargo, si una civilización ataca a otra en un turno, entonces sus soldados no estarán disponibles para defender su ciudad hasta el próximo turno. Es decir, en el caso de que un oponente decida atacar a su civilización, los soldados no son contados en la fórmula de defensa total ni en la repartición de daños.

4.2. Investigación de Tecnologías

Durante el desarrollo del juego, las civilizaciones se dedicarán a investigar nuevas tecnologías, para lo cual necesitan acumular puntos tecnológicos. Los encargados de obtener dichos puntos son los ayudantes, los que entregan un total de puntos tecnológicos P_T al **inicio** de cada turno en base a la siguiente fórmula:

$$P_T = \frac{IQ_{\text{ayud}}}{100}$$

Donde IQ_{ayud} corresponde a la suma de IQ todos los ayudantes de la civilización. Así, turno por turno se suman los valores de puntos generados P_T en la cantidad total de puntos tecnológicos de la civilización C_{tec}.

Cabe recordar que las civilizaciones empiezan sin el DCCowork, por lo que no generarán puntos tecnológicos hasta que se construya y se hayan creado ayudantes.

A medida que se obtienen puntos tecnológicos, se irán desbloqueando tecnologías importantes tras alcanzar distintas metas de puntaje sobre C_{tec}, las que se muestran a continuación junto a su efecto sobre la civilización:

- **Matemáticas (30 ptos. tecnológicos)**: Aplicando los avances en las matemáticas, los ingenieros inventaron el trebuchet, la mejor arma de asedio jamás creada. Desde ahora en adelante la fuerza de los soldados aumenta en un 50 %.
- **Educación (70 ptos. tecnológicos)**: Los ayudantes son más eficientes, ya que estudian para aprender y no para obtener una buena nota, aumentando su IQ en un 10 %.
- **Medicina (140 ptos. tecnológicos)**: Los avances en la medicina aumentan en un 50 % el *HP* máximo de todos los trabajadores, soldados y ayudantes, tanto actuales como futuros. A los que ya estaban heridos se les aplicará la bonificación considerando su *HP* actual.
- **Computación (250 ptos. tecnológicos)**: ¡Se descubre el maravilloso mundo de la computación, obteniendo la **¡Victoria Científica**!.

4.3. Estado del Sistema (Estadísticas)

El programa debe poder mostrar la información de cada civilización para que el jugador pueda conocer los estados actuales de sus ejércitos, recursos, entre otras características. Estas civilizaciones deben poder ordenarse de acuerdo a su P_A o a su P_D.

Para esto es que se debe mostrar la siguiente información por cada civilización:

- Cantidad de recursos almacenados por tipo.
- Cantidad de personas por especialización.
- Cantidad de edificaciones por tipo.
- Metas tecnológicas alcanzadas y puntos tecnológicos totales C_{tec}.
- Poder total de ataque P_A.
- Poder total de defensa P_D.
- Puntaje total P_{total}.

  El puntaje total de una civilización se calcula con la siguiente fórmula:

  $$P_{\text{total}} = P_A + P_D + C_S + C_{\text{tec}}$$

  Donde P_A corresponde al poder de ataque total, P_D corresponde a la defensa total, C_S es la cantidad de soldados y C_{tec} es la cantidad de puntos tecnológicos de la civilización.

4.4. Término del Juego

Para ganar el juego existen 3 condiciones de victoria distintas:

- **Victoria por Dominación**: Ocurre cuando una civilización destruye a todas las otras.
- **Victoria Científica**: Ocurre cuando una civilización investiga todas las tecnologías.
- **Victoria por Tiempo**: Ocurre cuando se alcanzan los `MAX_TURNOS` turnos (Subsubsección 6.1.5). Gana el jugador que tiene el mayor puntaje total P_{total}.

Si en algún momento el jugador es destruido por otra civilización, se le deberá notificar al jugador de que perdió y el juego terminará. De otra forma, el juego continuará hasta que una civilización cumpla con alguna de las condiciones de victoria, mostrando quién ganó y qué tipo de victoria logró, para luego proceder a finalizar la partida.

5. Interacción por consola

Para que se pueda utilizar tu juego, deberás crear un menú para interactuar por consola con las distintas funcionalidades del programa. En cada turno se podrán realizar diversas acciones, las que deberán ser ejecutadas en el mismo orden que el jugador las eligió. Las acciones que puede realizar el jugador están restringidas a tus recursos, es decir, si no tienes suficientes recursos o trabajadores para realizar algo, no es posible realizar dicha acción. Los efectos de dichas acciones no se verán reflejados hasta el siguiente turno aunque, los recursos utilizados para completar tal acción son reducidos inmediatamente después de verificar que son los suficientes para abarcar el costo.

A continuación se detallan las opciones que el menú le provee al usuario para jugar:

- Al iniciar el programa se debe poder elegir entre crear una partida nueva o cargar una existente. En el primer caso, se tendrá la posibilidad de elegir una civilización de entre un grupo de opciones (Sección 3). En el caso de que se haya escogido cargar una partida existente, se deberán mostrar las partidas guardadas y se debe poder instanciar toda la información de la partida seleccionada para poder continuar con ella. Luego de seleccionar la partida con la que se jugará, comienza el juego. En cada turno se le debe permitir al jugador realizar varias opciones.
- Una de las acciones que pueden realizar en un turno, es la recolección de recursos. Se le debe permitir al jugador escoger que tipo de recurso quiere recolectar, y cuantos trabajadores asignará para recolectarlo. Al momento de asignar el número de trabajadores, serán considerados inmediatamente como trabajadores ocupados, y no podrán realizar otras acciones en ese turno. Posteriormente los recursos recolectados aparecerán al inicio del turno siguiente.
- Otra de las acciones que se pueden realizar en un turno, es la creación de personas. Se le debe indicar al jugador cuales son las especializaciones de personas que se pueden crear considerando los edificios existentes, la cantidad de recursos actuales y la cantidad máxima de personas de la civilización. Al momento de elegir alguna especialización de persona, será considerada **inmediatamente** dentro del límite de población. Posteriormente la persona aparecerán al inicio del turno siguiente.
- Similar al punto anterior, una acción que se puede realizar en un turno es la creación de edificios. Se debe indicar al usuario cuales son los edificios que se pueden construir, considerando los recursos del jugador, los edificios existentes y el límite de edificios por tipo. Al momento de elegir crear un edificios, será considerado **inmediatamente** dentro del límite de edificios, y luego se deberá esperar la cantidad de turnos necesarios para crearla. Posteriormente la entidad aparecerá al inicio del turno siguiente. Por ejemplo, si se decide crear un edificio que toma 2 turnos, entonces este estará en construcción durante el turno actual y el siguiente turno, y al inicio del subsiguiente aparecerá en la ciudad.
- El otro tipo de acción que se puede realizar en un turno es atacar a otra civilización (Subsección 4.1). Para seleccionar a quién atacar, se debe mostrar una lista de las civilizaciones rivales. Luego, se debe poder seleccionar una para atacar. El turno siguiente, se deberán mostrar los resultados del ataque (si se logró derrotar al oponente o solo dañarlo). Solo se puede atacar una vez por turno.
- Al comienzo del turno, se deberá notificar al jugador si es que fue atacado por otra civilización el turno anterior. Debes indicar los daños sufridos, y en caso de ser derrotado, se debe indicar que está

fuera de la partida y terminar el juego.

- Si es que ya no se quieren realizar más acciones se debe poder seleccionar la opción de esperar al siguiente turno. Esto finalizará el turno del usuario y el programa deberá ejecutar los **turnos y acciones de las demás civilizaciones** según el archivo de acciones (Subsección 6.1). Al finalizar esas acciones, volverá a ser su turno.
- Una acción que siempre se ejecuta al comienzo del turno es la investigación de nuevas tecnologías (Subsección 4.2). Esto debe mostrarse mediante un mensaje al usuario notificando los puntos tecnológicos P_T desarrollados en el turno, la cantidad puntos totales C_{tec}, y en caso de alcanzar una nueva meta de nuevas tecnología, indicar la tecnología recién descubierta y sus efectos sobre la civilización.
- Para poder ver el estado de las civilizaciones rivales, el menú deberá tener una opción de mostrar las estadísticas de todas las civilizaciones (Subsección 4.3).
- Como parte de los requisitos se debe poder guardar el estado de la partida. Por lo tanto, el menú debe poseer una opción de "guardar" o "guardar y salir". También debe estar la opción de "salir sin guardar", lo cual cierra el juego, pero se pierde todo el progreso logrado desde la última vez que se guardó (Subsección 6.2).
- Es posible que durante la selección de las distintas opciones se quiera volver hacia atrás, por lo que se debe poder tener la opción de volver. Esto se debe aplicar a **todos los sub-menú del programa**, para así facilitar la interacción por consola y mejorar la usabilidad del juego. Además, el menú **debe ser a prueba de inputs erróneos o no válidos**. Así, si es que se ingresa una opción o parámetro no válido, el programa deberá pedir uno nuevo.

Este es un menú guía, no es necesario que tenga exactamente las mismas opciones, en el mismo orden. Lo importante es que se puedan acceder a todas las funcionalidades de manera fácil e intuitiva.

En el caso que decidas no realizar un menú que permite la interacción con el programa, debes entregar un archivo con el nombre `main.py` que muestre el funcionamiento de tu tarea. Entiéndase, un archivo que instancie las clases modeladas y ejecuten métodos que muestren las funcionalidades implementadas de las civilizaciones. Recuerda: **si no se pueden probar las funcionalidades, éstas no tendrán puntaje**. Además, ten en cuenta que si no construyes un menú, no obtendrás puntaje en interacción con la consola.

Hint: Para crear un menú a prueba de errores, se recomienda crear uno que solo requiera *inputs* numéricos y por lo tanto, cualquier otro input que no sea número es rechazado. De forma visual, el menú debería ser del estilo:

```
Seleccione una opción:
[1] Crear partida
[2] Cargar partida

Indique su opción (1 o 2): (input de usuario)
```

6. Archivos

6.1. Archivos entregados

Para que logres crear DCCivilization, el **Gran Polea** te hará entrega de cuatro archivos en formato CSV separados por `','`: `civilizaciones.csv`, `edificios.csv`, `acciones.csv` y `acciones_pendientes.csv`, los que contienen la información de las civilizaciones, los diferentes tipos de edificios, las acciones a tomar

durante el juego por las civilizaciones enemigas y sus acciones pendientes respectivamente, además de un archivo Python, `constantes.py`, que contendrá todas las constantes mencionadas en el enunciado. Todos estos archivos estarán almacenados en la carpeta `data/`. Con ellos deberás ser capaz de crear, guardar y cargar una partida para que el usuario pueda jugar o seguir jugando más adelante.

6.1.1. `civilizaciones.csv`

Al comenzar un juego nuevo las civilizaciones comienzan con una cantidad inicial de recursos y personas. La información inicial de las civilizaciones se encuentra en el archivo `civilizaciones.csv`, en donde cada fila representa una civilización con su información. Cada columna del archivo representa lo indicado en la siguiente tabla:

Nombre	Tipo de dato	Comentarios
`id`	`integer`	Id de la civilización.
`tipo`	`string`	Tipo de civilización.
`centro_urbano`	`boolean`	Presencia o ausencia de un centro urbano en la civilización.
`cuartel`	`boolean`	Presencia o ausencia de un cuartel en la civilización.
`dccowork`	`boolean`	Presencia o ausencia de un DCCowork en la civilización.
`muralla`	`integer`	Cantidad de murallas de la civilización.
`trabajador`	`integer`	Cantidad de trabajadores de la civilización.
`soldado`	`integer`	Cantidad de soldados de la civilización.
`ayudante`	`integer`	Cantidad de ayudantes de la civilización.
`oro`	`integer`	Cantidad de oro de la civilización.
`madera`	`integer`	Cantidad de madera de la civilización.
`piedra`	`integer`	Cantidad de piedra de la civilización.
`puntos_de_tecnologia`	`integer`	Los puntos de tecnología de la civilización.
`usuario`	`boolean`	Indica si la civilización está asociada al usuario.

Cuadro 1: Estructura de `civilizaciones.csv`

El tipo de civilización puede ser `'DCC'`, `'Comarca'` o `'Cobreloa'`.

La ausencia de un edificio significa que actualmente la construcción no se encuentra en la civilización y que por lo tanto puede construirse posteriormente. En el caso especifico del centro urbano, si éste no está presente en la civilización significa que actualmente esta fue derrotada y que por ende ya no puede realizar acciones. Notar que este archivo siempre comenzará con el Centro Urbano y sin Cuartel o DCCowork, para un juego nuevo, pero se explica la representación general ya que este archivo será reutilizado en la Subsección 6.2.

Finalmente, la variable usuario hace referencia a si la civilización se encuentra asociada al usuario de la partida (Subsección 6.2), por lo que en cada archivo a lo más un `usuario` puede ser verdadero para una civilización.

Ejemplo de fila:

```
0,DCC,True,False,False,3,24,19,5,217,178,56,11,True
```

[2]Se explica con mayor claridad en la Subsección 6.2: 'Persistencia de datos'.

6.1.2. `edificios.csv`

La información de las construcciones existentes se encuentra en el archivo `edificios.csv`. Cada fila representa un tipo de edificio con su respectiva información. Cada columna del archivo representa lo indicado en la siguiente tabla:

Nombre	Tipo de dato	Comentarios
`nombre`	`string`	Nombre del edificio.
`tiempo`	`integer`	Turnos que demora en construirse el edificio.
`c_trabajadores`	`integer`	Trabajadores necesarios para construir el edificio.
`c_oro`	`integer`	Oro necesario para construir el edificio.
`c_madera`	`integer`	Madera necesaria para construir el edificio.
`c_piedra`	`integer`	Piedra necesaria para construir el edificio.
`hp`	`integer`	El *HP* que proporciona el edificio.
`p_oro`	`integer`	Oro necesario para producir una persona.
`p_madera`	`integer`	Madera necesaria para producir una persona.
`p_piedra`	`integer`	Piedra necesaria para producir una persona.

Cuadro 2: Estructura `edificios.csv`

El nombre del edificio puede ser `'centro urbano'`, `'cuartel'`, `'dccowork'` o `'muralla'`.

Ejemplo:

```
dccowork,2,7,30,25,55,225,15,5,10
```

Como podrás haber notado en Subsección 3.3: 'Edificios', solo los Centros urbanos, Cuarteles y DCCowork's producen personas, por lo que el valor que se entregue en los costos de producción de personas en las murallas es despreciable.

6.1.3. `acciones.csv`

La información de las acciones que realiza cada civilización (en caso de no ser controlada por el jugador) durante el juego se encuentra en el archivo `acciones.csv`, cada fila representa una acción con su descripción. Cada columna del archivo representa lo indicado en la siguiente tabla:

Nombre	Tipo de dato	Comentarios
`turno`	`integer`	Turno en que se realiza la acción.
`id_civilizacion`	`integer`	Id de la civilización.
`tipo_accion`	`string`	Tipo de acción a realizar.
`input_accion`	`par de strings`	Inputs de la acción separados por `';'`.

Cuadro 3: Estructura `acciones.csv`

El tipo de acción a realizar puede ser `'Recolectar recurso'`, `'Construir edificio'`, `'Crear persona'`, `'Atacar civilización'` o `'Nada'`.

El orden en que se entrega el *input* de las acciones es: `tipo_elemento;cantidad`.

Ejemplo

```
5,0,Construir edificio,muralla;6
5,1,Recolectar recurso,madera;5
```

6.1.4. `acciones_pendientes.csv`

La información de las acciones que quedaron pendientes durante la partida (por ejemplo: la construcción de un edificio) durante el juego deben ser almacenados en el archivo `acciones_pendientes.csv`, cada fila representa una acción pendiente con su descripción. Cada columna del archivo representa lo indicado en la siguiente tabla:

Nombre	Tipo de dato	Comentarios
`turno`	`integer`	Turno en que se realizó la acción.
`id_civilizacion`	`integer`	Id de la civilización.
`tipo`	`string`	Tipo persona o edificio a producir.
`turnos_pendientes`	`integer`	Turnos pendientes para terminar de producir.

Cuadro 4: Estructura `acciones.csv`

El tipo de persona o edificio a producir puede ser `'trabajador'`, `'soldado'`, `'ayudante'`, `'cuartel'`, `'dccowork'` o `'muralla'`.

Ejemplo:

```
27,2,ayudante,3
```

Al crear una partida nueva claramente no pueden existir acciones pendientes, por lo que el archivo que se encuentra en `data/` no contiene información. En la Subsección 6.2: 'Persistencia de datos' se explicará su uso.

6.1.5. `constantes.py`

La información de las constantes mencionadas anteriormente se encuentra en el archivo `constantes.py`, en donde cada fila almacena una constante con su respectivo valor. Debes importar este archivo como un módulo y así usar los valores almacenados.

Ejemplo:

```
MAX_POBLACION_DCC = 200
MIN_IQ_AYUDANTE = 75
MAX_IQ_AYUDANTE = 165
```

6.2. Persistencia de datos

Para que el **Gran Polea** pueda jugar DCCivilization en cualquier momento y lugar, te pide que tu programa sea capaz de guardar y cargar partidas. Para esto tendrás que hacer uso de las carpetas `data/` y `save/`.

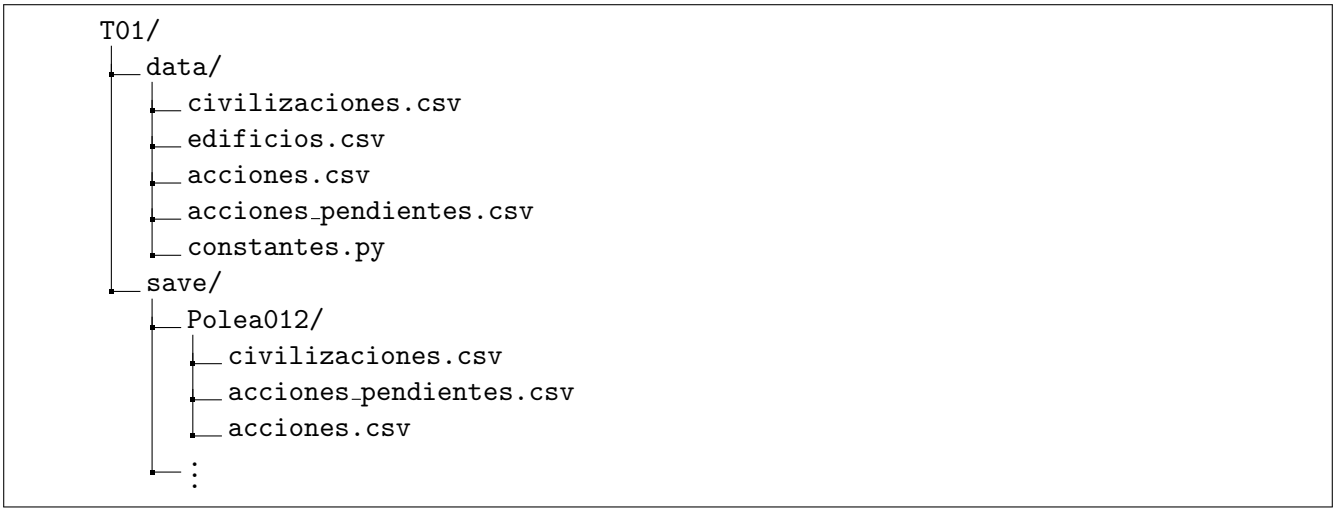

Figura 1: Árbol de direcciones de los archivos y carpetas.

En `data/` se encuentran almacenados los archivos que fueron mencionados en el punto anterior. Estos archivos almacenan la información necesaria para crear una nueva partida de DCCivilization.

En `save/` se deberán almacenar las subcarpetas que tendrán como nombre el usuario de la partida, estas contendrán el estado actual de la partida guardada. Cada estado de una partida está compuesto por los archivos `civilizaciones.csv`, `acciones_pendientes.csv` y `acciones.csv`.

A continuación se explica cómo deberás desarrollar el guardado y la carga de los archivos de una partida.

6.2.1. Guardar una partida

Para almacenar la información actual de cada civilización y las acciones restantes a realizar[3] deberás hacer lo siguiente:

- Crear una carpeta dentro de `save/` cuyo nombre sea el nombre del usuario.
- Dentro de la carpeta debes almacenar los siguientes archivos.
  - `civilizaciones.csv`: este archivo tendrá el estado actual de cada civilización, por lo que deberás actualizar la información del archivo original y guardarlo en su nueva carpeta (el archivo que se utilizó para cargar o crear la partida debe mantener en su estado original).
  - `acciones_pendientes.csv`: este archivo almacena la información de las acciones que se están realizando actualmente, es decir, almacena la información de los edificios o personas que se encontraban en producción cuando se guardó la partida y aún no se completan.
  - `acciones.csv`: este archivo almacena la información de las acciones futuras (de las civilizaciones enemigas), es decir, las acciones que no se han ejecutado hasta el momento del archivo `acciones.csv` con el cual se cargaron las acciones para el resto del juego. Para esto deberás utilizar el mismo formato del archivo de las acciones para guardar las acciones restantes.
- Los datos que se calculan a base de una distribución uniforme y las constantes entregadas en el archivo `constantes.py` no deben ser almacenados en ningún archivos, lo que se hará con estos

[3]Esto puede ser tanto las acciones que se están realizando actualmente (por ejemplo, la construcción de un edificio), como la acciones futuras de las otras civilizaciones.

datos se explicará en el siguiente punto.

6.2.2. Cargar una partida

Para cargar una partida ya almacenada, deberás realizar lo siguiente:

- Ofrecer al usuario un listado de todas las partidas ya almacenadas. El usuario debe poder seleccionar una para cargar su información.[4]
- Se debe revisar que en la carpeta seleccionada se encuentre todos los archivos necesarios para cargar la partida (`civilizaciones.csv`, `acciones_pendientes.csv` y `acciones.csv`). En caso de que falte alguno se le deberá informar al usuario y anular la carga de la partida.
- Además, deberás acceder a los archivos `edificios.csv` y `constantes.py` que se encuentran en la carpeta `data/` para cargar la partida.
- Deberás leer los archivos y cargar su información en tu programa, al igual que lo harías si es que crearas una nueva partida.
- Los datos que no se encuentran especificados en los archivos, como los valores aleatorios que son asignados a las personas y edificios, deben volver a calcularse al momento de cargar la partida.
- Para saber cuál civilización está controlando el usuario, utiliza la columna `usuario` del archivo `civilizaciones.csv`. Las civilizaciones que no sean controladas por el usuario basaran sus acciones en lo indicado en el archivo `acciones.csv`.
- Para saber el turno en el que actualmente va la partida, debes fijarte en la primera línea del archivo `acciones.csv`.

7. 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. Como avance de esta tarea, a entregar la primera semana de la tarea, se pide realizar este diagrama. Esto permitirá entregarles *feedback* de cómo va la modelación de su programa. Luego de esto, deben entregar una versión final de su diagrama junto a la entrega final, que represente fielmente la modelación del problema de su programa.

En ambos casos, el diagrama debe contener las clases junto con sus atributos, y métodos y todas las relaciones existentes entre ella (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.

[4] *Hint:* Para esto puedes usar la función `listdir()` de la biblioteca `os`.

8. 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 3.6.
- Si no se encuentra especificado en el enunciado, asume 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 24 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).