Gwen't

Proyecto semestral

Nancy Hitschfeld
Profesora

Matías Toro
Profesor

Ignacio Slater
Auxiliar

Daniel Ramírez
Auxiliar

Beatriz Graboloza
Auxiliar

Santiago, Chile
2023-04-12
v1.0.2

Índice

1. Descripción

El proyecto a realizar será crear un clon (simplificado) del juego de cartas *Gwent* desarrollado por *CD Projekt RED*.
A grandes rasgos el juego tendrá dos jugadores, uno controlado por el usuario y otro controlado por la computadora.
Cada jugador tendrá un mazo y una mano de cartas que puede jugar en un tablero en una partida al mejor de tres rondas.
A continuación se explicarán en detalle los requisitos que debe cumplir su programa al finalizar el proyecto.

1.1. Jugador

Llamaremos jugador a cada uno de los participantes de una partida, estos serán controlados por un usuario y una computadora[1].
Cada jugador tiene:

- **Nombre:** Es el identificador del jugador. Es una manera de distinguir a los distintos participantes en la partida.
- **Sección del tablero:** Es la zona del tablero que le corresponde al jugador. En ella, el jugador ubicará sus cartas.
- **Contador de gemas:** Es la cantidad de gemas que tiene el jugador. Estas gemas representan la «vida» del jugador.
- **Mazo de cartas:** Es el conjunto de cartas que tiene el jugador. Desde aquí, podrá robar cartas.
- **Mano de cartas:** Es el conjunto de cartas que el jugador tiene en su mano. Estas cartas pueden ser utilizadas para jugar durante la partida.

Las acciones que un jugador puede realizar son:

- **Jugar una carta de su mano:** Seleccionar una carta de su mano y colocarla en el tablero para realizar una acción.
- **Robar una carta del mazo:** Tomar una carta del mazo y agregarla a su mano.

1.2. Tablero

El tablero es la representación física del campo de batalla en el que los jugadores colocan sus cartas. Está dividido en dos secciones simétricas, una para cada jugador, y cada sección se subdivide en tres zonas: la **zona de combate cuerpo a cuerpo**, la **zona de combate a distancia** y la **zona de asedio**. Utilizaremos indistintamente los términos *tablero* y *campo de batalla*, así como los términos *zona* y *fila*.

Además de las zonas de combate, existe una zona adicional para las cartas de clima que afectan el campo de batalla y pueden tener efectos como modificar la fuerza de las unidades en diferentes zonas. Esta zona es compartida por ambos jugadores, por lo que solo puede haber una carta de clima en el campo.

La única diferencia entre las zonas del tablero son las cartas que pueden ser colocadas en cada una de ellas.

1.3. Cartas

En el juego, las cartas desempeñan un papel fundamental. Existen dos tipos principales de cartas: las **cartas de unidad** y las **cartas de clima**.
Las cartas de unidad son aquellas que cada jugador debe colocar estratégicamente en las zonas correspondientes del campo de batalla. Estas cartas tienen tres clasificaciones: cuerpo a cuerpo, a distancia y de asedio, cada uno de los cuales determina la ubicación en la que puede ser colocada. Por ejemplo, las cartas cuerpo a cuerpo solo pueden ser colocadas en la zona de combate cuerpo a cuerpo, mientras que las cartas de distancia solo pueden ser colocadas

[1] El comportamiento de la computadora se explica en la sección 1.5.1

en la zona de combate a distancia. Además, cada carta de unidad tiene un número que representa su valor de fuerza.

Por otro lado, las cartas de clima son cartas especiales que pueden ser colocadas en la zona de clima. Estas cartas tienen el poder de afectar el campo de batalla y brindar ventajas o desventajas a los jugadores, dependiendo del tipo de clima que se haya elegido.

Cada carta tiene un nombre y su clasificación que la identifica. Finalmente, cada carta de unidad puede tener su propia habilidad especial, esto se explica en la sección 1.4.

1.4. Efectos

Las cartas de unidad pueden tener una habilidad especial que se activa cuando la carta es colocada en el campo de batalla y que se mantiene activo mientras la carta permanezca en el tablero[2].

Considere que no todas las cartas de unidad tienen una habilidad especial, pero todas las cartas de clima sí poseen una.
A continuación se presentan los efectos que debe implementar su programa:

1.4.1. Cartas de unidad

- **Refuerzo moral:** Cuando la carta entra en el campo, añade +1 a la fuerza de todas las cartas en su fila, excepto a sí misma.
- **Vínculo estrecho:** Si ya existe una carta con el mismo nombre en la fila, duplica la fuerza de esa(s) carta(s), incluyéndose a sí misma.

1.4.2. Cartas de clima

- **Escarcha mordiente:** Establece el valor de fuerza de **todas** las cartas de combate cuerpo a cuerpo en 1.
- **Niebla impenetrable:** Establece el valor de fuerza de **todas** las cartas de combate a distancia a 1.
- **Lluvia torrencial:** Establece el valor de **todas** las cartas de asedio a 1.
- **Clima despejado:** Elimina todos los efectos climáticos actualmente en efecto en el campo de batalla.

1.5. Reglas del juego

El objetivo del juego es acumular más fuerza en el campo de batalla que el oponente al final de cada ronda. Al finalizar una ronda, el jugador que tenga menos fuerza total en su zona pierde una gema. El juego finaliza cuando un jugador pierde todas sus gemas. Si se da la situación que ambos jugadores tengan la misma fuerza al término de una ronda, esta es considerada un empate, en cuyo caso ambos perderán una gema.

Cada jugador cuenta con un mazo de 25 cartas de cualquier clasificación y 2 gemas. Al inicio de la primera ronda, ambos jugadores barajan sus mazos y roban 10 cartas cada uno. Luego, para las siguientes rondas, barajan sus mazos y solo podrán robar 3 cartas cada uno, pudiendo tener hasta un máximo de 10 en sus manos.

Cada ronda se divide en turnos alternados, donde cada jugador tiene la oportunidad de jugar una sola carta de su mano o pasar su turno, es decir, no jugar ninguna carta. Cuando un jugador pasa su turno, el otro puede jugar tantos turnos como desee hasta que no tenga más cartas que pueda jugar o hasta que también pase su turno. Si un jugador no tiene cartas que pueda jugar en su mano, entonces pasa su turno automáticamente.

[2]Piense que al implementar un proyecto real este comportamiento podría cambiar al agregar nuevas cartas y mecánicas al juego. ¿Cómo podría hacer para que su programa sea flexible a este tipo de cambios?

1.5.1. Computadora

La computadora jugará cada turno de forma automática siguiendo las siguientes reglas:

1. Si la suma de fuerza de las cartas que tiene en el campo de batalla y en mano es mayor que la del campo de batalla del oponente, entonces juega una carta al azar de su mano hasta que el oponente pase su turno.
2. Si la suma de fuerza de las cartas que tiene en el campo de batalla y en mano es menor que la del campo de batalla del oponente, entonces intentará jugar una carta de clima al azar. En caso de que no tenga una carta de clima, pasará su turno.

2. Arquitectura

La resolución de este proyecto se hará siguiendo el patrón arquitectónico *Modelo-Vista-Controlador*, donde primero se implementará el *modelo*, luego el *controlador* y por último la *vista*. Este patrón se explicará en más detalle en el transcurso del curso, pero en el contexto del proyecto estos componentes serán como se explica a continuación.

Modelo Para la primera parte se le solicitará que cree todas las entidades necesarias que servirán de estructura base del proyecto y las interacciones posibles entre dichas entidades. Las entidades en este caso se refieren a los elementos que componen el juego.

Vista Se le pedirá también que cree una interfaz de consola simple para el juego que pueda responder al input de un usuario y mostrar toda la información relevante del juego en pantalla.

Controlador Servirá de conexión lógica entre la vista y el modelo, se espera que el controlador pueda ejecutar todas las operaciones que un jugador podría querer efectuar, que entregue los mensajes necesarios a cada objeto del modelo y que guarde la información más importante del estado del juego en cada momento.

3. Evaluación

3.1. Puntaje

- **Código fuente (4.0 puntos):** Este ítem se divide en 2:
  - **Funcionalidad (1.5 puntos):** Se analizará que su código provea la funcionalidad pedida. Para esto, se exigirá que testee las funcionalidades que implementó. **Si una funcionalidad no se testea, no se podrá comprobar que funciona y, por lo tanto, NO SE ASIGNARÁ PUNTAJE EN ESTE APARTADO por ella.**
  - **Diseño (2.5 puntos):** Se analizará que su código provea la funcionalidad implementada utilizando un buen diseño.
- **Coverage (1.0 puntos):** Sus casos de prueba deben crear diversos escenarios y contar con un *coverage* de al menos un 90 % de las líneas. No está de más decir que sus tests deben *testear* algo, es decir, no ser tests vacíos o sin *asserts*.
- **Documentación y estilo (1.0 puntos):** Cada clase, interfaz y método debe ser debidamente documentado. Siga la guía de documentación dejada en la sección de Enlaces.

[2]Se le recomienda enfáticamente que piense en cuales son los casos de borde de su implementación y en las fallas de las que podría aprovecharse un usuario malicioso.

Adicionalmente, para guiar la realización de este proyecto, se plantearán objetivos pequeños en forma de *entregas parciales* que buscarán enfrentar una problemática a la vez.

- **Entrega parcial (0.5 puntos c/u):** Dichas entregas deberán ser enviadas en la fecha señalada y no serán evaluadas directamente con nota. Solamente será evaluado que al momento de entregarla se cumpla con el objetivo planteado, **independientemente de su diseño**.

La no entrega de una entrega parcial supondrá un descuento en la cantidad de puntos señalada. Las tarea 1 tendrá una entrega parcial, la tarea 2 tendrá dos, y la tarea 3 incluirá dos, o posiblemente tres.

3.2. Modalidad de entrega

Todo su trabajo debe ser subido al repositorio privado que se le entregó a través de *Github Classroom*. La modalidad de entrega será mediante un archivo de texto **(.txt)** entregado mediante la sección Tareas de *U-cursos* que tenga el formato:

```
Nombre: <Su nombre>
PR: <URL del Pull Request>
```

Cuando tenga una entrega lista para ser revisada, deberá realizar un **pull request** a la rama main, esta será la versión que será revisada. Tenga en cuenta que si hace *push* de otros *commits* en esa rama se actualizará el PR, por lo que deberá crear una nueva rama para seguir trabajando.

3.3. Bonificaciones

Además del puntaje asignado por cumplir con los requisitos anteriores, puede recibir puntos adicionales (*que se sumarán a su nota total*) implementando lo siguiente:

TAREA 1,2,3 **Readme** (0.2 pts.): Obtendrá puntaje extra si utiliza su archivo README.md para detallar explicaciones sobre qué hizo, por qué lo hizo y las intenciones detrás de su código.

TAREA 2,3 **Manejo de excepciones** (0.3 pts.): Se otorgará puntaje por la correcta utilización de excepciones para manejar casos de borde en el juego. Recuerde que atrapar *runtime exceptions* es una mala práctica, así como arrojar un error de tipo `Exception` (esto último ya que no es un error lo suficientemente descriptivo). Bajo ninguna circunstancia una de estas excepciones debiera llegar al usuario.

TAREA 3 **Interfaz gráfica avanzada** (0.5 pts.): Si su interfaz gráfica cumple más de los requisitos mínimos podrá recibir una bonificación de 0.5 pts.

TAREA 3 **Patrones de diseño adicionales** (0.5 pts.): Si realiza más (y distintos) patrones de diseños enseñados en el curso, podrá obtener esta bonificación.

4. Recomendaciones

Como se mencionó anteriormente, no es suficiente que su tarea funcione correctamente. Este curso contempla el diseño de su solución y la aplicación de buenas prácticas de programación que ha aprendido en el curso. Dicho esto, no se conforme con el primer diseño que se le venga a la mente, intente ver si puede mejorarlo y hacerlo más extensible.

NO EMPIECE LA TAREA A ÚLTIMO MOMENTO. Esto es lo que se dice en todos los cursos, pero es particularmente importante y cierto en este. **Si usted hace la tarea a último minuto lo más seguro es que no tenga tiempo para reflexionar sobre su diseño, y termine entregando un diseño deficiente o sin usar lo enseñado en el curso.**

Haga la documentación de su programa en inglés (no es necesario). La documentación de casi cualquier programa *open-source* se encuentra en este idioma. Considere esta oportunidad para practicar su inglés.

Si hay algo que no está especificado en este enunciado, usted tiene la total libertad de implementarlo a su gusto, siempre que cumpla con lo solicitado y aplicando el contenido enseñado.

Considere (si lo desea) una baraja estándar como:

- 18 Cartas de unidad, repartidas equitativamente entre las tres clasificaciones,
- 6 - 10 Cartas de unidad con efecto, idealmente que no sean todas de la misma clasificación, y
- 7 Cartas de clima.

Por último, el orden en el que escriben su programa es importante, se le sugiere que para cada funcionalidad que quiera implementar:

1. Cree los *tests* necesarios para verificar la funcionalidad deseada, de esta manera el enfoque está en como debería funcionar ésta, y no en cómo debería implementarse. Esto es muy útil para pensar bien en cuál es el problema que se está buscando resolver y se tengan presentes cuales serían las condiciones de borde que podrían generar problemas para su implementación.
2. Escriba la firma y la documentación del método a implementar, de esta forma se tiene una definición de lo que hará su método incluso antes de implementarlo y se asegura de que su programa esté bien documentado. Además, esto hace más entendible el código no solo para alguna persona que revise su programa, sino que también para el mismo programador.
3. Por último implemente la funcionalidad pensando en que debe pasar los tests que escribió anteriormente y piense si estos tests son suficientes para cubrir todos los escenarios posibles para su aplicación, vuelva a los pasos 1 y 2 si es necesario.

[2]Entender un programa mal documentado que haya escrito uno mismo después de varios días de no trabajar en él puede resultar bastante complicado.