Battle Cards

> Proyecto de Programación II. Facultad de Matemática y Computación - Universidad de La Habana. Curso 2022.

El propósito de este proyecto es implementar una biblioteca de clases y una aplicación visual para una variante de la familia de juegos conocidos como *Trading Cards Games* de su propia concepción.

Usted deberá entregar una solución compuesta, al menos, por dos proyectos:

- Una biblioteca de clases, implementada en C# 10, .NET Core 6, donde se implemente toda la funcionalidad lógica de su solución.
- Una aplicación visual, implementada con tecnologías de libre elección (preferible en el lenguaje de programación C#) que permita visualizar el resultado de los juegos.

Descripción del juego

Los juegos de tipo *Trading Cards* tienen varias características en común que describimos a continuación, pero tenga en cuenta que dentro de estas características, la diversidad y flexibilidad de este tipos de juegos es muy alta, por lo que usted tendrá una libertad considerable en términos de diseño del juego, reglas, condiciones de victoria, etc.

La característica fundamental de este tipo de juegos es que cada jugador tiene un conjunto de cartas, que puede ir coleccionando a partir de diferentes partidas. Por ejemplo, en un juego ambientado en la mitología de Tolkien, las cartas pueden representar unidades tales como arqueros elfos, guerreros enanos, magos, fortalezas, monstruos como el Balrog, o héroes como Gandalf.

Cada carta tiene características o atributos propios que dependen del tema o dominio del juego. Siguiendo con el ejemplo anterior, cada carta puede tener una vitalidad, una puntuación de magia, un nivel de moral, resistencia a diversos elementos, etc. Las cartas también pueden tener estados temporales, por ejemplo, envenenado, quemado, baja moral, etc.

Además, cada carta tiene un conjunto de acciones que puede realizar, que puede depender de la existencia de otras cartas sobre la mesa, o incluso del valor de ciertos atributos. Además, cada acción puede tener varios efectos, tales como modificar el valor de algún atributo (e.j., reducir la vitalidad de un enemigo), aplicar o quitar un efecto, etc. Incluso hay variantes donde una acción puede producir que cartas de un jugador pasen a otro, ya sea temporalmente o no.

Una partida consiste entonces en un enfrentamiento entre dos (o más jugadores), que pueden estar o no en equipos, y que pueden tener una cantidad fija, aleatoria, o preseleccionada de cartas. Estos jugadores toman turnos, y en cada turno escogen una o más cartas, y usan una o más de las acciones, en función de las reglas de cada juego específico. Las condiciones para que la partida termine pueden ser múltiples según el juego, tales como que todas las cartas hayan sido eliminadas, o cierta cantidad de turnos, etc.

Teniendo en cuenta esta descripción general, usted debe escoger una temática que desee, diseñar un conjunto de atributos y reglas específicos para su temática, e implementar un sistema que permita a uno o más usuarios jugar partidas, así como crear cartas propias (atendiendo a las restricciones de su juego), e incluir al menos un jugador virtual sencillo.

Aplicación visual

Para mostrar sus resultados usted debe implementar una aplicación visual que permita ejecutar una partida de su juego, configurando todas aquellas opciones que definen una partida, ya sea el conjunto de cartas de cada jugador, o elementos como el terreno, la cantidad de turnos, etc., en función de lo que permita su diseño.

La aplicación puede ser tan sencilla como una aplicación de línea de comandos (o sea, en la consola prieta), una aplicación web, una aplicación con gráficos en 3D, una aplicación móvil, etc. Queda a su decisión cuánto esfuerzo ponerle a dicha aplicación, pero al menos tiene que tener las siguientes funcionalidades.

- Permitir al usuario configurar una partida de su juego personalizando todos los elementos que su juego permite variar, incluyendo los jugadores virtuales a utilizar.
- Mostrar el desarrollo de la partida paso a paso de forma interactiva, o directamente hasta el final. Dependiendo de la tecnología que escoja, esto puede ser desde imprimir en la terminal la carta y acción jugada hasta mostrarla en 3D.
- Permitir al usuario crear nuevas cartas utilizando un mini-lenguaje de programación (definido por usted) para las acciones.

Lenguaje de programación para crear cartas

A través de su aplicación visual, el usuario deberá poder crear nuevas cartas, personalizando no solo los atributos (e.j., nombre, descripción, vitalidad, magia, etc.) sino fundamentalmente las acciones, incluyendo las condiciones para ser ejecutadas y los efectos que cada acción realiza.

Para esto, usted diseñará un mini-lenguaje de programación, con una sintaxis definida por usted, que permita especificar todo lo necesario para definir una carta nueva.

Usted es libre de decidir cuán complejo y expresivo es su lenguaje, pero al menos debe permitir las siguientes funcionalidades:

- Definir los atributos iniciales de la carta, e.j., nombre, puntos de vida o magia, ataque, defensa, …
- Definir un listado de acciones disponibles para dicha carta, e.j., ataque de rango, lanzar bola de fuego, subir moral, …
- Por cada acción, definir los efectos que dicha acción tiene, e.j., reducir los puntos de vida en X cantidad sobre la carta “atacada”, activar el estado “quemado”, “revivir” una carta eliminada, …
- Por cada acción, definir las condiciones para poder activarla, e.j., que los puntos de vitalidad sean más de X, que exista al menos una carta de tipo Y en la partida, …

No basta con que usted defina un conjunto básico de efectos parametrizables (e.j., `BajarVida(float x)`). Su lenguaje debe permitir al usuario crear efectos que dependan del estado de la partida y de los atributos de las cartas existentes. Algunos ejemplos de efectos que podría ser posible programar con su lenguaje son:

- Eliminar tantos puntos de vida como puntos de magia tenga la carta lanza la acción.
- Al hacer un daño X a una carta, rebota hacia otra carta aleatoria y hace un daño X/2, y así sucesivamente hasta que el daño llegue a 0.
- Actuar sobre las N cartas más débiles del contrario.
- Coger la carta más fuerte del contrario, e intercambiar sus puntos de vida con los de la carta que lanza la acción.
- …

Igualmente, con las condiciones, no basta solamente implementar algo como `TenerVidaMayorQue(float x)`, sino que debe ser posible definir condiciones que dependan de los valores de los atributos y estados de las cartas en la partida. Por ejemplo:

- Que el contrario tenga al menos una carta con vitalidad mayor que la carta que lanza la acción.
- Que las tres cartas más “fuertes” del contrario tengan mayor vitalidad que la carta que lanza la acción.
- Que la última vez que esta carta lanzó una acción, haya eliminado a la carta “atacada”.
- …

Por supuesto, debe ser posible definir condiciones compuestas mediante operadores lógicos, por ejemplo, tener vida mayor que X1 y tener magia mayor que X2.

Note que los ejemplos anteriores son solo eso, ejemplos, que puede que no tengan sentido en su juego según la temática que escoja. La idea que debe llevarse es que necesita diseñar un lenguaje con suficiente nivel de expresividad tal que algunos efectos y condiciones de complejidad similar a los ejemplos mostrados sean programables.

Jugadores virtuales

Su implementación debe incluir al menos un jugador virtual que implemente una estrategia no aleatoria. No es requisito que este jugador sea infalible, ni siquiera notablemente bueno, pero al menos debe tener algún tipo de lógica a la hora de escoger la carta y acción a ejecutar que dependa del estado actual del juego (o sea, no aleatorio) y que sea al menos localmente óptima bajo algún criterio. Esto no quita que su jugador tenga algún componente aleatorio, ya sea para dar variedad al juego o para distraer al oponente (o ambos).

Un ejemplo sencillo puede ser escoger aquella carta que tiene la acción que realizaría el mayor daño teniendo en cuenta las cartas que tiene el contrario. Note que esta no tiene por qué ser una estrategia óptima a largo plazo, ya que dicha carta puede que sea mejor guardarla para otro momento, pero al menos es una estrategia óptima en el turno actual si se mira como criterio de optimalidad el daño realizado al jugador contrario.

Usted puede implementar más de un jugador virtual con diferentes niveles de “inteligencia” si así lo desea, o simplemente para explorar diferentes estrategias.

Ingeniería de software

El propósito fundamental de este proyecto es evaluar el uso de las buenas prácticas de desarrollo de software que se han enseñado y se seguirán enseñando en el curso. Por tal motivo, se pondrá especial énfasis en evaluar el diseño de las clases, interfaces, y métodos, y se profundizará durante la evaluación en la justificación que usted tenga para las decisiones de diseño de software que ha tomado.

Esto tendrá mayor peso en la evaluación final que cualquier consideración sobre la funcionalidad en sí. La calidad de la interfaz gráfica, los efectos de sonido, la inteligencia artificial de los jugadores, todo eso aporta y será considerado, pero es secundario ante un buen diseño de clases que sea extensible y mantenible.

Algunos consejos:

- Recuerde los principios SOLID y diseñe en consecuencia.
- Priorice las interfaces sobre las clases abstractas, y la composición sobre la herencia.
- Recuerde que las abstracciones se diseñan encima de las funcionalidades a

implementar, y no solamente encima de los “conceptos” o “entidades” convencionales del dominio.
- Recuerde el principio DRY y las estrategias de encapsulamiento existentes para reutilizar código y evitar la duplicidad de funcionalidades.
- Utilice nombres de clases y variables descriptivos, y adicione comentarios en su código para explicar las motivaciones detrás de su diseño.
- Y sobre todo, recuerde que ningún diseño pensado a priori sobrevive la prueba del tiempo. Empiece con el diseño más sencillo posible e introduzca abstracciones a medida que descubre que las necesita.

Entrega

Dada la complejidad del proyecto, el mismo se hará en equipos de dos o tres personas. Usted tiene la posibilidad de hacerlo individualmente, pero tenga en cuenta que la complejidad será mucho mayor.

La entrega del proyecto será exclusivamente mediante un repositorio en Github.

En la raíz de su proyecto debe haber un archivo `Readme.md` con **todas las instrucciones necesarias** para ejecutar su código, así como todos los detalles de instalación (si necesitara algo más allá de .NET Core 6) y el uso de su interfaz gráfica.

Además, usted debe tener en la raíz de su repositorio un archivo de nombre `Report.pdf` que contendrá un reporte técnico de su proyecto. Este reporte tendrá una extensión mínima de 5 cuartillas, y ahí debe detallar su diseño de clases, exponiendo las motivaciones detrás de cada abstracción (clase, interfaz, método, etc.) así como todos los detalles de implementación que sean necesarios. Incluya fragmentos de código y capturas de pantalla para ayudar en la explicación.

La evaluación se realizará en dos partes. Primero, su mentor evaluará el reporte y hará los comentarios que considere necesarios a su proyecto. Una vez que el mentor considere que su proyecto está en condiciones de ser expuesto, le dará una cita para exponer en persona en la Facultad. La nota final será decidida por un tribunal que tendrá en cuenta las consideraciones de su mentor así como el código fuente y el reporte entregado.

> Los detalles sobre la fecha de entrega y formato exacto serán especificados más adelante.