

Universidad de la Habana

FACULTAD DE MATEMÁTICA Y COMPUTACIÓN

*Proyecto Intrasemestral de Programación
Ave Cesar*



Autores:
Amanda Cordero Lezcano
Christopher Guerra Herrero

Diciembre, 2022

1 Información básica

1.1 Breve Introducción

El presente trabajo constituye el proyecto de programación, de segundo semestre de primer año de los autores. El mismo consiste en crear un videojuego con una interfaz visual sobre la temática de Juegos de Cartas Coleccionables. En este se reta a los estudiantes a cumplir con determinadas tareas sin dejar de acatar los principios y buenas prácticas de la ingeniería de software. A lo largo de este documento se explicará como los autores enfrentaron estos problemas y cuales fueron las razones que los llevaron a tomar cada decisión.

1.2 Datos generales de las herramientas usadas

Para la creación de este software se han usado diferentes programas y recursos. La confección del código y de las bibliotecas de clases se realizó con .Net6.0, mientras que para el diseño visual se recurrió a Unity19.04. Además se necesitaron herramientas complementarias como GIMP, Audacity y La-Tex.

1.3 Sobre el juego

La trama del videojuego se basa en el enfrentamiento de *Hackers* y *Algoritmos*. Las reglas del juego son explicadas en el Readme.md adjunto en el proyecto. El ejecutable es compatible con las distribuciones de Linux y en pantallas de resolución 16:9, este último aspecto es un requisito para la correcta visualización del programa.



2 Aspectos sobre la jerarquía de clases

En la carpeta del proyecto hay dos grandes grupos de documentos .cs: la biblioteca BattleCards, que contiene las funciones lógicas del software, y un conjunto de scripts que permiten el funcionamiento de la interfaz visual.

2.1 Radiografía para Ave Cesar

2.1.1 La clase Game

La clase mencionada en el enunciado contiene el estado del juego en un momento dado. Esta es la encargada de contar los turnos transcurridos y contener a los jugadores(instancias de la clase Player). Por su parte, la clase Player modela las características de un jugador, tiene campos referente a la cantidad de información encriptada, las cartas en la mano y en el campo de los mismos. La clase Life es instanciada en los jugadores, mas también es el modelo usado en la vida de otros objetos del juego(como los Algoritmos). La clase Field está compuesta de contenedores de cartas para el campo de los Hackers, los Algoritmos y las clases de efecto especial, al igual que la clase Hand. Una instancia de la clase *Container < T >* es la responsable de almacenar y disponer de los métodos necesarios para gestionar un conjunto de elementos T (where T : Card), ya sean esto un campo o la mano de los jugadores.

2.1.2 La clase Card

Esta estructura contiene los elementos inherentes a todas las cartas del juego(nombre, descripción), así como un array con los elementos que esta clase no puede explotar por su nivel de abstracción. La información de las cartas del juego es almacenada en .txt, y el constructor de Card asimila esta información.

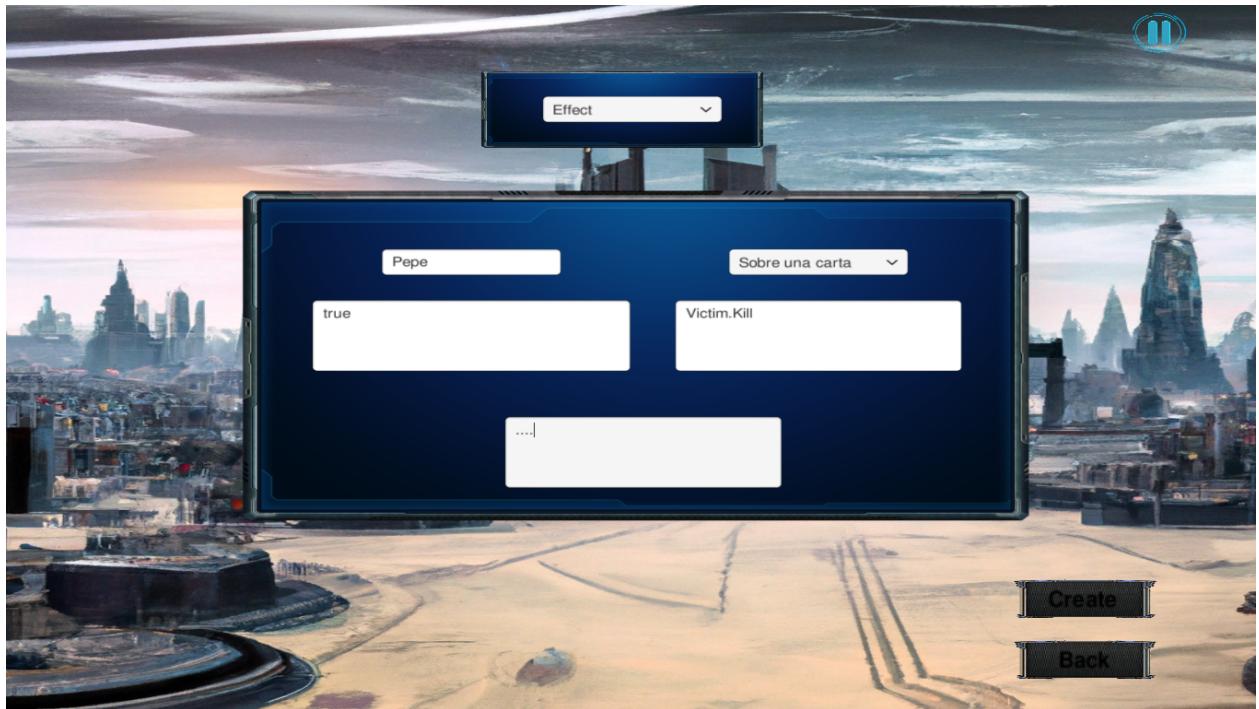
De la clase Card heredan las clases Hacker, Algorithm y Effect, y de esta última heredan las clases EffectOverField y EffectOverCard.

2.1.3 El jugador Virtual



Entre los retos del proyecto se encontraba la confección de un jugador virtual sencillo. Para el nuestro no se ha recurrido a principios teóricos complejos(IA), sino a condicionales que hacen que el mismo sea poco estúpido. El mismo prioriza su defensa en todo caso, defendiendo aquellas posiciones que pueden perjudicar más la vida del jugador. En un segundo momento el jugador selecciona una carta efecto bajo un criterio probabilístico; en el futuro esto pudiera modificarse haciendo los estados del juego Comparables con el fin de poder simular algunas jugadas y decidir que movimiento es el mejor, pero decidimos no reparar en esto por el momento. Por último el jugador intenta usar un hacker para debilitar la vida de su enemigo, ubicándolo en la posición menos defendida. Tenga en cuenta que para lo anterior, los players cuentan con un número finito predeterminado de jugadas por turnos.

2.1.4 Para crear cartas



Nuestro juego cuenta con una sección para la confección de cartas. Desde dicha sección se pueden crear los tres tipos de cartas que se usan en el juego(para ser consecuentes). En el caso de los tipos Algoritmo y Hacker el usuario debe proveer un nombre, una descripción y un número(capacidad del Hacker o resistencia del Algortimo); esta información es almacenada en un .txt, que luego los constructores de las clases que heredan de Card interpretarán.

En el caso del tipo Effect, el usuario debe proveer nombre y descripción, como en los casos anteriores, además la condición para activar la carta, y el efecto que esta debe realizar al ser activada.

La condición:

De la interfaz IAssessable que contiene el método Evaluate() heredan las clases CompoundExpression y SimpleExpression, donde SimpleExpression evalúa una expresión atómica:

- Literales (true, false)
 - Comparación (la clase Comparation también hereda de IAssessable)
 - Predicados (ForAll y Exist mediante el método ReadBool de la clase Reader)
- Y CompoundExpression evalúa una concatenación mediante conjunciones (and), disyunciones (or) y agrupaciones por parentesis de estas expresiones atómicas. El proceso de evaluación es recursivo

Las cartas de tipo Effect se subdividen en dos grandes grupos (clases Effec-

tOverCard y EffectOverField que heredan de la clase abstracta Effect). En dependencia de a que grupo pertenece la carta a crear su propiedad Function coincide con un delegado:

```
public delegate void Function(); // para EffectOverField  
public delegate void ActionOver< T >(T Victim) where T : Card;  
// para EffectOverCard
```

En la clase Analyzer los métodos estáticos Create(string action) y *Create* < *T* >(string action) se encargan de crear la propiedad Function en dependencia del texto recibido del usuario. Estos métodos separan el string por ; para quedarse con cada una de las acciones que introdujo el usuario y reconocen el efecto mediante palabras reservadas del lenguaje que representan métodos de la clase BasicMethods y sus respectivos parámetros antecedidos por _. Estas acciones son agregadas a Function y asignadas a la carta para ser ejecutadas cuando la carta es activada.

2.2 Fotografía para Ave Cesar

En los scripts usados para la interfaz de usuario se explota al máximo la biblioteca UnityEngine y UnityEngine.UI. Note que en estos documentos se recibe la entrada del usuario, y por tanto se tuvo que ser muy explícito para poder receptionar dichos eventos correctamente. No es objetivo de este documento especificar la arquitectura de estos scripts.