Ah-Gi-Oh Trading Card Game

Introducción

Ah-Gi-Oh es un divertido y dinámico videojuego de cartas coleccionables creado en la plataforma Windows Forms por los estudiantes de la facultad de Matemática y Computación, Universidad de La Habana en la carrera Ciencias de la Computación: Yoel Enriquez Sena, Jean Carlo García Wong y Raciel Alejandro Simón Domenech. Tomando inspiración directa del famoso juego de cartas coleccionables Yu Gi Oh, creamos un sistema enfocado en el *singleplayer* con algunos giros que hacen la experiencia interesante desde un punto de vista estratégico al eliminar y retocar paradigmas fundamentales del género donde el usuario se verá implicado en aspectos nunca antes vistos del mismo.

Sistema de Juego

En el núcleo, Ah-Gi-Oh funciona como cualquier Trading Card Game, en específico posee mecánicas similares a Yu-Gi-Oh, del cual toma inspiración. Se cuenta con un mazo de 15 cartas, mano, cementerio y un campo de juego para cada jugador, además cada uno cuenta con 5000 puntos de vida. El campo de juego se divide en dos filas donde se pueden colocar un máximo de 5 cartas cada una; en la fila superior, también llamada campo de monstruos, se puede colocar las cartas cuyo tipo sea monstruos; y en la inferior, también llamada campo de magias y trampas, se pueden colocar las cartas de tipo magia y trampa. El juego termina cuando se logren reducir los puntos de vida del adversario a 0 o si la cantidad de cartas en el deck de cualquiera de los jugadores sea 0; para esto se debe hacer uso de las cartas tipo monstruo y atacar con estas a los monstruos del rival, el monstruo con menos puntos de ataque es destruído y enviado al cementerio y el jugador tomará daño(osea se reducirá sus puntos de vida) igual a la diferencia entre sus ataques. Los monstruos también cuentan con habilidades especiales que llamamos *efectos,* las cuales permiten cambiar las tornas de la batalla dándonos acceso a múltiples alternativas como destruir una carta rival o robar

una carta adicional de nuestro mazo entre otras. Los monstruos más poderosos(indicado por el nivel) requieren enviar al cementerio uno o dos monstruos(dependiendo del nivel) para poder ser usados, acción que llamamos sacrificar. Además solo se puede usar(invocar) un monstruo por turno, si quieres ejecutar más acciones en tu turno se debe hacer uso de las cartas magias y trampas. Las cartas mágicas y trampas son cartas especiales que nos permiten hacer más acciones durante nuestro turno, no atacan, no interactúan con otros monstruos y no hacen daño a no ser que se indique lo contrario en su efecto, los cuales al igual que el efecto de los monstruos pueden variar desde destruir otras cartas, permitirnos robar cartas adicionales, recuperar puntos de vida, entre otras.

Esto suena muy familiar, pero: ¿qué hace único a Ah Gi OH? Una de las acciones más importantes y definitorias de los Trading Card Games es la *creación de mazos*, se podría decir que gran parte del juego se decide en gran medida en que tan equilibrado y bien construído está tu mazo, el cual determina la diferencia entre la victoria y la derrota, eliminando el factor suerte en favor de un correcto equilibrio en la construcción de este.
Sin embargo Ah Gi Oh ELIMINA por completo la creación de mazos, construyéndose de forma aleatoria. En su lugar esta mecánica se sustituye por la *creación de cartas customizables*, dando al jugador la posibilidad de crear sus propias cartas, permitiendo gran control sobre las posibilidades que ofrece el juego. Sin embargo crear cartas muy poderosas y usarlas para destruir al rival no es divertido, por esto las 15 cartas del mazo se seleccionan de forma aleatoria, de modo que el oponente puede terminar con cartas que el propio jugador creó, incitando así que exista un equilibrio razonable a la hora de crear una carta: hazla muy poderosa y puedes terminar combatiendo contra un muro irrompible, hazla muy débil y puedes terminar con un ladrillo inutilizable en tu mano por el resto del juego. A continuación se exponen las posibilidades al crear una carta.

Creación de Cartas

Tipos y atributos:

Existen 3 tipos de cartas a crear: monstruos, magias y trampas, los cuales fueron explicados anteriormente.

Además cada carta cuenta con unos atributos específicos que la definen:
-Nombre de la Carta: Simplemente eso, su nombre.
-Elemento de la Carta: El elemento que esta posee.
-Tipo de la Carta: En caso de que la carta sea Mágica o Trampa el tipo será igual al elemento, si es un Monstruo éste será tipo bestia, máquina, serpiente marina, etc.

Si la carta es Mágica o Trampa esta no poseerá ataque, defensa y nivel.
-Ataque: Solamente para las cartas tipo Monstruo. El valor deberá ser solamente de tipo entero(números).
-Defensa: Solamente para las cartas tipo Monstruo. El valor deberá ser solamente de tipo entero(números).
-Nivel: Solamente para las cartas tipo Monstruo. El valor deberá ser solamente de tipo entero(números).

-Efecto: Operación especial que realiza la carta una vez por turno independientemente de su Rol(Monstruo,Mágica o Trampa) y este debe ser añadido o escrito según como lo define nuestro ¨mini-lenguaje¨ de programación el cual explicamos más adelante.

-FrontImage: Es la imagen que se le guarda a esta carta personalizada la cual se guarda en el archivo de la carta en forma de url. El usuario deberá seleccionar una imagen para ponerle a su carta para así poder crearla.

-BackImage: Esta propiedad guarda un string con la url con la imagen trasera de las cartas de yu-gi-oh. Se da por defecto, el usuario no debe seleccionar que imagen trasera le pondrá a su carta.

Efectos:

Además cada carta obviamente cuenta con su efecto, el cuál el jugador puede crear siguiendo unas reglas definidas por un lenguaje creado por nosotros que se explicará a continuación. Se dispone de una variedad de efectos a elegir, además de una serie de condiciones aplicables para que estos sean efectivos, dando pie a la creatividad y customización de estos.

Funcionamiento del lenguaje:

El lenguaje implementado en Ah-Gi-Oh! es simple de comprender para el usuario, consta de tokens básicos resumidos al nombre de los efectos que tenemos predefinidos en nuestro juego. Contiene el sistema if , else para el uso de condicionales que solamente pueden ser usados en ella los specialtokens(via y vim) los cuales hacen referencia a la vida del propio jugador y a la del adversario y además se puede usar cualquiera de las operaciones matemáticas básicas combinadas, una forma correcta de utilizar esta condicional sería:

if{via<vim}
inv;
else
atq;

Notemos que es algo parecido al lenguaje C#, solo que en lugar de usar paréntesis en la sintaxis del if usamos llaves y al final de cada línea debemos usar siempre el carácter “;”.

Tokens:

-inv: Hace referencia al efecto de Invocación Especial, solo se invocan monstruos desde la propia mano.
-red: Hace referencia al efecto Reducir Vida, se le reducen los puntos de vida al adversario en 500.
-rea: Hace referencia al efecto Reducir Ataque, se le reducen los puntos de ataque a un monstruo en el campo del adversario en 500.

-atq: Hace referencia al efecto Atacar Especial, esta carta puede atacar una vez más en ese turno.
-cam: Hace referencia al efecto Cambiar, puede cambiar de posición a un monstruo en el campo del adversario.
-roc: Hace referencia al efecto Robar Carta(Draw), puedes robar una carta de tu propio deck.
-con: Hace referencia al efecto Controlar, puedes controlar un monstruo del campo del adversario y trasladarlo para tu campo.
-inc: Hace referencia al efecto Incrementar Ataque, incrementa el ataque de un monstruo en tu propio campo en 500.
-sub: Hace referencia al efecto Subir, subes un monstruo del campo del adversario a su mano.
-rov: Hace referencia al efecto Robar Vida, le restas 500 puntos de vida al adversario y se los incrementa al usuario.
-des: Hace referencia al efecto Destruir, puedes destruir cualquier carta sobre el campo del oponente.

```
condicionToken = new List<string>();
condicionToken.Add("if"); condicionToken.Add("else");

tokens = new List<string>();
tokens.Add("inv"); tokens.Add("sub");
tokens.Add("red"); tokens.Add("eva");
tokens.Add("rea"); tokens.Add("evp");
tokens.Add("atq"); tokens.Add("dev");
tokens.Add("cam"); tokens.Add("rov");
tokens.Add("roc"); tokens.Add("des");
tokens.Add("rev");
tokens.Add("con");
tokens.Add("neg");
tokens.Add("inc");

specialTokens = new List<string>();
specialTokens.Add("via"); specialTokens.Add("vim");

cards = new List<Card>();
```

Implementación

Ah Gi Oh está implementado en C#, usando la plataforma windows forms para la interfaz gráfica. A continuación se ofrece un resumen de la lógica y las clases usadas.

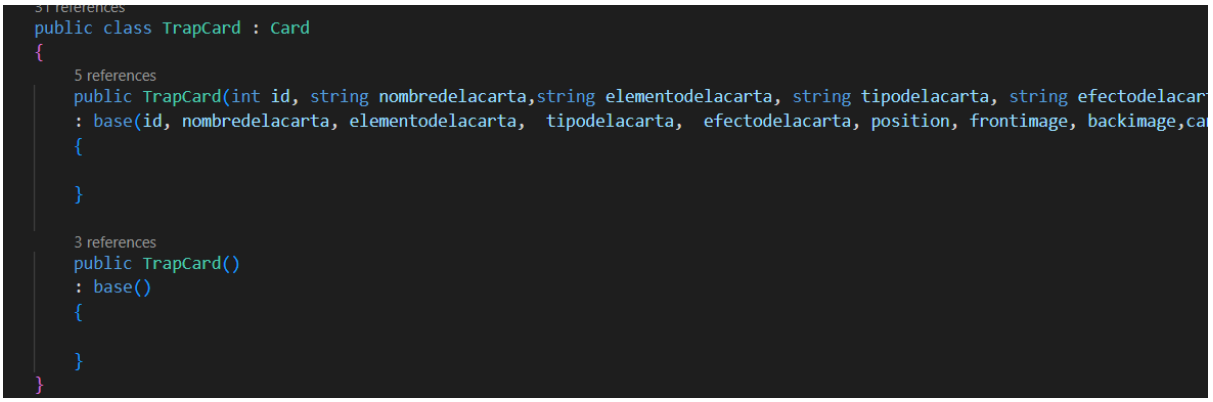
Clases

Card: Usada para definir una carta, cuenta con los campos:

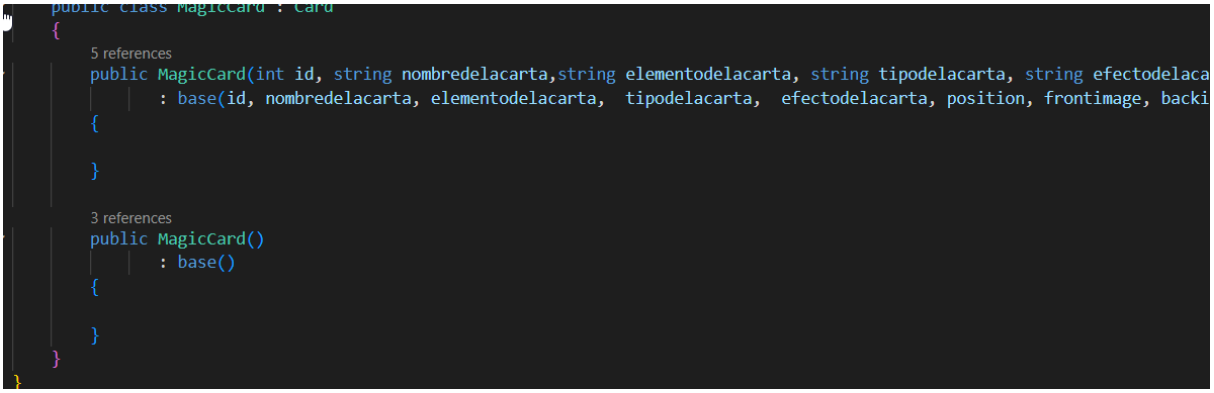
id: para identificar de forma única la carta en cuestión.
nombredelacarta: nombredelacarta.
elementodelacarta: elementolacarta.
tipodelacarta: si es monstruo, trampa o magia.
efectodelacarta: la habilidad de la carta.
position: si se encuentra en defensa o ataque.
frontimage: imagen de la parte delantera de la carta.
backimage: imagen de la parte trasera de la carta.
caneffect: campo usado para determinar si se cumplen las condiciones para activar el efecto.

```
public class Card
{
    4 references
    protected int id;
    4 references
    protected string nombredelacarta;
    4 references
    protected string elementodelacarta;
    4 references
    protected string tipodelacarta;
    4 references
    protected string efectodelacarta;
    4 references
    protected bool position;
    4 references
    protected string frontimage;
    4 references
    protected string backimage;
    4 references
    protected bool caneffect;   //True si puede activar effecto
```

MagicCard y TrapCard: usadas para definir los tipos específicos magia y trampa, ambas heredan de Card.



```
public class TrapCard : Card
{
    5 references
    public TrapCard(int id, string nombredelacarta,string elementodelacarta, string tipodelacarta, string efectodelacar
    : base(id, nombredelacarta, elementodelacarta,  tipodelacarta,  efectodelacarta, position, frontimage, backimage,ca
    {

    }

    3 references
    public TrapCard()
    : base()
    {

    }
}
```



```
public class MagicCard : Card
{
    5 references
    public MagicCard(int id, string nombredelacarta,string elementodelacarta, string tipodelacarta, string efectodelaca
            : base(id, nombredelacarta, elementodelacarta,  tipodelacarta,  efectodelacarta, position, frontimage, backi
    {

    }

    3 references
    public MagicCard()
            : base()
    {

    }
}
```

MonsterCard: Usada para definir el tipo monstruo, también hereda de Card pero este tipo posee ciertas propiedades propias:

niveldelacarta: el nivel define que tan poderoso es un monstruo y si requiere sacrificios o no para ser usado.
ataquedelacarta: cuanto ataque posee un monstruo.
defensadelacarta: cuanta defensa posee un monstruo.
canatq: si puede atacar, normalmente los monstruos solo pueden atacar una vez.

```
66 references
public class MonsterCard : Card
{
    4 references
    protected int niveldelacarta;

    4 references
    protected int ataquedelacarta;

    4 references
    protected int defensadelacarta;

    4 references
    protected bool canatq;          // True: si puede atacar, False: si no puede atacar

    5 references
    public MonsterCard(int id, string nombredelacarta,string elementodelacarta, string tipodelacarta, string efectodela
                       bool position,string frontimage, string backimage,bool caneffect, int niveldelacarta, int ataque
    : base(id, nombredelacarta, elementodelacarta,  tipodelacarta,  efectodelacarta, position, frontimage, backimage,ca
    {
        this.niveldelacarta = niveldelacarta;
        this.ataquedelacarta = ataquedelacarta;
        this.defensadelacarta = defensadelacarta;
        this.canatq = true;
    }
```

CustomMagicCard, CustomTrapCard, CustomMonsterCard: Usadas para diferenciar las cartas por defecto en el juego de las cartas creadas por el usuario. Estas clases existen por motivos referentes a la implementación.

```
0 references
public class CustomMonsterCard : MonsterCard
{
    4 references
    List<CustomAttribute> attribute;

    0 references
    public CustomMonsterCard(int id, string nombredelacarta,string elementodelacarta, string tipodelacarta, string efe
                       bool position,string frontimage,string backimage,bool caneffect, int niveldelacarta, int ataque
    : base(id, nombredelacarta, elementodelacarta,  tipodelacarta,  efectodelacarta, position,frontimage,backimage,can
          niveldelacarta, ataquedelacarta, defensadelacarta, canatq)
    {
        attribute = new List<CustomAttribute>();
    }
```

Player: En esta clase se define todo lo que tiene un jugador a su disposición para jugar. Cuenta con los campos:

deck: el mazo del jugador
hand: las cartas en la mano del jugador
monsterField: las cartas en el campo de monstruos del jugador
magicField: las cartas en el campo de magias y trampas del jugador
graveyard: las cartas en el cementerio del jugador

caninvk: bool donde se comprueba si el jugador tiene disponible su invocación normal, osea si puede colocar un monstruo en el campo de monstruos
Vida: puntos de vida restantes del jugador

Además en esta clase se implementan algunos métodos para manipular estos campos.

```
public class Player
{
    5 references
    private int live;
    5 references
    private List<Card> deck;
    5 references
    private List<Card> hand;

    5 references
    private List<Card> monsterField;

    5 references
    private List<Card> magicField;

    5 references
    private List<Card> graveyard;

    5 references
    private bool caninvk;   // True: si puede invocar  False: no puede invocar

    8 references
    public int Vida{get => live; set => live = value;}
    0 references
    public List<Card> Deck{get => deck; set => deck = value;}
    4 references
    public List<Card> Hand{get => hand; set => hand = value;}

    4 references
    public List<Card> MonsterField{get => monsterField; set => monsterField = value;}

    2 references
    public List<Card> MagicField{get => magicField; set => magicField = value;}

    2 references
    public List<Card> Graveyard{get => graveyard; set => graveyard = value;}

    0 references
    public bool CanInvoke {get => caninvk; set => caninvk = value;}
```

IABot: En esta clase se implementa todo lo relacionado con la IA rival.

```
2 references
public class IABot
{
    2 references
    private int count;

    1 reference
    public IABot(int count)
    {
        this.count = count;
    }

    1 reference
    public  void IAStart(Player IA, Player adversary)
    {
        Operations.DrawNormal(IA);
        InvokePhase(IA);
        if(count != 0)
            BattlePhase(IA,adversary);
        EndPhase(IA);
    }

    int IABot.SacrificeCount(Card card)
    1 reference
    public int SacrificeCount(Card card)
    {
        int result = 0;
        if (((MonsterCard)card).Nivel > 5)
            result = 1;

        return result;
    }
```

Operations: En esta clase se implementan todos los efectos posibles en el juego, tanto por defecto, como los que puede implementar el usuario.

```
public class Operations
{
    2 references
    public static bool InvocarEspecial(Player player, int cardId,Card cardgen) ···

    5 references
    public static bool InvocarNormal(Player player, int n, List<int> sacrifice, bool position = false)   //false signific

    2 references
    public static bool Destruir(Player player, Player playerAdversary, int cardId,Card cardgen) ···

    4 references
    public static void ReducirVida(Player player, Player playerAdversary,Card cardgen, int reduccion = 500) ···

    3 references
    public static bool ReducirAtaque(Player player,Card cardgen,Player playerAdversary, int cardId, int reduccion = 500)
```

Compiler: En esta clase se implementan todas las reglas a la hora de crear efectos. Las reglas se explican en el apartado *Creación de Cartas.*

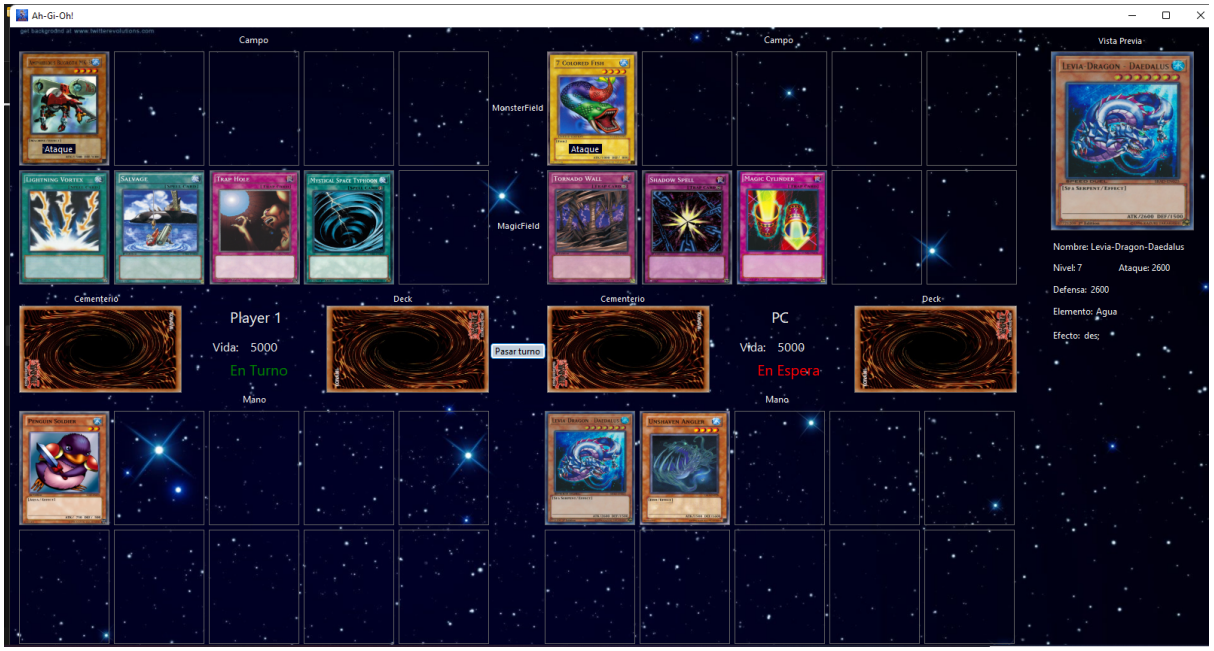
```
0 references
public class Compiler
{
    11 references
    private List<string> symbols;

    36 references
    private List<string> tokens;

    18 references
    private List<string> operators;

    25 references
    private List<string> conditionOperators;

    7 references
    private List<string> condicionToken;

    12 references
    private List<string> specialTokens;

    4 references
    private List<Card> cards;

    6 references
    private Player p1;

    6 references
```

Lógica del juego e Interfaz

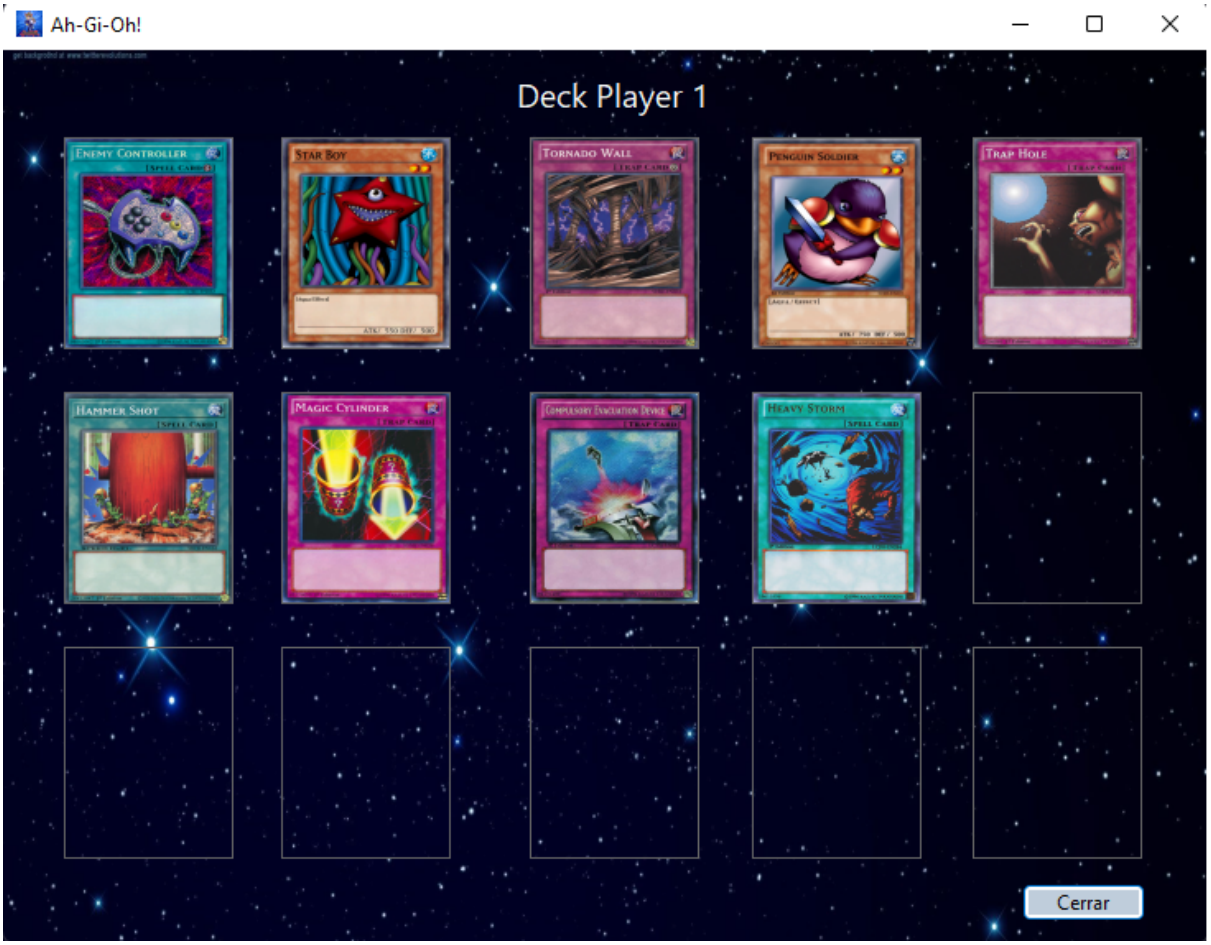
La interfaz, así como la lógica del juego se implementan usando el motor Windows Forms incluído en la IDE Visual Studio 2022.

Menú Principal:

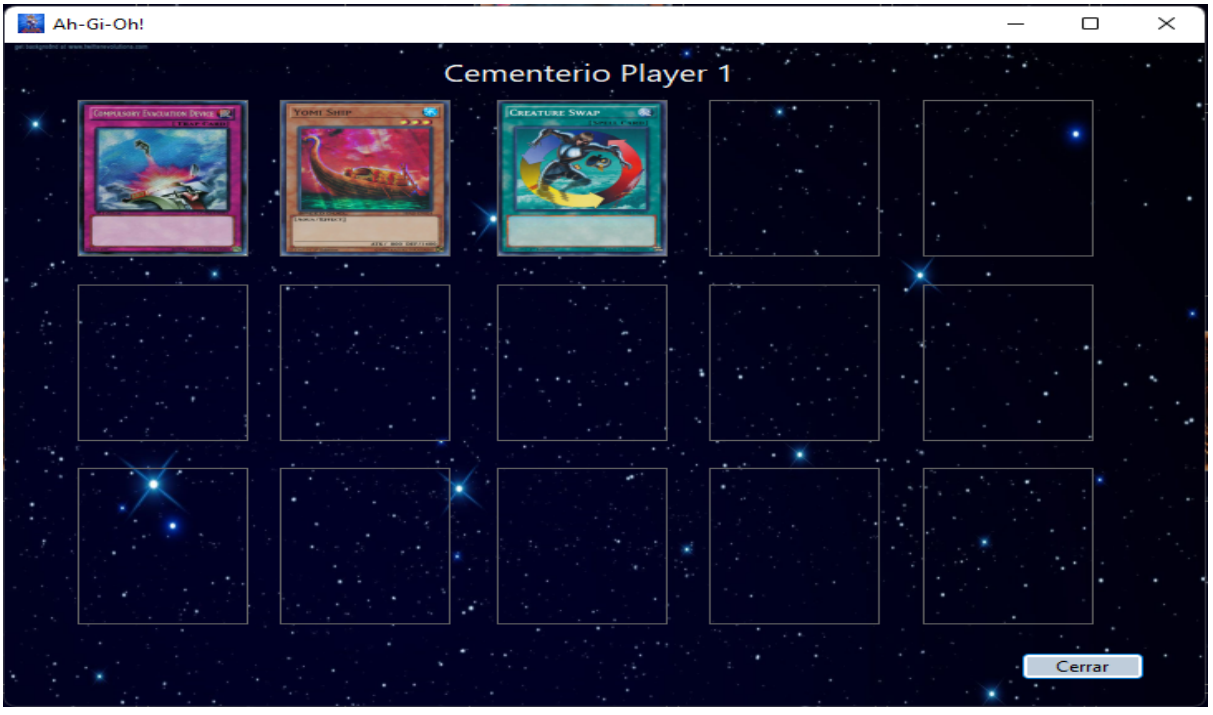
Duelo:



Visualizador de Deck:

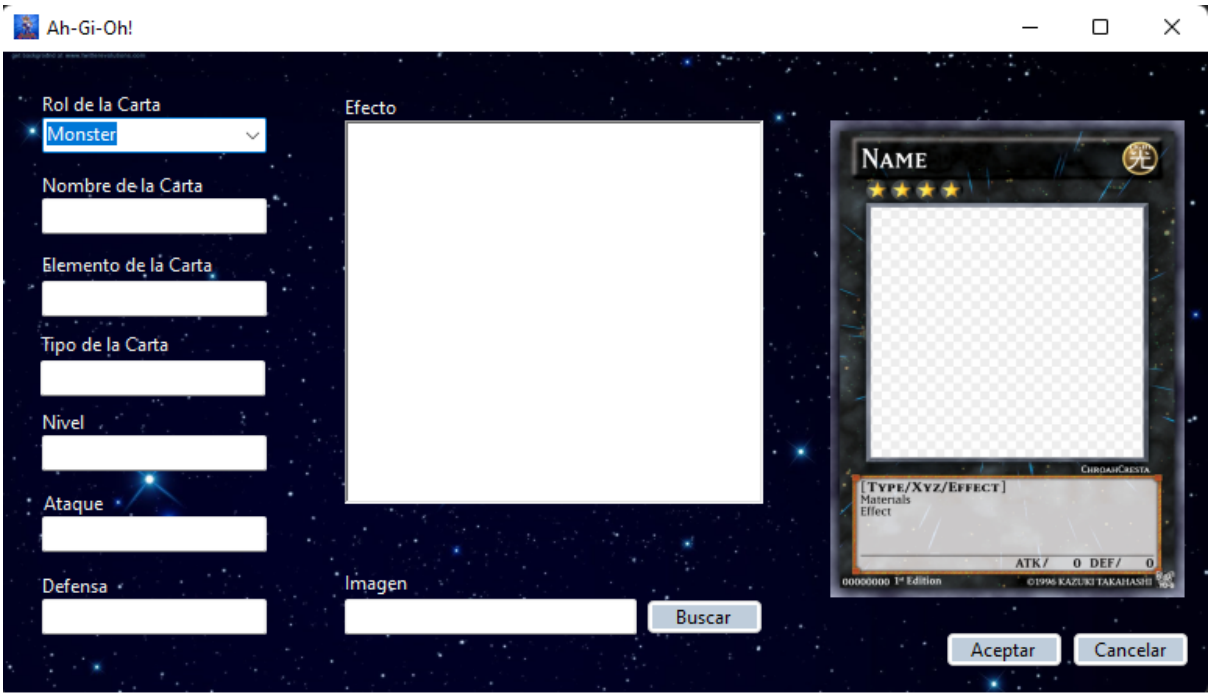


Visualizador de Cementerio:



Menú de Selección de Opción:

Menú de Creación de Cartas:



Requisitos:

- Sistema Operativo: Windows
- net SDK 6.0
- Windows Forms
- Que te corra el buscaminas🙂