Facultad de Ciencias

Diseño y desarrollo de un de juego de estrategia de cartas coleccionables (TCG) con inteligencia artificial.

Design and development of a collectible card strategy game (TCG) with artificial intelligence.

Trabajo de Fin de Grado
para acceder al

GRADO EN INGENIERÍA INFORMÁTICA

Autor: José Mª Herrerías Santos

Director: Inés Gonzalez Rodríguez

Febrero - 2018

Resumen

Este proyecto se divide en dos objetivos claramente diferenciables, el primero es el diseño y desarrollo de una plataforma digital que permita a dos usuarios jugar a un TCG (Trading Card Game), mientras que el segundo consiste en desarrollar un sistema experto que actúe como jugador no humano, pudiendo sustituir a uno o ambos usuarios en el juego.

El desarrollo del TCG se divide nuevamente en dos partes, la creación de un motor de juego que compone la lógica de la aplicación y el diseño de una interfaz que permita la interacción con el usuario.

Para el diseño del sistema experto definiremos una base de reglas basadas en el proceso de toma de decisiones de un jugador humano, que alimentarán a un motor de inferencia fundamentado en encadenamiento hacia delante, permitiendo a nuestra IA (Inteligencia Artificial) tomar parte en la aplicación como un jugador más.

Palabras Clave: Encadenamiento hacía delante, TCG, Sistema Experto

Abstract

This project is divided into two clearly differentiable objectives, the first one is the design and development of a digital platform that allows two users to play a TCG (Trading Card Game), while the second one is to develop an expert system that acts as a non-human player, capable of replacing one or both users in the game.

The development of the TCG is divided again into two parts, the creation of a game engine that makes up the logic of the application and the design of an interface that allows to interact with the user.

For the design of the expert system we will define a base of rules inspired in the decision-making process of a human player, which will feed an inference engine based on forward chaining, allowing our AI to take part in our application as another human player.

Palabras Clave: Forward Chaining, TCG, Expert System

Índice general

Índice de figuras

Índice de tablas

Capítulo 1
Introducción

A lo largo de este primer capítulo se proporciona al lector mediante tres sencillos apartados la idea general detrás de este proyecto.

Para ello se expone en primer lugar la motivación que ha impulsado este trabajo, seguidamente los objetivos que se esperan alcanzar y en último lugar procedimiento de trabajo que se ha aplicado.

1.1 Motivación

En el campo de la inteligencia artificial, un **sistema experto** es un sistema computacional que emula la capacidad de tomar decisiones de un humano experto [1].

Un sistema experto, como todos los sistemas basados en conocimiento, se compone fundamentalmente de dos subsistemas: una base de conocimiento tanto genérico como específico que nuestro sistema tiene de su entorno y un motor de inferencia capaz de evaluar el estado actual de la base de conocimiento y deducir nuevos hechos.

Por tanto, únicamente es necesario disponer de la base de conocimiento adecuada para un entorno concreto para crear un nuevo sistema experto, lo que nos proporciona un gran abanico de posibilidades en cuantos a los diferentes campos en los cuales pueden ser aplicables. En nuestro caso, los videojuegos nos presentan un espacio propicio a la aplicación de técnicas de inteligencia artificial. Esto se debe a que proporcionan entornos suficientemente ricos, pero a la vez totalmente controlables y fáciles de interpretar. Por esta razón el proyecto se enfoca al diseño y desarrollo de un sistema experto y la posterior aplicación de técnicas de IA, para permitir que emule a un jugador no humano en nuestro TCG.

Hemos elegido el género **Trading Card Game**, debido a que se basa fundamentalmente en la toma de decisiones de los dos jugadores implicados, lo que beneficia la implementación de un sistema experto**.** Con el objetivo de agilizar el proceso de implementación de la plataforma digital nos hemos basado en "Magic: The Gathering", un popular juego de este género. No obstante, este juego, así como la gran mayoría de TCG comerciales presentan una serie de características que de ser implementadas supondrían un aumento de la complejidad que excedería el ámbito de un TFG. Por tanto hemos optado por omitir las mecánicas más complejas y desarrollar un TCG que conserve la mayor parte de las características fundamentales de otros juegos comerciales que los hacer interesantes y proporcionan un entorno suficientemente rico para el desarrollo de un TFG.

1.2 Objetivos del proyecto

De manera más detallada los objetivos de este proyecto son los siguientes:

1. Diseñar e implementar un sistema de juego de cierto nivel de complejidad que permita emular de manera digital un TCG.
2. Modelar una interfaz de usuario atractiva e intuitiva.
3. Diseñar e implementar un sistema experto que sirva como jugador no humano en nuestro juego.

1.3 Procedimiento

Siguiendo los objetivos marcados anteriormente, hemos dividido el proyecto en diferentes etapas de investigación, diseño y desarrollo. En el caso de estas últimas, hemos utilizado una metodología de desarrollo iterativo incremental [2] organizado en varias iteraciones que se detallan en la [Sección 2.3].

A continuación, se muestra una visión general de la estructura del desarrollo del nuestro proyecto, detallando las fases realizadas:

1. Análisis de requisitos para el desarrollo de la aplicación**.**

   Se lleva a cabo un análisis en el que se obtiene información de la estructura fundamental de un Trading Card Game.

2. Diseño del software.

   En base a la información obtenida en la fase anterior se identifican las clases del diseño y se construye un modelo cercano a la programación orientada a objetos [Sección 2.4].

3. Elección del entorno de programación

   Se consideran las ventajas y desventajas de varias herramientas de programación [Sección 2.2].

4. Implementación de la aplicación**.**

   Se recogen las reglas del sistema de juego (Apéndice I) y los requisitos del software [Sección 2.3] y realizamos varias iteraciones de codificación del software.

5. Análisis y diseño de la estrategia de juego.

   Se lleva a cabo un análisis de la estrategia de juego de un jugador humano a lo largo de varias partidas. Descomponiendo la táctica en una serie de reglas que formaran la estrategia de nuestro jugador no humano (Sección 3.5).

6. Búsqueda y análisis de librerías de sistemas expertos.

   Analizamos las ventajas y desventajas expuestas en la sección [Sección 2.2.3] de las diferentes tecnologías para representar una base de conocimiento y realizar inferencias que podemos utilizar para construir nuestro sistema experto.

7. Implementación del sistema experto

   Se construye el sistema experto, descrito en la sección [Sección 2.4.4] en base a nuestro diseño de estrategia de juego y el motor de inferencia NRules.

Capítulo 2

Diseño y Desarrollo

En este capítulo se exponen los procesos relacionados con el análisis, diseño e implementación del software correspondiente a nuestro TCG propiamente dicho, así como los cambios producidos en las diferentes iteraciones del proceso de desarrollo.

2.1. Descripción general

Nuestro software debe implementar todos los elementos esenciales de nuestro TCG, omitiendo únicamente las mecánicas más complejas y permitiendo a dos jugadores humanos que conozcan las reglas (Apéndice I) jugar uno contra el otro. Adicionalmente más adelante añadiremos, como ya hemos comentado en los objetivos, la posibilidad de sustituir a uno o incluso a ambos jugadores por un sistema experto capaz de jugar, por tanto, nuestra aplicación debe permitir el acoplamiento de un jugador no humano inteligente.

Nuestro software tiene varios modos de ejecución: jugador contra jugador, jugador contra IA e IA contra IA. El primero de los modos es completamente funcional con independencia de nuestro sistema experto y permite a dos usuarios jugar uno contra el otro, además de permitirnos comprobar el correcto funcionamiento del motor de juego. El segundo consiste en la introducción de un jugador no humano, controlado por nuestro sistema experto, que actúa como oponente del usuario. Por último, en el modo IA contra IA, podemos ver a nuestro sistema experto jugar contra sí mismo, lo que resulta interesante para comprobar su rendimiento, ya que nos permite, por ejemplo, enfrentar a IAs más desarrolladas con otras más simples.

Además de desarrollar la lógica necesaria para nuestra aplicación, hemos querido proporcionar al jugador con una interfaz gráfica intuitiva y de aspecto profesional, por lo que hemos utilizado varias herramientas para mejorar la experiencia de usuario.

2.2. Tecnologías y herramientas

2.2.1. Análisis de alternativas

Ante el comienzo del desarrollo nos encontramos con la disyuntiva de elegir las herramientas a utilizar. Frente a este problema llevamos a cabo una investigación de diferentes opciones aplicables a nuestro caso.

La decisión más fundamental era escoger que plataforma de software utilizaríamos para desarrollar el propio videojuego. Las opciones consideradas fueron: utilizar el Entorno de Desarrollo Integrado (IDE) Eclipse realizando la programación en Java, usar el motor de juego Unity 3D realizando la programación en C# o utilizar el motor de juego Unreal Engine realizando la programación en C++.

Además, otra decisión importante fue la selección del motor de inferencia que utilizaríamos para implementar a nuestro jugador no humano. En esta selección consideramos como opciones: CLIPS y NRules. Esta decisión se ve fuertemente afectada por la anterior debido a las posibilidades de integración con la plataforma software elegida. En el caso de CLIPS, a pesar de estar implementado en C tiene soporte oficial tanto para Java (CLIPS JNI) como para C# (CLIPS .NET) y por lo tanto es compatible con todas las opciones, aunque al investigar descubrimos que CLIPS .NET, lleva mucho tiempo sin recibir actualizaciones mucho tiempo, por lo que tiene problemas de compatibilidad con software que utilice versiones modernas de C#, como Unity 3D. Por su parte NRules es un motor de inferencia de código abierto implementado en C# que únicamente resultaría compatible con Unity 3D.

De esta manera concluimos que nuestras opciones eran utilizar Eclipse y CLIPS, Unity 3D con NRules o Unreal Engine con CLIPS. Finalmente decidimos que Unity 3D y NRules eran la mejor opción, debido a que aunque NRules es menos fiable y potente que CLIPS utilizar un software especializado en desarrollo de videojuegos favorece a nuestro proyecto y Unity 3D es una opción mucho más sencilla y accesible que Unreal Engine.

2.2.2. Unity 3D

Para la programación del software correspondiente a la plataforma de juego utilizamos el Game Engine Unity 3D. Unity es un motor de videojuegos multiplataforma muy popular que nos presenta varias ventajas: es un software especializado en la creación de videojuegos por lo que cuenta con múltiples herramientas que facilitan este fin, permite exportar el programa resultante a diferentes plataformas, tanto móviles (Android, iOS), de escritorio (Windows, Linux, Mac) y de juego (Xbox One, PS4, PSVita), cuenta con una amplia comunidad y una documentación muy completa y su scripting se basa en Mono, una implementación de código abierto de .NET Framework, lo que nos otorga compatibilidad con una amplia cantidad de librerías y programas externos.

Como lenguaje de programación Unity nos da la opción de elegir entre JavaScript o C#, de entre los cuales escogeremos C#, debido a que en el ámbito de los videojuegos tiene una aceptación mucho mayor, lo que facilita la búsqueda de material de apoyo. C# es un lenguaje de programación orientado a objetos, concurrente y compilado[3] que deriva de C/C++ utilizado en el .NET Framework de Microsoft.

2.2.3. DOTween

Además de las herramientas propias de Unity, en el apartado gráfico hemos utilizado el motor de Tweening DOTween. Tweening es la abreviación del término inglés *in-betweening*, que se corresponde en castellano con intermediación o interpolación. Por tanto, un motor de tweening es un sistema software que nos permite crear animaciones a través de código haciendo uso de algoritmos interpolación de movimiento. El uso de una aplicación de este tipo nos permite crear una interfaz más profesional y agradable para el usuario de forma sencilla.

2.2.4. NRules

Como comentamos con anterioridad para el desarrollo de nuestro sistema experto decidimos utilizar NRules. NRules es un motor de producción de reglas de código abierto para .NET (C#) [4]. El cual se basa, como CLIPS y la gran mayoría de motores de inferencia, en el algoritmo Rete [Sección 3.3] y utiliza un DSL (Domain Specific Language) para la creación de reglas.

2.3. Especificación de requisitos

Según las necesidades de nuestra aplicación descritas anteriormente en la sección 2.1, y las reglas de nuestro TCG (Apéndice I), extraemos una serie de requisitos funcionales y no funcionales que enumeramos en esta sección:

2.3.1. Primera iteración

Durante la primera iteración del software nos limitamos a modelar el sistema de juego y toda su lógica con las reglas definidas (Apéndice I), e implementamos un sistema de simulación gráfica muy básico. Los requisitos funcionales de esta iteración se pueden encontrar en la tabla 2.1.

ID	DESCRIPCIÓN
RF001	El software implementará las reglas de juego especificadas en el Apéndice I.
RF002	Se debe mostrar el estado del juego mediante una simulación gráfica.
RF003	El usuario participará en el juego utilizando el ratón tal y como se especifica en el Apéndice II.
RF004	La aplicación se ejecutará por completo en modo de ejecución Jugador vs. Jugador.
RF005	La aplicación contará con un mecanismo de persistencia que permite almacenar la información sobre las cartas.

Tabla 2.1: Requisitos funcionales de la primera iteración

2.3.2. Segunda iteración

En la segunda iteración de desarrollo, añdimos funcionalidades de interfaz juego, se introducen algunas mejoras de eficiencia e implementamos sistemas para hacer frente a casos de uso inesperados. Los requisitos funcionales y no funcionales de esta iteración se pueden encontrar en las tablas 2.2. y 2.3 respectivamente.

ID	DESCRIPCIÓN
RF006	Se añade un menú principal donde el usuario puede seleccionar el modo de ejecución de la aplicación o salir de ésta.
RF007	Se añade un menú que permite al usuario seleccionar las cartas que utilizarán tanto él como su oponente durante la partida.
RF008	Se añade un menú que indica que jugador es el ganador al terminar una partida.
RF009	Se añade un cronómetro con un tiempo máximo por turno para prevenir la inactividad de un jugador.

Tabla 2.2: Requisitos funcionales de la segunda iteración

ID	DESCRIPCIÓN
RNF001	Se rediseña la interfaz para ser más atractiva al usuario.

Tabla 2.3: Requisitos no funcionales de la segunda iteración

2.3.3. Tercera iteración

Para la tercera iteración incorporamos nuestro sistema experto tal y como se detalla en la [Sección 2.4.4.] los requisitos funcionales de esta iteración se pueden encontrar en la tabla 2.4.

ID	DESCRIPCIÓN
RF010	Se implementa al menos un sistema experto.
RF011	La aplicación se adapta para permitir a un jugador no humano tomar parte en el juego.
RF012	Se permite iniciar una partida en cualquiera de sus modos de ejecución (Jugador vs. Jugador, jugador vs. IA e IA vs. IA.

Tabla 2.4: Requisitos funcionales de la tercera iteración

2.4. Diseño de la aplicación

En las siguientes subsecciones, explicamos el diseño de la aplicación especificada en los anteriores apartados del capítulo desde un nivel alto de abstracción hasta el detalle.

2.4.1. Diseño de alto nivel

Para la arquitectura de la aplicación empleamos una estructura modular por capas. De esta manera la aplicación queda dividida en tres capas: la capa de presentación, la capa lógica y la capa de persistencia. Además, en el caso de la capa lógica se subdivide en dos paquetes: el paquete de la lógica del TCG y el paquete del sistema experto.

En esta sección omitiremos la capa de persistencia, ya que está implementada por completo con herramientas propias de Unity. En la figura 2.1 se muestra la separación en capas.

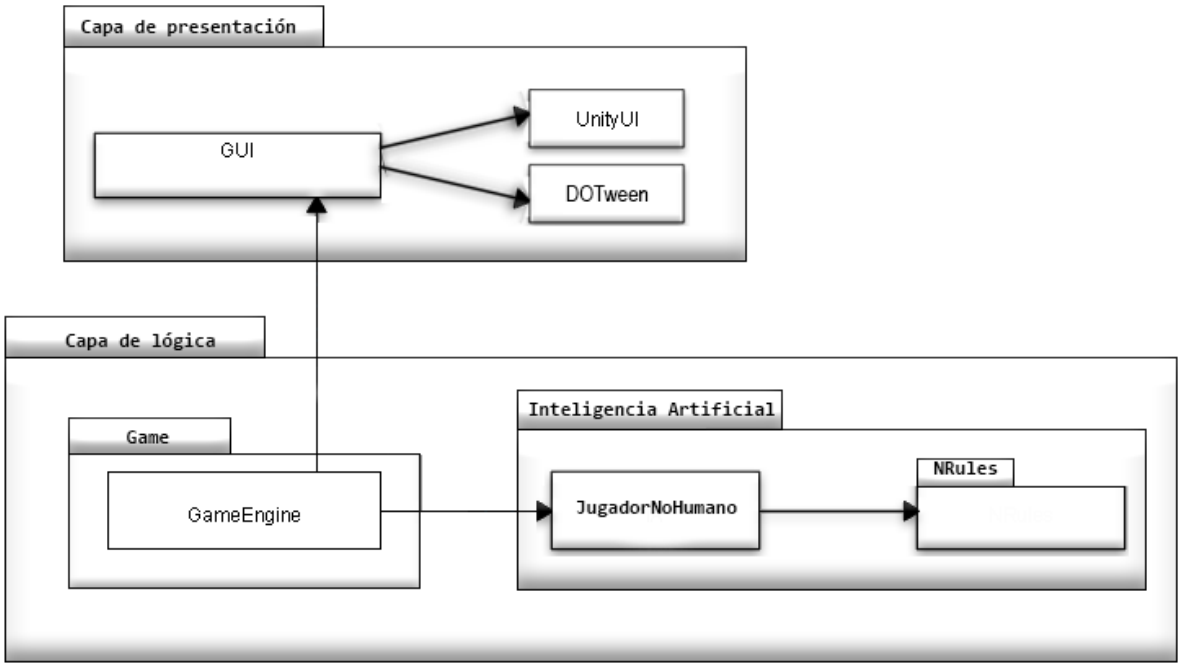

Figura 2.1: Diagrama de arquitectura en capas: paquetes principales

2.4.2. Motor de juego

En esta sección mostramos con mayor detalle el diseño del motor de juego, cuyos componentes están ubicados en el paquete Game. Estos componentes fueron desarrollados por completo en la primera iteración.

Los componentes básicos son comunes a cualquier TCG: AssetCarta (representación de una carta), Mano (representación de las cartas en mano), Mazo y Mesa (representación de las cartas sobre la mesa).

Además, nuestra clase Jugador representa a cada uno de los jugadores y por tanto cada jugador tendrá su propio mazo, mano y mesa. En la figura 2.2 se muestra la organización general del paquete.

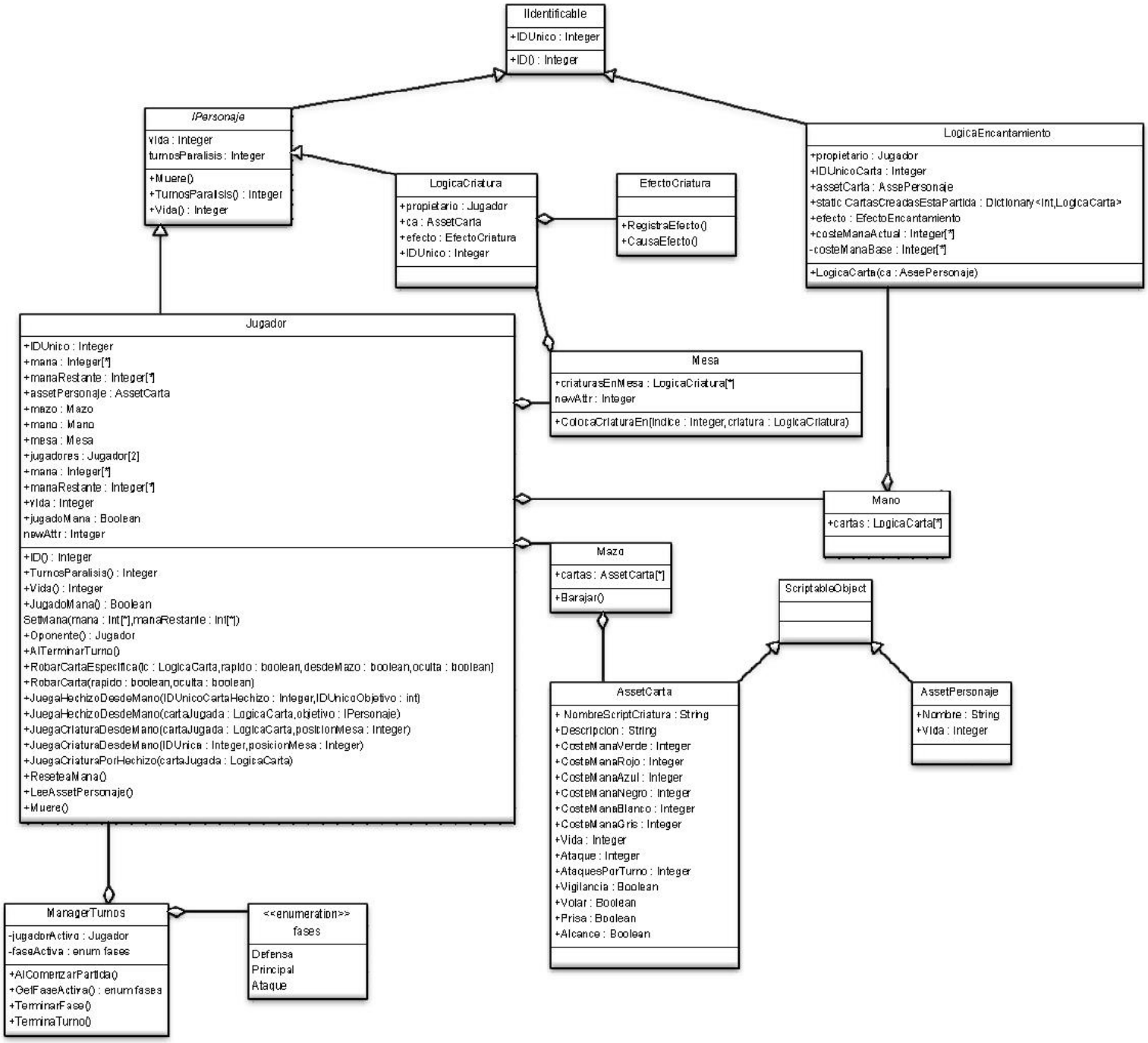

Figura 2.2: Diagrama de clases del motor de juego

2.4.3. Simulación gráfica

Para el diseño de la vista gráfica de la aplicación, realizado en su mayoría durante la segunda iteración del proyecto, se crean las clases expuestas en la figura 2.3. Tanto VisualMazo, VisualCriatura y VisualMano utilizan el paquete DOTween para crear animaciones a partir de scripting.

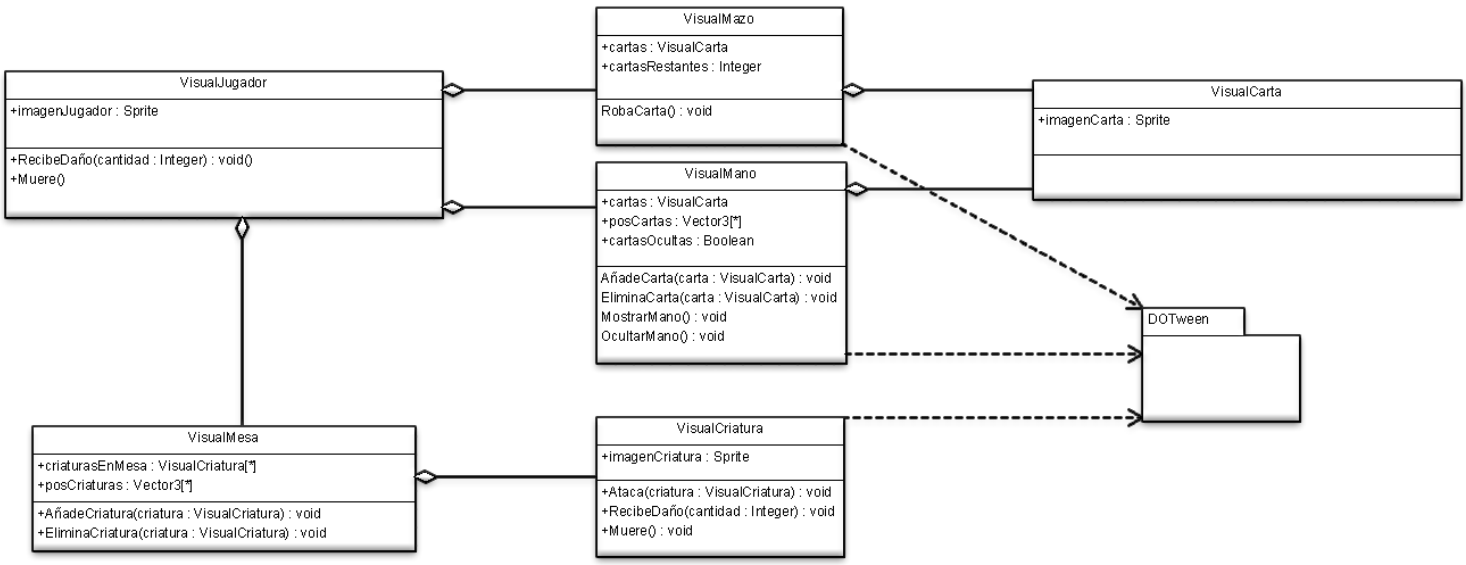

Figura 2.3: Diagrama de clases para la simulación gráfica

2.4.4. Jugador no humano

Finalmente, en la implementación del jugador no humano durante la última iteración, creamos la jerarquía que se muestra en la figura 2.4.

En nuestro diseño se implementa cada regla de nuestro sistema experto como una clase que hereda de la clase Rule del paquete NRules.

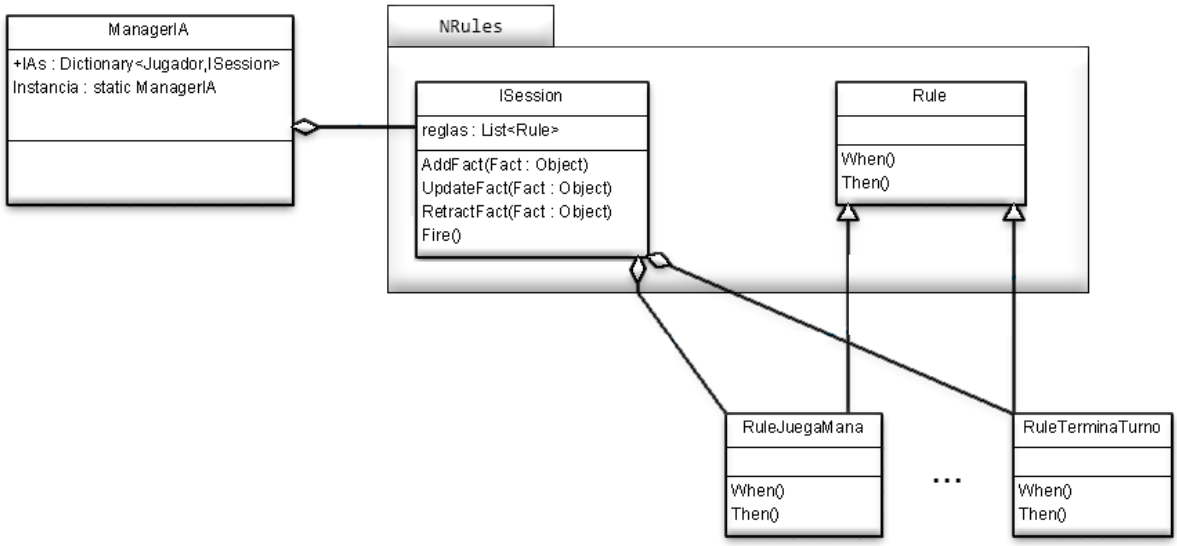

Figura 2.4: Diagrama de clases para el jugador no humano

2.5. Casos de uso

2.5.1. Identificación de los actores

Como tarea previa a la definición de los casos de uso, debemos identificar a los actores de nuestra aplicación. En nuestro caso, se identifica únicamente un tipo, que definimos como **jugador**.

2.5.2. Casos de uso

Nombre	Comenzar partida
Actor Principal	Jugador.
Descripción	El jugador puede comenzar una partida, en cualquiera de los modos disponibles, desde el menú principal de la aplicación.
Evento de Activación	El jugador escoge una de las opciones de comienzo de partida.
Precondición	
Garantías de éxito	La partida da comienzo.
Garantías mínimas	La aplicación se mantiene consistente.
Escenario principal	1. Se muestra un menú donde se seleccionan los mazos de cartas que usaran los jugadores. 2. El jugador escoge los mazos. 3. El jugador indica al sistema que quiere continuar. 4. El sistema carga la escena de la partida.
Extensiones	2a , 3a. El jugador pulsa el botón cancelar. 1. La aplicación vuelve al menú principal

Tabla 2.5 Caso de uso Comenzar partida

Nombre	Terminar fase actual.
Actor Principal	Jugador.
Descripción	El jugador puede poner fin manualmente a la fase actual de su turno para realizar las acciones propias de la siguiente fase.
Evento de Activación	El jugador escoge la opción “Terminar fase”
Precondición	Es el turno del jugador.
Garantías de éxito	Se cambia de fase.
Garantías mínimas	La aplicación se mantiene consistente.
Escenario principal	1. El sistema cambia el indicador de fase actual a la siguiente fase correspondiente 2. Se desbloquean las acciones propias de la siguiente fase.
Extensiones	1a. No quedan más fases del turno del jugador actual 1. Se termina el turno del jugador.

Tabla 2.6 Caso de uso Terminar fase actual.

Nombre	Juega una carta sin objetivo.
Actor Principal	Jugador.
Descripción	El jugador puede durante la fase principal de su turno, jugar cartas de criatura, maná o hechizos sin objetivo arrastrando la carta correspondiente al campo.
Evento de Activación	El jugador arrastra una carta de su mano al campo.
Precondición	La carta debe ser una criatura, maná o hechizo sin objetivo. El jugador debe encontrarse en la fase principal de su turno. El usuario debe tener suficiente maná como para pagar el coste de la carta.
Garantías de éxito	Se lleva a cabo el efecto de la carta.
Garantías mínimas	La aplicación se mantiene consistente.
Escenario principal	1. El usuario suelta la carta sobre el campo. 2. La aplicación muestra la animación de la carta entrando en juego. 3. Se ejecuta el efecto de la carta.
Extensiones	1a . El jugador suelta la carta sobre otro lugar que no sea su parte del campo 1. La carta vuelve a la mano del jugador

Tabla 2.7 Caso de uso Juega una carta sin objetivo.

Nombre	Juega una carta con objetivo.
Actor Principal	Jugador.
Descripción	El jugador puede durante la fase principal de su turno con objetivo, arrastrando el marcador del hechizo sobre el objetivo.
Evento de Activación	El jugador arrastra una carta de su mano al campo.
Precondición	La carta debe ser un hechizo con objetivo. El jugador debe encontrarse en la fase principal de su turno. El usuario debe tener suficiente maná como para pagar el coste de la carta.
Garantías de éxito	Se lleva a cabo el efecto de la carta.
Garantías mínimas	La aplicación se mantiene consistente.
Escenario principal	1. La aplicación muestra un marcador que indica cual es el objetivo. 2. El jugador arrastra el marcador sobre una criatura o personaje. 3. Se ejecuta el efecto de la carta.
Extensiones	2a . El jugador suelta el marcador sobre un objetivo no valido. 1.El hechizo no se ejecuta.

Tabla 2.8 Caso de uso Juega una carta con objetivo.

Nombre	Atacar.
Actor Principal	Jugador.
Descripción	El jugador puede utilizar una criatura que tenga ataques restantes para atacar a su oponente.
Evento de Activación	El jugador hace click en una criatura durante la fase de ataque.
Precondición	La criatura debe tener ataques restantes este turno. El jugador debe encontrarse en la fase de ataque de su turno.
Garantías de éxito	Se marca la criatura como atacando.
Garantías mínimas	La aplicación se mantiene consistente.
Escenario principal	1. Aparece una marca sobre la criatura que indica que está atacando.
Extensiones	1a . El jugador hace click de nuevo sobre la criatura. 1. Se cancela el ataque.

Tabla 2.9 Caso de uso Atacar.

Nombre	Defender.
Actor Principal	Jugador.
Descripción	El jugador puede utilizar una criatura que tenga ataques restantes para defenderse del ataque de una criatura oponente.
Evento de Activación	El jugador arrastra una de sus criaturas sobre un atacante oponente.
Precondición	La criatura debe tener ataques restantes este turno. El jugador debe encontrarse en la fase de defensa de su turno.
Garantías de éxito	Se produce un enfrentamiento entre criaturas.
Garantías mínimas	La aplicación se mantiene consistente.
Escenario principal	1. Se produce un enfrentamiento entre atacante y defensor. 2. Ambos reciben el daño correspondiente y pierden un ataque restante. 3. Si la vida de alguno es igual a cero, esa criatura muere.
Extensiones	1a . La criatura defensora no es válida para bloquear al atacante. 1. No se produce el encuentro

Tabla 2.10 Caso de uso Defender.

Nombre	Seleccionar maná neutral.
Actor Principal	Jugador.
Descripción	Cuando un jugador juega una carta que tiene un coste en maná neutral (gris), puede seleccionar con qué tipo de maná quiere pagarlo.
Evento de Activación	El jugador arrastra a su campo una carta con coste de maná neutral.
Precondición	El jugador debe encontrarse en la fase principal de su turno. El usuario debe tener suficiente maná como para pagar el coste de la carta. La carta jugada debe tener coste en maná gris.
Garantías de éxito	El jugador selecciona con que maná quiere pagar.
Garantías mínimas	La aplicación se mantiene consistente.
Escenario principal	1. Se abre un menú que permite seleccionar con que manás pagar, así como el pago restante. 2. El jugador selecciona manás hasta que el pago restante es cero. 3. Se pone en juego la carta y se cierra el menú.
Extensiones	2a . El turno termina antes de poder seleccionar todos los manás. 1. Se realiza el pago restante seleccionando de forma automática manás para cubrir el coste.

Tabla 2.11 Caso de uso Seleccionar maná neutral.

Capítulo 3

Jugador no humano

A continuación, describimos nuestro TCG como problema (desde la perspectiva del diseño de agentes inteligentes), comentamos la elección de sistema experto como jugador no humano, explicamos los diferentes métodos de trabajo con motores de inferencia, presentamos una breve descripción del algoritmo empleado por nuestro motor de inferencia y por último exponemos de forma sencilla las estrategias que hemos implementado.

3.1. Descripción del problema

En un TCG, se enfrentan dos oponentes, que tienen conocimiento sobre las cartas que forman parte de su mazo (aunque no el orden en el que se colocan), las cartas que conforman su mano, las criaturas colocadas en la mesa y el estado de ambos jugadores.

Por tanto, ambos jugadores deben tratar de jugar sus cartas actuando de acuerdo con una estrategia para derrotar a su oponente al mismo tiempo que frustran la de éste, a pesar de no conocerla.

De esta manera, el entorno de nuestro TCG, como todos los de su género, cuenta con una serie de características a tener en cuenta a la hora de diseñar un jugador no humano, a saber:

- **Parcialmente observable**: Se dice que un entorno de trabajo es parcialmente observable si los sensores no detectan todos los aspectos que son relevantes en la toma de decisiones y las medidas de rendimiento.

- **Entorno competitivo:** Si el siguiente estado del medio está totalmente determinado por el estado actual y la acción ejecutada por el agente, entonces el entorno es determinista; de otra forma es estocástico. Pero en el caso de que el medio sea determinista, excepto para las acciones de otros agentes, decimos que el entorno es competitivo.

- **Estático:** Cuando nos encontramos con un entorno que no cambia mientras el agente está deliberando decimos que es estático.

- **Discreto:** Dado que el número de posibles estados del juego es finito.

3.2. Sistema experto

Hemos decidido abordar el diseño e implementación del jugador no humano como el desarrollo de un agente basado en conocimiento, concretamente, un sistema experto donde la base de reglas codifica el conocimiento experto sobre cómo jugar. Como todo sistema basado en reglas nuestro agente está compuesto por:

- Una **base de conocimiento**, en nuestro caso se trata de una base de reglas (cláusulas de Horn con cabeza) que representa el conocimiento concreto sobre el estado del agente, es decir el estado del juego cuando el jugador no humano debe actuar.

- Un **motor de inferencia** que utilice un algoritmo de inferencia para la producción de reglas. En este proyecto se corresponden con NRules y el algoritmo Rete.

En nuestro caso hemos desarrollado una primera base de conocimientos sencilla, que se corresponde con una oponente de baja dificultad y una segunda que supone una ampliación de la primera y por tanto un oponente de mayor dificultad. Las bases de conocimiento que se han implementado para nuestro jugador no humano se especifican en la Sección 3.5 de forma superficial.

La implementación de estas reglas se ha realizado utilizando el DSL (Domain Specific Language) que nos facilita NRules y equivale a un pequeño y sencillo lenguaje de programación con una sintaxis única diseñado concretamente para esta tarea.

3.3. Modos de un motor de inferencia

Ya que hemos decidido que nuestro agente inteligente será un sistema experto y de acuerdo con la definición que dimos anteriormente [Sección 1.1], a continuación, presentamos una breve descripción de los diferentes modos de funcionamiento de un motor de inferencia.

En sistemas expertos basados en reglas nos encontramos con dos tipos de inferencia:

Forward Chaining o encadenamiento hacia delante, este sistema consiste en empezar el proceso desde algunas premisas en nuestra base de conocimiento, a partir de las cuales comenzamos un ciclo en el que se comprueban que reglas podemos disparar, se selecciona aquella de mayor prioridad y finalmente se ejecuta antes de volver a empezar.

Debemos tener en cuenta que la ejecución de cada una de estas reglas suele conllevar que nuestra base de conocimiento se vea alterada, lo que a su vez provoca que nuevas reglas se disparen.

Backward Chaining o encadenamiento hacia atrás, en este modo realizaríamos el proceso contrario. Partimos de un objetivo que intentamos alcanzar y realizamos el siguiente ciclo, comprobamos que reglas tienen como consecuencia nuestro objetivo, analizamos su antecedente, si este no se encuentra en la memoria recomenzamos el ciclo con el antecedente como nuevo objetivo. Finalmente, cuando el antecedente esté en memoria habremos probado cierto nuestro objetivo y por tanto tendremos una secuencia de acciones que realizar para alcanzarlo. Si por el contrario alcanzamos un punto en el que ninguna regla tenga nuestro objetivo como consecuencia, desechamos este camino, detenemos la recursión y buscamos otra forma de alcanzarlo.

En nuestro problema necesitamos extraer conclusiones desde un estado concreto de la partida, por tanto, lo adecuado es realizar una inferencia dirigida por los datos, es decir, un motor de inferencia con encadenamiento hacia delante.

3.4. Algoritmo Rete

Los modos de funcionamiento de un motor de inferencia, tal como se han explicado en el apartado anterior estaban originalmente implementados mediante un paradigma de búsqueda exhaustiva o fuerza bruta, es decir, comprobaríamos cada una de nuestras reglas contra los hechos en nuestra base de conocimiento. Desafortunadamente incluso para bases de reglas y hechos de un tamaño moderado este proceso es demasiado lento [5].

Es por esta razón que el motor de inferencia que hemos escogido implementa el algoritmo Rete, una implementación eficiente del encadenamiento hacia delante.

El algoritmo Rete, diseñado por Charles L. Forgy en 1974, basa su mejora de eficiencia en la idea de prevenir repetidas iteraciones sobre la memoria almacenando información entre ciclos. Esto se debe a que el paso que puede requerir iterar sobre la memoria es determinar si un patrón dado concuerda con un dato que forma parte de esta. Pero la iteración se puede evitar almacenando con cada patrón una lista de los elementos con los que coincide, que se actualizara cuando haya cambios en la memoria.

Con esta información se construye los que llamaremos una red Rete, que utilizaremos como filtro para realizar el mínimo número de comprobaciones sobre los hechos posibles.

3.5. Estrategia implementada

En este subapartado comentaremos de forma superficial las estrategias que se han implementado para el jugador no humano de nuestro TCG.

La estrategia implementada, responde al termino de rush (carrera), es una táctica habitual en los trading card games, que consiste en jugar de manera muy agresiva, intentando derrotar al rival en los primeros turnos de la partida, antes de que tenga tiempo para reaccionar. Podemos resumir la estrategia en:

- Intentar tener siempre más criaturas que el contrario
- Hacer siempre el máximo daño posible al oponente.
- Utilizar hechizos de daño para terminar con el rival en los últimos turnos, cuando sea más complicado hacer daño.

A pesar de que no se especifica, cada una de las reglas que conforman las estrategias tiene una prioridad de ejecución que resulta decisiva cuando varias las condiciones de varias reglas se cumplen a la vez. En el caso de que dos reglas de la misma prioridad puedan ejecutarse a la vez el motor de inferencia escoge una de forma aleatoria, esto añade un factor de impredecibilidad y realismo a la toma de decisiones.

3.5.1. Estrategia de rush simple

Antes de implementar la estrategia completa, decidimos realizar una implementación simplificada, para comprobar el correcto funcionamiento del motor de inferencia y que finalmente paso a formar parte del proyecto como oponente de dificultad fácil.

Esta estrategia responde a las siguientes reglas:

- Si tenemos un numero de criaturas en mesa menor que el máximo permitido, podemos jugar criaturas.
- Si estamos en fase principal de nuestro turno y podemos pagar una carta, entonces podemos jugarla.
- Si tenemos una carta de maná y podemos jugarla, la jugamos.
- Si tenemos una criatura en mano y podemos jugarla, la jugamos.
- Si tenemos un hechizo de daño y podemos jugarlo, lo lanzamos directamente contra el oponente.
- Si podemos jugar criaturas, tenemos un hechizo de invocación en mano y podemos jugarlo, lo lanzamos.
- Si tenemos una criatura que puede atacar, ataca.

3.5.2. Estrategia de rush ampliada

Finalmente, y utilizando las reglas de la estrategia fácil como base implementamos una extensión del plan de juego, permitiendo a la IA cambiar de comportamiento en función del rumbo de la partida y realizar poderosos combos.

Para esta implementación se añadiendo a la anterior las siguientes reglas:

- Si un jugador tiene 6 puntos de vida más que su oponente, entonces ese jugador tiene ventaja en puntos de vida.
- Si un jugador tiene más criaturas en campo que su oponente, entonces tiene ventaja numérica.
- Si un jugador tiene 8 puntos de vida o menos, entonces está cerca de la derrota.
- Si un jugador tiene al menos 4 criaturas en mesa, entonces tiene muchas criaturas.
- Si una criatura tiene características 1/1 y no tiene efecto, entonces es sacrificable.
- Si un jugador tiene una criatura en campo, que puede atacar, con un efecto que provoca que sus características aumenten al jugar hechizos, entonces el jugador prioriza lanza hechizos.
- Si un jugador tiene una criatura en campo, que puede atacar, con un efecto que provoca que sus características aumenten al jugar criaturas, entonces el jugador prioriza jugar criaturas y hechizos de invocación.
- Si un jugador tiene ventaja numérica, entonces prioriza lanzar hechizos de daño contra el oponente.
- Si un jugador tiene desventaja numérica, entonces lanza hechizos de daño contra criaturas oponentes que no pueden sobrevivir a ellos.

- Si un jugador tiene desventaja numérica, entonces prioriza el lanzamiento de hechizos de invocación.
- Si un jugador tiene desventaja en puntos de vida, entonces utiliza criaturas sacrificables para defender.
- Si el oponente está cercano a la derrota, entonces prioriza el lanzamiento de hechizos de daño contra él.

Capítulo 4

Pruebas y experimentación

En este capítulo se describen las pruebas que se han ejecutado sobre el software para verificar que el sistema funciona correctamente de acuerdo con los requisitos expuestos en la fase de análisis y las pruebas que hemos llevado a cabo para comparar las dos implementaciones de estrategias de juego.

4.1. Pruebas

Las pruebas verifican el funcionamiento de piezas de software que pueden ser probadas de forma separada. Las pruebas unitarias se han llevado a cabo a lo largo de la fase de implementación, para comprobar su correcto funcionamiento. Para hacer estas pruebas hemos utilizado las herramientas que nos brindan Unity y Visual Studio, ya que nos permite poner puntos de ruptura en el código, pausar la ejecución de la aplicación de forma manual en cualquier momento y monitorizar en tiempo real el valor de todas las variables del sistema para asegurar la correcta ejecución de los métodos.

4.2. Experimentación

Como hemos mencionado con anterioridad, durante el desarrollo del jugador no humano hemos implementado dos bases de reglas, correspondientes a una estrategia básica y su posterior ampliación, especificadas en la Sección 3.5. Por tanto, nos ha parecido interesante realizar una comparación de ambas estrategias, para ello hemos utilizado el modo “IA vs. IA” que se incluye en nuestra aplicación y hemos simulado una serie de partidas entre los diferentes jugadores no humanos.

A continuación, se muestran los resultados obtenidos:

ID	Ganador	Turnos
1	Estrategia ampliada	5
2	Estrategia ampliada	6
3	Estrategia ampliada	7
4	Estrategia básica	7
5	Estrategia ampliada	6
6	Estrategia ampliada	7
7	Estrategia ampliada	5
8	Estrategia básica	8
9	Estrategia básica	7
10	Estrategia ampliada	7
11	Estrategia ampliada	7
12	Estrategia ampliada	6
13	Estrategia ampliada	8
14	Estrategia ampliada	7
15	Estrategia ampliada	7

Tabla 4.1 Resultado de la experimentación

Debido al planteamiento de las estrategias elegidas consiste en jugar de forma agresiva para intentar conseguir la victoria de forma rápida tal como se explican en la Sección 3.5, hemos considerado que los parámetros más apropiados para compararlas son número de victorias y el número de turnos dura la partida. Según los datos obtenidos en la experimentación observamos que:

- Al enfrentar a la estrategia básica con la más compleja esta última gana en un 80% de las ocasiones.
- La media de turnos de duración de la partida para las victorias de la estrategia básica es 7,33 turnos.
- La media de turnos de duración de la partida para las victorias de la estrategia ampliada es 6,5 turnos.

A la luz de los datos obtenidos, podemos concluir que nuestra estrategia ampliada supone una mejora con respecto a su versión más simple ya que no solo es capaz de vencerla en un 80% de las ocasiones, sino que además lo hace un 11,32% más rápido. No obstante, destacamos que debido a la naturaleza del juego hay una cierta influencia del azar en los resultados, debido a que el reparto de cartas para cada jugador es impredecible.

Capítulo 5
Conclusiones

De acuerdo con los objetivos definido para este proyecto [Sección 1.2], hemos implementado una plataforma digital que emula un sistema de juego basado en nuestra versión de “Magic: The Gathering” permitiendo a dos jugadores participar en una partida, hemos aplicado nuestros conocimientos sobre los fundamentos de los sistemas expertos y los motores de inferencia adquiridos durante los estudios de grado, hemos diseñado e implementado una estrategia basada en reglas que permite a un jugador no humano participar en el juego y por último hemos introducido un sistema experto en la aplicación capaz de ejecutar nuestra estrategia.

La aplicación de una estrategia de encadenamiento hacia delante al ámbito de los videojuegos supone una aportación original al mundo de la IA en los videojuegos puesto que no se trata de una técnica popular en este terreno. Esto ha supuesto una dificultad añadida debido a una menor disponibilidad de fuentes de información y herramientas especializadas para esta tarea.

Por otra parte, también hemos realizado la construcción de la propia plataforma de juego. Tarea cuya complejidad residía en lograr una adaptación digital fiel de un sistema de juego tan rico y complejo como es “Magic: The Gathering” o en nuestro caso nuestra versión de este TCG.

No obstante, considero importante resaltar la excelente capacidad que presenta el marco de los videojuegos para el estudio y evaluación de las técnicas de inteligencia artificial. Nuestro TCG nos proporcionó un entorno en el que se puede apreciar de forma visual el comportamiento de los agentes inteligentes, así como comprobar de forma sencilla el buen o mal rendimiento de una estrategia de juego concreta.

Referencias

[1] Peter Jackson, (1998), *Introduction to Expert Systems*, Addison-Wesley.

[2] Basili Larman, (2003), *Iterative and incremental development: A brief history*. *Computer*, vol. 6, pp. 47-56.

[3] Microsoft Corporation, (2013), *C# Language Specification 5.0*

[4] NRules, *Getting Started Guide*, [sitio web]. [Consulta: 11 febrero 2018].

[5] Charles L. Forgy, *Rete: A Fast Algorithm for the Many Pattern/Many Object Pattern Match Problem*, pp 17-37

Apéndice I: Reglamento Básico TCG

Objetivo

Cada jugador comienza la partida con 20 vidas, y 60 cartas en su mazo (biblioteca), el objetivo es derrotar al oponente reduciendo su vida a 0.

Colores de las cartas

Existen cartas de 5 colores diferentes, que corresponden a 5 tipos de maná diferente.

Figura Apéndice 1.1: Representación de los manás y colores

Generar maná

El maná es de uno de los cincos colores de Magic o es incoloro. Cuando un coste requiera maná de color, verás símbolos de maná de color (Ej.: ✸ Para maná blanco o ● para maná rojo) y cuando puedas usar cualquier clase de maná para pagar el coste, verás un símbolo con un número en él (Ej.: 2).

Las tierras se juegan para agregar un maná de ese color a tu reserva de maná. (Tu reserva de maná es donde guardas el maná hasta que se gasta.) Algunas criaturas, artefactos y hechizos también producen maná. Dicen algo así como "Agrega ● a tu reserva de maná"

Girar cartas

Las cartas son giradas por diferentes razones, las criaturas entrar en juego giradas debido al “mareo de invocación” una criatura se gira tras atacar y adicionalmente se pueden girar cartas por efecto de un hechizo.

Las cartas giradas no pueden realizar ninguna otra acción en ese turno, al comienzo del siguiente volverán a su posición original.

Estructura de las cartas

Figura Apéndice 1.2: Esquema de estructura de las cartas

Tipos de carta

Tierra

Aunque las tierras son permanentes, no se juegan como hechizos. Para jugar una tierra, simplemente ponla en juego. Esto ocurre inmediatamente, así que ningún jugador puede jugar nada en respuesta. Puedes jugar una tierra sólo durante una de tus fases principales mientras la pila esté vacía. No puedes jugar más de una tierra por turno. Usarás las tierras para producir el maná que necesitas para pagar tus hechizos y habilidades.

Conjuro

Un conjuro puede ser jugado sólo durante una fase principal de uno de tus propios turnos. No puedes jugarlo mientras otro hechizo esté en la pila. (Fases y pila se explican más adelante.) Un conjuro realiza su efecto (Sigues las instrucciones en la carta) y luego lo colocas en tu cementerio.

Encantamiento

Un encantamiento es un conjuro permanente. Esto significa dos cosas: Puedes jugar uno en cualquier momento en que pudieras jugar un conjuro y, después de jugarlo, lo colocas en la mesa frente a ti, cerca de tus tierras.

Algunos encantamientos se anexan a otra carta y le afectan mientras está en juego. Cuando la carta deja el juego, el aura se pone en el cementerio de su propietario.

Artefacto

Es como un encantamiento, con la diferencia de que si se anexionan a otra carta y esta es destruida el artefacto se conserva, para ser anexionado de nuevo. Además, los artefactos son incoloros, así que puedes jugar uno sin importar qué tipo de tierra controles

Criatura

Las criaturas luchan por ti. Son permanentes, pero al contrario de otro tipo de permanentes, las criaturas pueden atacar y bloquear. Todas las criaturas tienen valores de fuerza y resistencia. La fuerza de una criatura (el primer número) es cuánto daño hace en combate. Su resistencia (el segundo número) es cuánto daño debe recibir en un único turno para ser destruida. Las criaturas atacan y bloquean durante la fase de combate. Puedes bloquear con una criatura o jugar sus otras habilidades sin importar cuánto haya estado en juego.

Tablero

Figura Apéndice 1.3: Esquema del tablero de juego

Biblioteca

Tu biblioteca es tu montón para robar. La biblioteca se mantiene boca abajo y las cartas se quedan en el orden en que estaban al principio del juego. Nadie puede mirar las cartas que hay en tu biblioteca, pero puedes saber cuántas cartas hay en la biblioteca de cada jugador. Cada jugador tiene su propia biblioteca.

Mano

Cuando robas cartas, van a tu mano. Sólo tú puedes ver las cartas de tu mano. Cada jugador tiene su propia mano. Al final de cada turno si tienes más de siete cartas en la mano, elige y descarta hasta que te queden sólo siete cartas.

En juego

Comienzas el juego sin nada en juego. En cada uno de tus turnos, puedes jugar una tierra de tu mano y tantas criaturas, artefactos y encantamientos como tu maná permita. Tu oponente debe poder verlos todos y saber cuándo están girados. Esta zona es compartida por ambos jugadores.

Cementerio

Tu cementerio es tu montón de descarte. Tus hechizos de instantáneo y de conjuro se van a tu cementerio cuando se resuelven. Tus cartas van a tu cementerio cuando se descartan, se destruyen, se sacrifican, son contrarrestadas o llegan ahí por un efecto. Además, tus criaturas van a tu cementerio si reciben daño en un mismo turno igual o mayor que su resistencia o si su resistencia es reducida a 0 o menos. Las cartas de tu cementerio siempre estarán boca arriba, no pueden realizar ninguna acción y cualquiera debe poder verlas. Cada jugador tiene su propio cementerio.

La pila

Los hechizos y habilidades existen en la pila. Esperan allí para resolverse hasta que ambos jugadores eligen no jugar nuevos hechizos o habilidades. Luego el último hechizo o habilidad puesto en la pila se resuelve, y los jugadores tienen una nueva oportunidad de jugar hechizos y habilidades. (Aprenderás más sobre cómo jugar hechizos y habilidades en la próxima sección.) Esta zona es compartida por ambos jugadores.

Habilidades

Habilidades estáticas

Una habilidad estática se aplica constantemente mientras la carta está en juego. Por ejemplo, Furia primitiva es un encantamiento con la habilidad “Una criatura de tu control obtiene +1,+1”. Simplemente hace lo que dice.

Las habilidades estáticas más recurrentes son:

Volar: Una criatura que vuela no puede ser bloqueada excepto por criaturas que tengan la habilidad de volar o alcance.

Alcance: Las criaturas con alcance pueden bloquear a criaturas con volar.

Arrollar: Si una criatura que controles fuera a hacer suficiente daño de combate a sus bloqueadores como para destruirlos, el resto del daño que no haya sido bloqueado se hace al jugador defensor.

Daña primero: Si una criatura A con daña primero ataca o se defiende de otra criatura B, que no tenga la habilidad daña primero, y se da el caso de que el ataque de A es mayor que la resistencia de B, B es enviada al cementerio y A no recibe daño.

Defensor: Las criaturas con la habilidad de defensor no pueden atacar

Prisa: Una criatura con la habilidad prisa no entra al juego girada, es decir, puede atacar tan pronto como entra bajo tu control.

Velo: Un permanente con la habilidad de velo no puede ser objetivo de hechizos o habilidades; ni siquiera las tuyas. Los jugadores también pueden ganar la habilidad de velo.

Vigilancia: Una criatura con vigilancia puede bloquear, aunque esté girada.

Vínculo vital: Siempre que un permanente con vínculo vital haga daño, su controlador gana esa misma cantidad de vida.

Encantar: Aparece en los encantamientos, quiere decir que el encantamiento en cuestión se anexiona con otra carta, transfiriéndole cierta capacidad.

Equipar: El equivalente a encantar para artefactos. Te dice cuánto cuesta anexar el equipo a una de tus criaturas. Puedes jugar esta habilidad sólo durante tu fase principal, cuando no haya hechizos o habilidades en la pila.

Habilidades disparadas

Una habilidad disparada es un texto que sucede cuando ocurre un evento específico en el juego. Se disparan automáticamente siempre que suceda el evento. No puedes elegir retrasar o ignorar una habilidad disparada. Sin embargo, si la habilidad tiene un objetivo, no sucede nada si no puedes elegir un objetivo legal.

Atacar y bloquear

La principal manera de ganar el juego es atacar con tus criaturas. Si una criatura atacante no es bloqueada, hace daño igual a su fuerza a tu oponente.

En tu fase de combate (Las fases se explicarán en la sección siguiente), eliges qué criaturas atacarán y luego las giras.

Esas criaturas atacan al mismo tiempo. Todas atacan a tu oponente, no a sus criaturas. Puedes atacar con una criatura sólo si está enderezada y sólo si estaba en juego bajo tu control al comienzo de tu turno. Tu oponente elige cuáles de sus criaturas bloquearán. Las criaturas giradas no pueden ser declaradas como bloqueadoras.

Para bloquear no importa cuánto estuvo la criatura en juego. Cada criatura puede bloquear sólo a un atacante. Las criaturas no se giran para bloquear. Una vez que se eligieron las bloqueadoras, se asigna el daño de combate. Todas las criaturas, tanto atacantes como bloqueadoras, hacen daño igual a su fuerza.

- Las criaturas atacantes que no fueron bloqueadas hacen daño a tu oponente.

- Las criaturas atacantes que fueron bloqueadas hacen daño a las criaturas bloqueadoras. Si una de tus criaturas atacantes es bloqueada por varias criaturas, decides cómo dividir el daño de combate entre ellas.

- Las criaturas bloqueadoras hacen daño a las atacantes que bloquean.

- Si haces daño a tu oponente, ¡pierde esa cantidad de vida! Si una criatura recibe daño igual o mayor a su resistencia durante un mismo turno, esa criatura es destruida y va al cementerio de su propietario. Si una criatura recibe daño que no es letal, permanece en juego, pero el daño no desaparece hasta que termine el turno.

Fases de un turno

1. FASE DEFENSA

a. En caso de que el oponente nos haya atacado comenzaremos el turno en esta fase.

b. Seleccionamos cuales de nuestras criaturas bloquearan.

2. FASE INICIAL

a. Paso de enderezar Enderezas todos tus permanentes girados.

b. Paso de mantenimiento Esta parte del turno se menciona en algunas cartas. Si algo debe suceder una vez por turno, justo al comienzo, la habilidad se disparará en esta fase. Los jugadores pueden jugar instantáneos y habilidades activadas.

c. Paso de robar Roba una carta de tu biblioteca.

3. PRIMERA FASE PRINCIPAL

Puedes jugar cualquier cantidad de conjuros, instantáneos, criaturas, artefactos, encantamientos y habilidades activadas. Además, puedes jugar una tierra durante esta fase. Tu oponente puede jugar instantáneos y habilidades activadas.

4. FASE DE COMBATE

a. Paso de inicio del combate Los jugadores pueden jugar instantáneos y habilidades activadas.

b. Paso de declarar atacantes Tú decides cuáles, o si ninguna, de tus criaturas atacantes atacará. Esto gira las criaturas atacantes. Los jugadores pueden jugar instantáneos y habilidades activadas.

c. Paso de declarar bloqueadores Tu oponente decide cuáles, o si ninguna, de sus criaturas enderezadas bloqueará a tus criaturas atacantes. Los jugadores pueden jugar instantáneos y habilidades activadas.

d. Paso de daño de combate Cada criatura asigna su daño de combate al jugador defensor (si está atacando y no bloqueada), a la criatura o criaturas que la bloquean o a la criatura que está bloqueando. Los jugadores pueden luego jugar instantáneos y habilidades activadas. Una vez que se hayan resuelto, se hará el daño de combate.

5. FASE FINAL

a. Paso de final del turno Las habilidades que se disparan "al final del turno" se ejecutan. Los jugadores pueden jugar instantáneos y habilidades activadas.

b. Paso de limpieza Si tienes más de siete cartas en la mano, elige y descarta hasta que te queden sólo siete cartas. A continuación, todo el daño que se haya hecho a las criaturas y los efectos de "hasta el final del turno" desaparecen. Nadie puede jugar hechizos o habilidades a menos que una habilidad se dispare durante este paso.

Apéndice II: Guía de usuario

A pesar de que el proyecto se ha realizado pretendiendo que la aplicación resultante sea sencilla de utilizar e intuitiva para cualquier usuario, en este apéndice detallamos el uso de la misma.

Menú Principal

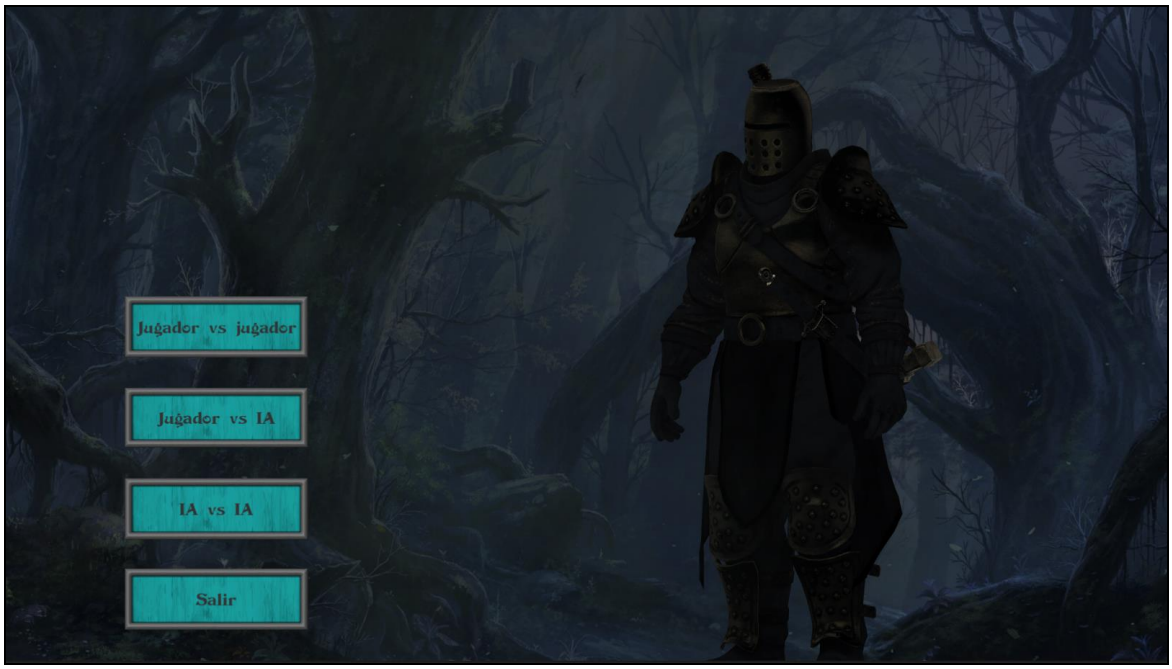

Figura Apéndice 2.1: Interfaz del menú principal

Este será el menú que recibirá al jugador al abrir la aplicación. Su función es permitirle al usuario seleccionar el modo de ejecución de la partida o abandonar la aplicación, para ello únicamente deberá hacer click en uno de los siguientes botones:

- **Jugador vs. jugador**
- **Jugador vs. IA**
- **IA vs. IA**
- **Salir**

Menú de selección de mazo

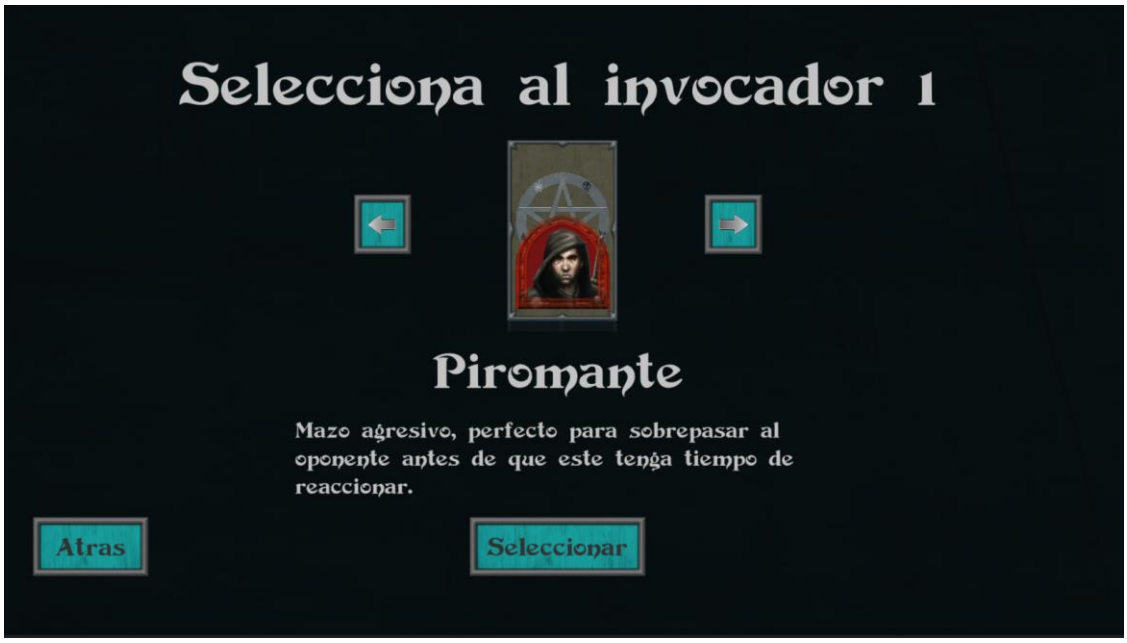

Figura Apéndice 2.2: Interfaz del menú de selección de mazo

Una vez que el usuario haya seleccionado el modo de ejecución de la partida se le presentará con este menú. En él, el usuario debe seleccionar que mazos de cartas utilizaran los jugadores durante la partida. Para realizar esta tarea el usuario cuenta con los siguientes elementos de interfaz:

- **Flechas de selección:** Permiten navegar adelante y atrás entre los mazos disponibles.
- **Botón de seleccionar:** Selecciona el mazo que se muestra en pantalla para el jugador que corresponda. Si es el jugador 1, se repetirá el proceso para el jugador 2. En caso de que fuera el mazo del jugador 2 la partida da comienzo.
- **Botón de atrás:** Este botón permite volver al menú principal en caso de que no se hubiese seleccionado ningún mazo o cancelar la primera selección en caso de que se hubiese hecho

Control durante la partida

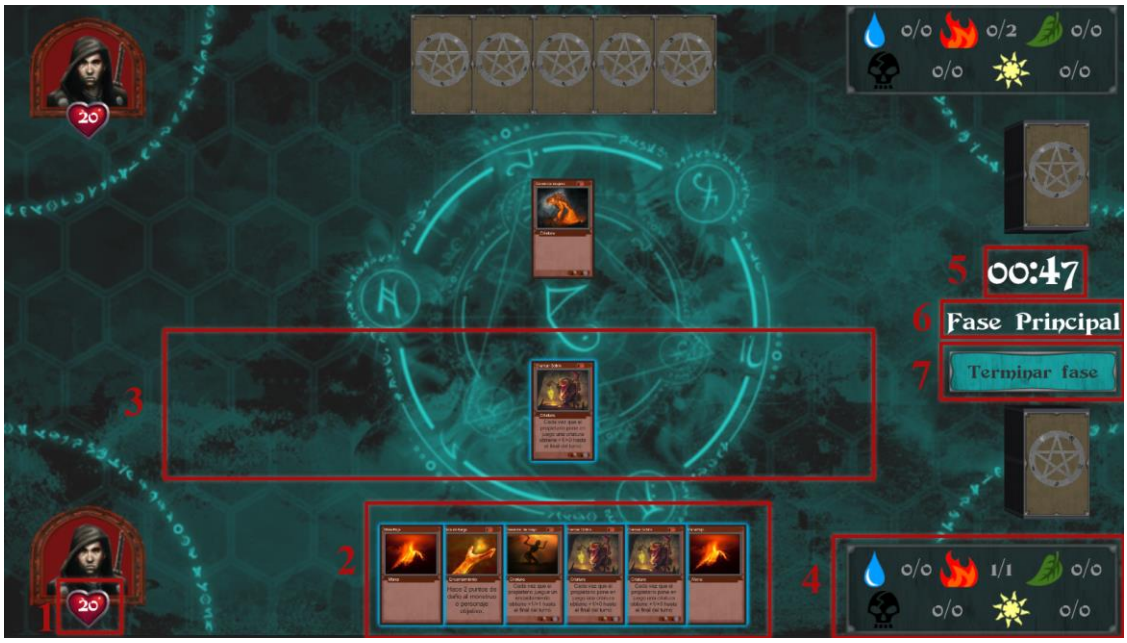

Figura Apéndice 2.3: Interfaz del menú de selección de mazo

Una vez la partida de comienzo, transcurrirá de acuerdo con las reglas especificadas en el Apéndice I. Para interactuar con el juego disponemos de una serie de elementos:

1. **Indicador de vida:** Muestra la salud restante del jugador.
2. **Mano:** Muestra las cartas que el jugador tiene en su mano. Se resaltan con un aura azul aquellas cuyos requisitos para ser jugadas se cumplen. Para jugar una carta únicamente se debe arrastrar desde la mano a la mesa, o a su objetivo en caso de ser un hechizo (se explica más adelante).
3. **Mesa:** Cada jugador tiene su lado de la mesa, donde se muestran las criaturas que posee. Se resaltan en azul aquellas que tienen acciones restantes.
4. **Indicador de maná:** Muestra el maná total y resaltante del jugador.
5. **Tiempo restante:** Indicador del tiempo que queda hasta que el turno del jugador actual acabe automáticamente.

6. **Fase actual:** Indicador que muestra en qué fase del juego nos encontramos.
7. **Botón de fin de fase:** Permite al usuario dar por terminada una fase de su turno y comenzar la siguiente.

Jugar hechizos con objetivo

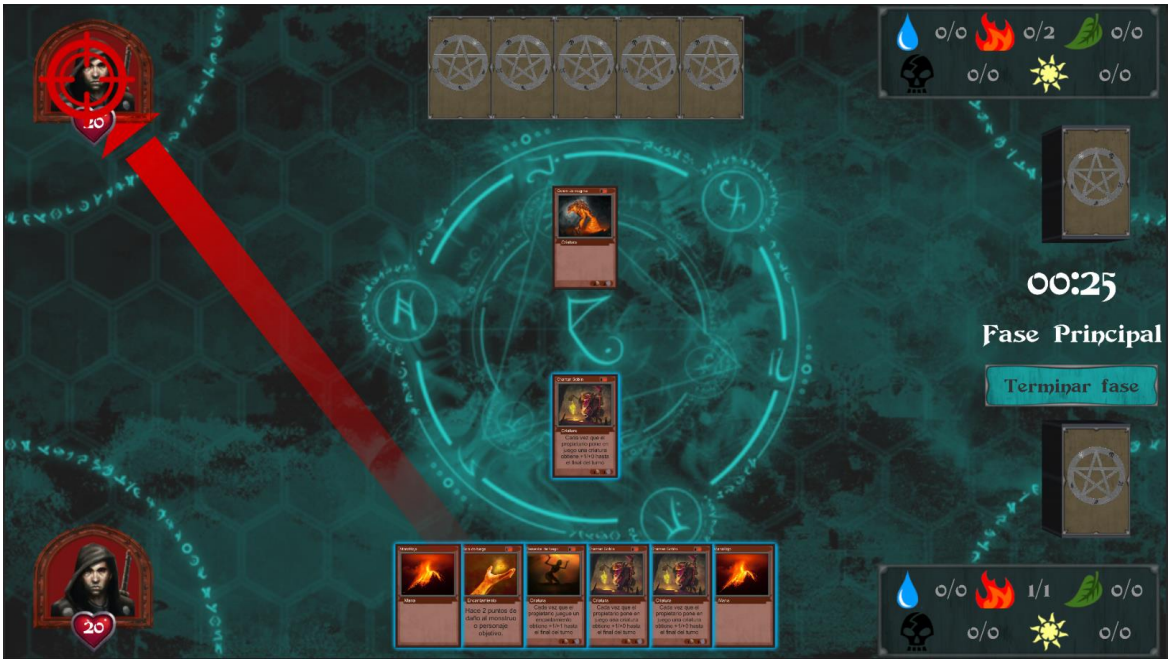

Figura Apéndice 2.4: Lanzamiento de un hechizo con objetivo

En el caso de la mayoría de las cartas basta con arrastrarlas a la mesa para jugarlas, pero otras, como en el caso de los hechizos de daño, requieren un objetivo.

Para lanzarlas arrastraremos la carta y en ese momento aparecerá una retícula roja, arrastraremos esta retícula y la soltaremos sobre un objetivo, si es válido el hechizo se jugará.

Vista detallada de una carta

Figura Apéndice 2.5: Lanzamiento de un hechizo con objetivo

Si mantenemos el puntero sobre cualquier carta visible de la mesa se nos mostrará una ampliación de la misma, para que podamos leer sus efectos y ver sus estadísticas fácilmente.

Atacar

En nuestra fase de ataque, podemos hacer click en una criatura que aun tenga acciones restantes para hacer que ataque al oponente. Para señalizar que una criatura atacará al oponente, esta carta se adelantará hacia el oponente y se rodeará de un aura roja.

Figura Apéndice 2.6: Atacar con una criatura

Defender

Cuando nuestro oponente nos ataque comenzaremos nuestro turno en fase de defensa. En esta fase podremos usar nuestras criaturas para defendernos de ataque.

Para ellos arrastramos nuestras criaturas, haciendo que aparezca una retícula (idéntica al caso expuesto para lanzar hechizos con objetivo) y soltamos el click sobre la criatura que queramos bloquear.

Menú de fin de partida

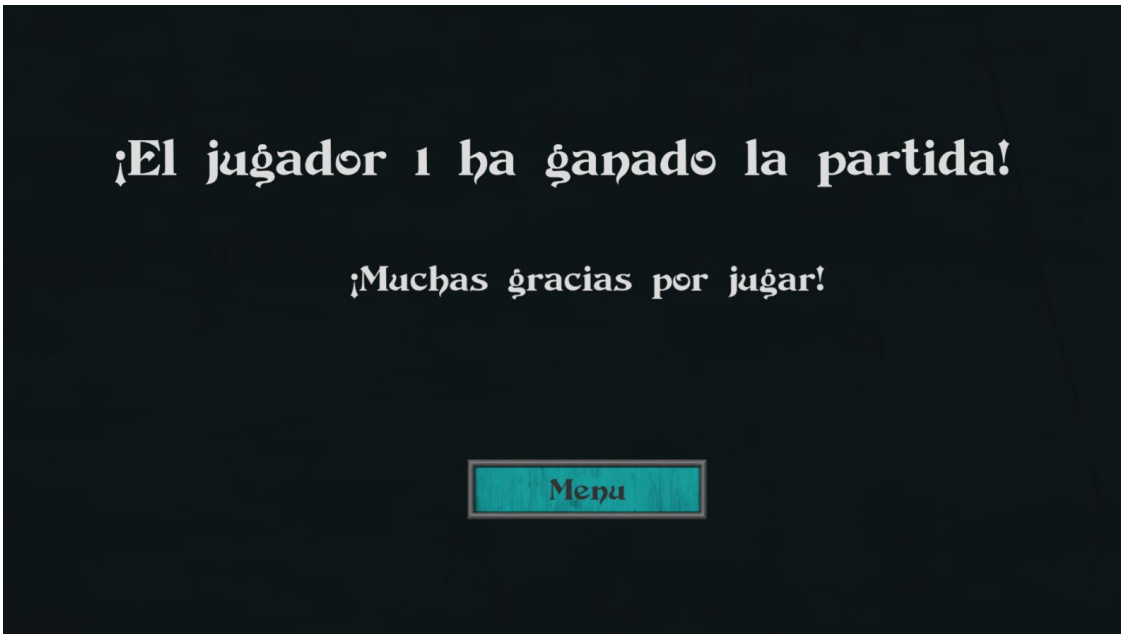

Figura Apéndice 2.7: Menú de fin de partida

Una vez que la salud de uno de los jugadores sea reducida a cero la partida habrá terminado y se mostrará al usuario esta última pantalla. En ella solo hay una acción posible, hacer clic en el botón menú para volver al menú principal.