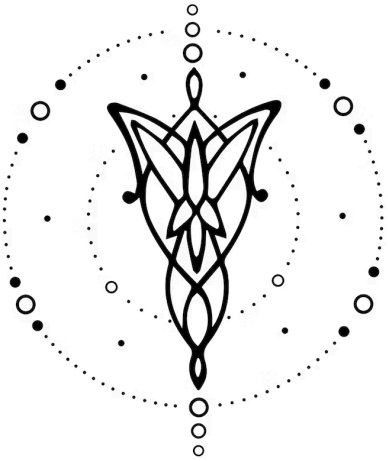

Especificación

Diseño e Implementación de un Lenguaje

v1.0.0

1. Equipo

Nombre	Apellido	Legajo	E-mail
Ian Franco	Tognetti	61215	itognetti@itba.edu.ar
Tomas	Freire	62027	tfreire@itba.edu.ar
Ramiro	Garcia	61133	rgarcia@itba.edu.ar
Lautaro	Farias	60505	lfarias@itba.edu.ar

2. Repositorio

La solución y su documentación serán versionadas en: **CardGames**

3. Dominio

Desarrollar un lenguaje que permita crear un videojuego de cartas. Se debe permitir crear un personaje y especificar los tipos y valores numéricos de las cartas con las que se jugará. Se ofrecerá la posibilidad de inventar las reglas acerca de qué cartas son mejores y la posibilidad de describir un uso específico para cada una o para un grupo de ellas. Finalmente se permitirá jugar contra la computadora.

Para reducir la complejidad del proyecto, se utilizarán bibliotecas externas para la creación de la interfaz visual. Se hará uso de un enfoque de programación orientada a objetos, dado que los componentes del juego pueden tratarse como tales y que dicho paradigma es intuitivo y fácil para el usuario. Además, se proveerán funciones básicas para la creación de reglas y usos de cada carta.

La implementación satisfactoria de este lenguaje permitirá programar, ejecutar y visualizar los juegos. Las partidas serán frente a la computadora.

4. Construcciones

El lenguaje desarrollado debería ofrecer las siguientes construcciones, prestaciones y funcionalidades:

(I). Se podrá crear un *juego* con su nombre, cantidad de cartas del mazo, cantidad de cartas en mano y cantidad de rondas para ganar.
(II). Se podrán crear una o varias *cartas* con su tipo y su número.
(III). Se podrán crear reglas para cartas con usos especiales.
(IV). Las variables podrán ser de tipo *type*, *value* y *game*.
(V). Se proveerán operadores relacionales como <, >, =, ≠, ≤ y ≥.
(VI). Se proveerán operaciones aritméticas básicas como +, -, * y /.
(VII). Se proveerán operaciones lógicas básicas como AND, OR y NOT.
(VIII). Se proveerán estructuras de control básicas de tipo IF-THEN-ELSE, FOREACH.
(IX). Se podrá crear el juego con todos los objetos creados anteriormente.
(X). Se podrá jugar al juego de cartas contra una computadora.

5. Casos de Prueba

Se proponen los siguientes casos iniciales de prueba de **`aceptación`**:

(I). Un programa que genere un juego de cartas contra una computadora.
(II). Un programa que permita programar mazos de cartas genéricos.
(III). Un programa que permita programar cartas con usos específicos.
(IV). Un programa que genere un juego de cartas como el “Club Penguin Cards”.

(V). Un programa que genere un juego con cartas determinando las condiciones de victoria del juego.
(VI). Un programa que permita generar un diseño de cartas estándar.
(VII). Un programa que permita personalizar cartas agregando arte, color y grosor de marco, descripción y disposición de los atributos.
(VIII). Un programa que permita personalizar la interfaz visual modificando el fondo de tablero.

Además, los siguientes casos de prueba de **rechazo**:

(I). Un programa malformado.
(II). Un programa que permita jugar sin determinar un mazo y sus condiciones de juego.
(III). Un programa que permita jugar sin determinar el diseño de las cartas.
(IV). Un programa que intente operar entre tipos de datos incompatibles.

6. Ejemplos

Creación de un jugador, un mazo con su cantidad de cartas y sus reglas y finalmente el juego con su condición de victoria y cantidad de jugadores:

```
//Se crea un mazo con información importante
game DeckOfHeroes has:
	NumbersOnDeck(13) //Cantidad de cartas por tipo
	CardsByPlayer(3) //Cantidad de cartas que tiene el jugador en mano
	RoundsToWin(3) //Cantidad de rondas para poder ganar una partida
	Design() //Diseño estándar de cartas

//Crea una regla para los tipos Water, Air y Fire
rule0 for types Air, Fire, Water:
	Air > Fire > Water
	Water > Air

//Crea una regla general para las cartas de tipo Aire
rule1 for type Air:
	if(value > 7) {
		shuffle()
	}

//Crea una regla para todas las cartas con valor 4
rule2 for value 4:
	wins()

//Utilizamos la función winsOver(values) dada por nuestro lenguaje para declarar
//cuando una carta explota
rule3 for values 1, 5, 8:
	winsOver(2, 6, 9)
//Crea una regla para todas las cartas con valor 9
```

```
//Esta regla utiliza la función lose dada por nuestro lenguaje para declarar
//cuando se pierde una partida (sin tener en cuenta la cantidad de rondas)
rule4 for value 9:
       lose()

//Reglas para una carta específica
rule5 for type Fire:
       if(value == 1) {
              wins();

//Equivalente a la anterior, forma distinta de acceder
rule6 for value 1:
       if(type == Fire) {
              wins();

//Este tipo de reglas es utilizado para poder definir comparaciones entre
//cartas
rule7 for type Game:
      if(CompareType(User, Opponent))
              greaterValueWins();

//Esta regla muestra una forma de acceder a varias cartas
rule8 for type Water:
       if(value == 2 || value == 4 ||...){ //Numeros pares
              winsOver(Air)
       }
       if(value == 1 || value == 3 ||...){ //Numero impares
              winsOver(Fire)
       }

//Esta regla muestra una forma de acceder a varias cartas usando iterador
//foreach
// Ambas reglas son equivalentes
rule9 for type Water:
      foreach(value){
         if(value % 2 == 0){ //Numeros pares
              winsOver(Air)
         }
      }
      foreach(value){
          if(value % 2 == 1){ //Numero impares
              winsOver(Fire)
          }
      }
```