FIUBA – 75.07

Algoritmos y programación 3

Trabajo práctico Nº 2:

FonTruco

2015 - 2º cuatrimestre

(Trabajo grupal)

Alumnos:

Nombres y Apellidos	Padrón	Mail
Cozza, Fabrizio Luis	97.402	fabrizio.cozza@gmail.com

Fecha de entrega final: Miércoles 2/12/2015 - Jueves 3/12/2015

Tutor:

Nota Final:

Entrega Nº: **1**

//Supuestos

Modelo de dominio

Programa realizado con IDE Eclipse.

Aquellas notaciones aparecidas con negrita en lineas generales representan clases dentro del modelo planteado.

Lo primero que se hizo fue poner en práctica el pensamiento respecto a un modelo, desarrollado de manera de documentar en diagramas UML, tanto de clases como de secuencia, para prever como debería representarse la solución al problema presentado, sin considerar a grandes rasgos problemas detallistas que podrían surgir durante el desarrollo del programa, por medio de una reunión grupal de todos los integrantes.

Se optó en una estructura basada tal que la jerarquía de crecimiento de juego de truco varíe desde una clase **Partido** */*(subdividida en los casos posibles 1vs1, 2vs2, 3vs3, no se sabe aún si se trabajará con clases individuales o por parámetro saber la cantidad de jugadores)*/*, que sería el nivel más "alto" encargado del manejo del partido genérico en desarrollo, luego continua con una **Ronda**, con sus variantes de **RondaRedonda** o el caso **RondaPicaPica** para el caso de 3vs3 posible, que trabaja con las rondas que trascurren durante el partido, y finalmente una clase **Mano** donde los jugadores bajarían sus cartas y elegirían la producción de una **Acción** (ya sea cantar Truco, Envido, etc.). En cuanto a los jugadores, se optó por la posible creación de **JugadorHumano** o **JugadorVirtual**, referenciando en el primer caso a personas, y en el segundo a una maquina con inteligencia artificial mínima. Estos a su misma vez pueden trabajar individualmente o colaborar en un juego dentro de un Equipo.

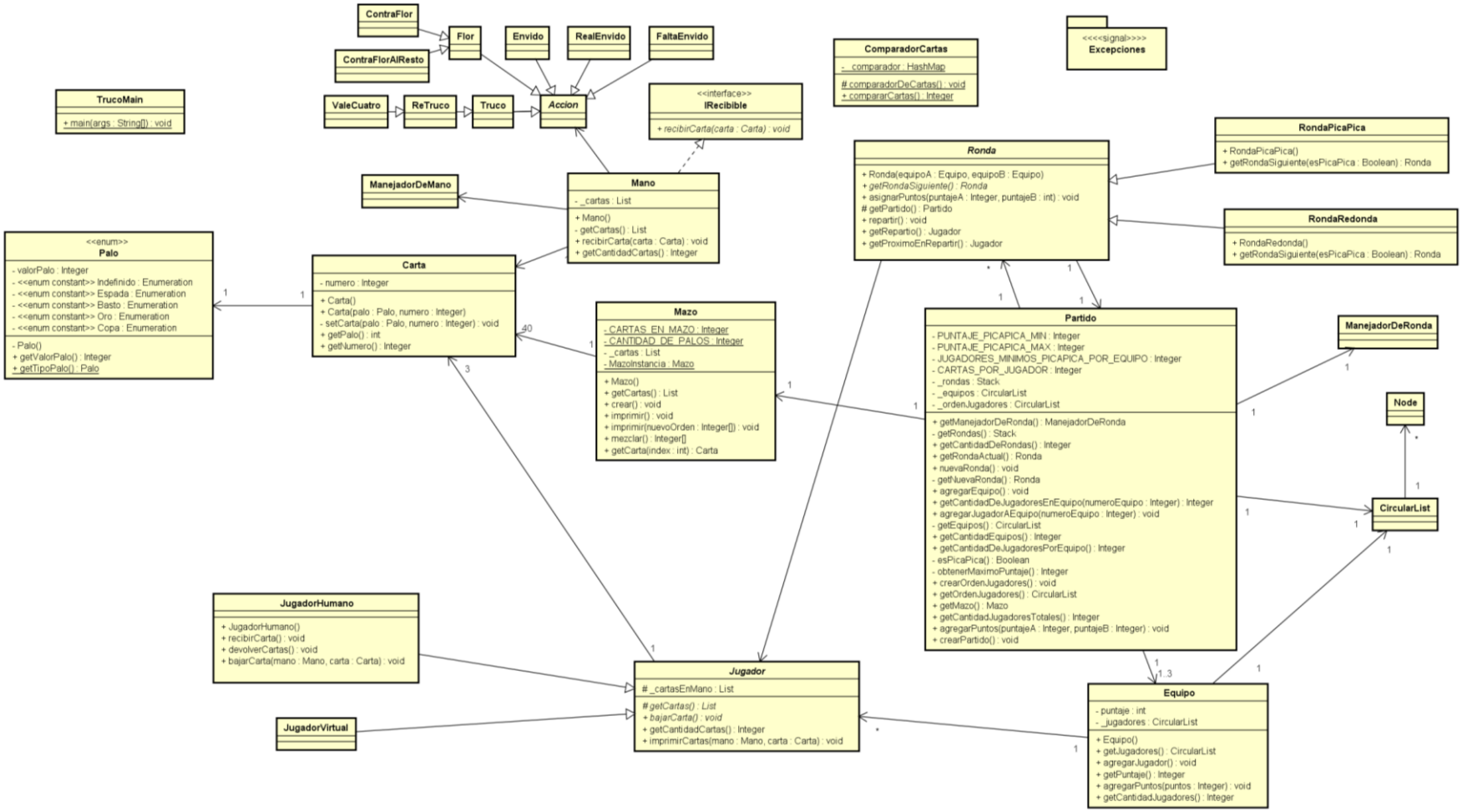
Saliendo fuera del incremento del juego, existen otras clases principales como **Carta** y **Mazo**, vitales en el software armado, ya que sin ellos, no podría jugarse la partida. La entidad **Carta** tiene la posibilidad de crearse con un valor y un tipo de **Palo**, para denotar su importancia en el truco según las reglas establecidas. Todas las cartas inicializadas por "default" en un **Mazo** conocido de 40 cartas (españolas si se quiere), son, valga la redundancia, creadas por una clase **Mazo**, teniendo este la posibilidad de repartir cartas y mezclarse cuando se necesite principalmente.

Otras clases importantes de mencionar serán aquellas pertenecientes a **Acción** (ya mencionada), que funcionaran como Decorator (en lo tenido en mente desde un principio), para cantar las posibles variantes para ganar puntos que el juego implica, como también será importante el **ComparadorCartas** que sabe cómo funcionan las reglas de juego y que carta le gana a cual otra, como también *(a desarrollar)* debería haber uno encargado de comparar por ejemplo, el Envido.

Aquellas otras clases que pueden aparecer en general son reguladores de juego, como los **Manejadores**, que le indican, según corresponda, como deben comportarse o ser algunos objetos (por ejemplo si una **Ronda** corresponde que sea redonda o picapica). Como también la aparición de una **CircularList** para la implementación de la **Ronda**.

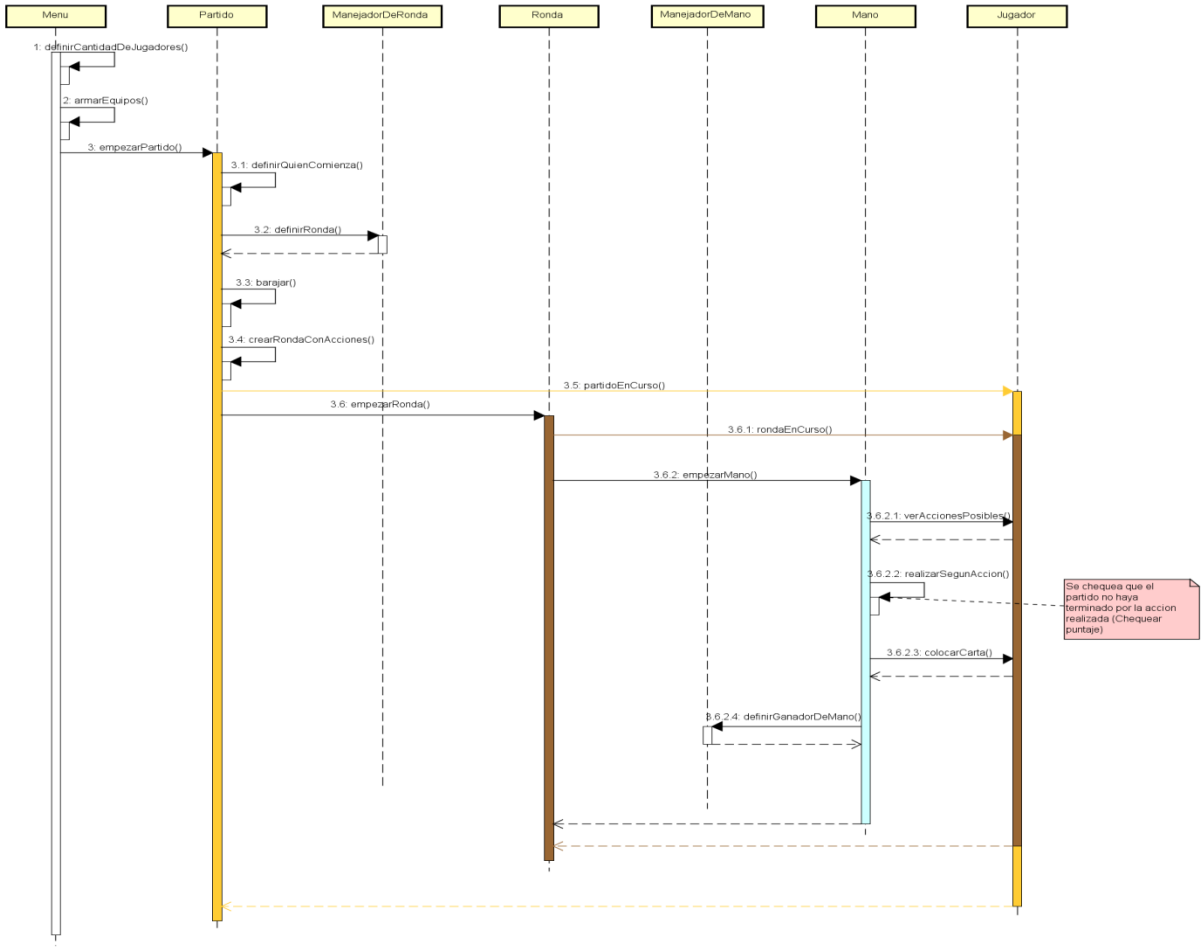
Diagrama de clases

Diagrama de clases general (en desarrollo)



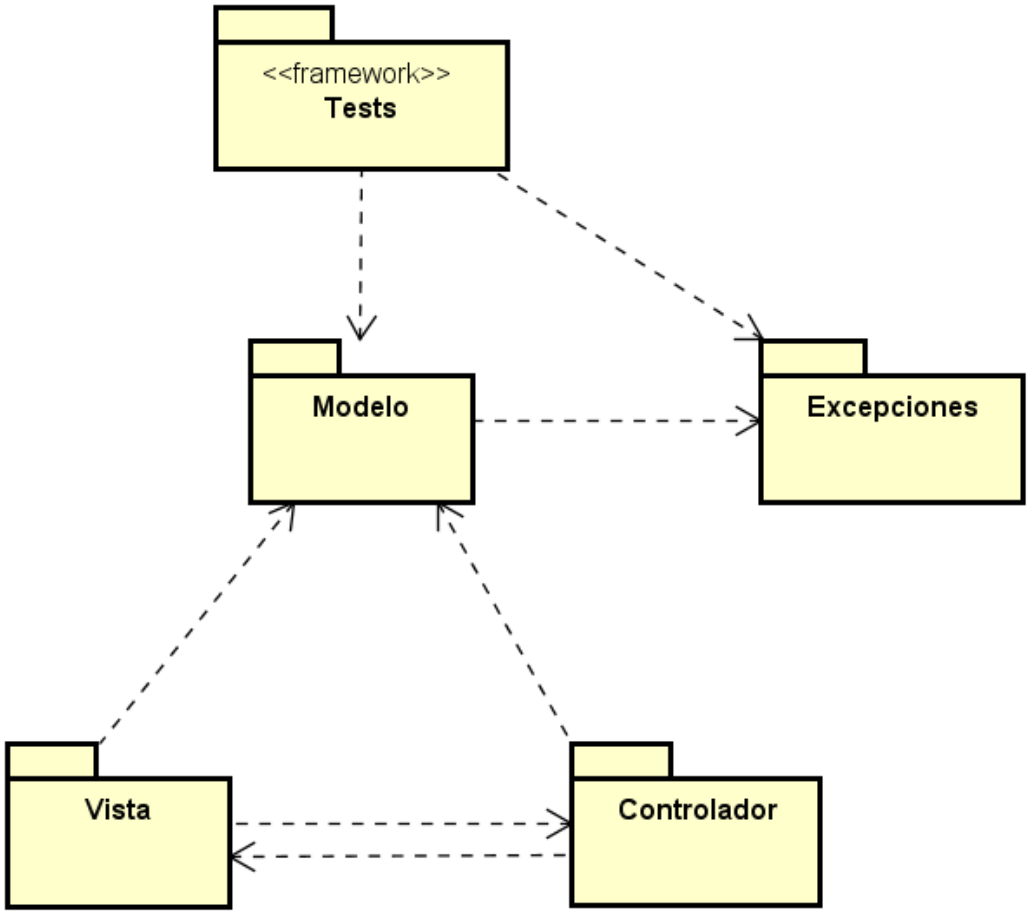
Diagramas de secuencia

Diagrama de secuencia de posible crecimiento de juego (en desarrollo)



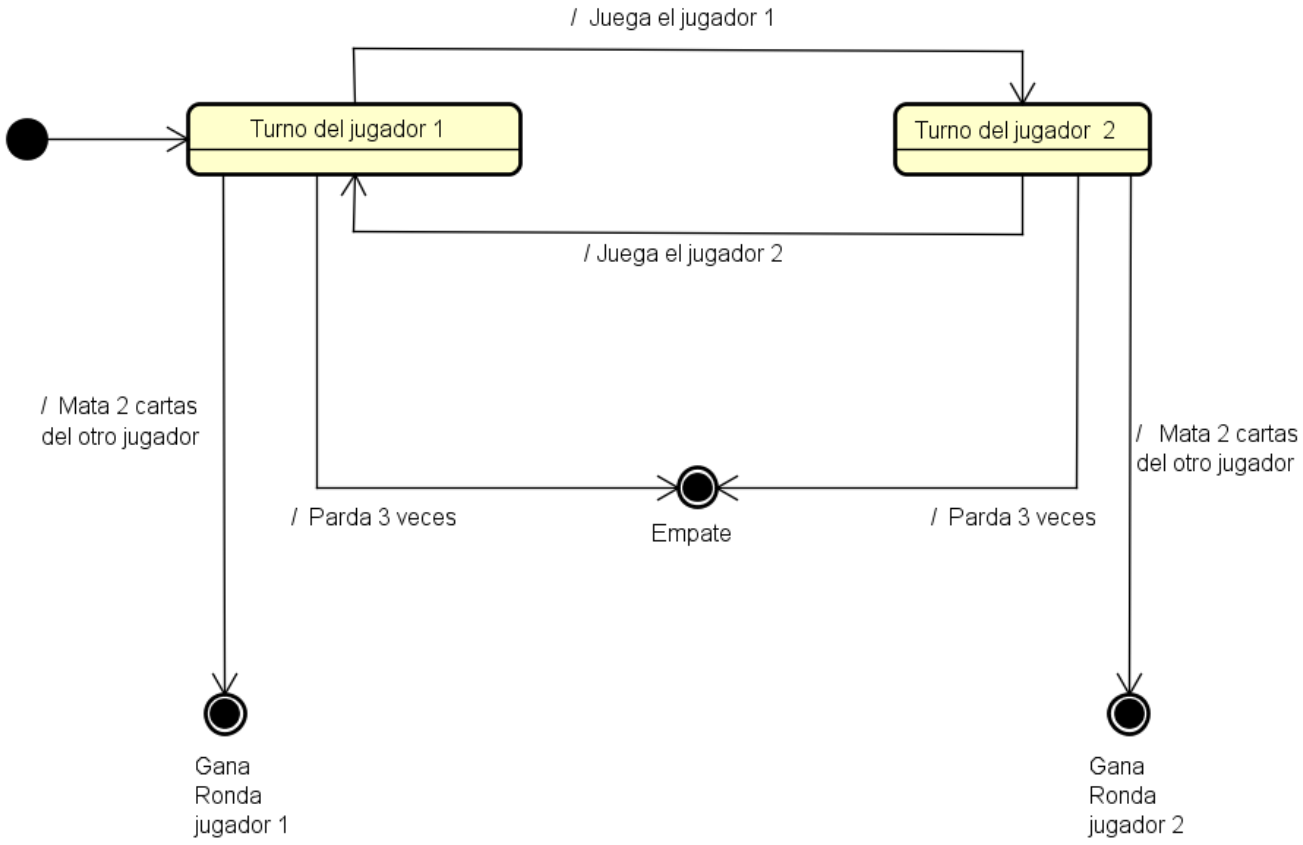
Diagramas de paquete

Diagrama de paquetes, básico por ahora.



Diagramas de estado

Diagrama de estado de partida entre dos jugadores.



Detalles de implementación

//Las implementaciones en esta primera entrega por ahora son simples y entendibles.

Excepciones

Las excepciones son:

- NoHayEquiposException: en el caso de solicitar equipos para trabajar con ellos y estos no han sido creados aún o no existen, se arrojará una excepción en representación de este suceso.
- NumeroFueraDeRangoException: en caso de que la carta a crear no cumpla los requisitos en referencia al valor que poseerá (comprendido entre 0 y 7 como también entre 10 y 12), se arrojará una excepción.
- EmptyListException: cuando se quiere obtener una lista característica desarrollada en el juego , se arrojará una excepción si está vacía
- ValueNotFoundException