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
Piñeiro Vázquez Manuel	96.947	
Angel Martin Ignacio	95.445	
Scardapane Adriano	94.079	

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

Tutor:

Nota Final:

Entrega Nº:

Modelo de dominio

Programa realizado con IDE Eclipse.

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

Aquellas notaciones aparecidas con cursiva en lineas generales representan patrones de diseño 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**, 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 conjunto a la ronda que se esté jugando corresponden una lista de **Vuelta**s que pueden ser una, dos o tres según el transcurso de las **Accion**es desarrollas durante la ronda, y finalmente una clase **Mano** donde los **Jugador**es tendrían y bajarían sus cartas. 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, que no termino de ser completamente funcional. Estos a su misma vez pueden trabajar individualmente o colaborar en un juego dentro de un **Equipo**.

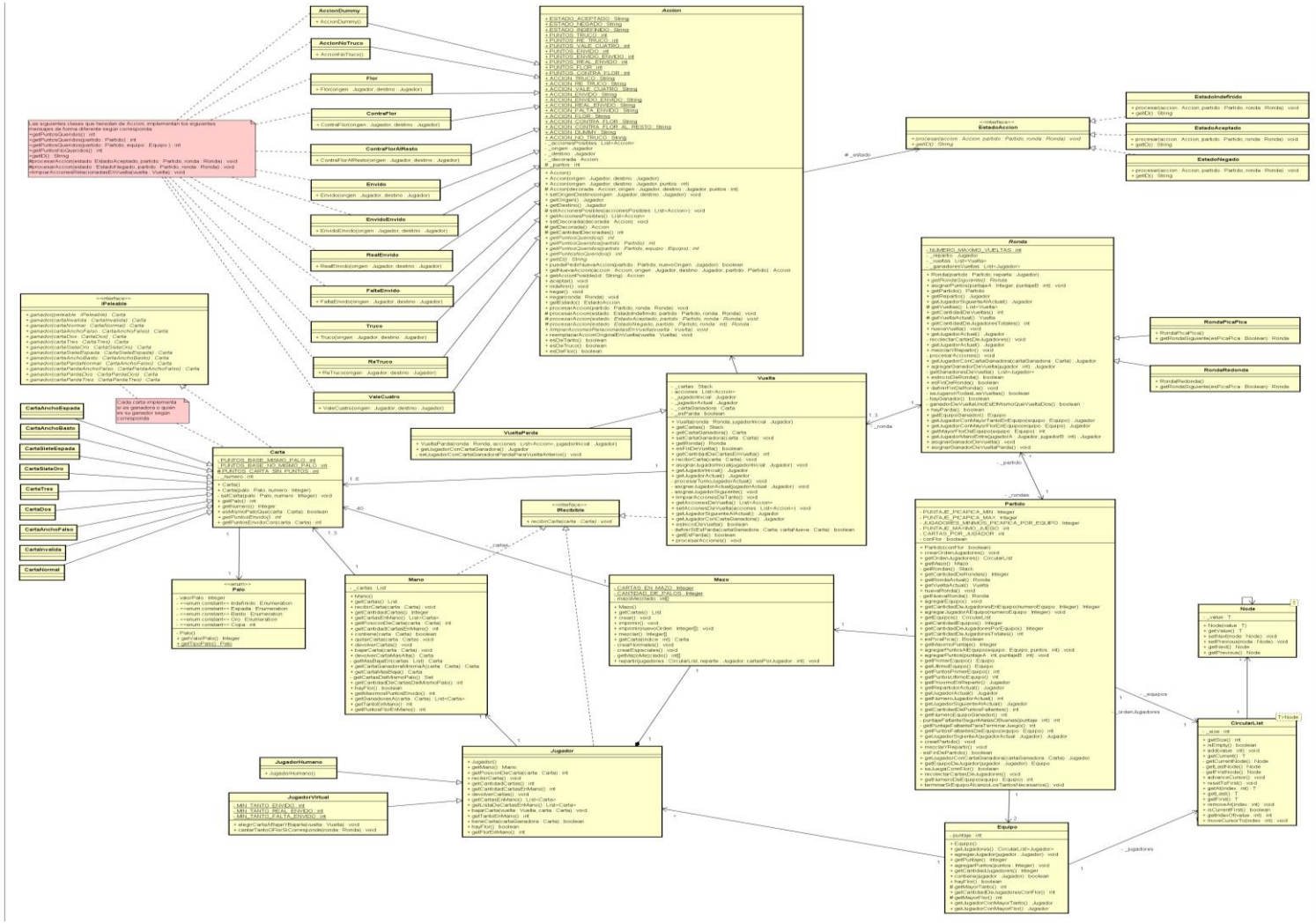
Saliendo fuera del incremento del juego, existen otras clases principales como **Carta** y **Mazo**, entidades de las cuales no podría jugarse la partida si no existiesen. La entidad **Carta** tiene la posibilidad de crearse con un valor y un tipo de **Palo**, para denotar su importancia en el truco 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. Las **Carta**s por su lado, se subdividen en las "Especiales" por decirlo de alguna forma y las "Normales" (siendo estos tipos de cartas clases aparte) e implementan una interfaz llamada **IPeleable**, que permite la implementación de un ganador en relación a una **Carta** específica contra otra **Carta**, y cada carta correspondiente sabe a quién o no le gana por medio de una implementación de *DoubleDispatch*.

Otras clases importantes de mencionar serán aquellas pertenecientes a **Accion** (ya mencionada), vitales en el software, que funcionaran como reguladoras del puntaje de juego durante el transcurso de las rondas. Los usuarios, por medio de la interfaz visual darán a conocer la acción que se quiere realizar y ellas mismas sabrán, por medio de una interfaz llamada **EstadoAccion**, como deben procesar el puntaje según qué acción sean y en qué estado estén ellas mismas, según las respuestas que se generen de un jugador a otro por medio de la interfaz visual y sus correspondientes controladores.

Con lo que respecta a la interfaz visual, no termino subdividiéndose correctamente lo que respecta al MVC pensado prematuramente para el desarrollo de esta misma, sino que la visual tiene una clase **ImprimirTablero** (con un patrón de diseño *Singleton*), que tiene bastante peso en lo que es reaccionar según las acciones y sus correspondientes **EventHandlers**, lo cual debería subdivirse este tipo de responsabilidades en otro paquete.

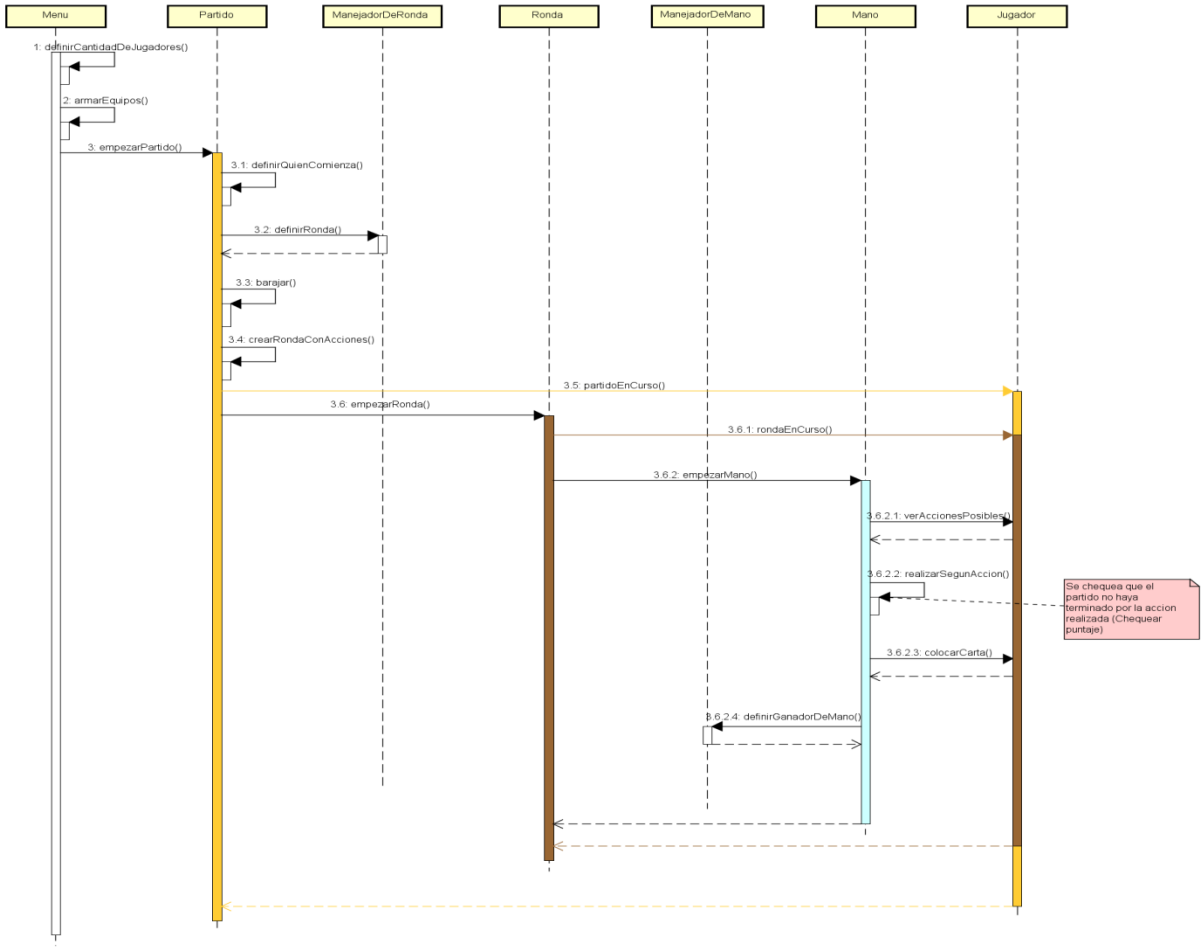
Vale también mencionar la implementación de una **CircularList** para el transcurso del juego.

Diagrama de clases

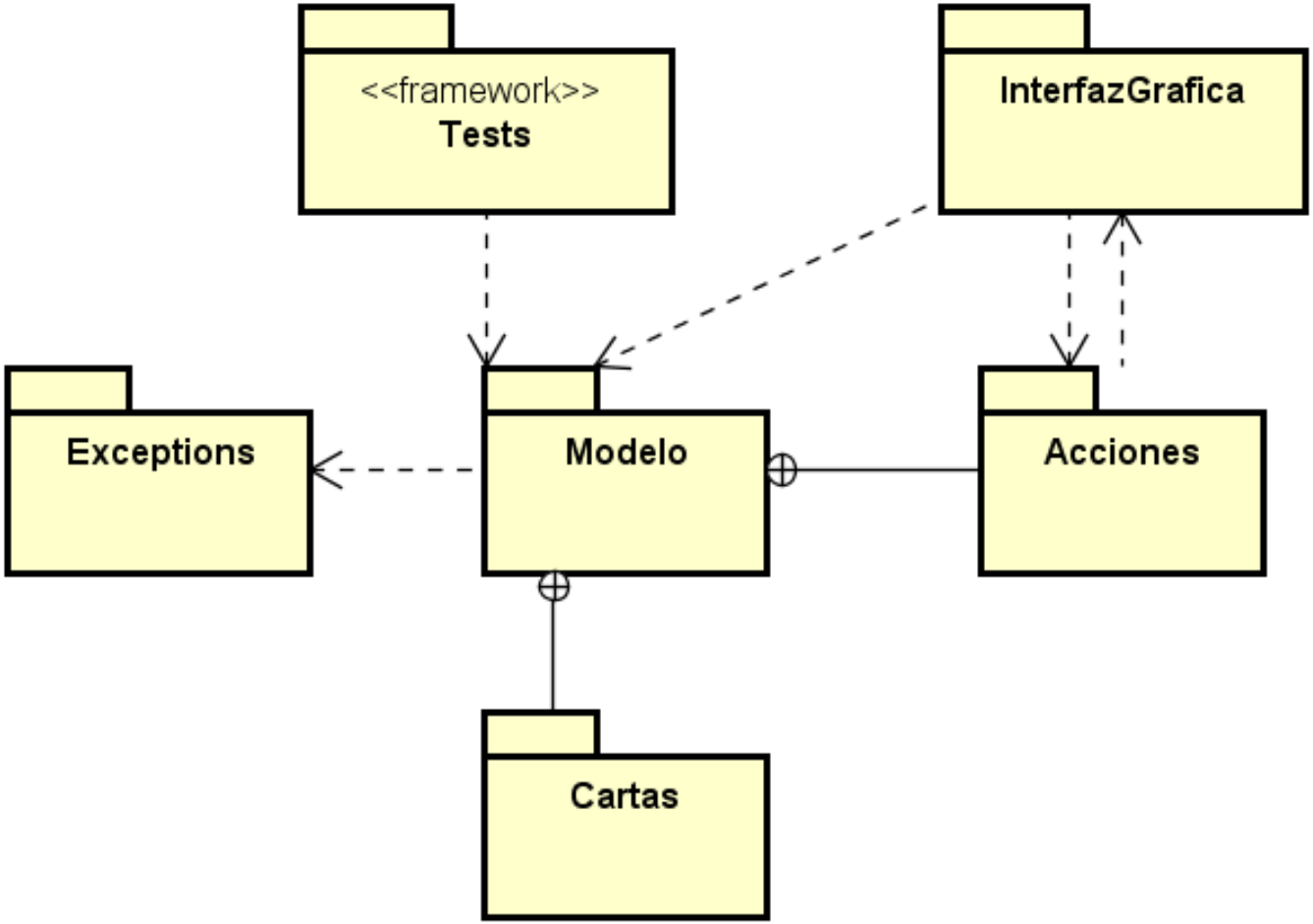


Diagramas de secuencia

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

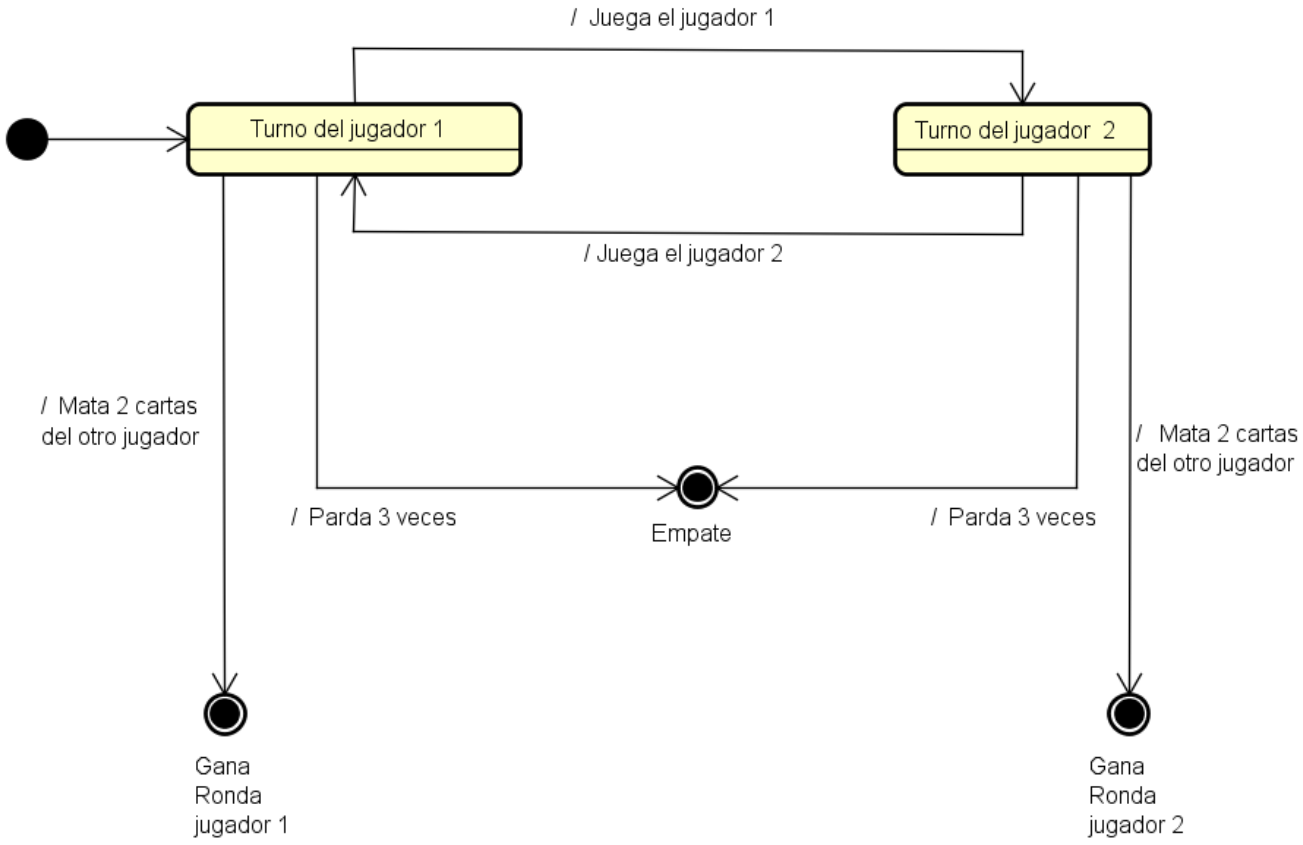


Diagramas de paquete



Diagramas de estado

Diagrama de estado de partida entre dos jugadores.



Detalles de implementación

Lo importante a entender es como está organizada la jerarquía en relación Partido -> Ronda -> Vueltas. Dentro de esto lo único destacable es la relación con una interfaz **IRecibible**, para recibir las cartas por parte del **Jugador** y la **Mano**, como también la **Vuelta** correspondiente para así saber, dentro de las cartas presentadas en la mesa, cual es la ganadora (esto no por parte de la interfaz).

Luego lo más importante y destacable, es la implementación de las **Accion**es. Por medio de una clase **Accion** central, las acciones correspondientes de **Truco**, **ReTruco**, **ValeCuatro**, **Envido**, **EnvidoEnvido**, **RealEnvido**, **FaltaEnvido**, **Flor**, **ContraFlor** y **ContraFlorAlResto**, implementan tanto métodos como una correspondiente interfaz llamada **EstadoAccion**, que permite un **EstadoAceptado**, **EstadoNegado** o **EstadoIndefinido** (implementan también), que permite a las acciones poseer un estado en determinado momento de la partida/ronda el cual procesaran según corresponda generando así el incremento del puntaje y el avance del juego.

También es destacable la implementación ya mencionada al comienzo de las **Carta**s con la interfaz **IPeleable**.

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.
- **AccionDummyException**: en caso de solicitar una acción decorada a la **AccionDummy** se arrojará esta excepción debido al propósito que tiene la **AccionDummy**, que es no tener una acción específica.
- **EmptyListException**: arroja una excepción indicando que la lista esta vacía en relación a la implementación de la **CircularList** desarrollada por el equipo.
- **EstadoIndefinidoException:** en el caso que se desee procesar una acción que tiene el estado indefinido se arrojará esta excepción.
- **JugadorNoPuedeCantarAccionConcatenadaConsecutivamenteException:** en el momento de definir una nueva acción, si no se puede pedir una nueva acción se arrojará esta excepción.

- **NoContieneCartaException:** si por algún motivo la carta a bajar por parte del jugador no está contenida en su mano, se arrojará esta excepción.
- **NoExisteAccionException:** si se busca una acción para procesarla o para otro motivo y no se la encuentra se arrojará esta excepción.
- **NoHayAccionesException:** si quiere obtenerse la lista de acciones posibles y no hay ninguna guardada o registrada la lista estará vacía y se arrojará esta excepción.
- **NoHayEquiposException:** si no se agregaron equipos al partido y quiere obtenerse algún tipo de estado de parte de ellos se arrojará esta excepción.
- **NoHayGanadorException:** si se quiere obtener el número del equipo ganador y aun no es posible ya sea por el estado actual del juego o porque no hay un ganador se arrojará esta excepción.
- **NoHayVueltasException:** si no hay vueltas disponibles en la lista de vueltas de la ronda y se quiere obtener una de ellas se arrojará esta excepción.
- **NoSeEncuentraJugadorException:** si quiere obtenerse que jugador es mano entre dos, y no hay un resultado se arrojará esta excepción.
- **NoSePuedeNoQuererException:** en el caso de que se cante Flor y algún jugador, por algún motivo tuvo la posibilidad de no quererlo, se arrojará esta excepción ya que por default la Flor es querida siempre cuando un jugador la tenga.
- **NoSePuedenRecibirMasCartasException:** si se llega al número de cartas máximo que un jugador puede tener y se desea recibir otra carta se arrojará esta excepción.
- **PaloInvalidoException:** si se quiere inicializar un ancho falso con un palo no correspondiente (es decir, basto o espada), se arrojará esta excepción.
- **ValueNotFoundException**: si se quiere obtener un valor para cierto índice de una CircularList y no se encuentra, se arrojará esta excepción.