FIUBA - 75.07

Algoritmos y programación III

Trabajo práctico 2: FonTruco

2do cuatrimestre, 2015

(trabajo grupal)

Alumnos:

Nombre	Padrón	PuertoEstelarTerranMail
Daciuk, Alexis	97630	adaciuk@gmail.com
Llauro, Manuel	95736	llauromanuel@gmail.com
Risaro, Lucas	94335	lucasrisaro@gmail.com
Rozanec, Matias	97404	rozanecm@gmail.com

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

Tutor:

Nota Final:

Introducción

Objetivo del trabajo

Aplicar los conceptos enseñados en la materia a la resolución de un problema, trabajando en forma grupal y utilizando un lenguaje de tipado estático (Java)

Consigna general

Desarrollar la aplicación completa, incluyendo el modelo de clases e interfaz gráfica. La aplicación deberá ser acompañada por prueba unitarias e integrales y documentación de diseño. En la siguiente sección se describe la aplicación a desarrollar.

Descripción de la aplicación a desarrollar

Se deberá desarrollar una aplicación que implemente el juego de cartas popular Truco (https://es.wikipedia.org/wiki/Truco_(juego_de_naipes)) .

Las características del mismo serán:

- Se podrá seleccionar la variante con/sin Flor.
- El juego será entre jugadores humanos.
- Se podrá jugar en las variantes 2 jugadores, 4 jugadores y pica-pica.
- En la variante de 2 jugadores, se podrá jugar contra la computadora, con una inteligencia mínima y determinística (esto se comentará en clase).

Se desarrollará la interfaz visual para la interacción entre los jugadores.

Entregables

- Código fuente de la aplicación completa, incluyendo también: código de la pruebas, archivos de recursos
- Script para compilación y ejecución (ant)
- Informe, acorde a lo especificado en este documento

Formas de entrega

Habrá **4 entregas formales.** Las mismas tendrán una calificación de **APROBADO** o **NO APROBADO** en el momento de la entrega.

Aquél grupo que acumule 3 no aprobados, quedará automáticamente desaprobado con la consiguiente pérdida de regularidad en la materia. En cada entrega se debe traer el informe actualizado.

1er Entrega: Mínimamente (se aconseja avanzar lo más posible) pruebas de integración y unitarias funcionando que contemplan:

- Modelado de las cartas y los valores entre ellas
- Modelado de 'manos' y 'mesa'
- Modelado de cálculo envido y flor para una 'mesa' dada

2da Entrega: **Modelo del Juego Completo** con todas las pruebas unitarias y de integración que contemplen todos los casos del enunciado, simulando partidas completas en todas sus variantes.

3er Entrega: Interfaz gráfica Parcial. A determinar por el ayudante.

4ta y última Entrega: Trabajo Práctico completo funcionando, con interfaz gráfica final, sonidos e informe completo.

Fechas de entrega programadas

1er Entrega: Miércoles 11 / 11 / 2015 - Jueves 12 / 11 / 2015

2da Entrega: Miércoles 18 / 11 / 2015 - Jueves 19 / 11 / 2015

3er Entrega: Miércoles 25 / 11 / 2015 - Jueves 26 / 11 / 2015

4ta y última Entrega: Miércoles 2 / 12 / 2015 - Jueves 3 / 12 / 2015

Informe

Supuestos

[Documentar todos los supuestos hechos sobre el enunciado. Asegurarse de validar con los docentes]

- Al momento de poder cantar envido, es legal repetir el llamado infinitamente (i.e. no hay límite en cuanto a cantidad de llamadas).

Modelo de dominio

[Explicar los elementos más relevantes del diseño. Es decir: qué entidades se han creado, qué responsabilidades tienen asignadas, cómo se relacionan, etc]

Player: Interfaz implementada por HumanPlayer y CpuPlayer. Contiene los métodos que permiten realizar los cantos propios del juego, así como aceptarlos y poder rechazarlos o irse al mazo.

Slot: Clase que contiene las cartas jugadas por un determinado jugador. Tiene métodos para consultar las cartas en todas las posiciones (tiene 3 posiciones).

Scoreboard: Clase que se ocupa de llevar los puntajes de los jugadores (o equipos, según sea el caso). Se encarga de sumar puntos a quien le corresponda; tiene métodos de consulta de puntajes de jugadores.

Judge: Clase que evalúa rondas de cartas, se encarga de evaluar las cartas que tienen los jugadores en su Slot para decidir quién gano la ronda o la mano, en caso de que un jugador o equipo haya ganado una mano informa al Scoreboard sobre quién y qué juego ganó; en caso de que sea una ronda, le avisa a table quién ganó, así puede empezar la próxima ronda desde ese jugador

Builder: Es la clase encargada de preparar el entorno para que se pueda desarrollar el juego; crea los Players, Teams, Table, Judge, Scoreboard y settea todas las referencias entre cada clase

Hand: Clase que contiene las cartas de la mano del jugador. Una mano es capaz de informar su composición (devolviendo las cartas que la conforman), el puntaje de envido y si forma flor o no.

Team: Clase que contiene una lista de jugadores que conforman un equipo. Tiene métodos que devuelven si un jugador esta en el Team y métodos varios para consultas sobre el Team.

Deck: Clase que representa a un mazo. Tiene un método que devuelve una carta semi-aleatoria de las cartas que tiene

Card: Clase que representa una carta. Tiene un número, palo y valor.

Croupier: Clase que se encarga de devolver una mano que consta de 3 cartas semi-aleatorias listas para jugar, tiene un método que devuelve la mano preparada y un metodo para avisarle que empieza una nueva ronda para que use un mazo entero

Score: Clase que permite manejar los puntajes de un equipo

Diagramas de clases

Se adjuntan aparte ya que no entran aca

Diagramas de secuencia

Idem diagrama de clases

Diagrama de paquetes

Idem diagrama de clases

Diagramas de estado

Idem diagrama de clases

Detalles de implementación

[Deben **detallar/explicar** qué estrategias utilizaron para resolver los puntos más conflictivos del trabajo práctico. (Incluida la persistencia)]

Un punto conflictivo del trabajo fue poder saber exactamente en qué estado se encontraba la partida, i.e. si estaba en juego una llamada de envido, si el truco/retruco/vale 4 ya fueron cantados o no, etc., para poder conocer la cantidad de puntos que habría que otorgarle a los ganadores de la ronda una finalizada cada una de ellas, y poder verificar de alguna manera (no demasiado rebuscada) si una llamada que quisiera hacer algún jugador sería válida en el contexto dado.

Para solucionar esto recurrimos al patrón de diseño state. Así, la mesa guarda el estado en que se encuentra, pudiendo verificarse rápidamente todo lo que se debe verificar, ya que cada estado que puede almacenar la mesa tiene guardada la información necesaria (i.e. posibles llamadas desde ese estado – por ej.: estando en truco se puede cantar retruco, pero no vale 4 o Envido) – y el puntaje que otorga ese estado – por ej.: truco otorga 2 puntos).

Excepciones

Se crearon las siguientes excepciones para el manejo de errores:

DonTHaveThatCardException: Excepción que se lanza cuando un jugador trata de jugar una carta que no posee.

FirstTeamWonException: Excepción que lanza el Scoreboard cuando el primer equipo llegó a 30.

InvalidGameCallException: Excepción que lanza la clase Table cuando se quiere jugar un juego que no corresponde (Ej: Envido después de la primera ronda, Vale 4 sin Truco y Retruco antes).

InvalidNumberOfPlayersException: Excepción que lanza el Builder cuando se pasa una cantidad de jugadores impar o mayor a 6.

NobodyWonYetException: Excepción que lanza el un método privado de la clase Scoreboard, es de manejo interno.

NotCardThrownException: Excepción que lanza el Slot cuando se consulta por alguna carta en cierta posición y esa posición del Slot esta vacía.

NotYourTurnException: Excepción que lanza la Table cuando un jugador trata de jugar y no es su turno.

PlayerDoesNotExistsException: Excepcion que lanza un Team cuando no tiene un Player específico.

SecondTeamWonException: Excepción que lanza el Scoreboard cuando el segundo equipo gano el juego.

TeamDoesntExistException: Excepción que lanza Scoreboard cuando se busca el puntaje de un equipo inexistente.

Checklist de corrección

Esta sección es para uso exclusivo de los docentes, por favor no modificar.

Carpeta

Generalidades

- ¿Son correctos los supuestos y extensiones?
- ¿Es prolija la presentación? (hojas del mismo tamaño, numeradas y con tipografía uniforme)

Modelo

- ¿Está completo?¿Contempla la totalidad del problema?
- ¿Respeta encapsulamiento?
- ¿Hace un buen uso de excepciones?
- ¿Utiliza polimorfismo en las situaciones esperadas?

Diagramas

Diagrama de clases

- ¿Está completo?
- ¿Está bien utilizada la notación?

Diagramas de secuencia

- ¿Está completo?
- ¿Es consistente con el diagrama de clases?
- ¿Está bien utilizada la notación?

Diagrama de estados

- ¿Está completo?
- ¿Está bien utilizada la notación?

Código

Generalidades

- ¿Respeta estándares de codificación?

- ¿Está correctamente documentado?