****

**U.B.A. FACULTAD DE INGENIERÍA**

**Departamento de Informática**

**75.59 – Técnicas de Programación Concurrentes I**

**Primer Proyecto: ConcuPig**

**Curso: 2013 – 1er Cuatrimestre**

|  |  |
| --- | --- |
| **APELLIDO, Nombres** | **N° PADRÓN** |
| Rodriguez, Sebastian | 90202 |
| Schenkelman, Damián | 90728 |
| Servetto, Matías | 91363 |
| **Fecha de Aprobación :** |  |
| **Calificación :** |  |
| **Firma de Aprobación :** |  |

**Observaciones:**

Índice

[Análisis del problema 2](#_Toc355188145)

[Hipótesis 2](#_Toc355188146)

[Casos de uso 3](#_Toc355188147)

[Actores 3](#_Toc355188148)

[Casos de uso 3](#_Toc355188149)

[Diagrama de casos de uso 4](#_Toc355188150)

[Especificación de casos de uso 4](#_Toc355188151)

[Resolución del problema 6](#_Toc355188152)

[Diagrama de clases 6](#_Toc355188153)

[Diagrama de estados del jugador 6](#_Toc355188154)

# Análisis del problema

Nos encontramos ante el problema de implementar la simulación del juego “**Chancho va**”.

Este juego posee la dinámica de varios jugadores interactuando entre ellos al mismo tiempo, donde la interacción se da ente jugadores vecinos. En esta interacción, un jugador envía una carta a un vecino y recibe la carta del otro.

El juego impone la regla de que esta acción la deben realizar al mismo tiempo todos los jugadores e inmediatamente levantadas las cartas los jugadores deben comprobar si su mano es ganadora.

En el caso que un jugador gana, debe poner su mano (no las cartas) en el centro de la mesa y, ante este evento, el resto de los jugadores debe colocar la mano sobre la del ganador sin importar lo que estaban realizando en ese momento, ni el juego de cartas que tenga en la mano.

Para entrar en este estado de juego, existe un memento previo donde se deben repartir cartas a todos los jugadores. La baraja del juego varía según la cantidad de jugadores y debe ser elegido al comienzo. No hay una regla que indique si las cartas las reparte un “**Dealer**” ajeno al juego o los jugadores deben tomar el rol.

Este proceso de acciones “**Repartir**” luego “**Jugar**” se debe repetir tantas veces como sea necesario para llegar al estado en que un jugador pierda 7 veces (por cada ver que pierda se le asigna de puntaje una letra de la palabra “**CHANCHO**” hasta que completa la palabra).

Dado el previo análisis y las técnicas con las que se requiere que se resuelvan nos encontramos con los siguientes problemas del punto de vista de la concurrencia:

* Debemos implementar la simulación de manera que todos los jugadores realicen las interacciones jugador-jugador “**al mismo tiempo**”.
* Realizar la acción del propio jugador sin un orden especifico, o sea, que el enviar y recibir no ocurran de manera secuencial siempre en el mismo orden, deben ocurrir “**al mismo tiempo**”.
* Debemos buscar un mecanismo para sincronizar el reparto de cartas y tanto el comienzo como el fin de cada ronda.
* Implementar una manera de informar a todos los procesos del fin de la partida para que cada uno libere de manera correcta los recursos utilizados.

# Hipótesis

* No delegamos la responsabilidad de “**Dealer**” a los jugadores, contaremos con una entidad encargada de repartir las cartas.
* La ejecución de la simulación termina cuando un jugador consigue las 7 letras. No se elimina este jugador y sigue la partida.
* Dado que la baraja se mezcla de manera puramente aleatoria ya que no tenemos el problema real de que la baraja posee un determinado orden y que depende de la manera que el “**Dealer**” baraje se mezclen “**bien**” o “**mal**”, el “**Dealer**” de la simulación repartirá las cartas siempre empezando con el jugador 0 y no realizara la rotación de jugador como en una partida con personas reales. Si se alternara entre jugador y jugador la carta repartida.

# Casos de uso

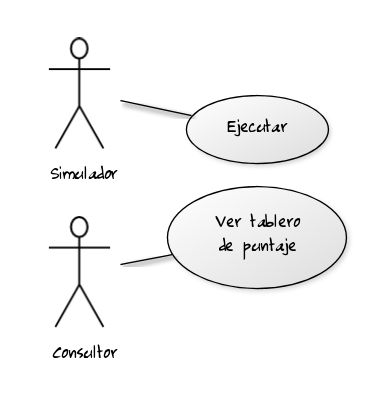
## Actores

* Simulador: Persona encargada de lanzar la simulación del sistema.
* Consultor: Persona que consulta el tablero de puntaje de la simulación.

## Casos de uso

* Ejecutar: El programa es iniciado con las opciones requeridas por el mismo (cantidad de jugadores y modo de ejecución para realizar o no el loggeo de las acciones). Es lanzado por el usuario con el rol de “Simulador”.
* Ver tablero de puntaje: El programa muestra por consola el tablero de puntaje de los jugadores al momento del pedido. Es lanzado por el usuario con el rol de “Consultor”.

## Diagrama de casos de uso



## Especificación de casos de uso

|  |  |
| --- | --- |
| **Especificación de caso de uso** | |
| **Nombre** | Ejecutar |
| **Descripción** | El programa es iniciado con las opciones requeridas por el mismo (cantidad de jugadores y modo de ejecución para realizar o no el loggeo de las acciones) |
| **Actor** | Simulador |
| **Pre condiciones** | - |
| **Post condiciones** | * La simulación se encuentra en ejecución. * Se generan archivos semilla para las estructuras de concurrencia utilizadas en la carpeta donde se ejecuta la simulación. * Se genera un archivo de log en la carpeta donde se ejecuta la simulación si la ejecución es modo “debug”. * Se ve en pantalla el mensaje para dar la orden de ver puntaje. |
| **Flujo normal de eventos** | |
| 1. El usuario envía la instrucción de ejecución ingresando las opciones de ejecución (cantidad de jugadores y modo de ejecución). 2. Si se ingresaron correctamente los parámetros **{A}** 3. Si no **{E}** | |
| **Flujos alternos** | |
| **{A} Los parámetros se ingresaron correctamente**   1. El sistema lee la cantidad de jugadores y genera los archivos y procesos correspondientes. 2. El sistema lanza los procesos de sincronización de jugadores. 3. Si se ingreso en modo debug, el sistema genera el log. 4. El sistema lanza el proceso del puntaje y este muestra por pantalla la opción de “ingresar una tecla para mostrar puntaje”. 5. El sistema comienza a jugar. 6. Fin del caso de uso. | |
| **Flujos de excepción** | |
| **{E} Se ingresaron parámetros incorrectos**   1. El sistema muestra un texto de ayuda donde muestra como ejecutar el programa. 2. Termina la aplicación. 3. Fin del caso de uso. | |

|  |  |
| --- | --- |
| **Especificación de caso de uso** | |
| **Nombre** | Ver tablero de puntaje |
| **Descripción** | El programa muestra por consola el tablero de puntaje de los jugadores al momento del pedido. |
| **Actor** | Consultor |
| **Pre condiciones** | - |
| **Post condiciones** | * Por consola se ve el puntaje * Se vuelve a mostrar el mensaje para dar la orden de mostrar el puntaje. * El proceso de consulta terminado si se ingresa la opción de terminarlo. |
| **Flujo normal de eventos** | |
| 1. El consultor ingresa una tecla a la consola donde se muestra el mensaje con la instrucción para ver el puntaje. 2. El sistema lee la tecla. 3. Si la tecla es “Escape”, el sistema termina con el proceso de consulta. 4. Si no, muestro puntaje, luego muestro el mensaje de instrucción para pedir el mensaje. 5. Fin del caso de uso. | |
| **Flujos alternos** | |
| - | |
| **Flujos de excepción** | |
| - | |

# Resolución del problema

## Procesos involucrados en la simulación

Se comenzara la descripción de la resolución del problema explicando la estructura de los procesos generados:

Se pueden dividir en dos grandes rangos:

* Los procesos del jugador.
* Los procesos de sincronización de la partida.

#### Los procesos del jugador

Se crearon 3 procesos que representan a la entidad de jugador en la simulación:

***La cabeza***, implementada en la clase “**PlayerHead**”. Se encarga de crear los procesos hijos de envió y recepción. Tiene la lógica para la selección de la carta a enviar, saber si gano la partida, poner la mano en la mesa cuando gana o cuando es informado que otro jugador puso la mano y para sincronizar el envió y recepción de cartas.

***El receptor de cartas***, implementado en la clase “**PlayerCardReceiver**”. Se encarga de recibir una carta de su vecino y de compartir la carta recibida con “*la cabeza*”.

***El remitente de cartas***, implementado en la clase “**PlayerCardSender**”. Se encarga de enviar la carta al otro vecino que *la cabeza* le comparte.

#### Los procesos de sincronización de la partida

Se crearon los siguientes procesos:

***La mesa***, implementada en la clase “**Table**” y en el “**main**” del programa. Este proceso, como “**main**” del programa, inicializa los archivos y estrucutras que se utilizan en el programa. Lanza a los procesos *cabeza* de los jugadores, y el resto de los procesos que se explicaran posteriormente. También se encarga de la terminación de los procesos y el cierre de los recursos una vez terminada la simulación.

Como “**Table**”, tiene la responsabilidad en el loop principal del juego: reparte las cartas y envía la señal de la mano en la mesa. El loop termina cuando se detecta que un jugador recibe las 7 letras de “**CHANCHO**”.

***El tablero de puntaje***, implementador en la clase “**ScoreBoardController**”. Se encarga de leer la tecla de la pantalla y mostrar el puntaje ante ese evento o terminar, dependiendo esto último de si la tecla es “**Escape**” o no. También debe compartir los puntajes y la lógica de la condición de corte de juego con el proceso de *la mesa*.

***El sincronizador de jugadores***, implementado en la clase “**PlayerSynchronizer**”. Se encarga de dar la señal de pasaje de cartas cuando los jugadores están listos para realizar la acción.

|  |
| --- |
| **Aclaración**: la frase “***dar la señal***” que se uso en la sección de “**procesos involucrados**” no hace referencia al mecanismo de señales de concurrencia. |

## Comunicación entre procesos

#### Comunicación entre los procesos del jugador

\*\*\*Describir como se comunica la cabeza con las manos.

#### Comunicación entre el jugador y los procesos de sincronización

\*\*\*Describir el resto de las comunicaciones.

# Diagrama de clases

\*\*\*Acá va el diagrama

# Diagrama de estados del jugador

\*\*\*Acá va el diagrama de estados