ESTRUCTURAS DE DATOS Y DE LA INFORMACIÓN

Ingenierías Técnicas de Informática de Gestión y Sistemas

PRÁCTICA FINAL

Primer cuatrimestre
Curso 2009/2010

JUEGO DE LAS 21

Objetivo

Desarrollo de un programa principal y de las clases necesarias para crear una aplicación que simule jugar al juego de cartas llamado "las veintiuna" (http://es.wikipedia.org/wiki/Blackjack).

El juego de las veintiuna

En cada turno del juego de las veintiuna, cada jugador va robando cartas con el objetivo de sumar 21 puntos. Gana el jugador que, sin pasarse, más se aproxime a 21 puntos.

El juego vamos a dividirlo en *turnos*.

- En cada turno, juega un único jugador, donde dicho jugador irá robando cartas hasta que decida plantarse. Cada oportunidad de robar lo llamaremos *ronda*.
- Un jugador se plantará obligatoriamente al llegar a o pasarse de 21 puntos.
- Cuando finalice el turno del último jugador, se mostrarán las cartas para calcular la puntuación obtenida por cada uno de ellos.
- Gana el jugador que haya llegado a 21 puntos, o aquél que, sin pasarse, más se aproxime a 21 puntos.

Para nuestra implementación:

- Usaremos una baraja de Póker, y asumiremos que cada carta suma el valor correspondiente a su número.
- En particular, la *J* sumará 11 puntos, la *Q* 12 puntos, y la *K* 13 puntos.

Diseño de la solución

Para resolver el problema anterior se propone el siguiente diseño de clases.

Lista de Clases básicas involucradas:

- Carta
- Baraja
- Jugador
- Juego
    - Juego21

A continuación se describen los elementos básicos de las clases propuestas.

1. Clase Carta

Descripción: representa una carta del juego.

Atributos:

- *palo*: string. Ej.) "diamantes"
- *número*: entero. Ej.) 5

Métodos:

- *Carta(), Carta(elPalo, elNumero)*

- *getPalo()*
- *getNumero()*
- *operator<<()*: ejemplo de salida "5 de diamantes".

Ejemplo,
```
Carta c("diamantes", 5);
```

2. Clase Baraja

Descripción: representa una baraja de cartas en el juego (conjunto de cartas). Para su implementación, se hará uso de algún **contenedor STL** adecuado.

Atributos:
- *lasCartas*: estructura STL que contiene un conjunto de cartas.

Métodos:
- *Baraja(), Baraja(int tipoBaraja)*: tipo 1 es Póker y tipo 2 es Española.
- *barajar()*: cambia el orden de las cartas en la baraja.
- *getNumCartas()*
- *robarCarta()*: devuelve (y elimina de la baraja) la carta situada en la parte superior de la baraja.
- *addCarta(carta)*: añade una carta al final de la baraja.

Ejemplo,
```
Baraja b_poker(1), b_espa(2);
```

3. Clase Jugador

Descripción: representa un jugador que participa en un juego de cartas.

Atributos:
- *nombre*: string con el nombre del jugador.
- *lasCartas*: conjunto de cartas que tiene actualmente el jugador en su poder. Para su implementación, se hará uso de una clase adecuada.
- *esHumano*: si true, lo maneja una persona; si false, lo maneja el computador.

Métodos:
- *Jugador(), Jugador(nombre, esHumano)*
- *setNombre(nombre)*
- *getNombre()*
- *getCartas()*
- *addCarta(carta)*: añade una carta a la mano del jugador.

Ejemplo,
```
Jugador j1("Manuel", true), j2("Rafael", false);
```

4. Clase Juego

Descripción: clase "abstracta" que representa un juego de cartas, con su baraja y un conjunto de jugadores.

Atributos:
- *laBaraja*: baraja de cartas asociada al juego actual.
- *losJugadores*: conjunto de jugadores participantes en el juego.
- *jugadorActual*: contiene el índice del jugador actual.
- *losScores*: contiene la puntuación actual de cada jugador. Inicialmente serán 0.

Métodos:
- *addJugador(unJugador)*: añade un nuevo jugador a la partida.
- *puntuarJugador(unJugador)*:
    1. Dice la puntuación del jugador objetivo según las cartas que tiene actualmente en la mano.
    2. Este método será redefinido por los descendientes de la clase según el juego. Se debe definir como "*virtual*".
- *numJugadores()*: número de jugadores actuales.
- *setBaraja(laBaraja)*: establecer la baraja del juego.
- *getScore(jugador_index)*: devuelve la puntuación actual del jugador dado por el índice pasado como argumento.
- *getJugadorActual()*: devuelve el índice del jugador que está activo actualmente.

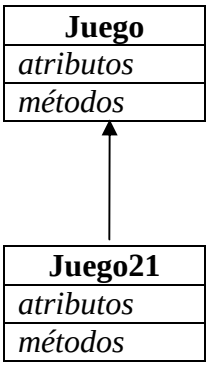
5. Clase Juego21

Descripción: clase que hereda de la clase Juego. Representa un juego de cartas concreto, en este caso el juego de "las 21".

Atributos:
- Los heredados de Juego.
- *rondaActual*: número de ronda en cada momento. Se entiende por ronda el número de veces que lleva jugando (opción a robar) el jugador actual.

Métodos:
- Los heredados de Juego.
- *jugar()*:
    1. Este método comienza la simulación del juego de "las 21".
    2. Antes de llamar a este método debemos añadir los jugadores.
- *nextJugada()*:
    1. Evalúa la siguiente jugada.
    2. En cada jugada pueden darse las siguientes situaciones:
        i. El jugador actual roba y no se planta;
        ii. El jugador actual roba y se planta, por tanto, se pasa al siguiente jugador.
    3. Este método devuelve algún código para saber si el juego ha terminado o se puede seguir jugando.
    4. Se termina el juego cuando el último jugador se planta.

- *getRonda()*: devuelve el número de ronda para el jugador actual.
- *guardarEstadoActual(fichero)*: guarda en fichero el estado actual de la partida.
- *recuperarPartida(fichero)*: recupera una partida guardada en fichero.
- *addPuntuacion(ficheroScores)*: actualiza en un fichero el número de victorias del jugador ganador de la última partida.



Enunciado de la práctica

- Desarrolle el código correspondiente a las clases previamente descritas
  - *Carta.cpp/.h*
  - *Baraja.cpp/.h*
  - *Jugador.cpp/.h*
  - *Juego.cpp/.h*
  - *Juego21.cpp/.h*

  para generar un programa que permita jugar a las veintiuna.

- El programa principal (contiene función *main*)
  - Se llamará *jugar21.cpp*.
  - La sintaxis de llamada al programa será:

```
$> ./jugar21 [–n <numJugadores>] [-h <unJugadorHumano>] [-i <ficheroPartidaGuardada>] [-o <ficheroSalidaPartida>]
```

Donde:

- `-n <numJugadores>`: número entero que indica el número de jugadores de la partida. Entre 2 y 6 jugadores.
- `-h <unJugadorHumano>`: toma el valor 0 para indicar que no existe ningún jugador humano en la partida, y toma el valor 1 para indicar el jugador número 1 es controlado por un humano.
- `-i <ficheroPartidaGuardada>`: indica el nombre del fichero de entrada que contiene una partida guardada para ser cargada.
- `-o <ficheroSalidaPartida>`: indica el nombre del fichero de salida donde se puede guardar el estado actual del juego.
- `-s <ficheroScores>`: indica el nombre del fichero donde se almacenan las victorias de cada jugador humano. Si no se incluye esta opción, se usará el fichero *victorias.dat*.

El programa principal soportará los siguientes **modos de juego**:

1) **Simulación**: no existe jugador humano.
   a. Opción por defecto si **no** se usa "`–h 1`" en la llamada al programa.
2) **Interactivo**: existe un (único) jugador humano.

a. Este modo de juego se activará si en la llamada al programa se incluye la opción "`-h 1`".
b. Se considerarán los siguientes detalles:
    i. En este modo, el primer jugador será siempre el humano. Antes de comenzar la partida se pedirá el nombre del jugador humano para actualizar el fichero de victorias.
    ii. Después de cada partida, el programa dará las opciones de
        1. volver a jugar,
        2. terminar el programa,
        3. guardar el estado actual (número de jugadores, número de juegos terminados y victorias asociadas a cada jugador en la ejecución actual del programa),
        4. mostrar estadística de victorias (absoluto y porcentaje) para los jugadores actuales (humano y computadores actuales).

Detalles técnicos

- Fichero de victorias:
    - Se generará (si no existe) un fichero *victorias.dat* donde se almacene para cada jugador humano (identificado por su nombre, introducido al inicio de la partida), el número de veces que ha jugado dicho jugador y el número de victorias conseguidas.
    - Cada línea del fichero tendrá el siguiente formato:

```
Nombre_jugador Número_partidas_jugadas Número_victorias
```

    - Corresponde al método: *Juego21::addPuntuacion()*

- Fichero de sesión guardada:
    - En la primera línea, un entero que indica el número de jugadores en la partida.
    - Segunda línea, contiene un entero que indica el número de partidas jugadas hasta el momento en la sesión guardada.
    - Siguientes líneas, en cada línea el nombre del jugador y el número de juegos ganados por dicho jugador.
    - Corresponde a los métodos: *Juego21::guardarEstadoActual()*, *Juego21::recuperarPartida()*.

- Comportamiento del juego del computador:
    - Modelaremos el comportamiento del computador de tal modo que conforme más puntos acumule en la mano, más probable es que se plante.
    - Por tanto, la probabilidad de plantarse *P(x)*, puede definirse como:

$$P(x) = \frac{1}{1+\exp(-w \cdot x)}$$

Donde, *x* es el número de puntos en mano y *w* es un parámetro que controla la velocidad con que aumenta el valor de *P(x)*. Este valor debe ser fijado por el alumno. Por ejemplo, *w = 0,05*.

El valor de P(x) estará en [0,1], por tanto, podemos usar *rand()* para generar un número aleatorio, y usar dicho valor para decidir si plantarse de la siguiente forma:

```
int v_rand = rand()%100;
float prob = P(puntos)*100;
if (v_rand < prob)
     plantarse = true;
else
     plantarse = false;
```

Entrega

- Los ficheros fuente desarrollados (*.cpp, .h, makefile, README.txt*) se entregarán mediante la tarea Moodle habilitada para ello.

- Todos los ficheros estarán dentro de un único fichero zip con el siguiente formato de nombre:
  <Apellido1>_<Apellido2>_<Nombre>_edi.zip

  Ejemplo: *Marin_Garcia_Rafael_edi.zip*

- La **fecha límite** de entrega es el día **15 de enero de 2010**.

OBLIGATORIO:

- Incluir fichero *makefile* que permita generar el programa principal *jugar21* y limpiar los ficheros objeto generados.
  - Será optativo el incluir una opción para generar la documentación de código con la herramienta Doxygen.
- Incluir un fichero *README.txt* que describa cómo compilar y ejecutar el programa desarrollado.
- Para que la práctica sea corregida, el alumno debe asegurarse de que sus clases desarrolladas son compatibles con el fichero de test (contiene función *main*) que estará disponible en Moodle. Cualquier práctica que no supere la compilación y ejecución correcta con dicho fichero de test, **no será corregida**.

OBSERVACIONES:

- Se valorará la generación de documentación de código mediante la herramienta Doxygen.
- Cualquier código total o parcialmente copiado será automáticamente calificado con 0.