1. DESCRIPCIÓN DE LA PRÁCTICA

El objetivo de la práctica es desarrollar el juego de cartas **los seises**. Como muchos otros juegos de cartas, se juega encima de una mesa y participan cuatro jugadores, posicionados cada uno en los cuatro lados (norte, sur, este y oeste) de la mesa. Las reglas del juego son las siguientes:

- La baraja tiene 52 cartas agrupadas en 4 palos (tréboles, corazones, espadas y diamantes). Cada palo tiene 13 cartas, con valor de 1 a 13 (el As (A=1), las numeradas del 2 al 10, la sota (J=11), la reina (Q=12) y el rey (K=13)).
- Al inicio del juego, se reparten todas las cartas, cogiendo una a una por encima de la baraja (ver Figura 1), y repartiéndolas entre los 4 jugadores. Cada jugador las va colocando en el mismo orden en que las va recibiendo (ver Figura 2) y se queda con 13 cartas.
- El juego se juega de la siguiente forma:
    - El objetivo es ir formando filas, una por cada palo, encima de la mesa. Cada fila estará formada solamente por cartas del mismo palo y en orden numérico. La formación de una fila sólo puede empezar con un 6 y posterior, se irán colocando los menores a su izquierda y los mayores a su derecha, respetando el orden numérico.
    - Normalmente, el jugador que empieza el juego es el que tiene el 6 de corazones,
    - El modo de jugar de cada jugador es intentar poner una carta que tiene en la mano:
        - Primero, analiza y selecciona una carta
        - Segundo, intenta poner la carta encima de la mesa:
            - Si puede la pone en la fila correspondiente a su palo.
            - Si no puede ponerla, la carta vuelve a la mano.
        - El turno pasa al siguiente jugador.
    - Las opciones para que un jugador pueda poner una carta encima de la mesa son:
        - Es su turno.
        - La carta que quiere poner debe ser seleccionable (sólo la primera).
        - La carta es un 6 y puede colocarla en cualquier fila que esté vacía (ver Figura 3).
        - Si la carta no es un 6, pero es el siguiente de alguna de las cartas que están en la mesa. Por ejemplo, si la carta del jugador es el 4 de trebol y en la mesa está el 5 de trebol, ó si la carta es el 7 de diamantes y en la mesa está el 6 de diamantes (ver Figura 3).
        - Si no ha podido ponerlo, pierde su turno (lo que se conoce como realizar "paso") y se incrementa el número de pasos de dicho jugador (ver el valor "f" de los jugadores en las Figuras adjuntas).
    - El juego finaliza cuando todas las filas se han completado o todos los jugadores ya no tienen cartas en la mano.
    - El ganador del juego es el primer jugador que ha puesto todas sus cartas en las filas, es decir, no tiene más cartas en la mano. También será el que menos "pasos" ha realizado.
- Restricciones de movimientos en el juego:
    - Para colocar una carta en alguna de las filas se tiene que cumplir: 1) que todas las cartas estén repartidas; y 2) anteriormente la carta tiene que estar en las manos de algún jugador. Es decir, no se puede colocar una carta directamente desde la baraja.
    - Si un jugador intenta poner una carta en una fila, pero se cumple que: 1) no es su turno; 2) la carta no es seleccionable, o 3) no se puede poner, entonces la carta vuelve a su origen.

Anexo I.

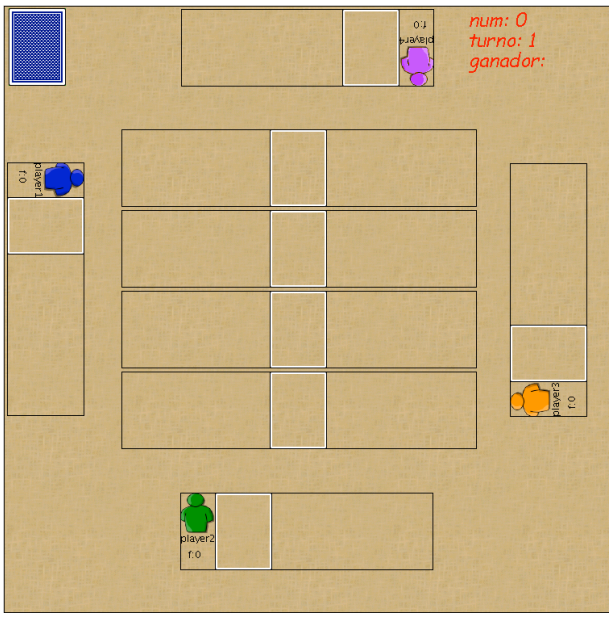

Figura 1. Repartiendo cartas

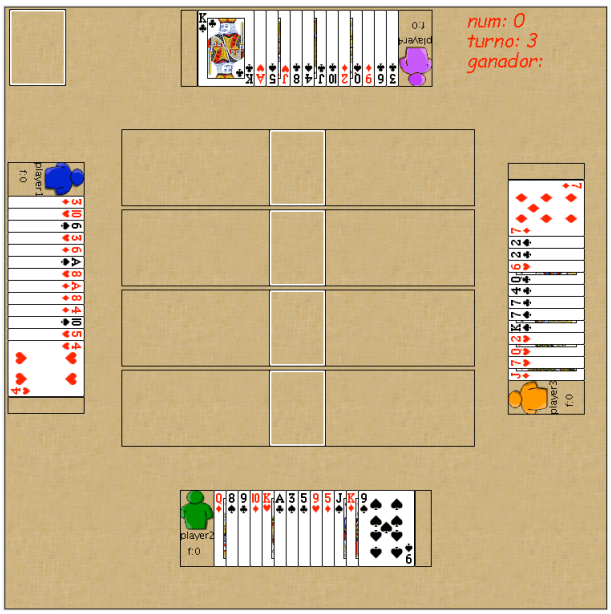

Figura 2. Todas las cartas repartidas

Nota: La Figura 1 muestra como se han posicionado los jugadores (p.ej. player1 está en el OESTE y player2 en el SUR). En la Figura 2 se puede observar que todas las cartas están repartidas y que el jugador que sale a jugar es el player3 (según indica el valor del turno en el marcador, que indica el 3.er jugador desde la derecha de la baraja,).

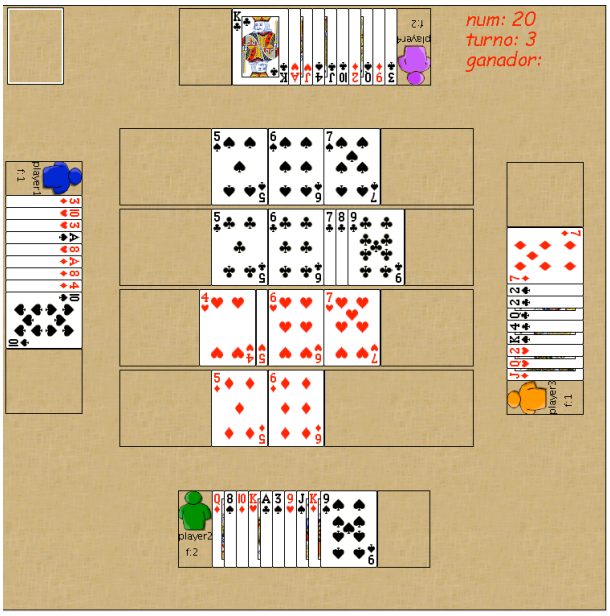

Figura 3. Situación del juego después de 20 turnos

Nota: En la parte superior derecha de la Figura 3 se indica que se han jugado 20 turnos y ésta va a ser la 21, y es turno del jugador 'player3'. También se puede observar el número de pasos que ha realizado cada jugador: 1) f: 1; 2) f:2; 3) f:1 y el 4) f:2. En cuanto al palo de cada fila, comentar que no están prefijadas de antemano, si no se establecen con la colocación del primera carta, es decir, un 6.

2. DISEÑO Y FUNCIONALIDAD DEL JUEGO

La información central del juego es la representación de las cartas (`Card`) y los distintos conjuntos de cartas que componen el juego: la baraja de cartas (`Deck`), las cartas que tienen en las manos los 4 jugadores (`Player`) y las 4 filas de cartas (`CardRow`) que hay que componer siguiendo las reglas del juego. Y toda esta información está agrupada encima de la mesa de juego (`TableGame`). Además de estas entidades, la entidad turno (`Turn`) controlará el turno de cada jugador. La Figura 4 muestra las dependencias entre estas entidades.

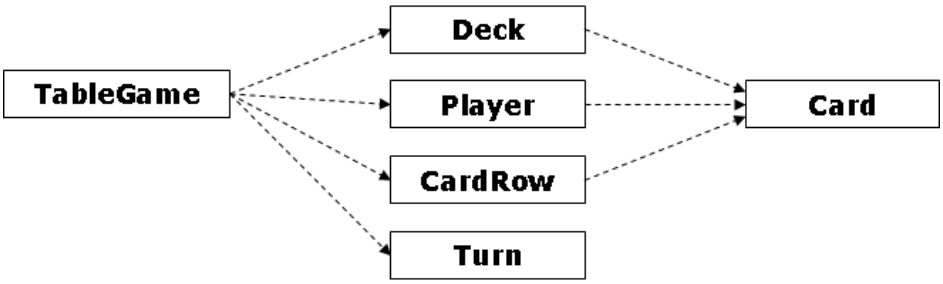

Figura 4. Resumen del diseño del juego mediante el diagrama de clases UML

La Figura 5 describe con más detalle la representación de las entidades más importantes del juego:

- La clase `Card` representa la información de una carta: el palo (`suit`) al que pertenece, su valor (`value`) y el color de la parte trasera (`colour`). Los tipos de datos `Suit` y `Value`, enumeran los posibles palos y valores que pueden describir una carta. El atributo `accepter` identifica a qué conjunto de cartas (jugador o fila) pertenece la carta, para poder implementar las reglas del juego (p.ej. para poder poner una carta en una fila, la carta tiene que estar en las manos de un jugador). Los demás atributos (`flipped` y `canDrag`) están relacionados con la visualización y la interacción del juego.

- La clase `Deck` representa la baraja de cartas, la cual está compuesta por 52 cartas que pertenecen a 4 palos distintos, 13 cartas en cada palo.

- La clase `Player` representa la información de un jugador respecto al juego: su nombre, el número de "pasos" realizados (`failures`) y las cartas que tiene en la mano (`cards`). Estas cartas las agrupa en una estructura de datos para ir añadiendo las cartas según las va recibiendo.

- La clase `CardRow` representa una fila de cartas. Ésta sólo puede pertenecer a un palo (`suit`), que se establecerá al colocar la primera carta con el valor 6. Las cartas de la fila se van guardando en una estructura de datos según el orden correspondiente de visualización.

- La clase `TableGame` representa todo el juego. Por lo tanto, define una baraja de tipo `Deck`, 4 jugadores agrupados en un array de tipo `Player`, 4 filas agrupadas en otro array de tipo `CardRow` y un objeto de tipo `Turn`, para controlar el orden y turno de los jugadores. También utilizará el tipo de dato `Position`, para enumerar las posiciones de los jugadores en la mesa.

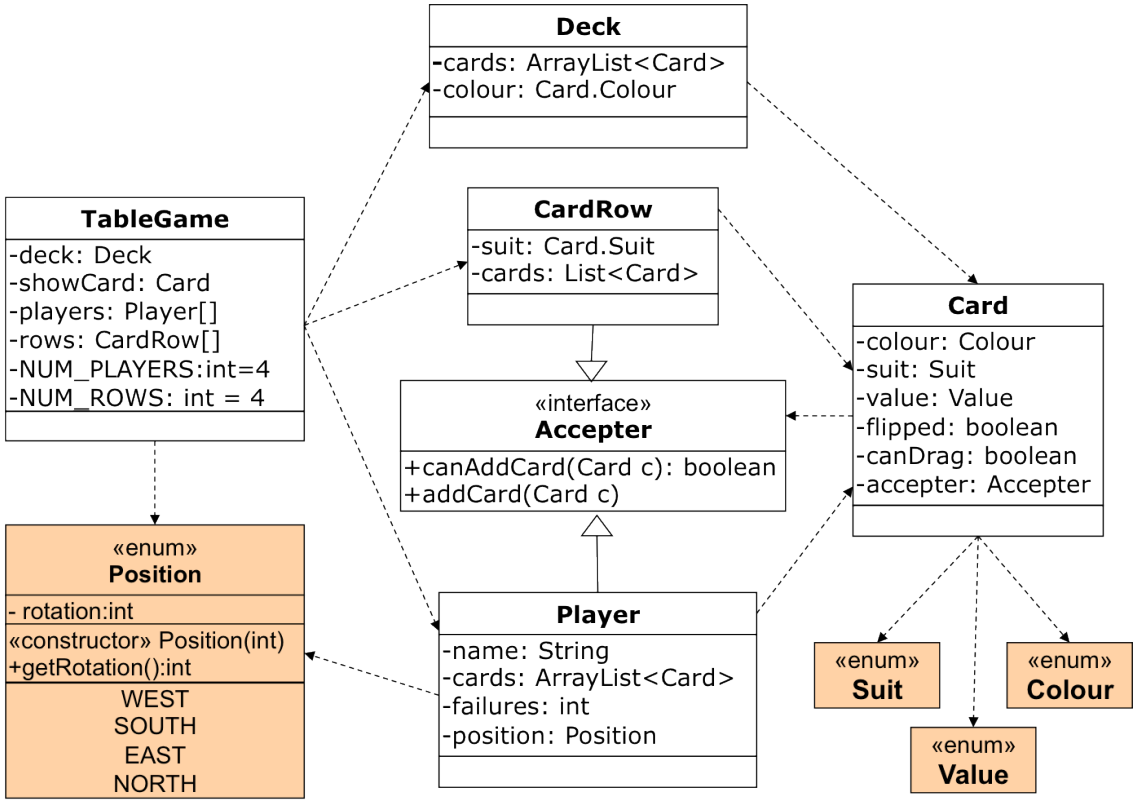

Figura 5. El diseño del juego representado mediante el diagrama de clases UML

2.1. FUNCIONALIDAD DEL JUEGO

El objetivo de la práctica es implementar algunos métodos de las clases comentadas anteriormente para que den soporte a la funcionalidad y reglas de juego comentadas en el apartado 4. Además de la funcionalidad requerida, el juego tendrá una interfaz gráfica con la opción de que el usuario pueda interaccionar en el juego mediante el ratón.

2.1.1 Material y funcionalidad ya implementada

Además del lenguaje de programación Java, se ha decidido utilizar el Framework Greenfoot [2] para facilitar el desarrollo de la práctica. El Framework Greenfoot se compone de 5 clases [3]: `World`, `Actor`, `GreenfootImage`, `Greenfoot` and `MouseInfo`, las cuales facilitan al programador el desarrollo de cualquier programa de juegos o simulación con interfaz gráfica.

El archivo *seises.zip* contiene el proyecto con una implementación parcial, que el alumno/a deberá de ir complentando según el apartado 2.1.2.

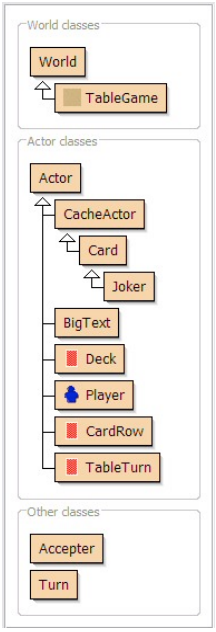

Figura 6. Las Clases

La Figura 6 muestra, la clase `TableGame` que representará el mundo de Greenfoot y los actores principales serán las cartas (`Card`), los jugadores (`Player`) y las filas de cartas (`CardRow`). La funcionalidad que se implementa en los métodos de las clases, básicamente se divide en tres grupos: consulta y modificación del estado de los objetos, visualización y control del programa. En éste último es donde el Framework Greenfoot nos ayuda mediante el método `act`() del mundo y los actores, así como las clases `Greenfoot` y `MouseInfo` [3].

El control del juego incial, se divide entre los métodos `act`() de las clases `TableGame` y `Card`. Dichos métodos están ya implementados, así como parte de los métodos para la visualización de las cartas.

El método `act`() de la clase `TableGame` es sencilla y controla: 1)si el juego ha finalizado o no, y 2) si el usuario hace *un-click* encima de la baraja: a) si se hace *un-click* con el botón *izquierdo* del ratón, se muestra la carta que está encima de la baraja y se deja encima de la mesa para que sea el usuario el que lo reparta de forma manual (arrastrando la carta al correspondiente jugador, según el turno indicado en el marcador) y b) si se hace *un-click* con el botón *derecho* del ratón, se reparten todas las cartas entre los jugadores.

El método `act`() de la clase `Card` es más complicada y controla todos los eventos que el usuario realiza con el ratón sobre la carta, tales como, hacer *un-click*, hacer *doble-click*, *presionar* botón, *arrastrar* o *terminar el arrastre* [3]. En cada caso delegará la acción a otro método de la propia clase o alguna otra clase.

Una vez iniciado el juego, éste se juega interactivamente con el ratón. Algunas de las interacciones que podemos realizar son las siguientes:

- Hacer *un-click* con el botón *izquierdo* encima de la baraja: se muestra la carta de encima.
- *Presionar* con el botón izquierdo sobre la carta mostrada de la baraja, *arrastrar* hasta que esté encima de un jugador y *soltar* el botón: si el turno es del jugador lo aceptará y si no, la carta vuelve a la posición inicial.
- Hacer *un-click* con el botón *derecho* encima de la baraja: se reparten todas las cartas, respetando el turno de cada uno.

- *Presionar* con el botón izquierdo sobre una carta, *arrastrar* hasta que esté encima de una fila y *soltar* el botón: si la carta se puede poner en esa fila (es seleccionable y pertenece al jugador que es su turno) la fila lo aceptará y si no, la carta vuelve a la posición inicial.
- Hacer *doble-click* sobre una carta y se intenta colocar automáticamente en alguna de las filas. La operación será satisfactoria si la carta es seleccionable y pertenece al jugador que es su turno.

Por otro lado, en este juego es importante la visualización de las cartas porque se solapan entre ellas. Greenfoot nos ofrece métodos para modificar las posiciones de objetos en 2D (coordenadas *x* e *y*) pero no en 3D, es decir, la coordenada *z* o de profundidad. Eso no quiere decir que no lo controle. Greenfoot controla dicha coordenada según el orden en que se han ido añadiendo los objetos al mundo. El objeto que más se ve es el último que se ha añadido y el objeto que más atrás (oculto) se encuentra es el primero. Por ello, cuando se quiere traer una carta más adelante que otras, ejecutaremos el método `reAdd()` de la clase `Card` para posicionar su profundidad.

Tampoco debemos olvidar que el entorno de desarrollo de Greenfoot permite *pausar* la ejecución del juego y *ejecutar* cualquier método de los objetos que tenemos en el mundo de Greenfoot. Una funcionalidad importante para analizar el estado de los objetos mientras estamos probando la ejecución del juego.

Para finalizar, comentar que para saber más de las clases y métodos ya implementados tenéis que consultar la documentación creada en JavaDoc, que se encuentra en el directorio /doc del proyecto.

2.1.2 Funcionalidad que tiene que implementar el alumno

1- Define el tipo de dato enumerado `Position` (ver Figura 5), con los cuatro puntos cardinales como valores por defecto y relaciónalos con los grados de rotación de Greenfoot para las imágenes.
2- Implementa el método `addPlayers()` de la clase `TableGame`, para que añada 4 jugadores en la mesa, posicionando cada una de ellos en una de las posiciones definidas por el enumerado `Position`. Para las imágenes de cada jugador utiliza 'ppl1.png', 'ppl2.png', 'ppl3.png' y 'ppl4.png del directorio 'images'.
3- Implementar los métodos indicados 'To-Do' de `Player`
4- Implementa el método `isSixHearts()` de la clase `Card`, que determina si la carta es un 6 de corazones o no.
5- Implementa el método `whoIsFirst()` de la clase `TableGame`, el cuál analiza las cartas de los jugadores y devuelve el jugador que iniciará la partida. El primer jugador que puede empezar a poner una carta es el que posee el 6 de corazones y debe empezar jugando con dicha carta.
6- Implementa el método `areAllRowsEmpty()` de la clase `TableGame`, que determina si todas las filas están vacías.
7- Implementa el método `setTurn()` de la clase `TableGame`, para que determine que jugador empieza la partida, establezca el turno y lo refleje en el marcador (`Board`).
8- Implementa el método `whereCanPlace()` de la clase `Player`, que dado una carta analiza en que fila puede colocar la carta. Si puede ponerla en alguna fila, devuelve dicha fila y si no puede, devuelve null.
9- Implementa el método `placeOnArow()` de la clase `Player`, que dado una carta intenta colocarlo en una de las filas. Si se coloca se emite un sonido y devuelve True. Si no se puede poner, entonces incrementa los 'pasos' y devuelve False. Dicho método se ejecuta haciendo *doble-clik* encima de la carta de un jugador.

10- Desarrolla la jerarquía de clases que especializa la clase `Player`, tal cómo se indica en la siguiente figura, con el objetivo de poder definir jugadores automáticos que tengan distintas estrategias:

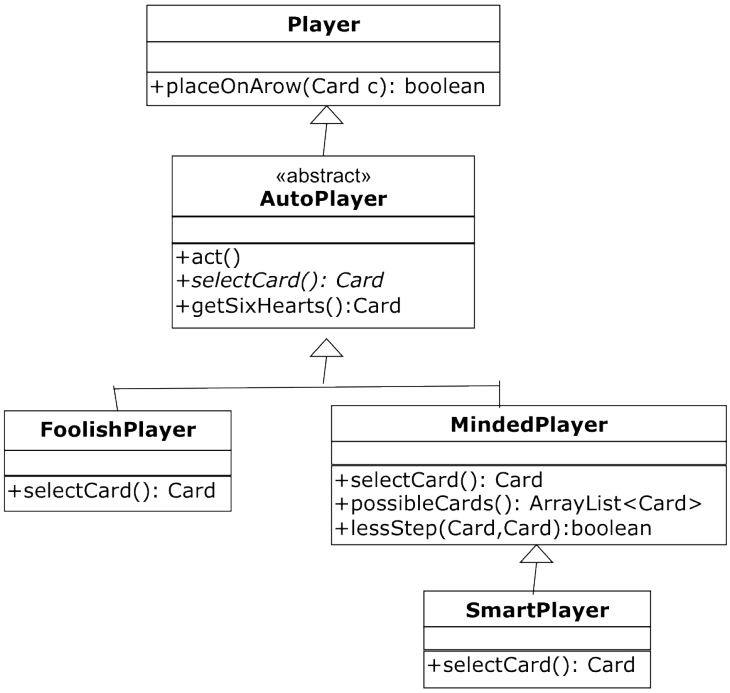

Figura 3. Detalle de los distintos tipos de jugador

a. La clase `AutoPlayer` describe un jugador automático:
   i. Implementa el método `getSixHearts()` que analiza todas las cartas y devuelve el 6 de corazones, en caso de que lo tenga en las manos y `null`, en caso contrario.
   ii. Implementa el método `act()` de forma que 1) comprueba si todas las cartas están repartidas y si es su turno, 2) si es así seleccionará una carta y la intentará poner en la fila correspondiente al palo de la carta. Recordar que independiente de la estrategia de cada tipo de jugador, el juego requiere que la primera carta siempre sea el 6 de corazones. Finalmente, se pasa el turno al siguiente jugador.
   iii. El método `selectCard()` define la estrategia del jugador, será un método abstracto.
b. La clase `FoolishPlayer` describe un jugador tonto, cuya estrategia para seleccionar una carta es escoger la primera (de izquierda a derecha) que puede ponerla. Esta estrategia se implementará en el método `selectCard()`.
c. La clase `MindedPlayer` describe un jugador que piensa, cuya estrategia es analizar que cartas puede poner, las memoriza y entre todas ellas escoge la que menos recorrido de cartas ofrece para los demás jugadores. Por ejemplo, entre el 5 de trébol o 4 de corazones, escoge el 4 de corazones. Esta estrategia se implementará en el método `selectCard()`.
   i. Implementa también el método `getPossibleCards()`, analiza si puede poner alguna de las cartas de la mano y devuelve el conjunto de cartas que realmente tiene opción de poner.
   ii. Implementa también el método `lessSteps()`, que dado dos cartas devuelve True si el primero tiene menos recorrido que el segundo. **Nota**: Tener en cuenta que de 13 cartas, la carta del medio es el 7 y no el 6.
d. Modifica el método `addPlayers()`, para que añada por lo menos un jugador de cada tipo.

11- Implementa el método `isFinished()` de la clase `TableGame`, que determina si el juego ha terminado o no.

12- Implementa los métodos `isTheWinner()`, `setTheWinner()` y `setTheWinner()` de la clase `TableGame`. El primero determina quién es el ganador, el segundo lo establece antes de que termine el juego y el tercero devuelve una cadena de caracteres con el nombre del ganador seguido del tipo de jugador, entre paréntesis. Por ejemplo, *player4(SmartPlayer)*.

2.1.3 Funcionalidad opcional (para puntuación extra)

13- Define la clase `SmartPlayer` para describir un jugador inteligente, cuya estrategia es analizar que cartas puede poner y entre todas ellas escoger la carta que más le conviene:
   a. Mira si posee una carta del mismo palo que puede ponerla más adelante, es decir, en el mismo recorrido creciente o decreciente.
   b. Si no posee ninguna que cumpla lo anterior, entonces sigue la misma estrategia que un jugador del tipo `MindedPlayer`.
   c. Modifica el método `addPlayers`, para que añada por lo menos un jugador de cada tipo.

14- Una vez que se haya comprobado que todos los jugadores automáticos funcionan correctamente, todos las cartas de los jugadores automáticos deberían de estar boca abajo, es decir, los demás jugadores no las deberían de ver. Implementa el método correspondiente.

3. LA BIBLIOGRAFÍA

1. Resumen de UML, ver 'Material complementario' en Moodle.
2. Greenfoot (v2.0). http://www.greenfoot.org
3. Documentación de las clases de Greenfoot. http://www.greenfoot.org/doc