Proyecto 1: Siete y medio

Introducción

El **siete y medio** es un juego de cartas que habitualmente utiliza la baraja española de 40 cartas. El juego consiste en obtener siete puntos y medio, o acercarse a esta puntuación lo más posible. Las cartas tienen, indistintamente de su palo, el valor que indica su propio índice, excepto las figuras (sota, caballo y rey) que tienen un valor de medio punto cada una, y los (ocho, nueve, diez) en caso de utilizar una baraja española de 48 cartas o una baraja de póker.

El objetivo es ganar los puntos apostados en cada ronda. Cada jugador compite contra la banca. Para ganar la apuesta, cada jugador ha de intentar sumar **siete y medio** o el número más cercano sin pasarse de esta cantidad superando a la banca.

Resumen del juego

- El número de jugadores debe estar entre 2 y 6.
- Se jugará un máximo de 30 manos.
- Cada jugador inicia la partida con 20 puntos.
- Inicialmente, se reparte una carta a cada jugador, que establecerá las prioridades de cada uno. El de mayor prioridad será la banca.
- Una vez establecidas las prioridades, se devuelven las cartas al mazo. Se establecen las apuestas de cada jugador.
- Seguidamente, el programa reparte una carta a cada jugador y comienza la primera ronda. Cada jugador decide si quiere recibir más cartas del mazo. Si no quiere más cartas, se planta. Puede pedir tantas cartas del mazo como crea conveniente y se puede plantar cuando quiera.
- Cuando todos han acabado de escoger cartas, juega la banca, si la banca no se pasa, se compara la puntuación de la banca con la de cada jugador.
    - Si el jugador tiene igual de puntos o menos que la banca, paga lo apostado a la banca.
    - Si el jugador se ha pasado, igualmente paga lo apostado a la banca.
    - Si el jugador tiene siete y medio y la banca no, gana el doble de lo apostado y pasará a ser uno de los candidatos a la banca.
    - En caso de no tener siete y medio, pero tener más puntos que la banca, la banca le paga lo apostado.

- Si la banca se pasa, pagará a todos los jugadores que no se hayan pasado, lo que hayan apostado.
- El jugador que pierde todos sus puntos, queda eliminado de la partida. Los otros pueden seguir jugando más rondas.
- Una vez realizado los pagos, si la banca no ha sacado siete y medio, y hay candidatos a la banca, el jugador con más prioridad de entre los candidatos, pasará a ser la nueva banca.
- En caso de quedar eliminada la banca, si no hay candidatos a la banca, el jugador con más prioridad pasará a ser la nueva banca.
- El jugador ganador es el que más puntos ha obtenido después de todas las rondas jugadas.

Desarrollo del juego

Habrá un máximo de 6 jugadores que se establecerán antes de empezar la partida.

Podrán jugar tanto jugadores humanos como jugadores «máquina»

Se establecerá una baraja de entre las posibles para jugar.

Cada jugador tiene un perfil de riesgo del tipo:

- «atrevido» que se cuantifica como 50
- «normal» que se cuantifica como 40
- «prudente» que se cuantifica como 30

El perfil de un jugador definirá qué cantidad de puntos apuesta en función de los que le quedan, y también lo que arriesgará a la hora de pedir una nueva carta.

Se reparten 1 carta del mazo a cada jugador para establecer prioridades. El jugador que consiga la carta más alta es el que tiene mayor prioridad. En caso de que 2 o más jugadores saquen la carta más alta, la prioridad es:

1. Oros
2. Copas
3. Espadas
4. Bastos

En caso de jugar una baraja de póker, por ejemplo, la prioridad de las cartas se establecen según los equivalentes a oros, copas, espadas, bastos.

El jugador con mayor prioridad será la banca, y los turnos irán en función de la prioridad, primero el jugador con menor prioridad, luego el siguiente con menor prioridad, y por último la banca.

Cada jugador inicia la partida con 20 puntos.

Habrá un máximo de 30 rondas.

Antes de repartirse ninguna carta, se establecen las apuestas para cada ronda, cada jugador apuesta según su perfil. Hay que tener en cuenta que un jugador nunca apostará más puntos de los que tiene la banca, dado que sería incoherente.

Por turnos, según prioridad, cada jugador en cada ronda:

- Al iniciar la ronda se le ha repartido una carta.
- Seguidamente, debe decidir si desea recibir más cartas del mazo.
    - Si no lo desea, debe indicarlo diciendo que se planta.
    - Si por el contrario, desea cartas para intentar acercarse lo más posible a sumar **siete y medio**, podrá pedir todas las que quiera de una en una, pudiéndose plantar cuando quiera.
- El último jugador en cada ronda es la banca.

Juega la banca

Una vez hayan hecho las apuestas todos los jugadores, le llega el turno a la banca.

- Si quedara algún jugador que no se hubiese pasado de siete y medio, y, por lo tanto, está todavía en condiciones de poder ganar su apuesta, la banca procederá a su vez a jugar.

La banca no hace apuestas, simplemente recibe las de los jugadores. Juega como los demás jugadores:

- plantándose (si cree que así gana a todos o algunos de los jugadores que quedan)
- dándose cartas, de una en una, pero con todas sus cartas boca arriba hasta decidir plantarse.

- Si la banca ve que no supera a suficientes jugadores y que puede quedarse sin puntos (eliminada) se verá obligada a pedir carta, sea cual sea el riesgo de pasarse.

Reglas para repartir los puntos apostados

- La *banca* juega contra todos y cada uno de los jugadores, y, por lo tanto, si ella se ha pasado, deberá pagar su apuesta a todos aquellos jugadores que se hubieran plantado sin pasarse.
- Si la *banca* se ha plantado, comprueba con cada jugador su jugada para ver a quién vence y con quién pierde. En cada apuesta vence quien más se acerque a siete y medio.
- En caso de empate gana la *banca*; por lo tanto, si la banca tiene siete y medio, gana automáticamente a todos los jugadores.
- La *banca* debe pagar la cantidad apostada, a cada jugador con el que pierda, y a la inversa, cada jugador que pierda con la *banca* debe pagarle a esta lo apostado.
- Si un jugador tiene siete y medio (y la *banca* no) cobra el doble de lo apostado y además toma pasa a ser candidato a banca la ronda siguiente.
- En caso de que haya más de un jugador con siete y medio, y la *banca* NO, el jugador con más prioridad de entre los que tienen siete y medio pasará a ser la *banca*.
- El jugador que pierde sus puntos queda eliminado de la partida.
- Si la *banca* queda eliminada y ningún jugador tiene siete y medio, el jugador con más prioridad pasará a ser la *banca*.
- Si la *banca* ha de pagar a varios jugadores, pero no tiene suficientes puntos para pagar a todos, irá realizando los pagos que pueda según la prioridad de los jugadores. Es decir:
    - Primero paga al jugador con más prioridad, luego al segundo, y así sucesivamente mientras tenga puntos para pagar. Es posible que el último jugador al que tenga que pagar no reciba los puntos que debería (la banca no tiene más puntos).
    - Si la banca se queda sin puntos, quedará eliminada y se elegirá una nueva banca:
        - o bien entre los jugadores que tengan 7.5, si hay alguno,

        - o bien entre todos si no hay ningún jugador con 7.5, siempre en función de la prioridad de los jugadores.
- La partida continúa hasta que:
    - un único jugador se hace con todos los puntos en juego, quedando los demás eliminados
    - o bien hasta que se disputa el máximo de manos fijadas para la partida, que puede ser hasta un máximo de 30, en cuyo caso el vencedor es quien acumula mayor cantidad de puntos al final de la partida.
- La cantidad de puntos a apostar, salvo en el caso de un jugador humano, que podrá establecer la apuesta manualmente, se realizará en función del perfil de jugador.

Lógica de pedir cartas

Cuando un jugador decide pedir una carta, lo hará en función de las cartas que hay repartidas, y lo hará calculando la probabilidad de pasarse si recibiese una nueva carta:

- Si un jugador tiene por ejemplo un cinco, calculará todas las posibles cartas que supongan pasarse de 7 y medio, así como el número total de cartas que quedan por salir.
- La división entre el número de cartas que supongan pasarse de 7 y medio y el número de cartas que quedan por salir multiplicado por 100 nos da la probabilidad de pasarnos de 7 y medio. Si quedan 10 cartas que suponen pasarnos de 7 y medio y un total de 20 cartas por salir, esta probabilidad sería (10/20)*100 = 50%.

Si esta probabilidad supera el perfil de riesgo del jugador (prudente–30, normal–40, arriesgado-50) se plantará. Salvo en el caso de:

- un humano que juegue manualmente
- la *banca*, que está obligada a pedir cartas si ve que puede quedar eliminada o no gana a ningún jugador.

La *banca* sigue el siguiente criterio:

- Si ya tiene 7.5 no pedirá más cartas.
- Si ya se ha pasado, no pedirá cartas.
- Si no tiene 7.5, pero después de repartir puntos se quedase sin puntos, pedirá carta.

- Si ya gana a todos los jugadores, no pedirá carta.
- Si no supera a ningún jugador, pedirá carta.
- Si ya supera a algún jugador, y con esta situación no se queda sin puntos, pedirá carta según su perfil de riesgo.

Especificaciones funcionales

M01 Sistemas Operativos

- Utilizar una cuenta de Azure. (Con la cuenta de uno del grupo ya estaría)
- Alojar la Base de Datos en Azure.

M02 Bases de datos

- La aplicación tiene que ser capaz de mostrar distintos informes conectándose a la base de datos de Azure.
- La BD tiene que tener una vista de ranking con la siguiente información de cada jugador:
    - ganancias obtenidas:
        - suma de los puntos que ha conseguido, lo que gana en cada partida es la diferencia entre los puntos al finalizar la partida y los que tenía al inicio.
    - partidas jugadas :
        - el total de partidas en que ha participado.
    - minutos jugados :
        - el total de minutos que ha invertido, la suma de los minutos de las diferentes partidas en que ha participado..

  Esta vista servirá para la query que usemos en la opción de menú que muestra el Ranking.

- Necesitamos sacar informes, se han de realizar un **mínimo de 7**, a escoger entre las siguientes propuestas:

Núm	Propuesta	Dificultad
1	Carta inicial más repetida por cada jugador, solo se tienen en cuenta jugadores que hayan jugado un mínimo de 3 partidas. *Los datos a mostrar de cada jugador son*: identificador jugador palo carta inicial más repetida carta inicial mas repetida número de veces que se ha repetido total de partidas que ha jugado	ALTO
2	Jugador que realiza la apuesta más alta en cada partida, es el que haya hecho la apuesta más alta en alguna de las rondas de la partida. *Los datos a mostrar de cada partida son:* identificador partida identificador jugador apuesta más alta	MEDIO
3	Jugador que realiza la apuesta más baja por partida. *Los datos a mostrar de cada partida son:* identificador partida identificador jugador apuesta más baja	MEDIO
4	Porcentaje de rondas ganadas por jugador en cada partida (%), así como su apuesta media de la partida. Un jugador gana en una ronda cuando es el que obtiene más puntos (mayor diferencia entre los puntos al comenzar y al finalizar la ronda). PUEDE QUE EN UNA PARTIDA UN JUGADOR,que participe en ella, NO GANE NINGUNA RONDA, también se ha de mostrar. *Los datos a mostrar son:* identificador partida identificador jugador rondas partida apuesta media rondas ganadas jugador porcentaje ganadas por jugador	ALTO

5	Lista de partidas ganadas por Bots. El ganador es quien ha conseguido más puntos al finalizar la partida, los puntos conseguidos son la diferencia entre los puntos que tenía al iniciar la partida y los puntos al acabar. *Los datos a mostrar son:* identificador partida (ganada por bot) puntos ganados	ALTO
6	Cuantas rondas gana la banca en cada partida, PUEDE QUE NO GANE NINGUNA y también se ha de mostrar. *Los datos a mostrar son:* identificador partida rondas ganadas banca	ALTO
7	Cuántos usuarios han sido la banca en cada partida. *Los datos a mostrar son:* identificador partida cantidad usuarios banca	BAJO
8	Calcular la apuesta media por partida. *Los datos a mostrar son:* identificador partida apuesta media	BAJO
9	Calcular la apuesta media de la primera ronda de cada partida. *Los datos a mostrar son:* identificador partida apuesta media (primera ronda)	MEDIO
10	Calcular la apuesta media de la última ronda de cada partida. *Los datos a mostrar son:* identificador partida apuesta media (última ronda)	MEDIO

- Estos informes han de ser mostrados por la pantalla del terminal y ser guardados en un fichero xml con el formato correspondiente.

M03 Programación

El juego empezará con un menú muy básico:

1) Add/Remove/Show Players
2) Settings
3) Play Game
4) Ranking
5) Reports
6) Exit

La primera opción nos servirá para poder crear jugadores, tanto humanos como «bots», y para añadir o eliminar jugadores que ya han sido creados y, por tanto, están en BBDD.
Una partida no puede empezar si no hay mínimo 2 jugadores.

La segunda opción nos llevará a un menú donde podremos establecer: Los jugadores participantes en la partida, la baraja de cartas que se va a utilizar (en función de las que tengamos en la BBDD) y el máximo de rondas.

1)Set Game Players
2)Set Card's Deck
3)Set Max Rounds (Default 5 Rounds)
4)Go back

Como mínimo deberemos poder elegir entre dos barajas diferentes y tendremos al menos 6 jugadores en la BBDD.

La tercera opción empezará la partida, siempre y cuando haya al menos 2 jugadores y se haya establecido la baraja con la que se va a jugar.

La cuarta opción nos llevará a un menú donde podremos ver el ránking de jugadores segun la opción que escojamos:

1)Players With More Earnings
2)Players With More Games Played
3)Players With More Minutes Played

La quinta opción nos servirá para ver una serie de reportes que conllevarán consultas a la BBDD.
Cuando empiece la partida, los jugadores humanos tendrán la opción de:

- ver sus stats
- ver los stats de todos los jugadores
- cambiar su apuesta
- jugar en modo automático (como si fuese un bot)
- pedir una nueva carta y plantarse.

M04 Lenguajes de marcas

- **Millora**: Se les dará un fichero XML con la información de las cartas que se leerá la primera vez que se repartan cartas, a partir de ese momento se trabajará con las cartas en "memoria"
- **Millora**: Se tiene que generar los archivos DTD que valida el fichero XML anterior
- **Obligatori**: Desarrollar el portal del juego con las siguientes páginas web:
  - index.html - Se explican las reglas del juego
  - tutorial.html - Se explica como usar el juego
  - equip.html - Se listan los participantes y que ha hecho cada uno
  - Documentacion_programacion.html

    Se ha de documentar con todo tipo de detalle, en formato html, cómo se han resuelto los problemas siguientes, (Evaluación en M3: 2 puntos).:

    - ➔ Lógica de juego de la banca.
    - ➔ Lógica del juego de un boot.
    - ➔ Establecimiento de la prioridad inicial de los jugadores.
    - ➔ Inserción en BBDD de las tablas cardgame, player_round, player_round_game.

- Condiciones:
  - Todas las páginas tienen un menú para cambiar de página
  - Todas las páginas tienen el logotipo del juego
  - Las páginas deben tener un mínimo de dos fotografías
  - Las páginas se deben adaptar a móvil (sin scroll lateral)
  - En global debe haber 2 al menos 2 transiciones y 2 animaciones
  - La estética debe ser profesional según las tendencias actuales
  - El CSS debe estar en archivos aparte de los .html
  - El posicionamiento debe hacerse con FlexBox

M05 Entorns de desenvolupament

- Se realizará un diagrama de flujo del proyecto, de la opción principal: "3) Play Game". La herramienta para hacerlo puede ser una elección personal, aunque se recomienda Draw.io por su integración con Drive y facilidad de uso.
- Control de versiones con Github:
  - El proyecto debe estar gestionado desde el primer día en Github.
  - El proyecto tendrá un fichero "README.md" con la definición y las instrucciones para la instalación y utilización del proyecto. Así como también la información de contacto de sus autores (email, twitter, etc).
  - Cada alumno creará y trabajará en su propia rama de trabajo. La rama de trabajo llevará su nombre.
  - Cada alumno hará al menos 5 commits al proyecto de Github.

Especificaciones no funcionales

M02 Bases de datos

- A partir del diagrama entidad-relación, generar el diagrama-relacional correspondiente, implementarlo con MYSQL-WORKBENCH.
- La Base de datos ha de contener todo lo necesario para su consistencia (PK , FK, tipo de los datos).
- Almacenar los DML de creación de las tablas, en una carpeta del proyecto.
- Algunos detalles del modelo:
  - La base de datos ha de ser capaz de almacenar los datos de todos los jugadores de las partidas.
    - id_jugador, nombre, nivel de riesgo
    - los jugadores pueden ser humanos o bots.
  - Para la partida se tienen que guardar los siguientes datos.
    - hora inicio y fin de partida
    - el número de jugadores.
    - el número de rondas
    - baraja que se ha usado, por defecto será la española
  - Se deben guardar todas las cartas de la baraja, con sus valores reales y

los valores del juego, así como el tipo(palo de la baraja) que son. En este juego se usa por defecto la baraja española, pero al iniciarse el juego puede escogerse otra baraja.

- Se deben guardar las rondas de cada partida con los datos de las mismas.
  - Puntos del jugador al inicio de la ronda.
  - Puntos del jugador al final de la ronda, si ha ganado sumará puntos, si ha perdido restará.
  - Apuesta que realiza el jugador (excepto si el jugador es la banca)

M03 Programación

Estructuras de datos y variables importantes en el juego:

Utilizaremos las siguientes estructuras de datos en el juego:

```
cartas = {
"001": {"literal": "As de Oros", "value": 1,  "priority": 4, "realValue": 1},
"002": {"literal": "Dos de Oros", "value": 2,  "priority": 4, "realValue": 2},
"003": {"literal": "Tres de Oros", "value": 3,  "priority": 4, "realValue": 3},
.....}
```

Es decir, un diccionario para las cartas, cuyos keys son del tipo:

- O01 --> 1 de oros; B12 --> 12 de bastos, E05 --> 5 de espadas ... etc

Como datos, un nuevo diccionario, donde tenemos el nombre literal de la carta, el valor de la carta, el valor de la carta en el juego y la prioridad (oros 4, copas 3, espadas 2, bastos 1).

La prioridad nos da la opción de desempatar cartas que tienen el mismo valor.

Este diccionario nos servirá para todo lo relacionado con las cartas.

```
players = {
"11115555A":{"name":"Mario","human":True,"bank":False,"initialCard":"","priority":0
,"type":40,"bet":4,"points":0,"cards":[],"roundPoints":0},
"22225555A":{"name":"Pedro","human":True,"bank":False,"initialCard":"","priority":0
,"type":40,"bet":4,"points":0,"cards":[],"roundPoints":0},
...}
```

Es decir, un diccionario donde sus keys son los NIF's de los jugadores disponibles almacenados en BBDD y como valor un nuevo diccionario con los elementos:

- nombre

- humano del tipo booleano
- banca del tipo booleano
- carta inicial, que será la que determine la prioridad de los jugadores.
- Prioridad, que vendrá determinada por la carta inicial.
- Tipo que será 50 en el caso atrevido, 40 en el caso normal y 30 en el caso prudente.
- Apuesta
- Puntos, representa los puntos de cada jugador en la partida.
- Cartas, que será una lista donde almacenaremos los id's de cartas en cada turno.
- Puntos ronda, que serán los puntos conseguidos durante el turno.

En este diccionario tendremos almacenados todos los jugadores que se hayan creado en algún momento y ya estén en BBDD y nos servirá para gestionar todo los relacionado con los jugadores.

Crearemos una lista, por ejemplo game=[], donde tendremos los NIF de todos los jugadores que participen en la partida en cada momento.

Crearemos una lista, por ejemplo mazo=[], donde tendremos todos los id's de las cartas que componen el mazo en cada momento.

Crearemos un diccionario, por ejemplo context_game={}, donde tendremos una serie de variables de contexto a las que podamos acceder desde cualquier sitio.

Por ejemplo context_game[«game»] = lista de jugadores en la partida actual

context_game[«round»] = ronda actual de la partida.

Tal y como indica su nombre, este diccionario nos será de utilidad para tener, de forma ordenada, variables que pueden ser de tipo global.

Con estas variables, ya podemos gestionar prácticamente todos los aspectos del juego.

Algunas sugerencias:

Para la inserción de datos en BBDD, sería conveniente crearse un diccionario para cada una de las tablas que tengamos que actualizar durante el juego.

Por ejemplo (en rojo las claves):

cardgame = {'cardgame_id': id de partida, 'players': Numero de jugadores, 'start_hour':Hora de inicio de artida (datetime), 'rounds': Número de rondas, 'end_hour': hora final de partida (datetime) }

player_game = {id_game:{id_player_1:{initial_card_id:"card id", starting_points:"puntos al inicio", ending_points:"puntos al final de partida},...,id_player_n:{initial_card_id:"card id", starting_points:"puntos al inicio", ending_points:"puntos al final de partida}}"

player_game_round = {round:{id_player_1:{is_bank:"0 ó 1",bet_points:"apuesta en la ronda",starting_round_points:"puntos al inicio de la partida,cards_value:"puntos obtenido en la actual ronda",endind_round_points:"puntos al final de la ronda"},...,{id_player_n:{is_bank:"0 ó 1",bet_points:"apuesta en la ronda",starting_round_points:"puntos al inicio de la partida,cards_value:"puntos obtenido en la actual ronda",endind_round_points:"puntos al final de la ronda"}

Funciones que os serán útiles para el desarrollo del proyecto:

def playGame():

Esta es la función principal del proyecto. Una vez establecido el número de rondas, la baraja con la que se va a jugar, y los jugadores que participan en la partida, ésta será la función que gestione toda la partida. Para ello, hará uso de otras funciones auxiliares como:

> def setGamePriority(mazo):
>
> Esta función establece las prioridades de los jugadores.
>
> Se recibe una lista con los id's de la baraja (mazo), se mezclan, se reparte una carta a cada jugador, se ordenan la lista de jugadores de la partida (contextGame["game"]) según la carta que han recibido, y se establecen las prioridades.
>
> def resetPoints():
>
> Función que establece los 20 puntos iniciales en todos los jugadores.
>
> def fill_player_game(player_game,gameID,*fields):
>
> Función para insertar datos en el diccionario player_game

def fill_player_game_round(player_game_round,round,*fields):

Función para insertar datos en el diccionario player_game_round

def checkMinimun2PlayerWithPoints():

Función que verifica que al menos haya dos jugadores con puntos.

def orderAllPlayers():

Función que ordena los jugadores de la partida (contextGame[“game”]) de forma que pone la banca al principio y el resto de jugadores después, ordenados según prioridad

def setBets():

Función que establece las apuestas de cada jugador en función del tipo de jugador.

def standarRound(id, mazo):

Función que realiza la tirada de cartas de un jugador en función del tipo de jugador que es y teniendo en cuenta si el jugador es banca o no.

def humanRound(id, mazo):

Función que gestiona la tirada de un jugador humano. Nos muestra el menú de opciones:

1)View Stats

2)View Game Stats

3)Set Bet

4)Order Card

5)Automatic Play

6)Stand

Option:

Y ejecuta la acción que elijamos

def distributionPointAndNewBankCandidates():

Función que realiza el reparto de puntos una vez finalizada una ronda y devuelve una lista con los candidatos a la banca (los que tienen 7,5)

def printStats(idPlayer="", titulo=""):

Esta función nos imprime los stats de todos los jugadores de la partida:

```
*******************************************************************************************************************************************
                                    _____                          ___              __   __  __      ______
                                   / ___/___ _   _____  ____      /   |  ____  ____/ /  / / / /___ _/ / __/
                                   \__ \/ _ \ | / / _ \/ __ \    / /| | / __ \/ __  /  / /_/ / __ `/ / /_
                                  ___/ /  __/ |/ /  __/ / / /   / ___ |/ / / / /_/ /  / __  / /_/ / / __/
                                 /____/\___/|___/\___/_/ /_/   /_/  |_/_/ /_/\__,_/  /_/ /_/\__,_/_/_/

********************************************************* Round 0 , Turn of Mario **********************************************************

Name                Boot2                                   Boot1                                   Mario
Human               False                                   False                                   True
Priority            1                                       2                                       3
Type                30                                      40                                      40
Bank                False                                   False                                   True
Bet                 6                                       8                                       8
Points              20                                      20                                      20
Cards               001;010;B02                             E07                                     C07
Roundpoints         3.5                                     7.0                                     7.0
```

def orderPlayersByPriority(listaJugadores):

Ordenamos la lista de jugadores de la partida (contextGame[“game”]) según prioridad.

def printWinner():

Función que muestra el ganador de la partida:

```
**********************************************************************************************************************************************************
                    .d8888b.                                    .d88888b.
                   d88P  Y88b                                  d88P" "Y88b
                   888    888                                  888     888
                   888         8888b.  88888b.d88b.   .d88b.    888     888 888  888  .d88b.  888d888
                   888  88888     "88b 888 "888 "88b d8P  Y8b   888     888 888  888 d8P  Y8b 888P"
                   888    888 .d888888 888  888  888 88888888   888     888 Y88  88P 88888888 888
                   Y88b  d88P 888  888 888  888  888 Y8b.       Y88b. .d88P  Y8bd8P  Y8b.     888
                    "Y8888P88 "Y888888 888  888  888  "Y8888     "Y88888P"    Y88P    "Y8888  888


**********************************************************************************************************************************************************
                              The winner is 11115555A - Mario, in 2 rounds,  with 44 points
                              Enter to continue
```

def insertBBDDCardgame(cardgame):

Función que guarda un nuevo registro en la tabla cardgame.

Esta función debería llamarse justo después de acabar una partida.

def insertBBDD_player_game(player_game,cardgame_id):

Función que guarda en la tabla player_game de la BBDD el diccionario player_game.

Esta función debería llamarse justo después de acabar una partida

def insertBBDD_player_game_round(player_game_round,cardgame_id):

Función que guarda en la tabla player_game_round de la BBDD el diccionario player_game_round.

Esta función debería llamarse justo después de acabar una partida.

Una posible estrategia para esta función sería:

Establecer prioridades de los jugadores

Resetear puntos

Crear diccionarios cardgame,player_game,player_game_round

Crear un id de partida

Mientras hayan dos jugadores o más con puntos, y no nos pasemos del máximo de rondas:

ordenar jugadores, banca al final y resto de prioridad menor a mayor.

Crear una lista con los id's de cartas (mazo).

Barajar el mazo.

Establecer apuestas

Ejecutar jugadas de cada jugador.

Repartir puntos.

Eliminar los jugadores sin puntos.

Establecer nueva banca si es necesario.

Insertar en BBDD los diccionarios creados para tal propósito.

Mostrar el ganador.

def getOpt(textOpts="", inputOptText="", rangeList=[], exceptions=[]):

Función para la gestión de menús. Le pasamos un texto, que nos mostrará un menú, un rango de opciones válidas, y una lista de excepciones, y nos devuelve la opción elegida por el usuario.

def orderPlayersByPoints(listaJugadores):

Función que ordena los jugadores según sus puntos.

def chanceExceedingSevenAndHalf(id, mazo):

Función que calcula la probabilidad de pasarse de siete y medio

def printPlayerStats(id):

Esta función nos muestra los stats de un jugador humano.

name Mario

type 40

human True

bank True

initialCard E11

priority 3

bet 8

points 20

cards C07

roundPoints 0

def logToFile(text):

```
f = open("logfileSevenAndHalf.txt", "a")
f.write(text)
f.close()
```

Esta función nos puede servir para enviar mensajes de texto al archivo “logFileSevenAndHalf”, que puede sernos útil a modo de debug.

def baknOrderNewCard(id, mazo):

Función que evalúa si la banca pedirá una nueva carta.

def newPlayer(dni, name, profile, human):

Función que devuelve una tupla con dos elementos, el primero es el dni del nuevo jugador, el segundo, un diccionario con las claves: name, human, bank, initialCard, priority, type, bet, points, ards, roundPoints

def addRemovePlayers():

Función que nos muestra el menú despues de escoger la opción 1 del menu principal:

1)New Human Player

2)New Boot

3)Show/Remove Players

4)Go back

Option:

def settings():

Función que gestiona el menú settings, donde podemos establecer los jugadores que participarán en una partida, la baraja con la que se va a jugar y el número máximo de rondas.

def setMaxRounds():

Función que pide al usuario el número de rondas de la siguiente partida y lo establece en el diccionario contextGame, contextGame["maxRounds"]

def newRandomDNI():

Función que devuelve un dni válido con números aleatorios

def setNewPlayer(human=True):

Función que gestiona la creación de un nuevo jugador que insertaremos en la BBDD

def showhPlayersGame():

Función que muestra los jugadores seleccionados cuando estamos añadiendo jugadores a la partida:

```
******************Actual Players In Game******************
11111111A   Boot1              Boot      Moderated
22222222A   Boot2              Boot      Cautious
11115555A   Mario              Human     Moderated
```

def setPlayersGame():

Función para establecer los jugadores que conformarán la partida siguiente

def removeBBDDPlayer():

Función que nos muestra los jugadores disponibles en BBDD, y elimina el que seleccionemos

def printStats(idPlayer="", titulo=""):

Esta función nos imprime los stats de todos los jugadores de la partida:

```
*****************************************************************************************************************************************
                                   _____                        ___              __   __  __      ______
                                  / ___/___ _  _____  ____    /   |  ____  ____/ /  / / / /___ _/ / __/
                                  \__ \/ _ \ | / / _ \/ __ \  / /| | / __ \/ __  /  / /_/ / __ `/ / /_
                                 ___/ /  __/ |/ /  __/ / / / / ___ |/ / / / /_/ /  / __  / /_/ / / __/
                                /____/\___/|___/\___/_/ /_/ /_/  |_/_/ /_/\__,_/  /_/ /_/\__,_/_/_/

******************************************************** Round 0 , Turn of Mario *********************************************************

Name                Boot2                                   Boot1                                   Mario
Human               False                                   False                                   True
Priority            1                                       2                                       3
Type                30                                      40                                      40
Bank                False                                   False                                   True
Bet                 6                                       8                                       8
Points              20                                      20                                      20
Cards               001;010;B02                             E07                                     C07
Roundpoints         3.5                                     7.0                                     7.0
```

def reports():

Función que nos muestra el menú de reportes, y una vez elegida una opción, el reporte correspondiente

def getPlayers():

Función que extrae los jugadores definidos en la BBDD y los almacena en el diccionario contextGame[“players”]

def setCardsDeck():

Elegimos una baraja, y a partir de esa baraja, establecemos el diccionario de cartas contextGame["cards_deck"]

def savePlayer(nif,name,risk,human):

Función que guarda en BBDD un nuevo jugador.

def delBBDDPlayer(nif):

Función que elimina un jugador de la BBDD

def getGameId():

Función que devuelve un id no existente en la tabla cardgame.

def getBBDDRanking():

Función que crea la vista player_earnings, y retorna un diccionario con los datos de ésta,

player_id | earnings | games_played | minutes_played.

def ranking():

Función queMuestra el menú del ranking y el ranking según la opción elegida

def returnListRanking(field="earnings"):

Función que retorna una lista con los id de jugadores del diccionario que retorna la función getBBDDRanking(), ordenados según la opción del ranking elegida

M04 Lenguajes de marcas

- El fichero XML de configuración ha de tener el siguiente formato:
  - Tag Raíz: Inital_Config
  - Tag Clave de configuración: “nombre de la clave”
- El fichero de configuración ha contener las siguientes claves:
  - Num_Min_Players
  - Num_Max_Players
  - Num_Max_Rounds
  - Num_Initial_Points

- Is_Allowed_Auto_Mode

- El fichero de configuración se ha de llamar: Basic_Config_Game.xml
- El fichero XML de cartas debe tener el siguiente formato:
  - Tag Raíz: Cards
  - Tag Carta: Card (Cada card contendrá la siguiente información)
    - Tag Identificador: Id
    - Tag Valor: Value
    - Tag tipo (palo): Suit
    - Tag Carta activa: Is_Active
- El fichero XML con el resultado de veces jugadas debe ser:

  <players_list>

  Raíz que contiene la lista de veces jugadas con elementos (user)

  <player>

  Elemento que contiene la información de una fila

  <player_id>

  Identificador del jugador

  <top_initial_card>

  Carta inicial que s'ha tret més cops

  <times_initial_card>

  Cops que ha sortit la carta "top"

  <times_played>

  Cops que s'ha jugat (ha de ser superior o igual a 3)

M05 Entorns de desenvolupament

- La rama principal debe llamarse "main".
- Existirá una rama llamada "pre-producción", donde se harán los "merge" necesarios antes de subir cambios a la rama principal.
- La entrega del proyecto se hará en GitHub a través de una "release". La llevará a cabo el propietario del repositorio.
- También debe entregarse en la tarea del moodle de M5 habilitada para el proyecto:
  - un fichero con la imagen del Diagrama de flujo.

- la URL del repositorio GitHub del proyecto.

Gestión y fechas del proyecto

9 de Enero al 22 de Enero incluido (A partir del 23 presentaciones.)

Normativa y criterios de evaluación/corrección

Cada uno de los módulos participantes en el proyecto tendrá una nota, la nota final del proyecto quedará ponderada según las horas de cada módulo. Si las horas de cada módulo son:

M1 - 4

M2 - 5

M3 - 7

M4 - 3

M5 - 2

Nota final primero proyecto M15 =

4/21 Nota M1 + 5/21 Nota M2 + 7/21 Nota M3 + 3/21 Nota M4 + 2/21 Nota M5

M01 Sistemas operativos

- Correcta conexión de la base de datos, 10 puntos.

M02 Bases de datos

Partiendo de una puntuación inicial de 10 (+1 punto adicional si los objetos de la base de datos: tablas, columnas.. están en inglés), se aplicarán las siguientes penalizaciones que restarán puntos a la puntuación inicial:

PENALIZACIÓN	puntos
NO tener tabla de cartas con las cartas de la baraja	2,0
NO tener tabla de jugadores, partidas y jugadores-partida, con la información de las partidas jugadas	2,0
NO tener tabla de rondas, con la información de las rondas jugadas en cada partida.	2,0
NO tener la vista para la consulta de ranking	2,0

NO tener implementados con éxito un **mínimo de 7 informes** de los 10 propuestos.

- cada informe no realizado, o que no funciona correctamente, se penaliza con 0,30 puntos. Màxima penalización 2,0.
- si se realizan más de 7 informes, cada informe extra sumará 0,45 puntos a la puntuación final.

M03 Programación

- Si el desarrollo de una partida es totalmente correcto (6 puntos).

  Se ha de poder al menos jugar una partida con un número de jugadores configurable, y la lógica del juego ha de ser correcta en todos los aspectos.

- Presentaciones:

  El formato de las presentaciones han de ser correcto, no han de ser como las del ejemplo que se os mostrará, pero funcionalmente han de ser equivalentes.(2 puntos).

- Documentación:

- Se ha de documentar con todo tipo de detalle, en formato html, como parte de M4, cómo se han resuelto los problemas siguientes:(2 puntos).
  - ➔ Lógica de juego de la banca.
  - ➔ Lógica del juego de un boot.
  - ➔ Establecimiento de la prioridad inicial de los jugadores.
  - ➔ Inserción en BBDD de las tablas cardgame, player_round, player_round_game.

M04 Lenguaje de marcas

- 5% Extra si se hace la mejora XML con información de las cartas
- 5% Extra si se hace la mejora de generar los DTDs
- 75% Páginas web
  - 10% menos si faltan páginas
  - 10% menos si no hay menú o no funcionan los enlaces

        - 10% menos si no hay 2 fotos en cada página
        - 10% menos si no se adaptan a móbil o tienen scroll lateral
        - 10% menos si faltan las animaciones o transiciones
        - 10% menos si mezcla CSS y HTML
        - 10% menos si no usa usa FlexBox
        - 5% menos si no tiene estética profesional
    - 25% Ficheros XML correctos
        - 10 % menos si falta algún elemento o clave de configuración
        - 10 % menos si falta algún elemento en las cartas
        - 5% menos si falta algún elemento de cartas jugadas

M05 Entornos de desarrollo

Siguiendo las indicaciones del apartado “Especificaciones no funcionales.M05”. Se valorará aplicando los siguientes criterios:

CRITERIO	%
Diagrama de flujo	70%
Uso de GitHub	30%