



UNIÓN EUROPEA  
Fondo Social Europeo  
El FSE invierte en tu futuro



**Junta de Andalucía**  
Consejería de Desarrollo Educativo  
y Formación Profesional



FORMACIÓN  
PROFESIONAL  
ANDALUZA

# UT1. Generación dinámica de páginas web

## Tarea 2: Juego de mesa

Desarrollo Web en Entorno Servidor – 2ºDAW

Luis Ramón López López - [luis.ramon.lopez@g.educaand.es](mailto:luis.ramon.lopez@g.educaand.es)

Enrique Casanova Gallardo - [ecasgal895@g.educaand.es](mailto:ecasgal895@g.educaand.es)

# Introducción

¿Qué se pretende con la tarea?



# Objetivos

- **Principal:** Crear una aplicación web que no sea la típica CRUD
- **Secundarios**
  - Implementar una jerarquía de clases que dé soporte a un juego de mesa de varios jugadores dado
  - Generar un modelo de datos que permita persistir la información que necesitan los jugadores y las partidas del juego de mesa
  - Integrar la mecánica del juego con lo anterior para producir una aplicación jugable

# ¿Por qué un juego de mesa?

- Porque las normas suelen estar bien definidas y ser ampliamente conocidas
- Porque se quiere que se permita la interacción entre varios usuarios, cada uno en su navegador



# Información común

A tener en cuenta para todos los juegos



# Idioma de la aplicación

- La aplicación (incluyendo nombres de los *scripts*, variables, clases, métodos, funciones y comentarios) se programará en idioma **inglés**
- El juego, cuando sea utilizado por el usuario final, tendrá toda su interfaz en idioma **inglés**



# Secciones de la aplicación

- Existen varias secciones de la aplicación que hay que definir:
  - **login.php** : Permite identificar al jugador que quiere entrar en la aplicación
  - **lobby.php** : Permite a un jugador autenticado seleccionar una partida ya existente o crear una nueva
  - **waiting.php** : Muestra el listado de jugadores apuntados a la partida actual mientras no esté completa
  - **???.php** : Implementa el juego en sí, dependiendo del juego concreto a implementar que se haya elegido

# *Script* de entrada

- El script por defecto, **index.php**, se comportará de manera distinta según el estado de la sesión:
  - Si aún no ha entrado ningún usuario, redirigir a **login.php**
  - Si hay un usuario autenticado pero aún no ha elegido partida activa, redirigir a **lobby.php**
  - Si hay una partida activa, redirigir al *script* correspondiente según el estado de la misma (dependerá de la opción elegida)



# Jugadores

- Todas las opciones deberán contemplar jugadores (*players*) previamente registrados en la base de datos
  - Nombre de usuario
  - Nombre completo
  - Contraseña
- Para poder entrar en la aplicación (*script **login.php***), habrá que introducir el nombre de usuario y contraseña de un jugador válido

# Lobby

- Una vez entre un jugador, se mostrará el *lobby*
- El *lobby* será una lista de partidas activas en las que está el jugador, a continuación otra lista de partidas que están esperando a un jugador (no activas) y, al final del todo, un botón que permitirá crear una nueva
- Cuando se seleccione la partida, se almacenará la misma en la sesión para que el resto de *scripts* sepan la cuál es la partida activa

# Partida

- Una partida puede estar activa o no
  - Se dice que una partida está activa cuando se han apuntado todos los jugadores y ha comenzado
  - Mientras no esté activa, se puede seleccionar por cualquier jugador que quiera participar en ella
- Una partida puede estar finalizada o no
  - Las partidas finalizadas no se muestran en el *lobby* ni se pueden seleccionar

# Activación de una partida

- Si se selecciona una partida que aún no tiene el número de jugadores adecuados, debe redirigir a **waiting.php**
- En el *script* se muestra la lista de jugadores actuales e indicar cuántos faltan
  - En este caso, la página se recargará cada 5 segundos
  - Usa **header("refresh: 5");** para que se recargue sola
- Cuando estén todos los jugadores, se marcará como activa y se redirigirá a **???.php** (depende del juego elegido)

# Seguridad de la partida

- Si se intenta acceder a cualquier *script* de la partida y tanto el jugador como la partida activa no están en la sesión, se eliminará la sesión y redirigirá al usuario a la pantalla de entrada





# Opción A: Ajedrez

Partida de ajedrez para dos jugadores

# Información

- Implementar una partida de ajedrez para dos personas
- Una vez los dos jugadores se hayan unido a la partida...
  - Se asignará blancas o negras a cada jugador de forma aleatoria
  - Se inicializará el tablero de la partida
  - Se activará la partida
- Por cada partida se almacenará en la base de datos
  - Qué jugador es cada color
  - El tablero (las piezas que hay colocadas en él)
  - Estado actual: blanco elige pieza, blanco elige destino, negro elige pieza o negro elige destino o finalizada
  - La posición de la pieza elegida el jugador actual (si está a la espera de elegir destino)

# Script del juego

- El *script* principal del juego mostrará el estado del tablero
- Si la partida ha finalizado, se mostrará un mensaje que indique el ganador (blancas, negras o tablas) y se ofrecerá un botón para salir de la partida y volver al *lobby*
- Si lo carga el jugador que no tiene el turno, se indicará con un mensaje que le toca esperar y recargará automáticamente cada 5 segundos

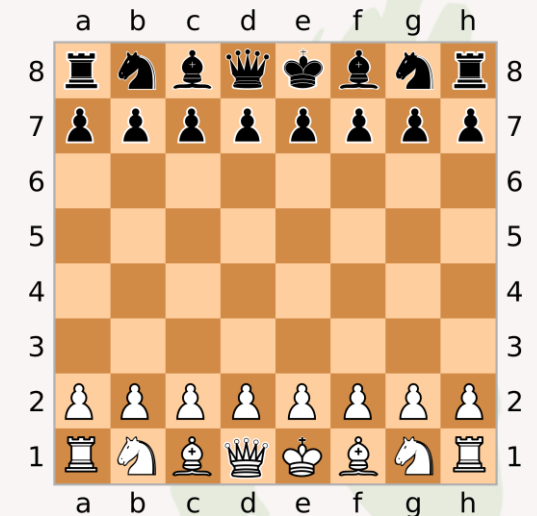
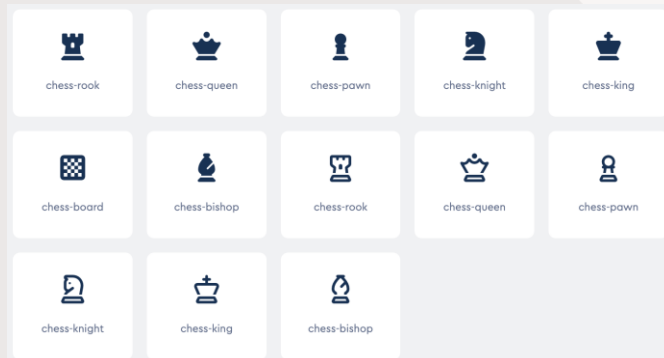


# *Script* del juego cuando se tiene el turno

- En la fase de “elige pieza”, las piezas que pueda mover se pondrán entre etiquetas <a> para que el usuario pueda hacer clic sobre ellas
  - Una vez elegida una ficha, se pasa a la fase “elige destino”
- En la fase de “elige destino”, las casillas a las que puede mover la pieza seleccionada se marcan con etiquetas <a>
  - Al hacer clic sobre una de ellas se procesa el movimiento
  - Si se detecta un ganador o tablas, se marca la partida como finalizada
  - Si no hay ganador o tablas, se pasa el turno al “elige pieza” del otro jugador

# Dibujo del tablero y las piezas

- Usa una tabla de 8x8
  - O de 9x9: Intenta dibujar también el número de cada fila y la letra de cada columna
- Se ofrecen varios ficheros con imágenes de las distintas piezas del ajedrez y de casillas blancas y negras
- Como alternativa, usa iconos de [FontAwesome](#)



# Consejos para implementar la lógica (I)

- Expresa la posición (fila y columna) como enteros entre 0 y 7
  - Row (fila) 1..8 → Entero de 0 a 7
  - Column (columna) A..H → Entero de 0 a 7
- Crea una clase **Board** para almacenar el tablero
  - Internamente almacena un *array* de 8x8 piezas. Si no hay ninguna en esa posición, contendrá **null**
  - Cuando se cree una instancia, coloca las piezas del tablero en su posición inicial creando los objetos correspondientes y colocándolos en el *array*
  - Implementa el método **getPieceAt(row, col)** que devuelva la pieza en esa posición o **null** si no hay ninguna
  - Implementa el método **setPieceAt(piece, row, col)** que almacene la pieza indicada ahí
  - Implementa el método **canMoveTo(color, row, col)** que compruebe si alguna de las piezas del color indicado puede moverse a la fila y columna indicadas

# Consejos para implementar la lógica (II)

- Crea una clase abstracta **Piece** que almacene el color (*white* o *black*) y el tablero (*board*) al que pertenece
  - Inicializa todos esos datos en el constructor y encapsula con *getters*
  - Para expresar el color, define dos constantes enteras dentro de la clase **WHITE** (0) y **BLACK** (1)
  - Define un método abstracto **canMove(fromRow, fromCol, toRow, toCol)** que devuelva **true** si la pieza puede moverse desde la fila y columna primeras a la fila y columna indicadas después
    - Cada tipo de pieza concreta implementará el cómo se hace
    - Si la posición no es válida, devolver **false**
    - Usa el tablero para saber qué hay en esa posición (puede ser un hueco, haber una pieza del mismo color o de otro color)

# Consejos para implementar la lógica (III)

- Crea una clase abstracta **Piece** que almacene el color (*white* o *black*) y el tablero (*board*) al que pertenece
  - Define un método abstracto **countMoves(fromRow, fromCol)** que devuelva el número de movimientos posibles de esa pieza si está en la posición indicada por los parámetros



# Consejos para implementar la lógica (IV)

- Crea tantas clases hijas de **Piece** como piezas hay en el ajedrez
  - **Pawn, Queen, King, Bishop, Knight y Rook**
- Implementa el método **canMoveTo(row, col)** en función de los movimientos que puedan hacer
  - Cuidado con el rey: no puede moverse a una casilla aunque esté vacía si al moverse allí está amenazada por una pieza del otro color

# Consejos para implementar la lógica (V)

- Para guardar el tablero en la base de datos, puedes serializar el tablero en una columna de tipo TEXT
  - **serialize(\$v)** : Devuelve una cadena que contiene el parámetro de forma serializada
- Para recuperar el objeto original, puedes usar el método **unserialize(\$cadena)** que devuelve el objeto reconstruido

# Reglas importantes

- Si el rey del jugador está amenazado (jaque o *check*) cuando comienza su turno, sólo puede mover el rey
- Si el jugador está en esa situación y no puede mover el rey a ningún otro sitio, se produce el jaque mate (*check mate*) y pierde la partida
- Cuando un jugador en su turno, sin estar en jaque, no puede realizar ningún movimiento legal la partida termina automáticamente en tablas (no hay ganador)



# Simplificaciones opcionales

- Si un peón llega al extremo del tablero, promociona automáticamente a reina (en las reglas originales, se puede elegir cualquier pieza salvo el rey)
- No es necesario implementar el enroque del rey con las torres
- No es necesario implementar el “comer al paso” de los peones
- No hay que implementar otras formas de terminar en tablas, como la de insuficiencia de material
  - Es decir, cuando es rey contra rey, rey y caballo contra rey, etc.

# Si eres “pro” y quieres ampliar

- Implementa todo lo que era opcional en los movimientos
  - Para el enroque tendrás que almacenar en la base de datos como columnas de la partida si se ha movido alguna vez: el rey negro, la torre izquierda negra, la torre derecha negra y lo mismo para las blancas (para saber si se puede hacer o no)
- Añade campos a la partida: comienzo turno (*timestamp*), total segundos blancas y total segundos negras
  - Cuando comienza el turno de un jugador, almacena el *timestamp* en la base de datos
  - Cuando selecciona el destino del movimiento, calcula la diferencia del *timestamp* actual respecto al almacenado en la partida y súmale esos segundos al total de su color
  - Muestra el total de cada jugador en formato MM:SS al acabar la partida



# Opción B: Dominó

Partida de dominó para entre 2 y 4 jugadores

# Información


- Implementar una partida de dominó para 2 a 4 personas
- Una vez los dos jugadores se hayan unido a la partida...
  - Si hay al menos dos jugadores, cualquiera de ellos podrá dar un botón en **waiting.php** para comenzar la partida
  - Si hay 4 jugadores, directamente se activará la partida sin esperar a nada más

# Información

## ■ Al iniciar la partida

- Se barajarán las 28 fichas (*tiles*). Al conjunto desordenado se le denomina pozo (*stock*)
- Se repartirán 7 fichas a cada jugador (si juegan 2 jugadores) o 5 fichas (si juegan 3 o 4)
- Comenzará el jugador que tenga el 6 doble; si no lo hay, el 5 doble y así sucesivamente. Si ninguno tiene dobles, empezará aquel que tenga la ficha más alta. El jugador que empieza se dice que “tiene la mano”
- Se sigue el orden normal después de él

# Información

- Por cada partida se almacenará en la base de datos
  - El orden de los jugadores
  - El jugador que tiene la mano (es decir, el que empieza la partida)
  - El jugador que tiene el turno actual 
  - La pieza que ha elegido el jugador que tiene la mano (si está a la espera de indicar dónde colocarla)
  - El pozo (las piezas que aún no se han repartido y que están barajadas)
  - Las fichas que hay desplegadas en el tablero, en el orden en que están conectadas
  - Las fichas que tiene cada jugador para elegir

# Script del juego

- El *script* principal del juego mostrará las fichas que ya se han colocado (recuerda que siempre habrá al menos una porque el jugador que tiene la mano coloca siempre la ficha que le ha dado el turno)
- Debajo aparecerán las fichas de todos los jugadores, en posición vertical
  - A la hora de dibujarlas, dibujar la parte de atrás (todo negro) si no son del jugador actual para permitir saber cuántas tiene pero no qué contienen
- Si la partida ha finalizado, se mostrará quién ha ganado y cuántos puntos ha conseguido. Haciendo clic a un botón, se le llevará de nuevo al *lobby*
- Si lo carga el jugador que no tiene el turno, se indicará con un mensaje que le toca esperar y recargará automáticamente cada 5 segundos

# Script del juego cuando se tiene el turno

- En la fase de “elige ficha”, las fichas que pueda colocar en la mesa se pondrán entre etiquetas <a> para que el usuario pueda hacer clic sobre ellas
  - Una vez elegida una ficha, si sólo puede colocarse a la izquierda o a la derecha, se coloca automáticamente y pasa el turno
  - Si la ficha puede colocarse en ambos lados, se pasa a la fase “elige posición”
- Si ninguna de las fichas que tiene pueden colocarse, el jugador que tiene el turno tendrá que robar un ficha del pozo hasta que salga alguna que pueda colocar
  - Si el pozo se queda vacío y aún así no puede colocar, su turno pasa automáticamente



# *Script* del juego cuando se tiene el turno

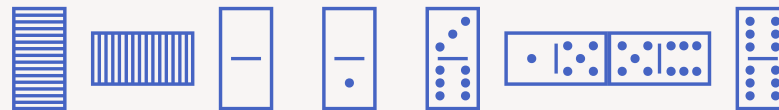
- En la fase de “elige posición”, se muestra la ficha elegida a ambos lados de las fichas ya colocadas en un color distinto y entre etiquetas <a>
  - Al hacer clic sobre una de ellas se coloca la ficha ahí
  - Si se detecta un final de partida, se marca la partida como finalizada
  - Si la partida continúa, se pasa el turno al “elige ficha” del otro jugador

# Fin de la partida

- Cuando ningún jugador puede colocar fichas y el pozo está vacío se habla de **cierre** (*block*)
  - Gana el jugador que menos puntos sume. Si hay empate, gana el primer jugador que haya empatado comenzando a contar desde el jugador que tiene la mano en adelante
- Cuando el jugador actual coloca su última ficha en la mesa (**dominó** o *domino*), es el ganador
- El jugador ganador obtiene su puntuación como la suma de los puntos de las fichas de sus contrincantes

# Dibujo de las fichas

- Se proporcionan imágenes de libre uso para todas las combinaciones de las fichas
- Alternativamente, Unicode incluye representaciones de todas las piezas en horizontal y vertical que pueden usarse como texto directamente
  - <https://www.alanwood.net/unicode/domino-tiles.html>



# Consejos para implementar la lógica (I)

- Implementa el estado del juego con clases y objetos
- Crea la clase **Tile** que encapsule una ficha especificando los puntos (*pips*) de cada extremo
  - El constructor recibe los puntos de los extremos (deben ser valores correctos) y exponerlos al exterior con *getters*
  - Implementa el método **connectsTo(Tile t)** que devuelva **true** si la ficha puede conectarse con la pasada como parámetro
  - Implementa el método **countPips()** que devuelva el número de puntos de la ficha
- Crea la clase **Stock** que, al instanciarse, genere las 28 fichas del juego barajadas
  - Implementa el método **getTile()** que devuelva una de las fichas restantes. Si se intenta obtener una ficha cuando no quedan, lanzar una excepción
  - Implementa el método **isEmpty()** que devuelva true si el pozo está vacío

# Consejos para implementar la lógica (II)

- Crea la clase **Table** que encapsule las fichas colocadas en la mesa
  - Guardará un *array* de fichas, inicialmente vacío
  - Implementa el método **getTiles()** que devuelva un *array* con las fichas en el orden en que se muestran en pantalla
  - Implementa el método **addTileLeft(Tile t)** que añada la ficha 't' al extremo izquierdo. Si no conecta con ese extremo, lanzar una excepción
  - Implementa el método **addTileRight(Tile t)** que añada la ficha 't' al extremo derecho. Si no conecta con ese extremo, lanzar una excepción
  - Implementa el método **canConnectToTheLeft(Tile t)** y **canConnectToTheRight(Tile t)** que devuelvan true si la ficha pasada como parámetro puede colocarse en el extremo izquierdo o derecho respectivamente

# Consejos para implementar la lógica (III)

- Crea la clase **Player** que encapsule las fichas que tiene un jugador
  - Guardará un *array* de fichas, inicialmente vacío
  - Implementa el método **getTiles()** que devuelva un *array* con las fichas del jugador
  - Implementa el método **addTile(Tile t)** que añada la ficha al jugador
  - Implementa el método **downTile(Tile t)** que quita la ficha indicada al jugador. Lanzar una excepción si no la tiene
  - Implementa el método **countPips()** que devuelve la suma de todos los puntos de sus fichas

# Consejos para implementar la lógica (IV)

- Crea la clase **Game** que encapsule la partida
  - Guardará un *array* con los *username* de los jugadores
  - Guardará en otro *array* asociado a cada *username* un objeto tipo **Player**
  - Almacenará un objeto de la clase **Stock** (para el pozo) y otro de la clase **Table** (para guardar lo colocado en la mesa)
  - Almacenará cuál es el *username* del usuario actual
  - Almacenará cuál es el *username* del usuario que tiene la mano
  - Guardará la ficha que ha seleccionado el usuario que tiene el turno (por si está pendiente de indicar dónde colocarla)
- Para guardar la partida en la base de datos, puedes serializar el objeto Game en una columna de tipo TEXT con **serialize(\$v)**, que devuelve una cadena que contiene el parámetro de forma serializada
- Para recuperar el objeto original, puedes usar el método **unserialize(\$cadena)** que devuelve el objeto reconstruido

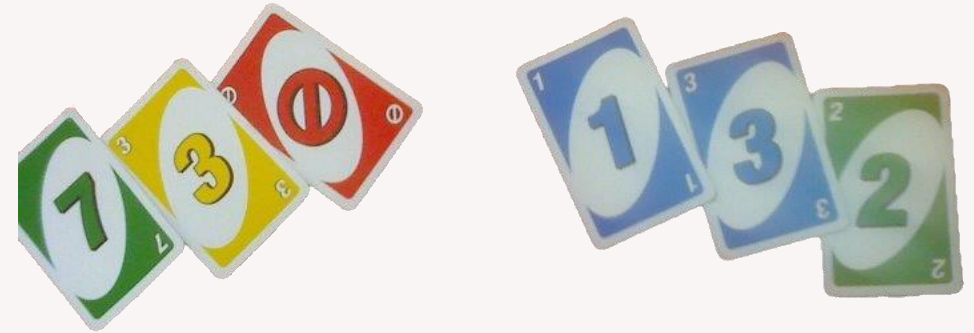
# Simplificaciones opcionales

- Haremos partidas de una sola ronda
  - En el dominó original, antes de empezar una partida se determina entre los jugadores una puntuación para ganar
  - A partir de ahí se echan tantas rondas como sean necesarias. En cada una de ellas, el ganador suma puntos. Cuando al acabar una ronda alguno de los jugadores supera la puntuación acordada al comienzo, gana la partida



# Si eres “pro” y quieres ampliar

- En lugar de hacer una ronda por partida, haz que en **waiting.php** se permita establecer cuál será la puntuación objetivo
- Implementa tantas rondas como sean necesarias y acumula los puntos hasta que algún jugador llegue a la puntuación
  - El jugador que tiene la mano comienza la primera ronda
  - En la segunda y siguientes empieza a quien le tocaba el turno después de la última jugada realizada y puede colocar la ficha que quiera, no necesariamente el 6 doble o similar
- Al finalizar cada ronda, muestra la puntuación acumulada de cada jugador y cuál es la puntuación objetivo. Al jugador que tiene el turno (y que comienza la nueva ronda) debe seleccionar en esta pantalla con qué ficha va a empezar la ronda. Los demás jugadores esperan viendo la tabla de puntuaciones



# Opción C: UNO

Partida de UNO para entre 2 y 10 jugadores

# Información

- Implementar una partida de UNO para 2 a 10 personas
- Una vez los dos jugadores se hayan unido a la partida...
  - Si hay al menos dos jugadores, cualquiera de ellos podrá dar un botón en **waiting.php** para comenzar la partida
  - Si hay 10 jugadores, directamente se activará la partida sin esperar a nada más

# Información

- Al iniciar la partida se asignan 0 puntos a cada jugador. Una partida se compone de muchas rondas
- Al iniciar una ronda
  - Se barajarán las 108 cartas (*cards*) en el mazo de reparto (*draw pile*)
  - Se prepara el mazo de descarte (*discard pile*) que recibirá las cartas que se vayan descubriendo
  - Se repartirán 7 cartas del mazo a cada jugador
  - Se elegirá un jugador al azar para empezar
  - Se sacará otra carta del mazo de reparto y se colocará en el mazo de descarte

# Información

- Composición de un mazo de UNO
  - 19 cartas *Red* (0 al 9, dos de cada salvo del 0)
  - 19 cartas *Blue* (0 al 9 , dos de cada salvo del 0)
  - 19 cartas *Green* (0 al 9 , dos de cada salvo del 0)
  - 19 cartas *Yellow* (0 al 9 , dos de cada salvo del 0)
  - 8 cartas de pasar turno (*Skip*), dos de cada color
  - 8 cartas de cambio de sentido (*Reverse*), dos de cada color
  - 8 cartas de roba dos (*Draw 2*), dos de cada color
  - 4 cartas de roba cuatro (*Wild Draw 4*)
  - 4 cartas de comodín (*Wild Card*)

# Información

- Por cada partida se almacenará en la base de datos
  - El orden de los jugadores y cuántos puntos tiene cada uno
  - El número de ronda
  - El jugador que tiene el turno actual y el sentido del turno
  - Las cartas que hay en el mazo de descarte y en el mazo de reparto
    - Es especialmente importante la última de la pila de descarte, que es la que determina la próxima jugada
  - Las cartas que tiene cada jugador en su mano
  - Si el jugador actual está pendiente de seleccionar un color (por colocar un comodín)
  - El color activo (*red, yellow, green* o *blue*) porque no tienen por qué coincidir con los de la última carta en la pila de descarte

# Script del juego

- El *script* principal del juego mostrará la última carta de la pila de descarte y cuántas cartas quedan en el mazo de reparto
- A continuación se indicará de quién es el turno y se mostrarán las cartas del jugador de todos los jugadores (las cartas de otros jugadores se mostrarán por la parte de atrás)



# Script del juego cuando se tiene el turno

- Si el jugador actual está pendiente de seleccionar el nuevo color activo, mostrar 4 botones para que lo elija y pase el turno
- Si no está pendiente de elegir color, el jugador actual tendrá que seleccionar una de las cartas que puede jugar (usar etiquetas <a> para convertirlas en un enlace)
  - Si no puede colocar nada, robará una carta de mazo de reparto. Si no puede jugarla, roba otra carta y pasa el turno directamente
    - Si se acaban las cartas del mazo de reparto, mover todas las cartas del mazo de descarte menos la última (la que está al descubierto) al primero y barajarlas
  - Sacar la carta de su mazo y colocarla en el mazo de descarte
  - Si es un comodín y puede elegir el color activo, marcar que está pendiente de seleccionarlo
  - Si no necesita cambiar el color, cambiar el turno al siguiente jugador



# *Script* del juego cuando se tiene el turno

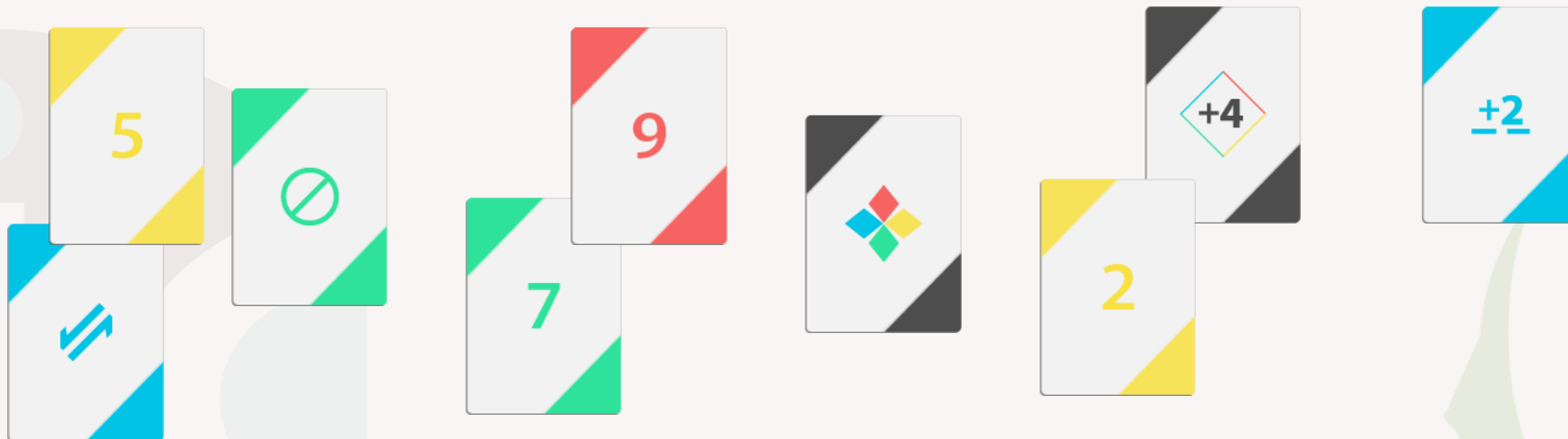
- Recuerda que la carta que se juegue tiene que ser del mismo color o del mismo símbolo que la última del mazo de descarte
- Si es un +2 o un +4 el jugador robará tantas cartas como se indique del mazo de reparto y pasará su turno
- No se puede jugar un +4 salvo que sea la única carta que el jugador pueda jugar

# Fin de la partida

- El primer jugador que se quede sin cartas gana la ronda y suma puntos
- Los puntos se obtienen de las cartas que les quedan en las manos al resto de jugadores según esta tabla:
  - Cartas con número: Tantos puntos como indique la carta
  - *Draw 2*: +20 puntos
  - *Reverse*: +20 puntos
  - *Skip*: +20 puntos
  - *Wild Card*: +50 puntos
  - *Wild Draw 4*: +50 puntos
- Cuando un jugador termine una ronda con 500 o más puntos, habrá ganado la partida

# Dibujo de las cartas

- Se proporcionan imágenes de libre uso de todas las cartas
  - Es posible saber el nombre del fichero de la imagen a partir del tipo de carta
- Se recomienda dejarlas en una subcarpeta “*images*” para que no se mezclen con el resto de *scripts*



# Consejos para implementar la lógica (I)

- Implementa el estado del juego con clases y objetos
- Crea una clase **Card** que encapsule una carta especificando el color y el símbolo
  - El constructor recibe el color (entero) y el símbolo (entero) y exponlos al exterior con *getters* (deben ser valores correctos, lanza excepción si no es así)
    - Para expresar el color, define dos constantes enteras dentro de la clase: **BLUE** (0), **GREEN** (1), **RED** (2), **YELLOW** (3) y **BLACK** (4)
    - Para expresar el símbolo, define constantes enteras dentro de la clase: **PICKER** (10), **REVERSE** (11), **SKIP** (12), **COLOR\_CHANGER** (13) o **PICK\_FOUR** (14). Los números del 0 al 9 se escriben directamente
  - Implementa el método **getImage()** que devolverá el nombre del fichero que dibuja la carta
  - Implementa el método **matches(Card c)** que devuelve **true** si la carta pasada como parámetro puede colocarse encima
  - Implementa el método **getPoints()** que devuelva cuántos puntos vale la carta

# Consejos para implementar la lógica (II)

- Crea una clase abstracta **Pile** (mazo) que encapsule un conjunto de cartas
  - Almacena las cartas como un *array* inicialmente vacío
  - Implementa un método **addCard(Card c)** que añada una carta al final del *array*
  - Implementa un método **shuffle()** que desordene las cartas del *array*
- Implementa la clase **DrawPile** como hija de **Pile**
  - Al instanciarse, se rellenará con las 108 cartas barajadas. Se repetirá el barajado hasta que la primera de ellas no sea un +4
  - Implementa el método **drawCard()** que retira la primera carta del *array* y la devuelve como valor de retorno
  - Implementa el método **addPileButLast(Pile p)** que añada al *array* todas las cartas del mazo indicado como parámetro menos la última
  - Implementa el método **countCards()** que devuelve cuántas cartas quedan en el mazo

# Consejos para implementar la lógica (III)

- Implementa la clase **DiscardPile** como hija de **Pile**
  - Al instanciarse, estará vacía
  - Implementa el método **peekLastCard()** que muestra cuál es la última carta añadida al mazo
  - Implementa el método **clear()** que elimina todas las cartas del mazo menos la última
- Implementa la clase **PlayerPile** como hija de **Pile**
  - Al instanciarse, estará vacía
  - Implementa el método **getCards()** que devuelve una array con todas las cartas del mazo
  - Implementa el método **playCard(Card c)** que elimina la carta indicada del mazo. Si no estaba en el mazo, lanzar excepción

# Consejos para implementar la lógica (IV)

- Implementa la clase **Game** que encapsule una partida
  - Guardará un *array* con los *username* de los jugadores
  - Guardará en otro *array* asociado a cada *username* un objeto tipo **PlayerPile**
  - Almacenará un objeto de la clase **DrawPile** (para el mazo de reparto) y otro de la clase **DiscardPile** (para el mazo de descarte)
  - Almacenará cuál es el *username* del usuario actual
  - Indicará si el jugador actual tiene que elegir color por haber colocado un comodín
  - Sentido del turno: hacia adelante o hacia atrás