




Presentación *Practica de curso*

Entrega I

Juego del Connect 4

Aplicaciones para Dispositivos Móviles

Guión

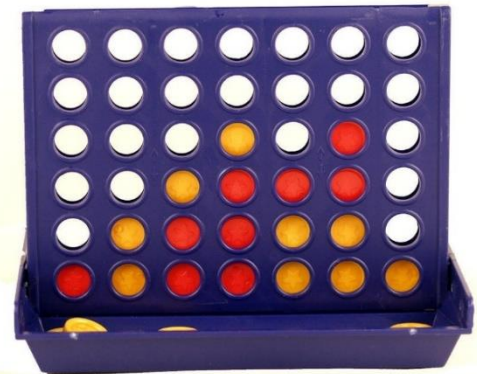
- ▶ **Presentación**
 - ▶ Consistencia
 - ▶ Mapa de navegación
 - ▶ Resultados
 - ▶ Requisitos
 - ▶ Pautas y claves de resolución
 - ▶ Parte Opcional
 - ▶ Criterios puntuación y entrega
- 

Consistencia del juego

Juego popular

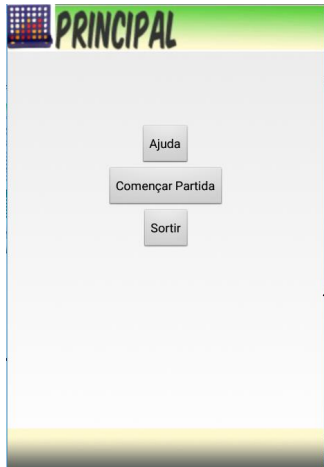
- ▶ El Connect 4 (o 4 en línea) es un juego estratégico de mesa para dos jugadores (el jugador humano y el sistema), cuyo objetivo es alinear cuatro fichas consecutivas del color asignado al jugador (e.g. rojo para el jugador humano) antes que tu oponente
 - Las fichas se pueden alinear en cualquier dirección (horizontal, vertical o diagonal), teniendo en cuenta que es un tablero vertical
- ▶ Gana la partida el jugador que primero consigue alinear cuatro fichas consecutivas de su color en horizontal, vertical o diagonal, teniendo en cuenta que es un tablero vertical

Gana el primer jugador que consigue alinear 4 fichas de su color !

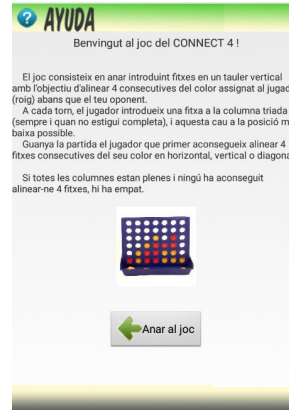


Mapa de navegación

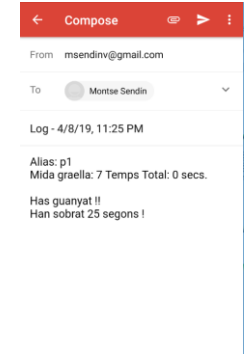
Menu Principal



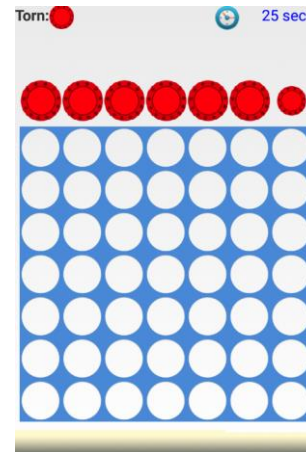
Ayuda



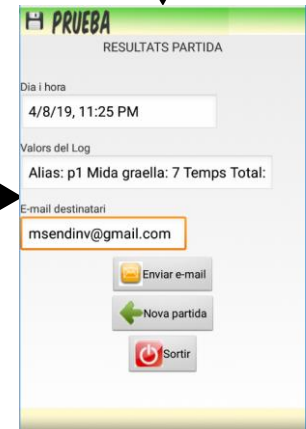
Cliente correo



Configuración



Desarrollo Juego



Resultados



Instrucciones mapa de navegación

Aspectos de navegación

- ▶ La pantalla de **Ayuda** te retorna al menú principal
- ▶ En el instante en que finaliza la partida la app te lleva automáticamente a la pantalla de **Resultados**, dando la opción de enviar el log por email
- ▶ El usuario podrá jugar tantas partidas como desee sin salir de la app, volviendo atrás desde la pantalla de resultados
- ▶ Reiniciar nueva partida supone pasar por la pantalla de **Configuración**, en la que estableces los parámetros del juego
 - La pantalla de configuración te lleva a la pantalla **Desarrollo del juego**
- ▶ El usuario podrá salir de la app desde el **Menú principal** y también desde la pantalla de **Resultados**

No debe quedar ninguna activity en la back stack !!

Pantalla de Configuración

Configuración de la partida...

- ▶ Pantalla donde se establecen los parámetros de cada partida:
 - **Alias del jugador** (*text*)
 - **Tamaño de parrilla:** un *RadioGroup* con 3 *RadioButton* exclusivos (por defecto: 7)
 - **Control del tiempo:** un *CheckBox*
 - Si habilitado se activa el control del tiempo, por lo que el usuario dispone de un tiempo máximo
 - **Otras opciones:**
 - Nivel de dificultad
 - Tiempo máximo de partida

CONFIGURACION

Alias identificatiu

p1

Mida graella

7 6 5

Control del temps

Començar

Pantalla de Resultados

Presentación de los resultados, listos para su envío por email...

► Pantalla donde se visualizan los siguientes datos:

- **Día y hora** de finalización de la partida
- **Valores del log:** log de la partida, a enviar como cuerpo del email
- **Email del destinatario**
 - Mostrará un valor establecido por defecto, o un *hint*
 - Dispondrá del foco
 - El asunto del email será: **Log – fecha y hora**

Todos ellos como *EditText*, y por tanto editables

PRUEBA

RESULTATS PARTIDA

Dia i hora
Mar 22, 2017 1:53:14 PM

Valors del Log
Alias: p1Heu empatat !! Han sobrat -3

E-mail destinatari
pepitogrillo@gmail.com

Enviar e-mail

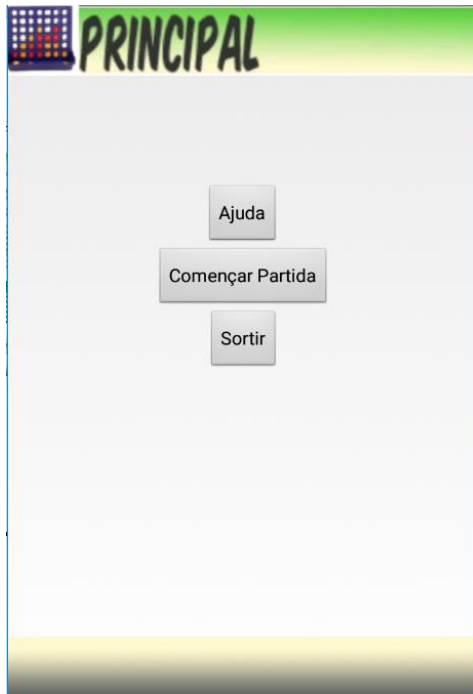
Nova partida

Sortir

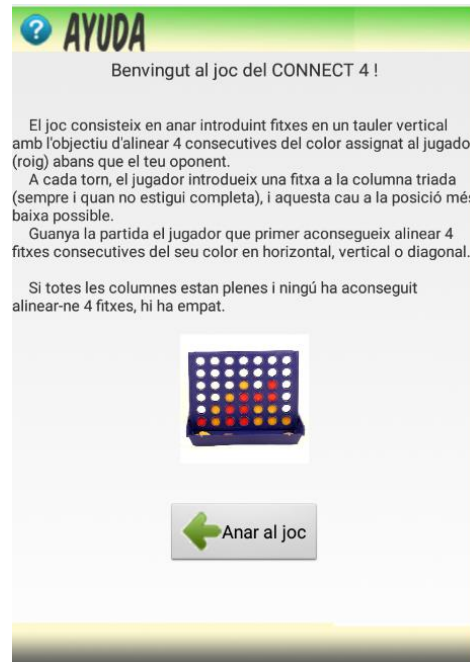
Otras pantallas

Resto de pantallas

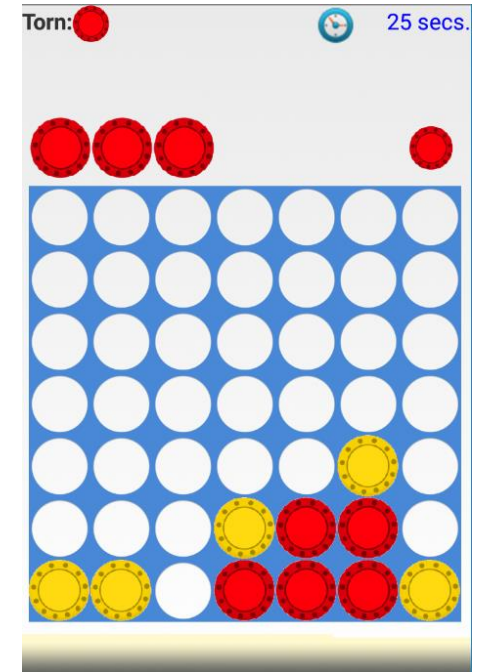
Menu Principal



Ayuda



Desarrollo Juego



Resultados

- ▶ Una vez concluida la partida, la app debe:
 1. Mostrar el *log* con el resumen de la partida
 2. Dar la posibilidad de enviar un e-mail con el log resultante
 - Es conveniente proporcionar una dirección de correo electrónico por defecto, así como dejar el foco en este campo de texto
- A tener en cuenta:
 - Los datos del log empiezan a conformarse desde la pantalla de Configuración, para luego irse desarrollando a lo largo del juego, hasta su finalización (pantalla de *Resultados*)

Resultados

► Datos a recopilar en el log resultante de la partida:

Datos comunes:

- Alias usuario
- Tamaño parrilla
- Tiempo total empleado

Alias: p1
Mida graella: 7 Temps Total: 0 secs.

Además de

datos específicos propios del resultado del juego

Resultados

► Datos a recopilar en el log resultante de la partida:

Datos dependientes del resultado del juego:

- Si control del tiempo:

Temps total: 9 secs.

Alias: p1
Mida graella: 7 Temps Total: 9 secs.

Has guanyat !!

- Partida perdida por tiempo agotado

- Log : *Has esgotat el temps!!*

Alias: p1
Mida graella: 7 Temps Total: 31 secs.


Has esgotat el temps !! |

- Partida empatada

- Log : *Heu empatat !!*

Heu empatat !!

Guión

- ▶ Presentación
 - ▶ **Requisitos**
 - ▶ Pautas y claves de resolución
 - ▶ Parte Opcional
 - ▶ Criterios puntuación y entrega
- 

Requisitos

- ▶ **Implementación:** La parrilla se implementará utilizando el control de selección *GridView* compuesta por una matriz de objetos *ImageView*
- ▶ **Rotaciones:** La app debe estar preparada para poder cambiar la orientación de la pantalla *sin que ello suponga perder ningún tipo de información*
 - Gestionar la **recreación**, especialmente de la pantalla de desarrollo del juego
 - **Personalizar** adecuadamente las vistas para la orientación horizontal
- ▶ **Control del tiempo:** El control del tiempo debe ser lo más preciso posible
 - Iniciar justo al inicio del juego
 - Finalizar justo en el instante de finalización de la partida
 - Ir mostrando el tiempo restante en cada clic (o en tiempo real), y también en cada rotación de pantalla (pantalla del juego)
 - ***Control de si tiempo agotado en cada click y en cada rotación de pantalla !***
 - El valor de *tiempoMáximo* estará establecido (e.g. 25 segundos), o bien a introducir por el usuario

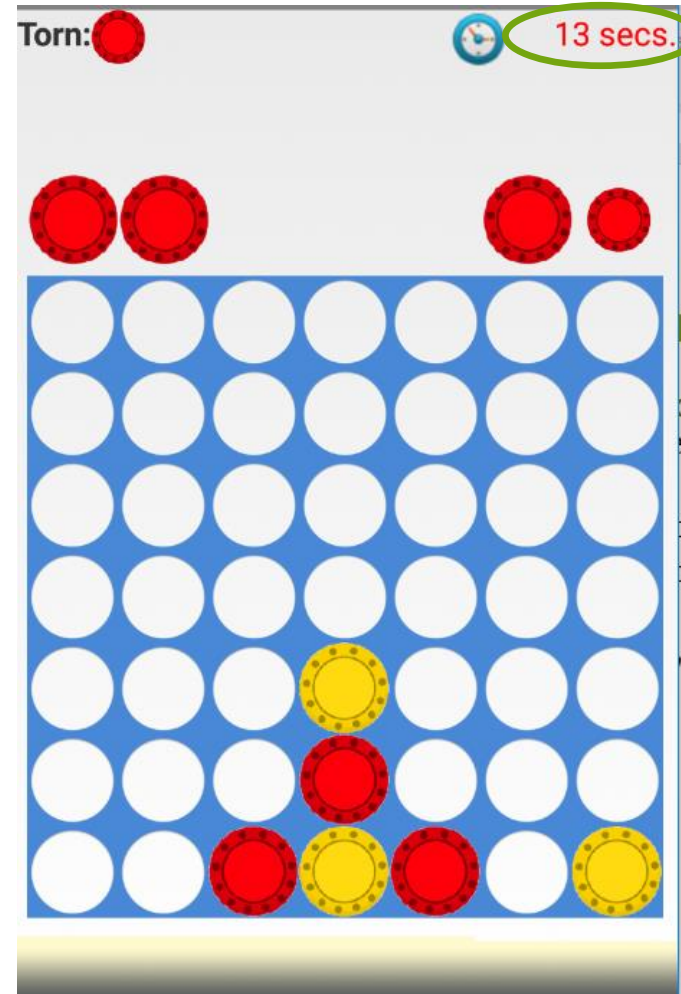
Requisitos

Tratamiento de información durante el juego

- ▶ **En cada clic**, además de refrescarse la parrilla del juego se actualizará el valor del tiempo restante. A mostrar:

- ▶ En color azul si no se está controlando el tiempo
- ▶ En color rojo si sí se está controlando el tiempo

A actualizar si cambio de orientación antes del clic



Requisitos

Tratamiento adecuado de la finalización del juego

- ▶ Existen 4 situaciones en las que el juego alcanza su finalización:

Tablero queda sin completar

- ▶ Tiempo agotado


Partida finalizada

- Gana el jugador humano (fichas rojas)
- Gana el sistema (fichas amarillas)
- Se produce un empate



La app deberá tener en cuenta todas estas consideraciones y tratarlas oportunamente, generando el log apropiado a cada caso, así como proceder a la finalización del juego, y de la activity correspondiente de manera instantánea

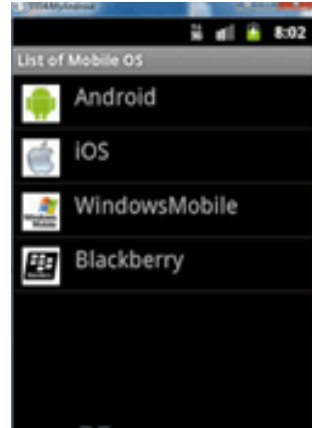
Guión

- ▶ Presentación
 - ▶ Requisitos
 - ▶ Pautas y claves de resolución
 - ▶ Parte Opcional
 - ▶ Criterios puntuación y entrega
- 

Pautas y claves de resolución

Personalizar un Adapter

- ▶ Por defecto, un *ArrayAdapter* crea un objeto *TextView* por cada elemento del array (fuente de datos de un *ArrayAdapter*)
 - El mapeo de la fuente de datos a la vista se hace invocando a *toString()* por cada ítem y pasando el contenido a un *TextView*



Eso es muy limitado

¿Y si me interesa acompañar cada literal con una imagen distinta ?

¿Qué hacer ?

1. Para controlar la asignación de los datos y crear una vista específica por cada ítem es necesario extender *ArrayAdapter*
 - Redefiniendo el método *getView()* de manera que retorne la vista (layout, la cual incluirá un *TextView*) deseada para cada ítem
 - **Método *getView()*** : Genera, da formato y retorna uno a uno cada uno de los objetos a ser visualizados en el control

Es la opción más sencilla

Pautas y claves de resolución

Personalizar un Adapter

2. Para que un adapter construya *Views* diferentes a *TextView* (e.g. un *ImageView*) por cada elemento del array, es necesario extender la clase base de *ArrayAdapter*: *BaseAdapter*

- ▶ Redefiniendo el método *getView()* de manera que retorne el tipo de *View* deseado para el control
- ▶ Además, es necesario redefinir los métodos *getCount()*, *getItem()* y *getItemId()*
- ▶ *Es la opción más potente*

<http://developer.android.com/reference/android/widget/BaseAdapter.html>

- Por ejemplo: *ImageAdapter*

Pautas y claves de resolución

Personalizar un Adapter

- ▶ Ej: clase *ImageAdapter*

- ▶ Conveniente implementar una constructora

`public ImageAdapter (Context c)`

Puede resultar útil para pasar parámetros adicionales, además del contexto

```
public class ImageAdapter extends BaseAdapter {  
  
    public ImageAdapter (Context c) {  
  
    }  
  
    @Override  
    public int getCount() {  
        // TODO Auto-generated method stub  
        return 0;  
    }  
  
    @Override  
    public Object getItem(int position) {  
        // TODO Auto-generated method stub  
        return null;  
    }  
  
    @Override  
    public long getItemId(int position) {  
        // TODO Auto-generated method stub  
        return 0;  
    }  
  
    @Override  
    public View getView(int position, View convertView, ViewGroup parent) {  
        // TODO Auto-generated method stub  
        return null;  
    }  
  
}
```

Pautas y claves de resolución

Personalizar un Adapter

► Métodos imprescindibles a implementar:

- ***getCount()***, que retorna el nº de objetos a ser visualizados en el control (longitud del array de ítems)
- ***getView()***, que genera y retorna uno a uno cada uno de los objetos a ser visualizados en el control

- Ambos harán referencia al array de ítems a manejar en el control

```
public class ImageAdapter extends BaseAdapter {  
  
    public ImageAdapter (Context c) {  
  
    }  
  
    @Override  
    public int getCount() {  
        // TODO Auto-generated method stub  
        return 0;  
    }  
  
    @Override  
    public Object getItem(int position) {  
        // TODO Auto-generated method stub  
        return null;  
    }  
  
    @Override  
    public long getItemId(int position) {  
        // TODO Auto-generated method stub  
        return 0;  
    }  
  
    @Override  
    public View getView(int position, View convertView, ViewGroup parent) {  
        // TODO Auto-generated method stub  
        return null;  
    }  
  
}
```

→ En nuestro caso la estructura que guarda la
configuración del tablero de juego !!

Pautas y claves de resolución

Personalizar un Adapter

► Métodos en los que sirve la implementación por defecto:

- *getItem()*, como *getView()* pero retornando un *Object*.
Puede dejarse retornando null
- *getItemId()*, retorna el ID del ítem.
Puede dejarse retornando el propio valor del argumento (*position*)

```
public class ImageAdapter extends BaseAdapter {  
  
    public ImageAdapter (Context c) {  
  
    }  
  
    @Override  
    public int getCount() {  
        // TODO Auto-generated method stub  
        return 0;  
    }  
  
    @Override  
    public Object getItem(int position) {  
        // TODO Auto-generated method stub  
        return null;  
    }  
  
    @Override  
    public long getItemId(int position) {  
        // TODO Auto-generated method stub  
        return 0;  
    }  
  
    @Override  
    public View getView(int position, View convertView, ViewGroup parent) {  
        // TODO Auto-generated method stub  
        return null;  
    }  
  
}
```

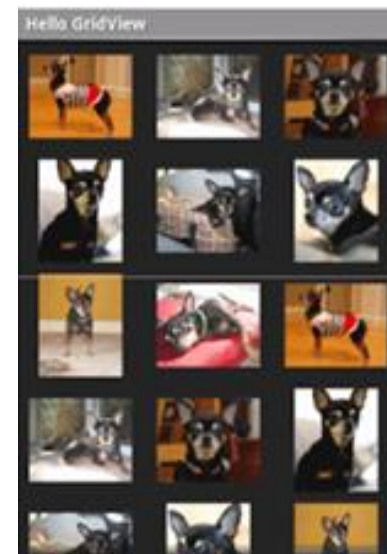
Pautas y claves de resolución

Personalizar un Adapter

- Definición de un adaptador cuando los elementos a manejar son *ImageViews*
- Establecer el adaptador al control *GridView*

```
public class ImageAdapter extends BaseAdapter {  
    private Context mContext;  
  
    public ImageAdapter(Context c) {  
        mContext = c;  
    }  
  
    public int getCount() {  
        return mThumbIds.length;  
    }  
  
    public Object getItem(int position) {  
        return null;  
    }  
  
    public long getItemId(int position) {  
        return 0;  
    }  
  
    // create a new ImageView for each item referenced by the Adapter  
    public View getView(int position, View convertView, ViewGroup parent) {  
        ImageView imageView;  
        if (convertView == null) { // if it's not recycled, initialize some  
            imageView = new ImageView(mContext);  
            imageView.setLayoutParams(new GridView.LayoutParams(85, 85));  
            imageView.setScaleType(ImageView.ScaleType.CENTER_CROP);  
            imageView.setPadding(8, 8, 8, 8);  
        } else {  
            imageView = (ImageView) convertView;  
        }  
  
        imageView.setImageResource(mThumbIds[position]);  
        return imageView;  
    }  
}
```

```
gridview.setAdapter(new ImageAdapter(this));
```



```
// references to our images  
private Integer[] mThumbIds = {  
    R.drawable.sample_2, R.drawable.sample_3,  
    R.drawable.sample_4, R.drawable.sample_5,  
    R.drawable.sample_6, R.drawable.sample_7,  
    R.drawable.sample_0, R.drawable.sample_1,  
    R.drawable.sample_2, R.drawable.sample_3,  
    R.drawable.sample_4, R.drawable.sample_5,  
    R.drawable.sample_6, R.drawable.sample_7,  
    R.drawable.sample_0, R.drawable.sample_1,  
    R.drawable.sample_2, R.drawable.sample_3,  
    R.drawable.sample_4, R.drawable.sample_5,  
    R.drawable.sample_6, R.drawable.sample_7  
};
```

Pautas y claves de resolución

Personalizar un Adapter

- Implementación básica del método *getView()* –un caso de ejemplo:
 - Genera, da formato y retorna uno a uno cada uno de los objetos a ser visualizados en el control. En este ejemplo, cada uno de los elementos *Button* a desplegar en un *GridView*

```
public View getView(int position,
                    View convertView, ViewGroup parent) {
    Button btn;

    if (convertView == null) {
        // if it's not recycled, initialize some attributes
        btn = new Button(mContext);
        btn.setLayoutParams(new GridView.LayoutParams(GridView.LayoutParams.WRAP_CONTENT, 50));
        btn.setPadding(8, 8, 8, 8);
    }

    else {
        btn = (Button) convertView;
    }

    btn.setText(numblist.get(position));
    btn.setTextColor(Color.BLACK);

    btn.setBackgroundResource(R.drawable.grayrectanglebutton);
    btn.setOnClickListener(new MyOnClickListener(position));

    btn.setId(position);

    return btn;
}
```

Pautas y claves de resolución

Personalizar un Adapter

► Establecer el manejador de eventos del control:

- Con objeto de que los botones reaccionen al ser seleccionados, el manejador de eventos debe ser **onClickListener**, a establecer para cada botón

```
// Set the onclicklistener so that pressing the button fires an event
// We will need to implement this onclicklistener.
btn.setOnClickListener(new MyOnClickListener(position));
```

- Definir el manejador de eventos como una clase. Eso permitirá **personalizarla**:
 - Definiendo la constructora con un argumento entero: la posición del elemento (botón) seleccionado dentro del *GridView*
 - **OPC**: Añadiendo cualquier otro argumento, así como nuevos métodos para la manejadora de eventos, si así se requiere

```
class MyOnClickListener implements OnClickListener
{
    private final int position;

    public MyOnClickListener(int position)
    {
        this.position = position;
    }

    public void onClick(View v)
    {
        // Perform a function based on the position
        someFunction(this.position)
    }
}
```

Pautas y claves de resolución


Personalizar un Adapter

► Qué debe hacer el método *onClick* en nuestra aplicación:

¿Qué hacer cada vez que el jugador selecciona una columna de la parrilla?

- Actualizar el tiempo transcurrido, así como las casillas colocadas por ambos jugadores y las casillas pendientes
- Cambiar el turno
- Comprobar si partida finalizada, y por qué causa
 - Desencadenar los pasos pertinentes
- ➔ Actualizar la parrilla (nueva casilla colocada y cambios pertinentes)
notifyDataSetChanged()
 - Notifica a los observadores registrados que los datos subyacentes han cambiado y cualquier *View* asociada al conjunto de datos requiere refrescarse

Guión

- ▶ Presentación
 - ▶ Requisitos
 - ▶ Pautas y claves de resolución
 - ▶ **Parte Opcional**
 - ▶ Criterios puntuación y entrega
- 

Parte Opcional

- ▶ Parte Obligatoria: peso del **95%**
- ▶ Parte Opcional: **15%** de la nota (10% *extra* !!)

Por ejemplo....


7,5%: Mejora de la experiencia de usuario

- ▶ Introducir mejoras en la interfaz
 - **Controles** de IU más específicos (*spinner, seekbar*, etc.), **sonidos**, etc.
 - Reproducir una **animación** al hacer seleccionar la columna escogida
- ▶ Resolver la funcionalidad del ***Undo*** i/o ***Redo*** 1 ó más movimientos
- ▶ Personalizar los Toast

7,5%: Niveles de dificultad

- ▶ Introducir **niveles de dificultad**
 - Mediante la incorporación de heurísticas (inteligencia artificial)

Guión

- ▶ Presentación
 - ▶ Requisitos
 - ▶ Pautas y claves de resolución
 - ▶ Parte OPCional
 - ▶ Criterios puntuación y entrega
- 

Criterios puntuación

► Funcionamiento correcto y requisitos:

- **Robustez** (obtención de los resultados de la partida correctamente en todos los casos)
- **Cumplimiento** de todos los **requisitos** definidos
- **Validación de la entrada y otros aspectos como** uso de valores por defecto o *hints*, alerta de errores por parte del usuario, etc.

Criterios puntuación

► Elegancia de código:

- Código entendedor en el conjunto de las distintas componentes
- Descomposición adecuada del código (Diseño Descendiente)
- Definición de clases propias. Se valorará:
 - Definición adecuada de clases (estructura y comportamiento)
 - Empleo que se hace de ellas para diversos propósitos
 - *Ex:* métodos *getSerializable()* y *putSerializable()* en clases que implementen *Serializable*
 - Lo mismo es válido para *Parcelable*
- *Adicionalmente* se valorarán otros criterios de diseño como:
 - Máxima cohesión, mínimo acoplamiento
 - Aplicación de patrones de diseño, etc.

Criterios puntuación

► Seguimiento de las pautas facilitadas para su resolución:

- Resolver los distintos **aspectos generales** tal y como se ha presentado en clase
 - *Por ejemplo, resolver la gestión de los cambios de orientación de la pantalla* según se ha estudiado en clase
- Resolver los **aspectos específicos de la práctica** tal y como han sido presentados en este documento

► Presentación y diseño:

- **Aspectos de diseño** que ofrezcan una experiencia de usuario satisfactoria
 - Aplicar **criterios de usabilidad**: interfaz agradable, intuitiva, gráfica, libre de *lags*
 - Definición adecuada de las vistas, extensible a ambas orientaciones del dispositivo

Criterios puntuación

Recomendaciones

► La definición de clases ...

- Facilita un código más elegante y entendedor
- Facilita **futuras ampliaciones** o **funcionalidades extra**
 - Desarrollo parte OPCional
 - Desarrollo entregas posteriores de la práctica

► Diseño Descendiente

- Facilita un código más entendedor

Criterios puntuación

▶ Parte Obligatoria (9,5 puntos):

- Funcionamiento y requisitos:
2 puntos
- Elegancia de código:
3 puntos
- Seguimiento de las pautas:
2,25 puntos
- Presentación y diseño:
2,25 puntos

▶ Ambas partes (11 puntos):

- Funcionamiento y requisitos:
2,5 puntos
- Elegancia de código:
3,5 puntos
- Seguimiento de las pautas:
2,25 puntos
- Presentación y diseño:
2,75 puntos

Entrega

Plazo: 1 mes (10 de Mayo)

- ▶ A desarrollar preferiblemente en grupos de 2 personas
- ▶ Entrega por el *Campus Virtual*, **preferiblemente** mediante repositorio **GitHub**
 - Se entrega tanto el proyecto (empaquetado o bien enlace a GitHub), como el **.apk** resultante
 - **Entrega presencial** en casos dudosos, por parte de uno o ambos miembros del grupo

Peso en la nota:

- ▶ Peso de la Práctica de curso: **50%** de la nota de la asignatura
 - ▶ Peso de la 1ª entrega de la práctica: **25%** de la nota de la asignatura
- 