Universidad de Concepción
Facultad de Ingeniería
Departamento de Ing. Informática y
Cs. de la Computación

# Informe Tarea 2: Solitario

Desarrollo de Aplicaciones para Dispositivos Móviles (543780-1)
Mario Medina Carrasco

**Alumnos:**

Nicolás Araya

Martina Cádiz

**Ayudantes:**

Jorge Palacios
Isaac Goicovich

**Fecha:**

03/07/20

## Descripción de la tarea

La tarea consiste en crear una aplicación que implemente el juego “Senku”. Para poder llevar a cabo esta tarea, el contenido gráfico fue desarrollado con la herramienta externa Adobe Photoshop. El proyecto está compuesto por 6 layouts.
La aplicación lleva por nombre “Solitario”. Al abrir la aplicación se despliega un Menú con 4 opciones, estas son Jugar, ¿Cómo Jugar?, Créditos y Salir. Al pulsar sobre Jugar, permite elegir qué tablero jugar. Una vez dentro se presenta el tablero en la parte central de la pantalla. En la parte superior se encuentran 5 botones, en orden estos permiten volver al menú principal, reiniciar el tablero, volver atrás una jugada, saltar al tablero siguiente y sugerir una jugada(se requiere tener pulsada una casilla previamente).

La aplicación es capaz de:

1. Al seleccionar una ficha, dar sugerencia de una posible jugada al cliente.
2. Cambiar de tablero sin volver al Menú Principal
3. Poder volver un movimiento atrás
4. Poder volver al estado inicial el juego.
5. Identificar cuando un tablero ya no dispone de movimientos válidos
6. Cuando se gana, puede verificar si la última ficha quedó en la posición inicial.
7. Es posible jugar horizontal y verticalmente.

## Descripción de clases y sus métodos

### 1. Casillas y Movimientos

Para poder desarrollar la solución se crearon las clases auxiliares: “Casillas” que es la clase que determina y modifica el estado de cada sección del tablero y “Movimientos” que es la clase que se utilizará para guardar las jugadas.
Los métodos relevantes de “Casillas” son:

1. setLineas() y setColumna(): SetLineas está enfocada para el tablero pirámide, ya que esta permitirá reconocer en la columna que se encuentra cada casilla, se enumeran las casillas de izquierda a derecha y de derecha a izquierda. En cambio setColumna() está enfocada para el tablero cruz, acá solo se enumeran las columnas de izquierda a derecha.

2. setClick() y update(): Ambos tableros utilizarán estos métodos para cambiar el estado del los ImageButton. Si el atributo clicks es igual a 0 se considera la casilla vacía, si es 1 está seleccionada por el jugador, 2 se considera que la casilla está con una ficha y 4 es el estado cuando el jugador pide una sugerencia. Para 3,5,6,7 y 8 son los ImageButton que realizan otras funciones.

   Los métodos de “Movimientos” son getters y nos entregan el valor de los atributos privados asignados. Esta clase se asemeja a un vector triple, ya que almacena 3 valores. Cada vez que se realiza un movimiento, se crea una instancia de esta clase con las coordenadas de las 3 casillas que fueron modificadas, para luego ser almacenadas en un Stack propio de cada Tablero.

## 2. TableroCruz y TableroPiramide

Ambas clases poseen los mismos métodos y utilizan OnClickListener, sin embargo la implementación es diferente, ya que son tableros diferentes.
Para crear ambos tableros, a grandes rasgos, fue utilizando arreglos. Se creó un arreglo de tipo Imagebutton llamado "Ibtns", uno de tipo "Casillas" que contuviese la información del estado de cada casilla del tablero y fue llamado "info". Además se utilizaron las estructura de datos MutableMap y Stack. Las claves del map "bloques" era de tipo ImageButton y su valor era un objeto de tipo "Casillas". Por otra parte el Stack "movs" contenía objetos de tipo "Movimientos".

Acerca de los métodos:

1. OnClick(), OnCreate(): OnClick es el método que utilizan los ImageButton, este recibe la información de los eventos y se la entrega al método btnSelected(button:ImageButton), le entrega el botón seleccionado. OnCreate() es el método que instancia el ambiente, entregando el estado inicial del tablero. La única diferencia en los métodos de cada tablero es que, en el de forma piramidal se toma el "bloques[btns[1]]" como la sección vacía y para todas las "Casillas" se hace el llamado a setLineas(). En el tablero cruz, toma el "bloques[btns[17]]" como la sección vacía y para todas las "Casillas" se hace el llamado a setColumnas(). Comienza el juego con puntaje y movimientos nulos. Cabe destacar que la cantidad de movimientos realizados está dada por el tamaño del Stack.

2. btnSelected(ImageButton): Este está implementado de la misma manera en ambos tableros. Este método es el que recibe el ImageButton seleccionado. Lo primero que hace es llamar a juegoTerminado() para verificar si es posible realizar más movimientos. Luego analiza el estado de cada bloque, si es que hubo uno seleccionado previamente o si se realizó una sugerencia(el bloque con la sugerencia es regresado a su estado inicial para que no interfiera con las próximas jugadas). En el caso de haber un bloque seleccionado previamente, el jugador puede solicitar una sugerencia o realizar una jugada. Si escoge solicitar una sugerencia, se llamará al método Sugerencia(int,boolean) y si decide realizar una jugada se llamará a EvaluarJugada(int,int), este método recibe los movimientos realizados, es decir el index correspondiente a cada ImageButton seleccionado.
   Este método también recibe la información de los bloques que no pertenecen al tablero, es decir los ImageButtons que aparecen en la parte superior del juego.

3. EvaluarJugada(int,int): En este método se analiza la jugada del cliente y cabe destacar que en las clases se implementa de manera totalmente diferente. A grandes rasgos ambos cumplen la función de avisar, mediante un Toast, si la jugada no es posible de realizar y en caso contrario cambiar el estado de cada ImageButton mediante los bloques.
   Primero, en ambos casos, se utilizará la variable "comp", esta representa la distancia, entre las filas que tienen las bloques. Además se busca cual es el bloque que se presenta primero en el arreglo y el menor. Esto es para que posteriormente la comparación sea mucho más sencilla.

Solución 1 : Tablero Piramidal

Acá es necesario llamar a setLineas(), ya que esta permitirá identificar en qué diagonal se encuentran los bloques. Se comienzan a contar de izquierda a derecha las diagonales y se guarda el valor en la variable “lineaVerticalIzqADer” y para identificar en que diagonal se encuentra de derecha a izquierda, se guarda en “lineaVerticalDerAIzq”.
El primer bloque if es para identificar si los bloques seleccionados por el cliente se encuentran en la misma fila. Para esto se utiliza la variable “comp”, si su diferencia da 0 es porque cumplen con la condición anterior, en caso contrario se verifica si el valor absoluto de la diferencia es igual 2, ya que para moverse verticalmente necesariamente tienen que saltarse una fila entre sí.
Si entra en el primer bloque if, se comprueba si el bloque que está entre los dos está con una “ficha”, si cumple el programa cambia el estado y settea todas las variables para que considere la jugada válida y se cuenten los puntos, y sino es asi, el el movimiento que realiza no es válido y el programa avisa que no es posible realizarlo.
Si entra en el bloque if else, se analiza cuando el cliente juega verticalmente. Acá se busca que los bloques seleccionados estén en la misma línea vertical, ya sea de izquierda a derecha o de derecha a izquierda. Sin embargo no pueden estar en las mismas líneas verticales de derecha a izquierda y de izquierda a derecha al mismo tiempo. Los bloques tienen que coincidir las líneas sólo de una orientación, es decir pertenecen a la misma línea de izquierda a derecha o pertenecen a la misma línea de derecha a izquierda. Si la casilla que se encuentra entre las dos seleccionadas no está llena, o no se cumplen las condiciones anteriores, el movimiento no es válido y el programa arroja un Toast advirtiendo esto.

Solución 2 : tablero cruz

A diferencia del otro tablero, acá se utiliza el setColumnas(), el primer bloque if cumple la misma función que el tablero anterior. Este corresponde a los bloques que son seleccionados en la misma fila. El bloque if else verifica la distancia que hay entre las filas y compara las columnas, esto último es gracias a los datos entregados al momento de settear setColumna() de cada bloque. Se comparan las distancias en las que se encuentran los bloques seleccionados, necesariamente tienen que estar con una diferencia de 2 entre las filas y deben poseer el mismo valor en el atributo “columna”, además se comprueba la distancia del botón que está al medio con respecto a los seleccionados y su estado. En caso de realizar un movimiento no válido, el programa avisa al cliente que la jugada no es posible mediante un Toast.

4. Resetear(): Este método, en ambas clases, permite que el tablero y todo el ambiente vuelva a estar en su estado inicial.

5. MovAtras(): Este método, en ambas clases, permite volver múltiples jugadas hacia atrás. Si el jugador realizó movidas anteriormente. Para ello utiliza el stack de movimientos, ve el elemento top() y modifica las casilla que fueron guardadas. Luego hace pop().

6. Sugerencia(Int, Boolean): Este método recibe el index del bloque que es seleccionado y un booleano que se le llamará “asked”. Cabe destacar la importancia de este booleano. Si el booleano es verdadero, el bloque que es considerado como una posible sugerencia será marcado, sin embargo si es false no cambiará el estado del bloque. Solo cambiará la variable que “posiblecambio” en el caso de existir un cambio, el booleano será false cuando se llame desde el método juegoTerminado(), ya que la función de este método no es entregar sugerencia solo avisar si no es posible seguir jugando.
   En resumen, funciona igual que el método descrito arriba. Se analiza la posible siguiente jugada mediante el ciclo for que se utiliza.

7. juegoTerminado():Este método es implementado con un ciclo for que recorre todas las casillas y busca una posible sugerencia con Sugerencia(Int, Boolean), el booleano que le entrega es false, ya que no se busca cambiar el estado de los bloques, sino que buscar si existen movimientos. Si encuentra uno, sale del ciclo for y continua el método. En caso de no encontrar ningún movimiento, cambia el texto principal para avisarle al jugador que no puede seguir jugando.

8. openActivity(): este método permite cambiar de mapa entre uno u otro, para esto es asignado al botón skip de la GUI. Y cierra la actividad actual y abre la que corresponde.

9. openHome(): este método permite volver al menú principal. Es asignado al botón home de la aplicación. Cierra la Actividad correspondiente a Tablero y ejecuta la Actividad Menú.

### 3. Menu, Selector, Créditos, Instrucciones:

Todas éstas corresponden a Actividades de la Aplicación.

1. La actividad Menú corresponde al main de la aplicación, esta cuenta con 4 botones: “Jugar”, “Como Jugar”, “Créditos” y “Salir”.

2. La Actividad Selector, es una actividad lanzada cuando es presionado el botón “Jugar” en menú. Selector permite mediante botones elegir el tablero a jugar.

3. La Actividad Créditos, es desplegada por la actividad Menú al pulsar sobre el botón “Créditos”. Esta actividad solo muestra el nombre de los desarrolladores y permite con un botón volver al menú

4. La Actividad Instrucciones es similar a la anterior, es lanzada por Menú al pulsar sobre el botón “Cómo Jugar”. Muestra información acerca del juego describe los botones de la interfaz. Cuenta con un botón para volver al menú

## Dificultades encontradas

1. Entender el patrón del tablero pirámide, para poder solucionar este problema, se tuvo que analizar los tipos de movimientos válidos y las distintas maneras que se podía almacenar el tablero. Encontrando soluciones como la implementación con matrices, sin embargo la opción que permitió una mejor adaptación fue utilizar arreglos y calcular el arreglo.

2. Guardar el estado de una actividad cuando es rotada. Es decir, que al rotar el dispositivo no se reinicie el progreso del tablero. Para esto investigamos y usando OnSaveInstanceState junto con las funciones putSerializable para guardar los objetos. Y luego restaurarlos en el método OnCreate de la Actividad utilizando getSerializable.

3. Poder entregar las jugadas anteriores, la implementación de este método fue un problema inicialmente. Primero se probó utilizando un arreglo auxiliar, sin embargo al investigar las estructuras de datos que existen, nos dimos cuenta que era mucho mejor usar stacks para poder ir regulando el orden de entrada y de salida. La utilización de stack facilitó el código considerablemente y permitió incluso guardar todas las jugadas anteriores.

4. Mantener un orden de los elementos en distintos tamaños de pantalla nos quitó bastante tiempo. Para esto utilizamos RelativeLayout para así mantener siempre el mismo orden en distintos tipos de pantallas. Y también los tamaños asignados fueron con unidades en “dp”, para mantener una escala relativamente consistente en los variados tamaños de pantalla.