Universidad Simón Bolívar
Departamento de Computación y Tecnología de la Información
CI-2691 - Laboratorio de Algoritmos y Estructuras I
Trimestre Abril-Julio 2015

Proyecto: Lineas de colores y rectángulos

El objetivo de este proyecto es la implementación de un programa no trivial, haciendo uso del análisis descendente y de las técnicas vistas a lo largo del curso de teoría y práctica de Algoritmos y Estructuras I.

1. Planteamiento del Problema

Se quiere que usted implemente un juego llamado *Lineas de colores y rectángulos*. El juego consiste básicamente en construir lineas de 5 ó más círculos de colores y rectángulos de al menos $2x2$ cuadros, en un tablero de $9x9$ cuadros. El objetivo del juego es acumular la mayor cantidad de puntos posibles al formar las líneas y los rectángulos. Este juego es una variante del juego *Color Lines*, originalmente hecho para MS-DOS [3]. Existen hoy en día varias variantes de *color lines* que son accesibles. En el proyecto GNOME está disponible el juego *Five o more* [1].

1.1. Desarrollo y reglas del juego

El juego se desarrolla en un tablero de $9x9$ cuadros. Hay siete tipos de objetos divididos en dos grupos:

- Seis bolas de diferentes colores.
- Un cuadrado de cualquier color.

Al comienzo del juego en el tablero aparecen tres objetos seleccionados al azar, que se colocan en posiciones aleatorias. Además del tablero de juego, en el panel del juego se pueden visualizar los próximos tres objetos que van a ser colocados en el tablero una vez el jugador consuma su turno.

El objetivo del juego consiste en alinear los objetos en el tablero para que desaparezcan del mismo. La alineación va a depender del tipo de objeto. Para que una serie de objetos desaparezcan del tablero de juego se deben formar:

- Lineas de al menos 5 bolas, que sean del mismo color. Las líneas pueden ser horizontales, verticales o diagonales
- Rectángulos de al menos 2x2 cuadros de objetos cuadrados.

Una vez presentado el panel con el tablero y con los próximos objetos a agregar, el jugador debe escoger uno de los objetos del tablero, y moverlo a una posición donde no haya ningún objeto. Sólo se pueden mover objetos que tengan al menos un cuadro a su alrededor sin ningún objeto. En la figura 1 se muestra un ejemplo con un par de objetos que no pueden ser seleccionados por el jugador para moverlos en el tablero.

Una vez seleccionado el objeto a mover, el jugador debe indicar el cuadro en donde va a mover el objeto. Este cuadro debe estar vacío. Luego de que el objeto es movido,

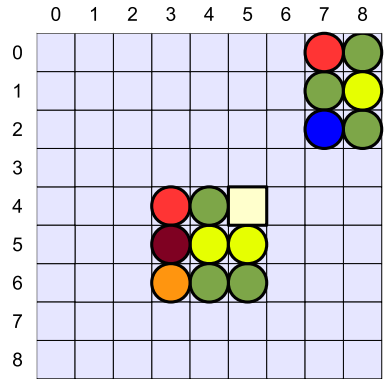
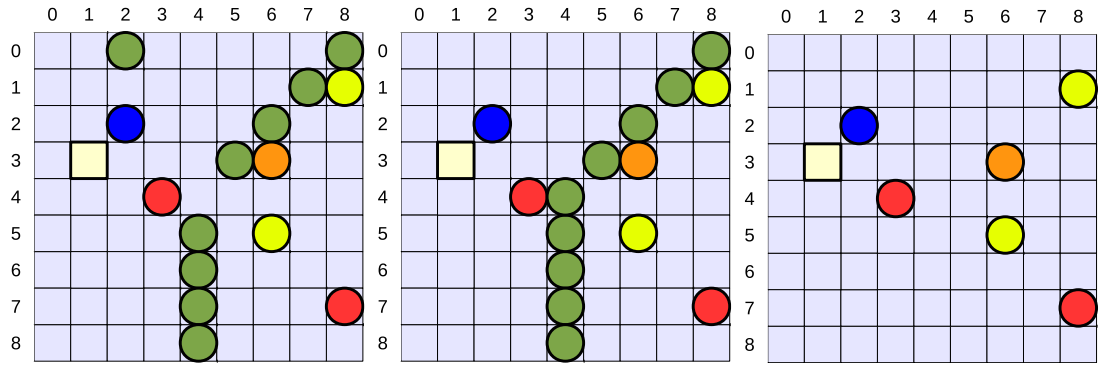


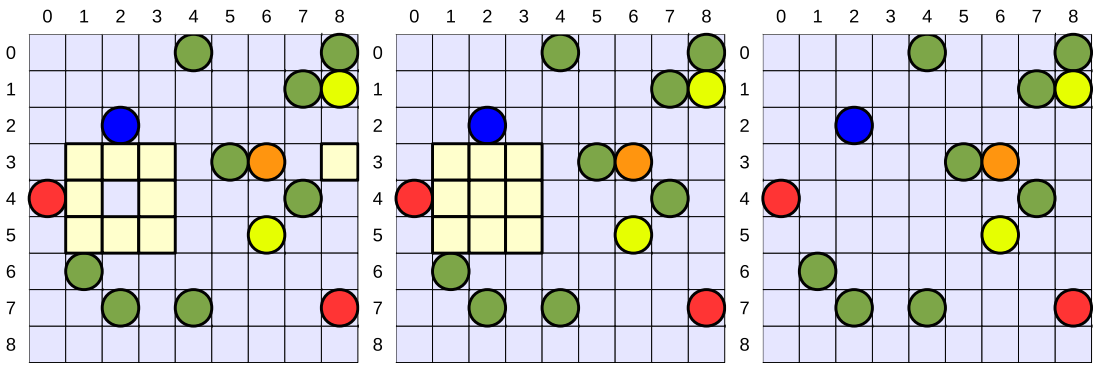
Figura 1: Ejemplo de objetos que no se pueden mover en el tablero. Las bolas amarillas que se encuentran en las posiciones (5,4) y (1,8), no pueden ser movidas por el jugador

la aplicación verifica si se ha formado una línea de 5 o más bolas, o un rectángulo de al menos de $2x2$ cuadrados. En el caso de que se hayan encontrado elementos correctamente alineados, entonces estos objetos desaparecen del tablero y el jugador gana un nuevo turno. En caso contrario se colocan en el tablero los tres objetos que en el panel se indican que son los próximos a agregar. En la figura 2 se muestra un ejemplo de un turno de un jugador en el cual se forman dos líneas de cinco bolas del mismo color, una diagonal y otra vertical. Estos objetos desaparecen del tablero y el jugador obtiene un nuevo turno para mover otro objeto. En la figura 3 se muestra un turno de un jugador que al mover un cuadrado forma un rectángulo de más de 2x2. Cuando se dice que el jugador gana un nuevo turno, se quiere indicar que no son colocados en la cuadrícula los objetos que son los próximos a agregar. En la figura 4 se muestra el caso de un turno de un jugador en el que no se produce ni una línea de 5 o más bolas ni un rectángulo de más de $2x2$ cuadrados. En este caso los objetos que están señalados como los próximos a agregar en el panel, son colocados en el tablero en cuadros que no contienen objetos, los cuales fueron escogidos de forma aleatoria.



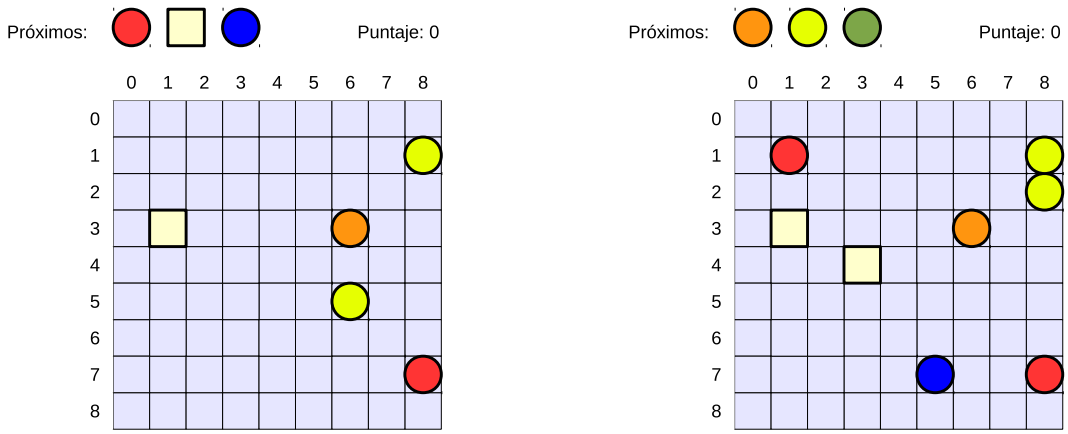
(a) Se escoge el objeto en (0,2) (b) Se mueve el objeto a (4,4) (c) Las líneas de 5 o más objetos desaparecen

Figura 2: Ejemplo de un turno de un jugador. En 2a se muestra el tablero al inicio del turno. El jugador escoge mover la bola verde de la posición (0,2) a la posición (4,4). En 2b se muestra la jugada efectuada por el jugador. Se forman dos líneas de cinco bolas verdes. Las líneas desaparecen en 2c



(a) Se escoge el objeto en (3,8) (b) Se mueve el objeto a (4,2) (c) Los rectángulos de 2x2 cuadros o más desaparecen

Figura 3: Ejemplo de un turno de un jugador. En 3a se muestra el tablero al inicio del turno. El jugador escoge mover el cuadrado de la posición (3,8) a la posición (4,2). En 3b se muestra que la jugada realizada por el jugador forma un cuadrado de 3x3. Como el cuadrado formado es mayor que 2x2 desaparece como se muestra en 3c



(a) (b)

Figura 4: Ejemplo de un turno de un jugador. En el 4a observamos un panel de juego al comienzo del turno de un jugador. El jugador escoge mover la bola amarilla de la posición (5,6) a la posición (2,8). En 4b vemos el resultado de la jugada. Como no se formó ninguna línea mayor de 5 objetos ni ningún cuadrado 2x2 o mayor, entonces los objetos que estaban apuntados como los próximos en 4a se agregan al tablero en posiciones aleatorias

La política a seguir para escoger los próximos objetos a agregar en el tablero, que se muestran en el panel, es la siguiente. Los dos primeros objetos son escogidos al azar. El tercer objeto va a ser del tipo que se encuentre en menor cantidad en el tablero. Si hay algún tipo de objeto que no se encuentre en el tablero, entonces el tercer objeto a colocar en la lista de próximos debe ser de este tipo.

Una vez que se haya completado una línea o un rectángulo en el tablero, estos desaparecen del mismo. Luego se debe proceder a contar los puntos obtenidos y sumarlos a los puntos acumulados por el jugador. El número de puntos que se obtienen en un turno va a depender del número de objetos que se eliminen del tablero en un turno. El objetivo del juego es acumular la mayor cantidad de puntos. En la tabla 1 se muestra el número de puntos que se obtiene por cada objeto alineado.

Número de objetos alineados	Puntos que se obtienen
4	5
5	10
6	12
7	18
8 o más	40

Tabla 1: Cantidad de puntos que obtiene un jugador por alinear un número determinado de objetos en un turno

El juego continúa su desarrollo y termina cuando todos los cuadros del tablero son ocupados por objetos. En la figura 5 se muestra un tablero que representa el final del juego. Llegado a este punto se debe comparar el puntaje obtenido por el jugador con el mejor puntaje obtenido por algún jugador en la historia que tiene grabada el juego. En caso de que el puntaje del jugador sea el mejor de todos los puntajes obtenidos, entonces se procede a guardar éste en la memoria que contiene el mejor puntaje obtenido hasta el momento.

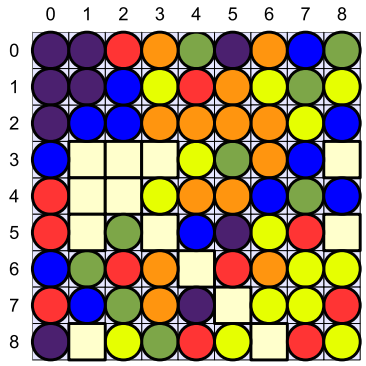


Figura 5: Ejemplo de un tablero totalmente lleno que representa el final del juego

1.2. Requerimientos y características de la aplicación a desarrollar

La aplicación *Lineas de colores y rectángulos* debe interactuar con el usuario a través de una interfaz gráfica y de la consola de comandos. La interfaz gráfica debe mostrar un panel que contenga:

- Una cuadrícula de $9x9$ que corresponde al tablero de juego. Cada fila y columna debe estar identificada con un número como se mostró en las figuras anteriores.
- La tres próximos objetos a agregar al finalizar el turno del jugador.
- El puntaje acumulado por el jugador.

Por la consola de comandos la aplicación le debe pedir al usuario el objeto a mover y la posición hacia donde quiere mover el objeto. Vamos a establecer la convención de que el usuario debe indicar las posiciones del tablero proporcionando primero el número de la fila y luego el número de la columna. Todas las posiciones que da el usuario deben ser validadas. En caso de que el usuario trate de hacer una jugada inválida, la aplicación debe indicarle el error y permitirle ingresar de nuevo otras coordenadas.

La interfaz gráfica se debe realizar haciendo uso de la librería Pygame [2]. Esta librería esta diseñada para desarrollar juegos, es robusta y está apliamente documentada.

Para poder hacer que la aplicación tenga memoria del mejor puntaje alcanzado hasta el momento, se va almacenar el valor del puntaje récord en un archivo llamado `record.txt`. Este archivo sólo tendrá escrita una línea con el valor del mejor puntaje. Usted puede tener creado este archivo, con un valor 0 en su primera línea, a la hora de iniciar la primera ejecución de su aplicación. Una vez finalizada una partida la aplicación debe abrir el archivo `record.txt` y leer el valor del puntaje récord. Luego se debe comparar el puntaje obtenido por el jugador con el puntaje récord. Si el puntaje obtenido por el jugador es menor que el puntaje récord, entonces se debe desplegar un panel en el que se muestre el puntaje del jugador y el puntaje récord y luego la aplicación finaliza. Si por el contrario el puntaje del jugador supera al puntaje récord, se debe desplegar un panel en el que se felicite al jugador por haber alcanzado una nueva marca y se deben mostrar el puntaje del jugador y del récord. Luego se debe guardar la nueva marca en el archivo `record.txt`. El archivo `record.txt` siempre tendrá una sola línea, por lo que en caso de ser modificado se debe eliminar el valor que se encontraba anteriormente. Una vez realizada esta acción la aplicación finaliza.

Recuerde que es importante la modularidad y el dividir el problema en varios problemas más pequeños. La claridad del código, el estilo, el nombre adecuado de las variables de los procedimientos y de las funciones, así como los comentarios que expliquen lo que hace el código en sectores complicados, tendrán un peso importante en la evaluación. Todo el código debe estar debidamente documentado. Para cada método se debe indicar, en comentarios, su descripción, la descripción de los parámetros de entrada y salida, y las precondiciones y postcondiciones en lenguage natural. También debe implementar las precondiciones, postcondiciones, invariantes y funciones de cota que correspondan.

2. Estructura de datos sugeridas

En el desarrollo de todo programa se debe escoger qué estructura de datos serán utilizadas para almacenar los datos que maneja el programa.

Para representar a la cuadrícula se sugiere crear una matriz llamada `tablero` de NxN, donde $N = 9$. Cada casilla de la matriz puede contener un número entero del 0 al 6. Los números representan cada uno de los objetos que son parte del juego. Es conveniente tener una estructura para almacenar las coordenadas del objeto a mover por el usuario y del cuadro hacia donde se va a mover ese objeto. Para almacenar una jugada válida podemos crear un arreglo de longitud cuatro, llamado `jugada`. Se tiene que `jugada[0]` y `jugada[1]` contendrían los valores de la fila y columna del objeto a mover. Asimismo

`jugada[2]` y `jugada[3]` guardarían los valores de la fila y columna del cuadro vacío hacia donde se dirigirá el objeto a mover. Debido a que los próximos objetos a colocar en la cuadrícula se deben estar mostrando constantemente en el panel, es bueno tener una estructura que represente a esos tres objetos. Se propone un arreglo de longitud tres, llamado `proximosObjetos` en donde cada posición tenga un número entero del 0 al 6, que identifica al próximo objeto a ser agregado en la cuadrícula, una vez terminado el turno del jugador. Para llevar el puntaje del jugador puede crear una variable entera llamada `puntaje` en el cuerpo del programa principal.

3. Esquema de la solución algorítmica sugerida

El diseño de la solución del juego lo hacemos aplicando la técnica del *análisis descendente*, mediante el cual un problema es dividido en varios subproblemas de menor complejidad.

En primer lugar recomendamos tener separados los componentes de la interfaz gráfica de los componentes que forman parte de la lógica del juego. La independencia de ambos componentes debe ser tal, que se podría sustituir la parte gráfica por una salida por consola sin alterar la componente lógica del juego. En la figura 6 se muestra una propuesta de esquema general de solución algorítmica para el juego.

```
program LineasRectangulosColores
|[
     Inicializar-Juego;
     Inicializar-Tablero;
     Obtener-Proximos-Objetos;
     do !Fin-Juego ->
         Mostrar-Estado-del-Juego;
         Obtener-Jugada-Valida;
         Mover-Objeto-Seleccionado;
         Mostrar-Estado-del-Juego;
         Procesar-Objetos-del-Tablero;
         if No-se-elimino-ningun-objeto-en-el-tablero ->
             Agregar-Proximos-Objetos;
             Obtener-Proximos-Objetos;
             Determinar-si-es-fin-juego
         | -> skip
         fi;
         Actualizar-Estado-del-Juego;
     od;
     Procesar-Final-Juego
]|
```

Figura 6: Esquema general de solución del juego

A continuación explicaremos el esquema presentado en 6. El subproblema `Inicializar-Juego` corresponde a la inicialización con valores adecuados de las variables del programa principal y de las estructuras de datos a utilizar. El siguiente subproblema `Inicializar-Tablero` coloca en el *tablero* los tres objetos iniciales con los que comienza el juego. Luego es necesario obtener los próximos objetos a ser agregados al tablero, una vez que termine el turno del jugador. De esto trata el subproblema `Obtener-Proximos-Objetos`. Podemos observar

que el próximo paso es un lazo **do od**. Esta lazo tiene como función permitir el desarrollo de las acciones del juego. En el lazo se van a se desarrollar los turnos del jugador hasta que el juego haya finalizado por haberse llenado el tablero de juego. El primer subproblema presentado dentro del lazo es `Mostrar-Estado-del-Juego`. Éste consiste en mostrar al usuario el estado del juego por medio de un panel gráfico. El siguiente paso en las acciones del juego es obtener del jugador, por medio de la consola, una jugada que cumpla con todas las condiciones anteriormente expuestas. El subproblema asociado a esta operación es `Obtener-Jugada-Valida`. Una vez validada la jugada del jugador, se procede a mover el objeto de una posición a otra. De esto se trata `Mover-Objeto-Seleccionado`. Una vez que se mueve el objeto de un cuadrado a otro, es necesario mostrar el nuevo estado del juego al usuario, por lo que es necesario aplicar las acciones de `Mostrar-Estado-del-Juego`. Llegado a este punto hay que procesar la jugada hecha por el usuario de la aplicación. Esto quiere decir que se tiene que determinar si el usuario formó alguna fila de 5 o más bolas o si construyó un rectángulo de al menos $2x2$ cuadrados. En caso de haberse producido alguna alineación correcta, entonces procede a eliminarla del tablero. Luego una vez eliminadas todas las alineaciones correctas se puede proceder a contar el número de objetos en el tablero. Esto se hace con el fin de poder comparar el número de objetos antes y después de la jugada del usuario. De esta manera podemos saber cuantos objetos exactamente se eliminaron en el turno o si en su defecto, en la jugada no se eliminó ningún objeto. Todas estas operaciones son llevadas a cabo en el subproblema `Procesar-Objetos-del-Tablero`. Llegado a este punto hay que determinar si se le va a permitir al jugador efectuar otro turno, o si se va agregar al tablero los objetos listados en próximos. Entonces tenemos que determinar si `No-se-elimino-ningun-objeto-en-el-tablero`. Si esto es cierto entonces se debe agregar los objetos que se encontraban en espera y debe crear una nuevo trío de objetos próximos a agregar. Esto es modelado en los subproblemas `Agregar-Proximos-Objetos` y `Obtener-Proximos-Objetos`. Tenemos que `Determinar-si-es-fin-juego` nos indica si el tablero se encuentra lleno de objetos y por lo tanto es el fin del juego o si por el contrario todavía existen cuadros sin objetos. El subproblema `Actualizar-Estado-del-Juego` involucra realizar básicamente dos acciones. La primera es determinar los puntos obtenidos por el jugador en su turno y la segunda es la actualización de las variables y las estructuras de datos para el próximo turno del jugador. Una de las variables que es necesario actualizar, es aquella que lleva el puntaje del jugador. El último subproblema es `Procesar-Final-Juego` e involucra acciones como comparar el puntaje obtenido con el puntaje récord, el mostrar el panel adecuado al jugador dependiendo si obtuvo un nuevo récord o no y todo lo concerniente a la manipulación adecuada del archivo `record.txt` para que siempre posea un sólo registro que es el valor récord.

4. Funciones y procedimientos recomendados

En esta sección se presentarán los encabezados de algunos procedimientos y funciones que están relacionados con los subproblemas expuestos anteriormente. Por convención los valores enteros que siguen a un arreglo o matriz en los parámetros de un procedimiento o función, corresponden a la longitud de los mismos.

El procedimiento 1 agrega al tablero inicial del juego, tres objetos escogidos al azar. Para escoger los próximos objetos a ser agregados en el tablero podemos hacer uso de un procedimiento como el 2. Con el tablero se puede determinar cuál es el tipo de objeto que se encuentra en menor cantidad.

Para poder `Mostrar-Estado-del-Juego` hacemos uso de una interfaz gráfica. A la variable que contiene la interfaz gráfica la llamamos `panel` y suponemos que es un tipo

Procedimiento 1

```
proc inicializarTablero:(in out tablero : array[][] of int, in N : int)
{pre ...}{post ...}
```

Procedimiento 2

```
proc obtenerProximosObjetos:(in out proximosObjetos : array[] of int, in M : int,
                             in tablero : array[][] of int, in N : int)
{pre ...}{post ...}
```

especial que depende de la librería gráfica a usar, a ese tipo de datos lo llamamos `Pantalla`. El procedimiento 3 nos permitiría, por medio de un panel, observar la cuadrícula que representa el tablero, los próximos objetos a agregar y los puntos acumulados por el jugador.

Procedimiento 3

```
proc mostrarJuego:(in out panel : Pantalla, in proximosObjetos : array[] of int,
            in M : int, in tablero : array[][] of int, in N : int, puntaje : int)
{pre ...}{post ...}
```

Para obtener una jugada que sea válida se puede implementar un procedimiento como 4, en el que las coordenadas del objeto a mover y de su punto de destino, se encuentren en el arreglo jugada.

Procedimiento 4

```
proc obtenerJugadaValida:(in out jugada : array[] of int, in M : int,
                          in tablero : array[][] of int, in N)
{pre ...}{post ...}
```

Luego de obtener una jugada válida por parte del jugador, el programa puede proceder a mover el objeto seleccionado, de un punto a otro en el tablero. Para ello presentamos el procedimiento 5.

Procedimiento 5

```
proc moverObjetoSeleccionado:(in out tablero : array[][] of int, in N : int,
                              in jugada : array[] of int, in M)
{pre ...}{post ...}
```

El subproblema `Procesar-Objetos-del-Tablero` puede ser resuelto con dos procedimientos y una función. Se tiene un procedimiento 6 que determina todas las líneas de más de 5 objetos de tipo círculo del mismo color que hay en el tablero y las elimina del mismo. Luego se puede aplicar el procedimiento 7 que elimina del tablero todos los cuadrados de tamaño mayor que $2x2$. Por último se puede aplicar la función 1 para obtener el número de objetos resultantes, después de aplicar procedimientos 6 y 7. Con el valor del número de objetos resultantes y el número de objetos en el tablero antes de las llamada a los procedimientos 6 y 7, se puede obtener el número de objetos que logró eliminar el jugador en

su turno.

Procedimiento 6

```
proc procesarCirculos:(in out tablero : array[][] of int, in N : int)
{pre ...}{post ...}
```

Procedimiento 7

```
proc procesarCuadrados:(in out tablero : array[][] of int, in N : int)
{pre ...}{post ...}
```

Función 1

```
fun obtenerNumerosDeObjetos:(in out tablero : array[][] of int, in N : int) -> int
{pre ...}{post ...}
```

Si en turno del jugador se determina que no se ha eliminado ningún objeto del tablero, entonces se procede a agregar los objetos de la lista de próximos objetos al tablero. Para ello se propone el procedimiento 8

Procedimiento 8

```
proc agregarObjetos:(in out tablero : array[][] of int, in N : int,
                     in proximosObjetos : array[] of int, in M : int)
{pre ...}{post ...}
```

Para saber si la cuadrícula tiene en todas sus posiciones objetos podemos emplear una función como la 2 que verifique si todas las posiciones del tablero están ocupadas o no.

Función 2

```
fun esFinDeJuego:(in out tablero : array[][] of int, in N : int) -> boolean
{pre ...}{post ...}
```

5. Condiciones de la entregas

La realización del proyecto se hará por equipos de hasta dos personas que pertenezcan a la misma sección de laboratorio. La entrega del proyecto se llevará a cabo en dos partes.

5.1. Entrega 1 (valor 15 puntos)

La primera entrega consiste en implementar en Python el juego completo exceptuando dos subproblemas: `Procesar-Objetos-del-Tablero` y `Procesar-Final-Juego`. Estos dos subproblemas forman parte del esquema general de solución que se muestra en la figura 6, y fueron explicados en la sección *Esquema de la solución algorítmica sugerida.*

Sin esos subproblemas la aplicación resultante es un juego en el cual el jugador en su turno mueve un objeto de una posición a otra, pero nunca se le elimina una línea o un rectángulo, ya que no se verifica ni elimina ninguna de esas alineaciones. Siempre después del turno del jugador, la aplicación agrega los objetos de la lista de próximos a la cuadrícula de juego. Una vez que la cuadrícula este llena la aplicación termina. No se debe verificar si hay un nuevo récord o no, ya que todo lo de la manipulación de archivos corresponde al subproblema `Procesar-Final-Juego`.

La primera entrega debe realizarse el día martes 09 de Junio de 2015, antes de la 9:00 am, y consiste en los siguientes elementos:

- Un sobre sellado y identificado con su nombre, carné y profesor de laboratorio, que debe contener dos documentos:
  - El código impreso de su programa.
  - La "Declaración de Autenticidad para Entregas" firmada por los integrantes del equipo.
- Un archivo comprimido del tipo `tgz` con el código fuente de su proyecto, que debe ser entregado en la página del curso en el Aula Virtual. El nombre del archivo deber ser **Entrega1ci2691AbrJun15-X-Y.tgz** donde **X** y **X** son los números de carné de los autores del proyecto.

5.2. Entrega 2 (Valor 25 puntos)

Para la segunda entrega debe proporcionar un juego que cumpla con todos los requerimientos que le han sido solicitados. Además debe realizar un informe que contenga los siguientes puntos:

- **Portada:** Debe incluir los nombres y los números de carnes de los integrantes del grupo, así como el nombre del profesor encargado.
- **Introducción:** Breve descripción del problema resuelto, cuales son los objetivos planteados y el alcance de la solución. Debe indicar cual es el contenido del informe.
- **Diseño:** Explicación del análisis descendente hecho para resolver el problema. Descripción de las estructuras de datos utilizadas. Explicación de los subproblemas claves del programa.
- **Estado Actual del Proyecto**: Se debe indicar el grado de operatividad del programa, es decir, si funciona perfectamente o no; en caso negativo indique cuales son los errores.
- **Conclusiones:** Resultados obtenidos, dificultades presentadas, su visión sobre la experiencia del desarrollo del proyecto y las recomendaciones que considere necesarias para mejorar los próximos cursos de algoritmos I.
- **Bibliografía**

La segunda entrega debe realizarse el día martes 23 de junio de 2015, antes de la 9:00 am. Debe entregar al docente de su sección un sobre manila debidamente identificado que contenga el código del proyecto impreso, el informe y la "Declaración de Autenticidad para Entregas" firmada por los integrantes del equipo. Además el código fuente debe ser entregado en la página del curso, contenido en archivo llamado **Entrega2ci2691AbrJun15-X-Y.tgz** donde **X** y **X** son los números de carné de los autores del proyecto.

6. Consideraciones Finales

- Es obligatorio que el equipo trabaje de forma coordinada. El profesor podrá hacer un interrogatorio individual a cada integrante del equipo, y el no responder adecuadamente a las preguntas puede tener como consecuencias penalizaciones en la nota del proyecto.
- Cualquier error que sea hallado en este enunciado, así como cualquier tipo de observación adicional sobre el proyecto, serán publicadas como fe de erratas en la página web del curso.
- No debe haber copia, ni intercambio de información específica, ni ayuda detallada entre los equipos. El incurrir en cualquiera de las acciones descritas anteriormente tendrá como consecuencia sanciones severas.
- La plataforma en que se debe ejecutar su proyecto es Linux. Si el código de alguna entrega no puede ser ejecutado por el interpretador, entonces la entrega será calificada con cero.
- El no cumplimento de todos los requerimientos podría resultar en el rechazo de su entrega.

Referencias

[1] GNOME. Five o more. `https://wiki.gnome.org/Apps/Five%20or%20more`, 2015.

[2] PYGAME.ORG. Pygame. `http://www.pygame.org/news.html`, 2015.

[3] WIKIPEDIA. Color lines. `http://en.wikipedia.org/wiki/Color_Lines`, 2015.

Guillermo Palma / gvpalma@usb.ve / Mayo 2015