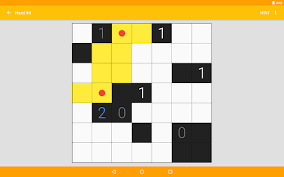
Proyecto *Akari Puzzle o* *Light Up*

Fecha de entrega de la versión 1: 11 de abril de 2023 a las 9:00



(Imagen extraída de Google Play)

El objetivo del proyecto es desarrollar un programa que permita jugar a una versión del juego [*Light Up o Akari Puzzle*](https://play.google.com/store/apps/details?id=cz.danurbanek.lightup&hl=es_419&gl=US). Iremos desarrollando diversas versiones que permitirán poner en práctica los distintos conocimientos de la asignatura.

# El juego en detalle

*Light up* es un juego para un jugador consistente en iluminar los espacios libres de un tablero rectangular o cuadrado a base de colocar bombillas.

El tablero está dividido en celdas que se pueden ser de tres tipos:

* **Celdas pared**. No pueden tener bombillas ni ser iluminadas. Además, impiden que el haz de luz de una bombilla siga su camino.
* **Celdas con bombilla**. Tienen una bombilla y, por lo tanto, están iluminadas.
* **Celdas libres**. Pueden estar o no iluminadas, dependiendo de que el haz de luz de alguna bombilla las alcance.

Las acciones que puede ir realizando el jugador son *poner bombilla* -con la consiguiente actualización de la iluminación de las celdas alcanzables por los haces de luz de la misma- o *quitar bombilla* -que hace que la celda en la que se encuentra vuelva a estar libre (y no iluminada), y que desaparezca la iluminación de todas las celdas alcanzadas por sus haces de luz, salvo que dichas celdas estén también iluminadas por otra bombilla).

A la hora de poner bombillas hay que tener en cuenta tres restricciones:

1. Por ***ahorro energético*** no se permite colocar una bombilla en una celda iluminada.
2. Por ***imposibilidad técnica*** las bombillas no se pueden colocar en celdas tipo pared.
3. Por ***diseño de la instalación eléctrica*** hay celdas tipo pared que deben tener un cierto número de bombillas (de 0 a 4) colocadas en sus celdas adyacentes (aquellas con las que comparte una cara, no las celdas vecinas de las esquinas). Por ejemplo, una celda tipo pared con un 4 deberá tener 4 bombillas alrededor de ella, colocadas en sus celdas adyacentes; una celda con un 0 no debe tener ninguna bombilla en sus celdas adyacentes.

Una vez colocada una bombilla en una celda libre dicha bombilla ilumina la celda donde se ha puesto y también todas aquellas libres alcanzables por los haces de luz que emite en horizontal y en vertical, iluminando así toda la fila y la columna en la que se encuentra a menos que un haz sea bloqueado por una celda tipo pared (en ese caso, el haz no ilumina las celdas situadas al otro lado de la celda tipo pared). El juego ***finaliza*** cuando toda la estancia está iluminada cumpliendo las restricciones indicadas.

# Versión 1: Jugando a *Light Up*

La aplicación para jugar a *Light Up* comenzará cargando un tablero desde fichero. A continuación, el jugador irá quitando y poniendo bombillas, introduciendo la posición (fila y columna) de la celda que quiere modificar en cada acción. Si la posición corresponde a una celda libre se colocará una bombilla en ella, salvo que la celda esté previamente iluminada, y si corresponde a una celda con bombilla, la quitará, actualizando en ambos casos la iluminación de otras celdas, según se ha indicado anteriormente. Introducir una posición no válida o correspondiente a una celda tipo pared o a una celda iluminada sin bombilla dentro, no surte ningún efecto.

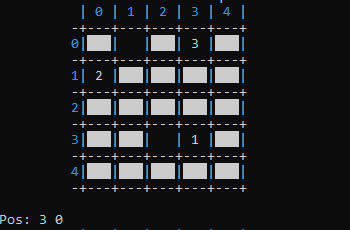
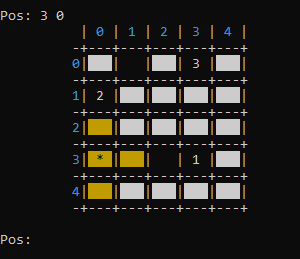
El programa termina cuando todas las celdas libres estén iluminadas y se cumplan todas las restricciones (las relativas a la colocación de bombillas y las relacionadas con el *diseño de la instalación eléctrica*). Asimismo, el jugador debe poder abandonar el juego o ‘resetearlo’ (quitar todas las bombillas puestas) en cualquier momento. Cuando el programa termine deberá informar al usuario de cuántas acciones se han ejecutado.

En esta versión de *Light up* vamos a visualizar el tablero y las distintas celdas como se indica a continuación (ver figura inferior): las celdas libres son amarillas o blancas, dependiendo de que estén iluminadas o no, las celdas con bombilla son amarillas con un asterisco rojo y las celdas tipo pared son negras.

En la figura puedes ver que aparecen en azul los valores de cada fila y columna del tablero (empezando en 0). Hay una única bombilla colocada en la celda situada en (3, 0), que aparece en fondo amarillo y con un ‘\*’. Su haz de luz vertical está bloqueado, en dirección norte, por la celda tipo pared situada en (1, 0), y su haz de luz horizontal está bloqueado en la dirección este por la celda pared colocada en (3, 2). Los haces de luz, evidentemente, también son bloqueados por los bordes del tablero. Observa que las casillas pared aparecen en negro cuando no tienen restricciones de bombillas --por ejemplo, en la posición (0, 1)-- o aparecen en fondo negro con un número en blanco que indica el número de bombillas que deben rodearla, como ocurre en la posición (0, 3).

Asimismo, puedes ver en la figura que cuando el usuario introduce la posición de la celda donde quiere llevar a cabo una acción debe proporcionar dos números; el primero corresponde a la fila y el segundo a la columna. Para **abandonar** el juego, introducirá la posición (-1, 0); y para **resetear** el tablero, introducirá posición

(-1, -1).

# Detalles de implementación

La aplicación contará con los siguientes tipos de datos:

* Para representar el estado de cada celda del tablero emplearemos la siguiente pareja de tipos

typedef enum tTipo {PARED, BOMBILLA, LIBRE};

typedef struct {

tTipo tipo;

int numBombillas;

} tCelda;

con el siguiente convenio:

* + Si la celda es una pared, el campo tipo tendrá el valor enumerado PARED y el campo numBombillas contendrá un valor en el intervalo [-1,4], donde -1 significa que es una celda que no tiene restricción en cuanto el número de bombillas que la pueden rodear, mientras que los valores del 0 al 4 indican el número de bombillas que la tienen que rodear (situadas en celdas adyacentes).
  + Si la celda contiene una bombilla, el campo tipo tendrá el valor enumerado BOMBILLA, y en ese caso el valor de campo numBombillas es irrelevante.
  + Si la celda está libre, el campo tipo tendrá el valor enumerado LIBRE, y el campo numBombillas indicará el número de bombillas que la iluminan.
* Para representar el estado del tablero emplearemos las siguientes constantes y tipos:

const int MAX\_FILS = 10;

const int MAX\_COLS = 10;

typedef struct {

int nFils, nCols;

tCelda tablero[MAX\_FILS][MAX\_COLS];

} tTablero;

de manera que el tablero será una variable de tipo tTablero, es decir, una matriz de dimensiones variables, con tamaño máximo 10 x 10, cuyas celdas contienen el estado de cada celda del tablero. Permite disponer de tableros cuadrados y rectangulares.

* Para representar las direcciones hacia las que se extiende un haz de luz definiremos el tipo typedef enum tDir {NORTE, SUR, ESTE, OESTE};

Estos tipos y las operaciones asociadas a ellos se distribuirán en los siguientes módulos:

* ***Módulo celda***

Este módulo contendrá las definiciones de los tipos tTipo y tCelda, y las siguientes operaciones:

* char celdaToChar(const tCelda& celda): Devuelve el carácter que representa a la celda. Concretamente, si la celda es de tipo BOMBILLA, entonces devuelve ‘\*’; si es una pared numerada, devuelve su número de bombillas. En cualquier otro caso devuelve un blanco. Esta función se utilizará para mostrar el tablero.
* tCelda charToCelda(char c): Dado un carácter, devuelve la celda que representa. Concretamente:
* ‘X’: la celda es una pared sin restricciones
* Carácter numérico: la celda es una pared, y el carácter numérico representa el número de bombillas que la deben rodear.
* ‘.’: celda libre (en ella se pueden colocar bombillas).
* ‘\*’: celda que contiene una bombilla.

Esta función se utilizará durante la carga del tablero.

* bool esPared(const tCelda& c): Devuelve true si y sólo si la celda es tipo pared.
* bool esParedRestringida(const tCelda& c): Devuelve true si la celda es tipo pared y tiene restricciones sobre el número de bombillas que deben rodearla.
* int numParedRestringida(const tCelda& c): Devuelve el número de bombillas que deben rodear a la celda c, de tipo pared.
* bool esBombilla(const tCelda& c): Devuelve true si y sólo si la celda es tipo bombilla.
* bool estaApagada(const tCelda& c): Devuelve true si la celda está libre y su número de bombillas es 0.
* bool estaIluminada(const tCelda& c): Devuelve true si la celda está libre y su número de bombillas es mayor que 0.
* void apagaCelda(tCelda& c): Pone su tipo a LIBRE y su número de bombillas a 0.
* void actualizaIluminaciónCelda(tCelda& c, bool iluminar): Incrementa o decrementa el número de bombillas que la iluminan.
* void ponBombilla(tCelda& c): coloca una bombilla en la celda.
* ***Módulo tablero***

Este módulo contendrá la definición del tipo tTablero, las constantes MAX\_FILS y MAX\_COLS y las siguientes operaciones:

* int getNumFilas(const tTablero& tab): Devuelve el número de filas del tablero.
* int getNumCols(const tTablero& tab): Devuelve el número de columnas del tablero.
* tCelda celdaEnPos(const tTablero& tab, int x, int y): Devuelve la celda que ocupa la posición (x,y) en el tablero. El parámetro x es la fila e y es la columna.
* void ponCeldaEnPos(tTablero& tab, int x, int y, const tCelda& c): Coloca la celda c en la posición (x,y) del tablero.
* bool leerTablero(ifstream& archivo, tTablero& tab): Carga el tablero de archivo.
* void mostrarTablero(const tTablero& tab): Muestra en consola el tablero.

Como alternativa a las dos últimas funciones puedes sobrecargar los operadores ‘>>’ y ‘<<’:

* ifstream& operator>>(ifstream& archivo, tTablero& tab): carga el tablero de archivo.
* ostream& operator<<(ostream& out, const tTablero& tab): muestra en consola el tablero.

El formato del fichero de entrada y el resultado de mostrarlo por consola son análogos a lo siguiente:

|  |  |
| --- | --- |
| tablero.txt  5 5  .X.3.  2....  .....  ..X1.  .....  C:\Users\Puri\AppData\Local\Microsoft\Windows\INetCache\Content.Word\Captura de pantalla 2023-03-01 135427.png |  |

* + - La primera línea contiene dos números enteros que representan el número de filas y el número de columnas, en ese orden, que tiene el tablero.
    - A continuación, hay tantas líneas como filas tiene el tablero. En cada línea hay tantos caracteres como número de columnas tiene el tablero. El i-ésimo carácter de una fila representa el estado de la i-ésima celda de la fila, de acuerdo con el siguiente convenio:
      * Un **.** representa una celda libre.
      * Una **X** representa una pared sin restricción en el número de bombillas que se pueden colocar en las celdas adyacentes.
      * Un carácter numérico del 0 al 4 representa una pared con restricción en el número de bombillas que puede haber en las celdas adyacentes.

**De forma opcional** podemos admitir además información sobre bombillas que debemos colocar en el tablero. En este caso el formato del fichero es igual que el anterior pero, una vez descrito el tablero, aparecen más líneas. Por ejemplo:

tablero.txt

5 5

.X.3.

2....

.....

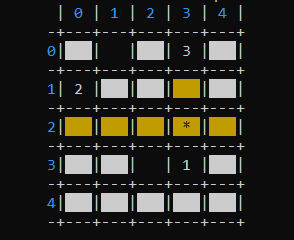
..X1.

.....

1

2 3

La primera fila, después de la descripción del tablero, indica el número de bombillas que hay que colocar (en este caso, 1). A continuación, se enumeran las posiciones en que hay que colocarlas (una posición por línea). En el ejemplo, hay que colocar una bombilla en la posición (2,3) e iluminar las casillas correspondientes alcanzadas por la bombilla. Puedes asumir que las posiciones son correctas, es decir cumplen todas las restricciones del juego. Por lo tanto, si procesamos el fichero anterior, visualizaremos un tablero de la forma:



* ***Módulo reglasJuego***

Este módulo contiene la lógica del juego, es decir las reglas que determinan cómo jugar, y que ya se han explicado con anterioridad. Las funciones públicas que ofrece este módulo son las siguientes:

* bool estaTerminado(tTablero const& tab): Indica si el tablero está completamente iluminado y cumple todas las restricciones, es decir, si ha finalizado el juego.
* bool esPosQuit(int x, int y): comprueba si la posición introducida por el usuario es la (-1, 0), que es la correspondiente a abandonar el juego.
* void ejecutarPos(tTablero& tab, int x, int y): Se encarga de poner o quitar, según proceda, una bombilla en la posición (x,y) del tablero si se puede (i.e., es una posición válida), siguiendo las reglas explicadas anteriormente, y ajustar la iluminación de todas las celdas alcanzables desde la celda que se encuentra en dicha posición. Para implementar esta función necesitarás definir funciones auxiliares (privadas); usa tantas como necesites para que el código sea legible. Si la posición indica resetear o abandonar el juego, se actuará en consecuencia.
* ***Función main***

La función main estará en un fichero .cpp de nombre main.cpp. Dicha función, tras cargar el tablero, ejecutará un bucle en el que se pedirá una posición por teclado al usuario, y se invocará a la función ejecutaPos del módulo reglasJuego, hasta que o bien termine el juego o el usuario desee finalizar la partida. Además, en dicho bucle tendrás que contabilizar el número de pasos que ha realizado el jugador antes de terminar la partida con éxito o bien abandonarla.

Si decides implementar la parte opcional, ten en cuenta que una vez que leas el tablero, tendrás que leer la información sobre la modificación de algunas casillas, que pasarán a tener bombilla. Cada vez que leas una posición, tendrás que llamar a ejecutarPos del módulo reglasJuego, para que ponga la bombilla e ilumine las celdas correspondientes. Así funcionará correctamente, ya que llamarás a ejecutarPos con una celda libre (sin bombillas ni iluminada), y por lo tanto colocará la bombilla e iluminará las celdas correspondientes.

El siguiente módulo contiene definiciones de colores tanto de la fuente como del fondo. Puedes utilizarlo para pintar el tablero (no tiene un cpp asociado).

**“colores.h”**

//===Color fuente===//

#define BLACK "\x1B[30m"

#define RED "\x1b[31m"

#define GREEN "\x1b[32m"

#define YELLOW "\x1b[33m"

#define BLUE "\x1b[34m"

#define MAGENTA "\x1b[35m"

#define CYAN "\x1b[36m"

#define WHITE "\x1B[37m"

#define ORANGE "\x1B[38;2;255;128;0m"

#define ROSE "\x1B[38;2;255;151;203m"

#define LBLUE "\x1B[38;2;53;149;240m"

#define LGREEN "\x1B[38;2;17;245;120m"

#define GRAY "\x1B[38;2;176;174;174m"

#define RESET "\x1b[0m"

//===Color fondo===//

#define BG\_BLACK "\x1B[40m"

#define BG\_RED "\x1B[41m"

#define BG\_GREEN "\x1B[42m"

#define BG\_YELLOW "\x1B[43m"

#define BG\_BLUE "\x1B[44m"

#define BG\_MAGENTA "\x1B[45m"

#define BG\_CYAN "\x1B[46m"

#define BG\_WHITE "\x1B[47m"

#define BG\_ORANGE "\x1B[48;2;255;128;0m"

#define BG\_LBLUE "\x1B[48;2;53;149;240m"

#define BG\_LGREEN "\x1B[48;2;17;245;120m"

#define BG\_GRAY "\x1B[48;2;176;174;174m"

#define BG\_ROSE "\x1B[48;2;255;151;203m"

Ejemplo de uso:

cout << BG\_YELLOW << RED << " \* " << RESET;

Pintaría una celda con bombilla (fondo amarillo y asterisco rojo). RESET restablece la combinación de colores por defecto de la consola.