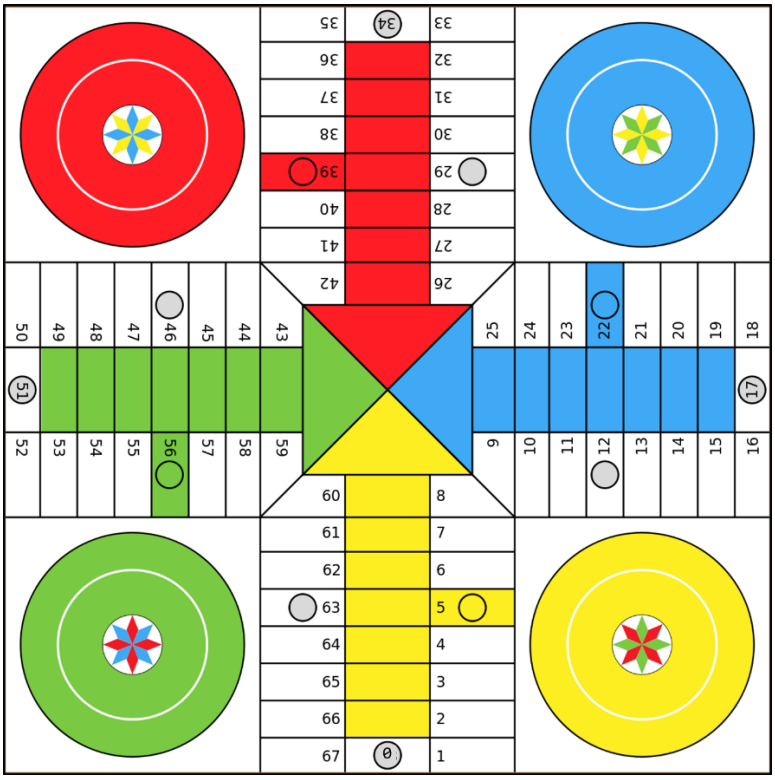
Proyecto *Parchís*
Versión 2

Fecha de entrega: 7 de enero



El objetivo del proyecto es permitir a un usuario humano jugar al parchís contra otros jugadores controlados por la computadora, por medio de un programa en C++ del que ya hemos desarrollado la primera versión. En esta segunda versión permitiremos que cuatro jugadores humanos jueguen una partida de parchís completa.

1. El juego: Parchís

Recordemos los elementos y las reglas de nuestro juego del parchís*:

- Hay cuatro colores: amarillo, azul, rojo y verde.
- Cada jugador juega con cuatro fichas de su color asignado.
- Se utiliza un dado para determinar los movimientos para las fichas.
- *Casas*: zonas cuadradas con círculos en color que hay en las esquinas del tablero; las fichas de cada jugador comienzan encerradas en su casa.
- Alrededor de las casas hay 68 casillas, numeradas del 0 al 67. No puede haber más de dos fichas en una misma casilla.

* Recuerda que en nuestro caso la casilla 68 está numerada como 0, de forma que la siguiente a la 67 es la 0.

- *Salida*: casillas rectangulares en color que hay junto a cada casa (5, 22, 39 y 56).
- Los jugadores sacan una ficha a su salida cuando el dado marca un cinco.
- *Seguros*: casillas rectangulares con un círculo gris (0, 12, 17, 29, 34, 46, 51 y 63), además de las salidas.
- *Metas*: casillas triangulares en color que hay en el centro del tablero.
- *Zanatas*: casillas junto a las subidas a meta (0, 17, 34 y 51); cada ficha accede a la subida a meta desde la zanata de su jugador.
- *Puente*: dos fichas de igual color en una misma casilla (si están en un seguro es una barrera que no se puede franquear).
- Si un jugador debe sacar una ficha de casa y en la salida ya hay dos fichas, alguna de ellas de otro jugador, se manda la última que haya llegado a casa.
- Las fichas se mueven siempre en el sentido creciente del número de casilla, siendo un circuito continuo (se pasa de nuevo a la casilla 0 desde la casilla 67).
- Una ficha que ya haya salido de casa y no esté bloqueada se podrá mover tantas casillas como indique el dado.
- Un jugador gana cuando todas sus fichas han llegado a la meta; las fichas sólo pueden llegar a la meta con el número de movimientos exacto.
- *Comer*: una ficha llega a una casilla, que no sea un seguro, en la que hay otra ficha de un contrario, enviando esta última a su casa y ganando una jugada extra de 20 movimientos.
- Cuando una ficha llega a la meta, el jugador gana una jugada extra de 10 movimientos.
- Cuando todas las fichas de un jugador están fuera de casa los seises se cuentan como sietes.
- Cuando un jugador juega un seis, tiene derecho a una nueva jugada. Pero al tercer seis consecutivo se manda a casa a la última ficha movida por el jugador, excepto si ese último movimiento fue en la subida a meta.

2. La segunda versión

En esta segunda versión jugaremos con normalidad al parchís:

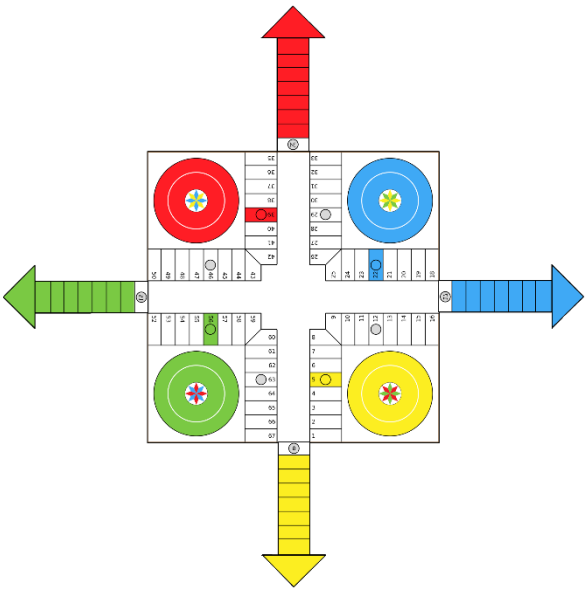
- Juegan los cuatro jugadores.
- Cada uno de los jugadores dispone de cuatro fichas.
- El jugador amarillo será siempre el de la primera posición, seguido del azul, el rojo y el verde. Se elegirá aleatoriamente quién de ellos juega en primer lugar.
- Las tiradas del dado se generarán aleatoriamente.
- Las reglas son las que se han explicado en el primer apartado.

Esta vez tenemos que contemplar las subidas (caminos) a meta, que son ocho casillas de color, la última de las cuales (la triangular) es la propia meta. Las fichas que van llegando a la meta se mantienen ahí. El primer jugador que consigue meter sus cuatro fichas en la meta es el que gana. Y, como ya sabemos, las fichas deben llegar a la meta con el número exacto de movimientos que les resten (hay versiones del juego en las que las fichas *rebotan* si mueven más de lo necesario, volviendo hacia atrás por el camino a meta; en nuestro caso no será así).

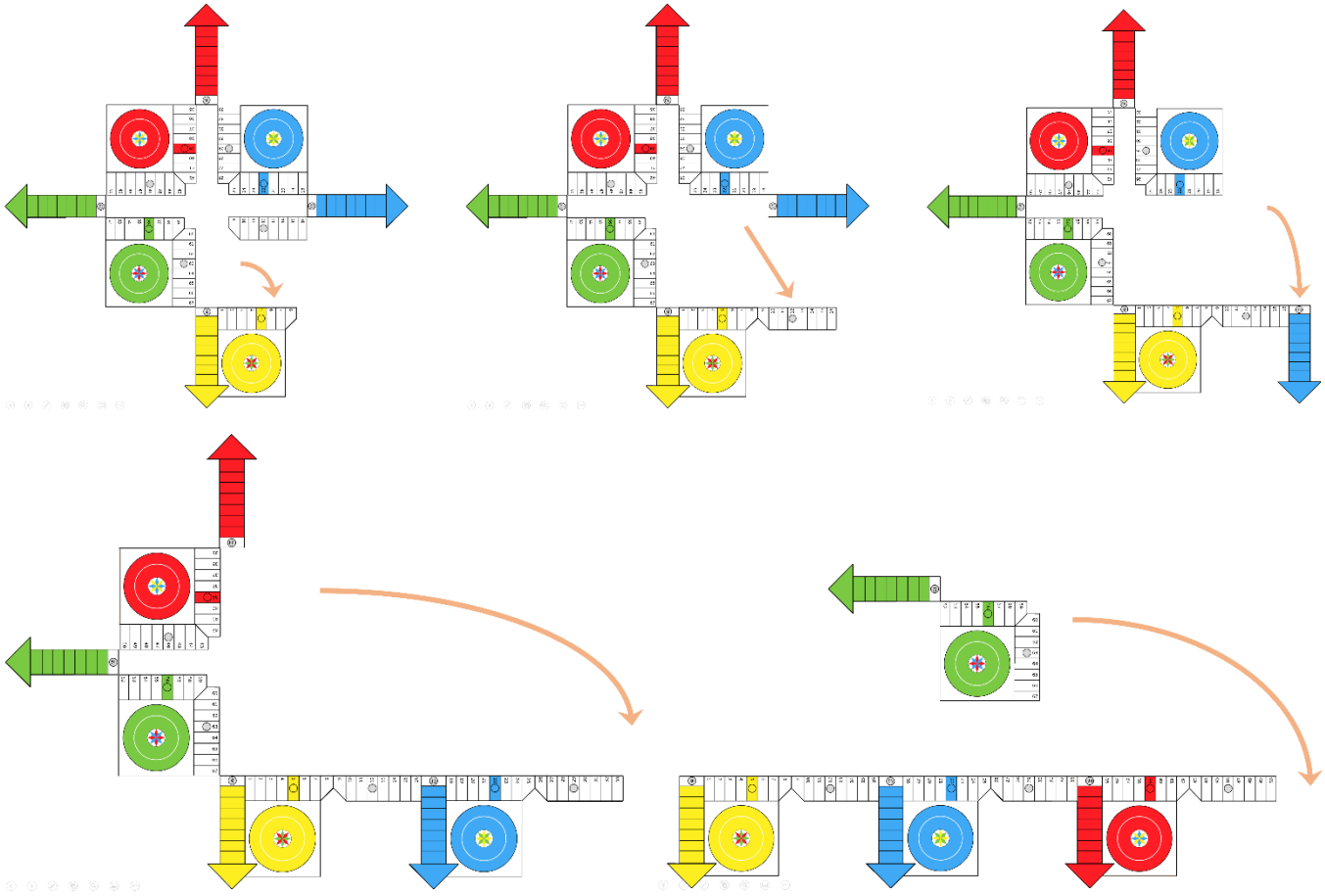
3. Representación del tablero

Vamos a linealizar el tablero y añadir los caminos a meta de la forma que sigue (recuerda que hemos cambiado la casilla 68 a casilla 0).

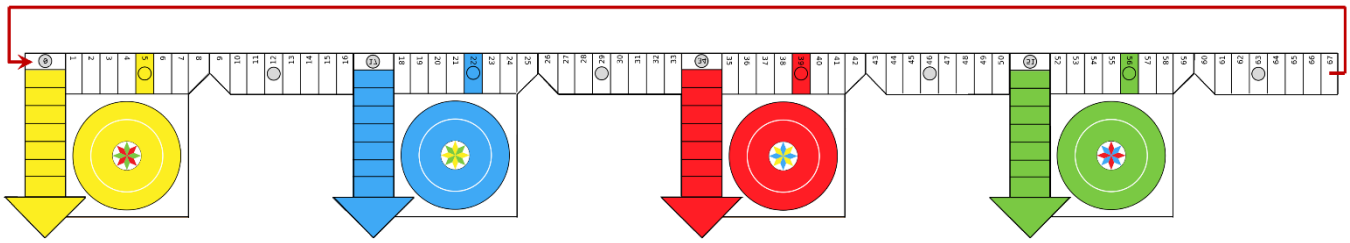
En primer lugar sacamos las subidas a meta hacia el exterior:



A continuación vamos colocando en línea las casillas normales, tal como hicimos en la primera versión, pero manteniendo hacia abajo los caminos a meta y las casas:



De esta forma conseguimos una representación que es mucho más manejable por un programa que se ejecuta en una ventana de consola:



Vamos a representar en el programa ese tablero de forma que resulte relativamente sencillo generar la situación del juego en cada momento. Inicialmente todas las fichas de los cuatro jugadores estarán en sus casas:

```
0000000000111111111122222222223333333333444444444455555555556666666667
0123456789012345678901234567890123456789012345678901234567890123456789
>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>

o----o------o----o----o------o----o----o------o----o----o------o----

V>>>>^>>>>>>>>>>>>V>>>>^>>>>>>>>>>>>V>>>>^>>>>>>>>>>>>V>>>>^>>>>>>>>>>>>
VV    1           VV    1           VV    1           VV    1
VV    2           VV    2           VV    2           VV    2
VV    3           VV    3           VV    3           VV    3
VV    4           VV    4           VV    4           VV    4
VV                VV                VV                VV
VV                VV                VV                VV
VV                VV                VV                VV
..                ..                ..                ..
..                ..                ..                ..
```

Ahora ya hay sitio para dos fichas en cada casilla (por debajo y por encima de la línea de guiones y círculos; si hay una sola ficha, estará abajo). Los círculos siguen indicando las casillas que son seguras. Las puntas de colores que apuntan hacia arriba (^) indican la casilla de salida de las fichas de ese color y las que apuntan abajo (V) indican la zanata, es decir, el camino hacia la meta de las fichas de ese color. En los caminos a meta hay siete casillas en las que igualmente caben dos fichas (del mismo jugador esta vez) y la meta está representada por cuatro puntos, cada uno para una de las fichas del jugador.

Cada ficha sale de casa a su casilla de salida (cuando el dado marca un 5) y a partir de ahí avanza hacia la derecha, de forma que si sale por la parte derecha (casilla 67) sigue por la izquierda (casilla 0). Se aplicarán todas las reglas del juego sobre posibilidad de avanzar, comida de fichas de otros jugadores, contar movimientos extra si se come (20) o si otra ficha llega a la meta (10), repetición de jugada si sale 6, etcétera.

Las fichas de cada jugador estarán numeradas (1 a 4) para poder identificarlas.

A continuación se muestra el aspecto del tablero tras una serie de jugadas y el menú que pregunta al usuario la ficha que quiere mover.

```
000000000011111111112222222222333333333344444444445555555556666666666
012345678901234567890123456789012345678901234567890123456789012345678
>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
                                                                      1
o----o------o----o----o------o----o----o------o----o----o------o----
3          4     3                    3   4              4          4
V>>>>^>>>>>>>>>>>V>>>>^>>>>>>>>>>>V>>>>^>>>>>>>>>>>V>>>>^>>>>>>>>>>>
VV   ^           VV   ^           VV   1           VV   ^
2V   ^           VV   ^           VV   ^           VV   2
VV   ^           12   ^           VV   ^           VV   ^
VV   ^           VV   ^           VV   ^           VV   ^
VV               VV               VV               VV
VV               VV               2V               VV
VV               VV               VV               VV
1.               ..               ..               ..
..               ..               3.               ..

Turno para el jugador Verde... Sale un 2
Por favor, elige la ficha que quieres mover...
1: De la casilla 56 a la casilla 58
3: De la casilla 0 a la casilla 2
4: De la casilla 56 a la casilla 58
Ficha (0 para salir):
```

4. Detalles de implementación

Modifica el tipo enumerado `tColor` de forma que incluya también el color gris, así como un valor `Ninguno` que utilizaremos para indicar que no hay ficha en una casilla:

```
typedef enum {Amarillo, Azul, Rojo, Verde, Ninguno, Gris} tColor;
```

El jugador en la primera posición será siempre el amarillo, seguido del azul, el rojo y el verde. Así se puede usar `tColor(jugador)` para saber el color del `jugador` (de 0 a 4).

Declara constantes para el número de jugadores (4), el número de fichas por jugador (4) y el número de casillas en la calle del tablero (68).

Otros tipos de datos

Declara el tipo `tCasillas` como un array de 68 colores (`tColor`). Si hay una ficha en una casilla, el valor en el array será el color de la ficha; si no hay ficha su valor será el color `Ninguno`.

Necesitamos dos arrays de este tipo, uno para cada hueco de las casillas. Si sólo hay una ficha en la casilla, el color de esta estará siempre en el primer array. Si hay dos fichas, la última que haya llegado a la casilla será la que esté en el segundo array.

Declara el tipo `tFichas` como un array de 4 enteros, siendo cada uno el lugar en el que se encuentra una de las fichas de un jugador: -1 si está en casa, 0 a 67 si está por la calle y 101 a 108 si está subiendo a la meta, siendo la 108 la propia meta.

Declara el tipo `tJugadores` como un array bidimensional de 4 jugadores (`tFichas`), siendo cada elemento las fichas de un jugador.

Otros datos generales

Aparte del array de jugadores y los arrays del tablero, para el juego necesitaremos saber:

- Cuántos seises seguidos han salido.
- Cuál ha sido la última ficha movida por el jugador que saca seises (por si se va a casa).
- El número de jugador al que le toca jugar (0 a 3).
- La tirada actual del dado.

Subprogramas

De la versión anterior ya tienes algunas funciones de utilidad:

```
bool esSeguro(int casilla);
int salidaJugador(int jugador);
int zanataJugador(int jugador);
string colorACadena(tColor color);
char colorALetra (tColor color);
```

Desarrolla nuevos subprogramas de utilidad:

- `void iniciar(tJugadores jugadores, tCasillas calle1, tCasillas calle2, tColor& jugadorTurno)`: Inicializa el generador de números aleatorios, con `srand()`, y los elementos del juego (fichas y tablero). También pone el color de la fuente en gris sobre fondo blanco (*ver más adelante*) y devuelve el jugador que empieza a jugar (aleatorio de 0 a 3).
- `bool puente(const tCasillas calle1, const tCasillas calle2, int casilla)`: Devuelve `true` si en la `casilla` hay dos fichas del mismo color, o `false` en otro caso.
- `int cuantasEn(const tFichas jugador, int casilla)`: Devuelve el número de fichas del `jugador` que hay en una `casilla` (valores válidos -1 casa, 0 a 67 ó 101 a 108).
- `int primeraEn(const tFichas jugador, int casilla)`: Menor índice de ficha del `jugador` que está en esa `casilla` (-1 si no hay ninguna ahí). Se usa para `mostrar` y para sacar de casa la primera ficha del jugador (`saleFicha`).
- `int segundaEn(const tFichas jugador, int casilla)`: Mayor índice de ficha del `jugador` que está en esa `casilla` (-1 si no hay ninguna ahí). Se usa para `mostrar`, para determinar qué ficha se manda a casa (`aCasita`) y en el manejo de los puentes (`abrirPuente` y `procesa6` donde se gestiona la apertura forzosa de un puente).
- `void saleFicha(tJugadores jugadores, tColor jugadorTurno, tCasillas calle1, tCasillas calle2): )`: Sale de casa una ficha del `jugador`, concretamente la de menor índice que esté en casa. La que sale va a la calle 1 y si hubiera otra se coloca en la calle 2.
- `void aCasita(tJugadores jugadores, int casilla, tCasillas calle1, tCasillas calle2)`: Manda a casa a la última ficha que llegó a la `casilla`, la que está

en `calle2`. En caso de que ambas fichas de la casilla sean del mismo jugador, se irá a casa la de mayor índice de las dos.

- `bool todasEnMeta(const tFichas jugador)`: Devuelve `true` si las cuatro fichas del `jugador` están en la meta (casilla 108) y `false` en otro caso.

Desarrolla también estos subprogramas para la dinámica del juego:

- `void abrirPuente(tJugadores jugadores, int casilla, int casilla2, int& premio, tColor jugadorTurno, int& ultimaFichaMovida, tCasillas calle1, tCasillas calle2)`: Abre el puente del jugador en `casilla`, llevando la ficha con mayor índice en el puente a la `casilla2` y ganando el `premio` si se come otra ficha. El movimiento en sí mismo se realizará invocando desde aquí a la función `mover()` para realizar el movimiento correspondiente.
- `bool procesa5(tJugadores jugadores, tColor jugadorTurno, int& premio, bool& pasaTurno, tCasillas calle1, tCasillas calle2)`: Intenta sacar una ficha de casa del jugador, si queda alguna. La función devuelve `true` si se ha podido sacar ficha o `false` en caso contrario. Si ya hay dos fichas en la casilla de salida, entonces si ambas son suyas no sale ninguna de casa. Si de las dos una es de otro jugador, se come esa, y si las dos son de otros jugadores, se come la última que haya llegado. Si se come otra ficha, se gana un premio de 20 movimientos extra y el jugador conserva el turno. Estas son las únicas responsabilidades de esta función. Si no se puede sacar ninguna ficha por el motivo que sea (porque no hay fichas que sacar o por bloqueo de la salida con dos fichas propias), entonces no será responsabilidad de esta función intentar mover cinco con alguna ficha en la calle. Ni lo hará ella, ni ella invocará a otra para que lo haga.
- `bool procesa6(tJugadores jugadores, tColor jugadorTurno, int& premio, bool& pasaTurno, int& seises, int& ultimaFichaMovida, int& tirada, tCasillas calle1, tCasillas calle2)`: Actualiza la `tirada` a 7 si no quedan fichas en casa y gestiona los dos casos especiales con una tirada de 6: manda `ultimaFichaMovida` a casa si es el tercer seis consecutivo, o abre obligatoriamente un puente que tenga el jugador (si tiene dos, el usuario ha de decidir cuál). La función devolverá `true` si el movimiento se ha llevado a cabo de alguna de esas dos maneras (incluso si tocaba abrir un puente pero ninguna ficha pudo moverse para hacerlo). El jugador actual mantendrá el turno si no es el tercer 6. Si no hay ningún movimiento obligatorio (porque no hay puente o porque hay más de un puente posible para abrir), entonces el movimiento no se llevará a cabo en esta función (en este caso, el movimiento se llevará a cabo más adelante en la función `jugar()`, función que no será invocada desde aquí). De manera similar a la función `procesa5()`, fíjate en que estas son las únicas responsabilidades de esta función.
- `bool jugar(tJugadores jugadores, tColor jugadorTurno, int& premio, bool& fin, int& seises, int& ultimaFichaMovida, int tirada, tCasillas calle1, tCasillas calle2)`: Esta función se invocará cuando no haya salido una ficha de casa con un 5, no se haya ido una ficha a casa con un 6, ni haya habido que abrir un puente

con un movimiento obligatorio. La función preguntará al usuario qué ficha quiere mover `tirada` movimientos (de entre las que realmente puedan hacer tal movimiento) hasta que escoja una ficha válida para mover. Entonces se llevará a cabo el movimiento de dicha ficha invocando a la función `mover()` (que también se ocupará de gestionar los premios por comer o llegar a meta, así como de mandar a casa a la ficha que corresponda en el primer caso). Además, se actualizará la `ultimaFichaMovida`. Si el usuario introduce 0 en lugar de un número de ficha, `fin` valdrá true, lo que provocará que el resto del programa finalice el juego. La función devuelve `true` si el turno debe pasar al siguiente jugador.

- `bool puedeMover(const tJugadores jugadores, tColor jugadorTurno, int ficha, int& casilla, int tirada, const tCasillas calle1, const tCasillas calle2)`: Indica la `casilla` a la que puede ir la `ficha` del jugador con esa `tirada`. Devuelve `true` si se puede mover la `ficha` y `false` en caso contrario.
- `void mover(tJugadores jugadores, tColor jugadorTurno, int ficha, int casilla, int& premio, int& ultimaFichaMovida, tCasillas calle1, tCasillas calle2)`: Ejecuta el movimiento de la `ficha` (debe usarse después de haber usado `puedeMover()`), conforme a lo explicado anteriormente.

Desarrolla, si quieres, otros subprogramas que puedas considerar de utilidad.

Importante: ¡No se pueden usar estructuras (`struct`) en esta versión de la práctica!

Colores en la consola

El cambio de colores en las ventanas de consola se puede conseguir por medio se *secuencias de escape*, que son secuencias de caracteres especiales que el sistema de visualización de la computadora interpreta no como literales, sino como instrucciones.

Como habrás visto, en nuestra representación el fondo es blanco (un poco *sucio*) sobre el que imprimimos los elementos generales en gris y los de cada jugador en su correspondiente color.

Para cambiar, por ejemplo, el texto a azul sobre fondo blanco basta con enviar a `cout` la secuencia de escape `"\x1b[34;107m"`. Para verde sobre fondo blanco será `"\x1b[32;107m"`.

Se usará el siguiente procedimiento para cambiar de color. Dado que las secuencias de escape funcionan tanto en las consolas de Windows, como en los terminales de Linux, la rutina es válida para ambos sistemas operativos.

```
void setColor(tColor color) {
   switch (color) {
   case Amarillo:
      cout << "\x1b[33;107m";
      break;
   case Azul:
      cout << "\x1b[34;107m";
      break;
   case Rojo:
      cout << "\x1b[31;107m";
```

```
      break;
   case Verde:
      cout << "\x1b[32;107m";
      break;
   case Gris:
   case Ninguno:
      cout << "\x1b[90;107m";
      break;
   }
}
```

Visualización del tablero

La visualización del tablero se realiza con el subprograma `mostrar()`:

```
void mostrar(const tJugadores jugadores, const tCasillas calle1,
             const tCasillas calle2);
```

Los subprogramas `mostrar()` y `setColor()` se proporcionan en el espacio virtual.