Universidad de La Laguna

Trabajo de fin de grado

Implementación del juego de Buscaminas basado en hardware con Arduino y display táctil

Autor: Adrian Monzón Rodríguez
Tutor: Evelio Jose González González

Resumen

Vamos a desarrollar el juego del Buscaminas utilizando una placa Arduino Uno R3 y una pantalla táctil WHADDA. El objetivo del juego es despejar el campo sin revelar ninguna mina para ganar; de lo contrario, se pierde. Comenzaremos diseñando nuestra pantalla principal y programando el botón de inicio, de manera que al pulsarlo nos lleve a la pantalla de selección de menú. En esta pantalla de selección de menú, elegiremos una dificultad que determinará el modo de juego, con un campo y un número de minas predefinidos para cada nivel. Al finalizar una partida, ya sea ganando o perdiendo, el juego nos devolverá a la pantalla de inicio.

Abstract

We are going to develop the Minesweeper game using an Arduino Uno R3 board and a WHADDA touch screen. The objective of the game is to clear the field without revealing any mines to win; otherwise, you lose. We will start by designing our main screen and programming the start button, so that once pressed, it takes us to the menu selection screen. In this menu selection screen, we will choose a difficulty level that will determine the game mode, with a predefined field and a set number of mines for each level. Once a game is finished, whether by winning or losing, the game will return to the start screen.

ÍNDICE

- Introducción

Buscaminas

El Buscaminas es un videojuego para un jugador creado por Curt Johnson y Robert Donner en 1989. El objetivo del juego es despejar un campo de minas sin detonar ninguna.

Algunas de las casillas tendrán un número, el cual indica la cantidad de minas que hay en las casillas circundantes. Así, si una casilla tiene el número 3, significa que de las ocho casillas que hay alrededor (si no es en una esquina o borde) hay 3 con minas y 5 sin minas. Si se descubre una casilla sin número indica que ninguna de las casillas vecinas tiene mina y éstas se descubren automáticamente. Si se descubre una casilla con una mina se pierde la partida.

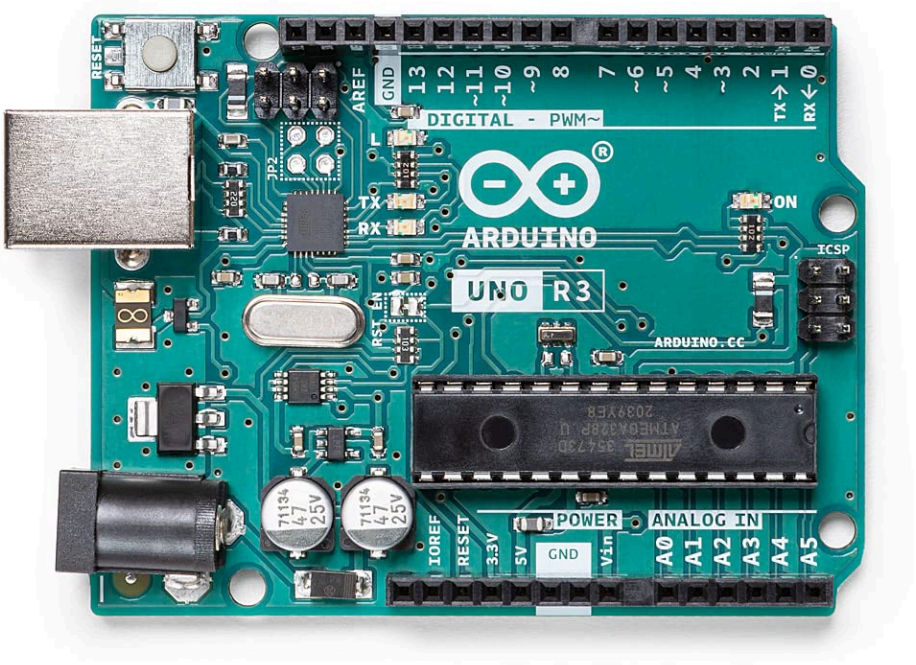
Se puede poner una marca en las casillas que el jugador piensa que hay minas para ayudar a descubrir las que están cerca.

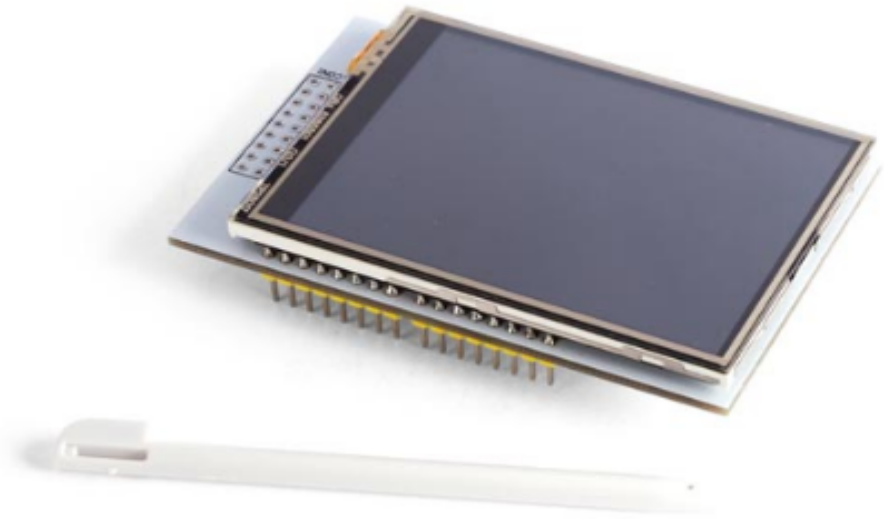
Este proyecto incluirá la implementación de una interfaz principal y una pantalla de selección de dificultad. Tras la elección del nivel de dificultad, se generará un tablero de juego cuyo tamaño y número de minas se ajustarán en función de la dificultad seleccionada.

Material empleado

Para este proyecto utilizaremos la placa Arduino Uno R3, en la que se integrará una pantalla táctil de la marca Velleman.

La placa Arduino Uno R3 es un microcontrolador basado en el ATmega328P, conocido por su facilidad de uso y versatilidad en proyectos de electrónica y programación. Esta placa ofrece 14 pines digitales de entrada/salida, 6 entradas analógicas, y una frecuencia de reloj de 16 MHz. La pantalla táctil de la marca Velleman, compatible con Arduino, permitirá una interacción intuitiva del usuario con la interfaz del juego, facilitando la selección de opciones y la visualización del tablero.





La elección de la placa Arduino Uno R3 se debe a su amplia difusión y soporte dentro de la comunidad de desarrolladores, lo cual asegura una abundancia de recursos y

documentación para solucionar posibles problemas durante el desarrollo. Asimismo, la pantalla táctil de Velleman se seleccionó por su compatibilidad y facilidad de integración con la placa Arduino, lo que simplifica la implementación de una interfaz gráfica interactiva.

Para llevar a cabo esta programación, utilizaremos el entorno de desarrollo conocido como Arduino IDE. Este entorno de desarrollo integrado (IDE) se utiliza para escribir el código en una computadora y cargarlo en la placa física. El Arduino IDE destaca por su simplicidad, lo cual ha sido un factor crucial en la popularidad de Arduino.

Entre las ventajas del Arduino IDE se encuentran su simplicidad de uso, su compatibilidad con múltiples sistemas operativos (Windows, macOS y Linux), y una amplia colección de bibliotecas predefinidas que facilitan la programación de componentes específicos, como la pantalla táctil Velleman. Además, la comunidad activa de usuarios y desarrolladores de Arduino contribuye con una vasta cantidad de recursos y ejemplos de código que pueden ser utilizados como referencia.

• Objetivos

El objetivo principal de este proyecto es aplicar y consolidar conocimientos en programación mediante la creación de una versión simplificada del juego Buscaminas. La elección de este juego se justifica por su popularidad y su simplicidad inherente, lo cual permite enfocar el desarrollo en aspectos clave como la gestión de la interfaz gráfica y la generación procedural de tableros.

El objetivo técnico de integrar la placa Arduino Uno R3 con la pantalla táctil Velleman es proporcionar una plataforma robusta y eficiente para el desarrollo del videojuego. Esta integración permitirá crear una experiencia de usuario fluida y responsiva, crucial para la jugabilidad y la interacción con el Buscaminas.

• Planificación.

Antes de comenzar con el desarrollo del videojuego, es esencial verificar que todas las funcionalidades del hardware estén operativas. Este paso es crucial para asegurar que no habrá problemas técnicos durante el desarrollo y que la integración entre el hardware y el software será fluida. A continuación, se describen los pasos a seguir para esta comprobación y el desarrollo subsecuente del videojuego.

Comprobación de Funcionalidades del Hardware

Para asegurarnos de que nuestro hardware funciona correctamente, utilizaremos programas de muestra proporcionados en la página web del fabricante de la pantalla táctil. Estos programas de muestra nos permitirán:

1. Verificar la funcionalidad táctil.
2. Comprobar la calidad de la pantalla y la precisión del toque.
3. Asegurar la correcta comunicación entre la pantalla y el sistema principal.

Una vez que hayamos confirmado que el hardware funciona perfectamente, podemos proceder con el desarrollo del videojuego.

Pasos para el Desarrollo del Videojuego

El desarrollo del videojuego se dividirá en varias etapas claras y definidas para asegurar un proceso ordenado y eficiente. Estas etapas son:

- **Diseñar y Programar la Pantalla Principal del Videojuego**
    - **Objetivo:** Crear una interfaz inicial donde el jugador pueda iniciar el juego.
    - **Tareas:**
        - Diseño gráfico de la pantalla principal.
        - Implementación de botones táctiles, como "Start" y "Settings".
        - Programación de la funcionalidad para iniciar el juego al tocar el botón "Start".

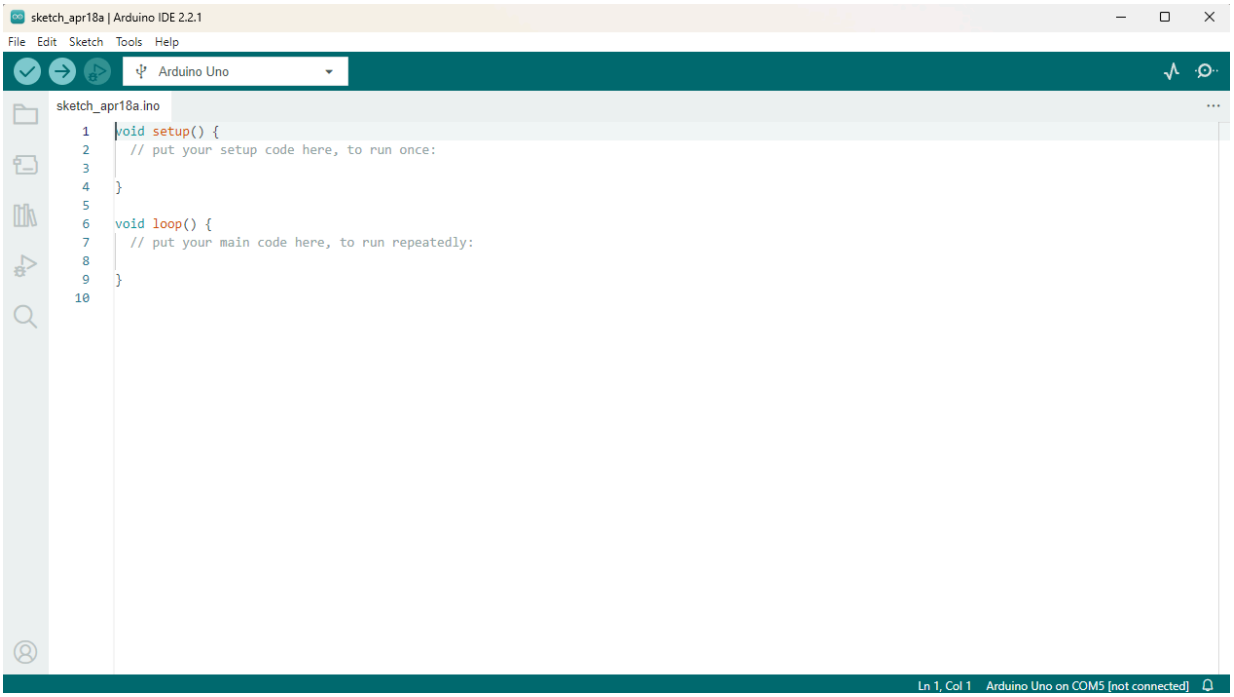
- **Diseñar y Programar la Pantalla de Selección de Dificultad**
    - **Objetivo:** Permitir al jugador seleccionar el nivel de dificultad antes de comenzar el juego.
    - **Tareas:**
        - Diseño gráfico de la pantalla de selección de dificultad.
        - Implementación de botones táctiles para las opciones de dificultad: fácil, medio y difícil.
        - Programación de la transición a la pantalla de juego correspondiente según la dificultad seleccionada.

- **Desarrollar el Juego Adaptado a la Dificultad Elegida**

- **Objetivo:** Implementar la lógica del juego para que se adapte a la dificultad seleccionada por el jugador.
- **Tareas:**
  - Definir las configuraciones del tablero y el número de minas para cada nivel de dificultad.
  - Programar la lógica del juego, incluyendo la colocación de minas, el conteo de minas adyacentes y la revelación de celdas.
  - Implementar la funcionalidad de bandera para marcar las celdas sospechosas de contener minas.
  - Desarrollar la lógica de victoria y derrota, incluyendo la transición al estado GAME_OVER y la visualización de mensajes correspondientes.

• Resultado y análisis.

Comenzaremos instalando el entorno de desarrollo Arduino IDE. Una vez finalizada la instalación, procederemos a abrir el programa, encontrándonos con el siguiente entorno:



Podemos destacar dos puntos importantes en este entorno de programación: el *void setup()* y el *void loop()*, los cuales explicaremos a continuación. La función *void setup()*

es la parte del código que se ejecuta una sola vez, al iniciar el programa. Esta sección se utiliza para configurar los parámetros iniciales del videojuego, como la inicialización de la pantalla táctil y la configuración de los pines de la placa Arduino. Una vez que se ha ejecutado *void setup()* y hemos iniciado nuestra primera partida, esta parte del código no se ejecutará nuevamente, incluso si volvemos a la pantalla principal. Por otro lado, la función *void loop()* es responsable de la ejecución continua del programa. Este bucle se repite indefinidamente, gestionando la lógica del juego y actualizando la interfaz de usuario hasta que se desconecte la alimentación de la placa Arduino.

Una vez comprendido el funcionamiento del entorno de programación Arduino IDE, procederemos a ejecutar código de prueba para verificar que tanto nuestra placa Arduino como la pantalla táctil funcionan correctamente. Para ello, visitaremos la página web de WHADDA, fabricante de nuestra pantalla táctil, y en la sección de descargas, obtendremos la carpeta que contiene todas las pruebas disponibles para la pantalla. Utilizaremos dos pruebas específicas: la primera, denominada "simple test", nos permitirá verificar que la pantalla imprime correctamente, y la segunda, llamada "Touch", simula de manera sencilla la herramienta Microsoft Paint para comprobar la funcionalidad táctil de la pantalla.

La primera prueba, 'simple test', cargará una serie de gráficos y textos en la pantalla táctil para verificar su capacidad de impresión. Esperamos ver gráficos claros y nítidos sin artefactos visuales, lo cual confirmará que la pantalla funciona correctamente. La segunda prueba, 'Touch', simula una herramienta básica de dibujo similar a Microsoft Paint. Esta prueba nos permitirá verificar la precisión y la responsividad del sensor táctil. Durante esta prueba, dibujaremos en la pantalla y observaremos si las trazas se corresponden con nuestros movimientos táctiles.

Una vez finalizadas las pruebas, procederemos a programar y diseñar nuestro juego. Primero, definiremos las dimensiones de nuestra pantalla, así como las funciones táctiles correspondientes. Esto es fundamental para asegurar que los elementos gráficos y las interacciones táctiles se ajusten correctamente a las características de la pantalla. A continuación, definiremos las constantes utilizadas por la biblioteca Touchscreen e incluimos las librerías necesarias en el código así como una función llamada GameState para diferenciar entre los distintos estados de nuestro juego.

```
#include <MCUFRIEND_kbv.h>
#include <TouchScreen.h>
#include <stdio.h>

MCUFRIEND_kbv tft;

#define BOXWIDTH 150
#define BOXHEIGHT 70

// Define la configuración de la pantalla táctil
#define TS_MINX 100
#define TS_MINY 70
#define TS_MAXX 920
#define TS_MAXY 900

// Definiciones para las constantes utilizadas por la biblioteca TouchScreen
#define XP 8
#define YP A3
#define XM A2
#define YM 9

enum GameState {
  MAIN_MENU,
  SECOND_SCREEN,
  GAME_PLAY,
  GAME_OVER
  // Agrega otros estados
};

```

Ahora que hemos definido las dimensiones de nuestra pantalla, las bibliotecas necesarias y nuestros *GameState*, procederemos al diseño y programación de nuestra pantalla principal. Esta será la pantalla de título del juego e incluirá un botón táctil que, al ser pulsado, nos llevará a la siguiente pantalla, que será la de selección de dificultad.

Utilizaremos las siguientes funciones para diseñar nuestra pantalla principal. Empezaremos por dibujar los elementos clásicos del Buscaminas, como una bandera y una mina, así como el título del juego. Posteriormente, implementaremos el botón táctil y su correspondiente función de interacción.

```
312  void drawMine(int x, int y) {
313    // Dibuja una mina en la posición especificada (x, y)
314    tft.fillCircle(x, y, 10, 0x0000);  // Círculo negro
315
316    // Dibuja líneas adicionales para simular la forma de una mina
317    tft.drawFastHLine(x - 15, y, 30, 0x0000);  // Línea horizontal
318    tft.drawFastVLine(x, y - 15, 30, 0x0000);  // Línea vertical
319    tft.drawLine(x - 11, y - 11, x + 11, y + 11, 0x0000);  // Línea diagonal
320    tft.drawLine(x - 11, y + 11, x + 11, y - 11, 0x0000);  // Línea diagonal
321  }
322
323  void drawFlagLeft(int x, int y) {
324    // Dibuja un triángulo rojo a la izquierda de la mina
325    tft.fillTriangle(x, y - 25, x - 20, y - 15, x, y - 5, 0xF800);  // Triángulo rojo
326
327    // Dibuja el asta de la bandera (línea negra) desde el vértice inferior derecho del triángulo hacia abajo
328    tft.drawFastVLine(x, y - 5, 10, 0x0000);  // Línea vertical negra
329  }
330
331  void drawFlagRight(int x, int y) {
332    // Dibuja un triángulo rojo a la derecha de la mina
333    tft.fillTriangle(x, y - 25, x + 20, y - 15, x, y - 5, 0xF800);  // Triángulo rojo
334
335    // Dibuja el asta de la bandera (línea negra) desde el vértice inferior izquierdo del triángulo hacia abajo
336    tft.drawFastVLine(x, y - 5, 10, 0x0000);  // Línea vertical negra
337  }
```

Estas son las funciones que utilizaremos para nuestra pantalla principal. Ahora procederemos a integrarlas en el programa principal. Este programa invocará las funciones de diseño de la pantalla principal y, desde nuestro programa se mostrará el título del juego. A continuación, se presenta el código del programa principal que estructura el flujo del juego desde la pantalla de inicio hasta la selección de dificultad.

```
409  void loop() {
410
411    TSPoint touch = ts.getPoint();
412    pinMode(XM, OUTPUT);
413    pinMode(YP, OUTPUT);
414    int tam;
415    int16_t adjustedX = map(touch.y, TS_MINY, TS_MAXY, 0, tft.width());
416    int16_t adjustedY = map(touch.x, TS_MINX, TS_MAXX, 0, tft.height());
417
418    if (currentState == MAIN_MENU) {
```

```
418  if (currentState == MAIN_MENU) {
419    //tft.fillScreen(0xFFFF);  // Llena la pantalla de blanco
420    tft.setTextSize(3);
421    tft.setTextColor(0x0000);  // Color blanco
422    tft.setCursor(75, 20);
423    tft.print("Buscaminas");
424    // Dibuja una mina debajo del título
425    drawMine(160, 90);
426
427    // Dibuja la bandera izquierda
428    drawFlagLeft(110, 100);  // Ajusta la posición (x,y)
429
430    // Dibuja la bandera derecha
431    drawFlagRight(210, 100);  // Ajusta la posición (x,y)
432
433    // Calcula la posición para centrar el cuadrado "Start"
434    int startX = (tft.width() - BOXWIDTH) / 2;
435    int startY = (tft.height() - BOXHEIGHT) / 2 + 50;  // Ajuste vertical hacia abajo
436
437    // Dibuja el cuadrado "Start"
438    tft.drawRect(startX, startY, BOXWIDTH, BOXHEIGHT, 0x0000);  // Color negro
439    tft.setTextSize(3);
440    tft.setTextColor(0x0000);  // Color negro
441    tft.setCursor(startX + 30, startY + 25);
442    tft.print("Start");
```

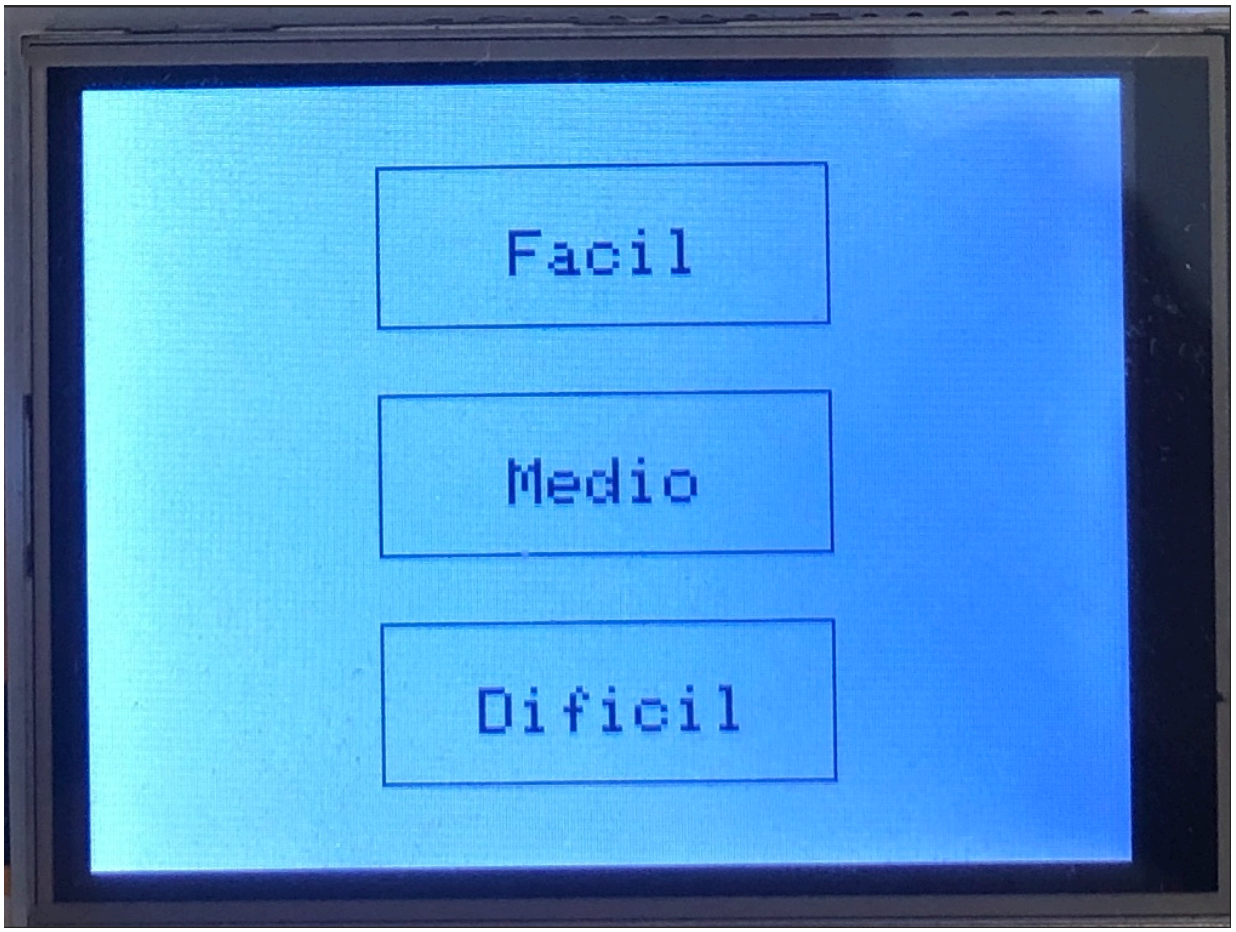
Observamos que dentro de nuestro programa principal estamos definiendo y programando las funciones táctiles de nuestra pantalla, así como estableciendo las coordenadas de interacción. Además, es importante destacar que hemos configurado nuestro juego para iniciar siempre en el estado `MAIN_MENU`. Esto implica que, al finalizar cualquier partida, el juego volverá automáticamente al menú principal. Esto asegura una estructura coherente y permite a los usuarios navegar fácilmente entre las distintas secciones del juego.

A continuación, procederemos a implementar la funcionalidad táctil en el botón 'Start'. Para lograrlo, utilizaremos la siguiente sentencia en nuestro código.

```
444   if (adjustedX > startX && adjustedX < startX + BOXWIDTH && adjustedY > startY && adjustedY < startY + BOXHEIGHT && touch.z > 0) {
445     currentState = SECOND_SCREEN;
446     drawDifficultyMenu();
447   }
```

Una vez que el usuario pulse este botón, el programa lo detectará y cambiará nuestro estado a *SECOND_SCREEN*. Además, se invocará la función *drawDifficultyMenu()*, la cual se define como sigue:

```
339  void drawDifficultyMenu() {
340    tft.fillScreen(0X8430);  // Llena la pantalla de gris
341    tft.setTextSize(2);
342    tft.setTextColor(0x0000);  // Color negro
343
344    // Dibuja los rectángulos de selección de dificultad
345    tft.drawRect(90, 25, 140, 50, 0x0000);  // Fácil
346    tft.setCursor(131, 45);
347    tft.print("Facil");
348
349    tft.drawRect(90, 95, 140, 50, 0x0000);  // Medio
350    tft.setCursor(130, 115);
351    tft.print("Medio");
352
353    tft.drawRect(90, 165, 140, 50, 0x0000);  // Difícil
354    tft.setCursor(120, 185);
355    tft.print("Dificil");
356  }
```

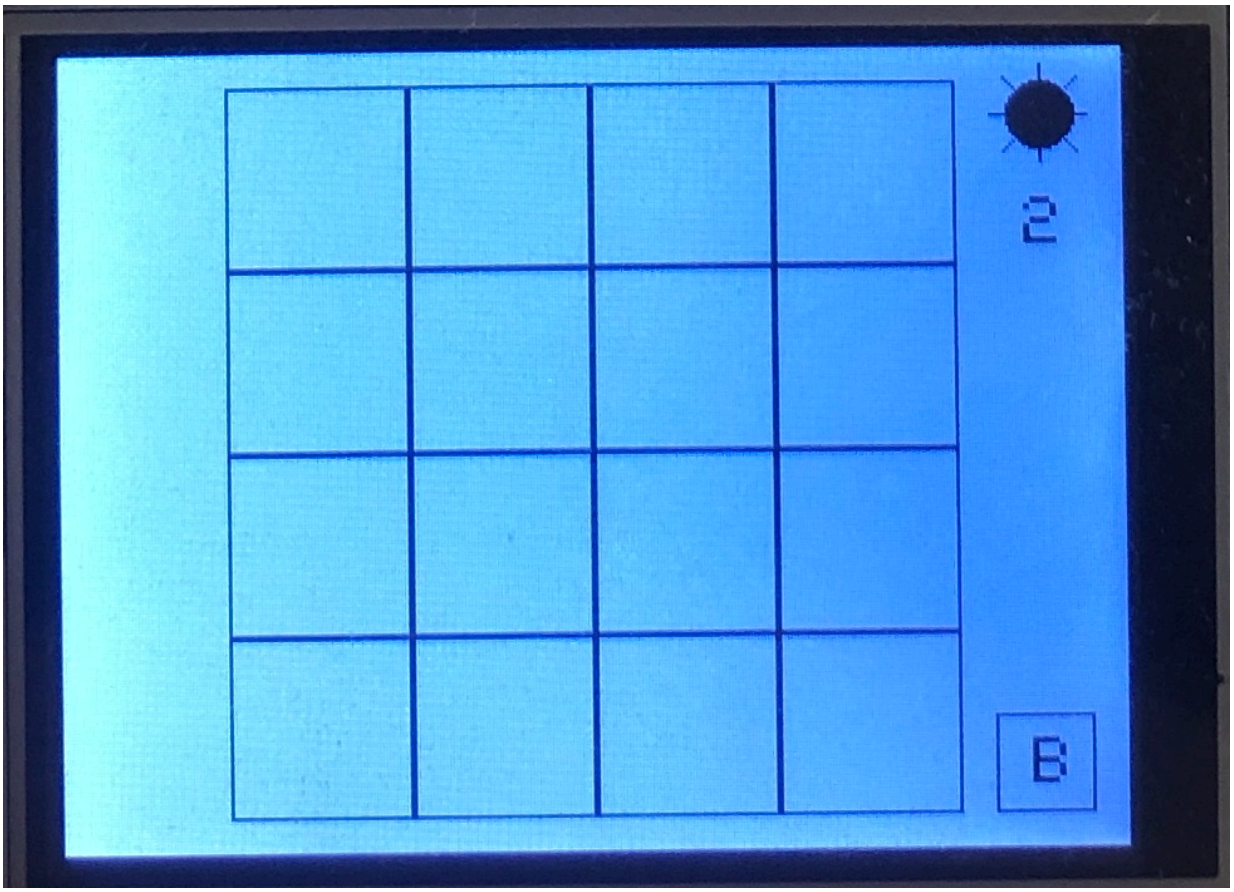


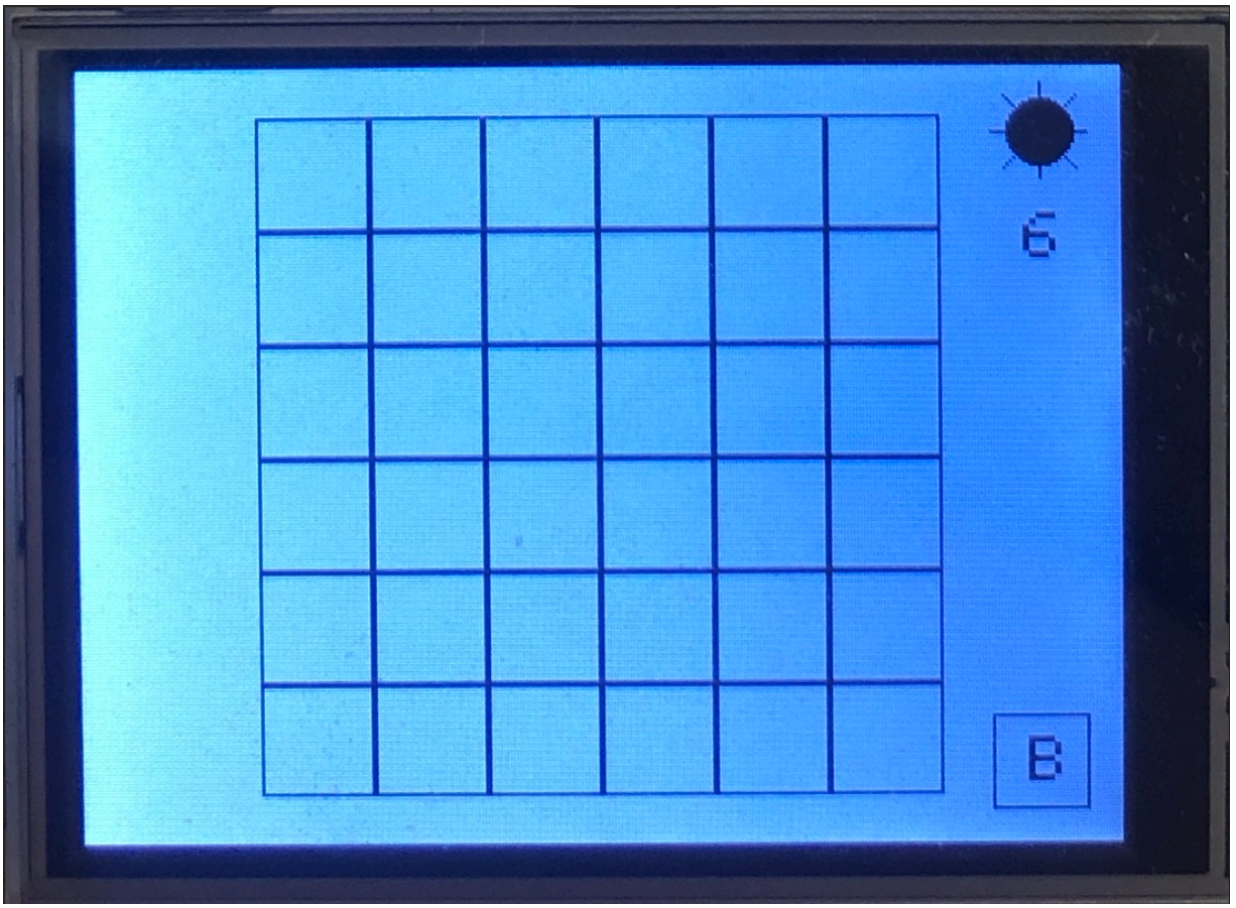
En este punto, el jugador habrá cambiado de pantalla y se le mostrará la pantalla de elección de dificultad, que consta de tres opciones: fácil, medio y difícil. Cada una de estas opciones tendrá configuraciones diferentes de tablero y minas. Por ejemplo, el modo fácil tiene un tablero de 4x4 con un máximo de 2 minas, el modo medio tiene un tablero de 6x6 con un máximo de 6 minas, y el modo difícil tiene un tablero de 8x8 con un total de 12 minas.

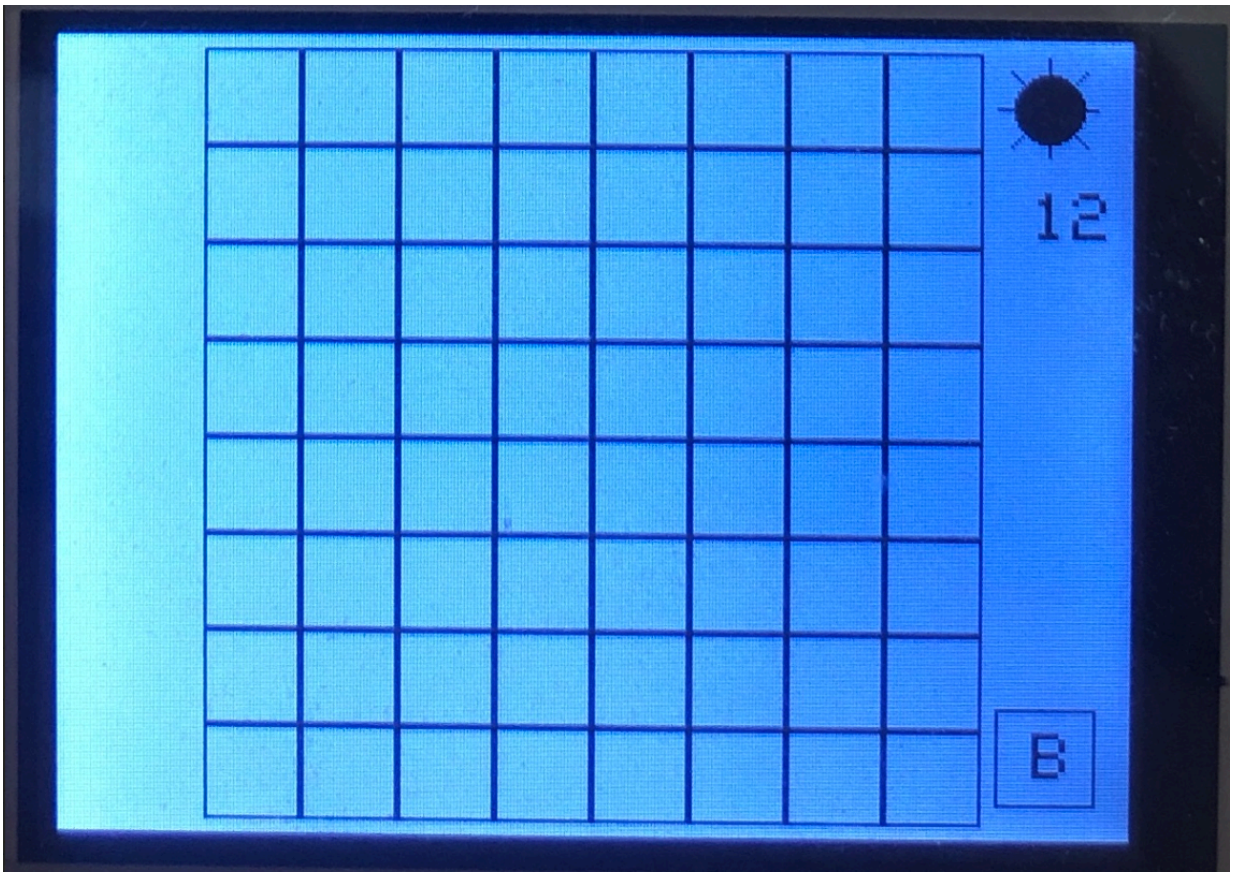
A continuación, procederemos a programar la funcionalidad táctil para cada botón de selección de dificultad. Una vez pulsado un botón, el programa cambiará al modo de juego correspondiente. Implementaremos esta funcionalidad de la siguiente manera:

```
448  } else if (currentState == SECOND_SCREEN) {
449
450    if (touch.z > 0) {
451      // Coordenadas de los rectángulos
452      int rectX = 90;
453      int rectYFacil = 25;
454      int rectYMedio = 95;
455      int rectYDificil = 165;
456      int rectWidth = 140;
457      int rectHeight = 50;
458
459      // Verifica si se toca "Fácil"
460      if (adjustedX > rectX && adjustedX < rectX + rectWidth && adjustedY > rectYFacil && adjustedY < rectYFacil + rectHeight) {
461        // Realiza acciones para la opción "Fácil"
462        tam = 5;
463        tft.fillScreen(0X8430);  // Llena la pantalla de gris
464        initializeGame(4, 4, 2);
465        currentState = GAME_PLAY;
466        drawBoard(tam);
467        drawMine(295, 20);
468        tft.setCursor(290, 45);
469        tft.print("2");
470        delay(100);
471      }
```

```
474      if (adjustedX > rectX && adjustedX < rectX + rectWidth && adjustedY > rectYMedio && adjustedY < rectYMedio + rectHeight) {
475        // Realiza acciones para la opción "Medio"
476        tam = 5;
477        tft.fillScreen(0X8430);
478        initializeGame(6, 6, 6);
479        currentState = GAME_PLAY;
480        drawBoard(tam);
481        drawMine(295, 20);
482        tft.setCursor(290, 45);
483        tft.print("6");
484        delay(100);
485      }
486
487      // Verifica si se toca "Difícil"
488      if (adjustedX > rectX && adjustedX < rectX + rectWidth && adjustedY > rectYDificil && adjustedY < rectYDificil + rectHeight) {
489        // Realiza acciones para la opción "Difícil"
490        tam = 1;
491        tft.fillScreen(0X8430);
492        initializeGame(8, 8, 12);
493        currentState = GAME_PLAY;
494        drawBoard(tam);
495        drawMine(295, 20);
496        tft.setCursor(290, 45);
497        tft.print("12");
498        delay(100);
499      }
500    }
```


2
B


6
B



Primero, definiremos las coordenadas para cada botón. Esto nos permitirá, al programar la función táctil, diferenciar en qué posición de la pantalla se está tocando. Utilizaremos estas coordenadas para identificar correctamente la interacción del usuario con los botones.

Una vez elegida la dificultad, se entrará en el modo de juego correspondiente se llamarán a varias funciones, la primera `initializeGame()`. Esta función será la encargada de definir los parámetros del tablero y el número de minas, además de colocar las minas de manera aleatoria, evitando que dos minas se ubiquen en la misma celda. La implementación se realizará de la siguiente manera:

```
55  void initializeGame(int rows, int cols, int minesCount) {
56    // Inicializa el juego y coloca las minas
57    memset(mines, false, sizeof(mines));
58    memset(revealed, false, sizeof(revealed));
59    memset(flagged, false, sizeof(flagged));
60
61    totalMines = minesCount;
62    boardRows = rows;
63    boardCols = cols;
64
65    for (int i = 0; i < totalMines; ++i) {
66      int row = random(boardRows);
67      int col = random(boardCols);
68
69      while (mines[row][col]) {
70        // Evita colocar una mina en la misma posición
71        row = random(boardRows);
72        col = random(boardCols);
73      }
74
75      mines[row][col] = true;
76    }
77  }
78
```

Además de la función `initializeGame()`, se llamará a la función `drawBoard()`, que será la encargada de dibujar el tablero en la pantalla. A esta función se le pasará un parámetro `tam`, que, definido dentro de cada selección de dificultad, ajustará el tamaño del tablero impreso en pantalla.

```
79  // Función para dibujar el tablero
80  void drawBoard(int tam) {
81    cellSize = min(tft.width() / boardCols, tft.height() / boardRows - tam);
82    int paddingX = (tft.width() - boardCols * cellSize) / 2;
83    int paddingY = (tft.height() - boardRows * cellSize) / 2;
84
85    for (int i = 0; i < boardRows; i++) {
86      for (int j = 0; j < boardCols; j++) {
87        int cellX = paddingX + j * cellSize;
88        int cellY = paddingY + i * cellSize;
89
90        tft.drawRect(cellX, cellY, cellSize, cellSize, 0x0000);
91      }
92    }
93  }
94
```

La función `drawMine()` se utilizará para dibujar una representación visual de una mina en la pantalla de juego. Esta función ya se utilizó para dibujar la mina del menú principal, por lo que haremos uso de nuevo de ella.

En el estado `GAME_PLAY` de nuestro juego, se invocan dos funciones fundamentales: `drawButton()` y `handleTouch()`. Estas funciones son las que determinan el comportamiento de nuestro juego y se programaron de la siguiente manera:

```
95  void handleTouch() {
96    TSPoint touch = ts.getPoint();
97    pinMode(XM, OUTPUT);
98    pinMode(YP, OUTPUT);
99
100   // Convierte las coordenadas táctiles a las coordenadas del tablero
101   int screenX = map(touch.x, TS_MINX, TS_MAXX, 0, tft.width());
102   int screenY = map(touch.y, TS_MINY, TS_MAXY, 0, tft.height());
103
104   if (screenX > BUTTON_X && screenX < BUTTON_X + BUTTON_WIDTH && screenY > BUTTON_Y && screenY < BUTTON_Y + BUTTON_HEIGHT && touch.z > 0) {
105     flagMode = !flagMode;
106     drawButton();
107     delay(100);
108     return;
109   }
110
111   // Verifica si el toque está dentro de los límites del tablero
112   if (screenX >= 0 && screenX < tft.width() && screenY >= 0 && screenY < tft.height()) {
113     // Calcula la celda en la que se tocó
114     int paddingX = (tft.width() - boardCols * cellSize) / 2;
115     int paddingY = (tft.height() - boardRows * cellSize) / 2;
116
117     // Ajusta las coordenadas para resolver el problema
118     int adjustedX = screenX - paddingX;
119     int adjustedY = screenY - paddingY;
120
```

En primer lugar, declaramos las configuraciones necesarias para habilitar la funcionalidad táctil en nuestro juego. Una vez establecidas estas configuraciones, procederemos a diseñar y programar un botón específico para la colocación de banderas en el tablero. Al pulsar este botón, se activará una condición que invocará nuevamente nuestra función `drawButton()`. A continuación, se presenta la implementación detallada de dicha función:

```
358 void drawButton() {
359   tft.drawRect(BUTTON_X, BUTTON_Y, BUTTON_WIDTH, BUTTON_HEIGHT, 0x0000);  // Dibuja el rectángulo del botón
360   tft.setTextSize(2);
361   tft.setTextColor(flagMode ? 0xF800 : 0x0000);  // Rojo si el modo de bandera está activado, negro si no
362   tft.setCursor(BUTTON_X + 11, BUTTON_Y + 7.5);  // Centrar el texto
363   tft.print("B");
364 }
```

La función `drawButton()` es responsable de dibujar el botón de bandera en la interfaz del juego. Al pulsar este botón, el estado del botón cambiará, indicando que está listo para colocar una bandera en la ubicación seleccionada en el tablero. Este cambio de estado se representará visualmente cambiando el color del botón de negro a rojo.

La función `handleTouch()` es esencial para gestionar las interacciones del usuario mediante toques en la pantalla. Una de sus responsabilidades clave es verificar que el toque del usuario se encuentra dentro de los límites del tablero de juego. Esta verificación se

realiza mediante cálculos precisos que aseguran que solo se tomen acciones válidas cuando el usuario interactúa con el tablero.

```
111   // Verifica si el toque está dentro de los límites del tablero
112   if (screenX >= 0 && screenX < tft.width() && screenY >= 0 && screenY < tft.height()) {
113     // Calcula la celda en la que se tocó
114     int paddingX = (tft.width() - boardCols * cellSize) / 2;
115     int paddingY = (tft.height() - boardRows * cellSize) / 2;
116
117     // Ajusta las coordenadas para resolver el problema
118     int adjustedX = screenX - paddingX;
119     int adjustedY = screenY - paddingY;
120
121     // Calcula las coordenadas de la celda correspondiente
122     int boardX = adjustedX / cellSize;  // Invierte el cálculo
123     int boardY = adjustedY / cellSize;
124
125     // Verifica si las coordenadas están dentro del tablero y si el juego está en curso
126     if (currentState == GAME_PLAY && boardX >= 0 && boardX < boardRows && boardY >= 0 && boardY < boardCols) {
127
```

Una vez que se ha verificado que el toque está dentro de los límites del tablero, el siguiente paso es programar la funcionalidad que permite al usuario colocar y eliminar marcas de bandera en las celdas del tablero. Esta funcionalidad debe manejar adecuadamente dos situaciones: la colocación de una bandera en una celda no revelada y la eliminación de una bandera en celdas tanto no reveladas como ya reveladas.

Colocación de la Bandera

Cuando el botón de bandera está activado y el usuario toca una celda, se colocará un círculo rojo en dicha celda para indicar la presencia de una bandera.

Eliminación de la Bandera

La eliminación de una marca de bandera requiere analizar dos posibles situaciones:

1. **Celda No Revelada:** Si la marca de bandera se encuentra en una celda que no ha sido revelada previamente, al eliminar la marca, la celda volverá a estar disponible para ser revelada.
2. **Celda Revelada:** Si, por error, se ha colocado una bandera en una celda que ya ha sido revelada y esta contenía el número de minas alrededor, al eliminar la marca, la celda debe volver a su estado revelado, mostrando el mismo número.

```
125     // Verifica si las coordenadas están dentro del tablero y si el juego está en curso
126     if (currentState == GAME_PLAY && boardX >= 0 && boardX < boardRows && boardY >= 0 && boardY < boardCols) {
127
128       if(flagMode){
129         if (flagged[boardX][boardY]) {
130           // Si la casilla ya está marcada, eliminar la marca
131           flagged[boardX][boardY] = false;
132           removeFlag(boardX, boardY);
133           // Redibuja el número si la casilla estaba revelada y no es una mina
134           if (revealed[boardX][boardY] && !mines[boardX][boardY]) {
135             drawNumber(boardX, boardY, countAdjacentMines(boardX, boardY));
136           } else {
137             // La casilla no estaba revelada, la vuelve a estar disponible para ser revelada
138             revealed[boardX][boardY] = false;
139           }
140         } else {
141           drawFlag(boardX, boardY);
142           flagged[boardX][boardY] = true;
143         }
```

En caso de que el usuario no haya activado el modo de colocación de banderas y toque una celda aleatoria, el juego debe garantizar que la primera celda tocada no contenga una mina. Si la primera celda tocada contiene una mina, el juego redistribuye las minas para asegurar que el jugador no pierda en el primer movimiento.

```
144       } else {
145         // Verifica si es la primera celda tocada
146         if (!flagged[boardX][boardY]) {
147           static int revealedCount = 0;
148
149           if (!firstCellTouched && mines[boardX][boardY]) {
150           // Vuelve a inicializar el juego para colocar las minas aleatoriamente
151             initializeGame(boardRows, boardCols, totalMines);
152             return;  // Sale de la función para evitar revelar la celda actual
153           }
154
155           if (!firstCellTouched) {
156           // Si es la primera celda, revela la celda seleccionada y las celdas adyacentes sin minas
157             revealCell(boardX, boardY);
158             revealSafeCells(boardX, boardY, revealedCount);
159             firstCellTouched = true;
160           } else {
161           // Si no es la primera celda, revela la celda seleccionada
162             revealCell(boardX, boardY);
163           }
```

En esta sección, se describen las dos funciones clave que se invocarán para gestionar la interacción del usuario con el tablero de juego. La primera de estas funciones es `revealCell()`, cuya responsabilidad principal es revelar las celdas seleccionadas por el usuario.

En esta sección, se describen las dos funciones clave que se invocarán para gestionar la interacción del usuario con el tablero de juego: `revealCells` y `revealSafeCells`. Estas funciones son complementarias y juegan un papel fundamental en la dinámica del juego.

Función `revealCells`

La función `revealCells` es responsable de revelar de forma recursiva las celdas seleccionadas por el usuario. Si la celda seleccionada no tiene minas adyacentes, la función también revelará automáticamente las celdas adyacentes. Este comportamiento simula la lógica de expansión que es característica del juego clásico de Buscaminas. Además, esta función dibuja un 0 en las celdas que no tienen minas cercanas.

Función `revealSafeCells`

La función `revealSafeCells` se encarga de revelar todas las celdas seguras. Una celda segura es aquella que no contiene una mina y, generalmente, se define como una celda que ha sido determinada como segura a través de la lógica del juego, como por ejemplo, celdas que no tienen minas adyacentes.

```
232  void revealSafeCells(int row, int col, int &revealedCount) {
233    if (revealedCount >= 2 || revealed[row][col] || mines[row][col]) {
234      return;  // No reveles más celdas, celdas ya reveladas o celdas con minas
235    }
236
237    revealed[row][col] = true;
238    revealedCount++;
239
240    // Recorre las celdas adyacentes
241    for (int i = max(0, row - 1); i <= min(boardRows - 1, row + 1); ++i) {
242      for (int j = max(0, col - 1); j <= min(boardCols - 1, col + 1); ++j) {
243        if (!mines[i][j]) {
244          revealSafeCells(i, j, revealedCount);
245        }
246      }
247    }
248  }
249
```

```
250  void revealCell(int row, int col) {
251    if (revealed[row][col] || mines[row][col]) {
252      return;  // No hagas nada si la celda ya ha sido revelada o contiene una mina
253    }
254
255    revealed[row][col] = true;
256    int numMines = countAdjacentMines(row, col);
257    int paddingX = (tft.width() - boardCols * cellSize) / 2;
258    int paddingY = (tft.height() - boardRows * cellSize) / 2;
259    int cellX = paddingX + col * cellSize;
260    int cellY = paddingY + row * cellSize;
261
262    if (numMines == 0) {
263      // Si no hay minas cercanas, revela las celdas adyacentes recursivamente
264      for (int i = max(0, row - 1); i <= min(boardRows - 1, row + 1); ++i) {
265        for (int j = max(0, col - 1); j <= min(boardCols - 1, col + 1); ++j) {
266          revealCell(i, j);
267        }
268      }
```

```
268      }
269
270      tft.setTextSize(2);
271      tft.setTextColor(0x0000);
272      tft.setCursor(cellX + cellSize / 2 - 5, cellY + cellSize / 2 - 10);
273      tft.print("0");
274
275    } else {
276      // Si hay minas cercanas, muestra el número en la celda
277      drawNumber(row, col, numMines);
278    }
279  }
280
```

En esta sección, se describen las funciones adicionales que se invocan dentro de `revealCell()` para completar su lógica. Para lograr su funcionalidad completa, esta función invoca a otras dos funciones: `countMinesAround()` y `drawNumber()`.

Función countAdjacentMines()

La función `countAdjacentMines()` es crucial para el correcto funcionamiento de la función `revealCell()`, ya que proporciona la información necesaria sobre el entorno

inmediato de una celda seleccionada. A continuación, se describe en detalle su propósito, implementación y cómo se integra con `revealCell()`.

La función `countAdjacentMines()` tiene como objetivo contar el número de minas que se encuentran en las celdas adyacentes a una celda dada en el tablero de juego. Esta información es esencial para determinar qué número se debe mostrar en la celda revelada y para decidir si se deben revelar automáticamente las celdas adyacentes sin minas.

Función drawNumber ()

La función `drawNumber()` se encarga de imprimir el número de minas adyacentes en la celda revelada. Este número indica al jugador cuántas minas están cerca de la celda seleccionada, proporcionando información crucial para el progreso del juego. Una vez que `countAdjacentMines()` ha devuelto un valor, este se pasa a la función `drawNumber()`, que es responsable de dibujar el número correcto dentro de la celda específica.

```
300  void drawNumber(int row, int col, int number) {
301    int paddingX = (tft.width() - boardCols * cellSize) / 2;
302    int paddingY = (tft.height() - boardRows * cellSize) / 2;
303    int cellX = paddingX + col * cellSize;
304    int cellY = paddingY + row * cellSize;
305
306    tft.setTextSize(2);
307    tft.setTextColor(0x0000);
308    tft.setCursor(cellX + cellSize / 2 - 5, cellY + cellSize / 2 - 10);
309    tft.print(number);
310  }
311
```

En esta sección se describe cómo programar y diseñar la lógica que determina las condiciones de victoria y derrota en el juego. La condición de derrota se activa cuando el jugador pulsa una celda que contiene una mina. En ese momento, el juego debe finalizar y todas las minas del tablero deben revelarse. A continuación se describe la implementación de esta lógica. La condición de victoria se activa, cuando hemos revelado todas las celdas seguras y solo quedan las celdas con minas disponibles para revelar. Se implementara de la siguiente manera:

Funcion drawGameOver()

```
165                 // Verifica si se ha tocado una mina
166                 if (mines[boardX][boardY]) {
167                   // Se ha tocado una mina, muestra la pantalla de Game Over
168                   currentState = GAME_OVER;
169                   revealAllMines();
170                   delay(3000);
171                   drawGameOver();
```

```
196   void drawGameOver() {
197     tft.fillScreen(0X8430);  // Llena la pantalla de blanco
198     tft.setTextSize(3);
199     tft.setTextColor(0xF800);  // Color rojo
200     tft.setCursor(80, 100);
201     tft.print("Game Over");
202     delay(5000);
203     tft.fillScreen(0X8430);
204     resetGame();
205   }
```


Game Over

Funcion drawVictory()

```
172             } else {
173               // Verifica si todas las celdas no minadas han sido reveladas, en cuyo caso el jugador ha ganado
174               bool allNonMineCellsRevealed = true;
175               for (int i = 0; i < boardRows; ++i) {
176                 for (int j = 0; j < boardCols; ++j) {
177                   if (!mines[i][j] && !revealed[i][j]) {
178                     allNonMineCellsRevealed = false;
179                     break;
180                   }
181                 }
182               }
183               if (allNonMineCellsRevealed) {
184                 // Todas las celdas no minadas han sido reveladas, el jugador ha ganado
185                 currentState = GAME_OVER;
186                 drawVictory();
187               }
188             }
189           }
190         }
191       }
192     }
193     delay(100);
194   }
```

```
207   void drawVictory() {
208     tft.fillScreen(0X8430);  // Llena la pantalla de blanco
209     tft.setTextSize(3);
210     tft.setTextColor(0x07E0);  // Color verde
211     tft.setCursor(95, 100);
212     tft.print("Ganaste");
213     delay(5000);
214     tft.fillScreen(0X8430);
215     resetGame();
216   }
217
```

Una vez que se cumple la condición de victoria o derrota, el estado del juego cambiará a `GAME_OVER` y se imprimirán los mensajes correspondientes. Además, se invocará una función resetGame() para reiniciar el juego.

La función `resetGame()` es fundamental para restablecer el juego a su estado inicial, permitiendo que los jugadores comiencen una nueva partida sin problemas. Esta función debe reiniciar todas las variables modificadas durante el juego y devolver al jugador a la pantalla principal.

```
289  void resetGame() {
290    currentState = MAIN_MENU;
291    touchHandled = false;
292    firstCellTouched = false;
293    flagMode = false;
294  }
```

- Presupuesto

Para elaborar el presupuesto de nuestro proyecto, identificamos los componentes y recursos necesarios, junto con sus costos estimados. En este caso, se mencionan dos componentes fundamentales: la placa de desarrollo Arduino Uno R3 y una pantalla táctil.

Componente	**Cantidad**	**Costo unitario**	**Costo total**
Arduino UNO R3	1	30 euros	30 euros
Pantalla tactil	1	20 euros	20 euros
Mano de obra	1	70 euros	70 euros
IGIC			8,4 euros
Total			128,4 euros

- Conclusiones

Desarrollo de Habilidades Técnicas

El proyecto permitió el desarrollo y fortalecimiento de habilidades en programación y diseño de sistemas embebidos. La implementación del Buscaminas en la plataforma Arduino requirió una comprensión profunda tanto de hardware como de software, demostrando la capacidad para integrar y optimizar ambos aspectos.

Integración de Componentes

La integración de la pantalla táctil con la placa Arduino UNO R3 presentó desafíos que fueron abordados con éxito. Esto incluyó el manejo de bibliotecas de control táctil, la gestión de la interfaz gráfica y la implementación de algoritmos de juego eficientes, demostrando la capacidad para manejar proyectos complejos y multifacéticos.

Interacción Usuario-Máquina

El proyecto destacó la importancia de una interfaz de usuario intuitiva y responsiva. La implementación de una pantalla táctil permitió una interacción directa y natural con el juego, mejorando la experiencia del usuario. Este aspecto es importante en el desarrollo de aplicaciones y juegos embebidos, donde la usabilidad es fundamental.

- Conclusions

Development of Technical Skills

The project allowed for the development and strengthening of skills in programming and embedded systems design. The implementation of Minesweeper on the Arduino platform required a deep understanding of both hardware and software, demonstrating the ability to integrate and optimize both aspects.

Component Integration

The integration of the touch screen with the Arduino UNO R3 board presented challenges that were successfully addressed. This included managing touch control libraries, handling the graphical interface, and implementing efficient game algorithms, demonstrating the ability to handle complex and multifaceted projects.

Human-Machine Interaction

The project highlighted the importance of an intuitive and responsive user interface. The implementation of a touch screen allowed for direct and natural interaction with the game, enhancing the user experience. This aspect is crucial in the development of embedded applications and games, where usability is fundamental.

- Bibliografía

[1] Ruben Velazco (2024).¿Vas a programar en Arduino? Todo lo que debes saber. https://www.softzone.es/programas/lenguajes/programar-arduino/

[2] WHADDA exciting electronics. https://whadda.com/product/2-8-inch-touch-screen-for-uno-mega-development-board-wpsh412/

[3] Arduino.cc forum. https://forum.arduino.cc/t/error-de-compilacion/315062

[4] Jose Domingo Muñoz (2018). Como programar buscaminas paso a paso. https://openwebinars.net/blog/como-programar-el-buscaminas-paso-paso/

[5] Programación en C Juego de buscaminas. https://elrincoinformatico.blogspot.com/2014/12/programacion-en-c-juego-de-buscaminas.html

[6] Juan David Meza Gonzalez (2021). Matrices en C++. Uso, declaración, y sintaxis de las matrices en C++. https://www.programarya.com/Cursos/C++/Estructuras-de-Datos/Matrices#google_vignette

[7] Entorno de Programación de Arduino (IDE). https://aprendiendoarduino.wordpress.com/2016/03/29/entorno-de-programacion-de-arduino-ide/

[8] Rotating the Display. https://learn.adafruit.com/adafruit-gfx-graphics-library/rotating-the-display

[9] Introducción al IDE de arduino. https://cpham-perso-univ--pau-fr.translate.goog/LORA/HUBIQUITOUS/solution-lab/arduino-lora-tutorial/introduction_arduino_ide/introduction_arduino_ide/?_x_tr_sl=en&_x_tr_tl=es&_x_tr_hl=es&_x_tr_pto=rq#:~:text=The%20Arduino%20IDE%20(Integrated%20Development,reason%20Arduino%20became%20so%20popular.