Juego del Gato implementado en la tarjeta Stellaris LaunchPad con interfaz LCD-Touch

Felipe Santiago Espinosa

Universidad Tecnológica de la Mixteca, Instituto de Electrónica y Mecatrónica.

Carretera a Acatlima km 2.5, Huajuapan de León Oaxaca.

Tel. (01-953) 53 203 99 Ext. 555

fsantiag@mixteco.utm.mx

Felipe de Jesús Trujillo Romero

Universidad Tecnológica de la Mixteca, Unidad de Posgrado.

Carretera a Acatlima km 2.5, Huajuapan de León Oaxaca.

Tel. (01-953) 53 203 99 Ext. 300

ftrujillo@mixteco.utm.mx

Fermín Hugo Ramírez Leyva

Universidad Tecnológica de la Mixteca, Instituto de Electrónica y Mecatrónica.

Carretera a Acatlima km 2.5, Huajuapan de León Oaxaca.

Tel. (01-953) 53 203 99 Ext. 555

hugo@mixteco.utm.mx

Resumen

Las pantallas LCD-Touch actualmente se encuentran en una diversidad de sistemas electrónicos, proporcionando una interfaz amigable y de fácil manejo. Su manipulación con un microcontrolador, aparentemente compleja, se simplifica al utilizar una biblioteca con los elementos necesarios para la generación rápida de salidas gráficas y el vínculo a funciones hechas por el programador para la atención de los eventos que ocurren en la zona táctil. Con la finalidad de evaluar el manejo de una pantalla LCD-Touch, en una aplicación demostrativa complementada con un procedimiento de inteligencia artificial, se implementó el clásico juego del gato. El sistema permite tres modos de juego: entre dos jugadores, jugador contra sistema y sistema contra jugador; para los dos últimos, el sistema responde con base en la búsqueda

minimax. La plataforma de desarrollo es la tarjeta Setellaris LaunchPad LM4F120 de Texas Instruments. La pantalla LCD-Touch utilizada es la LM4F120-L35 y es producida por la compañía KenTec Display.

Palabras Claves: juego del gato, LCD-Touch, minimax, Stellaris LaunchPad.

1. Introducción

Desde el año 2010, buscando alcanzar un lugar en el mercado de los microcontroladores (también referidos como MCU, *Micro Controller Unit*), la empresa Texas Instruments inició con la promoción de diversos kits de desarrollo, denominando a la serie como LaunchPad, siendo estos de muy bajo costo. Cada kit incluye una tarjeta de evaluación, la cual contiene un MCU, LEDs, botones y conectores para expansión. Las tarjetas se programan vía USB con el entorno Code Composer Studio, el cual está disponible de manera gratuita. Los kits LaunchPad también incluyen el cable USB para su programación y transferencia de datos, y tienen un costo entre $10 y $20 dólares, dependiendo del modelo del MCU [1].

En este trabajo se utilizó la tarjeta Stellaris LaunchPad, sus principales recursos son: Microcontrolador Stellaris LM4F120H5QR, dos conectores USB micro-B, LED RGB de usuario, dos botones de usuario e interfaz para depuración. El microcontrolador contiene un núcleo ARM Cortex-M4F de 32 bits con 256 KB de memoria Flash, 32 KB de SRAM, operación a 80 MHz, conexión USB, módulo de hibernación y otros periféricos [2]. La Fig. 1(a) muestra a la tarjeta Stellaris, un puerto USB es para la programación y depuración, y el otro es para que el MCU se comporte como un dispositivo en una red USB 2.0, permitiendo transferir datos a una velocidad de hasta 12 MHz.

Para ampliar la capacidad de las tarjetas LaunchPad, Texas Instruments ha desarrollado diversos módulos de expansión compatibles con sus diferentes modelos, a estos se les denomina BoosterPacks y con ellos se puede manejar una diversidad de periféricos, desde entradas y salidas básicas con LEDs e interruptores, manejo de sensores de diferentes tipos y algunos protocolos de comunicación, entre otros.

Texas Instruments mantiene una invitación abierta para que empresas o desarrolladores independientes diseñen nuevos BoosterPacks, siempre que se mantenga la compatibilidad con las LaunchPad. De esta manera, la compañía KenTec Display promueve al BoosterPack EB-LM4F120-L35, enfocado a las tarjetas Stellaris, el cual contiene una pantalla LCD gráfica de 3.5” con una resolución de 320 x 240 pixeles y una profundidad de 16 colores, además de una capa táctil con tecnología resistiva, la pantalla se muestra en la Fig. 1(b).

Fig. 1. Elementos empleados: (a) Tarjeta Stellaris LaunchPad y (b) Pantalla LCD-Touch.

Con la tarjeta Stellaris y la pantalla LCD-Touch se implementó el juego del gato, la programación se hizo en lenguaje C, empleando al entorno de desarrollo Code Composer y dos bibliotecas de funciones: La Biblioteca de Controladores Periféricos (*Stellaris Peripheral Driver Library*) y la Biblioteca de Gráficos (*Stellaris Graphics Library*). La primera contiene funciones dedicadas al uso de los recursos internos del microcontrolador [3], y la segunda incluye las funciones necesarias para la inicialización y manejo de la pantalla, así como para la gestión de los eventos producidos en la zona táctil [4].

Una vez que el juego del gato es puesto en operación, las respuestas del sistema están basadas en el método de búsqueda conocido como minimax, el cual pertenece a un conjunto de procedimientos enfocados a juegos entre dos adversarios empleando un tablero, como

ajedrez o damas chinas, en donde una opción conduce a otras, por lo que se genera un árbol del juego. Minimax es un procedimiento de búsqueda hacia adelante, mediante el cual se analiza la situación actual para valorar las posibles alternativas y elegir la que proporcione una mayor ventaja [5].

2. Desarrollo

Para la implementación del sistema se utilizó la Metodología Simplificada para Desarrollar Sistemas basados en Microcontroladores, la cual es descrita en [6] y comprende los pasos que se irán cubriendo en los apartados de esta sección.

2.1. Planteamiento del Problema

El clásico juego del gato, también conocido como tic tac toe o tres en raya, es un juego de destreza entre dos participantes. Requiere de un tablero con 9 casillas, formadas con dos líneas horizontales y dos verticales, cruzadas de tal manera que se forme el símbolo #. Tradicionalmente se usan la ‘X’ y la ‘O’ como las fichas de los jugadores, el objetivo para cada jugador consiste en la alineación horizontal, vertical o diagonal de tres de sus fichas, gana quien alcance el objetivo o el juego se empata si las casillas se agotan sin alinear tres fichas. Durante una tirada, el jugador decide si conseguirá alcanzar la alineación o si es mejor bloquear al contrincante para que no lo haga.

Puesto que se utilizará una pantalla LCD-Touch, este recurso será suficiente para funcionar como salida, para mostrar el tablero del juego; pero además, cada zona del tablero será una entrada, con un toque el jugador hará su tirada y el sistema colocará la ficha que le corresponda. El sistema debe validar que no se intente tirar en una posición ocupada.

Para seleccionar el modo de juego, el sistema debe contar con tres botones. Además, es conveniente la presencia de una barra de estado, en donde el sistema muestre mensajes que permitan conocer la situación actual del juego, indicando de quién es el turno, quién ganó o si

el juego se empató. Los botones para el modo y la barra de estado también ocuparán un espacio en la pantalla LCD-Touch. Organizando los elementos, el sistema final debe tener la apariencia de la Fig. 2.

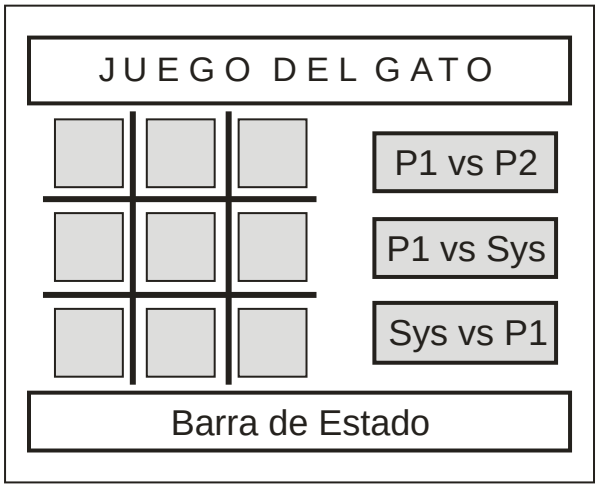


Fig. 2. Elementos esperados en el sistema (en gris se presentan las zonas táctiles).

El sistema iniciará con la matriz del juego vacía y en espera de que se seleccione el modo, sin atender los eventos que ocurran en el tablero.

2.2. Requerimientos de hardware y software

En cuanto al hardware, con la Stellaris LaunchPad y la pantalla LCD-Touch es suficiente. El microcontrolador de la tarjeta tiene un núcleo ARM de 32 bits, 256 KB de Flash y 32 KB de SRAM, por lo que sin problema puede alojar al árbol de juego generado con el procedimiento recursivo minimax. Por otra parte, todos los elementos requeridos para la interfaz con el usuario, y que se muestran en la Fig. 2, pueden acomodarse en la pantalla, proporcionando salidas visuales y entradas táctiles.

En lo que refiere al software, es necesario emplear las funciones de las bibliotecas proporcionadas por Texas Instruments, específicamente se requiere de la Biblioteca de Controladores Periféricos (*Stellaris Peripheral Driver Library*) y la Biblioteca de Gráficos (*Stellaris Graphics Library*). La primera incluye las funciones para inicializar y tener acceso a los recursos internos del MCU [3] y la segunda incluye las funciones necesarias para la inicialización y manejo de la pantalla, así como para la gestión de los eventos producidos en la zona táctil [4]. Ambas bibliotecas se agregan al Code Composer Studio en el momento en que se instala al programa StellarisWare, también de TI, que es una suite extensiva de software diseñada para simplificar y acelerar el desarrollo de aplicaciones basadas en un microcontrolador Stellaris [7].

Adicionalmente, para el proyecto se requiere de los controladores del LCD-Touch, éstos se encuentran por separado, la biblioteca **Kentec320x240x16_ssd2119_8bit** contiene las funciones para inicializar la pantalla LCD y la biblioteca **touch** las funciones para la inicialización de la zona táctil.

2.3. Diseño del hardware

El microcontrolador LM4F120 tiene 6 puertos, los puertos A, B, C y D son de 8 bits, el puerto E es de 6 bits y el puerto F es de 5. Sin considerar las terminales dedicadas a la programación y depuración del dispositivo, en la tarjeta LaunchPad quedan disponibles 35 terminales para entrada/salida de propósito general, éstas están distribuidas en cuatro conectores de 10 terminales, ubicándose dos conectores en cada extremo de la tarjeta, como puede verse en la figura 1(a).

Por otra parte, la pantalla LCD-Touch es un BoosterPack para la tarjeta Stellaris LaunchPad, de manera que se realiza una conexión directa entre estos elementos, sin la necesidad de un ajuste por parte del usuario. El LCD se maneja con una interfaz de 8 bits, por lo que además de los datos, se requieren pines para las señales de control: RD, WR, RS, CS y BL. Para la detección de eventos en la zona táctil se necesitan cuatro terminales, puesto que tiene tecnología resistiva, dos terminales son para el eje *x* y dos para el eje *y*; en una de ellas se

establece un voltaje de referencia y en la otra se mide un voltaje analógico, la magnitud es proporcional al punto de presión en la capa flexible de la pantalla [8]. En la tabla 1 se muestra la lista de las terminales de la tarjeta Setellaris LauchPad que son empleadas para el manejo de la pantalla LCD-Touch, indicando su función en la pantalla y la posición en uno de los cuatro conectores de la tarjeta.

Conector	PIN	Terminal MCU	Símbolo	Función
J1	3	PB0	LCD_D0	Bit 0 del bus de datos del LCD
J1	4	PB1	LCD_D1	Bit 1 del bus de datos del LCD
J2	2	PB2	LCD_D2	Bit 2 del bus de datos del LCD
J4	3	PB3	LCD_D3	Bit 3 del bus de datos del LCD
J1	7	PB4	LCD_D4	Bit 4 del bus de datos del LCD
J1	2	PB5	LCD_D5	Bit 5 del bus de datos del LCD
J2	7	PB6	LCD_D6	Bit 6 del bus de datos del LCD
J2	6	PB7	LCD_D7	Bit 7 del bus de datos del LCD
J2	8	PA4	LCD_RD	Control de Lectura en el LCD
J1	8	PA5	LCD_WR	Control de Escritura en el LCD
J1	9	PA6	LCD_RS	Selector entre registro y dato, en el LCD
J1	10	PA7	LCD_CS	Chip Select del LCD
J4	1	PF2	LCD_BL	Control ON/OFF de la luz de fondo
J1	5	PE4	TOUCH_XP	Terminal izquierda del resistor Touch
J1	6	PE5	TOUCH_YP	Terminal superior del resistor Touch
J2	9	PA3	TOUCH_XN	Terminal derecha del resistor Touch
J2	10	PA2	TOUCH_YN	Terminal inferior del resistor Touch

Tabla 1. Conexión entre el MCU y la pantalla LCD-Touch.

2.4. Diseño del software

La biblioteca de gráficos, también referida como GRLIB, es fundamental para el desarrollo del programa; ésta incluye funciones para dibujar elementos estáticos o primitivos que aparecerán en la pantalla y permanecerán sin cambios, a menos que se vuelvan a repintar. Los elementos primitivos se colocan en un contexto gráfico (*context*) indicando sus características, como ubicación, longitud y color, posteriormente el contexto es enviado a la pantalla mediante la función *grFlush*. De acuerdo con la Fig. 2, las líneas del tablero se deben dibujar con este esquema.

Los lienzos (*canvas*) son objetos de la biblioteca GRLIB ampliamente utilizados, un lienzo puede ubicarse directamente en pantalla, sin la necesidad de un contexto, colocando mensajes estáticos con características propias de color de fondo, margen, tipo, color y tamaño de letra. Para el juego bajo desarrollo, el título de la pantalla y la barra de estado se colocarán sobre lienzos.

Un lienzo también puede funcionar como un contenedor de *widgets*, los cuales son objetos disponibles para la interacción con el usuario. En un *widget*, además de sus características gráficas, se debe incluir el nombre de una función que será ejecutada inmediatamente después de que se detecte un evento sobre el objeto. Los tres botones para la selección del modo de juego y las nueve zonas del tablero deben ser *widgets*, sin embargo, manejarlos en forma independiente implica la necesidad de 12 funciones diferentes para la atención de sus eventos; además, conlleva al uso de variables globales que posteriormente deben organizarse en forma matricial, para la evaluación del estado del juego.

Por lo tanto, conviene más el manejo de dos arreglos, uno con nueve botones rectangulares para las zonas del juego y el otro con tres botones rectangulares para la selección de modo. De esta forma, sólo se emplearán dos funciones, una por arreglo, cada función debe iniciar con una evaluación que determine el elemento del arreglo que generó el evento. Cada arreglo será colocado en un lienzo; así, se tendrá el lienzo del modo y el lienzo del tablero.

Los lienzos y *widgets* se vinculan mediante una estructura jerárquica tipo árbol, generalmente un lienzo es el padre y puede tener a otros lienzos o a *widgets* como hijos; en un arreglo, el primer elemento es tratado como hijo de un lienzo y el elemento siguiente del arreglo es su hermano, esta relación se mantiene hasta alcanzar al último elemento, quien ya no tiene

hermanos. La relación jerárquica se establece en el momento en que se declaran los lienzos o *widgets*, además de indicar el nombre de la función de atención a eventos, en el caso de los *widgets*.

La raíz en esta jerarquía es una estructura definida como WIDGET_ROOT. Los lienzos que van a aparecer en la pantalla deben ser vinculados a la raíz con la función *WidgetAdd*, para que posteriormente, todos sean pintados con la llamada a la función *WidgetPaint(WIDGET_ROOT)*.

En la Fig. 3 se muestra el comportamiento del programa principal, el sistema tiene tres modos de juego, la variable que determina el modo y la matriz del tablero son variables globales, esto debido a que se modifican en las funciones que atienden a los eventos que ocurren en la zona táctil.

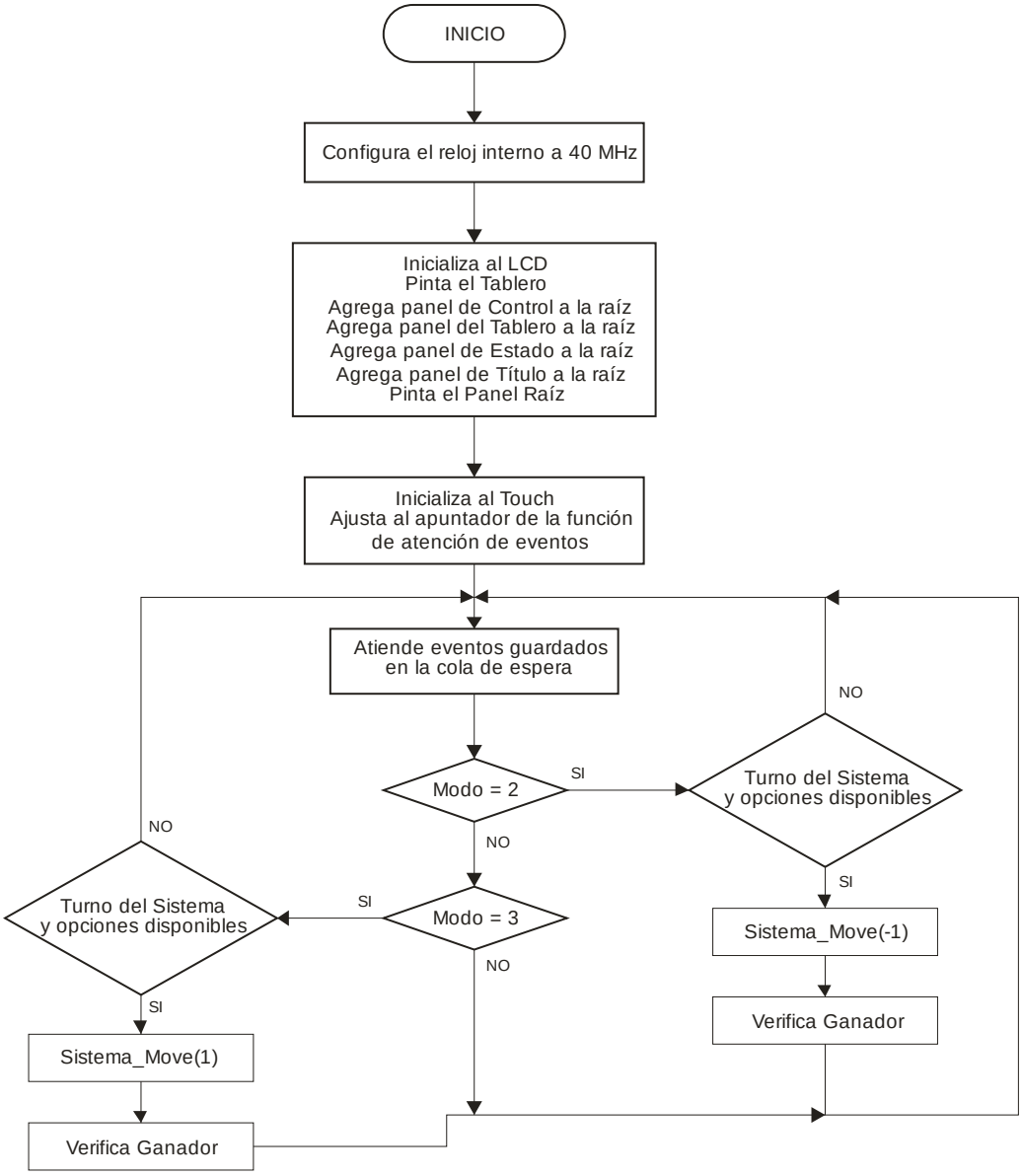


Fig. 3. Comportamiento del programa principal del sistema.

El modo 1 no se observa en el diagrama de flujo porque es un modo entre dos jugadores humanos, la validación del tiro en el tablero y la evaluación de la existencia de un ganador se hacen en una de las funciones que atiende los eventos que se generan en la zona táctil. En los modos 2 y 3, con la misma función se hacen las validaciones para el turno del jugador humano; por ello, en el programa principal únicamente se revisa si es el turno del sistema, se

realiza la tirada a partir de la función *Sistema_Move* y después se evalúa la existencia de un ganador.

Puede notarse que la función *Sistema_Move* en el modo 2 recibe -1 y en el modo 3 recibe 1, esto se debe a que en el modo 2 el sistema es el segundo en tirar y en el modo 3 tira primero. La selección de la casilla del tiro se hace a partir de la búsqueda minimax, el cual es un método que construye el árbol con todas las posibilidades y calcula, mediante una búsqueda en profundidad, la ventaja que le proporciona a cada jugador tirar en una determinada casilla, la ventaja del primer jugador se lleva con un número positivo y la del segundo con número negativos. De acuerdo con [5], al contendiente que lleva los números positivos se le denomina jugador maximizador y a quien lleva los números negativos será el minimizador, esto significa que el sistema buscará ganar cuando tira primero (maximizar su ventaja) y no perder si tira en segundo término (minimizar la ventaja del contrario).

Los eventos que ocurren en la zona táctil son detectados con uno de los ADC's del microcontrolador, empleando dos canales para sondear ambos ejes. La biblioteca incluye una función que da servicio a la interrupción que indica el fin de una conversión (ISR, *interrupt service routine*), el ACD opera en un modo de carrera libre, de manera que, en cada ejecución de la ISR se revisa la ocurrencia de eventos, en caso de existir, se van encolando para su posterior atención en el programa principal. Son dos las funciones que se pueden invocar cuando ocurre un evento en la zona táctil *OnPressControl* y *OnPressJuego*.

La función *OnPressControl* se invoca cuando uno de los botones de modo es presionado, con ello, primeramente se identifica al elemento del arreglo que generó el evento para registrar el modo en una variable global; posteriormente se inicializa al tablero y a la matriz del juego, quedando listos para iniciar una partida. Si se trata del modo 3, en el que el sistema debe tirar primero, se genera un número aleatorio entre 0 y 8 para hacer la primera tirada. En la barra de estado se indica el modo del juego (para los modos 1 y 2) o que es el turno del jugador humano (para el modo 3).

La función *OnPressJuego* se invoca cuando una zona del tablero de juego es presionada, también inicia con la identificación del elemento del arreglo que generó el evento, observa si en esa posición de la matriz hay un cero, indicando una casilla disponible, para después

colocar un 1 o -1 dependiendo del jugador en turno, en el texto del *widget* se colocará 'X' o 'O' y se repintará al lienzo del tablero, para que el usuario vea reflejada su tirada. El contador de tiros se incrementa y en caso de ser mayor de 4, se evalúa si hay un posible ganador. Cuando el contador llega a 9 y no hay ganador, se declara el juego empatado. En la barra de estado se colocan los mensajes que correspondan, indicando quién tiene el turno, quién ganó o si el juego se empató.

2.5. Implementación del hardware

Dado que se está empleando una tarjeta de evaluación en conjunción con una pantalla LCD-Touch totalmente compatible, no se requieren más implementaciones de hardware, por lo que en este paso de la metodología ya no hay actividades por hacer.

2.6. Implementación del software

Por el tamaño del programa, sólo se presentan algunos segmentos relevantes del mismo, explicando su contenido.

En la Fig. 4 se muestra cómo se declara al lienzo *control* con el arreglo que contiene los botones para definir el modo del juego. Se presenta un conflicto en la declaración porque cuando el lienzo se instancia ya debe estar definido el arreglo con los botones, dado que el lienzo los tomará como hijos; sin embargo, al declarar cada botón del arreglo, ya debería estar instanciado el lienzo para que los elementos lo refieran como padre. Debido a esto, primero se declara al lienzo como externo, sin los detalles de su instanciación; posteriormente se declara el arreglo con los botones, refiriendo al lienzo externo, para que, finalmente, se declare al lienzo de manera local, con todos sus atributos. En la declaración de cada botón se observa la jerarquía, el lienzo es el padre y los botones son hermanos. Entre sus atributos se manejan 4 colores, el color del texto y el color de fondo, para el botón sin presionar y presionado.

```
extern tCanvasWidget Control;          // Canvas para los 3 botones (externo)

// ARREGLO CON LOS BOTONES DE SELECCIÓN DEL MODO DE JUEGO

tPushButtonWidget Bot_Ctrl[] = {

RectangularButtonStruct(&Control, Bot_Ctrl + 1, 0,&g_sKentec320x240x16_SSD2119, 205,
      50, 102, 30, PB_STYLE_OUTLINE | PB_STYLE_TEXT_OPAQUE | PB_STYLE_TEXT |
      PB_STYLE_FILL, ClrWhite, ClrGray, ClrRed, ClrRed, &g_sFontCmss22b, "P1 vs P2", 0,
      0, 0, 0, OnPressControl),

RectangularButtonStruct(&Control, Bot_Ctrl + 2, 0,&g_sKentec320x240x16_SSD2119, 205,
      110, 102, 30, PB_STYLE_OUTLINE | PB_STYLE_TEXT_OPAQUE |PB_STYLE_TEXT |
      PB_STYLE_FILL, ClrWhite, ClrGray, ClrRed, ClrRed, &g_sFontCmss22b, "P1 vs Sys",
      0, 0, 0, 0, OnPressControl),
RectangularButtonStruct(&Control, 0, 0,&g_sKentec320x240x16_SSD2119,205, 170, 102, 30,
      PB_STYLE_OUTLINE | PB_STYLE_TEXT_OPAQUE | PB_STYLE_TEXT | PB_STYLE_FILL,
      ClrWhite, ClrGray, ClrRed, ClrRed, &g_sFontCmss22b, "Sys vs P1", 0, 0, 0, 0,
      OnPressControl)
};

Canvas(Control, 0, 0, Bot_Ctrl, &g_sKentec320x240x16_SSD2119, 170, 28, 150, 200,
               CANVAS_STYLE_FILL, ClrBlack,0,0,0,0,0,0);
```

Fig. 4. Declaración del lienzo con los botones para definir el modo del juego.

De forma similar se declara el lienzo con el tablero del juego. Estas declaraciones son globales, así como la declaración del lienzo para el título y para la barra de estado. Los lienzos también pueden referenciar a una función que invocarán cuando sean dibujados, al lienzo del tablero se le vincula con la función *paintTablero*, la cual pintará las líneas que lo conforman. Con esto, en la pantalla únicamente se colocarán los cuatro lienzos. En la Fig. 5 incluyendo el código necesario para la configuración del reloj interno, la inicialización del LCD y la ubicación de los lienzos en la pantalla.

```
   SysCtlClockSet(SYSCTL_SYSDIV_4|SYSCTL_USE_PLL|SYSCTL_OSC_MAIN|SYSCTL_XTAL_16MHZ);

   Kentec320x240x16_SSD2119Init();                        // Inicializa al LCD
   GrContextInit(&sContext, &g_sKentec320x240x16_SSD2119);
   ClrScreen();                                           // Limpia la pantalla

   WidgetAdd(WIDGET_ROOT, (tWidget *)&Control);           // Lienzo de control
   WidgetAdd(WIDGET_ROOT, (tWidget *)&Panel_Tablero);     // Lienzo del tablero
   WidgetAdd(WIDGET_ROOT, (tWidget *)&Titulo);            // Barra de título
```

```
   WidgetAdd(WIDGET_ROOT, (tWidget *)&Estado);             // Barra de estado

   WidgetPaint(WIDGET_ROOT);                               // Dibuja en la pantalla
```

Fig. 5. Configuración del reloj e inicialización de la pantalla LCD.

Con respeto a la zona táctil de la pantalla, en la Fig. 6 se muestran las llamadas a las funciones que lo inicializan.

```
   TouchScreenInit();

   TouchScreenCallbackSet(WidgetPointerMessage);
```

Fig. 6. Inicialización de la zona táctil.

El código que se ejecuta en el lazo infinito se presenta en la Fig. 7, éste corresponde al diagrama de flujo de la Fig. 3.

```
while(1)
{
   WidgetMessageQueueProcess();  // Atiende eventos pendientes de la zona  táctil

   if( modo == 2 && ganador == 0 && tiros < 9 && turno == -1) {   // Si al sistema le
                                                                  // toca tirar
       pos_sist = sistemaMove(-1);                  // Obtiene la posición del tiro
       PushButtonTextColorSet(Tablero + pos_sist, ClrYellow);     // Ubica tirada
       PushButtonTextSet(Tablero + pos_sist, "O");
```

```
        board[pos_sist] = -1;
        turno = 1;                                              // Cambia turno
        CanvasTextSet(&Estado, "Turno del Jugador Humano");
        tiros++;
        if(tiros > 4)
           ganador = hay_ganador();
        if( ganador == 0 && tiros == 9)
           CanvasTextSet(&Estado, "Juego Empatado");
        else if( ganador == -1)
           CanvasTextSet(&Estado, "Gana el Sistema");
        WidgetPaint((tWidget *)&Panel_Tablero);             // Actualiza del tablero
        WidgetPaint((tWidget *)&Estado);                    // Actualiza del mensaje
    }
    if( modo == 3 && ganador == 0 && tiros < 9 && turno == 1) {    // Si al sistema le
                                                                   // toca tirar
        pos_sist = sistemaMove(1);                    // Obtiene la posición del tiro
        PushButtonTextColorSet(Tablero + pos_sist, ClrYellow);     // Ubica tirada
        PushButtonTextSet(Tablero + pos_sist, "X");
        board[pos_sist] = 1;
        turno = -1;
        CanvasTextSet(&Estado, "Turno del Jugador Humano");
        tiros++;
        if(tiros > 4)
            ganador = hay_ganador();
        if( ganador == 0 && tiros == 9)
            CanvasTextSet(&Estado, "Juego Empatado");
        else if( ganador == 1)
            CanvasTextSet(&Estado, "Gana el Sistema");
        WidgetPaint((tWidget *)&Panel_Tablero);             // Actualiza del tablero
        WidgetPaint((tWidget *)&Estado);                    // Actualiza del mensaje
       }
}
```

Fig. 7. Codificación del lazo infinito del sistema.

Para elegir la casilla de tiro, la función *sistemaMove* evalúa todas las posibilidades, colocando su tiro y obteniendo el beneficio por medio de la función *minimax*, la función regresa la posición que le dio una mayor ventaja. En la Fig. 8 se muestra el código de la función, la bandera *procesando* se pone en alto para que cuando *minimax* llame a la función que verifica si hay ganador, no cambie el aspecto del tablero al encontrar combinaciones ganadoras.

```
// Regresa el tiro del sistema
int sistemaMove(int player) {

       int move;
       int score = -2;
       int k;
       int tempScore;

       procesando = 1;
       for(k = 0; k < 9; k++) {
             if(board[k] == 0) {                            // Si hay una casilla vacía
```

```
                    board[k] = player;                      // Tira
                    tempScore = -minimax(-1*player);        // Calcula beneficio
                    board[k] = 0;                           // Quita el tiro
                    if(tempScore > score) {
                          score = tempScore;                // Respalda la posición con
                          move = k;                         // un mayor beneficio
                    }
              }
        }
        procesando = 0;

        return  move;                                       // Regresa posición
}
```

Fig. 8. Función que obtiene la mejor posición para el tiro del sistema.

La función *minimax* es recursiva, la recursividad termina cuando se encuentra un ganador o cuando ya no hay opciones disponibles en el tablero de juego. En la Fig. 9 se muestra el código de la función *minimax*, la función *hay_ganador* regresa 0 si no encuentra combinaciones ganadores; o bien, -1 ó 1, dependiendo del jugador que consigue el triunfo. Es por ello que el producto *winner*player* dará un 1 si el jugador que hizo el tiro obtiene el triunfo y -1 si el triunfo es del contrario. La variable *score* inicia con -2, por lo que el triunfo de cualquier jugador dará un resultado mayor. Si ya no hay movimientos por realizar y en los realizados no ha habido ganador, la variable *move* mantendrá su valor inicial, que es -1, con ello la función recursiva regresará 0. Esta es una condición de empate, dado que 0 es mayor a -1, se prefiere el empate a que gane el contrario (tiene una mayor ventaja). Para la llamada recursiva de la función *minimax* se cambia el signo del jugador actual, por la alternancia entre jugadores; el resultado de esa llamada se multiplica por -1 porque la ventaja de un jugador se vuelve la desventaja del contrario, o viceversa.

```
//Implementación del algoritmo recursivo de Minimax

int minimax(int player) {

int move = -1;
int score = -2;
int winner = hay_ganador();
int k;
int thisScore;

      if(winner != 0) return winner*player;         // Termina si hay ganador
```

```
	for(k = 0; k < 9; k++) {                    // Evalúa las diferentes opciones
		if(board[k] == 0){
			board[k] = player;
			thisScore = -minimax(-1*player);       // Llamada recursiva
				if(thisScore > score) {
					score = thisScore;
					move = k;
				}
			board[k] = 0;
		}
	}

	if(move == -1)                  // Si no hay casilla, regresa 0
		return 0;

	return score;                   // Regresa el valor de la mejor opción
}
```

Fig. 9. Algoritmo recursivo de búsqueda Mini-Max.

2.7. Integración y Evaluación

El programa se editó y compiló con el entorno de desarrollo Code Compuser Studio versión 5.5. Desde ese entorno se realizó la programación del microcontrolador Stellaris. Para una adecuada compilación, el código fuente con las funciones para el manejo de la pantalla LCD-Touch debe ser agregado al proyecto. Además, en las propiedades del proyecto se debe especificar que el archivo grlib.lib debe ser considerado en la etapa de ligado del proyecto.

La integración hardware-software se hizo sin problema y el sistema funcionó de manera favorable.

2.8. Ajustes y Correcciones

Para la interfaz gráfica se hicieron una serie de ajustes, buscando una relación adecuada en el tamaño y color de los elementos de la interfaz. Afortunadamente la combinación del LaunchPad con el Code composer permite modificar e evaluar casi inmediatamente.

El sistema funcionó favorablemente, se le agregó la presentación de una imagen al inicio de la ejecución del programa con el propósito de usarlo en pláticas de difusión de la maestría en electrónica que se imparte en la UTM, esto debido a que las tarjetas se han empezado a usar en cursos de esta maestría, porque se tiene una plataforma para evaluar un núcleo ARM de 32 bits a un precio muy bajo.

La imagen primero debe procesarse para que tenga el tamaño de la pantalla y una resolución de 16 colores, los datos se colocan en un arreglo para su posterior ubicación en el contexto gráfico que después será mostrado. En la Fig. 10 se muestra cómo se proyecta una imagen en la pantalla.

```
GrImageDraw(&sContext, g_pucImage, 0, 0);    // Mueve el arreglo
GrFlush(&sContext);                          // Proyecta en pantalla
SysCtlDelay(SysCtlClockGet());               // Espera
ClrScreen();                                 // Limpia la pantalla
```

Fig. 10. Código para proyectar una imagen en la pantalla.

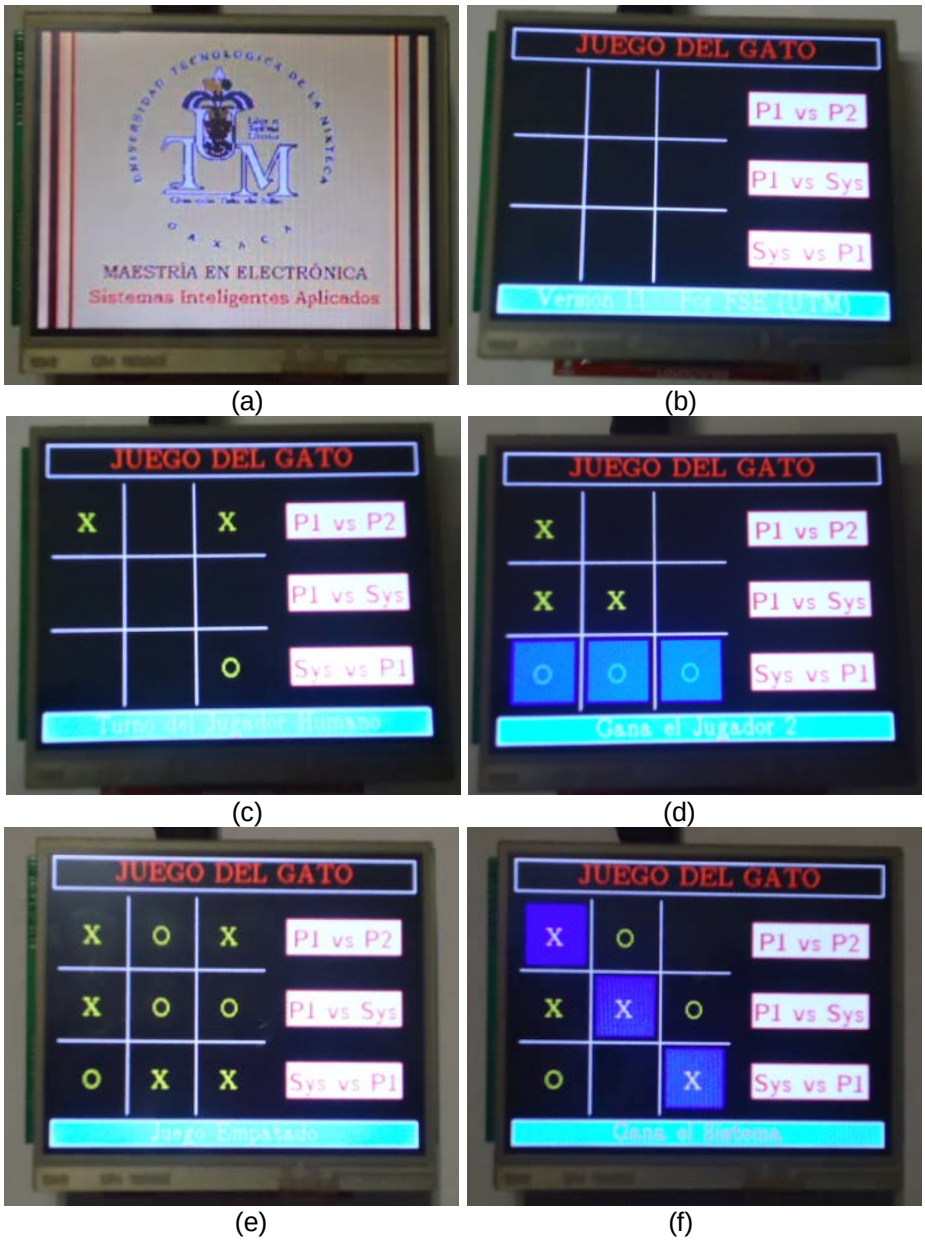
Además de los ajustes en la interfaz gráfica y el complemento con la imagen de presentación, no se requirió de otro ajuste o corrección.

3. Resultados

En la tabla 2 se muestra el porcentaje de uso de la memoria de código (FLASH) y de la memoria de datos (SRAM), se puede ver que hay espacio suficiente para aplicaciones con mayores requerimientos de procesamiento.

Memoria	Disponibilidad	Uso	Porcentaje
FLASH	256 Kbytes	49 418 Bytes	18.85 %
SRAM	32 Kbytes	4 136 Bytes	12.63 %

Tabla 2. Porcentaje en el uso de memoria.



(a) (b)

(c) (d)

(e) (f)

Fig. 11. Situaciones temporales en el juego del gato: (a) Pantalla de inicio, (b) Tablero sin iniciar una partida, (c) Turno del jugador 2, (d) Triunfo del jugador 2, (e) Juego empatado y (f) Gana el sistema.

En la Fig. 11 se muestran diferentes situaciones en el juego, la pantalla de inicio, el tablero vacío, algunas tiradas, un empate y el triunfo del sistema. Con excepción de la figura 9(a), en todos los casos el mensaje de la barra de estado sirve de auxiliar al usuario.

4. Discusión

El rendimiento del sistema es muy bueno, éste se determinó midiendo el tiempo de respuesta en el peor de los casos, es decir, cuando el sistema hace más procesamiento. Antes de determinar en dónde dará su respuesta, el sistema explora recursivamente todas las posibilidades, de manera que, la peor situación ocurre en el modo Jugador contra Sistema porque después del tiro del jugador quedan 8 posibilidades por explorar. En este caso, el tiempo promedio de respuesta fue de 1.8 segundos, el tiempo se midió enviando un carácter por el puerto serie de la tarjeta antes y después de la tirada, se utilizó un programa de monitoreo serial que muestra el tiempo en que ocurre cada evento, para después obtener la diferencia entre eventos sucesivos.

En el modo Sistema contra Jugador, la primera tirada es inmediata porque se define con la generación de un número aleatorio, esto proporciona versatilidad al inicio del juego. Después debe tirar el jugador y para la próxima tirada del sistema van a quedar 6 posibilidades por explorar, con esta cantidad de niveles el tiempo de respuesta es menor a un segundo, lo cual se concluye porque el monitor serial muestra el antes y después del tiro con el mismo valor de tiempo.

5. Conclusiones

El juego del gato se implementó favorablemente, los tres modos fueron evaluados por diversos usuarios con el funcionamiento esperado. Debido a que el método de búsqueda minimax explora todas las posibilidades, no se le puede ganar al sistema, aun cuando el jugador tire primero, en esos casos el mejor resultado para el humano será un empate.

El uso de la biblioteca de gráficos y las bibliotecas para el manejo de la pantalla LCD-Touch son fundamentales, simplifican el manejo de los recursos permitiendo que el programador se concentre únicamente en las acciones que se realizarán cuando ocurra un evento. La respuesta del sistema aparentemente es inmediata, dado que el ojo humano no detecta el intervalo de tiempo que transcurre desde que se genera un evento hasta que se realiza su atención.

La metodología empleada resultó muy conveniente, aunque en el paso correspondiente a la implementación del hardware no hubo nada que hacer. Pero aun utilizando tarjetas de evaluación y desarrollo como las LaunchPad de Texas Instruments, la metodología es funcional porque en muchas aplicaciones se va a contar con otros elementos de hardware que se podrán manipular a través de los conectores disponibles.

Con esta conjunción Stellaris LaunchPad y LCD-Touch se tiene una plataforma con una capacidad de procesamiento muy alta, dado que está soportada por un procesador de 32 bits puede operar hasta 200 MHz con la ayuda de la circuitería PLL interna; y además, se cuenta con una interfaz humana atractiva y de fácil manejo para el usuario. Por lo tanto, esta plataforma podrá adecuarse a aplicaciones con exigencias de procesamiento relativamente altas.

6. Referencias

[1] TI LaunchPad Evaluation Ecosystems. http://www.ti.com/launchpad. Consulta: Junio de 2014.

[2] Stellaris® LM4F120 LaunchPad Evaluation Board. User Manual. Texas Instruments Incorporated. August 2012–Revised July 2013.

[3] Stellaris® Peripheral Driver Library. User's Guide. Texas Instruments Incorporated. SW-DRL-UG-10636. Copyright © 2006-2013.

[4] Stellaris® Graphics Library. User's Guide. Texas Instruments Incorporated. SW-GRL-UG-10636. Copyright © 2008-2013.

[5] P. H. Winston, Inteligencia Artificial. 3a. Edición, Año 1992. Editorial Addison-Wesley Iberoamericana. ISBN 0-201-51876-7.

[6] F. Santiago Espinosa, Los Microcontroladores AVR de Atmel. 1a. Edición, Año 2012. Editado e Impreso por la Universidad Tecnológica de la Mixteca. ISBN: 978-607-95222-7-8.

[7] Stellaris Complete. http://www.ti.com/tool/sw-lm3s. Consulta: Junio de 2014.

[8] Introduction to Touch Solutions, White Paper, Revision 1.0 A, Densitron Corporation, August 21, 2007.

7. Autores

M. C. Felipe Santiago Espinosa es Maestro en Ciencias con especialidad en Electrónica por parte del INAOE, incorporado al IEM de la Universidad Tecnológica de la Mixteca, en donde es Profesor-Investigador desde 1998. Actualmente está cursando el Doctorado en Robótica en la misma institución. En el año de 2012 publicó su libro titulado "Los Microcontroladores AVR de ATMEL".

Dr. Felipe Trujillo Romero obtuvo el grado de Dr. en Sistemas Informáticos por el Instituto Nacional Politécnico de Toulouse, Francia. Actualmente es profesor-investigador de la División de Estudios de Posgrado de la Universidad Tecnológica de la Mixteca.

M. C. Fermín Hugo Ramírez Leyva obtuvo su título de Maestría en Ciencias con especialidad en electrónica por el Instituto Nacional de Astrofísica Óptica y Electrónica. Actualmente es profesor investigador, adscrito al Instituto de Electrónica y Mecatrónica de la UTM. Es

estudiante de doctorado en Ingeniería Mecatrónica en la Universidad Popular Autónoma del Estado de Puebla.