trabalho microcontroladores 3

Marco Antonio

1 Introdução

Esse relatorio visa explicar a dinamica e o funcionamento do jogo inspirado no jogo snake do nokia. O codigo é devido bot.h e bot.c que são os arquivos de configuração dos botões, os arquivos do nokia, e uma .h para os simbolos.

2 bot.c

Essa .h configura os botões que serão usados no jogo.GPIOPIN0 (pino 0) do PortF (GPIOPORTFBASE): Este pino é usado para ler a entrada do botão SW2. A função GPIOPinWrite(GPIOPORTFBASE, GPIOPIN4, 0xff) configura o pino 4 do PortF como saída alta (ligada) para acionar o botão SW2. Em seguida, a função GPIOPinRead(GPIOPORTFBASE, GPIOPIN0 GPIOPIN1 GPIOPIN2 GPIOPIN3) lê a entrada dos pinos 0, 1, 2 e 3 do PortF para determinar qual botão foi pressionado. GPIOPIN0 (pino 0) do PortB (GPIOPORTBBASE): Este pino é usado para ler a entrada do botão SW5. A função GPIOPinWrite(GPIOPORTBBASE, GPIOPIN0, 0xff) configura o pino 0 do PortB como saída alta (ligada) para acionar o botão SW5. Em seguida, a função GPIOPinRead(GPIOPORTFBASE, GPIOPIN0 — GPIOPIN1 — GPIOPIN2 — GPIOPIN3) lê a entrada dos pinos 0, 1, 2 e 3 do PortF para determinar qual botão foi pressionado.

GPIOPIN1 (pino 1) do PortB (GPIOPORTBBASE): Este pino é usado para ler a entrada do botão SW7. A função GPIOPinWrite(GPIOPORTBBASE, GPIOPIN1, 0xff) configura o pino 1 do PortB como saída alta (ligada) para acionar o botão SW7. Em seguida, a função GPIOPinRead(GPIOPORTFBASE, GPIOPIN0 GPIOPIN1 — GPIOPIN2 — GPIOPIN3) lê a entrada dos pinos 0, 1, 2 e 3 do PortF para determinar qual botão foi pressionado.

GPIOPIN5 (pino 5) do PortB (GPIOPORTBBASE): Este pino é usado para ler a entrada do botão SW10. A função GPIOPinWrite(GPIOPORTBBASE, GPIOPIN5, 0xff) configura o pino 5 do PortB como saída alta (ligada) para acionar o botão SW10. Em seguida, a função GPIOPinRead(GPIOPORTFBASE, GPIOPIN0 — GPIOPIN1 — GPIOPIN2 — GPIOPIN3) lê a entrada dos pinos 0, 1, 2 e 3 do PortF para determinar qual botão foi pressionado.

```
#include "bot.h"
#include <stdint.h>
```

```
#include <stdbool.h>

#include "inc/hw_gpio.h"
#include "driverlib/gpio.h"
#include "inc/hw_ints.h"
#include "driverlib/interrupt.h"
#include "driverlib/pin_map.h"
#include "driverlib/sysctl.h"
#include "inc/hw_memmap.h"
#include "inc/hw_types.h"

#define WRITE_REG(x)                        (*((volatile
    uint32_t *)(x)))

void ConfigureButtons(){

    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOB);
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF);

    int i;

    for(i = 0; i < 10; i++);

    WRITE_REG(GPIO_PORTF_BASE+GPIO_O_LOCK) =
       GPIO_LOCK_KEY;
    WRITE_REG(GPIO_PORTF_BASE+GPIO_O_CR) = 0x01;

    GPIOPinTypeGPIOInput(GPIO_PORTF_BASE, GPIO_PIN_0 |
        GPIO_PIN_1 | GPIO_PIN_2 | GPIO_PIN_3);

    GPIOPadConfigSet(GPIO_PORTF_BASE, GPIO_PIN_0 |
       GPIO_PIN_1 | GPIO_PIN_2 | GPIO_PIN_3,
       GPIO_STRENGTH_2MA, GPIO_PIN_TYPE_STD_WPD);
    GPIOPinTypeGPIOOutput(GPIO_PORTF_BASE, GPIO_PIN_4)
       ;
    GPIOPinTypeGPIOOutput(GPIO_PORTB_BASE, GPIO_PIN_0
       | GPIO_PIN_1 | GPIO_PIN_5);


}

int GetButton(){

    int i, aux;

    for(i = 0; i < 4; i++){
```

```
GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_0 |
   GPIO_PIN_1 | GPIO_PIN_5 , 0x00);
GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_4, 0x00
   );

switch(i){

    case 0:

        GPIOPinWrite(GPIO_PORTF_BASE,
           GPIO_PIN_4 , 0xff);

        aux = GPIOPinRead(GPIO_PORTF_BASE,
           GPIO_PIN_0 | GPIO_PIN_1 |
           GPIO_PIN_2 | GPIO_PIN_3);

        if(aux & GPIO_PIN_0) return 0;
        else if(aux & GPIO_PIN_1) return 1;
        else if(aux & GPIO_PIN_2) return 2;
        else if(aux & GPIO_PIN_3) return 3;

        break;

    case 1:

        GPIOPinWrite(GPIO_PORTB_BASE,
           GPIO_PIN_0 , 0xff);

        aux = GPIOPinRead(GPIO_PORTF_BASE,
           GPIO_PIN_0 | GPIO_PIN_1 |
           GPIO_PIN_2 | GPIO_PIN_3);

        if(aux & GPIO_PIN_0) return 4;
        else if(aux & GPIO_PIN_1) return 5;
        else if(aux & GPIO_PIN_2) return 6;
        else if(aux & GPIO_PIN_3) return 7;

        break;

    case 2:

        GPIOPinWrite(GPIO_PORTB_BASE,
           GPIO_PIN_1 , 0xff);

        aux = GPIOPinRead(GPIO_PORTF_BASE,
```

```
                    GPIO_PIN_0 | GPIO_PIN_1 |
                    GPIO_PIN_2 | GPIO_PIN_3);

                if(aux & GPIO_PIN_0) return 8;
                else if(aux & GPIO_PIN_1) return 9;
                else if(aux & GPIO_PIN_2) return 10;
                else if(aux & GPIO_PIN_3) return 11;

                break;
            case 3:

                GPIOPinWrite(GPIO_PORTB_BASE,
                    GPIO_PIN_5 , 0xff);

                aux = GPIOPinRead(GPIO_PORTF_BASE,
                    GPIO_PIN_0 | GPIO_PIN_1 |
                    GPIO_PIN_2 | GPIO_PIN_3);

                if(aux & GPIO_PIN_0) return 12;
                else if(aux & GPIO_PIN_1) return 13;
                else if(aux & GPIO_PIN_2) return 14;
                else if(aux & GPIO_PIN_3) return 15;

                break;
        }
    }

    return BUTTON_NOT_PRESSED;
}
```

3 simbols.h

nessa .h estão os bitmap utilizados no jogo , esses bitmaps estavam na Nokia.h, mas foram realocados no em uma .h a parte.

```
#ifndef SIMBOLOS_H
#define SIMBOLOS_H

#include "Nokia5110.h"

static const uint8_t ASCII[][5] = {
        {0x00, 0x00, 0x00, 0x00, 0x00} // 20
        ,{0x00, 0x00, 0x5f, 0x00, 0x00} // 21 !
        ,{0x00, 0x07, 0x00, 0x07, 0x00} // 22 "
        ,{0x14, 0x7f, 0x14, 0x7f, 0x14} // 23 #
```

```
,{0x24, 0x2a, 0x7f, 0x2a, 0x12} // 24 $
,{0x23, 0x13, 0x08, 0x64, 0x62} // 25 %
,{0x36, 0x49, 0x55, 0x22, 0x50} // 26 &
,{0x00, 0x05, 0x03, 0x00, 0x00} // 27 '
,{0x00, 0x1c, 0x22, 0x41, 0x00} // 28 (
,{0x00, 0x41, 0x22, 0x1c, 0x00} // 29 )
,{0x14, 0x08, 0x3e, 0x08, 0x14} // 2a *
,{0x08, 0x08, 0x3e, 0x08, 0x08} // 2b +
,{0x00, 0x50, 0x30, 0x00, 0x00} // 2c ,
,{0x08, 0x08, 0x08, 0x08, 0x08} // 2d -
,{0x00, 0x60, 0x60, 0x00, 0x00} // 2e .
,{0x20, 0x10, 0x08, 0x04, 0x02} // 2f /
,{0x3e, 0x51, 0x49, 0x45, 0x3e} // 30 0
,{0x00, 0x42, 0x7f, 0x40, 0x00} // 31 1
,{0x42, 0x61, 0x51, 0x49, 0x46} // 32 2
,{0x21, 0x41, 0x45, 0x4b, 0x31} // 33 3
,{0x18, 0x14, 0x12, 0x7f, 0x10} // 34 4
,{0x27, 0x45, 0x45, 0x45, 0x39} // 35 5
,{0x3c, 0x4a, 0x49, 0x49, 0x30} // 36 6
,{0x01, 0x71, 0x09, 0x05, 0x03} // 37 7
,{0x36, 0x49, 0x49, 0x49, 0x36} // 38 8
,{0x06, 0x49, 0x49, 0x29, 0x1e} // 39 9
,{0x00, 0x36, 0x36, 0x00, 0x00} // 3a :
,{0x00, 0x56, 0x36, 0x00, 0x00} // 3b ;
,{0x08, 0x14, 0x22, 0x41, 0x00} // 3c <
,{0x14, 0x14, 0x14, 0x14, 0x14} // 3d =
,{0x00, 0x41, 0x22, 0x14, 0x08} // 3e >
,{0x02, 0x01, 0x51, 0x09, 0x06} // 3f ?
,{0x32, 0x49, 0x79, 0x41, 0x3e} // 40 @
,{0x7e, 0x11, 0x11, 0x11, 0x7e} // 41 A
,{0x7f, 0x49, 0x49, 0x49, 0x36} // 42 B
,{0x3e, 0x41, 0x41, 0x41, 0x22} // 43 C
,{0x7f, 0x41, 0x41, 0x22, 0x1c} // 44 D
,{0x7f, 0x49, 0x49, 0x49, 0x41} // 45 E
,{0x7f, 0x09, 0x09, 0x09, 0x01} // 46 F
,{0x3e, 0x41, 0x49, 0x49, 0x7a} // 47 G
,{0x7f, 0x08, 0x08, 0x08, 0x7f} // 48 H
,{0x00, 0x41, 0x7f, 0x41, 0x00} // 49 I
,{0x20, 0x40, 0x41, 0x3f, 0x01} // 4a J
,{0x7f, 0x08, 0x14, 0x22, 0x41} // 4b K
,{0x7f, 0x40, 0x40, 0x40, 0x40} // 4c L
,{0x7f, 0x02, 0x0c, 0x02, 0x7f} // 4d M
,{0x7f, 0x04, 0x08, 0x10, 0x7f} // 4e N
,{0x3e, 0x41, 0x41, 0x41, 0x3e} // 4f O
,{0x7f, 0x09, 0x09, 0x09, 0x06} // 50 P
,{0x3e, 0x41, 0x51, 0x21, 0x5e} // 51 Q
```

```
,{0x7f, 0x09, 0x19, 0x29, 0x46} // 52 R
,{0x46, 0x49, 0x49, 0x49, 0x31} // 53 S
,{0x01, 0x01, 0x7f, 0x01, 0x01} // 54 T
,{0x3f, 0x40, 0x40, 0x40, 0x3f} // 55 U
,{0x1f, 0x20, 0x40, 0x20, 0x1f} // 56 V
,{0x3f, 0x40, 0x38, 0x40, 0x3f} // 57 W
,{0x63, 0x14, 0x08, 0x14, 0x63} // 58 X
,{0x07, 0x08, 0x70, 0x08, 0x07} // 59 Y
,{0x61, 0x51, 0x49, 0x45, 0x43} // 5a Z
,{0x00, 0x7f, 0x41, 0x41, 0x00} // 5b [
,{0x02, 0x04, 0x08, 0x10, 0x20} // 5c '\'
,{0x00, 0x41, 0x41, 0x7f, 0x00} // 5d ]
,{0x04, 0x02, 0x01, 0x02, 0x04} // 5e ^
,{0x40, 0x40, 0x40, 0x40, 0x40} // 5f _
,{0x00, 0x01, 0x02, 0x04, 0x00} // 60 `
,{0x20, 0x54, 0x54, 0x54, 0x78} // 61 a
,{0x7f, 0x48, 0x44, 0x44, 0x38} // 62 b
,{0x38, 0x44, 0x44, 0x44, 0x20} // 63 c
,{0x38, 0x44, 0x44, 0x48, 0x7f} // 64 d
,{0x38, 0x54, 0x54, 0x54, 0x18} // 65 e
,{0x08, 0x7e, 0x09, 0x01, 0x02} // 66 f
,{0x0c, 0x52, 0x52, 0x52, 0x3e} // 67 g
,{0x7f, 0x08, 0x04, 0x04, 0x78} // 68 h
,{0x00, 0x44, 0x7d, 0x40, 0x00} // 69 i
,{0x20, 0x40, 0x44, 0x3d, 0x00} // 6a j
,{0x7f, 0x10, 0x28, 0x44, 0x00} // 6b k
,{0x00, 0x41, 0x7f, 0x40, 0x00} // 6c l
,{0x7c, 0x04, 0x18, 0x04, 0x78} // 6d m
,{0x7c, 0x08, 0x04, 0x04, 0x78} // 6e n
,{0x38, 0x44, 0x44, 0x44, 0x38} // 6f o
,{0x7c, 0x14, 0x14, 0x14, 0x08} // 70 p
,{0x08, 0x14, 0x14, 0x18, 0x7c} // 71 q
,{0x7c, 0x08, 0x04, 0x04, 0x08} // 72 r
,{0x48, 0x54, 0x54, 0x54, 0x20} // 73 s
,{0x04, 0x3f, 0x44, 0x40, 0x20} // 74 t
,{0x3c, 0x40, 0x40, 0x20, 0x7c} // 75 u
,{0x1c, 0x20, 0x40, 0x20, 0x1c} // 76 v
,{0x3c, 0x40, 0x30, 0x40, 0x3c} // 77 w
,{0x44, 0x28, 0x10, 0x28, 0x44} // 78 x
,{0x0c, 0x50, 0x50, 0x50, 0x3c} // 79 y
,{0x44, 0x64, 0x54, 0x4c, 0x44} // 7a z
,{0x00, 0x08, 0x36, 0x41, 0x00} // 7b {
,{0x00, 0x00, 0x7f, 0x00, 0x00} // 7c |
,{0x00, 0x41, 0x36, 0x08, 0x00} // 7d }
,{0x10, 0x08, 0x08, 0x10, 0x08} // 7e ~
// ,{0x78, 0x46, 0x41, 0x46, 0x78} // 7f DEL
```

```
        ,{0x1f, 0x24, 0x7c, 0x24, 0x1f} // 7f UT sign
};

static const uint8_t SPECIAL_SIMBOLS[][5] = {

        {0xff, 0xff, 0xff, 0xff, 0xff}
};

static const uint8_t MIX_SIMBOLS[][2] = {

        {0x0, 0x0} // blank square
        ,{0x3, 0x3} // black square
        ,{0x2, 0x1}
        ,{0x1, 0x2}
        ,{0x3, 0x0}
        ,{0x0, 0x3}
};


#endif
```

4 main.c

O código fornecido é um programa de jogo do Snake para um dispositivo que usa uma placa Nokia5110 para exibir a interface do jogo. O programa utiliza interrupções de SysTick para controlar a lógica do jogo e botões para permitir a interação do usuário.

Aqui está uma documentação do código, descrevendo as principais partes e sua funcionalidade:

Macros e Constantes

SNAKEHEAD‘, ‘SNAKEUP‘, ‘SNAKEDOWN‘, ‘SNAKELEFT‘, ‘SNAKERIGHT‘, ‘SNAKEFOOD‘, ‘SNAKEBLOCK‘, ‘SNAKEBLANK‘, ‘SNAKEEASTEREGG‘, ‘SNAKEINVALID‘: constantes que representam os diferentes tipos de células no mapa do jogo.

‘SNAKESTARTX‘, ‘SNAKESTARTY‘: coordenadas iniciais da cabeça da cobra.

INVALIDPOSITION: valor usado para indicar uma posição inválida no jogo.

FOODSTART: posição inicial inválida para a comida.

‘STARTINGTICKS‘: número inicial de ticks para controlar a velocidade do jogo.

‘SNAKESTARTSIZE‘: tamanho inicial da cobra.

Variáveis Globais

‘map[24][42]‘: matriz que representa o mapa do jogo, contendo os diferentes tipos de células.

‘speeduppoints[STARTINGTICKS - 1]‘: array que define os pontos em que a velocidade do jogo aumentará.

‘lastbutton‘: guarda o valor do último botão pressionado pelo usuário.

‘beforebutton‘: guarda o valor do botão pressionado anteriormente.

‘head[]‘, ‘tail[]‘: coordenadas da cabeça e da cauda da cobra.

‘foodexist‘: flag que indica se há comida no mapa.

‘aux‘: variável auxiliar para controlar o movimento da cobra.

‘rand‘: guarda um valor aleatório gerado pelo ‘SysTickValueGet()‘.

‘foodimage‘: indica o tipo de imagem da comida.

‘squares‘: contador de pontos (quantidade de células preenchidas pela cobra).

‘foodticks‘, ‘ticks‘: variáveis para controlar os ticks relacionados à comida e à velocidade do jogo.

‘tickcur‘, ‘foodtickcur‘: contadores de ticks para controle da lógica do jogo.

Funções

‘MapStart()‘: inicializa o mapa do jogo, definindo os limites e a posição inicial da cobra.

‘lost()‘: lida com a lógica quando o jogo é perdido (cobra colide com uma parede ou consigo mesma). Mostra uma mensagem de acordo com a pontuação alcançada e reinicia o jogo.

‘setfood()‘: define a posição da comida no mapa, verificando se a posição está vazia.

‘draw()‘: desenha o mapa do jogo na tela.

‘move()‘: lida com o movimento da cobra, atualizando a posição da cabeça e da cauda. Também verifica colisões com a comida e controla o crescimento da cobra.

‘SysTickHandler()‘: manipulador da interrupção de SysTick. Controla a lógica do jogo, chamando as funções necessárias de acordo com os ticks.

Função ‘main()‘

Configuração do clock do sistema.

Configuração dos botões.

Inicialização do mapa e da tela Nokia5110.

Exibição do título do jogo na tela.

Aguarda o pressionamento de um botão para iniciar o jogo.

Limpa a tela e exibe as instruções do jogo.

Aguarda o pressionamento de um botão para começar a jogar.

Habilita as interrupções de SysTick.

Configuração do período do SysTick.

Habilita as interrupções gerais.

Loop infinito para atualizar o valor do último botão pressionado e continuar o jogo.

Observações

Este código não contém a implementação das funções ‘ConfigureButtons()‘ e ‘GetButton()‘, que são responsáveis pela configuração dos botões e a leitura do botão pressionado, respectivamente.

O código também depende de uma biblioteca chamada ”Nokia5110.h” para controlar a exibição na tela Nokia5110.

```
#include "Nokia5110.h"
#include "bot.h"

#include "driverlib/systick.h"
#include "driverlib/interrupt.h"
#include "driverlib/sysctl.h"

#define SNAKE_HEAD              1
#define SNAKE_UP                2
#define SNAKE_DOWN              3
#define SNAKE_LEFT              4
#define SNAKE_RIGHT             5
#define SNAKE_FOOD              6
#define SNAKE_BLOCK             7
#define SNAKE_BLANK             0
#define SNAKE_EASTER_EGG        8
#define SNAKE_INVALID           255

#define SNAKE_START_X           12
#define SNAKE_START_Y           21
#define SNAKE_START             {SNAKE_START_X,
    SNAKE_START_Y}
#define INVALID_POSITION        255
#define FOOD_START              {INVALID_POSITION,
    INVALID_POSITION}
#define STARTING_TICKS          7

#define SNAKE_START_SIZE        2

char map[24][42];

char speed_up_points[STARTING_TICKS - 1] = {2, 5, 8,
    13, 20, 35};

int last_button = BUTTON_NOT_PRESSED;
int before_button;

char head[] = SNAKE_START;
char tail[] = SNAKE_START;

char food_exist = 0;

int __aux = SNAKE_START_SIZE;

char _rand;
```

```
char _food_image = 2;

char _squares = 0;

char _food_ticks = 5;
char _ticks = STARTING_TICKS;

char _tick_cur = 0;
char _food_tick_cur = 0;

void MapStart(){

    int i, j;

    for(i = 0; i < 24; i++) for(j = 1; j < 42; j++)
       map[i][j] = SNAKE_BLANK;

    for(i = 0; i < 24; i++) map[i][0] = SNAKE_BLOCK;

    for(i = 0; i < 24; i++) map[i][41] = SNAKE_BLOCK;

    for(j = 0; j < 42; j++) map[0][j] = SNAKE_BLOCK;

    for(j = 0; j < 42; j++) map[23][j] = SNAKE_BLOCK;

    map[head[0]][head[1]] = SNAKE_HEAD;
}

void lost(){

    Nokia5110_Clear();

    Nokia5110_SetCursor(1, 3);
    if(_squares < 16)       Nokia5110_OutStringInv("
       YOU SUCK ");
    else if(_squares < 24)  Nokia5110_OutStringInv("
       GAME OVER ");
    else if(_squares < 32)  Nokia5110_OutStringInv("
       NOT SO BAD");
    else if(_squares < 40)  Nokia5110_OutStringInv("
       NOT BAD  ");
    else if(_squares < 50)  Nokia5110_OutStringInv("
       GOOD WORK ");
    else if(_squares < 60)  Nokia5110_OutStringInv("
       VERY GOOD ");
```

```
    else                    Nokia5110_OutStringInv("
       YOU'RE GOOD");

    Nokia5110_SetCursor(5, 2);
    Nokia5110_OutCharInv('0' + _squares%10);
    Nokia5110_OutCharInv('0' + (_squares/10)%10);
    Nokia5110_OutCharInv('0' + (_squares/100)%10);

    uint32_t loop;

    for(loop = 0; loop < 0xfffff; loop++);

    while(GetButton() == BUTTON_NOT_PRESSED);

    for(loop = 0; loop < 0xfff; loop++);

    head[0] = tail[0] = SNAKE_START_X;
    head[1] = tail[1] = SNAKE_START_Y;

    MapStart();

    _squares = 0;

    __aux = SNAKE_START_SIZE;

    _ticks = STARTING_TICKS;

    last_button = BUTTON_NOT_PRESSED;

    Nokia5110_Clear();
}

void set_food(){

    char food[2];

    for(;;){

        food[0] = 2 + _rand%21;
        food[1] = 2 + (_rand - 24)%39;

        if(map[food[0]][food[1]] == SNAKE_BLANK){

            map[food[0]][food[1]] = SNAKE_FOOD;

            food_exist = 1;
```

```
            return;
        }

        _rand -= 50;
    }
}

void draw(){

    int i, j, k;

    char food_not_found = 1;

    for(i = 0; i < 24; i += 4) for(j = 0; j < 42; j++)
       {

        char aux[4] = {(map[i+3][j] == SNAKE_FOOD) ?
           _food_image : (map[i+3][j] != 0),
                       (map[i+2][j] == SNAKE_FOOD) ?
                          _food_image : (map[i+2][j]
                          != 0),
                       (map[i+1][j] == SNAKE_FOOD) ?
                          _food_image : (map[i+1][j]
                          != 0),
                       (map[i][j]   == SNAKE_FOOD) ?
                          _food_image : (map[i][j]
                          != 0)};

        for(k = 0; k < 4; k++) if(aux[k] ==
           _food_image) food_not_found = 0;

        Nokia5110_DrawMix(aux[0], aux[1], aux[2], aux
           [3]);
    }

    if(food_not_found) set_food();

    Nokia5110_SetCursor(0, 0);
}

void move(){

    char prev_tail[2];
    char prev_head[2];
```

```
prev_tail[0] = tail[0];
prev_tail[1] = tail[1];

prev_head[0] = head[0];
prev_head[1] = head[1];

int next_value = SNAKE_INVALID;

if(last_button == 1){

    next_value = map[head[0] + 1][head[1]];

    if(next_value == SNAKE_BLANK || next_value ==
       SNAKE_FOOD) head[0]++;
    else return lost();

    map[prev_head[0]][prev_head[1]] = SNAKE_UP;
}
else if(last_button == 9){

    next_value = map[head[0] - 1][head[1]];

    if(next_value == SNAKE_BLANK || next_value ==
       SNAKE_FOOD) head[0]--;
    else return lost();

    map[prev_head[0]][prev_head[1]] = SNAKE_DOWN;
}
else if(last_button == 4){

    next_value = map[head[0]][head[1] + 1];

    if(next_value == SNAKE_BLANK || next_value ==
       SNAKE_FOOD) head[1]++;
    else if(next_value == SNAKE_BLOCK){

        if(head[0] == 2){

            map[head[0]][head[1]] =
               SNAKE_EASTER_EGG;
        }
        else return lost();
    }
    else if(next_value != SNAKE_EASTER_EGG) return
        lost();
```

```
    if(map[prev_head[0]][prev_head[1]] !=
       SNAKE_EASTER_EGG) map[prev_head[0]][
       prev_head[1]] = SNAKE_LEFT;
    else{

        head[0] = SNAKE_START_X;
        head[1] = SNAKE_START_Y;
    }
}
else if(last_button == 6) {

    next_value = map[head[0]][head[1] - 1];

    if(next_value == SNAKE_BLANK || next_value ==
       SNAKE_FOOD) head[1]--;
    else return lost();

    map[prev_head[0]][prev_head[1]] = SNAKE_RIGHT;
}

if(next_value == SNAKE_FOOD){

    __aux++;
    _squares++;
    food_exist = 0;
    if(_ticks > 1) if(_squares == speed_up_points[
       STARTING_TICKS - _ticks]) _ticks--;
}

int tail_value = map[prev_tail[0]][prev_tail[1]];

if(__aux != 0){

    if(last_button != BUTTON_NOT_PRESSED) __aux--;
}
else{

    if(tail_value == SNAKE_UP) tail[0]++;
    else if(tail_value == SNAKE_DOWN) tail[0]--;
    else if(tail_value == SNAKE_LEFT) tail[1]++;
    else if(tail_value == SNAKE_RIGHT) tail[1]--;
    else if(tail_value == SNAKE_EASTER_EGG){

        tail[0] = SNAKE_START_X;
        tail[1] = SNAKE_START_Y;
    }
```

```
        map[prev_tail[0]][prev_tail[1]] = SNAKE_BLANK;
    }

    map[head[0]][head[1]] = SNAKE_HEAD;
}

void SysTickHandler(){

    char _bool = 0;

    if(++_tick_cur >= _ticks && last_button !=
       BUTTON_NOT_PRESSED){

        if(!food_exist) set_food();

        move();
        _tick_cur = 0;

        before_button = last_button;

        _bool = 1;
    }

    if(++_food_tick_cur >= _food_ticks){

        if(_food_image == 3) _food_image = 2;
        else _food_image = 3;

        _food_tick_cur = 0;

        draw();
    }
    else if(_bool) draw();
}

int main(void) {

    SysCtlClockSet(SYSCTL_SYSDIV_20 | SYSCTL_USE_PLL |
        SYSCTL_XTAL_16MHZ | SYSCTL_OSC_MAIN);

    ConfigureButtons();

    MapStart();

    Nokia5110_Init();
```

```
Nokia5110_Clear();

Nokia5110_SetCursor(4, 2);

Nokia5110_OutStringInv("Snake");

int loop = 0;

while(GetButton() == BUTTON_NOT_PRESSED) if(loop++
    == 0xffff) break;

Nokia5110_Clear();

Nokia5110_SetCursor(1, 4);
Nokia5110_OutStringInv("SW2  - UP  ");

Nokia5110_SetCursor(1, 3);
Nokia5110_OutStringInv("SW5  - LEFT");

Nokia5110_SetCursor(0, 2);
Nokia5110_OutStringInv("SW7  - RIGHT");

Nokia5110_SetCursor(1, 1);
Nokia5110_OutStringInv("SW10 - DOWN");

loop = 0;

while(GetButton() == BUTTON_NOT_PRESSED) if(loop++
    == 0x20000) break;

Nokia5110_Clear();

SysTickEnable();

SysTickIntEnable();

SysTickPeriodSet(400000);

IntMasterEnable();

while (1){

    int aux = GetButton();

    if(aux == 1 || aux == 4 || aux == 9 || aux ==
```

```
            6){

             if(aux == 1 && before_button == 9)
                continue;
             else if(aux == 4 && before_button == 6)
                continue;
             else if(aux == 9 && before_button == 1)
                continue;
             else if(aux == 6 && before_button == 4)
                continue;

             last_button = aux;

             _rand = SysTickValueGet();
        }
    }
}
```