Breakout Game

Grupo: José Gabriel, Pedro Mendes, Thiago Sena
Disciplina: TEC499
Professor: Thiago de Jesus

Problema e Objetivos

- Desenvolver um jogo inspirado em Breakout;
- Capturar o movimento do jogador com acelerômetro existente no Kit desenvolvimento DE1-SoC;
- Acesso e controle do jogo por meio do uso de botões;
- Utilizar a interface VGA para visualização do jogo em um monitor CRT;

Requisitos

- O código deve ser escrito em linguagem C;
- O sistema só poderá utilizar os componentes disponíveis na placa.

Comunicação do processador com a FPGA e seus periféricos

- Utilizações dos **botões** para acesso e controle do jogo;

```c
#ifndef KEYS_H
#define KEYS_H

/**
 * Function KEY_open: opens the pushbutton KEY device
 * Return: 1 on success, else 0
 */
int KEY_open (void);

/**
 * Function KEY_read: reads the pushbutton KEY device
 * Parameter data: pointer for returning data. If no KEYs are pressed *data = 0.
 *  If all KEYs are pressed *data = 0b1111
 * Return: 1 on success, else 0
 */
int KEY_read (int * /*data*/);

/**
 * Function KEY_close: closes the KEY device
 * Return: void
 */
void KEY_close (void);

#endif
```

Comunicação do processador com a FPGA e seus periféricos

- Utilização do **acelerômetro** para controle do jogador;

```
#ifndef accel_H
#define accel_H

/**
 * Opens the 3D-accelerometer accel device
 * Return: 1 on success, else 0
 */
int accel_open (void);

/**
 * Function: accel_read: reads data from the 3D-accelerometer accel device
 * Parameter ready: pointer for returning ready signal (1 if new data, else 0)
 * Parameter tap: pointer for returning the tap signal (1 if tap event, else 0)
 * Parameter dtap: pointer for the double-tap signal (1 if double-tap event,
 *  else 0)
 * Parameter x: pointer for returning acceleration data in the x direction
 * Parameter y: pointer for returning acceleration data in the y direction
 * Parameter z: pointer for returning acceleration data in the z direction
 * Parameter mg_per_lsb: pointer for returning the acceleration-data scale factor
 * Return: 1 on success, else 0
 */
int accel_read (int * /*ready*/, int * /*tap*/, int * /*dtap*/, int * /*x*/,
    int * /*y*/, int * /*z*/, int * /*mg_per_lsb*/);

/**
 * Function: accel_init: initializes the 3D-acceleration device
 * Return: void
 */
void accel_init (void);
```

Comunicação do processador com a FPGA e seus periféricos

- Utilização do **VGA** para visualização do jogo em um monitor CRT;

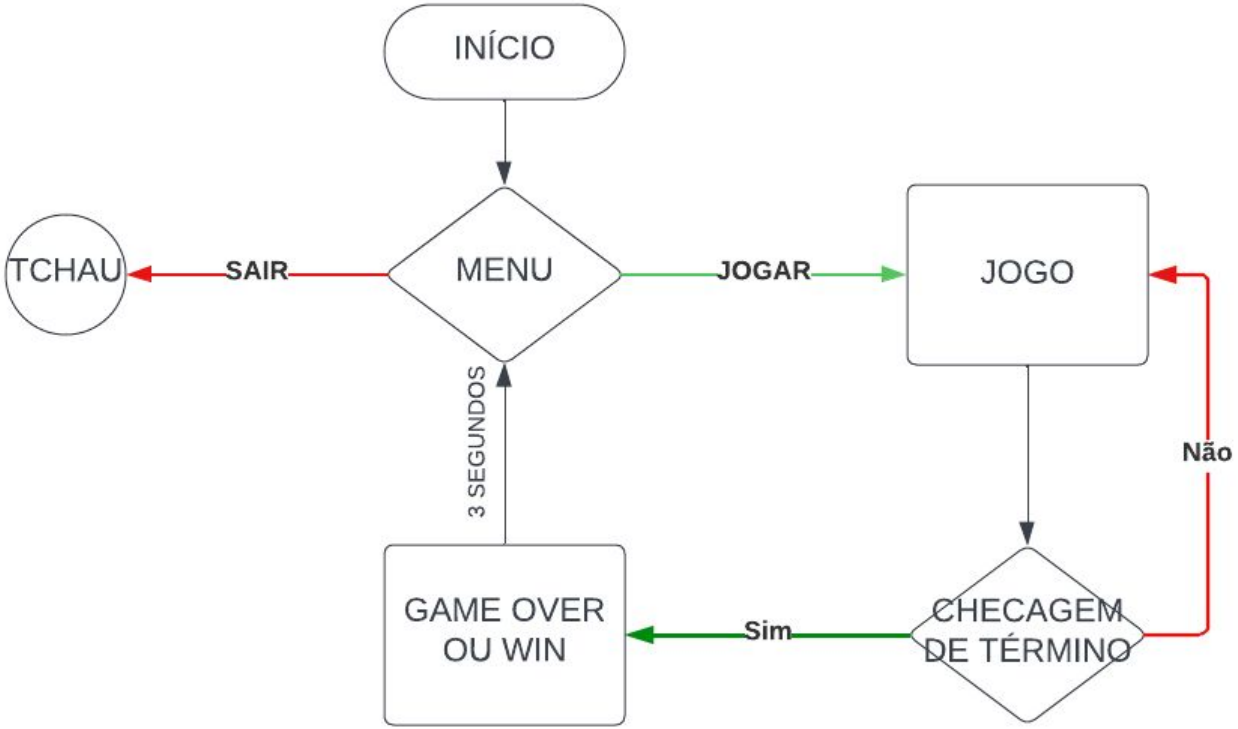
```
#ifndef video_H
#define video_H

#define video_WHITE     0xFFFF   // Define some colors for video graphics
#define video_YELLOW    0xFFE0
#define video_RED       0xF800
#define video_GREEN     0x07E0
#define video_BLUE      0x041F
#define video_CYAN      0x07FF
#define video_MAGENTA   0xF81F
#define video_GREY      0xC618
#define video_PINK      0xFC18
#define video_ORANGE    0xFC00

/**
 * Function video_open: opens the VGA video device
 * Return: 1 on success, else 0
 */
int video_open (void);

/**
 * Function video_read: reads from the video device
 * Parameter cols: pointer for returning the number of columns in the display
 * Parameter rows: pointer for returning the number of rows in the display
 * Parameter tcols: pointer for returning the number of text columns
 * Parameter trows: pointer for returning the number of text rows
 * Return: 1 on success, else 0
 */
int video_read (int * /*cols*/, int * /*rows*/, int * /*tcols*/, int * /*trows*/);
```

Fluxo do Jogo



Lógica do Jogo

- Objetos do jogo:

```
typedef struct{
    int x, y; // Valor inicial de X e Y do player
    int color;
} playerStruct;
```

```
typedef struct{
    int x1, y1;
    int x2, y2;
    int isBroken ; // 1 = broken, 0 = not broken
    int color;
} blockStruct;
```

```
typedef struct{
    int x1, y1; // Valor inicial de X e Y da bola
    int x2, y2; // Valor final de X e Y da bola
    int speedX; // Velocidade da bola no eixo X
    int speedY; // Velocidade da bola no eixo Y
    int color; // Cor da bola
} ballStruct;
```

Lógica do Jogo

- Movimento:

```
void movePlayer(playerStruct *player, int x){

    int moduleX = (int) x / PLR_SENSITIVITY;


    if (moduleX > PLR_SPD){
        moduleX = PLR_SPD;
    } else if (moduleX < PLR_SPD * -1){
        moduleX = PLR_SPD * -1;
    }

    player->x += moduleX;


    // Se o valor de x for menor que 20, mover para a esquerda e verificar se o player não está na parede da esquerda
    if (player-> x <= WALL_SIZE){
        player->x = WALL_SIZE;
    // Se o valor de x for maior que 20, mover para a direita e verificar se o player não está na parede da direita
    } else if ((player->x + PLR_WID) >= (SCR_WID - WALL_SIZE)){
        player->x = SCR_WID - WALL_SIZE - PLR_WID;
    }
}
```

```
void moveBall(ballStruct *ball){
    // Calcula o valor de X e Y de acc
    ball->x1 += ball->speedX;
    ball->y1 += ball->speedY;
    ball->x2 = ball->x1 + BALL_SIZE;
    ball->y2 = ball->y1 + BALL_SIZE;
}
```

Lógica do Jogo

- Colisão:

```
void checkBlockCollision(ballStruct *ball, blockStruct blocks[], int * score){
    int numberOfBlocks = BLK_PER_LINE*COLUMNS_BLK;
    int isBrokeBlock = 0;
    for (int i = 0; i < numberOfBlocks; i++){
        // Verificar se o bloco está quebrado, se não estiver, verificar a colisão
        if (!blocks[i].isBroken){
            // Verificar colisão entre a bola e o bloco
            if (checkBallCollision(ball, blocks[i].x1, blocks[i].y1, BLK_HEI, BLK_WID)){
                // Quebrar o bloco e inverter o ângulo da bola
                if(!isBrokeBlock){
                    calculateBallCollision(ball, blocks[i].x1, blocks[i].y1, BLK_HEI, BLK_WID);
                    isBrokeBlock = 1;
                }
                *score += 100;
                blocks[i].isBroken = 1;

                printf("Colidindo com o bloco %d\n", i);
            }
        }
    }
}
```

Lógica do Jogo

- Colisão:

```
int checkBallCollision(ballStruct *ball, int x, int y, int height, int width){
    return ball->x1 <= x + width && ball->x2 >= x && ball->y1 <= y + height && ball->y2 >= y;
}

//Função que verica qual lado da bola colidiu com alguma estrutura dos lados ou em em cima e em baixo
void calculateBallCollision(ballStruct *ball, int x, int y, int height, int width){
    // Se a bola colidir verticalmente
    if (ball->y1 >= y && ball->y2 <= y + height){
        ball->speedX = invertSpeedSignal(ball->speedX);

    }
    // Se a bola colidir horizontalmente
    else if (ball->x1 >= x && ball->x2 <= x + width){
        ball->speedY = invertSpeedSignal(ball->speedY);
    }
    // Acho que dá pra juntar e virar um ELSE só (Esse caso ele verifica se tocou numa quina)
    else {
        ball->speedY = invertSpeedSignal(ball->speedY);
        ball->speedX = invertSpeedSignal(ball->speedX);
    }
}
```

Lógica do Jogo

- Colisão:

```
void checkPlayerCollision(ballStruct *ball, playerStruct *player){
    // Se a bola colidir com o player
    if (ball->speedY > 0 &&checkBallCollision(ball, player->x, player->y, PLR_HEI, PLR_WID)){
        // Verificar se ela colidiu na metade do player e o angulo dele, para inverter a jogada
        if (ball->x2 >= player->x + PLR_WID/2 && ball->speedX < 0 && ball->speedY > 0){
            ball->speedX = invertSpeedSignal(ball->speedX);
            ball->speedY = invertSpeedSignal(ball->speedY);
        }
        else if (ball->x1 <= player->x + PLR_WID/2 && ball->speedX > 0 && ball->speedY > 0){
            ball->speedX = invertSpeedSignal(ball->speedX);
            ball->speedY = invertSpeedSignal(ball->speedY);
        }
        // se cair do lado normal
        else{
            ball->speedY = invertSpeedSignal(ball->speedY);
        }
    }
}
```