**UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ**

FERNANDO I. N.

# Projeto Estrutura de Dados

Jogo “Snake”

**CURITIBA – PR**

**2021**

## 1. INTRODUÇÃO

Neste documento se está relatado em como fora implementado o programa “Jogo Snake” como projeto final para a disciplina Estrutura de Dados e Algoritmos (ELP33), onde fora utilizado a estrutura Lista Simplesmente Encadeada como principal lógica e funcionamento do mesmo.

## 2. DESCRIÇÃO E CONCEITOS

O “Jogo Snake” consiste em o usuário controlar a direção em que a cobrinha deve percorrer o tabuleiro deve seguir a fim de comer a comida. Em termos mais técnicos, as coordenadas ‘x’ e ‘y’ da cabeça da snake deve ser o mesmo que a da comida. O jogador deve repetir tais feitos a fim de ganhar pontos. A cada vez que a snake come a comida, seu corpo cresce, dificultando o movimento. Colisões com os limites do tabuleiro ou com o próprio corpo da cobra significa fim de jogo.

```
typedef struct sSnakeSegment
{
    int x;
    int y;
    struct sSnakeSegment* prox;
} Snake;
```

*Fig. 1: struct nó da Lista Encadeada. Corpo da cobra.*

O deslocamento do corpo, a imagem do tabuleiro, e identificação das coordenadas foram feitas a partir do auxilio da função gotoxy da <windows.h> e seus auxiliares.

```
void GotoXY(int x, int y)// coordenadas
{
    HANDLE a;
    COORD b;
    fflush(stdout);
    b.X = x;
    b.Y = y;
    a = GetStdHandle(STD_OUTPUT_HANDLE);
    SetConsoleCursorPosition(a,b);
}
```

*Fig. 2: função GotoXY.*

Para a lógica principal do jogo, o movimento, deslocamento, a cabeça da snake fica no início da Lista Encadeada, o primeiro elemento. Assim, para dar a noção de deslocamento uma nova cabeça é inserida no início a cada frame, de acordo com a direção do input, e o ultimo pedaço de sua cauda é removida do final da lista. Tal lógica pode ser diferente do senso comum em que: ao comer a comida se é inserido mais um pedaço da cauda no início da lista, porém tal método se demonstrou muito complexo para quando se é necessário mudar de direção, já que para isso todos as coordenadas do corpo teriam que ter suas informações atualizadas por frame.

O input das direções em tempo real, direita e esquerda é feito via função KeyAsyncKeyState.

```
int bKeyLeft = 0, bKeyRight = 0, bKeyLeftOld = 0, bKeyRightOld = 0;

while(aDead!=2 && bDead!=1)
{
    fflush(stdin);
    bKeyRight = (0x8000 & GetAsyncKeyState((unsigned char)('\x27'))) != 0;
    bKeyLeft = (0x8000 & GetAsyncKeyState((unsigned char)('\x25'))) != 0;

    if (bKeyRight && !bKeyRightOld)
    {
        nSnakeDirection++;
        if (nSnakeDirection == 4)
            nSnakeDirection = 0;
    }

    if (bKeyLeft && !bKeyLeftOld)
    {
        nSnakeDirection--;
        if (nSnakeDirection == -1)
            nSnakeDirection = 3;
    }

    bKeyRightOld = bKeyRight;
    bKeyLeftOld = bKeyLeft;
```

*Fig. 3: leitura do input pelo usuário.*

```c
void pop_back(Snake* l)// remove ultimo elemento
{
    Snake* p;
    for (p=l; p->prox->prox!=NULL; p=p->prox);
    p->prox=NULL;
}
int ultimo_x(Snake* l)// func. aux. retorna ultima coordenada X
{
    Snake* p=l;
    for (; p!=NULL; p=p->prox)
        if(p->prox == NULL)
            return p->x;
}
int ultimo_y(Snake* l)// func. aux. retorna ultima coordenada Y
{
    Snake* p=l;
    for (; p!=NULL; p=p->prox)
        if(p->prox == NULL)
            return p->y;
}
Snake* push_back(Snake* l, int ultimo_x, int ultimo_y)// faz a cobrinha crescer ao comer a Food
{
    Snake* p;
    Snake* novo = (Snake*) malloc(sizeof(Snake));
    for (p=l; p->prox!=NULL; p=p->prox);
    novo->x = ultimo_x;
    novo->y = ultimo_y;
    novo->prox = NULL;
    p->prox = novo;
    return l;
```

*Fig. 4: funções auxiliares ao deslocamento*

```c
//Lógica do jogo | movimento
switch (nSnakeDirection)
{
case 0: // UP
    snake = insere(snake, snake->x, snake->y-1);
    break;
case 1: // RIGHT
    snake = insere(snake, snake->x+1, snake->y);
    break;
case 2: // DOWN
    snake = insere(snake, snake->x, snake->y+1);
    break;
case 3: // LEFT
    snake = insere(snake, snake->x-1, snake->y);
    break;
}
// Deslocamento ao cortar o rabo da cobra
pop_back(snake);
```

*Fig. 5: Lógica do movimento*

Para a checagem de fim de jogo: A cobra não deve encostar nos limites do tabuleiro e não deve morder o próprio corpo. Também se foi implementado o código de meio a comida não ser gerada dentro do corpo da snake, fora do tabuleiro, ou dentro do mesmo.

```
int ourobouros(Snake* snake)// caso a cobra morda ela mesma
{
    Snake* p;
    for(p=snake->prox; p!=NULL; p=p->prox)
        if(snake->x == p->x && snake->y == p->y)
            return 1;//bDead = true
    return 0;
}

int colisao_parede(Snake* snake)// colisao com o limite do tabuleiro
{
    if (snake->x == 0 || snake->x == nScreenWidth)
        return 2;
    else if (snake->y == 1 || snake->y == nScreenHeight)
        return 2;
    else
        return 0;
}
```

*Fig. 6: funções de checagem de fim de jogo.*

```
int dentro(Snake* snake, int food_x, int food_y)// func. aux para impedir que Food seja gerada dentro do corpo da cobra
{
    Snake* p;
    for(p=snake; p!=NULL; p=p->prox)
        if(food_x==p->x && food_y==p->y)
            return 1;
    return 0;
}

// Colisão Cobra X Food | Gerar novo Food
if (snake->x == nFoodX && snake->y == nFoodY)
{
    nScore++;
    srand(time(NULL));
    while(dentro(snake,nFoodX,nFoodY)==1)
    {
        nFoodX = (rand() % nScreenWidth);
        if(nFoodX==0)
            nFoodX=1;
        nFoodY = (rand() % nScreenHeight);
        if(nFoodY==0 || nFoodY==1)
            nFoodY = 2;
    }
    for (int i = 0; i < 3; i++)
        snake = push_back(snake, ultimo_x(snake), ultimo_y(snake));
}
```

*Fig. 7: Prevenção de bug na geração da comida.*

## 3. RESULTADOS

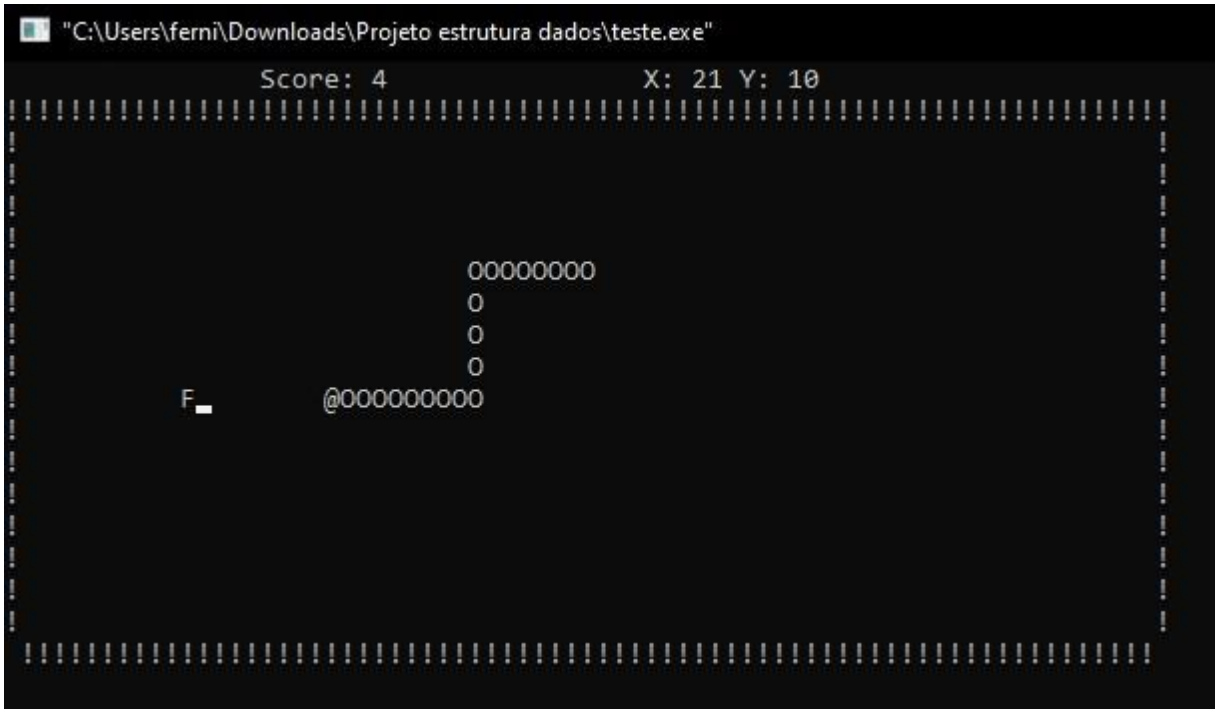


*Fig. 8: Funcionamento do jogo.*

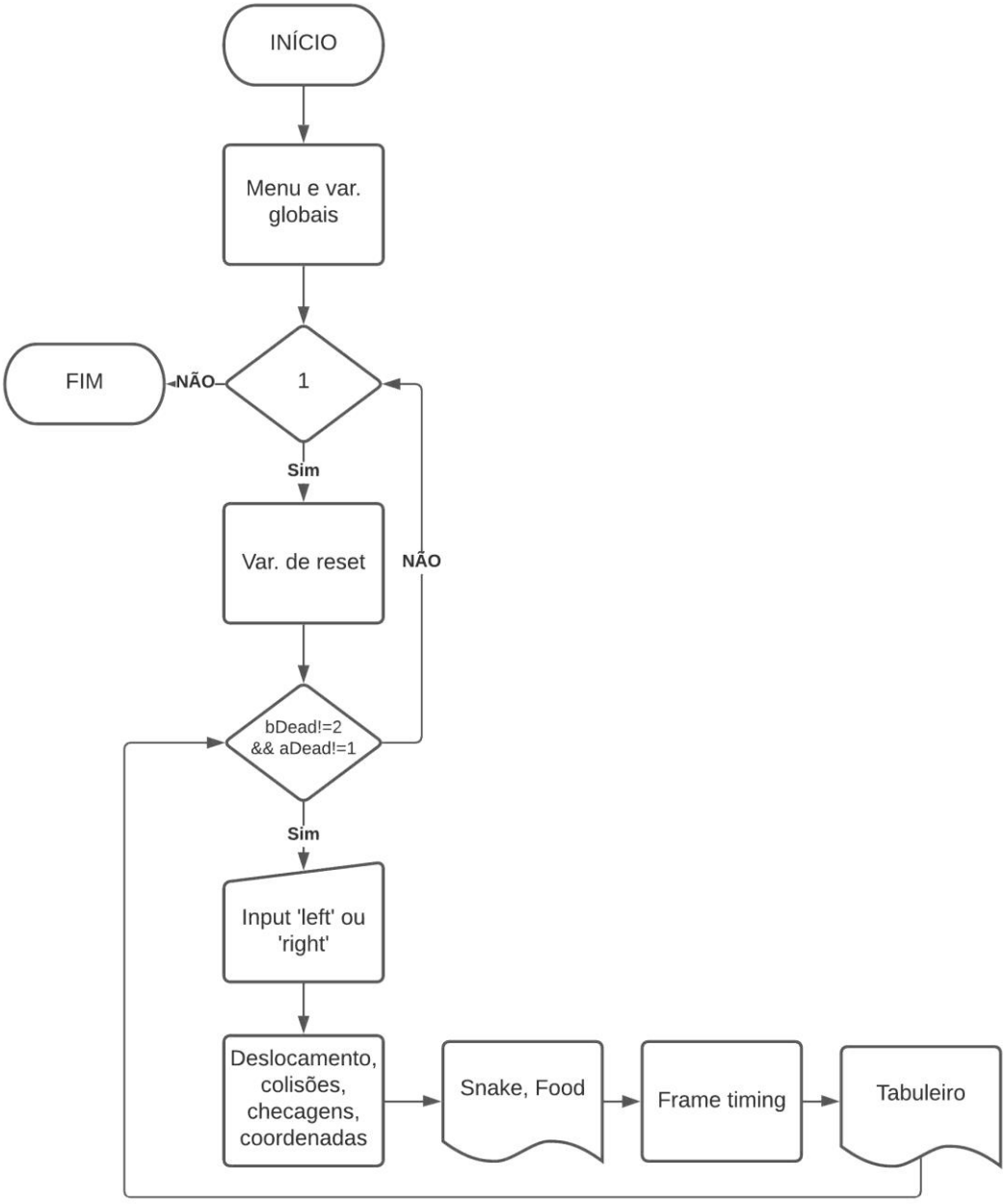


*Fig. 9: Fluxograma.*

## 4. REFERENCIAS

JAVIDX9. “Code-It-Yourself! Snake! - Programming from Scratch (Quick and Simple C++)”. Youtube channel (javidx9). Disponível em: https://www.youtube.com/watch?v=e8lYLYlrGLg&t=891s