Lançamento de Projéteis

LINGUAGENS DE PROGRAMAÇÃO

Grupo 14 | Linguagens para Computação Numérica | 7.05.2018

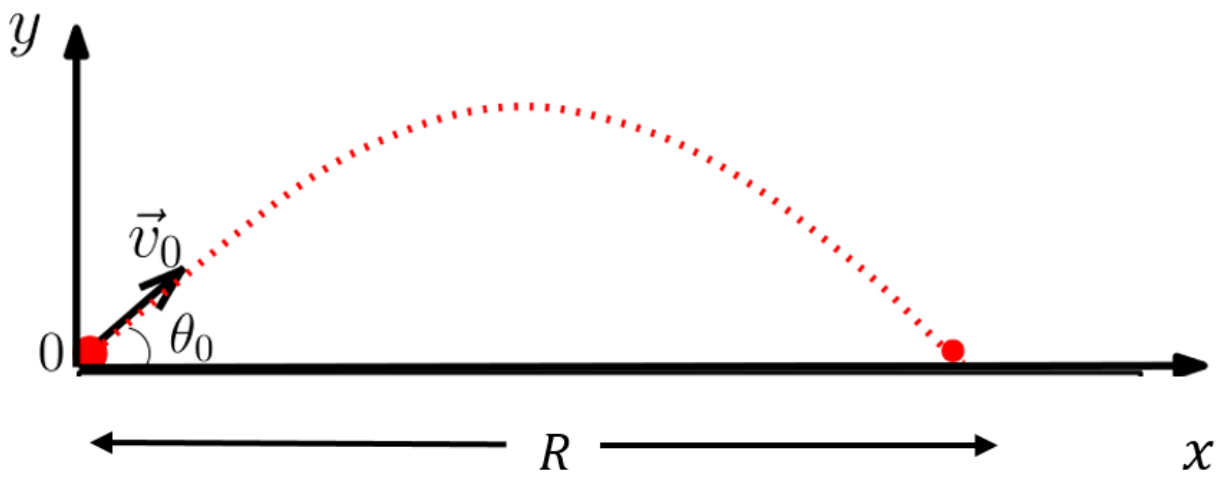
Gilberto Cunha
Henrique Lopes

Introdução

Este projeto foi desenvolvido no âmbito da Unidade Curricular de Linguagens para Computação Numérica, consiste num jogo programado na linguagem "C". Neste jogo, os jogadores têm como objetivo acertar em alvos gerados pelo mapa, através da introdução de dados que vão definir trajetórias.

Os cálculos para este jogo são feitos aplicando as leis da física, mais especificamente, as leis do movimento, após a introdução de uma velocidade e ângulo pelo utilizador, tendo em consideração a força de resistência do ar.

Cada jogador tem 5 turnos para acertar num alvo e o vencedor é aquele que tiver acertado em mais alvos e/ou tiver estado mais próximo destes ao longo dos turnos.



Definição de Objetivos

Para a elaboração deste jogo foi então necessário o desenvolvimento de diferentes funções e algoritmos que funcionem em conjunto de modo a definir e interligar os diferentes aspetos que devem estar presentes no jogo. Foram então delimitados os seguintes objetivos:

1. Estética do Jogo: Não necessariamente uma função, mas um conjunto de impressões que contribuem para uma experiência graficamente mais apelativa e completa durante a execução do jogo;

2. Menu: Uma função interativa que interliga os objetivos do(s) utilizador(es) a correr o programa e o que a função iria executar, constituída pelas opções de jogar e de consultar as instruções do jogo;

3. Alvos: O objetivo principal desta função seria a criação de um algoritmo capaz gerar alvos aleatórios no mapa do jogo, alvos esses que o utilizador irá tentar atingir;

4. Posição do jogador: De modo análogo á geração de alvos, deveria também ser implementada uma função com um algoritmo que criasse uma posição aleatória para o jogador, sempre na parte inferior do mapa;

5. Impressão: Uma função que imprimisse (em tempo real) a trajetória do projétil, bem como o mapa do jogo;

6. Trajetória: Para o desenvolvimento desta função era pretendido que a mesma guardasse os dados da trajetória do jogador e efetuasse todos os cálculos necessários para a criação e posterior impressão da mesma;

7. Pontuação: Para esta função tinha-se o objetivo da criação de um algoritmo de atribuição de um score para cada jogador, de modo a terem uma noção da sua prestação. Tal deve ser imprimido dentro de um ficheiro de texto;

Algoritmos e Funções Desenvolvidas

Para o bom funcionamento deste jogo, tiveram então que ser desenvolvida uma série de funções e algoritmos, que serão agora descritos:

1. Menu

```
int* menu()
{
    int SCANF;
    int c=1;
    int CICLO=1;
    int *a=malloc(sizeof(int)*2);
    int b;

    /* Escolher 1 ou 2 (ASCII - 65 a 122)*/

    printf("1 - Novo Jogo       2 - Instruções\n");

    while(CICLO)
    {
        if( scanf("%d", &b)!=1 || (b!=1 && b!=2) )
            printf("Introduza o número 1 ou 2\n");

        else
            break;
    }

    /* Intruções */

    if(b==2)
    {
        system("clear");
        ANGLYBIRDS();
        printf("O jogo consiste em cada jogador formular
        printf("Deseja voltar ao menu inicial ?\n1 - SHOT

        while(CICLO)
        {
            if(scanf("%d", &c)!=1 || (c!=1 && c!=2) )
                printf("Introduza o número 1 ou 2:\n");

            else if(c==2)
            {
                a[0]=234;
                return a;
            }

            else if(c==1)
            {
                a[0]=233;
                return a;
            }

        }

    }

    /* Jogar */
    else if(b==1)
    {
        /* Dificuldade */

        system("clear");
        ANGLYBIRDS();

        printf("1 - Fácil       2 - Médio       3 - Dific
        while(CICLO)
        {
```

Nesta parte do código do programa foi criada uma função que dá um apontador com 2 posições como return. Esta função não recebe nada como argumento.

Este array está associado ao modo como a função foi desenvolvida. Esta função é um menu gráfico interativo, constituído maioritariamente por printfs e scanfs, que permitem ao jogador selecionar se quer começar um Novo Jogo (premir 1) ou verificar as instruções do mesmo (premir 2).

As instruções são constituídas por um simples printf que detalha o funcionamento do jogo de modo a que o jogador o saiba jogar de modo apropriado.

Dentro da opção "Novo Jogo", o(s) jogador(es) podem também escolher entre três dificuldades:

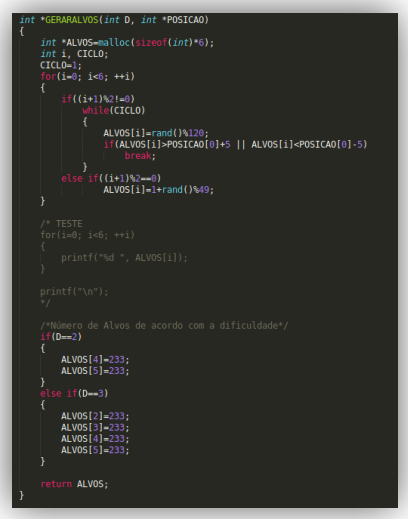
- Fácil: Baixa força de atrito e geração de três alvos para serem atingidos;
- Médio: Força de atrito intermédia e geração de dois alvos;
- Difícil: Força de atrito elevada e apenas um alvo gerado;

Após a escolha da dificuldade, é também pedido na função o número de jogadores, sendo o mínimo permitido 1 jogador e o máximo 4.

A dificuldade e o número de jogadores representam então cada uma das duas posições do array que a função retorna, permitindo a sua interligação com outras funções do jogo.

Estes dados são posteriormente guardados numa estrutura constituída pelas definições do jogo.

2. Gerar Alvos



```
int *GERARALVOS(int D, int *POSICAO)
{
    int *ALVOS=malloc(sizeof(int)*6);
    int i, CICLO;
    CICLO=1;
    for(i=0; i<6; ++i)
    {
        if((i+1)%2!=0)
            while(CICLO)
            {
                ALVOS[i]=rand()%120;
                if(ALVOS[i]>POSICAO[0]+5 || ALVOS[i]<POSICAO[0]-5)
                    break;
            }
        else if((i+1)%2==0)
                ALVOS[i]=1+rand()%49;
    }

    /* TESTE
    for(i=0; i<6; ++i)
    {
        printf("%d ", ALVOS[i]);
    }

    printf("\n");
    */

    /*Número de Alvos de acordo com a dificuldade*/
    if(D==2)
    {
        ALVOS[4]=233;
        ALVOS[5]=233;
    }
    else if(D==3)
    {
        ALVOS[2]=233;
        ALVOS[3]=233;
        ALVOS[4]=233;
        ALVOS[5]=233;
    }

    return ALVOS;
}
```

Esta função gera alvos aleatóriamente pela matriz que contém o mapa do jogo.

Tal foi conseguido através do uso da função srand(), onde se utilizou o "tempo" do computador, presente na biblioteca "time.h", como semente para a geração dos números aleatórios, de modo a evitar a geração de uma sequência recursiva de números "aleatórios" utilizando apenas a função rand(), que iria resultar numa criação de alvos que se iriam repetir ao longo do tempo.

Para cada alvo foi atribuido um x e um y dentro dos limites da matriz, gerados pelo algoritmo descrito anteriormente.

De modo a tornar o jogo mais diversificado, a função verifica qual a dificuldade escolhida pelo jogador, de modo a criar o número de alvos (utilizados) pretendido. No caso da dificuldade ser média ou difícil, 1 ou 2 alvos (respetivamente) são descartados na main, devido á atribuição de um número não aleatório fora do alcance utilizado para a função rand() ás coordenadas do alvo dentro da função "Gerar Alvos".

É finalmente dado como return um apontador com 6 posições (2 para cada alvo) que é associado na main a 3 alvos distintos, sendo cada um um array com 2 posições.

Esta função é executada dentro de dois ciclos na main, de modo a gerar alvos diferentes para cada jogador em cada turno.

3. Gerar Posição

Em tudo idêntica á função anterior. Difere apenas no facto em que gera apenas 1 número aleatório: A posição "x" do jogador.

A posição "y" é definida como zero para todos os casos.

Tal como a função "Gerar Alvos", é chamada para cada jogador, em cada turno.

```
int *GERARPOSICAO()
{
    int *POSICAO=malloc(sizeof(int)*2);
    POSICAO[0]=rand()%120;
    POSICAO[1]=0;
    return POSICAO;
}
```

4. Imprime

```
void IMPRIME(char MAPA[120][50])
{
    int i, j;
    printf("                    ");
    for(i=0; i<122; ++i)
        printf("*");
    printf("\n");
    for (j=0; j<50; ++j)/*Imprimir o Mapa*/
    {
        printf("                     ");
        printf("*");
        for(i=0; i<120; ++i)
            printf("%c", MAPA[i][49-j]);

        printf("*\n");
    }
    printf("                    ");
    for(i=0; i<122; ++i)
        printf("*");
    printf("\n");
}
```

Responsável pela impressão da matriz que contém o mapa do jogo. Recebe apenas essa matriz como argumento.

Consiste num ciclo dentro de um ciclo, representando o y e o x (respetivamente), onde imprime cada caracter da matriz, sendo então constituída apenas por ciclos for e printfs.

Esta matriz é previamente preenchida por espaços na main de forma a evitar erros de impressão, sendo posteriormente alterada para obter a trajetória pretendida.

De modo similar a funções descritas previamente, é chamada 1 vez por cada jogador, 1 vez por cada turno, mas também 1 vez por cada "x" percorrido na função disparo, que será descrita mais tarde, de modo a que o jogador possa receber uma atualização em tempo real da impressão da trajetória.

Esta função não tem return.

5. Trajetória

O núcleo matemático do programa, onde são efetuados quase todos os tipos de cálculos presentes no mesmo. Sendo que é o principal motivo do uso da biblioteca “math.h”.

A função começa por pedir ao utilizador uma velocidade e um ângulo para definir a sua trajetória. A velocidade é diretamente guardada numa estrutura com os dados de cada jogador. O ângulo, no entanto, passa por um algoritmo elementar de conversão de graus em radianos primeiro (a introdução do numero 45, por exemplo, resultará em guardar pi/4 na estrutura dos dados do jogador), devido ás funções trigonométricas do “math.h” trabalharem em radianos e não em graus, sendo então este passo elementar crucial para o jogo funcionar apropriadamente.

Segue-se um ciclo (coordenada x) que percorre a matriz do mapa, começando no “x” da posição do jogador. Por cada passagem neste ciclo são efetuados diversos cálculos, tornando esta função talvez a mais extensa e complexa de todo o programa:

- Tempo: De modo a facilitar o cálculo da altura e a impressão a tempo real da matriz, foi criada uma variável tipo *float, que guarda o tempo passado desde o início da trajetória (Tempo[0]=0) para cada coordenada x, calculado através da fórmula de x(t) que constava no enunciado do projeto.
- Altura: Análogamente, criou-se uma variável *float, que calcula a altura para cada tempo e posição x, onde a matriz MAPA[coordenadax][coordenaday] é posteriormente substituída por um “o”, representando que o projétil se encontra lá naquele momento.
- Altura Máxima: A altura máxima é atingida quando a velocidade no eixo dos yy é zero. Deste modo, o Vy na equação Vy(t) foi substituído por zero e

escreveu-se essa equação em ordem ao tempo. Posteriormente escreveu-se essa fórmula na função, em que o resultado era guardado na variável TempoMax. De modo análogo, calcula-se a altura para o TempoMax, obtendo assim a altura máxima atingida na trajetória.

- Distância Percorrida: Um cálculo que exigiu um pouco de raciocínio de modo a ser simplificado. O método utilizado foi o seguinte: Calculou-se a distância entre cada dois pontos seguidos (de acordo com o eixo do xx) que fazem parte da trajetória. No final somou-se as distâncias, obtendo uma boa aproximação da distância total percorrida.

- Distância Mínima a um dos Alvos: Calculou-se a distância mínima entre cada ponto (x, y) pertencente á trajetória e os diferentes alvos não "descartados". Posteriormente, através de ciclos e o uso do "INT_MAX", da biblioteca "limits.h", descobriu-se qual das distâncias era a menor, guardando-a na estrutura dos dados do jogador. Essa distância é a Distância Mínima.

- Acertar num dos Alvos: Um simples if(), que verifica se a distância mínima é igual a zero. Em caso afirmativo (TRUE), é guardado o valor 1 na variável ACERTO[turno] pertencente novamente á estrutura dos dados do jogador. Utilizou-se o 1 para representar acertar um alvo e zero a representar não acertar.

Para além de todo este cálculo, foram usadas outras duas funções relevantes:

- usleep(): Pertence á biblioteca "unistd.h" e suspende a execução do código por um certo número de microssegundos. Usou-se como argumento um intervalo de tempo correspondente ao Tempo[i]-Tempo[i-1]. Este intervalo de tempo corresponde ao tempo que decorre entre cada dois pontos seguidos da trajetória. Utilizando-o como argumento da função usleep e chamando posteriormente a função Imprime(), conseguiram-se imprimir as trajetórias em tempo real. Como esta função recebe o tempo em microssegundos, o intervalo de tempo foi multiplicado por 10^6.

- system(): Também pertencente á biblioteca "unistd.h", esta função executa códigos de um terminal UNIX. Foi usada especificamente na forma system("clear"), de forma a "limpar" o terminal onde está a ser executado o programa sistematicamente, tornando o programa mais apelativo visualmente. Resultados similares podiam ter sido obtidos com, por exemplo, um printf com vários "\n". Optamos, no entanto, por esta solução de certo modo mais minimalista e simples.

É importante denotar que estas duas funções tornam o programa direcionado a sistemas operativos Unix.

6. Pontuação

Esta funcionalidade foi dividida em 2 funções, Pontuação1() e Pontuação2().

A “Pontuação1()” atribui o score apropriado a cada jogador. O score utilizado é do tipo comparativo. No nosso caso, isto significa que o jogador com a menor distância mínima obtém sempre 10 de score, independentemente do jogador que obtém score 10 noutros turnos ter obtido uma distância mínima maior ou menor. Deste modo, dividiu-se o jogo em 2 casos possíveis: Singleplayer e Multiplayer .

- Singleplayer: O score do jogador é a soma das distâncias mínimas de cada jogada. Quanto menor o score, melhor a prestação do jogador. Optou-se por esta opção porque o score comparativo em singleplayer não fazia sentido, visto que se iria obter sempre a pontuação máxima por não haver outros jogadores a quem comparar.

- Multiplayer: Score Comparativo. O jogador com a menor distância mínima obtém o maior score. As distâncias mínimas foram guardadas por ordem em 4 variáveis: m1, m2, m3 e m4, através do mesmo método como o da determinação da distância mínima entre alvos. Comparando então a distância mínima de cada um dos jogadores (guardada numa estrutura) e estas 4 variáveis, atribuem-se os scores de 10, 9, 8 e 7, respetivamente. Caso os jogadores tenham distâncias mínimas iguais, utilizou-se um processo mais complexo que é discutido na conclusão, devido a ter sido uma das maiores complicações no desenvolvimento deste projeto. Teve-se também em consideração o bónus no caso de acertos consecutivos. Isto foi facilmente alcançado criando uma variável bónus e dizendo que “bónus=bónus+1” no caso da distância mínima ser zero(acerto), sendo que bónus é inicializado como sendo igual a zero. Isto faz com que o bónus seja progressivamente maior á medida que se acerta em mais alvos consecutivos. No caso de não se acertar (distância mínima diferente de zero), o bónus é igualado novamente a zero.

Chegamos então à “Pontuação2()”. Função que apenas passa os dados relativos ao score para um ficheiro de texto, através do uso de FILEs, fprintfs e ciclos. Em adição, e para tornar o jogo um pouco mais competitivo e detalhado, guardaram-se as alturas máximas, distâncias percorridas e distâncias mínimas de cada jogador em cada turno, bem como a taxa de acerto em alvos e distância mínima média (calculadas através de métodos óbvios), numa outra estrutura. Estes dados são também imprimidos dentro do ficheiro, de modo a permitir aos jogadores verificarem as estatísticas do jogo.

Conclusão

Em síntese, os objetivos propostos foram todos atingidos. No entanto, deparámo-nos com algumas adversidades, nomeadamente a impressão das trajetórias em tempo real e o caso de empate em distância mínima.

Na primeira, inicialmente pensou-se que a função sleep fosse resultar, porém esta não funciona de modo adequado com argumentos do tipo float. Deste modo, utilizou-se a função usleep, em tudo idêntica á sleep, mas que recebe um argumento em microssegundos e corre adequadamente com floats, resolvendo o prévio problema.

Quanto ao empate na distância mínima, desenvolvemos um ciclo a percorrer todos os jogadores. Dentro desse ciclo, outro que os percorre novamente. No caso dos jogadores serem diferentes (posições diferentes do apontador) e os seus scores nesse turno serem iguais, aquele que disparou posteriormente (posição superior no apontador), terá o seu score diminuído em 1 unidade. Contudo, isto não era suficiente, pois ao baixar o score desse jogador, este vai igualar outros com distâncias mínimas superiores nesse turno. Novamente foi criado outro ciclo, dentro dos outros 2, também a percorrer os jogadores. Caso os jogadores sejam todos diferentes e o seu score no turno seja igual ou inferior ao score do jogador que viu o seu score reduzido (significa que a sua distância mínima é superior), o score nesse turno do jogador neste novo ciclo será também reduzido por uma unidade. Desta forma, o objetivo pretendido para o funcionamento do score do jogo foi atingido.

Finalmente, relativamente a possíveis melhorias no programa, consideramos a otimização do código a principal possível melhoria. O nosso programa é relativamente extenso e "pesado", pelo que uma redução na extensão do código através do uso de outros métodos mais elementares no cálculo e execução de funcionalidades tornaria o projeto mais fluido e amigável ao hardware do computador a executá-lo.