Universidade Federal de Minas Gerais

Documentação sobre o Trabalho Prático da disciplina
Jogo R-Type
Programação e Desenvolvimento de Software 1

Aluno: Daniel Nogueira Junqueira
Matrícula: 2021072244

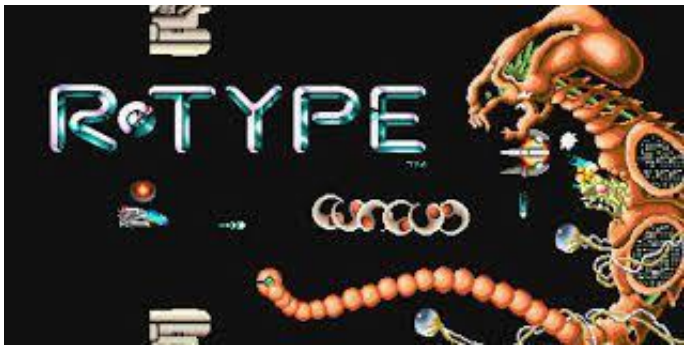

Imagem do jogo, lançado em 1987

1. Introdução

1.1 Descrição

A seguinte documentação tem por objetivo relatar e descrever brevemente a criação e o desenvolvimento de um jogo de tiro bidimensional chamado R-Type, lançado pela fabricante de software japonesa Irem em 1987 no Japão. O desenvolvimento desse jogo foi possibilitado pelo conhecimento adquirido ao longo do curso de Programação e Desenvolvimento de Software 1 (PDS 1) que, juntamente com a utilização da biblioteca Allegro, famosa por facilitar a criação de jogos em duas dimensões (2D), forneceu maior acesso a recursos e habilidades para os alunos para tornar esse trabalho possível.

No protótipo do jogo, o jogador controla uma nave, cujo formato é um triângulo, e seu objetivo é destruir a raça alienígena Bydo que ameaça acabar com a humanidade e o planeta Terra, que possui o formato de um círculo preenchido, portanto depende do jogador para impedir este feito. O jogador possui uma arma capaz de atirar de duas formas diferentes, que também possui o formato de um círculo preenchido, porém menor, o tiro básico, que quando colide com um inimigo o destrói e é destruído, e o tiro mais forte,

em que o raio do tiro aumenta de tamanho duas vezes e destrói todos os inimigos que estiverem em sua frente, até o fim da tela. Além disso, ainda há um obstáculo de forma retangular que passa pela tela que não é possível de ser destruído, cabendo ao jogador desviar desse bloco. O movimento da nave, dos inimigos e do bloco são limitados ao tamanho da tela, e caso o jogador colida com um inimigo ou com o bloco o jogo acaba, mostrando sua pontuação juntamente com um recorde (caso houver), sendo que o sistema de pontuações possui dois critérios: inimigos pequenos valem um ponto, e inimigos grandes valem dois pontos.

1.2 Controles

Tecla W: move a nave para cima.

Tecla S: move a nave para baixo.

Tecla D: move a nave para a direita.

Tecla A: move a nave para a esquerda.

Tecla espaço: aperte para o tiro normal ou aperte e segure durante um segundo e meio e depois solte para o tiro mais forte.

2. Descrição do código

2.1 Bibliotecas e constantes

No início do código há a implementação das bibliotecas utilizadas, tanto do Allegro quanto da linguagem C para o funcionamento do restante do código, e logo depois começa as constantes definidas por meio do ´´const int´´ que são utilizadas ao longo do algoritmo armazenando valores que não podem ser modificados ao longo da execução do jogo.

2.2 Estrutura dos objetos necessários ao jogo

Para cada tipo de objeto no cenário do jogo, há uma struct declarada para uma maior organização e manipulação deles, começando na linha 36:

Struct nave, possuindo as variáveis direção x (dir_x), direção y (dir_y), coordenadas x e y, velocidade (vel), e sua cor.

Struct bloco, possuindo as coordenadas x e y, seu tamanho (w), sua altura (h), e sua cor.

Struct bola, que representa o tiro, possuindo as coordenadas x e y, o raio, ativo (para quando for o tiro normal), poder (para quando for o tiro mais forte), e um contador (cont) para que seja possível realizar o tiro mais forte.

Struct inimigo possuindo as coordenadas x e y, raio, velocidade do inimigo (vel_i), ativo (para ver se tem inimigo ativo na tela), e sua cor.

2.3 Funções criadas e procedimentos

Sumário:

1. *Void init globais*( ), possuindo como código o comando para criar a cor do cenário.

2. int newRecord (*int score, int *record*), função criada para abrir um arquivo ´´recorde.txt´´ para a leitura do recorde, e se o recorde atual for menor que o score (pontuação) o arquivo é aberto novamente para a escrita da nova pontuação que será o novo recorde.

3.1. void initBloco (*Bloco *bloco*), função criada para definir os valores das variáveis passadas para a struct do Bloco citada anteriormente.

3.2. void initNave (*Nave *nave*), função criada para definir os valores das variáveis passadas para a struct da Nave citada anteriormente.

3.3. void initBola (*Bola *bola, Nave *nave),* função criada para definir os valores das variáveis passadas para a struct da Bola (tiro) citada anteriormente.

3.4. void initInimigo (*Inimigo *inimigo, Bloco *bloco*), função criada para definir os valores das variáveis passadas para a struct do Inimigo citada anteriormente.

4. void desenhaCenario( ), procedimento criado para desenhar e limpar o cenário para a cor desejada.

4.1. void desenhaNave (*Nave nave*), procedimento criado que utiliza uma outra função da biblioteca Allegro para criar um triângulo com suas coordenadas x e y dos seus três pontos, e sua cor.

4.2. void desenhaBloco(*Bloco bloco*), procedimento criado que utiliza uma outra função da biblioteca Allegro para criar um retângulo com suas coordenadas x e y do ponto superior esquerdo e do ponto inferior direito, e sua cor.

4.3. void desenhaBola(*Bola bola*), procedimento feito que utiliza uma função da biblioteca Allegro para criar um círculo preenchido com as coordenadas x e y, o raio e a cor do círculo.

5. void atualizaBloco (*Bloco *bloco*), procedimento criado para a movimentação do bloco da direita para a esquerda na tela, e a criação de um novo bloco caso o antigo ultrapasse por completo o limite da tela.

5.1. void atualizaNave (*Nave *nave*), procedimento feito para dar movimento a nave e limitar a sua movimentação nos limites tanto laterais quanto inferiores e superiores da tela.

5.2. void atualizaBola (*Bola *bola, Nave *nave, Bloco *bloco*), procedimento criado para caso o tiro ultrapasse a tela ele é reinicializado, e independente se for o tiro normal (bola->ativo) ou o tiro mais forte (bola->poder) a velocidade do tiro é a mesma, e a coordenada do tiro recebe as coordenadas da nave para o tiro ficar na ponta da nave.

6. void liberaInimigo (*Inimigo *inimigo*), procedimento feito para que de tempos em tempos, quando um número randômico entre 0 e 499 for igual a 0, faz com que libere mais inimigos dificultando o jogador.

6.1. void atualizaInimigo (*Inimigo *inimigo, Bloco *bloco*), procedimento criado para dar movimento ao inimigo e delimita-lo dentro da tela e caso ele ultrapasse o limite da tela, reinicie ele.

6.2. void desenhaInimigo (*Inimigo *inimigo*), procedimento feito para desenhar os vários inimigos usando uma função da biblioteca Allegro para desenhar um círculo preenchido utilizando as coordenadas x e y do círculo, o raio e a cor.

7. int colisaoBolaInimigo *(Bola *bola, Inimigo *inimigo, Bloco *bloco, Nave *nave)*, procedimento criado para verificar a colisão entre o tiro e o inimigo, valendo tanto para o tiro mais forte (bola->poder) quanto para o tiro normal (bola->ativo), utilizando teorema de Pitágoras, em que o cateto1 é a coordenada x do inimigo menos a coordenada x do tiro, e o cateto2 é a

coordenada y do inimigo menos a coordenada y do tiro, e é calculada a distancia por meio de Pitágoras, e se essa distancia for menor que a soma do raio do inimigo e o raio do tiro é porque houve colisão. Também é analisado nos dois tipos de tiros o raio do inimigo, se o raio for maior ou igual a oitenta (80) é para somar 2 no score (pontuação), já se for menor que 80 soma apenas 1 na pontuação.

7.1. int colisaoNaveBloco (*Nave nave, Bloco bloco),* analisa a colisão entre a nave e o bloco, retornando 0 se não houver colisão ou 1 se houver colisão.

7.2. int colisaoBolaBloco *(Bola *bola, Bloco *bloco, Nave *nave),* analisa a colisão entre o tiro e o bloco, e se houver apenas o tiro é destruído, enquanto o bloco permanece intacto.

7.3. int colisaoNaveInimigo (*Nave *nave, Inimigo *inimigo),* analisa a colisão entre a nave e um inimigo, retornando 0 se não ocorrer ou 1 se ocorrer a colisão.

7.4. int colisaoNaveInimigos (*Nave *nave, Inimigo *inimigos*), procedimento criado de forma a facilitar o código e verificar a colisão com todos os inimigos, ao invés de só um, chamando a função anterior e caso ela aconteça (que significa que vai ter colisão) é retornado 1, e se não houver colisão retorna 0.

7.5. int colisaoInimigoBloco (*Inimigo *inimigo, Bloco *bloco),* procedimento criado para verificar a colisão entre um inimigo e o bloco.

7.6. void colisaoInimigoInimigo (*Inimigo *inimigo),* procedimento criado para verificar a colisão entre os inimigos, percorrendo-os com ´´loops´´ feitos utilizando duas variáveis diferentes, e verificando se eles estão ativos na tela, e depois é calculado a distancia entre esses inimigos utilizando a fórmula de distância entre dois pontos da geometria, e caso essa distância seja menor que a soma do raio desses dois inimigos, significa que eles colidiram então eles não estão mais ativos e recebem 0.

2.4 Função principal (int main)

2.4.1. No início eu declaro um char my_score[20] para receber o score onde será computado a pontuação mais tarde, e é feito a rotina de inicialização padrão do Allegro, começando na linha 413 até a linha 497, e depois disso eu apenas faço a chamada das funções inicializadoras já citadas, que começam com ´´init´´, e declaro um vetor para o Inimigo e faço um loop em especifico para a função de initInimigo, pois é necessário para que apareça vários inimigos na tela ao invés de apenas um.

2.4.2. Agora, enquanto o jogo é executado (while(playing)) é definido que playing = 1, e se o tipo de evento for um evento do temporizador (linha 526), eu incremento o contador do tiro (bola.cont) e faço a chamada das funções desenha, atualiza, libera e colisão que já foi explicada anteriormente, e caso haja colisão entre o tiro e o inimigo (colisaoBolaInimigo), uso um sprintf para processar o score (pontuação) e atualizar a pontuação. Logo depois disso, é utilizado uma função da biblioteca Allegro para que a palavra ´´score´´ e o atual score apareça no canto superior esquerdo da tela do jogador, com o tamanho e cor desejado. Depois disso, o jogo só é executado enquanto ´´playing´´ for diferente da resposta da colisão entre a nave e o bloco e a colisão entre a nave e os inimigos, por exemplo enquanto não há colisão é retornado 0, e já definimos anteriormente que playing = 1, então o jogo continua. Depois é utilizado uma função da biblioteca Allegro para atualizar a tela.

2.4.3. Agora, na linha 566, se o tipo de evento for o fechamento da tela (clicando no x da janela), o jogo é fechado.

2.4.4. Depois disso, é analisado se o tipo de evento for o pressionar de uma tecla, possuindo o key_down (tecla abaixada) e key_up (tecla levantada), em que é colocado o movimento preciso da nave pelas teclas W, S, A, D por meio desse modo, e a tecla espaço usada para atirar, em que o contador do tiro (bola.cont) citado anteriormente recebe 0 quando a tecla é abaixada, e quando ela é levantada eu analiso se o tempo que ela ficou abaixada foi maior ou igual ao TEMPO_SUPER_TIRO, constante definida no começo do código e, se for, o tiro mais forte se torna verdadeiro (bola.poder = 1) e o raio do tiro é dobrado, enquanto o tiro normal está apenas no evento de tecla levantada (bola.ativo = 1).

2.4.5. Na linha 628 é declarado o fim de procedimentos enquanto o jogo está executando.

2.4.6. Agora, na linha 630 eu declaro uma string chamada char my_text[20] com o objetivo de receber um texto e depois imprimi-lo por meio do sprintf para poder visualizar o ´´Score: ´´ (pontuação) em uma nova tela quando o jogo acabar e utilizando uma função do Allegro para desenhar o texto. Depois disso, é chamado a função inicial já citada newRecord e se ela for verdadeira, é porque o recorde foi batido e é imprimido na tela ´´New Record!´´, e caso isso não seja verdadeiro, é colocado o recorde que já estava antes, se houver algum, junto com o score. A partir disso, na linha 647 é utilizada uma função do Allegro para reinicializar a tela, e logo em seguida

uma outra função dessa biblioteca em que a nova tela com o Score e o Record é para ser mostrado durante 2 segundos.

2.4.7. A partir da linha 656, procedimentos de fim de jogo e retornando a 0.