# Trabalho Prático: R-type

Gabriela Tavares Barreto
Matrícula: 2018074657

## Introdução:

Como trabalho prático da disciplina de programação e desenvolvimento de software I, no semestre de 2021/2, foi proposto utilizar a biblioteca da Allegro, que reúne uma coleção de funções diversas e que permite, com relativa facilidade e simplicidade, criar jogos utilizando a linguagem C. A partir dessa biblioteca, e usando todos os conhecimentos adquiridos ao longo do semestre, eu pude construir uma versão simplificada do jogo R-type. Dessa forma, eu pude ver de forma muito palpável possibilidades de aplicação do que foi aprendido na disciplina. Por causa disso, eu gostaria de ressaltar o quanto esse projeto foi pessoalmente bastante gratificante para mim, principalmente como forma de avaliar meus conhecimentos, e com certeza foi a atividade mais divertida que fiz nesse semestre.

## Como jogar

Neste jogo o jogador controla uma nave espacial de formato triangular, que enfrenta dois tipos de inimigos: blocos e bolas. É possível controlar a nave usando as teclas A, W ,S e D do teclado.

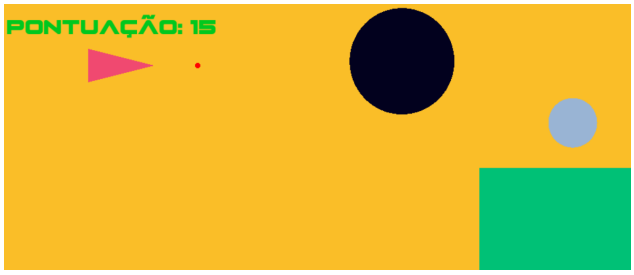

**Tecla A ->** a nave vai para a esquerda

**Tecla D ->** a nave vai para a direita

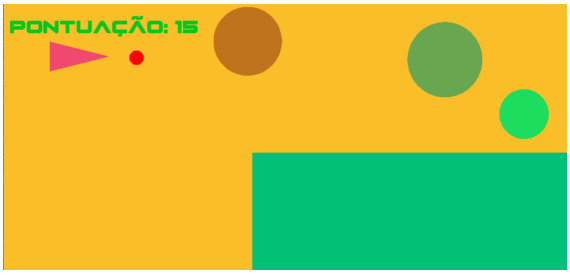
**Tecla W ->** a nave vai para cima

**Tecla S ->** a nave vai para baixo

Além disso, é possível acionar o comando de tiro usando a **barra de espaço.** Clicando na barra uma vez é acionado um tiro do tipo simples, que mata o inimigo do tipo bola e ressurge na nave novamente após colidir com um inimigo ou se tiver percorrido toda a tela. Já se a barra espaço for apertada continuamente, o tiro cresce até atingir um tamanho máximo, tornando-se um tiro avançado. Ao soltá-lo, esse tipo de tiro mata todos os inimigos pela frente, e só ressurge na nave após atravessar toda a tela.




**Tiro simples**




**Tiro avançado**

Só as bolas inimigas podem ser destruídas usando os tiros. Os blocos são indestrutíveis, e devem ser desviados pela nave espacial. Se a nave esbarrar em qualquer um dos dois tipos de inimigos (bloco ou bola), o jogador perde e o jogo acaba.

## Funções e estruturas

A partir da linha dezenove, até a linha trezentos e sessenta do arquivo **rtype.c**, foram implementadas diferentes estruturas e funções a serem usadas no jogo. Agora, irei explicar um pouco sobre cada uma delas.

### Estruturas

Linhas 19 a 40:

- **struct Nave:** estrutura para armazenar as coordenadas (x, y) da nave, direção de movimento, velocidade e cor.
- **struct Bloco:** estrutura para armazenar altura, largura, cor e coordenadas do bloco do jogo.
- **struct Tiro**: estrutura para armazenar as coordenadas, raio e cor do tiro do jogo.
- **struct Inimigo:** estrutura para armazenar as coordenadas, raio, velocidade e cor do tiro do jogo.

### Funções:

Linhas 41 a 360:

- **void desenha_cenario():** função para desenhar o cenário do jogo, mantendo a mesma cor de fundo.
- **float randf():** gera números aleatórios em 0 e 0,99.
- **void initNave(Nave *nave ):** recebe uma variável do tipo Nave como parâmetro por passagem de referência, e atribui valores aos seus campos.
- **void initBloco(Bloco *bloco ):** recebe uma variável do tipo Bloco como parâmetro por passagem de referência, e atribui valores aos seus campos.
- **void initTiro (Tiro *tiro, Nave nave):** recebe uma variável do tipo Tiro por passagem de referência, e uma variável nave, para atribuir valores às coordenadas do tiro em função do das coordenadas da nave.
- **void init_inimigo(Inimigo **array_enemies, int j):** recebe um array com itens do tipo Inimigo, por passagem de referência, e um inteiro correspondente a um índice no array. Essa função acessa uma única posição no array, com o objetivo de inicializar um único inimigo, com valores aleatórios de posição, cor e tamanho, usando as funções randf() e rand().
- **void init_todos_inimigos(Inimigo **array_enemies, int num_enemies):** inicializa todos os campos, de todos inimigos que estão dispostos em um array de inimigos, com valores aleatórios de posição, cor e tamanho, usando as funções randf() e rand().
- **void desenha_tiro(Tiro tiro):** desenha o tiro obedecendo os valores dos campos do tiro passado como parâmetro.

- **void desenha_nave(Nave nave):** desenha a nave seguindo os valores dos campos da nave passada como parâmetro.
- **void desenha_bloco(Bloco bloco):** desenha o bloco seguindo os valores dos campos do bloco passado como parâmetro.
- **void desenha_inimigos(Inimigo *array_enemies, int num_enemies);** desenha todos os inimigos do array de Inimigos, de acordo com suas coordenadas e cores respectivas.
- **void atualiza_Tiro(Tiro *tiro, Nave nave, int *atira, int colidiu_tiro_bloco, int aumenta_tiro):** atualiza o tiro de acordo com três possíveis situações;
    1. O jogador mandou a nave atirar **(atira == 1)**: o tiro terá sua coordenada x atualizada somente, para que se locomova na tela no sentido esquerdo.
    2. O tiro colidiu com o bloco **(colidiu_tiro_bloco != 0)** ou passou da largura da tela: as coordenadas do tiro serão atualizadas para que ele ressurja novamente na ponta da nave.
    3. O jogador está pressionando a tecla de espaço continuamente para lançar um tiro avançado **(aumenta_tiro == 1):** o raio do tiro é aumentado a uma taxa de 1,01, até atingir o valor máximo de 12 pixels.
    4. Nenhuma das situações acima: as coordenadas do tiro serão correspondentes a da nave.
- **void atualiza_bloco(Bloco *bloco):** atualiza a coordenada x do bloco, para que ele se locomova para a esquerda ao longo do tempo.
- **void aloca_inimigos(Inimigo **array_enemies, int num_enemies):** cria, por alocação dinâmica de memória, um vetor com espaço de memória para uma quantidade num_enemies de estruturas do tipo Inimigo.
- **int colisao_inimigo_bloco(Inimigo *array_enemies, Bloco bloco, int i):** verifica se houve uma colisão do bloco com algum dos inimigos. Retorna 1 no caso de colisão, e 0 caso contrário.
- **int colisao_tiro_bloco(Tiro tiro, Bloco bloco):** verifica se houve uma colisão do tiro da nave com o bloco. Retorna 1 no caso de colisão, e 0 caso contrário.
- **int colisao_tiro_inimigo(Tiro tiro, Inimigo *array_enemies, int i, Nave nave, int *pontuacao):** verifica se houve uma colisão entre o tiro e algum dos inimigos do jogo, e caso sim, armazena a pontuação equivalente ao acerto do inimigo na variável pontuação. Retorna 1 no caso de colisão, e 0 caso contrário.
- **int colisao_inimigo_inimigo(Inimigo *array_enemies, int j, int i)**: verifica se os inimigos colidiram entre si. Retorna 1 no caso de colisão, e 0 caso contrário.
- **int colisao_nave_inimigos(Nave nave, Inimigo *array_enemies, int i):** verifica se a nave colidiu com algum dos inimigos. Retorna 1 no caso de colisão, e 0 caso contrário.
- **void atualiza_inimigos(Inimigo **array_enemies, int num_enemies, Bloco bloco, Tiro *tiro, Nave nave, int *atira, int *playing, int *pontuacao):** usando um **for**, essa função percorre cada inimigo do **array de inimigos (array_enemies)**, conferindo possíveis colisões dos inimigos, os atualizando. Os casos contemplados são:
    1. O inimigo **passou da coordenada x=0 da tela:** ele é inicializado novamente com uma posição, cor e velocidade aleatória pela função init_inimigo.

2. O inimigo colidiu com o bloco **(colisao_inimigo_bloco(*array_enemies, bloco, i) == 1**): ele é inicializado novamente com uma posição, cor e velocidade aleatória pela função init_inimigo.
3. Algum inimigo colidiu com o tiro **(colisao_tiro_inimigo(*tiro, *array_enemies, i, nave, pontuacao) == 1 ):** o inimigo é inicializado pela função init_inimigo, e caso o tiro não seja do tipo avançado, é chamada a função initTiro, ou seja, o tiro irá parar de percorrer pela tela e ressurgir na ponta da nave.
4. A nave colidiu com algum inimigo **(colisao_nave_inimigos(nave, *array_enemies, i) == 1):** nesse caso a variável **playing é atualizada para 0,** para que o jogo seja encerrado.
5. Um segundo for é utilizado para verificar se houve alguma colisão entre os inimigos, chamando a função **colisao_inimigo_inimigo,** e caso positivo, os inimigos são reiniciados.
6. Por fim, é atualizada a coordenada x de todos inimigos do array, de acordo com suas respectivas velocidades.

- **int colisao_nave_bloco(Nave nave, Bloco bloco):** verifica se a nave colidiu com o bloco, se sim, ela retorna 1, e caso contrário, retorna 0.
- **int Recorde(int pontuacao, int *recorde):** ao jogar pela primeira vez essa função cria um arquivo do tipo .dat, onde irá armazenar o valor da pontuação recorde do jogo. Nas vezes seguintes que o jogo for usado, ela irá acessar o arquivo recorde.dat já existente para que sempre seja exibido ao final do jogo o recorde previamente estabelecido pelo jogador.

## Programa principal

Para explicar de forma sucinta o **main** do programa rtype.c, podemos dividi-lo em 4 partes:

1. **Linhas 360 a 460:** são feitos procedimentos padrão de inicialização, como inicializar o Allegro, instalar primitivas, instalar o teclado, criar uma fila de eventos , criar variáveis de jogo (nave, inimigos e bloco), iniciar o timer, entre outros procedimentos.
2. **Linhas 460 a 593**: é rodado um while em função de uma variável **playing** ser true. Dentro desse while são esperados eventos, e de acordo com o tipo de evento registrado, são chamadas funções diferentes. O while só será encerrado quando for verificado a colisão da nave com um dos inimigos ou bloco, ou se o jogador fechar a tela.
    a. Evento de tipo **tempo:** são chamadas as funções que verificam se houve colisões entre os inimigos, bloco e nave. Depois disso, são chamadas as funções que atualizam as variáveis dessas estruturas do jogo, para que sejam chamadas às funções que desenham essas estruturas de forma atualizada.
    b. Evento do tipo **fechamento de tela:** o while é encerrado e o jogo acaba.
    c. Evento do tipo **pressionamento de alguma tecla** do teclado: verifica se a tecla pressionada é alguma das teclas usadas no jogo(A,S,W,D ou espaço), e atualiza determinadas variáveis de jogo de acordo com isso.
3. **Linhas 595 a 618:** agora, é rodado um while que é verdadeiro quando o playing for falso. Dentro desse while a tela é atualizada, é comandada a exibição da pontuação do jogador e

é chamada a função Recorde para exibir o recorde também. Esse while só é encerrado quando o jogador comanda o fechamento da tela.

4. **Linhas 621 a 629:** por fim, são feitos os procedimentos de fim de jogo , como limpar a tela, terminar o timer e destruir a fila de eventos. Depois, zero é retornado e o programa se encerra.