UNIVERSIDADE FEDERAL DE MINAS GERAIS

PROGRAMAÇÃO E DESENVOLVIMENTO DE SOFTWARE I

Documentação do Trabalho Prático

R-Type

Bruno de Souza Lages

Belo Horizonte

2021

Introdução **4**

Jogabilidade **4**

Controles 4

Obstáculos 5

Paredes 5

Chão 5

Inimigos 5

Inimigos simples 5

Robôs 5

Robôs de chão 5

Canhões 6

Addons e power-ups 6

Chave de fenda 6

Gasolina 6

Organização do Código **6**

equation.h 7

Paralelogram 7

Circle 8

points_distance() 8

angle() 8

wave_equation() 8

background.h 9

Estrelas 9

obstacles.h 9

spaceship.h 9

shot.h 9

supershot.h 10
enemy_shot.h 10
enemies.h 10
basic_enemies 10
Robôs 11
Robôs de chão 11
Canhões 11
power_up.h 12
collision_equations.h 12
Colisão retangular 12
Colisão circular 12
Colisão entre círculos e retângulos 12
collisions.h 13
Implementação **13**

1. Introdução

Na disciplina de Programação e Desenvolvimento de Software I foi-se proposto como trabalho prático a elaboração do jogo R-type. R-type é um jogo arcade bidimensional de tiros lançado em 1987. Nele, é preciso pilotar a nave R-9 enquanto se enfrenta os diversos inimigos do império Bydo. O jogo se mantém enquanto a nave não é atingida.

Para fins práticos e criativos, algumas funcionalidades foram implementadas ou alteradas. O jogo pode ser acessado clicando aqui ou pelo link: https://drive.google.com/drive/folders/1VqC9vIry3Y2t4_nEgj6ZAP9E6p3MVnlb?usp=sharing

2. Jogabilidade

2.1. Controles

Os jogadores podem apertar as teclas:

- Seta esquerda ou A: para movimentar a nave para a esquerda.
- Seta direita ou D: para movimentar a nave para a direita.
- Seta para cima ou W: para movimentar a nave para cima.
- Seta para baixo ou S: para movimentar a nave para baixo.
- Espaço: para atirar um tiro simples.
- Esc: para pausar ou despausar o jogo.

Ou ainda, pressionar as teclas:

- Seta para cima ou W: para movimentar a nave para cima em uma velocidade maior.
- Seta para baixo ou S: para movimentar a nave para baixo em uma velocidade maior.
- Espaço: para carregar um tiro especial, que ao soltar eliminará todos inimigos e seus tiros no caminho.

2.2. Obstáculos

Obstáculos são blocos com velocidade constante que não podem ser destruídos por nenhum outro elemento presente no jogo. Existem dois tipos de obstáculos no jogo:

2.2.1. Paredes

As paredes são blocos de tamanho e posição que variam.

2.2.2. Chão

O chão é um bloco que é constante no canto inferior do mapa.

2.3. Inimigos

Existem quatro tipos de inimigos no jogo. Entre os inimigos, tem-se:

2.3.1. Inimigos simples

São inimigos de formato circular, que andam em grupos e tem um movimento em formato de onda. Oferecem pouco risco, no geral, e dão poucos pontos.

2.3.2. Robôs

São inimigos de formato retangular, que andam de modo independente uns dos outros. A cada poucos segundos, existe a possibilidade de se aproximarem da nave verticalmente. Além disso, possuem um tiro que mira a direção da nave e pode atingi-la. Oferecem um risco mediano, e dão alguns pontos.

2.3.3. Robôs de chão

São inimigos de formato retangular, que andam no chão de modo independente uns dos outros. A cada poucos segundos, atiram um tiro que mira a posição da nave. Ao contrário dos outros, ao chegar no fim da tela, mudam de direção para continuar atirando. São extremamente perigosos e oferecem grande pontuação.

2.3.4. Canhões

Canhões são armas inimigas grudadas nas paredes. A cada segundo, atiram um tiro que mira a nave. Podem ser de grande risco e oferecem uma quantia razoável de pontos.

2.4. Addons e power-ups

Power ups são itens que dão vantagem para a nave. Apenas um power up aparece por vez, e, como todos os elementos do jogo, ~~com exceção dos obstáculos~~, podem ser destruídos por meio de colisões, como inimigos e seus tiros. Apesar de serem resistentes aos tiros da nave, podem ser destruídas pelo tiro especial. Existem dois power ups no jogo, sendo eles:

2.4.1. Chave de fenda

Um objeto circular contendo ferramentas que ajudam a consertar a nave, e dando-lhe um ponto de vida extra.

2.4.2. Gasolina

Um galão de gasolina, que repõe o combustível da nave e aumenta sua velocidade.

3. Organização do Código

Para entender a implementação do jogo, é preciso saber sobre a divisão das bibliotecas, que são essenciais para o funcionamento do programa. Entre elas, se encontram:

- equations.h
- background.h
- obstacles.h
- spaceship.h
- shot.h
- supershot.h
- enemy_shot.h

- enemies.h
- collision_equations.h
- collisions.h
- power_up.h

Para inserir qualquer elemento dentro do jogo, é preciso no mínimo um struct, que dá informações sobre o objeto, e três tipos de funções.

- **init:** funções que configuram o elemento e suas coordenadas/status.
- **update:** funções que atualizam a posição ou o status do elemento no mapa.
- **draw:** funções que desenham o elemento na tela, por meio de um sprite ou imagem passado como argumento.

Visando evitar o uso de repetições, elas serão referidas como funções triviais pelo resto da documentação.

3.1. equations.h

Nesta biblioteca se encontram formatos geométricos e fórmulas matemáticas importantes para todo o jogo. Nela se encontram os structs e funções:

3.1.1. Paralelogram

Struct representando paralelogramas. Possui como elementos os números flutuantes:

- x1: representando o lado esquerdo de paralelogramas
- x2: representando o lado direito de paralelogramas
- any1: representando o topo de paralelogramas
- y2: representando a parte inferior de paralelogramas

3.1.2. Circle

Struct que representa circunferências. Possui como elementos os números:

- x1: representando a coordenada x do centro da circunferência
- y1: representando a coordenada y do centro da circunferência
- radius: representando o raio da circunferência

3.1.3. points_distance()

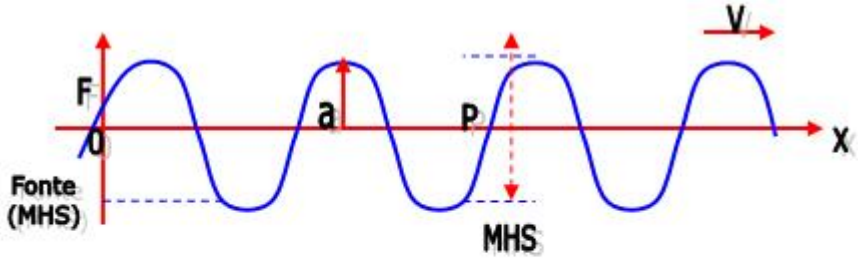
Função que calcula a calcula a distância do centro de duas ciricunferências, por meio da fórmula $\sqrt{(x2 - x1)^2 + \sqrt{(y2 - y1)^2}}$.

3.1.4. angle()

Função que calcula o ângulo entre dois pontos, por meio da função atan2 das coordenadas da distância dos dois pontos menos pi.

3.1.5. wave_equation()

Função que calcula uma coordenada Y em função da posição da tela, usando a fórmula:



O M.H.S. será representado pela função horária:

$y = a \cos(\omega t + \varphi_0)$ ou
$y = a \cos(2\pi f t + \varphi_0)$

3.2. background.h

Nesta biblioteca se encontram funções que vão desenhar elementos do fundo do jogo.

3.2.1. Estrelas

As estrelas são structs circulares de posição aleatória que vão passando pelo mapa ao serem atualizadas. Elas possuem uma variável speed, que define a velocidade em que passam pela tela. Ao chegarem ao fim da tela, retornam ao final. Possuem apenas as funções triviais.

3.3. obstacles.h

Os obstáculos são structs que possuem como atributos as coordenadas de um paralelograma, sua altura e sua largura. Nesta biblioteca encontram-se as três funções triviais para a parede, que recebe coordenada y e altura variável na inicialização, e para o chão, que possuem coordenada y e altura fixas por meio de uma constante. Quando o chão chega ao fim da tela, ele retorna ao final.

3.4. spaceship.h

A nave é um struct retangular com atributos de velocidade, quantidade de vidas, direção(1 se está indo para a direita e 0 se está indo para trás), multiplicador e altura. Ao invés de ter uma função update, possui severas funções de atualização de coordenadas. Além disso, possui uma função spaceship_damaged, que é chamada ao sofrer dano, reduzindo a quantidade de vidas da nave e fazendo uma animação de dano.

3.5. shot.h

Os tiros são uma struct circular que possui como atributo se foi atirado ou não, e a direção em que foi atirado. As funções triviais só são ativadas quando o tiro foi atirado. Para isso, existe a função shoot_out, que atualiza as coordenadas

iniciais para a posição do canhão da nave no momento em que foi atirado, além da direção.

3.6. supershot.h

O tiro especial é um struct circular que, assim como o tiro, tem atributos para saber se foi atirado, sua direção, e desta vez, se está carregando ou não. Além das funções triviais, tem a função load_supershot, que carrega o supertiro, definindo sua direção e aumentando o raio da circunferência até determinado valor. Há também a função reload_supershot, que recarrega a posição do tiro especial para o canhão da nave e retorna ao tamanho original.

3.7. enemy_shot.h

O tiro inimigo é um struct circular que possui como atributos se foi atirado, sua velocidade e o ângulo entre seu ponto de origem e a posição da nave no momento do tiro. Além das funções triviais, possui a função enemy_shoot_out, que atualiza suas coordenadas para o lugar em que foi atirado e calcula o atributo do ângulo. Ao atualizar sua posição, a coordenada y segue é atualizada em seno(ângulo do tiro), fazendo com que ande vá até o ponto y da nave no momento em que foi atirada.

3.8. enemies.h

Nesta biblioteca se encontram as funções triviais dos inimigos.

3.8.1. basic_enemies

Os inimigos simples são structs circulares que possuem como atributos o ponto de origem da onda, que é a partir deste ponto que vão para cima ou para baixo, a velocidade, se foram destruídos e o tipo de inimigo simples. Existem dois tipos de inimigos simples, os líderes, que são os primeiros na fila, e os seguidores, que são todos os outros. O líder é inicializado em uma posição aleatória, já os seguidores, se iniciam atrás do líder ou seguidor seguinte. A partir destas funções,

existem as funções de inicialização de inimigos simples, que recebem um array de inimigos e o seu tamanho, iniciando o primeiro como líder e todos os outros como seguidor, e a função de atualização de fila de inimigos, que atualiza a posição de todos inimigos simples a partir da posição deles na tela por meio da função da onda. Quando o último inimigo chegar ao fim da tela, a fila é reiniciada. Ao desenhar, dependendo do ângulo em que se encontram, possuem diferentes imagens.

3.8.2. Robôs

Os robôs são structs retangulares que possuem como atributos sua velocidade, se foram destruídos, e um struct de tiro inimigo. Em suas funções triviais iniciam o tiro, e na atualização, a cada determinados segundos, se estiver abaixo da nave, sobe, se estiver acima, desce, de modo que persegue a nave. Além disso, simulando a jogada de um dado, caso um número aleatório entre 0 e 6 dê 0 e o tiro já não esteja no mapa, atira. Caso chegue em um ponto anterior ao início da tela, é reiniciado. Possui uma animação que divide um segundo em frames e a cada frame atualiza a posição do sprite.

3.8.3. Robôs de chão

Os robôs de chão são inimigos retangulares com atributos de direção, velocidade, e um array de tiros inimigos. Nas funções triviais, inicia os tiros inimigos em um array, possui coordenada y fixa, e ao ultrapassar a tela possui direção negativa(vai para o lado esquerdo), e se chegar a um ponto anterior ao início da tela, a direção fica positiva(vai para o lado direito). Assim como o robô, divide cada segundo em frações, e nessas frações muda a posição do sprite.

3.8.4. Canhões

Os canhões são inimigos retangulares com atributos de se foram destruídos e um array de tiros. Nas funções triviais, possuem altura fixa, e iniciam os tiros inimigos em um array. A cada segundo, se estiver na tela, e a nave abaixo do canhão, atira. Caso chegue em um ponto anterior ao início da tela, é reiniciado. Ao atirar, emite um efeito sonoro de tiro.

3.9. power_up.h

Os power ups são um struct único, que é circular e retangular ao mesmo tempo. Possuem como atributo o tipo de power up, e se é circular ou retangular. Existem 2 power ups, e ao ser inicializado, podem ser gasolina, ou chave de fenda. Caso seja gasolina, é retangular, e caso seja chave de fenda, é circular. Ambos compartilham a mesma distância como ponto de origem e a mesma largura/comprimento. Caso chegue em um ponto anterior ao início da tela, é reiniciado. Dependendo do tipo de power up, é desenhado determinado sprite.

3.10. collision_equations.h

Nesta biblioteca se encontram as funções que analisam se duas formas geométricas estão se sobrepondo, ou seja, colidindo. Existem três tipos de colisões: entre dois retângulos, entre duas circunferências e entre um retângulo e uma circunferência.

3.10.1. Colisão retangular

Para verificar esta colisão, é preciso analisar se o lado esquerdo ou direito de um triângulo está entre os dois lados de outro triângulo e ao mesmo tempo possui a parte inferior ou superior entre as partes inferiores e superiores de outro retângulo.

3.10.2. Colisão circular

Verifica se a distância entre o centro de duas circunferências é menor que a soma de seus raios usando a função de distância entre dois pontos.

3.10.3. Colisão entre círculos e retângulos

Verifica se o ponto mais à direita ou esquerda da circunferência está entre os lados esquerdo e direito do retângulo, ao mesmo tempo que o ponto mais acima ou abaixo da circunferência se encontra entre as partes superiores e inferiores do retângulo. Isto cuida da maior parte do retângulo, mas ainda é preciso calcular as bordas do retângulo, para isso, basta calcular a distância entre o centro da

circunferência e as quatro pontas do objeto, caso seja menor que o raio, também estão colidindo.

3.11. collisions.h

Para calcular a colisão dos elementos do jogo, basta utilizar as funções da biblioteca de equações de colisões, e caso haja esta colisão, reiniciam a posição de um dos elementos. Na Tabela abaixo, é possível verificar as possíveis colisões entre cada objeto da linha com cada objeto da coluna.

	Obstacles	Spaceship	shot	shupershot	enemy_shot	basic_enemy	robot	floor_robot	cannon	power ups
Obstacles		x	x	x	x	x	x	x	x	x
Spaceship					x	x	x	x	x	x
shot					x	x	x	x	x	x
shupershot					x	x	x	x	x	x
enemy_shot						x	x	x	x	x
basic_enemy						x	x	x	x	x
robot							x	x	x	x
floor_robot								x	x	x
cannon									x	x
power ups										

Em preto, estão as funções já implementadas ou que não seriam viáveis, como por exemplo colisões entre a nave e seus próprios tiros, dois tiros (o motivo será visível na instanciação dos tiros), e power ups com ele mesmo. Utilizando as linhas e colunas, é possível também saber que objeto possui preferência sobre o outro, desta forma, os obstáculos sempre se mantêm vivos, pois não possuem elementos acima/anteriores a eles. A nave, apesar de sofrer dano dos outros elementos, chamando a função de dano, só é destruída durante a colisão com os obstáculos, caso contrário é destruída por ter ficado sem vidas. Já o power up, caso colida com a nave, aumenta o número de vidas, caso seja chave de fenda, ou a velocidade da nave, caso seja gasolina.

Um detalhe importante é que as colisões costumam ser chamadas apenas quando os objetos estão a uma determinada distância da tela, reduzindo o processamento do jogo.

No caso das colisões de obstáculos, em relação ao chão, ela simplesmente irá calcular se o elemento está ultrapassando a altura mínima do chão, que é constante.

4. Implementação

O código principal se inicia com a inclusão das bibliotecas criadas, das bibliotecas padrões allegro, contendo as funções básicas, de imagens, fontes e áudio. Então se tem as configurações das bibliotecas allegro, como configurações, a inclusão das imagens, sprites, áudio, fontes, timer, tela, teclado, fila de eventos, etc.

Em seguida tem-se a inicialização dos elementos das bibliotecas, inicia-se a música de fundo, um temporizador do jogo e um para a animação do carregamento do super tiro.

Cria-se então as funções de balanceamento do jogo, entre elas, o nível de dificuldade do jogo, que irá configurar a velocidade do jogo. Enquanto a nave estiver com vidas, o programa rodará.

As taxas de quadro começam com 30FPS, e vão até 90FPS. Isto ocorre por meio da fórmula ($90 - \frac{60}{score * 10^{-4 + dificuldade} + 1}$). Desta forma, inicia-se com 30 e vai até um total de 90, pois quanto maior a pontuação, menor a subtração.

A quantidade de tiros da nave começam com o um tiro e vai até um número máximo estipulado na variável TOTAL_SHOTS, seguindo a fórmula $TOTAL\ SHOTS\ - \frac{TOTAL\ SHOTS - 1}{1 + 10^{1 - dificuldade}}$.

A velocidade dos inimigos é dada por SPEED - $\frac{MAX\ SPEED - 1}{10^{1 - DIFUCULDADE}}$ ∨ 1.

E a quantidade de inimigos é dada por $MAX\ ENEMIES\ - \frac{0{,}7 * MAX\ ENEMIES}{10^{-DIFICULDADE} + 1}$.

Verifica-se então se o jogo está pausado, se sim, pula para a próxima iteração do loop do jogo.

Começa-se então a implementação das funções triviais do background, com três matrizes de estrelas com velocidades diferentes, que são atualizadas e desenhadas, dando uma impressão de estrelas se movendo.

Depois, há um loop sobre as vidas da nave, que desenham um coração para cada vida no topo da tela, e um texto mostrando a pontuação atual.

Inicia-se então as funções triviais, atualizando e desenhando a parede e o chão.

Após isso, desenha a nave e observa colisões da nave com obstáculos.

Observa-se então as colisões envolvendo o array de tiros disponíveis com a parede, o chão, power ups, inimigos simples, robôs robôs de chão, canhões e os tiros de cada um, para então atualizar sua posição e desenhá-lo caso tenha sido atirado.

Ocorre o mesmo com o super tiro, e, caso ele esteja carregando ou tenha sido atirado, converte o FPS para desenhar a animação de carregamento, apenas atualizando a imagem a cada 1/ 6 de segundo.

Então percorre o array de inimigos básicos e observa colisões envolvendo eles, para atualizá-los e colocá-los na tela.

O mesmo ocorre para os robôs, robôs de chão, canhões e power ups e, por fim, atualiza a tela.

No segundo timer, atualiza-se a posição do tiro e do super tiro.

Nos eventos, começa-se observando a tecla de fechamento de janela, que ao ser pressionada, zera a vida da nave, terminando o jogo.

Ao apertar as teclas de w ou seta para cima, caso a nave esteja abaixo da altura dos corações, chama a função de movimento para cima. Ao apertar as teclas a/setinha esquerda ou d/setinha direita, se estiver após o início da tela e antes do fim da tela, chama a função de movimento para a esquerda e direita, respectivamente. Ao apertar s ou setinha para baixo, caso esteja acima do limite vertical da tela, chama a função de movimento para baixo.

Ao apertar a tecla esc, a variável pausa é ativada, fazendo com que o programa não atualize enquanto a tecla não for apertada novamente.

Ao apertar a tecla de espaço, percorre os tiros disponíveis, e, caso tenha algum que não foi atirado, faz um som de tiro e atira o tiro.

Ao pressionar as teclas de w ou seta para cima, caso a nave esteja abaixo da altura dos corações, chama a função de movimento para cima mais rápido, que aumenta a velocidade da subida e coloca outra animação. Ao pressionar as teclas a/setinha esquerda ou d/setinha direita, se estiver após o início da tela e antes do fim da tela, chama a função de movimento para a esquerda e direita, respectivamente. Ao apertar s ou setinha para baixo, caso esteja acima do limite vertical da tela, chama a função de movimento para baixo mais rápido, que aumenta a velocidade da descida e coloca outra animação.

Ao pressionar a tecla espaço, verifica se o super tiro já não foi atirado ou está carregando, e se não for o caso, recarrega a posição do super tiro e começa a carregá-lo.

Ao soltar a tecla espaço, caso o super tiro esteja carregando, encerra o carregamento, e caso tenha atingido um determinado tamanho, atira.

Ao soltar as teclas de movimentação, retorna o sprite da nave ao normal

Após tudo isso, quando o jogo se encerra, verifica-se a existência de um arquivo de recorde, caso não exista, cria e coloca a nova pontuação nele, colocando na tela a mensagem de novo recorde, e a pontuação total.

Caso exista e a pontuação seja menor, coloca o recorde na tela e abaixo a pontuação obtida.

5. Observações

Devido ao tempo disponível, não foi possível terminar de implementar todas as funcionalidades e animações planejadas, sendo este o motivo de algumas imagens e funções incompletas, entre elas, o modo multiplayer. Caso queira observar uma prévia, é possível apertar a tecla M, controlando a primeira nave pelas

teclas A, W, S e D, e a segunda pelas setas do teclado, todas as colisões funcionam, com exceção das duas naves entre si. Entretanto, há um problema ao movimentar as duas naves ao mesmo tempo, além dos tiros.