
**Nome:** Luiz Felipe de Sousa Faria - 2020027148
**Matéria**: Programação e Desenvolvimento de Software I
**Professor**: Pedro O.S Vaz de Melo

# 1. Instruções do jogo

---

## 1.1. Descrição

**Combat** é um jogo *multiplayer* um contra um, lançado para o console *Atari* 2600 em 1977, na qual cada jogador tem controla um tanque e tem o objetivo de destruir o tanque inimigo, marcando pontos a cada vez que acerta um tiro. Ganha quem conseguir mais pontos até o tempo finalizar. Na versão original para *Atari*, o jogo contava com um modo em que os jogadores controlavam aeronaves ao invés de aviões.

Na seguinte implementação do jogo, os jogadores podem controlar apenas tanques e ganha o primeiro que atingir cinco tiros no tanque inimigo. Os cenários das partidas são gerados de maneira aleatória, criando obstáculos espelhados para os campos de ambos os jogadores. Ao todo, são 8 obstáculos (4 para no campo de cada jogador) . Os resultados das partidas jogadas são armazenados em um arquivo e exibido ao final de cada rodada.

## 1.2. Requisitos

O jogo é compatível com Windows e Linux. Para rodar o jogo, é necessário o compilador GCC (no Windows, o GCC deve ser instalado com o MinGW), da biblioteca Make (já instalada por padrão na maioria das distribuições Linux). Após a instalação desses dois pacotes, o usuário pode acessar a pasta raiz do projeto e executar o comando “make”. Após isso, o jogo será compilado, juntamente com todas as suas dependências e módulos, e automaticamente executado, estando pronto para jogar

### 1.3. Telas

O jogo possui duas telas básicas. A tela do campo de batalha (imagem 1) e a tela do resultado final (imagem 2). Na tela de combate, a pontuação é exibida na parte superior. O restante é preenchido pelos tanques e pelos obstáculos gerados. Após um dos jogadores atingir 5 pontos, o jogo finaliza, levando os usuários para a tela de resultado, que mostra a pontuação da partida atual e o histórico de todas as partidas jogadas pelos jogadores.

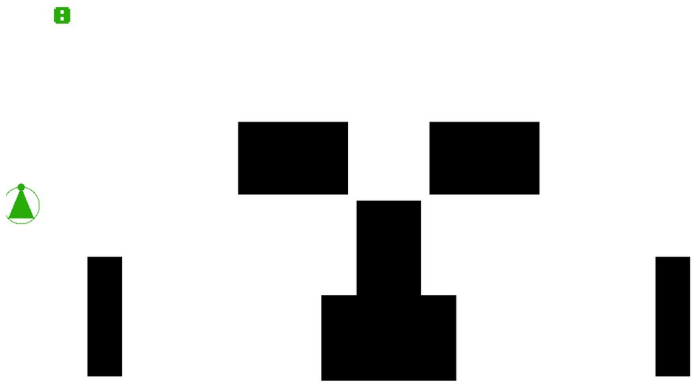


*Imagem 1 - Tela de combate*


*Imagem 2 - Tela de Resultado*

### 1.4. Controles

Jogador 1:

- Movimentação: W (para cima), S (para baixo), A (rotação para a esquerda) e D (rotação para a direita);
- Tiro: Q;

Jogador 2:

- Movimentação: Seta para cima (para cima), Seta para baixo (para baixo), Seta para a esquerda (rotação para a esquerda) e Seta para direita (rotação para a direita);
- Tiro: Enter;

## 2. Estrutura do código

---

### 2.1. *Structs*

O código apresenta ao todo 3 *structs*. São elas: *Tank, Point* e *Obstacle*

- *Point:* trata-se da estrutura básica que compõe as outras. Ela guarda os pontos X e Y de um determinado objeto no plano cartesiano. Ambos os pontos são descritos como números flutuantes.

- *Tank:* guarda informações a respeito da posição do tanque, bem como sua velocidade e velocidade angular em um determinado momento; se o tanque realizou um disparo e a posição desse disparo em cada instante; a cor do tanque; e a quantidade de pontos que ele marcou na rodada.

- *Obstacle:* guarda a posição referente a um determinado objeto presente na tela. Essa posição é do tipo *Point* e descreve as coordenadas dos pontos referentes a borda superior esquerda e a borda inferior direita.

### 2.2. Estrutura de arquivos

O presente código possui uma estrutura de arquivos divididas em *Modules, Screens, Core*, todos divididos internamente em *Includes,* que guarda os arquivos *headers*, e *Src (source),* que guarda os arquivos ".c". Além da pasta *Assets*, que guarda as fontes e sons utilizados no desenvolvimento do projeto, e a pasta *Libs*, que guarda a pasta contendo o Allegro.

- *Modules:* representa todos os arquivos que compõe os elementos do jogo. Dentro dele está a implementação dos tanques, das colisões, dos funções de desenho e dos obstáculos.
- *Screens:* representa as funções responsáveis por gerar as telas da aplicação, inserindo todos os elementos implementados nos *Modules* nas suas respectivas telas.
- *Core*: representa o principal arquivo da aplicação, bem como as configurações básicas do jogo, como velocidade do tanque, do tiro, tamanho da tela, etc. É responsável também por carregar as fontes e sons necessários e por gerar a fila de eventos da biblioteca Allegro, que permite a execução do jogo e a captura de eventos do usuário, como do teclado para a movimentação dos tanques.

## 3. Código por arquivo

---

### 3.1. *Core*

#### 3.1.1. Arquivo /combat.c

Responsável pelo carregamento e criação inicial das variáveis de fontes, sons, fila de eventos, e por executar as funções que criam a tela a ser apresentada pelo usuário. Inicialmente, são criadas as estruturas básicas para a execução do jogo, que são os tanques e obstáculos.. O número de obstáculos totais totais é multiplicado por 2, já que são espelhados verticalmente na tela.

Após isso, são declaradas as variáveis *playing* e *winner*. Enquanto *playing* for igual a 1, a fila de eventos observará os eventos de *timer* e de *inputs* do usuário, para desenhar os elementos na tela e capturar a movimentação desejada para os tanques, respectivamente. É feito um *switch* para saber qual a tela atual. Esse *switch* observa a variável *screen*, que possui valores referentes ao *enum SCREEN*. O evento de timer

também verifica a cada momento se há um vencedor, executando a função *gameWinner* e substituindo o valor da variável *winner*. Se tiver um ganhador, a tela de *score* é exibida.

Após 5 segundos da tela de *score* ser exibida, o jogo é reiniciado, com novos valores para os obstáculos.

3.1.1. Arquivo /config.h

Arquivo que guarda constantes responsáveis pelas definições de aspectos padrões do projeto, tais como tamanho e velocidade do tanque e do seu tiro, tamanho da tela, número de obstáculos, FPS e a definição da *struct Point*.

## 3.2. *Modules*

*3.2.1.* Arquivo /collision.c

- *shotOutOfScreen()*: função responsável por receber um ponteiro de uma *struct Tank* e verificar, para cada, se o tiro dele passou dos limites da tela. Se sim, executa a função *resetShotPosition()*, que volta o tiro para suas coordenadas iniciais.

- *collisionTankScreen():* recebe um ponteiro de uma *struct Tank* e verifica se o mesmo passou dos limites da tela. Caso verdadeiro, muda os pontos X ou Y do tanque para o raio do tanque (caso tente atravessar parte superior e esquerda) ou para o tamanho da largura ou altura menos o raio do tanque (para direita e inferior, respectivamente).

- *collisionBetweenTanks()*: recebe como parâmetro o ponteiro referente a cada tanque e verifica se eles colidiram. Essa verificação pega a distancia entre as coordenadas do centro de cada tanque e verifica se ela é menor que a distância entre os raios. Se sim, quer dizer que colidiram. Assim, cria-se a variável *distanceToMove*, que seria a diferença entre as somas dos raios e a distância entre os centros. Se o tanque estiver com a velocidade negativa (quer dizer que esta movimentando para frente), então soma-se os pontos X e Y pela multiplicação de *distanceToMove* e os vetores dos eixos X e Y do tanque. Caso a velocidade for positiva (tanque movimentando para trás), faz-se a subtração desses valores.

- *collisionTankShot()*: recebe o ponteiro dos dois tanques e o um ponteiro referente a um ALLEGRO_SAMPLE, que contem o som disponível em um arquivo. Calcula a colisão entre o tiro e um dos tanques, de forma semelhante a função *collisionBetweenTanks().* Se houve a colisão, o tiro retorna para as coordenadas iniciais do tanque que realizou essa ação. Ao acertar o tiro, também executado o arquivo de som de uma explosão em 8-bits.

- collisionTankObstacle*()*: recebe o ponteiro de uma *struct Tank* e um obstáculo. Através das coordenadas do obstáculo, obtêm-se o centro, altura e largura dele. Com isso, calcula-se a diferença entre o valor do eixo X do obstáculo e o X do tanque, armazenando na variavel *distX*. O mesmo se faz para Y. Calcula-se também a distância entre essas duas coordenadas também, armazenando na variável *diagonalDistance*. Verifica se *distX* é menor que a largura do obstáculo dividido por 2. Se for, muda a variável *distanceToMove* para altura / 2 + raio – *distY*. Se *distY* for menor que altura / 2, *distanceToMove* assume largura / 2 + raio – *distX*. Se *diagonalDistance* for menor que o raio do tanque (caso colida na borda do obstáculo), *distanceToMove* assume raio –

*diagonalDistance*. Após, é feita a mesma lógica da função collisionBetweenTanks para reajuste da posição do tanque.

- collisionShotObstacle*()*: recebe o ponteiro de uma *struct Tank,* de um obstáculo e de um som. Utiliza a mesma lógica da função collisionTankObstacle para verificar a colisão entre uma circunferência e um retângulo. Caso colida, executa *resetShotPosition* e o som de colisão com um obstáculo.

*3.2.2.* Arquivo /drawer.c

- *drawScenario():* responsável por mudar o fundo da tela para branco.
- *drawTank():* recebe uma *struct Tank* como parâmetro e utiliza funções do Allegro para desenhar o campo de força do tanque, a partir das coordenadas do centro e do raio, e o próprio tanque, a partir dos pontos A, B e C (vértices de um triângulo).
- *drawShot():* recebe uma *struct Tank* como parâmetro e verifica se o tanque atirou. Se sim, ele pega as coordenadas X e Y do tiro. Se não, ele pega as coordenadas do ponto A do tanque.
- *drawPoints():* recebe uma cor, um int representado o *score* do jogador, uma fonte, e um float X e Y, representando as coordenadas onde o texto será escrito. Desenha os pontos de cada tanque na parte superior da tela. A cor da pontuação é a mesma do tanque que ela se refere.
- *drawScoreScreen():* recebe a *struct* dos dois tanques, um vetor representando o score de cada jogador e um ponteiro de uma fonte. Desenha a pontuação do jogo atual e o histórico de partidas.
- *drawObstacle():* recebe uma *struct Obstacle* como parâmetro e desenha um retângulo na cor preta baseado nas coordenadas informadas pela *struct*.

*3.2.2.* Arquivo /obstacle.c

- *createObstacle()*: recebe um vetor da *struct Obstacle* e cria blocos para cada posição desse vetor. A posição dos blocos é gerada de forma randômica através da função interna *generateRandomPositions()*. Possui largura e altura mínimas, além de uma margem de segurança do tamanho do diâmetro do tanque mais 10. As formas são geradas para metade da tela e depois espelhadas para a outra metade através de função *createMirroredObstacles()*.

*3.2.3.* Arquivo /tank.c

- *createTank()*: recebe um ponteiro de uma *struct Tank* e as coordenadas inicias do tanque. Atribui os valores iniciais para os atributos do tanque.
- *rotateTank()*: função que recebe um ponteiro para a *struct Tank* e aplica uma rotação das coordenadas X e Y de cada vértice do triângulo que compõe o tanque, permitindo que a parte frontal do tanque aponte para a direção que o tanque esta locomovendo, além de definir os valores dos vetores X horizontal e Y vertical do tanque.
- *updateTank()*: recebe um ponteiro para a *struct Tank* e atualiza os valores das coordenadas do centro, fazendo a multiplicação da velocidade atual do tanque pelo valor do vetor X ou Y. Executa primeiramente a função *rotateTank()* para atualizar o valor dos vetores.
- *updateShot*(*):* recebe um ponteiro para a *struct Tank* e atualiza a posição do tiro do tanque, caso ele tenha sido disparado. Caso não, aplica a rotação nas coordenadas do

centro do tiro. A atualização ocorre de forma semelhante a função *updateTank()*. No entanto, o valor dos vetores X e Y se mantém fixos aos valores dos vetores do tanque no momento em que o tiro foi disparado. A velocidade do tiro é constante.
- *tankShot()*: função que representa a ação de disparo do tanque. Recebe um ponteiro para a *struct Tank* e atualiza os vetores do tiro com os valores dos vetores do tanque no momento do disparo, e as coordenadas do centro do tiro, para que sejam referentes ao plano da tela, e não ao centro do tanque.

## 3.3. *Screens*

### 3.3.1. Arquivo /game.c

- *restartGame()*: recebe um ponteiro dos dois tanques, o ponteiro da variável *winner* e *screen*, além do vetor contendo os obstáculos exibidos na última partida. Utiliza a função *createTank()* para reiniciar a posição e atributos do tanque, e a função *createObstacles()* para substituir os obstáculos antigos por novos. Atribui 0 a variável *winner* e *GAME_SCREEN* para *screen*.

- gameWinner(): recebe os dois tanques, o ponteiro da variável *winner* e *screen.* Verifica se algum jogador atingiu 5 pontos. Caso sim, atribui para *winner* o valor 1 caso tenha sido o primeiro jogador o vencedor ou 2 caso tenha sido o segundo. Muda a tela atual para a *SCORE_SCREEN*.

- *renderGame()*: executa as funções descritas nos módulos *tank.c, obstacle.c* e *collision.c*. Usa a função do Allegro *al_flip_display()* para sempre reestruturar a tela com as novas informações de cara objeto.

### 3.3.2. Arquivo /score.c

- *renderScore()*: responsável por executar a função *drawScoreScreen()*, mostrando o resultado final da partida e o histórico de jogos. Caso não exista um arquivo de histórico, ele cria um novo e atribui 1 ponto para o jogador vencedor. Caso já exista, atribui 1 ao valor preexistente do jogador vencedor. Cria um *timeout* de 5 segundos, que exibe a tela por esse período. Ao final, a função *restartGame()* é executada.