Universidade Federal de Minas Gerais
Gustavo Freitas Cunha

Trabalho Prático em Programação e Desenvolimento de Software
Professor Pedro O. S. Vaz de Melo

COMBAT

ATARI

COMBAT

SOBRE O JOGO

Nesta reinvenção do clássico jogo eletrônico da década de 1970, dois jogadores poderão interagir simultaneamente. Cada jogador possui o controle de um **tanque de força**, que pode se mover livremente pela galáxia. Mas atenção: no meio do cenário há um **buraco negro**. Caso o **campo de força do tanque** (círculo em que o tanque está inscrito) ultrapasse o **campo gravitacional** do buraco (delimitado por um retângulo colorido - **FIGURA 2**), ele automaticamente perderá a partida.

DICA: Fique atento aos movimentos do seu adversário e tente encurralá-lo contra o buraco, assim, você vencerá a partida, sem precisar acertar tiros nele!

Cada tanque é equipado com um **atirador**, que dispara um tiro por vez. A cada vez que um tiro do jogador acerta seu advesário, **1 ponto** é creditado. O **placar** em tempo real é exibido nos cantos superiores da tela e a cor dos pontos corresponde à cor do tanque respectivo **(FIGURA 2)**.

Os tanques começam o jogo em posições opostas na galáxia. O **Tanque 1** inicia o jogo disposto na região à esquerda do buraco negro e o **Tanque 2**, à direita dele. O piloto que atingir a marca de **5 pontos** primeiro, vence a partida!

No final de cada partida, o placar da rodada é exibido, juntamente com o histórico de vitórias do Tanque 1 e do Tanque 2 **(FIGURA 3)**.

Ao iniciar a execução, os jogadores irão se deparar com uma tela inicial. Quando estiverem prontos, é só apertar **ENTER** para iniciar o combate!

FIGURA 1: TELA INICIAL DO JOGO

COMBAT

LAYOUT

Após o início, pela tecla **ENTER**, o jogo estará valendo e será exibida a tela de jogo:

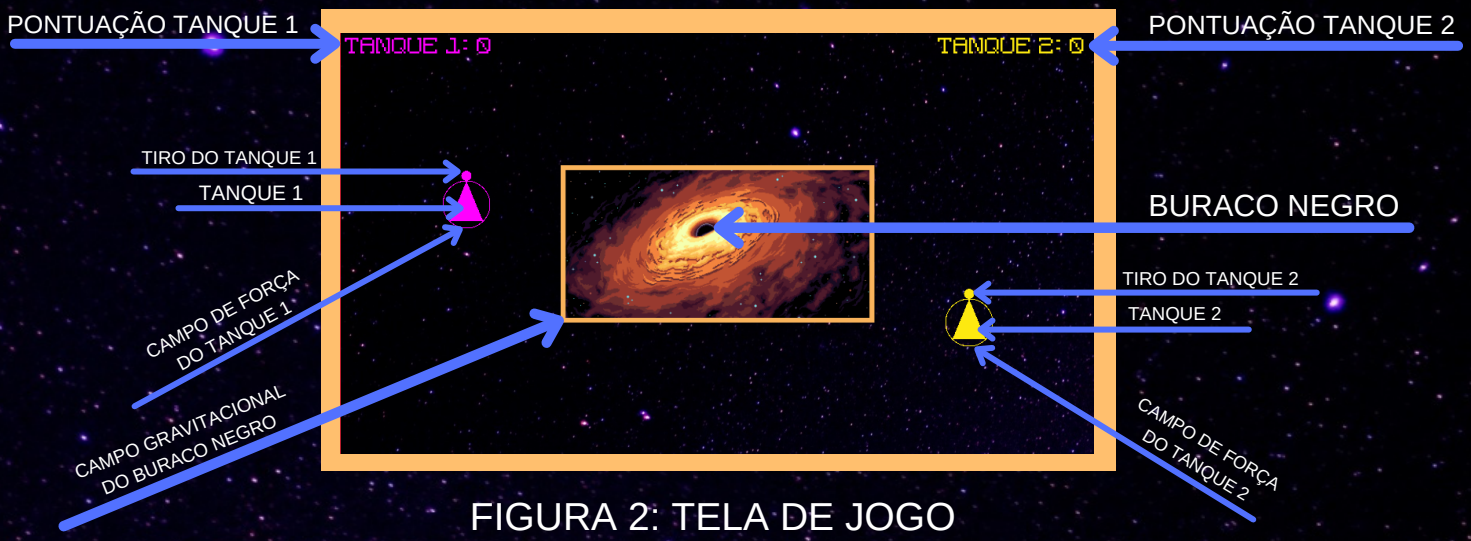

FIGURA 2: TELA DE JOGO

No fim da partida, isto é, quando um dos jogadores atingir cinco pontos ou se algum tanque colidir com o campo de força do buraco negro, a tela final mostrará informações como:

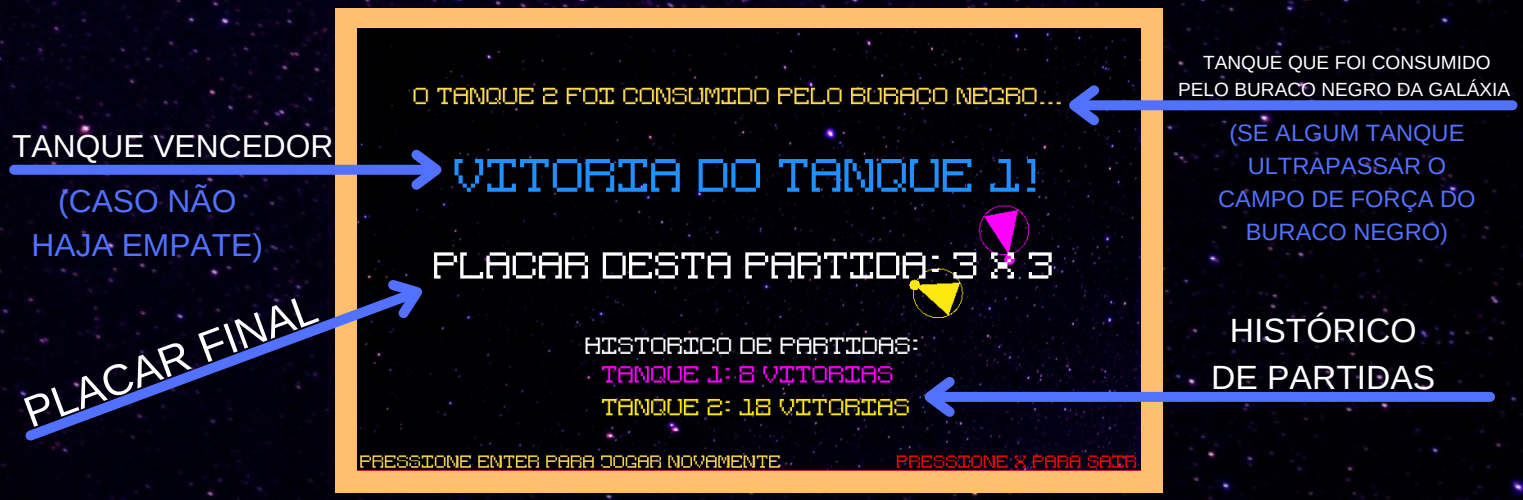

FIGURA 3: TELA FINAL DO JOGO

Caso os jogadores queiram jogar novamente, é só teclar **ENTER**. Caso queiram sair, é só teclar **X** ou clicar no botão **FECHAR.** Os players também podem clicar em **FECHAR** a qualquer momento durante o jogo, para encerrá-lo e sair:

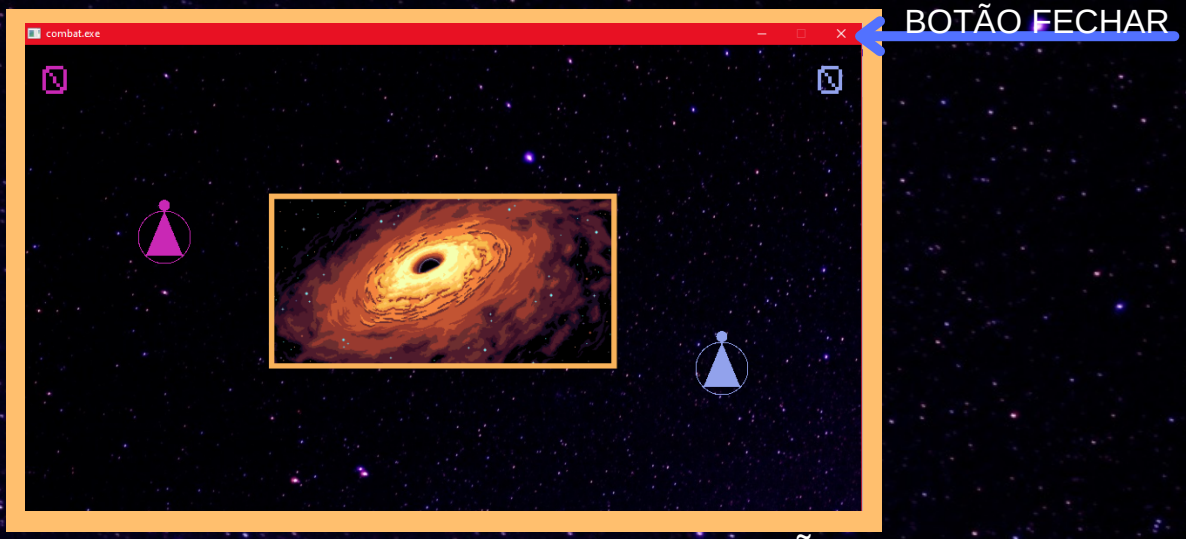

FIGURA 4: TELA DE JOGO COM A OPÇÃO "FECHAR"

Caso o jogo seja encerrado dessa forma, a tela final **(FIGURA 3)** será exibida com a pontução final e com o histórico de partidas. Então, é só teclar **X** ou clicar em **FECHAR** novamente para finalizar a execução do game.

COMBAT

COMO JOGAR

Tanque 1:

- Tecla W: o tanque se move para frente;
- Tecla S: o tanque se move para trás;
- Tecla D: gira o tanque em sentido horário;
- Tecla A: gira o tanque em sentido anti-horário;
- Tecla Q: atira imediatamente.

Tanque 2:

- Tecla ↑: o tanque se move para frente;
- Tecla ↓: o tanque se move para trás;
- Tecla →: gira o tanque em sentido horário;
- Tecla ←: gira o tanque em sentido anti-horário;
- Tecla ENTER: atira imediatamente.

FIGURA 5: ILUSTRAÇÃO DAS TECLAS UTILIZADAS NO JOGO EM UM TECLADO COMUM

COMBAT CODIFICACAO DO JOGO EM C

Início do código

Nas primeiras linhas, são incluídas as bibliotecas necessárias para implementação do jogo e declaradas algumas variáveis constantes, para facilitar na manipulação de alguns detalhes genéricos, como largura e altura da tela de jogo; dimensões do campo de força do buraco negro; raio, angulação e velocidade dos tanques e tiros, dentre outros.

Estruturas

O manuseio das entidades do jogo é simplificado, pois foram criadas structs, que contém, internamente, informações específicas de cada tipo de dado, através do **typedef**. Structs criadas: **Ponto**, **Tiro**, **Tanque**, **Obstaculo**.

Funções

A maior parte da codificação do jogo é feita por funções, para que o código fique mais legível, limpo e sua manutenção seja otimizada: "dividir para conquistar!".

- **Função desenhaCenario**

Esta função coloca na tela as camadas mais básicas do que se vê na tela de jogo **(FIGURA 2)**: a **imagem de fundo** e a **pontuação** de cada tanque nos cantos superiores opostos. Para isso, ela recebe como parâmetros os tanques e as fontes utilizadas, declarados e carregados mais adiante, nas rotinas de inicialização, na função main.

- **Função distanciaPontos**

Esta função utiliza a fórmula de **distância Euclidiana** para calcular e retornar a distância entre dois pontos, que são recebidos como parâmetros.

COMBAT CODIFICACAO DO JOGO EM C

- **Funções de verificação de colisões:**
  colisaoTanques; colisaotirotanque; colisaotirotela; colisaoTanqueObstaculo; colisaoTiroObstaculo; colisaoTelaTanquesY; colisaoTelaTanquesX.

As funções acima enumeradas possuem nome intuitivo, isto é, conseguimos saber sobre quais entidades verificam a colisão. Todas utilizam lógica semelhante: como os parâmetros recebidos não são passados por referência, mas sim **por valor**, há uma **simulação**. Antes da verificação da colisão, a posição do objeto é incrementada, para verificar se num movimento haveria ou não colisão, mas esse incremento é feito apenas internamente nas funções. Após a simulação, verifica-se, pela **distância entre os dois objetos**, se há ou não colisão. As funções **colisãoTelaTanquesY**, **colisãoTelaTanquesX** e **colisaotirotela** se diferenciam porque verificam a colisão do Tiro/Tanque com os limites da tela de jogo e impede que os objetos a ultrapasse.

- **Funções de verificação de distâncias:**
  distFiguras e distFiguras2.

Estas duas funções atuam de forma semelhante e estão ligadas às funções anteriormente listadas (colisões): caso a **distância** entre o Tanque (função **distFiguras**) ou Tiro (função **distFiguras2**) e o Obstaculo (campo gravitacional do buraco negro) seja **menor ou igual ao raio** do Tanque ou Tiro, haverá colisão.

- **Funções de inicialização e representação dos Tanques:**
  initTanque1; initTanque2 e desenhaTanque.

Esses procedimentos são responsáveis por determinar aspectos iniciais dos Tanques, como cor na partida (escolhida aleatoriamente, através da função **rand()**), posição inicial, velocidades iniciais (zeradas), pontuações iniciais (zeradas) e aspectos geométricos para alinhamento e simetria do triângulo e do campo de força do tanque. Ainda, a última função desenha os tanques, conforme posições determinadas nos dois outros procedimentos.

- **Funções de atualização dos Tanques e Tiros:**
  Rotate, rotacionaTanque e atualizaTanque.

Depois da função main, essas são as funções mais importantes do programa. Utilizam quase todas as funções anteriormente criadas, para implementar o movimento e rotação dos Tanques e Tiros. Os procedimentos **Rotate** e **rotacionaTanque** se encarregam da parte angular do movimento dos tanques, utilizando as variáveis **vel_angular** e **angulo**, definidas na struct Tanque. A função Rotate utiliza trigonometria avançada, através de seno e cosseno e altera as posições dos pontos A, B e C do triângulo interno do tanque (também definidos na struct Tanque), quando chamada pela função rotacionaTanque, que também incrementa a vel_angular e faz atribuições à **x_comp** e **y_comp**, variáveis que serão multiplicadas pela velocidade do Tanque, quando este se movimentar.

A função **atualizaTanque** utiliza as funções anteriores para incrementar ou decrementar as coordenadas do centro dos Tanques e dos Tiros, quando solicitado e caso não haja colisão prevista pelos procedimentos responsáveis chamados (funções de verificação de colisões). Essa função também incrementa a variável **pontos**, definida na struct Tanque, quando um Tanque acerta um tiro em seu adversário, além de voltar o tiro para o Tanque de origem, quando ele colide com algum obstáculo ou com a tela.

COMBAT CODIFICACAO DO JOGO EM C

Main

- ## Rotinas de inicialização

Neste trecho do código, são realizados alguns **procedimentos padrões da biblioteca Allegro**, como inicialização do Allegro, de módulos de primitivas, módulos para carregamento de imagens e fontes, criação do temporizador e da tela, instalação do teclado e criação e registro de **eventos do tipo ALLEGRO_EVENT.**

- ## Algumas declarações

Ainda nas rotinas de inicialização, são feitas declarações importantes de variáveis para carregamento de imagens e também declarado o **obstaculo**, variável do tipo **Obstaculo**, que contém as informações sobre o buraco negro. Em seguida, o **temporizador** é iniciado e é declarada a variável **playing**, que auxiliará no **ciclo do jogo**.

- ## While da Tela de Início

Para exibição da tela de início, é implementado um laço while, com condição **(playing==1)**. Dentro dele, temos a implementação da imagem de fundo e a impressão de alguns inscritos na tela, conforme mostrado na **FIGURA 1**. Há também a criação de um evento, para atualizar a tela (caso evento seja do tipo **ALLEGRO_EVENT_TIMER)**; para fechar a tela (caso seja de tipo **ALLEGRO_EVENT_DISPLAY_CLOSE)**; ou para iniciar o jogo caso seja do tipo **ALLEGRO_EVENT_KEY_DOWN**. Neste último, caso a tecla **ENTER** seja pressionada, a variável **playing recebe o valor 2**, que finaliza este laço e inicia o próximo.

- ## Outras declarações importantes

O trecho que segue, é para **declaração e inicialização (utilizando initTanque) dos dois tanques** (tanque1 e tanque2), que irão jogar e para declaração das variáveis **vencedor** (que recebe, incialmente, zero) e **consumido**, que também recebe o valor zero, de início.

- ## While da Tela de Jogo

Este laço é para **execução do jogo propriamente dito**, e tem como condição (playing==2). Nele, outro evento é criado, de forma análoga ao while da tela de início, e as funções até aqui descritas, são utilizadas em eventos específicos. O movimento e rotação dos tanques são feitos conforme o jogador pressiona (**ALLEGRO_EVENT_KEY_DOWN**) e solta (**ALLEGRO_EVENT_KEY_UP**) as teclas. É feita também, a cada segundo, através de evento do tipo **ALLEGRO_EVENT_TIMER**, a verificação, com auxílio das funções respectivas, se algum dos jogadores chegou à 5 pontos ou se houve colisão entre tanques e o campo gravitacional do buraco negro. Caso haja retorno 1 para a primeira opção, a variável vencedor recebe 1 (caso o tanque1 tenha marcado 5 pontos), ou 2 (caso seja o tanque2 que marcou a quíntupla). Caso a segunda opção retorne 1, isto é, caso algum tanque tenha colidido com o campo gravitacional do buraco negro, ele será o perdedor e a variável vencedor recebe o número de seu adversário. Além disso, variável **playing recebe o valor zero** novamente, e este laço é quebrado.

While de Fim de Jogo e While Macro

Esse laço final tem como condição **(!(playing))** e outro evento é criado. Novamente, temos a implementação das camadas de fundo (com os tanques nas posições em que finalizaram o jogo), a declaração de algumas strings, para impressão de textos na tela **(FIGURA 3)**. Dentre esses, temos a impressão do **jogador vitorioso**, através da variável **vencedor**, a impressão do tanque que perdeu por ser consumido pelo buraco negro (caso haja, através da variável **consumido**), a impressão do **placar da partida** (registrado pelas variáveis **pontos** dos dois tanques) e a impressão do **histórico de partidas**, que é registrado através das variáveis **h1** e **h2** (declaradas antes do início desse laço, quando são lidas e incrementadas conforme o jogador que venceu), que são manipuladas e colocadas em um **arquivo.txt** (através das funções **fscanf** e **fprintf**), presente no mesmo diretório do arquivo executável do jogo. A tela de fim de jogo **(FIGURA 3)** fica aguardando que um dos usuários pressione **ENTER**, (o jogo irá recomeçar e a tela de inicio **(FIGURA 1)** será exibida), ou que pressione X ou clique em **FECHAR** (o jogo será finalizado e fechado). A finalização do jogo é executada atribuindo o **valor 11 para playing**, para quebrar esse laço, e **atribuindo zero à variável macro**, que é o parâmetro de um **laço universal do jogo**, que começa no início da função main *(linha 453)* e termina em seu final *(linha 919)*.

Últimas linhas

Nas últimas linhas, temos as rotinas de fim de jogo, para limpeza da memória, o fim da função main e o fim do programa.

COMBAT AUTORIA E AGRADECIMENTOS

Gustavo Freitas Cunha

Matrícula: 2020085498

Professor Pedro O. S. Vaz de Melo

Programação e Desenvolvimento de Software I

DCC, UFMG, 2020.2