

**Among Invaders**

Nome: Ana Flávia de Matos Souza

Disciplina: Programação e Desenvolvimento de Software I

Professor: Pedro O. S. Vaz de Melo

Introdução:

**Among Invaders** é um jogo desenvolvido para a disciplina de Programação e Desenvolvimento de Software I e foi criado utilizando a linguagem C e a biblioteca Allegro 5. Ele é um jogo de arcade de tiro e foi baseado nos jogos já existentes Space Invaders e Among Us.

Você joga como um alien impostor que tinha conseguido se infiltrar entre os membros da tripulação, porém os tripulantes conseguiram descobrir que você é o assassino entre eles. Com isso, eles estão tentando te perseguir para te expulsar da nave antes que seja tarde demais. O seu objetivo é eliminar todos os tripulantes antes que eles consigam te alcançar ou que a nave chegue em seu destino.

Controles:



Move o impostor para a esquerda



Move o impostor para a direita



Atira a faca

Implementação:

O código inicia com os includes das bibliotecas usadas e a declaração de algumas constantes e variáveis usadas ao longo de todo o programa. Logo em seguida começa a definição de funções e estruturas, que serão explicadas uma por uma.

**Estruturas:** Foram declaradas três estruturas – Impostor, Crew e Faca – muito semelhantes entre si. Abaixo estão os detalhes sobre cada uma dessas:

* **Impostor:** Essa é a estrutura utilizada para criar o personagem jogável – o impostor. Ela possui os seguintes parâmetros:
  + altura define a altura de cada sprite dentro da folha,
  + largura define a largura de cada sprite dentro da folha,
  + colunas\_folha define quantos sprites cada linha da folha tem,
  + coluna\_atual define em qual coluna da folha a animação está,
  + linhas\_folha define quantos sprites cada coluna da folha tem,
  + linha\_atual define em qual linha da folha a animação está,
  + regiao\_x\_folha define a posição x da folha que será mostrada,
  + regiao\_y\_folha define a posição y da folha que será mostrada,
  + pos\_x define a posição x da tela em que será mostrado o sprite,
  + pos\_y define a posição y da tela em que será mostrado o sprite,
  + vel\_x define a velocidade horizontal do sprite na tela,
  + vel\_y define a velocidade vertical do sprite na tela.
* **Crew:** Essa é a estrutura utilizada para criar os alvos do impostor. Ela possui exatamente os mesmos parâmetros da estrutura acima.
* **Faca:** Essa é a estrutura utilizada para criar a faca que será atirada pelo impostor. Ela possui os seguintes parâmetros:
  + pos\_x define a posição x da tela em que será mostrado o sprite,
  + pos\_y define a posição y da tela em que será mostrado o sprite,
  + altura define a altura do sprite,
  + largura define a largura do sprite.

**Funções:** Foram declaradas diversas funções para serem utilizadas ao longo do programa. Isso ajudou bastante na legibilidade e na debugação do código. Abaixo estão os detalhes sobre cada uma delas:

* int inicializar() – Essa função foi criada para juntar todas as rotinas de inicialização em um só lugar. Isso foi feito para limpar a main, facilitando achar as partes do código necessárias. Estão inclusas aqui as rotinas de inicialização dos módulos do Allegro, do timer, da fila de eventos, da fonte utilizada, dos bitmaps, dos arquivos de áudio, do mouse e do teclado. Retorna 1 se tudo ocorrer corretamente e 0 caso contrário.
* void initImpostor(Impostor \*imp) – Essa função inicializa o Impostor criado com a estrutura definida anteriormente. Ela recebe um Impostor como argumento e atribui valores para cada um de seus parâmetros.
* void initCrew(int i, int j, Crew \*crewmate) – Essa função inicializa um membro da tripulação criado com a estrutura definida anteriormente. A única diferença dela para a função anterior são os valores atribuídos e o fato de ela receber dois novos argumentos: i e j. Como os tripulantes são criados dentro de uma matriz esses dois argumentos servem para alterar a posição em que os tripulantes são exibidos na tela, evitando que eles sejam criados um em cima do outro, e também para mudar a região da folha que origina cada sprite, o que possibilita que eles sejam criados em cores diferentes.
* void initFaca(Faca \*knife, Impostor imp) – Outra função muito semelhante às duas anteriores. Recebe um objeto Faca e um Impostor como argumentos. Preenche a Faca com valores e utiliza o Impostor para posicionar a Faca escondida atrás do Impostor, evitando que ela seja vista antes de ser atirada.
* void desenhaImpostor(Impostor \*imp) – Essa função desenha o impostor na tela. Recebe um objeto do tipo Impostor já preenchido como argumento e utiliza os parâmetros dele para desenhar o bitmap com base na velocidade. Se o impostor estiver se movimentando para a direita ele será desenhado virado para esse lado, se ele estiver indo para a esquerda ele será desenhado para esse lado e se ele estiver parado ele será desenhado por padrão virado para a direita.
* void desenhaCrew(Crew \*crewmate) – Essa função desenha um tripulante na tela. A única diferença dela para a função anterior será que ela receberá um objeto do tipo Crew ao invés de Impostor e que não ocorrerá um caso em que os tripulantes estão parados.
* void desenhaFaca(Faca knife) – Essa função desenha a faca na tela. Funciona de forma semelhante às duas últimas, porém muito mais simples. Simplesmente desenha a faca na posição já armazenada nos parâmetros da Faca.
* int colisaoCrewSolo(Crew crewmate) – Essa função recebe um objeto Crew como argumento e verifica se ele encostou com o chão. Retorna 1 se houve colisão e 0 caso contrário.
* int colisaoCrewImpostor(Crew crewmate, Impostor imp) – Essa função recebe um objeto Crew e um objeto Impostor como argumentos e verifica se eles colidiram. Retorna 1 se houve colisão e 0 caso contrário.
* int colisaoCrewFaca(Crew crewmate, Faca knife) – Essa função recebe um objeto Crew e um objeto Faca como argumentos e verifica se eles colidiram. Retorna 1 se houve colisão e 0 caso contrário.
* void deleteCrew(Crew \*crewmate) – Essa função deleta um membro da tripulação. Para isso ela recebe um objeto Crew como argumento e altera os seus parâmetros para 0.
* int novoRecorde(int score, int \*recorde) – Essa função verifica se a pontuação do jogador bateu o recorde anterior e salva o novo recorde se esse for o caso. Para isso ela recebe como parâmetros um inteiro com a pontuação atingida pelo jogador e um inteiro para armazenar o recorde anterior. A função abre o arquivo onde o recorde se encontra, armazena ele na variável fornecida e o compara com a pontuação do jogador. Se a pontuação do jogador for maior que o recorde essa pontuação é armazenada no arquivo e a função retorna 1. Caso contrário a função retorna 0.

**Main:** Após a criação das estrutras e a declaração das funções temos a main do programa. Aqui é onde as funções e as estruturas são utilizadas para criar o jogo em si. Na ordem de aparição, o funcionamento está descrito abaixo:

* Primeiramente são declaradas quatro variáveis que serão utilizadas ao longo dos códigos para iterar os loops;
* É chamada a função inicializar() e caso ocorra algo de errado o programa retornará -1 e será encerrado;
* São registrados na fila os eventos de tela, tempo, teclado e mouse;
* São declaradas duas variáveis: frames\_sprite, que define quantos frames devem se passar antes de trocar de sprite, e cont\_frames, que conta quantos frames já passaram;
* São declarados e criados o impostor, a tripulação e a faca. O impostor e a faca são criados utilizando as estruturas previamente definidas (Impostor e Faca respectivamente) e preenchidas com as funções initImpostor e initFaca respectivamente. Já a tripulação é criada com uma matriz que utiliza a estrutura Crew e preenchida por dois loops que iteram a matriz e preenchem um tripulante por vez com a função initCrew;
* Logo em seguinda é inicializado o temporizador;
* São declaradas mais três variáveis. Essas variáveis são waiting, playing e mostraRecorde e elas são utilizadas para controlar os três loops principais do programa, descritos logo abaixo;
* O primeiro loop é o waiting. Ele é executado assim que o programa é iniciado e a sua função é esperar o jogador apertar uma tecla ou clicar na tela para só então começar o jogo. Isso serve para evitar que o jogo se inicie sem que o usuário esteja pronto. Sobre o funcionamento dele:
  + É criada uma variável de evento e logo em seguida o programa começa a esperar por um evento;
  + Caso esse evento seja do tipo timer, ou seja, caso haja passada de tempo, será desenhado na tela o impostor, a tripulação e a faca em suas posições iniciais, mas todos parados;
  + Além disso é desenhado um texto instruindo o jogador a apertar uma tecla para iniciar a execução do jogo. Esse texto só é exibido em segundos alternados;
  + Caso o evento seja uma tecla ou um botão do mouse sendo pressionados, a variável playing passa a ser 1 e a variável waiting passa a ser 0. Isso faz com que o programa saia do loop de espera e entre no loop do jogo em si;
  + Por fim, caso o evento seja o usuário apertando o botão de fechar, a execução do programa é interrompida e o jogo é encerrado.
* Logo antes do loop do jogo ser iniciado o som de início de jogo é executado uma vez;
* É iniciado então o segundo dos loops principais do programa. Esse loop é o playing e ele é responsável pela execução do jogo em si. Sobre o seu funcionamento:
  + A música de fundo é iniciada novamente após o som de início de jogo;
  + É criada uma variável de evento e logo em seguida o programa espera por um evento para ser armazenado nela;
  + Caso o evento seja do tipo timer ocorrem os seguintes processos:
    - A variável cont\_frames é incrementada por 1;
    - Se cont\_frames passar do valor de frames\_sprite, o valor de cont\_frames é resetado para 0;
    - Com isso o parâmetro coluna\_atual do impostor é incrementado por 1. Caso ele passe do valor do parâmetro colunas\_folha o valor de coluna\_atual é volta a ser 0 e linha\_atual é incrementado. Caso linha\_atual seja maior do que linhas\_folha, o seu valor volta a ser 0;
    - Então são calculadas as região\_x\_folha e região\_y\_folha com base nos valores de linha\_atual, coluna\_atual, altura e largura;
    - Esse mesmo processo é repetido para os tripulantes, porém dessa vez dentro de dois loops que iteram a matriz para garantir que todos os tripulantes serão atualizados;
    - Após a atualização dos sprites a velocidade x e y do impostor é somada em suas posições x e y respectivamente para movimentá-lo;
    - Verifica-se então se o impostor está encostando em uma das bordas da tela e, caso ele esteja, a sua posição x fica fixa enquanto esse for o caso para evitar que ele saia da tela;
    - Itera-se novamente pela matriz de tripulantes, dessa vez para checar se algum deles está encostando no canto da tela;
    - Se esse for o caso, é realizada uma iteração pela matriz dentro dos dois loops aninhados que inverte a velocidade horizontal de todos os tripulantes e soma a velocidade vertical deles em sua posição y;
    - Após verificar se os tripulantes encostaram na borda da tela, todos eles têm a sua velocidade x somada a sua posição x, para mover a tripulação horizontalmente;
    - É então verificado se a faca passou do limite superior da tela. Se esse for o caso a variável que armazena o estado da faca (parada ou se ela foi atirada) volta a ser 0;
    - Desenha-se então na tela o fundo, a tripulação (novamente através de dois loops que iteram por toda a matriz), a faca e o impostor;
    - Se a faca tiver sido atirada ela sobe 5 pixels a cada iteração desse loop;
    - São declaradas 3 novas variáveis: colidiuFaca, colidiuSolo e colidiuImpostor. Elas servem para, respectivamente, armazenar se houve colisão da da faca com um tripulante, da tripulação com o solo e de um tripulante com o impostor;
    - Itera-se novamente pela tripulação e são chamadas as funções colisaoCrewFaca, colisaoCrewSolo e colisaoCrewImpostor para verificar se houve alguma colisão;
    - Caso tenha ocorrido uma colisão da faca com um tripulante, é chamada a função deleteCrew para deletar esse tripulante. Além disso a pontuação é incrementada por 1 e é possível atirar novamente a faca. É executado aleatóriamente um dentre dois sons possíveis para sinalizar a morte de um tripulante;
    - Caso a colisão tenha sido entre o tripulante e o chão ou entre o tripulante e o impostor, a variável responsável playing passa a ser 0 e a variável mostraRecorde passa a ser 1, o que encerra o loop de jogo e inicia o próximo loop no final dessa iteração. A tela é congelada por 1 segundo e então é dado um comando break para sair do loop;
    - Desenha-se então a pontuação atual na tela e é dado o comando de al\_flip\_display para atualizar a tela.
  + Caso o evento seja o usuário apertando o botão de fechar, a execução do programa é interrompida e o jogo é encerrado;
  + Caso o evento seja uma tecla sendo apertada ocorrem os seguintes processos:
    - Se a tecla apertada tiver sido ou o impostor tem seu parâmetro vel\_x igualado a -VEL\_IMP;



* + - Se a tecla apertada tiver sido ou o impostor tem seu parâmetro vel\_x igualado a VEL\_IMP;



* + - Se a tecla apertada tiver sido a faca é atirada.
  + Caso o evento seja uma tecla sendo solta ocorrem os seguintes processos:
    - Se a tecla soltada tiver sido ou o impostor tem seu parâmetro vel\_x igualado a 0;



* + - Se a tecla soltada tiver sido ou o impostor tem seu parâmetro vel\_x igualado a 0;
  + Por fim, se a pontuação do jogador for igual ao número de tripulantes que haviam no início do jogo, a variável playing passa a ser 0, a variável mostraRecorde passa a ser 1 e a tela é congelada por 1 segundo. Isso encerra o loop de jogo e inicia o próximo loop no final dessa iteração.
* Logo antes do próximo loop ser iniciado o som de fim de jogo é executado uma vez;
* É iniciado então o último dos loops principais do programa. Esse loop é o mostraRecorde e ele é responsável por mostrar na tela a pontuação final do jogador e o recorde das pontuações. Sobre o seu funcionamento:
  + É criada uma variável de evento e logo em seguida o programa começa a esperar por um evento;
  + Caso esse evento seja do tipo time ocorrem os seguintes processos:
    - É declarada uma variável recorde que irá armazenar a maior pontuação já obtida no jogo;
    - É desenhado o fundo na tela;
    - Por cima desse fundo é desenhada a pontuação obtida pelo jogador em vermelho;
    - Verifica-se se a pontuação do jogador é maior do que o recorde anterior utilizando a função novoRecorde. Se esse for o caso é exibido na tela “NOVO RECORDE!” e caso contrário o recorde é exibido.
  + Caso o evento seja o usuário apertando o botão de fechar, a execução do programa é interrompida e o jogo é encerrado.
* Por fim, são destruídos o timer, o display, a fila de eventos, os bitmaps, a fonte e os arquivos de áudio utilizados ao longo do programa.