Universidade Federal do Rio Grande do Sul
Instituto de Informática
INF01202 - Algoritmos e Programação - Turmas G/H - 2023/2
Prof. Renan Maffei

Especificação do Trabalho Final

1. Motivação e Objetivos

O trabalho final tem por objetivo exercitar as habilidades e conceitos de programação desenvolvidos ao longo da disciplina INF01202 através da **implementação de um jogo de computador em linguagem C**. Este trabalho também permitirá ao aluno exercitar as habilidades de pesquisa, uma vez que será possível utilizar bibliotecas e conceitos ainda não trabalhados em aula para implementar certas funcionalidades. Outro objetivo do trabalho é exercitar a habilidade de desenvolvimento em equipe, uma vez que o trabalho (implementação) deve ser **realizado por duplas** de alunos.

2. Requisitos Administrativos

2.1. Etapas de entrega

O trabalho será entregue em 3 etapas:

A. Andamento Data: 19/01/2024

Envio do código parcial do trabalho, acompanhado de breve relato para explicar o andamento do trabalho e mostrar o que já está feito. É obrigatório ter alguma parte do código do trabalho funcional e que possibilite uma demonstração (em video pré-gravado, duração < 5 min). O não cumprimento desta etapa implicará em desconto de 10% da nota final obtida pelo aluno!

OBS: A formalização das duplas deve ser feita e informada ao professor com no mínimo uma semana de antecedência (até 12/01/2024). Será disponibilizado um espaço no Moodle para isso.

B. Entrega Final Data: 14/02/2024

A dupla deverá submeter via Moodle um arquivo zip contendo:

- Um relatório com a descrição do trabalho realizado, a especificação completa de como os elementos do jogo foram representados, como foi implementada a interação dos componentes interativos, bem como as estruturas e funções utilizadas e uma explicação de como usar o programa.
- Os códigos fontes devidamente organizados e documentados (.c).
- Bibliotecas necessárias para executar os códigos.

C. Apresentação Data: 15/02/2024

Os alunos devem ser capazes de explicar todos os recursos da linguagem, comandos e bibliotecas utilizados no seu código. É permitido usar recursos não vistos em aula desde que se saiba explicar como funcionam e para que servem. Ambos membros da dupla devem saber responder perguntas sobre qualquer trecho do código e estes serão avaliados separadamente.

2.2. Avaliação

A avaliação levará em conta os seguintes critérios:

- Atendimento aos requisitos definidos;
- Estruturação do código em módulos;
- Uso das estruturas e funcionalidades vistas em aula;
- Documentação geral do código (comentários);
- Indentação do código;
- "Jogabilidade" do jogo.

Importante: Trabalhos copiados não serão considerados. Existem ferramentas que possibilitam a detecção automática de plágio, as quais serão utilizadas na correção. **Se for detectado plágio, todos os trabalhos relacionados serão desconsiderados**.

3. Descrição Geral e Regras do Jogo a Ser Implementado

O jogo que deverá ser implementado é o jogo eletrônico de quebrar blocos onde se escolhe a direção de tiro da bolinha. Existem muitas versões desse jogo com diferentes nomes como "*Bricks Ball Crusher*", "*Bricks n Balls*", "*Brick Breaker*", "*Bricks vs Balls*", portanto para simplificar vamos chamá-lo de **Bricks**.

No jogo há várias fileiras de tijolos no topo da tela e se dispara uma bolinha da parte inferior em direção aos tijolos, podendo escolher a direção que a bola será lançada. A bola rebate nos tijolos, destruindo-os, e nas paredes do topo e das laterais. Cada tijolo pode ter um valor diferente, indicando quantas vezes a bolinha deve bater nele para que ele seja destruído. Como só é possível controlar o lançamento da bolinha, ela acaba saindo pela parte inferior da tela após rebater nos tijolos. O objetivo do jogo é eliminar o máximo de tijolos até o número de disparos se esgotar.

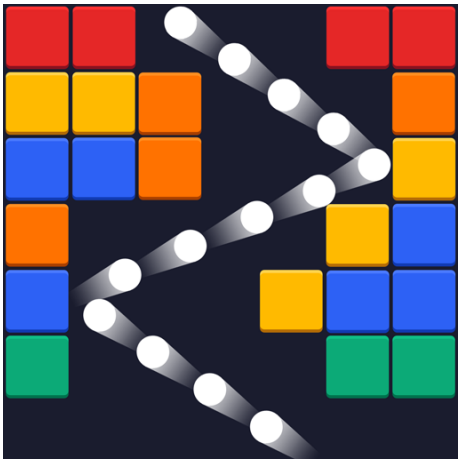

Fig. 1: Bricks

obs: não confundir com o jogo Breakout do Atari e suas variações, onde o jogador controla uma barrinha na parte inferior da tela que se desloca horizontalmente para rebater a bola e evitar que ela caia.

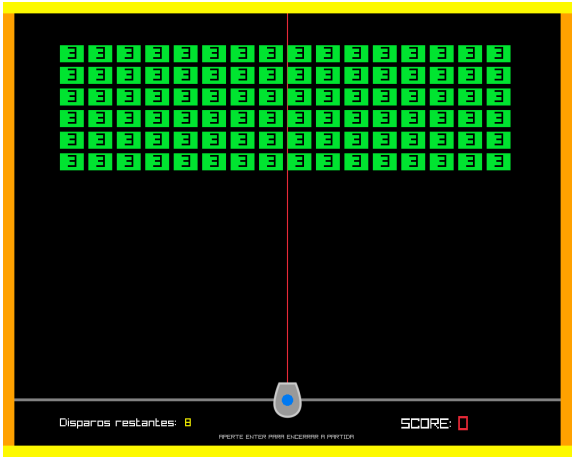

Fig. 2: Exemplo de configuração inicial

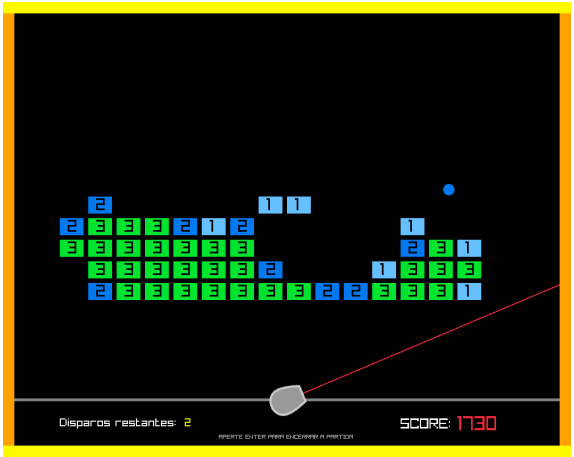

Fig. 3: Andamento do jogo

3.1. Características do jogo

1. O jogador controla apenas a **direção** do "canhão" que dispara bolinhas e o **gatilho** que indica o momento do disparo das bolinhas.
2. **Somente uma bolinha atua por vez no espaço de jogo**. O canhão só é recarregado, permitindo um novo disparo, após a bolinha anterior sair pela parte inferior da tela.
3. No momento em que o gatilho é disparado, a bolinha começa a se mover com **velocidade constante** tendo como **orientação inicial a direção do canhão** no instante do disparo.
    - A velocidade pode ser decomposta em duas componentes: vertical e horizontal.
4. Quando a bolinha rebate em algum obstáculo (paredes ou tijolos) sua velocidade é espelhada
    - Se rebate em uma superfície horizontal, a velocidade vertical é espelhada.
    - Se rebate em uma superfície vertical, a velocidade horizontal é espelhada.
5. Sempre que a bolinha colide com um tijolo, o valor deste tijolo é decrescido de uma unidade. Caso o valor do tijolo seja 1, o tijolo desaparece (é destruído).
    - Cada colisão com um tijolo conta pontos.
6. OPCIONAL: a cada novo disparo de uma bolinha, os tijolos descem uma linha para baixo, se aproximando do canhão e dificultando o processo para o jogador (vide Fig. 3).

O jogo termina quando o número de disparos se esgota ou quando todos os tijolos são destruídos.

3.2. Parâmetros do jogo

- Basicamente tudo no jogo é parametrizável. **Usar defines!**
- Exemplo
    - Largura e altura da tela (no exemplo, 1000x800 pixels)
    - Número de tijolos por linha e por coluna (no exemplo, 16x18)
    - Largura e altura de tijolo (no exemplo, 42x30 pixels)
    - Valores iniciais de cada tijolo (no exemplo, 3)
    - Número de disparos (no exemplo, 8)

- Pontuação a cada colisão com um tijolo (no exemplo, 10 pontos)
- Ângulos mínimo e máximo da direção do canhão (no exemplo, -80 a +80 graus do eixo vertical)
- Raio da bola (no exemplo, 10 pixels)
- Velocidade da bola (no exemplo, 10 pixels p/ iteração)
- ...

3.3. Informações a serem mostradas ao jogador durante o jogo

- Score (pontuação baseada no total de tijolos atingidos)
- Número de disparos restantes

4. Requisitos Mínimos de Implementação

A implementação realizada pelos alunos deverá respeitar os seguintes requisitos mínimos:

4.1. Visualização da tela de jogo, com jogo funcional

- Os elementos visuais do jogo devem ser implementados e exibidos visualmente. A representação visual pode ser em modo texto ou com o carregamento de imagens (por exemplo, o canhão nas Figs. 2 e 3 são uma imagem carregada e rotacionada).
  - A biblioteca `raylib.h` permite a manipulação de imagens, como as usadas na implementação de exemplo.
- A tela de jogo deve mostrar a bolinha, os tijolos (ainda não destruídos), a linha de mira (para auxiliar no disparo da bolinha) e as marcações textuais (score e disparos restantes).
- O jogo deve funcionar minimamente: o canhão deve poder ser direcionado, a bolinha disparada e os tijolos destruídos.

4.2. Tela de abertura e Carregamento de Configuração Salva

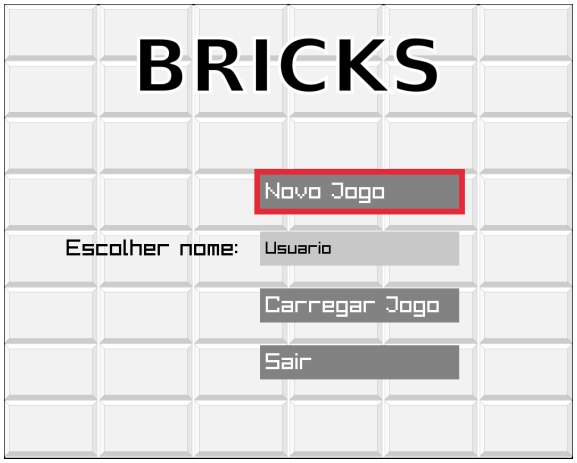

Fig. 4: Exemplo de tela de abertura

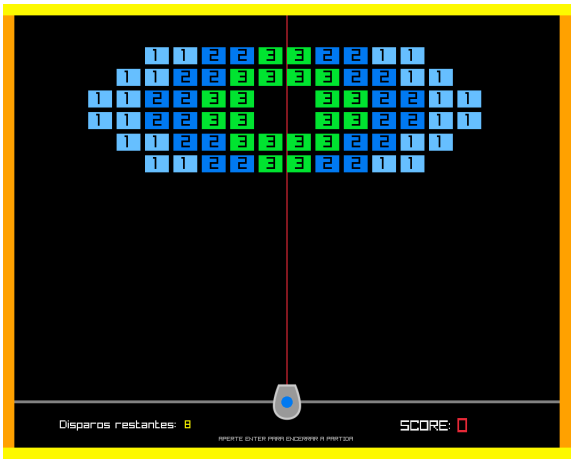

Fig. 5: Exemplo de config. carregada

- No início, o programa deverá permitir a escolha das opções:
  - **Novo Jogo**
  - **Carregar Jogo**
  - **Sair**
- Se for escolhido **novo jogo**, o programa deve ir para a tela de jogo e iniciar com uma configuração padrão de tijolos, como no exemplo da Fig. 2.

- Se for escolhido **carregar**, o programa deve carregar um arquivo contendo a descrição de uma configuração salva (pode ser um nome padrão).
  - A configuração a ser carregada é a descrição dos valores dos tijolos. Escolher carregar uma configuração existente, começará um jogo do inicio, mas com essa configuração de tijolos descrita no arquivo.
  - Ver exemplo de arquivo de configuração `conf.txt`, que gera a configuração da Fig. 5.
- Se for escolhido **sair**, o programa deve ser fechado
- OPCIONAL: possibilitar editar nome do jogador para futuramente mostrar nos scores

4.3. Tela de encerramento

- Quando o jogador solucionar o jogo (destruir todos os tijolos), for derrotado (esgotar o número de disparos), ou quando o mesmo apertar a tecla ENTER durante o jogo, o programa deverá ir para uma tela de encerramento.
- Similar à tela de abertura, ela conterá as seguintes opções:
  - **Resetar Jogo**
  - **Voltar a tela inicial**
  - **Sair**

Fig. 6: Exemplo de tela de encerramento

- Se for escolhido **resetar jogo**, o programa deve ir para a tela de jogo e reiniciar com a mesma configuração de tijolos recém-jogada (mas com os tijolos em suas posições e valores iniciais).
- Se for escolhido **voltar**, o programa deve retornar para a tela de abertura.
- Se for escolhido **sair**, o programa deve ser fechado
- Nesta tela, o programa também deverá mostrar o score final do jogo que se encerrou.

4.4. Controle por parte do jogador

O controle poderá ser feito via teclado ou mouse (ou ambos). Usando a raylib, as duas formas tem grau de dificuldade similar. **Ao menos uma das duas formas precisa ser implementada**.

- **Se usar teclado**, o jogador moverá a direção do canhão da esquerda para a direita, e vice-versa, através das **teclas direcionais** ($\leftarrow$, $\rightarrow$). O disparo do canhão deverá ser feito apertando-se a tecla de ESPAÇO.
- **Se usar mouse**, o jogador moverá a direção do canhão através do movimento do mouse. Neste caso, a mira deve ser computada pela reta entre o ponto de origem do canhão e a posição momentânea do mouse. O disparo do canhão deverá ser feito apertando-se o botão esquerdo do mouse.

  - Caso deseje-se usar teclado e mouse simultaneamente, poderá haver conflito na escolha da posição do cursor. Uma possível solução é decidir por um ou outro, por exemplo, caso o mouse esteja sobre a tela de jogo ele é quem indica a direção do canhão, caso esteja posicionado fora da tela de jogo, o teclado indica a direção do canhão.

4.5. Estruturas

- O programa deverá conter ao menos uma estrutura para representar o **estado da bola**, e essa estrutura conterá a sua **posição** e sua **velocidade**. Pode ter mais variáveis, como uma flag dizendo se já começou a se mover ou não.
- Deverá conter ao menos uma estrutura para representar o **estado do jogo**, contendo o **ângulo do canhão**, o **número de disparos restantes**, o **score** e o **estado da bola**.
- Deverá conter ao menos uma estrutura para representar um **tijolo**, contendo **posição** e **valor**.
- Mais estruturas podem ser usadas, a critério de cada um.

4.6. Vetores e Matrizes

- A descrição dos tijolos deverá envolver o uso de uma matriz, declarada na *main*.
  - Evitar uso de variáveis globais.
- Os eventuais vetores (ex: strings) e matrizes utilizados deverão ter tamanhos compatíveis com o que se espera deles.
- Os tamanhos devem ser definidos via `#define`.

4.7. Modularização

- Devido ao grande tamanho esperado, é muito importante que o programa seja altamente modularizado através de **funções**
  - As variáveis necessárias (vetores, matrizes, estruturas, etc) devem ser cuidadosamente passadas como parâmetro na chamada das funções, via cópia ou referência, dependendo da necessidade.
- Para uma melhor modularização do código, sugere-se também a organização do programa em **módulos descritos em diferentes arquivos** (por exemplo, separando o gerenciamento do jogo, gerenciamento das telas de abertura e encerramento, etc)
  - Cada módulo deve ter interfaces bem definidas (arquivos `.c` e `.h` para cada módulo).
- Abaixo estão algumas funções que podem ser desenvolvidas neste trabalho:
  - função para inicializar as variáveis do jogo (configuração inicial do canhão, bola, tijolos, ...);
  - função para desenhar/atualizar as telas de menu;
  - função para desenhar/atualizar a tela do jogo;
  - função para checar a ação feita pelo jogador (via teclado ou mouse);
  - função para checar colisão com tijolos;
  - função para atualizar estado da bola;
  - etc.

5. Tarefas extras

Segue abaixo algumas sugestões de tarefas extras que podem ser implementadas. Embora opcionais, essas tarefas melhorarão a avaliação final do seu trabalho. Também fique a vontade para implementar outras funcionalidades que achar interessante (mas não esqueça dos requisitos mínimos).

5.1. Escolha do nome do usuário

- Permitir que o jogador escolha o seu nome para posteriormente listar o score obtido junto ao nome.
- Para tal, é preciso possibilitar a edição de uma string na tela de abertura (vide Fig. 4). Na página da Raylib há um exemplo de como fazer isso.

5.2. Carregamento de múltiplas configurações de jogo

- Além de permitir o carregamento de uma configuração padrão salva, permitir o carregamento de qualquer configuração salva informando o nome do arquivo. (Usar mesma forma de leitura de string da tarefa anterior).

5.3. Mostrar lista de scores de todos os jogos disputados durante a execução do programa

- Na tela de encerramento, mostrar não apenas o último score obtido, mas também todos os nomes e scores obtidos durante a execução atual do programa.

5.4. Armazenamento em arquivo dos scores dos jogos

- Manter a lista de scores salva em um arquivo de forma persistente.
- Sempre que um jogo for encerrado (quer seja porque o jogador concluiu a partida, ou porque ele desistiu do jogo), o score do jogo deve ser **salvo em um arquivo** contendo todos os scores já jogados.
- Mostrar na tela de encerramento as 5 maiores pontuações já obtidas (lendo os valores do arquivo de scores). Mostre **em ordem decrescente** na tela essas 5 pontuações e os nomes dos jogadores que as conseguiram.

5.5. Adicionar variações ao jogo

- É comum nos jogos deste tipo existirem variações, como a descida dos tijolos a cada novo disparo. Além disso, é possível criar posições especiais que caso atingidas pela bola gerem alguma ação, como um disparo simultaneo em todos os tijolos daquela linha ou coluna.