Trabalho Prático: Tower Defense

# 1. Objetivo Geral

Desenvolver um jogo funcional do gênero *Tower Defense* utilizando a biblioteca **Processing**. O projeto deverá aplicar de forma prática os conceitos de **Teoria dos Grafos**, o algoritmo de menor caminho de **Dijkstra** para a inteligência artificial dos inimigos, e o uso de **Filas** para o gerenciamento das ondas de ataque.

# 2. Contexto do Jogo

Sua base está sob ataque! Hordas de inimigos estão surgindo no campo de batalha e avançando implacavelmente em direção ao seu "Núcleo de Energia". Sua missão é usar recursos limitados para construir uma defesa estratégica, posicionando paredes, torres e armadilhas de areia para impedir que os inimigos alcancem o núcleo. A sobrevivência depende da sua capacidade de forçar os inimigos a tomar os caminhos mais longos e perigosos.

# 3. Requisitos Técnicos Obrigatórios

1. **Linguagem/Framework:** O projeto deve ser desenvolvido inteiramente em **Processing**.
2. **Representação do Mundo:** O jogo deve ocorrer em um **grid 2D** (matriz) de N x M células (ex: 40x30).
3. **Algoritmo de Dijkstra:**
   * O grid do jogo deve ser modelado como um **grafo**. Cada célula do grid é um **nó**.
   * Existem **arestas** entre nós adjacentes (não-diagonais), representando a possibilidade de movimento.
   * Os inimigos **devem** usar o algoritmo de Dijkstra para calcular o caminho de **menor custo** do seu ponto de partida até o "Núcleo de Energia".
   * O **custo (peso) da aresta** é o fator central da estratégia. Ele deve ser manipulado pelas construções do jogador.
4. **Estrutura de Dados (Fila):**
   * Uma **Fila (Queue)** deve ser utilizada para gerenciar as ondas de inimigos. A cada nova onda, os inimigos são enfileirados e, em intervalos de tempo, um inimigo é "desenfileirado" e colocado no mapa.

# 4. Regras e Mecânicas do Jogo

* **O Grid e o Cenário:**
  + Um **"Núcleo de Energia"** (bloco a ser protegido) deve estar posicionado em uma ou mais células na coluna mais à esquerda do grid.
  + Os inimigos sempre surgem (spawn) de uma ou mais células na coluna mais à direita.
* **Inimigos:**
  + Possuem **Pontos de Vida (HP)** (ex: 3 HP).
  + Movimentam-se de um nó a outro do grid, seguindo o caminho calculado por Dijkstra.
  + **Importante:** Se o jogador construir uma parede que bloqueia o caminho atual do inimigo, o inimigo deve ser capaz de **recalcular um novo caminho** a partir de sua posição atual.
* **Recursos do Jogador:**
  + O jogador começa com uma quantia inicial de **dinheiro** (ex: 100 moedas).
  + O jogador ganha uma recompensa em dinheiro por cada inimigo derrotado.
* **Estruturas Defensivas (Construídas pelo Jogador):**
  + O jogador usa o mouse para selecionar e construir estruturas no grid.
  + **Parede:**
    - **Custo:** [Definir um custo, ex: 25 moedas]
    - **Efeito:** Impede totalmente a passagem. Para o algoritmo de Dijkstra, o custo de se mover para um nó com parede é **infinito**.
  + **Areia:**
    - **Custo:** [Definir um custo, ex: 10 moedas]
    - **Efeito:** O inimigo se move com **metade da velocidade** sobre a areia. Isso deve ser implementado no algoritmo de Dijkstra da seguinte forma: o **custo da aresta** para se mover para um nó de areia deve ser o **dobro** do custo de um nó normal. (Custo normal = 1, Custo areia = 2).
  + **Torre de Defesa:**
    - **Custo:** [Definir um custo, ex: 40 moedas]
    - **Efeito:** Atira projéteis nos inimigos que entram em seu **raio de alcance**.
    - **Dano:** 1 HP por tiro.
    - **Cadência de Tiro:** A torre deve ter um intervalo entre os disparos (ex: 1 tiro por segundo).
* **Condições de Vitória e Derrota:**
  + **Derrota:** O jogo termina se **um único inimigo** alcançar o "Núcleo de Energia".
  + **Vitória:** O jogador vence se sobreviver a um número pré-determinado de ondas de inimigos (ex: 10 ondas).

# 5. Desafios Adicionais (Até mais 2 pontos)

* **Inimigos Variados:** Criar diferentes tipos de inimigos (ex: um rápido com pouca vida, um lento com muita vida).
* **Melhoria de Torres:** Permitir que o jogador gaste dinheiro para melhorar o dano, o alcance ou a cadência de tiro de uma torre existente.
* **Venda de Estruturas:** Implementar a funcionalidade de vender uma estrutura construída (talvez por metade do preço de compra).
* **Algoritmo A**\*: Para grupos que terminarem antes, substituir o Dijkstra pelo algoritmo A\* (A-Estrela), que é mais eficiente para *pathfinding* em jogos, e explicar a diferença de performance. Ou usar algoritmos que independem de conhecer o grafo, como o bug-2.
* **Efeitos Visuais e Sonoros:** Adicionar sons e animações para tiros, explosões e movimentos para tornar o jogo mais polido.

# 8. Entregáveis

1. **Código-Fonte:** Projeto completo e comentado, enviado em um arquivo .zip.
2. **Apresentação Final:** Apresentação explicando os conceitos e estratégias utilizadas para solução dos problemas.
3. **Vídeo:** Uma breve demonstração ao vivo (5-10 minutos) do jogo funcionando.

# 8. Critérios de Avaliação

* **Funcionalidade,**
* **Apresentação,**
* **Video,**
* **Pontos extra.**