Asteroides (Click)

Documentação integrada: Fluxo Técnico + Arquitetura Técnica (Versão 1)

Este documento descreve como o jogo funciona em tempo de execução (estados, eventos e loops) e como o código deve se organizar (módulos, responsabilidades e contratos).

Escopo: PWA (browser) e TWA (APK) com 1 asteroide por vez, 1-hit-kill, tela inicial, HUD de tempo, pausa, fullscreen, modal de personalização com draft/apply, e modal de confirmação ao encerrar.

Diagrama UML (PDF): uml_asteroides_click_v1.pdf (mantenha este arquivo na mesma pasta deste PDF para abrir pelo link).

Sumário

1. Visão geral do runtime

O jogo é dirigido por eventos (cliques, teclas e botões), por uma máquina de estados de UI, e por um loop de atualização que movimenta o asteroide e conta o tempo de partida.

Loop principal: a engine calcula delta time (dt) e atualiza somente quando a partida está rodando e não está pausada.

Restrição v1: existe no máximo 1 asteroide ativo. Quando ele é destruído ou foge da tela, um novo é gerado.

Componentes em tempo de execução:

- **UI**: StartScreen, HUD, Modal de Personalizar, Modal de Confirmação de Encerramento.
- **Engine**: estado da partida, spawn, movimento, hit-test, pause/resume, estatísticas.
- **Infra**: fullscreen, storage (last run), helpers.

Clock/tempo: o contador exibido no HUD reflete o tempo efetivo de jogo (não inclui períodos em pausa). Na implementação, some dt apenas quando *isPaused=false*.

Runtime (alto nível)

```
[UIController] ---- eventos DOM ----> [GameEngine] ---- callbacks ----> [UIController]
 |                                                 |
 |---- render HUD / telas ---->                     |---- atualiza estado ----> (state snapshot
 |
 +---- Storage.save(lastRun) <----- onEnd(stats)
```

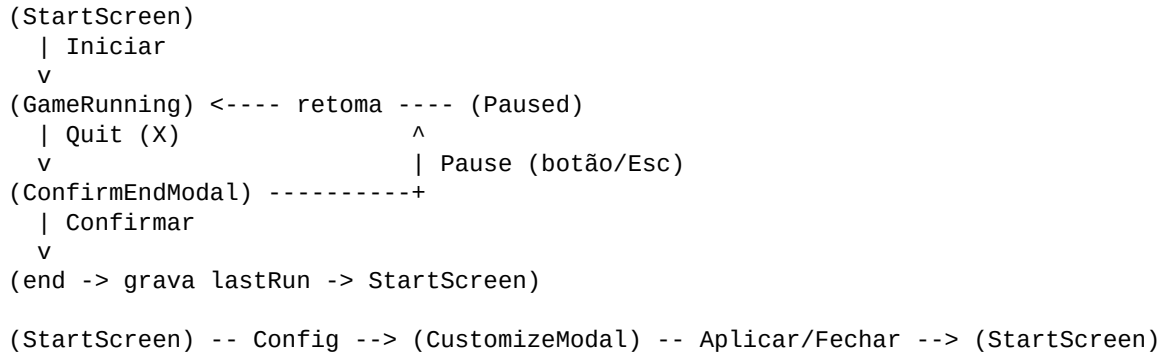
2. Máquina de estados (UI + jogo)

A UI e o jogo compartilham uma máquina de estados simples. Alguns estados são visuais (telas/modais) e outros são lógicos (pausa).

2.1 Estados

Estado	Tipo	Descrição	Entradas	Saídas
StartScreen	UI	Menu inicial com estatísticas da última partida.	Init / End	Start → GameRunning Config → CustomizeModal Fullscreen → (toggle)
CustomizeModal	UI (modal)	Ajustes em *draft*; só aplicam no botão **Aplicar**.	Config (StartScreen)	Fechar → StartScreen Aplicar → StartScreen
GameRunning	Jogo	Partida ativa com HUD e asteroide atravessando (1 por vez).	Start	Pause → Paused Quit (X) → ConfirmEndModal End → StartScreen
Paused	Lógico	Congela tempo e updates do asteroide.	Pause	Resume → GameRunning Quit (X) → ConfirmEndModal
ConfirmEndModal	UI (modal)	Confirma encerramento; ao abrir, pausa o jogo.	Quit (GameRunning/Paused)	Cancelar → retoma estado anterior Confirmar → End

2.2 Diagrama de transição (simplificado)



3. Fluxos detalhados (sequências)

Nesta seção, descrevemos as sequências de eventos e chamadas entre UI e Engine, com ênfase em pontos críticos (pausa, modais e persistência).

3.1 Iniciar partida

```
Usuário           UIController                 GameEngine
  | click Iniciar     |                           |
  |------------------>| start(settings)           |
  |                   |-------------------------->|
  |                   | reset stats/time          |
  |                   | isRunning=true            |
  |                   | isPaused=false            |
  |                   | spawnAsteroid()           |
  |                   |<--------------------------|
  |                   | show HUD / hide Start     |
  |                   | begin loop (rAF/timer)    |
```

Regras: resetar contadores (destruídos, fugas, cliques), zerar o tempo de partida, e inicializar o asteroide único.

3.2 Tick do jogo (movimento + tempo)

```
Loop (rAF/timer)
  dt = now - last
  if isRunning && !isPaused:
    timeMs += dt
    speed = Difficulty(...)
    asteroid.update(dt, speed)
    if asteroid saiu da tela:
      stats.escapes++
      asteroid = null
      spawnAsteroid()
  UIController.onTick(snapshot) -> atualiza HUD
```

O tick deve ser determinístico: dt em milissegundos. Se velocidade estiver em px/s, converta deslocamento por frame: *dx = v * (dt/1000)*.

3.3 Clique para destruir (1-hit-kill)

```
Usuário click na tela/canvas
  UIController captura ponteiro (x,y)
  stats.clicks++
  engine.handlePointerDown(x,y)
    if asteroid.hitTest(x,y):
      stats.destroyed++
      asteroid = null
      spawnAsteroid()
  UI atualiza destroyed/precisão
```

Precisão recomendada: *destroyed / clicks* (com proteção para divisão por zero).

3.4 Encerrar com confirmação (modal pausa o jogo)

```
Usuário click X (encerrar)
  UIController.openConfirmEnd()
  engine.pause() (se ainda não pausado)
  mostra modal (ConfirmEndModal)
  Cancelar/Backdrop:
```

```
    UIController.closeConfirmEnd()
    esconde modal
    engine.resume() (retoma estado anterior)

  Confirmar:
    UIController.confirmEnd()
    engine.end()
    isRunning=false
    finalize stats (precisão, tempo)
    UIController.onEnd(stats)
    Storage.saveLastRun(stats)
    volta StartScreen
```

4. Arquitetura de código (módulos e dependências)

A arquitetura v1 prioriza separação entre regras do jogo (core), apresentação (ui) e integrações (infra). A Engine não conhece DOM.

4.1 Estrutura de diretórios recomendada

```
/src
  /core
    GameEngine.js
    Asteroid.js
    Difficulty.js
    Stats.js
  /ui
    UIController.js
    domRefs.js
  /infra
    Storage.js
    Fullscreen.js
  main.js
index.html
styles.css
manifest.webmanifest
/icons/...
```

4.2 Dependências permitidas

- UIController pode depender de core e infra.
- Core (GameEngine/Asteroid/Difficulty/Stats) não depende de DOM nem de infra.
- Infra não depende de core nem de UI (funções isoladas).

4.3 Responsabilidades por módulo

Módulo	Responsabilidade principal	Não deve fazer
GameEngine	Loop, estado da partida, spawn/movimento, hit-test, pause/end.	Manipular DOM diretamente.
Asteroid	Modelar asteroide ativo e lógica local (update/hit/out).	Ler settings globais; usar localStorage.
Difficulty	Mapear níveis de velocidade e progressão opcional.	Alterar stats/UI.
Stats	Acumular e finalizar métricas da partida.	Decidir layout de exibição.
UIController	Bind de UI, telas/modais, render HUD/start, aplicar settings.	Regra de movimento/colisão.
Storage	Persistir/ler lastRun.	Conhecer detalhes do DOM.
Fullscreen	Toggle e HTML do ícone; helpers.	Controlar fluxo do jogo.

Referência: o diagrama UML completo está no PDF uml_asteroides_click_v1.pdf.

5. Contratos e dados (interfaces, eventos, storage)

Para evitar acoplamento, o core expõe métodos simples e publica snapshots para a UI via callbacks.

5.1 API mínima recomendada do GameEngine

```
GameEngine
  - start(settings)
  - setSettings(settings)          // aplicado após 'Aplicar' no modal
  - handlePointerDown(x, y)
  - pause() / resume() / togglePause()
  - end()                          // finaliza stats e dispara callback onEnd

Callbacks (injetados no construtor ou via setters)
  - onTick(snapshot)               // chamado em cada tick útil
  - onEnd(stats)                   // chamado ao finalizar
  - onPauseChange(isPaused)        // opcional
```

5.2 Estruturas de dados

Settings (exemplo): opacity (0.1..1.0, passo 0.1), speedLevel (1..5 com mínimo efetivo v1 = nível 3), difficultyProgressionEnabled (boolean).

Stats (exemplo): destroyed, escapes, clicks, precision, timeMs (tempo efetivo).

5.3 Persistência (Última partida)

Persistência local via localStorage. O engine não persiste; a UI persiste quando recebe onEnd(stats).

```
Storage.saveLastRun(stats):
  localStorage.setItem("lastRun", JSON.stringify(stats))

Storage.loadLastRun():
  const raw = localStorage.getItem("lastRun")
  return raw ? JSON.parse(raw) : null
```

6. Regras de configuração (draft/apply, opacidade, velocidade, progressão)

A tela de personalização opera em duas camadas: **draft** (o que o usuário está ajustando) e **settings** (o que está aplicado).

6.1 Draft/Apply

- Ao abrir o modal: copiar settings → draft.
- Ao mover sliders: atualizar apenas draft.
- Ao clicar Aplicar: copiar draft → settings e chamar engine.setSettings(settings).

6.2 Opacidade (UI)

- Intervalo: 10% a 100%, em passos de 10%.
- Aplica-se a textos e painéis exibidos no jogo (HUD e overlays).
- Implementação típica: variável CSS --ui-alpha e uso de rgba().

6.3 Velocidade por níveis (5 níveis)

O slider apresenta níveis discretos. Internamente, mapeie para velocidade base em px/s. A diferença entre níveis deve ser perceptível, com teto mais alto.

Nível	Velocidade base	Observação
1	260 px/s	mais lento
2	340 px/s	lento
3	450 px/s	mínimo efetivo v1
4	620 px/s	rápido
5	820 px/s	muito rápido

6.4 Progressão de dificuldade (liga/desliga)

Quando habilitada, a progressão aumenta a velocidade ao longo da partida. Regra determinística sugerida:

```
if difficultyProgressionEnabled:
  factor = 1 + min(0.75, (timeMs/60000)*0.15 + destroyed*0.01)
  speed = baseSpeed * factor
else:
  speed = baseSpeed
```

6.5 Trajetória perto do centro (com tolerância)

A geração de trajetória deve concentrar passagem próxima ao centro, mas com amplitude configurável ("mais distante do centro"). Implementação típica: escolher um ponto-alvo no eixo central com jitter controlado e interpolar.

7. Apêndice: checklist técnico v1

Funcionalidades core

- [ ] 1 asteroide por vez
- [ ] 1-hit-kill por clique
- [ ] Contador de tempo (pausa não conta)
- [ ] Pontuação por destruídos
- [ ] Botão pausa (ícone) + Esc (se aplicável)
- [ ] Botão fullscreen (ícone) no menu e no jogo
- [ ] X encerrar com modal de confirmação (modal pausa o jogo)
- [ ] Última partida exibida no StartScreen (localStorage)

Personalização

- [ ] Modal sobrepondo todo o StartScreen, com ícone de fechar
- [ ] Opacidade 10%..100% em passos de 10%
- [ ] Velocidade por níveis (5), com diferenças perceptíveis
- [ ] Botão 'Aplicar alterações'
- [ ] Toggle de progressão de dificuldade

Empacotamento (TWA)

- [ ] PWA em HTTPS
- [ ] manifest.webmanifest acessível
- [ ] ícones 192/512 acessíveis
- [ ] assetlinks.json publicado no escopo correto
- [ ] APK/AAB assinado com keystore e versionCode incrementado a cada release