Relatório Técnico - Desenvolvimento de Jogo da Cobrinha Mobile

**Professor:** Davi Camara

**Alunos:** Caio Menezes, Edivaldo Victor, Aredson Mário, Antônio Filipe

**Data:** 02 de Junho de 2025

**1. Introdução**

O presente relatório técnico descreve o processo de desenvolvimento de um jogo mobile do tipo "Snake" (jogo da cobrinha), realizado pelos alunos Caio Menezes, Edivaldo Victor e Aredson Mário, sob a orientação do Professor Davi Camara. O projeto teve como plataforma de desenvolvimento o framework Ionic em conjunto com Angular, visando a criação de uma aplicação híbrida. O escopo do trabalho envolveu a construção, teste, adaptação e implementação de um código base preexistente, com o objetivo de entregar um jogo funcional e interativo para dispositivos móveis. Este documento detalha as etapas de desenvolvimento, as funcionalidades implementadas, as dificuldades encontradas e as conclusões obtidas ao final do projeto.

**2. Desenvolvimento**

O desenvolvimento do jogo da cobrinha foi centrado na adaptação e expansão de um código TypeScript, predominantemente contido no arquivo home.page.ts. A arquitetura da aplicação seguiu o padrão de componentes do Angular, aproveitando os recursos do Ionic para a interface do usuário e interações mobile.

**2.1. Estrutura e Tecnologias Base**

* **Frameworks:** Ionic e Angular foram as tecnologias centrais, permitindo o desenvolvimento de uma aplicação híbrida.
* **Linguagem:** TypeScript.
* **Plugins e Módulos:**
  + @ionic/angular para componentes de UI e navegação.
  + @awesome-cordova-plugins/vibration/ngx para feedback tátil (vibração).
  + @capacitor-community/admob para integração de anúncios (monetização).
  + GestureController do Ionic para manipulação de gestos de toque.

**2.2. Lógica Central e Mecânicas do Jogo**

A lógica principal do jogo foi encapsulada e inicializada através da função gInit(). Esta função estabelece as entidades fundamentais e as regras do jogo:

* **Entidades do Jogo:**
  + SnakeTile: Representa cada segmento da cobra.
  + FoodTile: Representa o alimento que a cobra deve consumir.
  + BoardTile: Representa as células do tabuleiro.
* **Estrutura do Tabuleiro:** Um Grid (THAT.g.Grid) foi utilizado para gerenciar o estado de cada célula do tabuleiro, identificando se está vazia, ocupada pela cobra ou pelo alimento.
* **Renderização:** O jogo é renderizado em um elemento <canvas> HTML, permitindo atualizações visuais dinâmicas e animações. A função requestAnimationFrame é utilizada para otimizar o loop de renderização (THAT.g.step).
* **Estados do Jogo:** A variável isPlaying controla o fluxo do jogo, alternando entre:
  + 0: Menu Principal/Inicial.
  + 1: Jogo em Andamento.
  + 2: Tela de Fim de Jogo.
  + 3: Tela de Configurações. Funções como startGame(), endGame(), showMenu(), e showSettings() gerenciam as transições entre esses estados.

**2.3. Controles e Interação**

Foram implementados e adaptados dois esquemas de controle para o jogo:

* **Controles por Gestos (Mobile):** Utilizando o GestureController do Ionic, o jogo captura gestos de deslizar (swipe) nas direções horizontal e vertical. A função onMove(detail) processa os deltas (deltaX, deltaY) para determinar a direção do movimento da cobra (buttonPressUp(), buttonPressDown(), buttonPressLeft(), buttonPressRight()). Esta é a principal forma de interação em dispositivos móveis.
* **Controles por Teclado (Desenvolvimento):** Para facilitar os testes em ambiente de desktop, o jogo também responde a comandos do teclado (teclas W, A, S, D e setas direcionais). O evento keydown é capturado e processado para acionar as mesmas funções de movimento.

**2.4. Sistema de Pontuação e Progressão**

* A pontuação (score) é incrementada cada vez que a cobra consome um FoodTile.
* A velocidade da cobra (updateTickMax) aumenta gradualmente conforme a pontuação sobe, tornando o jogo progressivamente mais desafiador.
* A pontuação é exibida dinamicamente na interface do usuário.

**2.5. Configurações do Usuário e Persistência**

O jogo permite ao usuário personalizar diversas opções, que são salvas no localStorage do dispositivo para persistência entre sessões:

* **Vibração:** Chave VIBRATION\_KEY.
* **Controle por Gestos:** Chave GESTURE\_CONTROL\_KEY.
* **Exibição da Tela de Instruções:** Chave HOWTO\_KEY. A função changeSettings(type) é responsável por atualizar essas preferências.

**2.6. Feedback Tátil**

Para melhorar a experiência do usuário, a vibração foi adicionada no android:

**Vibração:** O plugin @awesome-cordova-plugins/vibration/ngx é utilizado para fornecer feedback tátil (doVibrate()) em eventos como marcar pontos ou ao final do jogo.

**2.7. Interface do Usuário e Elementos Visuais**

* **Animações Lottie:** Para telas de "Fim de Jogo" (endgame.json) e "Como Jogar" (swipe-right.json), foram utilizadas animações Lottie, adicionando um apelo visual moderno.
* **Tela de Instruções:** Uma tela inicial (howToScreen) guia o novo jogador sobre como interagir com o jogo.
* **Layout Responsivo:** A lógica de resize() dentro do StatePlay ajusta as dimensões do tabuleiro e dos elementos do jogo com base no tamanho da tela do dispositivo.

**2.8. Integração com AdMob**

O código inclui a inicialização e exibição de anúncios através do AdMob (AdMobInitialize() e AdMob.showBanner()), visando uma possível monetização do aplicativo. O sistema também lida com a solicitação de autorização de rastreamento (trackingAuthorizationStatus()), uma exigência para plataformas como iOS.

**2.9. Testes e Adaptação do Código Base**

Uma parte significativa do trabalho consistiu em entender a estrutura do código base fornecido, que implementava a lógica fundamental do jogo da cobrinha. Os testes foram realizados continuamente, tanto em navegadores desktop quanto em tentativas de deploy para dispositivos móveis. A adaptação envolveu:

* Integrar a lógica do jogo com os componentes e ciclo de vida do Ionic/Angular.
* Mapear os controles de teclado existentes para gestos de toque.
* Ajustar a renderização e o layout para diferentes tamanhos de tela.
* Implementar a persistência de configurações e o feedback específico para mobile (vibração).

**3. Dificuldades Encontradas**

Durante o desenvolvimento do projeto, a equipe enfrentou desafios técnicos significativos que impactaram o cronograma e a complexidade do trabalho.

**3.1. Exigência Computacional do Android Studio**

A principal barreira técnica foi a utilização do Android Studio para compilação, emulação e debugging da aplicação para a plataforma Android. O ambiente de desenvolvimento integrado (IDE) demonstrou uma alta exigência de recursos computacionais (CPU, RAM), o que resultou em:

* **Lentidão:** Processos de compilação (build) e deploy para emuladores ou dispositivos físicos eram demorados.
* **Instabilidade:** Em máquinas com configurações mais modestas, o Android Studio apresentava travamentos e lentidão excessiva, dificultando o fluxo de trabalho.
* **Consumo de Tempo:** Uma parcela considerável do tempo que poderia ser dedicado ao desenvolvimento de funcionalidades do jogo foi, na verdade, consumida esperando processos do Android Studio ou solucionando problemas relacionados ao próprio ambiente.

**3.2. Desafios no Deploy para Dispositivos Móveis (Android)**

A transição do aplicativo, que funcionava corretamente em navegadores desktop (servidor de desenvolvimento do Ionic), para um arquivo .apk instalável em dispositivos Android revelou-se uma fonte de grandes dificuldades:

* **Incompatibilidades de Frameworks e Plugins:** Apesar da promessa de desenvolvimento multiplataforma do Ionic/Capacitor, foram encontradas diversas incompatibilidades entre os frameworks (Ionic/Angular), os plugins Cordova/Capacitor e o ambiente nativo do Android. Essas incompatibilidades se manifestavam de formas variadas:
  + **Problema da "Tela Cinza" (Gray Screen):** Este foi o problema mais recorrente e frustrante. Após a instalação do .apk, o aplicativo iniciava, mas exibia apenas uma tela cinza, sem nenhum conteúdo da interface do usuário. Isso indicava falhas na inicialização da WebView, no carregamento dos componentes Angular, ou conflitos com plugins nativos. A depuração desse tipo de erro é complexa, pois muitas vezes não são gerados logs claros no console do Android Studio.
  + **Falhas em Plugins Nativos:** Plugins como o de vibração ou AdMob, por vezes, não funcionavam como esperado no dispositivo, embora não apresentassem erros durante a compilação.
  + **Performance:** Em alguns casos, mesmo quando a interface carregava, a performance no dispositivo era inferior à observada no navegador.
* **Complexidade na Depuração Mobile:** Identificar a causa raiz dos problemas no ambiente mobile foi significativamente mais complexo do que no desenvolvimento web tradicional. Ferramentas como o adb logcat e o inspetor de WebViews do Chrome, embora úteis, nem sempre forneciam informações diretas sobre a origem dos erros de incompatibilidade.

**3.3. Aumento da Complexidade e do Volume de Código**

As tentativas de solucionar os problemas de compatibilidade e os desafios impostos pelo Android Studio levaram a um aumento na complexidade geral do projeto:

* **Código Adicional:** Foram necessárias diversas configurações específicas para a plataforma Android, ajustes em arquivos de manifesto, build.gradle, e, em alguns casos, a adição de código para contornar problemas específicos de plugins ou do comportamento da WebView.
* **Projeto Pesado:** A inclusão de múltiplas dependências e as configurações para o build nativo contribuíram para um projeto final com um volume considerável de arquivos e um .apk de tamanho relativamente grande para um jogo simples.

**3.4. Alocação de Tempo de Desenvolvimento**

Como consequência direta das dificuldades mencionadas, a maior parte do tempo de desenvolvimento não foi dedicada à implementação de novas funcionalidades ou ao refinamento da lógica do jogo, mas sim à configuração do ambiente de build para Android, à depuração de problemas específicos da plataforma e à busca por soluções para as incompatibilidades encontradas. Isso representou um desvio significativo do planejamento inicial.

**4. Conclusão**

O desenvolvimento do jogo da cobrinha utilizando Ionic e Angular proporcionou uma experiência de aprendizado prática e desafiadora. A equipe conseguiu adaptar e implementar o código base fornecido, resultando em um jogo mobile com as funcionalidades essenciais, incluindo controles por gestos, sistema de pontuação, configurações personalizáveis, feedback audiovisual e tátil, e integração com anúncios AdMob.

No entanto, o projeto foi intensamente marcado pelas dificuldades técnicas relacionadas ao uso do Android Studio e, principalmente, aos desafios de compatibilidade e execução do aplicativo em dispositivos móveis Android. A alta demanda de recursos do Android Studio e os problemas como a "tela cinza" consumiram uma parcela substancial do tempo e esforço da equipe, levando a um projeto mais pesado e com mais linhas de código do que o idealizado inicialmente.

Apesar dos obstáculos, a equipe demonstrou resiliência e capacidade de solucionar problemas complexos. A experiência adquirida no processo de depuração de aplicações híbridas, na configuração de ambientes de desenvolvimento mobile e na lida com incompatibilidades de frameworks é de grande valia para futuros projetos. O projeto, embora com percalços, atingiu o objetivo de entregar uma versão funcional do jogo, evidenciando o potencial das tecnologias híbridas, mas também expondo suas complexidades inerentes quando se trata do deploy e performance em plataformas nativas.

Link do projeto: https://github.com/sayCain/Jogo-da-cobrinha-android