Universidade Federal do Rio de Janeiro

Escola Politécnica

Departamento de Eletrônica e de Computação

Educational Space Invaders

Autor:

Kauli Rigoni Dias Gutierrez

Orientador:

Prof. Sérgio Barbosa Villas-Boas, Ph.D.

Examinador:

Prof. Antônio Cláudio Gómez de Sousa, Dr.

Examinador:

Profa. Jonice de Oliveira Sampaio, D.Sc.

DEL

Setembro de 2013

UNIVERSIDADE FEDERAL DO RIO DE JANEIRO

Escola Politécnica – Departamento de Eletrônica e de Computação

Centro de Tecnologia, bloco H, sala H-217, Cidade Universitária

Rio de Janeiro – RJ CEP 21949-900

AGRADECIMENTO

Dedico este trabalho a todos os professores que contribuíram de forma significativa à minha formação e à minha família. Este projeto é uma pequena forma de retribuir o investimento e confiança em mim depositados.

Agradeço também à orientação do Professor Sérgio Villas-Boas, que me ajudou a conduzir este projeto com muita competência e paciência.

RESUMO

Projeto de Graduação apresentado à Escola Politécnica/COPPE/UFRJ como requisito necessário para a obtenção do grau de Engenheiro Eletrônico e de Computação.

Este projeto consiste em um jogo educativo para Android baseado no mesmo conceito do famoso jogo *Space Invaders (1978).* A ideia principal do projeto é que diferentes temas podem ser abordados pelo jogo sem a necessidade de modificação ou atualização do software. Através do carregamento de imagens armazenadas em um servidor, diferentes versões do jogo podem ser criadas sem qualquer conhecimento em programação.

Palavras-Chave: Jogos Educativos, Space Invaders, Programação Orientada à Objetos, Smartphone, Aplicativos.

ABSTRACT

Undergraduate Project presented to POLI/COPPE/UFRJ as a partial fulfillment of requirements for the degree of Electronics and Computer Engineer.

This project consists of an educational game for Android based on the same concept of the famous game Space Invaders (1978). The main idea of the project is that different issues can be addressed by the game without the need to modify or update the software. By loading images stored on a server, different versions of the game can be created without any programming knowledge.

Key-words: Educational Game, Space Invaders, Object Oriented Programming, Smartphone, Applications.

SIGLAS

ESI – Educational Space Invaders

FTP – File Transfer Protocol

FPS – Frames per second

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

1.1 – Tema

O Educational Space Invaders é um projeto que pretende ser um aplicativo educacional para Smartphones. Este trabalho abordará temas como programação orientada à objetos através da linguagem java e desenvolvimento de jogos para mobile.

1.2 – Delimitação

Normalmente, os jogos educacionais são desenvolvidos visando desenvolver alguma habilidade específica do usuário ou expandir conhecimentos sobre algum conceito. Quando uma ferramenta educacional é utilizada inúmeras vezes e todo conhecimento é absorvido pelo jogador, o jogo se torna obsoleto para o usuário. Muitos jogos são desenvolvidos em diversas áreas da educação, mas muitos possuem temas limitados e assuntos muito específicos. Este projeto propõe um modelo de jogo educacional onde diferentes áreas da educação poderão ser opções de temas a serem explorados.

Games educacionais são uma poderosa ferramenta para aprendizado. Psicólogos e educadores apoiam a ideia de que um aprendizado inserido no mundo dos games é muito mais atrativo do que o aprendizado convencional. Além disso, o conceito de personalização de jogos eletrônicos pelo próprio usuário é cada vez mais comum. Inúmeros jogos permitem que os jogadores criem seus próprios níveis, fazendo com que os desafios sejam infinitos. Com a facilidade de desenvolvimento e publicação de aplicativos e jogos para celulares e tablets, o Educational Space Invaders pode ser amplamente acessado. Este projeto pretende ser uma ferramenta geral de ensino para diversas áreas da educação.

1.3 – Justificativa

Apesar do mercado de jogos para smartphones ter crescido bastante nos últimos anos, os jogos educativos ainda têm pouco espaço neste mercado. Além disso, a prática de programação de jogos com a obtenção de valores de funções provenientes de um recurso externo ainda é pouco explorada. Este projeto é mais uma alternativa aos métodos tradicionais de aprendizado. Não pretende substituí-los, mas sim, complementá-los. Pode fornecer a oportunidade do aprendizado ter lugar fora da sala de aula, em um ambiente livre de pressões e bem mais atrativo para crianças e jovens.

1.4 – Objetivos

O objetivo é desenvolver um jogo para Smartphone, educativo, que segue um conceito parecido com o do famoso jogo *Space Invaders*. Com a temática livre, este aplicativo será capaz de listar e baixar arquivos compactados contendo bitmaps de um servidor da web, que posteriormente serão carregados como as imagens dos componentes do jogo. Sem conhecimento de qualquer linguagem de programação, será possível armazenar inúmeras versões do jogo no servidor somente manipulando os arquivos contendo as imagens selecionadas. Apesar da lógica do jogo ser definitiva, cada usuário poderá ser capaz de personalizar os desafios através do conteúdo das imagens carregadas.

1.5 – Metodologia e Materiais

A metodologia utilizada para o desenvolvimento do projeto será a *Scrum Academic*, uma versão acadêmica do Scrum, onde são feitas entrevistas com o cliente conforme o projeto avança. O cliente será representado pelo orientador deste projeto, o professor Sérgio Villas-Boas. O aplicativo será desenvolvido através do IDE Eclipse, seguindo as orientações e procurando alcançar o objetivo traçado em cada reunião. O kit de desenvolvimento para Android do Eclipse possui todas as bibliotecas necessárias para o desenvolvimento de aplicações para Smartphones e é possível realizar testes em emuladores e em mobiles reais.

O aplicativo será elaborado por meio de linguagem java e seu progresso seguirá um roteiro típico de desenvolvimento de jogos eletrônicos através da programação orientada a objetos.

A lógica do jogo, assim como o famoso *Space Invaders*, consiste em um tanque na base da tela que atira tentando evitar que os alienígenas se aproximem da terra. A figura 1.1 é uma captura de tela do jogo Space Invaders original.

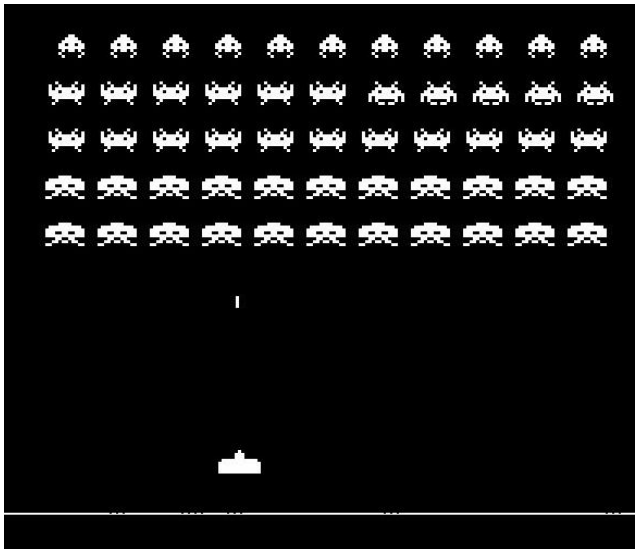

Figura 1.1: O jogo Space Invaders(1978) original. Fonte: pcguia.sapo.pt

A diferença conceitual entre este projeto e o Space Invaders é que o nosso tanque terá três opções de tiros para acertar em três tipos diferentes de naves. Além disso, o tema do jogo não será limitado à alienígenas e tanques de guerra. Os alienenígenas e os botões de tiros poderão ser, respectivamente, desafios (perguntas, associações, charadas) e respectivas respostas de tema livre.

A diversidade de assuntos será possível graças às diferentes versões do mesmo jogo armazenadas em um servidor. Este será acessado pelo aplicativo através de uma conexão FTP.

Este trabalho, primeiramente, foi utilizado como o projeto de conclusão da disciplina "Software para Smartphones" e tinha como objetivo somente a criação do jogo simplesmente usando as imagens salvas como recurso do aplicativo. O acesso ao servidor foi adicionado posteriormente, permitindo a variedade da temática do Educational Space Invaders (ESI).

1.6 – Descrição

O capítulo 2 faz uma breve introdução do conceito de programação a objetos voltada para jogos e explica a idéia do Educational Space Invaders.

O capítulo 3 fala sobre o funcionamento de alguns recursos da programação voltada para aplicativos de Smartphones e explica como foi feita a tela de menu principal.

O capítulo 4 disserta sobre a Arquitetura da tela de jogo e como foram desenvolvidos todos os mecanismos que fazem o jogo funcionar.

O capítulo 5 explica como é realizada a conexão com o servidor para a realização do download das diferentes versões do jogo, falando, também, da tela de download da aplicação.

O capítulo 6 aborda o carregamento dessas imagens e a tela de seleção de jogo.

O capítulo 7 somente apresenta alguns exemplos de jogos criados.

No Capítulo 8, temos a conclusão do trabalho.

Capítulo 2

Programação para Jogos

"O que exatamente é um objeto em termos de Programação Orientada a Objetos? Bom, literalmente pode ser qualquer coisa. Em um jogo, podemos ter um objeto para um partícula - digamos, uma faísca emitida por uma explosão - ou o tanque que causou a explosão. Na verdade, o jogo inteiro pode ser um objeto. O propósito do objeto é conter a informação e dar ao programador a habilidade de manipular essa informação."[1]

2.1 – O Conceito da Modelagem Através de Classes

Este projeto foi totalmente desenvolvido em java, através da programação orientada a objetos. Isso nos dá a vantagem de modelar o jogo inteiro através de classes e objetos. Entidades podem ser criadas reutilizando código e animações e movimentos podem ser modelados através de métodos.

Todos os elementos do jogo são objetos pertencentes às classes criadas ou já existentes na biblioteca java ou do kit de desenvolvimento do Android. Além disso, toda ação que cada objeto realiza - modificando atributos dele próprio ou de outros objetos – pode ser realizada através de seus métodos.

Em outras palavras, as classes são um meio de definir objetos. Podemos pensar em uma classe como uma espécie de modelo para objetos. Elaboramos a classe apenas uma vez e podemos criar quantos objetos for preciso.

2.2 – O Educational Space Invaders

Quando elaboramos um jogo educativo, devemos levar em conta dois aspectos. O jogo deve aprimorar algum conhecimento do usuário ou estimular a sua criatividade.

Além disso, o jogo precisa ser atrativo e divertido. Assuntos abordados em sala de aula podem ser vistos em um ambiente livre de pressões, como o mundo dos jogos.

A ideia do Educational Space Invaders é, essencialmente, fazer com que o usuário possa criar o seu próprio jogo escolhendo apenas as imagens de fundo, dos botões, das naves, etc. No Educational Space Invaders, a missão de criar os desafios não cabe somente ao programador, mas também, ao usuário.

A figura 2.1 é uma captura de tela de uma das versões do jogo, criada apenas com o intuito de apresentar como o jogo funciona. Chamaremos esta versão do jogo, por simplicidade, de "Default" e ela pode ser encontrada no servidor disponível para download.

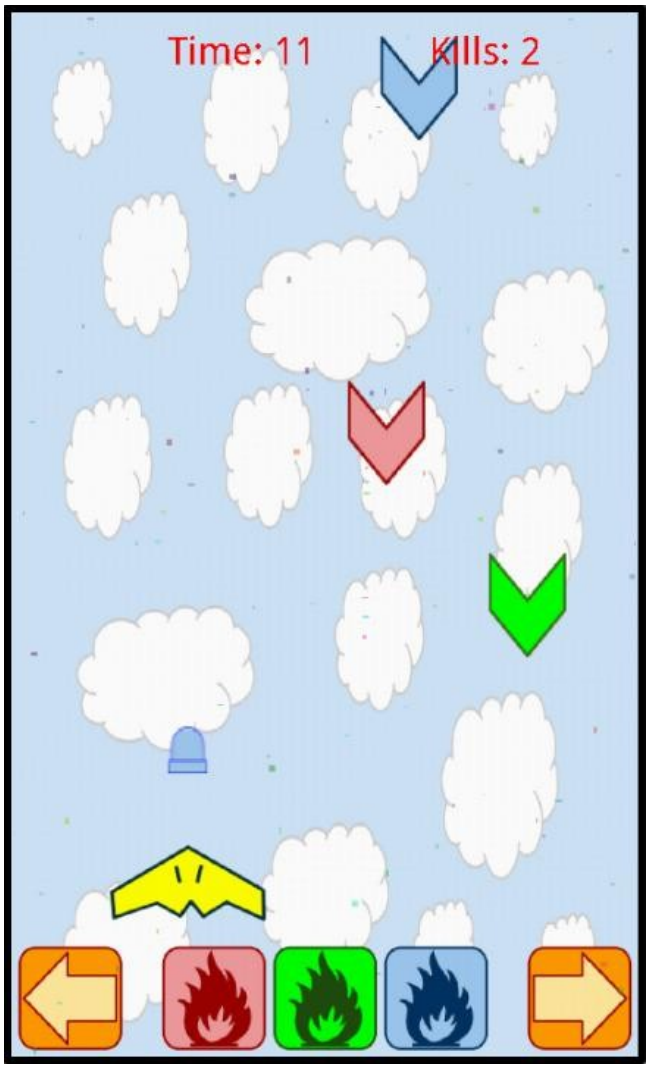

Figura 2.1: O jogo Default.

Esta versão é simples. Inimigos de três cores diferentes aparecem no topo da tela e descem tentando alcançar a base. Por conveniência, vamos nos referir a estes inimigos

como “naves”. O jogador controla este bombardeiro amarelo que se movimenta na vertical que chamaremos de “canhão”. Este canhão atira, mas somente os tiros das cores certas destroem as naves certas. Note que há três botões de tiro na base da tela: tiros vermelhos, verdes e azuis.

O ESI pode ser qualquer coisa além de associações de cores. Com a liberdade de escolher as imagens de fundo e os bitmaps dos elementos do jogo, qualquer tema poderá ser abordado.

Capítulo 3

Desenvolvimento de Aplicativos

3.1 – Activities e Views

Para o desenvolvimento de aplicativos em Android, é necessário o entendimento de algumas classes importantes utilizadas. Primeiramente, precisamos entender o conceito de Activities e Views.

Uma Activity é uma componente chave no Android e está relacionada ao que o aplicativo pode fazer. É, basicamente, uma classe gerenciadora de UI (interface com o usuário). Esta é a classe encarregada de criar uma janela – flutuante ou full-screen – na qual é apresentada a interface editada através da classe View. Cada tela do aplicativo (tela de menu, tela do jogo, tela de download, etc.) representa uma Activity criada. Neste projeto, quatro classes herdadas de Activity foram criadas. MenuActivity.java, DownloadGameActivity, ShowDescriptionActivity, GameActivity.

Tudo em Android acontece dentro de uma Activity. A Activity é a responsável por criar uma View. A View representa uma porção retangular da janela gerenciada pela activity e nela é onde tudo acontece. É onde o toque se realiza e onde a imagem resultante é exibida. É através desta classe que a interface com o usuário e o layout da tela serão construídos. Este layout pode ser desenvolvido através de um arquivo editado na forma .xml ou criando uma classe totalmente nova herdada da classe View. Nesta classe está definida, por exemplo, que partes da tela são sensíveis ao toque ou que elementos visuais serão mostrados ou desenhados na tela.

3.2 - As Activities do Aplicativo

Quando criamos um jogo ou outro aplicativo, normalmente precisamos de mais de uma Activity. Cada tela do aplicativo é representada por uma Activity. Neste projeto, por exemplo, foram criadas mais três Activities, além da tela principal do jogo. Foi

necessária a criação de telas para o menu principal, download e pré-jogo. A tabela 3.1 descreve a função de cada Activity criada para este aplicativo.

Tabela 3.1 - Descrição de cada Activity utilizada no aplicativo.

Nome da Activity	Descrição
MenuActivity	É a tela de menu inicial. É ativada quando o aplicativo é iniciado.
DownloadGameActivity	Representa a tela onde são listados os jogos disponíveis no servidor. O download de um dos jogos é iniciado ao escolhermos um dos itens da lista.
ChooseGameActivity	Esta Activity é referente à tela "pré-jogo". Nela são listados os jogos já baixados e disponíveis na memória do aparelho. O jogo é iniciado ao ser escolhido um dos itens da lista.
GameActivity	Representa a tela de jogo.

3.3 – A MenuActivity

Quando abrimos o aplicativo, a MenuActivity é acionada e a View correspondente (a MenuView) é exibida na tela. Esta Activity representa, basicamente, uma tela de acesso às outras Activities através da utilização dos botões adicionados à View. Ela também serve como uma tela de apresentação onde é exibida o título do jogo. Os botões foram criados utilizando a classe MyButton, que será explicada mais detalhadamente no capítulo 4.

O botão “Download a Game”, ao ser pressionado, faz o aplicativo abrir a tela onde podemos baixar os jogos do servidor. Os jogos disponíveis no servidor são listados na tela e quando um deles é selecionado, o aplicativo realiza os procedimentos necessários para o download do jogo e, em seguida, volta para a tela de menu principal.

O botão “Start a Game” é selecionado quando queremos começar um dos jogos já baixados. Novamente será apresentada uma lista, desta vez de jogos já baixados (sem a necessidade de conexão com a internet) onde o usuário poderá escolher qual jogo iniciar. Todas estas outras telas serão explicadas com mais detalhes nos próximos capítulos.

O botão “Quit” finaliza o aplicativo. Na figura 3.1, podemos ver como o layout do menu principal.

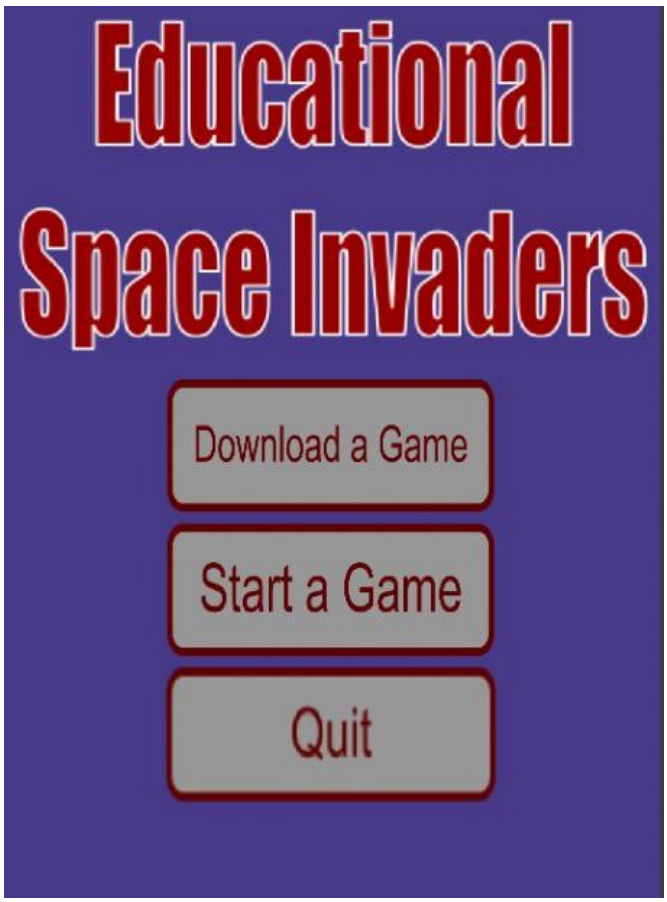

Figura 3.1: A tela de menu principal do ESI

3.4 – Comunicação entre Activities

Podemos falar das Activities de forma isolada, já que o desenvolvimento de cada uma é independente. Porém, as Activities se comunicam entre si quando, por exemplo, usamos uma Activity para abrir outra. Existe um padrão de comunicação entre as Activities, utilizada neste projeto, usando a classe Intent do Android.

Um objeto da classe Intent é uma descrição abstrata de uma operação a ser realizada. São determinados comandos que podemos enviar ao Sistema Operacional Android para realizar alguma ação, com Intents podemos iniciar Activities e trocar dados entre as elas.

Quando trocamos de tela, o que estamos fazendo, na verdade, é usando um método startActivity(Intent intent) na Activity atual para enviar dados de requisições da

tela que queremos abrir. Podemos abrir uma outra Activity enviando informações da Activity atual utilizando o método putExtra("nome_de_identificação", valor). Este método insere na intent algum dado que possa ser identificado e recuperado pela outra Activity. O primeiro parâmetro deste método é um nome para que o dado possa ser identificado mais tarde e o segundo parâmetro um valor que pode ser String, boolean, int, float, etc. A Activity de destino chama um outro método para recuperar os dados inseridos na intent, utilizando o mesmo identificador usado como parâmetro.

A figura 3.2 mostra como podemos navegar entre as telas do aplicativo. Ou seja, quais Activities podem se comunicar entre si através de intents.

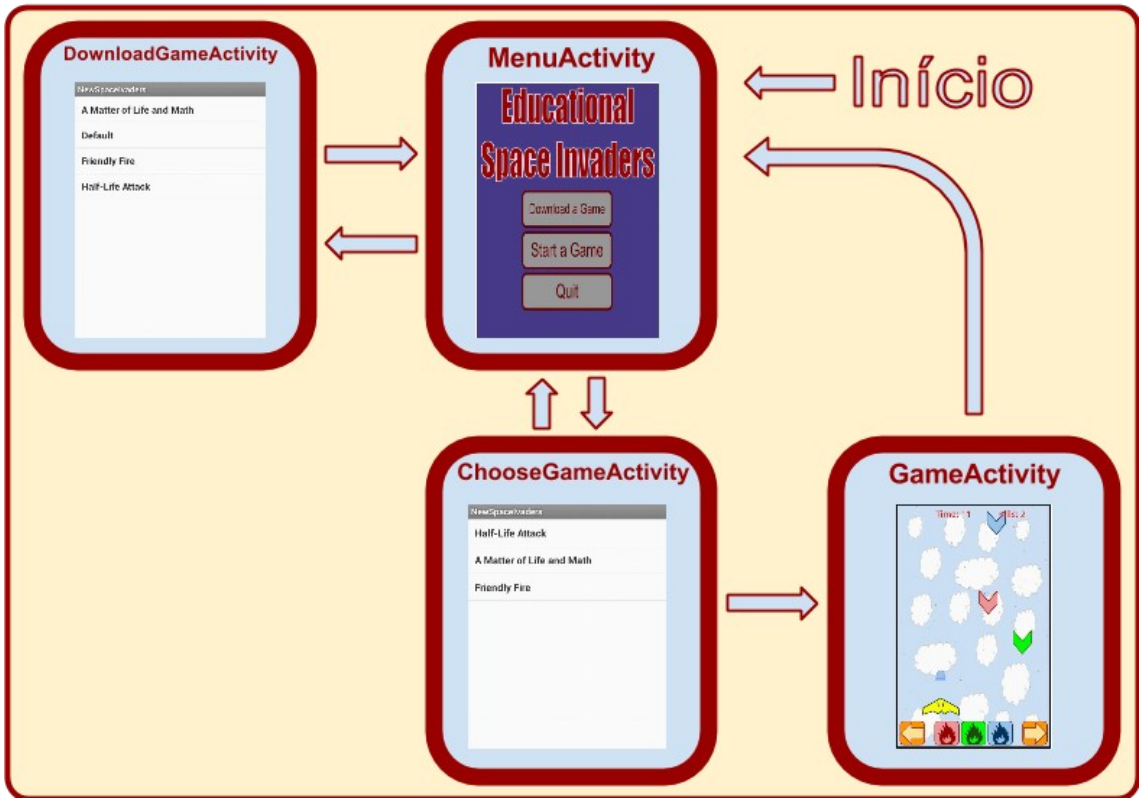

Figura 3.2: Interação entre as Activities

Capítulo 4
A Arquitetura do Jogo

4.1 – Introdução

Neste capítulo, falaremos especificamente do funcionamento da GameActivity e da GameView. Como esta Activity é a mais importante e a mais complexa, separamos um capítulo inteiro para abordarmos o assunto mais detalhadamente.

4.2 – A Criação da GameView

A GameView é a classe chave do jogo. Representa a tela onde podemos jogar o ESI. Assim que a GameActivity é acessada, a GameView é ativada . As chamadas de seus métodos através de uma thread é o que define a mecânica do jogo. A GameView é uma classe estendida de SurfaceView do Android. Esta classe pode ser definida como uma superfície onde é desenhada toda a parte gráfica do jogo. A classe também implementará SurfaceHolder.Callback para ganharmos acesso a alterações da superfície quando, por exemplo, esta é destruída ou a orientação do dispositivo for alterada. Também precisamos bloquear a superfície quando o método onDraw (que será explicado mais adiante) desenha na tela e isso somente pode ser feito através da surfaceHolder.

4.3 – A Lógica Através dos Métodos da GameView

4.3.1 – O Loop Básico do Jogo

O loop básico do jogo segue uma mecênica ilustrada na Figura 4.1. Todos os métodos relacionados à mudança de estados, gerenciamento da entrada do usuário e

atualização de gráficos são fornecidos pela GameView e as chamadas desses métodos são mediadas por uma thread.

Uma thread é uma unidade de execução paralela. Ele tem sua própria pilha de chamadas a métodos, seus argumentos e variáveis locais. Cada aplicação tem pelo menos uma thread rodando quando é iniciada. No caso deste jogo, foi criada uma classe estendida de Thread (a MainThread) para lidar com a execução do jogo. O passo do loop do jogo é dado pelo método run( ) da thread. A figura 4.1 exibe um modelo superficial do que acontece no loop do jogo.

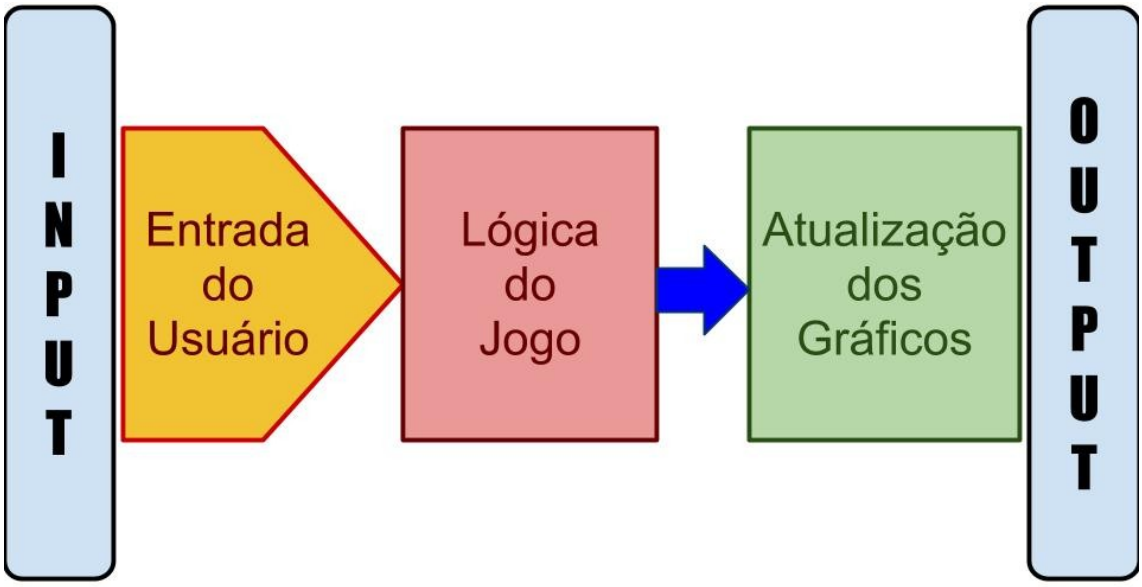

Figura 4.1: O loop do jogo.

Os principais métodos da GameView serão explicados a seguir. Estes métodos correspondem aos mecanismos apresentados na Figura 4.1. Cada um possui um bloco de execuções que definirá os estados do jogo.

4.3.2 – surfaceCreated( )

Este método é criado automaticamente quando implementamos a SurfaceHolder.Callback. No método surfaceCreated é onde definimos alguns parâmetros iniciais de cada objeto do jogo e inicializamos a thread. Este método é o responsável por definir qual é o estado inicial do jogo. É chamado no momento em que a superfície já tiver sido criada e, portanto, o loop do jogo pode ser iniciado de forma segura.

4.3.3 – surfaceDestroyed( )

Este método é chamado imediatamente antes da View ser destruída. Neste bloco, colocamos o código que garante que o mecanismo que gerencia o loop será desligado corretamente. Quando isto acontece, a thread é simplesmente bloqueada e morre.

4.3.4 – update( )

Este é o módulo de lógica do jogo. É o responsável por definir como alterar os estados de todos os componentes a cada loop da thread. Cada objeto possui atributos e métodos que definem o estado atual e como estes estados serão modificados. Por exemplo, o canhão controlado pelo jogador possui atributos como posição e velocidade. Ao tocarmos nos botões de deslocamento, estes atributos são alterados através das chamadas aos métodos utilizados para modificar estes valores.

4.3.5 - onDraw( )

Este é o método encarregado de processar o estado do jogo e exibi-lo na tela. O método onDraw é o responsável pelo que será desenhado na tela. Este módulo, a princípio, tem a função de desenhar um frame a cada ciclo do método run da MainThread.

Para medirmos a velocidade com que os frames aparecem na tela, usamos a unidade FPS, que significa frames por segundo. Se tivermos 30 FPS, significa que exibimos 30 imagens por segundo. Portanto, quanto maior o FPS mais suave será a animação. O funcionamento dos mecanismos de controle de frames será explicado mais adiante.

4.3.6 – onTouchEvent( )

Neste bloco, colocamos as funções responsáveis por alterar o estado do jogo em função da entrada do usuário. Em Android, temos eventos do tipo *MotionEvent*, que são gerados pelo toque na tela em uma das áreas sensíveis definidas. A GameView é quem monitora estes eventos e as coordenadas de cada toque são passadas como parâmetro para o método onTouchEvent. Se as coordenadas estiverem dentro das áreas de controle definidas (aqui, entende-se como botões), alguma funcionalidade entrará em ação.

Todos os botões do jogo possuem métodos para lidar com o toque, atributos que

definem o tamanho da área ocupada na tela e sua posição. Por exemplo, se o toque ocorre na área do botão designada para mover o canhão para a direita, o mecanismo do jogo é notificado e o canhão é instruído a mover-se.

4.4 – Construtor x surfaceCreated

Algumas variáveis são criadas no construtor da GameView, outras, no método surfaceCreated. Ambos são chamados assim que a GameView é chamada, mas alguns atributos iniciais não podem ser definidos no construtor, mas sim, na surfaceCreated que somente é chamada quando a superfície é completamente criada.

Por exemplo, todas as imagens que o aplicativo utiliza são gravadas em variáveis do tipo Bitmap. Ao serem simplesmente carregadas para variáveis e, posteriormente, exibidas na tela do celular, as imagens mantém o seu tamanho original e seu numero de pixels. Isso se torna um problema uma vez que cada Smartphone possui um tamanho de tela diferente e as dimensões e o posicionamento das imagens devem ser proporcionais ao tamanho da tela.

Este problema pode ser contornado utilizando métodos da SurfaceView que fornecem as dimensões da tela dentro do método surfaceCreated. As dimensões da superfície criada podem ser adquiridas simplesmente chamando os métodos getHeight( ) e getWidth( ). Os objetos são criados com as imagens no seu tamanho original no construtor e redimensionados no método surfaceCreated. Como as dimensões da tela ainda não estão disponíveis no construtor (a superfície ainda não é totalmente criada neste ponto), os objetos só podem ser redimensionados a partir do método surfaceCreated, já que este método só é chamado quando a superfície estiver pronta.

Neste projeto, foram usadas duas maneiras de obter os bitmaps: a partir dos recursos do aplicativo e através do carregamento das imagens da memória do cartão SD. A imagens que são fixas (não mudam, independente do jogo escolhido) - como as imagens dos botões de movimento do canhão e da tela de game over – foram gravadas como recursos do aplicativo. Estas imagens são simplesmente copiadas para a pasta res do projeto e o plugin executa os scripts necessários para a criação de um identificador no arquivo R.java. Assim, conseguimos referenciar a imagem quando quisermos

carregá-la em um objeto da classe Bitmap. O R.java detém todos os identificadores de recursos.

No capítulo 6 veremos mais detalhadamente como carregamos as imagens da memória do Cartão SD para as variáveis do programa.

4.5 - O Passo do Loop

Como já foi dito anteriormente, o loop do jogo é o coração do aplicativo e seu ritmo é ditado pelo método run da MainThread. Um loop mais rudimentar possível é um loop "while"que mantém a execução de algumas instruções até que se dê o sinal para que ela pare, geralmente, definindo uma variável booleana de controle para falso.

Este tipo de loop, porém, fará com que o jogo tenha velocidades diferentes em aparelhos diferentes. Como a velocidade da execução das funções dentro de um loop while depende do desempenho do processador, as naves levarão tempos diferentes para chegar na base da tela em diferentes Smartphones, por exemplo.

A thread, a cada loop, chama os métodos update e onDraw da GameView. Ou seja, ela atualiza os estados do jogo e desenha na tela. Idealmente, os métodos de atualização e renderização são chamados o mesmo número de vezes por segundo. Uma taxa de 25 FPS é geralmente suficiente para que uma animação em um Smartphone não aparente ser lenta para nós, seres humanos.

Por exemplo, se projetarmos um jogo rodando a 25 FPS, temos que chamar o método onDraw a cada 40ms (1000/25 = 40ms, 1000ms = 1s). Precisamos ter em mente que o update também é chamado antes do método de exibição. Para chegarmos a 25 FPS, temos que ter certeza de que a sequência onDraw-update é executada em exatamente 40ms. Se demorar menos de 40ms, então temos um quadro de FPS mais alto. Se demorar mais do que isso, teremos um jogo mais lento.

A figura 4.2 mostra exatamente o que acontece em uma taxa de 1 FPS. Para criarmos um frame, é preciso executar em exatamente um segundo, um ciclo inteiro de atualização e renderização. Isso significa que você vai ver a imagem na tela mudar a cada segundo.

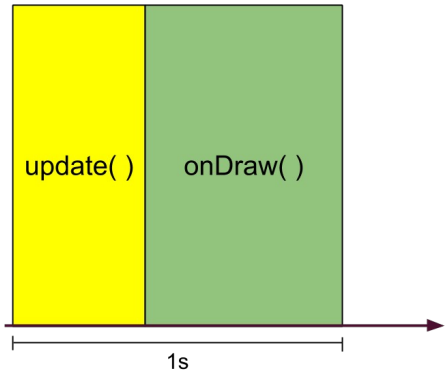

Figura 4.2: Um ciclo da thread em 1 FPS

Esta, porém, é uma situação ideal onde o tempo de execução dos métodos dura exatamente o tempo desejado. Na prática, celulares diferentes executam os mesmos métodos em tempos diferentes e podemos ter um ciclo onDraw-update com a duração mais longa ou mais curta que o desejado. Estes dois quadros são representados através dos diagramas da figura 4.3.

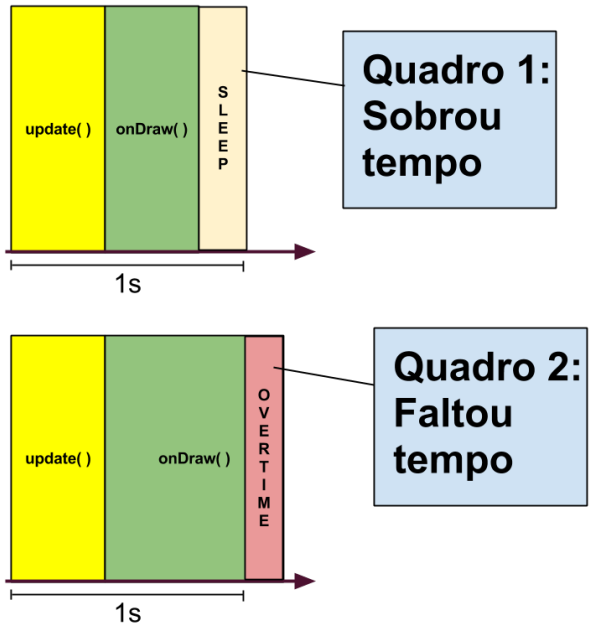

Figura 4.3: Diferentes durações de um ciclo em 1 FPS

Como podemos verificar, no quadro 1, o ciclo termina antes do prazo fixado e, por isso, temos uma pequena quantidade de tempo livre antes de executar o próximo

ciclo. Esta é uma situação desejada. Não é necessário fazer nada além de instruir o loop a dormir (método sleep(time)) durante o período restante e somente acordar quando o próximo ciclo é iniciado. Além disso, podemos economizar a energia da bateria neste tempo de sono. Se não fizermos isso, o jogo vai rodar mais rápido do que o pretendido.

A segunda situação requer uma abordagem diferente. No caso das instruções de atualização e renderização durarem mais do que um segundo, precisamos fazer uma atualização extra antes de entrar no próximo ciclo. Assim, o loop do jogo consegue alcançar o frame perdido pulando um processo de onDraw com o custo de alguns frames perdidos. Utilizando estas duas abordagens, a velocidade do jogo fica constante, independente do aparelho e da quantidade de instruções contidas nos métodos.

4.6 – Os Elementos do jogo

4.6.1 – Introdução

Agora que entendemos como funcionam os mecanismos que fazem o jogo funcionar, podemos falar especificamente do Educational Space Invaders. Falaremos do design do jogo, como cada elemento se comporta, seus atributos e como são atualizados dependendo da entrada do usuário. A figura 4.4 explica o que significa cada objeto ativo no jogo.

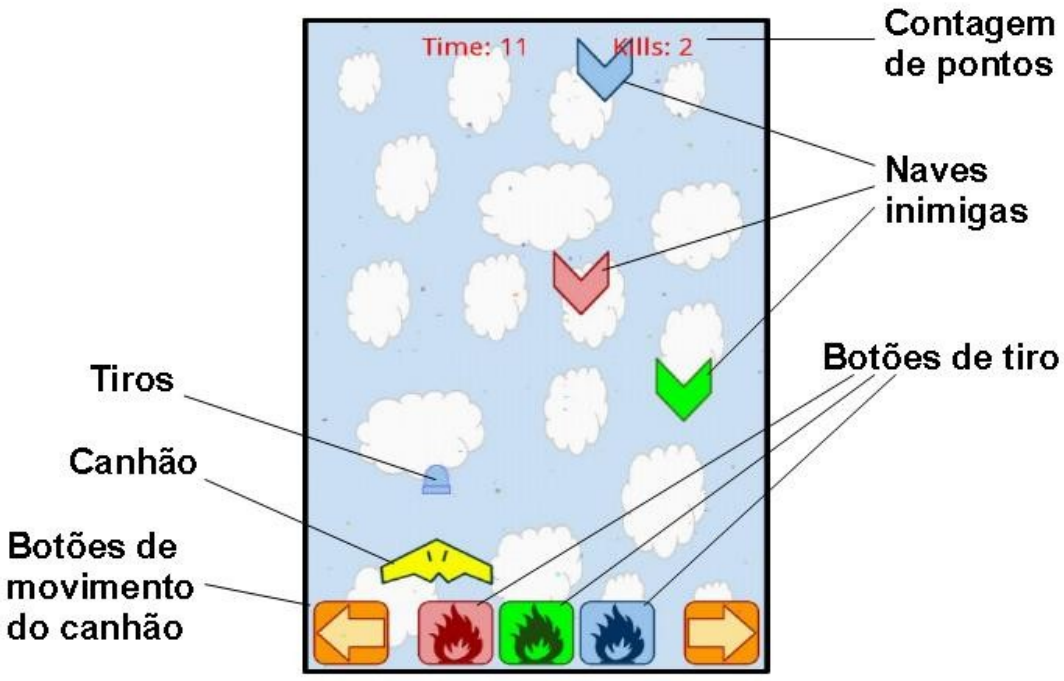

Figura 4.4: Os elementos do jogo.

Note que os botões utilizados durante o jogo foram posicionados na base da tela para facilitar a jogabilidade. Desta forma, o aparelho pode ser segurado na vertical como um mini-game antigo e os botões serem apertados com os polegares.

Para cada tipo de objeto foi criada uma classe com diferentes atributos – como posição e velocidade - que são atualizados através dos métodos chamados nos blocos de atualização e de sensibilidade ao toque. Além disso, estes elementos possuem seus próprios métodos de desenho que são chamados nos módulos de renderização. A tabela 1 mostra a descrição de cada classe usada como componente do jogo.

Tabela 4.1: As classes auxiliares.

Nome da Classe	**Descrição**
MyButton	Representa uma área sensível ao toque na tela que, ao ser pressionada, provoca uma ação.
Cannon	Na tela de jogo, é o "canhão" controlado pelo jogador que se movimenta horizontalmente na base da tela.
Ship	Esta classe representa as "naves" inimigas que o jogador deve evitar.
Fire	Representa os projéteis disparados pelo "canhão" para evitar que as "naves" cheguem na base da tela.
Explosion	Representa uma explosão de particulas criada na posição e no momento em que um projétil acerta uma nave do mesmo tipo.
Particle	Representa uma partícula da explosão de partículas.
Speed	É usada como atributo de velocidade de cada objeto animado do jogo.

Estas classes também possuem um estado booleano state, que podem assumir os valores STATE_ALIVE ou STATE_DEAD, dependendo se os objetos estão ativos ou inativos. Todos os métodos de atualização e renderização de cada objeto só são chamados quando este está no modo ativo. Cada uma dessas classes será explicada mais detalhadamente a seguir.

4.6.2 – Os Botões

Para implementar os botões, foi necessário criar uma classe em que é definida uma área na tela sensível ao toque. O construtor da classe MyButton é responsável por

isso. Todas as áreas da tela sensíveis ao toque (com exceção dos itens das listas DownloadGameActivity e ChooseGameActivity que já possuem seus próprios métodos para lidar com itens selecionáveis) são objetos do tipo MyButton e, ao serem pressionados, provocam uma ação.

No caso da GameView, cinco objetos da classe MyButton foram criados – dois para movimento da nave e três para atirar. Cada botão possui um método chamado handleActionDown. Este método faz com que, ao tocarmos uma área de tela definida através do construtor de cada botão, um atributo booleano chamado *touched* passa a ser definido como true. Os métodos de cada botão são chamados dentro do bloco do método onTouchEvent, aonda as coordenadas do toque são passadas como parâmetro. A cada loop da thread, os atributos touched de todos os objetos do tipo MyButton são verificados em cada atualização estado.

Por exemplo, se o botão que faz o canhão se mover para a direita é tocado, as coordenadas do toque são passadas para o método onTouchEvent e, dentro desse método, verifica-se se estas coordenadas correspondem a alguma área definida por algum dos botões. Quando o método handleActionDonw do botão de movimento para a direita é chamado, o seu atributo touched passa a ser definido com true. Quando é feita a atualização, este atributo é avaliado e, uma vez ativo, o canhão se move para a direita.

4.6.3 – A Classe Speed

Esta classe é usada para definir a velocidade de cada objeto que se movimenta no jogo. Ela possui dois atributos principais e úteis para um jogo 2D, o valor da velocidade na horizontal e na vertical. Esta velocidade pode ser traduzida em quantos pixels um objeto anda a cada atualização.

Devemos tomar cuidado com a quantidade de pixels que cada objeto deve andar a cada atualização. Com uma velocidade fixa, um objeto se moverá sempre a mesma quantidade de pixels a cada atualização independente do tamanho da tela e isto pode se tornar um problema.

Vamos tomar como exemplo um jogo rodando em uma tela de 500 pixels de largura. Se considerarmos um objeto (um ponto) se movimentando na horizontal de um lado da tela para o outro com uma velocidade de 5 pixels/frame, ele chegará ao outro lado da tela em 100 frames. Se considerarmos o mesmo objeto em uma tela de 200 pixels, ele fará o mesmo percurso em 40 frames.

Para contornar esse problema, cada valor de velocidade do jogo é definida em função da largura e altura da tela através dos métodos getWidth( ) e getHeight( ). Com isso, assim como o tamanho dos objetos, a velocidade também é proporcional ao tamanho da tela.

4.6.4 – O Canhão

Este é o elemento principal do jogo. É o "personagem" que controlamos e com ele evitamos a invasão de naves inimigas. Este objeto está definido pela classe Cannon e possui um altura fixa no jogo. Ou seja, só se movimenta na horizontal através dos botões de movimento que alteram o seu atributo speed e, consequentemente, a sua posição.

A largura também é importante para a detecção de quando o canhão alcança as bordas laterais. Quando a posição do canhão, depois de uma atualização se encontra fora destes limites, ele é reposicionado para a sua última posição antes de escapar da tela.

4.6.5 – Os Tiros

Quando os botões de tiro são acionados, um objeto do tipo Fire é criado e posicionado exatamente onde se encontra o canhão no momento do tiro. A classe Fire representa os projéteis atirados pelo canhão. Na verdade, um vetor de objetos do tipo Fire (vetor *fireArray* de tamanho 15) foi criado no construtor da GameView para termos a possibilidade de dar mais de um tiro.

Primeiramente, este vetor é totalmente preenchido com null (situação em que nenhum tiro é disparado). Quando um dos botões de tiro é selecionado, o método onTouchEvent entra em ação e um objeto do tipo Fire é criado e alocado neste vetor. Ao toque de cada botão de tiro, o vetor de tiros é percorrido e o novo objeto é alocado na primeira posição em que for encontrado "null" ou outro objeto inativo, substituindo-o (por exemplo, um tiro que já acertou um inimigo ou saiu da tela sem acertar ninguém).

Todos objetos da classe Fire possuem o mesmo valor de velocidade na vertical de baixo pra cima e não possuem componentes de velocidade na horizontal. Também possuem um atributo referente a qual dos três tipos de tiro o objeto pertence. O tipo de tiro será correspondente a qual dos três botões de tiro será pressionado. A imagem carregada para representar o tiro na tela dependerá do tipo selecionado.

4.6.6 – As Naves Inimigas

Estes são os inimigos os quais, acertando os tiros corretos, devemos evitar que cheguem na base da tela. São as charadas sem resposta ou associações desassociadas. Assim como o vetor de tiros, um vetor de naves (shipWave de tamanho 10) também foi criado. O preenchimento deste vetor com objetos da classe Ship é análogo ao do vetor de tiros, com a diferença de que novas naves são criadas cada vez que a nave anterior alcança uma certa altura da tela. Surgindo do topo da tela, estes objetos da classe Ship representam a parte imprevisível do jogo, uma vez que alguns de seus atributos são gerados aleatoriamente.

Todos o valores inteiros aleatórios do aplicativo são gerados a partir da manipulação do método nextInt(int) da classe SecureRandom. Este método gera um número inteiro aleatório que pode assumir os valores entre 0 e o valor passado como parâmetro para a função.

O primeiro atributo aleatório é a posição X em que as naves aparecem. As naves podem aparecer em qualquer ponto do topo da tela contanto que estejam dentro dos limites da tela e com o cuidado para que possa ser acertada com um tiro.

O segundo atributo gerado aleatoriamente é o tipo de nave que aparece. Assim como os tiros, as imagens carregadas para cada objeto dependerão deste atributo e somente um tiro de mesmo tipo conseguirá destruir uma nave. No exemplo do jogo Default, as naves vermelhas, verdes e azuis aparecem aleatoriamente.

O terceiro atributo aleatório é qual das imagens será carregada, uma vez definido o tipo. Para cada tipo de Ship, uma ou mais imagens correspondentes são usadas para representar cada nave. No jogo Default, não podemos verificar isso, mas podemos utilizar o exemplo de um jogo hipotético em que os três tipos de nave podem ser mamíferos, anfíbios ou répteis. Se a nave criada for do tipo mamífero, a figura poderá ser escolhida aleatoriamente entre as fotografias de um elefante, um leão ou um morcego, por exemplo, e somente o tiro proveniente do botão “Mamífero” poderá destruí-la.

Objetos deste tipo possuem uma única componente de velocidade na vertical para baixo (não se movimentam na horizontal) e seu valor depende de quantas naves já foram destruídas.

Uma última observação a ser feita em relação à esta classe é relativo ao seu método update. As funcionalidades relativas à posição da nave continuam ativas mesmo depois que uma nave morre. Isto é útil pois o momento do aparecimento de uma nova

nave depende da posição Y da nave anterior. Assim, a posição da nave continua sendo atualizada mesmo que os outros métodos de desenho e atualização sejam desativados.

4.6.7 – A Colisão entre um Tiro e uma Nave

O método update da GameView possui um cálculo para verificar se, a cada ciclo de atualização, um tiro está encostado em uma nave. Este cálculo leva em conta, as posições atuais, as alturas e larguras dos tiros e das naves envolvidas. Se uma colisão for detectada, uma de duas coisas acontecem.

Se o tiro e a nave forem do mesmo tipo, a nave e o tiro são destruídos e acontece uma explosão de partículas. Se forem de tipo diferente, a nave fica invencível por um tempo e um "campo de força" aparece ao redor da nave neste intervalo de invencibilidade. Este campo de força é representado por um círculo colorido ligeiramente maior que a nave. As cores variam entre roxo e amarelo a cada atualização para dar uma sensação animada de vivacidade.

4.6.8 – A Explosão de Partículas

Quando um tiro acerta uma nave do mesmo tipo, além desta ser destruída, uma explosão de partículas que lembra fogos de artifício acontece no local e no momento do impacto. Esta explosão consiste em um grupo de partículas (neste caso, pixels coloridos aleatoriamente) que se espalham pela tela provenientes de um único ponto.

Duas classes foram criadas para representar esta explosão: a classe Particle (que representa apenas uma partícula desta explosão) e a classe Explosion (que representa a explosão em si). Um objeto do tipo Explosion contém um conjunto de objetos do tipo Particles que é instanciado no próprio construtor da Explosion.

Quando uma explosão ocorre, algumas partículas são criadas e posicionadas nas coordenadas em que se encontra a nave no momento em que foi destruída. O efeito visual resultante do movimento das partículas é o resultado da definição aleatória de cores, tamanho e velocidades iniciais de cada partícula. Uma partícula não é nada mais do que um pequeno retângulo (pode ser uma imagem, um círculo ou qualquer outra forma mas, neste caso, um retângulo) com algumas propriedades.

Uma destas propriedades é o tempo de vida da partícula. Depois de criada, a partícula permanece um tempo ativa antes de desaparecer. Esta propriedade é regulada através de uma variável definida como a "idade" da partícula que é incrementada a cada ciclo da thread. Quando o valor da idade alcança o tempo de vida máximo, a particula é

desativada e deixa de ser exibida na tela. Todas as partículas de explosões do jogo têm o mesmo tempo de vida.

As cores, os tamanhos e as velocidades de cada partícula são randomizadas no momento de sua criação. Para isso, foram criados métodos auxiliares para a geração aleatória de cores, largura, altura e coordenadas dos vetores de velocidade. Uma vez que uma partícula possui atributos definidos como posição e velocidade, o método de atualização da partícula se torna muito análogo aos outros objetos animados no jogo.

Outra característica interessante destas partículas é que a componente alpha (responsável pela transparência das cores) da partícula é decrementada a cada ciclo. Isso faz com que a cor da partícula enfraqueça aos poucos e, a cada atualização, temos a impressão de que a partícula está sumindo gradativamente até ficar completamente transparente. A figura 4.5 ilustra o momento em que uma explosão ocorre.

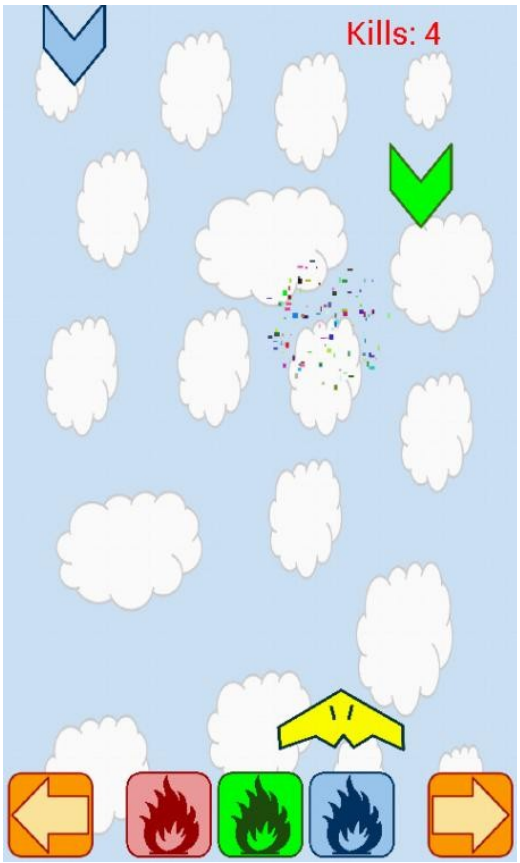

Figura 4.5: A explosão de partículas.

As partículas também conseguem detectar colisões com os cantos da tela para que não escapem. O código de colisão é bastante simples e seus métodos são chamados

no bloco update( ). Basicamente, quando uma partícula alcança uma borda da tela, uma das componentes da velocidade da partícula é invertida (se toca uma borda lateral, a componente X é invertida, se toca a borda superior ou inferior, a componente Y é invertida), fazendo com que ela retorne para dentro da área da tela.

4.7 - A Tela de Game Over

Quando uma das naves inimigas consegue descer até alcançar a altura do canhão sem que seja destruída, o jogo termina. Com isso, o jogo muda o seu estado para "Game Over". Neste estado, o canhão é destruído, as naves inimigas param de surgir, uma mensagem de Game Over aparece na tela e o usuário passa a ter três opções. Reiniciar o jogo (botão Restart), sair do aplicativo (botão Quit) ou voltar para a tela de menu principal (botão Main Menu).

Estes botões e a mensagem, assim como os outros elementos da GameView, foram criadas no construtor, mas posicionadas e dimensionadas através do método surfaceCreated. Porém, apesar de serem criados ao mesmo tempo que os outros componentes do jogo, estes objetos somente são exibidos no final do jogo. Isso ocorre pois a exibição destes objetos são imediatamente desativadas através da alteração do valor do atributo *active*. Como já foi explicado anteriormente, este é um atributo booleano é comum a todos os objetos e, neste caso, recebem o valor false quando são criados (os atributos active dos botões de tiro e de movimento recebem o valor true quando criados e, no estado Game Over, passam a ser desativados e recebem o valor false). A figura 4.6 mostra a tela de Game Over.

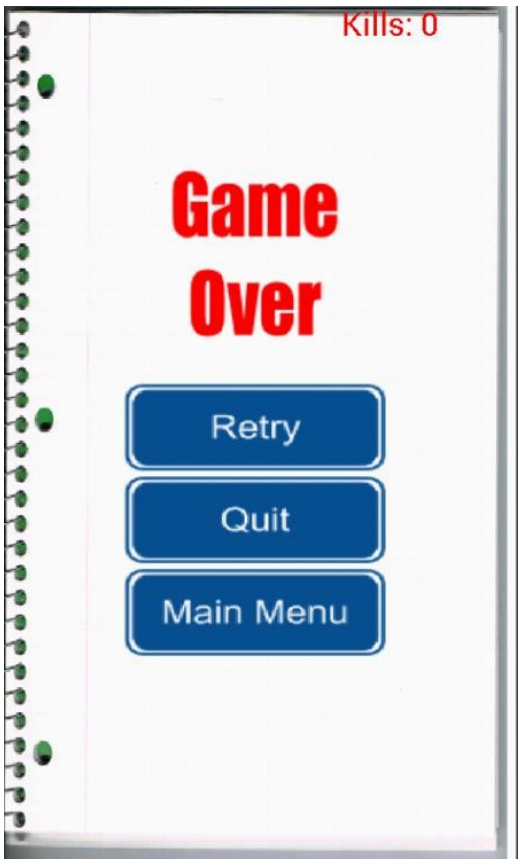

Figura 4.6: A tela de Game Over

4.8 – O Número de Kills

O objetivo do jogador, diferente de outros jogos tradicionais, não é alcançar um objetivo final, mas sim, obter o máximo de pontos que conseguir. O número de acertos é determinado pela variável *kills* e é constantemente mostrado no canto superior direito da tela e é incrementada a cada nave destruída.

Outra dificuldade que o jogador irá enfrentar é que, a cada 25 naves destruídas, a velocidade vertical com que as naves se aproximam da base aumenta. Assim, o jogo vai se tornando mais desafiador e dinâmico na medida em que o jogador constrói o seu placar. A velocidade inicial do jogo é razoavelmente lenta para dar tempo de um jogador iniciante entender o que está acontecendo e o que ele deve fazer inicialmente. Se a velocidade continuasse a mesma durante toda a rodada, a jogabilidade iria se tornar maçante e previsível.

Capítulo 5
A DownloadGameActivity

5.1 – Introdução

Um jogo como descrito até agora – envolvendo naves invasoras, canhões e explosões – não pode ser considerado educativo. Mas se mantivermos a mecânica do jogo e alterarmos somente as imagens, podemos ter um jogo totalmente novo. Por exemplo, se no lugar das naves invasoras, criarmos expressões matemáticas e no lugar dos botões de tiro, os respectivos resultados, teremos um jogo que exige um rápido raciocínio matemático.

Na figura 5.1 temos uma captura de tela do jogo "A Matter of Life and Math", criado como exemplo para demonstrar que tipo de jogo educativo pode ser desenvolvido. Da mesma forma que foi explicado nos capítulos anteriores, o tiro certo deve acertar as naves certas, neste caso, as expressões. Para a expressão 54/6 atiramos com o botão 9, para 63/9, com o botão 7 e assim por diante.

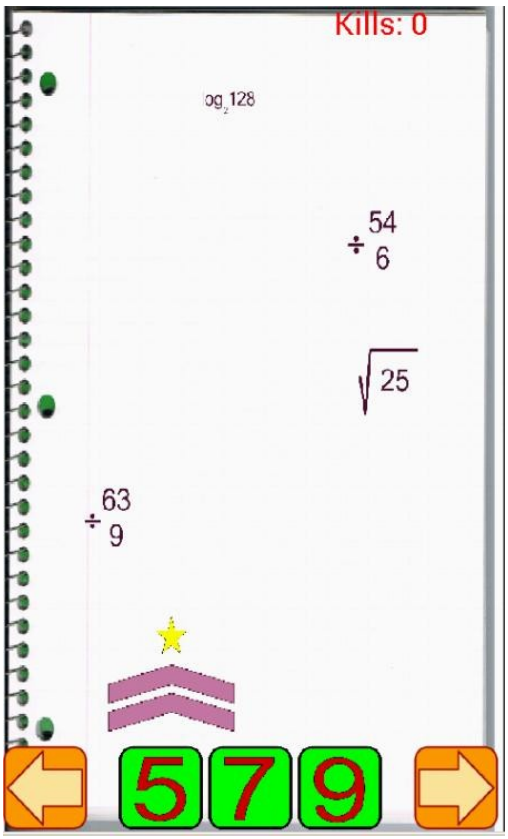

Figura 5.1: A Matter of Life and Math

As possibilidades de tipos de aprendizado que podemos ter com esse tipo de abordagem são inúmeras e foi necessário o uso de um servidor para o armazenamento das imagens de cada jogo. E é justamente a DownloadGameActivity que irá prover a conexão com este servidor. A seguir, temos uma breve explicação de como funciona este servidor e de como são estruturados os arquivos armazenados.

5.2 - Servidor, arquivos e pastas

Um drive de armazenamento online foi criado para funcionar como um servidor FTP e guardar os arquivos contendo as imagens de cada jogo personalizado. O aplicativo estabelece uma conexão FTP com o servidor no momento em que a DownloadGameActivity é acessada.

O servidor escolhido foi o www.drivehq.com[5], que permite armazenamento online, acesso através do protocolo FTP e, além de ser gratuito, possui um razoável espaço para armazenamento. Para implementar o lado cliente FTP no aplicativo, o

pacote *commons-net-3.2.jar* (pertencente à biblioteca *Apache Commons Net* ™[4]*)* foi adicionado como recurso ao projeto. A biblioteca *Apache Commons Net* ™[4] possui inúmeros recursos que podem ser utilizados em projetos na linguagem java que gerenciam conexões e transferência de dados utilizando protocolos básicos da Internet. No caso deste projeto, funcionalidades da conexão FTP entre o aplicativo e o servidor serão implementadas através da importação da classe FTPClient. Esta classe possui métodos úteis para o estabelecimento de sessões FTP, listagem de arquivos e download de pacotes.

Cada versão do jogo corresponde a um arquivo compactado no formato ZIP disponível no servidor. A ideia é que qualquer usuário possa editar e armazenar o seu próprio ZIP no servidor, sem a necessidade de qualquer conhecimento em programação ou desenvolvimento de software. Quando o aplicativo finaliza o download de um ZIP, este é descompactado automaticamente na memária do cartão SD do aparelho.

Posteriormente, a GameActivity irá carregar as imagens descompactadas como componentes do jogo de acordo com a localização das imagens nas pastas correspondentes. Por isso, ao criarmos o ZIP, devemos considerar a organização e os nomes das pastas compactadas. A estrutura de arquivos compactada deve ser organizada da forma ilustrada na Figura 5.1.

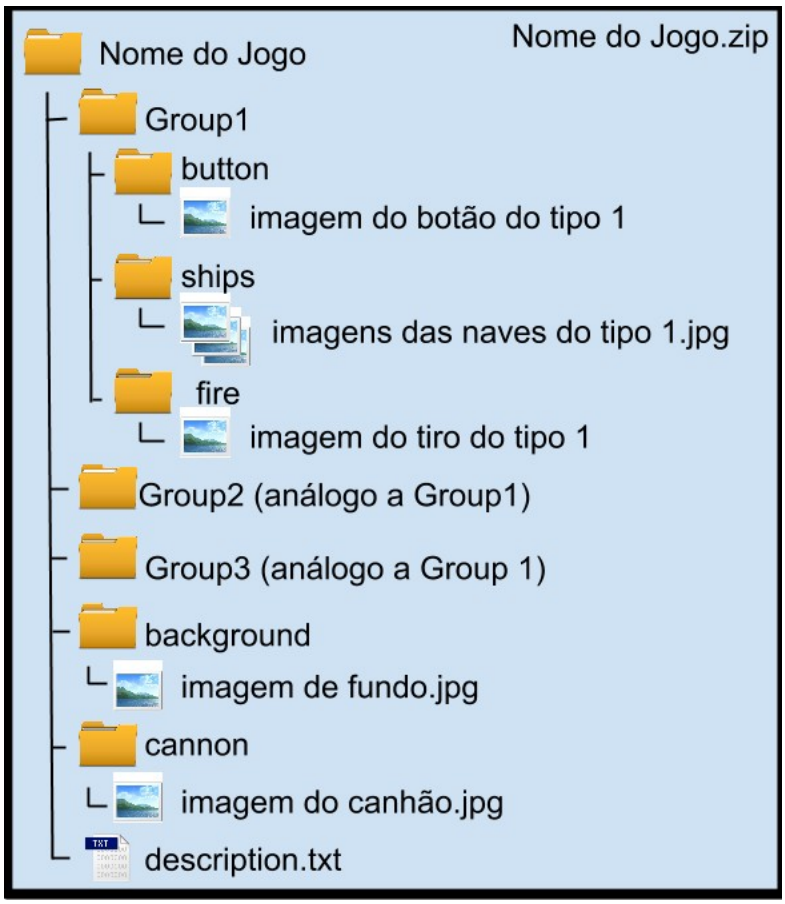

Figura 5.1: A organização das pastas no servidor.

O nome do jogo definido será o mesmo nome do ZIP e da pasta raiz da estrutura de arquivos. Nesta pasta está localizado o arquivo **description.txt** no qual está gravado o texto exibido na ChooseGameActivity como uma breve descrição do jogo. A pasta **background** e a pasta **cannon** contém as imagens correspondentes à imagem de fundo e ao canhão controlado pelo usuário, respectivamente. Além destas pastas, a pasta raiz também contém as pastas **Group1**, **Group2** e **Group3**. Em cada uma destas pastas estão contidas as imagens correspondentes aos três tipos de botões, naves inimigas e tiros e são guardadas nas pastas **button**, **ships** e **fire**, respectivamente.

O conteúdo destas pastas deve ser pensado pelo criador do jogo como charadas, perguntas ou desafios e suas respectivas respostas. Podemos citar novamente o exemplo hipotético de uma versão do ESI onde diferentes imagens de animais representam as naves inimigas e cada botão representa, por exemplo, mamíferos, répteis e anfíbios. Isto significa que os botões serão representados por imagens contendo as repostas em si (Group1: mamífero, Group2: réptil e Group3: anfíbio) e as naves representadas pelas imagens dos animais, cada grupo de animais dentro da sua respectiva pasta.

Note que na pasta **ships**, poderemos ter uma ou mais imagens representando um único tipo de nave. Isso faz sentido pois desejamos que inúmeros desafios com a mesma resposta seja implementada. Quanto mais imagens são colocadas nesta pasta como charadas, mais desafiador se torna o jogo. Os nomes das imagens utilizadas é de preferência do autor, contanto que estas estejam localizadas nas pastas corretas. As imagens podem ser do tipo jpg ou png.

5.3 - A ListActivity

A DownloadGameActivity e a ChooseGameActivity são de um tipo especial de classe filha de ListActivity. Este tipo de Activity exibe uma lista de itens ligados a uma fonte de dados como um vetor ou um conjunto de informações. Possui métodos que lidam com os eventos resultantes quando o usuário seleciona um dos itens. A figura 5.3 demonstra como é a aparência da tela quando a DownloadGameActivity é acionada. Esta é a lista de jogos disponíveis no servidor.

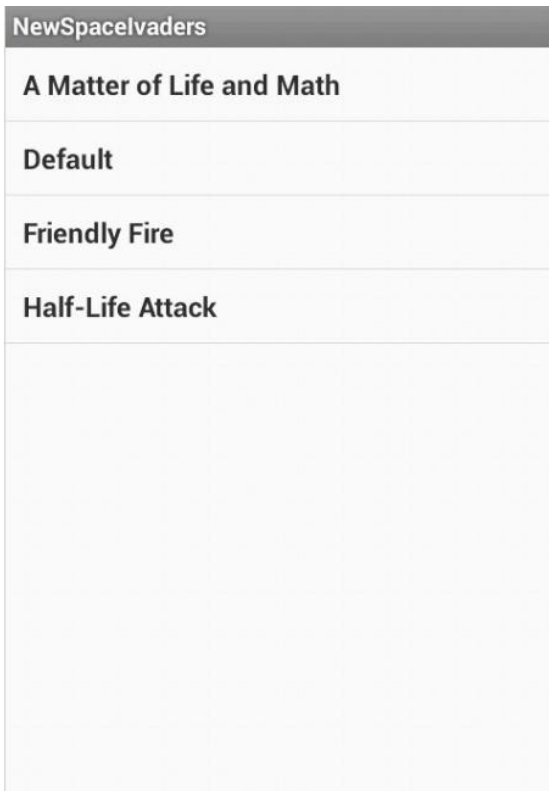

Figura 5.3: A lista da DownloadActivity.

Cada item da lista é uma View elaborada através de um arquivo .xml contendo apenas um objeto do tipo TextView. Este objeto é ligado a uma String que contém o nome de cada jogo. Para formar esta lista, precisamos iniciar uma sessão com o servidor para possibilitar o acesso aos itens disponíveis. Por isso, assim que esta Activity é aberta, a conexão é estabelecida e os métodos do pacote *commons-net-3.2.jar* são chamados. Com esse pacote, podemos criar um objeto do tipo FTPClient que representa o lado cliente da conexão ftp e usamos seus métodos para gerenciar a comunicação. Na tabela 5.1, apresentamos as descrições de alguns métodos importantes presentes no código do aplicativo para lidar com a conexão no momento da listagem e posteriormente, do download.

Tabela 5.1: Métodos utilizados para a conexão

Método	Descrição
connect(server, port)	Estabelece a conexão com o servidor especificado utilizando a porta selecionada.
login(username, password)	Inicia a sessão utilizando o nome do usuário e a senha, se necessário.
changeWorkingDirectory(directory)	Seleciona o diretório a ser utilizado.
listNames(names)	Lista os nomes dos arquivos contidos no diretório selecionado.
setFileType(fileType)	Indica que tipo de arquivo será baixado
retrieveFileStream(fileName)	Grava o arquivo (.zip) disponível no servidor em uma variável do tipo InputStream que posteriormente será copiada para a pasta desejada na memória do aparelho.
disconnect( )	Finaliza a sessão.

Caso haja falha na conexão, o aplicativo retorna ao menu principal e uma mensagem de falha na conexão é exibida na tela.

5.4 - Realizando o Download

5.4.1 – Como Funciona o Download

Quando um dos itens é selecionado, quatro coisas acontecem em sequência: 1) a pasta esi é criada na memória do cartão SD do aparelho (somente no primeiro download); 2) o download da pasta compactada do jogo selecionado é efetuado; 3) é feita a descompactação do zip correspondente para a pasta esi; 4) o aplicativo retorna para o menu principal.

Foram criadas duas classes auxiliares para lidar com essa parte do código. O EsiFileManager e o EsiZipManager. A primeira foi elaborada para prover a criação e a exclusão das pastas necessárias para o armazenamento das imagens e a segunda, para executar as atividades relacionadas à descompactação do zip para as devidas localizações.

Por exemplo, quando o jogo "Default" é selecionado, o download do arquivo Default.zip é inicializado através do método retrieveFile( ). O arquivo é gravado em uma variável do tipo InputStream e, em paralelo, outra variável do tipo OutputStream é criada e direcionada para a pasta esi. Após o download, a variável InputStream é copiada para a OutputStream e passamos a ter o arquivo Default.zip localizado na pasta esi. Essa cópia é feita graças a outra classe do pacote da Apache, a IOUtils. Esta classe contém métodos úteis para lidar com leitura, escrita e cópia de bytes para variáveis do tipo *stream*.

Assim que o download e a cópia são finalizados, automaticamente começa a descompactação da Default.zip. A classe EsiZipManager é a encarregada de descompactar as pastas para os locais corretos. Agora, temos uma pasta (de nome Default), com a mesma estrutura explicada anteriormente, localizada na pasta esi. Assim que a descompactação termina, o arquivo Default.zip é deletado, o aplicativo retorna ao menu principal e é exibida uma mensagem dizendo que o download foi concluído. Caso o download seja cancelado por falha na conexão, uma mensagem de falha no download é exibida.

5.4.2 - A Espera do Download

Todas essas operações de conexão, download e descompactação levam tempo e não resta nada ao usuário além de esperar que todos os procedimentos sejam concluídos. Por isso, todos estes processos são realizados por uma segunda thread, enquanto compete à thread principal a função de criar e exibir uma caixa de diálogo comunicando o usuário que o download está em progresso. A figura 5.4 ilustra um download em andamento.

Figura 5.4: A caixa de diálogo de espera de download em andamento.

Capítulo 6
Escolhendo o jogo

6.1 – A ChooseGameActivity

Quando tocamos o botão "Start Game" no menu principal, o aplicativo fecha a MenuActivity e abre a ChooseGameActivity. Esta activity, assim como a DownloadGameActivity, é uma extensão da classe ListActivity, o que significa que uma lista de itens será exibida na tela sem a necessidade da elaboração de uma View. Assim, o layout desta tela é semelhante ao da DownloadGameActivity como podemos ver na figura 6.1.

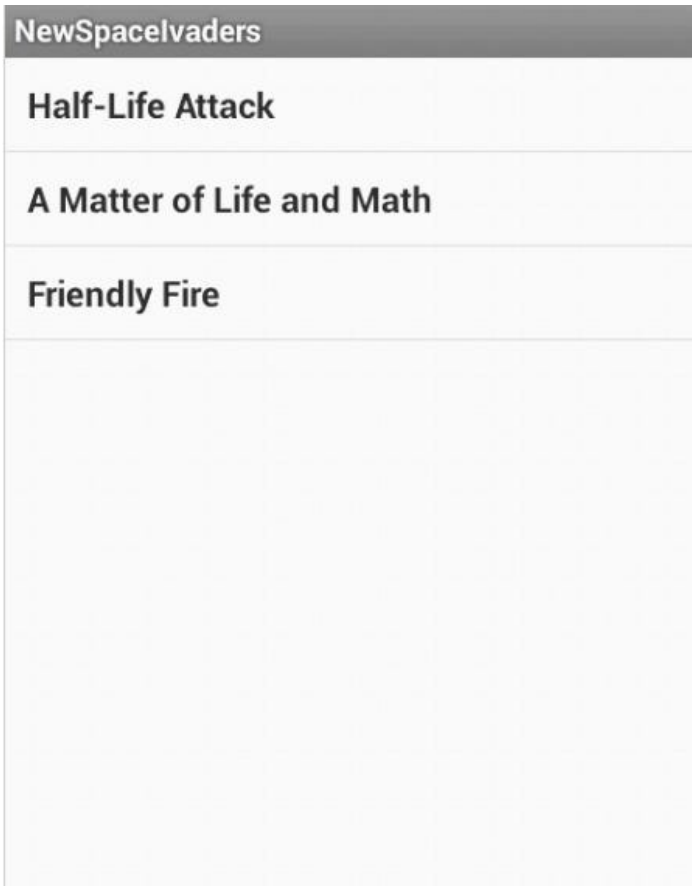

Figura 6.1: A lista da ChooseGameActivity.

Esta lista será formada pelos jogos já baixados do servidor e disponíveis na memória do aparelho. A pasta esi é percorrida e todas as subpastas são listadas. Lembrando que, neste ponto, os arquivos localizados dentro da pasta esi são as pastas descompactadas dos zips baixados do servidor. Como cada pasta tem o mesmo nome do jogo, basta exibir o nome de cada pasta localizada na esi como um item da lista.

A função desta activity é dar a opção do jogador de escolher qual das versões será carregada ao começar o jogo. Assim que uma das opções é escolhida, uma caixa de diálogo é exibida contendo a descrição do jogo. Ela também pergunta ao usuário ele quer realmente começar o jogo ou não. É neste ponto que carregamos a descrição do jogo a partir do arquivo description.txt localizado na pasta do jogo selecionado. A caixa de diálogo é ilustrada na figura 6.2. O jogo começa ao selecionarmos "Yes".

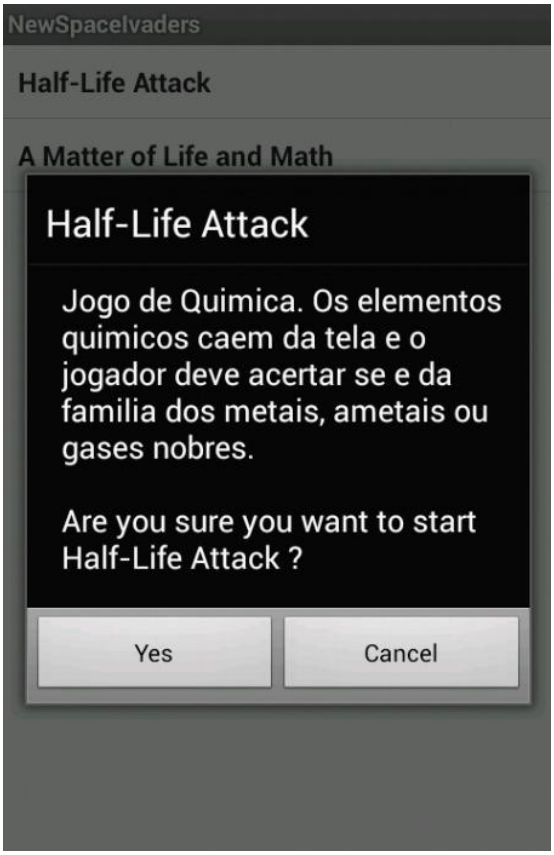

Figura 6.2: Caixa de diálogo de inicialização do jogo.

Esta Activity também possui um vetor de Strings que irá guardar o caminho da localização de cada imagem. Quando um jogo é selecionado, os caminhos de todas as imagens do jogo correspondente são alocados neste vetor que imediatamente é enviado

para a GameActivity. Na GameActivity, estes caminhos serão utilizados como parâmetros para os métodos de carregamento de imagens.

Diferente da lista da DownloadGameActivity, esta é uma lista que contém itens locais (na memória do smartphone) e não em um servidor. Logo, o usuário tem permissão para deletar um jogo salvo na memória. Isto pode ser feito nessa activity através de um toque de longa duração (long click) em um dos itens da lista. Ao fazermos isso, mais uma vez uma caixa de diálogo aparece. Desta vez, para perguntar se o usuário tem certeza se quer mesmo deletar a versão do jogo selecionada. Podemos observar esta caixa de diálogo na figura 6.3.

Figura 6.3: Caixa de diálogo de exclusão de um jogo.

Ao deletarmos um jogo da memória, a pasta correspondente ao jogo e seu conteúdo (subpastas, imagens e o arquivo description.txt) é deletado e só pode ser recuperado baixando novamente do servidor.

Se não houver nenhum jogo salvo na memória, a lista será preenchida com um único item com um aviso de que não existe nenhuma versão a ser carregada e um clique neste item fará com que o aplicativo retorne ao menu principal.

6.2 – O Carregamento das Imagens na GameActivity

Quando um jogo é selecionado na ChooseGameActivity, todas as imagens contidas nas pastas descompactadas são referenciadas por um caminho. Este caminho é, simplesmente, uma String que indica em quais pastas estão localizadas cada imagem do jogo. Por exemplo, o caminho da imagem de fundo do jogo 'Default' é "mnt/sdcard/esi/Default/background/background.png". Cada caminho servirá como referência para possibilitar o carregamento de cada imagem para uma variável do tipo Bitmap e ser manipulada no jogo. Note que estas Strings são formadas a partir do nome do jogo escolhido que, não por acaso, é o mesmo nome da pasta.

Antes de inicializar a GameActivity, a ChooseGameActivity armazena todos os caminhos das imagens do jogo selecionado na intent. Estes dados, posteriormente, são recuperados na GameActivity e são usados como referência para o carregamento das imagens. Com o identificador utilizado pela intent, podemos recuperar cada imagem através do método putExtras("identificador", valor), distinguindo o que é uma imagem de uma nave, de um tiro ou de fundo de tela.

Na GameActivity os métodos getString(string) e getStringArray(string) são utilizados para recuperar as strings passadas pela intent. Estas strings são utilizadas como parâmetro pelo método decodeFile(string) da classe BitmapFactory. Este método consegue criar objetos do tipo Bitmap carregando um arquivo de imagem pelo caminho indicado. Caso o aplicativo não consiga encontrar a imagem solicitada, o método decodeResource(res, int) é acionado para carregar as imagens do jogo Default guardadas na pasta res do projeto.

Capítulo 7

Outros Exemplos de Jogos

Neste capítulo será mostrado que tipo de jogo educacional poderá ser criado apenas manipulando as imagens, guardando-as nas pastas corretas, compactando-as e transferindo o zip para o servidor. Foram criados três jogos (além do jogo Default) para demonstrar a ideia do Educational Space Invaders.

O primeiro jogo criado foi um jogo de Matemática chamado A Matter of Life and Math, já explicado anteriormente no capítulo 6. Outro exemplo é o Half-Life Attack. Este é um jogo de Química onde as naves inimigas, na verdade, são elementos químicos. O jogador deverá acertar se os elementos pertencem à classe dos metais, ametais ou gases nobres. A figura 7.2 mostra o Half-Life Attack em ação. Note que todas as imagens do jogo foram modificadas.

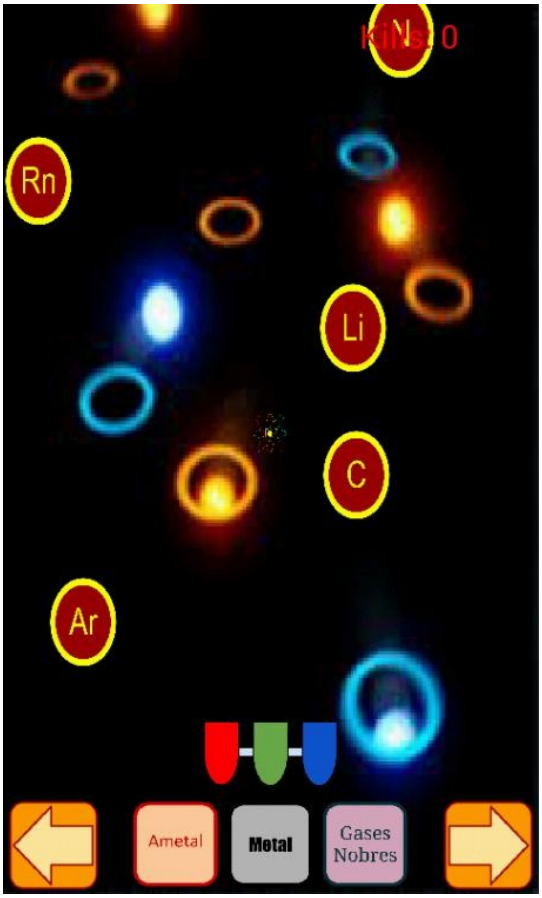

Figura 7.1: O Half-Life Attack.

O jogo Friendly Fire é um jogo onde as naves são retratos de personalidades e o jogador deverá saber a nacionalidade de cada uma. As bandeiras da Itália, França e EUA foram utilizadas para representar os botões de tiro. A figura 19 é uma captura de tela do Friendly Fire.

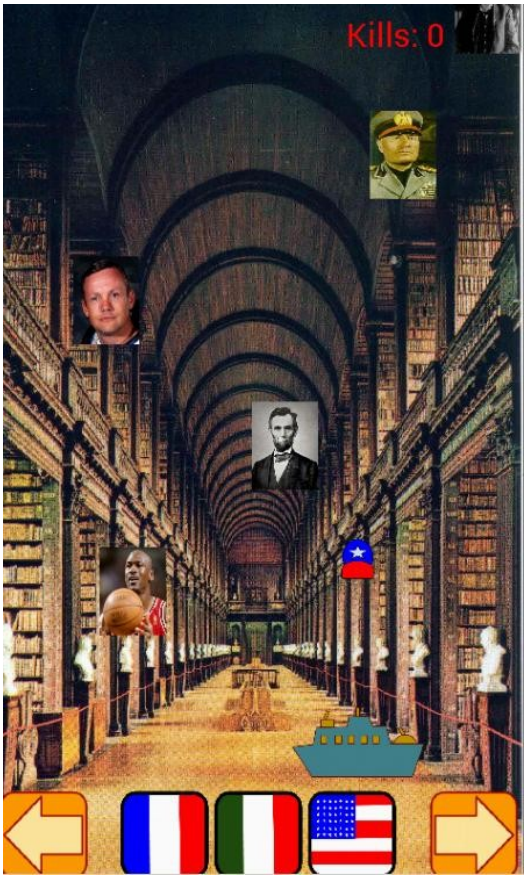

Figura 7.2: O Friendly Fire.

Capítulo 8

Conclusões

Este projeto final abordou alguns conceitos aprendidos durante a graduação, como programação orientada a objetos e desenvolvimento de softwares para Smartphone, além de concentrar-se em um tipo de inovação de design de jogos. Algumas dificuldades no projeto, como sincronizar e ordenar a pilha de execuções do com a da thread auxiliar, forma superadas. Outros problemas, como a velocidade dos objetos para tamanhos de tela diferentes, ainda persistem, quando a tela possui dimensões muito reduzidas.

A programação para jogos ainda é um tema pouco explorado nas universidades e este projeto pode ser um estímulo para o aprendizado através do desenvolvimento de aplicativos voltado para jogos. Este projeto pode ser usado para difundir conhecimentos em linguagens de programação de uma forma mais acessível.

Algumas funcionalidades do aplicativo ainda não estão disponíveis para os que desejam criar seus próprios jogos. Por enquanto, os jogos estão armazenados em um servidor privado. Este aplicativo poderia ser aprimorado com a criação de uma página na internet onde seria possível criar e armazenar as versões do jogo de forma mais intuitiva.

Bibliografia

[1] MCGUGAN, W., *Beginning Game Development with Python and Pygame*. New York, Apress, 2007.

[2] KLOPFER, E., OSTERWEIL, S., SALEN, K., "Moving Learning Games Forward", Massachusetts Institute of Technology, The Education Arcade, 2009.

[3] KYOURTZOGLOU, B., "Android Game Development Tutorials", 2011, http://www.javacodegeeks.com/2011/06/android-game-development-tutorials.html , (Acesso em 15 de Agosto 2013).

[4] "Apache Commons™", http://commons.apache.org/proper/commons-net/, 2013, (Acesso em 05 de Maio 2013).

[5] "Drive Headquarters™", http://www.drivehq.com/ , 2003, (Acesso em 22 de Julho 2013).

[6] TAROUCO, L. M. R., ROLAND, L. C., FABRE, M. J. M., KONRATH, M. L. P., "Jogos Educacionais", http://www.cinted.ufrgs.br/ciclo3/af/30-jogoseducacionais.pdf, 2011, CINTED/UFRGS, (Acesso em 30 de Julho 2013).

[7] "Eclipse.org", http://http://www.eclipse.org/ , 2001, (Acesso em 05 de Maio 2013).

Anexo A

Computador na Educação

A importância do uso dos computadores e das novas tecnologias na educação deve-se hoje não somente ao impacto desta ferramenta na nossa sociedade e às novas exigências sociais e culturais que se impõe, mas também ao surgimento da Tecnologia Educativa. Eles começaram a ser utilizados no contexto educativo a partir do rompimento com o paradigma tradicional e surgimento do construtivismo, que enfatiza a participação e experimentação do sujeito na construção de seu próprio conhecimento, através de suas interações. Com isso a capacidade do professor e o conteúdo dos livros constituem uma condição necessária mas não suficiente para garantir a aprendizagem, pois ela envolve um processo de assimilação e construção de conhecimentos e habilidades, de natureza individual e intransferível.

Os efeitos do computador na escola dependem de diversos fatores, contudo a generalidade da investigação aponta para a possibilidade de desenvolvimento de novas competências cognitivas, entre elas: maior responsabilidade dos alunos pelo trabalho, novos laços de entre-ajuda e novas relações professor-aluno. Assim, o computador se constitui numa ferramenta poderosa, que pode (e deve) ter todas as suas potencialidades utilizadas com propósitos educacionais, proporcionando ao professor a possibilidade de enriquecer sua prática pedagógica com recursos multimídia, tais como jogos educacionais, vídeos, animações, gráficos e outros materiais que possibilitem ao aluno aprender de forma prazerosa, cativante, divertida e motivadora.

Neste sentido, os jogos educacionais podem ser um elemento catalisador, capaz de contribuir para o "processo de resgate do interesse do aprendiz, na tentativa de melhorar sua vinculação afetiva com as situações de aprendizagem" (Barbosa, 1998). A vinculação afetiva exerce um papel fundamental, pois, cansado de muitas vezes tentar e não alcançar resultados satisfatórios no chamado "tempo" da escola, o aluno experimenta sentimentos de insatisfação constantes os quais funcionam como bloqueadores nos avanços qualitativos de aprendizagem.[6]