Experimentos em plataforma móvel

Hallison Oliveira da Paz

17 de maio de 2012

Resumo

Este trabalho tem por finalidade descrever o processo de concepção, desenvolvimento e coleta de resultados de atividade parcial do projeto *Realidade Aumentada Móvel*. Esta descrição refere-se às atividades transcorridas entre 24 de fevereiro de 2012 e 17 de maio de 2012. Objetivando a integração e ambientação do aluno ao desenvolvimento de aplicações em dispositivos móveis, foi proposta a criação de um aplicativo no formato de um jogo eletrônico, com maior ênfase nos aspectos técnicos do que lúdicos.

1 Aspectos Gerais

Para implementação do aplicativo, foi utilizado o *kit de desenvolvimento de software* do sistema operacional *Android* – sistema utilizado em celulares e tablets de diversas marcas disponíveis no mercado – e o *framework Cocos2D*. O *Android SDK (Software Development Kit)* é um conjunto de ferramentas para desenvolvimento, teste e análise de erros em aplicações para o sistema operacional *Android*. O ambiente de desenvolvimento *Eclipse [4]* foi utilizado por possuir um plugin oficial para integração de várias funções do *SDK* e algumas referências na documentação desta ferramenta [3]. O *framework Cocos2D* provê uma estrutura que agiliza e facilita os processos de exibição e manipulação de objetos bidimensionais, sendo baseado na biblioteca *OpenGl ES*.

O jogo possui um cenário quadriculado, possibilitando o desenho de um labirinto, com ponto de entrada e ponto de saída, a partir da composição dos quadrados. O objetivo do jogador é traçar, tocando a tela do dispositivo, um caminho entre a entrada e a saída – esse caminho não necessariamente é único. Para adicionar um desafio ao jogador, foram inseridas armadilhas, em alguns pontos do cenário, que quando tocadas terminam o jogo. Estas armadilhas são exibidas em tela somente durante alguns segundos no início do jogo, após isso, são camulfladas no cenário, com o intuito de que o jogador lembre-se de suas posições para evitá-las. Foram ainda implementadas funcionalidades para agregar o uso da câmera e do acelerômetro do dispositivo. O aparelho utilizado para testes durante o desenvolvimento foi o *LG-P500h* com a versão 2.3.3 do sistema operacional *Android*. Não é garantido o funcionamento em versões do sistema anteriores à 2.2.

2 Construção e implementação

Para melhor organização e análise do processo de implementação, foram propostas 5 etapas. Na primeira etapa, objetivou-se criar uma estrutura de dados capaz de mapear o espaço bidimensional da tela do dispositivo de exibição, referenciado em pixels, para uma grade de posições em que são exibidos quadrados de tamanhos iguais, porém com propriedades diferentes. Nesta etapa, não foi dada ênfase à implementação das propriedades dos quadrados, porém foi atingido o objetivo parcial de exibição de quadrados visualmente diferentes.

Criou-se uma estrutura de dados responsável por esse mapeamento. A partir de métodos disponíveis nas bibliotecas utilizadas, pôde-se obter as dimensões da tela do dispositivo de exibição. A partir daí, determina-se uma quantidade de quadrados a serem distribuídos na menor dimensão. A quantidade distribuída na maior dimensão, assim como a quantidade total de quadrados e o tamanho do lado de cada quadrado, podem ser calculados de maneira única a partir dessas informações. Sejam l o tamanho do lado do quadrado, D a maior dimensão do dispositivo, d a menor dimensão, n o número de quadrados dispostos na menor dimensão e N, na maior dimensão, as relações entre esses atributos estão representadas nas equações 1 e 2.

$$l = \frac{d}{n} \tag{1}$$

$$N = \left[\frac{D}{l}\right] \tag{2}$$

Em que [.] representa a função maior inteiro. Desta maneira garante-se que sempre é possível desenhar ao menos um quadrado em tela.

A figura 1 ilustra um cenário em que foi definido o valor 8 para a quantidade de quadrados na menor dimensão.

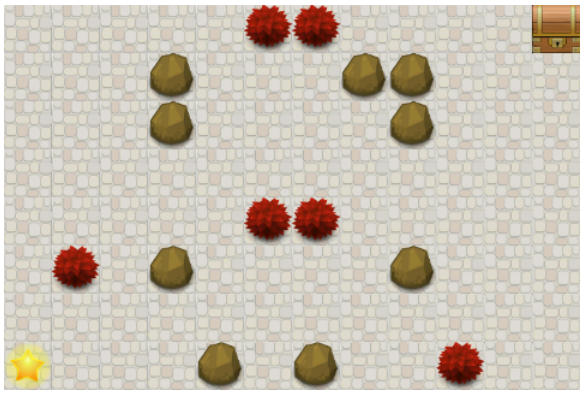


Figura 1: Cenário do jogo. Os quadrados brancos representam caminhos por onde pode-se traçar livremente. As pedras são obstáculos, que podem ser tocados, porém não podem ser atravessados. Os "arbustos" vermelhos representam as armadilhas e o baú no canto superior direito, é o ponto de saída do labirinto. Todas as imagens utilizadas neste cenário foram obtidas em [8].

Inicialmente, houve alguns problemas no desenvolvimento dessa etapa, ocasionados, principalmente, pela falta de uma documentação organizada da versão do *framework Cocos2D* no sistema *Android*. A maioria dos problemas ocorria quando se tentava incrementar – especializar – alguma classe da biblioteca do *Cocos2D*. A solução encontrada para esses problemas foi estudar diretamente o código fonte do projeto de desenvolvimento da versão do *Cocos2D* utilizada. Este código encontra-se disponível em endereço eletrônico [6] constante na bibliografia.

Em uma segunda etapa do desenvolvimento, foi dada atenção às propriedades de cada quadrado. Para os quadrados do jogo, de uma maneira geral, foi criada uma classe abstrata derivada de *CCSprite* para representar suas características comuns e, posteriormente, para cada tipo de quadrado, foi implementada uma classe específica como "caminho", "obstáculo" e "armadilha". A classe *CCSprite* é utilizada para representar objetos do tipo *sprite*, que consistem em imagens bidimensionais que podem ser integradas a uma cena maior, porém manipuladas de maneira independente. Este tipo de jogo, em que uma determinada quantidade de quadrados são desenhados repetidamente para compor um cenário, é conhecido como *jogo em tiles (tile game)*. *Tile* – do inglês, azulejo ou ladrilho – é o nome dado aos quadrados que compõem o jogo, e é o termo que será utilizado no restante deste documento.

Cada *tile* pode ser visto como um nó de um grafo, em que as propriedades e relações de adjacência e alcance foram estabelecidas em etapas posteriores, de acordo com o tipo de cada um. Com isso, finalizou-se a construção do cenário.

3 Toque e movimentação

A partir do toque do usuário, com o auxílio do método *ccTouchesMoved* do *Cocos2D*, pode-se obter as coordenadas em pixels do ponto tocado. Cada um dos *tiles* exibidos possui uma referência armazenada em uma matriz bidimensional, o que permite o acesso a qualquer um deles durante a execução do aplicativo. Para qualquer pixel presente na tela do dispositivo de exibição, existe um elemento correspondente na referida matriz. Assim, quando o usuário toca na tela do aparelho, pode-se saber qual o *tile* correspondente àquela posição.

Foram implementados dois métodos de retorno *booleano* para a classe *tile*. Um deles indica se é permitido ao jogador traçar um caminho por sobre aquela instância e é utilizado na construção lógica do cenário. Como o cenário possui uma estrutura de grafo, todos os nós que este método retorna valor lógico falso, possuem grau nulo, ou seja, não estão conectados a nenhum outro nó do grafo. O segundo método de retorno *booleano* nos permite verificar se o *tile* tocado provoca o término do jogo e a derrota do jogador. Com essa estrutura, pôde-se montar um cenário que permite uma movimentação coerente com a ideia do jogo. O cenário é lido a partir de uma matriz com três números inteiros distintos, cada um representando um tipo de *tile*: caminho, obstáculo ou armadilha. A classe responsável pela montagem do cenário se encarrega de analisar o número recebido como entrada, configurar as propriedades da abstração correspondente e formular a correta relação de adjacência entre os nós do grafo. Com isso, pode-se inserir o ponto inicial em um local qualquer do cenário e quando o usuário quiser traçar na tela e mover o "personagem", basta verificar se o *tile* que ele tocou está conectado com o *tile* anterior. Isso evita que o usuário tente traçar um caminho descontínuo ou que ele tente atravessar paredes ou outros locais não permitidos, visto que *tiles* não caminháveis não estão conectados a nenhum outro nó. Cada nó guarda consigo uma lista de referências aos nós adjacentes.

4 Complementos

Ao finalizar essa etapa, foi trabalhada a parte de desafio do jogo. Como mencionado anteriormente, a ideia é que o jogador trace um caminho entre o ponto inicial e o ponto final sem passar por cima de nenhuma armadilha, que apenas é exibida no início do jogo. Para isso, definiu-se na classe que representa a armadilha, um tempo de 5 segundos para que o método que troca a textura seja chamado. Assim, a partir de sua criação, cada instância de armadilha exibe sua textura natural por 5 segundos e, logo após, passa a exibir uma imagem igual à de um caminho, camuflando-se.

Neste ponto, para tornar a aplicação jogável, faltava apenas implementar as mudanças de tela representando o fim do jogo, tanto para a derrota do jogador, quanto para vitória. O *Cocos2D* pemite configurar métodos para serem chamados constantemente pelo sistema, com uma frequência determinada pelo programador. Isso é feito com o uso do método *schedule*, presente em todas as classes derivadas de *CCNode* – classe base de todas as classes que geram objetos

desenháveis em tela. Com isso, foram configurados um método de *game over* e outro de vitória, que executam suas rotinas, respectivamente, quando o jogador traça por sobre uma armadilha ou quando ele consegue chegar ao ponto final.

5 Câmera

O objetivo desta etapa é capturar uma imagem a partir da câmera do dispositivo e fazer um tratamento básico, que permita construir um cenário – no modelo aceito pelo jogo – semelhante à imagem. Isso permitiria que pessoas interagissem construindo desafios umas para as outras. Como a câmera é apenas necessária para a captura da imagem, não há necessidade de construir uma aplicação de câmera desde o começo. Pode-se aproveitar a aplicação de câmera que vem por padrão no sistema operacional *Android*, solicitando que ela retorne, para o jogo, a imagem capturada como um objeto do tipo *bitmap*. A partir do objeto *bitmap*, toma-se as dimensões em pixels de largura e altura da imagem e divide-se cada valor pela quantidade de tiles por linha e por coluna. Assim, obtém-se a quantidade de pixels que cada *tile* compreende.

Com esses dados, seleciona-se o pixel correspondente à posição do centro de cada tile do cenário e toma-se as coordenadas tricromáticas no sistema *CIE-RGB* desses pixels. As coordenadas são dadas por valores inteiros de 0 a 255, inclusive. Executa-se um processo de normalização em que a coordenada de maior magnitude é levada ao valor 255 (maior valor possível) e as demais são ajustadas de acordo com a equação 3. Essa normalização permite uma melhor comparação entre as coordenadas e reduz problemas relacionados à diferença de iluminação entre ambientes.

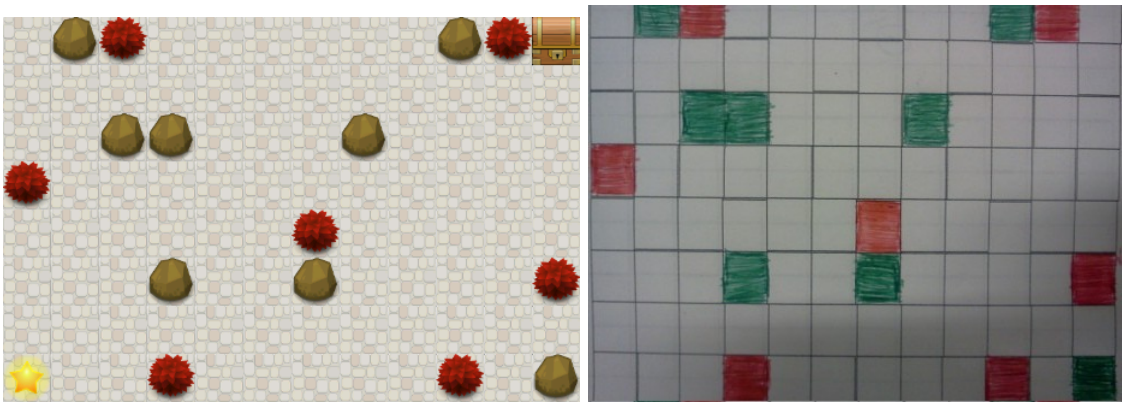
$$C = \frac{C * 255}{máx(R, G, B)} \tag{3}$$

Em que C representa a componente da coordenada tricromática que está sendo atualizada e *máx(R,G,B)* representa o maior valor entre essas três componentes.

O objetivo desses cálculos é determinar se vamos aproximar a cor obtida para azul, vermelho ou branco. Para construção do cenário, inicialmente, foi arbitrado que a cor vermelha seria traduzida em armadilha, a cor azul, em obstáculo e as cores próximas ao branco, permaneceriam como caminho comum. Com as coordenadas normalizadas, foram fotografadas várias imagens em ambientes com iluminações diferentes, com o intuito de se observar os valores retornados para os diversos tons de azul, vermelho e branco.

No entanto, os resultados com a cor azul tiveram algumas distorções, dificultando a aplicação de uma métrica simples e eficaz. Isso provavelmente aconteceu porque o aparelho de visão do ser humano é menos sensível à cor azul e, consequentemente, nossos dispositivos de captura de imagens também o são. Para contornar este problema sem a necessidade de fazer um tratamento matemático

complexo da imagem, optou-se por utilizar a cor verde para representar os obstáculos. Das três cores primárias do sistema *CIE-RGB*, o olho humano é mais sensível ao verde. As figuras 2 (a) e (b) ilustram o resultado.



(a) Cenário exibido no jogo

(b) Fotografia base para o cenário

Figura 2: Comparação entre fotografia e cenário exibido

Se a diferença entre as componentes verde e vermelha da cor estiver acima de um determinado limiar, aproxima-se a cor para aquela de maior magnitude. Caso contrário, a cor analisada é levada ao branco. Essa métrica funciona bem, dado que as imagens fotografadas foram coloridas com esse padrão de cor. Foram executados testes em ambientes com muita e pouca iluminação. Com o uso da cor verde ao invés da azul, a interferência de sombras foi bastante reduzida. A figura 3 mostra uma fotografia da mesma imagem base para o cenário, porém com iluminação menos intensa devido a sombras. O resultado obtido foi o mesmo da figura 2 (a).

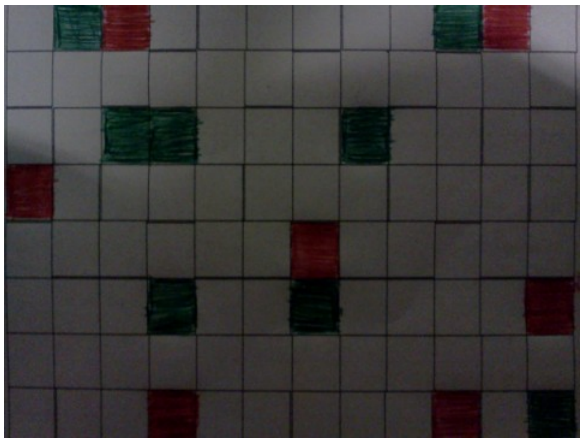


Figura 3

6 Acelerômetro

Como um elemento adicional, visando utilizar o recurso de acelerômetro, foi implementada uma movimentação diferente no jogo, gerando algumas adaptações. Com isso, o jogador deveria movimentar uma bolinha pelo cenário, inclinando o dispositivo. O método *ccAccelerometerChanged* do *framework Cocos2D* retorna valores do tipo ponto flutuante - *float* - de acordo com o posicionamento do dispositivo. Esses valores podem ser manipulados e adicionados à posição dos objetos que deseja-se movimentar.

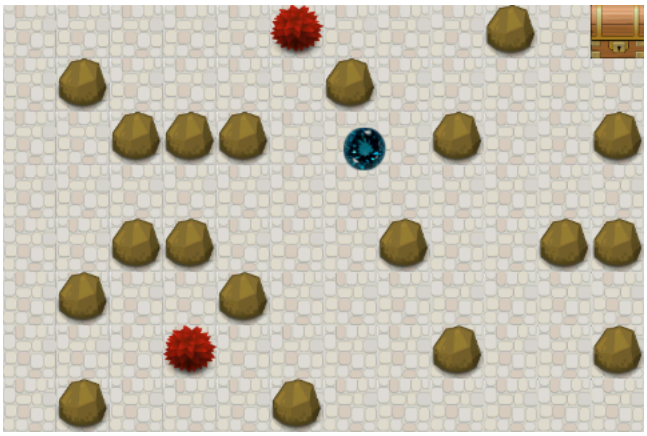


Figura 4: Próximo ao centro, bola azul movimentada pelo acelerômetro

7 Conclusão

Ao final deste período, obteve-se um jogo que explora um pouco dos recursos presentes nas plataformas móveis atuais. A inserção de elementos de visão computacional, serviu como um contato inicial com as dificuldades de se utilizar sensores para captar informações do mundo real e buscar métricas adequadas para tratar essas informações. Como trabalho futuro, pode-se pensar em uma maneira de tornar a aplicação jogável diretamente sobre o cenário desenhado, ao invés de utilizar uma fotografia para construir o cenário. Esta proposta enquadra-se na definição de realidade aumentada de exibir conteúdo que mescle elementos virtuais com elementos reais em tempo real.

Referências

[1] Cocos2D para iphone - http://www.cocos2d-iphone.org/about

[2] http://dan.clarke.name/2011/04/how-to-make-a-simple-android-game-with-cocos2d/ - acesso em 3 de abril de 2012

[3] Documentação para desenvolvedores Android - http://developer.android.com/guide/index.html

[4] *Eclipse IDE* - http://www.eclipse.org/

[5] http://en.wikipedia.org/wiki/Sprite_(computer_graphics) - acesso em 28 de abril de 2012

[6] https://github.com/ZhouWeikuan/cocos2d/tree/master/cocos2d-android - último acesso em 20 de abril de 2012.

[7] LECHETA, Ricardo R. Google Android – Aprenda a criar aplicações para dispositivos móveis com o Android SDK, 2ª edição, Ed. Novatec. São Paulo, 2010

[8] http://www.lostgarden.com/2007/05/dancs-miraculously-flexible-game.html – acesso em 10 de maio de 2012