Experiências com Desenvolvimento Ágil de um Jogo Casual para Plataformas Móveis usando o Motor Gráfico *Unity*

Daniel Valente de Macedo Maria Andréia Formico Rodrigues

Universidade de Fortaleza (UNIFOR)
Programa de Pós-Graduação em Informática Aplicada (PPGIA)
Av. Washington Soares 1321, J(30)
60811-905 Fortaleza-CE Brasil

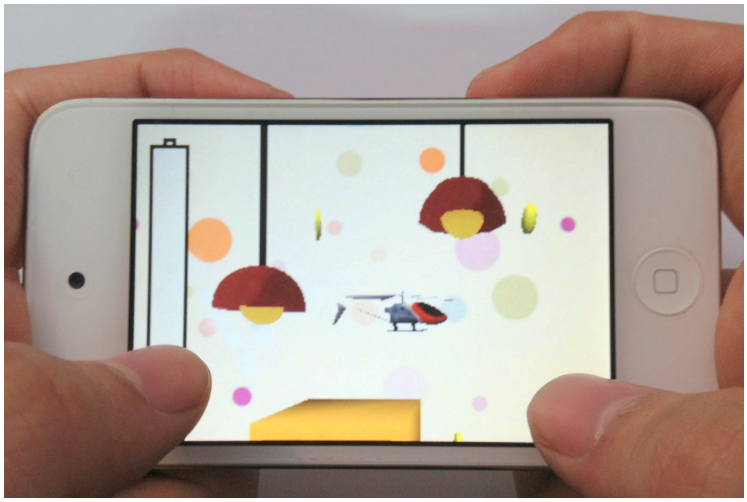

Figura 1: Um dos cenários do *FunCopter*.

Abstract

In this work, we present *FunCopter,* a casual game for mobile platforms (*iPhone 3G*, *iPhone 3GS*, *iPhone 4/4S* e *iPad*) that we have designed and implemented using *Unity* engine. We describe our personal experiences on casual game development, with emphasis on the existing constraints related to graphics aspects. Additionally, we detail the main activities realized at each phase of the game development process and the lessons we learned during its implementation. These descriptions include details from the initial concept to the fully realized playable game, which are particularly interesting and useful for casual game developers in general.

Keywords: development process, casual game, mobile platforms, *Unity* game engine

Resumo

Neste trabalho, apresentamos *FunCopter,* um jogo casual para plataformas móveis (*iPhone 3G*, *iPhone 3GS*, *iPhone 4/4S* e *iPad*) que projetamos e implementamos usando o motor gráfico *Unity*. Descrevemos nossas experiências pessoais no desenvolvimento de jogos casuais, com ênfase nas restrições existentes quanto aos aspectos gráficos. Adicionalmente, detalhamos as atividades realizadas em cada fase do processo de desenvolvimento de jogos e as lições que aprendemos durante a sua implementação. Estas descrições incluem detalhes sobre o conceito inicial do jogo até a sua finalização, as quais são particularmente interessantes e úteis para desenvolvedores de jogos casuais em geral.

Palavras-chave: processo de desenvolvimento, jogo casual, plataformas móveis, motor gráfico *Unity*

Contato dos autores:
{danielvalentemacedo,andreia.formico}@gmail.com

1. Introdução

A indústria de desenvolvimento de jogos tem apresentado um crescimento bastante acentuado nas últimas décadas [Cheah and NG 2005]. Além disso, a indústria de jogos tem influenciado uma ampla e variada gama de perfis de usuários, tornando-se parte importante da vida cotidiana da maioria das pessoas, sem restrição de faixa etária [Kurkovsky 2009, Bell *et al*. 2006].

Atualmente, há uma grande variedade de jogos disponíveis para computadores, consoles, dispositivos

móveis e até mesmo para a televisão digital. Contudo, jogos para dispositivos móveis são as principais aplicações de entretenimento no momento e um dos segmentos de mais rápido crescimento na indústria de jogos [Jhingut *et al.* 2010, Jie *et al.* 2011]. Paralelamente, novos modelos de *smartphones* estão constantemente sendo lançados e os consumidores têm apresentado o hábito de comprar jogos para os seus celulares, desta forma, influenciando diretamente os desenvolvedores a projetar frequentemente novos jogos para as plataformas mais recentes.

Contudo, o desenvolvimento de jogos para dispositivos móveis envolve muitas questões críticas e apresenta desafios consideráveis para os desenvolvedores, por exemplo, as limitações dos recursos computacionais [Chehimi 2008]. Simultaneamente, jogos desenvolvidos para essas plataformas devem satisfazer uma série de requisitos críticos e não-funcionais para atender às expectativas dos usuários e das empresas, tais como: portabilidade, jogabilidade, nível de realismo gráfico, entre outros.

Neste trabalho, apresentamos *FunCopter* (Figura 1), um jogo casual voltado para dispositivos móveis, simples e fácil de jogar, projetado e desenvolvido utilizando-se o motor gráfico *Unity* [Creighton 2010, Blackman 2011]. Mais especificamente, *FunCopter* oferece suporte para vários tipos de plataformas móveis: *iPhone 3G*, *iPhone 3GS*, *iPhone 4/4S* e *iPad*. Dentre as suas principais características podemos destacar: facilidade de aprendizado e compreensão, comandos simples de toque na tela, uso de acelerômetro, interação com outros jogadores e ciclo de desenvolvimento rápido. Destacamos também aspectos importantes de cada fase do processo de desenvolvimento do jogo [McConnell 1996, Menard 2011, Bates 2004], bem como as principais limitações encontradas sob o ponto de vista gráfico. Estas descrições incluem detalhes sobre o conceito inicial do *FunCopter* até a sua finalização, sendo particularmente interessantes e úteis para desenvolvedores de jogos em geral.

2. Processo de Desenvolvimento

O processo de desenvolvimento de jogos é um procedimento que tem geralmente duas etapas básicas: pré-produção e produção [Bates 2004]. Estas duas etapas principais, por sua vez, envolvem algumas fases mais específicas, por exemplo: Análise, Projeto, Desenvolvimento e Testes.

A fase de Análise engloba o conceito inicial do jogo, *storyboarding*, detalhes do *gameplay*, etc. Já a de Projeto e Desenvolvimento retratam o leiaute do jogo, os gráficos/animações e a programação. A fase de Teste inclui os testes funcionais e do próprio jogo. Essas fases se relacionam entre si e o nível de detalhamento de cada uma delas depende da complexidade, enredo, elementos e objetivos do jogo a ser desenvolvido. Cada uma dessas fases, no processo de desenvolvimento do *FunCopter,* é mostrada na Figura 2. Nas próximas seções, iremos abordar todas essas fases mais detalhadamente.

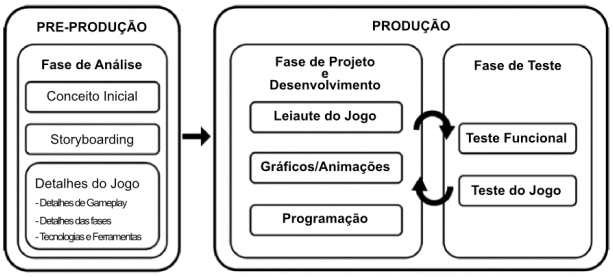

Figura 2: Fases de desenvolvimento do *FunCopter.*

2.1 Fase de Análise

Na Engenharia de *Software*, o objetivo da fase de análise é produzir uma descrição completa do domínio do problema e dos problemas a serem resolvidos [Lichtl and Wurzer 2001].

No entanto, na área de jogos digitais as exigências não são definidas pelo cliente, mas pela concepção do jogo. Mais especificamente, pelo documento de concepção do jogo, no qual são listadas e descritas as suas características fundamentais, por exemplo: jogabilidade, personagens, enredo, mecanismos de pontuação, fluxo de menus, ferramentas, tecnologia, etc. A seguir, apresentamos as principais atividades realizadas em cada fase da Engenharia de *Software* e as lições que aprendemos durante o desenvolvimento do *FunCopter*.

2.1.1 Concepção do Jogo

A concepção inicial do *FunCopter* foi baseada em um jogo simples e clássico conhecido como *SFCave* [Sunflat Jogos 1998], que nos inspirou a criar um pequeno helicóptero de controle remoto, voando através de ambientes temáticos (por exemplo, cozinhas, dormitórios, salas de jantar, banheiros, etc).

Cada cena tem seus detalhes gráficos específicos, diversos obstáculos que devem ser evitados e itens que devem ser coletados durante o jogo. O objetivo é controlar o helicóptero e coletar a maior quantidade de moedas espalhadas pelo cenário o mais rápido possível, antes que a bateria do helicóptero acabe.

No final de cada partida, os jogadores podem enviar suas respectivas pontuações para um *ranking online*, estimulando ainda mais a competição entre eles. Para finalizar cada nível de jogo, é necessário atingir um número específico de pontos, em um tempo limite, de acordo com o nível de dificuldade inicialmente escolhido para o jogo.

2.1.2 *Storyboard*

O *storyboard* é o planejamento básico do jogo, no caso, uma série de desenhos que definem a seqüência das cenas, as quais são usadas para descrever o fluxo básico do jogo. A Figura 3 ilustra um exemplo simples de esboço do *storyboard* criado para o *FunCopter*, contendo seis quadros-chave. Cada quadro representa uma determinada cena ao longo do jogo.

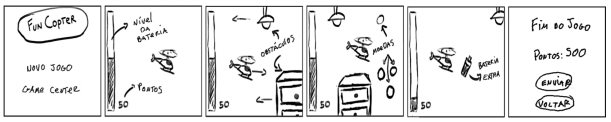

Figura 3: *Storyboard* do *FunCopter*.

2.1.3 Principais Características do Jogo

Inicialmente, para identificar as principais características do *FunCopter,* analisamos alguns aspectos básicos de alguns dos jogos casuais mais populares, entre os quais, *Super Monkey Ball* [Sega 2004], *NinJump* [Backflip Studios 2010], *Cube Runner* [Andy Qua 2008], *Nuts!* [Limbic 2011], *Simple Launch* [Eyesix Games 2011] e *Doodle Jump* [Lima Sky 2009].

Doodle Jump corresponde a um jogo *multiplayer* cujo objetivo é fazer com que o personagem alcance a maior altura possível, pulando de plataformas em plataformas. Já *Nuts!* é um jogo de ação, no qual um esquilo coleta moedas e usa itens especiais enquanto escala árvores. *Simple Launch* é um jogo no qual o jogador controla um foguete voando no céu, evitando a ocorrência de colisões com obstáculos durante o percurso. Esses jogos casuais mencionados são de fácil aprendizado, contudo, ainda muito divertidos de serem jogados.

Várias são também as opções de ferramentas para o desenvolvimento ágil de jogos, entre as quais: *Game Maker*, *Flash* e *Unity* [Guimarães 2011, Ollila *et al.* 2008, Menard 2011, McConnell 1996]. Outra característica que os jogos casuais apresentam são elementos de jogo que podem ser reutilizados durante a partida. Isso inclui não apenas modelos 3D e arquivos de imagem, mas também a dinâmica do jogo, por exemplo, um tiro de canhão ou um certo tipo de vôo de um helicóptero.

Certamente, os jogos casuais são um segmento em expansão na indústria de jogos que têm proporcionado novas oportunidades também para pequenos grupos de desenvolvedores (ou mesmo para indivíduos) [Jie *et al.* 2011, Blackman 2011]. Adicionalmente, apresentam comandos simples e, consequentemente, não precisam de muita dedicação para serem assimilados. Possuem vários níveis de jogabilidade e se encaixam em qualquer gênero. Além disso, oferecem aos usuários a possibilidade de aprender a jogar um jogo em poucos minutos, motivando o entretenimento em momentos esporádicos do dia-a-dia.

Atualmente, a maioria dos jogos casuais mais populares usam o acelerômetro, um recurso para medir a orientação do dispositivo móvel, o qual auxilia no processo de interação com a aplicação. Além disso, outras novas formas de interação com o dispositivo estão surgindo, que vão desde telas sensíveis ao toque até a realidade aumentada [Jie *et al.* 2011].

Outro objetivo que os desenvolvedores de jogos casuais têm em mente é o de atingir o maior número possível de usuários, incluindo os que não apresentam um perfil tradicional de jogador. Com a possibilidade de maior acesso à Internet, os jogos casuais também têm oferecido aos usuários alguma forma de interação com outros jogadores, tal como o uso de *ranking* e pontuação *online,* como motivação adicional ao jogo.

Dada a grande popularidade dos dispositivos móveis, outra questão importante que abordamos na concepção do *FunCopter* é a portabilidade [Cheah and NG 2005, Jie *at al.* 2011, Menard 2011]. Neste contexto, não só o motor *Unity* é bastante flexível e simples de aprender, mas também facilita bastante o ciclo de desenvolvimento, além de oferecer uma excelente integração com o *Blender* [Blender Foundation 2009]. Adicionalmente, vem ganhando bastante popularidade ao longo dos últimos anos, com uma comunidade expressiva de desenvolvedores [Rogers 2012, Blackman 2011]. *Unity* também disponibiliza uma boa documentação (guias de usuário, manuais de referência, tutoriais e vídeos). Para garantir que o nosso jogo casual alcançasse um número considerável de usuários, focamos o seu desenvolvimento para a plataforma iOS.

Levando em consideração essas cinco características básicas anteriormente mencionadas (facilidade de aprendizado, comandos simples e sensíveis ao toque na tela, uso de acelerômetro, possibilidade de interação com outros jogadores e ciclo de desenvolvimento rápido), definimos o núcleo das características do *FunCopter*.

2.2 Fase de Projeto e Desenvolvimento

Com a concepção do *FunCopter* definida, escrevemos o documento de concepção do jogo, utilizado para produzir uma descrição detalhada de cada uma das suas funcionalidades: fases, elementos gráficos, motores, lógica interna, mecanismos de controle, etc.

2.2.1 Leiaute e Gráficos do Jogo

A modelagem geométrica do helicóptero, personagem principal do nosso jogo, mostrado na Figura 4, teve inspiração em desenhos ilustrativos de helicópteros de controle remoto.

Para o processo de modelagem 3D do helicóptero, carregamos o desenho ilustrativo do nosso modelo no *Blender* (Figura 5).

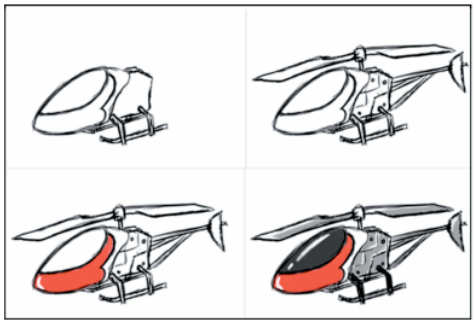

Figura 4: *Concept* do helicóptero.

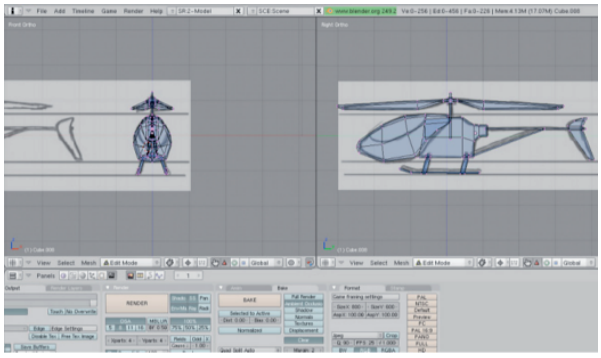

Figura 5: Carregamento da imagem de referência para a modelagem 3D.

Em particular, aplicamos a técnica de modelagem *Box Modeling* para gerar o objeto [Blender Foundation 2009]. Além disso, para economizar tempo e acelerar o processo de produção, utilizamos um modificador de espelhamento (uma ferramenta do *Blender* que automaticamente espelha uma malha ao longo das coordenadas locais *x*, *y*, *z* do centro do objeto). Em seguida, criamos uma textura com base no mapeamento de coordenadas UV, um processo tradicional, que mapeia o objeto 3D em uma textura (plano 2D) para colorir o objeto (Figura 6).

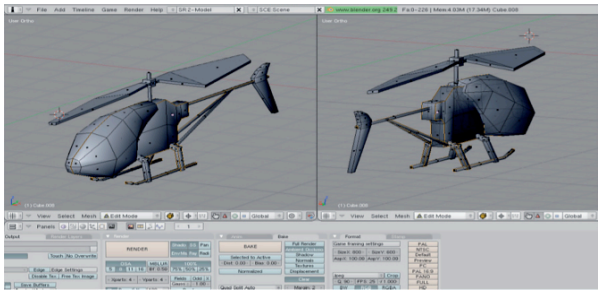

Figura 6: Cortes do mapeamento UV.

FunCopter foi projetado para executar em dispositivos móveis mais populares, os quais, geralmente, possuem telas pequenas. Em decorrência desse fator, todos os outros itens modelados para o jogo são objetos simples, já que uma alta complexidade gráfica pode resultar em queda de desempenho. Sendo assim, optamos por utilizar somente texturas para representar o detalhamento das superfícies dos objetos. O próximo passo foi a exportação da imagem de referência em coordenadas UV, utilizada como guia para a criação das texturas da superfície do helicóptero (Figura 7).

Vale a pena ressaltar a importância de seguir rigorosamente a imagem original de referência, a fim de garantir que ela possa ser exportada corretamente para o motor de jogo a ser utilizado na etapa de desenvolvimento. Antes de exportar o modelo geométrico mostrado na Figura 8 para o *Unity* [Creighton 2010], criamos uma animação da hélice do helicóptero, conforme descrito na Subseção 2.2.2.

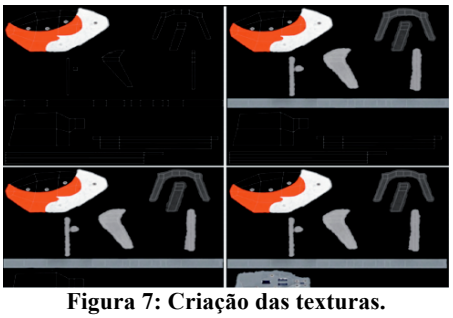

Figura 7: Criação das texturas.

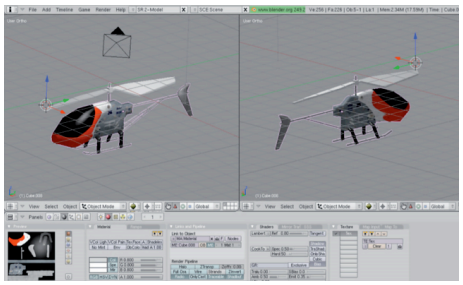

Figura 8: Helicóptero 3D.

2.2.2 Animações

Para obter uma representação perceptível dos aspectos dinâmicos do helicóptero, criamos uma animação utilizando a técnica *keyframe* [Parent 2002], um processo simples e eficaz que envolve a alteração das posições de alguns objetos e a inclusão de quadros-chave. Com isso, os quadros intermediários (ou *inbetweens*) são gerados interpolando-se linearmente as posições definidas nos quadros principais. Por exemplo, na hélice de helicóptero, adicionamos quadros-chave para representar o seu movimento de rotação.

Consideramos este tipo de animação como um exemplo típico de um método rápido que pode ajudar os desenvolvedores de jogos voltados para plataformas móveis. Para gerar a animação e facilitar ainda mais o

processo, criamos inicialmente uma estrutura de esqueleto, que ajuda na manipulação do objeto 3D durante a animação (definimos um "osso" para a carcaça do helicóptero e um outro para a sua hélice, como mostra a Figura 9).

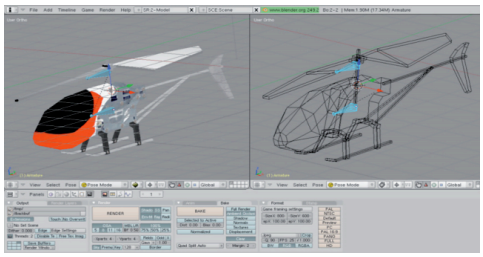

Figura 9: Esqueleto do helicóptero.

Para representar o movimento das hélices do helicóptero adicionamos um quadro-chave no "osso" da hélice a cada 90°, até completar uma rotação anti-horária de 360° em torno do eixo *z*. Depois de terminar a modelagem 3D, o mapeamento de textura e a animação do helicóptero, o passo seguinte foi importar os objetos do *FunCopter* para o *Unity* (Figura 10).

Figura 10: Um dos quartos modelados para o *FunCopter*.

2.2.1 Programação

Como mencionado anteriormente, para o desenvolvimento do *FunCopter* escolhemos o *Unity* [Creighton 2010]. Seu editor executa no *Windows* e *Mac OS X*. Pode produzir jogos para *Windows*, *Mac*, *Wii*, *iOS* (*iPhone, iPod Touch* e *iPad*), *Android*, *Xbox 360*, *Playstation 3* e para, praticamente, todos os navegadores. Uma vez que o *Unity* pode exportar o mesmo jogo para várias plataformas, o desenvolvedor também poupa bastante tempo durante esta fase. Outro ponto positivo que identificamos no *Unity*, foi a forma produtiva de desenvolvimento oferecido pelo seu editor "*What You See Is What You Get*", muito útil para prototipagem rápida de jogos.

Como descrevemos anteriormente, o *Unity* tem uma excelente integração com o *Blender*. Uma vez que optamos por usar o *Blender* para modelar objetos em 3D e o *GIMP* (um editor de gráficos 2D *open-source*) para a criação das texturas e componentes da interface, tivemos uma redução dos custos para o desenvolvimento do projeto.

Para o armazenamento *online* do *ranking*, optamos pelo *Game Center* [Apple 2010], uma ferramenta para auxiliar nos aspectos sociais de jogos, essencialmente devido à sua facilidade de uso e à baixa complexidade do *FunCopter*. O *Game Center* pode ser utilizado até para conectar os jogadores. Por exemplo, os usuários podem convidar seus amigos para jogar um jogo, adquirir itens especiais e/ou estabelecer metas, e visualizar suas respectivas pontuações *online* (*ranking*). Para gerenciar esses recursos usamos o *iTunes Connect*, ferramenta *web* da *Apple* utilizada para gerenciar todos os aspectos do *Game Center*, tornando possível, por exemplo, a criação de tabelas de classificação *online* e o envio/recebimento das pontuações através do *SDK* do *Game Center,* disponível na plataforma *iOS*.

Para a implementação do *FunCopter*, utilizamos algumas outras estratégias com o objetivo de obter o melhor desempenho possível. Por exemplo, uma parte importante do cenário do *FunCopter* é um túnel sem fim, no qual o helicóptero pode voar livremente. Para modelar esse túnel, procedemos da seguinte maneira: (1) criamos e conectamos dois planos para representar as imagens de fundo para o jogo, com texturas aplicadas em cada um deles, de tal forma a criar uma ilusão de continuidade geométrica; e (2) alternamos a exibição desses dois planos no *frustum* da câmera, ao mesmo tempo em que movimentamos os planos para a esquerda, gerando uma ilusão de *looping*, como mostrado na Figura 11.

```
using UnityEngine;
using System.Collections;

public class CenarioScript : MonoBehaviour {

    public int speed;
    public int offset;
    public Vector3 startPosition;

    void Update () {
        float amountToMove = speed * Time.deltaTime;
        transform.Translate(Vector3.left * amountToMove);
        if(transform.position.x < offset) {
            this.ResetPosition();
        }
    }

    void ResetPosition() {
        Vector3 position = transform.position;
        position.x = startPosition.x;
        position.y = startPosition.y;
        transform.position = position;
    }
}
```

Figura 11: *Script* para criar o efeito de *looping* com as imagens de fundo.

Aplicamos também uma técnica de reutilização de objetos para os obstáculos gráficos que são gerados no cenário do jogo, definida da seguinte forma: sempre que um objeto deixa de ser visível pela câmera, reaproveitamos este mesmo objeto para representar um novo que aparecerá como obstáculo do helicóptero, evitando a necessidade de criação contínua de novos objetos. A estratégia que utilizamos na implementação dos obstáculos é bastante semelhante ao esquema criado para representar o movimento do plano de fundo do cenário. No entanto, a *ResetPosition* gera novas posições aleatórias para o objeto durante a animação, como mostrado na Figura 12.

```
public void ResetPosition() {
    Vector3 position = transform.position;
    position.x = Random.Range(MIN_RANGE, MAX_RANGE);
    position.y = Random.Range(minYPosition, maxYPosition);
    transform.position = position;
}
```

Figura 12: *Script* para gerar obstáculos gráficos no jogo.

Para gerar um pouco mais de realismo gráfico, acrescentamos um efeito especial que simula a fumaça na cauda do helicóptero, utilizando o sistema de partículas do *Unity*. Esse efeito especial consiste em três componentes: um emissor de partículas, um animador de partículas e um processador de partículas. As partículas no *FunCopter* são representadas por *billboards*. Esta é uma boa recomendação para a geração de fumaça e de efeitos especiais relativamente simples que simulam explosão. No *FunCopter*, cada vez que o jogador coleta uma moeda (isto é, o helicóptero atinge uma moeda), um *script* de explosão é usado para criar um efeito visual, gerando um maior realismo.

Os cálculos relativos à detecção e resposta a colisões podem se tornar custosos (com relação à memória e processamento) e, muitas vezes, dependendo da complexidade, são inviáveis de serem implementados em dispositivos móveis. De fato, a detecção de colisão é um dos grandes gargalos que influenciam no desempenho das aplicações gráficas interativas 3D, cujas taxas de exibição devem ser elevadas e constantes (30 a 60 quadros/s), limitando o tempo disponível para a fase de detecção de colisão.

Uma maneira simples de verificar colisões usando o *Unity* é adicionando-se o componente de *Rigidbody*, que corresponde a um objeto cujo movimento é controlado por leis físicas (por exemplo, movimentos de helicópteros ou aviões, via aplicação de forças por intermédio de *script*s). Para o sistema de colisão modelado no *FunCopter*, adicionamos um *Collider* e um *Rigidbody*, habilitado como *IsKinematic*. Isso impede que o helicóptero sofra a influência das leis físicas, mas permite que receba todos os eventos disparados durante uma colisão. Para mover o objeto, modificamos o seu componente de transformação, ao invés de utilizar a aplicação de forças.

Para capturar as moedas, adicionamos um *script* no objeto que representa o helicóptero, que tem o método *OnTriggerEnter* implementado. Este método é chamado automaticamente pelo *Unity* quando dois objetos marcados como *Triggers* colidem. Neste método é verificado se o objeto que colidiu com o helicóptero é uma moeda. Em caso afirmativo, uma animação chamada *ExplosionCoin* é criada na posição em que a moeda estava.

Em seguida, o método *ResetPosition* é chamado novamente para reciclar o objeto da moeda (Figura 12). Então, o jogador é premiado com 50 pontos para cada moeda obtida, como mostrado na Figura 13. Neste *script*, o *prefab* é um objeto previamente definido e armazenado no projeto, o qual pode ser reutilizado a qualquer momento nas cenas da animação. Vários objetos idênticos (por exemplo, moedas) podem ser criados a partir de um *prefab*, permitindo ao desenvolvedor que instancie várias moedas idênticas e posicione-as em lugares diferentes da cena, de forma rápida.

```
void OnTriggerEnter(Collider otherObject) {
    if(otherObject.gameObject.tag == "coin") {
        Instantiate(Resources.Load("Prefab/ExplosionCoin"), otherObject.transform.position, transform.rotation);
        ((MoedaScript)otherObject.gameObject.GetComponent(typeof(MoedaScript))).ResetPosition();
        score+=50;
    }
}
```

Figura 13: *Script* para coletar as moedas durante a colisão.

Além do desenvolvimento da mecânica do jogo, um outro aspecto fundamental é o controle da taxa de exibição de quadros. Uma taxa de quadros por segundo constante pode ser uma boa solução para jogos casuais em dispositivos móveis. Por padrão, utilizamos 30 quadros/s para a execução do *loop* de renderização do *FunCopter*.

Outro aspecto importante é que o *Unity* no *iOS* consulta o acelerômetro 60 vezes por segundo. No entanto, para que os jogos executem satisfatoriamente em dispositivos móveis, muitas vezes é recomendado que sejam definidas configurações mais modestas. Ao evitar que o jogo execute com os recursos computacionais no limite, o tempo de bateria, por exemplo, pode ser poupado. A fim de não prejudicar a usabilidade do usuário, priorizamos a interação via toque e o uso do acelerômetro, em detrimento da renderização de alta qualidade (mais realista).

Jogar *FunCopter* em um *smartphone* fazendo uso de recursos modernos, tais como a movimentação do helicóptero via acelerômetro, aumenta a qualidade da experiência do usuário. Ao girar o dispositivo para a direita ou esquerda, o helicóptero é movido nessas direções, respectivamente, evitando dessa forma, a colisão com obstáculos do cenário. Além disso, implementamos os movimentos verticais do helicóptero, por exemplo, ao tocar na tela do dispositivo ele se move para cima e, fazendo o oposto, para baixo. Note que se um jogo usa muitas entradas do acelerômetro, isto pode ter um impacto negativo sobre o seu desempenho geral.

Usamos uma classe da *Unity*, chamada *Input*, que facilita a captura de tais entradas. No caso, usamos o *Input.acceleration.x*, o *Input.acceleration.y* e o *Input.acceleration.z*, respectivamente, para controlar os movimentos horizontais, verticais e de profundidade do helicóptero (isto é, para retornar os valores correntes do acelerômetro destes três eixos). Usando o valor armazenado no acelerômetro, o *script* mostrado na Figura 14 é chamado para estimar a distância que o helicóptero deve ser movido. Comandos de toque na tela são detectados utilizando-se o *Input.touchCount*,

que retorna o número de eventos de toque, em um dado momento.

```
float amoutToMove = playerSpeed * Input.acceleration.y * Time.deltaTime;
transform.Translate(Vector3.right * amoutToMove);
```

Figura 14: *Script* para capturar os dados do acelerômetro e movimentar o helicóptero.

Para o desenho da *HUD* (ou *Heads-Up Display*) do ícone da bateria, um *script* foi adicionado à câmera, conforme mostra a Figura 15. O evento *OnGUI* é chamado para desenhar todos os componentes da interface do usuário, tais como as três peças do ícone da bateria. Para cada quadro do jogo, o método de atualização é chamado para exibir o nível de bateria disponível no helicóptero e decrementá-lo ao longo do vôo e manobras pelo cenário. De acordo com o nível de bateria disponível, os atributos de visualização do ícone que representa a bateria também são atualizados.

```
void OnGUI() {
    GUI.DrawTexture(batteryPart1, texturePart1, ScaleMode.StretchToFill);
    GUI.DrawTexture(batteryPart2, texturePart2, ScaleMode.StretchToFill);
    GUI.DrawTexture(batteryPart3, texturePart3, ScaleMode.StretchToFill);
    GUI.DrawTexture(batteryFuel, textureFuel, ScaleMode.StretchToFill);
}

void Update () {
    PlayerScript.actualFuel-=60.0f * Time.deltaTime;
    if(PlayerScript.actualFuel < 0) {
        PlayerScript.actualFuel = 0;
    }
    batteryFuel.height= (285.0f / PlayerScript.maxFuel) * PlayerScript.actualFuel;
}
```

Figura 15: *Script* para desenhar o ícone da bateria e mostrar o nível de energia.

3. Fase de Teste

Apesar de sua jogabilidade simples, *FunCopter* ainda é um jogo casual desafiador. Testes iniciais mostraram que as atividades dos jogadores, ao tentar marcar o maior número de pontos possível, exigiam que jogassem de forma precisa e bastante rápida, motivando a competição entre eles.

Constatamos que o nível de jogabilidade variou de acordo com as diferentes faixas etárias de cada jogador. No geral, as experiências vivenciadas pelos jogadores durante o jogo foram extremamente positivas, evidenciando a aceitação e o interesse pelo *FunCopter*. Mais ainda, os usuários se divertiram bastante enquanto jogavam contra outros jogadores, especialmente quando se deparavam com situações que exigiam uma tomada de decisão imediata para conseguir mais moedas, ou para evitar colisões com obstáculos (entre o helicóptero e os objetos do cenário), ao longo do túnel.

4. Discussão e Trabalhos Futuros

Na área de jogos digitais, o processo de criação e de desenvolvimento é representado por um *pipeline*, a partir dos dados 2D, até a geração do produto final.

Nossa percepção geral, relativa ao *FunCopter,* é que a mecânica do jogo funcionou de forma eficiente, em diferentes plataformas: *iPhone 3G*, *iPhone 3G*, *iPhone 4/4S* e *iPad*. Contudo, observamos que o desempenho do *iPhone 3G* foi inferior aos desempenhos dos outros modelos de dispositivos, provavelmente, devido à sua menor capacidade de processamento.

Além das diferenças inerentes dos dispositivos quanto à capacidade de processamento, também verificamos questões relativas a diferentes resoluções de tela. Identificamos que os dispositivos com resoluções de tela superiores (por exemplo, *iPad* e *iPhone 4*) exigiram um maior nível de detalhe gráfico, com texturas de alta resolução e efeitos especiais mais realistas. Ou seja, se um jogo for voltado para dispositivos com tela de alta resolução, a geração de imagens com uma maior qualidade seria a mais recomendada. Uma boa prática, portanto, seria a geração de imagens de alta resolução, as quais poderiam ser opcionalmente redimensionadas para resoluções mais baixas, dependendo das necessidades e perfis da plataforma de execução escolhida.

O uso de um motor gráfico, como o *Unity*, aumentou consideravelmente a produtividade da equipe de desenvolvimento do *FunCopter*, principalmente devido ao seu editor visual e à possibilidade de exportar o jogo para várias outras plataformas. Em particular, nossa equipe de desenvolvimento foi composta por somente duas pessoas e, de fato, o *Unity* mostrou-se bastante adequado às necessidades de desenvolvimento do jogo que projetamos.

Além disso, com poucos ajustes de código, é possível fazer com que o *Unity* também possa ser usado para simplificar a tarefa de portar um jogo de *smartphone* para ser executado em um *browser* (por exemplo, *Firefox*, *Chrome*, *IE*, *Safari*), ou até mesmo para criar um executável em computadores pessoais tradicionais, com sistema operacional *Mac OS* e *Windows* (e, muito em breve, *Linux*). Portanto, constatamos que o *Unity* não requer muito esforço do desenvolvedor para trabalhar com múltiplas plataformas. Na verdade, para portar um jogo de um dispositivo móvel para um outro tipo de plataforma, a maior dificuldade diz respeito à interface de entrada, uma vez que os tipos de comandos para o controle do jogo variam consideravelmente entre as plataformas (teclados, *touch screen*, acelerômetro, etc).

Para que *FunCopter* possa executar tanto em computadores pessoais tradicionais, quanto em dispositivos móveis, antecipamos que seria necessário implementar uma camada transparente para o desenvolvedor modificar ou alterar a forma de entrada, ou até mesmo substituir o código original da entrada de teclado por um novo, com suporte à tela sensível ao toque e acelerômetro, por exemplo. Em algumas circunstâncias, portar para uma nova plataforma pode ser um processo menos ágil e menos direto (por exemplo, nos casos em que o código do jogo faz uso de

um recurso específico, disponível em apenas uma das plataformas). No caso do *FunCopter*, utilizamos o *Game Center*, uma rede social voltada para jogadores, exclusiva para os dispositivos *iOS*. Consequentemente, portar o *FunCopter* para celulares *Android*, por exemplo, exigiria o uso de uma outra plataforma social, tal como o *OpenFeint* [Aurora Feint 2009], disponível para o *iOS* e *Android*.

Como trabalhos futuros, planejamos evoluir a qualidade do *FunCopter*. Mais especificamente, o realismo do jogo poderia ser melhorado com o uso de mapeamento de texturas específicas para efeitos de iluminação e sombreamento. Além disso, uma avaliação mais detalhada sobre a sua jogabilidade (baseada na experiência do usuário) e sobre o grau de satisfação do usuário (em relação a características específicas, tais como gráficos, estória, etc) poderia ser realizada. Isso envolveria, por exemplo, o uso de heurísticas para jogos casuais, com foco em usabilidade, mobilidade e jogabilidade [Korhonen e Koivisto 2006]. Um dos objetivos subsequentes seria então avaliar quais aspectos da implementação [Evans 2004] poderíamos otimizar para melhorar a concepção do *FunCopter*, bem como a experiência dos usuários, particularmente os que pertencem à faixa etária de 7 a 12 anos, clientes-alvo do nosso jogo.

Agradecimentos

Daniel Valente de Macedo e Maria Andréia Formico Rodrigues gostariam de agradecer à FUNCAP-CE (Processo No. 0008-00096.01.29/12) e ao CNPq (Processo No. 310434/2010-6), respectivamente, pelo apoio financeiro recebido.

Referências

APPLE. 2010. Game Center. Disponível em http://www.apple.com/game-center/. Último acesso em 17/07/2012.

ANDYQUA. 2008. Cube Runner. Disponível em http://itunes.apple.com/app/cuberunner/id284596345?mt/8. Último acesso em 17/07/2012.

AURORA FEINT. 2009. OpenFeit. Disponível em http://openfeint.com/. Último acesso em 17/07/2012.

BACKFLIP STUDIOS 2010. NinJump. Disponível em http://itunes.apple.com/us/app/ninjump/id379471852?mt=8. Último acesso em 17/07/2012.

BELL, M. , CHALMERS, M. , BARKHUUS, L. , HALL, M., SHERWOOD, S. , TENNENT, P. , BROWN, B., et al. 2006. Interweaving mobile games with everyday life. In Proceedings of the 2006 CHI. ACM, New York, 417–426.

BLENDER FOUNDATION. 2009. Blender. http://www.blender.org.

CHEAH, T.C.S. and NG, K-W. 2005. A practical implementation of a 3-D game engine. In Proceedings of the 2005 Computer Graphics, Imaging and Vision: New Trends (CGIV). IEEE. Washington, DC, 351-358.

CHEHIMI, F., COULTON, P., ANDEDWARDS, R. 2008. Evolution of 3D mobile games development. Pervasive Ubiquit. Computing 12, 19–25.

CREIGHTON, R. H. 2010. Unity Ref: Unity 3D game development by example beginner's guide. Packt Publishing.

EVANS, E. 2004. Domain-driven design—tackling complexity in the heart of software. Addison Wesley.

EYESIXGAMES. 2011. Simple launch. Disponível em http://itunes.apple.com/us/app/a-simple launch/id425901529?mt=8. Último acesso em 17/07/2012.

JHINGUT, M.Z., GHOORUN, I.M., NAGOWAH , S.D., MOLOO , R., ANDNAGOWAH, L. 2010. Design and development of 3D mobile games. In Proceedings of the 3rd International Conference on Advances in Computer-Human Interactions, 119–124.

LICHTL,B. AND WURZER, G. 2001. Software engineering in games. Institute of Computers Graphics. Technical University Vienna. Seminar lecture handout, 1-23. http://www.cg.tuwien.ac.at/courses/Seminar/SS2001/se/sein games.pdf.

LIMASKY. 2009. Doodle Jump. Disponível em http://itunes.apple.com/us/app/doodle-jump-be-warned-insanely/id307727765?mt/8&ign-mpt/uo%3D6. Último acesso em 17/07/2012.

LIMBIC. 2011. Nuts! TM. Disponível em http://itunes.apple.com/us/app/nuts/id430404688?mt=8. Último acesso em 17/07/2012.

PARENT, R. 2002. Computer Animation—algorithms and techniques. Academic Press.

KORHONEN,H. AND KOIVISTO, E. M. I. 2006. Playability heuristics for mobile games. In Proceedings of the 8th Conference on Human-computer Interaction with mobile devices and services (MobileHCI). ACM, New York, 9–16.

KURKOVSKY, S. 2009. Engaging students through mobile game development. In Proceedings of the 40th ACM technical symposium on Computer science education (SIGCSE). ACM, New York, 44–48.

SEGA. 2004. Super Monkey Ball. Disponível em http://itunes.apple.com/app/super-monkey ball/id281966695?mt=8#. Último acesso em 17/07/2012.

SUNFLATGAMES. 1998. SFCave Game. Disponível em http://www.sunflat.net/iphone/app/sfcave/?lang=en. Último acesso em 17/07/2012.

BLACKMAN, S. 2011. Beginning 3D game development with Unity: all-in-one, multi-platform game development. Apress Berkely.

GUIMARÃES, M. 2011. Game development with Game Maker, Flash and Unity. In Proceedings of the 49th Annual Southeast Regional Conference (ACM-SE). New York, 1-9.

JIE, J., YANG, K., HAIHUI, S. 2011. Research on the 3D game scene optimization of mobile phone based on the Unity 3D engine. In Proceedings of the 2011 International Conference on Computational and Information Sciences (ICCIS). IEEE, Washington, D.C., 875-877.

ROGERS, M. 2012. Bringing Unity to the classroom. *Journal of Computing Sciences in Colleges*, 27 (5), 171-177.

OLLILA, E., SUOMELA, R., HOLOPAINEN, J. 2008. Using prototypes in early pervasive game development. Computers in Entertainment (CIE). ACM, New York, 6 (2), 1-17.

BATES, B. 2004. Game Design, Second Edition. Thomson Course Technology.

MENARD, M. 2011. Game development with Unity. Course Technology PTR.

MCCONNELL, S. 1996. Rapid development: taming wild software schedules. Microsoft Press.