Faculdade de Tecnologia de Americana

Curso Superior de Tecnologia em Desenvolvimento de Jogos Digitais

DESENVOLVIMENTO DO JOGO "WPBOUNCE" PARA WINDOWS PHONE USANDO XNA

BRUNO CESAR MENDES DA SILVA

Americana, SP

2012

Faculdade de Tecnologia de Americana

Curso Superior de

DESENVOLVIMENTO DO JOGO "WPBOUNCE" PARA WINDOWS PHONE USANDO XNA

BRUNO CESAR MENDES DA SILVA

bc.mendess@gmail.com

Trabalho de Conclusão de Curso desenvolvido em cumprimento à exigência curricular do Curso Superior de Tecnologia em Desenvolvimento de Jogos Digitais, sob a orientação do Prof. Me. Kléber de Oliveira Andrade.

Área: Jogos Digitais

Americana, SP

2012

BANCA EXAMINADORA

Prof. Me. Kleber Andrade (Orientador)

Prof. Fernando José Ignácio

Prof. Me. Cleberson Eugenio Forte

AGRADECIMENTOS

Em primeiro lugar gostaria de agradecer a Deus por ter me dado o dom da vida e oportunidade de estudar em uma instituição de excelência como é a FATEC.

A minha família e namorada, por terem me apoiado e torcido por mim nesta etapa da minha vida.

Ao meu professor e orientador Kléber de Oliveira Andrade que me auxiliou e sempre esteve a disposição para me orientar e sanar minhas dúvidas.

A todos os meus professores da FATEC que me ensinaram, orientaram e fizeram com que eu chegasse até este ponto do curso. Em especial ao professor Cleberson Eugenio Forte, que de forma não oficial foi meu co-orientador neste trabalho.

E por último, mas não menos importante, a todos que contribuíram de forma direta ou indireta para eu conseguisse finalizar este documento.

DEDICATÓRIA

À minha família, minha namorada, meus professores e aos amigos.

RESUMO

Este presente texto contextualiza sobre o desenvolvimento de jogos para a plataforma Windows Phone utilizando o XNA Framework, ambos desenvolvidos pela Microsoft, detalhando diversos aspectos da plataforma e todas as camadas deste framework de desenvolvimento de jogos. Alem de discutir a respeito de conceitos teóricos referentes ao assunto citado, o presente trabalho também mostra na pratica os principais pontos da codificação e concepção de um jogo bidimensional do tipo arcade, o WPBounce, o qual teve sua ideia principal retirada e evoluída a partir do game KBounce (desenvolvido pela KDE).

Palavras Chaves: Jogos, XNA, Windows Phone.

ABSTRACT

This present text contextualizes about game developing for the Windows Phone platform using the XNA Framework, both developed by Microsoft, it's detailing various aspects of the platform and all layers of the framework used for game development. Besides discussing about the theoretical concepts relating to the subject cited above, the present study also shows in practice the main points of coding and design of a two-dimensional arcade game, the WPBounce, which has its main idea evolved and extracted from the KBounce game (developed by KDE).

Keywords: Games, XNA, Windows Phone.

SUMÁRIO

1 INTRODUÇÃO 12

2 WINDOWS PHONE 14

2.1 METRO 15

2.1.1 HUBS E PANORAMAS 16

2.1.2 NAVEGAÇÃO SIMPLIFICADA 17

3 XNA FRAMEWORK 19

3.1 ESTRUTURA 20

3.1.1 CAMADA DE JOGOS 21

3.1.2 CAMADA DE EXTENSÕES 21

3.1.3 CAMADA DE NÚCLEO 22

3.1.4 CAMADA DE PLATAFORMA 22

3.2 CICLO DE VIDA 22

3.3 PRINCIPAIS COMPONENTES 24

3.3.1 MECÂNICA 24

3.3.2 GRÁFICOS 24

3.3.3 ENTRADA DO JOGADOR 25

3.3.3.1 DESKTOP 25

3.3.3.2 CONSOLE 26

3.3.3.3 DISPOSITIVO MÓVEL 26

4 PROJETO DO JOGO 27

4.1 FERRAMENTAS UTILIZADAS 28

4.1.1 VISUAL STUDIO 2010 28

4.1.2 XNA FRAMEWORK 4.0 28

4.1.3 WINDOWS PHONE SDK 7.1 29

4.1.4 WINDOWS PHONE EMULATOR 30

4.2 IDÉIAS E RASCUNHO DO JOGO 32
4.3 DEFINIÇÕES DO JOGO 32
4.4 A MECÂNICA DO JOGO.....33
4.5 A INTERFACE COM O USUÁRIO 33
5 CONCEPÇÃO DO JOGO 35
6 CONSIDERAÇÕES FINAIS 39
7 REFERÊNCIAS BIBLIOGRÁFICAS 40

LISTA DE FIGURAS

Figura 1: Modelo de um dispositivo com WP. 15

Figura 2: Demonstração de como é o conceito HUBs e panoramas por traz dos panos. 16

Figura 3: HUB de pessoas do WP.....17

Figura 4: Zune do WP. 18

Figura 5: Tela de templates de projetos do Visual Studio 2010. 19

Figura 6: A plaforma XNA e seus componentes e bibliotecas.....21

Figura 7: Ciclo de vida de um jogo em XNA.....23

Figura 8: Jogo KBounce desenvolvido pela KDE. 27

Figura 9: Emulador do WP: (a) tela do emulado; (b) ferramenta de simulaçaõ do acelerômetro. 30

Figura 10: Aba de localização. 31

Figura 11: Aba Screenshot.....31

Figura 12: Interface do WPBounce com o tema vermelho (cor de destaque) e escuro (tela de fundo) do dispositivo.....32

Figura 13: Interface do WPBounce com o tema azul (cor de destaque) e claro (tela de fundo) do dispositivo. 32

Figura 14: Rascunho da tela final do jogo. 34

Figura 15: Diagrama de classes do jogo WPBounce 35

Figura 16: Trecho de código do WPBounce que mostra como foi manipulado o toque na tela. 37

Figura 17: Método utilizado para tratar a colisão da bolinha com os blocos. 38

LISTA DE ABREVIATURAS E SIGLAS

API	*Application Programming Interface*
C#	C Sharp, linguagem de programação
IDE	*Integrated Development Environment*
GPS	*Global Positioning System*
MDX	Managed DirectX
SDK	*Software Development Kit*
WP	Windows Phone
XNA	*XNA's Not Acronymed*
KDE	*K Desktop Environment*

1 INTRODUÇÃO

Este trabalho discutirá a respeito do desenvolvimento de jogos para o mais novo sistema operacional (lançado em 21 de outubro de 2010) para dispositivos móveis criado pela Microsoft, o Windows Phone (WP), e seu *framework* de desenvolvimento de jogos, o XNA Framework, os quais atualmente se encontram nas versões 7.5 e 4.0, respectivamente.

Durante o decorrer deste documento serão abordadas as principais etapas necessárias para o desenvolvimento e criação de jogos bidimensionais para WP, e uma breve apresentação desta plataforma, seu SDK (*Software Development Kit*, ou pacote de desenvolvimento de software) de desenvolvimento e o *framework* de criação de jogos da Microsoft, o XNA, os quais quando combinados com a IDE (do inglês, *Integrated Development Environment* - ambiente de desenvolvimento integrado) Visual Studio 2010 se tornam uma fortíssima ferramenta para desenvolvimento de jogos.

O rumo para o qual este trabalho se direciona foi escolhido por considerar este assunto algo extremamente atual, inovador e, por intermédio do XNA, muito simples e amigável, diferentemente do que acontece em outras ferramentas de desenvolvimento de jogos como Managed DirectX[1], OpenGL[2] e outras APIs (do inglês, *Application Programming Interface* - Interface de Programação de Aplicativos) criadas para estes fins.

O objetivo deste trabalho é mostrar alguns conceitos básicos e etapas do desenvolvimento de jogos para WP utilizando XNA Game Studio e a linguagem C#. Também tem como alvo a demonstração na prática de alguns pontos do conteúdo abordado, através de um minitutorial (Capítulo 5), o qual mostrará alguns passos que necessitam de maior atenção na concepção do jogo WPBounce, o qual é bastante semelhante ao jogo KBounce[3] da KDE.

---

[1] Managed DirectX é um conjunto de bibliotecas criadas pela Microsoft a fim de melhorar a interface entre o desenvolvedor e a manipulação do hardware de um computador, o qual utiliza Windows como sistema operacional (Jones, 2004).

[2] OpenGL é uma biblioteca aberta e multiplataforma desenvolvida com o propósito de ser o mediador entre o desenvolvedor e o gerenciamento de hardware, assim facilitando o desenvolvimento de aplicações que utilizem recursos gráficos, como por exemplo jogos (Astle e Hawkins, 2001).

[3] KBounce é um jogo de um único jogador do tipo árcade desenvolvido pela KDE. Disponível em: <http://www.kde.org/applications/games/kbounce/>. Acesso em: 05/05/2012.

O capitulo dois abordará o sistema operacional WP, seu padrão de interface e seus requisitos. O capitulo três discutirá a respeito do Microsoft XNA Framework e sua estrutura. No capitulo quatro, será apresentado o projeto do jogo WPBounce, a sua mecânica, suas definições e as ferramentas utilizadas no desenvolvimento do mesmo entre outros assuntos. O capitulo cinco mostra a concepção do jogo WPBounce focando em como o mesmo foi estruturado. E finalmente o capitulo seis, apresenta a conclusão do projeto e os trabalhos futuros.

2 WINDOWS PHONE

O WP é o novo sistema operacional para dispositivos móveis desenvolvidos pela Microsoft, foi lançado no dia 21 de outubro de 2010, e graças ao novo conceito de interface criado pela mesma, Metro, o WP ganhou uma interface extremamente inovadora, agradável e intuitiva (WILLIAMS, C. G. e CLIGERMAN, G. W. , 2011).

Para evitar possíveis problemas relacionados à dispositivos com baixa capacidade de processamento ou com recursos insuficientes, a Microsoft criou uma lista de recursos mínimos[4] que seus fabricantes devem seguir rigorosamente na fabricação de seus produtos para que os mesmos possam ter funcionando o WP como sistema operacional.

Os recursos mínimos que um aparelho deve ter para obter autorização para a instalação do WP são:

- **Tamanho do display:** o dispositivo deve ter uma tela de 480 x 800 *pixels*;
- **Capacidade de toques:** o aparelho deve suportar quatro ou mais toques simultâneos na tela;
- **Sensores:** sensor de localização, acelerômetro, luz de ambiente, entre outros;
- **Câmera:** deve ter câmera digital com no mínimo cinco *megapixel* e *flash*;
- **Botões:** no mínimo três botões frontais, o *Back Button* (serve para voltar à ultima pagina aberta), *Start Button* (vai para pagina inicial) e o *Search Button* (abre Bing, motor de pesquisa na internet desenvolvido pela Microsoft, a fim de que o usuário possa realizar buscas), como mostrado na figura 1;
- **Memória:** no mínimo 256MB de memória RAM e oito GB ou mais de memória para armazenamento;

[4] O WP possui uma lista de recursos mínimos para garantir um bom funcionamento do sistema. Lista disponibilizada em: <http://msdn.microsoft.com/en-us/library/ff637514(v=vs.92).aspx>.

Alem dessas especificações citadas, a Microsoft também definiu um modelo padrão para os dispositivos, por este motivo todos os aparelhos possuem semelhante modelo, com um *display* de acordo com o tamanho informado na lista de requisitos e os três botões, *Back Button*, *Start Button* e *Search Button*. A Figura 1 exemplifica melhor como deve ser o modelo de um dispositivo com WP.

Figura 1: Modelo de um dispositivo com WP (MACHADO, 2011).

2.1 METRO

Pode-se dizer que Metro é um conceito de interface e foi implementado no WP. Este padrão é algo extremamente inovador no que diz respeito a navegação, janelas, ícones, entre outros pontos. Segundo Martins (2012), é possível afirmar que aplicações baseadas no Metro serão extremamente ricas e interessantes para a experiência do usuário (*User Experience*).

O Metro possui alguns principais conceitos em sua teoria que devem ser destacados, os quais serão abordados nos próximos subtópicos.

2.1.1 HUBS E PANORAMAS

Para mostrar todo o conteúdo na tela foi criado o conceito de HUBs, o qual exibe as informações de maneira inteligente com uma visão panorâmica, como exemplificado na Figura 2. Para visualizar o restante basta deslizar o dedo na tela movendo para um dos lados, desta forma a tela acompanha o movimento e disponibiliza todo o conteúdo para o usuário, porem uma parte do panorama por vez (Maciejewsky, 2011).

Figura 2: Demonstração de como é o funcionamento do conceito HUBs e panoramas (MACIEJEWSKY, 2011).

Além de ser uma forma de visualização de conteúdo, os HUBs também possuem uma grande e inovadora funcionalidade, eles agregam em um único local informações de diversos aplicativos, como por exemplo o HUB de pessoas, o qual reúne em um único local informações sobre contatos, sobre o que eles andam fazendo nas redes sociais, ou quais foram os contatos das ultimas ligações realizadas, como mostra a Figura 3.

Figura 3: HUB de pessoas do WP (MACIEJEWSKY, 2011).

2.1.2 NAVEGAÇÃO SIMPLIFICADA

Com ajuda dos HUBs e panoramas já citados, as interfaces Metro ganharam um novo conceito de navegação simplificada, a qual consiste em disponibilizar o maior número de informações referentes a um assunto em um único local (panorama), como exemplificado na Figura 4, a qual mostra o panorama do Zune do WP (aplicativo de reprodução de musicas, vídeos, rádios,...). Na primeira coluna fica um menu, o qual disponibiliza ações para que o usuário possa executa-las no aplicativo, já na segunda e terceira parte é exibido um histórico das últimas músicas ou vídeos reproduzidos, na quarta o que há de novo e assim por diante. Enfim, todo conteúdo em um único lugar ao rápido alcance de alguns toques e/ou deslizes, proporcionando assim, uma navegação simplificada (MARTIN, 2012).

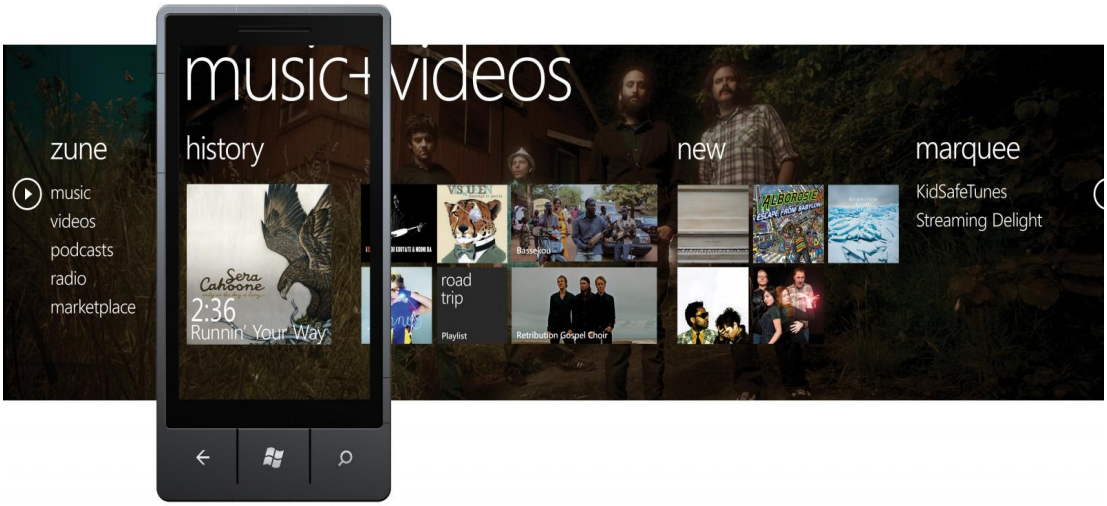


Figura 4: Zune do WP (KRAVITZ, 2010).

3 XNA FRAMEWORK

Segundo Júnior (2011), o XNA (sigla em inglês que significa *XNA's Not Acronymed*), como já citado anteriormente, é um *framework* que visa facilitar o desenvolvimento e a criação de jogos para PC (Windows Vista, Windows Seven...), Xbox 360 e WP.

O XNA pode ser considerado como uma evolução do MDX. Esse se encontra na versão 4.0 (2010), a qual trouxe consigo WP SDK 7.1, um pacote para desenvolvimento de aplicações para dispositivos móveis com WP. O XNA Game Studio 4.0 é uma extensão do Visual Studio, utiliza o XNA Framework e suportado por todas as versões do Visual Studio 2010 com suporte para C#, inclusive o Visual C# 2010 Express Edition (versão livre do Visual Studio).

Como ilustrado na Figura 5, instalação do XNA Framework e o SDK para WP, é adicionado ao Visual Studio uma série de tipos de soluções[5] referentes a jogos, e graças ao seu SDK, também são adicionados projetos voltados para a plataforma WP. Selecionando uma das *templates* da Figura 5 é criada uma solução com todo o código necessário para o jogo rodar e se iniciar o desenvolvimento sem se preocupar com coisas básicas referentes a cada tipo de projeto (*template*).

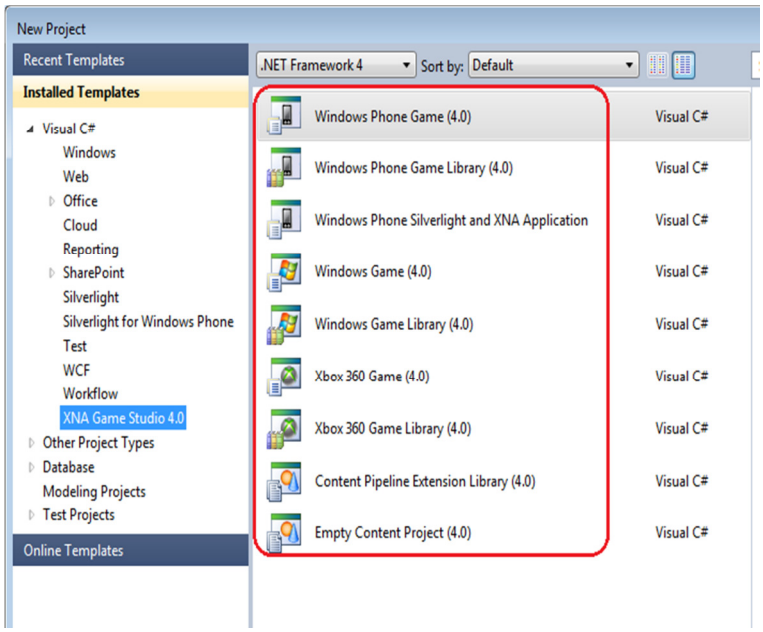


Figura 5: Tela de templates de projetos do Visual Studio 2010.

[5] O Visual Studio titula de "solução" uma aplicação completa, a qual possui um ou mais projeto com funções especificas.

Definição dos *templates* mais comuns para jogos na plataforma Microsoft:

- **Windows Phone Game:** cria uma solução com um projeto do tipo WP Game juntamente com todo o ambiente necessário para a criação de jogos para WP;
- **Windows Phone Game Library:** cria uma biblioteca de classes, a qual será utilizada por um WP Game;
- **Windows Phone Game Silverlight and XNA Application:** agrega o poder do Silverlight e do XNA Framework em uma unica solução;
- **Windows Game:** cria uma solução com todas as configurações necessárias para o desenvolvimento de um jogo para *desktop* (Windows 7, Windows vista, ...);
- **Windows Game Library:** cria uma biblioteca de classes, a qual será utilizada por um Windows Game;
- **Xbox 360 Game:** cria o esqueleto de um jogo para Xbox 360;
- **Xbox 360 Game Library:** biblioteca para projetos do tipo Xbox 360 Game;
- **Content Pipeline Extension Library:** monta o esqueleto de um projeto onde ficara todo o conteudo do jogo, sons, efeitos sonoros, texturas, modelos 3d, entre outros;
- **Empty Content Project:** estrutura um projeto do tipo *Content* vazio, ou seja, um projeto do mesmo tipo do *Content Pipeline Extension Library*, porem sem o esqueleto e as configurações padrões.

3.1 ESTRUTURA

A melhor maneira de se explicar a estrutura do XNA Framework é utilizando a Figura 6, a qual mostra todas as suas camadas e o que as mesmas contém.

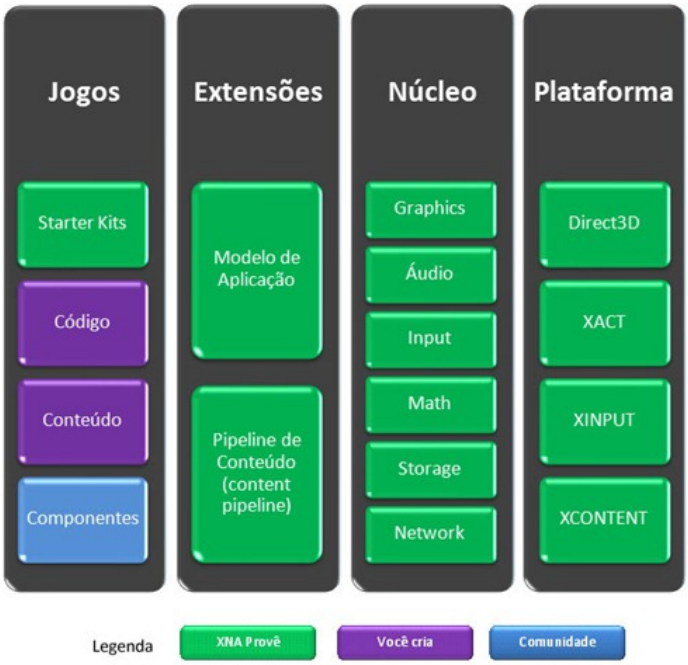


Figura 6: A plaforma XNA e seus componentes e bibliotecas (OLIVEIRA, 2009).

3.1.1 CAMADA DE JOGOS

Segundo Oliveira (2009), as *starter kits*[6] são pequenos jogos de vários gêneros, os quais são desenvolvidos pela Microsoft, e tem como objetivo impulsionar a criação de um jogo. Utilizando XNA o desenvolvedor somente fica responsável por criar o código e o conteúdo (sons, efeitos sonoros, texturas e modelos 3D) do jogo, como mostrado na Figura 6, e os componentes são por conta da comunidade (engines, bibliotecas, entre outros componentes feitos com o objetivo de facilitar a vida do desenvolvedor).

3.1.2 CAMADA DE EXTENSÕES

Todo jogo em XNA (em qualquer linguagem) é basicamente um *loop*, ou ciclo, que inicia quando o jogo começa e termina quando o jogo é finalizado, ou seja, o jogo fica constantemente gerenciando as ações do(s) jogador(es) e reage à estas de acordo com as regras do jogo, as quais foram criadas pelo desenvolvedor, o responsável por todo este gerenciamento é o modelo de aplicação.

[6] S*tarter kits* são distribuídas de maneira gratuita no seguinte endereço: <http://create.msdn.com/en-US/>.

O *Content Pipeline* é o encarregado por lidar com todo o conteúdo do game, ou melhor, as musicas, os efeitos sonoros, as texturas, ou ate mesmo os modelos 3D, fica a cargo desse componente gerenciar e otimizar a utilização dos mesmo pelo jogo, tirando esta tarefa das mãos do programador.

3.1.3 CAMADA DE NÚCLEO

Como o próprio nome já diz esta é a principal camada do XNA, é onde se encontra os principais recursos do mesmo: O componente Graphics (responsável por toda e qualquer redenrização de baixo nível do jogo, o mesmo roda sobre a plataforma Direct3D 9); Áudio (torna o gerenciamento de efeitos sonoros, musicas, entre outros, muito mais simples); Input (gerência entradas de dados feitas pelo usuário, sejam estas teclados, *mouses*, *joystick*, ...); Math (fornecer inúmeras funções matemáticas pré-definidas para auxiliar no trabalho com colisão, movimentação, física, vetores, entre diversos outros); Storage (facilita a manipulação de dados em disco, seja para salvar o jogo ou para o *ranking* do mesmo); e por ultimo o NetWork (o qual proporciona e facilita a criação de jogos em ambientes online, ou alguma conexão, caso seja necessária).

3.1.4 CAMADA DE PLATAFORMA

Enfim a última camada, a Plataforma, esta é a camada na qual todos os componentes anteriormente citados foram baseados, seja de forma direta ou indireta os mesmo utilizam esta plataforma.

3.2 CICLO DE VIDA

Diferentemente de uma aplicação comum, a qual possui uma interface gráfica que questiona o usuário para que o mesmo preencha formulários e clique em botões de OK ou cancelar. Um jogo não espera que o seu jogador diga o que fazer, ele fica em um loop desde o momento que é iniciado e lê todas as entradas que o usuário fez no intervalo entre iterações e reage concordantemente com as mesmas, fazendo

as personagens pularem, atirarem, entre outros diversos atos. Resumidamente, um jogo não possui registro de eventos, ao contrário de uma aplicação (Reed, 2009).

Graças ao XNA, desenvolver um jogo se tornou uma tarefa bastante simples e fez com que os desenvolvedores mantivessem o foco na ideia do jogo sem ter que se preocupar tanto com a sua plataforma (áudio, entradas, gráficos,...) em que o mesmo está sendo desenvolvido. Isso é possível não só por intermédio dos componentes citados anteriormente, mas também por conta de cinco métodos essências para qualquer jogo desenvolvido em XNA (Initialize, LoadContent, Update, Draw, UnloadContent). A Figura 7 mostra como são alocados os métodos anteriormente citados.

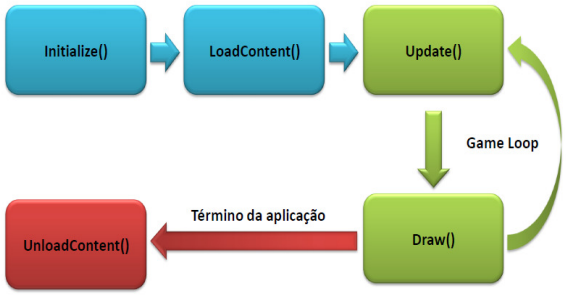


Figura 7: Ciclo de vida de um jogo em XNA (ANDRADE, 2010).

O método *Initialize* é o primeiro método a ser iniciado em um jogo e é onde todas as suas variáveis devem ser inicializadas. O método *LoadContent* é o local em que se encontra o código responsável por carregar todo o conteúdo que será utilizado no jogo (o qual fica alocado em um projeto do tipo *content pipeline*) para a memória do dispositivo (*desktop*, móvel ou console), ou seja, carrega as músicas, os efeitos sonoros, as texturas, os modelos 3D, etc. O *Update* é responsável por atualizar todos os objetos do jogo, como por exemplo mover um personagem, etc. O método *Draw* renderiza na tela tudo aquilo que foi "atualizado" pelo método *Update* e estes ficam em *loop* desde o início do jogo até seu término. Por fim, o método *UnloadContent* o qual é responsável por tirar da memória todo o conteúdo do projeto e finalizá-lo.

3.3 PRINCIPAIS COMPONENTES

Segundo Torres e Tavares (2009), os componentes abordados nos subtópicos desta sessão serão os mais utilizados pelo desenvolvedor durante a escrita de algum jogo, caso o mesmo opte por utilizar o XNA Framework.

3.3.1 MECÂNICA

Os tópicos a seguir destacam o principal objetivo de cada um dos componentes utilizados para promover a mecânica de um jogo em XNA.

- ***GameComponent*:** este componente é basicamente um esqueleto que serve para modelar qualquer tipo de componente que não dependa de gráficos para um correto funcionamento, por este motivo o mesmo não possui o método *Draw*;
- ***DrawableGameComponent*:** define uma estrutura para modelar componentes gráfico, ou seja, qualquer objeto visível na tela (sprites, fontes,...);
- ***GameComponentCollection*:** é uma classe do tipo lista que tem como objetivo agrupar os componentes, sejam estes do tipo *GameComponent* ou *DrawableGameComponent*, a fim de facilitar o gerenciamento dos mesmos;
- ***GameTime*:** é a classe responsável por computar todas as informações referentes ao tempo do jogo.

3.3.2 GRÁFICOS

Os seguintes tópicos discutirão a respeito dos componentes pertinentes aos gráficos de um jogo em XNA.

- ***Rectangle*:** é uma classe utilizada para calcular as coordenadas de um objeto na tela. Possui propriedades que informam exatamente a posição de cada lado do retângulo (*right, left, top, bottom*) na tela e

também possui um método nativo para o calculo da colisão, por exemplo, identificar se os tiros da nave estão atingindo os inimigos em jogo do tipo Space Invaders (jogo bidimensional de tiro);

- ***Vector2*:** é um objeto bastante utilizado para armazenagem da posição de um componente na tela e cálculos para movimentar o mesmo em um jogo bidimensional;
- ***Texture2D*:** normalmente utilizado para o carregamento e manipulação de texturas do jogo (cenários, sprites,...).
- ***Color*:** principalmente utilizado para colorir texturas, assim possibilita o reaproveitamento das mesmas. Também serve para criar efeitos de transparência, colisão *pixel*-a-*pixel* e efeitos de *fade-in* e *fade-out*.

3.3.3 ENTRADA DO JOGADOR

Esta sessão, tratará os tipos de entradas de dados de um jogo em XNA e os componentes necessários para trabalhar com as mesmas.

3.3.3.1 DESKTOP

Os seguintes tópicos explicarão sobre cada um dos componentes necessários para gerenciar as entradas do usuário a partir de um computador (Windows).

- ***Keyboard*:** é a classe usada para gerenciar e manipular o teclado. Contem todas as propriedades e métodos correspondentes a estas tarefas;
- ***KeyboardState*:** represa o estado atual do teclado do computador, é basicamente uma “foto” do teclado e é utiliza para detectar quais teclas foram apertadas;
- ***Keys*:** é um enumerador que representa todas as teclas do teclado, é necessário na identificação das teclas apertadas.

3.3.3.2 CONSOLE

A sessão aborda as classes utilizadas para gerenciar as entradas de dados a partir de um console (Xbox).

- ***GamePad*:** é a classe necessário para o gerenciamento e manipulação de controle do console, possui todas as propriedades e métodos pertinentes a estas ações;
- ***GamePadState*:** represa o estado do controle, ou seja, uma "fotografia" do mesmo informando todos botões pressionados;
- ***GamePadButtons*:** e uma estrutura com uma propriedade para cada botão do console, é também uma propriedade de GamePadState com a função de identificação de entradas do jogador;
- ***Buttons*:** é um enumerador que representa todos os botões de um controle para console, é utilizado na identificação de quais botões foram pressionados.

3.3.3.3 DISPOSITIVO MÓVEL

Esta sessão exibe as classes utilizadas para gerenciar as entradas do usuário a partir de um dispositivo móvel (WP).

- ***TouchPanel*:** é a classe responsável por fornecer métodos para obtenção de informações de entradas do jogador;
- ***TouchCollection*:** é uma coleção de *TouchLocation*, a qual é obtida a partir do método *GetState* do *TouchPanel*;
- ***TouchLocation*:** provê diversas propriedades com informações pertinentes à identificação da entrada do jogador (toque na tela), para que os objetos ou componentes do jogo possam reagir as mesmas;
- ***TouchLocationState*:** é um enumerador que representa todos os estados que um objeto do tipo *TouchLocations* pode ter.

4 PROJETO DO JOGO

Neste trabalho serão mostrados alguns conceitos e etapas do desenvolvimento de um jogo para WP. É um jogo do tipo *arcade* para um único jogador baseado no jogo KBounce[7]. KBounce é um jogo criado pela equipe internacional de tecnologia, o KDE. O jogo é parte de seu projeto, o qual consiste na criação de softwares livres para modernos e elaborados ambientes gráfico nas plataformas de Linux e Unix.

O jogo exige agilidade e estratégia. Possui um campo rodeado de paredes e duas ou mais bolas ficam se movendo de um lado para o outro, ricocheteando nas laterais da tela como mostra a Figura 8. O jogador tem como objetivo principal criar paredes e quando completar 75% dessa área vazia o mesmo vence e passa para o próximo nível, o qual uma bola é acrescentada.

O WPBounce teve sua ideia principal extraída do game KBounce, porém um pouco mais simplificado e compatível ao conceito de interface da Microsoft, o Metro.

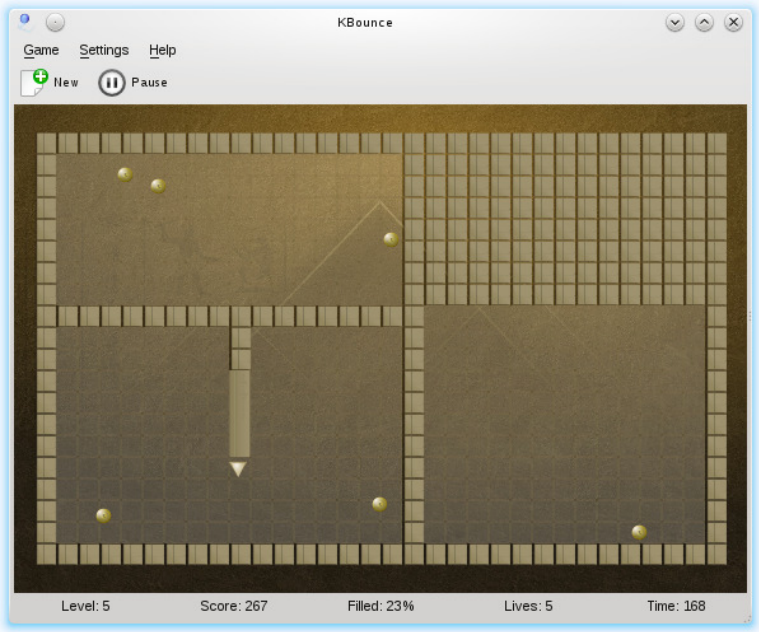


Figura 8: Jogo KBounce desenvolvido pela KDE.

[7] O seguinte vídeo mostra de maneira pratica o funcionamento e a mecânico do jogo KBounce: <http://www.youtube.com/watch?v=7oZw1YE5WJI>.

4.1 FERRAMENTAS UTILIZADAS

Para o desenvolvimento deste projeto foi utilizado, o Visual Studio 2010, o *framework* XNA 4.0, o SDK 7.1 do WP e seu emulador. Estas ferramentas serão melhor abordadas nos subtopicos desta sessão.

4.1.1 VISUAL STUDIO 2010

É uma IDE criada com o intuito de auxiliar no desenvolvimento de software para a plataforma .Net com as linguagens C#, Visual Basic, C++, J#, entre outras.

Não é apenas um editor de código e sim uma poderosa ferramenta com diversos recursos que facilitam o desenvolvimento de um software e aumentam sua qualidade final, tais como um ambiente completo para testes, ou seja, utilizando o Visual Studio[8] é possível criar código para testar a aplicação que está sendo desenvolvida, também possui um conceito de *Team builder*, ou seja, é possível desenvolver uma aplicação com diversos desenvolvedores trabalhando simultaneamente em diferentes partes do sistema. Enfim possui múltiplos recursos necessários para garantir a melhor qualidade de um software e a máxima velocidade na construção do mesmo (MARTIN, 2012).

4.1.2 XNA FRAMEWORK 4.0

O XNA Framework 4.0[9], a versão mais nova do *framework* discutido no Capitulo 3. É o mais utilizado no desenvolvimento de games para as plataformas Microsoft. Esta versão do XNA é compatível com todas as edições do Visual Studio 2010 com suporte a linguagem C#, inclusive o Visual C# Express Edition 2010 (Martin, 2012).

[8] O Visual Studio 2010 possui uma versão gratuita, o Visual C# Express Edition 2010, a qual está disponível no seguinte endereço: <http://www.microsoft.com/visualstudio/en-us/products/2010-editions/visual-csharp-express>.

[9] O Microsoft XNA Game Studio 4.0 está disponível gratuitamente no seguinte endereço: <http://www.microsoft.com/en-us/download/details.aspx?id=23714>.

4.1.3 WINDOWS PHONE SDK 7.1

Para desenvolver qualquer aplicativo ou jogo para WP é necessário antes instalar seu SDK[10], para que todas as templates e bibliotecas necessárias para o desenvolvimento para esta plataforma sejam instaladas e disponibilizadas para o desenvolvedor. O SDK do WP se encontra na versão 7.1 e é esta versão que será utilizada na criação do WPBounce (Martin, 2012).

Ao instalar o *SDK* também são instaladas algumas ferramentas auxiliares ao desenvolvimento:

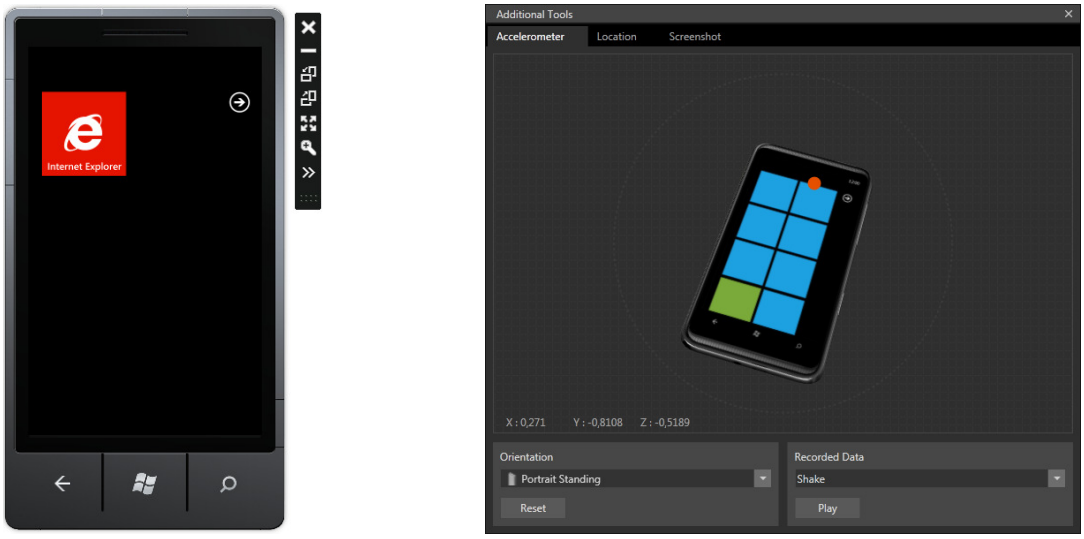
- **Emulador**: Emula um dispositivo com WP. Será melhor abordado no próximo tópico;
- ***Developer Phone Registration***: É um aplicativo que permite o desenvolvedor registrar até três dispositivos diferentes no WP MarketPlace (site que possibilita que desenvolvedores possam publicar e distribuir seus aplicativos e games), dessa forma é possível instalar o aplicativo direto em uma device e testa-lo em um ambiente real, permitindo assim a depuração do código, ou seja, analisar o código em tempo de execução e identificar possíveis falhas. Sem o registro do telefone não é possível instalar aplicativos no mesmo sem o intermédio do WP MarketPlace ou a ferramenta *Application Deployment*, a qual será abordada a seguir. Fazendo uso de um dos dois últimos meios de distribuição citados acima o desenvolvedor perde um valioso recurso, o *Debug* (Depuração do código);
- ***Application Deployment*:** É uma ferramenta que permite a distribuição de aplicativos do WP diretamente no dispositivo ou no emulador. A compilação de um aplicativo para WP resulta em um arquivo do tipo XAP, o qual é necessário para que o *Application Deployment* inicie o processo de instalação.

[10] WP SDK 7.1 está disponível de forma gratuita em: <http://create.msdn.com>.

4.1.4 WINDOWS PHONE EMULATOR

Junto com o SDK de desenvolvimento para WP já vem um emulador de um aparelho com WP instalado para que seja possível testar a aplicação desenvolvida sem a necessidade de um dispositivo real. O emulador possui a versão mais atual do WP e diversas características do mesmo, como por exemplo, a troca de tema, o botão de pesquisa, o qual abre o Bing (motor de busca online da Microsoft), o botão voltar, o navegador Internet Explorer, entre diversos outros recursos. A Figura 9.a mostra a pagina inicial de um emulador de WP (Martin, 2012).

Também é possível emular o acelerômetro (dispositivo que mede acelerações do aparelho, a fim de detectar movimentos externos, a orientação do mesmo, entre outras interações do usuário) do aparelho e testar aplicações que dependem deste recurso. A Figura 9.b mostra a janela de ferramentas adicionais do emulador com a aba de acelerômetro selecionada.



(a) (b)

Figura 9: Emulador do WP: (a) tela do emulado; (b) ferramenta de simulaçaõ do acelerômetro.

Ainda existem mais duas abas nas ferramentas adicionais do emulador do WP, Localização e *Screenshot*. A localização é basicamente uma GPS, ou seja, é possível definir localizações para testar aplicativos que necessitem deste recurso,

como por exemplo, algum sistema de navegação ou a rede social Foursquare (rede social que tem como principal objetivo mostrar a localização dos contatos do usuário, para que o mesmo saiba o que seus amigos estão fazendo ou se há algum próximo do mesmo). A Figura 10 mostra o simulador de localização do emulador do WP.

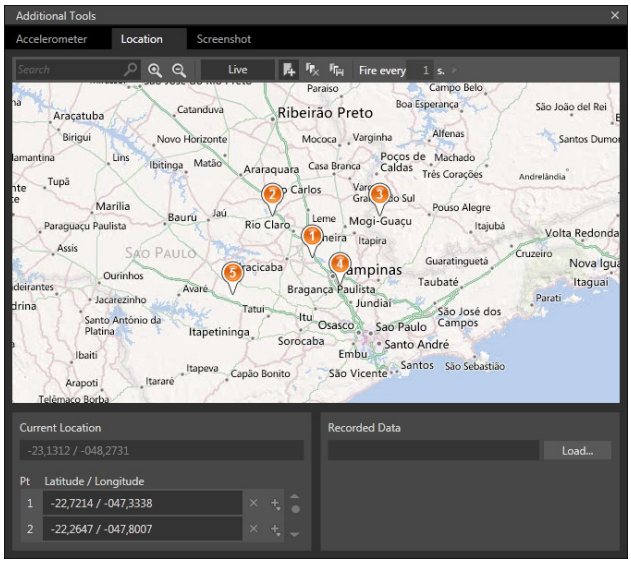


Figura 10: Aba de localização.

E por fim a aba *Screenshot* que permite ao usuário capturar a tela do seu emulador e exporta-la em formato de imagem. Funciona basicamente como a tecla *Printscreen* do computador. A Figura 11 mostra como é a aba *Screenshot*.

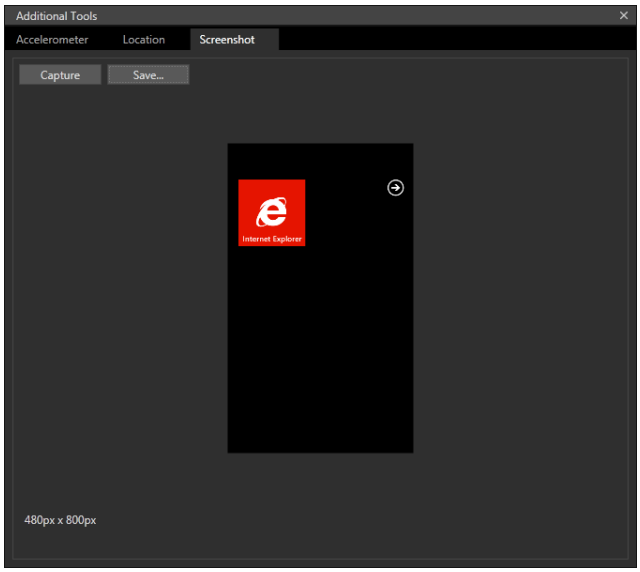


Figura 11: Aba Screenshot.

4.2 IDÉIAS E RASCUNHO DO JOGO

O WPBounce apesar de ser um jogo em diversos ponto semelhante ao KBounce possui uma inconfundível interface compatível ao novo conceito de interface gráfica da Microsoft, o Metro. As cores do jogo são trocadas concordantemente com os temas do sistema operacional. As Figuras 12 e 13 demonstram como ficará a interface do WPBounce conforme a troca de temas.

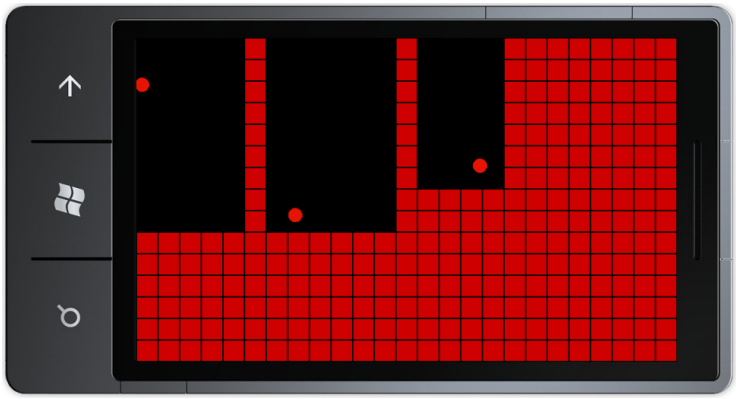


Figura 12: Interface do WPBounce com o tema vermelho (cor de destaque) e escuro (tela de fundo) do dispositivo.

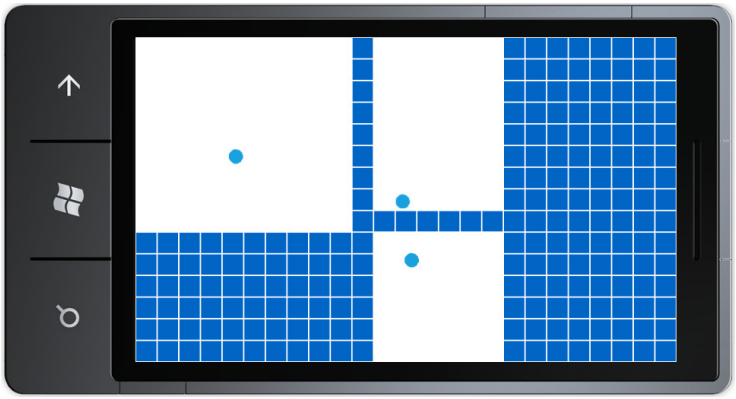


Figura 13: Interface do WPBounce com o tema azul (cor de destaque) e claro (tela de fundo) do dispositivo.

4.3 DEFINIÇÕES DO JOGO

Para a criação do jogo serão necessários dois objetos: a bola e o bloco. As Figuras 12 e 13 exemplificam a forma como os objetos ficarão dispostos na tela do jogo.

- **Bola:** circulo, com diâmetro de 30 *pixels*, que fica se movendo de um lado ao outro, ricocheteando nas laterais da tela. Se uma bola colidir com uma parede de blocos durante sua formação, esta finaliza o processo de construção e se mantém seu comprimento atual.
- **Bloco:** quadrado, junto os lados medem 30 *pixels*, que será utilizado para desenhar as paredes do jogo, estas são criadas em tempo de execução conforme as entradas do jogador.

4.4 A MECÂNICA DO JOGO

O objetivo do jogo é preencher toda a área do mesmo menos o espaço referente a quatro blocos por bola no jogo, porem as bolas ficam se movendo de um lado para o outro e as paredes só são construídas adequadamente se nenhuma bola as tocarem durante o processo de construção. Caso uma bola toquem a parede enquanto esta está sendo construída a mesma interrompe a construção e o jogador perde uma vida.

A cada nível concluído uma bola é acrescentada ao jogo, tornado o objetivo deste cada vez mais difícil de alcançar. O jogo pode chegar a ter no máximo 20 níveis, ou seja, 21 bolinhas.

4.5 A INTERFACE COM O USUÁRIO

A interface será bem simples, extremamente limpa e compatível com o conceito Metro, conforme mencionado anteriormente. Terá apenas duas cores, as quais seguirão o padrão do tema e cor de fundo do dispositivo, vermelho e preto (Figura 12), por exemplo. Ficarão dispostos na tela apenas dois objetos: a bola e o bloco, conforme já citado acima, e o placar do jogo, informado a quantidade de vidas e porcentagem da área do jogo preenchido por blocos.

A bola ficara ricocheteando de um lado a outro, e quando em contato com algum bloco, sua direção será alterada, nada mais pode interferir no comportamento da bola, a mesma é imune a intervenções do jogador.

Já os blocos são totalmente dependentes do jogador, pois os mesmos necessitam das entradas do jogador para iniciar a construção de uma parede, porem após o inicio deste processo somente as bolas podem interferir no mesmo e interrompe a criação do muro, mas somente enquanto a parede esta sendo construída, porque depois de finalizada a construção nada pode alterar seu estado.

A Figura 14 mostra um rascunho de como será a tela do jogo.

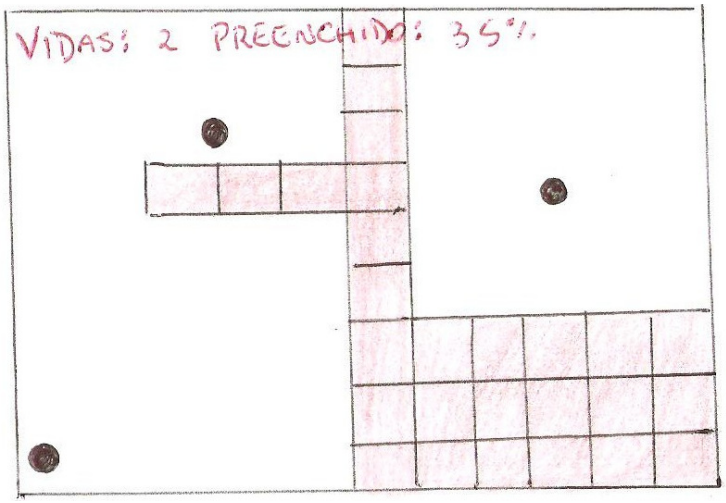


Figura 14: Rascunho da tela final do jogo.

5 CONCEPÇÃO DO JOGO

Juntamente com a elaboração deste documento, também foi concebido um protótipo, ou melhor, foi desenvolvido um jogo, o WPBounce. É um game bastante simples de se jogar, porém um pouco mais complicado de se programar, mas, graças ao XNA, seu desenvolvimento foi relativamente rápido, aproximadamente 160 horas de desenvolvimento.

A Figura 15 mostra o diagrama de classes do WPBounce, basicamente o jogo é dividido em componentes (ComponentBase, Ball, Block e GameArea) e a classe principal do jogo (WPBounce).

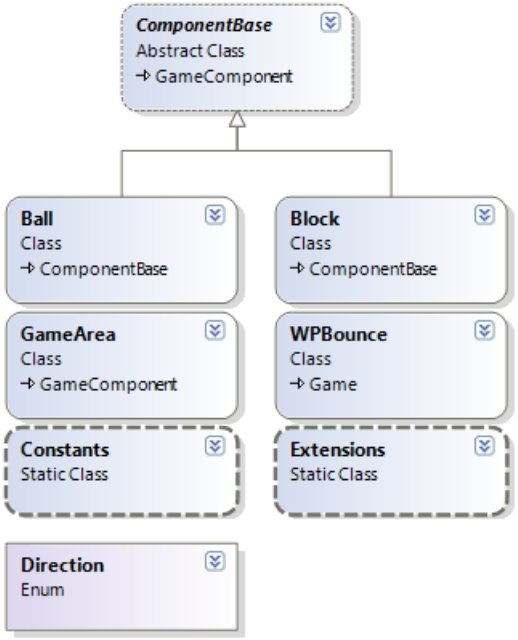


Figura 15: Diagrama de classes do jogo WPBounce

- **ComponentBase:** é uma classe abstrata que somente pode ser herdada e nunca instanciada. Ela serve como base para as classes Ball e Block, possui métodos que ambas deveriam possuir, desta forma a implementação fica alocada em um único local;
- **Ball:** é utilizada para manipular o objeto bola do jogo, o qual foi citado e descrito anteriormente;
- **Block:** é necessário na manipulação do objeto bloco, como já definido anteriormente;

- **GameArea:** é utilizado no gerenciamento da área do jogo, ou seja, manipulação dos muros ou paredes. Basicamente esta classe tem uma lista de blocos, os quais ficam dispostos em seus devidos lugares, porem estão invisíveis e quando o jogador executa alguma ação, os blocos do muro selecionado vão se tornando visíveis ate que o mesmo alcance outra parede visível ou uma bola colida com o mesmo ou encontre o limite da tela;
- **WPBounce:** é a classe principal do jogo e responsável por gerenciar o mesmo, detectar os toques na tela, detectar as colisões, desenhar a tela, entre outras ações;
- ***Constants*** / ***Extensions*** / ***Direction*****:** são classes auxiliares ao jogo e não possuem um comportamento em especial.

Entre diversos pontos de extrema importância na implementação do WPBounce, dois foram eleitos para serem destacados nos próximos parágrafos. Como foram manipulados os toques na tela e como foram tratadas as colisões das bolas com os blocos, pois estes são dois pontos chaves do jogo.

A Figura 16 exibe uma parte do código necessária para tratar os toques na tela do dispositivo e coordenar as ações seguintes. Basicamente quando o jogador toca a tela deve ser desenhado um muro na área do jogo a partir do ponto de partida do toque e na mesma direção em que o dedo se direciona, seja horizontalmente ou verticalmente. Para detectar esta ação é necessário utilizar o método GetState da classe estática TouchPanel, o qual retorna uma lista de TouchLocation e a partir deste objeto é possível obter o estado do toque e a localização do mesmo. O estado do toque é utilizado para se identificar o que o jogador fez, ou seja, se ele apenas pressionou a tela ou se deslizou o dedo sobre a mesma, no caso do WPBounce, somente reproduzem ações no jogo quando o usuário desliza o dedo sobre a tela, assim se inicia a construção do muro. A posição é indispensável para saber onde deve se iniciar a construção do muro e a direção do mesmo (vertical ou horizontal).

```
var touchLocations = TouchPanel.GetState();
foreach (var touchLocation in touchLocations)
{
    if (touchLocation.State == TouchLocationState.Pressed)
    {
        firstTouchPosition = touchLocation.Position;
        canUpdateBlocks = true;
    }
    else if (touchLocation.State == TouchLocationState.Released)
        canUpdateBlocks = true;
    else if (canUpdateBlocks && touchLocation.State == TouchLocationState.Moved &&
        touchLocation.Position != firstTouchPosition)
    {
        lastTouchPosition = touchLocation.Position;
        canUpdateBlocks = false;

        float firstX = firstTouchPosition.X, firstY = firstTouchPosition.Y;
        float actualX = touchLocation.Position.X, actualY = touchLocation.Position.Y;

        var resultX = Math.Abs(actualX - firstX);
        var resultY = Math.Abs(actualY - firstY);

        if (resultX > resultY)
            wallDirection = Direction.Horizontal;
        else
            wallDirection = Direction.Vertical;
    }
    else
        wallDirection = Direction.None;
}
```

Figura 16: Parte do código do WPBounce que mostra como foi manipulado o toque na tela.

O segundo ponto que merece atenção é a colisão das bolas com os blocos, como mostrado na Figura 17 foi criado um método (CheckCollisions) na classe WPBounce (classe principal do jogo), o qual é responsável por checar as colisões e alterar a direção da bola. Basicamente o método recebe um objeto Ball e verifica se o mesmo colidiu com os limites da tela, ou chocou-se com três blocos de uma só vez, ou com qual lado do bloco o mesmo colidou, e assim, de acordo com a colisão resultante do método, a direção da bola corrente é alterada, para que a mesma fique ricocheteando de um lado a outro da área do jogo.

```
public void CheckCollisions(Ball ball)
{
    var ballRectangle = ball.GetBounding();

    var intersectedBlocks = (from b in GameArea.Blocks
                             where
                                b.GetBounding().Intersects(ballRectangle) &&
                                b.Visibility == Visibility.Visible
                             select b).ToList();

    if (ball.Position.Y + ball.Texture.Height > graphics.PreferredBackBufferHeight || ball.Position.Y < 0)
        ball.Direction *= new Vector2(intersectedBlocks.Count >= 1 ? -1 : 1, -1);

    if (ball.Position.X + ball.Texture.Width > graphics.PreferredBackBufferWidth || ball.Position.X < 0)
        ball.Direction *= new Vector2(-1, intersectedBlocks.Count >= 1 ? -1 : 1);

    if (intersectedBlocks.Count == 3)
        ball.Direction *= new Vector2(-1, -1);
    else
    {
        foreach (var block in intersectedBlocks)
        {
            var blockRectangle = block.GetBounding();

            var blockCanCollideBottom = ballRectangle.Center.Y > blockRectangle.Center.Y ? block.CanCollideBottom : false;
            var blockCanCollideTop = ballRectangle.Center.Y < blockRectangle.Center.Y ? block.CanCollideTop : false;
            var blockCanCollideLeft = ballRectangle.Center.X < blockRectangle.Center.X ? block.CanCollideLeft : false;
            var blockCanCollideRight = ballRectangle.Center.X > blockRectangle.Center.X ? block.CanCollideRight : false;

            if (ball.Direction.X == 1 && blockCanCollideLeft && ballRectangle.Right >= blockRectangle.Left)
            {
                ball.Direction = new Vector2(-1, ball.Direction.Y);
                break;
            }

            if (ball.Direction.X == -1 && blockCanCollideRight && ballRectangle.Left <= blockRectangle.Right)
            {
                ball.Direction = new Vector2(1, ball.Direction.Y);
                break;
            }

            if (ball.Direction.Y == 1 && blockCanCollideTop && ballRectangle.Bottom >= blockRectangle.Top)
            {
                ball.Direction = new Vector2(ball.Direction.X, -1);
                break;
            }

            if (ball.Direction.Y == -1 && blockCanCollideBottom && ballRectangle.Top <= blockRectangle.Bottom)
            {
                ball.Direction = new Vector2(ball.Direction.X, 1);
                break;
            }
        }
    }
}
```

Figura 17: Método utilizado para tratar a colisão da bolinha com os blocos.

6 CONSIDERAÇÕES FINAIS

Desenvolver um jogo nunca foi uma tarefa simples, além de se preocupar com a ideia principal do jogo o desenvolvedor precisa se atentar a todos os recursos necessários para o mesmo funcionar, não somente com seu código, mas com todo o ambiente, plataforma e gerenciamento de hardware do dispositivo para o qual o mesmo foi desenvolvido (computador, dispositivo móvel, *console*,...).

O XNA além de facilitar o desenvolvimento de um game, como já citado, foi criado com o principal objeto de fazer com que o desenvolvedor foque seu trabalho quase que inteiramente na criação do jogo e aperfeiçoamento da ideia, ou seja, o mesmo deixa de se preocupar com recursos externos, como por exemplo o loop principal do game ou recursos gráficos da plataforma, e atenta-se de fato na criação do mesmo.

Como constatado durante o desenvolvimento do WPBounce e através dos passos abordados na sessão 5, o XNA atende de fato ao seu objetivo de aproximar o desenvolvedor à ideia principal do jogo que o mesmo esta desenvolvendo. Foi averiguado também que graças ao XNA e o SDK de desenvolvimento para WP, desenvolver jogos para WP se tornou uma tarefa bastante simples e focada no que realmente interessa: a ideia principal do jogo.

7 REFERÊNCIAS BIBLIOGRÁFICAS

ANDRADE, K. O. **Desenvolvimento de Jogos 2D com XNA**. Disponível em: <http://www.4shared.com/office/TqZw2nEw/Mini-curso_de_XNA_2D.html>. Acesso em 15 de abril de 2012.

ASTLE, D.; HAWKINS K. **The Exploration Begins... Again. In: Beginnig OpenGL Game Programming**. Boston, MA: Stacy L. Hiquet, 2004. p. 26-35.

BOCZKOWSKI, T.; TROUNEV, E. **Manual do KBounce**. Disponível em: < http://docs.kde.org/development/pt_BR/kdegames/kbounce/kbounce.pdf>. Acesso em: 10 de março de 2012.

JONES, W. **The What, Why, And How of Direct. In: Beginning DirectX 9**. Boston, MA: Stacy L. Hiquet, 2004. p. 3-7.

JÚNIOR, E. **Vamos aprender XNA?: Entenda as facilidades proporcionadas pelo Framework para desenvolvimento de games da Microsoft**. .Net Magazine, Grajaú, RJ, v. 86, p. 10-31, junho. 2011.

KRAVITZ, N. 7 **Ways Windows Phone 7 Is Better Than Android (And 5 Ways It Isn't)**. Disponível em: < http://www.maximumpc.com/article/7_ways_windows_phone_7_better_android_and_5_ways_it_isn%E2%80%99t>. Acesso em 26 de abril de 2012.

MACHADO, C. **Em 2015, Windows Phone será o n. 2. Sério?** Disponível em: <http://info.abril.com.br/noticias/blogs/estacaowindows/categoria/windows-phone/>. Acesso em: 26 de abril de 2012.

MACIEJEWSKY, J. **Windows Phone 7: Recursos de Softwares - Parte 1.** Disponível em: <http://blog.maciejewsky.net/blog/post/2011/06/20/Windows-Phone-7-Recursos-de-Softwares%E2%80%93Parte-1.aspx>. Acesso em: 26 de abril de 2012.

MARTIN, F. **Visão geral do conceito Metro**. Disponível em: <http://channel9.msdn.com/posts/O-Metro-Viso-Geral-do-Windows-Phone>. Acesso em: 26 de abril de 2012.

MARTIN, F. **Visão geral do emulador de Windows Phone**. Disponível em: <http://channel9.msdn.com/posts/O-Emulador-Viso-Geral-do-Windows-Phone>. Acesso em: 26 de abril de 2012.

MARTIN, F. **Visão geral das ferramentas de desenvolvimento para Windows Phone**. Disponível em: <http://channel9.msdn.com/posts/As-Ferramentas-Viso-Geral-do-Windows-Phone>. Acesso em: 26 de abril de 2012.

MARTIN, F. **Visão geral do SDK do Windows Phone**. Disponível em: <http://channel9.msdn.com/posts/O-SDK-Viso-Geral-do-Windows-Phone>. Acesso em: 26 de abril de 2012.

OLIVEIRA, S. **Conhecendo o XNA: Mergulhando nas suas camadas**. Disponível em: <http://www.nintendoblast.com.br/2009/12/gamedev-6-conhecendo-o-xna-mergulhando.html>. Acesso em: 20 de março de 2012.

REED, A. **Fun With Sprites. In: Learning XNA 3.0**. Sebastopol, CA: O'Reilly Media, 2009. P. 8-39.

TORRES, D.; TAVARES, E. **Projeto Hydra: Desenvolvimento de jogos digitais utilizando o framework Microsoft XNA**. 2009. 86 f. Trabalho de Conclusão de Curso (Bacharelado em Ciências da Computação) – Área de Ciências Exatas e Tecnologia, Centro Universitário do Pará, Belém, PA, 2009. Disponível em: <http://pt.scribd.com/doc/86656081/18/Principais-Componentes-do-XNA>. Acesso em: 30 de março de 2012.

WILLIAMS, C. G.; CLINGERMAN, G. W. **Getting to Know the Windows Phone 7 Device. In: Professional Windows Phone 7 Game Development: Creating Games using XNA Game Studio 4**. Indianapolis, IN: Wiley Publishing, 2011. p. 1-10.