FACULDADE DE TECNOLOGIA DE AMERICANA

Curso Jogos Digitais

Rodrigo Galzano Pereira de Lima

Desenvolvimento de Jogos Digitais Utilizando a Metodologia RAD, no Ambiente Delphi, com o Componente DelphiX

Americana, SP

2015

FACULDADE DE TECNOLOGIA DE AMERICANA

Curso Jogos Digitais

Rodrigo Galzano Pereira de Lima

Desenvolvimento de Jogos Digitais Utilizando a Metodologia RAD, no Ambiente Delphi, com o Componente DelphiX

Trabalho de Conclusão de Curso desenvolvido em cumprimento à exigência curricular do Curso Jogos Digitais sob a orientação do Prof. Esp. José William Pinto Gomes

Área de concentração: Jogos Digitais.

Americana, SP

2015

FICHA CATALOGRÁFICA – Biblioteca Fatec Americana - CEETEPS
Dados Internacionais de Catalogação-na-fonte

Lima, Rodrigo Galzano Pereira de

L71d Desenvolvimento de jogos digitais utilizando a metodologia RAD, na linguagem delphi, com o componente delphiX. / Rodrigo Galzano Pereira de Lima. – Americana: 2015.

43f.

Monografia (Graduação em Tecnologia em Jogos Digitais). - - Faculdade de Tecnologia de Americana – Centro Estadual de Educação Tecnológica Paula Souza.

Orientador: Prof. Esp. José William Pinto Gomes

1. Jogos eletrônicos 2. Linguagem de programação I. Gomes, José William Pinto II. Centro Estadual de Educação Tecnológica Paula Souza – Faculdade de Tecnologia de Americana.

CDU: 681.6
681.3.061

Rodrigo Galzano Pereira de Lima

Desenvolvimento de Jogos Digitais Utilizando a Metodologia RAD, no Ambiente Delphi, com o Componente DelphiX

Trabalho de graduação apresentado como exigência parcial para obtenção do título de Tecnólogo em Jogos Digitais pelo CEETEPS/Faculdade de Tecnologia – FATEC/ Americana.
Área de concentração: Jogos Digitais

Americana, 24 de junho de 2015.

Banca Examinadora:

José William Pinto Gomes
Especialista
Fatec Americana

Francesco Artur Perrotti
Mestre
Fatec Americana

Wagner Siqueira Cavalcante
Mestre
Fatec Americana

AGRADECIMENTOS

A todos que direta ou indiretamente fizeram parte da minha formação, o meu muito obrigado.

DEDICATÓRIA

Com muito carinho, dedico a minha mãe
Margaret Galzano, pela compreensão,
apoio e contribuição para minha formação
acadêmica.

RESUMO

Esta monografia descreve algumas técnicas de desenvolvimento rápido de jogos. O projeto foi desenvolvido utilizando Delphi e um componente para o auxílio chamado DelphiX.

O objetivo deste trabalho é mostrar os primeiros passos para criar um jogo para desktop utilizando Delphi e mostrando técnicas básicas para a criação de um jogo. A metodologia de desenvolvimento usada será o RAD "*Rapid Application Development*", uma metodologia de desenvolvimento de aplicações que visa ser simples e evita ser necessário digitar uma quantidade excessiva de códigos.

Palavras Chave: RAD, Delphi, Desenvolvimento.

ABSTRACT

This monograph describes some techniques Games Rapid Development. The project was developed using Delphi and a component called DelphiX.

The objective of this study is to show the first steps to create a game for desktop using Delphi and showing basic techniques for creating a game. The development methodology will be used RAD "Rapid Application Development", an application development methodology that aims to be simple and avoids be required to enter an excessive amount of codes.

Keywords: *RAD, Delphi*, *development.*

SUMÁRIO

LISTA DE FIGURAS

1 Introdução

Existem várias metodologias de desenvolvimento de software e podemos dividir elas em dois grupos distintos as metodologias tradicionais e ágeis. Dentre as metodologias de desenvolvimento de software tradicionais podemos citar o modelo cascata ele sugere um modelo sequencial e sistemático para o desenvolvimento de software, começando com o levantamento de necessidades, avançando pelas fases de planejamento, modelagem, construção, emprego e culminando no suporte contínuo do software. Os modelos ágeis são mais adaptativos a mudanças no panorama moderno, atualmente é difícil ou impossível prever como um sistema computacional irá evoluir com o tempo, muitas vezes não será possível levantar todos os requisitos antes do início do projeto. Os modelos ágeis se tornaram mais populares entre a comunidade quando em 2001, Kent Beck e outros dezesseis desenvolvedores, autores e consultores da área de software (batizados de "*Agile Alliance*" – "Aliança dos Ágeis") assinaram o "Manifesto para o desenvolvimento Ágil de Software" ("*Manifesto for Agile Software Development*") (Pressman, 2006).

A monografia foca na metodologia RAD "*Rapid Application Development*" é uma metodologia de desenvolvimento de sistemas criada para diminuir radicalmente o tempo necessário para projetar e implementar sistemas.

Existem diversas ferramentas que auxiliam no desenvolvimento de software, dentre elas é citado o Delphi que é uma aplicação RAD.

O Delphi é uma ferramenta de desenvolvimento onde o desenvolvedor tem em um único ambiente de trabalho, todos os recursos necessários para o desenvolvimento de seu software. Uma curiosidade é que o nome Delphi é inspirado na cidade Delfos, único lugar na Grécia antiga onde acreditava-se que era possível consultar o Oráculo de Delfos. A primeira versão do Delphi foi lançada pela Borland em 1995, uma das primeiras ferramentas RAD.

A linguagem utilizada no Delphi é uma extensão OOP (*Object-oriented programming*) da linguagem Pascal clássica, que Borland tem usado por muitos anos em seus compiladores Turbo Pascal (Cantu, 2003).

O desenvolvimento de jogos pode ser um processo demorado. Muitas vezes por ser tratar de um projeto que envolve diferentes ramos, um jogo eletrônico complexo exige grandes esforços em áreas distintas como: programação, modelagem 3D, concepção de arte, roteiro e gerência de projetos. Visando um desenvolvimento

mais rápido de jogos, é apresentada uma metodologia RAD em conjunto com o Delphi que é uma ferramenta de fácil aprendizagem e recomendado também para pessoas que estejam iniciando no desenvolvimento de software, sobretudo de jogos digitais.

Os Objetivos da monografia são mostrar como implementar técnicas de desenvolvimento utilizando RAD. Para um melhor entendimento foi desenvolvido um jogo bidimensional utilizando a ferramenta RAD Delphi.

De uma forma especifica é realizado um estudo da metodologia RAD de desenvolvimento, apresentando a ferramenta de apoio (Delphi) para a execução de um projeto, com o objetivo de implementar algumas funcionalidades em um jogo digital.

2 Referencial Teórico

Este capítulo apresenta a metodologia RAD de desenvolvimento e a tecnologia que é usada no desenvolvimento de um projeto.

2.1 Metodologia de desenvolvimento RAD

O modelo RAD (*Rapid Application Development*), ou modelo de desenvolvimento rápido de aplicações, é um tipo de modelo incremental que prima por um ciclo de desenvolvimento curto (tipicamente de até 90 dias). O modelo RAD é uma adaptação do modelo cascata, no qual o desenvolvimento rápido é obtido com o uso de uma abordagem de construção baseada em componentes. Para obter êxito na execução do projeto utilizando RAD é necessário que os requisitos sejam compreendidos e o objetivo do projeto seja restrito, para que as equipes possam criar uma aplicação plenamente funcional dentro de um curto período de tempo (Pressman, 2006).

Para algumas pessoas, o desenvolvimento rápido consiste na aplicação de um único conjunto de ferramentas ou método. Para o *hacker*, desenvolvimento rápido é codificar por 36 horas sem parar. Para o engenheiro de software, RAD é a combinação de ferramentas CASE, intensivo envolvimento do usuário e cronogramas apertados. Para o programador é a rápida prototipagem usando alguma versão do Microsoft Visual Basic ou Delphi.

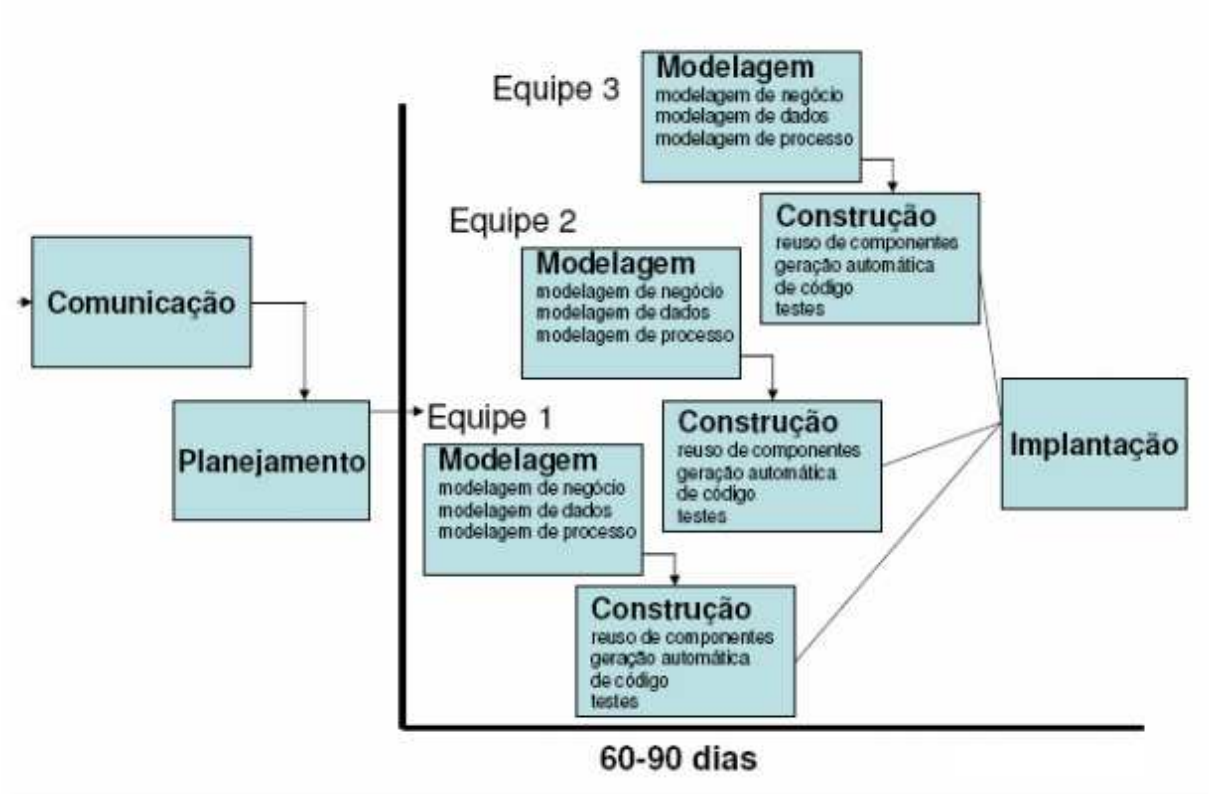
Cada uma dessas ferramentas e métodos são boas na sua medida e cada uma pode contribuir para o aumento da velocidade de desenvolvimento, mas para prover um pleno benefício cada uma precisa ser orquestrada como parte de uma estratégia. O "projeto de desenvolvimento rápido", é qualquer projeto que precisa enfatizar a velocidade de desenvolvimento (McConnell, 1996).

Como em outros modelos de processo, a comunicação é necessária para entender o plano de negócio e as características de informação que o software precisa acomodar. Um planejamento cuidadoso feito previamente é essencial para o sucesso, porque podem exisitr uma ou várias equipes de software trabalhando em paralelo em diferentes atividades relacionadas ao projeto. Após o planejamento é iniciada a modelagem que abrange as três principais fases – modelagem do negócio, modelagem dos dados e modelagem dos processos – que estabelecem

representações do projeto que servirão como base para toda a construção do projeto. A construção enfatiza o uso de componentes de software preexistentes e a aplicação da geração automática de código. A parte da implantação finalmente estabelece um ponto para as interações subsequentes, caso seja necessária alguma melhoria (Pressman, 2006).

Na figura 1 são apresentadas todas as etapas de desenvolvimento utilizando o RAD.

Figura 1 - "O modelo de RAD"



Fonte: Pressman (2006).

Algumas das vantagens de utilizar o modelo RAD são:

- Reutilização de componentes;
- Redução do tempo de desenvolvimento;
- Custos menores;
- Alta interação com o usuário;
- Respostas rápidas a mudanças;
- Viável integrar outras ferramentas auxiliares ao desenvolvimento RAD;

Embora existam ainda desvantagens como:

- A reutilização de componentes não garante a eficiência do código, podendo comprometer a qualidade;
- Alta dependência do código em relação à ferramenta;
- Preocupação com custo baixo pode comprometer a qualidade;
- Falta de escalabilidade;
- Dificuldade com reuso de módulos;
- Concentração excessiva na interface do usuário, pode resultar na falta de refinamento dos requisitos funcionais;

Muitos sistemas RAD atuais também incluem ferramentas de programação visual que facilitam e agilizam o desenvolvimento, permitindo que o projeto seja desenvolvimento interativamente. Em vez de programar de uma forma sequencial é possível desenvolver manipulando ícones gráficos que representam funções, dados, e componentes de interface com o usuário, e associa scripts de processamento a esses ícones. Com esta abordagem é possível que os programadores criem uma aplicação interativamente pela definição da interface.

2.2 Ferramenta Delphi

O Delphi como já foi mencionado é uma ferramenta RAD algumas das características que agilizam o desenvolvimento na ferramenta são:

O ambiente Delphi: O object Pascal, linguagem nativa do Delphi é conhecida por ser bastante detalhada e mais intelegível do que por exemplo a linguagem C e sua extensão OOP (*Object-Oriented Programming*) segue a mesma abordagem oferecendo a mesma potência da recente geração de linguagens OOP a partir de Java até C#.

Paleta VCL (*Visual Component Library*): Esta é a biblioteca de classes visuais. A VCL inclui embalagens dos controles nativos do Windows (como botões e caixas de edição), os controles comuns (como *TreeViews* e

ListViews), além de outros controles nativos do Delphi ligados ao conceito do Windows de janela. Além disso, a classe TCanvas envolve as chamadas gráficas básicas, que permite facilmente pintar na superfície de uma janela, por exemplo.

O Delphi é uma IDE e sua biblioteca de componentes (VCL) contribui para uma boa consistência interna e um pacote mais reconhecível (Embarcadero, 2015). Existem também algumas vantagens do Delphi que precisam ser citadas: a existência de uma grande quantidade de componentes prontos em sua biblioteca, facilidade de uso e aprendizado, desenvolvimento rápido e velocidade de execução do código.

2.3 Jogos Digitais

Para entender o que são jogos digitais é necessário compreender que a análise dos jogos numa visão mais ampla do termo, é uma atividade complexa devido a grandeza do campo de estudo a qual o mesmo se insere. Segundo Huizinga (2003), um jogo corresponde à um elemento muito primitivo, que antecede o surgimento da cultura na medida em que é um conceito compartilhado com outros animais. O autor exemplifica essa noção através de uma brincadeira realizada pelos caninos, na qual os animais convidam-se a participar de uma atividade lúdica em que disputam entre si, respeitando algumas regras. A atividade é dita lúdica pois a disputa em si não é real, mas sim fantasiada dentro dos limites estabelecidos.

Segundo Schuytema (2008), um jogo eletrônico é uma atividade lúdica formada por ações e decisões que resultam numa condição final. Tais ações e decisões são limitadas por um conjunto de regras e por um universo, que no contexto dos jogos digitais, são regidos por um programa de computador. O universo contextualiza as ações e decisões do jogador, fornecendo a ambientação adequada à narrativa do jogo, enquanto as regras definem o que pode e o que não pode ser realizado, bem como as consequências das ações e decisões do jogador. Além disso, as regras fornecem desafios a fim de dificultar ou impedir o jogador de alcançar os objetivos estabelecidos.

Os jogos digitais geralmente são compostos por três partes: o enredo, motor e a interface interativa. O enredo define o tema, a trama, apresenta como os objetivos

do jogo vão se devolver, passando por uma sequência de acontecimentos que também podem servir como plano de fundo para a jogabilidade. O motor do jogo o mecanismo que controla a reação do ambiente às ações e decisões, do jogador que podem resultar em mudanças no estado do ambiente do jogo. A interface interativa permite a comunicação entre o jogador e o motor do jogo, fornecendo um caminho de entrada para as ações do jogador e um caminho de saída para as respostas audiovisuais referentes às mudanças do estado do ambiente (Battaiola, 2000).

Uma característica marcante nos jogos digitais se refere à rigidez das regras. Apesar dos jogos, em geral, serem regrados, quando se trata de jogos não-digitais, sempre existe espaço para uma negociação das regras. Nessa negociação, por exemplo, pode-se optar ou não por algum tipo de punição quando um caso específico ocorre no decorrer do jogo, e tal negociação é realizada e respeitada pelos participantes na ocasião do início, ou mesmo durante a partida. No caso dos jogos digitais, essa flexibilidade não é comum, uma vez que as regras são traduzidas em algoritmos de computador, sendo assim sistematicamente seguidas. Em alguns jogos digitais pode até ser possível, através de configurações, personalizar algumas regras em casos específicos, mas ainda assim tais mecanismos não são triviais e tampouco flexíveis como os meios de negociação praticados nos jogos não-digitais.

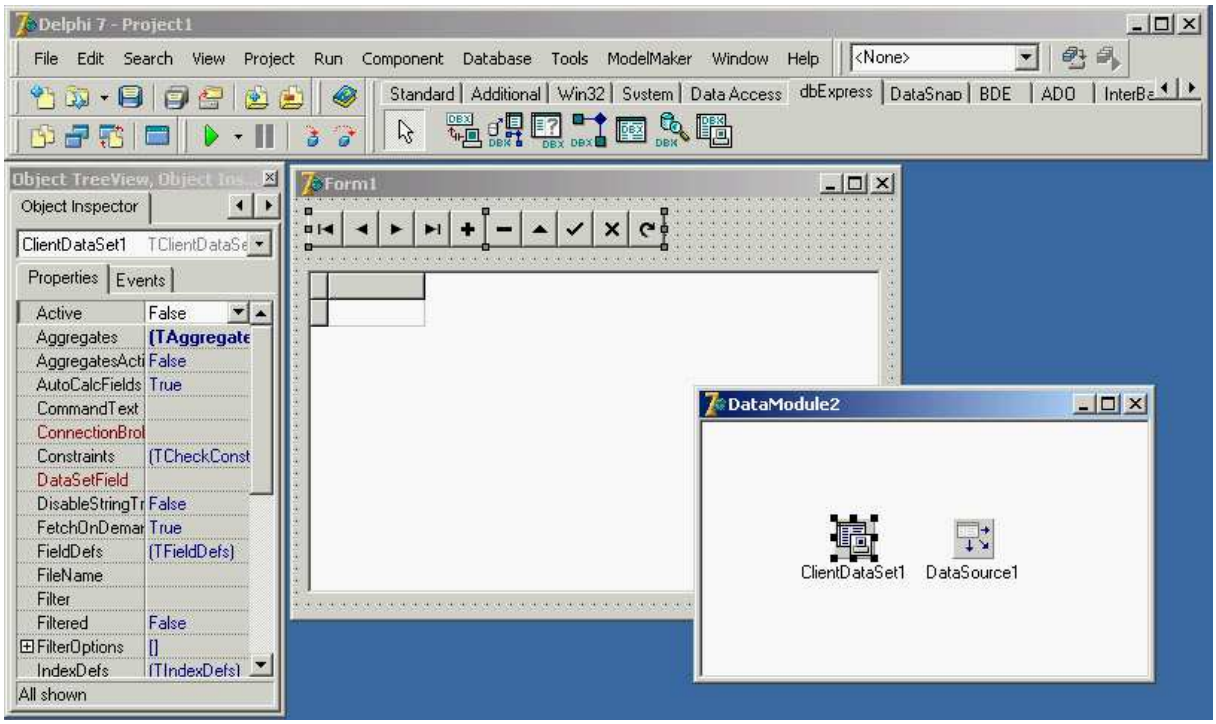
3 Desenvolvimento de Jogos Digitais utilizando Delphi

Neste capítulo é apresentado a ferramenta RAD Delphi, apresentando suas principais características para o desenvolvimento.

3.1 Características do Delphi

Segundo Cantú (2003), quando é trabalhado com um ambiente de desenvolvimento visual, seu tempo é gasto em duas porções diferentes da aplicação: designers visuais e editor de código. No modo de designer é possível trabalhar com componentes a nível visual (como quando é colocado um botão em um formulário) ou a um nível não-visual (como quando é colocado um componente *DataSet* em um módulo de dados). Em ambos os casos, a parte de designer permite a escolha dos componentes necessários e é possível definir o valor inicial das propriedades dos componentes. A figura 2, a seguir, é um exemplo.

Figura 2 - "Um *Form* e um *Data Module*"



Fonte: Cantú (2003).

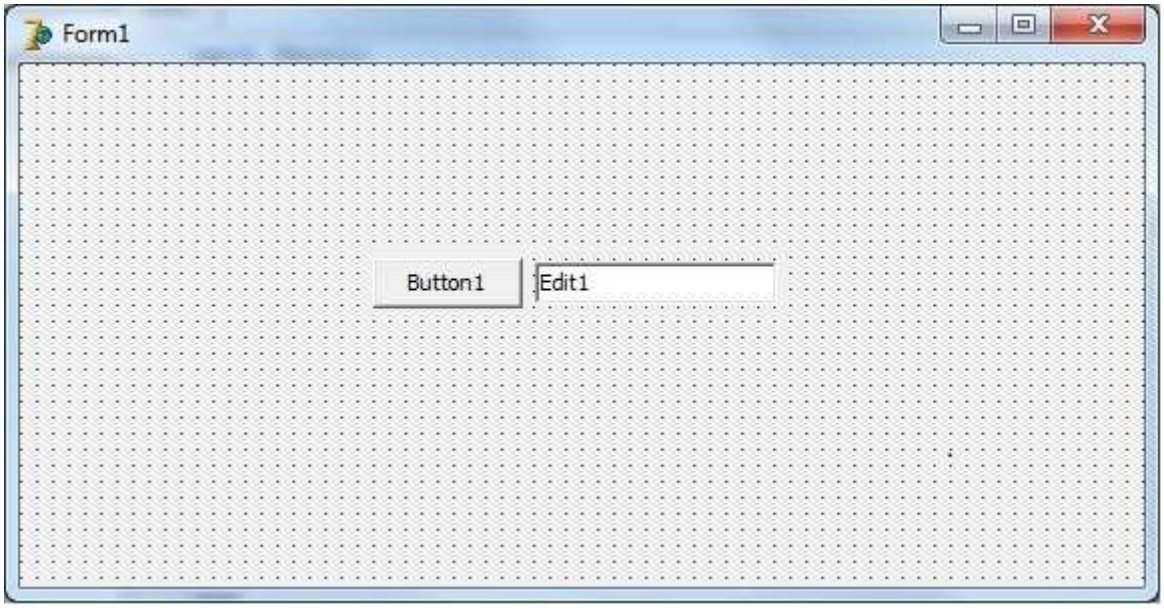
3.2 Ambiente Delphi

Neste item será apresentado o *Integrated Developer Environment* (IDE – em Português, Ambiente de Desenvolvimento Integrado) do Delphi, que possui um conjunto de ferramentas que permitem facilitar e agilizar a construção de programas, proporcionando uma melhor interação entre o programador e o ambiente de desenvolvimento.

3.2.1 Janela *Form*

O *Form* é a tela onde o desenvolvedor constrói sua aplicação. A partir de um *Form* é que se estabelece a interação, através de botões, rótulos e outros componentes, estabelecendo-se funções, métodos ou eventos que serão ativados. Os componentes são dispostos dentro da área útil do *Form,* representado na figura 3.

Figura 3 - “Form”



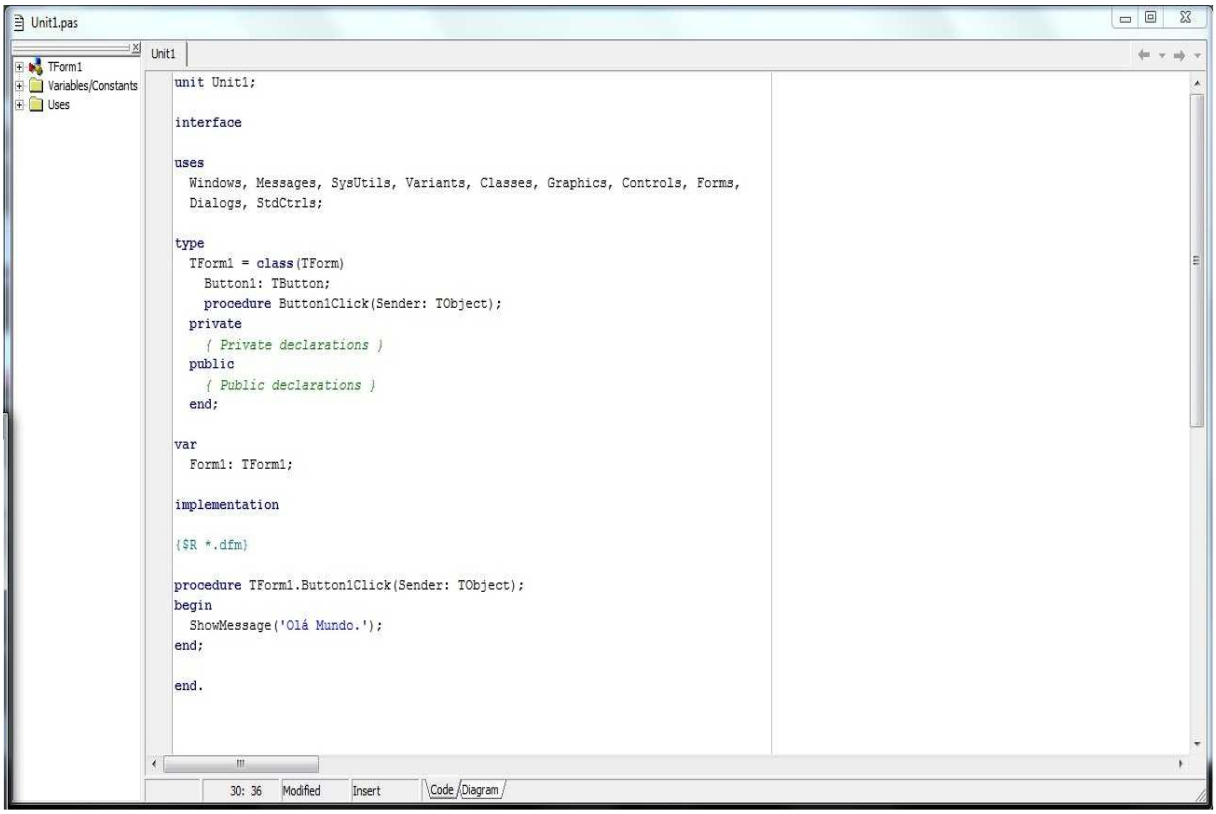
Fonte: Próprio autor.

No estudo de caso o *form* foi utilizado como o principal componente para o desenvolvimento do jogo e todos os outros componentes foram agrupados no *form*.

3.2.2 Editor de Código

O editor de código é onde código é escrito. A maneira mais óbvia de escrever código em um ambiente visual envolve responder a eventos, começando com eventos ligados a operações realizadas por usuários do programa, tais como clicar em um botão ou selecionar um item de uma caixa de seleção. Você pode usar a mesma abordagem para lidar com eventos internos, como eventos que envolvam alterações de banco de dados ou notificações do sistema operacional. A seguir na figura 4 está o editor de código.

Figura 4 - “Editor de código”



Fonte: Próprio autor.

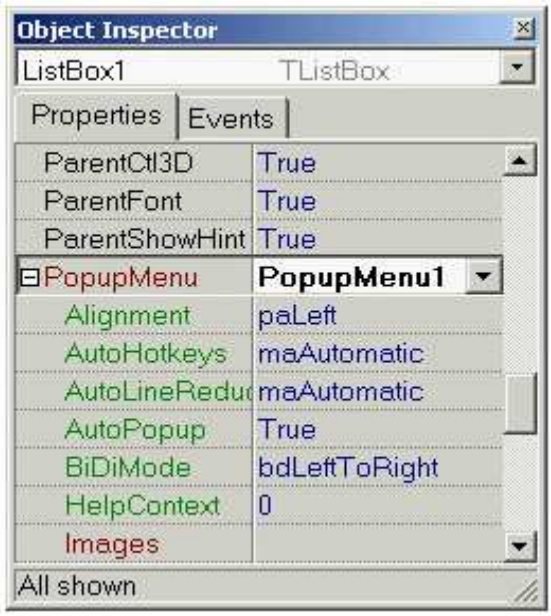
Os eventos em Delphi são uma importante característica que agilizam e facilitam a programação. Um evento é disparado quando um usuário faz alguma ação em um componente, como clicar nele, o componente que dispara um evento. Outros eventos são gerados pelo sistema, em resposta a uma chamada de método ou uma mudança de uma das propriedades do componente (ou até mesmo em um componente diferente do manipulado). Por exemplo, ao definir o foco em um

componente, aquele que atualmente tem o foco precisa perde-lo, desencadeando o evento correspondente.

3.2.3 Janela Object Inspector

Com o *Object Inspector* é possível ver e alterar as propriedades dos componentes colocados no *form* (ou em outro design) em tempo de design. Além disso a janela *Object Inspector* contém propriedades e eventos dos componentes inseridos em um *Form*, e do próprio *Form*. É na guia *Properties* (Propriedades), por exemplo, que se estabelecem as características de cada componente, como nome, fonte, altura, largura, etc. Já na guia *Events* (Eventos) estabelecem-se ações a serem tomadas pelo componente a partir de um evento associado ao mouse, teclado, sistema operacional, etc. A figura 5 é um exemplo de como é apresentado as propriedades e eventos dos componentes.

Figura 5 – "Object Inspector"



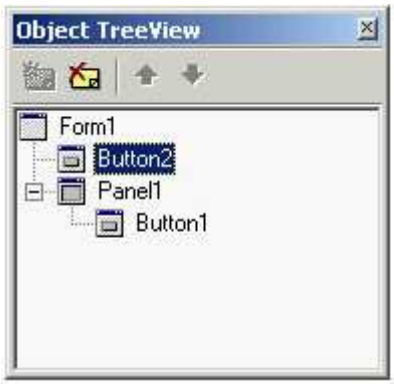
Fonte: Cantú (2003).

Para o desenvolvimento do jogo, o *Object Inspector* foi utilizado para definir as principais características dos componentes. Embora alguns parâmetros das propriedades mudem em tempo de execução configura-los antes agiliza o desenvolvimento.

3.2.4 Janela Object TreeView

O *Object TreeView* mostra todos os componentes e objetos em forma de árvore, representando suas relações. A mais óbvia é a relação pai / filho: Ao colocar um painel em um formulário, um botão dentro do painel, e um botão do lado de fora do painel, a árvore vai mostrar um botão sob a formulário e o outro sob o painel, como é representado na figura 6:

Figura 6 - "Object TreeView"



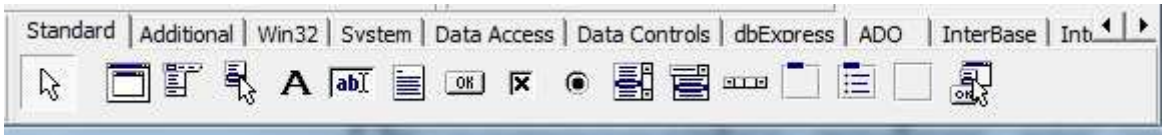
Fonte: Cantú (2003).

No desenvolvimento é importante padronizar uma hierarquia dos componentes de forma que estejam dispostos no *form* de uma forma clara, o que facilita o entendimento do projeto.

3.2.5 Paleta de Componentes

A paleta de componentes é utilizada para selecionar os componentes que se deseja adicionar ao projeto atual. A paleta de componentes é a biblioteca de classes que fornece recursos para o desenvolvimento visual em Delphi. Um exemplo de como os componentes ficam dispostos está na figura 7.

Figura 7 - "Paleta de Componentes"



Fonte: Cantú (2003).

Os componentes disponíveis na VCL (*Visual Component Library*) podem ser divididos entre:

- **Componentes Visuais** - podem ter sua forma e tamanho alterados no formulário (*Form*), além das propriedades e eventos no *Object Inspector*. Eles aparecem durante a execução do aplicativo exatamente como foram definidos durante o projeto.

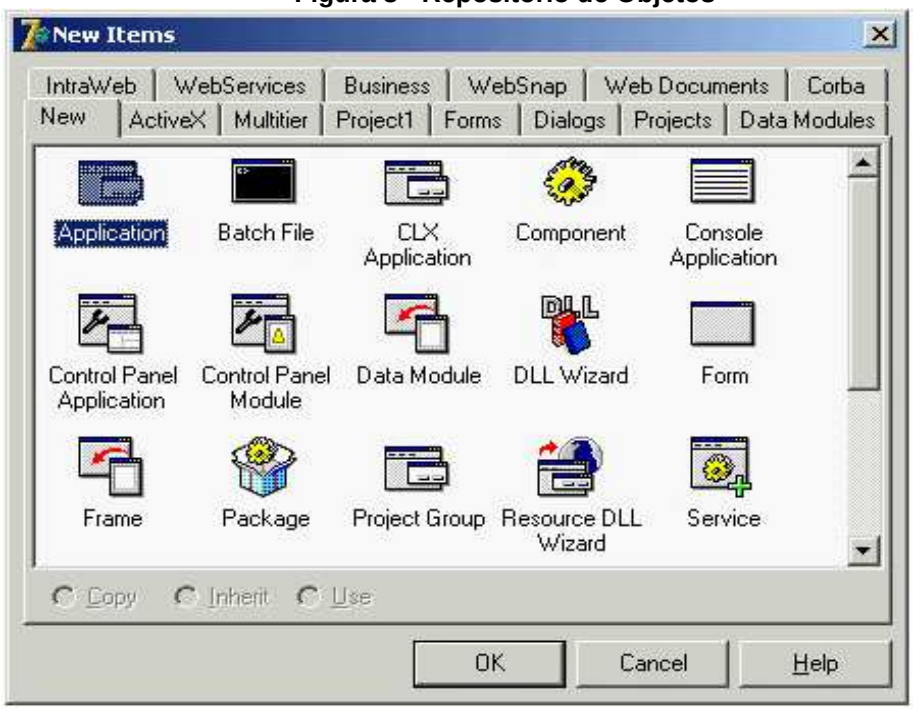
- **Componentes Não-Visuais** - ficam apenas como a representação de um ícone no formulário (*Form*), mas suas propriedades e eventos podem ser alterados no *Object Inspector*. Eles não aparecem no formulário durante a execução do aplicativo, podendo ser ativados por comandos específicos (por exemplo, podemos citar a caixa de diálogo abrir arquivo).

3.2.6 Repositório de Objetos

O Repositório de Objetos no Delphi tem comandos de menu para criar um novo *form*, uma nova aplicação, um novo módulo de dados, um novo componente, entre outros.

A caixa de diálogo *New Items* (como apresentado na figura 8) tem várias páginas, hospedando todos os elementos novos que são possíveis criar formulários e projetos armazenados no Repositório, assistentes Delphi existentes, e as formas de o projeto atual (por forma de herança). Quando é selecionado algum item do projeto atual é possível utilizar o conceito de herança no Delphi, em que toda a programação do componente atual é herdada para o novo. Como exemplo o clique de um botão do componente anterior já está implementado no novo componente, podendo ser antes complementa se necessário.

Figura 8 - Repositório de Objetos



Fonte: Cantú (2003).

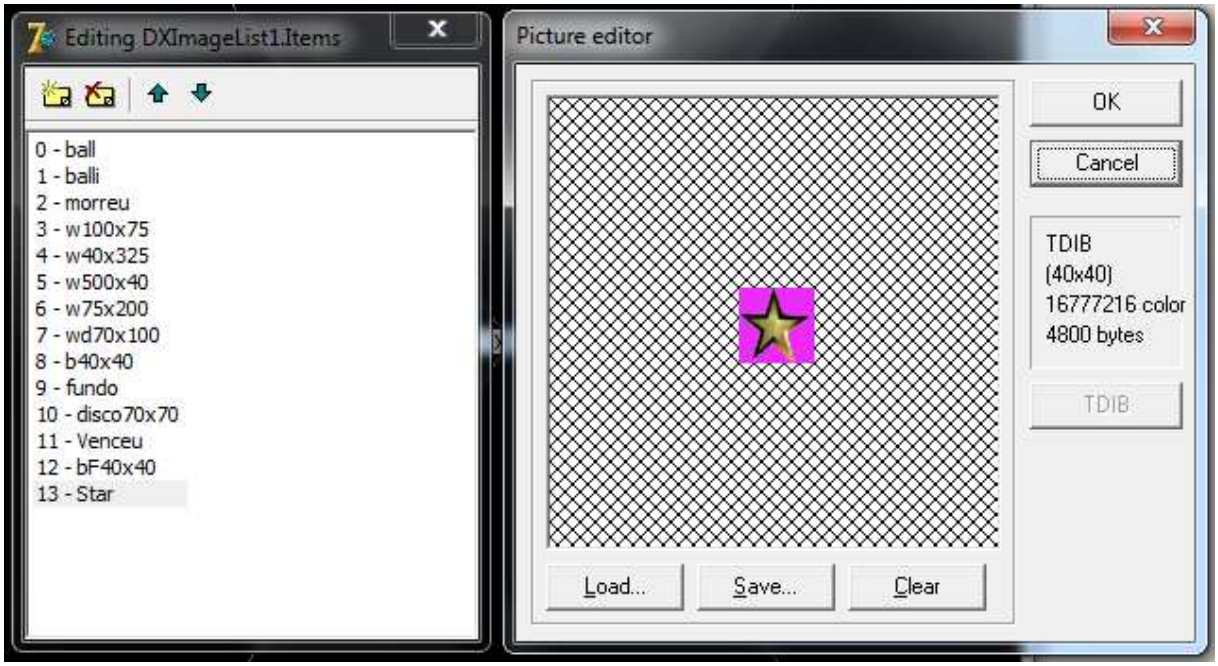
O Repositório de Objetos é o local onde podem ser acessadas de forma rápida, as principais ferramentas para o início de uma nova tela no jogo. Permite também acessar alguns objetos que foram previamente programados de forma que sirvam de base para a nova etapa da aplicação.

3.2.7 Componentes DelphiX

Os componentes DelphiX utilizados para o desenvolvimento no estudo de caso tem funções que auxiliam em algumas áreas do jogo. A seguir está a descrição dos componentes:

***TDXImageList*:** Este componente é responsável por agrupar todas as imagens em forma de uma “lista de imagens” a serem utilizadas no jogo. É possível inserir todas as imagens que necessitam ser carregadas em tempo de execução criando assim um índice para cada imagem como é apresentado na figura 9.

Figura 9 - "*ImageList*"



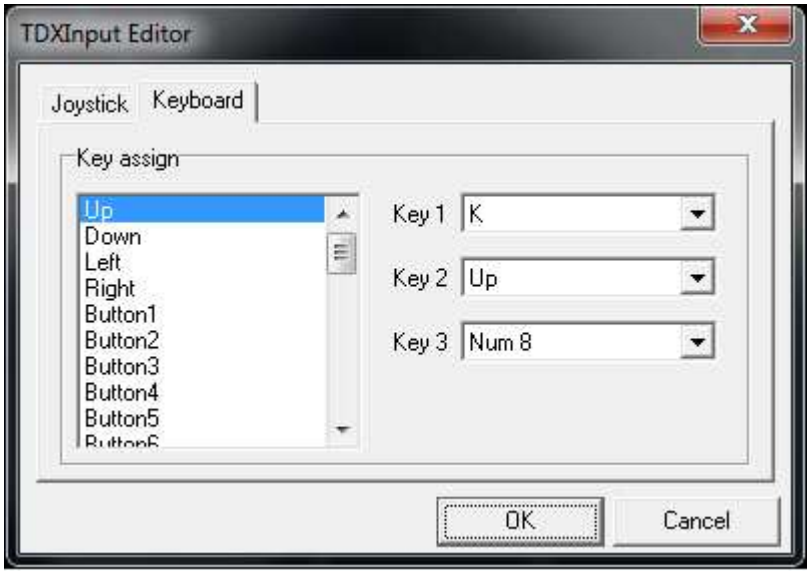
Fonte: Próprio Autor.

***TDXSpriteEngine*:** Utilizando este componente é possível realizar as principais funcionalidades do jogo, como a movimentação dos personagens e a detecção da colisão dos *sprites* do jogo.

***TDXTimer*:** Controla o “loop principal” do jogo, podendo modificar a velocidade dos intervalos. No evento OnTimer do componente é onde estão as principais funcionalidades a serem chamadas.

***TDXInput*:** Com este componente é possível configurar os comandos do jogo de forma simples, apenas atribuindo uma tecla a uma funcionalidade. Não apenas é possível reconhecer comandos do teclado, como também comandos de um controle. A figura 10 mostra a interface do componente para a configuração dos comandos.

Figura 10 - TDXInput



Fonte: Próprio autor.

***TDXSound*:** Tem a função de executar os efeitos sonoros do jogo. É possível reproduzir diversos sons em conjunto utilizando este componente.

***TDXWaveList*:** Tem o funcionamento semelhante do *TDXImageList*, porém armazena os sons do jogo os quais também ficam dispostos em forma de lista e permitem modificar algumas propriedades dos sons de forma individual, como por exempl,o se o som precisa de reproduzido continuamente ou não.

***TDXDraw*:** Componente principal do DirectX, responsável por renderizar as imagens, objetos, camadas e todos os componentes visuais utilizados. Existem propriedades intuitivas como altura e largura da tela de jogo.

3.2.7.1 Componente *TDXDraw*

Este componente é o primeiro a ser a usado quando é iniciado o desenvolvimento, necessitando o desenvolvedor estabelecer os limites de visualização da aplicação (Micrel, 2015).

3.2.7.2 Componente *TDXImageList*

No *TDXImageList* foram carregadas as imagens do jogo, que serão usadas como *sprites* no jogo. Neste componente foram carregadas as imagens em forma de lista, mas suas propriedades como a transparência no fundo podem ser configuradas individualmente (Micrel, 2015).

3.2.7.3 Componente *TDXTimer*

Para que o jogo tenha um “ritmo” o componente TDXTimer foi configurado com o *loop* principal do jogo. Configurando-se a propriedade *Interval* foi possível estabelecer a velocidade do jogo (Micrel, 2015).

3.2.7.4 Componente *TDXWaveList*

Este componente foi utilizado no jogo para armazenar os sons referentes aos personagens e ambiente do jogo. Semelhante ao funcionamento do *TDXImageList* que armazena em listas de itens, no *TDXWaveList* não é diferente permitindo também que suas configurações sejam modificadas de forma individual. Com o *TDXWaveList* também foi possível executar os sons do jogo de uma forma simples com o comando: *Items.Find*(“Nome do Som”).*Play.* Somente foi preciso substituir pelo nome do som que foi adicionado ao componente para que o som seja executado (Micrel, 2015).

3.2.7.5 Componente *TDXSpriteEngine*

Para a o desenvolvimento do jogo o *TDXSpriteEngine* foi de grande utilidade servindo com o “motor” do jogo. Através da função *DoCollision*, foi possível detectar qual o tipo de colisão estava acontecendo, como o tipo de objeto com o qual colidiu assim foi possível estabelecer um fluxo sobre qual ação deveria acontecer para que o jogo se comportasse de maneira correta (Micrel, 2015).

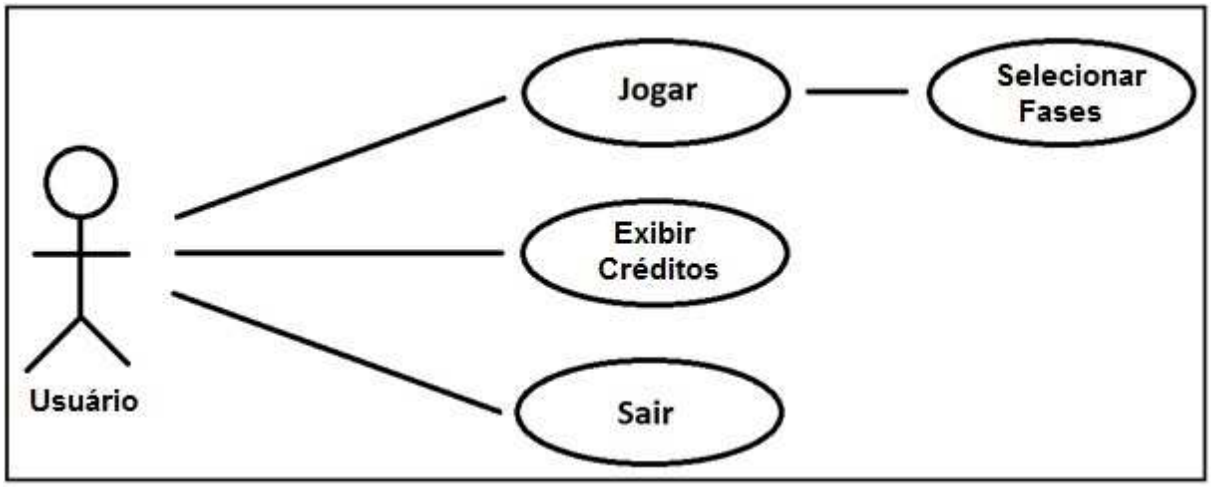
4 Estudo de Caso

Para o apoio à compreensão do texto foi desenvolvido um protótipo de um jogo do gênero de ação, utilizando a ferramenta RAD Delphi. Para auxiliar no desenvolvimento foi utilizado, além dos componentes nativos do Delph,i um componente chamando DelphiX.

4.1 Implementação dos menus

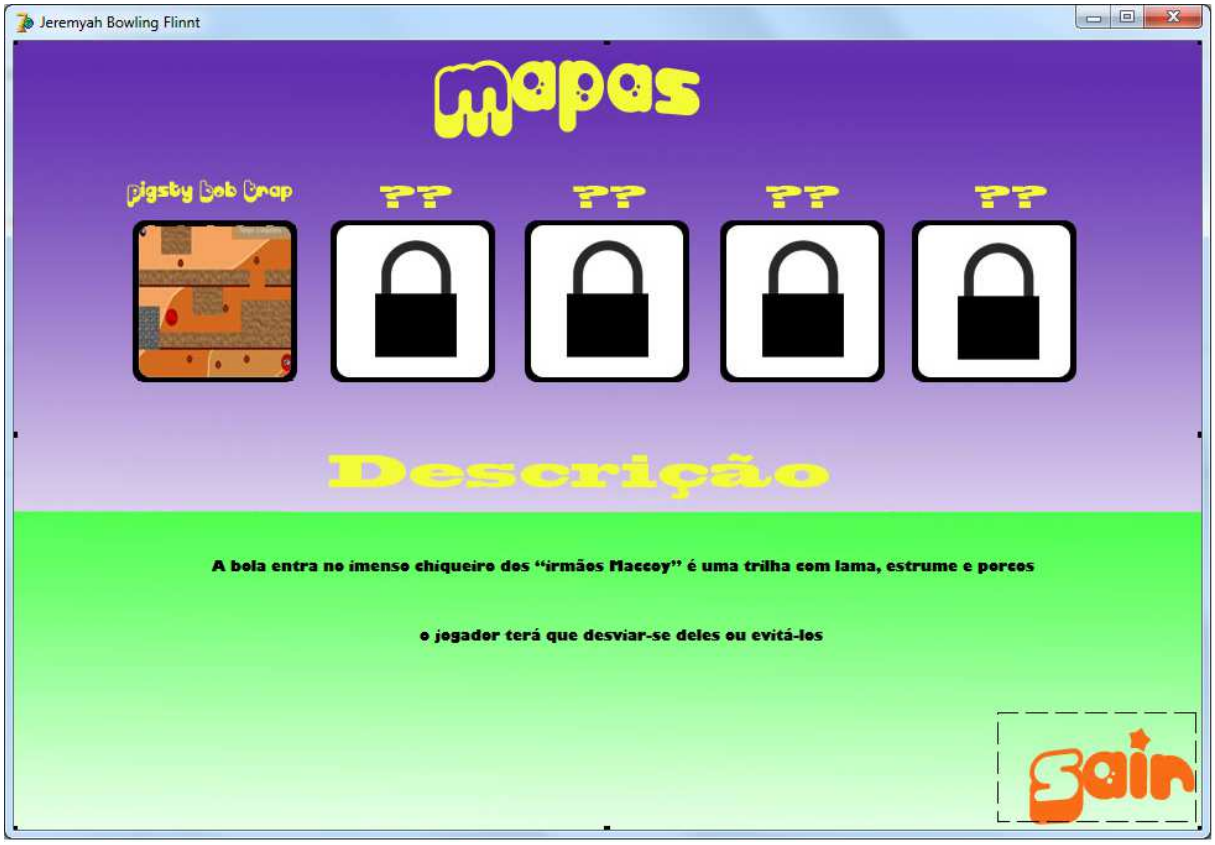
Inicialmente foram implementados os menus do jogo, com qual é possível que o jogador navegue pela interface representada na figura 11. Para a criação dos menus foram utilizados apenas *forms* e o componente TImage, o que possibilitou que as imagens dos menus possam ser exibidas no *form* como na figura 12:

Figura 11 - Diagrama de caso de uso do jogo



Fonte: Próprio Autor.

Figura 12 - Menus do jogo

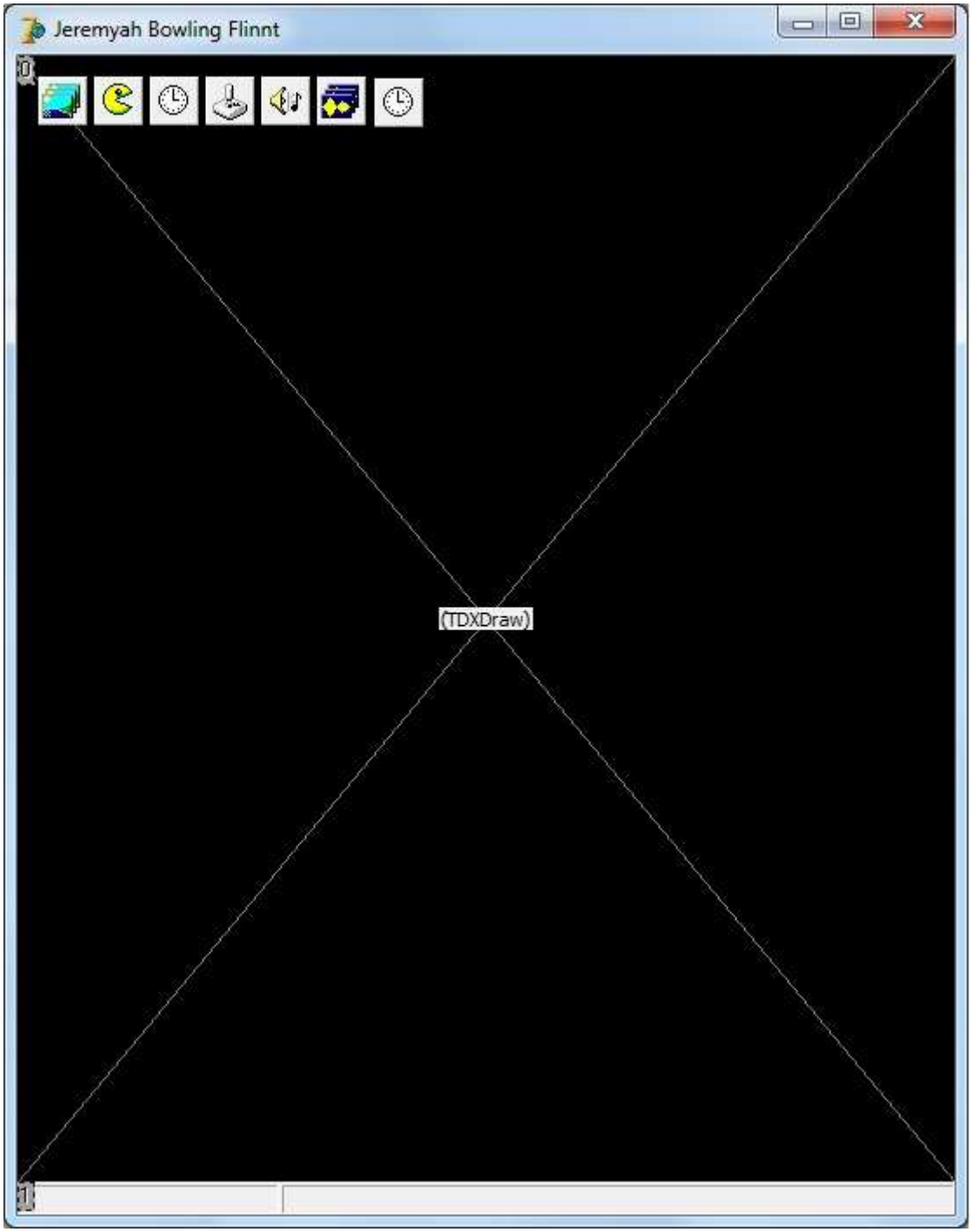


Fonte: Próprio autor.

4.2 Implementação do jogo

Para a implementação foi utilizado um *form* comum, no qual foram inseridos os componentes responsáveis pelo funcionamento propriamente dito do jogo. A figura 13 mostra os componentes já inseridos.

Figura 13 - Form principal e componentes



Fonte: Próprio autor.

4.3 Criação dos objetos

Antes de começar a programação do jogo propriamente dito, é necessário definir quais serão os objetos interativos do jogo, como: o personagem principal, os inimigos, limites do cenário (paredes e obstáculos) e pontos de objetivos.

Primeiramente serão definidas as diferentes classes para cada tipo, de “personagem” do jogo, o que facilita a interação entre eles no desenvolvimento do jogo. O código a seguir apresenta uma forma de realizar este procedimento:

```
Type
TBall = class(TImageSprite)
  public
  mov_x,mov_y:integer;
  protected
       procedure DoMove(MoveCount: Integer); override;
       procedure DoCollision(Sprite: TSprite; var Done: Boolean); override;
end;

Type
  TBotao = class(TImageSprite)
end;

Type
  TFim = class(TImageSprite)
end;

Type
TBalliH = class(TImageSprite)
  public
  mov_x,mov_y:integer;
  protected
       procedure DoMove(MoveCount: Integer); override;
       procedure DoCollision(Sprite: TSprite; var Done: Boolean); override;
end;

Type
TBalliV = class(TImageSprite)
  public
  mov_x,mov_y:integer;
  protected
       procedure DoMove(MoveCount: Integer); override;
       procedure DoCollision(Sprite: TSprite; var Done: Boolean); override;
end;

Type
TBalliSpec = class(TImageSprite)
  public
  mov_x,mov_y:integer;
  protected
       procedure DoMove(MoveCount: Integer); override;
```

```
end;

Type
  TParede = class(TImageSprite)
end;
```

Dessa forma há classe do personagem principal (TBall), inimigos (TBalliH, TBalliV e TBalliSpec), limites do cenário (TParede), objetos que tem funcionalidades específicas (TBotao e TFim) e as mensagens a serem exibas na tela (Tmsg).

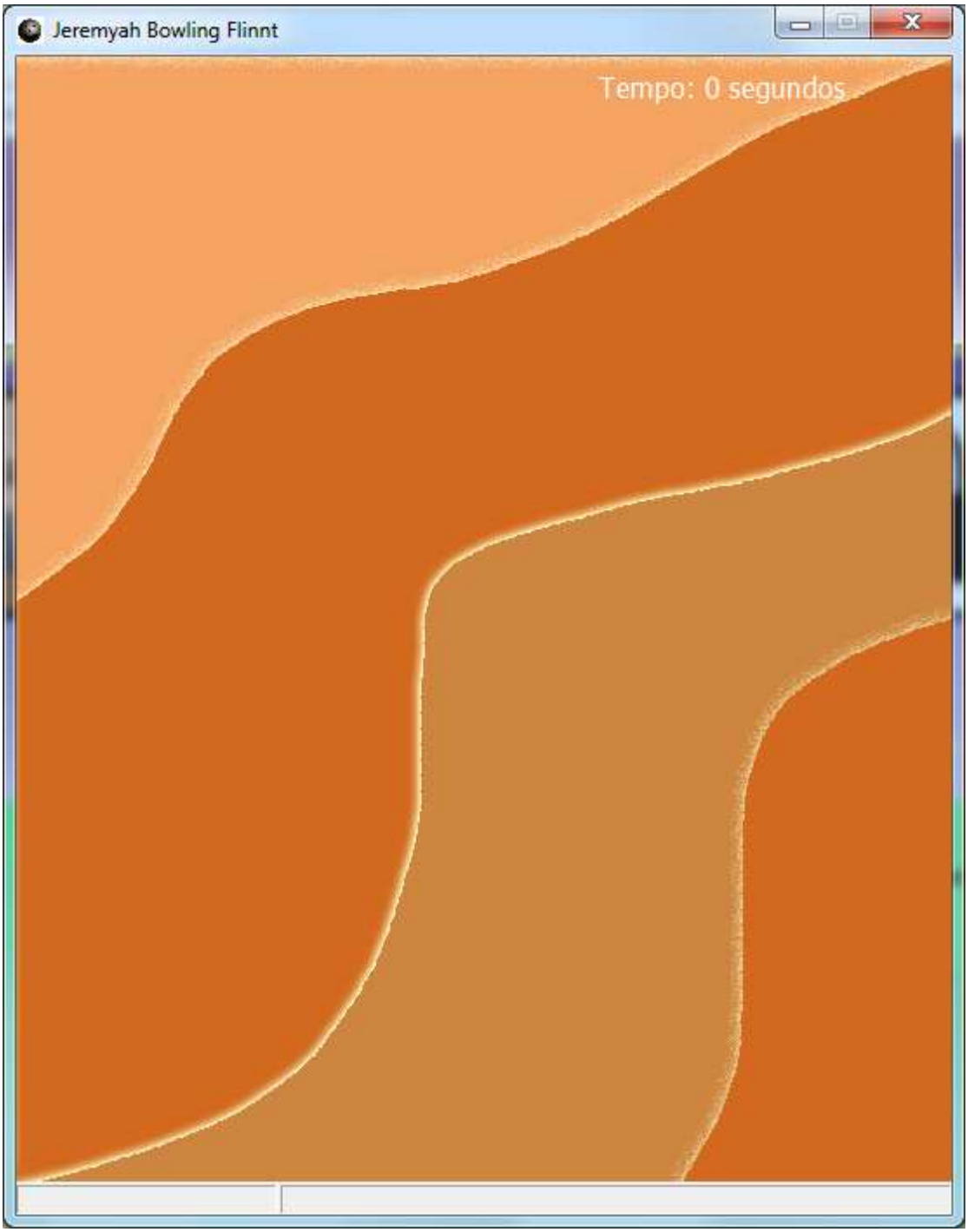
4.4 Construindo o cenário

O cenário do jogo é composto pelo fundo, os obstáculos e informações sobre o jogo, tais como: o tempo decorrido desde o início da fase, quantidade de vidas restantes e o nome da fase atual. A seguir está o algoritmo utilizado para criarmos o fundo e exibir o tempo atualizado da partida, ele foi implementado no evento OnTimer do componente TDXTimer, a imagem do fundo foi carrega através do componente TDXImageList, conforme código a seguir.

```
procedure TForm1.DXTimer1Timer(Sender: TObject; LagCount: Integer);
begin
        if not DXDraw1.CanDraw then exit;

        DXInput1.Update;
        DXSpriteEngine1.Move(LagCount);
        DXSpriteEngine1.Dead;
        DXImageList1.Items.find('fundo').Draw(DXDraw1.Surface,0,0,0);
        DXSpriteEngine1.Draw;

        with DXDraw1.Surface.Canvas do
        begin
                Brush.Style := bsClear;
                Font.Color := clwhite;
                Font.Size := 12;
                Textout(335, 8,'Tempo: '+inttostr(tempo) +' segundos');
                Release;
        end;
end;

DXDraw1.Flip;
```

Com este algoritmo obteve-se o seguinte resultado, que pode ser observado na figura 14:

Figura 14 - Fundo do jogo



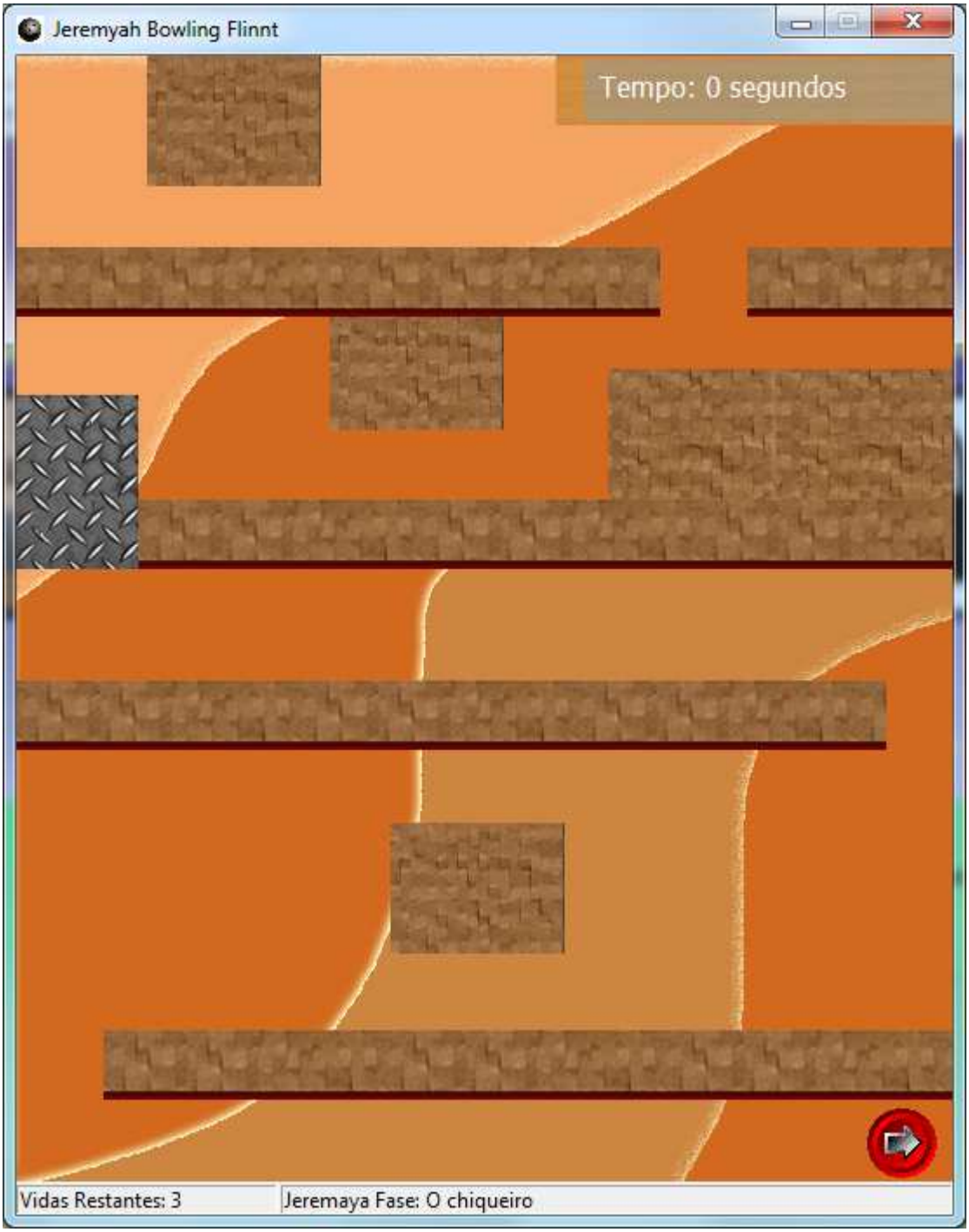
Fonte: Próprio autor.

Após a implementação do fundo é necessário adicionar as paredes e informações que serão exibidas ao jogador. Como já existem as classes dos obstáculos que compõem o cenário, somente é necessário criar os *sprites* (objetos gráficos geralmente bidimensionais que se movem numa tela) atribuindo-lhes sua classe. O algoritmo a seguir é um exemplo de como exibir com as informações do jogo e criar o *sprite* das paredes e as propriedades básicas como textura, posicionamento no ambiente do jogo e suas proporções, conforme código a seguir.

```
procedure TForm1.FormCreate(Sender: TObject);
var
  ParedeSprite:    Tsprite;
  BotaoSprite:     Tsprite;
  FimSprite:       Tsprite;

begin

  vidas := 3;
  Form1.StatusBar1.Panels[0].text := 'Vidas Restantes: '+inttostr(vidas);
  Form1.StatusBar1.Panels[1].text := 'Jeremaya Fase: O chiqueiro';

  ParedeSprite := TParede.Create(Form1.DXSpriteEngine1.Engine);
  with TParede(ParedeSprite) do
  begin
    Image := DXImageList1.Items.Find('w100x75');
    X := 75; Y := 0; Z := 2;
    Height:= 75; Width := 100;
  end;

end;
```

Com o resultado após implementação deste trecho, já é possível ver construção do cenário na figura 15:

Figura 15 - Mapa do jogo



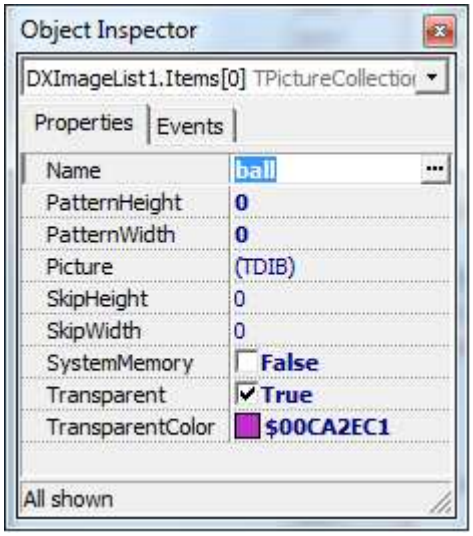
Fonte: Próprio Autor.

4.5 Personagens do jogo

Neste tópico serão inseridos os personagens do jogo. Semelhantemente aos cenários do jogo eles também são *sprites*, mas com algumas funcionalidades a mais como movimentação e sistema de detecção de colisões.

As imagens dos personagens foram adicionadas ao componente *TDXImageList*. Foi definido no jogo que a cor rosa serviria como chave para a transparência. Esta opção pode ser configurada diretamente pela interface do componente (propriedade *TransparentColor*) como ilustrado na figura 16:

Figura 16 - Interface TDXImageList



Fonte: Próprio Autor.

Agora é necessário realizar o posicionamento dos personagens no cenário do jogo, o código a seguir apresenta uma forma de realizar este passo:

```
procedure TForm1.FormCreate(Sender: TObject);
var

  PlayerSprite:          TSprite;
  InimigoSprite:         TSprite;
  InimigoSpecSprite: Tsprite;

  PlayerSprite := TBall.Create(Form1.DXSpriteEngine1.Engine);
  with TBall(PlayerSprite) do
  begin
    Image := DXImageList1.Items.Find('Ball');
    X := 5; Y := 13; Z := 2;
    mov_x:= 0; mov_y:= 0;
    Height:= 20; Width := 20;
  end;
```

```
InimigoSprite := TBalliH.Create(Form1.DXSpriteEngine1.Engine);
with TBalliH(InimigoSprite) do
begin
  Image := DXImageList1.Items.Find('Balli');
  X := 70; Y := 85; Z := 2;
  mov_x:= 0; mov_y:= 0;
  Height:= 20; Width := 20;
end;

InimigoSprite := TBalliV.Create(Form1.DXSpriteEngine1.Engine);
with TBalliV(InimigoSprite) do
begin
  Image := DXImageList1.Items.Find('Balli');
  X := 383; Y := 85; Z := 2;
  mov_x:= 0; mov_y:= 0;
  Height:= 20; Width := 20;
end;

InimigoSpecSprite := TBalliSpec.Create(Form1.DXSpriteEngine1.Engine);
with TBalliSpec(InimigoSpecSprite) do
begin
  Image := DXImageList1.Items.Find('disco70x70');
  X := 130; Y := -100; Z := 2;
  mov_x:= 0; mov_y:= 0;
  Height:= 70; Width := 70;
end;
```

Com os personagens posicionados no cenário é necessário implementar a movimentação dos personagens. Ao implementar a movimentação é preciso se atentar aos limites do cenário, que podem ser verificados através das propriedades *ClientWidth* e *ClientHeight* do componente *TDXDraw*. Todos os procedimentos estão exemplificados na *procedure* a seguir:

```
procedure TBall.DoMove(MoveCount: Integer);
begin
  Collision;
  mov_x :=0;
  mov_y :=0;
   if (X < 0) then
   begin
      mov_x:= 1;
      Form1.DXWaveList1.Items.Find('Sound1').Play(False);
   End
   else
   if (X + Width > Form1.DXDraw1.ClientWidth) then
   begin
    mov_x:= -1;

    Form1.DXWaveList1.Items.Find('Sound1').Play(False);
   end
   else
   if isLeft in Form1.DXInput1.States then
      mov_x:= -5
   else
   if isRight in Form1.DXInput1.States then
      mov_x:= 5;

   if (Y < 0) then
   begin
      mov_y:= 1;
      Form1.DXWaveList1.Items.Find('Sound1').Play(False);
   end
   else
   if (Y + Height > Form1.DXDraw1.ClientHeight) then
   begin
    mov_y:= -1;
    Form1.DXWaveList1.Items.Find('Sound1').Play(False);
   end
   else
   if isDown in Form1.DXInput1.States then
      mov_y:= 5
   else
   if isUp in Form1.DXInput1.States then
   begin
      mov_y:= -5;
   end;

   X := X + mov_x;
   Y := Y + mov_y;
```

Além da movimentação outra funcionalidade essencial para o funcionamento é o sistema de colisão. Antes mesmo de realizar um movimento, verifica-se se o personagem colidiu com algum dos diferentes objetos do cenário. Para cada tipo de objeto que o personagem principal venha a colidir uma ação diferente é acionada. Caso o objeto seja do tipo “Tparede” o personagem não pode avançar naquela direção em que foi ocasionada a colisão. Quando é um objeto do tipo “Tinimigo”, o personagem volta a sua posição inicial e é descontada uma vida. E existem também os tipos que causam modificações, anteriores no cenário como o TBotao. Após a colisão ser detectada é possível realizar mais ações como tocar um som predeterminado para que o jogador tenha um *feedback* da ação realizada. No trecho a seguir estão as implementações:

```
procedure TBall.DoCollision(Sprite: TSprite; var Done: Boolean);
begin

  if ((Sprite is TBalliH) or (Sprite is TBalliV) or (Sprite is TBalliSpec))then
  begin
    X := 13;
    Y:= 5;
    vidas := vidas - 1;
    Form1.StatusBar1.Panels[0].text:= 'Vidas Restantes: '+inttostr(vidas);
    Form1.DXWaveList1.Items.Find('Sound2').Play(False);
    if (vidas = 0 ) then
    begin
      dead;
      Tmsg(msg).mov_y:= 150;
      start := 50;
      Form1.DXWaveList1.Items.Find('ambi').Stop;
      Form1.DXWaveList1.Items.Find('morreu').Play(False);
    end;
  end;

  if (Sprite is TBotao)then
  begin
    TParede(ParedeDesSprite).X:= 50;
    TBalliSpec(InimigoSpecSprite).mov_y := 5;
  end;
```

```
  if (Sprite is TParede)then
  begin
   Form1.DXWaveList1.Items.Find('Sound1').Play(False);
    if mov_x > 0 then
    begin
       X:=X -10;
    end;
    if mov_x < 0 then
    begin
       X:=X +10 ;
    end;

    if mov_y > 0 then
    begin
       Y:=Y -10;
    end;
    if mov_y < 0 then
    begin
       Y:=Y +10 ;
    end;
  end;

  if (Sprite is TFim)then
  begin
    Form1.DXWaveList1.Items.Find('ambi').Stop;
    Form1.DXWaveList1.Items.Find('tuturu').Play(False);
    dead;
    Tmsg(msgV).mov_y:= 150;
    if (tempo < 35 ) then
    begin
      Tmsg(S3).mov_y:= 200;
    end;
    if (tempo < 50 ) then
    begin
      Tmsg(S2).mov_y:= 200;
    end;
    if (tempo < 60 ) then
    begin
      Tmsg(S1).mov_y:= 200;
    end;

    start := 50;
  end;

end;
```

5 Considerações Finais

Avaliando o projeto desenvolvido, conclui-se que os principais objetivos foram alcançados, visto que este trabalho foi desenvolvido para apresentar a tecnologia RAD no desenvolvimento de jogos 2D. Com relação ao uso da tecnologia Delphi foi possível criar um jogo sem muitos obstáculos para o desenvolvimento.

O Delphi, como plataforma de desenvolvimento de jogos em termos de recursos não se mostrou tão eficiente quanto outras ferramentas existentes no mercado especializadas em desenvolvimento de jogos, mas, em contrapartida, o Delphi apresentou uma forma simples de desenvolvimento sendo indicado até para pessoas com pouca experiência na área, podendo ser uma porta de entrada para quem quer ingressar no campo de desenvolvimento de jogos.

O desenvolvimento utilizando a metodologia RAD foi eficiente e principalmente rápido, duas características fundamentais para o projeto, que visa um desenvolvimento ágil (em um curto espaço de tempo). O planejamento detalhado de todas as funções presentes no projeto ajudou para que a escalabilidade não fosse drasticamente alterada, o que prejudicaria o desenvolvimento rápido.

Uma sugestão para futuras implementações é trocar o fundo do jogo com imagens estáticas por um sistema de *Parallax* e com isso criar cenário maiores proporcionando novas interações do ambiente com o jogador.

Para um trabalho futuro, sugere-se desenvolver um jogo com uma mecânica semelhante, mas com *multiplayer* cooperativo e a opção de jogar online. Com esta adição, será possível desenvolver várias formas de resolver novos obstáculos utilizando a mecânica de dois ou mais jogadores.

Referências Bibliográficas

Agile Manifesto (2001). Disponível em: <http://agilemanifesto.org/>, acessado em 6 de março de 2015.

Battaiola, A. L. **Jogos por computador: Histórico, relevância tecnológica e mercadológica, tendências e técnicas de implementação**. Anais do XIX Jornada de Atualização em Informática, 2000.

Cantu, M. **Mastering Delphi 7.** Califórnia, Alameda. Sybex, 2003.

Embarcadero (2015). Disponível em: <https://www.embarcadero.com/br/>, acessado em 30 de Março de 2015.

Huizinga, J. **Homo ludens: o jogo como elemento da cultura**. 5. Ed. São Paulo. Perspectiva, 2003.

McConnell, Steve C. **Rapid Development: Taming Wild Software Schedules**. Microsoft Press.Redmond, Washington. 1996.

Micrel (2015). Disponível em: <http://www.micrel.cz/Dx/>, acessado em 21 de maio de 2015.

Pressman, Roger. S. **Engenharia de Software**. 6. Ed. Tradução Ariovaldo Greiesi e Mario Moro Fecchio. São Paulo. AMGH, 2006.

Schuytema, P. **Design de games: uma abordagem prática**. São Paulo: Cengage Learning, 2008.