Um Framework para criação de jogos *towerdefense* baseado no motor ORX.

Gabriel M. Morais, Rodrigo P. Machado

Instituto Federal de Educação, Ciência e Tecnologia do Rio Grande do Sul - Campus Porto Alegre.
Curso de Tecnologia em Sistemas para Internet

Rua Coronel Vicente, 281. Centro Histórico - Porto Alegre CEP 90.030-041

{macielmorais90@gmail.com, rodrigo.prestes@poa.ifrs.edu.br}

Resumo. *Este trabalho apresenta um framework para construção de jogos do gênero towerdefense. O objetivo do trabalho é a construção do framework de desenvolvimento de um jogo de towerdefense que apresente uma taxa de quadros por segundo acima de 30. O projeto foi desenvolvido utilizando o motor Orx e linguagem C++. Foram desenvolvidas classes contendo métodos básicos que representam o comportamento geral presentes neste estilo. Serão demonstradas as funcionalidades das classes construídas e os testes de avaliação da taxa de quadros por segundo alcançados para verificação do desempenho. Com este trabalho concluiu-se que, a partir do foco em um estilo de jogo, pode ser desenvolvido um framework que prove uma estrutura básica para o desenvolvedor trabalhar a lógica do jogo.*

1. Introdução

O mercado de jogos no Brasil fatura mais de vinte milhões de reais por ano. As empresas mais antigas do mercado surgiram há 13 anos [Acigames, 2012] o que demonstra ser uma área relativamente nova no país. O Brasil começa a investir neste segmento e o governo, para estimular o desenvolvimento da área, organizou concursos que promovem a competição entre os desenvolvedores com prêmios em dinheiro [Ministério da Cultura, 2013]. Também estão ocorrendo mudanças no setor tributário, tendo o município Eng. Paulo Frontin, do Rio de Janeiro, reduzido tributos para empresas da área [Acigames, 2013]. Com isso, observa-se uma área do mercado que ascende em oportunidades e torna-se interessante a busca por novas soluções e melhorias nas tecnologias existentes.

Além do interesse monetário, pode-se ressaltar a existência de diversos estudos sobre os benefícios de jogos no contexto educacional de crianças e adultos. Através do jogo e de sua ligação com o prazer e desafio proporcionados é possível ao indivíduo expressar

sua criatividade, conhecer a si mesmo, tornar-se independente, melhorar a compreensão do mundo à sua volta, desenvolver a habilidade de entender e seguir regras [Andrade Pedroso, Crislaine, et al, 2013].

Tendo em vista este cenário, procurou-se trabalhar o problema da falta de motores de jogos (MJ) com ferramentas prontas para estilos de jogos simples e que possuem regras específicas poupando tempo de desenvolvimento, por exemplo, provendo estruturas pré-definidas para inimigos e jogadores em um jogo de tiro. Para este projeto foi escolhido o estilo de jogo *towerdefense (TD)* por possuir poucas estruturas e pela fácil compreensão de seus comportamentos, assim como ausência de física sobre os objetos. Através da abordagem a um único estilo, nesse caso o *towerdefense*, foram observados estruturas e comportamentos que se repetem nos jogos existentes nessa categoria e foram construídas classes para representar as principais estruturas de um jogo de TD. A par disso, procuramos prover o programador de classes e métodos direcionados aos problemas que são encontrados no desenvolvimento deste estilo de jogo como movimentação, interação entre os objetos e o controle deles. Jogos de um mesmo estilo compartilham características semelhantes entre si, através da identificação destas características busca-se criar um *framework* [Dirk, 2000] que facilite a sua construção. No estilo *towerdefense*, foi estudada sua estrutura básica, seus elementos fundamentais, regras e foram definidas as principais classes e métodos presentes na lógica do jogo.

O motor de jogos é uma das principais ferramentas utilizadas no desenvolvimento de jogos, portanto, foi necessário eleger o mais apropriado ao projeto devido às diferentes características que cada um apresenta como linguagem, plataforma, bibliotecas usadas e até mesmo a licença. Optou-se por escolher o motor Orx como alvo para o projeto por possuir licença que permite alterações e melhorias no seu código, possuir maior número de plataformas compiláveis, não possuir custos e estar com seu projeto em desenvolvimento ativo.

O objetivo do trabalho é construção de um *framework* para jogos *towerdefense*. Os programadores possuirão algumas das estruturas prontas e poderão ampliá-las e personalizá-las. Através de classes que foram desenvolvidas será possível a prototipação de um jogo no estilo *towerdefense*. Com a simples definição de algumas variáveis como número de inimigos, torres, rota a ser percorrida pelos inimigos, velocidade dos objetos na tela e também das imagens que serão usadas. Contudo, o *framework* permitirá que o programador incremente as funcionalidades e construa o diferencial de seu jogo sobre as classes disponibilizadas. Não foram abordados aspectos que são genéricos a todos os jogos e não fazem parte diretamente da lógica do jogo, como a construção de menus e controle de entrada de dados.

As seções seguintes deste artigo expõem informações sobre os componentes do projeto e sobre seu desenvolvimento. Nas seções 2 e 3 estão descritos conceitos relacionados ao trabalho; as seções 4 e 5 apresentam as tecnologias e as ferramentas utilizadas; a seção 6 apresenta o desenvolvimento do trabalho; a seção 7 descreve as validações utilizadas; a seção 8 traz a conclusão em que são discutidos os resultados obtidos, assim como as possíveis melhorias e trabalhos futuros.

2. O estilo *towerdefense*

A definição de jogo na computação é definida por [Koster, 2004] como uma experiência interativa que provê ao usuário uma sequência crescente de desafios correlacionados que eventualmente serão ultrapassados.

O estilo de jogo *towerdefense (TD)* classifica-se com um jogo de estratégia em tempo real (*RTS – Real Time Strategy*). Neste estilo de jogo os inimigos percorrem a tela de uma posição de início até uma posição final por um caminho predeterminado em uma velocidade constante. Ao jogador é permitido colocar torres ao longo do caminho, sem obstruí-lo, e que automaticamente atacam os inimigos na sua área de alcance. Assim que todos os inimigos forem derrotados ou escaparem, uma nova quantidade de inimigos surgirá no início no caminho propondo um novo desafio. O jogador inicia o jogo com um determinado valor em dinheiro virtual e adquire mais a cada inimigo derrotado utilizando-o para obter as torres. O fim do jogo é determinado quando um número predefinido de oponentes conseguirem escapar [Wetzel, 2011]. Vale ressaltar que estas regras são base do estilo, mas podem ser modificadas pelo *designer* de jogos.

Modificações comuns neste estilo são encontradas na movimentação dos inimigos, permitindo tomarem caminhos diferentes para aumentar a dificuldade. Existem jogos em que há a ação do jogador diretamente nos inimigos através de habilidades adquiridas. Também ocorre a disponibilização de novas estruturas no jogo a fim de expandir as possibilidades de estratégia e aumentar o tempo de jogo como “geradores” que seriam uma segunda fonte de recursos para adquirir novas torres.

3. Motores de Jogos

Motores de jogos são conjuntos de funções construídas para o uso no desenvolvimento de um jogo. Segundo [Gregory, 2009] uma das principais características de um motor de jogos é a arquitetura orientada aos dados ou *data-driven architecture*. Neste tipo de codificação valores mutáveis como a quantidade de inimigos, de vida e outros atributos importantes são armazenados em arquivos de configuração ou na linguagem de *script* utilizada. Desta forma, o código possui a capacidade de ser reutilizado e os ajustes nos comportamentos são modificados mais facilmente pelo desenvolvedor do jogo. Um MJ deve possuir a

capacidade de reutilização alta. Entretanto, não existe uma definição exata sobre os limites de especificidade que um motor de jogos pode atingir [Gregory, 2009].

Então, um motor construído especificamente para um determinado jogo poderia existir tendo em vista aspectos como desempenho e segurança frequentemente presentes em jogos de grande porte como os jogos de interpretação de personagens *online* e em massa para múltiplos jogadores (*Massively Multiplayer Online Role-playing Game*, *MMORPGs*) que necessitam de uma grande infraestrutura de banco de dados e rede, pois existem muitos usuários simultâneos, ou os chamados triplo A (*tripleA*), jogos desenvolvidos por grandes estúdios com orçamentos milionários.

4. O Motor Orx

Orx é um motor de jogos portável, leve, orientado a dados (*data-driven*) para jogos 2D, contudo pode gerar gráficos 3D. Foi criado para a construção e prototipação rápida de jogos [Orx, 2013], pois fornece ao programador diversas estruturas predefinidas e diversas funções. Seu código é desenvolvido na linguagem C, mas ele possui complementos (*plug-ins*) nas linguagens C++ que permitem trabalhar com o paradigma de orientação a objetos, e em *Objective-C*.

Sua arquitetura pode ser simplificada conforme se observa na figura (1) em que através do uso de bibliotecas já existentes como: SDL [SDL, 2013], OpenGL [OpenGL, 2013] e outras, cria-se uma nova abstração ao desenvolvedor que possui acesso direto a apenas um único *framework*.

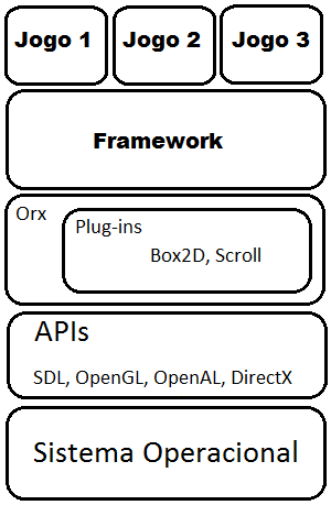

Figura 1. Representação simplificada do motor Orx

Sua principal estrutura é o *orxObject* capaz de armazenar diversas outras estruturas como relógios(*timers*), gráficos, animações, sons, textos e outros, permitindo, por exemplo, criar uma animação com textos e sons atribuídos à mesma estrutura.

Sua licença é a Zlib [Opensource, 2013] que permite usá-lo e modificá-lo livremente inclusive em projetos comerciais, sem nenhum custo e com pequenas restrições. Seu código é multiplataforma e, atualmente, pode ser compilado para *Windows*, *Linux*, *MacOS* X, *iPhone*, *iPodTouch* e *Android*. Tornou-se uma escolha atrativa por suas características técnicas e por ser um projeto em andamento no qual os responsáveis estão dispostos a corrigirem bugs e desenvolverem melhorias propostas pela comunidade através de sua página na internet.

Um dos *plug-ins* mais relevantes ao projeto é o *Scroll.* Através dele é possível que uma estrutura em linguagem C do motor Orx que possui texturas, sons e outras variáveis, seja manipulada por uma classe própria que deve, obrigatoriamente, estender à classe *ScrollObject* permitindo uma abordagem orientada a objetos sobre essas estruturas. Desta forma, podem-se implantar comportamentos iguais para um mesmo tipo de estrutura. Além disso, são herdados métodos *OnCreate*, *Update* e *OnDelete* que são executados automaticamente na instanciação, atualização e remoção, respectivamente, do objeto Orx, facilitando o controle dos objetos pelo programador. As variáveis de todos os objetos instanciados pelo Scroll podem ser lidas a partir dos arquivos de configuração e atribuídos ao método *OnCreate* facilitando os ajustes de variáveis fora do código fonte.

Os arquivos de configuração, figura 2, do motor Orx são arquivos texto que possuem, basicamente, três elementos. A chave (*key*) é um nome que identifica um valor (*value*). Valores podem representar caminhos a arquivos, valores absolutos como a latência de um relógio, referências a outras seções ou alguma das estruturas definidas pelo Orx como vetores, por exemplo. Por fim, seções auxiliam o motor a identificar coleções de pares de chave e valor pertencentes à mesma estrutura.

```
[Seção]
chave = valor;

[Torre]
Gráfico = G-Torre
Posição = (200,500,0)
Poder = 10
Alcance = 250.0
IntervaloDeAcao = 0.5

[G-Torre]
Textura = imagem.jpg
```

Figura 2. Representação do arquivo de configuração.

Vetores são estruturas que representam uma grandeza vetorial. Possuem orientação e são geralmente representados por setas. Podem representar a gravidade ou a direção de uma bola arremessada e também podem indicar uma simples posição. Possuem um valor para cada um dos eixos x, y e z, onde o terceiro é utilizado para representar a profundidade. Neste trabalho foram usados vetores do motor Orx que apresentam três dimensões representadas em valores do tipo *float*, mas como o ambiente de jogo é representado em apenas duas dimensões o valor de profundidade foi ignorado. Os vetores são usados para representar a posição de um elemento na tela. Através da posição atual e de um destino é calculada a direção na qual o elemento deve se mover, subtraindo esses vetores e através de um valor predeterminado, a velocidade é multiplicada sobre o vetor resultante da subtração e normalizado.

5. O *Framework* desenvolvido

A proposta deste trabalho foi de construir um *framework*, conforme figura 3, que facilite o desenvolvimento de um jogo no estilo *towerdefense*. As estruturas criadas formam uma camada de software colocada sobre o motor ORX, comportando-se como um complemento e não funcionam independentemente. Foram construídas classes representando os principais elementos no jogo. A seguir serão apresentadas as suas características e funções e descritas as classes do *framework*.

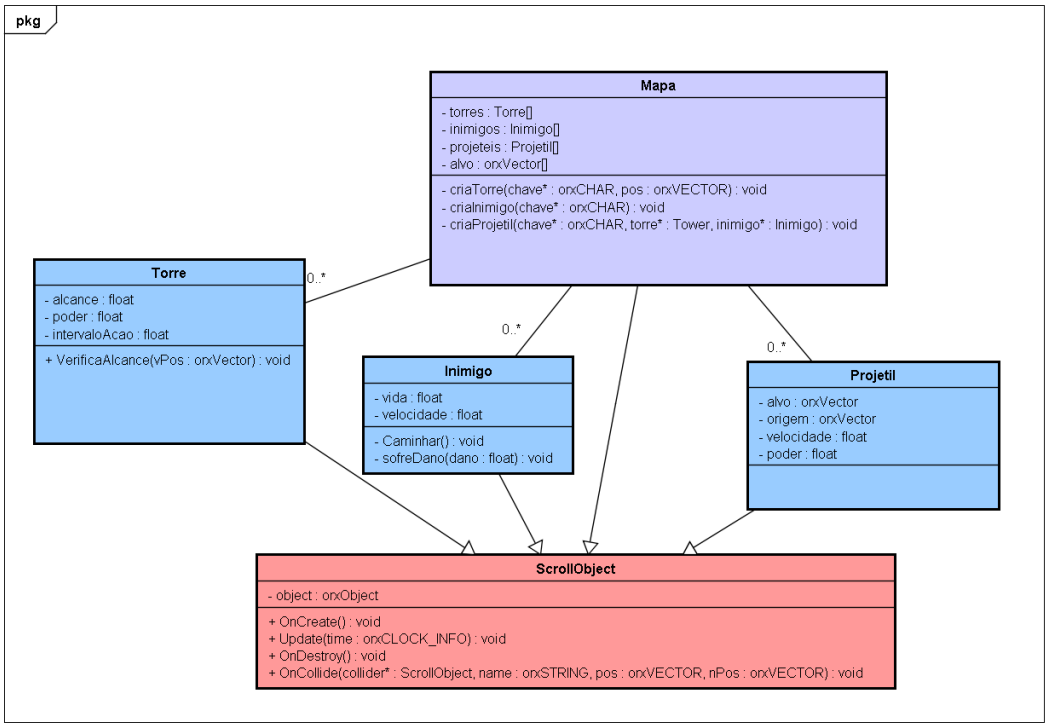

Figura 3: Representação simplificada das classes para melhor visualização do contexto geral.

A classe Mapa (*Map*) representa o cenário no qual os elementos irão interagir e será responsável por controlar as demais estruturas. Possui uma lista de vetores que representam o caminho por onde os inimigos devem passar. Os valores são lidos no arquivo de configuração. Possui dois *arrays* com as estruturas de inimigos e de torres, portanto as iterações de busca de inimigos pelas torres e criação de projéteis ocorrem no método *update* desta classe. Também é nesta classe que são instanciados projéteis, torres e inimigos.

A classe Torre possui atributos como alcance, poder. O alcance representa o raio a partir do seu centro que limita a distância dos seus ataques. O poder é um valor absoluto que representa o total de vida que será subtraído a cada ataque. Possui um método para buscar inimigos dentro de seu alcance que altera o ponteiro para um dos inimigos executado em seu método *update*.

A classe Inimiga (*Foe*) representa os inimigos que devem percorrer a lista de vetores do caminho no mapa. Possui atributos como quantidade de vida, velocidade. Sua trajetória é atualizada no seu próprio método *update* executado automaticamente pelo motor. Quando a quantidade de vida chega a zero são destruídos os inimigos e o seu preço será adicionado ao dinheiro do jogador. Se chegarem ao final do caminho no mapa o jogador perde uma vida. Acabando todas as vidas o jogo será encerrado.

A classe Projétil (*Bullet*) representa o disparo de uma torre contra um inimigo. Sua importância é apenas representativa para o jogador e tem como atributos a torre de origem e o alvo inimigo. Sua trajetória é calculada dentro do seu método *update.*

Todo o posicionamento dos elementos na tela assim como a movimentação dos inimigos e dos projéteis das torres utilizam cálculos com vetores já definidos e disponíveis no motor Orx.

A movimentação dos inimigos é realizada no método *update* da respectiva classe. Um vetor representa o destino a ser alcançado pelo objeto. O algoritmo de movimentação realiza a subtração da posição atual pelo vetor de destino, verifica se o objeto já está na localização desejada, se não, normaliza o vetor resultado e multiplica pela velocidade desejada. O vetor resultante é designado como a velocidade do objeto.

Através do uso dos arquivos de configuração próprios do motor ORX foi possível adotar uma arquitetura dirigida aos dados em que diversos inimigos com diferentes valores podem ser criados de forma rápida sem necessidade de alteração no código do jogo. Desta forma, podem ser instanciados inimigos com atributos variáveis, incluindo seus elementos gráficos de forma rápida apenas inserindo os valores no arquivo de configurações correspondente, sem a necessidade de escrever mais código.

6. Testes

A validação do projeto desenvolvido ocorreu por meio do sucesso ao instanciar o *framework* e de testes de sistema, ilustrados pela figura 4 que buscaram simular uma possível execução em um jogo completo. Foram observados os números de inimigos e torres presentes, pois afetam diretamente o número de iterações no processamento, assim como o número de quadros por segundo (*fps*). Este valor indica a fluidez com que as animações são dispostas na tela para o observador indicando possíveis atrasos indesejados quando há queda ou grande variação neste valor.

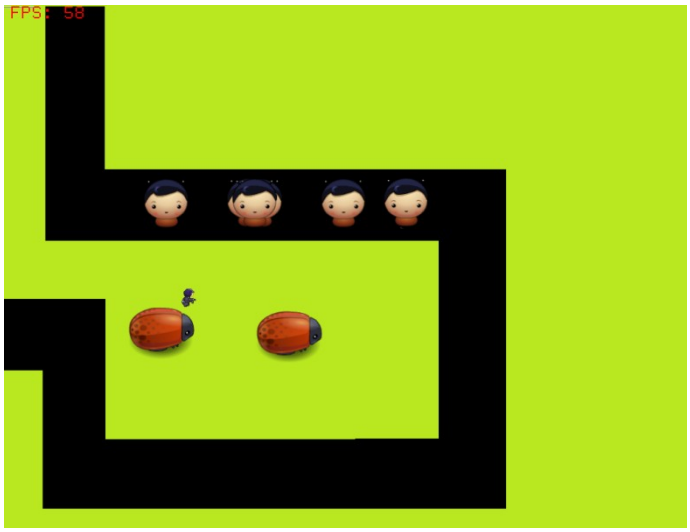

Figura 4: Exemplo de execução de teste.

A quantidade de elementos foi calculada observando os jogos existentes hoje no mercado, por exemplo, não se encontra um mapa com mil inimigos, mas algo em torno de cem inimigos e dezenas de torres tornando o teste realista e objetivo. Os valores aceitáveis para o número de quadros por segundo foi estabelecido de 30 a 60. [*TechForum*, 2013] Observando o quadro 1, a primeira coluna apresenta o número de objetos/torre instanciados; a segunda coluna, o número de objetos de inimigos instanciados e a terceira coluna apresenta o maior e o menor valores registrados durante a execução do teste.

Foram estabelecidos três casos de testes descritos no Quadro 1 que segue:

Quadro 1: Variáveis dos casos de teste e o valor de quadros por segundo resultante.

Número de torres	Número de inimigos	Fps
10	20	60 a 55
15	100	60 a 55
50	200	60 a 45

Os testes de quadros por segundo apresentaram variações significativas conforme aumenta o número de objetos. Apesar da queda no número de quadros por segundo, o fato não alterou a percepção das animações para o jogador. Contudo, será necessária a reavaliação dos resultados a cada incremento que poderá ser desenvolvido em trabalhos futuros, tendo em vista a restrição de funcionalidades apresentadas na versão atual.

7. Conclusão

Através de uma abordagem direcionada ao estilo de um jogo foi possível construir um conjunto de classes e funções características que podem facilitar o desenvolvimento de um protótipo, *demo*, ou até mesmo de um jogo completo.

Os testes tiveram resultados satisfatórios mostrando que foi possível instanciar o *framework.* Além disso, os algoritmos utilizados e a estrutura com que as classes são gerenciadas não resultaram em baixo rendimento dos quadros por segundo durante a execução. Com até 200 inimigos simultâneos e 50 torres não foi observado atraso no processamento gerando uma simulação, embora haja maior variação de quadros por segundo conforme se eleva o número de objetos no jogo. Devido ao baixo número de funcionalidades apresentadas na versão atual se comparados com jogos comerciais, torna-se necessária a ampliação das classes e métodos a fim de tornar o trabalho atrativo para o uso comercial. Portanto, trabalhos futuros poderão ser elaborados visando melhorias na versão atual com o desenvolvimento de menus, controle de entrada de dados e funcionalidades secundárias como modelos predefinidos de configurações com valores já testados que ofereçam um jogo de alto nível, além da aplicação de sons e animações facilitada. Também poderá ser feito estudos a procura de melhorias obtidas no processo desenvolvimento de jogos *towerdefense* utilizando este framework.

8. Referências

Acigames – As desenvolvedoras de games do Brasil. Disponível em: <http://www.acigames.com.br/wp-content/uploads/2012/05/Pesquisa-acigames-desenvolvedoras-de-games-2012.pdf>. Acesso em: 15 abr. 2013.

Acigames – Município do Rio de Janeiro será o primeiro a ter incentivos de carga tributária para a produção de *games*. Disponível em: <http://www.acigames.com.br/2013/05/municpio-do-rio-de-janeiro-ser-o-primeiro-a-ter-incentivos-de-carga-tributria-para-produo-de-games/>. Acessoem: 16 abr. 2013.

Dirk Riehle. ***Framework Design: A Role Modeling Approach*.** Ph.D. Thesis, No. 13509. Zürich,Switzerland, ETH Zürich, 2000.Disponível em: <http://www.riehle.org/computer-science/research/dissertation/diss-a4.pdf>. Acessoem: 01 jun. 2013.

Gregory Jason (2009). ***Game Engine Architecture***, A. K. Peters, Ltd., Natick, MA, USA.

Ministério da Cultura, *BRGames*. Disponível em: <http://www2.cultura.gov.br/site/2009/07/07/brgames-4/>. Acessoem 30 jan. 2013.

Open Source Initiative – The zlib/libpng License. Disponível em <http://opensource.org/licenses/Zlib>. Acesso em: 15 abr. 2013.

OpenGL– FAQ. Disponível em: <http://www.opengl.org/wiki/FAQ>. Acesso em: 15 abr. 2013.

Orx*Project*, *about*. Disponível em: <http://orx-project.org/about>. Acesso em: 10 abr. 2013.

Andrade Pedroso, Crislaine, et al. "**PAPEL DO BRINQUEDO NO DESENVOLVIMENTO INFANTIL.**" Disponível em: <http://scelisul.com.br/cursos/graduacao/PD/artigo2.pdf>. Acesso em: 15 dez. 2012.

Koster, Raph. (2004). ***A TheoryofFun for Game Design***. Phoenix, AZ: Paraglyph.

SDL – *SimpleDirectMediaLayer*.Disponível em: <http://www.libsdl.org>. Acesso em: 20 dez. 2012.

Tech SupportForum. Disponível em: <http://www.techsupportforum.com/4249-frames-rates-and-their-impact-on-different-game-genres/>. Acesso em: 26 jun. 2013.

Wetzel, B. **Temporal *Coverage in Tower Defense Games****: An Investigation of Strategy Representation.*Disponível em: <http://www-users.cselabs.umn.edu/classes/Spring-2011/csci4511/lectures/other/papers/wetzel%20FDG%20DC%20extended%20abstract.pdf>. Acesso em: 15 dez. 2012.