Universidade de São Paulo
Instituto de Matemática e Estatística

Trabalho de Formatura Supervisionado

aMaze Unknown

Gustavo Pinto Vilela
Itai de Ávila Soares
Luiz Ricardo Romagnoli

Orientador
Flávio Soares Corrêa da Silva

Dezembro de 2010

Sumário

1 Introdução

1.1 Motivação

A motivação inicial do projeto surgiu da oportunidade de dar continuidade a um jogo desenvolvido na disciplina Laboratório de Programação I. Essa proposta acabou não se concretizando, mas a idéia de desenvolver um jogo continuou atrativa e aparecia como uma boa oportunidade de explorar diversos conceitos computacionais de forma divertida.

Desenvolver um jogo dá a possibilidade de criar um projeto grande e ao mesmo tempo estudar e aplicar áreas da computação bastante distintas e sem a necessidade de estarem diretamente relacionadas. A oportunidade de conhecer e trabalhar independentemente com uma grande variedade de conceitos, sem precisar escolher uma única área para focar e aprofundar os estudos, foi determinante na escolha do projeto.

O desenvolvimento de um projeto grande é uma possibilidade de aproximar o projeto de uma situação real, onde é necessário trabalhar com uma linguagem, frameworks e plataformas que muitas vezes não são familiares.

1.2 Objetivos

Com a idéia definida, o principal objetivo foi desenvolver um jogo multiplayer, em rede, multiplataforma e preparado para ser estendido.

Esse último requisito é bastante abrangente e refere-se a criar uma estrutura que torne simples e fácil estender as funcionalidades iniciais. A idéia é que o jogo esteja preparado para mudanças e que o desenvolvedor interessado não necessite de muitos esforços para fazer alterações nas funcionalidades existentes ou adicionar novas com o objetivo de expandir o jogo.

1.3 Contexto

O desenvolvimento de jogos é um comum atrativo para muitos desenvolvedores que buscam uma oportunidade de colocar em prática suas idéias e criatividade.

Atualmente, a ascensão da área de jogos tem atraído muitos investimentos em diversas áreas, acarretando um grande crescimento, o surgimento de

diversos avanços tecnológicos e um crescente interesse e atenção de muitos especialistas.

Ao mesmo tempo em que é necessária muita criatividade e capacidade para criar um jogo envolvente, os jogos permitem estudar e aplicar diversas áreas do conhecimento, de matemática a design.

2 O Jogo

2.1 Estrutura

O jogo é segmentado em dois níveis: um jogo principal e diversos mini-games.

O jogo principal tem como cenário um labirinto no qual os jogadores se descolam e interagem livremente com o objetivo de ser o primeiro a conseguir escapar. Nesse trajeto, o jogador será submetido a diversas provas através de mini-games.

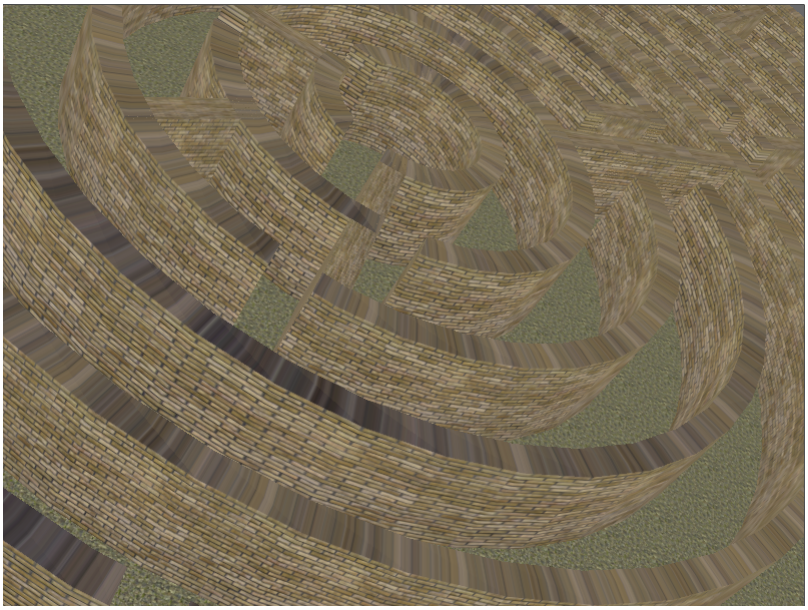

Figura 1: Cenário do jogo principal

Nos mini-games os jogadores têm a possibilidade de enfrentar ou cooperar buscando ganhar vantagem em relação ao restante dos adversários.

2.2 Conceitos Envolvidos

A criação do jogo envolve diversos conceitos que demandam algum estudo. A seguir, uma introdução de alguns desses conceitos, que serão explicados posteriormente.

Os modelos tridimensionais devem ser criados com cuidado para representar bem o que se deseja, tanto para os cenários quanto para os personagens. Não somente os modelos, mas as animações também necessitam de uma atenção especial no desenvolvimento, para que os modelos sejam leves, simples e realistas, sem prejudicar a qualidade da renderização.

No decorrer do jogo os jogadores trocam informações de posicionamento e mensagens de avisos (de início de um mini-game, por exemplo) através da rede. Essa comunicação é baseada em uma arquitetura cliente/servidor e foi implementada através de troca de mensagens via socket. Um dos jogadores, o que criou a sala, faz o papel de um servidor e aguarda pela comunicação dos outros jogadores, que serão os clientes. Quando as mensagens são enviadas, o servidor distribui a informação para todos os jogadores conectados.

A sensação de uma representação realista torna o jogo mais atrativo e por isso os fenômenos físicos foram cuidadosamente modelados para simular um comportamento bastante próximo do real.

Os jogadores não-humanos interagem com os jogadores humanos nos minigames. Com a idéia de fazer com que eles tenham um comportamento interessante, técnicas de Inteligência Artificial foram aplicadas para a tomada de decisão.

O requisito de extensibilidade do jogo é facilitado com a aplicação de padrões de design orientados a objetos, que tornam mais clara e simples a estrutura global das classes do jogo.

3 Mini-Games

Os mini-games são uma oportunidade efetiva dos jogadores se enfrentarem/colaborarem e eventualmente ganharem alguma vantagem. Cada minigame pode ser classificado em quatro classes: individuais, grupos, coletivos ou complôs.

No primeiro estilo, cada jogador escolhe suas decisões independentemente e tem o objetivo de vencer os demais jogadores; no segundo, os jogadores são separados em dois grupos e precisam se ajudar nas escolhas para vencer o grupo adversário; no estilo coletivo a principal característica é a cooperação: os jogadores devem todos colaborar para atingir um objetivo comum; e por

fim o estilo complô, quando um jogador enfrenta os outros, tentando impedi-los de atingirem suas metas.

3.1 Um Exemplo de Mini-Game

A versão atual do projeto apresenta um mini-game coletivo de exemplo.

O cenário do mini-game é uma plataforma móvel com dois níveis. No nível superior, a plataforma é transparente e permite que o outro nível seja visto. Já no nível inferior a plataforma é opaca e em forma de labirinto.

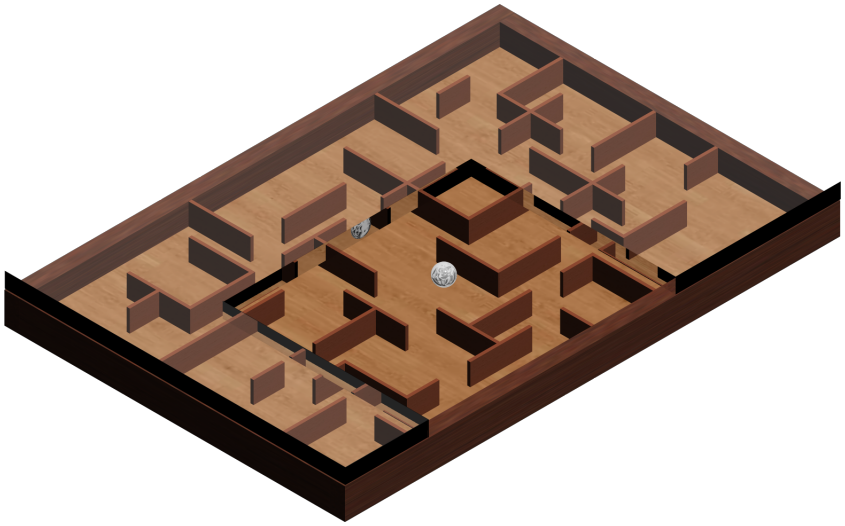

Figura 2: Cenário do Mini-Game

Inicialmente, existe uma bolinha em algum ponto deste labirinto e um buraco em algum outro ponto. O objetivo é que os jogadores trabalhem em conjunto para levar a bolinha até o buraco no menor tempo possível. Os jogadores ficam no nível superior, onde se deslocam livremente, inclinando a plataforma de acordo com os respectivos pesos.

O eixo de movimento da plataforma está localizado em seu centro, por isso um jogador mais distante do centro proporciona uma maior inclinação da plataforma.

4 Planejamento do Projeto

Antes de iniciar o desenvolvimento, foi elaborado um projeto utilizando técnicas de engenharia de software para guiar o andamento do projeto. A idéia não é ficar preso ao planejamento e às etapas estabelecidas, mas apenas tê-las como base para levar o projeto e ter uma forma de acompanhamento do que está sendo produzido.

O planejamento faz o papel de detalhar o que deve ser feito e se necessário estabelecer ordens e prioridades sobre as atividades. Portanto, não se trata de um cronograma.

O projeto aplicou o modelo incremental, que combina elementos do modelo em cascata sendo aplicados de forma iterativa, aproveitando o conhecimento adquirido com o processo e com o desenvolvimento e buscando sempre a melhoria contínua.

Ao final das etapas do processo, tudo que foi desenvolvido deve estar validado e uma última fase do projeto é necessária para cuidar dos últimos detalhes e finalizar os documentos necessários para a entrega definitiva.

4.1 Levantamento de Requisitos

Inicialmente o projeto passou por uma etapa de levantamento de requisitos. Esse período foi responsável por decidir o que seria necessário para a seqüência do desenvolvimento.

Todos os requisitos para o desenvolvimento do jogo devem aparecer nessa fase com base em um estudo sobre o escopo do projeto. Nesta etapa são feitas escolhas como a plataforma que servirá de base para o jogo, a linguagem utilizada, o ambiente, ferramentas necessárias e os principais conceitos que devem ser estudados.

4.2 Desenvolvimento Incremental

O desenvolvimento do projeto foi segmentado em diversas pequenas iterações. A idéia é que cada uma delas seja curta, bem definida e com resultados rápidos. Cada iteração consiste em: estudo de conceitos necessários, implementação de funcionalidades específicas, testes funcionais e integração com o restante do jogo.

4.2.1 Iterações

As iterações do projeto foram segmentadas da seguinte forma:

- Estudo de domínio;
- Início do desenvolvimento do jogo base (inclui modelagem gráfica);
- Aprimoramento do jogo base;
- Início da comunicação entre os jogadores;
- Implementação final da rede e tratamento de colisões;
- Início do desenvolvimento do mini-game de exemplo;
- Término do primeiro mini-game;
- Criação dos jogadores não-humanos;
- Últimos retoques no jogo (criação de menus e telas, por exemplo).

4.3 Testes

Em paralelo com o desenvolvimento das funcionalidades, os testes de corretude são naturais e feitos rotineiramente para validar a solução adotada. Após o desenvolvimento estar finalizado, alguns testes mais elaborados, de casos mais específicos ou problemáticos, são necessários.

Ao final de cada iteração, as novas funcionalidade devem ser submetidas aos testes de aceitação e após serem integradas ao restante do projeto devem passar por testes de integração para garantir que tudo continue funcionando corretamente.

5 O Panda3D

A plataforma escolhida para rodar o jogo foi o Panda 3D. "Panda3D é uma engine 3D: uma biblioteca de sub-rotinas para renderização 3D e desenvolvimento de jogos. A biblioteca é escrita em C++ com um conjunto de associações em Python. O desenvolvimento de jogos com o Panda3D geralmente consiste em escrever um programa em Python ou C++ que manipula a biblioteca Panda3D" [1].

A engine é muito poderosa e exatamente por isso a manipulação requer muito cuidado, conhecimento e habilidade. O Panda inclui gráficos, áudio, entrada e saída, detecção de colisão e diversos outros recursos relevantes para a criação de jogos, desde uma simples árvore de renderização até sub-rotinas que facilitam a comunicação em rede.

O Panda possui ainda a vantagem de ser multiplataforma. A engine não apenas funciona sobre diversas plataformas, como garante ao desenvolvedor que seu jogo funcionará em todas as plataformas sem que precise se preocupar com detalhes específicos de cada uma delas.

Desenvolvida pela Disney, atualmente a engine é mantida em conjunto com a universidade Carnegie Mellon.

O Panda apresenta a mesma dificuldade de se trabalhar com qualquer outro framework: a necessidade de acrescentar uma funcionalidade não disponível ou minimamente diferente da existente não é trivial.

5.1 ShowBase

O coração de uma aplicação desenvolvida com Panda3D está na classe ShowBase. Ela é a responsável por diversos objetos essenciais para a aplicação e implementa o método *run()*, que abre uma janela e começa a execução do programa. Portanto, é necessário que pelo menos uma classe entenda ShowBase, que será o ponto de partida da aplicação.

A classe ShowBase define, entre outros, os seguintes objetos: base, render, camera, loader, taskMgr, messenger, render2d, aspect2d e hidden. Esses objetos são responsáveis por organizar os recursos que servem de base para o

[1] Tradução livre do site oficial do Panda3D. Texto original disponível em: http://www.panda3d.org/manual/index.php/Introduction_to_Panda3D.

jogo, por exemplo, a câmera, as tarefas, as mensagens, o que será renderizado e formas simples de acesso a métodos importantes, como o de carregar um modelo.

5.2 Grafo de Cena

O Panda3D mantém uma relação, em forma árvore, dos objetos que devem ser renderizados. Um objeto será renderizado se e somente se pertencer a essa árvore. Essa árvore recebe o nome de Scene Graph, ou Grafo de Cena, e sua raiz é o objeto render.

Por definição, essa relação é hierárquica. Portanto, os objetos que estiverem abaixo de um objeto X serão renderizados "em relação à" X. Isso quer dizer que algumas propriedades desses objetos, como posição, orientação e tamanho, serão calculadas sempre em relação às propriedades de X. Por exemplo, o centro do eixo de coordenadas da posição desses objetos passa a ser a posição de X.

5.3 NodePath

A classe NodePath é a unidade fundamental de interação com o grafo de cena. Todos os objetos passíveis de serem renderizados são do tipo NodePath. Um objeto é inserido no grafo de cena com o método *reparentTo(outroObjeto)*. Para que um objeto X seja independente de outros objetos, deve-se inserí-lo na árvore exatamente abaixo da raiz, ou seja, fazer *X.reparetnTo(render)*; para que X seja renderizado em relação ao objeto Y deve-se fazer *X.reparentTo(Y)*.

5.4 Tasks

As Tasks são funções especiais que são executadas a todo frame, uma após a outra, cooperativamente. Um novo frame é iniciado somente quando todas as tasks terminarem suas execuções. Uma task é definida como uma função ou um método.

```
def taskExemplo (task) :
        #Faz algo
        return task.cont
```

Código 5.1: Tasks.

5.5 Task Manager

As tasks são atribuídas ao objeto global taskMgr, que é responsável por gerenciá-las e controlar suas execuções. Para que uma nova task seja iniciada, é necessário atribuí-la ao taskMgr pelo comando *taskMgr.add(taskExemplo, "nomeTask")*.

Assim, essa task começará sua execução a partir do próximo frame. O Task Manager mantém o controle de execução de todas as tasks, podendo continuar a executá-las, removê-las da lista de tasks ativas ou fazer algo quando uma task termina sua execução.

É possível, ainda, visualizar as tasks registradas através de uma interface gráfica através do comando *taskMgr.popupControls()*. Essa opção é útil na fase de desenvolvimento, especialmente para debug.

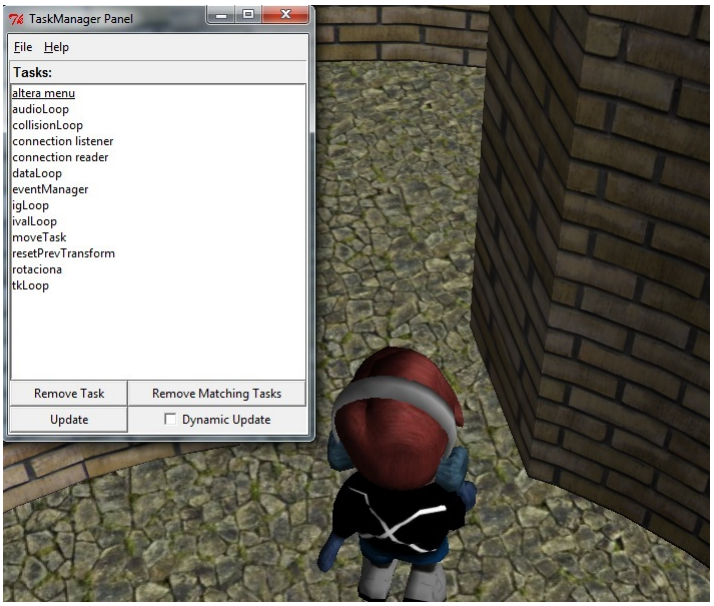

Figura 3: Interface gráfica com as tasks registradas.

5.6 Colisões

Uma parte importante de um jogo é o tratamento de colisões. O panda3D possui um mecanismo simples e eficiente para detectar e notificar as ocorrências de uma colisão. Mas antes disso, uma definição:

Uma colisão acontece quando um objeto, se movimentando no espaço, tenta ocupar um espaço já ocupado por outro objeto.

O programador é o responsável por definir os objetos passíveis de gerar uma colisão, os objetos que podem ser colididos e tratar as colisões detectadas no espaço.

5.6.1 Participantes

Primeiramente, vamos a duas definições:

Um objeto From é o objeto ativo da colisão, ou seja, aquele que colidiu; um objeto Into é o objeto passivo da colisão, ou seja, aquele que foi colidido.

Essa separação é importante pois o sistema mantém uma lista de todos os objetos *From* no espaço e checa, a todo frame, se eles estão colidindo com outros objetos. Um objeto estático, geralmente, é classificado como um objeto *Into*, enquanto um objeto móvel, um objeto *From*.

5.6.2 Sólidos de Colisão

O objeto central no sistema de colisões é o CollisionSolid, ou sólido de colisão. Para que um objeto no espaço seja detectado pelo sistema de colisões é necessário que ele possua um CollisionSolid associado.

Dentre os vários tipos de sólidos de colisão, está o ColisionSphere; é o sólido mais robusto e otimizado para ser utilizado em uma colisão. Além disso, é o único sólido que é capaz de gerar e de receber uma colisão com todos os outros tipos de sólidos. Especificamente, uma parte do objeto é colidível se está no interior da esfera de colisão.

O seguinte fragmento de código ilustra a criação de um sólido de colisão e sua associação a um objeto:

```
1 sphere = CollisionSphere(0, 0, 0, 1)
2 collisionNodePath = objeto.attachNewNode(CollisionNode("esferaColisao"))
3 collisionNodePath.node().addSolid(sphere)
```

Código 5.2: Sólido de colisão.

A linha 1 define uma esfera de colisão de centro (0, 0, 0) e raio 1. A linha 2 associa um CollisionNode a um objeto, que permitirá que se defina um ou mais sólidos de colisão para aquele objeto. Na linha 3, a esfera criada é associada ao objeto. Nesse momento, a posição do centro da esfera passa a ser relativo ao centro do objeto.

Outros tipos de sólidos de colisão disponíveis: CollisionTube, CollisionInvSphere, CollisionPlane, CollisionPolygon, CollisionRay, CollisionLine, CollisonSegment e CollisonParabola [2].

Existe outro tipo de definição de objetos colidíveis, que é feita no arquivo do modelo.

5.6.3 Tratando as Colisões

Para cada colisão detectada, um objeto do tipo CollisionEntry é criado. Esse objeto carrega informações como os objetos *Into* e *From*, o ponto de colisão e o vetor normal da colisão. Essas entradas de colisão são tratadas por um tratador de colisão, um CollisionHandler. Esse objeto captura as entradas geradas e define o modo como elas serão tratadas. O tratador mais simples é o CollisionHandlerQueue, que consiste de uma fila onde as entradas são armazenadas, na ordem em que foram geradas. Um exemplo de criação e uso de uma fila:

```
fila = CollisionHandlerQueue()
...
for i in range(queue.getNumEntries()):
        entrada = queue.getEntry(i)
        # Faz alguma coisa
```

Código 5.3: Tratador colisões.

Existem vários tipos de tratadores de colisão, a saber: CollisionHandlerEvent, CollisionHandlerPusher, PhysicsCollisionHandler, CollisionHandlerFloor.

5.6.4 Registrando as Colisões

O objeto responsável por checar por colisões é o CollisionTraverser. Ele mantém a lista de objetos colidíveis e realiza a verificação de colisões entre os

[2]Mais informações podem ser obtidas em https://www.panda3d.org/manual/index.php/Collision_Solids

objetos. Todos os objetos no espaço, desde que possuam um CollisionSolid associado, são objetos *Into*, já que todos podem receber uma colisão. Para um objeto ser também um objeto *From*, é necessário registrá-lo no traverser. Para isso, é necessário também associar qual tratador de colisões será responsável por tratar as colisões geradas por esse objeto.

```
traverser = CollisionTraverser("nome no traverser")
traverser.addCollider(objetoFrom, handler)
```

Código 5.4: Registro de colisões.

A partir de agora, o traverser checará, a todo frame, se o objeto objetoFrom está colidindo com alguém no espaço.

5.7 Interval

Existe um mecanismo de transição suave de valores chamado Interval. Ele consiste de alterar o valor de uma variável continuamente, sem gerar mudanças bruscas, durante um determinado intervalo de tempo. Ele possui a seguinte assinatura simplificada: *LerpInterval(duração, valorFinal, valorInicial=None)*, onde Lerp é abreviação para linearly interpolate, ou interpolação linear. Esse mecanismo possui várias versões, sendo que a mais utilizada é LerpPosInterval, que varia a posição de um objeto.

Ao iniciá-lo, o objeto irá de *posicãoInicial* até *posiçãoFinal* em *duração* segundos. Porém, esse mecanismo possui algumas limitações. A principal delas é que, iniciado o intervalo, o objeto sempre terminará na posiçãoFinal. Mais especificamente, se esse intervalo for interrompido antes do término, o objeto será imediatamente deslocado para posiçãoFinal. Isso é um problema, pois se no caminho entre um ponto

e outro ocorre uma colisão, o objeto não mais se movimentará devido ao objeto colidido e ficará inativo durante todo o tempo restante do intervalo. Isso acontece com todas as versões do LerpInterval.

Para evitar esse problema foi necessário escrever um mecanismo semelhante ao LerpInterval mas mais flexível.

A variação continua de um valor indo do valor A para o valor B é dada pela expressão

$$A + t(B - A), 0 \leq t \leq 1$$

Para que essa variação seja suave, é necessário variar continuamente o valor de t. Isso é feito seguindo a seguinte relação genérica:

$$t = \frac{(tempoAtual - tempoInicial)}{(tempoFinal - tempoInicial)}$$

onde *tempoFinal = tempoinicial + numeroSegundos*. Ou seja, seguindo essa relação, é possível variar o valor de t, indo de 0 a 1 durante *numeroSegundos* segundos. Mas ainda é necessário obter constantemente valores para tempoAtual e calcular os valores de t sem bloquear a Thread principal. Para isso, esse cálculo é feito em uma Task, que é executada enquanto *t != 1* (ou *tempoAtual < tempoFinal*) e que, a cada execução, atualiza o valor de *tempoInicial*. Agora basta, a cada frame, atualizar o valor da variável, fazendo variável = *A + t(B - A)*.

Como a variação é feita manualmente, é possível interrompe-la a qualquer momento sem que o objeto fique travado ou que seu valor se altere bruscamente, bastando finalizar a execução da task.

No projeto, esse novo mecanismo foi utilizado no clareamento da tela quando um mini-game é encontrado. Essa mudança é feita inserindo uma neblina na tela e aumentando sua densidade, cujo valor varia entre 0 e 1. *SetUp.fog.setExpDensity(t)*, $0 \leq t \leq 1$.

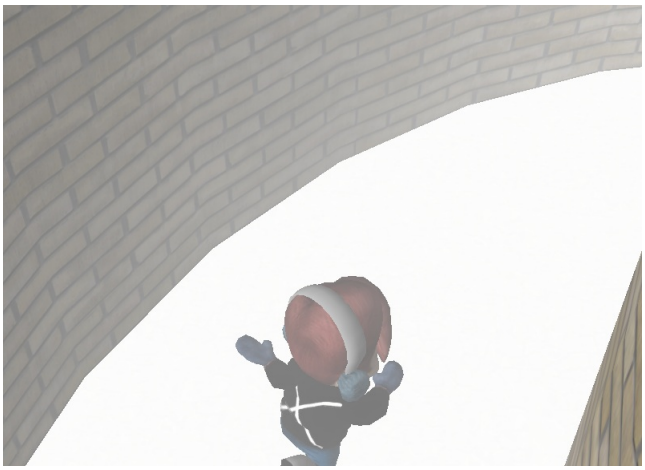

Figura 4: Efeito de transição para um mini-game.

5.8 O Formato *egg*

O Panda utiliza arquivos com extensão *egg* para carregar objetos e modelos gráficos que serão renderizados. Esse tipo de arquivo consiste de uma série de comandos seqüenciais e/ou hierárquicos que obedecem à seguinte forma genérica:

```
<comando> nome { conteúdo }
```

Código 5.5: Formato dos comandos do *egg*.

O arquivo também define as texturas aplicadas ao objeto:

```
<Texture> nome {
        nomeArquivo
        [opcoes]
}
```

Código 5.6: Textura no *egg*.

e as aplica em seu seus polígonos. Um polígono é definido por:

```
<Polygon> {
        <RGBA> { 1 1 1 1 }
        <TRef> { Textura1 }
        <VertexRef> { 1 2 3 <Ref> { vertices.verts } }
}
```

Código 5.7: Polígono no *egg*.

e o comando VertexRef faz referência a vértices definidos anteriormente:

```
<Vertex> 1 {
        x y z
        <UV> { u v }
        <Normal> { a b c }
}
```

Código 5.8: Vértice no *egg*.

Um modelo em *egg* pode ser feito manualmente ou com a ajuda de um software de modelagem gráfica compatível. A lista de softwares compatíveis é extensa mas os softwares recomendados, tanto por qualidade como por facilidade de exportar arquivos *egg* (via plugins disponibilizados pelo Panda), são Maya, 3D Studio Max e Blender.

6 O 3D Studio Max

No desenvolvimento de jogos a qualidade gráfica é um dos diferenciais. Desenvolver um jogo com código organizado, extensível, multiplataforma e não se preocupar com a parte gráfica certamente não será uma boa escolha quando for apresentado aos usuários finais.

A modelagem gráfica necessária ao jogo exige uma ferramenta com suporte para criação de modelos tridimensionais. Foi necessária uma pesquisa pra se escolher uma boa ferramenta, levando em consideração uma boa curva de aprendizagem, comunidade ativa, boa documentação, acessibilidade em questão de preço e compatibilidade com os modelos que o Panda utiliza.

Existem diversas opções poderosas nessas condições, porém a utilização não é trivial. Por combinar relativa facilidade de uso e recursos muito poderosos optou-se por utilizar o 3D Studio Max.

A ferramenta atende a todos os requisitos e apresenta diversas facilidades como suporte a texturas e animações. Além de possuir todos os recursos de outras ferramentas de modelagem gráfica poderosas (como o Blender, por exemplo), o 3D Studio Max ainda é compatível com a extensão *egg*, que é utilizada pelo Panda3D.

O único empecilho do 3D Max é o valor da licença do software. Por este motivo a solução foi adotar a versão 3D Max Trial.

6.1 Criação dos Modelos

Visando um aplicativo eficiente, a criação do ambiente foi projetada para aproveitar o máximo de desempenho, principalmente com relação às colisões, com o mínimo de perda de qualidade visual. A quantidade de vértices nos modelos é a mínima aceitável e o mapeamento de texturas foi feito com o mínimo de repetição.

Um dos problemas enfrentados na criação dos modelos é a definição da relação entre quantidade de vértices e qualidade visual. Um modelo com uma grande quantidade de faces e vértices é pesado para renderizar, enquanto um modelo com poucas faces e vértices não apresenta um visual aceitável.

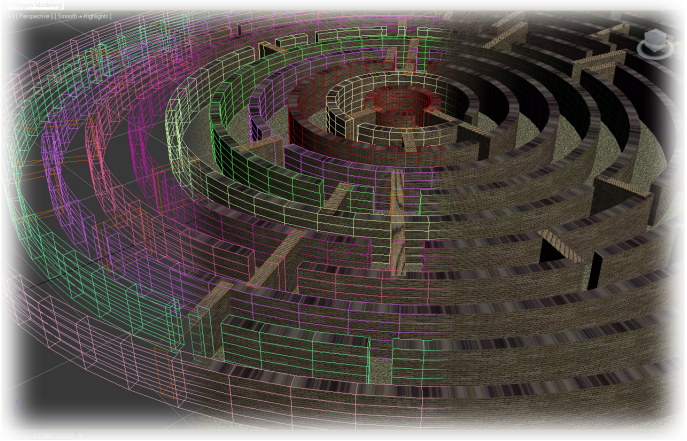

Figura 5: Modelo do jogo principal no 3Ds Max.

A solução para lidar com esse problema foi encontrar uma ponderação entre os dois fatores que combinasse satisfatoriamente a qualidade visual do modelo final e a complexidade envolvida na renderização do mesmo.

Outra otimização realizada, na busca por um melhor desempenho, foi a adaptação dos modelos para versões mais leves e simples. Essa adaptação inclui a redução do tamanho do arquivo do modelo, através da remoção de vértices não necessários (que não são exibidos na renderização, por exemplo, vértices internos a um polígono) e a unificação de objetos em que a forma de colisão teria o mesmo tratamento.

A primeira alteração permitiu a utilização de modelos mais leves e fez com que o carregamento e a renderização ficassem mais eficientes. A modificação para as colisões facilitou o tratamento. Anteriormente, era necessário verificar se existia colisão com todos os objetos que compunham o modelo (e muitas vezes o tratamento para essas colisões era o mesmo); após a alteração, esses objetos formavam um único componente, simplificando a verificação e os tratamentos das colisões.

7 Atividades Realizadas

No decorrer do desenvolvimento do projeto, diversos estudos em diversas áreas foram necessários. Alguns deles merecem destaque por tratarem de assuntos importantes e conceitos relevantes dentro do projeto.

Os primeiros passos do projeto foram baseados em estudos sobre os recursos do Panda e a modelagem gráfica, que já foram citados anteriormente. O passo seguinte foi a escolha da linguagem Python para implementar o projeto, por ser uma linguagem simples, dinâmica e com melhor documentação (em relação a C++, a outra possibilidade oferecida pelo Panda).

Os principais resultados desses estudos foram brevemente detalhados e alguns outros estudos importantes que devem ser lembrados são listados e explicados a seguir.

7.1 Modelagem Física

O problema da modelagem física está relacionado com a tentativa de aproximar o comportamento dos fenômenos físicos presentes no jogo do comportamento real. Para isso, a modelagem requer muito cuidado na aplicação de conceitos de física.

7.1.1 Movimento da Plataforma

No mini-game, o movimento da plataforma é definido de acordo com a posição de cada um dos jogadores. O cálculo considera a força peso atuando sobre cada jogador e calcula a força resultante para determinar a inclinação da plataforma.

A força resultante é calculada sobre o eixo de movimento da plataforma, que está localizado em seu centro. Por isso, a contribuição do peso de um jogador mais distante do centro é maior do que a de um jogador próximo.

Formalmente, dados n jogadores, seja P_i a posição o i-ésimo jogador em relação ao centro da plataforma. Considerando que todos os jogadores tem o mesmo peso, a força resultante F é o vetor cuja direção é dada por:

$$F = P_1 + P_2 + \cdots + P_n = \sum_{n}^{i=1} P_i$$

7.1.2 Inércia

A inércia é uma propriedade que diz que a velocidade de um corpo não é alterada se nenhuma força externa ao sistema agir sobre ele ou se a resultante das forças agindo for nula. Como conseqüência, se o corpo está parado, ele continuará parado e se estiver em movimento, continuará em movimento e sua velocidade permanecerá constante.

No caso do mini-game, a inércia está presente no movimento da bolinha sobre a plataforma. A única força externa agindo é a resultante dos pesos dos jogadores. Quando a resultante é não nula, a velocidade da bolinha aumenta de acordo com a inclinação causada na plataforma.

A ação da força sobre a bolinha parada simplesmente faz com que a bolinha passe a se mover na direção da inclinação. No caso em que a bolinha já está em movimento e a direção da inclinação não se altera, o movimento da bolinha não sofre alteração no sentido, mas altera a velocidade (aumentando caso a inclinação aumente ou reduzindo caso a inclinação diminua). Por último, no caso em que a bolinha está em movimento e a direção da inclinação é alterada, a bolinha vai aos poucos perdendo velocidade no sentido do movimento antigo e ganhando velocidade no sentido da nova inclinação, fazendo com que o movimento aparente se aproxime cada vez mais da inclinação atual da plataforma.

7.1.3 Colisões Inelásticas

A colisão inelástica ocorre quando a energia cinética não se conserva após a colisão. No caso do mini-game, pode-se observar esta propriedade na colisão entre a bolinha e as paredes do labirinto.

Quando a bolinha colide com a parede, a velocidade após a colisão é inferior à velocidade inicial. Nessa situação, o momento do sistema se conserva, mas a alteração na velocidade da bolinha indica uma variação na energia cinética do sistema. Parte dessa energia cinética é convertida em outros tipos de energia, como interna e térmica, que não foram aproveitadas.

7.1.4 Rotação da Bolinha [3]

A rotação da bolinha em 3D é a modelagem física mais delicada do projeto. A modelagem da rotação necessita das especificações da posição e da orientação da bolinha. Especificar a posição é relativamente simples, utilizando

coordenadas cartesianas ou translações em relação a uma origem conhecida. Especificar a orientação, porém, não é trivial.

Inicialmente, a orientação poderia ser dada como rotações em relação a uma origem conhecida. Porém, não é assim tão simples. Por exemplo, porque as rotações tridimensionais não comutam, implicando que a ordem de aplicação das operações nos diferentes eixos afeta o resultado final. Não é impossível representar as rotações utilizando os eixos cartesianos e ângulos, porém uma solução mais simples foi adotada: aplicar *quatérnios*.

Quatérnio é uma ramificação da matemática que generaliza o cálculo vetorial e os números complexos. Um quatérnio de rotação é determinado pelo cosseno do ângulo de rotação (parte real) e o eixo de rotação (parte imaginária).

$$q = (\cos\theta, (e_x, e_y, e_z))$$

A representação de um ponto $\vec{r}$ sobre o qual será aplicada uma rotação será um quatérnio com parte real nula, $p = (0, \vec{r})$. A rotação será representada por um quatérnio unitário $q = (s, \vec{v})$. O resultado da rotação de p por q é dado por:

$$\underline{R_q(p) = qp}$$
$$q$$

Outra vantagem da representação utilizando quatérnios é que a própria álgebra dos quatérnios realiza naturalmente a composição de rotações.

O Panda possui recursos que implementam a interpolação de orientações. Assim, dadas duas orientações representadas por quatérnios, o Panda determina os quatérnios que representam a seqüência intermediária de orientações.

7.2 Desenvolvimento da Rede

O problema do desenvolvimento da rede é um dos requisitos propostos pelo projeto. A comunicação entre os jogadores deve ser precisa, eficaz e eficiente, garantindo que toda a informação necessária é trocada e que o tráfego das mensagens não seja um problema no desempenho. A precisão das informações é importante para garantir a consistência do jogo.

A estrutura escolhida para a rede é baseada na arquitetura cliente/servidor. Um jogador, o criador da sala, faz o papel de servidor e fica aguardando pela comunicação dos outros jogadores, que fazem o papel dos clientes.

A escolha por utilizar uma arquitetura que centraliza o controle e a distribuição das mensagens exige uma atenção a mais para que não se crie um gargalo de informações no servidor. Toda a comunicação dos jogadores é baseada na troca de mensagens via socket.

Os jogadores não trocam as informações diretamente. Todas as mensagens são enviadas ao servidor, que tem o conhecimento de todos os clientes conectados e sabe resolver o conteúdo da mensagem e a encaminha para o destinatário correto.

As mensagens podem ser separadas em dois tipos principais: mensagens de aviso e de posicionamento. O primeiro tipo de mensagem tem função unicamente informativa e tem aplicação, por exemplo, no momento em que um novo cliente se conecta, um novo mini-game começa ou um mini-game termina. As informações de posicionamento servem para que todos os jogadores saibam onde estão os seus adversários.

O Panda possui algumas classes e funcionalidade que simplificam a criação da rede. Dentre as implementações existentes, estão classes que auxiliam tanto no momento de estabelecer conexões (na arquitetura cliente/servidor) como para transmitir informações. As funcionalidades implementadas suportam o uso dos protocolos TCP e UDP.

7.2.1 Formato das Mensagens

A decisão sobre quais informações devem fazer parte da mensagem é outro ponto interessante. É importante manter o tamanho da mensagem pequeno para que o tráfego na rede não seja grande. Por outro lado é necessário que toda a informação relevante esteja de alguma forma na mensagem.

O tamanho da mensagem é importante também para reduzir o tempo necessário para processá-la (extrair o conteúdo e iniciar as ações necessárias) do lado de quem a recebe. Do lado de quem envia, é importante para reduzir o tempo necessário para criar o datagrama e enviá-lo pela rede.

O formato final da mensagem contém toda e somente a informação necessária. A mensagem está organizada com a seguinte estrutura:

- Um inteiro que faz o papel de um identificador para o tipo da mensagem;
- Um inteiro que representa o id do jogador que está enviando a mensagem;
- Um inteiro que representa o id do jogador deve receber a mensagem (ou o valor -1 para indicar que é uma mensagem que deve ser enviada a todos os jogadores);
- O conteúdo da mensagem.

A estrutura inicial da mensagem é bem fixa, enquanto a parte do conteúdo pode variar bastante. No caso de mensagens informativas, apenas o identificador da mensagem é necessário, especificando qual ação tomar. No caso de mensagens de posicionamento, o conteúdo da mensagem contém as teclas que vão sendo pressionadas pelo jogador, de forma que a posição do personagem no jogo possa ser calculada.

A escolha por enviar as teclas que estão sendo pressionadas pelo jogador ao invés da posição real do personagem ocorreu em razão de uma dificuldade de tratar colisões. Diante dessa situação, a escolha por calcular a posição de cada personagem a partir das teclas pressionadas pelo jogador parece interessante, alem de permitir facilmente o controle sobre as colisões nesse percurso.

7.2.2 Tráfego de Mensagens

O tráfego das mensagens na rede é um ponto que merece uma atenção especial. A grande quantidade de informações trafegando na rede pode criar um gargalo no servidor, fazendo com que a distribuição das mensagens comece a atrasar e conseqüentemente pequenos travamentos começam a ocorrer no jogo. Outro problema que pode ser conseqüência desse atraso é a imprecisão das informações recebidas; o maior tempo entre o envio e recebimento da mensagem faz com que nem sempre o que está sendo exibido na tela do jogador reflita o estado atual do jogo.

Com o objetivo de amenizar esse problema, alguns testes sugerem a solução que deve ser adotada em cada situação específica. No projeto, a solução final encontrada foi enviar mensagens somente quando um dos jogadores pressionava ou soltava uma tecla, com mensagens de correção da sincronia em intervalos regulares.

A primeira tentativa feita foi a de enviar mensagens com a posição dos jogadores (coordenadas cartesianas e orientação) a cada frame. Claramente, essa solução não parece eficiente e na prática levou a um gargalo instantâneo do servidor. O acumulo de mensagens fez com que o tempo entre enviar o pacote e processar o pacote no destino chegasse a certa de 2 segundos.

A próxima tentativa foi a de enviar as mensagens em intervalos de tempo regulares. A dificuldade então era determinar o intervalo de tempo ideal para esse envio, tentando não criar um gargalo na rede e não prejudicar a realidade de movimento dos outros jogadores. Foram efetuados testes variando de 40 mensagens por segundo (que era aproximadamente o número de mensagens quando o envio era feito a todo frame) até 10 mensagens por segundo. Após alguns testes, a melhor solução encontrada com quatro jogadores foi trocar as mensagens a cada 0.1 segundos.

Essa parecia ser a solução ideal, mas ainda restava um pequeno atraso no movimento do personagem remoto e nessas duas soluções existe a dificuldade de controlar a animação do personagem dentro do jogo quando a posição era informada. A estrutura do Panda que é responsável por dados dois pontos, criar a animação do personagem indo de um ponto ao outro, não consegue tratar as colisões que ocorrem nesse trajeto. Assim, os personagens deixavam de colidir uns com os outros e também com o cenário.

O passo seguinte foi alterar o conteúdo da mensagem para o estado das teclas de cada personagem. Assim, o problema da animação não existia mais, já que o mecanismo de animação utilizado seria o mesmo que anima o personagem local, que é baseado nas teclas pressionadas. Porém o processamento da mensagem era mais lento e o tempo de resposta prejudicava a eficiência da comunicação.

Nesse momento surgiu a idéia de enviar somente as informações sobre as teclas pressionadas e fazendo o envio somente no momento em que uma nova tecla era apertada ou liberada. Essa solução não apresenta o problema da

animação, não cria um gargalo no servidor e o tempo de resposta é excelente, não prejudicando a precisão nem a eficiência da comunicação.

Essa solução apresenta como característica uma baixa quantidade de mensagens enviadas e tamanhos dos datagramas bem curtos, de fácil criação e rápido processamento.

O problema dessa solução é que em certas situações ela gera uma diferença mínima entre a posição real do personagem e posição remota. A propagação desse pequeno erro, porém, causa grandes efeitos em longo prazo e dá a impressão de não haver sincronização entre a posição do personagem nos clientes.

A causa dessa pequena diferença é o modo como se trata os eventos de teclas. Localmente, esse evento é processado em paralelo ao restante do jogo, inclusive a outros eventos de tecla, enquanto remotamente essa operação torna-se seqüencial. Assim, a situação onde esse erro aparece é quando o jogador pressiona e solta rapidamente uma tecla.

Localmente, essa ação de pressionar e soltar rapidamente a tecla gera dois eventos que serão processados em paralelo. A conseqüência disso é que o tempo em que a tecla pressionada será marcada como ativa e logo em seguida como inativa pode ser menor do que o tempo que leva para o jogo passar de um frame para o outro. O resultado é que a ação causada por essa rápida ação pode não existir.

Já remotamente, no momento em que a tecla é pressiona, um pacote é gerado e enviado pela rede com essa ação, em seguida, quando a tecla é liberada, outro pacote é criado e enviado com a nova ação. O cliente que recebe essa mensagem irá processar os pacotes de forma seqüencial, portanto irá gerar o resultado da primeira ação e em seguida o da segunda. A conseqüência disso é que a primeira ação foi executada, mesmo que rapidamente, gerando uma alteração na posição ou na orientação do personagem.

O fato de localmente a ação não ter gerado nenhum efeito e remotamente a ação ter existido, mesmo que minimamente, faz com que nesse momento apareça a diferença entre as posições ou orientações. Com o decorrer do jogo, a combinação desses pequenos erros leva a uma grande diferença nas posições dos personagens.

A solução mais simples para resolver esse problema é enviar pela rede, em intervalos de tempos regulares, a posição e a orientação do personagem. Esse intervalo de tempo não pode ser muito curto para não sobrecarregar a rede, mas também não pode ser muito longo para que não existam saltos entre as posições do personagem em razão da correção.

A solução final combina o envio das teclas com o envio periódico da orientação e da posição de cada personagem para sincronizá-los. O segundo tipo de mensagem é enviado em intervalos de 0.1 segundos. Apesar de essa solução aumentar o tráfego na rede, a quantidade total de mensagens não é um problema.

7.3 Jogadores não-humanos

O problema dos jogadores não-humanos está em desenvolver comportamentos inteligentes com uma baixa quantidade de processamento. O jogador não-humano deve simular o comportamento de um jogador humano; para isso ele deve saber tomar as atitudes adequadas em cada situação de forma simples.

O desenvolvimento desse jogador faz uso de técnicas de inteligência artificial para a tomada de decisão. A idéia é criar um agente inteligente que faça a melhor escolha localmente. Dessa forma o processamento final é simplificado já que somente as situações vizinhas precisam ser analisadas.

Apesar de o agente tomar a melhor decisão local, esta escolha é baseada em uma função objetivo global e tenta minimizar o valor dessa função. Dessa forma, a decisão do agente pode não ser a melhor decisão global, mas se aproxima de forma satisfatória dela.

7.3.1 Grafos e Busca em Largura

O primeiro passo para a criação do agente é levar o modelo contínuo do mapa para o mundo discreto. Nesse mapeamento, o movimento do agente é baseado em uma estrutura 4-conexa.

A representação do mapa no mini-game foi feita utilizando um grafo. Cada vértice do grafo representa uma posição no mapa real e as arestas representam uma vizinhança no mapa real e que não existe uma parede entre elas. Todas as arestas são bi-direcionais.

Inicialmente, o agente conhece o mapa e a menor distância partindo de cada um dos vértices até o buraco onde a bolinha deve ser levada. Para tal, o pré-processamento do mini-game inclui uma busca em largura para encontrar os melhores caminhos.

7.3.2 Subida de Encostas

O algoritmo escolhido para a tomada de decisão no mini-game foi o de *subida de encosta* [4], também conhecido por *subida da montanha* ou *hill climbing*. Essa é uma estratégia simples e popular de busca heurística baseada na busca em profundidade. É um algoritmo de busca local que não tem interesse no caminho percorrido, apenas na configuração final.

A idéia heurística por trás do algoritmo é que o número de passos para atingir um objetivo é inversamente proporcional ao tamanho deste passo. Assim, a decisão busca escolher o estado que leva para mais perto da solução. A implementação feita é baseada no *subida de encosta pela trilha mais íngreme*, que examina todos os vizinhos de um estado e escolhe aquele que está mais próximo da solução.

O algoritmo consiste em uma repetição que percorre o espaço de estados no sentido do valor decrescente e termina quando encontra um vale em que nenhum vizinho tem um valor mais baixo. O pseudocódigo é o seguinte:

```
SubidaDeEncosta (problema)
        estadoAtual <- estadoInicial(problema)
        enquanto true
                proxEstado <- melhorVizinho(estadoAtual)
                se valor(proxEstado) >= valor(estadoAtual)
                        então retorna estadoAtual
                estadoAtual <- proxEstado
```

Código 7.1: *Subida de Encosta.*

Algumas das vantagens desse algoritmo estão na simplicidade de processamento e na utilização de espaço de armazenamento. Por ser um algoritmo sem memória, as transições são Markovianas, nenhuma informação sobre o caminho percorrido precisa ser guardada, apenas o estado atual e o valor da função objetivo são armazenados, fazendo com que tenha um baixo consumo de memória. Em relação ao processamento, é simples porque não examina antecipadamente nenhum estado além dos vizinhos imediatos.

Esse algoritmo possui três situações em que não obtém sucesso: máximos (ou mínimos) locais, platôs e cordilheiras. Na primeira situação, dependendo do estado inicial, é possível que o algoritmo fique preso em máximos (ou mínimos) locais e não chegar a estados melhores e distantes; no caso de platôs pode-se encontrar uma área onde todos os vizinhos tem o mesmo valor e assim a escolha da melhor direção não pode ser feita localmente; por último, é uma região do espaço mais alta que os vizinhos e que não é possível de ser atravessada em um único passo.

Apesar de esses problemas serem bastante graves, no caso do mini-game não é preciso se preocupar diretamente com nenhum deles. O fato de o agente participar do jogo em conjunto com os jogadores humanos faz com que situações como essa sejam apenas momentâneas. A interação dos outros jogadores rapidamente tira a bolinha de situações que poderiam deixar o agente sem saber o que fazer.

7.4 Estrutura do Código

O problema da estrutura do código está em criar e manter um código de qualidade, bem estruturado e intuitivo. Além de ser um critério fundamental para auxiliar no requisito de extensibilidade do projeto.

A qualidade do código se relaciona com conseguir manter um código simples e bem modelado. As funcionalidades devem ter papéis bem definidos e separados, além de serem independentes umas das outras, em termos de implementação. As classes devem ser desacopladas entre si e cada uma ter uma responsabilidade específica.

Esses requisitos podem ser atingidos focando na organização e na clareza do código. A utilização de diagramas para projetar o que será desenvolvido também ajuda nesse processo.

Outra tentativa para elevar a qualidade do código é aplicar idéias para maximizar a coesão e minimizar o acoplamento entre as classes. Coesão refere-se à diversidade de responsabilidades associadas a uma classe; quanto menor a diversidade, maior a coesão. Acoplamento entre classes refere-se ao quanto duas classes são dependentes uma da outra; é uma medida do quanto uma classe conhece as outras.

A solução para maximizar a coesão é criar classes que representam apenas uma abstração e onde cada atributo descreve uma instância da classe. O problema do acoplamento pode ser resolvido com a idéia de criar classes que dependam somente da interface das outras e não da implementação.

Apesar de simples, esses detalhes tornam o código mais simples, fácil de entender, modificar, estender e reutilizar.

Um código de qualidade é um importante passo na busca por tornar o jogo preparado para ser estendido. A idéia é que qualquer pessoa interessada e com conhecimentos na linguagem e na plataforma tenha a possibilidade de adicionar novas funcionalidades ao jogo.

O critério de extensibilidade visa não somente ter um projeto passível de ser estendido, mas um projeto pensado, estruturado e preparado para tal. A inclusão de funcionalidades ou novos mini-games deve ser simples, fácil e não exigir muitos esforços por parte do desenvolvedor.

7.4.1 Design Patterns

Na busca por um código de qualidade, a aplicação de padrões de design orientados a objetos [5] aparece como uma solução bem interessante. Os padrões criam um vocabulário comum entre os engenheiros de software.

O uso de padrões permite elevar o nível de abstração do sistema e possibilita discutir de um nível mais alto a estrutura do projeto. Os padrões descrevem bases de soluções para problemas que ocorrem repetidamente.

O projeto aplicou padrões visando o critério de extensibilidade. Alguns dos padrões utilizados no desenvolvimento serão detalhados a seguir.

O *Singleton* é um padrão de criação que garante que será criada somente uma instância de uma determinada classe e esta pode ser facilmente acessível de qualquer ponto do programa. No projeto, o padrão *Singleton* foi utilizado no cenário, por exemplo. O cenário é sempre único, em qualquer momento da execução do jogo e deve estar facilmente acessível.

```
class Cenario() :

        _instancia = None

        def __init__ (self) :
                if Cenario._instancia :
                        raise Cenario._instancia
                Cenario._instancia = self
```

Código 7.2: *Singleton.*

O *Template Method* é um padrão comportamental que define um método modelo. Normalmente, é um método concreto em uma classe abstrata em que cada passo do algoritmo é definido por outro método abstrato. O uso desse padrão permite que somente um passo do método seja facilmente alterado sobrescrevendo um dos métodos que compõe o *Template Method*. No projeto, o *Template Method* foi utilizado para definir o método principal da classe abstrata *Mini-Game*. As subclasses, que representam os mini-games concretos, é que são responsáveis por definir cada um dos métodos e assim determinar o comportamento do mini-game.

A utilização deste padrão permitiu um resultado muito interessante para estender o jogo. Para a inclusão de um novo mini-game, por exemplo, basta criar os modelos e escrever um conjunto de cinco métodos.

```
class MiniGame () :

        def __init__ (self) :
                self.emAndamento = True

        def run (self):
                self.carrega()
                self.inicia()
                self.mainLoop = taskMgr.add(self.loop, "loop")
                while self.emAndamento :
                        self.mainLoop.step()
                self.finaliza()
                self.limpa()

        @abstractmethod
        def carrega (self):
                raise NotImplementedError

        @abstractmethod
        def inicia (self):
                raise NotImplementedError

        @abstractmethod
        def finaliza (self):
                raise NotImplementedError

        @abstractmethod
        def loop (self) :
        raise NotImplementedError

        @abstractmethod
        def limpa (self) :
                raise NotImplementedError
```

Código 7.3: *Template Method.*

O *Memento* é um padrão comportamental que captura e torna acessível externamente o estado interno de um objeto sem violar o encapsulamento. Assim, garante que este estado possa ser recuperado posteriormente. No projeto, o *Memento* é aplicado quando o jogador faz a transição entre o jogo base e um mini-game qualquer. No momento em que o mini-game tem início o estado do jogo base é capturado e armazenado. Assim, ao final do mini-game, este estado é restaurado e o jogo continua de onde havia parado.

```
def transicaoMiniGame(self):
        self.estado = EstadoMemento.salvaEstado(self.jogadores)
        ...

def voltaParaJogoBase(self):
        ...
        EstadoMemento.recuperaEstado(self.jogadores, self.estado)
```

Código 7.4: *Memento*.

O *Observer* é um padrão comportamental que define uma relação de dependência entre objetos de forma que quando o estado de um objeto muda, todos os seus dependentes são notificados. No projeto, o *Observer* aparece no momento em que um mini-game é iniciado, por exemplo. Quando um jogador encontra um mini-game o estado do jogo base é alterado e todos os dependentes, no caso os jogadores, precisam ser avisados.

O *Listener* é uma implementação específica do padrão *Observer* e no projeto está presente principalmente nas classes que representam o *Cliente* e o *Servidor*.

O *Command Dispatch* é um padrão comportamental exclusivamente para linguagens dinâmicas que encapsula um comando como um objeto. No projeto, o *Command Dispatch* foi aplicado na classe *Menu* para definir a rotação para o menu correto sem necessitar de criar vários condicionais para tal. O fato de a linguagem ser dinâmica permite chamar diretamente o método associado ao comando.

```
class Menu () :

def __init__ (self, main) :
        ...
        self.opcoesMenu = {
                SAIR_APRESENTACAO   : self.goToPrincipal,
                INICIAR_PRINCIPAL   : self.goToIniciar,
                OPCOES_PRINCIPAL    : self.goToOpcoes,
                SAIR_PRINCIPAL      : self.finaliza,
                CREDITOS_OPCOES     : self.goToCreditos,
                SOM_OPCOES          : self.finaliza,
                VOLTAR_OPCOES       : self.goToPrincipal,
                CRIAR_INICIAR       : self.finaliza,
                PROCURAR_INICIAR    : self.finaliza,
                VOLTAR_INICIAR      : self.goToBackPrincipal,
                VOLTAR_CREDITOS     : self.goToOpcoes,
        }
        ...

def goTo (self, menu) :
        func = self.opcoesMenu[menu]
        func()

def goToPrincipal (self) :
        self.menu.hprInterval(1, (0, 90, -90)).start()
        self.menuAtual = MENU_PRINCIPAL

def goToOpcoes (self) :
        self.menu.hprInterval(1, (0, 180, -90)).start()
        self.menuAtual = MENU_OPCOES

def goToBackPrincipal (self) :
        self.menu.hprInterval(1, (0, 90, -90)).start()
        self.menuAtual = MENU_PRINCIPAL

def goToIniciar (self) :
        self.menu.hprInterval(1, (0, 0, -90)).start()
        self.menuAtual = MENU_INICIAR

def goToCreditos (self) :
        self.menu.hprInterval(1, (0, 270, -90)).start()
        self.menuAtual = MENU_CREDITOS
```

Código 7.5: *Command Dispatch.*

8 Resultados

O resultado final do projeto foi uma base jogável em rede, multiplataforma, multiplayer e preparada para ser estendida. O jogo foi baseado na plataforma prevista e o projeto proporcionou uma boa oportunidade de conhecer, aprender e explorar a engine do Panda3D.

O jogo final apresenta todas as funcionalidades propostas, como a comunicação na rede com um estudo sobre a troca de mensagens, a criação dos modelos tridimensionais, a modelagem de fenômenos físicos presentes no jogo, o jogo base, um exemplo de mini-game, a aplicação de padrões, a utilização de técnicas de inteligência artificial para modelar os jogadores não-humanos e o fácil acoplamento de novos mini-games.

Figura 6: Acesso a um mini-game.

A realização de um projeto grande em cima de uma linguagem, plataforma e frameworks até então não familiares foi uma experiência de como é uma situação real. O projeto permitiu também aplicar diversas áreas e competências da computação.

Manuais de usuário e de desenvolvedor foram produzidos para facilitar o entendimento das partes e esclarecer todos os pontos do jogo. Futuramente, o jogo está preparado para crescer sem muitos esforços por parte dos desenvolvedores.

9 Conclusão

O desenvolvimento do projeto mostrou que a criação de um jogo não é uma tarefa simples. É preciso realizar uma série de estudos, desde a viabilidade do projeto inicial até uma implementação final com bom desempenho.

Apesar de parecer que os conceitos de computação apareceriam apenas no desenvolvimento dos mini-games, na realidade eles estão sempre presentes. Diversas tarefas aparentemente simples tem associados a elas sofisticados aspectos computacionais.

Pode-se ver na prática que é mais importante adaptar uma metodologia ao projeto do que o contrário. Por isso não se escolheu uma metodologia específica, mas um conjunto de técnicas de desenvolvimento e organização conhecidas.

O desenvolvimento em grupo foi importante e facilitou bastante o andamento do projeto. A experiência de trabalhar em equipe mostra como é importante existir e saber administrar o conflito de idéias para atingir um entendimento comum do problema e da solução.

Todos os estudos, escolhas e decisões tiveram influências positivas do trabalho em grupo. A discussão dos requisitos e das soluções permite escolher as melhores opções para o projeto, evitando erros, retrabalhos e desperdício de tempo ou esforços. Além disso, a soma dos interesses e conhecimentos específicos de cada integrante contribuiu com a busca por melhores soluções.

Trabalhar em equipe permite uma ampla troca de conhecimentos e, quando bem administrado, uma maior agilidade no cumprimento das tarefas. A colaboração de todo o grupo é essencial para alcançar mais rapidamente a maturidade das idéias.

O grupo conseguiu desenvolver com sucesso um projeto grande, trabalhando com ferramentas e framework não familiares. O projeto permitiu ainda conhecer, estudar e aplicar interessantes conceitos de diversas áreas da computação exploradas durante o curso de graduação.

9.1 Futuro do Projeto

Futuramente, o projeto poderá ser facilmente ampliado por qualquer pessoa interessada e que tenha algum conhecimento sobre a engine do Panda.

Atendendo a um dos requisitos iniciais, o projeto está preparado para ser estendido. Alterações pontuais nas funcionalidades existentes ou inclusões de novas funcionalidades e/ou mini-games podem ser feitas de forma simples, sem muitos esforços por parte do desenvolvedor e principalmente sem impactar negativamente no que já existe.

Algumas funcionalidades que podem ser incluídas no projeto:

- adicionar limite de tempo no mini-game de exemplo;
- incluir pontuação para os jogadores no jogo base;
- adicionar itens diversos no jogo base que causem vantagem/desvantagem aos jogadores;
- criar novos modelos para os personagens;
- colocar vantagens para os jogadores que vencerem os mini-games;
- incluir outros algoritmos de Inteligência Artificial para o comportamento dos jogadores não-humanos;
- criar e acoplar novos mini-games.

Por se tratar de um jogo, as opções de ampliação do projeto são muitas. Basta que um desenvolvedor motivado e criativo queira adicionar um novo recurso que o jogo está preparado para tal.

Lista de Figuras

Referências

[1] http://www.panda3d.org/

[2] http://usa.autodesk.com/adsk/servlet/pc/index?siteID=123112&id=13567410

[3] http://www.tecgraf.puc-rio.br/ mgattass/Quaternios.pdf

[4] Russell S. e Norvig P. *Artificial Intelligence - A Modern Approach*, Segunda Edição

[5] http://www.suttoncourtenay.org.uk/duncan/accu/pythonpatterns.html

10 Parte Subjetiva - Gustavo Vilela

Esta parte do documento apresenta a minha experiência pessoal com o desenvolvimento desse projeto. Os desafios e frustrações enfrentados, as disciplinas da graduação que serviram de base para o sucesso do projeto e quais são os próximos passos a seguir.

Em primeiro lugar gostaria de agradecer ao professor Flávio. Durante todo o projeto, ele nos deu o apoio necessário e ajudou com idéias e sugestões, principalmente no momento mais crítico, o início.

10.1 Desafios e Frustrações

Todos os novos projetos trazem junto novos desafios e esses são os responsáveis por garantir a qualidade final, sendo uma motivação a mais para seguir com o projeto. Algumas vezes, porém, os desafios vêm acompanhados de frustrações que ameaçam o bom andamento do projeto ou até a viabilidade do mesmo.

O primeiro desafio enfrentado estava na escolha do projeto. Na indecisão entre um estudo aprofundado em uma área específica ou explorar diversas áreas de forma mais superficial, a segunda opção me parecia mais atrativa. Quando surgiu a opção de fazer um jogo, era uma boa oportunidade de explorar a multidisciplinaridade e o desafio de conseguir obter sucesso em um projeto de grande porte e em equipe, que exige uma motivação a mais para administrar o andamento. A decisão do projeto estava tomada e teve início o desenvolvimento.

A fase mais complicada do nosso projeto foi o início. Os primeiros passos geralmente são mais delicados por serem os responsáveis por determinar um caminho a seguir no restante do projeto. É nesta etapa que o escopo é definido e um plano de trabalho é traçado.

Não bastasse o período delicado, a nossa maior frustração também apareceu nessa etapa. Inicialmente, nossa proposta era desenvolver um jogo para PSP, explorando a característica móvel do videogame e alguns outros recursos como o acelerômetro.

Estudos e pesquisas nessa linha foram realizados. Os primeiros programas com a linguagem foram escritos, o ambiente estava sendo preparado, as

ferramentas necessárias estavam sendo analisadas, as bibliotecas disponíveis foram exploradas e começaram a surgir os primeiros empecilhos.

Encontramos problemas complicados que atacaram diretamente a viabilidade do projeto. Entre as dificuldades estavam a inexistência de uma SDK oficial gratuita para o PSP, a pouca quantidade de documentação disponível, o não suporte a modelagem gráfica 3D de qualidade com as bibliotecas gráficas gratuitas e principalmente o emulador, o melhor que encontramos era muito lento e travava.

Nesse momento foi quando percebemos que não seria possível seguir com o projeto como desejado e optamos por alterar a proposta inicial. A idéia de desenvolver o jogo foi mantida, porém a plataforma PSP foi abandonada. O jogo seria então para PCs e teríamos que recomeçar todas as pesquisas e estudos que já estavam em andamento.

O maior problema dessa mudança é que já estávamos em junho e bastante tempo desde o início do projeto já tinha passado. Naquele momento me assustava a idéia de ter que recomeçar o projeto com praticamente metade do tempo já passado. A inviabilidade da proposta inicial foi a minha maior frustração no projeto.

O que eu considero o maior erro do projeto também foi cometido nessa fase. Quando alteramos a proposta, acabamos mudando apenas a plataforma e mantendo o restante do projeto. Em minha opinião esse foi o grande erro. Muito tempo já havia passado e muitos esforços haviam sido desperdiçados.

A proposta inicial era composta por três mini-games já que acreditávamos que os desafios e estudos necessários estariam nos problemas explorados neles, porém desenvolver um jogo de qualidade exige muitos outros estudos e pesquisas. Considerando esses esforços, a proposta inicial de três mini-games talvez fosse ambiciosa, com a alteração do projeto na metade do ano, essa proposta passou a ser uma utopia. Na minha opinião, manter essa proposta foi o grande erro.

Felizmente, esses problemas que considero mais graves aconteceram logo no início do projeto e o restante do planejado ocorreu bem, todos os outros itens propostos foram atingidos com sucesso. Após esse momento de instabilidade, o professor Flávio nos sugeriu utilizar a engine Panda, que foi o novo rumo do nosso projeto.

Os desafios seguintes foram adquirir certa maturidade com o Panda e com o 3D Studio Max. Ambos são muito poderosos, mas para explorar bem todos esses recursos é necessário ter um bom domínio. Em relação ao Panda, foi um grande desafio aprender a manipular eficientemente todas as bibliotecas e o resultado final me agrada bastante em relação à maturidade atingida com a engine.

O próximo desafio foi a escolha do mini-game. Criar um jogo divertido, que explore conceitos de computação e que não seja simples não é uma tarefa fácil. Acabamos com uma opção computacionalmente interessante e divertida. A implementação dessa solução trouxe outros desafios relacionados às pesquisas nas diversas áreas envolvidas.

No decorrer do projeto, muitos outros pequenos desafios ocorreram, como os estudos para conseguir ter realidade nos efeitos do jogo. O maior deles foi garantir a sincronia de movimentos do personagem na máquina do jogador com os jogadores remotos. A representação fiel do estado dos personagens exige tomar todos os cuidados relacionados com a rede, considerando o atraso das informações e o tráfego de mensagens.

Ao final do projeto, a integração do movimento do jogador não-humano com o restante do jogo não foi concluída com sucesso. Essa é outra frustração minha no projeto, mas não tão grande quanto à frustração inicial e nem tão grave por eu avaliar que o resultado final foi bastante satisfatório.

10.2 Disciplinas Relevantes e Aplicação dos Conceitos

Diversas disciplinas cursadas durante a graduação contribuíram, direta ou indiretamente, com o desenvolvimento do projeto. Algumas delas são fundamentais para a formação e desenvolvimento da lógica por parte do desenvolvedor, outras por abordarem assuntos relacionados aos explorados no projeto. A seguir estão listadas essas disciplinas.

- **MAC0110 - Introdução à Computação**
  MAC0122 - Princípios de Desenvolvimento de Algoritmos
  MAC0323 - Estruturas de Dados
  Muitas vezes estas disciplinas introdutórias são o primeiro contato do aluno com programação. Por isso, são fundamentais para o desenvolvedor criar uma base teórica sólida que será aplicada em todos os projetos que virá a desenvolver. Nestas disciplinas o aluno é estimulado a atingir a maturidade na programação, sendo capaz de fazer as melhores escolhas em termos de algoritmos e estruturas.

- **MAC0239 - Métodos Formais em Programação**
  Essa disciplina é a primeira a focar no raciocínio lógico do aluno. O desenvolvimento da lógica é fundamental para formar um bom desenvolvedor.

- **MAT0111 - Cálculo Diferencial e Integral I**
  FAP0126 - Física I
  Essas disciplinas são responsáveis por fornecer uma boa base matemática para o aluno. No projeto foram importantes para toda a modelagem física presente no jogo. No tratamento das colisões e principalmente na modelagem dos fenômenos físicos do mini-games: movimento da plataforma, inércia e colisões inelásticas da bolinha.

- **MAT0139 - Álgebra Linear para Computação**
  Completa a base matemática do aluno com conceitos de espaços vetoriais e transformações lineares. No projeto é a base para o tratamento da colisão da bolinha com o labirinto, no mini-game. Utilizada para encontrar os vetores e ângulos da colisão, do efeito e as projeções, por exemplo.

- **MAT0213 - Álgebra II**
  Importante para desenvolver a capacidade de abstração do aluno. No caso do projeto, foi o primeiro contato com conceitos como os quatérnios aplicados na modelagem da rotação da bolinha no mini-game.

- **MAC0211 - Laboratório de Programação I**
  A primeira disciplina em que o aluno tem que desenvolver um projeto grande, completo e em equipe. A importância de ter um projeto bem documentado e estruturado, o primeiro contato com bibliotecas, ferramentas e técnicas que auxiliam no desenvolvimento e com interfaces gráficas aparecem nessa disciplina. No nosso caso foi também o primeiro contato com *Visão Computacional* e onde desenvolvemos um jogo que serviu de motivação para este projeto. Aqui o aluno aprende a buscar soluções, estudar ferramentas por conta própria e começa a atingir a maturidade no desenvolvimento.

- **MAC0242 - Laboratório de Programação II**
  Assim como na disciplina anterior, o aluno tem que desenvolver um projeto grande, completo e em equipe. Os primeiros conceitos de orientação a objetos são vistos e tem-se o primeiro contato com linguagens de programação não tipadas. É mais um passo importante do aluno para atingir a maturidade no desenvolvimento.

- **MAC0328 - Algoritmos em Grafos**
  A disciplina onde o aluno aprende a trabalhar eficientemente com a estrutura de grafos. No projeto foi importante para mapear o modelo do cenário do mini-game para o mundo discreto, o labirinto foi representado como um grafo. Técnicas de busca de caminhos também foram aplicadas sobre este modelo discreto.

- **MAC0332 - Engenharia de Software**
  O conhecimento de técnicas para guiar o desenvolvimento de projeto de grande porte começa nessa disciplina. É onde somos introduzidos as etapas do desenvolvimento de software e a diferentes paradigmas, metodologias de projeto e boas práticas de programação. No nosso caso, serviu como uma linha base para guiar, acompanhar e avaliar o andamento do projeto.

- **MAC0441 - Programação Orientada a Objetos**
  Nesta disciplina o aluno conhece e explora a fundo os conceitos de orientação a objetos. No nosso caso foi importante também por ter sido uma disciplina que explorou conceitos de qualidade de código e o primeiro contato com *design patterns*, fundamentais para o critério de extensibilidade proposto no neste projeto.

- **MAC0431 - Introdução à Computação Paralela e Distribuída**
  MAC0438 - Programação Concorrente

São disciplinas onde se tem o primeiro contato com a manipulação e tratamento de tarefas rodando em paralelo e/ou distribuídas. No projeto, esses conceitos foram importantes para pensar e criar o fluxo de informações, considerando os jogares remotos. Não foi aplicada nenhuma técnica em especial, mas esse conhecimento foi importante para diagnosticar e resolver alguns problemas de sincronismo.

- **MAC0425 - Inteligência Artificial**
  Nessa disciplina são introduzidos os conceitos de agentes inteligentes e técnicas de tomada de decisão e busca de caminhos com robôs. No projeto, as idéias e algoritmos apresentados na disciplina serviram de base para a criação dos jogadores não-humanos.

- **PCS0210 - Redes de Computadores**
  O foco dessa disciplina é bastante diferente do restante do curso. Os conceitos vistos aqui foram fundamentais na criação da rede de comunicação do jogo. A estrutura aplicada, os estudos referentes ao tráfego de mensagens e ao formato ideal da mesma foram baseados em conhecimentos adquiridos nessa disciplina.

10.3 Continuação na Área

O área do desenvolvimento de jogos está crescendo e se fortalecendo cada vez mais. No futuro, seguir um caminho nessa área é uma alternativa bastante promissora e envolvida por boas expectativas.

A princípio, não devo seguir essa linha por ter interesse em estudar outras áreas, inclusive não totalmente voltadas para a computação, e que hoje não vejo com aplicação em jogos. Mas, caso optasse por continuar nesse caminho, o *aMaze Unknown* seria uma boa possibilidade.

Se fosse seguir na área de jogos, acho que estudar e explorar mais o *Panda* seria uma ótima forma de aprimorar os conhecimentos referentes à engine e ao desenvolvimento de jogos de forma geral. A engine se mostrou muito completa e poderosa. A experiência com o projeto mostrou que a facilidade em desenvolver aumenta quando se obtem uma maior fluência com o *Panda*. A engine possui muitos recursos que podem ser melhor explorados e o real poder da engine surge quando o desenvolvedor sabe utilizar tudo que a engine pode oferecer.

Outro caminho seria conhecer e desenvolver jogos em outras engines disponíveis para comparar a facilidade e resultados obtidos com a experiência do *Panda*.

Se escolhesse por continuar com o jogo atual, o resultado do projeto está preparado para crescer. Todo o planejamento foi pensando para que ele estivesse pronto para sofrer mudanças sem causar muitos impactos nas funcionalidades existentes. Seguindo essa linha, o futuro do jogo poderia ser através de melhorias nas funcionalidades existentes ou acrescentar novas e novos mini-games.

10.3.1 Melhorias e Extensão

O primeiro passo talvez fosse fazer melhorias no jogo existente. Algumas melhorias pontuais nas funcionalidades existentes.

O segundo passo, e talvez o mais promissor, seria estender o jogo existente. Existem diversas formas de expandir o jogo atual, adicionando novos mapas para o jogo base, incluindo outros modelos de personagens ou incluir funcionalidades no jogo como: pontos e algum tipo de vantagem por vencer os mini-games.

Pessoalmente, como gosto da área de inteligência artificial, se eu fosse estender o jogo gostaria de criar novos agentes inteligentes e aplicar outros algoritmos para a tomada de decisão. Outro aspecto também atrativo seria acoplar novos mini-games, já que essa tarefa é bastante simples. Aplicar neles novas idéias e conceitos que possam vir a me interessar.

11 Parte Subjetiva - Itai de Ávila Soares

Desenvolvimento para jogos foi uma área que pensei que me interessaria quando comecei a cursar a graduação. Depois da primeira experiência em Laboratório de programação me decepcionei muito dado a dimensão do projeto e como podem ser os resultados. Apesar disso não queria fazer um trabalho de conclusão de curso que fosse focado em alguma área somente, queria fazer algo que tentasse explorar a multidisciplinaridade e que desafiasse por ser um projeto que exigisse bastante da parte administrativa também. Após pensar muito, um jogo acabou cobrindo essas minhas exigências e muitas também de meus colegas.

11.1 Desafios e frustrações

Fazer um jogo também é algo que não é muito bem visto como um trabalho, pois a impressão as vezes que se passa é que é divertido fazer e que qualquer um poderia desenvolver em tempo curto e com bons resultados, gostaria muito que isso fosse verdade. As frustrações começaram quando se estava tentando implementar um jogo pra PSP, a documentação era ruim, o emulador dava muitos problemas e a biblioteca não tinha algumas funções do openGL. Recomeçar um trabalho praticamente no meio do ano é algo que não estava nos planos do grupo. Graças ao professor Flávio Soares, tivemos uma orientação de qual caminho seguir para continuarmos o projeto.

Foi muito difícil pensar em algum jogo que não fosse óbvio demais e que não fosse difícil demais e ao mesmo tempo em que pudesse ser abrangente. Depois de fechada a ideia, tínhamos em mente que teríamos muitos minigames e que o gráfico fosse muito melhor. Acredito que tivemos desafios com o Panda e com o 3DMax, como as tarefas foram distribuídas eu fiquei um pouco mais focado no 3DMax, e foi um desafio aprender do zero a modelar pra fazer algo suficientemente aceitável pro projeto, pequenas mudanças geravam horas de remodelagem.

Gostaria de ter tido um jogo mais completo, mas devido às dificuldades encontradas (muitas vezes em áreas que achávamos que seria trivial) conseguimos menos do que projetávamos, essa foi a maior frustração para mim.

11.2 Disciplinas Relevantes e Aplicação dos Conceitos

- **MAC0110 - Introdução a Computação**
  MAC0122 - Princípios de desenvolvimentos de algoritmos
  MAC0323 - Estrutura de dados
  Bases de um curso de Ciência da Computação, foi importante para termos os primeiros contatos e sabermos as boas práticas na programação. São matérias que nos deram as bases para o desenvolvimento, muito dificilmente as outras seriam bem feitas sem uma base sólida dessas três matérias.

- **MAC0211 - Laboratório de Programação I**
  MAC0242 - Laboratório de Programação II
  São disciplinas que nos deram as primeiras oportunidades de trabalhar em equipe, com projetos grandes, o desafio de implementar um jogo e de, em equipe, cumprir prazos e metas.

- **MAC0332 - Engenharia de Software**
  Uma matéria que penso ser uma das mais importantes pra vida profissional. Pudemos pela primeira vez ter um cliente real e com técnicas de desenvolvimento implementamos projetos grandes em equipe. Deu-nos bases para sabermos projetar, planejar e desenvolver o nosso trabalho de maneira organizada.

- **MAC0328 - Algoritmos em grafos**
  Na parte de inteligência do personagem foi fundamental o conhecimento de algoritmos que trabalham com grafos (em especial o do Dijkstra).

- **MAC0420 - Introdução à computação gráfica**
  Fundamental quando foi necessário implementar cenários computacionalmente robustos e leves para processamentos, também importante para saber que tipo de mapeamento, luz e projeção seriam usados. Também foi importante pra saber o que buscar no Panda, baseado em experiências com a matéria.

- **MAT0139 - Álgebra linear**
  Base para computação gráfica, para modelagem, para processamento e tratamentos de eventos do jogo.

- **MAT0213 - Álgebra II**
  Na modelagem de rotação da esfera no espaço de dimensão três, a álgebra II nos deu a ideia de utilizarmos quatérnios para rotação de forma que se aproximasse da realidade.

11.3 Continuação na Área

Desenvolver jogos é sempre um desafio, tanto pela dificuldade do projeto em si, como pela concorrência num mercado que existem dezenas de empresas grandes e milhares de empresas médias e pequenas. Eu gostaria de ter a oportunidade de trabalhar com uma biblioteca oficial para alguns consoles e aprender mais da parte de modelagem. Essa área para mim é mais um hobby do que uma intenção de escolha profissional.

12 Parte Subjetiva - Luiz Ricardo Romagnoli

12.1 Desafios e Frustrações

A primeira indecisão que tive foi no momento de decidir qual seria a orientação do meu trabalho, se ele teria um cunho teórico ou prático. A escolha pela primeira orientação trazia o problema do tema. Durante todo o curso, tive contato com várias áreas que despertaram meu interesse. Portanto, a escolha por um tema específico seria muito difícil.

O fato de eu gostar de várias áreas me levou à decisão natural de fazer um trabalho prático, onde eu teria possibilidade de estudar e aplicar, mesmo que não profundamente, essas várias áreas.

A escolha de fazer um jogo partiu da experiência anterior na disciplina Laboratório de Programação I. Nela, meu grupo fez um jogo muito interessante, apesar de simples, e de muito sucesso entre os colegas, chamado West Chicken. A idéia inicial para o TCC foi de expandir esse jogo.

Parte do grupo que participou desse projeto teve o mesmo pensamento em relação ao trabalho anterior e decidimos fazer o TCC juntos, seguindo a idéia de usar o West Chicken como base, seja usando seu tema, seja usando a experiência em desenvolver um jogo.

Implementar um jogo distribuído foi sempre o mote principal. Inicialmente, tentamos implementá-lo para uma plataforma móvel, o iPhone. Essa idéia foi bastante agradável para mim, que sempre me interessei por dispositivos móveis. Foi quando aconteceu a primeira grande frustração. Na pesquisa para montar o ambiente, descobrimos que a SDK oficial era paga. Além disso, eram necessários laptops MAC para o desenvolvimento. Tudo isso deixou a opção inviável.

A próxima tentativa foi com o videogame portátil PSP, que possui mecanismo de comunicação sem fio entre os aparelhos. Porém, não havia uma SDK oficial disponível e as livres eram bastante incompletas. Como completar uma SDK não era o objetivo do trabalho, a idéia foi abandonada.

Perdeu-se muito tempo nessas duas tentativas e chegou-se ao meio do ano sem praticamente nada definido, exceto a idéia inicial. Foi nesse momento que

o orientador, o professor Flávio Corrêa, fez sua primeira grande participação: nos indicou o Panda3D. A partir daí, o projeto fluiu bem até seu término.

12.2 Disciplinas Relevantes

- **MAC0110 - Introdução à Computação**
  MAC0122 - Princípios de Desenvolvimento de Algoritmos
  MAC0323 - Estruturas de Dados
  São as disciplinas que introduzem e aperfeiçoam a programação e o raciocínio lógico. São importantes para qualquer trabalho que envolva uma boa e eficiente programação.

- **FAP0126 - Física I**
  O conhecimento adquirido foi muito útil para modelagem física mais próxima possível da realidade.

- **MAT0139 - Álgebra Linear para Computação**
  Lidar com objetos em três dimensões requer destreza em manipulação de vetores tridimensionais e matrizes, estudadas nessa disciplina.

- **MAC0211 - Laboratório de Programação I**
  MAC0242 - Laboratório de Programação II
  Foram as disciplinas motivadoras do trabalho. Em Lab 1 pelo mote e Lab 2 foi a primeira matéria com contado a orientação a objetos. Ambas são muito preocupadas com boas práticas e organização, que são determinantes para o sucesso de um projeto de grande porte.

- **MAC0328 - Algoritmos em Grafos**
  Útil na implementação do agente inteligente. A aplicação foi bem simples, mas era necessário saber exatamente como ela funcionava.

- **MAC0332 - Engenharia de Software**
  O primeiro contato com metodologias de desenvolvimento, de grande importância na organização de um trabalho.

- **MAC0438 - Programação Concorrente**
  Vários aspectos do Panda acontecem em paralelo. Mesmo a maioria delas sendo transparentes ao usuário, é necessário manter o controle da concorrência quando se altera ou se implementa manualmente funcionalidades.

- **PCS0210 - Redes de Computadores**
  Conhecimento teórico para a implementação da rede e o entendimento de seu funcionamento, que permitiu que as melhores soluções fossem adotadas.

- **MAC0342 - Laboratório de Programação Extrema**
  Tive um contato direto com uma metodologia ágil, que foi a metodologia mais influente no trabalho.

12.3 Aplicação dos Conceitos

Vários conceitos foram abordados no trabalho. Conseqüentemente, quase nenhum foi estudado muito a fundo. Isso já era esperado devido à nossa intenção de fazer uma abordagem ampla aos estudos realizados durante a graduação.

12.4 Continuação na Área

A princípio, não tenho interesse em continuar desenvolvendo jogo, apesar de ter gostado e aprendido muito com a experiência.