MAC-0499
Trabalho de Conclusão de Curso: Implementação de um Jogo com Inteligência Artificial e Aprendizado com Probabilidades

Rogério Cazelato Papetti - NUSP: 5639082
Orientador: Prof. Marcelo Finger
Co-Orientador: Prof. Flávio Soares Corrêa da Silva

30 de novembro de 2009

Sumário

Lista de Figuras

Parte I
Descrição do Trabalho

1 Introdução

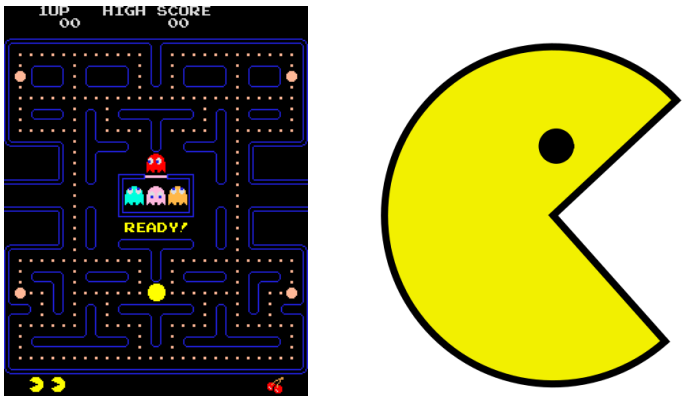
1.1 Motivação

1.1.1 Breve História do jogo Pac-Man

O Pac-Man é um dos jogos mais famosos em toda a história dos games. Foi criado pelo japonês Toru Iwatani, na década de 1980. Foi um grande sucesso nos fliperamas e portado, inclusive, para consoles, como o Atari (foram vendidas 7 milhões de cópias para esse console).

O jogo consiste em um personagem (o Pac-Man) percorrendo um labirinto de duas dimensões em busca de coletar prêmios e fugir de fantasmas que o perseguem. O grande sucesso do jogo é creditado, principalmente à inovação, já que na época a esmagadora maioria dos jogos eram os chamados 'space-shooters' (naves que atiravam em outras naves ou planetas). Esse novo estilo de jogo atraiu a atenção de milhões de novos jogadores. Desde então, se tornou um enorme sucesso.

Segundo pesquisa recente[1], 9 em cada dez americanos reconhecem o personagem Pac-Man ao ver uma imagem do mesmo.



(a) Pac-Man arcade

(b) O personagem Pac-Man

Figura 1: A personagem Pac-Man e seu jogo versão arcade

1.1.2 Regras do Jogo do Pac-Man

O objetivo do Pac-Man é coletar todas as "pac-dots" (bolinhas brancas) espalhadas pelo cenário. Pelo mapa estão espalhados quatro fantasmas, que se movimentam pelo labirinto tentando pegar o Pac-Man. Cada vez que o fantasma encosta no Pac-Man, o mesmo perde uma vida. Se todas as vidas são perdidas, o jogo acaba.

Existem também espalhados pelo mapa maçãs (que são bônus apenas, não precisam ser coletadas) e pontos piscantes (flash-dots), que ao serem ingeridos pelo Pac-Man tornam os fantasmas mais lentos e os impede de tirarem uma vida do Pac-Man. Caso haja uma colisão, o Pac-Man não sofre nada e o fantasma desaparece e volta no centro do mapa.

1.2 Inteligência Artificial (IA)

A inteligência artificial teve seu grande impulso após a Segunda Grande Guerra, principalmente por causa das pesquisas e artigos de Alan Turing. Entretanto, mesmo entre os especialistas da área, não há consenso sobre uma definição do conceito de inteligência artificial.

Dessa forma, os especialistas se dividem em duas correntes sobre o que é inteligência artificial e como trabalhar com ela: Inteligência artificial forte e Inteligência artificial fraca.

A inteligência artificial forte define um dispositivo como inteligente se o mesmo for capaz de raciocinar para resolver problemas, simulando o pensamento humano. O tema é altamente controverso, envolvendo debates sobre ética e consciência.

Já a inteligência artificial fraca considera um dispositivo como inteligente se o mesmo pode resolver problemas não determinísticos. Mesmo que não haja consciência e raciocínio, e sim um algoritmo previamente implementado.

1.2.1 Técnicas de Inteligência Artificial aplicada aos fantasmas no Pac-Man

A inteligência artificial dos fantasmas no Pac-Man é praticamente inexistente. A movimentação do fantasma consiste em seguir numa mesma direção até atingir uma esquina ou intersecção de caminhos, locais nos quais a trajetória é modificada. Entretanto, periodicamente, a direção do fantasma

pode ser alterada.

O fato da direção dos fantasmas ser modificada periodicamente está longe de querer dizer que o movimento do fantasma é aleatório. Os movimentos são determinísticos e foram determinados por engenharia reversa de movimentos que o Pac-Man faria para coletar todas as pac-dots no labirinto, o que permitiu que jogadores de alto nível desenvolvessem sequências de movimentos que os levavam à vitória sempre, os chamados padrões. Sempre há pelo menos um padrão vitorioso por fase.

Para que houvesse alguma diversidade nas movimentações entre os fantasmas, cada um deles, chamados Blinky, Pinky, Inky e Clyde, tem uma personalidade diferente. Um deles é um pouco mais rápido, outro muda de direção mais vezes, por exemplo.

Segundo o criador do jogo, esse sistema utilizado na movimentação do Pac-Man permite que o jogo não se torne 'excessivamente chato ou difícil'.

1.2.2 Aprendizado

Um sistema inteligente que aplica as técnicas de aprendizagem deve ser capaz de definir objetivos e tentar alcançá-los, refinando sua forma de tentar atingir essas metas conforme interage com o meio. [2]

A aprendizagem de máquinas com inteligência artificial ganhou muita força nos anos 80, quando começaram a aparecer as fraquezas da inteligência artificial limpa, na qual um algoritmo fixo e imutável era aplicado o tempo todo. Esse tipo de inteligência se mostrou ineficiente quando as condições do meio se modificavam constantemente e, principalmente, quando confrontada contra humanos, já que, encontrada uma falha na inteligência artificial, ela poderia ser explorada indefinidamente.

Hoje já há uma forte literatura sobre aprendizado, identificação de padrões com uso de estatística, utilizados principalmente em algoritmos de jogos para videogame e computadores, principalmente os que utilizam redes neurais.

1.3 Proposta

Já que os fantasmas do Pac-Man não apresentam nenhum tipo de inteligência artificial, abre-se então uma possibilidade muito interessante para a experimentação de algoritmos de inteligência artificial.

O projeto consistirá, então, na implementação de um jogo do tipo Pac-Man em 3 dimensões com algumas adaptações nas regras para que sejam implementados alguns algoritmos experimentais de inteligência artificial com probabilidades e seus resultados analisados num contexto prático.

Um desses algoritmos controlará a movimentação do fantasma, enquanto o outro controlará o aprendizado do fantasma a partir dos resultados obtidos de experiências anteriores. A proposta consiste, então na experimentação e análise desses algoritmos.

1.4 Objetivos

Os objetivos desse trabalho são:

a Desenvolver algoritmos de movimentação e de aprendizado para os fantasmas do jogo Pac-Man

b Implementar o jogo Pac-Man para que os algoritmos possam ser testados

c Verificar no contexto proposto a efetividade de cada algoritmo implementado, por meio de testes e análise dos resultados.

1.5 Organização Deste Documento

1.5.1 Descrição do Trabalho

Na seção 2 há uma breve descrição das ferramentas utilizadas; na seção 3 há a explicação sobre os algoritmos desenvolvidos; a seção 4 detalha todas as atividades que foram realizadas na construção do projeto; a seção 5 descreve o experimento; na seção 6 há a apresentação dos resultados; na seção 7 são exibidas as conclusões.

1.5.2 Apêndice

Há apenas uma seção com conceitos importantes do Panda-3D para entendimento do projeto.

1.5.3 Parte Subjetiva

A seção 1 contem os agradecimentos aos colaboradores do projeto; a seção 2 apresenta o futuro do projeto; a seção 3 revela os desafios, frustrações e vitórias durante o trabalho; a seção 4 discorre sobre quais matérias cursadas durante a graduação colaboraram no desenvolvimento do trabalho.

2 Ferramentas

2.1 Maya

O Maya é um dos programas de modelagem, animação, renderização e efeitos especiais em 3D mais utilizados no mundo por profissionais da computação gráfica. Além de ser uma ferramenta muito poderosa, é multiplataforma. Infelizmente, entretanto, não é gratuita. A versão utilizada no projeto foi o AutoDeskMaya 2008 Trial (versão gratuita que expirava após 30 dias). [3]

Figura 2: Renderização de uma face feita com o Maya

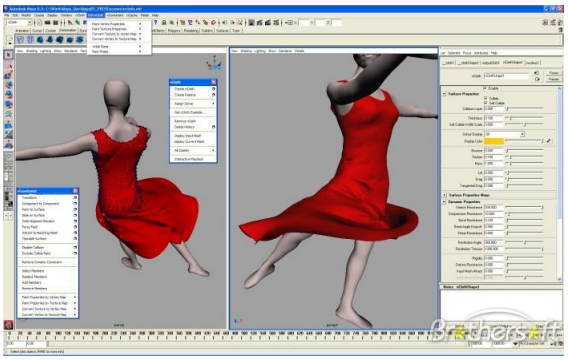


Figura 3: Modelagem do Maya de elementos com articulações e animações

O Maya foi utilizado para modelar todos os elementos que constituíram o cenário do jogo: Pac-Man, fantasmas, pac-dots e o próprio mapa. A modelagem, entretanto, foi bem simplificada, sem efeito de luzes ou renderização e textura complexas. Isso devido à dois fatores: à compatibilidade com o Panda-3D (apenas algumas modelagens básicas eram aceitas pelo Panda-3D sem a necessidade de modificações mais profundas nos arquivos gerados pelo

Maya) e à complexidade do próprio Maya (já que a modelagem gráfica não era o foco principal do projeto, não havia motivo para perder tempo em demasia na tentativa de dominar o programa por completo para fazer algo muito sofisticado).

2.2 Panda-3D

O Panda-3D é um engine 3D com diversas bibliotecas que facilitam a vida do programador que pretende desenvolver um jogo. Esse ambiente é multi-plataforma e gratuito, inclusive para uso comercial. Apesar de suportar C++, é fortemente aconselhável o uso de Pyhton para desenvolvimento. [4]

Figura 4: Cenário com reflexo na água gerado pelo Panda-3D

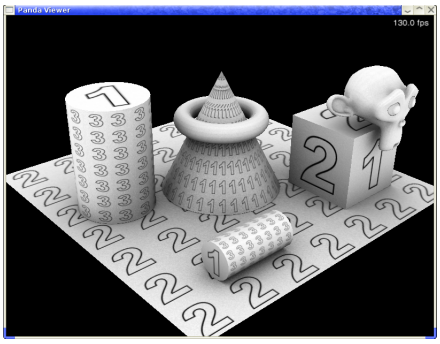


Figura 5: Modelos gerados pelo Panda-3D

Esse ambiente fornece uma vasta gama de facilidades ao desenvolvedor: classes para tratamento e detecção de colisões, raio de visão, tratamento para articulações de personagens com esqueleto, implementação de módulos básicos de física e controle de câmera.

Esse engine foi, obviamente, utilizado o tempo todo durante o projeto para carregar todos os objetos em 3 dimensões e aplicar toda a lógica inerente ao jogo (algoritmos de Inteligência artificial, colisões, pontuação, movimentação das personagens).

Muitos termos e conceitos do Panda-3D serão citados durante as seções seguintes. No apêndice há uma seção detalhada sobre o framework, com explicações sobre cada conceito utilizado.

3 Os Algoritmos Utilizados

Apesar de estarem relacionados, os dois algoritmos de inteligência artificial utilizados no projeto serão explicados separadamente. Entretanto, ficará visível onde ambos se interligam.

3.1 Regras Modificadas em Relação ao Jogo Original

Antes de explicitarmos os algoritmos, é necessário deixar bem claro que algumas mudanças foram feitas nas regras de forma a tornar os experimentos mais rápidos e plausíveis. As mudanças em relação às regras originais do Pac-Man foram:

- Tempo máximo de jogo. Após esse tempo, caso o Pac-Man não tenha coletado todas as 'pac-dots', perde.
- Fantasma pode ficar parado e trocar de direção quantas vezes desejar.
- Não há maçãs que valham bônus ou bolinhas que dêem super-poderes ao Pac-Man.

Ao se olhar com cuidado para as novas regras, pode-se verificar que há uma brecha para uma estratégia sempre vencedora: o fantasma simplesmente parar sobre uma 'pac-dot', impedindo o Pac-Man de coletá-la. Vale lembrar que essa brecha é intencional e é exatamente isso que buscamos verificar: O quão perto ou longe de atingir essa estratégia vencedora chegará o fantasma. Em suma, o quanto ele aprendeu.

3.2 O Algoritmo de Movimentação

O fantasma deve decidir a partir de informações que coleta do cenário o que deve fazer a cada frame. Essa é a premissa básica do algoritmo de inteligência artificial que o fantasma utiliza para se movimentar pelo cenário. Esse algoritmo não recebe entrada e não manipula saída. Desde o lançamento do raio de visão até a ação ser executada, tudo ocorre dentro desse método:

```
1: repeat
2:    lança raioDeVisao
3:    objetoMaisPerto ← colHandler.devolveMaisPerto()
4:    if fantasma.estaSobrePacDot() = TRUE then
5:       sorteia ação e executa
6:    else
```

```
7:      if fantasma.estaPertoDeParedeFrontal() = TRUE then
8:        if fantasma.existeEntradaLateral() = TRUE then
9:          Vira na entrada Lateral
10:       else
11:         Vira 180 graus e volta
12:       end if
13:     else
14:       Sorteia ação baseado no objeto que está mais próximo e executa
15:     end if
16:   end if
17: until jogo terminar
```

3.2.1 Explicação do Algoritmo

Para coletar informações sobre o que está acontecendo ao seu redor, o fantasma tem sensores que o ajudam a responder cinco perguntas:

1 Estou em cima de uma 'pac-dot' ?

2 Estou vendo uma 'pac-dot'?

3 Estou vendo o Pac-Man?

4 Estou vendo outro fantasma?

5 Estou vendo apenas uma parede?

A cada verificação, no máximo uma das perguntas pode ser respondida positivamente. Pois, o conjunto de ações que o fantasma terá ao seu dispor para decidir depende de qual das perguntas acima foi respondida positivamente. A grande pergunta que aparece, então, é a seguinte: Como garantir que apenas uma pergunta é respondida positivamente?

A primeira pergunta que é feita é a pergunta 1. Assim, a cada frame há a verificação se o fantasma está colidindo com uma 'pac-dot'. Em caso afirmativo, as outras perguntas não são feitas. O fantasma já tem informações suficientes para sua decisão (deve decidir apenas se fica sobre a pac-dot ou se sai de cima da mesma). Dessa forma, apenas a pergunta 1 tem resposta positiva. Caso o fantasma não esteja sobre uma 'pac-dot', ele recorre ao seu raio de visão para verificar o que está enxergando e, dessa forma, decidir por algo.

O raio de visão foi implementado como um segmento de reta que sai do olho do fantasma e tem um alcance fixo. Esse raio poderá colidir com nenhum, um ou mais objetos. Caso não haja colisão, as perguntas de 2 a 5 têm todas respostas negativas e então o cenário não ajudou o fantasma a decidir sua movimentação. Dessa forma, ele se movimentará de forma aleatória. Caso ele enxergue algum objeto, os mesmos são ordenados por distância e o que está mais perto é o objeto enxergado (já que está na frente dos outros). Assim, apenas uma das perguntas 2 a 5 terá resposta positiva. E, assim, o fantasma poderá escolher quais das ações entre as disponíveis para cada uma das respostas afirmativas vai executar.

Para escolher qual ação executar, o fantasma conta com a probabilidade. Cada ação tem uma probabilidade de ser escolhida (inicialmente a probabilidade é a mesma para cada uma das ações em um dos casos e enviesada no outro caso - isso seráexplicado mais adiante). Um número é sorteado e, assim, decide-se qual ação é executada. No próximo frame, o algoritmo é ativado novamente.

3.2.2 Ações Disponíves para o fantasma

Para cada uma das perguntas, há um conjunto de ações disponíveis para o fantasma. Será detalhado, então, quais ações podem ser tomadas à partir de cada pergunta:

1 Estou em cima de uma 'pac-dot' ?

- Continuar em cima da 'pac-dot'.
- Andar na direção que está olhando.
- Andar na direção contrária a que está olhando.

2 Estou vendo uma 'pac-dot'?

- Avançar na direção da 'pac-dot'
- Andar na direção contrária.
- Ficar Parado.

3 Estou vendo o Pac-Man?

- Avançar na direção do Pac-Man.
- Andar na direção contrária.

- Ficar Parado.

4 Estou vendo outro fantasma?

- Avançar na direção do outro fantasma.
- Andar na direção contrária.
- Ficar Parado.

5 Estou vendo apenas uma parede?

- Andar na direção contrária.
- Andar em qualquer outra direção.
- Ficar Parado.

3.3 O Algoritmo de Aprendizado

O algoritmo de aprendizado funciona ao oferecer maior chance para que ações bem sucedidas anteriormente sejam executadas e menor chance para ações que fracassaram. Ou seja, a probabilidade de uma ação ser executada depende, basicamente, de quanto sucesso a mesma teve anteriormente. O sucesso que uma ação teve ao longo do tempo é refletida em pontos (que chamaremos de score). Quanto maior o score de uma ação, mais sucesso ela teve ao longo do tempo e, assim, mais provável que a mesma volte a ser executada. Fica, entretanto, uma pertinente pergunta: como decidir o quão alto é o score de uma jogada, ou seja, como decidir se uma jogada obteve sucesso ou não?

3.3.1 Score

O score de uma ação deve ser tão alto quanto foi bem sucedida a ação. Entretanto, é um pouco difícil definir o que significa o sucesso de uma ação. Assim, foi definido que o score de uma ação é atribuído da seguinte forma:

$$(pontuacaoDaRodada = \frac{tempoQueOcorreu}{tempoTotalDoJogo})$$

Assim, as ações que ocorreram mais perto do fim do jogo (e tiveram maior peso na vitória do fantasma) receberão um peso maior. E, ao fim do jogo, a pontuação que a ação teve até o presente momento será de:

$$pontuacaoAtual+ = pontuacaoDaRodada$$

É importante ressaltar que cada ação foi iniciada tendo 1000 pontos de score.

4 Atividades Realizadas

4.1 Pesquisa Inicial

Em seus primeiros meses, o projeto foi dedicado, primeiramente, ao estudo da linguagem Python, do Panda-3D e do Maya, com o intuito não só de aprendizado como também de verificar a viabilidade de passar por um processo um pouco longo (de modelagem 3D e setup de configurações básicas do jogo e do ambiente de desenvolvimento). Depois de constatado que seria viável a execução desses passos, foram estudadas cada uma das ferramentas.

A primeira ferramenta estudada foi o Maya. Antes mesmo de começar a modelagem com o Maya, foi estudado o Panda-3D, para evitar que existissem problemas de compatibilidade mais tarde. Após isso, iniciou-se o processo de modelagem no Maya e, assim que os modelos ficaram prontos, iniciou-se a implementação no Panda-3D.

4.2 Modelagem Usando o Maya

A modelagem no Maya é baseada em figuras geométricas, os polygons. A partir de formatos básicos, como cones, esferas e paralelepípedos, é possível fazer qualquer tipo de transformação: rotações, recortes, nivelamentos de trechos e composições de uma ou mais figuras num único modelo.

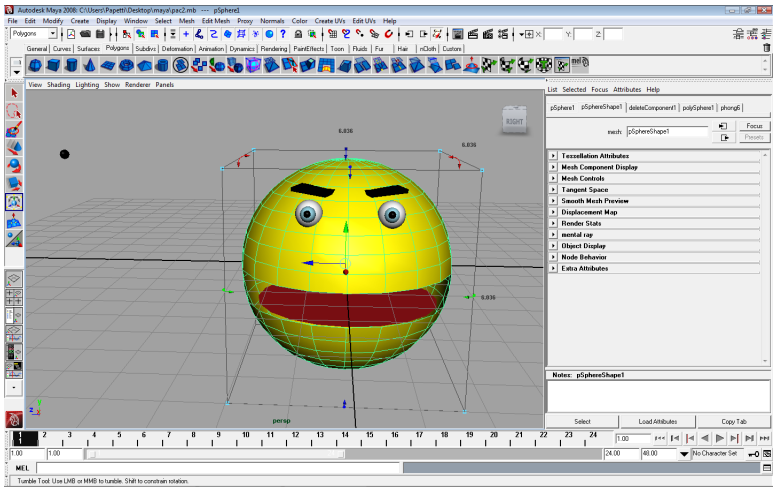


Figura 6: Modelagem do Pac-Man no Maya

É possível também sinalizar se aquela figura que está sendo construída

deverá ser considerada pelo Panda-3D como um objeto de colisão (um muro, por exemplo, pode ser indicado como uma barreira – barrier).

Todas essas configurações são gravadas num arquivo texto de formato egg, que será lido pelo Panda-3D, por meio de um script fornecido pelos fabricantes do Panda-3D.

4.3 Importação dos Arquivos EGG para o Panda-3D

O arquivo EGG guarda todas as informações relativas ao modelo que está sendo exportado. Ele contem os dados de cada um dos vértices, faces, arestas, coloração de cada face, largura de cada aresta. Usa um modelo de tags e é muito extenso, sendo praticamente impossível gerar um EGG sem o auxílio de um software.

Importar os arquivos egg gerados pelo Maya para o Panda-3D foi uma tarefa um pouco mais árdua do que se poderia imaginar. Alguns pequenos detalhes de geração de arquivos e configurações poderiam gerar situações adversas ao se executar a importação dos arquivos. Para a geração de cores, por exemplo, é necessário que se importe a cor de um arquivo jpeg ou png (caso a cor seja inserida diretamente na modelagem pela paleta de cores do Maya, não é reconhecida pelo Panda-3D, resultando num objeto completamente branco.

Entretanto, resolvidos esses pequenos problemas, a tarefa fica bem fácil, já que o Panda-3D consome o arquivo EGG e grava as informações num objeto do tipo PandaNode, mais especificamente numa subclasse do mesmo, chamada NodePath.

4.4 Implementação no Panda-3D

4.4.1 Colisões

Para implementar as colisões, foram utilizados três gerenciadores de colisão: um CollisionHandlerPusher, um CollisionHandlerEventer e um CollisionHandlerQueue. No Pusher foram adicionadas uma esfera de colisão do Pac-Man e uma de cada fantasma. Assim, ambos eram automaticamente bloqueados (não prosseguiam e atravessavam o objeto com quem colidiram) ao bater uns nos outros e nas paredes, que já haviam sido sinalizadas como barreiras durante a construção do mapa no Maya.

Já as pac-dots foram adicionadas no CollisionHandlerEventer. Entretanto, essa configuração apresentava um indesejável efeito colateral: Ao ocorrer a colisão de um fantasma ou do Pac-Man com uma pac-dot, o Pusher entrava em ação e afastava a personagem da pac-dot como se a mesma fosse uma parede.

Para evitar esse efeito, foi utilizado bitmask, evitando que as esferas do Pusher enxergassem as pac-dots. Uma nova esfera com bitmask adequado foi adicionada para os fantasmas e o Pac-Man e esses novos sólidos de colisão foram associados ao Eventer. Dessa maneira, sempre que uma personagem colidisse com uma pac-dot, apenas um evento seria gerado e tratado.

Por fim, os raios de visão frontal e lateral foram associados ao CollisionHandlerQueue. Dessa forma, a cada frame a fila era percorrida e verificado qual o objeto mais próximo cada raio colidiu.

4.4.2 Movimentação das Personagens

Cada fantasma instanciado tem uma Task associada a si. Essa task controla o movimento da personagem, executando diversas tarefas. Em seu inicio, lança os raios frontal e lateral, verifica se a distância para as paredes mudou, se o fantasma está sobre uma pac-dot e o que ele está enxergando. Com tudo isso considerado, uma decisão é tomada sobre o que fazer naquele frame.

5 Descrição Dos Experimentos

Para que fossem testados os algoritmos foram implementados dois experimentos, que consistiam ambos em 30 jogos com duração de 120 segundos, com o mesmo mapa ('mapaParaExperimentos.egg'), contendo um Pac-Man, controlado pelo testador e três fantasmas espalhados pelo mapa. Ambos os experimentos foram rodados num computador PentiumD HT2.8Mhz, com sistema operacional Ubuntu 9.04.

Foram testados dois cenários: num deles os fantasmas tinham exatamente a mesma probabilidade de executar cada uma das ações. No outro cenário, algumas das probabilidades foram estrategicamente enviesadas, na tentativa de simular um comportamento diferente no labirinto.

As ações mais comuns e que eram consideradas cruciais para que os fantasmas tivessem maior chance de vitória ou maior fluidez no movimento ganharam um incremento de de 20%. Ou seja, essas ações tinham 20% mais chances de serem executadas. As ações cujas probabilidades foram alteradas são:

- Sensor: está sobrepac-dot Ação:ficarSobrePac-dot
- Sensor: parede Ação:continuar na mesma direção
- Sensor: Pac-Man Ação: avançar na direção do Pac-Man
- Sensor: pac-dot Ação: avançar na direção da pac-dot

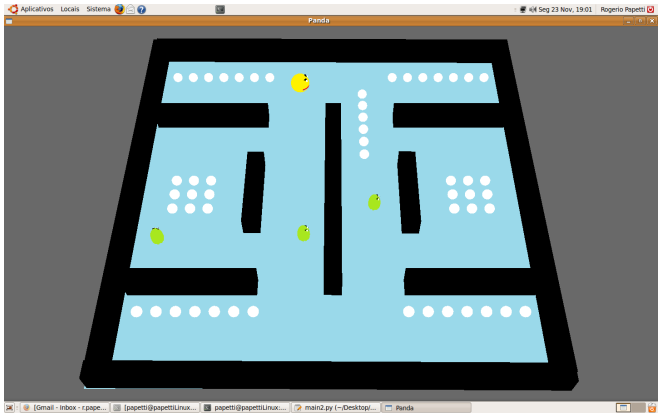


Figura 7: Mapa utilizado nos experimentos

6 Resultados

Serão apresentados os scores inicial e final e o índice de vitórias do fantasma tanto com as probabilidades enviesadas quanto com as não-enviesadas. Cada linha é composta por 3 colunas. A primeira indica qual sensor foi ativado, a segunda qual ação foi executada e a terceira qual o score da ação até o momento (a explicação sobre sensores, ações e score está na seção 3).

6.1 Probabilidades Não Enviesadas

Os 3 fantasmas não conseguiram nenhuma vitória nas 15 partidas disputadas.

O score inicial e o final de cada ação estão listados abaixo:

Score Inicial:
parede avancarParaParede 1000.0
parede andarNaDirecaoContraria 1000.0
pacman avancarParaPacman 1000.0
pacman andarNaDirecaoContraria 1000.0
pacman ficarParado 1000.0
pac-dot avancarParaPac-dot 1000.0
pac-dot ficarParado 1000.0
pac-dot andarNaDirecaoContraria 1000.0
sobrePac-dot continuarEmCima 1000.0
sobrePac-dot sairParaOndeOlha 1000.0
sobrePac-dot saiParaAsCostas 1000.0
fantasma avancarParaFantasma 1000.0
fantasma ficarParado 1000.0
fantasma andarNaDirecaoContraria 1000.0

Score Final:
parede avancarParaParede 4207.0
parede andarNaDirecaoContraria 4199.0
pacman avancarParaPacman 1051.0
pacman andarNaDirecaoContraria 1044.0
pacman ficarParado 1030.0
pac-dot avancarParaPac-dot 1195.0
pac-dot ficarParado 1166.0
pac-dot andarNaDirecaoContraria 1191.0
sobrePac-dot continuarEmCima 1095.0

sobrePacDot sairParaOndeOlha 1104.0
sobrePac-dot saiParaAsCostas 1104.0
fantasma avancarParaFantasma 1000.0
fantasma ficarParado 1000.0
fantasma andarNaDirecaoContraria 1000.0

6.2 Probabilidades Enviesadas

Os 3 fantasmas venceram 5 das 15 partidas disputas.
O score inicial e o final estão disponibilizados abaixo:

Score Inicial:
parede avancarParaParede 4000.0
parede andarNaDirecaoContraria 1000.0
pacman avancarParaPacman 4000.0
pacman andarNaDirecaoContraria 1000.0
pacman ficarParado 1000.0
pac-dot avancarParapac-dot 4000.0
pac-dot ficarParado 1000.0
pac-dot andarNaDirecaoContraria 1000.0
sobrepac-dot continuarEmCima 5000.0
sobrepac-dot sairParaOndeOlha 4000.0
sobrepac-dot saiParaAsCostas 1000.0
fantasma avancarParaFantasma 4000.0
fantasma ficarParado 1000.0
fantasma andarNaDirecaoContraria 1000.0

Score Final:
parede avancarParaParede 9792.0
parede andarNaDirecaoContraria 2503.0
pacman avancarParaPacman 4123.0
pacman andarNaDirecaoContraria 1107.0
pacman ficarParado 1009.0
pac-dot avancarParapac-dot 4592.0
pac-dot ficarParado 1060.0
pac-dot andarNaDirecaoContraria 1045.0
sobrepac-dot continuarEmCima 5583.0
sobrepac-dot sairParaOndeOlha 4172.0
sobrepac-dot saiParaAsCostas 1017.0
fantasma avancarParaFantasma 4000.0

fantasma ficarParado 1000.0
fantasma andarNaDirecaoContraria 1000.0

7 Conclusão

7.1 Probabilidades Não-Enviesadas

7.1.1 Algoritmo de Movimentação

Aspectos que não dependiam de probabilidades, como identificar paredes, entradas, enxergar objetos pelo mapa funcionaram de forma muito satisfatória. A cada frame o fantasma tinha à mão todos os elementos necessários para decidir o que fazer. Tanto os sensores de distância em relação à paredes quanto a visão do que estava mais próximo funcionaram muito bem.

A movimentação dos fantasmas, entretanto, ficou um pouco aquém do esperado ao aplicarem-se probabilidades não enviesadas. Isso porque, com chances iguais de executar as ações, o fantasma executava, na média cada vez uma ação diferente. Sem nenhum padrão, por muitas vezes o fantasma girava, ia e voltava e, no fim, acabava não saindo do lugar. Assim, tornou-se um pouco confuso e, dependendo de sua localização no mapa, um pouco inofensivo. Entretanto, se bem posicionado, sua imprevisibilidade era um fator positivo.

Esse problema de uma personagem poder andar para qualquer um dos lados com probabilidades iguais já foi inclusive estudado por estatísticos e apelidado de "andar do bêbado", já que o movimento de forma aleatória assemelha-se um pouco ao andar do bêbado, incerto. [5]

Dessa forma, pode-se concluir que o algoritmo seria bem aproveitado se o fantasma estivesse "guardando"algo, pois não iria muito longe e seria imprevisível. Entretanto, para longas distâncias (para percorrer todo o labirinto) o algoritmo não seria tão eficiente quanto se gostaria.

Como o mapa demandava muita movimentação dos fantasmas, que iniciavam em pontos não críticos do mapa, os fantasmas não conseguiram derrotar o Pac-Man.

7.1.2 Aprendizado

O algoritmo de aprendizado funcionou de forma correta. Entretanto, os resultados obtidos poderiam ser mais expressivos, pois os jogos duraram pouco e os fantasmas nunca venciam. Assim, aprendeu-se muito pouco e quase nada pode ser sentido em termos de mudanças práticas. Entretanto,

com muita prática e treinamento, é possível que o algoritmo tivesse obtido um sucesso maior.

7.2 Probabilidades Enviesadas

7.2.1 Algoritmo de Movimentação

A movimentação dos fantasmas apresentou uma sensível melhora ao serem efetuadas mudanças em algumas probabilidades, de forma que o fantasma já inicia o jogo "sabendo"algumas coisas. Sua movimentação ficou mais fluida, mais trechos do mapa foram explorados e, assim, os 3 fantasmas conseguiram complicar a vida do Pac-Man, conseguindo 5 vitórias nos 15 jogos disputados.

Apesar de ainda não ser a ideal, a movimentação apresentou sensível melhora, ainda mais se levarmos em conta que não há marcações pelo cenário para ajudar os fantasmas, os mesmos se guiam única e exclusivamente por suas visões e sensores.

7.2.2 Aprendizado

O fantasma apresentou um bom incremento no aprendizado porque venceu alguns jogos, ficou mais tempo vivo e teve, assim, mais experiências para computar. Algumas ações importantes, como avançar na direção da pac-dot ao vê-la tiveram um incremento considerável (15 % em 15 jogos, ao passo que outras, como ficar parado sobre uma pac-dot, aumentaram pouco (cerca de 5 %) mas, provavelmente, aumentariam mais conforme os fantasmas fosse mais treinados.

Dessa forma, o algoritmo parece ter sido mais bem sucedido nessa segunda implementação. Entretanto, há uma boa margem para melhoria e para uma intensificação da bateria de testes, para que resultados mais acintosos sejam obtidos.

7.3 Conclusão do Projeto

Por meio dos experimentos pode-se constatar que os algoritmos apresentaram comportamento bem diferentes, sendo o segundo uma evolução do primeiro. Ficou constatado também que, caso o fantasma já tenha um conhecimento prévio, sua tarefa de aprendizado torna-se muito mais fácil. Assim como com os seres humanos, foi mais fácil para o fantasma aprender à partir de algo que já conhecia. Por outro lado, o aprendizado à partir do zero não

rendeu muito bem.

Por fim, os frameworks utilizados mostraram-se de valiosos para o projeto, fornecendo ferramentas que pouparam muito tempo (como testadores de colisão ou scripts de conversão de formatos do Maya para formatos do Panda-3D). Apesar de muito divertida, porém, a tarefa foi bem trabalhosa, já que diversos aspectos da lógica do jogo devem ser observados.

8 Referências

Citações do projeto:

[1] Dbi Report: www.dbireport.com

[2] RUSSELL, Stuart ; NORVIG, Peter - “Artificial Intelligence: A Modern Approach”, Prentice Hall Series in Artificial Intelligence, 2ª Edição, 2003.

[3] Site Maya: http://usa.autodesk.com/adsk/servlet/pc/index?siteID=123112id=13577897

[4] Site Panda-3D: www.panda3d.org

[5] The Random Walk: www-math.mit.edu/phase2/UJM/vol1/RMONTE-F.PDF

Bibliografias utilizadas apenas para consulta:

[1] COSTA, Ernesto ; SIMÕES, Anabela - ”Inteligência Artificial - Fundamentos e Aplicações”, Editora de Informática, 2ª edição, 2008.

[2] Site Everything4Maya: www.everything4maya.com

[3] Midway Pac-Man Parts and Operating Manual. Chicago, Illinois: Midway Games, 1980.

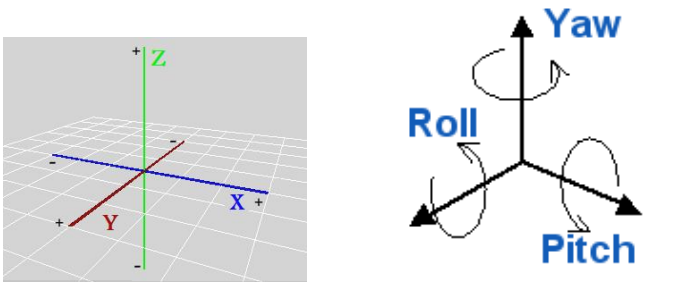
Parte II
Apêndice

A Conceitos do Panda-3D

A.1 Sistemas de Coordenadas

O Panda-3D utiliza o sistema XYZ de coordenadas cartesianas. Para a passagem de parâmetros do programa, deve-se usar os argumentos na ordem (x,y,z).

Já a medida de ângulos é feita usando o sistema de aviação HPR ou YPR (Yaw-Pitch-Roll) muito utilizado na aviação. Para a passagem de parâmetros do programa, deve-se utilizar os argumentos na Ordem (Y,P,R). Esse sistema pode ser algumas vezes um pouco difícil para quem nunca trabalhou com o HPR, dessa forma o Panda-3D também disponibiliza o uso de Quatérnios para trabalhar com medidas de ângulo.



(a) Orientação dos eixos (b) Eixos para os ângulos

Figura 8: Sistemas de coordenadas do Panda-3D

A.2 Grafo de Cena

O principal conceito utilizado pelo Panda-3D na composição de cenas é o Grafo de Cena. Qualquer objeto passa a ser visível para o renderizador somente se for inserido nessa árvore. A hierarquia de árvore torna o trabalho de posicionar objetos no cenário uma tarefa mais simples, já que o posicionamento de um objeto é relativo ao seu pai na árvore e qualquer mudança de renderização feita no pai propaga para os filhos. Assim, se o objetivo é, por exemplo, colocar um boné na cabeça de um personagem, basta inserir o

boné na árvore como filho do personagem e indicar que o boné é posicionado sempre no ponto (0,0,5), ou seja, 5 pixels acima do personagem. Assim, quando o personagem se mover, o boné será automaticamente reposicionado pelo grafo de cena. Esse conceito de hierarquia também é muito importante no sistema de colisões, que será explicado mais tarde.

Cada nó da árvore é um PandaNode, uma classe que é a superclasse de uma vasta gama de nodes: GeomNode, LightNode, CollisionNode, CameraNode. Um objeto da classe PandaNode é criado pelo Panda-3D toda vez que um arquivo EGG é consumido ou toda vez que é chamado um construtor de alguma das classes acima. É importante, aliás, não confundir essa classe com a NodePath, que contem um PandaNode mais um ID único para referência rápida e unicidade.

A.3 PandaNode e NodePath

Como já foi explanado, os NodePaths e PandaNodes são os nós que são inseridos no Grafo de Cena. Por meio deles, portanto, que são feitas todas as manipulações num modelo. Algumas funções dessas classes que merecem destaque são as funções para atribuir valores de escala, posicionamento e angulação ('setScale', 'setPos' e 'setHPR').

Outra função muito útil é a função 'lookAt', que faz com que um modelo seja rotacionado para que seu eixo Y passe a apontar na mesma direção que o eixo Y do qual ele deve "olha". Dessa forma, é possível, por exemplo, fazer com que alguns objetos passem a olhar para a câmera sem ter de calcular os ângulos HPR.

Existem também as funções 'hide' e 'show'. Essas funções, como o próprio nome sugere, permitem que os objetos passem a ser escondidos ou renderizados na cena. Com o auxílio das BitMasks, resultados muito interessantes podem ser obtidos, como permitir que apenas algumas câmeras "enxerguem"o modelo e outras não.

Há também o ActorNode, que é um nó que contem um ator, ou seja, um objeto com articulações ou juntas que se move. Normalmente vem acompanhado de animações e controle de cada um dos movimentos. Um jogador de futebol, por exemplo, é um ator.

A.4 Tarefas ('Tasks')

Tarefas ou 'Tasks' são funções especiais que são chamadas automaticamente pelo Panda-3D, frame por frame ou após um período delimitado pelo desenvolvedor. Para especificar que a função sendo definida é uma task, basta passar o parâmetro task na função, logo em seguida do argumento self.

Para especificar qual o tempo de atraso (chamado de 'delay') que uma task deve ter entre chamadas, basta atribuir um valor para o atributo 'task.delayTime'. Se o objetivo for saber há quanto tempo a Task está rodando, isso é guardado num outro atributo do objeto, 'task.time'.

Toda Task precisa retornar uma constante da classe Task, que indicará qual deve ser o comportamento subsequente da mesma. Existem quatro tipos básicos de retorno:

- Task.again: A Task será chamada no tempo de delay especificado.
- Task.cont: A Task será chamada no próximo frame.
- Task.done: A Task finalizou e não será chamada novamente.
- None: A Task finalizou e não será chamada novamente.

A.5 Câmera e Eventos de Teclado

A câmera é instanciada a partir de um objeto do tipo CameraNode, que normalmente será inserido na árvore como filho do render, ou seja, logo abaixo da raiz. Diversos tipos de configuração são possíveis para esse Node, como habilitar o mouse ou o botão de rolagem, distância mínima e máxima de alcance da lente. É possível também posicioná-la relativamente a um personagem e a câmera acompanhá-lo conforme se movimenta, como nos jogos em primeira pessoa (como o conhecido jogo Counter Strike).

Os eventos de teclado são gerados automaticamente pelo Panda-3D cada vez que uma tecla é apertada. Cabe ao desenvolvedor escolher quais eventos ele vai captar ou não. Cada evento captado pode ser ligado diretamente à uma função, que será chamada sempre que o evento for lançado pela engine.

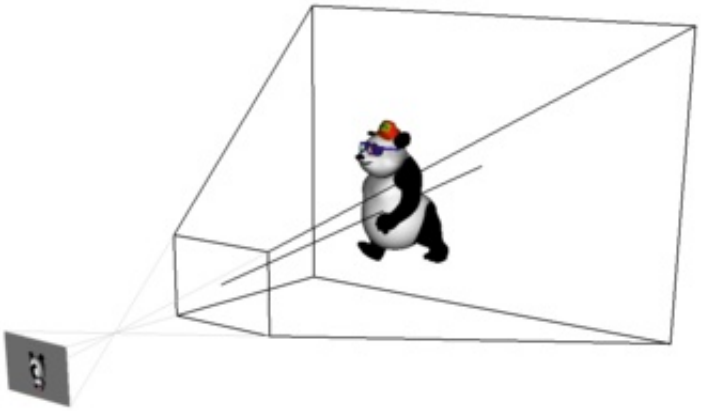


Figura 9: Câmera com lente em perspectiva do Panda-3D

A.6 Colisões

A engine do Panda-3D é muito completa e precisa no que diz respeito às colisões. Facilita muito o trabalho do desenvolvedor e disponibiliza diversos tipos de tratamentos e acessórios para as colisões.

A.6.1 Sólidos de Colisão

Para que uma personagem ou objeto colida com algum outro, é necessário que haja algum tipo de sinalização indicando ao Panda-3D que aquele objeto não é transpassável. Isso é feito envolvendo o objeto que deverá ter sua colisão testada pela engine num sólido de colisão. O sistema nomeia os objetos que causam a colisão como "from objects"e os que sofrem a colisão como "into objects".

- CollisionSphere
  Envolve o objeto com uma esfera, que delimitará até onde outros objetos podem chegar. Além de ser um objeto de uso muito comum, é o menos custoso para teste. Basta um cálculo de distância entre dois pontos seguido de uma comparação com o raio que já é possível saber se a colisão ocorreu ou não.

Figura 10: CollisionSphere

- CollisionTube
  É um cilindro circular. Seu teste de colisão já não é tão fácil quanto o da circunferência, sendo útil, por isso, evitar aplicá-lo em objetos que se movem.

- CollisionInvSphere
  Esse sólido é uma esfera invertida. Tudo que estiver FORA da esfera é considerado como colidindo com o objeto e tudo que estiver dentro da esfera não está colidindo. Esse sólido é útil para impedir que um objeto interno à esfera saia da mesma, pois sempre que atingir uma borda a colisão é automaticamente detectada.

- CollisionPlane
  É um plano infinito, que se estende em ambas as direções. Pode ser muito útil para construir um piso ou uma parede.

- CollisionPolygon
  É um polígono que pode ter formato baseado em triângulos e quadriláteros. Para construí-lo, basta informar quais pontos são seus vértices. Usar esse sólido, entretanto, acarreta um alto custo de performance no teste, sendo altamente desaconselhável seu uso como from object, ou seja, em objetos que se movam para colidir com outros.

- CollisionRay
  É uma semi-reta. É de grande utilidade para ser utilizado junto com a câmera, sendo possível assim saber quais objetos são intersectados por esse segmento e, dessa forma, estão aparecendo na câmera.

- CollisionLine
  É uma reta, ou seja, se estende infinitamente para ambos os lados.

- CollisionSegment
  É um segmento de reta, em que são especificados pontos de término.

Útil para, por exemplo, para constatar o alcance de um raio de visão ou da própria câmera.

- CollisionParabola
  É uma parábola de colisão. É de grande utilidade para tiro de projéteis, como balas de canhão, por exemplo.

A.6.2 Gerenciadores de Colisão

O Panda 3D permite que vários gerenciadores de colisão distintos sejam instanciados, permitindo tratar cada colisão de uma forma diferente. Os tipos disponíveis de gerenciadores de colisão são:

- CollisionHandlerQueue: A cada frame devolve uma fila com todas as colisões que aconteceram. É possível organizar por ordem cronológica e iterar sobre essa lista.

- CollisionHandlerEvent: Gera um evento assim que a colisão é detectada. Qualquer classe pode aceitar esse evento e indicar uma função que deva ser chamada ao ocorrer a colisão.

Existem três tipos básicos de eventos: 'in' (quando a colisão ocorre naquele frame), 'out' (quando a colisão deixa de ocorrer naquele frame) e 'again' (quando a colisão continuou ocorrendo naquele frame).

Para que o evento seja lançado, portanto, é necessário descrever ao handlerEvent que tipo de evento ele deve lançar. Isso é feito por meio do método addPattern. Assim, podemos ter:

```
handler.addInPattern(%fn-into-%in')
handler.addAgainPattern('%fn-again-%in')
handler.addOutPattern('%fn-out-%in')
```

Dessa forma, sempre que houver uma colisão, será lançado um evento do tipo 'nomeDoObjeto1-into-'nomeDoObjeto2'. Se algum objeto de alguma classe aceita esse evento, será imediatamente chamado.

- CollisionHandlerPusher: É um tipo mais sofisticado de gerenciador de colisão e herda todas as características do CollisionHandlerEvent. A grande diferença é que, além de lançar um evento, esse gerenciador automaticamente impede a progressão do objeto que colidiu. Esse handler

é muito útil para, por exemplo, fazer com que, ao bater num muro, um carro seja impedido de avançar mais ainda, entrando no muro, por mais que continue acelerando.

- PhysicsCollisionHandler: É talvez o tipo mais sofisticado de Handler presente no Panda-3D. Funciona de forma parecida com o Pusher, mas tem um cuidado especial com articulações e a forma com que a colisão ocorre. É indicado para ser usado com atores ou com objetos que se movem muito rápido ou de forma diferenciada, como fluidos ou projéteis.

- CollisionHandlerFloor: Gerenciador de colisão parecido com o Pusher, mas com design e adaptações exclusivas para como um chão funciona. Mesmo que o chão não esteja nivelado o CollisionHandlerFloor mantém o personagem sempre colado no chão, impedindo-o de flutuar ou de atravessar o chão. No caso de personagens com articulações, interage de forma perfeita com os mesmos, fazendo com que a queda pareça real, principalmente quando integrado ao PhysicsCollisionHandler.

A.6.3 Bitmask

Bitmask é um recurso muito interessante do Panda-3D no gerenciamento das colisões. Por meio dessas máscaras se torna possível estabelecer quais objetos devem colidir e quais não devem colidir. Cada nó de colisão tem duas bitmasks, uma para quando é um "in object"e outra para quando é um "from object"(o conceito de "in"e "from"object foi explicado no início desse capítulo).

Assim, se um objeto tem máscara de colisão para "in object"5 (em binário 101) e outro objecto tem máscara de colisão para "from object"3 (em binário 010) os dois objetos nunca colidirão, já que não há nenhuma casa na numeração binária na qual os dois números tenham um digito 1 em comum. Caso outro objeto tivesse máscara de colisão de "from object"7 (em binário 101), esses dois objetos seriam passíveis de colisão.

O recurso de Bitmask também é usado para que uma câmera "saiba"quais objetos deve enxergar, já que algumas vezes o objeto deve ser mostrado só por algumas câmeras.

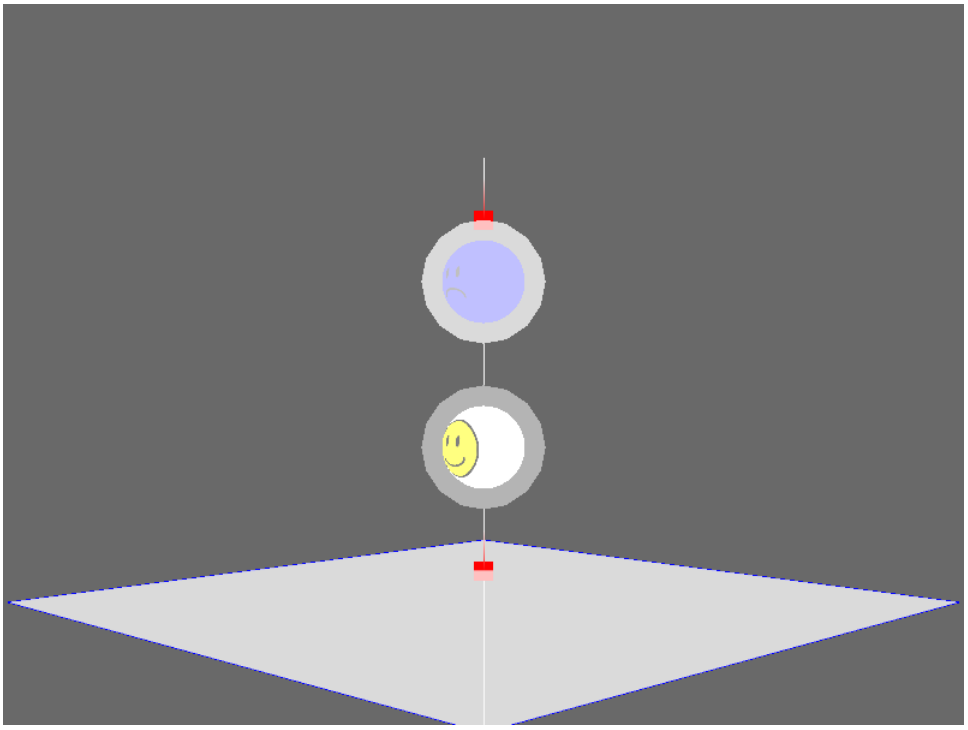


Figura 11: Colisão ignorada devido ao uso de bitmask

A.6.4 Collision Traverser

Collision Traverser é um tipo especial de verificador de colisão. Todos os CollisionHandlers instanciados devem ser associados a um CollisionTraverser. O papel do CollisionTraverser é, a cada frame ou chamada do método 'traverse' por parte do desenvolvedor, verificar todas as colisões que estão ocorrendo e as endereçar para o CollisionHandler adequado.

Para saber quais objetos ele deve checar e para quem endereçar, é necessário uma chamada ao método addCollider, como no exemplo abaixo (sendo 'traverser' um objeto do tipo CollisonTraverser):

traverser.addCollider(fromObject, handler)

Assim, a cada vez que o método 'traverse' for chamado, será checado se o fromObject está colidindo com alguém e, caso esteja, a colisão será informada ao handler, que decidirá o que deve ser feito.

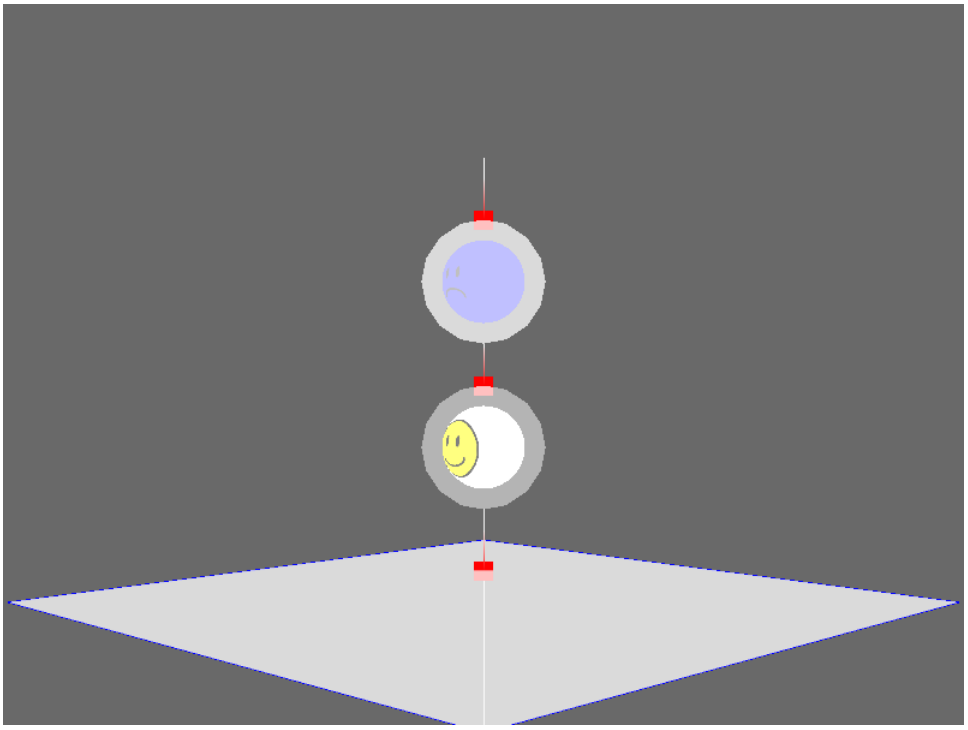


Figura 12: CollisionTraverser indicando quais colisões estão acontecendo

A.7 Classe Principal Do Panda-3D

A classe DirectObject é o coração do Panda-3D. Como variáveis dessa classe há um gerenciador de tasks ('taskMgr'), há um nó de câmera já instanciado('camera'), há a instância de um CollisionTraverser ('cTrav'), que é chamado automaticamente a cada frame e há o método 'run()', que inicializa o taskMgr e todas as outras constantes.

Parte III
Parte Subjetiva

1 Agradecimentos

Deixo registrado meus sinceros agradecimentos pela supervisão do Professor Marcelo Finger, que me atendeu por diversas vezes, mesmo em horários esdrúxulos e esteve presente durante todas as fases de desenvolvimento do projeto. Agradeço também à colaboração do Professor Flávio Soares Corrêa, que norteou o projeto em seu início e acompanhou seu desenvolvimento.

Por fim, agradeço aos membros do fórum do Panda-3D que, por mais de uma vez, foram fundamentais para me retirar de verdadeiros buracos durante o desenvolvimento do projeto.

2 Futuro do Projeto

Ainda há muito por fazer no desenvolvimento do projeto. Melhorias são possíveis em todas as áreas. Pode-se trabalhar num melhor desenvolvimento da parte gráfica, incluindo animações, efeitos de luz, sombra, anti-aliasing. O Maya pode ser muito mais explorado e resultados interessantes podem ser obtidos.

É possível ainda, é claro, modificar os algoritmos de inteligência artificial, deixando-os ainda mais efetivos ou até mesmo substituí-los totalmente, para testes e comparações. Dessa forma, há um campo aberto nesse sentido que pode ser aproveitado.

Outro ponto que pode ser corrigido é o fato de a posição inicial dos fantasmas e o número de pac-dots no mapa terem de ser definidas manualmente pelo programador. Sinalizar no mapa as posições de inicio e associar ao mapa um arquivo txt com informações essenciais também seria algo interessante de ser feito.

Assim, o projeto tem chances de ser continuado em diversas frentes e proporcionar a outras pessoas a possibilidade de melhorar o projeto, o que é interessante visto que o trabalho não é só um trabalho que morrerá após a conclusão da matéria e, sim, um legado que a faculdade terá à disposição para colaborar na formação de outros alunos.

3 Desafios, Vitórias e Frustrações

Uma das fases que acredito ser a mais crítica de um TCC é justamente a fase inicial, na qual se escolhe qual área da computação será escolhida para desenvolvimento de um projeto e, mais ainda, qual projeto será desenvolvido. Por já ter trabalhado numa iniciação científica por mais de um ano com o Professor Marcelo Finger, já sabia que escolheria a área de inteligência artificial. Escolher qual projeto seria desenvolvido, entretanto, exigiu uma reflexão mais aprofundada.

Escolhi que desenvolveria um jogo por três motivos: Uma das razões de ter me interessado por computação era saber como alguém fazia para tornar jogadores de futebol, terroristas, pilotos de jogos de videogame seres "inteligentes", com os quais os humanos podiam interagir. Levei em conta também que o projeto era longo e, portanto, gostaria de me divertir durante seu desenvolvimento. Nada melhor que um jogo para isso. Por fim, poderia ter um campo muito interessante na área de inteligência artificial, desenvolvendo algoritmos.

Assim, estava feita a escolha. O professor Flávio indicou os Frameworks para desenvolvimento do jogo e para modelagem 3D. Teria de aprender a linguagem Python e como trabalhar com esses frameworks. Comecei pela linguagem Python e não tive muitas dificuldades em assimilar os conceitos básicos. E seguida, passei à modelagem 3D e enfrentei os primeiros problemas.

O Maya (indicado pelo Panda-3D para modelagem) não era gratuito e sua versão paga era muito cara. Esse problema foi contornado por meio de uma versão Trial (que expiraria depois de 30 dias em cada computador instalado). Resolvido esse problema, havia outro maior: como fazer a modelagem 3D em um programa poderoso e cheio de opções como Maya sem nunca ter tido alguma experiência na área?

Após muitos tutoriais e tentativas, consegui um resultado satisfatório. Uma modelagem leve e bonita, o suficiente para continuar o desenvolvimento. Gostaria, entretanto, de ter mais tempo para desenvolver de forma mais avançada a modelagem, incluindo no projeto recursos como animações, efeitos de luzes e anti-aliasing. Mas, de qualquer forma, essa dificuldade estava superada.

O próximo passo foi mexer no Panda-3D. O framework me deixou muito

animado para a sequência do projeto. Parecia simples, poderoso (foi adotado inclusive pela Disney). Achei que o trabalho andaria bem rápido. Após alguns tutoriais, me senti pronto para começar.

Entretanto, por ser tão poderoso, o framework exige um conhecimento profundo de como funciona internamente, desde como configurar um arquivo para leitura de cores até como fazer o gerenciador de colisões detectar as barreiras indicadas nos mapas. Apesar de muito boa e extensa, seria muito difícil a documentação cobrir todos os aspectos desse framework. O fórum do Panda-3D, entretanto, foi mais do que suficiente para resolver minhas dúvidas.

Dessa forma fui desenvolvendo toda a lógica do projeto. Quando tudo já estava tratado (colisões, modelagem, lógica), implementei os algoritmos para teste. Vê-los funcionando foi, também uma vitória. Na primeira tentativa de inteligência artificial, entretanto, com as probabilidades não-enviesadas, vi que seria difícil chegar num resultado convincente.

Com o viés de probabilidades e mais algumas mudanças, pude chegar num resultado bem satisfatório. Fiquei muito satisfeito com o andamento do projeto, apesar de ter desejado torná-lo ainda melhor tanto na parte de inteligência artificial quanto na parte de computação gráfica.

4 Matérias Aplicadas Durante o Projeto

Entre todas as matérias cursadas ao longo dos 4 anos de graduação, algumas se mostraram especialmente úteis durante o desenrolar do projeto. Assim, listo aquelas que tiveram aplicação direta com o projeto:

- Introdução à Computação - MAC0110, Princípios de Desenvolvimento de Algoritmos - MAC0122 e Estrutura de Dados - MAC0323

  Inclusão um pouco óbvia, mas necessária. Não há como programar e fazer um projeto sem essas disciplinas devidamente assimiladas. Desde as bases da computação como "o que é um laço"até o estudo de estruturas de dados mais complexas (como grafos) e sua manipulação, tudo foi utilizado nesse projeto.

- Introdução à Computação Gráfica - MAC0420

  Fundamental para a modelagem 3D. Conceitos como normais de uma imagem, lentes em perspectiva e até mesmo o modelo de Phong de

coloração foram aplicados durante o desenvolvimento. Definitivamente uma matéria que foi fundamental para o trabalho.

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

  Essas duas matérias foram fundamentais já que, em um jogo, muitas coisas acontecem em paralelo e, ao menor sinal de lag na interface gráfica ou lentidão do fantasma na movimentação, perde-se muito em termos de realismo e o usuário passa a ter uma experiência desagradável. Dessa forma, o conceito de Threads e Tasks e seu gerenciamento foram importantes para que os algoritmos rodassem ao mesmo tempo.

- Engenharia de Software - MAC0332

  Inclusão obrigatória para todos aqueles que desenvolveram um projeto orientado à objetos. Conceitos como encapsulamento, herança, acoplamento, composição se tornam fundamentais. O estudo, mesmo que pouco aprofundado, de técnicas como Scrum e XP também ajudou, na medida em que podemos aplicar algumas das boas práticas aplicadas nessas matérias.

- Laboratório de Programação Extrema - MAC0342

  Apesar de não aplicar XP diretamente no projeto, algumas boas práticas puderam ser adotadas, como evitar a repetição de código por meio da refatoração e a tentativa de evitar nomes de métodos e classes confusos e ambíguos.

- Introdução à Inteligência Artificial - MAC0425

  Definitivamente uma matéria importante para o projeto. Muitos dos conceitos básicos implementados e toda a fundamentação para o desenvolvimento dos algoritmos foram obtidos por meio dessa matéria. O primeiro contato com Inteligência Artificial ocorreu nessa matéria.