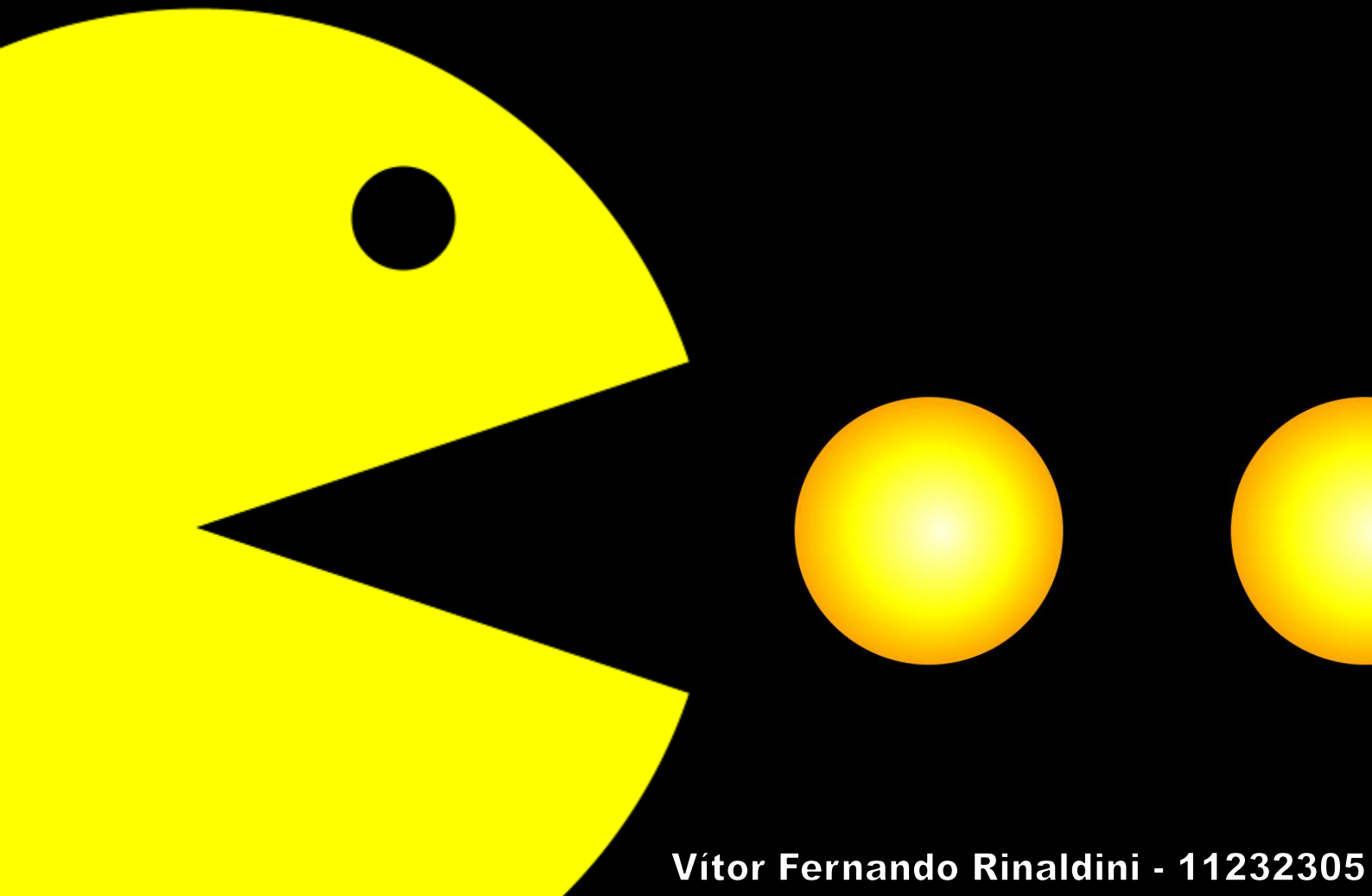
PAC MAN

- Manual do Sistema



Vítor Fernando Rinaldini - 11232305

PAC MAN

Manual do Sistema

- Versão 1.0

Vítor Fernando Rinaldini - 11232305

1. Sumário

2. Motor Gráfico

a. O Game Loop

Para a construção do motor gráfico foi necessário a utilização de uma Thread que ficaria atualizando o mapa e pintando a tela a uma frequência que gira em torno de 60 Hz (Quadros/segundo). Aqui tive meu primeiro problema: Ao utilizar uma Thread simples do Java, precebi que não havia forma de sincronizar o método 'run' com a taxa de atualização da tela. Isso causava travamento e lentidão consideráveis. Após pesquisar bastante, vi que o JavaFX possuía a sua própria thread que era perfeitamente sincronizada: a "Timeline" e foi através dela que implementei o loop principal do programa.

b. Pintando entidades

Não irei explicar detalhadamente como fiz cada entidade gráfica. Caso o leitor se interesse, poderá consultar o método "Draw(GraphicsContext g)" disponível em cada classe desenhável do jogo.

De uma forma geral, a tela é composta por um Canvas da biblioteca JavaFX. O loop principal extraí um objeto do tipo "GraphicsContext" através do método "getGraphics2D()" e com ele desenha todas as formas. Vale a pena o leitor consultar um material externo sobre esse pacote, pois há inúmeras possibilidades de formas geométricas e de materiais de pintura disponíveis. Alguns desses métodos são pesados e utilizam bastante memória, mas de uma forma geral, foram suficientes para representar as entidades desse game.

Vale salientar que praticamente todas as entidades possuem alguma animação. A mais fácil de visualizar é a do Pac-Man abrindo e fechando a boca. Elas foram implementadas com um ângulo theta ou com um contador que são sempre atualizados/verificados e até resetados na chamada do método update. Na hora de fazermos o desenho, usei uma função trigonométrica e multipliquei a devida coordenada para conseguir o devido resultado.

c. Principais problemas

Aqui eu tive dois problemas que serão mostrados adiante.

Caso o leitor analise a pasta do game, verá que não temos uma imagem guardando o conteúdo do tabuleiro (Que facilitaria muito o processo de draw). Visando num futuro fazer uma animação e tornar o método mais dinâmico para isso, fiz um método que pinta o tabuleiro em tempo real. À primeira vista, uma boa ideia, mas quando fui fazer os primeiros testes, vi que o mesmo estava muito pesado e o jogo se tornou uma “apresentação de slides” no quesito FPS. Portanto, para corrigir, decidi desenhar o tabuleiro em uma Thread separada (utilizando um Canvas secundário e o seu método snapchat) antes do inicio de cada fase. Isso resultou na tela de carregamento. Ou seja, enquanto uma linha de comando secundária está desenhando o mapa e configurando a fase, o loop principal exibe a tela de carregando. Como pode ver, isso corrigiu o problema.

Mais tarde quando estava perto da etapa final do projeto, fiz um arquivo ”jar” e pedi para o meu amigo testar. Foi um teste interessante, pois a priori o game não estava abrindo e quando pedi pra ele executar pelo prompt, vi que estava dando “OutOfMemoryError”. O programa funcionava, mas estava mal otimizado. Percebi que a cada instancia de um elemento do tabuleiro, ele criava várias “ImagePattern” para cada elemento. Esse obejeto seria utilizado para criar texturas que seriam utilizadas nas animações do objeto. Bom, a solução aqui foi criar uma classe TexturasGraficas que instaciaria uma única vez essas texturas e passa-la para cada elemento de mapa, já que as texturas são as mesmas indiferente o elemento. Isso resultou numa leve melhora na velocidade e numa otimização de memória significativa. O programa passou de 1GB de pico de Memória RAM para 20mb no máximo. Com isso, o problema foi corrigido

d. Pintando cada fase

De forma resumida, a fase se comporta como um objeto gráfico: possui um método draw, outro update, mas diferente deles, ela possui

elementos gráficos secundários: as entidades da Cena, o menu de Pause, as telas de Vitória e de Game Over, etc).

3. Controles

Foi criado uma Classe chamada InputKeyController (Famoso teclado) que é composta por varias chaves booleanas (True/False) que serve apenas para pegar o estado de uma determinada tecla do teclado (Se ela está apertada/ foi solta) ou de um determinado comando (útil para o menu ou para a fase). Essa classe deve ser instanciada no momento de criação da Janela para que possa ser setada por ela. Após isso, ela deve ser passada adiante para os componentes do GameLoop; Os controles estão explicados no Manual do Usuário.

4. Entidades

O pacote de entidades agrupa todos os objetos livres e que possuem algum tipo de interação com o jogador. Aqui está escrito o código do Pac-Man, dos diversos tipos de Fantasmas e dos elementos de Pontuação (Pac-Dots, Pílula e Frutinhas).

Esse projeto teve a entrega de duas partes. Na parte 1, o game funcionava apenas com inteiros (i, j) que era do tamanho de elementos do mapa. Esse espaço vetorial funcionava bem para a impressão no terminal. Quando tive que implementar a interface gráfica, passei pelo problema de estar trabalhando com dois espaços vetoriais ao mesmo tempo. Um onde foi desenvolvida toda e Engine e Inteligência Artificial e no outro responsável por desenhar os elementos na tela. Para resolver isso sem bagunçar uma parte considerável do programa, decidi criar métodos que convertessem de um espaço para outro. Assim, quando iríamos desenhar, o método passava as coordenadas para X e Y (Pixels) e quando iríamos usar alguma lógica, transformava as coordenadas em i e j.

5. Inteligência Artificial

a. Mover entidade

Para mover uma entidade, seja ela NPC ou jogador, eu decidi calcular primeiramente a distancia total que ele deve se mover naquele turno (Obviamente, recebendo a diferença de tempo para realizar esse calculo) e movia na devida direção. O método de mover confere se há uma parede no caminho ou se está tudo em ordem para a movimentação e, em seguida, executa essa ação. O NPC exclusivamente, antes de se movimentar, ele divide a distância que ele deve percorrer em quanto falta pra ele chegar até o destino e já se move. Após isso, ele calcula um novo destino e termina de se movimentar. Vale salientar que se a distancia na qual deve percorrer não for suficiente para chegar ao destino, ele simplesmente se move toda essa distancia e encerra o turno;

b. Fantasma Esperto: próximo destino

Com a ajuda de um grafo, o fantasma usa o método de **busca em largura** para encontrar o jogador e calcular o menor caminho até ele.Com isso, ele vai em direção ao próximo nó desse caminho. Quando ele chegar no próximo nó, ele realiza novamente a busca. Com isso, ele se atualiza com uma frequência relativamente boa de vezes e nunca se distancia do jogador. Vale salientar que se ele não encontrar o jogador, ele realizará uma busca aleatória.

c. Fantasma Aleatório: próximo destino

O fantasma aleatório se move como o próprio nome diz, aleatoriamente. Quando ele chega no Node destino do grafo, ele sorteia um outro Node aleatório para seguir.

d. Fantasma: fugir

A ideia aqui é calcular qual é o Node que deixaria o fantasma mais perto do jogador e seguir para um Node diferente desse. Afinal, esse é o conceito de fugir.

e. Fantasma: voltar ao Spawn

Quando ele morre, ele sorteia um lugar aleatória do Spawn, calcula o menor caminho até ele e segue o caminho completo até lá.

f. Curiosidades do grafo

Para construir o grafo, primeiramente ele varre o mapa todo e adiciona um Node para todo ponto que for caminho. Em seguida, ele liga os vizinhos separados por 1 unidade seja na vertical ou na horizontal. Para finalizar, ele remove Nodes que servem como Ponte, deixando apenas aqueles que conectam mais de dois Nodes ou que são pontos de curva.

Outro fato interessante é que os Nodes do Spawn ficam separados do Nodes do tabuleiro e através de uma variável booleana, ligamos/desligamos eles (Na localização de uma porteira obviamente). Os métodos responsáveis por fazer isso são os de “abrirPorteira”/”fecharPorteira”.

6. Engine Geral

O programa conta com um objeto Player, responsável por guardar informações que atravessam fases (Score, Vidas, etc). Há também um objeto Game que guarda a textura atual e o objeto de controle do teclado. Esses objetos são as primeiras coisas instanciadas. Após isso, ele inicia o loop e carrega o Menu.

Ao selecionar Novo Jogo, ele muda para a tela de carregamento, carrega o cenário do Nível 1 e inicia o jogo. Se o jogador ganhar, ele parte para o próximo nível. Se ele perder, ele retorna ao Menu principal.