UNIVERSIDADE FEDERAL DE MINAS GERAIS
DEPARTAMENTO DE CIÊNCIA DA COMPUTAÇÃO

PROGRAMAÇÃO MODULAR

Trabalho Final

PACMAN em Java

Alunos: Deiziane Silva
Dourival Pimentel
Nélio Sampaio
Samuel Oliveira

Prof.: Douglas Macharet

Belo Horizonte
18 de junho de 2017

1. Introdução

Pacman é um jogo eletrônico criado pelo Tōru Iwatani para a empresa Namco, e sendo distribuído para o mercado estadunidense pela Midway Games. Ele foi produzido originalmente para Arcade no início dos anos 1980, e como se tornou um dos jogos mais populares no momento, acabou tendo versões para diversos consoles e continuações para tantos outros, inclusive na atualidade.

O Pacman é uma bola redonda amarela com uma boca que abre e fecha, posicionado em um labirinto simples repleto de pastilhas e 4 fantasmas que o perseguem. A mecânica do game é simples: o Pacman come pastilhas fugindo de fantasmas em um labirinto. O objetivo é comer todas as pastilhas sem ser alcançado pelos fantasmas, aumentando progressivamente a dificuldade. É possível andar para todas as direções do labirinto, tem a cereja (bônus), e ainda as pílulas de poder, que permitem que o Pacman coma os fantasmas.

Nosso trabalho visa a implementação desse jogo utilizando a linguagem de programação Java e busca exercitar os conceitos aprendidos em sala sobre orientação a objetos e padrões de projetos.

2. Implementação

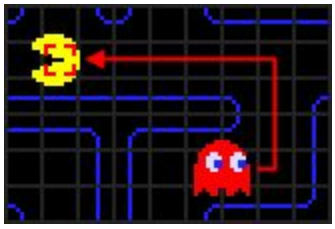
2.1. Inteligência Artificial

Comportamento dos fantasmas

Temos quatro fantasmas no jogo. Esses personagens são implementados de acordo com as características do jogo original. Cada fantasma tem uma “personalidade”, uma função no jogo.

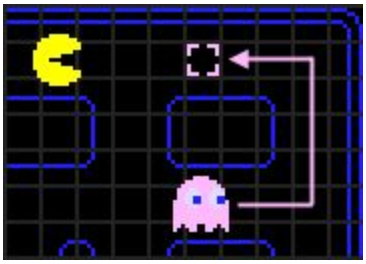
- Blinky (Vermelho) - É o principal fantasma do jogo, que constrói maior dificuldade. Começa o nível em uma velocidade constante igual dos outros fantasmas do jogo, porém após o Pacman devorar certa quantidade de

comida, ele começa a aumentar sua velocidade. Sempre escolhe o caminho mais curto até o Pacman.



Blinky sempre irá atrás do Pacman

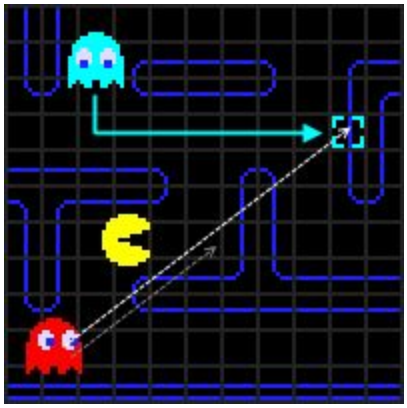
- Pinky (Rosa) - Ele está programado para sempre se posicionar na frente da boca do Pacman. Em nossa implementação, ele estará sempre perseguindo um ponto 4*16 (16 é o tamanho de cada “pedaço” do mapa) a frente do Pacman. Seu esquema de perseguição é tentar se mover para o lugar onde Pacman está indo, em vez de onde ele está atualmente. Sempre mantém sua aceleração.



Pinky sempre irá na frente do Pacman

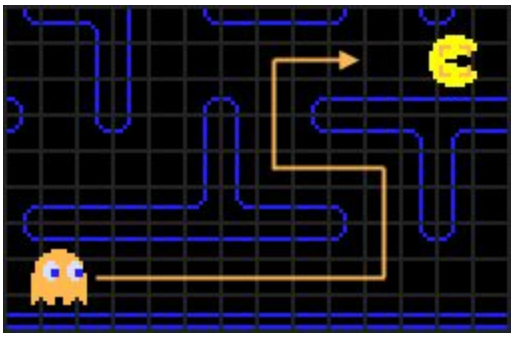
- Inky (Azul) - Inky é difícil de prever, pois ele é o único dos fantasmas que usa um fator diferente da posição do Pacman ao determinar seu destino. Além da posição do Pacman, ele também é controlado com a proximidade do Blinky. Caso Blinky esteja longe dele, seu comportamento é inesperado. Porém quando ele está perto, Inky começará a “imitar” seu comportamento. Para localizar o alvo da Inky, primeiro começamos selecionando a posição dois pedaços (*Chunks/Tiles)* na frente do Pacman, semelhante ao método de perseguição de Pinky. A partir daí, podemos imaginar um vetor da posição de Blinky para este *chunk/tile*, e depois dobrar o comprimento do

vetor. O novo ponto em que esse vetor novo atinge acabará será o alvo real da Inky.



Alvo de Inky, leva em consideração a posição de Binky

- Clyde (Laranja) - Possui dois modos distintos e alterna entre eles dependendo de sua posição em relação ao Pacman. Se estiver numa distância suficiente longe do Pacman, ele vai caçá-lo. Caso esteja perto, seu alvo é o mesmo que quando ele está em modo dispersar (SCATTER). Ou seja, ele vai ir em direção a sua “Home-corner”, um ponto inferior à esquerda do labirinto.



Longe do Pacman, persegue-o.

Perto do Pacman, vai para home-corner

Modos

Ao decorrer do jogo, os fantasmas também alternam entre modos. Esses modos são: perseguir (CHASE) assustado (FRIGHTENED) e dispersar (SCATTER), além de mais dois modos auxiliares, que são o piscando (BLINKING) e o retornando (RETURNING). O modo "padrão" dos fantasmas que perseguem o Pacman é o *Chase,* e este é o único que eles passam a maior parte do tempo. Enquanto no modo *Chase*, todos os fantasmas usam a posição do Pacman como fator na seleção do ponto alvo, embora seja mais significativo para alguns fantasmas do que outros, como discutido anteriormente. No modo *Scatter*, cada fantasma possui uma ponto alvo fixo, cada um dos quais estão localizados em um canto diferente do labirinto. Isso faz com que os quatro fantasmas se dispersem pelos quatro cantos sempre que estiverem nesse modo. O modo *frightened* é o único em que os fantasmas não possuem um ponto de destino específico. Em vez disso, eles pseudo randomicamente decidem qual caminho tomar a cada movimento.Um fantasma no modo Frightened também fica azul, se move muito mais devagar e pode ser comido pelo Pacman. Ao ser comido, esse fantasma volta para a gaiola.

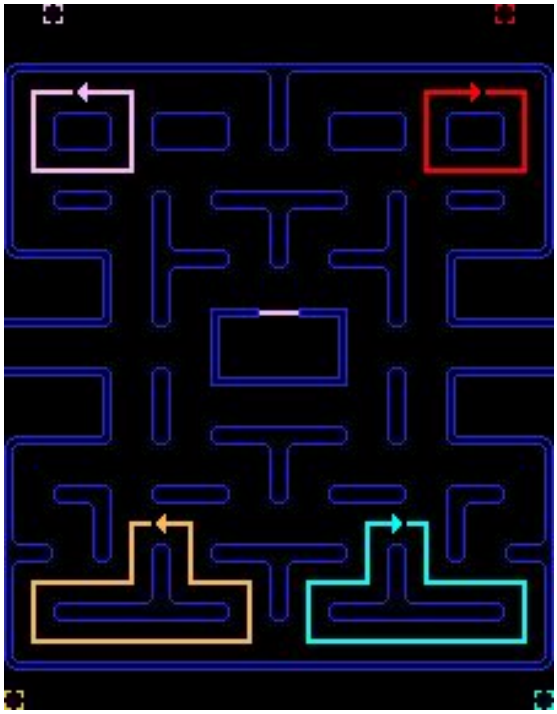
As alternações entre os modos *Chase* e *Scatte*r ocorrem em um tempo fixo através de um temporizador. Este temporizador é reiniciado no início da fase e sempre que uma vida é perdida. O temporizador fica pausado enquanto os fantasmas estão no modo *Frightened*, que ocorre sempre que o Pacman come um energizador (pílula de poder). Quando o modo *Frightened* termina, os fantasmas retornam ao modo anterior e o cronômetro retoma a contagem. Os

fantasmas começam no modo *Scatter*, e há quatro ondas de alternância *Scatter / Chase* definidas, após as quais os fantasmas permanecerão no modo Chase indefinidamente (até que o temporizador seja reiniciado). As durações dessas fases são:

- Dispersar (*Scatter*) por 7 segundos, então perseguir (*Chase)* por 20 segundos.
- Dispersar (*Scatter*) por 7 segundos, então perseguir (*Chase)* por 20 segundos.
- Dispersar (*Scatter*) por 5 segundos, então perseguir (*Chase)* por 20 segundos.
- Dispersar (*Scatter*) por 5 segundos, depois mude para o modo perseguir (*Chase)* de forma permanente.

Home-corners

Como citado acima, cada fantasma possui uma "*home-corner".* Esse é o nome dado a um ponto **fora do mapa** para onde cada fantasma irá se mover quando o modo *Scatter* (dispersar) começa. No entanto, uma vez que esses pontos são inacessíveis e os fantasmas não conseguem parar de mover ou inverter a direção, eles são obrigados a continuar seu caminho, mas tentarão ir para a *home-corner* o mais rápido possível. Isso resulta num movimento circular onde toda vez o fantasma tentará atingir o ponto, mas não conseguirá. Se deixado no modo *Scatter*, cada fantasma permaneceria em seu loop indefinidamente. Na prática, a duração do modo Scatter é sempre bastante curta, de modo que os fantasmas muitas vezes não têm tempo para chegar ao seu canto ou completar o circuito do loop antes de retornar ao modo *Chase*. As *home-corners* de cada fantasma são mostradas na imagem abaixo:



2.2. Classes Importantes

Main

A classe main possui um objeto JFrame que é a interface gráfica do jogo, além inserir os valores iniciais do jogo, inicia o loop de controle, as imagens e a música do jogo.

GameLoop

Essa classe controla o loop principal do jogo e controla todos os eventos relacionados. Essa classe possui uma thread que executa o jogo checando seus estados, A classe possui uma instância da classe GameCanvas que é responsável por controlar a interface gráfica do jogo e também uma referência a um objeto da classe ChunkedMap em um objeto da classe Map que contém todas as informações do mapa do jogo.

GameState

A classe guarda informações sobre o estado atual do jogo como pontos e vidas. A classe tem múltiplos listeners, que vão notificar os observadores. O FoodListener

será notificado quando qualquer tipo de "comida" for pega pelo Pacman (inclui pontos, bolas(poderes) ou frutas bônus). O StateListener será notificado se houver uma mudança de estado.

ChunkedMap

Essa classe implementa as interfaces Map, RenderEvent, StateListener e FoodListener, e sua principal responsabilidade é controlar todas as informações relativas ao mapa do jogo. O mapa do jogo é um labirinto composto por diversos elementos e é representado por uma matriz de números, onde cada número representa:

- 0 - Nada (espaço vazio onde os personagens podem caminhar)
- 1 - Bloco
- 2 - Comida
- 3 - Pilula de poder
- 4 - Jumper (nesse espaço, os personagens podem atravessar o mapa de um lado para o outro)
- 9 - Ponto de início

Actor

Essa classe abstrata é usada para representar os personagens do jogo, o Pacman - classe Pacman - e os demais fantasmas - classe Ghost - estes por sua vez, derivam-se de Actor. Essa classe implementa as interfaces MovementEvent, RenderEvent e StateListener. A interface MovementEvent em especial possui o método move que é sobrescrito de maneira distinta por cada personagem do jogo.

Pacman

Uma das principais classes do jogo é responsável por controlar o personagem principal do jogo o Pacman. É responsável por controlar eventos de colisão do Pacman com o mapa e também de interpretar os comandos do usuário. É uma classe filha de Actor e implementa as interfaces CollisionEvent e InputEvent.

Ghost

A classe Ghost é uma classe abstrata e implementa a interface CollisionEvent.
Essa classe é responsável pelo controle dos personagens que são fantasmas. A classe controla a velocidade(NORMAL, SLOW e FAST) e os eventos dependendo

do modo que os fantasmas se encontram(CHASE, SCATTER, FRIGHTENED, RETURNING, BLINKING). A classe também Inclui informações sobre onde o alvo (Pacman) está localizado.

Cage

Gaiola onde os fantasmas iniciam, esta classe gerencia seus estados e mudança de modos. Estende Actor e implementa a interface FoodListener. Ao iniciar o jogo, os fantasmas não são imediatamente atacados por todos os quatro fantasmas. Apenas um fantasma começa no labirinto atual, enquanto os outros estão dentro de uma pequena área no meio do labirinto, muitas vezes referido por nós como “gaiola”. Além do início de um nível, os fantasmas só retornarão a essa área se forem comidos por um Pacman energético, ou como resultado de suas posições serem reiniciadas quando o Pacman morre. A gaiola é, de outro modo, inacessível, e não é uma área válida para Pacman ou os fantasmas se movimentarem. A gaiola libera os fantasmas, cada um em seu tempo específico, sendo a primeira a Pinky, em seguida o Inky e, por último o Clyde. O único fantasma que não começa dentro da gaiola é o Blinky.

2.3. Princípios e Padrões de Projeto

Os detalhes abordados nesta seção são mais fáceis de ser vistos através do Diagrama de Classes (ANEXO A). No entanto, alguns recortes do diagrama ainda podem ilustrar esta seção.
Primeiramente, discutiremos a necessidade de usar e a atenção dada ao Princípio da Inversão de Dependência e o Princípio da Segregação de Interfaces [1]. Logo em seguida, falaremos dos Padrões de Projeto [2] utilizados e das vantagens adquiridas com cada um.

Princípio de Inversão de Dependência

Com o intuito de evitar rigidez e fragilidade, e não comprometer todo o projeto no momento de uma alteração futura, alguns pacotes como map e sound são usados para fornecer serviços às outras classes e não dependem de muitos outros, enquanto pacotes como main e character não são usados por ninguém, pois suas várias dependências os deixam instáveis e continuamente sujeitos a alterações.

Ainda assim, dentro de pacotes considerados estáveis, algumas classes mais genéricas (e frequentemente abstratas) podem ser usadas como base para construir uma outra classe mais específica, criando dependências seguras entre as classes dentro de um pacote. Como exemplo no pacote map, a Classe ChunkedMap que depende internamente apenas da Classe Point, que não depende de ninguém.

Princípio da Segregação de Interfaces

Para facilitar a legibilidade do código e evitar a repetição desnecessária, é recomendável que interfaces sejam o mais simples possível. Dessa forma, as classes que implementam uma interface não serão obrigadas a declarar métodos que elas não querem (ou devem) ter.
Esse princípio é bastante usado pelo pacote swing de java, mas além dele, outras pacotes que tiveram seu design orientado a eventos, como o pacote game e o pacote map, também contam com interfaces diversas e que são usadas por vários pacotes sem desperdício de código.

Padrões de Projeto - Singleton

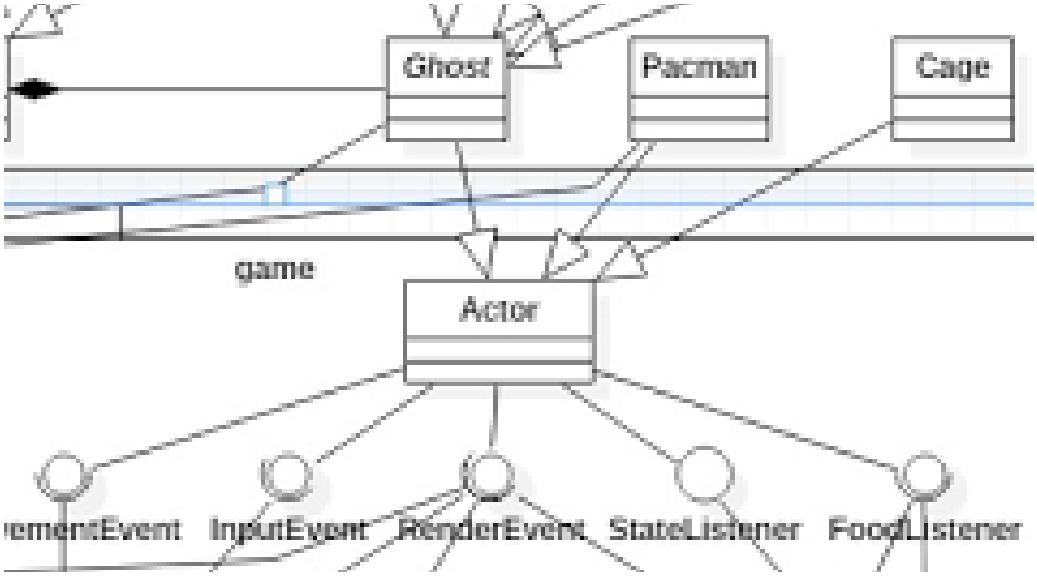
Algumas classes e responsabilidades são desenhadas pensando que apenas um objeto seja capaz de assumi-las. O papel de um líder em uma equipe é um bom exemplo de responsabilidade que deve ser única. Na Orientação a Objetos, tais tipos de responsabilidades podem ser protegidas com o Padrão Singleton, que além de garantir que apenas um objeto da classe seja instanciado, pode também oferecer uma interface que centralize o uso a esse objeto, tornando-o acessível com muita clareza.
No nosso projeto, algumas classes precisam dessa propriedade. E nesse caso a implementação foi feita usando enum, que enumerava sempre um número finito de instâncias e que podiam ser acessadas de fora através de NOME_DA_INSTANCIA.CLASSE. Como exemplo temos GameLoop e sua INSTANCE, e GameState e sua INSTANCE. Veja a implementação na figura abaixo.

```
public enum GameState implements RenderEvent, StateListener {
    INSTANCE;
    ...
}
```

Padrões de Projeto - Facade

Quando muitas classes externas dependem de recursos internos a um pacote para funcionar, o número de dependências pode dificultar a legibilidade do código e gerar uma certa rigidez e imobilidade. Uma alternativa para isso é fazer uma “Fachada” ou “Façade” para centralizar as chamadas externas a um pacote e, portanto, gerar um único lugar que possa ser alterado quando for necessário alterar as classes internas dos dois pacotes.

Nesse projeto, isso se fez necessário para simplificar o acesso às diversas interfaces do pacote game. Isso pôde ser feito através da Classe Actor que implementa todas as interfaces que podem ser úteis as classes do pacote character e que será estendida pelas classes do mesmo. Como pode ser visto na figura abaixo:



3. Conclusão

A programação modular tem como objetivo produzir um código que seja viável, não só computacionalmente, mas também economicamente e do ponto de vista de uma engenharia de produção. Modularizar pode ser sinônimo para simplificar, economizar e proteger, se feito da forma correta, e era isso que esperávamos desenvolver ao aplicar os conceitos de Programação Modular e Programação Orientada a Objetos em um projeto um pouco maior e mais realista.

Entretanto, nem sempre "quanto mais conceitos, melhor", mesmo que aplicados da forma correta, pois todo design tem um custo e nem sempre uma modularização perfeita é tangível para um prazo e para um orçamento. Nesse projeto, tivemos a oportunidade de aplicar Padrões de Projeto como o Builder, o Proxy ou o MVC, que teriam melhorado ainda mais a qualidade de código e sua manutenção futura e não implementamos. O custo seria alto em comparação aos benefícios e isso deve ser sempre avaliado.

Por último, é importante mencionar a experiência do usuário, pela interface, pela performance ou pelo fluxo do programa. Esse foi o primeiro projeto com interface, e por ser mais longo, o primeiro cuja performance se tornou relevante. Percebemos que interface é custosa, verbosa e demanda mais do que simples programação. Precisamos de banco de imagens para ilustrar a janela, percebemos que se a resposta do programa aos inputs do usuário for lenta, a experiência do jogo pode não ser boa e sentimos dificuldade ao inserir tantas linhas de código apenas para lidar com a interface. No entanto, sem se preocupar com isso, não haveria usuário, não haveria jogo e não haveria projeto. A experiência do usuário é tão parte do projeto quanto o código em sí.

4. Referências

[1] - https://robsoncastilho.com.br/2013/04/14/principios-solid-principio-da-segregacao-de-interface-isp/ acessado em 17 de junho de 2017.

[2] - *Design Patterns: Elements of Reusable Object-Oriented Software* by ErichGamma, RichardHelm, RalphJohnson, and JohnVlissides (the GangOfFour)

[3] - https://pt.wikipedia.org/wiki/Pac-Man

[4] - http://papeldiario.blogspot.com.br/2013/06/tutorial-java-pac-man-preparaca

o.html
[5] - http://gameinternals.com/post/2072558330/understanding-pac-man-ghost-behavior