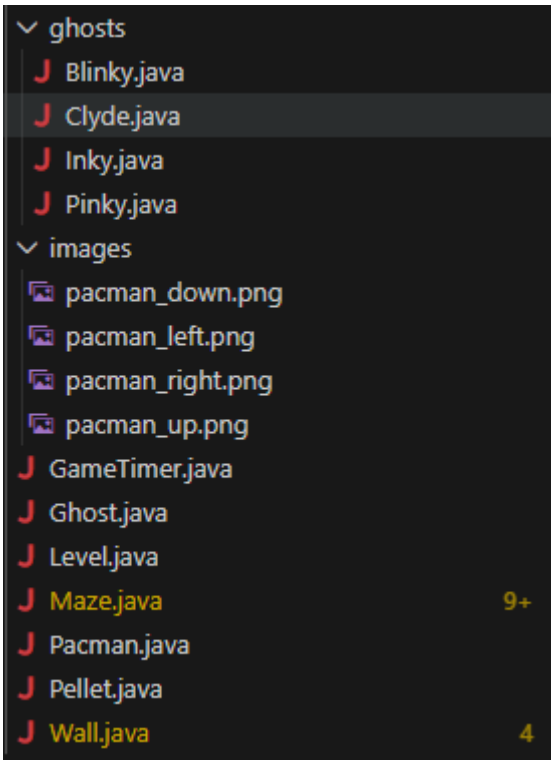
Técnicas de Construção de Programas

Trabalho Prático - Etapa 2

RELATÓRIO

Implementação



Estrutura de arquivos em /src

Em relação ao diagrama de classes da Etapa 1, a Hierarquia foi, em grande parte - pertinente às classes **públicas** *Pellet*, *Wall*, *Ghost*, *Pacman* e *GameTimer*, abordada através de serem instanciadas por *Maze*, a classe que serve como controle de fluxo do ciclo de jogo.

Maze

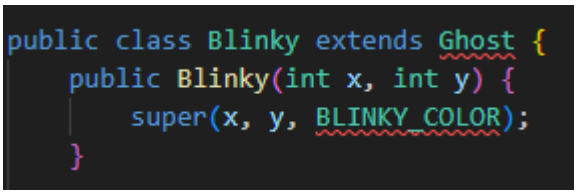
```
private ArrayList<Pellet> pellets;
private ArrayList<Wall> walls;
private ArrayList<Ghost> ghosts;
private Pacman pacman;
private Timer timer;
private int score = 0;
private boolean gameOver = false;
private int poweredUpTimer = 0;
private GameTimer gameTimer;
private final int POWERUP_DURATION = 400;
private final int GHOST_DEAD_DURATION = 20; // ticks the ghost stays dead before reviving
```

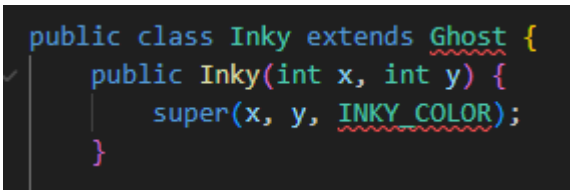
As entidades do jogo não foram separadas em *Element* e *Entity* como dispostas no diagrama da Etapa 1, em vez disso sendo simplesmente objetos com suas propriedades particulares que são chamados por Maze para construir o tabuleiro onde o jogo é então inicializado.

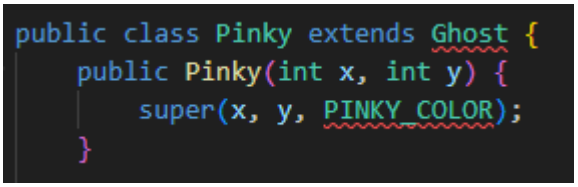
```
//X = wall, O = skip, P = pac man, ' ' = food, . = power pellet
//Ghosts: b = blue, o = orange, p = pink, r = red
private String[] grid = {
    "XXXXXXXXXXXXXXXXXXX",
    "X        X        X",
    "X XX XXX X XXX XX X",
    "X.               .X",
    "X XX X XXXXX X XX X",
    "X    X       X    X",
    "XXXX XXXX XXXX XXXX",
    "OOOX X       X XOOO",
    "XXXX X XXrXX X XXXX",
    "O       bpo       O",
    "XXXX X XXXXX X XXXX",
    "OOOX X       X XOOO",
    "XXXX X XXXXX X XXXX",
    "X        X        X",
    "X XX XXX X XXX XX X",
    "X. X     P     X .X",
    "XX X X XXXXX X X XX",
    "X    X   X   X    X",
    "X XXXXXX X XXXXXX X",
    "X                 X",
    "XXXXXXXXXXXXXXXXXXX"
};
```

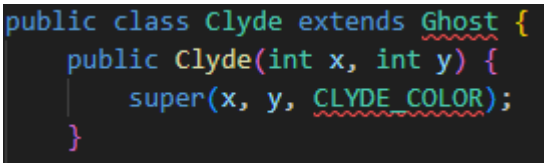
Ghost -> Inky, Blinky, Pinky, Clyde

Os fantasmas Inky, Blinky, Pinky e Clyde foram separados em subclasses de *Ghost* para que cada um tivesse seu comportamento particular. Neste quesito pode se observar a noção de Hierarquia, visto que *Ghost* agora é *composto* por *Inky*, *Blinky*, *Pinky* e *Clyde*.









A classe Ghost em si determina a inteligência simulada dos fantasmas em geral. Sendo assim, ela possui vários atributos usados para calcular o movimento e as colisões dos fantasmas em geral, bem como atributos para determinar a renderização de cada fantasma à parte.

```
private int x, y;
private int size = Maze.TILE_SIZE;
private int speed = 1;
private int returnSpeedMultiplier = 2; // speed multiplier while returning home
private Color color;
private int direction;
private Random random = new Random();
// Home (ghost house) coordinates (where the ghost respawns)
private int homeX, homeY;
private boolean returningHome = false; // true while walking back to home

public enum Mode { CHASE, SCATTER, FRIGHTENED, DEAD }
private Mode mode = Mode.CHASE;
private Mode previousMode = Mode.CHASE;
private int frightenedTimer = 0; // ticks remaining (decremented once per tick)
private int deadTimer = 0; // ticks remaining while paused at home before reviving
private Color baseColor;
private int scatterTargetX, scatterTargetY;
private int directionLockMs = 0; // prevent immediate redecisions to avoid oscillation

// Ghost colors
public static final Color BLINKY_COLOR = Color.RED;
public static final Color PINKY_COLOR = Color.PINK;
public static final Color INKY_COLOR = Color.CYAN;
public static final Color CLYDE_COLOR = Color.ORANGE;
```

Pacman

A classe Pacman existe para determinar a renderização, gráficos, comportamento, controles, movimento a cada frame, posição cartesiana do jogador no tabuleiro (para determinar e resolver colisões) e alguns valores para determinar se ele irá se mover em cada frame.

```
public class Pacman {
    private int x, y;
    private final int size = Maze.TILE_SIZE;
    private final int speed = 2;
    public static final int TURN_UP_QUEUE = 8;
    public static final int TURN_LEFT_QUEUE = 4;
    public static final int TURN_RIGHT_QUEUE = 6;
    public static final int TURN_DOWN_QUEUE = 2;

    private final Map<Integer, BufferedImage> images;

    public int currentDirection = KeyEvent.VK_RIGHT;
    public int turnDirection = 6;
    public boolean willUpdate = true;
```

Wall

A classe *Wall* existe apenas como representativo das paredes, ou obstáculos, do jogo. É uma classe pequena, porém é o elemento mais abundante do cenário e é utilizado diversas vezes nos cálculos de colisão.

```java
import java.awt.*;

public class Wall {
    private int x, y;
    private int size = Maze.TILE_SIZE;
    private Color color = new Color(0, 0, 200); // Blue color for walls

    public Wall(int x, int y) {
        this.x = x;
        this.y = y;
    }

    public void draw(Graphics g) {
        g.setColor(color);
        g.fillRect(x, y, size, size);

        // Add some detail to walls
        g.setColor(new Color(0, 0, 150));
        g.fillRect(x + 2, y + 2, size - 4, size - 4);
    }

    // Getters for collision detection
    public int getX() {
        return x;
    }

    public int getY() {
        return y;
    }
```

Pellet

Determina o comportamento tanto do pellet normal quanto da Power Pellet que altera o comportamento dos fantasmas. Também possui métodos para calcular colisão com Pacman, já que este é o núcleo relevante do papel dos pellets e power pellets no jogo.

Possui construtores diferentes para representar a renderização de um pellet comum ou Power Pellet.

```java
public class Pellet {
    private int x, y;
    private int size;
    private Color color = Color.WHITE;
    private boolean isPowerPellet = false;

    public Pellet(int x, int y) {
        this.x = x + Maze.TILE_SIZE/2 - Maze.TILE_SIZE/8; // Center in tile
        this.y = y + Maze.TILE_SIZE/2 - Maze.TILE_SIZE/8;
        this.size = Maze.TILE_SIZE / 4;
    }

    public Pellet(int x, int y, boolean isPowerPellet) {
        this.x = x + Maze.TILE_SIZE/2 - Maze.TILE_SIZE/4; // Center in tile
        this.y = y + Maze.TILE_SIZE/2 - Maze.TILE_SIZE/4;
        this.size = isPowerPellet ? Maze.TILE_SIZE / 2 : Maze.TILE_SIZE / 4;
        this.isPowerPellet = isPowerPellet;
    }
```

GameTimer

Classe que controla o tempo de jogo. Modificação conceitual da classe Time da Etapa 1, porém serve uma função extremamente similar.

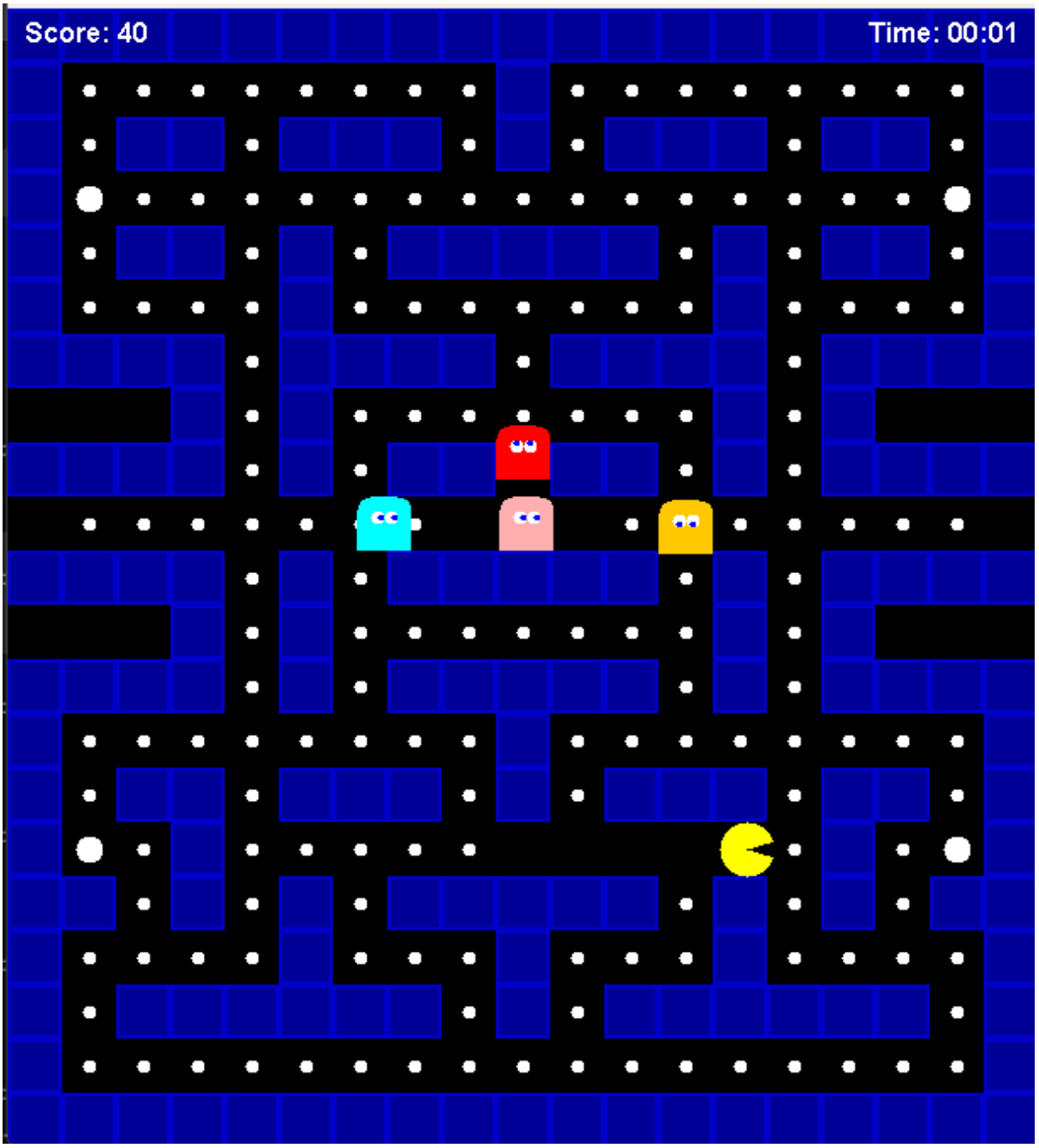
```java
import java.awt.*;
import java.text.SimpleDateFormat;
import java.util.Date;
import java.util.TimeZone;
import javax.swing.*;

public class GameTimer {
    private long startTime;
    private long elapsedTime;
    private boolean isRunning;
    private Timer timer;
    private SimpleDateFormat timeFormat;
```

Testes

Não foram feitos testes de unidade nesta aplicação.

Execução

Ao executar “pacman.jar”, o jogador vai ser levado à interface principal de jogo:



Houve algumas mudanças na tela em comparação à foto tirada do Pac-Man original na Etapa 1, mais proeminentemente em relação à fidedignidade gráfica.

O jogador utiliza as *setas do teclado* para movimentar pac-man. O objetivo é fazer com que ele encoste (“coma”) todos os Pellets e Power Pellets do tabuleiro, enquanto evita encostar nos quatro fantasmas.

Comer um Power Pellet altera, por um tempo limitado, o comportamento, cor, e movimento dos fantasmas, e faz com que Pac-Man possa encostar neles sem medo por um tempo, fazendo-os desintegrar e voltar para o meio do tabuleiro, onde voltarão a ser perigosos.

Encostar em um fantasma enquanto pac-man não está sob o efeito de um Power Pellet resultará em fim de jogo. Comer todos os pellets também, mas num estado de vitória.