Universidade do Algarve

Faculdade de Ciências e Tecnologias

Engenharia Informática

Programação Orientada aos Objetos
Snake

Autor(s)

Márcio Felício a76999
Miguel Rosa 76936
Afonso Sousa 80041

Docente(s)

José Valente de Oliveira
José Barateiro

Relatório apresentado em cumprimento dos requesitos da
Universidade do Algarve na cadeira de Programação Orientada aos Objetos para o grau de licenciatura em *Engenharia Informática*

10 de Abril, 2024

Indíce

Capítulo 1

Introdução

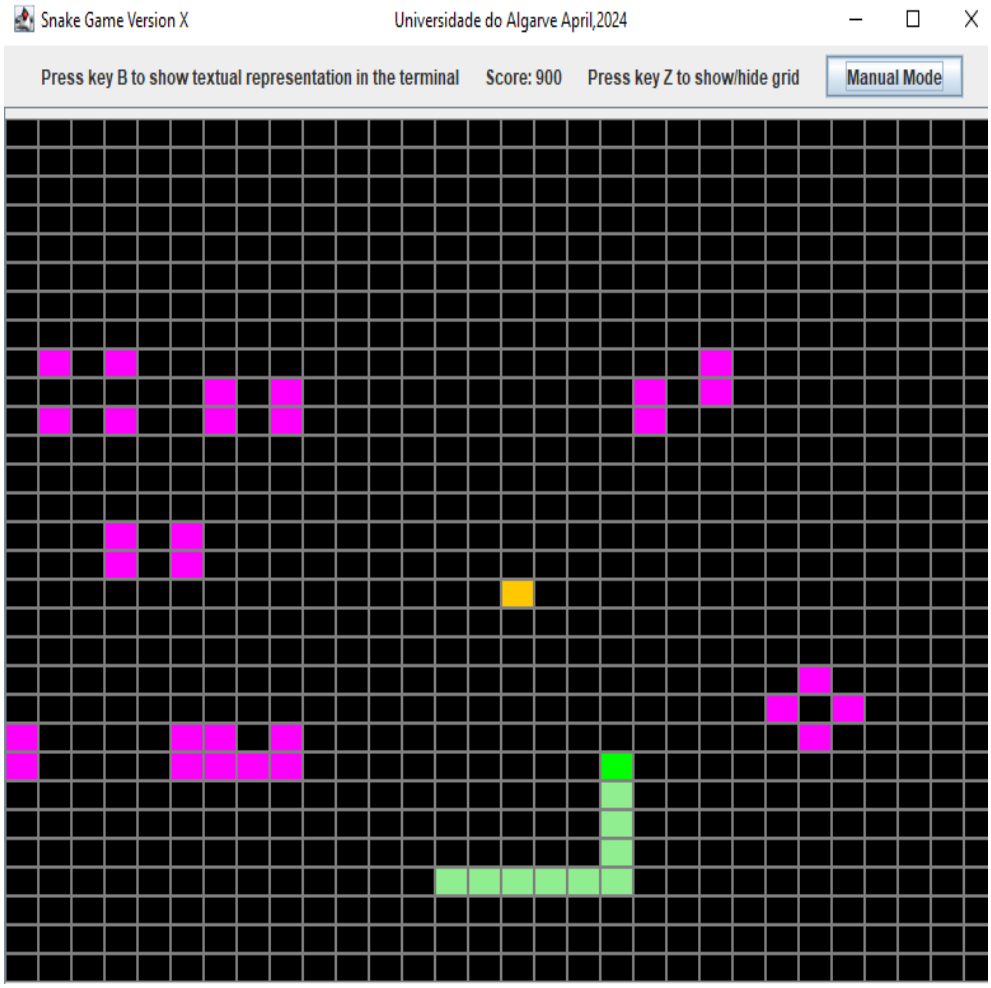

Figure 1.1: Snake Game Version X
Done by: Márcio Felício, Afonso Sousa, Miguel Rosa

O jogo OOPS – Object-Oriented Programmed Snake é uma versão adaptada do clássico jogo de arcade Snake, onde o jogador controla uma cobra que cresce ao consumir itens de comida enquanto evita colidir com obstáculos e as paredes da arena. Nesta adaptação, a arena de jogo é uma grelha onde cada célula pode conter uma parte da cobra, comida ou obstáculos, que podem ser figuras geométricas como triângulos, quadrados ou retângulos, sendo estes estáticos ou dinâmicos.

A cobra no jogo começa apenas com uma cabeça e cresce em comprimento conforme consome comida, adicionando quadrados à sua cauda. Os movimentos da cobra são contínuos e podem ser controlados pelo jogador em modo manual ou operados automaticamente em modo automático. No modo automático, o jogo emprega algoritmos de decisão para controlar a cobra, procurando otimizar a coleta de alimentos e a navegação segura através dos obstáculos sem intervenção direta do jogador. Essa funcionalidade permite a implementação de diversas estratégias de jogo, tornando o OOPS um campo de testes ideal para experimentar com diferentes técnicas de inteligência artificial e programação orientada a objetos.

O projeto OOPS visa não só implementar a lógica de jogo, mas também a aplicação dos conceitos de programação orientada a objetos, como encapsulamento, herança e polimorfismo, além do uso de padrões de projeto para estruturar o código de maneira eficiente e modular. Este jogo é um desafio prático que combina desenvolvimento de interface gráfica, gestão de estado de jogo, e interações baseadas em eventos, proporcionando uma plataforma rica para a aplicação de conceitos avançados de engenharia de software na linguagem de programação Java.

Capítulo 2

Diagrama de classes UML Snake

2.1 Apresentação do Diagrama

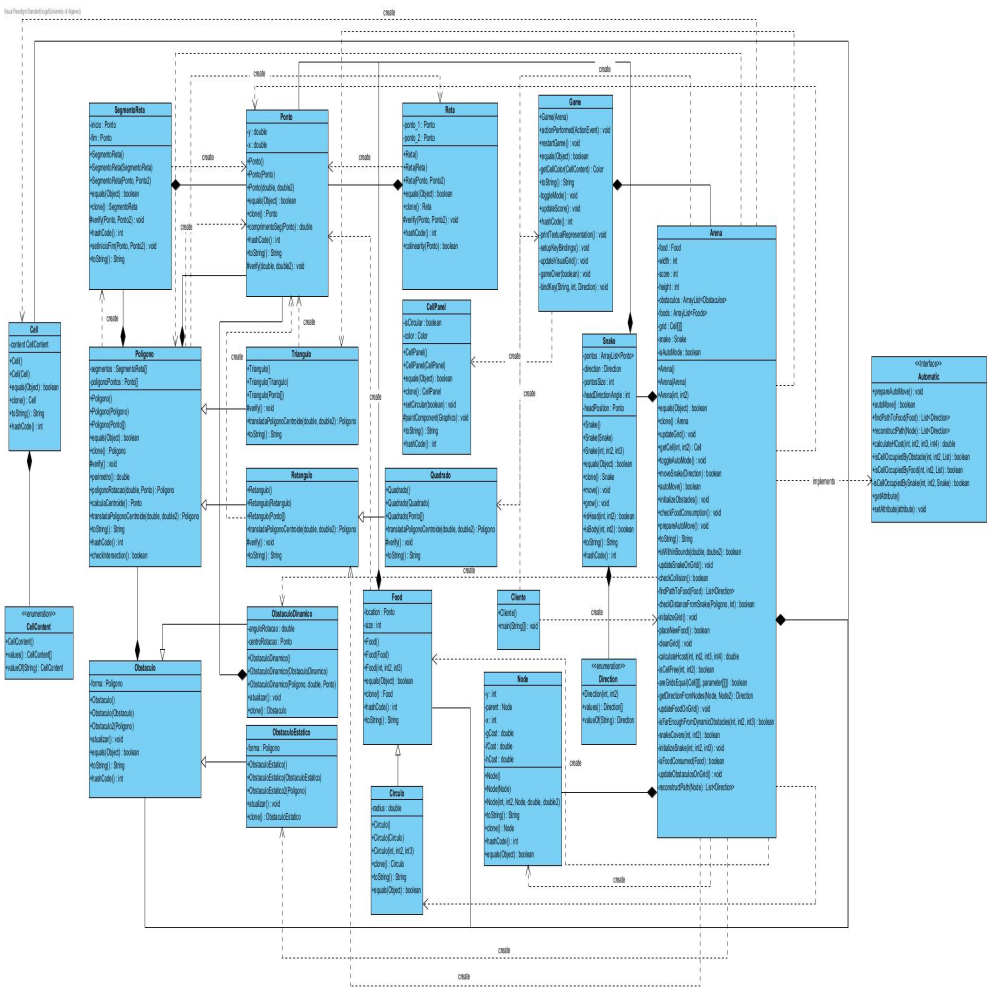

Figure 2.1: Diagrama UML do projeto (Para melhor visualização da imagem, a mesma encontra-se na pasta UML)

2.2 Breve Descrição e Responsabilidade de cada Classe

Classe Ponto

- Construtor que cria um objeto Ponto no plano cartesiano.
- Invariantes:
  - O ponto deve estar no primeiro quadrante.

Classe Reta

- Construtor que cria um objeto Segmento de Reta.
- Invariantes:
  - Diferença entre valores absolutos das coordenadas x e y de dois pontos deve ser $\geq 10^{-9}$.

Classe SegmentoReta

- Construtor que cria um objeto Segmento de uma Reta.
- Métodos para calcular o declive e verificar a intersecção de segmentos.
- Invariantes:
  - Comprimento do segmento deve ser maior que zero.

Classe Poligono

- Construtor que cria um objeto polígono.
- Invariantes:
  - Mínimo de três pontos.
  - Pontos consecutivos não colineares.
  - Segmentos não se cruzam.

Classe Quadrado

- Subclasse de Poligono e Retangulo.
- Invariantes:
  - Quatro segmentos de igual comprimento.

Classe Retangulo

- Subclasse de Poligono.
- Invariantes:
  - Quatro segmentos, lados opostos de igual comprimento.

Classe Triangulo

- Subclasse de Poligono.
- Invariantes:
  - A soma dos comprimentos de dois lados deve ser maior que o comprimento do terceiro lado.

Classe Food

- Representa um item de comida no jogo.
- Invariantes:
  - Localização não nula.
  - Tamanho positivo.

Classe Circulo

- Extende Food para representar uma comida em forma de círculo.
- Invariantes:
  - Raio positivo.
  - Tamanho igual ao dobro do raio.

Classe Abstrata Obstaculo

- Classe abstrata base para obstáculos dinâmicos ou estáticos.
- Invariantes:
  - Forma geométrica definida não nula.

Classe ObstaculoDinamico

- Representa um obstáculo dinâmico que pode ser rotacionado.
- Invariantes:
  - Centro de rotação não nulo.

Classe ObstaculoEstatico

- Representa um obstáculo estático.
- Invariantes:
  - Forma não pode ser alterada após a inicialização.

Classe Arena

- Gerencia o ambiente de jogo.
- Invariantes:
  - Dimensões positivas.
  - Grelha inicializada correspondente às dimensões.
- Representa uma célula na grelha do jogo.
- Pode conter diferentes tipos de conteúdo, como partes da Snake, comida, obstáculos ou estar vazia.
- Invariantes:
  - O conteúdo de uma célula nunca deve ser nulo.

Classe CellContent

- Representa uma célula na grelha do jogo. Uma célula pode conter diferentes tipos de conteúdos, como partes da Snake, comida, obstaculos ou estar vazia.
- Invariantes:
  - content != null, "O conteúdo de uma célula nunca deve ser nulo..

Enumeração CellContent

- Define os tipos possíveis de conteúdo para uma célula no jogo.
- Enums:
  - **EMPTY**: Indica que a célula está vazia.
  - **FOOD**: Indica que a célula contém comida.
  - **OBSTACLE**: Indica que a célula contém um obstáculo.
  - **SNAKE_BODY**: Indica que a célula contém uma parte do corpo da cobra.
  - **SNAKE_HEAD**: Indica que a célula contém a cabeça da cobra.

Classe CellPanel

- Estende JPanel para representar uma célula numa interface gráfica.
- Invariantes:
  - A cor nunca é nula.
  - `isCircular`: Determina a forma da célula, true para circular e false para retangular.

Enumeração Direction

- Define as direções básicas num plano bidimensional.
- Enums:
  - **UP**: Direção para cima.
  - **DOWN**: Direção para baixo.
  - **LEFT**: Direção para esquerda.
  - **RIGHT**: Direção para direita.

Classe Node

- Representa um nó num algoritmo de caminho.
- Guarda a posição no grid (x, y), custo de partida (gCost), custo estimado para o ponto final (hCost) e custo total (fCost).
- Invariantes:
  - O fCost é sempre a soma do gCost com o hCost.
  - Coordenadas x e y são não negativas.
  - O parent não é uma auto-referência.

Classe Snake

- Representa a cobra do jogo, composta por uma lista de pontos.
- Invariantes:
  - O tamanho de cada ponto deve ser maior que zero.

Interface Automatic

- Interface com a declaração dos métodos principais e alguns defaults inclusive que têm como principal objetivo ser a base do modo automático do jogo.

Classe Sound

- Classe responsável pelo som do jogo.

Capítulo 3

Opções de Projeto Tomadas

3.1 Factory Method

A estrutura do nosso projecto OOPS segue uma separação de responsabilidades entre o modelo (Arena), a visualização (Game), e os controles (bindings das teclas em Game).A classe Arena representa o modelo, gerenciando a lógica do jogo e o seu estado.A classe Game representa a visualização, lidando com a interface gráfica e textual do utilizador e as interações com o utilizador. Os bindings das teclas em Game servem como controles, capturando a entrada do utilizador e traduzindo-a em ações no modelo do jogo.

3.2 Observer (Parcialmente):

Embora não seja implementado explicitamente, há uma forma de padrão observador no loop do jogo em Game: O ouvinte de ação do temporizador (Classe Game) observa mudanças no estado do jogo e desencadeia atualizações na grade visual e na exibição da pontuação conforme necessário.

3.3 Singleton (Parcialmente):

Embora não seja totalmente implementado, a classe Cliente possui características de um padrão singleton com o seu método main: Garante que apenas uma instância do jogo seja criada e iniciada, gerenciando o ciclo de vida do jogo.

3.4 Strategy (Parcialmente):

O modo de jogo (automático ou manual) na classe Game utiliza uma forma parcial de padrão de estratégia.
O comportamento do jogo (se ele se move automaticamente ou aguarda entrada manual) pode ser alternado dinamicamente com base no modo de jogo atual.

3.5 Iterator (Parcialmente):

Embora não seja implementado explicitamente, há uma forma de padrão de iterador no loop do jogo em Game: Itera sobre os estados do jogo (como, por exemplo, o movimento da cobra, a atualização da pontuação) e atualiza a interface do utilizador conforme necessário em cada iteração.

Capítulo 4

Testes Unitários

Testes Unitários encontram-se no ficheiro Snake/src em anexo.

Capítulo 5

Observações

5.1 Observações Importantes sobre o Desenvolvimento do Projeto

- **Remoção da Cópia Profunda:** Inicialmente, o projeto fazia uso intensivo do método `clone` para garantir cópias profundas de objetos em operações de get e set. Essa abordagem foi adotada para manter o princípio de encapsulamento, um conceito fundamental em programação orientada a objetos que ajuda a proteger o estado interno dos objetos. No entanto, foi observado que a utilização de cópia profunda aumentava significativamente o consumo de memória heap, levando a uma degradação do desempenho e até mesmo à paralisação do jogo o que ainda acontece mas muito menos. Devido a isso, optou-se por remover a implementação do método `clone` nos getters e setters, reconhecendo que essa é uma limitação do projeto que compromete o encapsulamento.

- **Configuração da Snake:** A decisão de configurar a snake com dimensões de 1x1 na grid foi tomada para manter a experiência o mais próxima possível do jogo tradicional Snake.

- **Configuração do score:** O score foi configurado para caso a comida seja da classe circulo ser aumentado 50 pontos e caso contrário seja aumentado 100 pontos.

- **Configuração dos obstáculos:** Inicialmente colocou-se 6 obstáculos mas o número de obstaculos pode ser alterado na classe arena, função initializeObstacles() no primeiro for, basta trocar o "6"para "10"por exemplo.

- **Configuração do som:** Embora não se encontre no diagrama UML o jogo esta configurado com som!.

- **Configuração dos Controles:** Os controles para jogar utilizam as teclas tradicionais: UP (cima), DOWN (baixo), LEFT (esquerda) e RIGHT (direita). Para visualizar as funcionalidades. Note que, embora no auto-mode, a snake pode colidir com obstáculos dinâmicos, mas não com os estáticos.

- **Codificação e Tratamento de Exceções:** As exceções foram codificadas e estão armazenadas na respectiva pasta. Ao iniciar o jogo, pode ocorrer um erro durante a configuração da arena, como "reta:vi". Se isso acontecer, basta reiniciar o jogo para continuar, pois o tratamento de exceções já está implementado.

Capítulo 6

Conclusão

A conclusão do projeto OOPS, uma versão adaptada e orientada a objetos do clássico jogo Snake, reflete uma experiência de aprendizado profunda e produtiva em programação orientada a objetos. Através deste projeto, foi possível aplicar conceitos teóricos em um contexto prático, o que resultou em um software robusto, modular e de fácil manutenção. As decisões de projeto, especialmente a utilização de padrões de design como MVC, Singleton, Observer, Factory e Strategy, não só facilitaram a gestão do estado do jogo e a interação do usuário, mas também garantiram a extensibilidade e adaptabilidade do sistema.

A separação clara entre lógica de jogo, interface do usuário e controle de eventos através do padrão MVC permitiu focar em melhorias e testes específicos sem riscos de interferências cruzadas. Isso aumentou significativamente a qualidade do desenvolvimento e a eficiência na detecção e correção de erros. A implementação de um código responsivo e flexível através dos padrões Observer e Strategy mostrou-se fundamental para uma experiência de usuário fluida e agradável, que responde intuitivamente às ações dos jogadores.

Este projeto não apenas fortaleceu habilidades técnicas em Java e design de software, mas também enfatizou a importância do planejamento cuidadoso e da documentação detalhada para o sucesso de desenvolvimentos futuros. Além disso, a colaboração em equipe foi crucial, permitindo a troca de conhecimentos e a superação de desafios de maneira coletiva.

Em suma, o projeto OOPS foi uma oportunidade valiosa de crescimento profissional e pessoal, proporcionando insights essenciais para futuros projetos de software e reforçando a relevância dos padrões de design na construção de soluções eficazes e sustentáveis em programação orientada a objetos.

Capítulo 7

Bibliografia

GeeksforGeeks. ≪**A* Search Algorithm**≫
https://www.geeksforgeeks.org/a-search-algorithm/.

GeeksforGeeks. ≪**Encapsulation in Java**≫
https://www.geeksforgeeks.org/encapsulation-in-java/.

GeeksforGeeks. ≪**Java JFrame**≫
https://www.geeksforgeeks.org/java-jframe/.

GeeksforGeeks. ≪**Introduction to Java Swing**≫
https://www.geeksforgeeks.org/introduction-to-java-swing/.

GeeksforGeeks. ≪**Clone() Method in Java**≫
https://www.geeksforgeeks.org/clone-method-in-java-2/.