Linguagem de Programação II – Snaze - Trabalho Final*

Thiago Freire Cavalcante

[1]IMD - Universidade Federal do Rio Grande do Norte (UFRN) – Natal, RN - Brasil

Abstract. *This technical report presents the development of Snaze, an object-oriented implementation of the classic Snake game. The project implements multiple game modes, including single-player, multiplayer, and arcade mode. The system architecture focuses on modularity, featuring an automated opponent, collision detection system, and score persistence. The results demonstrate the successful application of object-oriented principles in game development.*

Resumo. *Este relatório técnico apresenta o desenvolvimento do Snaze, uma implementação orientada a objetos do clássico jogo Snake. O projeto implementa múltiplos modos de jogo, incluindo single-player, multiplayer e modo arcade. A arquitetura do sistema foca em modularidade, apresentando um oponente automatizado, sistema de detecção de colisões e persistência de pontuação. Os resultados demonstram a aplicação bem-sucedida dos princípios de orientação a objetos no desenvolvimento de jogos.*

1. Introdução

O Snake, um dos jogos mais icônicos dos anos 90, popularizou-se em celulares Nokia como um entretenimento simples e viciante. Este projeto visa recriar o jogo clássico em uma versão moderna, denominada Snaze, que inclui novos desafios como obstáculos no formato de labirinto e modos de jogo diversificados. Além de resgatar a essência nostálgica do Snake, o Snaze aplica conceitos avançados de programação orientada a objetos, fornecendo um código modular, escalável e reutilizável. Este relato aborda as etapas de desenvolvimento, as funcionalidades implementadas e as inovações introduzidas.

1.1. Objetivos

O objetivo principal deste projeto é implementar uma versão moderna do jogo Snake, mantendo os conceitos clássicos, mas introduzindo melhorias e inovações no design e na jogabilidade. Para isso, os seguintes objetivos específicos serão alcançados:

a. Implementar uma versão moderna do jogo: O jogo será desenvolvido com mecânicas atualizadas, melhorando a experiência visual e a dinâmica do jogo, mantendo, porém, a essência do clássico *Snake*.

b. Aplicar conceitos de Programação Orientada a Objetos (POO) e padrões de projeto: A implementação utilizará conceitos da POO para organizar o código de forma modular e reutilizável. A estrutura do código será dividida em classes que representem os principais elementos do jogo, como a cobra, o labirinto, a

comida e o sistema de pontuação.

c. Desenvolver múltiplos modos de jogo: O jogo incluirá diferentes modos de jogo, proporcionando uma experiência diversificada para os jogadores. Entre os modos previstos estão: modo Single-player, modo Multiplayer, Modo Arcade.

d. Implementar diferente algoritmos de busca para o encontrar um caminho até a comida.

Esses objetivos visam não só melhorar o jogo clássico, mas também garantir que o código seja bem estruturado e que a experiência ofereça novos desafios para o jogo.

2. Metodologia de Desenvolvimento

O desenvolvimento do Snaze foi organizado em etapas, desde o planejamento inicial até a implementação e testes. A seguir, são descritas as etapas do projeto:

a. **Planejamento e análise de requisitos:** Determinou-se as funcionalidades desejadas, como modos de jogo (Single-player, Multiplayer, Arcade) e a inclusão de lógicas para bots.

b. **Modelagem e design:** Elaborou-se o diagrama de classes, identificando as entidades principais e seus relacionamentos.

c. **Implementação:** As classes foram desenvolvidas seguindo princípios da POO e aplicando o padrão de projeto *State* para gerenciamento de estados do jogo.

2.1. Arquitetura do Sistema

A estrutura do diagrama de classes descreve as principais entidades e seus relacionamentos no jogo Snaze. Abaixo está a descrição em texto das classes e suas responsabilidades:

2.1.1 Classe SnakeGame:

Atributos:

a. gameMode: Define o modo de jogo atual (ex.: Single-player, Multiplayer, Arcade).

b. snake1: Representa a primeira cobra (se houver).

c. snake2: Representa a segunda cobra (caso o modo de jogo seja multiplayer).

d. labirinto: Representa o labirinto no qual as cobras se movem.

Métodos:

a. actionPerformed(): Método que lida com as ações realizadas durante o jogo, como movimento ou colisões.

b. keyPressed(): Método responsável por capturar as teclas pressionadas e mudar a direção das cobras ou realizar outras ações no jogo.

2.1.2 Classe Classe Snake (Abstrata):

Atributos:

a. corpo: Uma lista de objetos da classe Celula, que representa cada segmento da cobra.
b. direcaoX: A direção do movimento da cobra no eixo X.
c. direcaoY: A direção do movimento da cobra no eixo Y.

Métodos:

a. mover(): Método que movimenta a cobra de acordo com sua direção.
b. crescer(): Método que permite à cobra crescer quando ela come um alimento.

2.1.3 Classes SnakePlayer e SnakeBot:

Ambas são subclasses da classe Snake.

A classe SnakePlayer representa a cobra controlada pelo jogador, enquanto a classe SnakeBot representa uma cobra controlada por inteligência artificial (bot).

2.1.4 Classe GameState:

A classe SnakeGame possui uma associação com a classe GameState, pois o estado do jogo é uma parte crucial da implementação do jogo e será constantemente atualizado com o progresso do jogo.

Essa estrutura permite modularidade no código e facilita a expansão, como a adição de novos modos de jogo ou de comportamentos para diferentes tipos de cobras (jogador ou bot).

2.2. Padrões de Projeto Utilizado:

a. State Pattern: utilizado para gerenciar os diferentes estados do jogo, implementado na classe GameState.

2.3. Tratamento de Exceções:

A hierarquia de exceções do Snaze é estruturada a partir de uma classe base GameException, que estende Exception do Java. Esta classe base serve como fundamento para exceções específicas: LevelLoadException para erros de carregamento de níveis, PlayerDataException para problemas com dados dos jogadores e MovimentoInvalidoException para erros de movimento da cobra.

3. Funcionalidades da Aplicação

O Snaze oferece uma experiência diversificada, com as seguintes funcionalidades principais:

3.1. Modos de Jogo:

a. Single-player: O jogador controla uma única cobra.

b. *Multiplayer:* Dois jogadores competem no mesmo labirinto.
c. *Modo Arcade:* Inclui desafios adicionais, como níveis com tempo limitado e obstáculos dinâmicos.

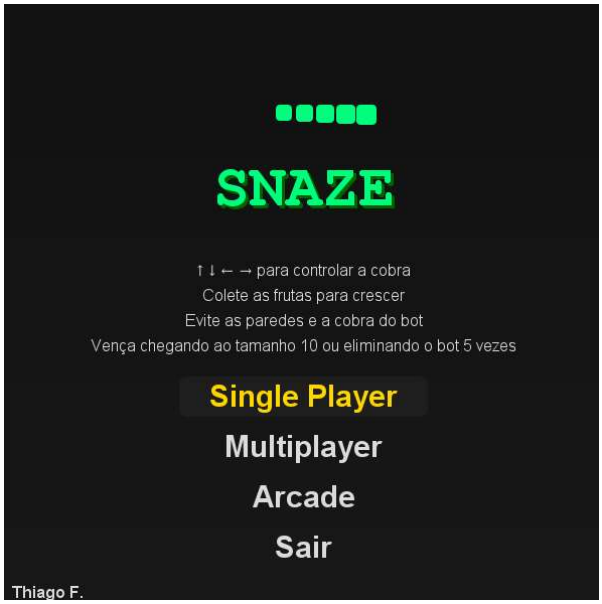

Figure 1. Menu Inicial

3.2. Regras para Bots:

As cobras controladas automaticamente utilizam algoritmos para buscar o caminho mais eficiente até a comida, evitando paredes e outras cobras.

3.3. Gerenciamento de Estados:

O padrão State garante que as transições entre estados do jogo (por exemplo, jogando, pausado, fim de jogo) sejam gerenciadas de forma consistente.

3.4. Tratamento de Exceções:

A hierarquia de exceções inclui GameException, LevelLoadException, PlayerDataException e MovimentoInvalidoException, garantindo a robustez do sistema.

4. Algoritmos

4.1. Logica do SnakeBot

4.1.1 Método calcularCaminho(Celula comida):

Este método calcula o melhor caminho para a cobra alcançar a comida, levando em consideração a posição atual da cabeça da cobra e a posição da comida.

Ele começa verificando se a célula onde a cabeça da cobra está localizada (determinada por cabeca) contém uma parede, chamando o método labirinto.ehParede() para verificar se há uma parede nas células adjacentes.

4.1.2 Evitar Paredes:

Caso a célula da cabeça da cobra tenha uma parede ao seu redor (em qualquer direção), o algoritmo tenta mudar a direção da cobra para evitar a colisão. Isso é feito da seguinte forma:

Se a cabeça da cobra está para a esquerda da comida (cabeca.getX() < comida.getX()), o algoritmo verifica se a direção da cobra pode ser alterada para a direita (aumento no valor de X, direcaoX = 1), sem colisão com uma parede.

Se a cabeça está para a direita da comida, o algoritmo tenta mover a cobra para a esquerda. Caso a cabeça está acima da comida, ele tenta mover para baixo. Mas se a cabeça está abaixo da comida, ele tenta mover para cima.

4.1.3 Mudança de Direção:

Se a direção da cobra ainda não foi alterada após verificar as paredes ao redor da cabeça, o algoritmo tenta encontrar a direção mais próxima da comida:

Ele verifica novamente as células ao redor da cabeça e tenta mover a cobra na direção mais próxima da comida, desde que não haja uma parede no caminho.

A lógica garante que a cobra não se mova para a direção oposta à que está indo (por exemplo, se ela está indo para a esquerda, ela não deve tentar ir para a direita).

5. Conclusão

O projeto Snaze alcançou o objetivo de modernizar o clássico Snake, introduzindo novas mecânicas e conceitos avançados de programação. A modularidade e a reutilização do código foram priorizados, permitindo futuras expansões e melhorias.

5.1. Resultados Alcançados:

a. Implementação bem-sucedida de múltiplos modos de jogo.

b. Sistema de tratamento de exceções.

c. Interface gráfica atrativa.

5.2. Resultados Não Alcançados:

a. Implementação de diferentes algoritmos de busca.

5 References

Deitel, P. and Deitel, H. (2015), *Java: Como Programar*,Pearson, 1th edition.

Ascencio, A. and Araújo, G. (2015), *Estruturas de Dados: algoritmos, análise da complexidade e implementações em JAVA e C/C++,* Pearson, 1th edition.

Rodrigo Freitas. (2021). Cursis de Java – JFrame, Interface Gráfica – Aula 32 – Programaçõa Iniciante. Disponível em: https://www.youtube.com/watch?v=fqyOD-wl1Cs . Acesso em: 10 dez. 2024

Bro Code. (2022). *How to Build a Snake Game in JavaJava snake game*. YouTube. Disponível em: https://www.youtube.com/watch?v=bI6e6qjJ8JQ . Acesso em: 20 dez. 2024