# Snack Game: Uma releitura de jogos tradicionais voltada para os moldes da Programação Orientada a Objetos utilizando o Pygame

1st Lucas Pimenta Braga
*Dep. de Engenharia de Sistemas*
*Universidade Federal de Minas Gerais*
Belo Horizonte, Brasil
2023034552

2nd Mateus de Souza Gontijo
*Dep. de Engenharia de Sistemas*
*Universidade Federal de Minas Gerais*
Belo Horizonte, Brasil
2020053530

3rd Victor Gabriel dos Santos Silva
*Dep. de Engenharia de Sistemas*
*Universidade Federal de Minas Gerais*
Belo Horizonte, Brasil
2023060804

***Abstract*—O presente relatório descreve a adaptação e aprimoração do famoso "jogo da cobrinha" (Snake), desenvolvido em Python e criado como projeto da disciplina de Análise, Projeto e Programação Orientada a Objetos na Universidade Federal de Minas Gerais, ministrada pela professora Gabriela Nunes. O sistema busca reproduzir a mecânica clássica do gênero, incluindo o controle do movimento da cobra, a geração aleatória de itens de pontuação (maçãs), a detecção de colisões e o gerenciamento dinâmico de placar. Para isso, foram aplicados os pilares da Programação Orientadas a Objeto, garantindo uma arquitetura modular e de fácil manutenção. Além disso, características nativas da biblioteca Python, assim como da biblioteca Pygame, foram empregadas de forma a possibilitar a entrega de uma experiência de jogo responsiva e de fácil extensão.**

***Index Terms*—Jogo, Programação Orientada a Objetos, Python, Bibliotecas, Pygames.**

## I. Introdução

A trajetória dos jogos eletrônicos remonta aos laboratórios de pesquisa da década de 1950, quando acadêmicos e engenheiros passaram a explorar as possibilidades lúdicas de mainframes e osciloscópios. Títulos pioneiros como Nimrod, OXO e Hutspiel surgiram como experimentos de inteligência artificial, interação humano–computador e simulações matemáticas, sem ainda visar o público consumidor. Em 1958, William Higinbotham desenvolveu o Tennis for Two para entretenimento de visitantes num osciloscópio, exibindo o primeiro grafismo de trajetória de uma "bola" em uma "quadra" minimalista. Poucos anos depois, em 1961, Steve Russell e colegas do MIT criaram o Spacewar no PDP-1, um jogo de tiro entre duas naves que passou a ser distribuído pela DEC como demonstração em todas as suas máquinas, inspirando entusiastas a programarem suas próprias aventuras digitais [1]. Um bom tempo depois, nos anos 2000 e 2010, a indústria de games se diversificou: além de consoles e PCs, passou a abranger jogos para celulares, redes sociais e dispositivos de realidade virtual. Gigantes como Nintendo, Sony e Microsoft passaram a investir em produções com orçamentos comparáveis aos cinematográficos, enquanto desenvolvedores independentes encontraram espaço em lojas digitais para levar ao público obras criativas e de baixo custo. Paralelamente, a "cultura gamer" se incorporou a outras mídias, filmes, quadrinhos, música e influenciou práticas educacionais e ambientes de trabalho por meio da gamificação, mostrando que jogar pode ser também aprendizado e colaboração [1].

No Brasil, esse fenômeno global ganhou força especialmente a partir da pandemia de 2020, que acelerou o consumo de entretenimento digital. Segundo a Pesquisa GameBrasil, hoje 74,5 da população brasileira joga com maioria de mulheres (51 por cento) e faixa etária principal entre 25 e 34 anos. Quase metade dos gamers (48,3 por cento) prefere o smartphone como plataforma, seguida por videogame (20 por cento ) e computador (15,5 por cento) [2]. Em 2021, o Brasil ultrapassou o México como maior mercado de games da América Latina, gerando US 1,4 bilhão em receita, valor que pode dobrar até 2026, alcançando US 2,8 bilhões e respondendo por 47,4 da receita regional [2].

Hoje o setor movimenta cerca de US 2,3 bilhões no país, sustenta milhares de empregos em desenvolvimento, design e streaming, e provê retorno econômico significativo, demonstrando o papel central dos jogos eletrônicos na cultura e na economia nacionais [3].

Nesse contexto, este relatório visa apresentar os conhecimentos e conceitos utilizados sobre a aplicação do "Snake Game", um jogo memorável que marcou presença e fez sez seu nome em uma época na qual os "celulares de botão" reinavam e o conceito de smatphones ainda não se encontrava difundido no imaginário popular brasileiro. O código, assim como o link do seu diagrama UML associado, encontra-se disponível em um repositório do GitHub [4].

## II. Metodologias

Dado a experiência satisfatória obtida pelo grupo no projeto anterior e a importância do emprego dos conceitos da arquitetura de software nesse âmbito, buscou-se seguir, para esta aplicação atual, o mesmo princípio de desenvolvimento que norteou a produção da aplicação passada (sistema de gerenciamento bancário). Nesse sentido, visando a modularidade, flexibilidade, manutenção/expansão da aplicação e

a melhor compreensão lógica e visual do código, utilizou-se novamente do conceito clássico de Arquitetura em Camadas e Padrões de Projeto Consolidados [5] somado a um Padrão MVC [6] adaptado (com models, views/telas e controllers/services), mas agora aliado a um Roteador de Telas e às facilitações (implementações), obtidas através de acervos associados à biblioteca Pygame [7], no que diz respeito ao módulo de engine do jogo. Tudo isso em conjunto com o uso do framework Flet [8]. Assim como no último projeto, a adoção dessa prática permitiu uma melhor separação de responsabilidades e a maior fluidez no desenvolvimento e na compreensão da lógica de funcionamento do programa, seguindo o princípio de ”responsabilidade única”.

Cabe destacar que a lógica do jogo já se encontrava implementada e presente em acervos/repositorios facilitadores via Pygame [9], sendo o trabalho do grupo majoritariamente implementar novas funcionalidades (ranking, telas, uso do banco de dados, sistema de login, etc) e adaptar a lógica encontrada previamente para os moldes da Programação Orientada a Objetos, adequando-o a esse padrão.

A seguir, serão apresentados uma visão geral do programa, seguindo para as implementações nativas já presentes no código versus as realizadas por nós e finalizando com a presença e aplicação da POO neste projeto, compondo, dessa forma, as abordagens metodológicas adotadas pelo nosso grupo.

### *A. Visão Geral*

O ”Snack Game” é uma releitura e adpatação do jogo ”Snake Game”, na qual uma cobrinha deve se alimentar de maçãs que aparecem aleatóriamente no cenário do jogo, de forma que quanto maior a quantidade de maçãs comidas, maior será a pontuação obtida pelo usuário, ao mesmo tempo que maior será a dificuldade apresentada pelo jogo. Aqui há uma brincadeira fonética entre as palavras snake (cobra) e snack (lanche) que soam muito parecidas em inglês, criando um jogo de ideias divertido, uma vez que durante a gameplay a cobrinha come itens, fazendo um verdadeiro ”snack” a cada ponto.

Agora voltando para o código, descobrimos, durante a realização dos projetos anteriores da disciplina, que a escolha de um modelo de arquitetura que faz ”sentido”, sendo compatível com o projeto, é uma decisão estratégica que irá impactar diretamente a qualidade e o desenvolvimento do código. Sob essa ótica, este projeto foi desenvolvido e aprimorado baseando-se majoritariamente em uma Arquitetura de Camadas, buscando separar as responsabilidades em diferentes seções lógicas a fim de promover a organização, manutenção e a escalabilidade do código, além da melhor integração entre o que já foi implementado e o que iríamos implementar/aprimorar.

O código conta com diversas camadas, sendo elas: a Camada de Interface (responsável por toda a interação com o usuário, sendo a parte visual do sistema construída com a biblioteca Flet), a Camada de Serviços (que contém a lógica da aplicação, orquestrando as operações como autenticar um usuário ou registrar uma nova conta, servindo como uma ponte entre a interface e os dados), a Camada de Modelos (que representa as estruturas de dados fundamentais do sistema, definindo o que é um ”Usuario”, uma ”Snake” ou uma ”Apple”), a Camada de Jogo (isola completamente a lógica do ”Snack Game”, sendo um subsistema independente que roda em Pygame), a Camada de Utilitários (fornece funcionalidades de suporte, como validação e logging) e a Camad de Dados (servindo como o repositório de persistência, isolado da lógica de aplicação).

A camada de Interface está relacionada com a experiência do usuário (como já foi apresentado anteriormente), sendo dividida em Telas e Componentes. As telas (divididas em TelaLogin, TelaCadastro, TelaMenuUsuario e TelaRanking) são as páginas da aplicação e apresentam cada uma funcionalidades específicas. Elas capturam a entrada do usuário e delegam a lógica de programa aos serviços, sendo construídas a partir dos elementos definidos no módulo de Componentes. Os Componentes abrigam um conjunto de classes e funções reutilizáveis que constroem os blocos fundamentais da interface gráfica, permitindo a reutilização de código e garantindo a consistência visual em toda a aplicação. Ao centralizar elementos comuns (botões, caixas, inputs, etc), evita-se a repetição de código e garantimos a aparência uniforme desses componentes em todas as telas.

Na base de toda estrutura está o módulo de Utilitários, onde encontramos classes como Config, que unifica caminhos e constantes de ambiente, e ValidarUsuario, responsável por garantir que nomes, senhas e pontuações obedecem a regras de integridade. Essa abordagem centralizada de validação evita redundância e garante que, qualquer que seja o ponto de entrada de dados no sistema (Flet ou JSON), as informações passem mesmos critérios de consistência.

A camada de Serviços (Services) implementa a lógica da aplicação, sendo composta pelo BaseService, AuthService, CadastroService e RankingService. O BaseService abstrai o acesso ao arquivo JSON na pasta ”DataBase”, garantindo uma forma padronizada de ler e escrever os dados. Quanto às demais classes, todas elas herdam de BaseService, mas apresentam implementações especificas, como autenticação, registro e ranking, respectivamente. O CadastroService controla o fluxo de criação de um novo usuário, validando duplicidade, instanciando Usuario e salvando-o, enquanto o AuthService gerencia sessões em memória, com métodos de login e logout, além de oferecer a consulta ”esta-logado()”, de modo que cada ação relevante é anotada via Logger, padronizando o registro de eventos. Por último, o RankingService carrega todos os usuários, ordena por pontuação e retorna as três maiores.

O módulo de Modelos (Models) reúne as classes que representam os principais elementos do sistema: Snake, Apple e Usuario. A classe Usuario encapsula dados sensíveis como nome e senha, utilizando um validador para impedir a construção de instâncias inválidas, enquanto a classe Snake mantém internamente uma lista de segmentos, registra mudanças de direção, controla crescimentos e detecta colisões por meio de métodos como ”mover()”, ”crescer()” e ”bateu-

na-parede()". A Apple gera novas posições livres dentro dos limites da janela, evitando que a maçã apareça sobre o corpo da cobra, além de desenhá-la na tela.

No módulo de Jogo, todo o funcionamento do "Snack" está isolado em duas partes que trabalham juntas de maneira simples e claras. Primeiro, a classe Game inicia uma janela usando o Pygame, lê o nome do jogador que chega como parâmetro de linha de comando e cria as entidades Snake e Apple. Em seguida, entra num laço contínuo onde, a cada iteração, ela capta eventos de teclado para mudar a direção da cobra, atualiza a posição dos objetos (movimenta a cobra, gera nova maçã quando necessário) e verifica colisões contra paredes ou o próprio corpo. Depois disso, ela redesenha toda a tela, exibindo a cobra, a maçã e a pontuação atual. Quando a partida termina, o método salvar-score() reabre o mesmo arquivo JSON usado pelo resto da aplicação, compara a pontuação obtida com o registro anterior do jogador e grava o valor maior. Para que essa lógica não trave a interface em Flet, foi usada a classe GameRunner, que simplesmente recebe o objeto Usuario já autenticado e chama o Pygame em um processo separado (subprocess.run), passando o nome como argumento. Assim, o jogo roda de forma independente, sem interferir na navegação da aplicação, e compartilha resultados apenas por meio do arquivo de dados.

Por último, destaca-se o "roteador". Apesar do roteamento de telas não representar um módulo por si só, convém citar as características do mesmo dado sua importância para o correto funcionamento e fluidez da aplicação do nosso grupo. Toda a lógica de troca de telas fica concentrada na classe Roteador, de modo que nenhuma tela precisa se comunicar diretamente umas com as outras. Ao ser criado, o Roteador recebe a instância de ft.Page usada pela aplicação Flet. Seu método principal, "navegar(route, *args)", funciona assim: primeiro, ele limpa a lista de views atuais ("self.page.views.clear()"), garantindo que nada deixado por telas anteriores permaneça exibido, em seguida, escolhe qual tela instanciar com base na "string route" recebida (cadastro, menu, ranking), chama o construtor da respectiva classe de tela e passa os argumentos necessários. Para sair de uma sessão, o método "fazer-logout()" unifica em um só lugar a chamada de AuthService.logout() e o redirecionamento imediato para a tela de login. Com essa abordagem, toda a navegação e tratamento de fluxo ficam em um ponto central, o que simplifica a manutenção (caso precisassemos adicionar uma nova rota bastaria inserir uma nova condição em navegar, sem alterar o código de cada tela) e garante consistência na transição entre views, mantendo a interface sempre sincronizada com o estado da aplicação.

### *B. Implementações*

O Pygames não apenas conta com uma extensa biblioteca que facilita a implementação de diversos jogos, como também conta uma comunidade de desenvolvedores altamente engajada. A comunidade Pygame tem um papel fundamental no crescimento e na vitalidade do desenvolvimento de jogos com Python, funcionando como um verdadeiro ecossistema colaborativo que acolhe desde iniciantes até desenvolvedores experientes, possibilitando o aprendizado contínuo ao disponibilizar tutoriais, exemplos práticos, repositórios no GitHub e uma vasta coleção de jogos compartilhados no site oficial e em fóruns.

Nesse sentido, como já mencionado, o objetivo do grupo foi aprimorar e refinar uma aplicação já pronta disponível em um repositório [10] no GitHub, atuando implementando novos métodos e adequando a aplicação base para os moldes da POO.

Ao compararmos o ccodigo base do jogo [10] com a versão modularizada desenvolvida, percebe-se uma verdadeira evolução da aplicação. O primeiro, escrito em um único arquivo procedural, centraliza todo o estado do jogo em variáveis globais e concentra a lógica em funções soltas, algo que, embora suficiente para demonstrar a mecânica essencial, dificulta manutenção, reuso de código e expansão de funcionalidades. Já o desenvolvido pelo grupo adota uma arquitetura em camadas, alicerçada nos princípios da Programação Orientada a Objetos (POO), oferecendo uma experiência de usuário muito mais rica e um código mais sustentável. A seguir destalham-se os aprimoramentos desenvolvidos.

- **Organização do código e interface**: A modularização distribui responsabilidades em cinco camadas bem definida, conforme já abordado na seção anterior, afastando-se do design utilizado na aplicação original. Na camada Models, classes como Snake, Apple e Usuario encapsulam dados e comportamentos, afastando-se das simples listas e tuplas do código original. A camada Services concentra a autenticação, cadastro de usuários, ranking e a persistência em arquivo JSO, de forma que a aplicação não apenas executa o jogo, mas também gerencia perfis de jogadores e suas pontuações, algo inexistente no programa inicial.
  A interface do usuário é entregue pela camada Interface, construída com o framework Flet. Antes mesmo de o Pygame abrir sua janela, o jogador transita por telas de login, cadastro, menu principal e ranking, todas estilizadas e reutilizando componentes visuais padronizados. Um roteador dirige a navegação, garantindo que cada ação leve à tela correta.
  O módulo Game fica isolado da interface gráfica principal, onde o GameRunner inicia a engine de jogo (engine.py) em um processo separado, recebendo o nome do usuário logado como argumento.
  Por fim, a camada Utils reúne utilitários como logger, config e o validador de entrada, centralizando convenções e reduzindo duplicação de código.
- **Mecânica de jogo**: Dentro do Pygame, a jogabilidade foi significativamente refinada. A classe Snake agora oferece métodos claros para mover-se, crescer e mudar de direção, abstraindo a manipulação direta de listas presentes no programa original. A detecção de colisão foi ampliada: além de verificar impactos contra as paredes, o sistema identifica quando a cobra se choca consigo mesma, um requisito clássico que estava ausente. A lógica de alimentação também evoluiu, na qual surgiram

mações especiais que aceleram a cobra temporariamente, implementadas por meio de um atributo tipo na classe Apple e de um sorteio de probabilidade a cada nova fruta.

- **Funcionalidades inéditas**: A maior inovação do sistema modular desenvolvido é o ecossistema de usuários. Jogadores podem criar contas, fazer login e ver suas melhores pontuações serem salvas em um arquivo usuarios.json. O menu principal ”chama” o usuário pelo seu ”nickname”, e uma tela de ranking exibe os três melhores aproveitando-se do RankingService, estimulando uma competição saudável e a contínua vontade de jogar. Também há as telas de cadastro e login, que contam com validação de dados e feedback visual imediato graças ao componente Notificador.

  A experiência de jogo em si também ganhou polimento, com o uso de uma trilha sonora contínua, efeitos sonoros distintos (comer frutas, aumentar velocidade e colidir), a possibilidade do jogador pode pausar a partida com a barra de espaço e, ao perder, ver uma tela de Game Over que exibe seu nome, a pontuação e a instrução para recomeçar. Tudo isso sem fechar abruptamente o programa, como ocorria no script inicial.

O percurso do script monolítico para o sistema modular demonstra a transformação de um conceito básico de programação em um software robusto. O jogo original ilustra a essência do jogo da cobrinha, o desenvolvido, porém, ergue sobre essa base uma aplicação completa, com arquitetura sólida, persistência de dados, autenticação, interface moderna e uma série de recursos que enriquecem tanto a mecânica quanto a usabilidade. Em síntese, a versão modular não só preserva a diversão do jogo original, mas a amplifica, elevando-a a um patamar que atende às expectativas de um produto pronto para o usuário final.

### *C. Aplicação da Programação Orientada a Objetos*

Durante o desenvolvimento do sistema bancário, a Programação Orientada a Objetos (POO) foi essencial para deixar o código mais organizado, reutilizável e fácil de manter. Os quatro pilares da POO (abstração, encapsulamento, herança e polimorfismo) foram aplicados de forma prática para melhorar a estrutura e o funcionamento do sistema, a saber:

- **Abstração**: A abstração permite escolher quais atributos e métodos eram essenciais para representar uma determinada ”entidade”, permitindo dessa forma ocultar detalhes desnecessários para sua representação. No codigo a abstração está presente em vários níveis, tanto na estrutura das classes quanto em métodos específicos, a classe AuthService, por exemplo, é uma abstração do processo de autenticação. A TelaLogin, ao utilizá-la, não precisa saber os detalhes de como um login funciona, ela não sabe que um arquivo JSON é lido, que uma senha é comparada ou que uma sessão é criada em um dicionário, a complexidade é escondida. A TelaLogin simplesmente chama o método login(nome, senha) e espera um resultado, como obtemos esse resultado está abstraído.
- **Encapsulamento**: O encapsulamento permite proteger o estado de um objeto através do uso de atributos privados/protegidos, expondo apenas métodos verificados para acesso ou modificação de tais atributos. Em Usuario, os campos nome, senha e score são protegidos e só acessíveis por meio de property e setters. Para modificar o score, o código não permite uma atribuição direta a score, em vez disso, ele usa o setter de score. Aqui há uma validação (”novo-score é maior que self.score?”) para garantir a integridade do dado, impedindo que o recorde de um jogador seja substituído por uma pontuação menor. Os dados estão protegidos e sua modificação obedece a regras definidas dentro da própria classe.
- **Herança**: A herança possibilita criar classes que herdam atributos e métodos de uma outra classe base (conhecida como pai, geralmente), podendo acrescentar ou modificar comportamentos. No programa, BaseService já sabe ler e gravar o arquivo JSON, CadastroService, AuthService e RankingService herdam essa capacidade, usando ”carregar-usuarios()” ou ”salvar-usuarios() sem repetir código.
- **Polimorfismo**: O uso dele permitiu que diferentes classes fossem tratadas por uma interface comum, garantindo que cada uma responda de forma específica ao mesmo comando. No código do jogo, um exemplo conceitual disso se encontra no fato de que tanto a classe Snake quanto a classe Apple possuem um método chamado ”desenhar(self, tela)”, de forma que, no loop principal do jogo, o motor chama ”self.snake.desenhar(self.tela)” e ”self.apple.desenhar(self.tela)”. Embora Snake e Apple sejam classes completamente diferentes, ambas respondem ao mesmo comando de desenhar da sua forma, demonstrando o princípio de tratar objetos diferentes de uma maneira uniforme.

Os exemplos relatados acima representam apenas alguns casos dentre os outros vários presentes no código no qual há a aplicação da POO.

## III. Resultados

Esse projeto permitiu o desenvolvimento de novas funcionalidades e aplicações a partir de uma versão base e não modularizada de um protótipo, algo que culminou na entrega de uma aplicação aprimorada que exemplifica a aplicação de alguns princípios de engenharia de software, mas agora no âmbito de um jogo virtual . Nesse sentido, convém apresentar o produto final criado, com a apresentação de imagens que ilustram as sequências de telas e etapas com a qual o usuário se deparará. Inicialmente, como mostra a figura 1, o usuário se depara com uma tela de login, caso ele já possua um cadastro, ele será enviado para a tela de menu do jogo. Caso não possua, ele deve cadastrar uma conta, com numero de usuário e senha escolhida.

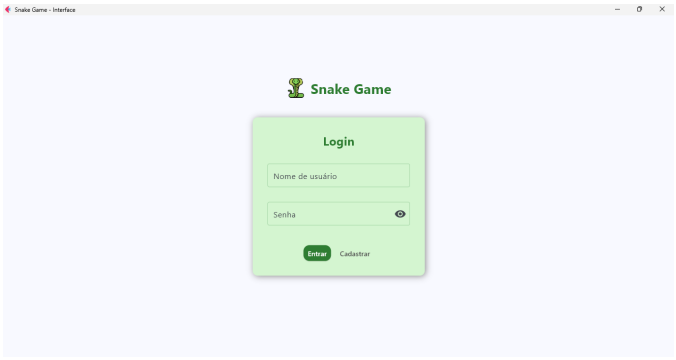


Fig. 1. Tela Inicial.

Impedimentos de login para usuários ainda não cadastrados, ou que estão inserindo senhas / nome de usuário incorretos, limitam o acesso ao sistema, como pode ser visto na figura 2. Essa etapa é de vital importância para o funcionamento do sistema, pois permite a validação correta e robusta dos dados, além de garantir uma autenticação segura do usuário.

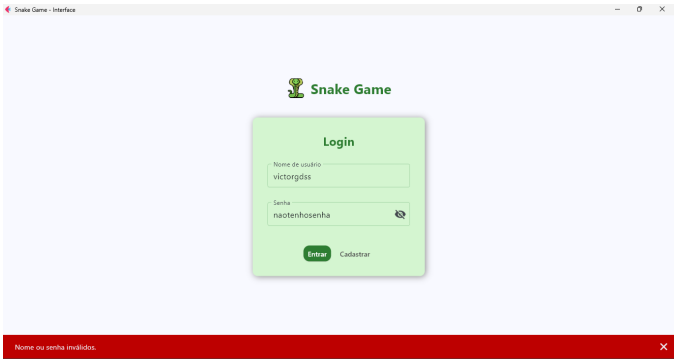


Fig. 2. Tentativa de login sem cadastro.

Em seguida, partimos para a tela de cadastro, por meio do uso do botão ”Cadastrar”. Nele, devemos criar um nome de usuário (diferente de um já exixtente) e uma senha. As figuras 3 e 4 ilustram a criação de um cadastro e o respectivo retorno de uma mensagem, pelo sistema, indicando o sucesso na realização do cadastro.

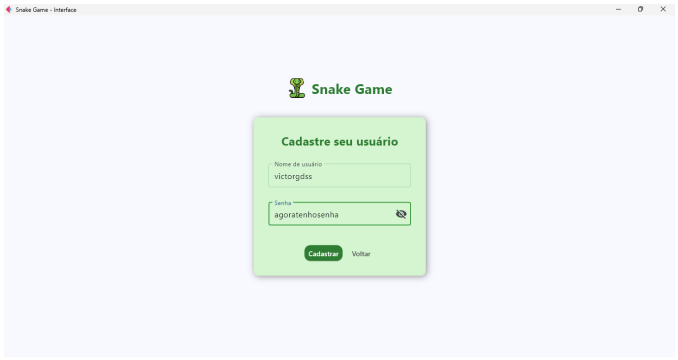


Fig. 3. Cadastro pt1

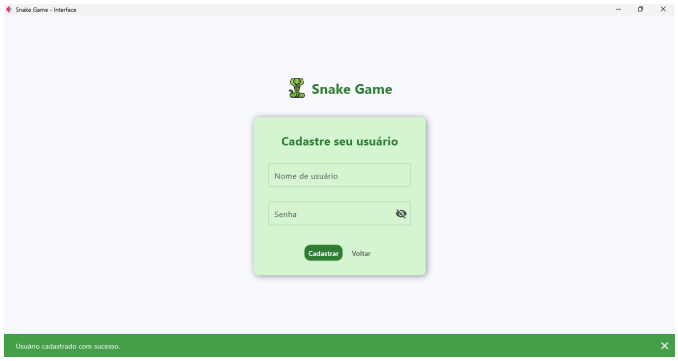


Fig. 4. Cadastro pt2

Com o cadastro do usuário finalizado, momento em que uma mensagem notificando a conclusão do cadastro é apresentada na tela, podemos realizar o login no sistema, algo que pode ser visto na figura 5. Também nesse momento, de realização do login, é retornada uma mensagem do sistema informando o sucesso na realização do login, como pode ser visto na mesma figura.

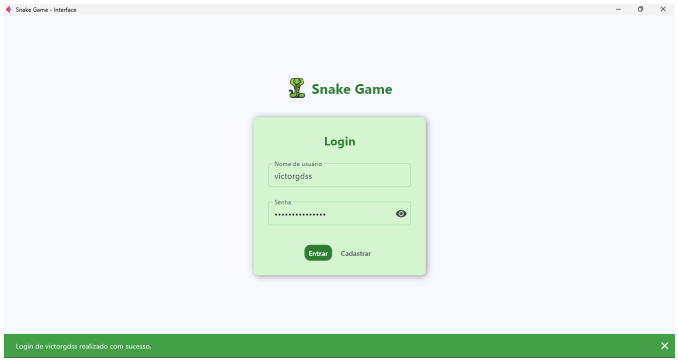


Fig. 5. Login

A realização do login parmite acessar o ambiente do jogo. A primeira tela com a qual o usuário logado terá contato é a tela de menu, representada na figura 6. Na tela de menu temos 3 opções, a saber: Jogar, ver o Ranking ou efetuar o Logout.

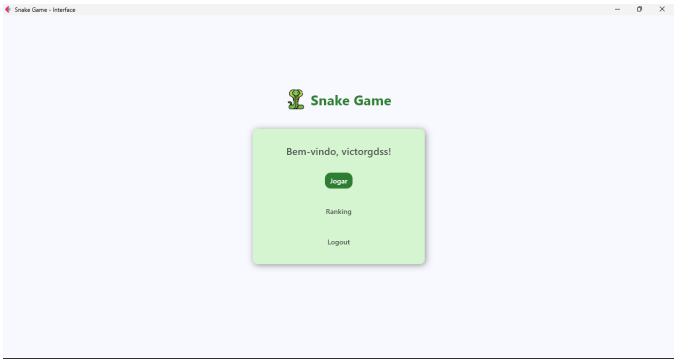


Fig. 6. Menu

Ao seleciotar a aba ”jogar” temos a possibilidade de iniciar o jogo. A mecânica central do jogo da cobrinha é baseada em três elementos principais: movimentação contínua, crescimento progressivo e colisão. O jogador deve direcionar a cobrinha utilizando comandos simples (setas direcionais), com o objetivo de coletar o máximo possível de itens sem colidir com as bordas do ambiente ou com o próprio corpo. A medida que a cobrinha consome itens, ela aumenta de tamanho, o que intensifica o desafio. O espaço jogável permanece constante, o que significa que o jogador precisa dde utilizar cada vez mais da estratégia e precisão para evitar colisões. A figura 7 ilustra o design do jogo.


Fig. 7. design jogo

No contexto do jogo da cobrinha, o game over ocorre quando a cobrinha colide com ela mesma ou quando atinge os limites do mapa. A autocolisão, que é o cenário mais simbólico de término da partida, acontece quando o jogador faz a cobrinha se mover em um caminho que a leva diretamente a interceptar seu próprio corpo. Essa forma de derrota carrega uma mensagem interessante: o fim se dá não por um agente externo, mas por uma consequência direta das próprias ações do jogador, reforçando a ideia de responsabilidade e planejamento dentro do jogo e ensinando de forma lúdica conceitos como antecipação, gestão de espaço e autocontrole.

Assim, no jogo da cobrinha, o game over não é apenas um fim, mas um feedback de que uma escolha tática levou a um impasse. Aqui, cabe ao usuário decidir jogar novamente através do botão ”ENTER”, buscando adquirir uma pontuação maior e, consequentemente, ocupar uma melhor posição no ranking ou retprnar ao menu através do botão ”ESC”. A figura 8 ilustra a situação descrita.


Fig. 8. Game over

A tela de ranking do sistema é um componente crucial que exibe os três jogadores com as maiores pontuações, além da pontuação pessoal do usuário logado. A sua importância vai além da mera exibição de dados, atuando como um pilar fundamental para o engajamento do usuário e a rejogabilidade. Ao criar um ambiente de competição saudável, a tela de ranking motiva os jogadores a superarem seus próprios recordes e a almejarem uma posição de destaque entre os melhores. Essa funcionalidade transforma o ato de jogar de uma experiência isolada para um desafio contínuo, incentivando o aprimoramento de habilidades e prolongando o interesse do usuário pelo jogo. A apresentação do score pessoal ao lado dos líderes serve como uma forte motivação, apresentando um objetivo tangível a ser alcançado. A figura 9 ilustra o ranking inicial, enquanto a figura 10, representa o ranking atualizado após uma partida.


Fig. 9. Ranking inicial

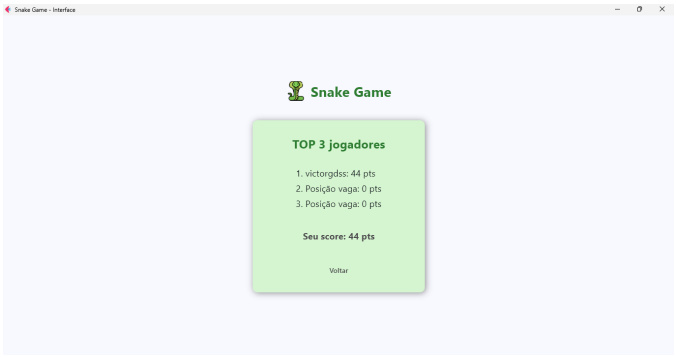


Fig. 10. Ranking atualizado

O jogo também conta com funcionalidades além das descritas. Durante a partida, o jogador pode pressionar a tecla ”ESPAÇO” a qualquer momento para pausar o jogo, algo que congela a movimentação da cobra e exibe um aviso centralizado na tela, conforme podemos ver na figura 11. Ao pressionar espaço novamente, o jogo é retomado do ponto onde parou. Outro destaque interessante encontra-se na trilha sonora ambiente que é reproduzida automaticamente enquanto o jogo roda, sendo pausada apenas ao pressionarmos ”ESPAÇO”.

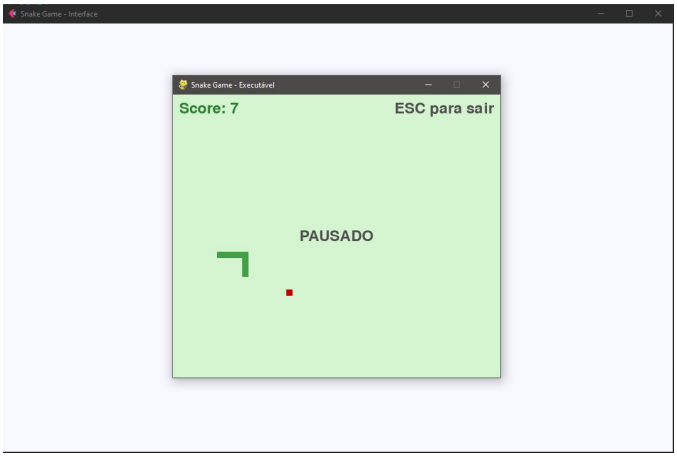


Fig. 11. Pause

Por último, quando alguém tenta entrar, o programa primeiro verifica se o nome de usuário digitado já existe no cadastro. Se não existir, ele para imediatamente e manda a tela de login avisar a pessoa com a mensagem ”Nome ou senha inválidos”. Se o nome for encontrado, o sistema dá o próximo passo e compara a senha que a pessoa digitou com a senha que está guardada para aquele usuário. Se as senhas ou usuário não baterem, ele também para e manda a tela mostrar o aviso de ”Nome ou senha inválidos”. Caso um usuário já cadastrato tente fazer cadastro, as validações internas retornam uma mensagem de ”Usuario já cadastrado” conforme pode ser visto na figura 12 .A importância disso é dupla, primeiro, para o jogador, fica muito mais fácil entender o que deu errado, em vez de receber uma mensagem genérica de ”erro no login”. Segundo, para o programa, essa separação de tarefas (uma parte verifica os dados, outra parte só mostra a mensagem) deixa tudo mais organizado e seguro, pois a tela de login nunca tem acesso direto às senhas salvas, apenas exibe os avisos que recebe.

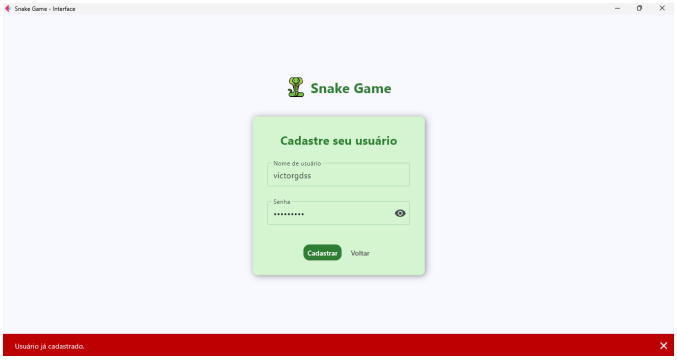


Fig. 12. Cadastro com usuario já ativo

## IV. Conclusão

A realização do projeto foi de grande valia para o grupo, permitindo acompanhar a transformação promovida partindo de uma versão inicial monolítica e enxuta até chegar a um sistema plenamente modularizado e com novas implementações, demonstrando, de forma exemplar, como boas práticas de desenvolvimento de software elevam tanto a qualidade quanto a escalabilidade de um projeto. Ao decompor o código em camadas bem definidas (utilitários, modelos, serviços, interface, roteamento e engine de jogo), não foram apenas desenvolvidas novas funcionalidades (como persistência de ranking, sessões de usuário, jogabilidade , etc) mas também tornou-se cada parte mais testável, legível e independente. Concluí-se, assim, que a evolução e transformação do código ilustra poderosamente como um design cuidadoso, apoiado em encapsulamento, herança, abstração e polimorfismo permite chegar em um software robusto, flexível e pronto para enfrentar requisitos cada vez mais complexos.

## References

[1] R. S. Bello. (2017) História e videogames: como os jogos eletrônicos podem ser pensados por historiadores. Acessado em 21 de junho de 2025. [Online]. Available: https://www.cafehistoria.com.br/historia-e-videogames/

[2] Sebrae. (2023) Mercado de games: tendências e oportunidades. Acessado em 21 de junho de 2025. [Online]. Available: https://sebrae.com.br/sites/PortalSebrae/artigos/mercado-de-games-tendencias-e-oportunidades,767cf253be2a6810VgnVCM1000001b00320aRCRD

[3] Latam Gateway. (2022) Saiba tudo sobre a história dos jogos eletrônicos no brasil. Acessado em 21 de junho de 2025. [Online]. Available: https://payments.latamgateway.com/pt/jogos-eletronicos-no-brasil/

[4] “Snack game,” https://github.com/lucaspimentab/Snack-Game, 2021, acessado em 21 de junho de 2025.

[5] DevMediaa, “Arquitetura de software: Introdução, camadas e concorrência,” 2025, vídeo de apresentação. Acessado em: 23 maio 2025. [Online]. Available: https://www.devmedia.com.br/arquitetura-de-software-introducao-camadas-e-concorrencia/26124

[6] DevMedia, “Introdução ao padrão mvc,” 2025, acessado em: 23 maio 2025. [Online]. Available: https://www.devmedia.com.br/introducao-ao-padrao-mvc/29308

[7] Pygame Community, “Pygame news,” https://www.pygame.org/news, 2024, notícias, atualizações e lançamentos da biblioteca Pygame. [Online]. Available: https://www.pygame.org/news
[8] Flet, “Flet documentation,” 2025, acessado em: 27 maio 2025. [Online]. Available: https://flet.dev/docs/
[9] Hashtag Treinamentos, “Pygame com python: Crie jogos e aprenda programação,” 2025, acessado em 22 de junho de 2025. [Online]. Available: https://www.hashtagtreinamentos.com/pygame-python
[10] D. Olímpio, “Jogo da cobrinha,” https://github.com/daniolimpiof/jogo-da-cobrinha, 2023, desenvolvimento básico de um Snake Game com Pygame. [Online]. Available: https://github.com/daniolimpiof/jogo-da-cobrinha