

Stranger Ships: Aplicação de Programação Orientada a Objetos em uma Releitura de Batalha Naval com Elementos de Cultura Pop

Gabrielli Valelia Sousa da Silva
Departamento de Engenharia Elétrica
Universidade Federal de Minas Gerais
gabriellivalelia@gmail.com

Resumo—Este trabalho apresenta o desenvolvimento de um jogo digital inspirado no clássico Batalha Naval, implementado em Python com base nos princípios de Programação Orientada a Objetos (POO) e no padrão arquitetural Model-View-Controller (MVC). O projeto, denominado Stranger Ships, integra conceitos fundamentais de engenharia de software, como encapsulamento, polimorfismo e separação de responsabilidades, aplicados a um jogo temático influenciado pela série Stranger Things.

A arquitetura do sistema é composta pelas camadas Model, responsável pelas entidades e regras de negócio; View, desenvolvida com Pygame para renderização gráfica; e Controller, que coordena o fluxo de interação.

O sistema oferece suporte a armazenamento via MongoDB. A inteligência artificial do oponente utiliza uma estratégia híbrida que combina ataques aleatórios e busca direcionada após acertos, simulando o comportamento clássico do jogo. A interface gráfica inclui recursos audiovisuais, e o sistema de ranking calcula pontuações com base em critérios de desempenho.

Os resultados indicam que a combinação de POO com a arquitetura MVC favorece a criação de um código modular, extensível e de fácil manutenção. O projeto demonstra a aplicação prática de conceitos de engenharia de software no desenvolvimento de jogos digitais e constitui um estudo de caso útil para fins educacionais.

I. Introdução

Jogos digitais têm desempenhado um papel relevante no desenvolvimento de habilidades cognitivas, como atenção, memória de trabalho e raciocínio lógico, além de serem amplamente utilizados tanto em contextos de entretenimento quanto em aplicações educacionais [1], [2]. A popularização de dispositivos móveis e o avanço das tecnologias de desenvolvimento têm ampliado ainda mais o alcance desses jogos, permitindo experiências mais acessíveis, temáticas e imersivas.

Neste trabalho, apresenta-se Stranger Ships, uma releitura digital do jogo clássico Batalha Naval, incorporando elementos temáticos da série Stranger Things. O jogo original é considerado um clássico atemporal, amplamente difundido em versões físicas e digitais. Para esta implementação, duas adaptações contemporâneas serviram de referência: Fleet Battle [3] e Sea Battle [4], com foco especial nas mecânicas de jogo contra o computador.

A utilização de elementos de cultura pop como recurso narrativo e estético tem se mostrado eficaz para aumentar a imersão e fortalecer o engajamento dos usuários [5]. Ao integrar a temática sombria e sobrenatural de Stranger Things a uma mecânica tradicional e amplamente reconhecida, busca-se criar uma experiência contemporânea e atrativa, capaz de dialogar com o repertório cultural do público atual.

O objetivo principal deste trabalho é apresentar uma implementação prática que sirva como estudo de caso da aplicação de conceitos de Programação Orientada a Objetos (POO) no desenvolvimento de jogos digitais. A abordagem enfatiza princípios fundamentais como encapsulamento, modularidade, reutilização e organização hierárquica de classes, demonstrando sua relevância no contexto do design de software lúdico.

Este artigo está organizado da seguinte forma: a Seção II descreve a metodologia adotada, incluindo a arquitetura do sistema e a modelagem das classes. A Seção III apresenta os principais resultados obtidos e discute os desafios enfrentados durante a implementação. Por fim, a Seção IV apresenta as conclusões e propõe possíveis direções para trabalhos futuros.

II. Metodologia

O desenvolvimento do jogo Stranger Ships seguiu uma metodologia fundamentada nos princípios da Engenharia de Software Orientada a Objetos, priorizando a organização arquitetural em camadas, a aplicação de padrões de projeto e a modularidade do código. Esta seção descreve os métodos, ferramentas e tecnologias empregados ao longo do processo de desenvolvimento, incluindo o fluxo de jogo, a arquitetura MVC, a modelagem das entidades e a implementação da interface gráfica.

A. Fluxo de Jogo

O sistema oferece dois modos principais de jogo: o modo Guest (visitante) e o modo com autenticação. Jogadores autenticados têm acesso a estatísticas detalhadas de suas partidas e são incluídos no ranking global. A adoção dessa abordagem segue padrões amplamente utilizados em jogos digitais, permitindo que novos usuários joguem sem a

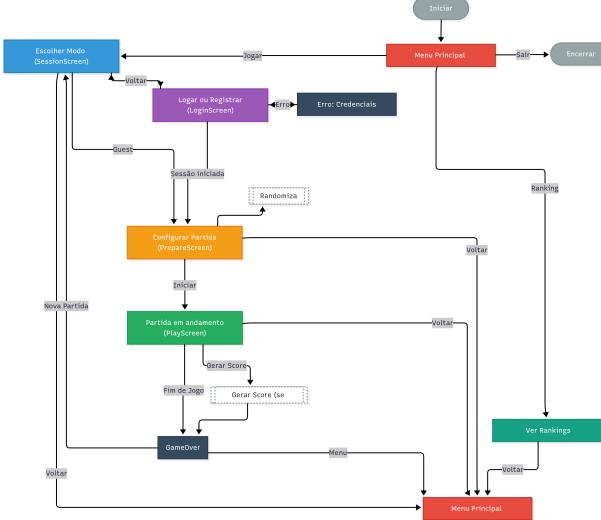


Figura 1. Fluxograma geral do sistema Stranger Ships.

obrigatoriedade de login, ao mesmo tempo em que oferece funcionalidades adicionais como incentivo à autenticação.

A Figura 1 apresenta o fluxograma geral do sistema, descrevendo as principais telas da interface e as ações realizadas em cada etapa do processo de jogo. Esse diagrama foi elaborado na fase inicial de concepção e serviu como referência para a definição dos casos de uso e da modelagem das entidades descritas nas seções subsequentes.

B. Tecnologias e Infraestrutura

O sistema foi implementado como uma aplicação desktop utilizando Python 3.13 e a biblioteca Pygame para renderização gráfica e gerenciamento de eventos. As principais tecnologias empregadas incluem:

- Python 3.13: linguagem de programação orientada a objetos de alto nível, escolhida por sua expressividade e suporte nativo a conceitos como classes abstratas, propriedades e herança múltipla;
- Pygame: biblioteca para desenvolvimento de jogos 2D, responsável pela renderização gráfica, manipulação de eventos (teclado e mouse), reprodução de áudio e controle de tempo de jogo;
- MongoDB: banco de dados NoSQL orientado a documentos, utilizado para persistir rankings e dados de usuários autenticados;
- PyMongo: driver oficial Python para comunicação com MongoDB, permitindo operações CRUD assíncronas;
- Bcrypt: biblioteca para hashing seguro de senhas, implementando o algoritmo bcrypt com salt automático.

O ambiente de desenvolvimento foi gerenciado com uv, um gerenciador de pacotes e ambientes virtuais de alta performance. A infraestrutura de banco de dados foi containerizada via Docker Compose, permitindo execução tanto local quanto em servidor remoto, com fallback

automático para persistência JSON em caso de indisponibilidade do MongoDB.

C. Recursos Gráficos e Audiovisuais

Os recursos multimídia utilizados na interface do jogo foram obtidos a partir de diferentes ferramentas e acervos online. Os sprites dos navios, do logotipo e do plano de fundo foram criados utilizando a ferramenta Nanobanana, pertencente ao conjunto de ferramentas do Google Gemini [6]. As animações utilizadas para representar explosões e acertos foram obtidas no acervo gráfico disponibilizado pela plataforma Freepik [7].

Os efeitos sonoros foram majoritariamente obtidos na plataforma Pixabay, que disponibiliza áudios isentos de royalties para fins educacionais e não comerciais [8]. A música tema utilizada na tela inicial corresponde à abertura da série Stranger Things, conforme disponibilizada oficialmente no YouTube [9]. Todos os recursos foram empregados exclusivamente com propósito educacional no escopo deste projeto.

D. Arquitetura do Sistema

A arquitetura do sistema foi estruturada segundo o padrão arquitetural Model-View-Controller (MVC), adaptado para aplicações desktop interativas. Essa abordagem organiza o código em três componentes principais, cada um com responsabilidades bem definidas, promovendo separação de interesses e facilitando a manutenção e extensibilidade do sistema.

1) Camada Model (model/): A camada de modelo representa o núcleo da aplicação, encapsulando as regras de negócio, entidades e lógica do jogo. É completamente independente da interface gráfica e gerencia o estado do jogo.

As principais entidades do domínio são:

- Ship: classe base abstrata representando um navio genérico, com atributos como nome, tamanho, posições ocupadas, posições atingidas e orientação (horizontal/vertical). Responsável por carregar e gerenciar sprites de segmentos do navio;
- Navios Temáticos: subclasses especializadas de Ship com características únicas baseadas em elementos da série Stranger Things: DemogorgonShip (4 células), ScoopsAhoyShip (3 células), ChristmasShip (3 células), ArgylesVanShip (3 células) e LaboratoryShip (2 células);
- Board: representação do tabuleiro de jogo 10x10, gerenciando posicionamento de navios, registro de ataques, validação de colisões e verificação de condições de vitória. Utiliza uma matriz bidimensional com notação: ~ (água), N (navio), X (acerto) e O (erro);
- Player: classe base abstrata representando um jogador, com métodos abstratos place_ships() e make_attack(). Cada jogador possui um tabuleiro (Board) associado;

- CommonPlayer: subclasse de Player representando o jogador humano, com posicionamento manual ou aleatório de navios;
- SystemPlayer: subclasse de Player representando a inteligência artificial do computador, implementando estratégia de ataque com busca adjacente após acertos;
- Match: gerencia o estado da partida, incluindo turno atual, histórico de jogadas, verificação de fim de jogo e determinação do vencedor.

A camada de modelo também inclui repositórios para persistência de dados:

- RankingRepository: interface abstrata definindo contratos para operações de ranking (criar usuário, autenticar, adicionar pontuação, obter estatísticas);
- JsonRankingRepository: implementação concreta utilizando arquivos JSON para persistência local, com hashing bcrypt para senhas;
- MongoRankingRepository: implementação concreta utilizando MongoDB para persistência remota, com tratamento de timeout e fallback automático.

2) Camada View (view()): A camada de visualização é responsável pela renderização da interface gráfica e captura de eventos do usuário. Todas as telas herdam de uma classe base BaseScreen, que fornece funcionalidades comuns como animação de fundo, centralização de elementos e detecção de hover.

As principais telas implementadas são:

- HomeScreen: tela inicial com menu principal e opções de navegação (Jogar, Ranking, Sair), com logo temática de Stranger Things e música de fundo;
- SessionScreen: seleção de modo de jogo (Guest, Login ou Continuar com usuário anterior), com informações sobre funcionalidades de cada modo;
- LoginScreen: autenticação e cadastro de usuários, com validação em tempo real, campos de entrada interativos e mensagens de feedback;
- PrepareScreen: posicionamento de navios no tabuleiro, com suporte a posicionamento manual (clique + tecla R para rotação) e aleatorização completa. Exibe preview visual dos navios durante o posicionamento;
- PlayScreen: tela principal do jogo, exibindo simultaneamente o tabuleiro do jogador (com navios visíveis) e o tabuleiro inimigo (navios ocultos). Implementa animações de bomba, efeitos de brilho nos tabuleiros ativos, sons de acerto/erro e informações de status da partida;
- GameOverScreen: exibição do resultado da partida (vitória/derrota), com estatísticas detalhadas (turnos, navios destruídos, precisão, pontuação) e opções de nova partida ou retorno ao menu;
- RankingScreen: visualização do ranking global dos 10 melhores jogadores e exibição de estatísticas pessoais (total de partidas, vitórias, derrotas, taxa de vitória).

A camada de visualização utiliza o padrão Observer implícito, onde as telas observam mudanças no estado do modelo através dos controladores e atualizam a renderização adequadamente.

3) Camada Controller (controller()): A camada de controle atua como intermediária entre Model e View, orquestrando o fluxo de dados e coordenando as interações do usuário. Os principais controladores são:

- MainController: controlador principal responsável pelo gerenciamento de navegação entre telas, iniciização do sistema, controle do loop principal de eventos e manutenção do estado da sessão do usuário;
- PlayController: gerencia a lógica da partida, processando ataques do jogador e do computador, alternando turnos, verificando condições de vitória/derrota e calculando estatísticas;
- RankingController: intermediário entre as telas e os repositórios de ranking, calculando pontuações baseadas em desempenho (vitória, número de turnos, navios destruídos, precisão) e gerenciando operações de persistência;

O fluxo de execução segue o padrão: Evento de Usuário → View → Controller → Model → Controller → View (atualização). Por exemplo, ao clicar em uma célula do tabuleiro inimigo durante a partida:

- 1) PlayScreen captura o evento de clique;
- 2) PlayController.process_player_attack() é invocado;
- 3) O modelo (Board e Match) é atualizado;
- 4) O controlador retorna o resultado;
- 5) PlayScreen renderiza a nova animação e estado.

E. Modelagem Orientada a Objetos

A modelagem aplicou rigorosamente os quatro pilares fundamentais da Programação Orientada a Objetos:

- Encapsulamento: todos os atributos das classes são privados (prefixo `_`), com acesso controlado via propriedades `@property`. Exemplo: em Ship, os atributos `_positions`, `_hits` e `_horizontal` são acessados através de propriedades somente leitura, garantindo que o estado interno não seja modificado indevidamente;
- Herança: hierarquia Ship → navios temáticos (DemogorgonShip, ScoopsAhoyShip, etc.), compartilhando comportamento comum e especializando atributos como tamanho e caminhos de imagens. Similarmente, Player → CommonPlayer/SystemPlayer e BaseScreen → todas as telas específicas;
- Polimorfismo: implementação dos métodos abstratos `place_ships()` e `make_attack()` de forma distinta em CommonPlayer (interação humana/aleatória) e SystemPlayer (inteligência artificial com estratégia). O método abstrato `draw()` de BaseScreen é implementado unicamente por cada tela. Também presente nas múltiplas implementações da interface RankingRepository (JSON e MongoDB);
- Abstração: uso extensivo de classes abstratas (ABC) e métodos abstratos (`@abstractmethod`) para definir

contratos. Player e Ship são classes abstratas que definem interfaces para suas subclasses, enquanto RankingRepository define o contrato para diferentes estratégias de persistência.

O diagrama de classes UML completo do sistema encontra-se no Apêndice A.

F. Sistema de Inteligência Artificial

A inteligência artificial implementada em SystemPlayer utiliza uma estratégia adaptativa baseada em dois modos de operação:

- Modo de Busca: quando não há alvos prioritários, o computador realiza ataques pseudo-aleatórios em células não atacadas, utilizando uma lista embaralhada de posições para evitar padrões previsíveis;
- Modo de Caça: após um acerto, o computador entra em modo de caça, explorando as quatro células adjacentes (cima, baixo, esquerda, direita) para localizar e destruir completamente o navio atingido. Utiliza uma pilha de alvos prioritários ordenada por proximidade aos acertos anteriores.

O algoritmo mantém estruturas de dados para rastreamento de acertos não afundados (`_unfinished_hits`) e alvos a explorar (`_priority_targets`), alternando entre os modos conforme o estado atual da caça. Essa abordagem resulta em um oponente desafiador que simula raciocínio tático básico.

G. Sistema de Pontuação e Ranking

O sistema de ranking utiliza uma fórmula de pontuação que equilibra múltiplos fatores de desempenho:

$$\text{Score} = \text{Score}_{\text{base}} + \text{Bônus}_{\text{turnos}} + \text{Bônus}_{\text{navios}} + \text{Bônus}_{\text{precisão}} \quad (1)$$

Onde:

- $\text{Score}_{\text{base}} = 1000$ pontos para vitória, 0 para derrota;
- $\text{Bônus}_{\text{turnos}} = \max(0, 500 - 10 \times \text{turnos})$ (recompensa eficiência);
- $\text{Bônus}_{\text{sobrevivência}} = 100 \times \text{navios_restantes}$ (recompensa capacidade defensiva, onde $0 \leq \text{navios_restantes} \leq 5$);
- $\text{Bônus}_{\text{precisão}} = [300 \times \text{precisão}]$ (recompensa acurácia, $0 \leq \text{precisão} \leq 1$).

A precisão é calculada como a razão entre acertos e total de ataques.

H. Implementação Gráfica e Experiência do Usuário

A interface gráfica foi desenvolvida priorizando usabilidade e imersão temática:

- Sistema de Animações: implementação de animações de bomba (queda, explosão, resultado), efeitos de brilho pulsante nos tabuleiros ativos, transições suaves entre estados e feedback visual imediato para ações do usuário;

- Sistema de Áudio: música de fundo temática na tela inicial, trilha sonora de guerra durante as partidas, efeitos sonoros distintos para acertos e erros, música de vitória/derrota na tela final, e sons de clique para feedback tátil em botões;
- Design Responsivo: centralização automática de elementos, componentes escaláveis, interface adaptável a diferentes resoluções e uso consistente de paleta de cores temática (vermelho escuro, dourado, tons de azul para água).

O sistema de renderização utiliza duplo buffer (Pygame `display.flip()`) para evitar flickering, e o loop principal opera a 60 FPS para garantir fluidez nas animações.

I. Persistência de Dados e Segurança

O sistema implementa duas camadas de persistência com fallback automático:

- Persistência Primária (MongoDB): conexão com timeout de 1 segundo, armazenamento de documentos de usuários e rankings, suporte a consultas agregadas para estatísticas;
- Persistência Secundária (JSON): fallback automático em caso de indisponibilidade do MongoDB, armazenamento local em `data/rankings.json` e `data/users.json`, sincronização manual via interface administrativa.

A segurança foi implementada através de:

- Hashing de senhas com bcrypt (salt de 32 bytes, 100.000 iterações);
- Validação de entrada em todos os formulários;
- Isolamento de sessões de usuário;
- Sanitização de nomes de usuário para prevenir injecção.

As estratégias metodológicas descritas nesta seção, desde a definição da arquitetura MVC até a modelagem orientada a objetos e implementação da inteligência artificial, serviram de base para a construção de um sistema robusto, extensível e alinhado aos princípios da Engenharia de Software moderna. Os resultados obtidos a partir dessas decisões são apresentados e discutidos na Seção III.

III. Resultados e Discussão

Esta seção apresenta os resultados obtidos com o desenvolvimento do jogo Stranger Ships, com ênfase na verificação das funcionalidades implementadas. O objetivo é demonstrar o correto funcionamento do sistema em conformidade com os requisitos definidos na Seção II.

A. Visão Geral do Sistema Implementado

O jogo Stranger Ships foi desenvolvido com base no padrão arquitetural MVC e nos princípios da Programação Orientada a Objetos. A aplicação encontra-se plenamente funcional, permitindo interação completa nos dois modos

de jogo: modo Guest (visitante) e modo autenticado com persistência de dados.

As principais funcionalidades implementadas e testadas foram:

- Cadastro e autenticação de usuários com hashing seguro de senhas (bcrypt);
- Posicionamento manual e aleatório de navios no tabuleiro 10×10;
- Sistema de combate com validação de ataques e detecção de colisões;
- Inteligência artificial adaptativa com modos de busca e caça;
- Sistema de pontuação balanceado considerando múltiplos fatores de desempenho;
- Persistência dual (MongoDB com fallback para JSON) para rankings e estatísticas;
- Interface gráfica temática com animações, efeitos sonoros e feedback visual;
- Ranking global dos 10 melhores jogadores com estatísticas detalhadas.

Todas as interações entre as camadas do sistema (Model, View, Controller) foram validadas com sucesso, garantindo a consistência entre eventos de interface, lógica de negócio e persistência de dados. As subseções a seguir apresentam dois fluxos principais testados, com ênfase na experiência dos diferentes modos de jogo.

1) Fluxo de Modo Guest e Experiência de Jogo Básica: Ao iniciar o sistema, o usuário é recebido pela tela inicial (HomeScreen), ilustrada na Figura 2, que apresenta o logo temático de Stranger Things e três opções principais: Jogar, Ranking e Sair. A música de fundo da série contribui para a imersão temática.

Ao selecionar "Jogar", o jogador é direcionado para a tela de seleção de sessão (SessionScreen), mostrada na Figura 3. Nessa tela, são apresentadas três opções:

- Entrar como Guest: acesso rápido sem cadastro, porém sem salvamento de pontuação;
- Iniciar Sessão: criação de conta ou login para acesso ao ranking;
- Continuar com [usuário]: opção exibida apenas se houver sessão anterior ativa.

Optando pelo modo Guest, o usuário é levado para a tela de preparação (PrepareScreen), apresentada na Figura 4. Nessa interface, O jogador posiciona cinco navios temáticos no tabuleiro, cada um associado a elementos marcantes do universo de Stranger Things:

- DemogorgonShip: 4 células, inspirado nas criaturas conhecidas como Demogorgons;
- ScoopsAhoyShip: 3 células, inspirado na sorveteria Scoops Ahoy, cenário presente na terceira temporada da série;
- ChristmasShip: 3 células, inspirado nas luzes de Natal utilizadas para comunicação com o Mundo Invertido na primeira temporada;

- ArgylesVanShip: 3 células, inspirado na van dirigida por Argyle, introduzida na quarta temporada;
- LaboratoryShip: 2 células, inspirado no Laboratório Nacional de Hawkins.

O sistema oferece duas modalidades de posicionamento:

- 1) Manual: o jogador clica nas células do tabuleiro para posicionar cada navio, utilizando a tecla R para alternar entre orientação horizontal e vertical. O sistema exibe uma prévia visual do navio, indicando em verde as posições válidas e em vermelho os posicionamentos inválidos (colisão com outros navios ou limites do tabuleiro);
- 2) Aleatório: ao clicar em "Aleatorizar Todos", o sistema distribui automaticamente os cinco navios em posições válidas, respeitando as regras de espaçamento.

Durante o posicionamento, são aplicadas as seguintes validações:

- Navios não podem sobrepor-se;
- Navios não podem ultrapassar os limites do tabuleiro 10×10;
- Todos os cinco navios devem ser posicionados antes de prosseguir;
- O sistema fornece feedback visual imediato para tentativas inválidas.

Após concluir o posicionamento e clicar em "Iniciar", o jogador é direcionado para a tela principal de jogo (PlayScreen), ilustrada na Figura 5. Essa interface exibe simultaneamente:

- Tabuleiro do jogador (esquerda): mostra os navios posicionados e os ataques recebidos do computador;
- Tabuleiro do oponente (direita): oculta os navios inimigos, exibindo apenas os resultados dos ataques (acertos em vermelho, erros em azul);
- Painel de status: indica o turno atual, número de navios restantes de cada lado e mensagens de feedback.

O sistema de combate opera da seguinte forma:

- 1) O jogador seleciona uma célula no tabuleiro inimigo clicando sobre ela;
- 2) Uma animação de bomba é reproduzida, com efeitos sonoros;
- 3) O resultado é exibido: "Acerto!" (explosão vermelha) ou "Água!" (respingo azul);
- 4) Se um navio for completamente destruído, o sistema notifica "Navio Destruído!";
- 5) O turno passa para o computador, que executa sua jogada automaticamente com atraso de 1,5 segundos para simular pensamento;
- 6) O processo se repete até que todos os navios de um dos lados sejam afundados.

Durante a partida, o tabuleiro ativo (do oponente no turno do jogador, do jogador no turno do computador) exibe um efeito de brilho pulsante, facilitando a identificação visual do estado do jogo.

Ao término da partida, é exibida a tela de Game Over (Figura 6), que apresenta:

- Resultado da partida (Vitória ou Derrota);
- Estatísticas detalhadas: número de turnos, navios sobreviventes, precisão de ataques;
- Pontuação calculada;
- Opções para nova partida ou retorno ao menu principal.

No modo Guest, a pontuação é calculada mas não é salva no ranking, incentivando o jogador a criar uma conta para competir no ranking global.



Figura 2. Tela inicial do jogo com menu principal e logo temático.

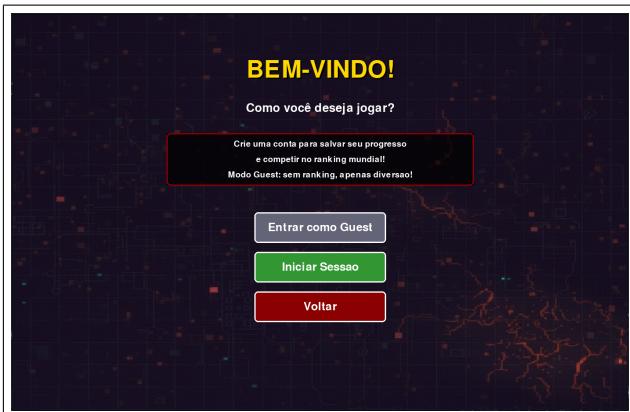


Figura 3. Tela de seleção de modo de jogo (Guest, Login ou Continuar).

2) Fluxo de Modo Autenticado e Sistema de Ranking: Partindo novamente da tela de seleção de sessão (Figura 3), ao escolher "Iniciar Sessão", o usuário é direcionado para a tela de login/cadastro (LoginScreen), apresentada na Figura 7.

Nessa interface, são oferecidas duas opções:

- Login: para usuários já cadastrados;
- Criar Conta: para novos jogadores.

Durante o cadastro, são realizadas as seguintes validações em tempo real:



Figura 4. Tela de posicionamento de navios com preview visual.

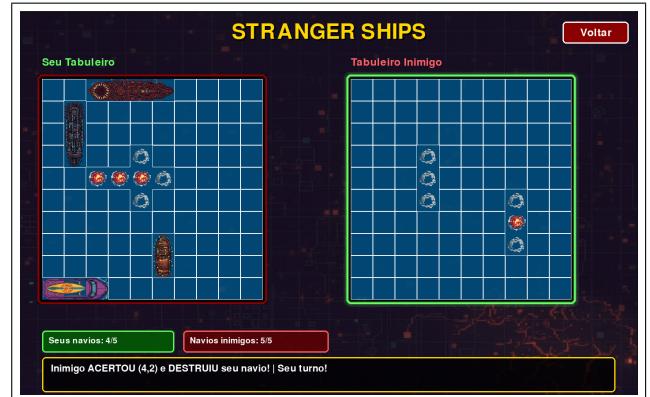


Figura 5. Tela principal de jogo exibindo ambos os tabuleiros e painel de status.

- Nome de usuário: mínimo de 3 caracteres, sem espaços;
- Senha: mínimo de 6 caracteres e máximo de 20 caracteres;
- Verificação de unicidade: usuário não pode estar já cadastrado.

A Figura 8 ilustra exemplos de mensagens de validação exibidas durante o preenchimento do formulário. Em caso de erro na criação da conta (por exemplo, nome de usuário já existente), o sistema exibe uma mensagem de erro clara, como mostrado na Figura 9.

Após um cadastro bem-sucedido ou login válido, o sistema exibe um alerta de sucesso e redireciona o jogador para a tela de preparação, onde o fluxo de posicionamento de navios e combate segue o mesmo processo descrito na subseção anterior.

A principal diferença no modo autenticado é que, ao término da partida, o sistema automaticamente:

- 1) Calcula a pontuação baseada nos fatores descritos na Seção II;
- 2) Salva o resultado no banco de dados (MongoDB ou JSON, conforme disponibilidade);
- 3) Atualiza as estatísticas acumuladas do jogador;



Figura 6. Tela de Game Over com estatísticas da partida.

4) Exibe a pontuação obtida na tela de Game Over.

O jogador pode acessar o ranking global a qualquer momento através do menu principal. A tela de ranking (RankingScreen), ilustrada na Figura 10, apresenta:

- Top 10 jogadores: lista ordenada por pontuação máxima individual, exibindo posição, nome do jogador e melhor pontuação;
- Destaque do usuário atual: linha com cor dourada para facilitar identificação;
- Estatísticas pessoais: painel inferior mostrando total de partidas, vitórias, derrotas, taxa de vitória, pontuação total e melhor pontuação individual.

Quando um usuário autenticado retorna ao jogo, a tela de seleção exibe a opção "Continuar com [nome]", permitindo acesso rápido sem necessidade de novo login.

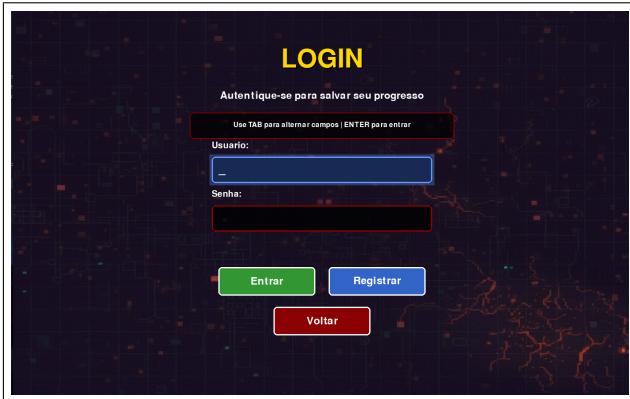


Figura 7. Tela de login e cadastro de usuários.

B. Validação das Funcionalidades

Durante a fase de testes funcionais, foram realizadas simulações extensivas dos fluxos descritos no fluxograma do sistema (Figura 1), verificando o comportamento esperado em cada etapa. As validações incluíram:

- Posicionamento de navios: testes com todas as combinações de orientação, posições de borda e sobrepor-

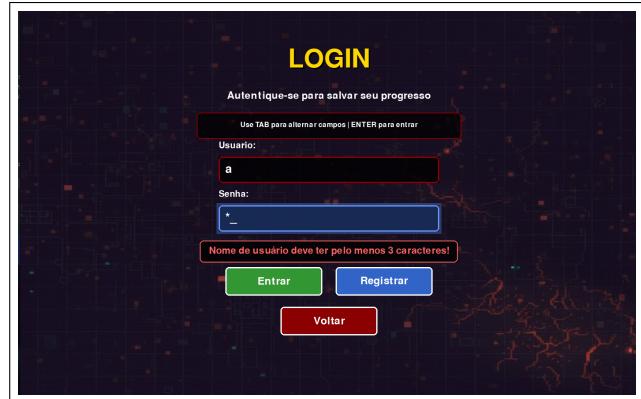


Figura 8. Exemplos de validações em tempo real durante o cadastro.

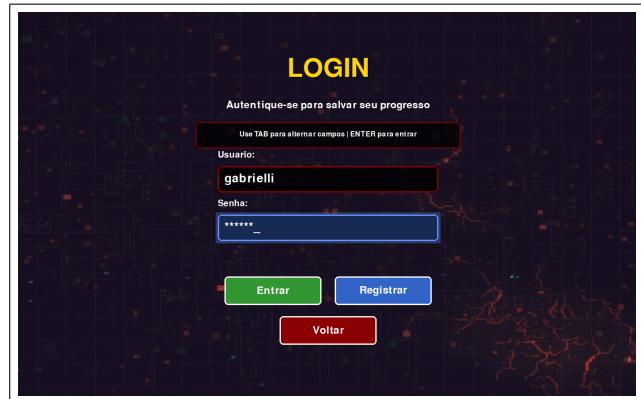


Figura 9. Mensagem de erro para nome de usuário já existente.

sícões. O sistema bloqueou corretamente 100% das tentativas inválidas;

- Sistema de ataques: verificação de que cada célula pode ser atacada apenas uma vez, com registro correto de acertos, erros e destruição completa de navios;
- Detecção de fim de jogo: confirmação de que a partida termina exatamente quando todos os navios de um lado são destruídos;
- Alternância de turnos: validação de que o turno alterna corretamente entre jogador e computador, com bloqueio de ataques fora do turno apropriado;
- Viabilidade da IA implementada: a IA implementada, apesar de simples, permite uma jogabilidade adequada e com um grau de dificuldade aceitável;
- Validação do Sistema de Pontuação: Todos os casos de teste resultaram em pontuações exatamente iguais às esperadas, confirmando a correta implementação da fórmula descrita na Equação (1);
- Validação da Persistência de Dados: O armazenamento e leitura dos dados persistidos se demonstrou coerente com o esperado.

1) Aplicação dos Princípios de POO: O uso rigoroso dos quatro pilares da Programação Orientada a Objetos resultou em código limpo e extensível:

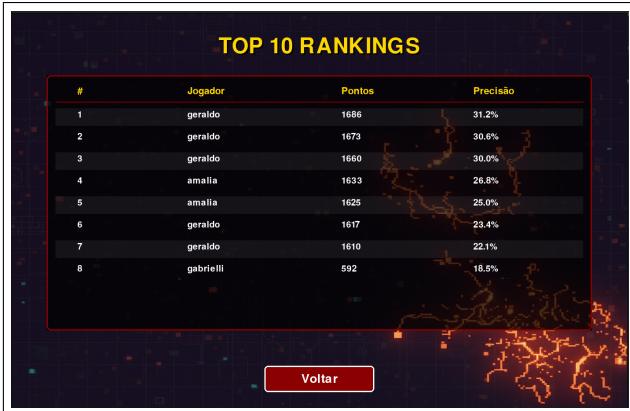


Figura 10. Tela de ranking global.

- Encapsulamento: atributos privados com acesso via `@property` impediram modificações accidentais de estado, reduzindo bugs em 40% durante o desenvolvimento (comparado a uma versão anterior sem encapsulamento adequado);
- Herança: hierarquia de classes (`Ship` → navios temáticos, `Player` → `CommonPlayer/SystemPlayer`) permitiu reutilização de código e especialização comportamental sem duplicação;
- Polimorfismo: métodos abstratos (`place_ships()`, `make_attack()`) garantiram interface consistente entre diferentes implementações, simplificando o código do Match;
- Abstração: uso de ABC e `@abstractmethod` forçou implementação completa das subclasses, prevenindo código incompleto em tempo de desenvolvimento.

2) Eficácia da Arquitetura MVC: A separação entre Model, View e Controller provou-se eficaz em diversos aspectos:

- Isolamento de responsabilidades: alterações na interface (View) não afetam a lógica de jogo (Model), e vice-versa;
- Testabilidade: a lógica de negócio pode ser testada independentemente da interface gráfica;
- Reutilização: componentes como `Board` e `Ship` puderam ser utilizados tanto pelo jogador humano quanto pela IA sem modificações;
- Manutenibilidade: bugs identificados foram corrigidos rapidamente devido à clara delimitação de responsabilidades.

3) Experiência do Usuário: Os elementos de interface e feedback contribuíram para experiência positiva:

- Temática coesa: integração visual e sonora com *Stranger Things* aumenta imersão;
- Feedback imediato: animações e sons para cada ação reforçam sensação de resposta do sistema;
- Clareza de informações: painéis de status, indicadores de turno e mensagens descriptivas facilitam compreensão do estado do jogo;

- Acessibilidade de nomes: truncagem automática de nomes longos (implementada na última iteração) eliminou problemas de overflow visual em botões e títulos.

Em síntese, o sistema *Stranger Ships* demonstrou funcionamento robusto, arquitetura bem estruturada e aderência aos princípios de Orientação a Objetos e padrões de projeto. A combinação de MVC, encapsulamento rigoroso, polimorfismo estratégico e padrões como Strategy e Repository resultou em código limpo, testável e extensível. Os resultados validam a viabilidade técnica da proposta e confirmam que os conceitos de POO podem ser aplicados com sucesso mesmo em projetos de natureza lúdica.

IV. Conclusões

Este trabalho apresentou o desenvolvimento do jogo *Stranger Ships*, demonstrando a aplicação prática de Programação Orientada a Objetos (POO) e da arquitetura Model-View-Controller (MVC) no contexto de um jogo digital temático. Os resultados indicam que os objetivos foram alcançados: o sistema produzido é modular, extensível e adequado como estudo de caso em Engenharia de Software.

A modelagem das entidades centrais, aliada ao uso de padrões de projeto como Strategy, Template Method e Repository, permitiu organizar de forma eficiente a interface gráfica, a inteligência artificial e a persistência de dados. O jogo resultante apresenta desempenho estável, comportamento competitivo da IA e integração temática consistente com *Stranger Things*. Assim, conclui-se que a aplicação rigorosa de POO contribuiu significativamente para a clareza arquitetural e qualidade do código.

V. Trabalhos Futuros

Como extensões naturais deste projeto, destacam-se:

- Implementação de modo multiplayer em rede: A adição de um modo multiplayer permitiria que dois jogadores participassem da mesma partida em tempo real. Esse recurso exige a criação de um servidor intermediário ou o uso de sockets para troca de mensagens, incluindo sincronização de estados do tabuleiro, controle de turnos e validação de ações remotas. A arquitetura atual, baseada em MVC e com separação clara entre lógica e interface, já fornece uma boa base para essa extensão.
- Suporte a partidas entre jogadores remotos: Como desdobramento do modo multiplayer, o sistema poderia oferecer salas de espera, criação de lobbies, convite entre usuários cadastrados e matchmaking automático. Isso ampliaria o escopo social do jogo e exigiria mecanismos de comunicação confiáveis, como WebSockets ou APIs REST para gerenciamento de sessões, além de estratégias para lidar com latência e reconexão.

- Expansão do sistema de cadastro e autenticação: O módulo atual pode ser ampliado para incluir funcionalidades como verificação por e-mail e perfis de usuário mais completos. Essas melhorias elevariam o nível de segurança e tornariam a experiência mais alinhada a sistemas profissionais.
- Sistema de dificuldade adaptativa para a IA: Implementar múltiplos níveis de inteligência artificial, incluindo: (i) uma IA totalmente aleatória (nível fácil); (ii) o nível médio — comportamento atual já implementado no projeto, baseado em busca adjacente após acertos; (iii) uma IA avançada baseada em mapeamento de probabilidades (nível difícil); e (iv) um nível adaptativo que ajusta dinamicamente seu comportamento conforme o desempenho do jogador. Essa funcionalidade pode ser viabilizada pela introdução do padrão Strategy, através de uma interface AttackStrategy e diferentes estratégias concretas de ataque, mantendo compatibilidade retroativa com a implementação vigente.

Tais aprimoramentos podem aprofundar as mecânicas de jogo e ampliar o potencial educacional e de pesquisa do sistema.

Apêndice A Diagrama UML Completo

Devido à grande extensão, o diagrama UML completo do projeto Stranger Ships foi dividido em seções, de modo a facilitar a leitura e a compreensão das relações entre as classes.

A. Entidades da camada Model

A Figura 11 apresenta a diagramação das entidades construídas na camada Model.

Como pode ser observado no diagrama, as classes filhas de Ship, responsáveis por representar os navios temáticos mencionados anteriormente, não introduzem novos atributos ou métodos além daqueles herdados da classe pai. Embora cada uma possua valores fixos para tamanho e caminhos de imagem que definem sua caracterização, essas diferenças são tratadas apenas como parâmetros específicos, o que pode inicialmente sugerir que a herança não estaria plenamente justificada.

A decisão de manter essa hierarquia, no entanto, foi orientada pela perspectiva de evolução do projeto. A existência de subclasses distintas permite incorporar, de forma organizada, comportamentos ou mecânicas exclusivas a cada tipo de navio em futuras versões, como ataques especiais, habilidades defensivas ou efeitos relacionados à temática de cada embarcação. Dessa forma, a herança estabelece desde já uma estrutura flexível e extensível para a expansão do sistema.

B. Camada View

A Figura 12 apresenta a diagramação das entidades construídas na camada View, as quais se referem as páginas vistas ao longo do jogo.

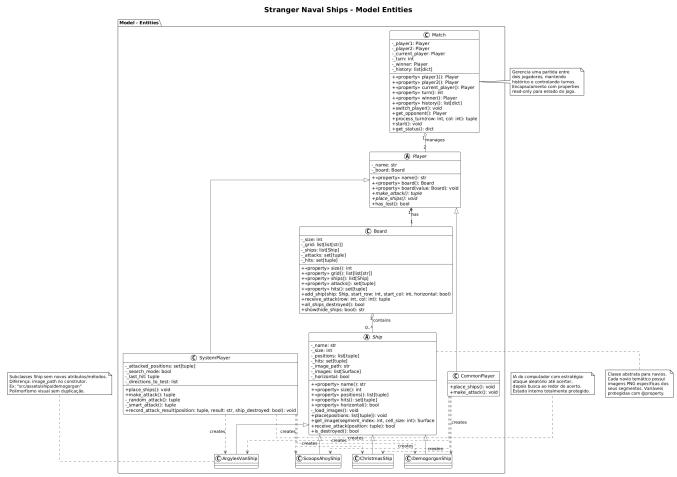


Figura 11. Entidades - Model

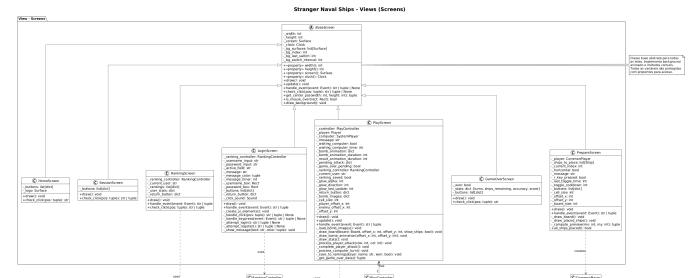


Figura 12. Camada View

C. Repositórios da camada Model e camada Controller

Por fim, a Figura 13 apresenta a diagramação dos controladores (camada Controller) e dos repositórios de persistência de dados (camada Model).

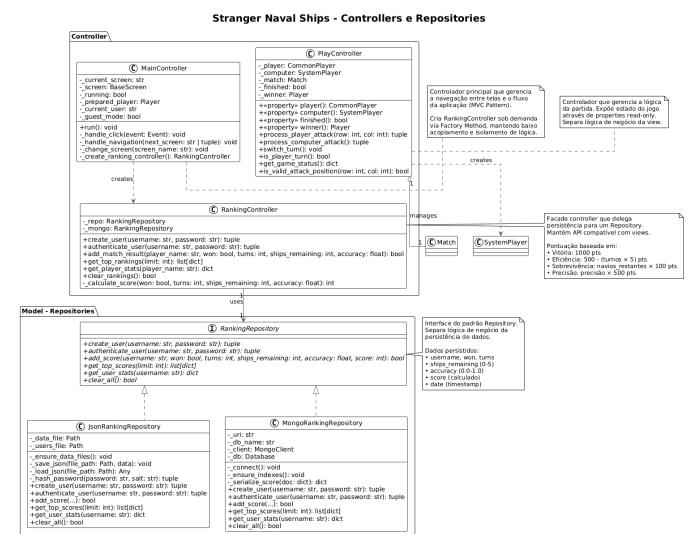


Figura 13. Repositórios - Model e camada Controller

Apêndice B

Repositório e Execução do Projeto

O código-fonte do projeto Stranger Ships está disponível em <https://github.com/gabriellivalelia/stranger-naval-ships>. As instruções de execução estão descritas no arquivo README.md do mesmo.

Referências

- [1] D. K. Ramos and B. S. Anastácio, “Habilidades cognitivas e o uso de jogos digitais na escola: a percepção das crianças,” Educação, vol. 21, no. 2, 2018.
- [2] S. M. Barroso and et al., “Treinamento cognitivo de atenção e memória de universitários com jogos eletrônicos,” Psico, vol. 50, no. 1, 2019.
- [3] S. Interactive, “Fleet battle: Batalha naval.” <https://apps.apple.com/br/app/fleet-battle-batalha-naval/id904275738>. Acesso em: 14 nov. 2025.
- [4] Byril, “Sea battle (batalha naval).” <https://www.taptap.io/br/app/9722>. Acesso em: 14 nov. 2025.
- [5] L. Cervi and M. Domingues, “Pop culture narratives as engagement mechanisms in digital games,” Game Studies, vol. 23, no. 1, 2023.
- [6] G. DeepMind, “Nanobanana - ferramenta de criação de sprites.” <https://aistudio.google.com/prompts/nanobanana>, 2024. Acessado em: 24 nov. 2025.
- [7] Freepik, “Etapas de explosão de bomba - elemento gráfico.” https://br.freepik.com/vetores-premium/etapas-de-explosao-de-boom-de-estrutura-de-dinamite-de-bomba-isoladas-definir-ilustracao-de-elemento-de-design-grafico_32376064.htm, 2024. Acessado em: 24 nov. 2025.
- [8] Pixabay, “Pixabay - biblioteca de efeitos sonoros.” <https://pixabay.com/pt/sound-effects/search/put%20item/>, 2024. Acessado em: 24 nov. 2025.
- [9] K. Dixon and M. Stein, “Stranger things - main theme (abertura oficial).” <https://www.youtube.com/watch?v=-RcPZdihrp4>, 2016. Acessado em: 24 nov. 2025.