

Relatório de Desenvolvimento - Etapa 1

Disciplina: INF01120 - Técnicas De Construção De Programas

Professora: Karina Kohl

○ **Turma A (25/2)** | **Grupo 08**

○ **Integrantes:**

- **Andrey Nunes | 00313765**
- **Érick Santos Matos | 00597911**
- **Fernando Tedesco | 00591001**
- **Pedro Blos Longhi | 00580421**

○ **Facilitador:** Érick Santos Matos

○ **Título:** A Generic Card Game

Principais Mudanças: No geral não houveram mudanças da etapa 0 para a 1, mantendo proposta e escopo como definidos anteriormente.

Requisitos de Sistema:

Requisitos Funcionais	Requisitos Não Funcionais
<p>RF-1: Menu Gráfico Contendo Botão “Início” e “Configurações”;</p> <p>RF-2: Integração do Mouse para seleção de objetos e botões;</p> <p>RF-3: Botão “Encerrar Partida”</p> <p>RF-4: Exibição de Conteúdo de Cartas Após Ação de Clique na mesma;</p> <p>RF-5: Botões de “Voltar” e “Selecionar” Após ação de Exibição de Carta;</p> <p>RF-6: Botões de ação baseados em cada carta;</p> <p>RF-7: Contador de dano e vida baseado em ataque inimigo, defesa do <i>player</i>/monstro e efeitos em campo</p> <p>RF-8: Contadores de tempo para fim da partida e fim de turno;</p> <p>RF-9: Bloqueio de ações durante turno oponente, com exceção do botão de encerrar;</p> <p>RF-10: Indicador de efeito de magia, caso houver.</p> <p>RF-11: Botão “Pausar”, “Continuar” e “Reiniciar”</p>	<p>RNF-1: O sistema deve manter coerência no estado do jogo, garantindo que nenhuma carta ou jogador execute ações fora de seu turno.</p> <p>RNF-2: Sistema deve apresentar resposta de menos de 1 segundo para ações comuns: compra de carta, invocação ou ataque, garantindo fluidez;</p> <p>RNF-3: Interface responsiva que permita interação no jogo com o mouse, sem atrasos perceptíveis;</p> <p>RNF-4: O código deve ser modularizado de forma que novas cartas ou efeitos possam ser adicionados sem alterar classes existentes.</p> <p>RNF-5: As mensagens de log devem registrar ações importantes (jogada, compra de carta, fim de turno, erro) para fins de debug e análise.</p>

Os Requisitos Funcionais propostos baseiam-se naquilo que o grupo entende como necessário para poder utilizar o programa, sendo escolhido:

RF01 pela necessidade de iniciar partidas e configurar elementos básicos como dimensões da tela.

RF02 torna-se necessário para eliminar obrigatoriedade de uso de teclado durante as partidas.

RF03 torna-se necessário para encerrar partidas, retornando ao menu e podendo iniciar nova partida.

RF04 torna-se necessário para que o jogador saiba o que cada carta faz, bem como realizar a seleção de botões das RF05 e RF06.

RF05 torna-se necessária para realizar seleção da carta, ou fechar a exibição da mesma.

RF06 torna-se necessário para realizar as ações possíveis dentro de cada carta e suas limitações.

RF07 torna-se necessário para definir um vencedor, bem como encerrar automaticamente a partida quando a vida for igual ou inferior a zero

RF08 torna-se necessário para encerrar a partida após duração padrão ser excedida, bem como ‘passar a vez’ caso o jogador demore muito durante uma ação.

RF09 torna-se necessário para evitar que o jogador realize qualquer tipo de jogada durante o turno do oponente, evitando desta forma trapaças.

RF10 torna-se necessário para que jogador e máquina saibam quais magias e/ou efeitos de campo, que possam interferir nas jogadas, estão ativos.

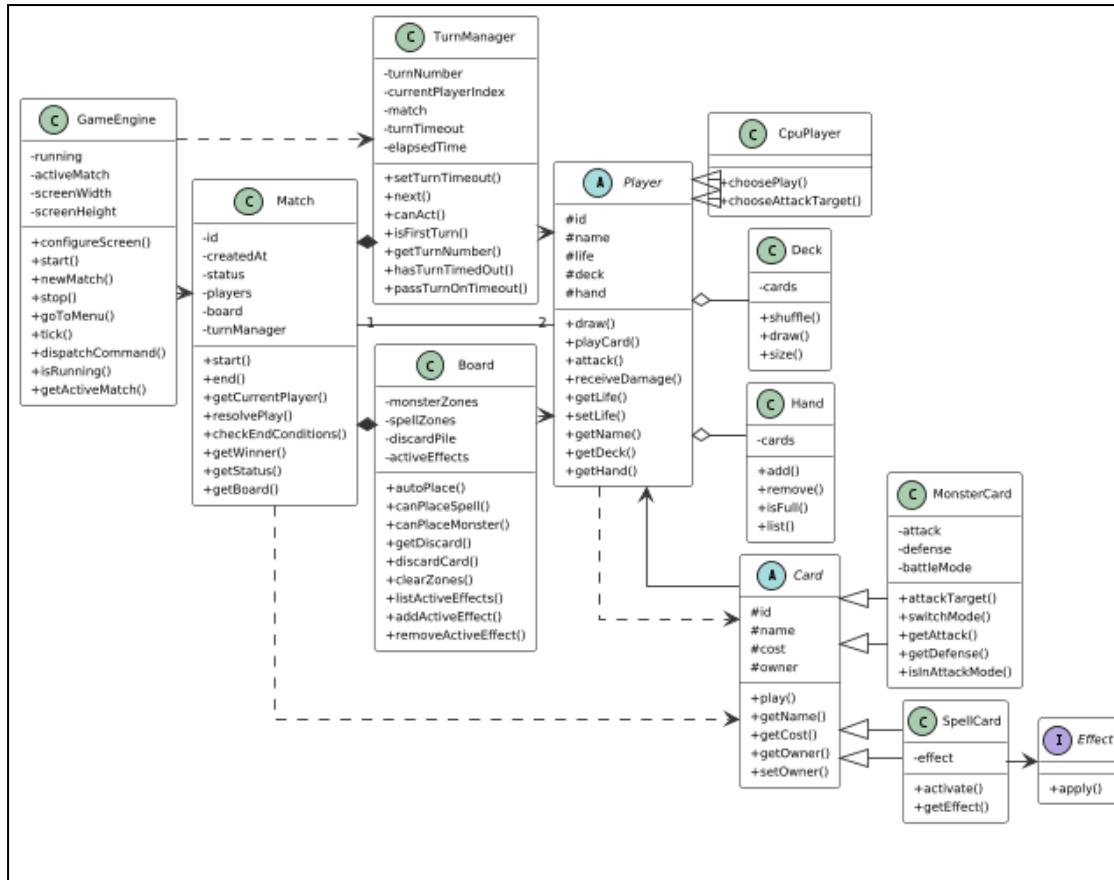
RF11 torna-se necessário para que seja possível interromper os contadores, em conjunto da partida quando o usuário quiser, bem como poder retomar ou reiniciar a partida.

A ordem escolhida baseia-se na ideia de efeito dominó, onde cada item posterior só faz sentido ser implementado se o anterior também foi implementado.

Os Requisitos Não Funcionais propostos baseiam-se em uma ordem de requisitos que prioriza a **coerência e estabilidade do jogo** (RNF-1), pois a integridade das regras é fundamental para o funcionamento correto. Em seguida, vêm aspectos de **performance e fluidez da experiência do jogador** (RNF-2 e RNF-3), garantindo respostas rápidas e interface responsiva. Por fim, foram priorizados requisitos voltados à **qualidade interna do código e facilidade de manutenção** (RNF-4 e RNF-5), essenciais para permitir evolução e depuração eficiente do sistema durante o desenvolvimento.

Projeto:

A imagem abaixo representam a proposta de desenvolvimento implementada em um Diagrama de classe onde as relações postas indicam os seguintes aspectos:



Associação(Nome/Direção/Cardinalidade)

Match – is_disputed_by -> Player [1->2 no diagrama]

(A partida tem 2 jogadores)

Generalização / Especialização (herança)

Player <|-- CpuPlayer

Card <|-- MonsterCard

Card <|-- SpellCard

Associação

Match --> Player

Card --> Player

SpellCard --> Effect

Agregação (diamante vazio)

Player o-- Deck

Player o-- Hand

Composição (diamante cheio)

Match *-- Board

Match *-- TurnManager

Dependência (tracejada)

GameEngine ..> Match

GameEngine ..> TurnManager

TurnManager ..> Player

Match ..> Card

Player ..> Card

Board ..> Player

O diagrama de classes divide o programa em componentes funcionais. A GameEngine é a classe que controla a execução do aplicativo e a criação das partidas. Cada partida é representada por um objeto da classe Match, que centraliza todos os elementos do jogo.

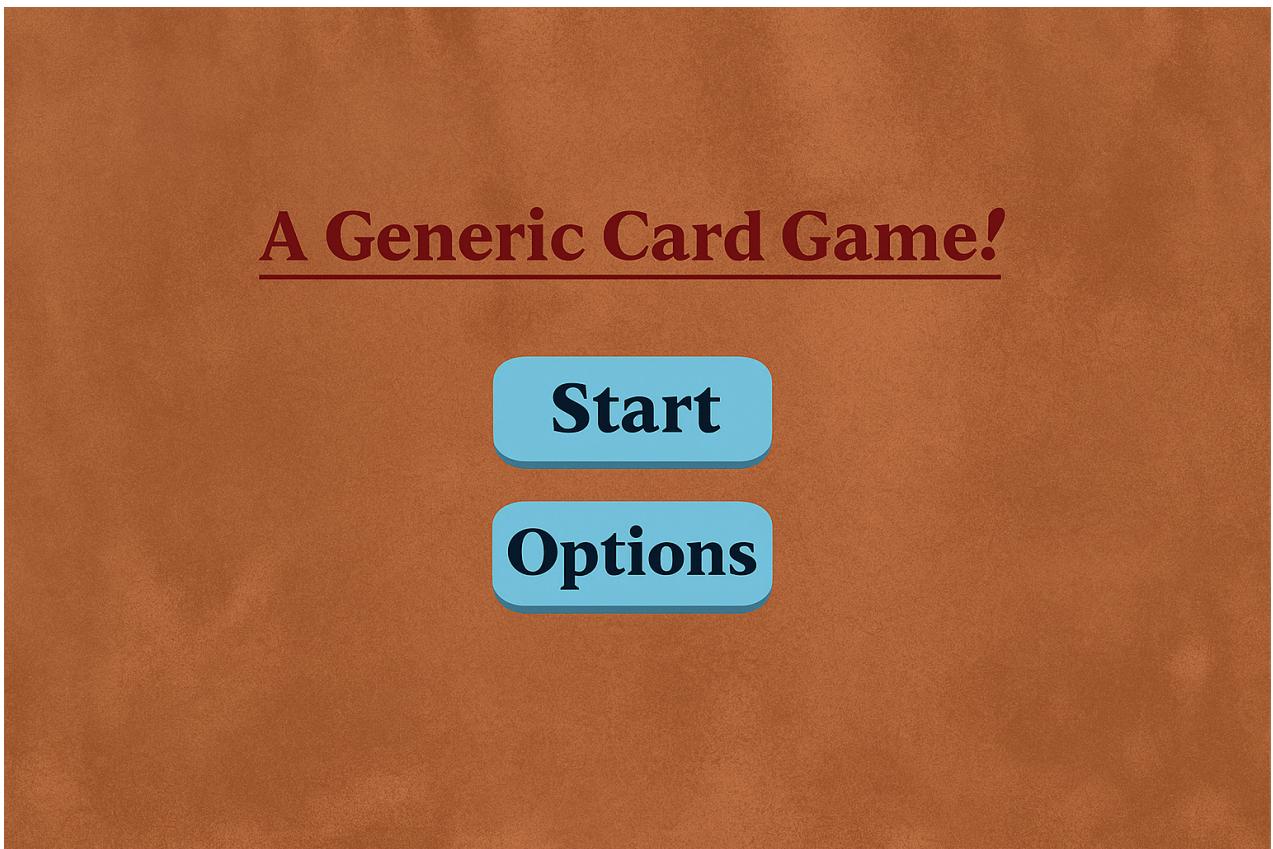
Dentro de uma Match, o Board e o TurnManager estão conectados por composição. Isso significa que eles são partes internas da partida e seus ciclos de vida estão atrelados ao dela. A responsabilidade de controlar a sequência de jogadas foi isolada na TurnManager para que a classe Match lidasse apenas com o fluxo geral do jogo, enquanto a TurnManager foca exclusivamente em quem joga e quando.

As classes Player e Card são abstratas para servirem como um modelo base para suas variações. A Match interage com objetos Player através dessa interface comum, sem precisar conhecer os detalhes de implementação de um CpuPlayer, por exemplo. Da mesma forma, Deck e Hand manipulam listas de Card de forma genérica, sem diferenciar os tipos específicos de cartas que contêm. A estrutura, portanto, separa as responsabilidades do sistema em classes com propósitos definidos.

Interface:

A imagem abaixo representa a proposta de interface de usuário para a implementação do CardGame, apresentando as áreas de **Menu**, **Jogo**, **Tela de Vitória/Derrota**, **Configurações**, **Menu de Pausa** e o **Layout das Cartas**.

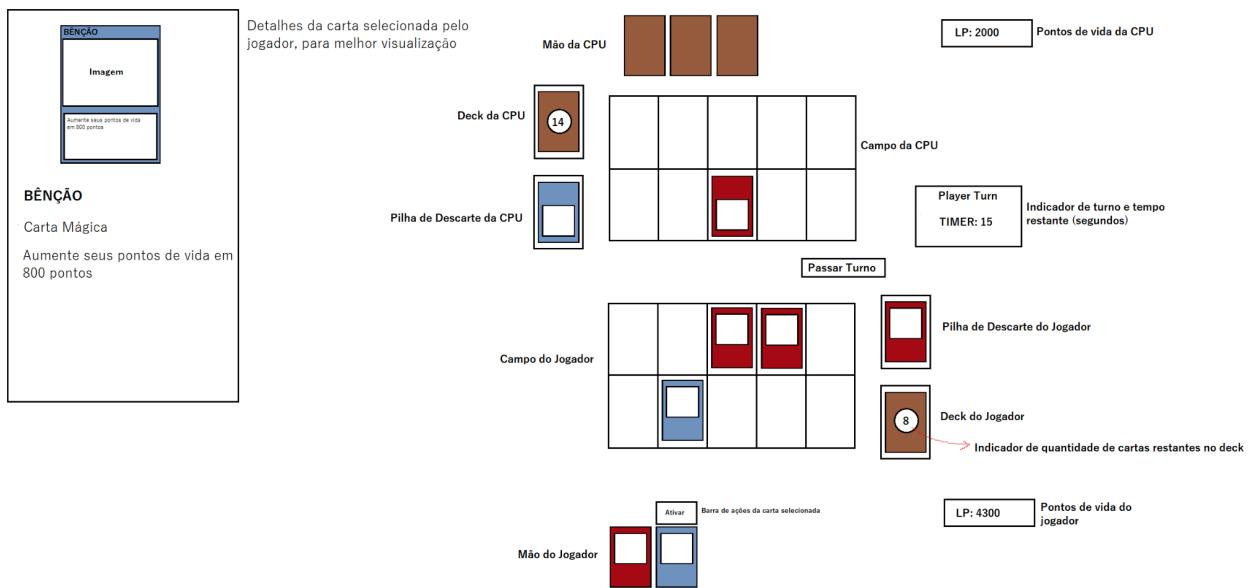
O Layout do Menu:



Botão Start inicia automaticamente uma nova partida aleatória

Botão Option exibe as opções do jogo

Jogo:



A mão do adversário deve ser mostrada para o jogador apenas como verso.

Enquanto há cartas restantes no deck do respectivo jogador, deve mostrar o ícone de verso de carta, com um contador indicando quantas cartas restam. Quando o deck está vazio, deve ser

mostrado a zona vazia.

A pilha de descarte deve exibir a última carta enviada para a zona. Se nenhuma carta tiver sido enviada, exibir a zona vazia.

Clicar numa carta que está sendo exibida de frente deve exibir detalhes sobre essa carta no canto da tela, assim como um menu de ações que podem ser tomadas com essa carta.



Botão Restart permite reiniciar a partida para uma nova combinação de cartas aleatórias

Botão Menu permite retornar ao menu inicial

Configurações:

A Generic Card Game!

Scale

640p ▾

🔊 Sound

100% ▾

Back

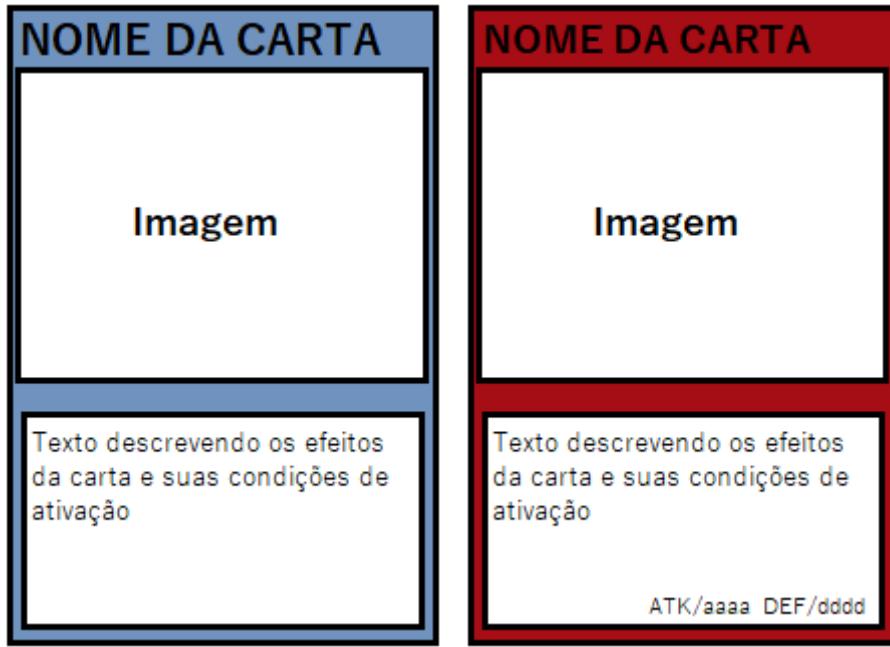
Botão Scale permite alterar o tamanho da tela dentre opções pré definidas
Botão Sound permite alterar volume da música de fundo de 0% a 100%

Menu de Pausa:



Botão Continue permite retornar para a partida do ponto em que parou
Botão Restart permite reiniciar a partida para uma nova combinação de cartas aleatórias

Layout das cartas:



Toda carta possui um nome e uma ilustração.

Cartas de monstro (direita) possuem pontos de ataque (**ATK**) e defesa (**DEF**) e podem não possuir um efeito. No caso de não possuir um efeito, simplesmente conterá o texto (**None**).

Cartas mágicas (esquerda) sempre possuem texto descrevendo seus efeitos.

Condições de ativação de efeito nem sempre existirão.

Cartas de monstro e cartas mágicas possuem bordas de cores diferentes.