Bruno Campeol Bartelle, Dionatan Pimentel Shcineider e João Vitor Maioli Marmentini – INFO III

**Relatório Técnico: Projeto Show do Milhão (Pygame)**

## 1. Descrição do Jogo e Regras

O projeto é uma implementação do jogo "Show do Milhão" utilizando a biblioteca Pygame em Python. O objetivo é que o jogador responda a uma série de 10 perguntas de múltipla escolha para ganhar um prêmio em dinheiro (virtual).

**Fluxo do Jogo:**

1. **Menu Principal:** Ao iniciar, o jogador é apresentado a um menu com as opções: "Iniciar Jogo", "Dificuldade", "Ranking" e "Sair".
2. **Dificuldade:** O jogador pode alterar a dificuldade usando as setas direcionais. Isso afeta as regras do jogo:
    - **Fácil:** 3 vidas e 35 segundos por pergunta.
    - **Médio:** 2 vidas e 25 segundos por pergunta.
    - **Difícil:** 1 vida e 18 segundos por pergunta.
3. **Início do Jogo:** Ao selecionar "Iniciar Jogo", o jogador é levado a uma tela para inserir seu nome.
4. **Quiz:** Após inserir o nome, o quiz começa.
    - Uma *cutscene* exibe o prêmio da rodada ("Valendo X reais!").
    - A pergunta é exibida com as opções. O jogador usa as setas (Cima/Baixo) para selecionar uma resposta e "ENTER" para escolher.
    - Uma tela de confirmação ("Você tem certeza?") é mostrada. O jogador deve pressionar "ENTER" novamente para confirmar ou "ESC" para voltar e escolher outra opção.
5. **Controles de Jogo:**
    - **Timer:** Um cronômetro regressivo é exibido. Se o tempo (time_left) chegar a zero, o jogador perde uma vida e a resposta é considerada errada.
    - **Vidas:** O jogador começa com um número de vidas definido pela dificuldade. Se errar uma pergunta ou o tempo acabar, perde uma vida.
    - **Cooldown:** Após cada resposta (certa ou errada), o jogo entra em um "cooldown" de 5 segundos, exibindo a tela da pergunta para que o jogador possa ver o resultado e ouvir o som de feedback antes de avançar.

**Fim de Jogo e Pontuação:**

- **Derrota (Game Over):** O jogo termina se o jogador perder todas as vidas (lives <= 0).
- **Vitória:** O jogo termina se o jogador responder corretamente a todas as perguntas do quiz (atualmente 10, com base no questions.json).
- **Pontuação (Score):** Ao final do jogo (vitória ou derrota), o método finish_game é chamado. Ele calcula uma pontuação final e uma "patente" com base na proporção de acertos:
    - score: Um valor normalizado de 0 a 10, baseado em (acertos / total_perguntas) * 10.
    - patente: Um título como "Iniciante", "Intermediário", "Avançado" ou "Expert", dependendo da pontuação.
- **Ranking:** O nome do jogador, a pontuação final e a patente são salvos no ranking.

---

**2. Diagrama de Classes**

O projeto é estruturado em torno de uma classe principal Game que gerencia o estado e agrega as demais classes responsáveis por cada tela ou lógica.

**Principais Classes:**

- **Game**: A classe central que contém o loop principal (run), a tela do Pygame (screen) e a variável de estado (self.state). Ela agrega instâncias de todas as outras telas (Menu, QuizManager, etc.) e gerencia a transição entre elas.
- **Menu**: Gerencia a lógica e o desenho do menu principal. Trata os inputs (setas, ENTER) para navegação e mudança de dificuldade.
- **QuizManager**: Controla todo o fluxo do quiz. É responsável por carregar as perguntas, exibir a pergunta atual, gerenciar a seleção e confirmação de respostas, e chamar a próxima pergunta ou finalizar o jogo.
- **Question**: Uma dataclass (ou classe similar) que serve como modelo de dados para uma única pergunta, contendo o texto, as opções e o índice da resposta correta.
- **Cutscene**: Uma tela de transição simples que exibe o prêmio da rodada por um tempo determinado (self.duration) antes de mudar o estado do jogo para "pergunta".
- **RankingManager**: Classe de lógica (não-visual) que gerencia a persistência dos dados do ranking. É responsável por ler (load) e escrever (save) no arquivo ranking.json.

- **RankingScreen**: Classe visual responsável por desenhar a tela de ranking, lendo os dados através do RankingManager.

---

### 3. Explicação do Banco de Dados e Estrutura do Ranking

O projeto não utiliza um banco de dados relacional (como SQL). Em vez disso, ele usa **arquivos JSON** para persistência de dados, o que é uma abordagem leve e adequada para este escopo.

**Estrutura de Dados**

Existem dois arquivos principais na pasta data/:

1. **questions.json**
    - **Função:** Armazena o "banco de dados" de perguntas do quiz.
    - **Estrutura:** É um *array (lista)* JSON. Cada elemento do array é um *objeto* que representa uma pergunta.
    - **Formato do Objeto:**

JSON

```
{
  "pergunta": "Texto da pergunta...",
  "opcoes": ["Opção 1", "Opção 2", "Opção 3", "Opção 4"],
  "resposta": 0
}
```

    - resposta: É o índice (base 0) da opção correta dentro do array opcoes.
    - **Gerenciamento:** O arquivo src/question.py contém funções para carregar (load_questions) e salvar (save_questions, append_question) dados neste arquivo.
2. **ranking.json**
    - **Função:** Armazena o ranking persistente dos jogadores.
    - **Estrutura:** É um *array (lista)* JSON. Cada elemento é um *objeto* que representa uma pontuação.
    - **Formato do Objeto:**

JSON

```
{
```

"name": "NomeDoJogador",

"score": 10,

"patente": "Expert"

}

**Gerenciamento do Ranking (Classe RankingManager)**

A classe RankingManager atua como uma camada de abstração de dados (Data Access Layer) para o arquivo ranking.json.

- **load()**: Chamado na inicialização. Lê o ranking.json e armazena seu conteúdo na variável self.entries. Possui tratamento de erros para caso o arquivo não exista ou esteja corrompido (JSONDecodeError).
- **add_entry(name, score, patente)**: Este é o método principal. Ele:
    1. Recebe os dados do jogador ao final de um jogo.
    2. Adiciona o novo registro (um dicionário) à lista self.entries.
    3. **Reordena** a lista self.entries inteira com base no score (do maior para o menor).
    4. Chama save() para persistir a lista atualizada.
- **save()**: Sobrescreve o arquivo ranking.json com o conteúdo atual (e ordenado) da lista self.entries, usando formatação indent=4 para legibilidade.
- **get_all()**: Método usado pela RankingScreen para obter a lista de pontuações e desenhá-la.

---

**4. Dificuldades Encontradas e Soluções Adotadas**

Durante a realização do código, foi possível inferir vários desafios comuns de desenvolvimento de jogos em Pygame. A principal delas foi a implementação deste em conjunto com o Git, principalmente na parte de sincronizar os dados de usuário no Visual Studio e ser possível o commit por parte dos integrantes. Outros pontos de dificuldade estão relacionados ao ranking e a maneira na qual ele estava alocado, além de configurar os sons da maneira correta e a interface gráfica do jogo. No início do trabalho, é importante ressaltar que as dependências e sua sincronização apresentaram problemas na interatividade e intuitividade com o usuário, resolvidos utilizando ferramentas externas.