FACULDADE DE INFORMÁTICA E ADMINISTRAÇÃO PAULISTA
FIAP

GABRIEL GENARO RM551986
PALOMA MIRELA RM551321
MURILO MARSOLA RM552117
VICTOR KENZO RM551649
RICARDO RAMOS RM550166

PRETTY VISITORS

Solução:

Doctobot

São Paulo – SP – 2022

SUMÁRIO

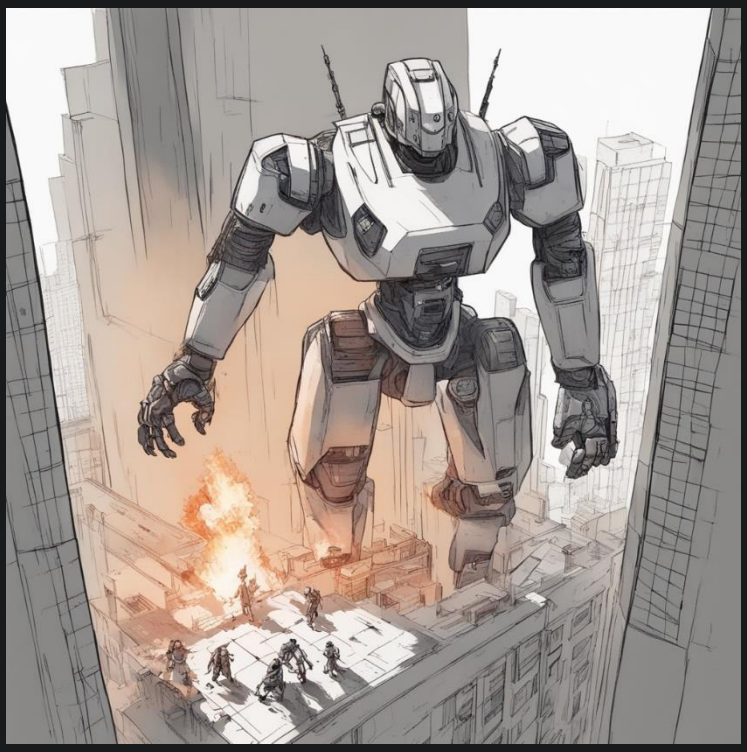
APRESENTAÇÃO DA SOLUÇÃO PROPOSTA

A nossa solução proposta se baseia em um dos exercícios que a LEPIC oferece pessoalmente aos residentes, o módulo Marbles, em que consiste em levar pequenos feijõezinhos de um ponto central para outros recipientes em volta do espaço, o exercício tem seu principal foco sendo a coordenação e precisão.

Seguindo a premissa desse exercício criamos um módulo de treinamento que visa imergir o médico residente na prática, visamos criar um mundo que chame a atenção do residente para que a prática não se torne algo banal, mas sim uma atividade que o residente sinta que está se desenvolvendo e ao mesmo tempo se divertindo.

O nosso mundo toma lugar em uma pequena cidade onde salvamentos podem ser feitos por mechas, o residente é uma dessas pessoas escolhidas para realizar salvamentos utilizando um robô mecha, seu objetivo é salvar pessoas de um incêndio no topo do prédio, o residente precisa utilizar de precisão para conseguir pegar essas pessoas do topo do prédio, tomando cuidado para não exercer pressão demais nas pessoas, já que seres humanos são frágeis, e calmamente coordenar seus movimentos para conseguir colocar essa pessoa em um dos recipientes nos prédios em volta, escolhendo o recipiente com base na cor da roupa da pessoa.

Queremos trazer esse lado da laparoscopia a um nível macro, sempre que pensamos em laparoscopia temos em mente um ambiente muito pequeno, com movimentos minúsculos realizando operações em pontos invisíveis a olho nu do corpo humano. Trazemos esse ambiente para trás aumentando tudo, então ao invés de você ser uma pessoa realizando uma operação em um ambiente minúsculo agora você controla um robô gigante que realiza suas operações no mundo real que, por conta de como você está vendo o mundo, tudo é diminuído.



Além do jogo em si a solução contará com diferentes dificuldades variando principalmente o tempo disponível para finalizar a partida, mas também contará com a adição de pequena movimentação nas pessoas em cima do prédio, cada pessoa terá uma cor de roupa para que o residente saiba exatamente em qual dos recipientes ao redor ele deverá deixar aquela pessoa.

O jogador terá acesso à sua posição do ranking, tanto pessoal, comparando suas partidas passadas entre si, quanto ao ranking total contando com todos os outros médicos residentes.

O médico coordenador terá acesso aos dados de cada um dos residentes, podendo consultar os dados de cada partida já realizada pelo jogador e estatísticas disponibilizadas em gráficos visuais para o coordenador.

Todas as etapas que não estão ligadas à realmente as partidas e a realização do exercício serão realizadas através da aplicação web, tanto o residente quanto o coordenador terão acesso à aplicação web, será lá que poderão conferir dados sobre sua performance, realizar login, consultar posições em rankings e, apenas para os coordenadores, dados sobre todos os residentes e funções administrativas.

A ideia em si do jogo é simples, e é nessa simplicidade que nossa equipe visa trabalhar tornando o jogo simples, intuitivo e extremamente eficaz quando se trata do desenvolvimento de coordenação e precisão, quanto mais tempo o residente passa jogando níveis mais difíceis, focando em melhorar sua performance, querendo subir no ranking de jogadores, ele vai se desenvolver de uma forma exponencial, nosso objetivo com esse mundo fantasioso é justamente trazer o residente fora de sua realidade para que ele possa se ligar à outro mundo enquanto desenvolve habilidades essenciais para sua formação.

PRINCIPAIS FUNCIONALIDADES & APLICABILIDADE AOS USUÁRIOS

Nossa solução proposta é uma plataforma de treinamento médico gameficada, ou seja, queremos tirar o máximo proveito possível da realidade virtual para dar uma cara mais amigável e menos repetitiva ao treinamento laparoscópico. O doctobot é uma solução que não apenas visa resolver a questão do residente poder treinar a hora que quiser, mas também quebra barreiras geográficas, facilmente um residente pode realizar todos os treinamentos laparoscópicos necessários sem precisar estar fisicamente na unidade de saúde/faculdade, o que é de enorme ajuda a todos que precisam se locomover vários quilômetros todos os dias e encarar trânsito por horas.

Abaixo estão algumas das principais funcionalidades onde apresentamos juntamente com cenários de aplicabilidade de usuários:

1. Acesso ao treinamento e gestão de desempenho (Web & VR)
    - O usuário antes de utilizar o óculos VR deve fazer login pela aplicação web para só então acessar o menu inicial do treinamento no óculos. Isso garante que apenas jogadores com uma conta ativa possam iniciar o treinamento.
    - O instrutor tem total acesso ao progresso e desempenho de cada jogador através da interface Web, garantindo que informações críticas fiquem restritas a pessoas autorizadas e seja possível consultar desempenhos e mais informações sobre partidas dos alunos à qualquer momento.
    - Cada aluno tem acesso ao seu próprio desempenho na aplicação web e à um ranking dos melhores alunos (sem mencionar nomes específicos)

Exemplo de aplicabilidade: Um médico residente realiza seu login facilmente através da interface web para iniciar a prática de seu treinamento simulado de laparoscopia. Seu desempenho é registrado e acessível tanto para ele quanto para o instrutor, que pode monitorar o progresso ao longo do tempo, esse mesmo residente checa o quadro de líderes e se sente inspirado à ficar melhor nos treinamentos para aparecer no ranking.

2. Simulação de habilidades médicas com controle de dificuldade
    - O sistema oferece 4 níveis de dificuldade que variam o tempo e complexidade dos movimentos por parte do cenário. Há também um modo livre, onde o usuário pode praticar sem restrições de tempo ou pontuação, se tornando um espaço mais tranquilo, e recomendado para melhorias individuais.
    - O jogador pode repetir os jogos quantas vezes quiser, o que é essencial para o desenvolvimento de habilidades. O jogo conta com a progressão do residente conforme ele faz mais pontos em menos tempo, liberando assim as dificuldades mais avançadas.
    - Durante o jogo, o jogador deve mover pessoas simuladas entre prédios com o uso de uma garra laparoscópica, exigindo precisão, coordenação e delicadeza levando em conta que movimentos muito bruscos ou pressionar demais as pessoas resulta em uma penalidade.

Exemplo de aplicabilidade: Em um cenário de treinamento, o residente escolhe a dificuldade intermediária para aprimorar sua coordenação motora e controle da pressão aplicada durante a simulação, sabendo que erros excessivos resultarão em feedback imediato.

3. Feedback e motivação contínua
    - Após cada sessão, o sistema oferece um resumo das pontuações do residente. Isso é complementado pelo ranking geral para encorajar mais o residente.

Exemplo de aplicabilidade: Após concluir uma sessão de treinamento, o médico visualiza seu desempenho, com foco nos erros cometidos ao manusear a garra, e se motiva a melhorar sua posição no ranking com base no feedback.

4. Facilidade e praticidade

    Tanto para o tutor quanto para o residente a facilidade proporcionada pela solução Doctobot é fenomenal. É possível realizar as sessões de treinamento onde quiser e quando quiser, basta uma conexão com internet e o equipamento de realidade virtual, e para o tutor é possível acessar o portal web da solução de qualquer dispositivo com conexão à internet, facilitando a comunicação e acompanhamento dos residentes.

Exemplo de aplicabilidade: Um residente irá passar alguns dias em outro estado, porém ele não pode simplesmente parar de treinar pois sabe que irá perder o foco e talvez um pouco de sua habilidade atual, portanto ele avisa seu tutor e leva para a viagem o equipamento de realidade virtual para realizar suas sessões de treinamento onde quiser.

O produto em sua totalidade contará com mais funções e atividades diferentes. Para nosso MVP deixamos as principais funcionalidades que são o core da solução, queremos mostrar que com pouco é possível dar ao usuário final uma experiência divertida que tenha uma melhora real em suas habilidades motoras, a fim de se tornar cada vez mais proficiente para sua profissão.

DIAGRAMAS

Diagrama do funcionamento do jogo

O diagrama baixa explica o passo a passo do jogador ao colocar os óculos VR e ao iniciar o treinamento, podendo acessar o menu, selecionar a dificuldade do jogo, realizar o treinamento, e consultar os pontos da sessão ativa.

A solução tem uma ideia geral básica pois o que importa não é necessariamente a complexidade da solução, e sim sua eficiência

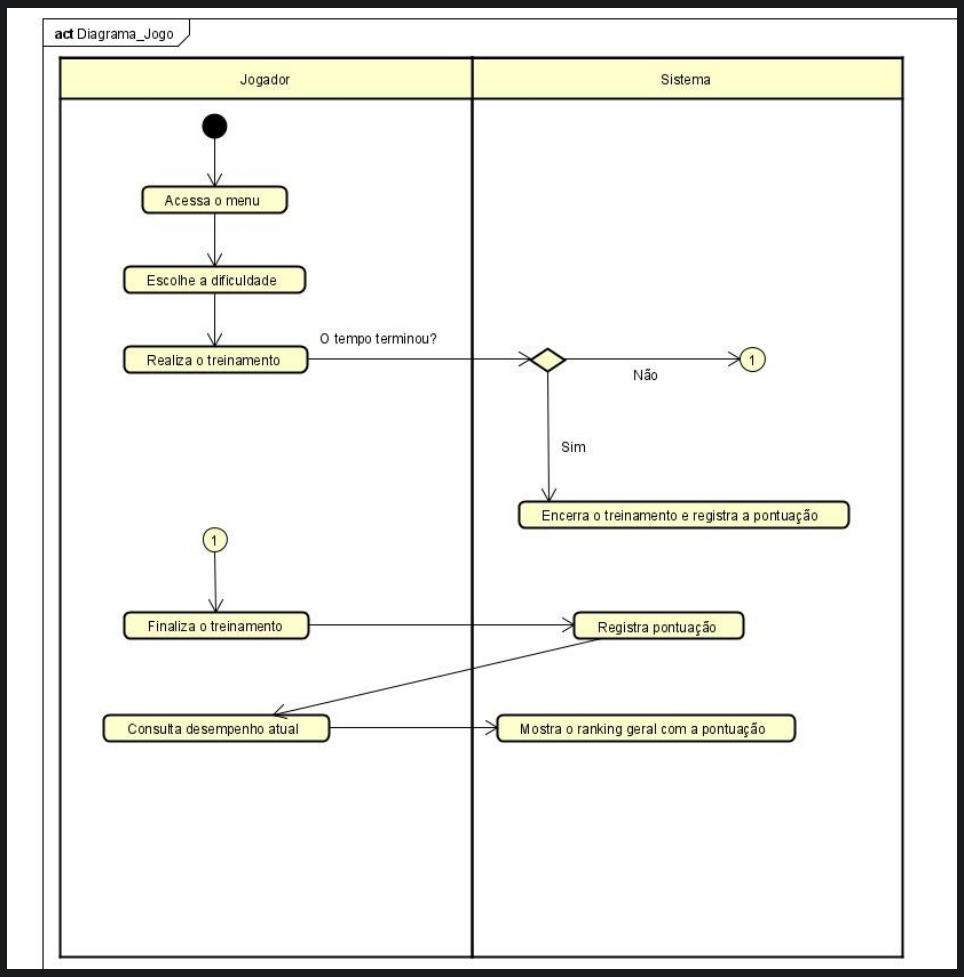
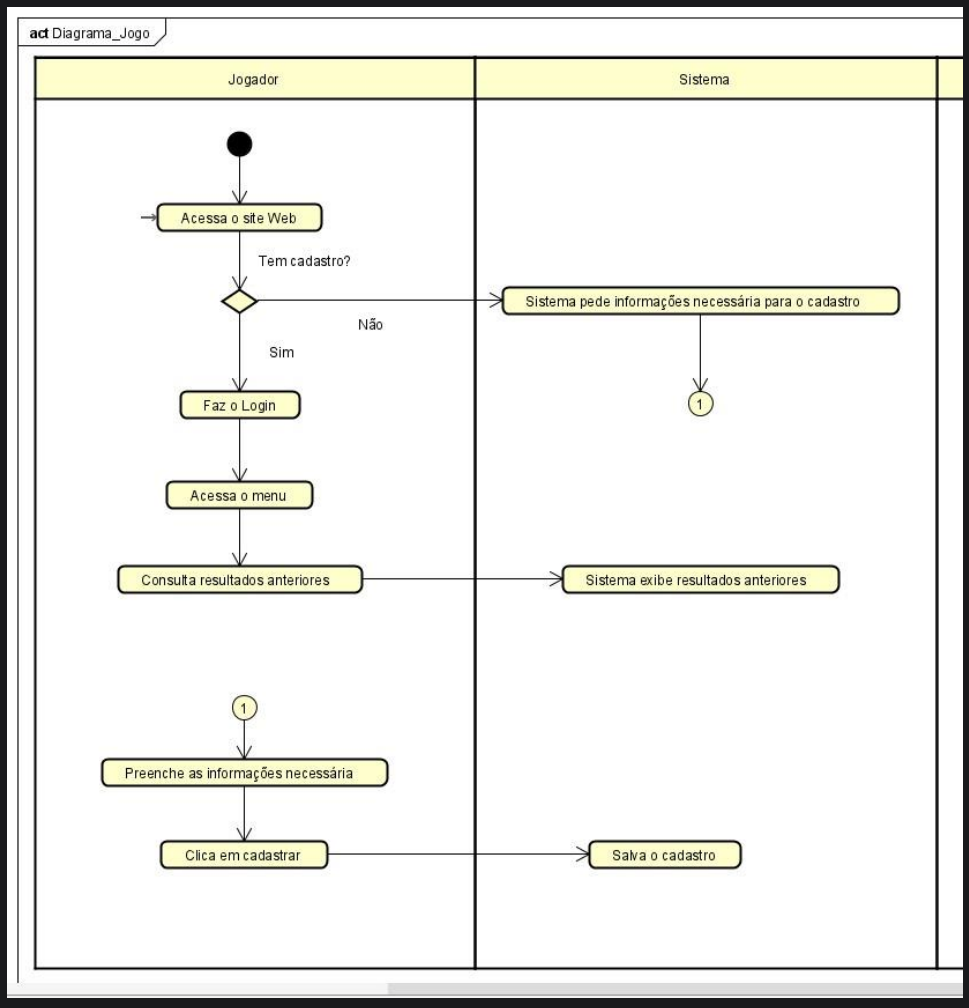


Diagrama do funcionamento do Portal Web

O diagrama abaixo explica como funcionará a página WEB do ponto de vista do jogador, podendo se cadastrar, realizar login e visualizar seus resultados.



CÓDIGOS

Código 1: Ranking Com Dicionário

```
pontuacoes = {}

def adicionar_pontuacao(nome, pontuacao):
    pontuacoes[nome] = pontuacao

def obter_pontuacao(item):
    return item[1]

def exibir_ranking():
    ranking = sorted(pontuacoes.items(), key=obter_pontuacao,
reverse=True)
    print("\nRanking:")
    for posicao, (nome, pontuacao) in enumerate(ranking, start=1):
        print(f"{posicao}. {nome} - {pontuacao} pontos")


def capturar_pontuacoes():
    while True:
        nome = input("Digite o nome do jogador (ou 'sair' para
finalizar): ")
        if nome.lower() == 'sair':
            break
        try:
            pontuacao = int(input(f"Digite a pontuação de {nome}: "))
            adicionar_pontuacao(nome, pontuacao)
        except ValueError:
            print("Por favor, insira um número válido para a pontuação.")


capturar_pontuacoes()
exibir_ranking()
```

Código 2: Ranking Com Lista Tuplas

```
pontuacoes = []

def adicionar_pontuacao(nome, pontuacao):
    pontuacoes.append((nome, pontuacao))

def exibir_ranking():
    for i in range(len(pontuacoes) - 1):
        for j in range(i + 1, len(pontuacoes)):
            if pontuacoes[i][1] < pontuacoes[j][1]:
                pontuacoes[i], pontuacoes[j] = pontuacoes[j],
pontuacoes[i]


    print("\nRanking:")
    for posicao, (nome, pontuacao) in enumerate(pontuacoes, start=1):
        print(f"{posicao}. {nome} - {pontuacao} pontos")


while True:
    nome = input("Digite o nome do jogador (ou 'sair' para finalizar):
").strip()
    if nome.lower() == 'sair':
        break
    try:
        pontuacao = int(input(f"Digite a pontuação de {nome}: "))
        adicionar_pontuacao(nome, pontuacao)
    except ValueError:
        print("Insira uma pontuacao valida!")

exibir_ranking()
```

Código 3: Ranking Com Listas Separadas

```
import os

azul = "\033[0;34m"
verde = "\033[0;32m"
vermelho = "\033[0;31m"
amarelo = "\033[0;33m"

lista_nomes=[]
lista_pontuacao = []

def organizador(pontos, nomes):

    indice_maior = 0
    indice_atual=0

    while indice_atual != len(pontos):

        indice_maior = indice_atual

        for i in range(indice_atual,len(pontos)):

            if pontos[i] > pontos[indice_maior]:
                indice_maior = i

        salva_ponto = pontos[indice_atual]
        salva_nome = nomes[indice_atual]

        pontos[indice_atual] = pontos[indice_maior]
        nomes[indice_atual] = nomes[indice_maior]

        pontos[indice_maior] = salva_ponto
        nomes[indice_maior] = salva_nome

        indice_atual+=1
```

```
while True:

    try:
        escolha = input(f"""{amarelo}
Bem-vindo ao sistema de placares:

Deseja Registrar uma pontuação ou Consultar o Ranking?

1 - {azul}Registrar
{amarelo}2 - {azul}Ranking
{amarelo}3 - {azul}Sair

""")

        os.system('cls')


        if escolha == "1":
            while True:
                nome = input(f"""{amarelo}(Caso queira voltar apenas deixe os
espaços em branco)

{azul}Informe seu nome: """)

                if nome == "":
                  os.system('cls')
                  print(f"{vermelho}Voltando ao menu...")
                  break

                else:
                    try:
                        pontuacao = int(input(f"{azul}Informe a pontuação de
'{verde}{nome}{azul}': {amarelo}"))

                        lista_nomes.append(nome)
                        lista_pontuacao.append(pontuacao)

                        os.system('cls')

                        print(f"{verde}Registro de '{nome}' feito com
sucesso!!")
                        organizador(lista_pontuacao,lista_nomes)
                        break

                    except:
                        break
```

```
        if escolha == "2":
            os.system('cls')
            print(f"{amarelo}Pressione ENTER para voltar ao menu")
            print(f"|{verde}(Pos) {azul}Nome{verde}-----------
>{amarelo}Pontuação")
            for posicao in range(0, len(lista_nomes)):
                print(f"|{verde}{posicao+1}°
{azul}{lista_nomes[posicao]}{verde} ----> {amarelo}{lista_pontuacao[posicao
]}")
            input()


        if escolha == "3":
            print(f"{vermelho}Finalizando programa...")
            break

        else:
            print(f"{azul}Escolha uma das opções demonstradas")
    except:
        os.system('cls')
        print(f"{vermelho}Escolha inválida...")
```

Código 4: Ranking Com Arquivo JSON

```
import json
import os

class ScoreManager:
    def __init__(self, filepath='pontuacoes.json'):
        self.filepath = filepath
        self.pontuacoes = {}
        self.load_scores()

    def adicionar_pontuacao(self, nome, pontuacao):
        if nome in self.pontuacoes:
            self.pontuacoes[nome] += pontuacao
        else:
            self.pontuacoes[nome] = pontuacao
        self.salvar_scores()  # Corrigido para chamar o método correto

    def obter_ranking(self):
        return sorted(self.pontuacoes.items(), key=lambda item: item[1],
reverse=True)

    def salvar_scores(self):
        try:
            with open(self.filepath, 'w') as f:
                json.dump(self.pontuacoes, f, indent=4)
        except IOError as e:
            print(f"Erro ao salvar as pontuações: {e}")

    def load_scores(self):
        if os.path.exists(self.filepath):
            try:
                with open(self.filepath, 'r') as f:
                    self.pontuacoes = json.load(f)
            except json.JSONDecodeError:
                print("Erro ao carregar as pontuações. O arquivo JSON está
corrompido.")
                self.pontuacoes = {}
            except IOError as e:
                print(f"Erro ao ler o arquivo de pontuações: {e}")
                self.pontuacoes = {}
        else:
            self.pontuacoes = {}
```

```
    def remover_jogador(self, nome):
        if nome in self.pontuacoes:
            del self.pontuacoes[nome]
            self.salvar_scores()
            print(f"Jogador '{nome}' removido com sucesso.")
        else:
            print(f"Jogador '{nome}' não encontrado.")

    def atualizar_pontuacao(self, nome, nova_pontuacao):
        if nome in self.pontuacoes:
            self.pontuacoes[nome] = nova_pontuacao
            self.salvar_scores()
            print(f"Pontuação de '{nome}' atualizada para {nova_pontuacao}.")
        else:
            print(f"Jogador '{nome}' não encontrado.")

def menu():
    manager = ScoreManager()
    while True:
        print("\n=== Sistema de Pontuação ===")
        print("1. Registrar Pontuação")
        print("2. Exibir Ranking")
        print("3. Remover Jogador")
        print("4. Atualizar Pontuação")
        print("5. Sair")
        escolha = input("Escolha uma opção: ").strip()

        if escolha == "1":
            nome = input("Digite o nome do jogador: ").strip()
            if not nome:
                print("Nome não pode ser vazio.")
                continue
            try:
                pontuacao = int(input(f"Digite a pontuação de {nome}: "))
                manager.adicionar_pontuacao(nome, pontuacao)
                print(f"Pontuação de {nome} registrada com sucesso!")
            except ValueError:
                print("Pontuação inválida. Por favor, insira um número
inteiro.")

        elif escolha == "2":
            ranking = manager.obter_ranking()
            print("\n=== Ranking ===")
            if not ranking:
                print("Nenhuma pontuação registrada ainda.")
```

```
            else:
                for pos, (nome, pontuacao) in enumerate(ranking, start=1):
                    print(f"{pos}. {nome} - {pontuacao} pontos")

        elif escolha == "3":
            nome = input("Digite o nome do jogador a remover: ").strip()
            if not nome:
                print("Nome não pode ser vazio.")
                continue
            manager.remover_jogador(nome)

        elif escolha == "4":
            nome = input("Digite o nome do jogador para atualizar: ").strip()
            if not nome:
                print("Nome não pode ser vazio.")
                continue
            try:
                nova_pontuacao = int(input(f"Digite a nova pontuação de
{nome}: "))
                manager.atualizar_pontuacao(nome, nova_pontuacao)
            except ValueError:
                print("Pontuação inválida. Por favor, insira um número
inteiro.")

        elif escolha == "5":
            print("Finalizando o programa...")
            break

        else:
            print("Opção inválida. Por favor, escolha uma das opções acima.")

if __name__ == "__main__":
    menu()
```

Código 5: Ranking Com Recursao e Arquivo JSON

```
import json
import os

class ScoreManager:
    def __init__(self, filepath='pontuacoes.json'):
        self.filepath = filepath
        self.pontuacoes = {}
        self.load_scores()

    def adicionar_pontuacao(self, nome, pontuacao):
        if nome in self.pontuacoes:
            self.pontuacoes[nome] += pontuacao
        else:
            self.pontuacoes[nome] = pontuacao
        self.save_scores()

    def obter_ranking(self):
        # Implementação do Merge Sort para ordenar as pontuações
        items = list(self.pontuacoes.items())
        sorted_items = self.merge_sort(items)
        return sorted_items

    def merge_sort(self, items):
        if len(items) <= 1:
            return items

        meio = len(items) // 2
        esquerda = self.merge_sort(items[:meio])
        direita = self.merge_sort(items[meio:])

        return self.merge(esquerda, direita)

    def merge(self, esquerda, direita):
        resultado = []
        i = j = 0

        while i < len(esquerda) and j < len(direita):
            if esquerda[i][1] > direita[j][1]:
                resultado.append(esquerda[i])
                i += 1
            else:
                resultado.append(direita[j])
                j += 1
```

```
        # Adiciona os restantes
        resultado.extend(esquerda[i:])
        resultado.extend(direita[j:])
        return resultado

    def save_scores(self):
        with open(self.filepath, 'w') as f:
            json.dump(self.pontuacoes, f, indent=4)

    def load_scores(self):
        if os.path.exists(self.filepath):
            with open(self.filepath, 'r') as f:
                self.pontuacoes = json.load(f)
        else:
            self.pontuacoes = {}

    def remover_jogador(self, nome):
        if nome in self.pontuacoes:
            del self.pontuacoes[nome]
            self.save_scores()
            print(f"Jogador '{nome}' removido com sucesso.")
        else:
            print(f"Jogador '{nome}' não encontrado.")

    def atualizar_pontuacao(self, nome, nova_pontuacao):
        if nome in self.pontuacoes:
            self.pontuacoes[nome] = nova_pontuacao
            self.save_scores()
            print(f"Pontuação de '{nome}' atualizada para {nova_pontuacao}.")
        else:
            print(f"Jogador '{nome}' não encontrado.")

def menu_principal(manager):
    while True:
        print("\n=== Sistema de Pontuação com Recursão ===")
        print("1. Registrar Pontuação")
        print("2. Exibir Ranking")
        print("3. Remover Jogador")
        print("4. Atualizar Pontuação")
        print("5. Sair")
        escolha = input("Escolha uma opção: ").strip()
```

```
        if escolha == "1":
            registrar_pontuacao(manager)
            menu_principal(manager)  # Chamada recursiva após a operação
            break

        elif escolha == "2":
            exibir_ranking(manager)
            menu_principal(manager)  # Chamada recursiva após a operação
            break

        elif escolha == "3":
            remover_jogador(manager)
            menu_principal(manager)  # Chamada recursiva após a operação
            break

        elif escolha == "4":
            atualizar_pontuacao(manager)
            menu_principal(manager)  # Chamada recursiva após a operação
            break

        elif escolha == "5":
            print("Finalizando o programa...")
            break

        else:
            print("Opção inválida. Por favor, escolha uma das opções acima.")
            menu_principal(manager)  # Chamada recursiva para opção inválida
            break

def registrar_pontuacao(manager):
    nome = input("Digite o nome do jogador: ").strip()
    if not nome:
        print("Nome não pode ser vazio.")
        return
    try:
        pontuacao = int(input(f"Digite a pontuação de {nome}: "))
        manager.adicionar_pontuacao(nome, pontuacao)
        print(f"Pontuação de {nome} registrada com sucesso!")
    except ValueError:
        print("Pontuação inválida. Por favor, insira um número inteiro.")

def exibir_ranking(manager):
    ranking = manager.obter_ranking()
    print("\n=== Ranking ===")
    for pos, (nome, pontuacao) in enumerate(ranking, start=1):
        print(f"{pos}. {nome} - {pontuacao} pontos")
```

```python
def remover_jogador(manager):
    nome = input("Digite o nome do jogador a remover: ").strip()
    if not nome:
        print("Nome não pode ser vazio.")
        return
    manager.remover_jogador(nome)

def atualizar_pontuacao(manager):
    nome = input("Digite o nome do jogador para atualizar: ").strip()
    if not nome:
        print("Nome não pode ser vazio.")
        return
    try:
        nova_pontuacao = int(input(f"Digite a nova pontuação de {nome}: "))
        manager.atualizar_pontuacao(nome, nova_pontuacao)
    except ValueError:
        print("Pontuação inválida. Por favor, insira um número inteiro.")

if __name__ == "__main__":
    score_manager = ScoreManager()
    menu_principal(score_manager)
```

ANÁLISE DOS CÓDIGOS

Recursão

Critério	Código 1	Código 2	Código 3	Código 4	Código 5
Utiliza recursão?	Não	Não	Não	Não	Sim (Merge Sort)

Análise:

- **Código 5**: Utiliza recursão no algoritmo merge sort para ordenação.
- **Códigos 1, 2, 3 e 4**: São totalmente iterativos e não fazem uso de recursão.

Medidas de Tempo em Segundos (Estimativas)

Operação	Código 1(Timsort)	Código 2 (Bubble Sort)	Código 3 (Selection Sort)	Código 4 (Timsort + I/O)	Código 5 (Merge Sort)
Adição de Pontuação	O(1)	O(1)	O(1)	O(1)	O(1)
Ordenação	O(n log n)	$O(n^2)$	$O(n^2)$	O(n log n)	O(n log n)
Exibição do Ranking	O(n)	O(n)	O(n)	O(n)	O(n)
Persistência de Dados	-	-	-	O(n) (Leitura/Gravação)	-

Análise:

- **Códigos 1 e 7**: Os mais eficientes, ambos com O(n log n) para a ordenação.
- **Códigos 2 e 3**: Menos eficientes, com $O(n^2)$ para a ordenação. Em listas grandes, terão um tempo de execução muito maior.
- **Código 4**: Desempenho comparável ao Código 1, mas com o acréscimo de leitura e gravação de dados.

Complexidade Computacional (Big O)

Operação	Código 1	Código 2	Código 3	Código 4	Código 5
Adição de Pontuação	O(1)	O(1)	O(1)	O(1)	O(1)
Ordenação	O(n log n)	$O(n^2)$	$O(n^2)$	O(n log n)	O(n log n)
Exibição	O(n)	O(n)	O(n)	O(n)	O(n)
Espaço	O(n)	O(n)	O(n)	O(n)	O(n)

Conclusão:

- **Códigos 1, 4 e 5**: Mais eficientes com complexidade O(n log n) para ordenação, oferecendo boa performance para grandes volumes de dados.
- **Códigos 2 e 3**: Com algoritmos de ordenação $O(n^2)$, o desempenho será pior para listas grandes, impactando a escalabilidade.

Facilidade de Codificar e Manter

Critério	Código 1	Código 2	Código 3	Código 4	Código 5
Estruturas de Dados	Dicionário	Lista de Tuplas	Duas Listas Separadas	Dicionário (JSON)	Lista
Modularidade	Alta	Média	Média	Alta	Alta
Leitura do Código	Simples	Moderada	Complexa	Simples	Moderada

Análise:

- **Código 1**: Fácil de codificar e manter. O uso de dicionários torna o código simples e eficiente.
- **Código 4**: Modular, fácil de manter e tem a vantagem de persistência de dados em JSON.

- **Códigos 2 e 3**: Mais difíceis de manter devido à implementação manual de algoritmos de ordenação. No código 3, a separação das listas torna o código mais confuso.
- **Código 5**: Embora use recursão, sua modularidade ajuda a manter o código organizado, embora não tão simples quanto o Código 1.

Facilidade de Modificar

Critério	Código 1	Código 2	Código 3	Código 4	Código 5
Extensibilidade	Alta	Média	Baixa	Alta	Média
Reutilização de Código	Alta	Média	Baixa	Alta	Alta
Facilidade de Adaptação	Alta	Média	Baixa	Alta	Média

Análise:

- **Código 1**: Extremamente fácil de modificar devido à estrutura simples e modular.
- **Código 4**: Flexível e modular, permitindo fácil adaptação para novas funcionalidades. A persistência em JSON oferece mais possibilidades.
- **Código 2**: Adicionar novas funcionalidades, como um algoritmo de ordenação mais eficiente, pode ser complexo.
- **Código 3**: Dificilmente modificável, principalmente devido à estrutura com duas listas separadas.
- **Código 5**: Boa modularidade, mas modificar a recursão pode ser mais complicado para quem não está familiarizado com algoritmos recursivos.

Resumo Comparativo

Critério	Código 1	Código 2	Código 3	Código 4	Código 5
Recursão	Não	Não	Não	Não	Sim
Tempo de Execução	O(n log n)	$O(n^2)$	$O(n^2)$	O(n log n)	O(n log n)
Complexidade	O(n log n)	$O(n^2)$	$O(n^2)$	O(n log n)	O(n log n)
Facilidade de Codificar	Alta	Média	Baixa	Alta	Média
Facilidade de Manter	Alta	Média	Baixa	Alta	Média
Facilidade de Modificar	Alta	Média	Baixa	Alta	Média

Conclusões e Recomendações:

1. **Código 1** é o mais eficiente e recomendado para a maioria dos casos devido ao uso de Timsort, facilidade de codificação, manutenção e modificação.
2. **Código 4** também é uma boa opção, especialmente se houver necessidade de persistência de dados. Sua modularidade e uso de JSON tornam-no flexível.
3. **Código 5** é interessante por sua aplicação de recursão, mas requer conhecimento mais avançado para modificar. Sua eficiência é comparável ao Código 1.
4. **Códigos 2 e 3** são os menos eficientes, especialmente para grandes volumes de dados, devido aos algoritmos de ordenação $O(n^2)$. Eles também são mais difíceis de manter e modificar.

Link do Github:

https://github.com/G3n4r00/DynamicProgSprint3.git