UNIVERSIDADE FEDERAL DE VIÇOSA - UFV - CAMPUS FLORESTAL

FUNDAMENTOS E TEORIA DA COMPUTAÇÃO

Trabalho Prático Final - CCF 131

Miguel Antonio Ribeiro e Silva - 4680

João Victor Graciano Belfort de Andrade - 4694

Mateus Henrique Vieira Figueiredo - 4707

Alan Gabriel MArtins Silva - 4663

Matheus Nascimento Peixoto - 4662

Florestal - MG

2023

Introdução

Esta documentação se refere ao Trabalho Prático Final da disciplina Fundamentos e Teoria da Computação (CCF 131). O projeto consistiu na implementação de um sistema de simulação de batalhas, que permitiu a aplicação prática dos conceitos de máquinas lógicas aprendidos durante o curso. O resultado final do projeto pode ser encontrado no repositório do GitHub disponível em [1].

Pré-requisitos

Para o desenvolvimento do projeto e a criação de uma interface gráfica interativa semelhante a um jogo, foram utilizadas as bibliotecas **Tkinter, Pillow** e **PyGame** em Python. Para garantir o funcionamento adequado do programa, é necessário instalar essas bibliotecas executando os seguintes comandos:

```
**pip install pillow**
**pip install tkinter**
**pip install pygame**
```

Além das bibliotecas mencionadas, é necessário ter o **Python 3** instalado em seu sistema para poder executar o programa. Certifique-se de ter uma versão recente do Python 3 instalada antes de prosseguir com a instalação das bibliotecas e a execução do projeto. O Python 3 pode ser baixado gratuitamente no site oficial do Python [2] e o processo de instalação é relativamente simples. Para executar é necessário estar no diretório src de cada arquivo.

Além disso, uma fonte externa chamada *"GodOfWar*" foi utilizada no projeto. Essa fonte pode ser baixada do site dafont [3], que está referenciado na bibliografia do trabalho. Certifique-se de baixar e instalar essa fonte para garantir que o sistema funcione corretamente.

Metodologia

O projeto foi estruturado seguindo conceitos de Programação Orientada a Objetos (POO) e tentando aplicar o padrão de arquitetura **MVC (Model-View-Controller)** e a linguagem escolhida foi **Python**.

Python foi escolhido como a linguagem para o projeto devido à familiaridade do grupo e à facilidade de implementação do padrão MVC e da interface gráfica usando a biblioteca Tkinter. A clareza da sintaxe do Python e a disponibilidade de bibliotecas e frameworks contribuíram para um desenvolvimento mais eficiente e eficaz do sistema.

A estrutura do projeto foi organizada em pastas, visando uma separação clara e modularização do código. A pasta "src" contém as subpastas referentes ao controle, à visão e ao modelo do sistema, seguindo a abordagem do padrão MVC. Essa divisão permite a fácil manutenção e extensibilidade do código.

Além dos arquivos de código-fonte, a pasta "**src**" também contém uma subpasta com as imagens utilizadas na interface gráfica, garantindo um visual mais atrativo para o projeto.

Outra pasta importante é a "**files**", que contém os arquivos de texto (.txt) das máquinas utilizadas no projeto. Esses arquivos são responsáveis por armazenar as configurações das máquinas lógicas utilizadas na simulação das batalhas. Essa abordagem permite uma fácil modificação e personalização das características das máquinas sem a necessidade de alterar diretamente o código-fonte do projeto.

Por fim, o arquivo "**main.py"** é o ponto de entrada do programa. Ele é responsável por inicializar o sistema e coordenar as interações entre as diferentes partes do projeto.

Python foi escolhido como a linguagem principal para o projeto de simulação de batalhas devido à familiaridade do grupo e à facilidade de implementação do padrão MVC e da interface gráfica usando a biblioteca Tkinter. A clareza da sintaxe do Python e a disponibilidade de bibliotecas e frameworks contribuíram para um desenvolvimento mais eficiente e eficaz do sistema.

Essa metodologia adotada no projeto permite uma estrutura organizada, modular e flexível, facilitando o desenvolvimento do sistema de simulação de batalhas.

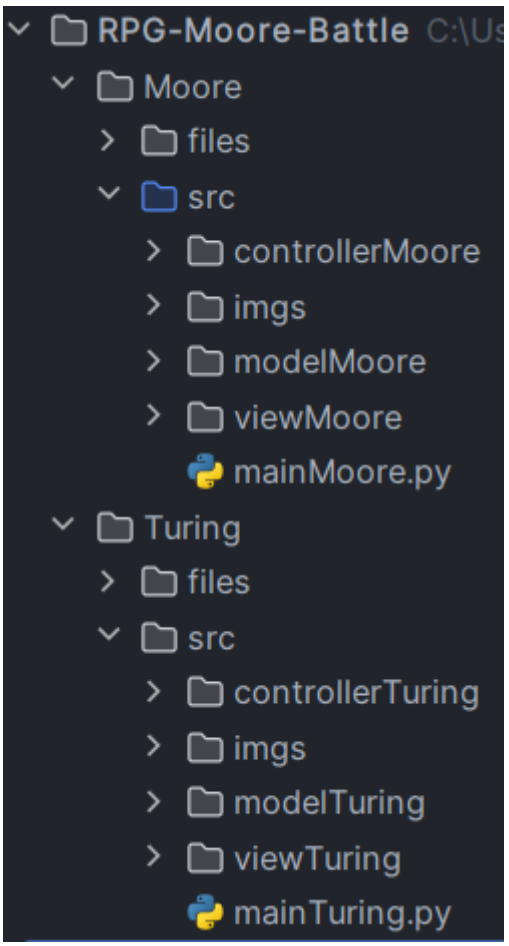

Imagem 1: Organização do projeto

Desenvolvimento

Máquina de Moore

- **Classe Duelist**

A classe **`Duelist`** representa um duelista que participa da batalha. Cada duelista possui um nome, um estado atual, uma lista de transições, uma lista de produções, uma lista de ações, um estado inicial, pontos de vida máximos e pontos de vida atual. O objetivo da classe é simular os turnos de um duelo entre dois duelistas, onde cada um executa a sua máquina lógica para decidir suas ações.

Métodos:

- **`__init__(self, name, machine_file, max_life)`**: O construtor da classe inicializa um duelista com um nome, arquivo de máquina lógica e pontos de vida máximos. Além disso, carrega a máquina lógica do arquivo fornecido.

- **`load_machine(self, machine_file)`**: Carrega a máquina lógica do arquivo fornecido. A função lê o arquivo, obtém as linhas contendo as informações necessárias e configura o estado inicial, as transições e as produções do duelista, escolhidas randomicamente.

- **`add_transition(self, current_state, next_state, input_)`**: Adiciona uma transição à lista de transições do Duelist. Se o estado atual ainda não existir na lista de transições, é criado um novo dicionário para esse estado.

- **`add_production(self, state, production)`**: Adiciona uma produção à lista de produções do Duelista.

- **`remove_actions(self)`**: Remove todas as ações registradas do Duelist.

- **`execute_machine(self, next_state, target)`**: Executa a máquina lógica do Duelist, determinando a produção com base no próximo estado. Com base na produção, realiza uma ação correspondente, como ataque, defesa ou cura. Atualiza o estado do Duelist.

- **`play_turn(self, opponent, choice)`**: Realiza um turno de jogo, onde o Duelist e o oponente escolhem suas ações com base na máquina lógica. As ações são registradas e, em seguida, resolvidas.

- **`resolve_actions(self, opponent)`**: Resolve as ações registradas pelo Duelist e pelo oponente. Com base nas ações, são determinados os danos, defesas e curas de cada Duelist, atualizando seus pontos de vida. Detalhes da implementação desse método pode ser visualizados nos comentários do código.

Atributos:

- **`name`**: O nome do duelist.
- **`state`**: O estado atual do duelist, inicializado com o estado de ínicio.
- **`transitions`**: As transições da máquina lógica do duelist.
- **`productions`**: As produções da máquina lógica do duelist.
- **`actions`**: As ações registradas pelo duelist.
- **`initial`**: O estado inicial da máquina lógica do duelist.
- **`max_life_points`**: Os pontos de vida máximos do duelist.
- **`life_points`**: Os pontos de vida atuais do duelist.

Constantes:

- **`max_attack`**: O valor máximo de ataque possível, padrão 20.
- **`max_heal`**: O valor máximo de cura possível, padrão 10.
- **`max_defense`**: O valor máximo de defesa possível, padrão 10.
  O grupo decidiu que o ataque é o estado mais vantajoso.

Detalhes de Implementação:

Na implementação do projeto, houve uma **modificação em relação à especificação** original. Enquanto a especificação original sugeria um duelo entre domínios, o grupo decidiu que seria mais interessante e esteticamente adequado ter um duelo direto entre Duelistas, eliminando a necessidade do domínio como parte do cenário do jogo. Essa mudança não impactou as demais funcionalidades do projeto.

Uma das adaptações feitas foi em relação ao estado inicial dos duelistas. Originalmente, o estado inicial deveria ter o formato '**IX**', onde 'X' representaria a inicial do domínio. No entanto, essa abordagem foi substituída pelo uso do estado '**S1**' (State 1) para ambos os duelistas. Essa modificação facilitou a geração aleatória das máquinas lógicas, conforme será descrito posteriormente.

Essa alteração foi feita com o objetivo de simplificar o jogo e tornar a interação mais intuitiva para os jogadores. A mudança não afetou a lógica geral do projeto e permitiu que os duelistas participassem do duelo de forma mais direta e imersiva.

```
class Duelist:
    Matt +1
    def __init__(self, name, machine_file, max_life):
        self.name = name
        self.state = "S1"
        self.transitions = {}
        self.productions = {}
        self.actions = {}
        self.initial = None
        self.max_life_points = max_life
        self.life_points = max_life
        self.load_machine(machine_file)
```

‘

Imagem 2: Classe Duelista

- **Enumeração Action**

A enumeração **`Action`** define os possíveis tipos de ação que um duelista pode executar durante um turno. As ações disponíveis são:

- **`ATAQUE`**: Representa a ação de ataque, onde o duelista causa dano ao oponente.
- **`DEFESA`**: Representa a ação de defesa, onde o duelista se protege e reduz o dano recebido.
- **`CURA`**: Representa a ação de cura, onde o duelista recupera pontos de vida.

A enumeração `Action` é utilizada no contexto do projeto para identificar e diferenciar as ações tomadas pelos duelistas durante o duelo. Cada ação possui um valor associado que é uma representação em string do seu nome.

A utilização dessa enumeração ajuda a melhorar a legibilidade do código, tornando mais fácil entender e identificar as diferentes ações disponíveis no sistema.

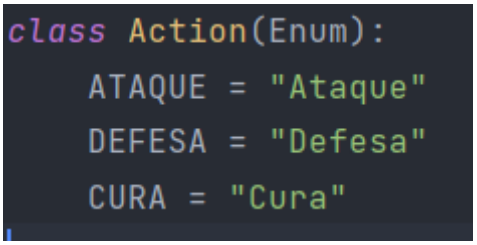

Imagem 3: Enum Action

- **generate_moore_machine.py**

Gera uma Máquina de Moore completa,e a escreve em um arquivo de saída.

Parâmetros:

num_states (int): O número de estados na Máquina de Moore.
output_file (str): O nome do arquivo de saída.

Descrição:

Esta função gera uma Máquina de Moore com o número de estados especificado e a escreve no arquivo de saída fornecido.

A Máquina de Moore é gerada da seguinte forma:

1. Geração de estados:
A função gera uma lista de nomes de estados, onde cada estado é representado por uma string no formato 'S' seguido por um número. O número total de estados é especificado pelo parâmetro num_states. Por exemplo, se num_states for 3, os estados serão ['S1', 'S2', 'S3'].

2. Estado inicial:
O estado inicial é definido como o primeiro estado da lista de estados gerada na etapa anterior ('S1').

3. Geração de transições:

A função gera as transições entre os estados. Para cada estado e para cada símbolo de entrada ('0', '1' e '2'), a função decide para qual estado o autômato deve transicionar.

- Se houver estados não alcançados ainda, ou seja, estados que não foram escolhidos como destino de nenhuma transição, a função dará prioridade para escolher esses estados.
- Se todos os estados já tiverem sido alcançados, a função escolherá um estado aleatório como destino da transição.

Cada transição é representada como uma tupla (estado_atual, símbolo_de_entrada, próximo_estado) e é armazenada na lista de transições.

4. Escrita no arquivo:

A função abre o arquivo de saída em modo de escrita e escreve a especificação da Máquina de Moore nele.

- Na primeira linha, escreve os estados, concatenando-os em uma única string separada por espaços e precedida pela etiqueta 'Q:'.
- Na segunda linha, escreve o estado inicial, precedido pela etiqueta 'I:'.
- A partir da terceira linha, escreve as transições na forma "estado_atual -> próximo_estado | símbolo_de_entrada" para cada transição, separadas por quebra de linha.

Após a execução desta função, o arquivo de saída conterá a especificação completa da Máquina de Moore gerada, sendo uma parte **extra a especificação do projeto.**

- **Classe StartWindow**

A classe `StartWindow` representa a janela de início do jogo, onde os jogadores podem inserir seus nomes e definir a quantidade de pontos de vida para o duelo.

Métodos:

- **`__init__(self, root)`:** O construtor da classe `StartWindow` recebe o parâmetro `root`, que representa a janela raiz do tkinter. Ele inicializa os atributos `life_entry`, `duelist1_name_entry`, `duelist2_name_entry` e `root`, e chama o método `create_widgets()` para criar os elementos visuais da janela.

- **`create_widgets(self)`:** O método `create_widgets()` é responsável por criar os elementos visuais da janela de início. Ele cria um rótulo de título, rótulos e campos de entrada para os nomes dos duelistas e a quantidade de vida, e um botão para começar o duelo. Os elementos são organizados utilizando o método `pack()`.

- **`check_life(self)`:** O método `check_life()` é chamado quando o botão "Começar duelo" é pressionado. Ele verifica se o valor inserido no campo de vida é um número inteiro maior que zero. Se a entrada for válida, o método `start_duel()` é chamado. Caso contrário, uma janela pop-up de aviso é exibida chamando o método `popup_message()`.

- **`popup_message(self, msg)`:** O método `popup_message()` é responsável por exibir uma janela pop-up com uma mensagem de aviso. Ele recebe a mensagem `msg` como parâmetro e cria uma nova janela pop-up utilizando a classe `tk.Tk()`. A mensagem é exibida em um rótulo e um botão "OK" é criado para fechar a janela pop-up.

- **`start_duel(self)`:** O método `start_duel()` é chamado quando todas as entradas são válidas e o duelo está prestes a começar. Ele obtém os nomes dos duelistas e a quantidade de vida inserida pelo usuário. Em seguida, a janela raiz (`self.root`) é destruída e uma nova janela de duelo é criada. As máquinas de estados dos duelistas são geradas utilizando a função `generate_moore_machine()`. Os objetos `Duelist` são criados com base nas informações fornecidas, e a classe `GameWindow` é instanciada para iniciar a janela principal do jogo.

Atributos:
- **`life_entry`:** O campo de entrada para a quantidade de vida do duelo.
- **`duelist1_name_entry`:** O campo de entrada para o nome do Duelista 1.
- **`duelist2_name_entry`**: O campo de entrada para o nome do Duelista 2.
- **`root`:** A janela raiz do tkinter.

Constantes:
-**`duelist1_states_quantity`** e**`duelist2_states_quantity`**: Essas constantes definem a quantidade de estados disponíveis para os duelistas. Ela também possui o valor 8, como padrão, indicando que os duelistas possuem 8 estados em sua máquina lógica.

-**`duelist1_machine_file`** e **`duelist2_machine_file`**: Essas constantes armazenam os caminhos dos arquivos que contém a máquina lógica dos duelistas. O valor é '../files/file2.txt', indicando que o arquivo está localizado no diretório "files" e possui o nome "file2.txt".

Uma janela de início será exibida, permitindo que os jogadores insiram seus nomes e a quantidade de vida para o duelo. Ao pressionar o botão "Começar duelo", a janela de início será fechada e a janela principal do jogo será aberta.

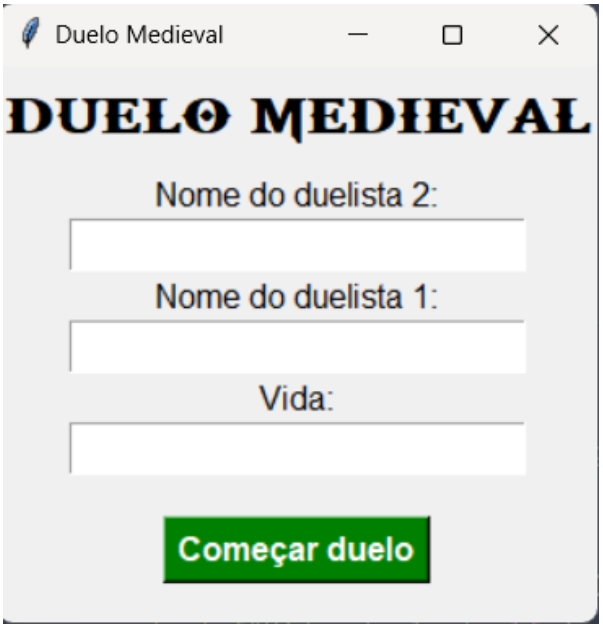

Imagem 4: Start Window

- **Classe Game Window**

A classe GameWindow representa a janela principal do jogo, onde ocorre o duelo entre dois jogadores (duelistas). Essa classe é responsável por exibir as informações dos jogadores, controlar o turno do jogo e atualizar as estatísticas.

Métodos:

- `**__init__(self, root, duelist1, duelist2)**`: Este método é executado quando um objeto da classe `GameWindow` é criado. Ele inicia a janela principal do jogo e configura os atributos iniciais. Os parâmetros `root`, `duelist1` e `duelist2` são passados para o método para definir a janela raiz, o Duelista 1 e o Duelista 2, respectivamente. Ele também configura outras variáveis e chama o método `create_widgets` para criar os elementos gráficos da janela.

- `**show_machines(self)**`**:** Este método abre uma duas janelas contendo a máquina de Moore de cada duelista. Essa função é chamada quando o jogador clica no botão "Show Machines" na janela principal do jogo. A nova janela exibe as informações sobre as máquinas de cada duelista, permitindo ao jogador visualizar os estados transições e produções.

- `**create_widgets(self)**`: Este método é responsável por criar e configurar todos os widgets (elementos gráficos) da janela do jogo. Ele cria rótulos, botões e campos de entrada, e define suas posições e comportamentos. Além disso, associa funções de callback aos botões para que ações sejam executadas quando eles forem clicados.

- **`update_stats(self)`:** Este método atualiza as informações exibidas na janela do jogo, como os nomes, pontos de vida, estado atual e turno dos duelistas. Ele é chamado sempre que há uma atualização nas estatísticas dos jogadores, como ao final de um turno ou ao iniciar o jogo.

- **`check_choice(self)`:** Este método é chamado quando o jogador pressiona o botão "Play" na janela do jogo. Ele verifica a escolha do jogador atual, obtida a partir do campo de entrada, e chama o método `play_turn` para executar o turno. Antes de chamar o método `play_turn`, ele também verifica se a escolha é válida, ou seja, se está dentro das opções disponíveis para o jogador.

- **`play_turn(self)`**: Este método executa o turno do jogador atual. Ele atualiza as estatísticas dos duelistas, como pontos de vida e estado atual, com base na escolha feita pelo jogador. Além disso, verifica se houve uma condição de vitória, derrota ou empate e atualiza a janela do jogo com o resultado do turno. Após a execução do turno, ele também alterna o jogador atual para que o próximo turno seja do outro jogador.

Atributos:

- **`root`:** Representa a janela principal do jogo criada usando a biblioteca Tkinter. É a janela onde todos os elementos gráficos do jogo são exibidos.
- **`duelist1`:** Uma instância da classe `Duelist` que representa o Duelista 1, um dos jogadores do jogo. É utilizado para acessar as informações e ações relacionadas ao Duelista 1.
- **`duelist2`:** Uma instância da classe `Duelist` que representa o Duelista 2, o outro jogador do jogo. É utilizado para acessar as informações e ações relacionadas ao Duelista 2.
- **`duelist1_state_label`:** Um rótulo de texto na janela do jogo que exibe o estado atual do Duelista 1. O estado pode ser "Normal", "Ataque", "Defesa" ou "Cura".
- **`duelist2_life_label`:** Um rótulo de texto na janela do jogo que exibe a quantidade atual de pontos de vida do Duelista 2.
- **`duelist2_state_label`:** Um rótulo de texto na janela do jogo que exibe o estado atual do Duelista 2. O estado pode ser "Normal", "Ataque", "Defesa" ou "Cura".
- **`play_button`:** Um botão na janela do jogo que o jogador pressiona para fazer uma jogada.
- **`duelist1_life_label`:** Um rótulo de texto na janela do jogo que exibe a quantidade atual de pontos de vida do Duelista 1.
- **`duelist1_actions_label`:** Um rótulo de texto na janela do jogo que exibe as produções alcançadas pelo Duelista 1.
- **`duelist2_actions_label`:** Um rótulo de texto na janela do jogo que exibe as produções alcançadas pelo Duelista 2.
- **`turn_label`:** Um rótulo de texto na janela do jogo que exibe o número do turno atual.
- **`choice_entry`:** Um campo de entrada na janela do jogo onde o jogador digita sua escolha de ação.
- **`result_label`**: Um rótulo de texto na janela do jogo que exibe o resultado da jogada, como "Vitória", "Derrota" ou "Empate".

- **`current_player_label`:** Um rótulo de texto na janela do jogo que exibe o nome do jogador atual.
- **`turn`:** O número do turno atual do jogo.
- **`current_player`:** Uma instância da classe `Duelist` que representa o jogador atual.
- **`duelist1_img_nrm`:** A imagem do Duelista 1 na forma normal.
- **`duelist2_img_nrm`:** A imagem do Duelista 2 na forma normal.
- **`duelist1_img_atk`:** A imagem do Duelista 1 na forma de ataque.
- **`duelist2_img_atk`:** A imagem do Duelista 2 na forma de ataque.
- **`duelist1_img_def`**: A imagem do Duelista 1 na forma de defesa.
- **`duelist2_img_def`:** A imagem do Duelista 2 na forma de defesa.
- **`duelist1_img_cura`**: A imagem do Duelista 1 na forma de cura.

A "Game Window" é a janela principal do jogo, proporcionando aos jogadores uma visão abrangente e interativa do duelo em andamento. Nesta janela, os duelistas são apresentados através de imagens representando suas formas de ataque, de defesa e quando estão derrotados. A vida de cada duelista é exibida em rótulos, permitindo que os jogadores acompanhem o estado atual do duelo e avaliem a situação estratégica. Além disso, a janela exibe rótulos com informações importantes, como os nomes dos duelistas, a quantidade de turnos já realizados e o turno atual. Os jogadores também têm acesso aos rótulos que mostram o estado atual de cada duelista, como estão postados, em modo de ataque, defesa ou cura. Por fim, os jogadores podem verificar as produções realizadas por cada duelista, que são exibidas nos respectivos rótulos. Além disso, a "Game Window" também apresenta um botão que permite aos jogadores visualizarem as máquinas de cada duelista. Ao clicar nesse botão, uma nova janela é aberta, exibindo a máquina de cada duelista. A "Game Window" oferece um espaço central onde os jogadores podem tomar decisões estratégicas, acompanhar as mudanças de estado dos duelistas e vivenciar a evolução do duelo em tempo real. **Observação: o nome Duelo Medieval foi substituído posteriormente a uma referência ao tipo da máquina utilizada**. **A interface também sofreu pequenas mudanças, mas nada que impacta na elaboração, apenas questões de estética que serão mostradas na apresentação.**

Imagem 5: Game Window

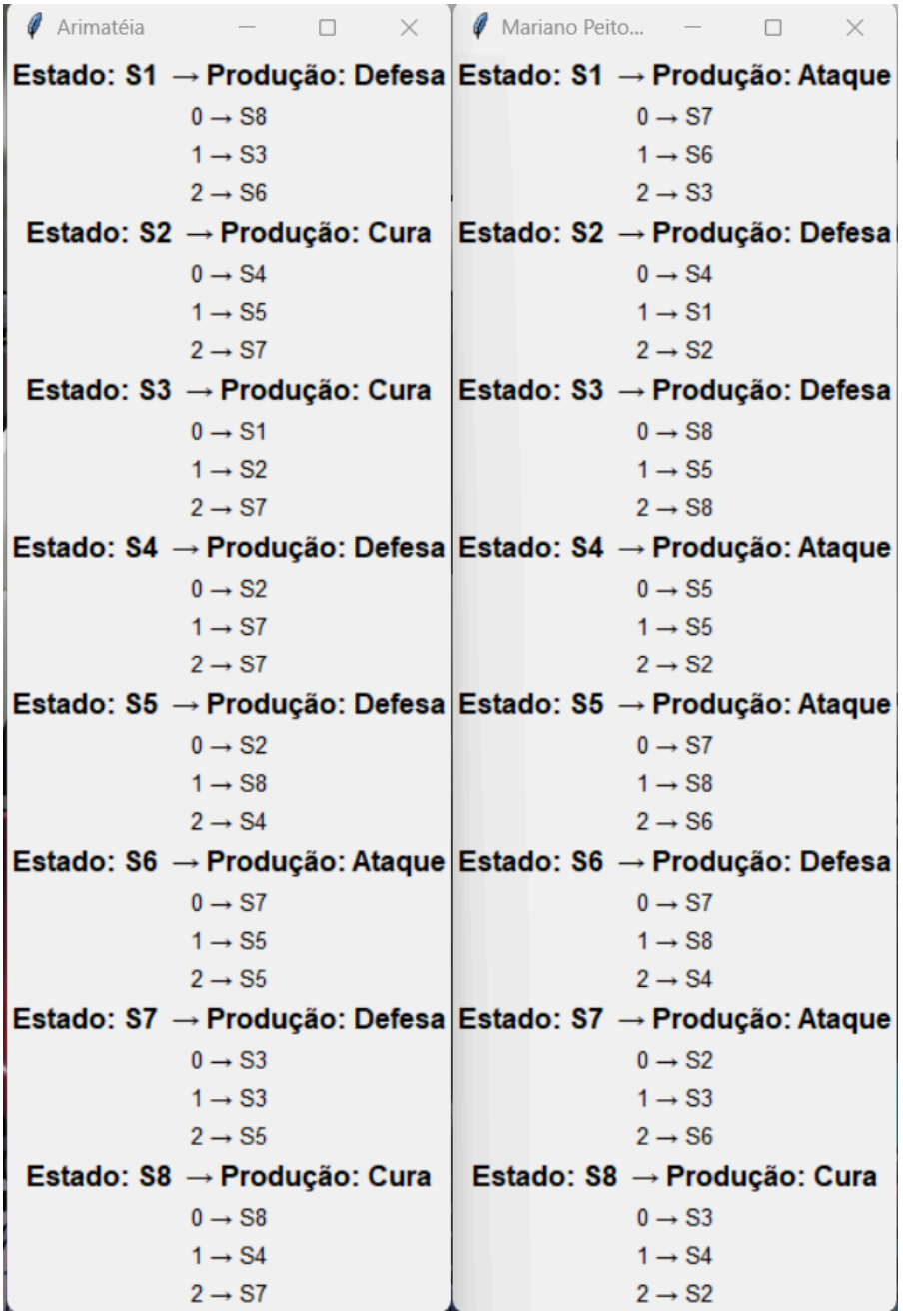

Imagem 6: Máquinas dos duelistas

- **subpasta `imgs`**

Dentro desta subpasta, estão armazenadas as imagens dos duelistas em seus respectivos estados: Ataque, Defesa, Cura e Morto. São ao todo 8 imagens, sendo 4 para cada duelista. As imagens são representadas em forma de pixel art do personagem Link, presente em [4]. Disponíveis em [5] e modificadas pelo grupo.

O Duelista 1, conhecido como Dark Link, possui uma modificação da skin padrão do Link. Sua imagem reflete sua natureza sombria e misteriosa, com tons mais escuros e uma aparência mais ameaçadora. Já o Duelista 2 utiliza a *skin* padrão do Link, mantendo a essência clássica e reconhecível do personagem.

Cada imagem retrata um dos duelistas em um estado específico. No estado de Ataque, os duelistas estão prontos para lançar seu poderoso golpe contra seus oponentes. No estado de Defesa, eles assumem uma postura mais protetora, se preparando para resistir aos ataques inimigos. No estado de Cura, os duelistas concentram suas energias para se recuperar de ferimentos, restaurando sua vitalidade. Já no estado de Morto, eles estão derrotados, sem nenhuma chance de continuar no combate.

Há também uma música de fundo, “Lost Woods” de The Legend Of Zelda Ocarina of Time.

Imagem 7: Dark Link em estado de ataque

Máquina de Turing (MT)

Como uma segunda máquina a ser implementada o grupo optou pela Máquina de Turing (MT), sendo esta do tipo transdutora. Tal escolha se deu pelo grande potencial dessa máquina e da possibilidade de manter parte da estrutura utilizada para implementar a Máquina de Moore, a qual foi explicada anteriormente nesse documento.

Como supracitado, foi possível reutilizar parte do código criado para a Máquina de Moore, fazendo algumas modificações que se mostraram necessárias para a implementação de uma MT. A seguir serão citadas as principais alterações.

- **Classe Duelist**

Métodos:

-‘__init__(self, name, machine_file, max_life)’: foi realizada a criação da fita (tape), a qual foi inicializada apenas com o símbolo de início (<) e com uma célula vazia a seguir.

- ‘load_machine(self, machine_file)’: a função de carregamento da Máquina foi alterada de modo que o arquivo pudesse ser corretamente interpretado, de modo que o código soubesse o que precisaria ser lido, escrito e qual direção seguir, utilizando-se das variáveis 'read', 'write' e 'direction'.

- ‘add_transition(self, current_state, next_state, read, write, direction)’: segue o mesmo sistema da máquina anterior, contudo passando algumas variáveis diferentes, necessárias para a MT, as quais foram citadas no método anterior.

- ‘execute_machine(self, choice, target)’: segue algo próximo à Máquina de Moore, contudo, alterando de 'produção' para a fita com suas respectivas características, realizando as ações correspondentes.

- ‘play_turn(self, opponent, choice)’ e **‘resolve_actions(self, opponent)’:** seguiram quase inalterados, realizando apenas uma pequena mudança para conseguir interpretar a ação que seria tomada.

Assim como na primeira máquina, o estado inicial 'S1' foi mantido pelo mesmo motivo, facilitar a implementação de uma geração aleatória das máquinas.

```
10  class Duelist:
11      def __init__(self, name, machine_file, max_life):
12          self.name = name
13          self.state = "S1"
14          self.transitions = {}
15          self.tape = ['<', ' ']
16          self.head = 1
17          self.actions = {}
18          self.initial = None
19          self.max_life_points = max_life
20          self.life_points = max_life
21          self.load_machine(machine_file)
```

Imagem 8: Classe Duelist para MT

Enumeração Action

Seguiu o mesmo formato anterior, fazendo apenas uma pequena alteração para um único caractere, ficando 'A' como 'ATAQUE', 'D' como 'DEFESA' e 'C' como 'CURA'

‘generate_turing_machine.py’

É nessa parte do projeto em que a Máquina de Turing é gerada e a escreve em um arquivo de saída. Os parâmetros necessários são os mesmos exigidos pela anterior, sendo o número de estados da máquina e o nome do arquivo de saída. Sua geração segue da seguinte maneira:

- De acordo com o valor passado como número de estados, estes são criados formando uma string em que cada estado começa com a letra 'S' seguida por um valor.

- Foi definido, como dito anteriormente, que o estado inicial seria sempre o 'S1'. Como a MT necessita saber para onde o cabeçote de leitura deve se movimentar a cada transição, foi definido pelo grupo que esta sempre se movimentaria para a direita ('D'), para assim conseguirmos com maior eficiência a geração aleatória.

- Assim como na Máquina de Moore criada, são realizadas transições para os símbolos '0', '1' e '2' e são realizadas as mesmas operações para definir como estas transições serão realizadas.

- No arquivo de saída sua primeira linha conta com todos os estados existentes para tal máquina. A seguir é informado o estado inicial e em sequência, já a partir da terceira linha, as transições aparecem no formato "estado_atual -> próximo_estado | símbolo_lido / símbolo_escrito direção"

A **Classe StartWindow** não precisou sofrer alterações entre uma máquina e outra, podendo manter todas as suas características para ambas.

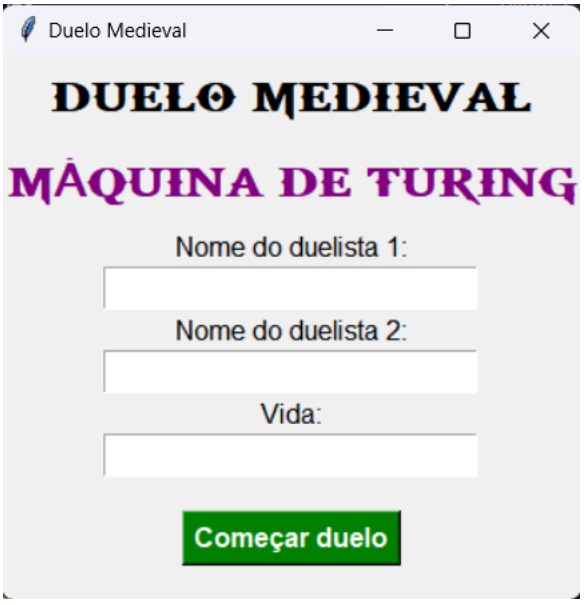

Imagem 9: StartWindow para MT

- **Classe Game Window**

No método **'show_machines(self)'**, assim como na anterior, abrem-se duas janelas para a visualização das máquinas criadas. Dessa vez essas janelas apresentam uma fita, a qual é atualizada com o decorrer do jogo e das transições realizadas, escrevendo as ações que foram tomadas.

No método **'update_stats(self)'** foram realizadas apenas pequenas alterações na qual foi passado a utilizar um dicionário que foi criado no começo do arquivo, o 'productions'. Esta modificação foi realizada para a correção no momento de apresentar as ações tomadas pelos duelistas, como 'ataque', 'defesa' e 'cura', se mostrando bastante eficaz para o caso. Os demais métodos não necessitaram passar por alterações significativas, mantendo as características descritas anteriormente.

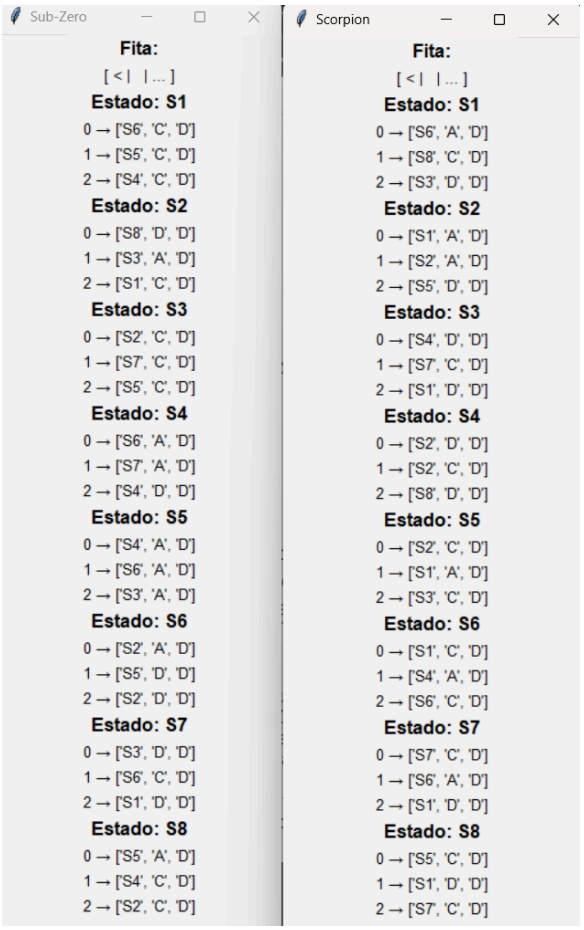

Imagem 10: Exemplo de Máquinas para MT

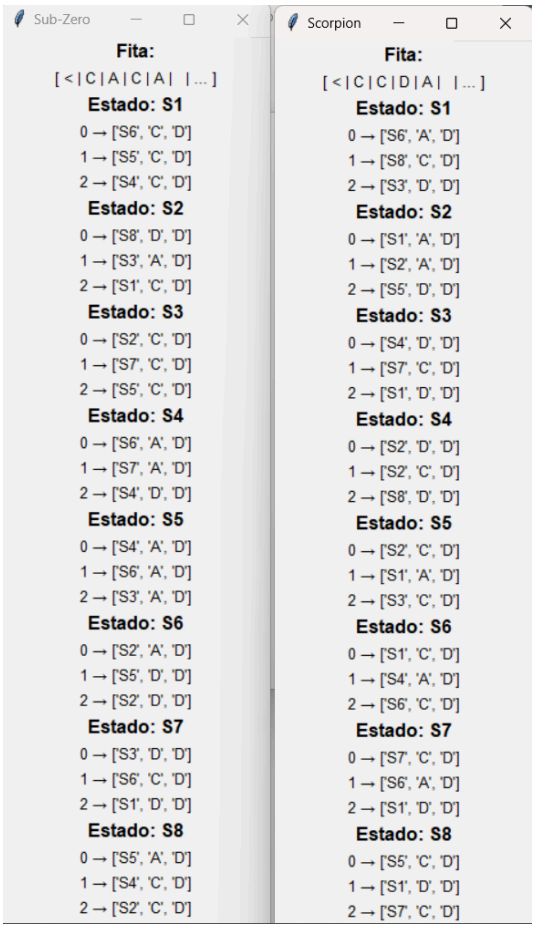

Imagem 11: Fita após a batalha

- **subpasta `imgs`**

Dentro desta subpasta, você encontrará imagens dos duelistas em diferentes estados: Ataque, Defesa, Cura e Morto, todas referentes aos personagens do jogo Mortal Kombat [6].

O Duelista 1 é o lendário Scorpion, conhecido por seu visual marcante com traje amarelo e máscara assustadora. Enquanto isso, o Duelista 2 é Sub-Zero, um mestre do gelo. Com seu traje azul e máscara intimidante.

Além das imagens, você também pode desfrutar da empolgante música tema de Mortal Kombat, que está tocando ao fundo, proporcionando uma atmosfera eletrizante e intensa.

Imagem 12: Exemplo de execução para MT

Conclusão

De forma geral, as diretrizes passadas pelo professor de CCF-424 foram seguidas para a realização do trabalho. Os resultados foram os esperados de acordo com as diretrizes, tentando conduzir uma abordagem mais prática possível. Deste modo o trabalho foi conduzido com criatividade e seriedade para poder-se ter uma boa ideia de funcionamento, aplicação e execução do tema referente.

Referências Bibliográficas

[1] https://github.com/miguelribeirokk/RPG-Moore-Battle
[2] Download Python
[3] God Of War | dafont.com
[4] Portal The Legend of Zelda | Jogos | Nintendo
[5] https://www.pngitem.com/pimgs/m/535-5351368_link-zelda-2-sprite-hd-png-download.png
[6] MKWarehouse: Mortal Kombat Sprites: Scorpion
[7] MKWarehouse: Mortal Kombat Sprites: Sub-Zero