

Pontifícia Universidade Católica do Rio Grande do Sul
Verificação e Validação de Software
Engenharia de Software

**Carolina Ferreira, Felipe Freitas, Mateus Caçabuena e Murilo
Kasperbauer**

Relatório do Trabalho 2

Porto Alegre
2024

Introdução

Neste relatório, iremos abordar a aplicação de testes em um sistema de complexidade média, um **Jogo da Velha com análise de tabuleiro por modelo de IA**. Este sistema consiste no frontend de um projeto desenvolvido em TypeScript, utilizando o framework Angular, e uma API desenvolvida em Python com o framework FastAPI para treinar os algoritmos que analisam o estado do jogo. Não há banco de dados além dos dados necessários para treinamento das IAs, que não são o objetivo dos testes.

O sistema tem como objetivo testar a eficácia de determinados modelos de aprendizado de máquina, por meio de uma interface agradável ao usuário com um jogo relativamente simples. Dentre as funcionalidades principais, destacam-se a possibilidade de jogar com um amigo ou contra o computador, integração com serviços externos e a aplicação de regras de negócio específicas do domínio do famoso jogo da velha.

Como o jogo é relativamente simples, nossa jornada de usuário se propõe a testar as funcionalidades básicas dos botões da aplicação (marcar X, marcar O, começar um novo jogo), validar casos de vitória do X, de vitória do O e de “Velha” (empate), a integração com o serviço de avaliação (se este está disponível ou não, e se seus resultados são obtidos) e se estes resultados são apresentados na tela após cada movimento do jogo. Ela pode ser descrita, mais formalmente, como: “Eu, como usuário do sistema de análise do tabuleiro de jogo da velha, quero conseguir interagir com um tabuleiro de jogo da velha e, principalmente, que o modelo analise corretamente meus movimentos e, se houver, os de meu adversário, para que eu consiga validar manualmente a eficácia do modelo que estou treinando.”.

Instruções de Compilação e Execução

Para compilar e executar o sistema, é necessário seguir os seguintes passos:

1. Requisitos:

- 1.1.1. Node ^16 (Recomendado: 20 LTS)
- 1.1.2. Python ^3.9 (Recomendado: 3.10.4)

1.2. NodeJS:

- 1.2.1. Verifique a versão instalada do Node:
 - 1.2.1.1. Em um terminal de sua preferência, digite:
 - 1.2.1.2. `node --version`
- 1.2.2. Caso necessário, instale a versão correta do Node a partir do [site oficial](#).

1.3. Python:

- 1.3.1. Verifique a versão instalada do Python:
 - 1.3.1.1. Em um terminal de sua preferência, digite:
 - 1.3.1.2. `python --version`
- 1.3.2. Caso necessário, instale a versão correta do Python a partir do [site oficial](#).

2. Configuração do Ambiente de Desenvolvimento:

- 2.1. Abra, em um terminal ou IDE de sua preferência, a pasta `TicTacToePWA`.
- 2.2. Se preferir, acesse e clone o repositório do projeto.
 - 2.2.1. Link: github.com/felipefreitassilvallearning/TicTacToePWA
- 2.3. Siga o passo-a-passo presente no arquivo `README.md`.
- 2.4. Para a API, recomendamos abrir em um terminal a pasta `API` e seguir as instruções no arquivo `README.md`.
- 2.5. Se preferir, o código original está disponível clicando no link abaixo:
 - 2.5.1. Link: github.com/EngenhariaSoftwarePUCRS/Inteligencia_Artificial/Trabalho01

3. Acessando o ambiente:

- 3.1. O frontend está disponível no endereço localhost:4200
- 3.2. A API não precisa ser acessada, mas está disponível localmente em localhost:8080

4. Como rodar os testes

- 4.1. Para rodar os testes, basta digitar em um terminal, na mesma pasta do frontend, o seguinte comando:
 - 4.1.1. `npm run tests`
 - 4.1.2. Alternativamente, rode cada teste separadamente, digite:
 - 4.1.2.1. `npm run test:unit`
 - 4.1.2.2. `npm run test:e2e`
- 4.2. Para rodar os testes de integração, digite em um terminal na pasta da API:
 - 4.2.1. `python test_integration.py`

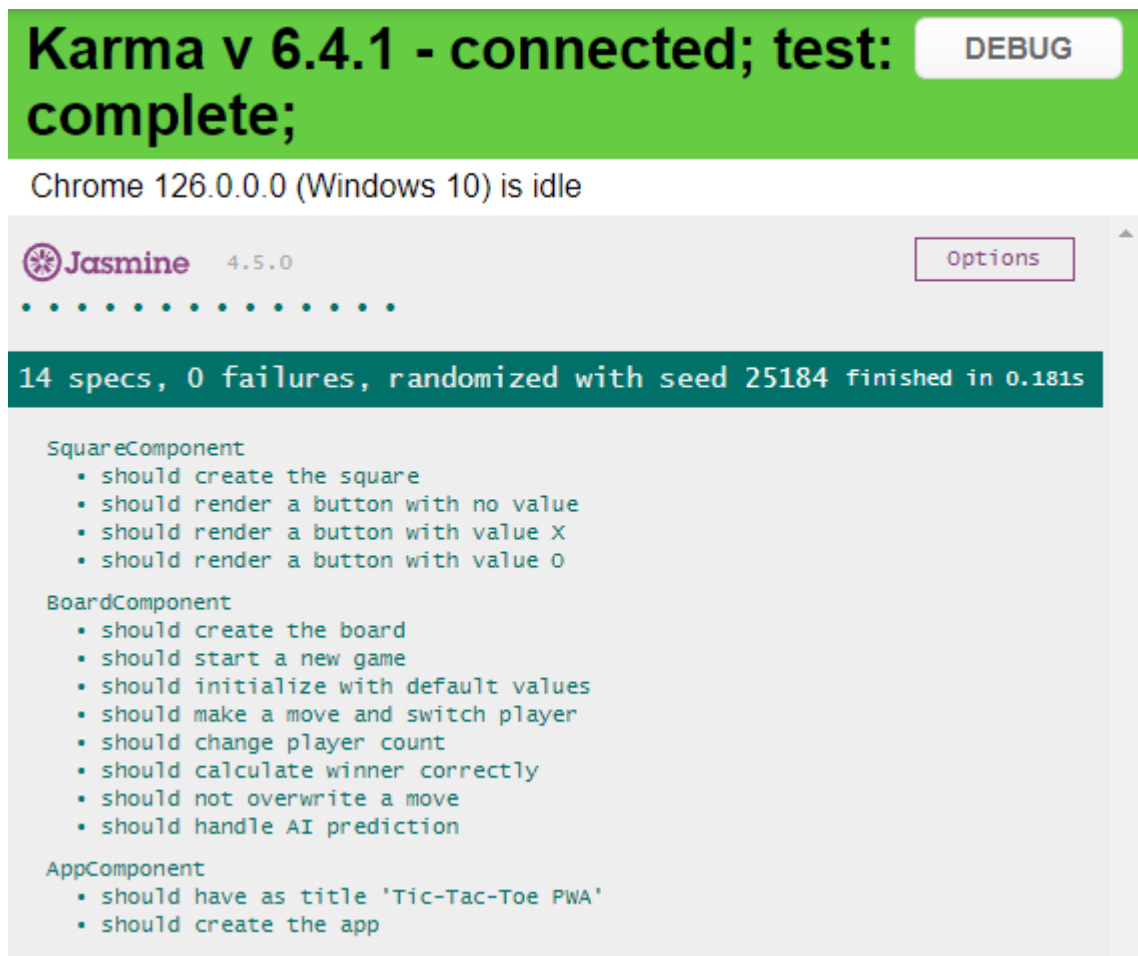
Com esses passos, o ambiente estará preparado para a execução dos testes e a análise crítica da qualidade dos mesmos, conforme os objetivos estabelecidos para este trabalho. É importante salientar que, para os testes funcionarem, é necessário que ambos os sistemas estejam rodando localmente.

Testes já disponibilizados no código

Testes padrão do Angular apenas para o caso de “Component Did Mount”, ou seja, o componente é criado. Apesar de já existirem, dos 5 testes, apenas 4 funcionam e não possuem qualidade pois não estão testando nada que agregue valor ao negócio.

Casos de Teste

Testes Unitários



1. Componente Célula (Quadrado – “Square”)

- a. Cria componente
- b. Cria botão sem valor algum
- c. Cria botão com valor X
- d. Cria botão com valor O

2. Componente Tabuleiro (“Board”)

- a. Cria componente
- b. Começa novo jogo
- c. É inicializado com valores padrão
- d. É capaz de fazer um movimento que troca o jogador ativo
- e. É capaz de mudar a quantidade de jogadores (VS: Computador ou VS: Amigo)
- f. Mostra vencedor corretamente
- g. Não pode sobre escrever uma célula
- h. Apresenta predição dos modelos de IA na tela

3. Componente Principal (“App”)

- a. Deve criar a página
- b. Deve possuir título correto da aplicação

Conclusão dos Testes Unitários

Todas as partes do sistema parecem estar funcionando conforme o esperado.

Testes de Integração

Data: 26/06/2024

Os testes de integração foram realizados para verificar o comportamento do endpoint da API do jogo TicTacToe desenvolvido. O objetivo foi validar se o servidor FastAPI retorna os resultados esperados para diferentes configurações do tabuleiro.

Resultados dos testes:

Caso de teste	Descrição	Resultado esperado	Resultado obtido	Status
1	Tabuleiro para 'X_GANHOU'	Vitória para 'X'	Vitória do jogador 'X'	Passou
2	Tabuleiro para 'TEM JOGO'	Jogo em andamento	Jogo continua em andamento	Passou
3	Tabuleiro para 'VELHA'	Empate	Empate entre os jogadores	Passou
4	Tabuleiro inválido	Erro do tamanho do	Error: Invalid board size	Passou

		tabuleiro inválido		
--	--	-----------------------	--	--

Detalhes dos casos de teste:

1. Caso de Teste 1: Tabuleiro para 'X_GANHOU'
 - Descrição: Enviar um tabuleiro que resulta na vitória do jogador 'X';
 - Resultado esperado: O servidor retorna "X GANHOU" como resultado.
 - Resultado obtido: O servidor retornou corretamente "X GANHOU";
 - Status: Passou.
2. Caso de Teste 2: Tabuleiro para 'TEM_JOGO'
 - Descrição: Enviar um tabuleiro que ainda possua um jogo em andamento dentro do sistema;
 - Resultado esperado: O servidor deve retornar "TEM JOGO";
 - Resultado obtido: O servidor retornou corretamente "TEM JOGO";
 - Status: Passou.
3. Caso de Teste 3: Tabuleiro para 'VELHA'
 - Descrição: Enviar um tabuleiro que resulta no empate de uma partida no sistema;
 - Resultado esperado: O servidor deve retornar "VELHA";
 - Resultado obtido: O servidor retornou corretamente "VELHA";
 - Status: Passou.
4. Caso de Teste 4: Tabuleiro inválido
 - Descrição: Enviar um tamanho de tabuleiro inválido (que não possua 9 células como exigido pelo sistema);
 - Resultado esperado: O servidor deve retornar um erro indicando o tamanho inválido do tabuleiro;
 - Resultado obtido: O servidor retorna corretamente o erro esperado do tamanho inválido do tabuleiro;
 - Status: Passou (O servidor deve retornar o status 400 no funcionamento do sistema).

Conclusão dos Testes de Integração:

Todos os casos de teste foram executados com sucesso e o servidor FastAPI demonstrou comportamento correto ao lidar com diferentes configurações de tabuleiro.

Testes de Sistema

```
TicTacToe
Winners
  ✓ should show that X has won (2469ms)
  ✓ should show that O has won (799ms)
  ✓ should show no winner (for tie) (1071ms)
```

(Run Finished)

Spec		Tests	Passing	Failing	Pending	Skipped
✓ gameStates.spec.cy.ts	00:06	3	3	-	-	-
✓ All specs passed!	00:06	3	3	-	-	-

1. Resulta em vitória para X
2. Resulta em vitória para O
3. Resulta em “Velha” (empate)
4. Aparecem os resultados do sistema

Conclusão dos Testes de Sistema

O sistema parece estar funcionando corretamente; as funcionalidades principais e possíveis estados de jogo estão sendo atingidos.