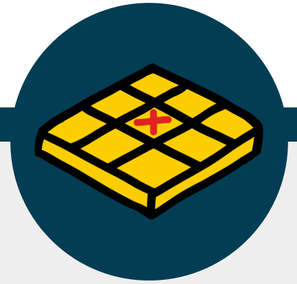
Sistema
TIC-TAC-TOE

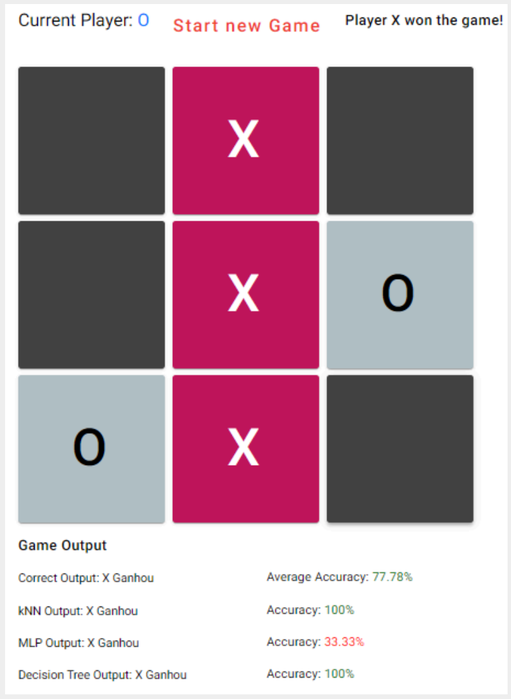
CAROLINA FERREIRA
FELIPE FREITAS
MATEUS CAÇABUENA
MURILO KASPERBAUER

VERIFICAÇÃO E VALIDAÇÃO DE SOFTWARE.
27/06/2024 PROFESSOR: JÚLIO MACHADO.

Sistema de Jogo da Velha com Análise de Tabuleiro por IA.



O frontend foi feito em TypeScript com o framework Angular, o que facilitou a integração com os testes. Para cada movimento, ele pergunta a uma API qual o estado do tabuleiro e reage de acordo.



O backend é feito em Python com uma simples API para receber as requisições do tabuleiro. Esta parte do sistema repassa o estado do tabuleiro para 3 modelos de aprendizado de máquina que, por sua vez, retornam o estado que acreditam ser correto para o momento do jogo.

Introdução

TESTES UNITÁRIOS	Criar a estrutura de validação dos testes para cada funcionalidade básica do sistema, tal qual as funcionalidades de algoritmo no backend ou funcionamento dos botões no frontend.
TESTE DE INTEGRAÇÃO	Crescer a cobertura dos testes fazendo com que as primeiras funcionalidades de backend e frontend comecem a se conversar, e da mesma forma os comportamentos de servidores do sistema.
TESTE DE SISTEMA	Validação de que o sistema funciona como esperado para ser utilizado pelo usuário ou cliente, tendo em vista que precisa ter finalizado a cobertura dos casos anteriores.

Testes Unitários

Com os testes unitários, buscamos a validação dos componentes mais específicos e cruciais do próprio sistema.

A cobertura, mesmo que simples, valida todos os componentes, desde a célula que irá estar vazia, receber um X ou O até os botões do tabuleiro para trocar de jogador, iniciar um novo jogo e o título com o nome do vencedor.

A ideia desta parte dos testes é de garantir que todas as unidades do sistema funcionem individualmente, pois, se nem isso funcionar, conseguimos observar rapidamente onde há um defeito.

```
14 specs, 0 failures, randomized with seed 15674

BoardComponent
  • should calculate winner correctly
  • should change player count
  • should make a move and switch player
  • should initialize with default values
  • should handle AI prediction
  • should not overwrite a move
  • should start a new game
  • should create the board

SquareComponent
  • should create the square
  • should render a button with no value
  • should render a button with value X
  • should render a button with value O

AppComponent
  • should create the app
  • should have as title 'Tic-Tac-Toe PWA'
```

```
it('should render a button with value O', () => {
  component.value = 'O';
  fixture.detectChanges();

  const button = fixture.debugElement.query(By.css('button'));
  expect(button).toBeTruthy();
  expect(button.nativeElement.textContent.trim()).toBe('O');
});
```

Testes de Integração

Com os testes de integração, buscamos a validação de alguns casos específicos que variam dentro do próprio sistema.

A cobertura, mesmo que simples, valida especificações de API necessárias como o tamanho inválido ou até mesmo como o servidor se comporta em caso de empate.

A ideia desta parte dos testes de fato é fazer com que o sistema local comece a ter um propósito conjunto com os servidores.

```python
def test_game_state_endpoint_for_has_game(self):
    # Teste com um tabuleiro com result TEM_JOGO
    board = "X,b,b,b,X,b,O,b,O"
    expected_output = { "correctOutput": TEM_JOGO }
    return self._test_board_successful(board, expected_output)
```

```python
def test_game_state_endpoint_for_invalid_board(self):
    # Teste com um tabuleiro inválido (não possui 9 células)
    board = "O,O,X,O,X,O,X,O"
    expected_output = {"detail": "Invalid board size"}
    response = requests.get(f"{self.base_url}/{board}")
    self.assertEqual(response.status_code, 400)
    response_data = dict(response.json())
    self.assertEqual(response_data, expected_output)
```

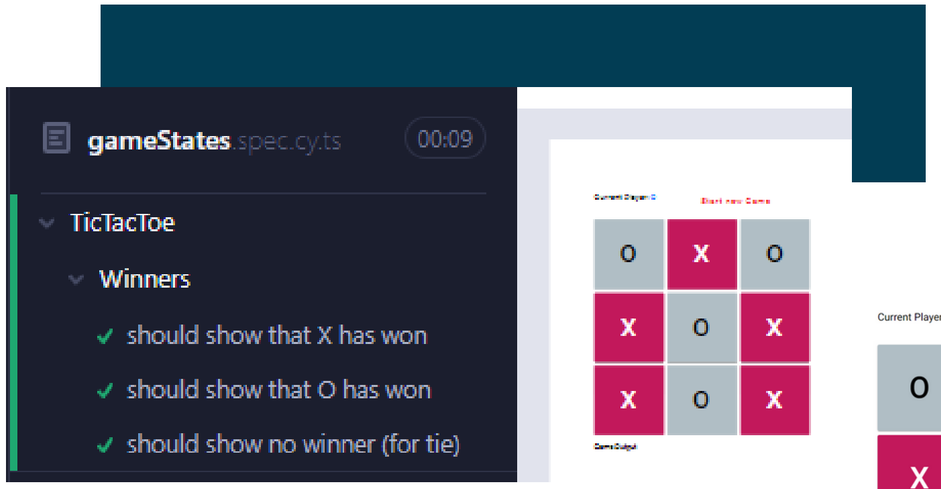
```python
def _test_board_successful(self,
                           board: str,
                           expected_output: dict[str, str],
                           expected_status_code: int = 200):
    response = requests.get(f"{self.base_url}/{board}")
    self.assertEqual(response.status_code, expected_status_code)
    response_data = dict(response.json())
    expected_key = list(expected_output.keys())[0]
    self.assertIn(expected_key, response_data.keys(), "Output key not found in response")
    self.assertEqual(
        response_data.get(expected_key),
        expected_output.get(expected_key),
        "Output value does not match expected value",
    )
```

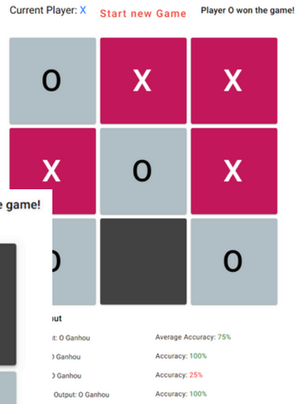
Testes de Sistema

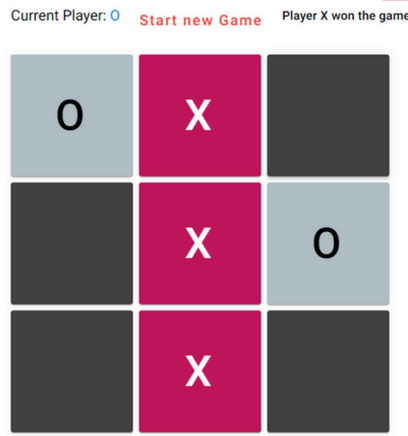
Com testes “de ponta a ponta”, estamos validando todas as funcionalidades e regras de negócio do sistema, simulando um usuário mesmo.

A cobertura aqui é completa, e deve abrangir o maior número de cenários possíveis, prováveis ou não, para termos uma ideia mais concreta das interações de um usuário.

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





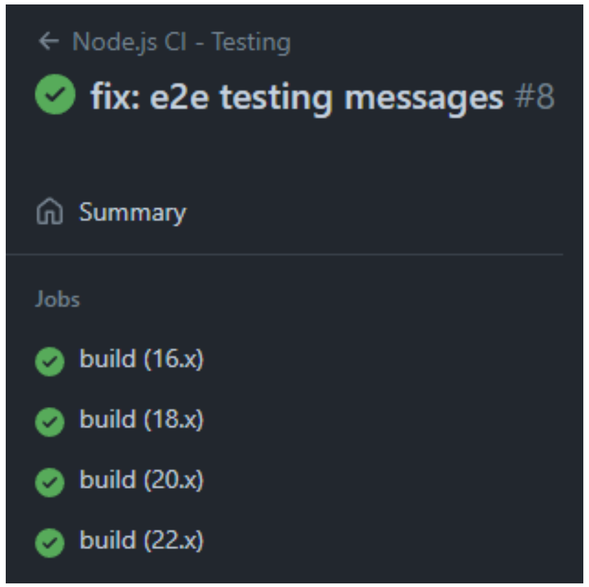


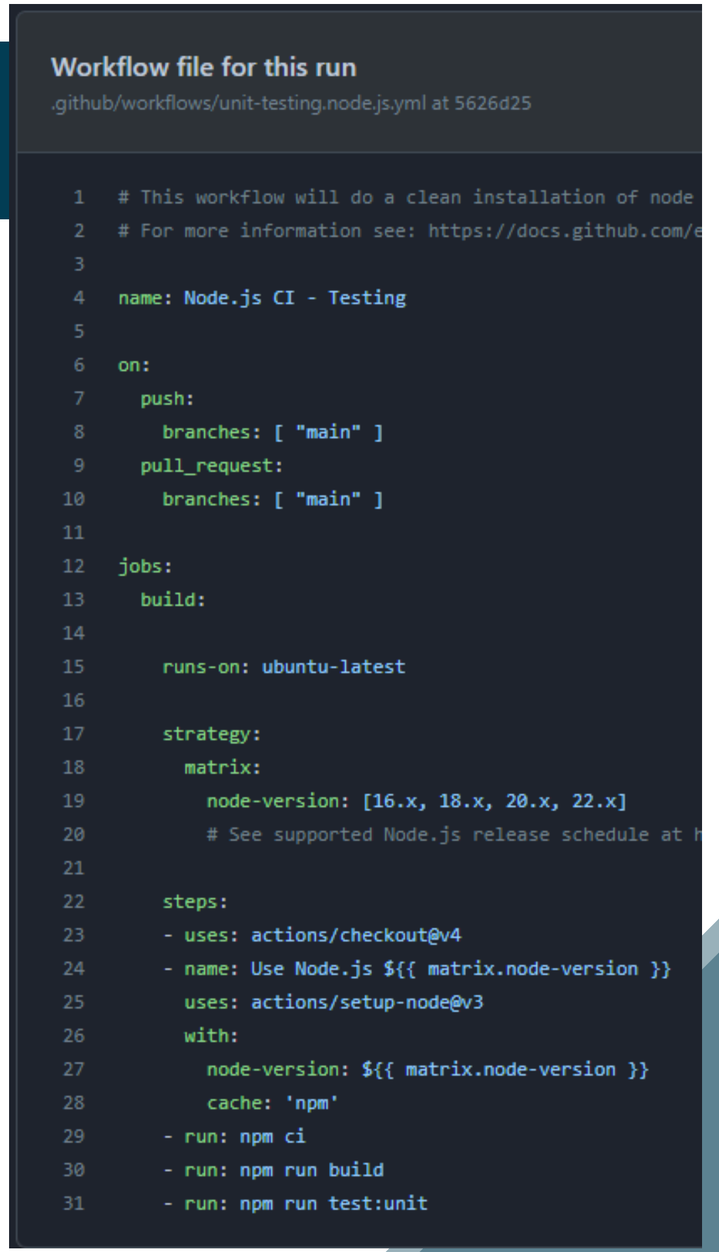

```
beforeEach(() => {
  cy.visit('http://localhost:4200');
  cy.viewport(1920, 1080);
  cy.get('[data-cy="new-game-button"]').click();
  cy.get('[data-cy="versus-button"]').click();
});

it('should show that X has won', () => {
  cy.get('[data-cy="button-1"]').click();
  cy.get('[data-cy="button-2"]').click();
  cy.get('[data-cy="button-3"]').click();
  cy.get('[data-cy="button-4"]').click();
  cy.get('[data-cy="button-5"]').click();
  cy.get('[data-cy="button-6"]').click();
  cy.get('[data-cy="button-7"]').click();
  cy.get('[data-cy="win-description"]').contains('Player X won the game!');
});
```

Extra - Integração Contínua

Este processo de “CI” é um meio moderno e automático para garantir a consistência dos testes de um determinado sistema.

Mais especificamente, para cada commit, rodamos os testes unitários em 4 versões maiores diferentes do Node, para garantir que tudo segue sempre funcionando.





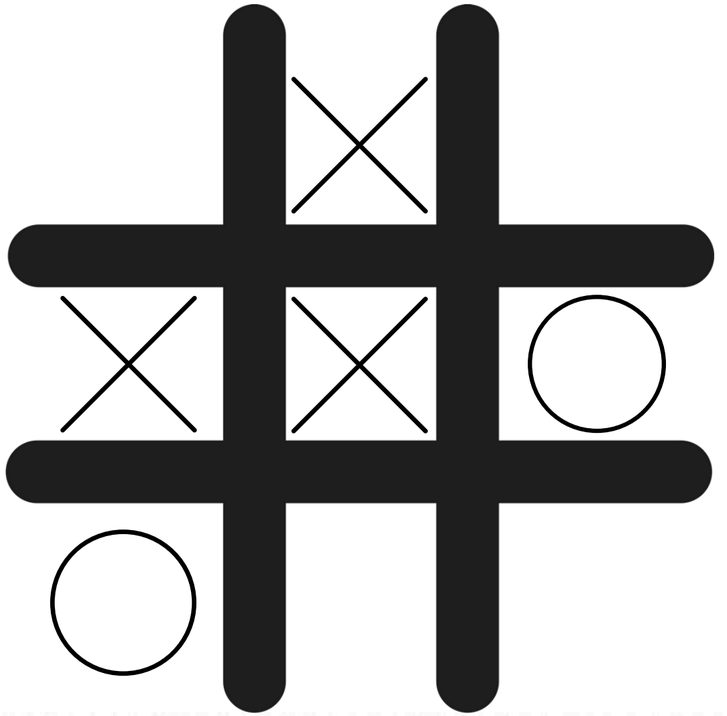
Conclusão

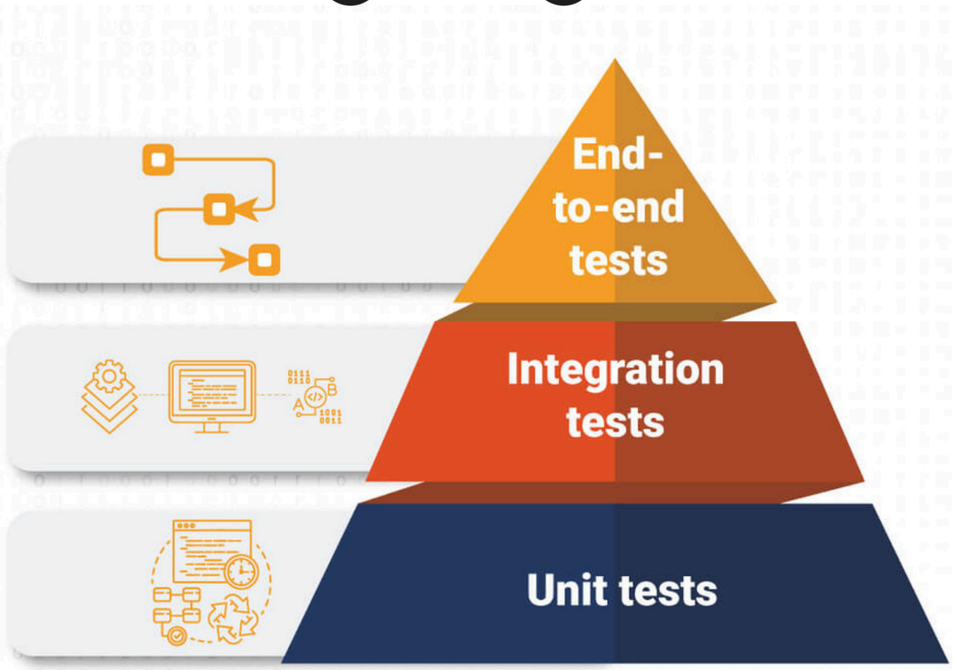
Importância

Investir em testes é investir na qualidade do software, na satisfação dos usuários e na tranquilidade de saber que, mesmo no sistema mais simples, se pode confiar que tudo rodará como esperado.

Variabilidade

Variar os testes é como testar temperos diferentes na gastronomia, torna o processo mais realista, completo e confiável, garantindo que o software estará pronto para enfrentar qualquer mudança de comportamento e que estará pronto para lidar com elas.





PUCRS

2024

OBRIGADO