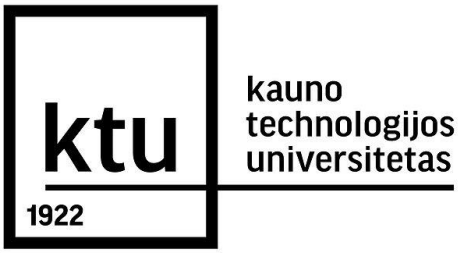


INFORMATIKOS FAKULTETAS

T120B162 Programų sistemų testavimas

2 laboratorinis darbas

Studentas: Ignas Jasonas IFF-6/6

Dėstytojas: doc. Šarūnas Packevičius

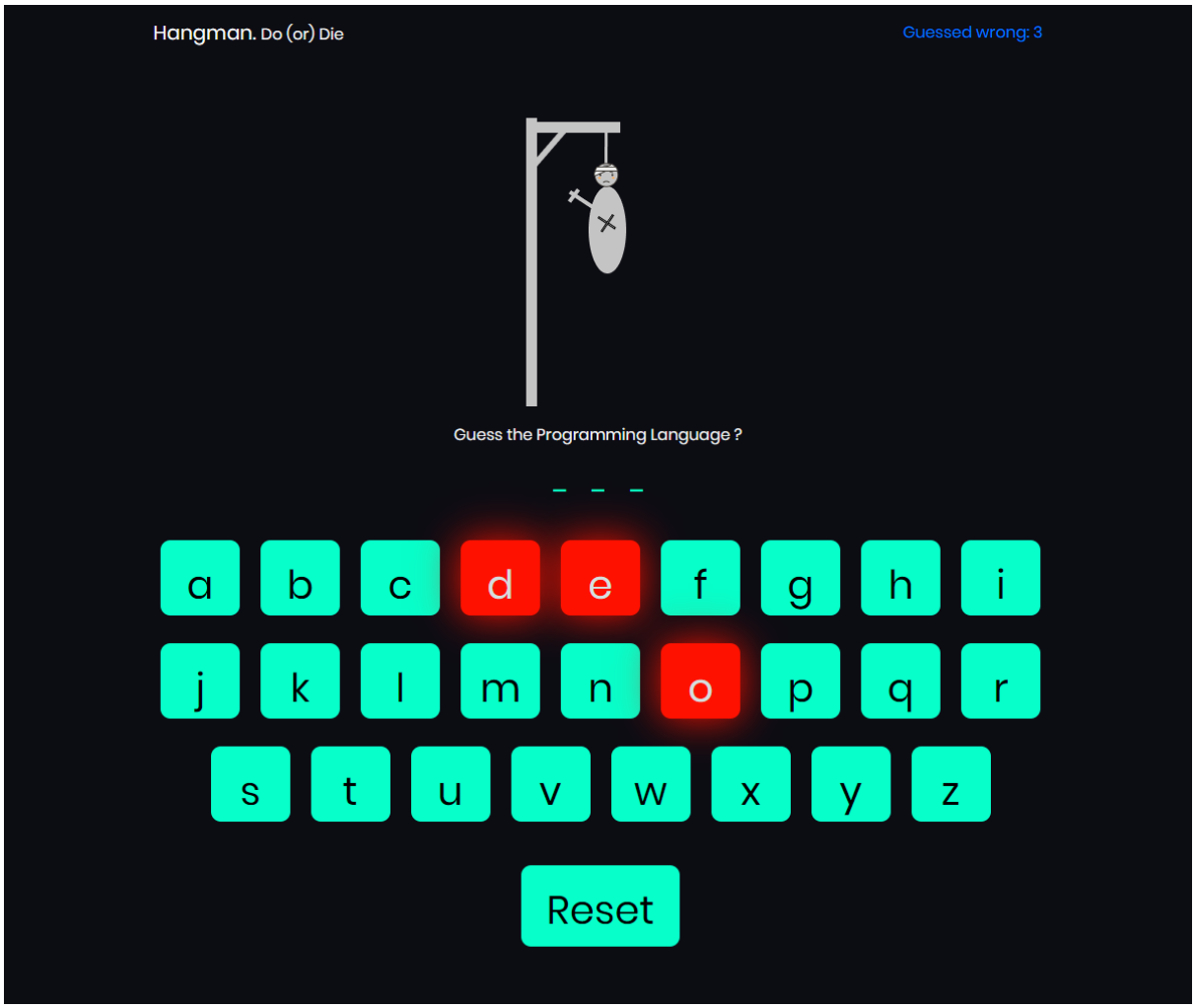
KAUNAS 2019

Turinys

1. Testuojama sistema 3
2. Unit testai 3
   2.1. resetButton 4
   2.2. guessedWord 4
   2.3. getMistake 4
   2.4. handleGuess 5
   2.5. getGameState 5
   2.6. getDisplayText 6
   2.7. randomWord 6
3. Mock'ai 7
4. Rezultatai 8
5. Išvados 8

1. Testuojama sistema

Testuojama sistema pasirinka React technologija realizuotas paprastas kartuvių žaidimas. Žaidimo tikslas – atspėti programavimo kalbą, spaudžiant ant spėjamą raidę. Projektas yra pateiktas kaip pavyzdys react github repozitorijoje. Testai realizuoti pasindauojant jest testavimo įrankiu.



2. Unit testai

Projekte buvo apie dešimt funkcijų, jas išsikėliau į atskirą failą, kad būtų patogiau testuoti ir sukūriau unit testus.

2.1. resetButton

Ši funkcija suveikia paspaudus ant reset mygtuko. Jos esmė atstatyti būsenos objektą į pradinę būseną.

```
export const resetButton = setState => () => {
  setState({
    mistake: 0,
    guessed: new Set(),
    answer: randomWord()
  });
};
```

```
test("Reset state", () => {
  resetButton(setState)();
  expect(mockState.mistake).toBe(0);
  expect(PROGRAMING_LANG).toContain(mockState.answer);
});
```

2.2. guessedWord

Ši funkcija išskaido spėjamą žodį į raidžių masyvą ir pagal spėtas raides parodo kiek žodžio yra jau atspėta.

```
export const guessedWord = state => {
  return state.answer
    .split("")
    .map(letter => (state.guessed.has(letter) ? letter : "_"));
};
```

```
test("Get current guessed word", () => {
  mockState.answer = "python";
  mockState.guessed = new Set("p");
  expect(guessedWord(mockState)).toEqual(["p", "_", "_", "_", "_", "_"]);

  mockState.answer = "python";
  mockState.guessed = new Set("r");
  expect(guessedWord(mockState)).toEqual(["_", "_", "_", "_", "_", "_"]);
});
```

2.3. getMistake

Ši funkcija pagal spėtą raidę, gauna reikiamą pridėti padarytų klaidų kiekį

```
export const getMistake = (state, letter) =>
  state.answer.includes(letter) ? 0 : 1;
```

```
test("Get mistake", () => {
  mockState.answer = "python";
  expect(getMistake(mockState, "p")).toBe(0);
});
```

2.4. handleGuess

Ši funkcija apdoroja spėjimą, atnaujina spėjamą žodį ir gautą klaidų kiekį

```
export const handleGuess = (setState, state) => evt => {
  let letter = evt.target.value;
  setState({
    guessed: state.guessed.add(letter),
    mistake: state.mistake + getMistake(state, letter)
  });
};
```

```
test("Handle guess", () => {
  let fakeEvt = { target: { value: "r" } };
  mockState.guessed = new Set();
  mockState.answer = "python";
  mockState.mistake = 0;
  handleGuess(setState, mockState)(fakeEvt)();
  expect(mockState.mistake).toEqual(1);

  fakeEvt = { target: { value: "p" } };
  mockState.guessed = new Set();
  mockState.answer = "python";
  mockState.mistake = 0;
  expect(mockState.mistake).toEqual(0);
});
```

2.5. getGameState

Ši funkcija gauna žaidimo statusą

```
export const getGameState = (isWinner, gameOver, gameStat) => {
  if (isWinner) {
    return "YOU WON";
  } if (gameOver) {
    return "YOU LOST";
  }
  return gameStat
};
```

```
test("Get game state", () => {
  expect(getGameState(true, false, "state")).toBe("YOU WON");
  expect(getGameState(false, true, "state")).toBe("YOU LOST");
  expect(getGameState(false, false, "state")).toBe("state");
});
```

2.6. getDisplayText

Ši funkcija gauna tekstą, kurį rodo ekrane

```
export const getDisplayText = (gameOver, state) =>
  !gameOver ? guessedWord(state) : state.answer;
```

```
test("Get display text", () => {
  mockState.answer = "python"
  mockState.guessed = new Set("p");

  expect(getDisplayText(true, mockState)).toBe("python");
  expect(getDisplayText(false, mockState)).toEqual(["p", "_", "_", "_", "_", "_"]);
});
```

2.7. randomWord

Ši funkcija iš programavimo kalbų masyvų išrenka atsiktinį žodį

```
export var PROGRAMING_LANG = [
    "python",
    "javascript",
    "mongodb",
    "json",
    "java",
    "html",
    "css",
    "c",
    "csharp",
    "golang",
    "kotlin",
    "php",
    "sql",
    "ruby"
];

export function randomWord() {
    return PROGRAMING_LANG[Math.floor(Math.random() * PROGRAMING_LANG.length)];
}
```

```
import { PROGRAMING_LANG, randomWord } from "../words";

test("Output one of the words", () => {
  expect(PROGRAMING_LANG).toContain(randomWord());
});
```

3. Mock'ai

Rašant testus buvo privaloma imituoti aplinką, kurioje paprastai butų vykdomi testai. Tam įgyvendinti buvo naudojami mock'ai. Šiuo atveju testuojamas komponentas turėjo savo būseną, ir jos nustatymo funkciją. Buvo sukurta dirbtinė būsena ir jos nustatymo funkcija.

```
let mockState = { mistake: 100, guessed: new Set(), answer: "python" };
const setState = state => {
  mockState = state;
};
```

Kai kuriais atvejai reikėjo sukurti ir kitų dirbtinių objektų, pvz. mygtuko paspaudimo event'o objektą. Kadangi testuojama funkcija naudoja tik dalį to objekto, tai ir buvo sukurtas dalinis dirbtinis event'o objektas

```
let fakeEvt = { target: { value: "r" } };
```

Taip pat buvo testuojamas ar komponentas teisingai atvaizduojamas ir sukuriamas aplinkoje.

```
it("renders without crashing", () => {
  const div = document.createElement("div");
  ReactDom.render(<App />, div);
});
```

Bet komponentas naudojo paveiksliukus, kurie buvo importuojami. Jest biblioteka negalėjo atkurti tikrų komponento naudojamų paveiksliukų bei stiliaus failo, todėl buvo sukurtos jų dirbtiniai variantai, pasinaudojant mock funkcija.

```
jest.mock('./images/0.jpg')
jest.mock('./images/1.jpg')
jest.mock('./images/2.jpg')
jest.mock('./images/3.jpg')
jest.mock('./images/4.jpg')
jest.mock('./images/5.jpg')
jest.mock('./images/6.jpg')
jest.mock('./index.css')
```

Testuojant pagrindinio komponento sukurimą, reikėjo sukurti dirbtinę DOM aplinką.

```
import ReactDOM from "react-dom";

jest.mock('react-dom')

require('../../index')

test('Renders the application', () => {
  expect(ReactDOM.render).toBeCalled()
})
```

4. Rezultatai

Visi testai įvykdomi tvarkingai, be jokių klaidų. Testais padengta 100% testuojamos sistemos.

```
Test Suites: 4 passed, 4 total
Tests:       10 passed, 10 total
Snapshots:   0 total
Time:        5.759s
Ran all test suites.
```

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Line #s
All files	100	100	100	100	
src	100	100	100	100	
App.js	100	100	100	100	
index.js	100	100	100	100	
src/components	100	100	100	100	
Hangman.js	100	100	100	100	
hangman-functions.js	100	100	100	100	
words.js	100	100	100	100	

5. Išvados

Laboratorinio darbo metu buvo išsirinkta testuojama sistema, ji išanalizuota ir pilnai ištestuota. Išmokta naudotis jest testavimo įrankiu, bei kurti mock‘us. Testais padengta 100% kodo pagal jest testavimo įrankį.