Gra w życie Conwaya

Politechnika Śląska

Autorzy :
Szymon Babula, Krystian Barczak
Aleksander Boronowski, Krzysztof Dragon
Wydział Matematyki Stosowanej
Kierunek Informatyka
V semestr - Grupa 2C

Spis treści

1 Opis programu

Webowa wersja gry w życie Conwaya. Oprócz podstawowych zasad i funkcjonalności wersja ta posiada takie funkcje jak:

- Dostosowanie planszy do różnych rozmiarów ekranów w urządzeniach mobilnych
- Wirtualizację ciągłą oraz krokową wraz ze zmienną szybkością
- Możliwość wyboru rozmiaru planszy
- Zapis oraz odczyt planszy

Program został wykonany w celu projektu zaliczeniowego z przedmiotu Inżynieria oprogramowania.

2 Instrukcja obsługi

1. **Opis gry**

   Gra toczy się na planszy podzielonej na kwadratowe komórki. Każda komórka ma ośmiu „sąsiadów" czyli komórki przylegające do niej bokami i rogami. Każda komórka może znajdować się w jednym z dwóch stanów: włączona lub wyłączona. Stany komórek zmieniają się w pewnych jednostkach czasu. Po tym czasie wszystkie komórki zmieniają swój stan dokładnie w tym samym momencie, a stan komórki zależy tylko od liczby jej „żywych" sąsiadów.

2. **Reguły gry**

   - **Standardowa**
     Wyłączona komórka, która ma dokładnie 3 żywych sąsiadów, staje się żywa w następnej jednostce czasu. Jeśli żywa komórka z 2 albo 3 żywymi sąsiadami pozostaje nadal żywa. Natomiast jeśli liczba żywych sąsiadów jest inna niż 2 lub 3, komórka umiera (z „samotności" albo „zatłoczenia").

   - **Maze**
     Wyłączona komórka, która ma dokładnie 3 żywych sąsiadów, staje się żywa w następnej jednostce czasu (rodzi się). Żywa komórka z 1,2,3,4 lub 5 żywymi sąsiadami pozostaje nadal żywa. Przy innej liczbie sąsiadów umiera (z „samotności" albo „zatłoczenia")

   - **High Life**
     Wyłączona komórka, która ma 3 lub 6 żywych sąsiadów, staje się żywa w następnej jednostce czasu (rodzi się). Żywa komórka z 2 albo 3 żywymi sąsiadami pozostaje nadal żywa. Przy innej liczbie sąsiadów umiera (z „samotności" albo „zatłoczenia")

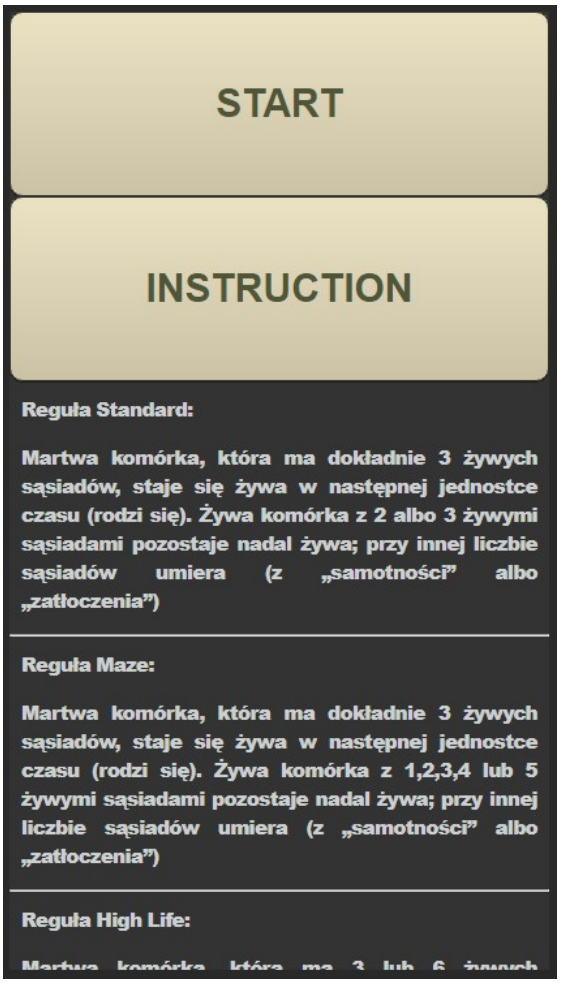
- **Ameba**
  Wyłączona komórka, która ma 3,5 lub 7 żywych sąsiadów, staje się żywa w następnej jednostce czasu (rodzi się). Żywa komórka z 1,3,5 lub 8 żywymi sąsiadami pozostaje nadal żywa. Przy innej liczbie sąsiadów umiera (z „samotności" albo „zatłoczenia")

- **Koral**
  Wyłączona komórka, która ma dokładnie 3 żywych sąsiadów, staje się żywa w następnej jednostce czasu (rodzi się). Żywa komórka z 4,5,6,7 lub 8 żywymi sąsiadami pozostaje nadal żywa. Przy innej liczbie sąsiadów umiera (z „samotności" albo „zatłoczenia")

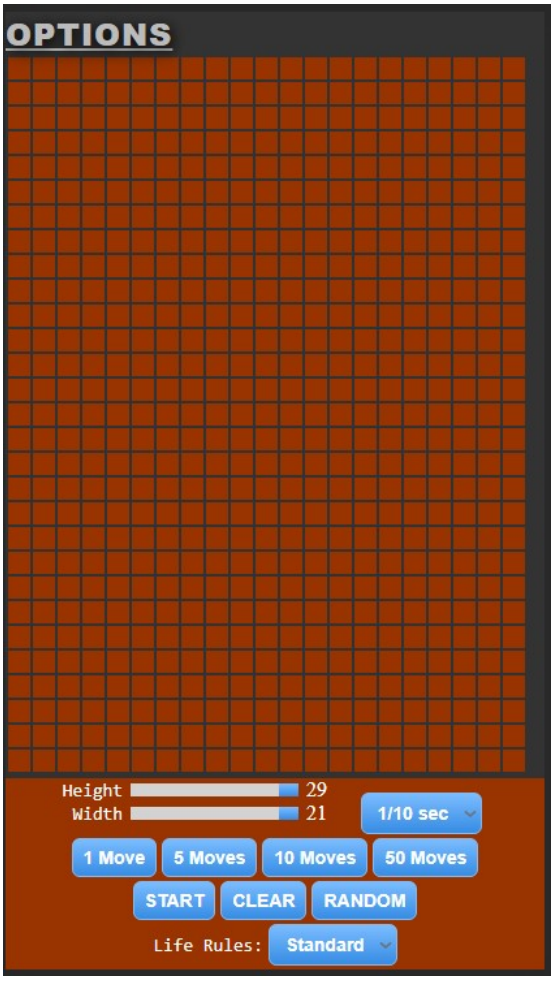
3. **Menu główne**

   Po wejściu na stronę wyświetla się menu opcji do wyboru:

   - **Start** - przejście do planszy gry
   - **Instruction** - wyświetlenie instrukcji gry na dole strony



Start

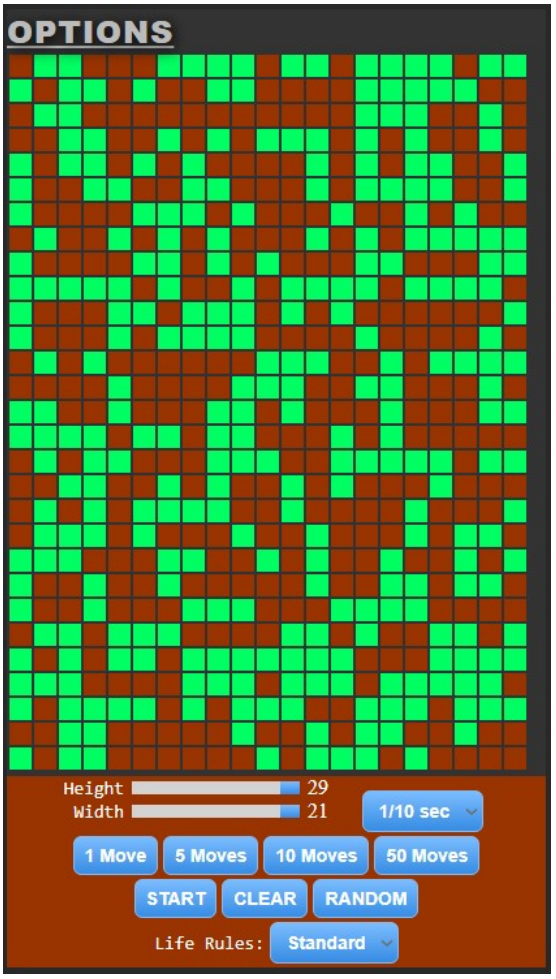


Ekran gry składa się z planszy oraz ustawień i sposobu rozgrywki. Kliknięcie na kwadrat powoduje, że zmienia się stan danej komórki. Można wybrać ile ruchów ma się wykonać, ustawić interwał automatycznego ruchu a także zmienić rozmiar planszy, wyczyścić ją lub zapełnić w sposób losowy. Możliwe są również zmiany reguł życia komórek.

Plansza

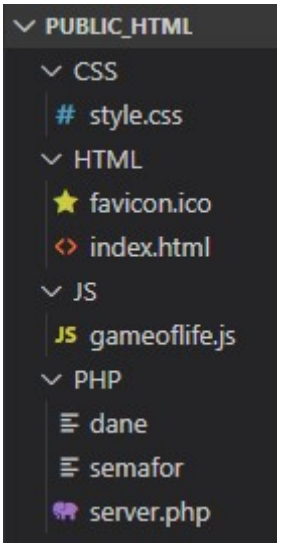
Ponadto pod przyciskiem "Options" znajduje się rozwijane menu, w którym znajdują się takie opcje jak Load - wczytanie planszy, Save - zapisanie aktualnego stanu planszy, Quit - wyjście z aktualnej rozgrywki. Kliknięcie pustego pola nad przyciskiem "Save lub pola z datą i godziną wyświetli listę dostępnych zapisów. Po wybraniu wystarczy kliknąć "Load", a wcześniej zapisana plansza zostanie wczytana. Pliki z zapisaną planszą znajdują się na serwerze.

Przykładowy wygląd planszy z losowo zaznaczonymi komórkami wygląda następująco:



3 Specyfikacja techniczna

Podział na pliki:



Kompilacja projektu:

Do stworzenia projektu wykorzystany został program Visual Studio Code oraz przeglądarki Google Chrome, Firefox, Opera. Menu główne napisane zostało w języku HTML oraz CSS w pełnej responsywności dla urządzeń mobilnych. Funkcjonalność gry natomiast napisana została w języku JavaScript, jQuery i PHP.
W celach testowych służyły urządzenia posiadające system Android oraz IOS posiadające różne przeglądarki(m.in. Safari, Internet Explorer).

4 Szczegóły techniczne

1. **Funkcja odpowiedzialna za życie komórek**

Funkcja odpowiedzialna za życie komórek napisana jest w języku JavaScript. Są to dwie funkcje, które odpowiadają za zliczanie sąsiadów danej komórki oraz za zmianę stanu komórki jeśli to potrzebne.

Funkcja sprawdzająca sąsiadów:

```
function countNeighbors(row, col) {
    var count = 0;
    if (row - 1 >= 0) {
        if (grid[row - 1][col] == 1) count++;
    }
    if (row - 1 >= 0 && col - 1 >= 0) {
        if (grid[row - 1][col - 1] == 1) count++;
    }
    if (row - 1 >= 0 && col + 1 < cols) {
        if (grid[row - 1][col + 1] == 1) count++;
    }
    if (col - 1 >= 0) {
        if (grid[row][col - 1] == 1) count++;
    }
    if (col + 1 < cols) {
        if (grid[row][col + 1] == 1) count++;
    }
    if (row + 1 < rows) {
        if (grid[row + 1][col] == 1) count++;
    }
    if (row + 1 < rows && col - 1 >= 0) {
        if (grid[row + 1][col - 1] == 1) count++;
    }
    if (row + 1 < rows && col + 1 < cols) {
        if (grid[row + 1][col + 1] == 1) count++;
    }
    return count;
}
```

Funkcja odpowiedzialna za stan komórki:

```
function applyRules(row, col) {
    var numNeighbors = countNeighbors(row, col);
    if (grid[row][col] == 1) {
        if (numNeighbors < 2) {
            nextGrid[row][col] = 0;
        } else if (numNeighbors == 2 || numNeighbors == 3) {
            nextGrid[row][col] = 1;
        } else if (numNeighbors > 3) {
            nextGrid[row][col] = 0;
        }
    } else if (grid[row][col] == 0) {
        if (numNeighbors == 3) {
            nextGrid[row][col] = 1;
        }
    }
}
```

2. **Funkcja odpowiedzialna za rysowanie planszy**

Funkcja odpowiedzialna za rysowanie planszy napisana jest w JavaScript. Funkcja ta tworzy tabelę o zadanych rozmiarach, a następnie zapełnia ją wyłączonymi ("martwymi") komórkami.

```
function createTable() {
    var gridContainer = document.getElementById('gridContainer');
    if (!gridContainer) {
        console.error("Problem: No div for the grid table!");
    }
    var table = document.createElement("table");
    table.setAttribute("id", "table");

    for (var i = 0; i < rows; i++) {
        var tr = document.createElement("tr");
        for (var j = 0; j < cols; j++) { //
            var cell = document.createElement("td");
            cell.setAttribute("id", i + "-" + j);
            cell.setAttribute("class", "dead");
            cell.onclick = cellClickHandler;
            tr.appendChild(cell);
        }
        table.appendChild(tr);
    }
    gridContainer.appendChild(table);
}
```

3. **Funkcje odpowiedzialne za zapis oraz odczyt planszy**

Funkcje odpowiedzialne za zapis oraz odczyt planszy napisane są w języku PHP oraz JSON. Zapis polega na sczytaniu oraz zapisaniu stanu każdej wyświetlanej komórki oraz wysłanie jej na serwer w JSON. Następnie w PHP dane zapisane są w pliku, której nazwa jest datą i godziną zapisu. Odczyt polega na wysłanie na serwer prośby do odczytu zawartości wybranego pliku. Serwer wysyła dane z pliku do JavaScript a ten wysyła je do funkcji, która wypełnia tabelę otrzymanymi danymi.

Zapis pliku w JSON:

```
function saveButton(table){
    var data = [];
    for (var i=0; i<table.rows.length; i++) {
        var rawData = {}
        for (var j=0; j<table.rows[i].cells.length; j++) {
            let id = table.rows[i].cells[j].id;
            let state = table.rows[i].cells[j].className;
            rawData[j] =
            {
                "id":id,
                "stan":state
            };

        }
        data.push(rawData);
    }
    AddToServer(data,rows,cols)
}
```

Odczyt pliku JSON:

```
function loadTable(){
    if(document.getElementById('plik').value){
        var request = new XMLHttpRequest();
        var plik = document.getElementById('plik').value;
        request.onreadystatechange = function() {
            if (this.readyState == 4 && this.status == 200) {
                var response = JSON.parse(this.responseText);
                LoadGridAndPopulate(response);
            }
        }
        request.open("POST", "../PHP/server.php", true);
        request.send(JSON.stringify({
            polecenie: 1,
            plik: plik+".data"
        }));
    }
}
```

Działanie serwera w PHP:

```
if(isset($daneJSON['polecenie']))
    {
        $polecenie = intval($daneJSON['polecenie']);
        switch($polecenie)
        {
            case 1:
                $wybranyPlik = $daneJSON['plik'];
                $plik = fopen($wybranyPlik, "r") or die("Blad odczytu pliku");
                $odczytPlik = fread($plik, filesize("dane"));
                fclose($plik);

                echo $odczytPlik;
            break;

            case 2:
                $wynik = '';
                $data = date("Y-m-d_H-i-s");
                $name = $data.'.data';
                file_put_contents($name,$suroweDane);
            break;

            case 3:
                $wynik = '';
                $files = glob('{*.data}', GLOB_BRACE);
                foreach($files as $file){
                    $file = substr($file, 0, -5);
                    if ($wynik === '') {
                        $wynik = '<option value="'.$file.'">';
                    } else {
                        $wynik .= '<option value="'.$file.'">';
                    }
                }
                print_r($wynik);
            break;

            default:
                $wynik = array('status' => false, 'kod' => 3, 'wartosc' => 'Podane zostalo
                    zle polecenie');
        }
    }
```

4. **Funkcje odpowiedzialne za ustawienie reguł życia**

Każda z poniższych funkcji sprawdza ilość sąsiadów oraz czy dana komórka jest martwa czy żywa. Następnie w zależności od wybranej reguły ustawiany jest odpowiedni schemat kolorystyczny oraz sprawdzane są warunki życia pod kątem wybranej reguły.

Reguła Standard:

```
        if(rule === "standard"){
    var numNeighbors = countNeighbors(row, col);
    if (grid[row][col] == 1) {
        if (numNeighbors < 2) {
            nextGrid[row][col] = 0;
        } else if (numNeighbors == 2 || numNeighbors == 3) {
            nextGrid[row][col] = 1;

            if(numNeighbors == 2) {
                color[row][col] = "blue";
            }
            else {
                color[row][col] = "cyan";
            }
        } else if (numNeighbors > 3) {
            nextGrid[row][col] = 0;
        }
    } else if (grid[row][col] == 0) {
        if (numNeighbors == 3) {
            nextGrid[row][col] = 1;
            color[row][col] = "yellow";
        }
    }
}
```

Reguła Maze:

```
        if(rule === "maze"){
    var numNeighbors = countNeighbors(row, col);
    if (grid[row][col] == 1) {
        if (numNeighbors < 1) {
            nextGrid[row][col] = 0;
        } else if (numNeighbors == 1 || numNeighbors == 2 || numNeighbors == 3 ||
            numNeighbors == 4 || numNeighbors == 5) {
            nextGrid[row][col] = 1;
            if(numNeighbors == 1) {
                color[row][col] = "blue";
            }
            else if(numNeighbors == 2){
                color[row][col] = "cyan";
            }
            else if(numNeighbors == 3){
                color[row][col] = "yellow";
            }
            else if(numNeighbors == 4){
                color[row][col] = "white";
            }
            else{
                color[row][col] = "pink";
            }
        } else if (numNeighbors > 5) {
            nextGrid[row][col] = 0;
        }
    } else if (grid[row][col] == 0) {
        if (numNeighbors == 3) {
            nextGrid[row][col] = 1;
            color[row][col] = "black";
        }
    }
}
```

Reguła High Life:

```
if(rule === "highLife"){
    var numNeighbors = countNeighbors(row, col);
    if (grid[row][col] == 1) {
        if (numNeighbors < 2) {
            nextGrid[row][col] = 0;
        } else if (numNeighbors == 2 || numNeighbors == 3) {
            nextGrid[row][col] = 1;
            if(numNeighbors == 2){
                color[row][col] = "cyan";
            }
            else{
                color[row][col] = "pink";
            }
        } else if (numNeighbors > 3) {
            nextGrid[row][col] = 0;
        }
    } else if (grid[row][col] == 0) {
        if (numNeighbors == 3 || numNeighbors == 6) {
            nextGrid[row][col] = 1;
            color[row][col] = "black";
        }
    }
}
```

Reguła Ameba:

```
if(rule === "ameba"){
    var numNeighbors = countNeighbors(row, col);
    if (grid[row][col] == 1) {
        if (numNeighbors < 1) {
            nextGrid[row][col] = 0;
        } else if (numNeighbors == 1 ||  numNeighbors == 3 || numNeighbors == 5 ||
            numNeighbors == 8) {
            nextGrid[row][col] = 1;
            if(numNeighbors == 1) {
                color[row][col] = "blue";
            }
            else if(numNeighbors == 3){
                color[row][col] = "cyan";
            }
            else if(numNeighbors == 5){
                color[row][col] = "yellow";
            }
            else {
                color[row][col] = "pink";
            }
        } else if (numNeighbors > 8) {
            nextGrid[row][col] = 0;
        }
    } else if (grid[row][col] == 0) {
        if (numNeighbors == 3 || numNeighbors == 5|| numNeighbors == 7) {
            nextGrid[row][col] = 1;
            color[row][col] = "black";
        }
    }
}
```

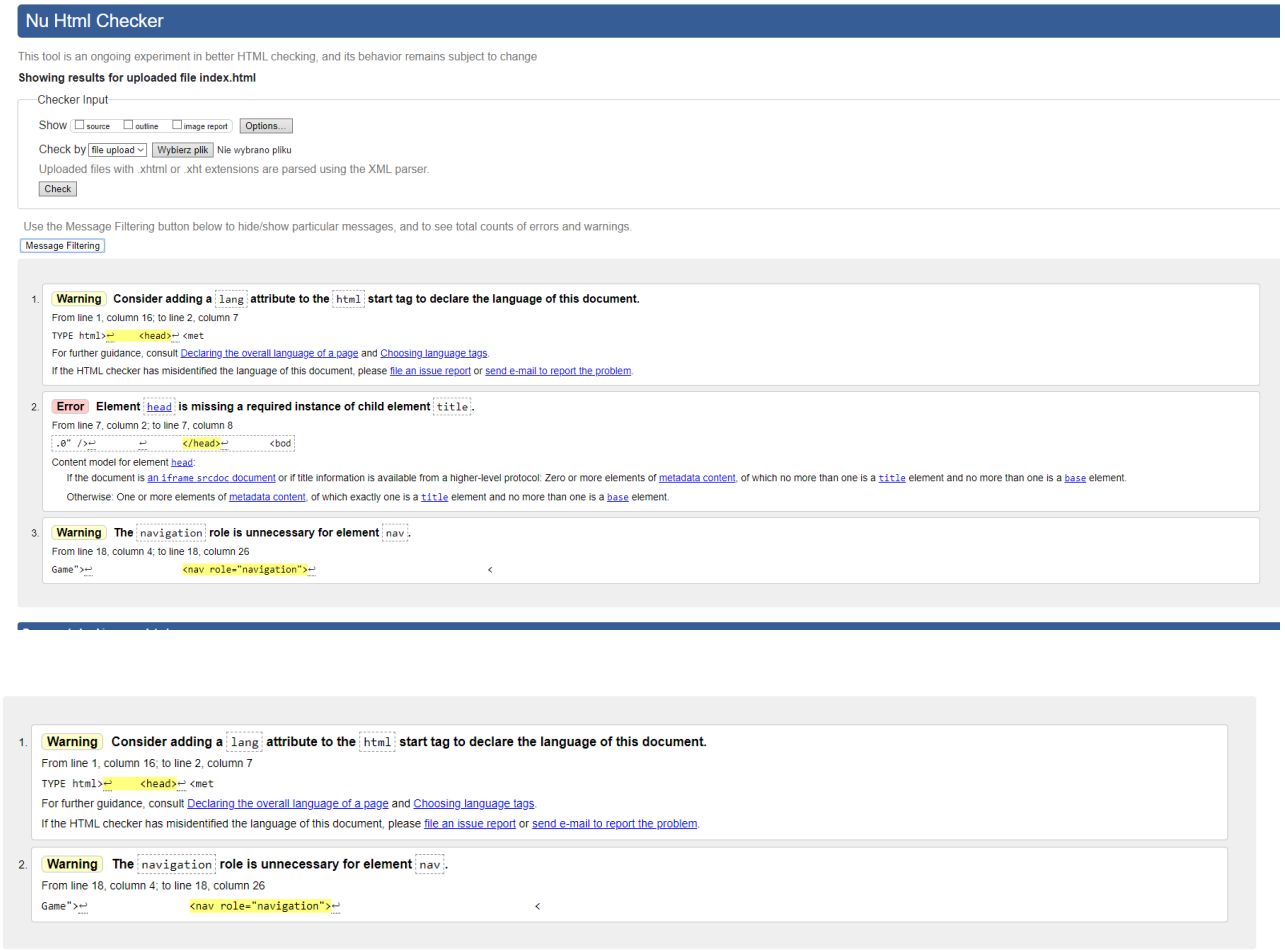
Reguła Koral:

```
if(rule === "koral"){
    var numNeighbors = countNeighbors(row, col);
    if (grid[row][col] == 1) {
        if (numNeighbors < 4) {
            nextGrid[row][col] = 0;
        } else if (numNeighbors == 4 ||  numNeighbors == 5 || numNeighbors == 6 ||
            numNeighbors == 7 || numNeighbors == 8) {
            nextGrid[row][col] = 1;
            if(numNeighbors == 4) {
                color[row][col] = "blue";
            }
            else if(numNeighbors == 5){
                color[row][col] = "cyan";
            }
            else if(numNeighbors == 6){
                color[row][col] = "yellow";
            }
            else if(numNeighbors == 7){
                color[row][col] = "white";
            }
            else{
                color[row][col] = "pink";
            }
        } else if (numNeighbors > 8) {
            nextGrid[row][col] = 0;
        }
    } else if (grid[row][col] == 0) {
        if (numNeighbors == 3) {
            nextGrid[row][col] = 1;
            color[row][col] = "black";
        }
    }
}
```

5. Testowanie aplikacji

Testowanie aplikacji odbywało się za pomocą testów manualnych, zautomatyzowanych (testy jednostkowe) i walidatora. Testy manualne wykonywane były na różnych urządzeniach posiadające system operacyjny Android i iOS. Przykładowe urządzenia, na których odbywały się testy: Xiaomi Mi 9T, Xiaomi Redmi Note 7, Sony Xperia Xa, iPad 3, Motorola e5.

Walidator:



Testy jednostkowe:

Do przeprowadzania testów jednostkowych wykorzystano biblioteki: chai oraz mocha. Testowaniu podlegało czyszczenie planszy i zliczanie sąsiadów.

Funkcje zliczania i czyszczenia:

```
let convert = {};

convert.countNeighbors = function(row, col,grid,cols,rows) {
    var count = 0;
    if (row - 1 >= 0) {
        if (grid[row - 1][col] == 1) count++;
    }
    if (row - 1 >= 0 && col - 1 >= 0) {
        if (grid[row - 1][col - 1] == 1) count++;
    }
    if (row - 1 >= 0 && col + 1 < cols) {
        if (grid[row - 1][col + 1] == 1) count++;
    }
    if (col - 1 >= 0) {
        if (grid[row][col - 1] == 1) count++;
    }
    if (col + 1 < cols) {
        if (grid[row][col + 1] == 1) count++;
    }
    if (row + 1 < rows) {
        if (grid[row + 1][col] == 1) count++;
    }
    if (row + 1 < rows && col - 1 >= 0) {
        if (grid[row + 1][col - 1] == 1) count++;
    }
    if (row + 1 < rows && col + 1 < cols) {
        if (grid[row + 1][col + 1] == 1) count++;
    }
    return count;
}
convert.resetGrids = function(grid) {
    for (var i = 0; i <= 9; i++) {
        for (var j = 0; j <= 9; j++) {
            grid[i][j] = 0;
        }
    }
        return grid;
}
if (module && module.exports) {
      module.exports = convert;
}
```

Ciało testu:

```
var assert = require('assert');
var expect = require('chai').expect;
var functions = require('C:/xampp/htdocs/public_html/functions.js'); //funkcje uzyte w naszym
    projekcie

function CreateAndPopulateGrid(row,col,boolean){ //funkcja pomocnicza do zapelniania tablicy
    MxN
  let array = [];
  for (var i = 0; i < row; i++) {
    array[i] = [];
  }
  if(boolean == true){
    for (var i = 0; i < row; i++) {
      for (var j = 0; j < col; j++) {
         array[i][j] = Math.round(Math.random());
      }
    }
  }else
  {
    for (var i = 0; i < row; i++) {
      for (var j = 0; j < col; j++) {
         array[i][j] = 0;
      }
    }
  }
  return array;
}

describe('Testy jednostkowe', function() {
  describe('Wszystkie funkcje, ktore zwracaja jakas wartosc,obiekt:', function() {
    it('resetGrids(grid)', function() {
      var grid = CreateAndPopulateGrid(10,10,true);
      var expected = CreateAndPopulateGrid(10,10,false);
      var result = functions.resetGrids(grid);
      expect(result).to.eql(expected);
    });
    it('countNeighbors(col,row,grid)', function() {
      var grid = [];
      for(var i = 0;i<3;i++)
      grid[i] = [];
      grid = CreateAndPopulateGrid(3,3,false);
      grid[0][1] = 1;
      grid[1][1] = 1;
      grid[2][1] = 1;
      grid[1][0] = 1;
      grid[1][2] = 1;
      var expected1 = 4;
      var expected2 = 3;
      var result1 = functions.countNeighbors(1,1,grid,3,3);
      var result2 = functions.countNeighbors(0,1,grid,3,3);
      assert.equal(result1,expected1);
      assert.equal(result2,expected2);
    });
  });
});
```

Testy i rezultaty:

```
Testy jednostkowe
  Wszystkie funkcje, które zwracają jakąś wartość,obiekt:
    √ resetGrids(grid)
    1) countNeighbors(col,row,grid)


1 passing (25ms)
1 failing

1) Testy jednostkowe
     Wszystkie funkcje, które zwracają jakąś wartość,obiekt:
       countNeighbors(col,row,grid):

    AssertionError [ERR_ASSERTION]: 3 == 4
    + expected - actual

    -3
    +4

    at Context.<anonymous> (test\unit_test.js:50:14)
    at processImmediate (internal/timers.js:439:21)
```

```
C:\xampp\htdocs\public_html>npm test

> public_html@1.0.0 test C:\xampp\htdocs\public_html
> mocha --watch --watch-extensions js:babel-register



  Testy jednostkowe
    Wszystkie funkcje, które zwracają jakąś wartość,obiekt:
      √ resetGrids(grid)
      √ countNeighbors(col,row,grid)


  2 passing (6ms)
```

Bibliografia

[1] *https://pl.wikipedia.org/wiki/Gra_w_życie*

[2] *https://www.sadistic.pl/gra-w-zycie-vt132225,15.htm*

[3] *https://junioritsociety.wordpress.com/2018/01/25/game-of-life/*

[4] *https://www.samouczekprogramisty.pl/game-of-life/*