Міністерство освіти і науки України

Національний університет «Львівська політехніка»

Кафедра «Електронних обчислювальних машин»



**Звіт**

з лабораторної роботи № 7-9

**з дисципліни:** «Кросплатформенні засоби програмування»

**Виконав:**

студент групи КІ-302

Козак О.П.

**Прийняв:**

Майдан М.В.

**Львів – 2023**

**Мета роботи:**

Гра 2048 на Javascript

**Вихідний код програми**

'use strict';

const button = document.querySelector('button');

const messageStart = document.querySelector('.message\_start');

const messageWin = document.querySelector('.message\_win');

const messageLose = document.querySelector('.message\_lose');

const gameScore = document.querySelector('.game\_score');

const fieldRows = document.querySelectorAll('.field\_row');

const size = 5;

let score = 0;

let isWin = false;

let newGameField;

let gameField = [, 0

  [0, 0, 0, 0, 0],

  [0, 0, 0, 0, 0],

  [0, 0, 0, 0, 0],

  [0, 0, 0, 0, 0],

];

button.addEventListener('click', e => {

  document.addEventListener('keydown', move);

  if (button.classList.contains('start')) {

    button.classList.replace('start', 'restart');

    button.innerText = 'Restart';

    messageStart.classList.add('hidden');

  } else {

    isWin = false;

    reset();

  }

  addNumber();

  addNumber();

  render();

});

function move(e) {

  newGameField = [...gameField];

  const leftArrow = 'ArrowLeft';

  const rightArrow = 'ArrowRight';

  const downArrow = 'ArrowDown';

  const upArrow = 'ArrowUp';

  switch (e.key) {

    case leftArrow:

      left(newGameField);

      break;

    case rightArrow:

      right(newGameField);

      break;

    case downArrow:

      down(newGameField);

      break;

    case upArrow:

      up(newGameField);

      break;

    default:

      return;

  }

  for (let row = 0; row < size; row++) {

    for (let col = 0; col < size; col++) {

      if (newGameField[row][col] !== gameField[row][col]) {

        gameField = newGameField;

        addNumber();

        render();

      }

    }

  }

  if (isWin) {

    messageWin.classList.remove('hidden');

  }

  if (!isPosibleToMove()) {

    messageLose.classList.remove('hidden');

    document.removeEventListener('keydown', move);

  }

}

function reset() {

  score = 0;

  gameField = [

    [0, 0, 0, 0, 0],

    [0, 0, 0, 0, 0],

    [0, 0, 0, 0, 0],

    [0, 0, 0, 0, 0],

  ];

  if (!messageLose.classList.contains('hidden')) {

    messageLose.classList.add('hidden');

  }

  if (!messageWin.classList.contains('hidden')) {

    messageWin.classList.add('hidden');

  }

};

function addNumber() {

  const [randomY, randomX] = findEmptyCell();

  gameField[randomY][randomX] = Math.random() < 0.9 ? 2 : 4;

};

function findEmptyCell() {

  const emptyCells = [];

  gameField.forEach((row, rowIndex) => {

    row.forEach((cell, columnIndex) => {

      if (cell === 0) {

        emptyCells.push([rowIndex, columnIndex]);

      }

    });

  });

  return emptyCells[Math.floor(Math.random() \* emptyCells.length)];

}

function render() {

  gameField.forEach((row, rowIndex) => {

    row.forEach((cell, colIndex) => {

      const elem = fieldRows[rowIndex].children[colIndex];

      if (cell === 0) {

        elem.textContent = '';

        elem.className = 'field\_cell';

      } else {

        elem.textContent = cell;

        elem.className = `field\_cell field\_cell--${cell}`;

      }

    });

  });

  gameScore.textContent = score;

};

function left() {

  if (!checkRows()) {

    return;

  }

  newGameField = newGameField.map(row => {

    const newRow = row.filter(cell => cell !== 0);

    newRow.forEach((cell, index) => {

      if (cell === newRow[index + 1]) {

        newRow[index] \*= 2;

        newRow.splice(index + 1, 1);

        score += newRow[index];

        if (newRow[index] === 2048) {

          isWin = true;

        }

      }

    });

    return newRow.concat(Array(size - newRow.length).fill(0));

  });

}

function right() {

  if (!checkRows()) {

    return;

  }

  reverseRows();

  left();

  reverseRows();

}

function reverseRows() {

  newGameField.forEach(row => row.reverse());

}

function down() {

  transposeGameField();

  right();

  transposeGameField();

}

function up() {

  transposeGameField();

  left();

  transposeGameField();

}

function transposeGameField() {

  newGameField = newGameField[0]

    .map((\_, colIndex) => newGameField.map(row => row[colIndex]));

}

function isPosibleToMove() {

  if (checkRows()) {

    return true;

  }

  transposeGameField();

  return checkColumns();

}

function checkRows() {

  let hasDuplicates = false;

  newGameField.forEach((row) => {

    if (row.some((cell, index) => cell === 0 || cell === row[index + 1])) {

      hasDuplicates = true;

    }

  });

  return hasDuplicates;

}

function checkColumns() {

  let hasDuplicates = false;

  newGameField.forEach((row) => {

    row.forEach((cell, index) => {

      if (cell === row[index + 1]) {

        hasDuplicates = true;

      }

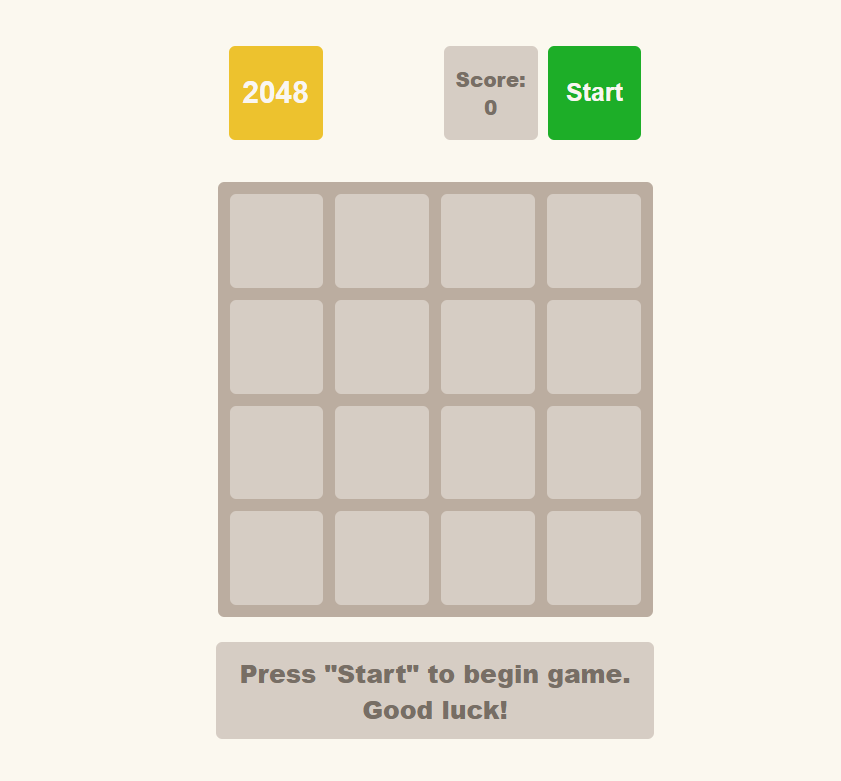
    });

  });

  return hasDuplicates;

}

**Результат виконання програми**



**Відповіді на контрольні запитання**

1. Дайте визначення терміну «параметризоване програмування».

Параметризоване програмування - це підхід до програмування, де класи та методи можуть приймати типи даних як параметри, що дозволяє створювати більш загальні та змінні структури даних та методи.

1. Розкрийте синтаксис визначення простого параметризованого класу.

class MyGenericClass<T> {

// Код класу з використанням параметра T

}

1. Розкрийте синтаксис створення об’єкту параметризованого класу.

MyGenericClass<Integer> obj = new MyGenericClass<>();

1. Розкрийте синтаксис визначення параметризованого методу.

<T> void myGenericMethod(T value) {

// Код методу з використанням параметра T

}

1. Розкрийте синтаксис виклику параметризованого методу.

MyClass.<Integer>myGenericMethod(42); // Виклик методу з параметром Integer

1. Яку роль відіграє встановлення обмежень для змінних типів?

Встановлення обмежень для змінних типів дозволяє обмежити допустимий тип даних, який можна використовувати як параметр типу.

1. Як встановити обмеження для змінних типів?

<T extends Number> void myMethod(T value) { /\* ... \*/ }

1. Розкрийте правила спадкування параметризованих типів.

Правила спадкування параметризованих типів включають здатність успадковувати параметризовані класи та методи з інтерфейсів або суперкласів та можливість використовувати абстрактні типи даних.

1. Яке призначення підстановочних типів?

Підстановочні типи (wildcards) використовуються для створення більш загальних методів та класів, які можуть приймати об'єкти різних типів.

1. Застосування підстановочних типів.

Застосування підстановочних типів включає використання ? для прийняття об'єктів різних типів, таких як <?> для довільного типу або <? extends T> для обмеження типу до підкласу T або <? super T> для обмеження типу до суперкласу T.

**Висновок**

Ознайомився з використанням параметризованого програмування. Створив клас який реалізує предметну область «множина». Та розробив клас драйвер який показує роботу параметризованого класу контейнера.