Федеральное государственное бюджетное образовательное учреждение

высшего образования

«Омский государственный технический университет»

Факультет информационных технологий и компьютерных систем

Кафедра «Информатика и вычислительная техника»

РАСЧЕТНО-ГРАФИЧЕСКАЯ РАБОТА

По дисциплине «Проектирование и тестирование программного обеспечения»

По теме «Проектирование и тестирование приложения «Программистский калькулятор» на языке C#»

Преподаватель:

Тюньков Д.А.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(Подпись, дата)

Студент группы ИВТ-172:

Зайцев В.А.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(Подпись, дата)

Омск 2019

**Содержание**

[ВВЕДЕНИЕ 2](#_Toc8568860)

[Описание проекта 2](#_Toc8568861)

[Разработка юнит-тестов 2](#_Toc8568862)

[Создание юнит-тестов 2](#_Toc8568863)

[Формулировка задания и схема алгоритма 2](#_Toc8568864)

[Приложение А 2](#_Toc8568865)

## ВВЕДЕНИЕ

Существующие на сегодняшний день методы тестирования ПО не позволяют однозначно и полностью выявить все дефекты и установить корректность функционирования анализируемой программы, поэтому все существующие методы тестирования действуют в рамках формального процесса проверки исследуемого или разрабатываемого ПО.

Такой процесс формальной проверки или верификации может доказать, что дефекты отсутствуют с точки зрения используемого метода. (То есть нет никакой возможности точно установить или гарантировать отсутствие дефектов в программном продукте с учётом человеческого фактора, присутствующего на всех этапах жизненного цикла ПО).

Существует множество подходов к решению задачи тестирования и верификации ПО, но эффективное тестирование сложных программных продуктов — это процесс в высшей степени творческий, не сводящийся к следованию строгим и чётким процедурам или созданию таковых.

Целью работы является создание программы и ряда модульных тестов к ней. Язык программы был выбран C#

## Описание проекта

Данная работа для изучения особенностей модульного тестирования программы на языке C#. Программы сделаны в среде “Microsoft Visual Studio 2019”. Данная программа является консольным аналогом популярной мобильной головоломки «2048».

## Разработка юнит-тестов

С целью автоматизации проверки работоспособности приложений было разработано множество разных подходов и типов тестирования. Одним из таких подходов является модульное тестирование, заключающиеся в том, что для конкретной ситуации задаются только входные значения и ожидаемый результат, если после выполнения теста они совпадают, тест считается пройденным, иначе – проваленным.

Цель модульного тестирования – изолировать отдельные части программы и показать, что по отдельности эти части работоспособны.

## Создание юнит-тестов

Модульные тесты были выделены в отдельный проект в той же сборке, что и тестируемый проект. Таким образом, из присутствующих в проекте классов были протестированы все калькуляторы и фабрики на все возможных наборах входных и выходных данных. Проект с модульными тестами находится в приложении В.

Созданные юнит-тесты состоят из 3 частей: инициализация данных (arrange), само действие (act) и постусловие, то есть то, что должно получится в итоге (assert).

Тестирование фабрик в целом схоже с тестированием калькуляторов, с той разницей, что проверяется не равенство значений, а принадлежность созданного фабрикой объекта определённому классу.

Также с помощью модульных тестов проверена работа приложения в ошибочных ситуациях.

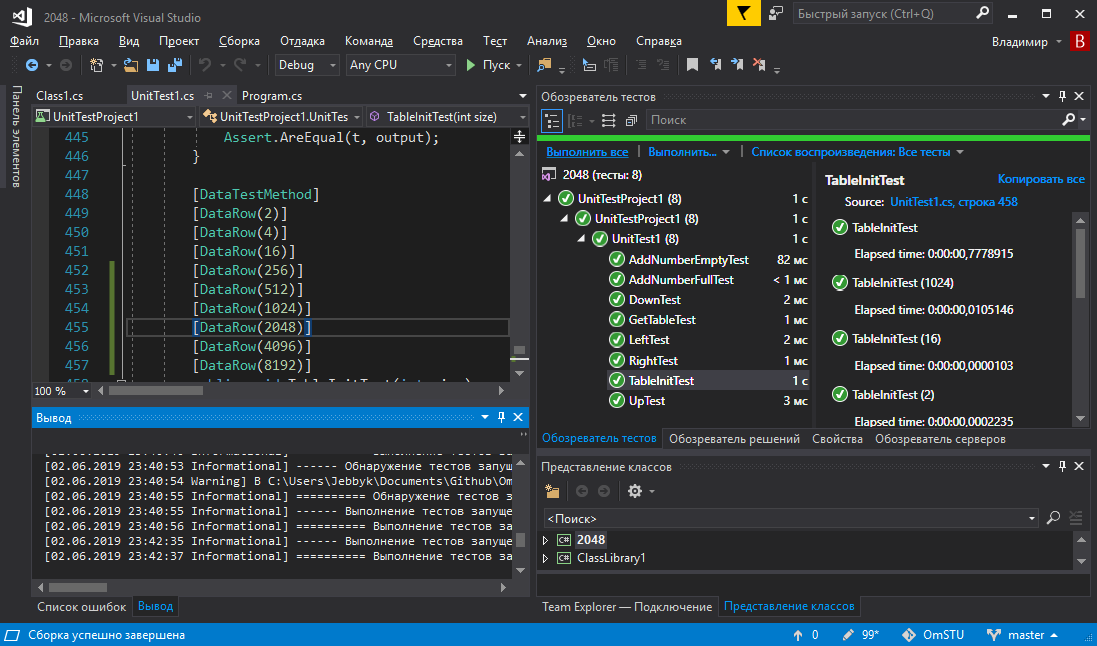


Рисунок 1 – Успешное прохождение тестов

## Формулировка задания и схема алгоритма

1. Формулировка задания

Написать программу «2048», являющуюся консольным аналогом популярной мобильной игры 2048.

1. Схема алгоритма для Program.cs

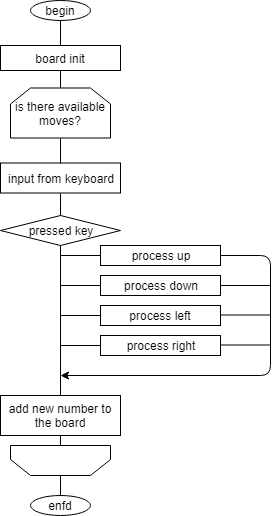


Рисунок 2 – Блок-схема алгоритма



Рисунок 3 – Внешний вид программы

## Приложение А

Текст программы с основными классами:

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace ClassLibrary1

{

public class Board

{

public int[,] table;

int size;

public int AddNumber()

{

bool empty = false;

foreach (int i in table)

{

if (i == 0) { empty = true; break; }

}

if (!empty) return -1;

Random r = new Random();

int x, y;

x = r.Next(0, size);

y = r.Next(0, size);

while (table[y, x] != 0)

{

x = r.Next(0, size);

y = r.Next(0, size);

}

table[y, x] = 2;

return 1;

}

public Board()

{

size = 3;

table = new int[3, 3];

AddNumber();

AddNumber();

}

public Board(int s)

{

size = s;

table = new int[s, s];

AddNumber();

AddNumber();

}

public string GetTable()

{

string s = "";

for (int i = 0; i < size; i++)

{

for (int j = 0; j < size; j++)

{

s = s + table[i, j] + " ";

}

s = s + "\n";

}

return s;

}

public void Right()

{

for (int i = 0; i < size; i++)

{

int num = 0;

bool flag = true;

for (int j = 0; j < size; j++)//combining numbers

{

if (table[i, j] != 0)

{

if (flag)

{

flag = false;

num = table[i, j];

}

else

{

if (num == table[i, j])

{

flag = true;

table[i, j] = num + num;

table[i, j - 1] = 0;

num = 0;

}

else

{

//flag = true;

num = table[i, j];

}

}

}

}

for (int j = size - 1; j >= 0; j--)//mooving numbers

{

if (table[i, j] != 0)

{

int shift = j;

while (shift + 1 < size && table[i, shift + 1] == 0)

{

table[i, shift + 1] = table[i, shift];

table[i, shift] = 0;

shift++;

}

}

}

}

}

public void Left()

{

for (int i = 0; i < size; i++)

{

int num = 0;

bool flag = true;

for (int j = size - 1; j >= 0; j--)//combining numbers

{

if (table[i, j] != 0)

{

if (flag)

{

flag = false;

num = table[i, j];

}

else

{

if (num == table[i, j])

{

flag = true;

table[i, j] = num + num;

table[i, j + 1] = 0;

num = 0;

}

else

{

//flag = true;

num = table[i, j];

}

}

}

}

for (int j = 0; j < size; j++)//mooving numbers

{

if (table[i, j] != 0)

{

int shift = j;

while (shift - 1 >= 0 && table[i, shift - 1] == 0)

{

table[i, shift - 1] = table[i, shift];

table[i, shift] = 0;

shift--;

}

}

}

}

}

public void Up()

{

for (int j = 0; j < size; j++)

{

int num = 0;

bool flag = true;

for (int i = size - 1; i >= 0; i--)

{

if (table[i, j] != 0)

{

if (flag)

{

flag = false;

num = table[i, j];

}

else

{

if (num == table[i, j])

{

flag = true;

table[i, j] = num + num;

table[i + 1, j] = 0;

num = 0;

}

else

{

//flag = true;

num = table[i, j];

}

}

}

}

for (int i = 0; i < size; i++)//mooving numbers

{

if (table[i, j] != 0)

{

int shift = i;

while (shift - 1 >= 0 && table[shift - 1, j] == 0)

{

table[shift - 1, j] = table[shift, j];

table[shift, j] = 0;

shift--;

}

}

}

}

}

public void Down()

{

for (int j = 0; j < size; j++)

{

int num = 0;

bool flag = true;

for (int i = 0; i < size; i++)

{

if (table[i, j] != 0)

{

if (flag)

{

flag = false;

num = table[i, j];

}

else

{

if (num == table[i, j])

{

flag = true;

table[i, j] = num + num;

table[i - 1, j] = 0;

num = 0;

}

else

{

//flag = true;

num = table[i, j];

}

}

}

}

for (int i = size - 1; i >= 0; i--)//mooving numbers

{

if (table[i, j] != 0)

{

int shift = i;

while (shift + 1 < size && table[shift + 1, j] == 0)

{

table[shift + 1, j] = table[shift, j];

table[shift, j] = 0;

shift++;

}

}

}

}

}

public int UpdateTable(ConsoleKey key)

{

switch (key)

{

case ConsoleKey.UpArrow:

{

Console.Clear();

Up();

return AddNumber();

}

case ConsoleKey.DownArrow:

{

Console.Clear();

Down();

return AddNumber();

}

case ConsoleKey.RightArrow:

{

Console.Clear();

Right();

return AddNumber();

}

case ConsoleKey.LeftArrow:

{

Console.Clear();

Left();

return AddNumber();

}

default: { Console.Clear(); return 1; }

}

}

}

}

Текст основного файла программы:

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using ClassLibrary1;

namespace \_2048

{

class Program

{

static int state = 0;

static void Game()

{

Console.Clear();

Board board = new Board();

Console.Write(board.GetTable());

while (true)

{

ConsoleKey key;

key = Console.ReadKey().Key;

if (key == ConsoleKey.Escape) { state = 0; return; }

if (board.UpdateTable(key) == 1)

{

Console.Write(board.GetTable());

}

else

{

Console.Clear();

Console.WriteLine("Game over, no places for new numbers");

if (Console.ReadKey().Key == ConsoleKey.Escape ) { state = 0; return; };

}

}

}

/\*static void LeaderBoard()

{

Console.Clear();

Console.WriteLine("Leader board!! There only best players... not you))))");

if (Console.ReadKey().Key == ConsoleKey.Escape) { Console.Clear(); state = 0; return; }

}\*/

static void Menu()

{

int currentItem = 0;

String[] items = { "new game", "exit" };

while(true)

{

Console.Clear();

for (int i = 0; i < items.Length; i++)

{

if (i == currentItem) items[i] = items[i].ToUpper();

else items[i] = items[i].ToLower();

Console.WriteLine(items[i]);

}

ConsoleKey key = Console.ReadKey().Key;

if(key == ConsoleKey.UpArrow)

{

if(currentItem > 0)

{

currentItem--;

}

else

{

currentItem = items.Length-1;

}

}

if (key == ConsoleKey.DownArrow)

{

if (currentItem < items.Length - 1)

{

currentItem++;

}

else

{

currentItem = 0;

}

}

if(key == ConsoleKey.Escape)

{

return;

}

if(key == ConsoleKey.Enter)

{

state = currentItem+1;

return;

}

}

}

public static void Main()

{

while (true)

{

switch(state)

{

case 0:

{

Menu();

if (state == 0) return;

else break;

}

case 1:

{

Game();

break;

}

/\* case 2:

{

LeaderBoard();

break;

}\*/

case 2:

{

return;

}

default: break;

}

}

}

}

}

Текст файла содержащего тесты:

using System;

using Microsoft.VisualStudio.TestTools.UnitTesting;

using ClassLibrary1;

namespace UnitTestProject1

{

[TestClass]

public class UnitTest1

{

[DataTestMethod]

[DataRow(3)]

[DataRow(4)]

[DataRow(5)]

[DataRow(6)]

[DataRow(7)]

[DataRow(8)]

public void AddNumberEmptyTest(int size)

{

int count = 0;

bool check = true;

Board b = new Board(size);

while (check)

{

if (b.AddNumber() == 1)

{

count++;

}

else check = false;

}

Assert.AreEqual(size \* size - 2, count);

}

[TestMethod]

[DataRow(3)]

[DataRow(4)]

[DataRow(5)]

public void AddNumberFullTest(int size)

{

int count = 0;

bool check = true;

Board b = new Board(size);

for (int i = 2; i < size \* size; i++)

{

b.AddNumber();

}

while (check)

{

if (b.AddNumber() == 1)

{

count++;

}

else check = false;

}

Assert.AreEqual(0, count);

}

[DataTestMethod]

[DataRow(new int[] {

2, 0, 0,

0, 0, 0,

0, 0, 0 },

"200" +

"000" +

"000")]

[DataRow(new int[] {

0, 2, 0,

0, 0, 0,

0, 0, 0 },

"020" +

"000" +

"000")]

[DataRow(new int[] {

0, 0, 2,

0, 0, 0,

0, 0, 0 },

"002" +

"000" +

"000")]

[DataRow(new int[] {

0, 0, 0,

2, 0, 0,

0, 0, 0 },

"200" +

"000" +

"000")]

[DataRow(new int[] {

0, 0, 0,

0, 2, 0,

0, 0, 0 },

"020" +

"000" +

"000")]

[DataRow(new int[] {

0, 0, 0,

0, 0, 0,

0, 0, 2 },

"002" +

"000" +

"000")]

[DataRow(new int[] {

2, 2, 0,

2, 0, 0,

0, 0, 0 },

"420" +

"000" +

"000")]

[DataRow(new int[] {

2, 2, 0,

2, 0, 0,

4, 0, 0 },

"420" +

"400" +

"000")]

[DataRow(new int[] {

4, 2, 0,

2, 0, 0,

2, 0, 0 },

"420" +

"400" +

"000")]

[DataRow(new int[] {

1024, 2048, 4096,

1024, 2048, 4096,

2, 0, 0 },

"204840968192" +

"200" +

"000")]

public void UpTest(int[] input, string output)

{

Board b = new Board(3);

for(int i = 0; i < 3; i++)

{

for(int j = 0; j < 3; j++)

{

b.table[i, j] = input[i \* 3 + j];

}

}

b.Up();

string t = "";

for (int i = 0; i < 3; i++)

{

for (int j = 0; j < 3; j++)

{

t = t + b.table[i, j];

}

}

Assert.AreEqual(t, output);

}

[DataTestMethod]

[DataRow(new int[] {

2, 0, 0,

0, 0, 0,

0, 0, 0 },

"000" +

"000" +

"200")]

[DataRow(new int[] {

0, 2, 0,

0, 0, 0,

0, 0, 0 },

"000" +

"000" +

"020")]

[DataRow(new int[] {

0, 0, 2,

0, 0, 0,

0, 0, 0 },

"000" +

"000" +

"002")]

[DataRow(new int[] {

0, 0, 0,

2, 0, 0,

0, 0, 0 },

"000" +

"000" +

"200")]

[DataRow(new int[] {

0, 0, 0,

0, 2, 0,

0, 0, 0 },

"000" +

"000" +

"020")]

[DataRow(new int[] {

0, 0, 0,

0, 0, 0,

0, 0, 2 },

"000" +

"000" +

"002")]

[DataRow(new int[] {

2, 2, 0,

2, 0, 0,

0, 0, 0 },

"000" +

"000" +

"420")]

[DataRow(new int[] {

2, 2, 0,

2, 0, 0,

4, 0, 0 },

"000" +

"400" +

"420")]

[DataRow(new int[] {

4, 2, 0,

2, 0, 0,

2, 0, 0 },

"000" +

"400" +

"420")]

[DataRow(new int[] {

1024, 2048, 4096,

1024, 2048, 4096,

2, 0, 0 },

"000" +

"204800" +

"240968192")]

public void DownTest(int[] input, string output)

{

Board b = new Board(3);

for (int i = 0; i < 3; i++)

{

for (int j = 0; j < 3; j++)

{

b.table[i, j] = input[i \* 3 + j];

}

}

b.Down();

string t = "";

for (int i = 0; i < 3; i++)

{

for (int j = 0; j < 3; j++)

{

t = t + b.table[i, j];

}

}

Assert.AreEqual(t, output);

}

[DataTestMethod]

[DataRow(new int[] {

2, 0, 0,

0, 0, 0,

0, 0, 0 },

"200" +

"000" +

"000")]

[DataRow(new int[] {

0, 2, 0,

0, 0, 0,

0, 0, 0 },

"200" +

"000" +

"000")]

[DataRow(new int[] {

0, 0, 2,

0, 0, 0,

0, 0, 0 },

"200" +

"000" +

"000")]

[DataRow(new int[] {

0, 0, 0,

2, 0, 0,

0, 0, 0 },

"000" +

"200" +

"000")]

[DataRow(new int[] {

0, 0, 0,

0, 2, 0,

0, 0, 0 },

"000" +

"200" +

"000")]

[DataRow(new int[] {

0, 0, 0,

0, 0, 0,

0, 0, 2 },

"000" +

"000" +

"200")]

[DataRow(new int[] {

2, 2, 0,

2, 0, 0,

0, 0, 0 },

"400" +

"200" +

"000")]

[DataRow(new int[] {

2, 2, 0,

2, 0, 0,

4, 0, 0 },

"400" +

"200" +

"400")]

[DataRow(new int[] {

4, 2, 2,

2, 0, 0,

2, 0, 0 },

"440" +

"200" +

"200")]

[DataRow(new int[] {

1024, 1024, 0,

2048, 2048, 0,

4096, 4096, 0 },

"204800" +

"409600" +

"819200")]

public void LeftTest(int[] input, string output)

{

Board b = new Board(3);

for (int i = 0; i < 3; i++)

{

for (int j = 0; j < 3; j++)

{

b.table[i, j] = input[i \* 3 + j];

}

}

b.Left();

string t = "";

for (int i = 0; i < 3; i++)

{

for (int j = 0; j < 3; j++)

{

t = t + b.table[i, j];

}

}

Assert.AreEqual(t, output);

}

[DataTestMethod]

[DataRow(new int[] {

2, 0, 0,

0, 0, 0,

0, 0, 0 },

"002" +

"000" +

"000")]

[DataRow(new int[] {

0, 2, 0,

0, 0, 0,

0, 0, 0 },

"002" +

"000" +

"000")]

[DataRow(new int[] {

0, 0, 2,

0, 0, 0,

0, 0, 0 },

"002" +

"000" +

"000")]

[DataRow(new int[] {

0, 0, 0,

2, 0, 0,

0, 0, 0 },

"000" +

"002" +

"000")]

[DataRow(new int[] {

0, 0, 0,

0, 2, 0,

0, 0, 0 },

"000" +

"002" +

"000")]

[DataRow(new int[] {

0, 0, 0,

0, 0, 0,

0, 0, 2 },

"000" +

"000" +

"002")]

[DataRow(new int[] {

2, 2, 0,

2, 0, 0,

0, 0, 0 },

"004" +

"002" +

"000")]

[DataRow(new int[] {

2, 2, 0,

2, 0, 0,

4, 0, 0 },

"004" +

"002" +

"004")]

[DataRow(new int[] {

4, 2, 2,

2, 0, 0,

2, 0, 0 },

"044" +

"002" +

"002")]

[DataRow(new int[] {

1024, 1024, 0,

2048, 2048, 0,

4096, 4096, 0 },

"002048" +

"004096" +

"008192")]

public void RightTest(int[] input, string output)

{

Board b = new Board(3);

for (int i = 0; i < 3; i++)

{

for (int j = 0; j < 3; j++)

{

b.table[i, j] = input[i \* 3 + j];

}

}

b.Right();

string t = "";

for (int i = 0; i < 3; i++)

{

for (int j = 0; j < 3; j++)

{

t = t + b.table[i, j];

}

}

Assert.AreEqual(t, output);

}

[DataTestMethod]

[DataRow(new int[] { 0,0,0,0,0,0,0,0,0}, "0 0 0 \n0 0 0 \n0 0 0 \n")]

[DataRow(new int[] { 8192, 16384, 32768, 65536, 131072, 262144, 524288, 1048576, 2097152 }, "8192 16384 32768 \n65536 131072 262144 \n524288 1048576 2097152 \n")]

public void GetTableTest(int[] input, string output)

{

Board b = new Board();

for (int i = 0; i < 3; i++)

{

for (int j = 0; j < 3; j++)

{

b.table[i, j] = input[i \* 3 + j];

}

}

string t = b.GetTable();

Assert.AreEqual(t, output);

}

[DataTestMethod]

[DataRow(2)]

[DataRow(4)]

[DataRow(16)]

[DataRow(256)]

[DataRow(512)]

[DataRow(1024)]

[DataRow(2048)]

[DataRow(4096)]

[DataRow(8192)]

public void TableInitTest(int size)

{

Board b = new Board(size);

int key = 0;

foreach(int i in b.table)

{

key++;

key += i;

}

Assert.AreEqual(key, size \* size + 4);

}

}

}