Оглавление

[Введение 2](#_Toc4161397)

[Цель разработки 3](#_Toc4161398)

[Обзор аналогов 3](#_Toc4161399)

[2048 Kitty Cat Island 3](#_Toc4161400)

[City of 2048 3](#_Toc4161401)

[City 2048 4](#_Toc4161402)

[BoneSwiper 4](#_Toc4161403)

[2048 5](#_Toc4161404)

[Постановка задачи 5](#_Toc4161405)

[Описание алгоритма работы программы 6](#_Toc4161406)

[Исходный код программы 6](#_Toc4161407)

[Тестирование программы 23](#_Toc4161408)

[Системные требования 23](#_Toc4161409)

[Руководство по установке 23](#_Toc4161410)

# Введение

На данный момент существует великое множество компьютерных игр, так же разнообразны подходы в их создании.

Например, популярные в последнее время небольшие игры (чаще всего многопользовательские, через сеть Интернет) создаются по технологии Flash с использованием языка ActionScript. Такие игры крайне популярны на разнообразных развлекательных интернет-ресурсах и а социальных сетях.

Большие, сложные игры с реалистичной 3D графикой пишутся на C++, отдельные модули к ним могут быть написаны на других языках (например AI - «искусственный интеллект» - на Lisp). Вообще, язык для создания игры выбирается как компромисс между требованиями к игре, к компьютерному железу и к стоимости разработки.

На подавляющем большинстве мобильных телефонов установлены ARM процессоры. Поэтому игры для мобильных телефонов на платформе Java пишутся на специальном языке для мобильных телефонов MobileBasic, а также на диалекте языка Pascal - MIDletPascal.

Также существуют специальные программы для написания игр. Яркий пример - программа GameMaker, автором которой является Марк Овермас. Эта программа позволяет написать захватывающую игру даже без знания языков программирования!

В рамках учебной практики необходимо разработать небольшую программу игры в «2048» поле размера 3х3 клеток с применением графики на языке высокого уровня с использованием объектно-ориентированного программирования.

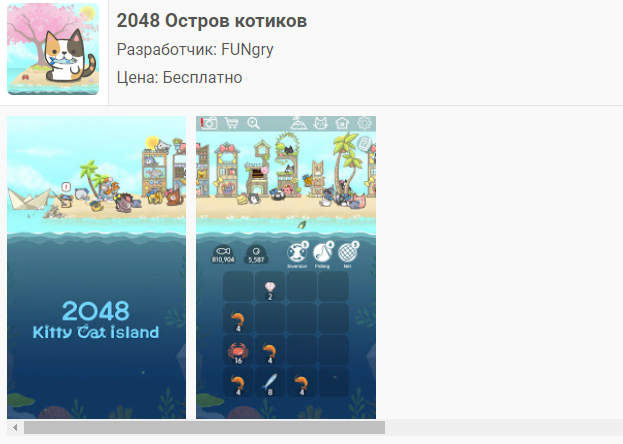
«2048» - популярная игра людей всех возрастов. Она отличается простотой, способствует развитию внимания и элементарной логики. В данной работе эта игра реализована на языке Pascal.

# Цель разработки

Создать интересную и логическую игру, для детей, игра будет в открытом доступе и все желающие могут воспользоваться.

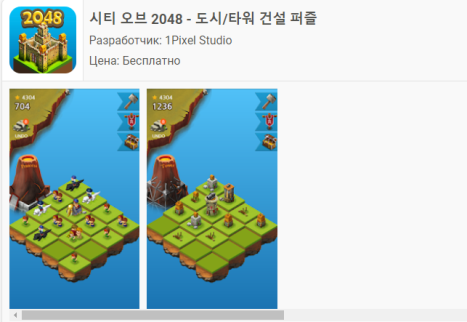
# Обзор аналогов

## 2048 Kitty Cat Island

Эта игра предложит помочь котикам, застрявшим на острове и пытающимся наловить рыбы. Именно поэтому все элементы на поле — это морские существа: начиная от креветок и заканчивая хищными млекопитающими. Помимо такой рыбалки, со временем вам придётся ещё и остров застраивать, который, кстати, в игре не один. https://cdn.lifehacker.ru/wp-content/uploads/lh-appbox/screenshots/5b53771eeb35929db25773b2f8b22705.jpg

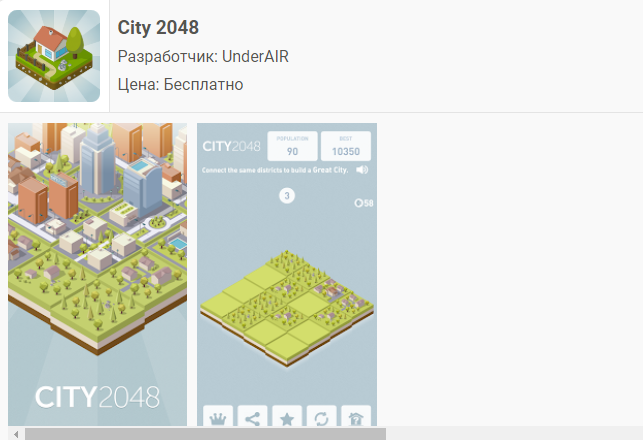
## City of 2048

Наборы элементов в этой головоломке пользователь изначально может выбрать. Это могут быть башни, замки, принцессы, деревья и сооружения эльфов. Вне зависимости от выбора смысл игры не меняется. По каждому из комплектов доступны отдельные таблицы лучших игроков.



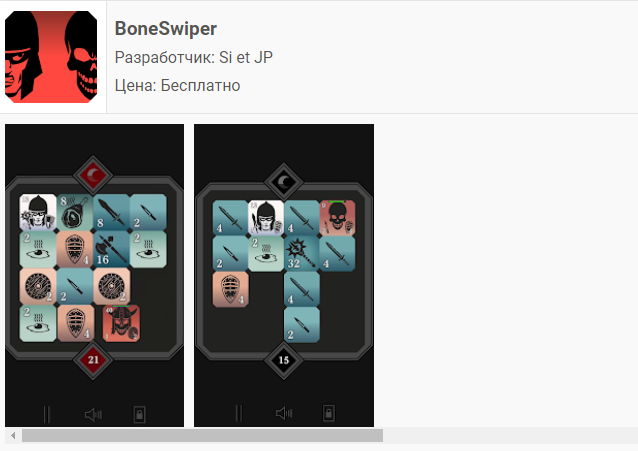
## City 2048

Ещё одна игра на развитие цивилизации. В ней необходимо постоянно увеличивать популяцию людей и возводить как можно больше различных построек. Ограничений никаких нет, играть можно очень и очень долго. Способствует этому и возможность периодически перемешивать все элементы на поле, что порой спасает из безвыходной ситуации.

 https://cdn.lifehacker.ru/wp-content/uploads/lh-appbox/screenshots/5b53771eeb35929db25773b2f8b22705.jpg

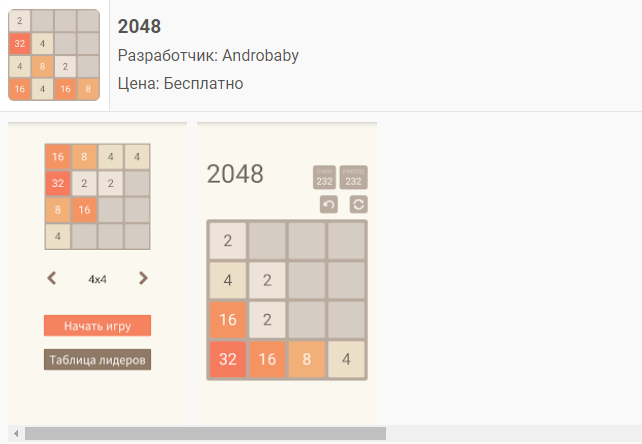
## BoneSwiper

Такая вариация «2048» подойдёт любителям по-настоящему сложных головоломок. В BoneSwiper игровыми элементами являются мечи, щиты, микстуры и провиант для вашего героя, который также размещён на игровом поле. Его совмещение с такими ячейками повышает показатели атаки и защиты, необходимые для сражения с появляющимися монстрами. Они, в свою очередь, тоже используют ячейки снаряжения, поэтому каждый ваш ход может помочь не только герою, но и противнику.



## 2048

В заключение напомним, как выглядит классическая «2048». Это наиболее популярный её вариант, в котором, помимо традиционного поля 4 × 4, можно играть с сеткой 5 × 5, 6 × 6 и даже 8 × 8. Игра продолжается даже после получения числа «2048». Есть таблица рекордсменов и возможность отменить последний ход.

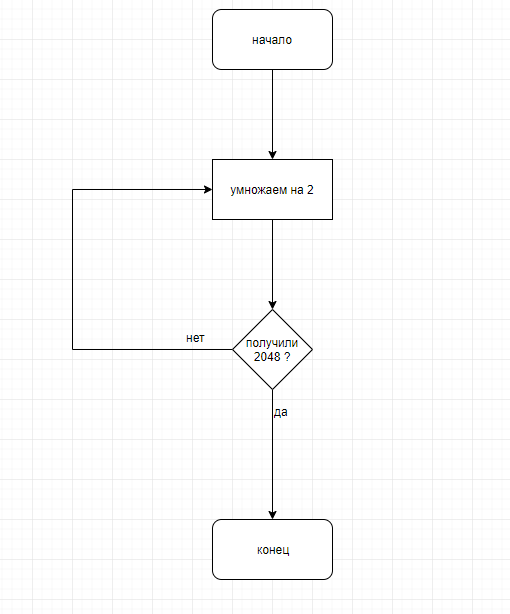


# Постановка задачи

1. Анализировать проектную и техническую документацию на уровне взаимодействия компонент программного обеспечения.
2. Проектирование и разработка ПО.
3. Основы разработки программного обеспечения.
4. Процесс и методология разработки ПО.
5. Участники процесса разработки ПО.

# Описание алгоритма работы программы

1. Отрисовать игровое поле.
2. Умножаем числа на 2.
3. Проверить дошли ли вы до 2048.
4. Если да – выиграли , иначе - вернуться к шагу 2.



# Исходный код программы

**program** g2048;

**uses**

GraphABC, ABCObjects, ABCButtons, timers, controlUtils;

**const**

/// размер поля

n = 4;

///размер фишки

tile = 100;

/// зазор между фишками

zz = 5;

/// отступ от левого и правого краев

x0 = 5;

/// отступ от верхнего и нижнего краев

y0 = 5;

///Степень округления углов у фигурок

roundRate = 5;

///заголовок окна

windowTitle = '2048 Game!';

//Настройки цветов

///фоновый цвет!

backgroundColor = RGB(187, 173, 160);

///фон пустой клетки

emptySquareColor = RGB(205, 193, 180);

///интервал таймера анимационного

animTimerInterval = 1;

///на сколько менять размер клеточки при анимации

animDelta = 6;

**var**

///ходил ли игрок? (это нужно будет для того, чтобы если ход не совершён, не создавать новую клеточку в таком случае)

moved: boolean := false;

///анимационный таймер

animationTimer: Timer;

///экземпляр класса для работы с диалогами

dialogs: TDialogs;

**type**

///возможные направления движения

TDirection = (tdUp, tdDown, tdLeft, tdRight);

///наш класс на основе SquareABC, чтобы хранить тут дополнительные переменные

TSquare = **class**(RoundRectABC)

///значение клеточки

value: integer = 0;

changed: boolean = false; //если произошло изменение значения (используется для анимации)

///фукция для установки значения клеточки, заодно нужных цветов

**procedure** setNumber(number: integer);

**end**;

///запись для хранения настроек цветов

TSquareColor = **record**

bColor, FontColor: color;

**end**;

**var**

gField: **array** [1..n, 1..n] **of** TSquare; //наше поле состоит из наших доделанных SquareABC!

///счёт, пожалуйста!

score: integer = 0;

{\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*}

//добавить очков

**procedure** addScore(add\_score: integer);

**begin**

score := score + add\_score;

moved := true; //раз очки прибавились, значит был совершён ход

**end**;

//обнулить количество очков

**procedure** clearScore();

**begin**

score := 0;

**end**;

//тут можно вывести куда-то количество очков

**procedure** updateScore();

**begin**

window.Title := windowTitle + '; Score: ' + IntToStr(score);

**end**;

//получить случайное начальное число (двойку или четвёрку)

**function** getRandomNumber(): integer;

**begin**

result := Random(100) > 85 ? 4 : 2; //делаем большую вероятность выпадения именно двойки

**end**;

//вспомогательная функция для создания записи TSquareColor из фонового цвета и цвета шрифта

**function** createColor(bColor, FontColor: Color): TSquareColor;

**begin**

result.bColor := bColor;

result.FontColor := FontColor;

**end**;

**procedure** clearSquare(**var** square: TSquare);

**begin**

square.Color := emptySquareColor;

square.Text := '';

square.value := 0;

**end**;

//получить клетку по позиции её

**function** getSquareByPos(x, y: integer): TSquare;

**begin**

result := gField[x, y];

**end**;

//получить цвет в зависимости от цифры

**function** getSquareColorByNumber(num: integer): TSquareColor;

**begin**

**case** num **of**

2: result := createColor(rgb(238, 228, 218), rgb(119, 110, 101));

4: result := createColor(rgb(237, 224, 200), rgb(119, 110, 101));

8: result := createColor(rgb(242, 177, 121), rgb(249, 246, 242));

16: result := createColor(rgb(245, 149, 99), rgb(249, 246, 242));

32: result := createColor(rgb(246, 124, 95), rgb(249, 246, 242));

64: result := createColor(rgb(246, 94, 59), rgb(249, 246, 242));

128: result := createColor(rgb(237, 207, 114), rgb(249, 246, 242));

256: result := createColor(rgb(237, 204, 97), rgb(249, 246, 242));

512: result := createColor(rgb(237, 200, 80), rgb(249, 246, 242));

1024: result := createColor(rgb(237, 197, 63), rgb(249, 246, 242));

2048: result := createColor(rgb(237, 194, 46), rgb(249, 246, 242));

4096: result := createColor(rgb(60, 58, 50), rgb(249, 246, 242));

**else** result := createColor(rgb(238, 228, 218), rgb(119, 110, 101));

**end**;

**end**;

**procedure** animateSquare(sq: TSquare; bigger: boolean);

**begin**

sq.changed := true;

**if** (bigger) **then**

**begin**

sq.left -= animDelta;

sq.Top -= animDelta;

sq.Width += animDelta;

sq.Height += animDelta;

**end else**

**begin**

sq.left += animDelta;

sq.Top += animDelta;

sq.Width -= animDelta;

sq.Height -= animDelta;

**end**;

**end**;

**procedure** TSquare.setNumber(number: integer);

**var**

sc: TSquareColor;

**begin**

value := number;

sc := getSquareColorByNumber(value); //получаем цвет клеточки и цвет шрифта в зависимости от номера

Color := sc.bColor; //устанавливаем цвет клеточки

FontColor := sc.FontColor; //устанавливаем цвет текста

Text := IntToStr(value); //пишем цифру

animateSquare(self, value > 2); //анимация клеточки в зависимости от значения

**end**;

//заполнить клеточку начальными пустыми значениями и параметрами

**procedure** createSquare(**var** square: TSquare);

**begin**

square.Bordered := false;

square.TextScale := 0.7; //размер текста?

square.value := 0;

**end**;

// создаём массив квадратиков

**procedure** CreateFields;

**var**

sq: TSquare;

**begin**

**for var** x := 1 **to** n **do**

**for var** y := 1 **to** n **do**

**begin**

sq := **new** TSquare(x0 + (x - 1) \* (tile + zz), y0 + (y - 1) \* (tile + zz), tile, tile, roundRate, emptySquareColor); //создаём клетку с указанными размерами и координатами и цветом

createSquare(sq); //заполним клетку начальными значениями

gField[y, x] := sq; //заносим экземпляр клетки в массив

**end**;

**end**;

**procedure** clearAllFields;//очистить все клетки!!!!!

**begin**

**for var** x := 1 **to** n **do**

**for var** y := 1 **to** n **do**

clearSquare(gField[x, y]);

**end**;

//проверяет, можно ли ходить куда-то

**function** canMove(): boolean;

**var**

field: TSquare;

**begin**

result := false;

//сначала проверка чисел по вертикалям

**for var** x := 1 **to** n **do**

**begin**

**for var** y := 2 **to** n **do**

**begin**

field := gField[x, y];

//если есть хоть одна пустая клетка, то игрок уж точно может ходить )

**if** (field.value = 0) **then**

**begin**

result := true;

**exit**;

**end**;

//Если можно сложить хоть одну пару клеток

**if** (field.value = gField[x, y - 1].value) **then**

**begin**

result := true; //то значит можно ходить!

**exit**; //И нужно выйти, ведь дальше незачем цикл гонять

**end**;

**end**;

**end**;

//а теперь проверка чисел по горизонталям

**for var** y := 1 **to** n **do**

**begin**

**for var** x := 2 **to** n **do**

**begin**

field := gField[x, y];

//снова проверка на пустую клетку

**if** (field.value = 0) **then**

**begin**

result := true;

**exit**;

**end**;

//снова проверка, можно ли сложить два числа по горизонтали

**if** (field.value = gField[x - 1, y].value) **then**

**begin**

result := true;

**exit**;

**end**;

**end**;

**end**;

**end**;

//получить случайную пустую клетку

**function** getRandomEmptyField(): TSquare;

**var**

positions: **array**[0..n \* n] **of** Point; //массив, куда будут складываться позиции свободных клеток

available\_count: integer = 0;

**begin**

result := nil; //по умолчанию результат будет НИЧЕГО

**for var** x := 1 **to** n **do**

**for var** y := 1 **to** n **do**

**begin**

**if** (gField[x, y].value = 0) **then** //если клетка свободна, то нужно добавить её в массив

**begin**

//сохраняем координаты очередной найденой пустой клетки

positions[available\_count].x := x;

positions[available\_count].y := y;

inc(available\_count); //увеличиваем счётчик количества свободных клеточек

**end**;

**end**;

**if** (available\_count > 0) **then** //если есть свободные клетки,

**begin**

**var** pos := positions[Random(available\_count)]; //то выбираем одну из них случайным образом

result := gField[pos.x, pos.y]; //и выдаём её в результат

**end**;

**end**;

//есть ли ещё пустые клетки?

**function** emptyFieldsExists(): boolean;

**begin**

result := false;

**for var** x := 1 **to** n **do**

**for var** y := 1 **to** n **do**

**begin**

**if** gField[x, y].value = 0 **then** //да, есть хоть одна пустая клетка!

**begin**

result := true;

**break**; //незачем дальше цикл гонять, раз есть хоть одна свободная клетка

**end**;

**end**;

**end**;

//создать случайное число (2 или 4) в случайный пустой квадратик

**procedure** putRandomSquare();

**var**

sq: TSquare;

**begin**

sq := getRandomEmptyField(); //получаем случайный пустой квадрат

**if** (sq = nil) **then** //Если nil, значит нет пустых клеток!

**begin**

//сначала проверить, есть ли возможность ходить куда-то

**if** (canMove()) **then**

**begin**

**exit**;

**end**;

//Иначе, раз нельзя уже ходить, то типа игра окончена и всё такое.

//согласен, что не очень правильно, что этот код находится в этой процедуре, но да ладно (

dialogs.showInfo('Игра окончена! Счёт: ' + IntToStr(score));

**exit**;

**end**;

sq.setNumber(getRandomNumber()); //устанавливаем случайный номер (2 или 4) клеточке

**end**;

//поменять местами две клеточки между собой

**procedure** replaceSquares(square1, square2: TSquare);

**var**

c, f: Color;

t: String;

v: Integer;

**begin**

//запомним параметры первой клетки

c := square1.Color;

f := square1.FontColor;

t := square1.Text;

v := square1.value;

//поменяем параметры первой клетки на параметры второй

square1.Color := square2.Color;

square1.Text := square2.Text;

square1.value := square2.value;

square1.FontColor := square2.FontColor;

//поменяем параметры второй клетки на заранее сохранённые параметры первой

square2.value := v;

square2.Text := t;

square2.Color := c;

square2.FontColor := f;

//system.Console.Writeline('MOVED! ' + t);

moved := true; //установим флаг, что ход был совершён!

**end**;

{\*\*}

**procedure** moveUp();

**var**

sq, new\_sq: TSquare;

**begin**

**for var** x := 1 **to** n **do**

**for var** y := 2 **to** n **do**

**begin**

sq := gField[y, x];

//sleep(100);

//sq.Color := clBlue;

**if** (sq.value <> 0) **then**

**begin**

**for var** i := y - 1 **downto** 1 **do** //пробежимся циклом вверх по Y до начала.

**begin**

new\_sq := gField[i, x];

**if** (new\_sq.value <> 0) **then** //если вдруг встретилась не пустая клетка, надо проверить, вдруг она складываемая!

**begin**

**if** (new\_sq.value <> sq.value) **then** //если значения разные, то

**begin**

replaceSquares(sq, getSquareByPos(i + 1, x)); //передвинуть клеточку рядом с той, что выше

**break**; //и просто АСТАНАВИТЕСЬ

**end else** //если совпало значение, то объединить их!

**begin**

new\_sq.setNumber(new\_sq.value \* 2); //установить новое значение для клеточки

addScore(new\_sq.value); //добавить это значение к количеству очков

clearSquare(sq); //очистить ту клеточку, которая сложилась с первой

**break**;

**end**;

**end**;

**if** (i = 1) **then**

replaceSquares(sq, new\_sq); //дошли до начала, значит, просто поменять местами клетки)

**end**;

**end**;

**end**;

**end**;

**procedure** moveDown();

**var**

sq, new\_sq: TSquare;

**begin**

**for var** x := 1 **to** n **do**

**for var** y := n - 1 **downto** 1 **do**

**begin**

sq := gField[y, x];

**if** (sq.value <> 0) **then**

**begin**

**for var** i := y + 1 **to** n **do** //пробежимся циклом вниз до конца по этому же столбцу Y

**begin**

new\_sq := gField[i, x];

**if** (new\_sq.value <> 0) **then** //если вдруг встретилась не пустая клетка, надо проверить, вдруг она складываемая!

**begin**

**if** (new\_sq.value <> sq.value) **then** //если значения разные, то

**begin**

replaceSquares(sq, getSquareByPos(i - 1, x)); //передвинуть клеточку к той, которая имеет значение иное

**break**; //и просто АСТАНАВИТЕСЬ

**end else** //если совпало значение, то объединить их!

**begin**

new\_sq.setNumber(new\_sq.value \* 2); //установить новое значение для клеточки

addScore(new\_sq.value); //добавить это значение к количеству очков

clearSquare(sq); //очистить ту клеточку, которая сложилась с первой

**break**;

**end**;

**end**;

**if** (i = n) **then**

replaceSquares(sq, new\_sq); //дошли до начала, значит, просто поменять местами клетки)

**end**;

**end**;

**end**;

**end**;

**procedure** moveLeft();

**var**

sq, new\_sq: TSquare;

**begin**

**for var** x := 2 **to** n **do**

**for var** y := 1 **to** n **do**

**begin**

sq := gField[y, x];

**if** (sq.value <> 0) **then**

**begin**

**for var** i := x - 1 **downto** 1 **do** //пробежимся циклом влево по текущей строке X

**begin**

new\_sq := gField[y, i];

**if** (new\_sq.value <> 0) **then** //если вдруг встретилась не пустая клетка, надо проверить, вдруг она складываемая!

**begin**

**if** (new\_sq.value <> sq.value) **then** //если значения разные, то

**begin**

replaceSquares(sq, getSquareByPos(y, i + 1)); //передвинуть клеточку к той, которая имеет значение иное

**break**; //и просто АСТАНАВИТЕСЬ

**end else** //если совпало значение, то объединить их!

**begin**

new\_sq.setNumber(new\_sq.value \* 2); //установить новое значение для клеточки

addScore(new\_sq.value); //добавить это значение к количеству очков

clearSquare(sq); //очистить ту клеточку, которая сложилась с первой

**break**;

**end**;

**end**;

**if** (i = 1) **then**

replaceSquares(sq, new\_sq); //дошли до начала, значит, просто поменять местами клетки)

**end**;

**end**;

**end**;

**end**;

**procedure** moveRight();

**var**

sq, new\_sq: TSquare;

**begin**

**for var** x := n - 1 **downto** 1 **do**

**for var** y := 1 **to** n **do**

**begin**

sq := gField[y, x];

**if** (sq.value <> 0) **then**

**begin**

**for var** i := x + 1 **to** n **do** //пробежимся циклом влево по текущей строке X

**begin**

new\_sq := gField[y, i];

**if** (new\_sq.value <> 0) **then** //если вдруг встретилась не пустая клетка, надо проверить, вдруг она складываемая!

**begin**

**if** (new\_sq.value <> sq.value) **then** //если значения разные, то

**begin**

replaceSquares(sq, getSquareByPos(y, i - 1)); //передвинуть клеточку к той, которая имеет значение иное

**break**; //и просто АСТАНАВИТЕСЬ

**end else** //если совпало значение, то объединить их!

**begin**

new\_sq.setNumber(new\_sq.value \* 2); //установить новое значение для клеточки

addScore(new\_sq.value); //добавить это значение к количеству очков

clearSquare(sq); //очистить ту клеточку, которая сложилась с первой

**break**;

**end**;

**end**;

**if** (i = n) **then**

replaceSquares(sq, new\_sq); //дошли до начала, значит, просто поменять местами клетки)

**end**;

**end**;

**end**;

**end**;

**procedure** move(direction: TDirection);

**begin**

moved := false; //сбросим флаг сделанного хода

//выполнить ход в нужную сторону

**case** direction **of**

tdUP: moveUp();

tdDown: moveDown();

tdLeft: moveLeft();

tdRight: moveRight();

**end**;

//обновить в заголовке окна количество очков после очередного хода

updateScore();

//Если ход был освершён, то добавить ещё клеточку

**if** moved **then**

putRandomSquare();

**end**;

**procedure** newGame();

**begin**

clearAllFields(); //При нажатии на Esc можно сделать что-то вроде меню и т.д.

clearScore();

putRandomSquare();

putRandomSquare();

**end**;

**procedure** KeyDown(Key: integer);

**begin**

//если открыто какое-то диалоговое окно!

**if** dialogs.modalVisible **then**

**begin**

**case** Key **of**

VK\_RETURN: dialogs.pressOK;

VK\_ESCAPE: dialogs.closeLastDialog();

**end**;

**exit**;

**end**;

//если нет открытого диалогового окна

**case** Key **of**

VK\_UP: move(tdUp);

VK\_DOWN: move(tdDown);

VK\_LEFT: move(tdLeft);

VK\_RIGHT: move(tdRight);

VK\_Escape: dialogs.showConfirm('Вы действительно хотите начать заново?', newGame);

**end**;

**end**;

//анимационный таймер

**procedure** animTimerProc;

**var**

sq: TSquare;

**begin**

//плавное изменение размеров обратно до изначального

**for var** x := 1 **to** n **do**

**for var** y := 1 **to** n **do**

**begin**

sq := gField[x, y];

**if** (sq.changed) **then**

**begin**

**if** (sq.Width > tile) **then** //если размер больше, надо уменьшать

**begin**

sq.left += 1;

sq.Top += 1;

sq.Width -= 1;

sq.Height -= 1;

**end else** //иначе увеличивать наоборот )

**begin**

sq.left -= 1;

sq.Top -= 1;

sq.Width += 1;

sq.Height += 1;

**end**;

**end**;

**if** (sq.Width = tile) **then** //если размер стал изначальным,

**begin**

sq.changed := false; //то пометить клетку, как нормальную, не изменённую, чтобы таймер больше не трогал её пока

**continue**;

**end**;

**end**;

**end**;

//обработчики событий для кнопок меню

**procedure** onNewGameClick;

**begin**

newGame();

**end**;

**var**

cu: TControlUtils;

**begin**

Randomize(); //инициализация генератора случайных чисел

Window.Title := windowTitle; //заголовок окошка

OnKeyDown := KeyDown; //зададим наш обработчик нажатия кнопок

SetWindowSize(2 \* x0 + (tile + zz) \* n - zz, 2 \* y0 + (tile + zz) \* n - zz); //задаём размер окошка, учитывая размеры всех клеток и отступов

Window.IsFixedSize := True; //чтобы нельзя было менять размер окошка

Window.Clear(backgroundColor); //"очистим" окно фоновым цветом

Window.CenterOnScreen(); //окно пусть будет по центру экрана

CreateFields(); //создадим поля-полюшки

//создаём и запускаем таймер, отвечающий за анимации

animationTimer := **new** Timer(animTimerInterval, animTimerProc);

animationTimer.start;

//создаём объект для работы с диалоговыми окошками

dialogs := **new** TDialogs;

{

cu := new TControlUtils();

cu.addMenuButton('Новая игра', onNewGameClick);

cu.addMenuButton('LOL?!', onTestGameClick);

cu.setToCenter();

}

//начинается всё с двух клеток

putRandomSquare();

putRandomSquare();

**end**.

# Тестирование программы



# Системные требования

Приложение совместимо с Windows 7,8,10 32-bit и 64-bit.

Минимальный размер внешнего носителя: 2 Мб

Максимальный размер внешнего носителя: неограниченно

Размер игры: 2 Мб

ОЗУ: 10 Мб

# Руководство по установке

Открыть скаченный файл, запустить 2048.exe.