Петрозаводский государственный университет
Институт математики и информационных технологий
Кафедра информатики и математического обеспечения

09.03.04 – Программная инженерия

Профиль направления подготовки бакалавриата

Системное и прикладное программное обеспечение

Отчет о проектной работе по курсу «Основы информатики и программирования»

Игра «Blockudoku»

Выполнил:
студент 2 курса группы 22207

Д. А. Костин _____
подпись

Место прохождения практики:
Кафедра информатики и математического обеспечения

Период прохождения практики:

17.01.2021–17.01.2021

Руководитель:
А. В. Бородин, старший преподаватель

_____
подпись

Итоговая оценка:

_____
оценка

Петрозаводск — 2021

Содержание

Введение

Цель практики: Создать приложение (игру) для мобильных устройств с графическим интерфейсом.

Задачи практики: Получить опыт в создании мобильных приложений, составить документацию, описывающую работу приложения

Приложение для мобильных, позиционируемая как игра, - «Blockudoku», - проект, созданный на языке Java в среде мобильной разработки Android Studio. Данный отчёт содержит подробное описание работы приложения с примерами кода, описанием алгоритмов и т.п.

1 Описание сути игры

Данная игра называется «Blockudoku», что означает объединение двух слов: «блок» и «судоку».

Интерфейс программы представляет собой клеточное поле 9 на 9. Игроку предоставляется выбор 3 фигур - фигуры составлены из квадратов, выбрав фигуру игрок нажимает на клетку поля, тем самым ставя данную фигуру в клеточное поле.

Если фигура успешно вмещается в выбранную область поля, то пользователь снова выбирает одну из 3 новых сгенерированных фигур и ставит её в поле. Задача игрока - заполнить квадрат 3 на 3 (таких квадратов на поле 9 штук), после заполнения всех клеток квадрата 3 на 3, он исчезает, а игроку добавляется счёт.

Игрок может проиграть, если 3 фигуры не помещаются не в какие свободные ячейки. При должной сноровке игрока, игра может длиться вечно.

2 Описание модулей

2.1 Модуль MainActvity

Данный модуль по большей части почти не отличается от автоматически сгенерированного модуля MainActivity. Функция setContentView() - функция отображения экрана активности теперь принимает объект SurfaceView - созданный класс GameView, который

будет производить отображение интерфейса игры, а также выполнять взаимодействие с игрой.

2.2 Модуль GameView

Данный модуль - это класс, наследованный от SurfaceView, предоставляет некоторую область для рисования, которую будет отрисовывать другой поток, определённый в модуле GameThread.

Работа с полотном для рисования осуществляется не напрямую через созданный класс, а с помощью объекта SurfaceHolder. Получить его можно вызовом метода getHolder(). Именно этот объект будет предоставлять нам canvas для отрисовки.

В конструкторе класса получаем объект SurfaceHolder и с помощью метода addCallback() указываем что хотим получать соответствующие обратные вызовы.

Листинг 1 – Класс GameView. Интерфейс SurfaceHolder.Callback

```
SurfaceHolder holder = getHolder();
holder.addCallback(new SurfaceHolder.Callback() {


    @Override
    public void surfaceCreated(@NonNull SurfaceHolder surfaceHolder) {
        gameThread.setRunning((true));
        gameThread.start();
    }
```

Класс содержит метод surfaceCreated - который создаёт область для рисования, а также запускает поток для отрисовки.

Заранее введём листинги классов GameLogic и Destinations:

Листинг 2 – Классы GameLogic и Destinations

```
    public static class GameLogic {
        int id;
        int colored = 0;
        int left;
        int right;
        int top;
        int bottom;
```

```
    }

    public class Destinations {
        int size;
        int destinations[];
    }
```

Класс потока запускает функцию отрисовки canvas - onDraw(), определённую в классе GameView.

В данной функции на основе класса GameLogic, который содержит поля для хранения координат ячеек на поле (а именно координаты левой, правой, верхней и нижней границ прямоугольника), значение того, что ячейка окрашена. Так вот, в функции onDraw() используются данные о ячейках из двумерного массива 9 на 9 класса GameLogic, для каждой ячейки происходит отрисовка с помощью функции canvas.drawRect().

Листинг 3 – Часть функции onDraw()

```
for (int i = 0; i < 9; i++) {
    for (int j = 0; j < 9; j++) {
        if (cells[i][j].colored == 1) {
            canvas.drawRect(cells[i][j].left, cells[i][j].top,
            cells[i][j].right, cells[i][j].bottom, cellsColored);
        } else {
            if ((i < 3 || i >= 6) && (j < 3 || j >= 6)
            || (i >= 3 && i < 6) && (j >= 3 && j < 6)) {
                canvas.drawRect(cells[i][j].left, cells[i][j].top,
                cells[i][j].right, cells[i][j].bottom, cellsColor2);
                canvas.drawRect(cells[i][j].left, cells[i][j].top,
                cells[i][j].right, cells[i][j].bottom, cellsColor);
            } else
                canvas.drawRect(cells[i][j].left, cells[i][j].top,
                cells[i][j].right, cells[i][j].bottom, cellsColor);
        }
    }
}
```

Как видно из листинга выше, цикл проходит по всем элементам двухмерного массива класса GameLogic cells[9][9], для каждого элемента вызывается функция drawRect, принимающая в себя координаты сторон прямоугольника. Наличие в цикле if обусловлено тем, что если cells имеет поле colored = 1, то клетку окрашиваем в синий цвет, другой if - некоторые поля 3 на 3 имеют различный цвет, для лучшего восприятия зон на поле, которые нужно заполнить для начисления очков.

Помимо отрисовки клеток, происходит отрисовка всех остальных элементов интерфейса, а именно: кнопка перезапуска, вывод текущего счёта, вывод выбора элементов для размещения на поле. Пожалуй, самое интересное это генерация элементов для размещения.

Для того, чтобы реагировать на нажатия на экран, была создана функция onTouchEvent(), которая:

- Получает координаты нажатия
- Проверят попали ли координаты в конкретную ячейку
- Если попадают, то происходит проверка возможности поставить выбранную фигуру
- Если фигура помещается, то происходит перебор всех подходящих под фигуру cells, для них cells.colored = 1.
- Вызывается функция проверки заполненность квадратов 3 на 3
- Вызывается функция генерации новых фигур

Рассмотрим по частям данную функцию.

Листинг 4 – Часть функции onTouchEvent()

```
case MotionEvent.ACTION_DOWN:
                x1 = event.getX();
                y1 = event.getY();
                for (int i = 0; i < 9; i++) {
                    for (int j = 0; j < 9; j++) {
                        if (inside((int)x1, (int)y1, cells[i][j])){
                            if (suitable(inButtonsDest[chose],i,j)){
                            ...
```

Из предыдущего листинга видно, что сначала получаются координаты места нажатия, затем происходит проход по всем элементам cells, вызываются функции inside - функция проверки того, что нажатие произошло в конкретной ячейке, и suitable - функция, проверяющая, что выбранная фигура построится из данной ячейки в последующие.

Листинг 5 – Часть функции onTouchEvent()

```
int i1 = i, j1 = j;
    for (int k = 0; k < inButtonsDest[chose].size; k++){
        if (k == 0){
           cells[i1][j1].colored = 1;
           continue;
        }
        if (inButtonsDest[chose].destinations[k] == 1){
            j1--;
            cells[i1][j1].colored = 1;
        }
        else {
        i1--;
        cells[i1][j1].colored = 1;
        }
}
```

Предыдущий листинг показывает, как происходит проход по ячейкам, которые занимаются поставленной на поле фигурой: существует класс Destinations, который хранит в себе массив int (k элемент массива - есть направление, куда размещается k клетка относительно предыдущей), элементы принимают значния 1 или 2 - 1 означает, что фигура растёт вверх, 2 - что влево. Если фигура растёт вверх, то меняется i индекс двухмерного массива cells[i][j], а именно он уменьшается, если 2 - то уменьшается j. После этого cells[i][j].colored = 1 - что означает - ячейка окрашена.

Перейдём к функции генерации новых фигур. Данная функция generateBlocks() принимает в себя index - то есть номер, для которого будет генерироваться новая фигура. Существует 3 ячейки для отображения сгенерированной фигуры. Получив номер, функция создаёт новый объект класса Destinations, который будет содержать массив направлений. Затем, генерируется псевдослучайное число - размер фигуры - то есть число блоков, из которых будет состоять фигура. Для каждого элемента массива генерируется число,

равно либо 1, либо 2, означающее направление построения фигуры. На основе данного класса происходит проверка, что фигура помещается в клетку нажатия, а также строится уменьшенное изображение фигуры для визуализации выбора.

Листинг 6 – generateBlocks()

```
public void generateBlocks(int index){
        int max = 4;
        int min = 1;
        max -= min;
        inButtonsDest[index] = new Destinations();
        inButtonsDest[index].size =(int)(Math.random()*++max)+min;
        inButtonsDest[index].destinations = new int[inButtonsDest[index].size];
        for (int j = 0; j < inButtonsDest[index].size; j++){
            inButtonsDest[index].destinations[j] = generateDestination();
        }
    }
```

Как уже было упомянуто, присутствует отображение того, какую фигуру игрок может выбрать из 3 сгенерированных. Данные фигуры представляют собой уменьшенные копии фигур, которые можно поставить. Они получаются в результате построения класса GameLogic на основе класса Destinations. То есть, у панели выбора вариантов 3 фигур есть свои координаты, в каждой ячейке выбора рисуется GameLogic inButton[3]

Он строится так: берётся направление из Destinations, в inButton каждая клетка строится в соответствии с направлением. Причём координаты клетки расчитываются для каждой ячейки выбора в зависимости от расположения ячейки. В результате, в каждой ячейке выбора строится уменьшенная копия фигуры.

Простыми словами говоря, класс GameLogic для визуализации выбора фигур строится в зависимости от сгенерированных направлений в классе Destinations. Ниже приведён листинг построения класса GameLogic.

Листинг 7 – Функция blocks() - визуализация фигуры

```
public void block(int index){
    generateBlocks(index);
    inButtons[index] = new GameLogic[inButtonsDest[index].size];
    int w=(1075-50)/3;
```

```
    int scewX = 0;
    for (int i = 0; i < inButtonsDest[index].size; i++){
        inButtons[index][i] = new GameLogic();
        if (inButtonsDest[index].destinations[i] == 1 || i == 0){
            inButtons[index][i].left = w - 25 +w*index-scewX;
            inButtons[index][i].right = w - 50 - 25 +w*index-scewX;
            inButtons[index][i].top = 1450-50*i+scewX;
            inButtons[index][i].bottom = 1500-i*50+scewX;
        }
        else {
            scewX+=50;
            inButtons[index][i].left = w - 25 +w*index-scewX;
            inButtons[index][i].right = w - 50 - 25 +w*index-scewX;
            inButtons[index][i].top = 1450-50*(i)+scewX;
            inButtons[index][i].bottom = 1500-(i)*50+scewX;
        }
    }
}
```

Остаётся разобрать функцию проверки того, что какой-либо квадрат 3 на 3 на поле был закрашен. Данная функция checkKub() проста: она проходит все элементы каждого куска поля 3 на 3, проверяя, что все cells[i][j] закрашены. Если это так, то все эти элементы cells[i][j].colored = 0.

Листинг 8 – Функция blocks() - визуализация фигуры

```
public void checkKub(){
    for (int i = 0; i < 3; i++){
         for (int j =0; j < 3;j++){
            if (cells[3*i][3*j].colored == 1 &&
                cells[3*i][3*j+1].colored == 1 &&
                cells[3*i][3*j+2].colored == 1 &&
                cells[3*i+1][3*j].colored == 1 &&
                cells[3*i+1][3*j+1].colored == 1 &&
```

```
                cells[3*i+1][3*j+2].colored == 1 &&
                cells[3*i+2][3*j].colored == 1 &&
                cells[3*i+2][3*j+1].colored == 1 &&
                cells[3*i+2][3*j+2].colored == 1)
                {
                cells[3*i][3*j].colored = 0;
                cells[3*i][3*j+1].colored = 0;
                cells[3*i][3*j+2].colored = 0;
                cells[3*i+1][3*j].colored = 0;
                cells[3*i+1][3*j+1].colored = 0;
                cells[3*i+1][3*j+2].colored = 0;
                cells[3*i+2][3*j].colored = 0;
                cells[3*i+2][3*j+1].colored = 0;
                cells[3*i+2][3*j+2].colored = 0;
                score+=9;
            }
        }
    }
}
```

2.3 Модуль GameThread

Логика отрисовки игры такова: создаётся новый поток, который будет выполнять отрисовку картинки.

Листинг 9 – Функция run()

```
@Override
public void run() {
    while (running) {
        Canvas c = null;
        try {
            c = view.getHolder().lockCanvas();
            synchronized (view.getHolder()) {
                view.onDraw(c);
```

```
            }
        } finally {
            if (c != null) {
                view.getHolder().unlockCanvasAndPost(c);
            }
        }
    }
}
```

Данная функция запуска потока создаёт новый объект Canvas, который затем закрывается держателем-holder, запускается функция из класса GameView - onDraw(), в который передаётся пустой canvas, на котором происходит отрисовка, после завершения отрисовки, canvas разблокируется и выводится на экран.

Данный цикл воспроизводится бесконечно, так как поток запускается из класса GameView, в моём приложении задержка между прорисовками не установлена, значит она минимальна - то есть ограничена временем выполнения функции onDraw.

Заключение

Данное мобильное приложение - игра Blockdoku - получилась схожей с её реальным аналогом. Её реализация выполнилась с помощью canvas на языке Java. Для отрисовки использовался второй поток, основные функции взаимодействия с игрой были реализованы в основном потоке.

В проекте были продемонстрированы навыки мобильной разработки, возможности взаимодействия двух потоков, взаимодействия на canvas.