Санкт-Петербургский Политехнический Университет Петра Великого
Институт компьютерных наук и технологий
Кафедра компьютерных систем и программных технологий

Алгоритмы и структуры данных

Отчет по курсовой работе
Игра змейка

Работу выполнил:
Курякин Д. А.
Группа: 23501/4
Преподаватель:
Вылегжанина К.Д.

Санкт-Петербург
2017

Содержание

1 Игра Змейка

1.1 Игровые принадлежности

Змейка — компьютерная игра, возникшая в середине или в конце 1970-х.

Игрок управляет длинным, тонким существом, напоминающим змею, которое ползает по плоскости ограниченной стенками, собирая еду, избегая столкновения с собственным хвостом и краями игрового поля. Каждый раз, когда змея съедает кусок пищи, она становится длиннее, что постепенно усложняет игру. Игрок управляет направлением движения головы змеи: вверх, вниз, влево, вправо, а хвост змеи движется следом. Игрок не может остановить движение змеи.

1.2 Порядок использования

После начала игры появляется еда(яблоко). Пользователь направлет голову змейки чтобы съесть яблоко. После съедения яблока змейка увеличивается на одно деление. Затем яблоко появляется в любом месте на карте.

2 Проектирование приложения

2.1 Концепция приложения

В ходе проектирования было разработана концепция продукта. Созданное приложение должно предполагать возможность: управления змейкой с помощью свейпа по экрану, появления яблок на карте, съедения яблок, увеличения размера змейки.

2.2 Минимально работоспособный продукт

Минимальном работоспособным продуктом было признано приложение, позволяющие производить игру(управлять змейкой, съедать яблоко, увеличивать длину после съедания).

2.3 Прецеденты использования

На основе разработанной концепции была составлена UML диаграмма прецедентов использования (рис.1).

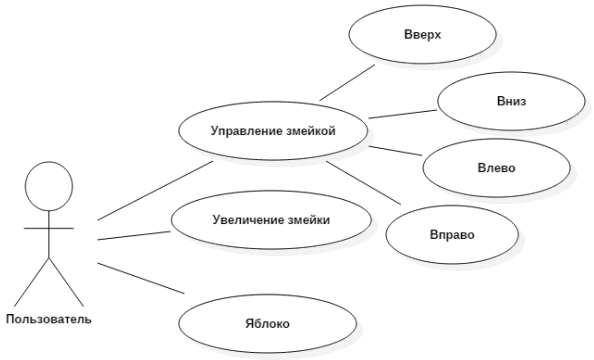

Рис. 1: Диаграмма прецедентов использования

2.4 Основные компоненты приложения

На основе анализа концепции и выделенных прецедентов использования было принято решение выделить два основных компонента, которые будут входить в состав продукта:

1. Библиотека

   Включает в себя игровую модель и реализует игровые механизмы. В ядре должно быть обеспечено регулярное обновление модели в ответ на действие пользователя. Кроме того, должна быть реализована обработка исключительных ситуаций, включающие в себя ошибки игрока, попытки выполнить запрещенные действия и прочее.

2. Графическое приложение

   Графически визуализирует игровую модель, предоставляет пользователю графический интерфейс для взаимодействия с ней и выполнения остальный действий предусмотренных в реализации библиотеки.

2.5 Используемые инструменты

Разработка в основном велась с использованием средств стандартной библиотеки Java в среде Android Studio. Для создания графического интерфейса применялась библиотеки из Android Studio.

2.5.1 Android Studio

Android Studio — это интегрированная среда разработки (IDE) для работы с платформой Android.

2.6 Выводы

Таким образом, была разработана концепция приложения, что позволило определить внешний вид продукта и выделить его основные компоненты.

3 Реализация приложения

3.1 Среда разработки

- Операционная система: Windows 10
- Интегрирование среда разработки: Android Studio 2.3
- Версия SDK: 26.0.2

3.2 Реализация основных компонентов приложения

3.2.1 Библиотека Core

Для реализации всех запланированных функциональностей было принято решение, создать два класса:

1. **Класс Coordinate** Данный класс представляет возможность управлять координатами обьекта. Конструктор класса **Coordinate** получает на вход координаты объекта **X** и **Y**. С помошью методов **setX** и **setY** предоставляется возможность изменять координаты объектов. Методы **getX** и **getY** предоставляют возможность возвращать координаты обьекта. Метод **equals** возвращает значение boolean и представляет возможность ставнивать обьекты друг с другом.

2. **Класс GameEngine** Данный класс отвечает работу игры в целом. Метод **initGame** отвечает за инициализацию всех объектов игры. Класс **addSnake** добавляет змейку, **addWalls** добавляет стены, **addApples** добавляет яблоки. Метод **Update** отвечает за обновление всех объектов игры. Используя класс **UpdateSnake** и перечисляемый тип Enum **Direction** метод **Update** обновляет змейку: направление (Noth, East, South, West), координаты всех частей змейки. Также с помощью перечисляемого типа Enum **GameState** метод **Update** осуществляет статусы игры (Ready, Running, Lost) Метод **getMap** создаёт карту используя перечисляемый тип Enum **TileType** (Nothing, Wall, SnakeHead, SnakeTail, Apple)

3.2.2 Графический интерфейс

Для создания графического интерфейса использовалась библиотеки Android Studio. В Android Studio было создано Main Activity в котором с помощью класса **SnakeView** осуществляется отрисовка. Также к Main Activity был применён интерфейс **View.OnTouchListener** для считывания касаний экрана.

1. **Класс SnakeView** Класс наследуется от класса **View** и осуществляет отрисовку игры методом **onDraw**.

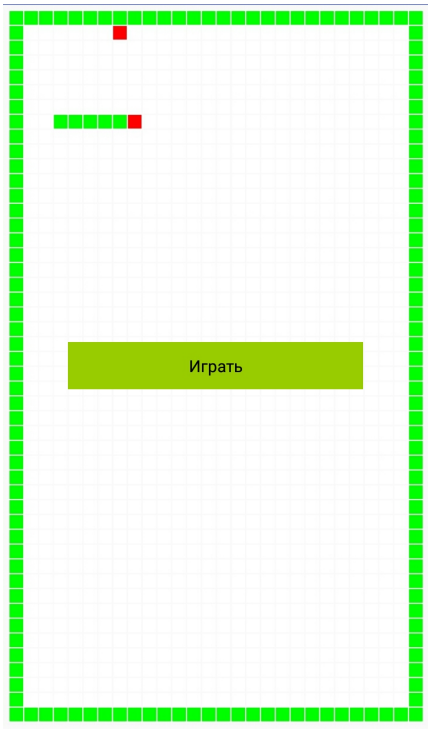

Рис. 2: Меню графического интерфейса

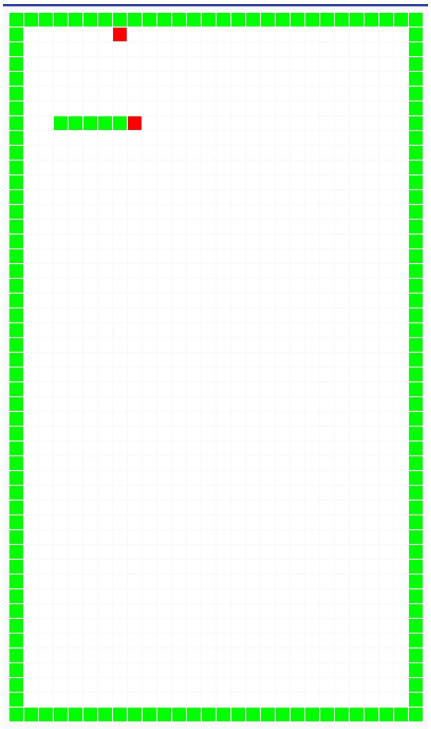

Рис. 3: Игровое окно графического интерфейса

На рисунках 2, 3 изображены основные окна графического интерфейса.

3.3 Теститрование

В ходе разработки проекта регулярно проводилось ручное тестирование.
Тестирование позволило обеспечить работоспособность продукта в ходе всего процесса разработки.

3.4 Демонстрации

Во время создания приложения было проведено 1 демонстрации, на которых группой людей, представляющих собой потенциальных пользоваталей разработываемого приложения.

4 Выводы

В ходе работы были получены навыки необходимые для написания программ на языке программирования Java. Во-первых были изучены библиотеки Android Studio и особенности данной среды разработки. Во-вторых был получен опыт, связанный с процессом разработки программного продукта.

5 Приложение 1

5.1 Листинги

```
1  package ru.kuryakin.snake.engine;
2
3
4  public class Coordinate {
5
6      private int x;
7      private int y;
8
9      public Coordinate(int x, int y){
10         this.x = x;
11         this.y = y;
12     }
13
14     public int getX(){
```

```
15          return x;
16      }
17
18      public void setX(int x){
19          this.x = x;
20      }
21
22      public int getY(){
23          return y;
24      }
25
26      public void setY(int y){
27          this.y = y;
28      }
29
30      @Override
31      public boolean equals(Object o) {
32          if (this == o) return true;
33          if (o == null || getClass() != o.getClass()) return false;
34
35          Coordinate that = (Coordinate) o;
36
37          if (x != that.x) return false;
38          return y == that.y;
39
40      }
41  }
```

```
1  package ru.kuryakin.snake.engine;
2
3  public enum Direction {
4      Noth,
5      East,
6      South,
7      West
8  }
```

```
1  package ru.kuryakin.snake.engine;
2
3
4  import java.util.ArrayList;
5  import java.util.List;
6  import java.util.Random;
7
8  public class GameEngine {
9      public static final int GameWith = 28;
10     public static final int GameHeight = 48;
11
12     private List<Coordinate> walls = new ArrayList<>();
13     private List<Coordinate> snake = new ArrayList<>();
14     private List<Coordinate> apples = new ArrayList<>();
15
16     private Random random = new Random();
17     private  boolean increaseTail = false;
18
19     private Direction currentDirection = Direction.East;
20
21     private GameState currentGameState = GameState.Running;
22
23     private Coordinate getSnakeHead(){
24         return snake.get(0);
25     }
26
27     public GameEngine(){}
28
29     public void initGame(){
30
31         addSnake();
32         addWalls();
33         addApples();
34     }
35
36     public void UpdateDirection(Direction newDirection){
37         if (Math.abs(newDirection.ordinal() - currentDirection.ordinal()) % 2 == 1){
38             currentDirection = newDirection;
```

```
39          }
40      }
41
42      public void Update(){
43          switch (currentDirection) {
44              case Noth:
45                  UpdateSnake(0, -1);
46                  break;
47              case East:
48                  UpdateSnake(1, 0);
49                  break;
50              case South:
51                  UpdateSnake(0, 1);
52                  break;
53              case West:
54                  UpdateSnake(-1, 0);
55                  break;
56          }
57
58          for (Coordinate w: walls) {
59              if (snake.get(0).equals(w)){
60                  currentGameState = GameState.Lost;
61              }
62          }
63
64          for (int i = 1; i < snake.size(); i++) {
65              if ( getSnakeHead().equals(snake.get(i))){
66                  currentGameState = GameState.Lost;
67                  return;
68              }
69          }
70
71          Coordinate appleToRemove = null;
72          for (Coordinate apple: apples) {
73              if (getSnakeHead().equals(apple)){
74                  appleToRemove = apple;
75                  increaseTail = true;
76              }
77          }
78          if (appleToRemove != null){
79              apples.remove(appleToRemove);
80              addApples();
81          }
82      }
83
84      public TileType[][] getMap(){
85          TileType[][] map = new TileType[GameWith][GameHeight];
86
87          for (int x = 0; x < GameWith; x++) {
88              for (int y = 0; y < GameHeight; y++) {
89                  map[x][y] = TileType.Nothing;
90              }
91          }
92
93          for (Coordinate wall: walls){
94              map[wall.getX()][wall.getY()] = TileType.Wall;
95          }
96
97          for (Coordinate s: snake){
98              map[s.getX()][s.getY()] = TileType.Wall;
99          }
100
101         for (Coordinate a: apples){
102             map[a.getX()][a.getY()] = TileType.Apple;
103         }
104
105         map[snake.get(0).getX()][snake.get(0).getY()] = TileType.SnakeHead;
106
107         return map;
108     }
109
110     private void UpdateSnake(int x, int y){
111         int newX = snake.get(snake.size()-1).getX();
112         int newY = snake.get(snake.size()-1).getY();
113
114         for (int i = snake.size() - 1; i > 0; i--) {
```

```
115             snake.get(i).setX(snake.get(i-1).getX());
116             snake.get(i).setY(snake.get(i-1).getY());
117         }
118 
119         if (increaseTail){
120             snake.add(new Coordinate(newX, newY));
121             increaseTail = false;
122         }
123 
124         snake.get(0).setX(snake.get(0).getX() + x);
125         snake.get(0).setY(snake.get(0).getY() + y);
126     }
127 
128     private void addSnake() {
129         snake.clear();
130         snake.add(new Coordinate(7, 7));
131         snake.add(new Coordinate(6, 7));
132         snake.add(new Coordinate(5, 7));
133         snake.add(new Coordinate(4, 7));
134         snake.add(new Coordinate(3, 7));
135         snake.add(new Coordinate(2, 7));
136     }
137 
138     private void addWalls() {
139         for (int x = 0; x < GameWith; x++) {
140             walls.add(new Coordinate(x,0));
141             walls.add(new Coordinate(x,GameHeight-1));
142         }
143 
144         for (int y = 0; y < GameHeight; y++) {
145             walls.add(new Coordinate(0,y));
146             walls.add(new Coordinate(GameWith-1, y));
147         }
148     }
149 
150     private void addApples(){
151         Coordinate coordinate = null;
152 
153         boolean added = false;
154 
155         while (!added){
156             int x = 1 + random.nextInt(GameWith - 2);
157             int y = 1 + random.nextInt(GameHeight - 2);
158 
159             coordinate = new Coordinate(x, y);
160             boolean collision = false;
161             for (Coordinate s: snake){
162                 if (s.equals(coordinate)){
163                     collision = true;
164                     break;
165                 }
166             }
167             for (Coordinate a: apples){
168                 if (a.equals(coordinate)){
169                     collision = true;
170                 }
171             }
172 
173             added = !collision;
174         }
175 
176         apples.add(coordinate);
177     }
178 
179     public GameState getCurrentGameState(){
180         return currentGameState;
181     }
182 
183     public int getSnakeSize(){
184         return snake.size();
185     }
186 
187 }
```

```
1 package ru.kuryakin.snake.engine;
2 
```

```
3
4 public enum GameState {
5     Ready,
6     Running,
7     Lost
8 }
```

```
1 package ru.kuryakin.snake.engine;
2
3
4 public enum  TileType {
5     Nothing,
6     Wall,
7     SnakeHead,
8     SnakeTail,
9     Apple
10 }
```

```
1 package ru.kuryakin.snake.views;
2
3
4 import android.content.Context;
5 import android.graphics.Canvas;
6 import android.graphics.Color;
7 import android.graphics.Paint;
8 import android.util.AttributeSet;
9 import android.view.View;
10
11 import ru.kuryakin.snake.engine.TileType;
12
13 public class SnakeView extends View {
14     private Paint mPaint = new Paint();
15     private TileType snakeViewMap[][];
16
17     public SnakeView(Context context, AttributeSet attrs) {
18         super(context, attrs);
19     }
20
21     public void setSnakeViewMap(TileType[][] map){
22         this.snakeViewMap = map;
23     }
24
25     @Override
26     protected void onDraw(Canvas canvas) {
27         super.onDraw(canvas);
28
29         if(snakeViewMap != null){
30             float tileSizeX = canvas.getWidth()/snakeViewMap.length;
31             float tileSizeY = canvas.getHeight()/snakeViewMap[0].length;
32
33             mPaint.setColor(Color.rgb(251, 249, 250));
34             canvas.drawRect(0, 0, canvas.getWidth(), canvas.getHeight(), mPaint );
35
36             for (int x = 0; x < snakeViewMap.length; x++) {
37                 for (int y = 0; y < snakeViewMap[x].length; y++) {
38                     switch (snakeViewMap[x][y]){
39
40                         case Nothing:
41                             mPaint.setColor(Color.WHITE);
42                             break;
43                         case Wall:
44                             mPaint.setColor(Color.GREEN);
45                             break;
46                         case SnakeHead:
47                             mPaint.setColor(Color.RED);
48                             break;
49                         case SnakeTail:
50                             mPaint.setColor(Color.GREEN);
51                             break;
52                         case Apple:
53                             mPaint.setColor(Color.RED);
54                             break;
55                     }
56                     canvas.drawRect(x * tileSizeX  + 1 + 10, y * tileSizeY  + 1 + 10,
57                             x * tileSizeX + tileSizeX - 1 + 10,
```

```
                            y * tileSizeY + tileSizeY - 1 + 10, mPaint );

                }

            }
        }
    }
}
```

```
package ru.kuryakin.snake;

import android.app.Activity;
import android.content.pm.ActivityInfo;
import android.os.Handler;
import android.os.Bundle;
import android.view.MotionEvent;
import android.view.View;
import android.widget.Button;
import android.widget.Toast;

import ru.kuryakin.snake.engine.Direction;
import ru.kuryakin.snake.engine.GameState;
import ru.kuryakin.snake.engine.GameEngine;
import ru.kuryakin.snake.views.SnakeView;

public class MainActivity extends Activity implements View.OnTouchListener {


    private Button btnStart;

    private GameEngine gameEngine;
    private SnakeView snakeView;
    private final Handler handler = new Handler();
    private final long updateDelay = 150;

    private float prevX, prevY;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setRequestedOrientation (ActivityInfo.SCREEN_ORIENTATION_PORTRAIT);
        setContentView(R.layout.activity_main);
        btnStart = (Button) findViewById(R.id.btnStart);
        snakeView = (SnakeView)findViewById(R.id.snakeView);
        snakeView.setVisibility(View.INVISIBLE);
        this.menu();
    }

    public void menu() {
        btnStart.setVisibility(View.VISIBLE);
        View.OnClickListener oclStart = new View.OnClickListener() {

            @Override
            public void onClick(View v) {
                btnStart.setVisibility(View.INVISIBLE);

                load();
            }
        };
        btnStart.setOnClickListener(oclStart);
    }

    private void load(){
        gameEngine = new GameEngine();
        gameEngine.initGame();

        snakeView.setOnTouchListener(this);
        snakeView.setVisibility(View.VISIBLE);

        startUpdateHandler();
    }

    private void startUpdateHandler(){
        handler.postDelayed(new Runnable() {
            @Override
            public void run() {
```

```
                gameEngine.Update();

                if (gameEngine.getCurrentGameState() == GameState.Running){
                    handler.postDelayed(this, updateDelay);
                }
                if (gameEngine.getCurrentGameState() == GameState.Lost){
                    onGameLost();
                    snakeView.setVisibility(View.INVISIBLE);
                    menu();
                }
                snakeView.setSnakeViewMap(gameEngine.getMap());
                snakeView.invalidate();
            }
        }, updateDelay);
    }

    private void onGameLost(){
        int res = gameEngine.getSnakeSize() - 6;
        Toast.makeText(this, "Вы проиграли \n Очки: " +  res, Toast.LENGTH_SHORT).show();
    }

    @Override
    public boolean onTouch(View v, MotionEvent event) {
        switch (event.getAction()){
            case MotionEvent.ACTION_DOWN:
                prevX = event.getX();
                prevY = event.getY();

                break;
            case MotionEvent.ACTION_UP:
                float newX = event.getX();
                float newY = event.getY();

                if (Math.abs(newX - prevX) > Math.abs(newY - prevY)){
                    if ( newX > prevX){
                        gameEngine.UpdateDirection(Direction.East);
                    }else {
                        gameEngine.UpdateDirection(Direction.West);
                    }
                } else {
                    if ( newY > prevY){
                        gameEngine.UpdateDirection(Direction.South);
                    }else {
                        gameEngine.UpdateDirection(Direction.Noth);
                    }
                }

                break;
        }

        return true;
    }
}
```

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
    android:theme="@android:style/Theme.Black.NoTitleBar"
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="@drawable/background"
    tools:context="ru.kuryakin.snake.MainActivity">


    <Button
        android:id="@+id/btnStart"
        android:layout_width="250dp"
        android:layout_height="40dp"
        android:layout_centerHorizontal="true"
        android:layout_centerVertical="true"
        android:background="@android:color/holo_green_light"
        android:text="Играть" />

    <ru.kuryakin.snake.views.SnakeView
        android:id="@+id/snakeView"
```

```
23            android:layout_width="match_parent"
24            android:layout_height="match_parent"
25            android:layout_alignParentTop="true"
26            android:layout_alignParentStart="true" />
27
28        <!--<ru.kuryakin.snake.views.ResultView-->
29            <!--android:id="@+id/resultView"-->
30            <!--android:layout_width="match_parent"-->
31            <!--android:layout_height="match_parent" />-->
32    </RelativeLayout>
```