**КУРСОВАЯ РАБОТА**

По дисциплине «Функциональное программирование»

Выполнил

студент гр. 3530904/80007 Базун М.Ф.

Руководитель Лукашин А.А.

16 декабря 2019 г.

Оглавление

[Введение: 3](#_Toc27675707)

[Описание задачи. 4](#_Toc27675708)

[Описание решения: 5](#_Toc27675709)

[Скриншоты 6](#_Toc27675710)

[Вывод 7](#_Toc27675711)

[Приложение. Код программы. 8](#_Toc27675712)

# Введение:

**Функциона́льное программи́рование** — раздел [дискретной математики](https://ru.wikipedia.org/wiki/%D0%94%D0%B8%D1%81%D0%BA%D1%80%D0%B5%D1%82%D0%BD%D0%B0%D1%8F_%D0%BC%D0%B0%D1%82%D0%B5%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0" \o "Дискретная математика) и [парадигма программирования](https://ru.wikipedia.org/wiki/%D0%9F%D0%B0%D1%80%D0%B0%D0%B4%D0%B8%D0%B3%D0%BC%D0%B0_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F), в которой процесс [вычисления](https://ru.wikipedia.org/wiki/%D0%92%D1%8B%D1%87%D0%B8%D1%81%D0%BB%D0%B5%D0%BD%D0%B8%D0%B5" \o "Вычисление) трактуется как вычисление значений [функций](https://ru.wikipedia.org/wiki/%D0%A4%D1%83%D0%BD%D0%BA%D1%86%D0%B8%D1%8F_(%D0%BC%D0%B0%D1%82%D0%B5%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0)" \o "Функция (математика)) в математическом понимании последних (в отличие от [функций](https://ru.wikipedia.org/wiki/%D0%A4%D1%83%D0%BD%D0%BA%D1%86%D0%B8%D1%8F_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)" \o "Функция (программирование)) как подпрограмм в [процедурном программировании](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D1%86%D0%B5%D0%B4%D1%83%D1%80%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5" \o "Процедурное программирование)).

Противопоставляется парадигме [императивного программирования](https://ru.wikipedia.org/wiki/%D0%98%D0%BC%D0%BF%D0%B5%D1%80%D0%B0%D1%82%D0%B8%D0%B2%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5" \o "Императивное программирование), которая описывает процесс вычислений как последовательное изменение [состояний](https://ru.wikipedia.org/wiki/%D0%A1%D0%BE%D1%81%D1%82%D0%BE%D1%8F%D0%BD%D0%B8%D0%B5) (в значении, подобном таковому в [теории автоматов](https://ru.wikipedia.org/wiki/%D0%A2%D0%B5%D0%BE%D1%80%D0%B8%D1%8F_%D0%B0%D0%B2%D1%82%D0%BE%D0%BC%D0%B0%D1%82%D0%BE%D0%B2" \o "Теория автоматов)). При необходимости, в функциональном программировании вся совокупность последовательных состояний вычислительного процесса представляется явным образом, например, как [список](https://ru.wikipedia.org/wiki/%D0%A1%D0%BF%D0%B8%D1%81%D0%BE%D0%BA_(%D0%B8%D0%BD%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0)).

Функциональное программирование предполагает обходиться вычислением результатов функций от исходных данных и результатов других функций, и не предполагает явного хранения состояния программы. Соответственно, не предполагает оно и изменяемость этого состояния (в отличие от [императивного](https://ru.wikipedia.org/wiki/%D0%98%D0%BC%D0%BF%D0%B5%D1%80%D0%B0%D1%82%D0%B8%D0%B2%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5" \o "Императивное программирование), где одной из базовых концепций является [переменная](https://ru.wikipedia.org/wiki/%D0%9F%D0%B5%D1%80%D0%B5%D0%BC%D0%B5%D0%BD%D0%BD%D0%B0%D1%8F_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)" \o "Переменная (программирование)), хранящая своё значение и позволяющая менять его по мере выполнения [алгоритма](https://ru.wikipedia.org/wiki/%D0%90%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC" \o "Алгоритм)).

На практике отличие математической функции от понятия «функции» в императивном программировании заключается в том, что императивные функции могут опираться не только на аргументы, но и на состояние внешних по отношению к функции переменных, а также иметь [побочные эффекты](https://ru.wikipedia.org/wiki/%D0%9F%D0%BE%D0%B1%D0%BE%D1%87%D0%BD%D1%8B%D0%B9_%D1%8D%D1%84%D1%84%D0%B5%D0%BA%D1%82_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)" \o "Побочный эффект (программирование)) и менять состояние внешних переменных. Таким образом, в императивном программировании при вызове одной и той же функции с одинаковыми параметрами, но на разных этапах выполнения алгоритма, можно получить разные данные на выходе из-за влияния на функцию состояния переменных. А в функциональном языке при вызове функции с одними и теми же аргументами мы всегда получим одинаковый результат: выходные данные зависят только от входных. Это позволяет средам выполнения программ на функциональных языках [кешировать](https://ru.wikipedia.org/wiki/%D0%9C%D0%B5%D0%BC%D0%BE%D0%B8%D0%B7%D0%B0%D1%86%D0%B8%D1%8F" \o "Мемоизация) результаты функций и вызывать их в порядке, не определяемом алгоритмом и распараллеливать их без каких-либо дополнительных действий со стороны программиста (что обеспечивают функции без побочных эффектов — чистые функции).

[Лямбда-исчисление](https://ru.wikipedia.org/wiki/%D0%9B%D1%8F%D0%BC%D0%B1%D0%B4%D0%B0-%D0%B8%D1%81%D1%87%D0%B8%D1%81%D0%BB%D0%B5%D0%BD%D0%B8%D0%B5) являются основой для функционального программирования, многие функциональные [языки](https://ru.wikipedia.org/wiki/%D0%AF%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F" \o "Язык программирования) можно рассматривать как «надстройку» над ними

# Описание задачи.

Написание игры «Snake», используя подходы функционального программирования.

Игрок управляет длинным, тонким существом, напоминающим [змею](https://ru.wikipedia.org/wiki/%D0%97%D0%BC%D0%B5%D1%8F), которое ползает по [плоскости](https://ru.wikipedia.org/wiki/%D0%9F%D0%BB%D0%BE%D1%81%D0%BA%D0%BE%D1%81%D1%82%D1%8C_(%D0%B3%D0%B5%D0%BE%D0%BC%D0%B5%D1%82%D1%80%D0%B8%D1%8F)) (как правило, ограниченной стенками), собирая еду (или другие предметы), избегая столкновения с собственным хвостом и краями игрового поля. В некоторых вариантах на поле присутствуют дополнительные препятствия. Каждый раз, когда змея съедает кусок пищи, она становится длиннее, что постепенно усложняет игру. Игрок управляет направлением движения головы змеи (обычно 4 направления: вверх, вниз, влево, вправо), а хвост змеи движется следом. Игрок не может остановить движение змеи.

# Описание решения:

Игра идет до тех пор, пока змейка не заденет своё тело или же не соприкоснется со границей поля. Управление идет за счет клавиш: up, down, right, left.

//The game of Snake in Scala

package snake

import javax.swing.{JFrame,JPanel,Timer}

import java.awt.Graphics

import java.awt.event.{ActionEvent,ActionListener,KeyListener,KeyEvent}

import scala.actors.Actor

import scala.actors.Actor.\_

case class Listen(actor: Actor)

case class SnakeMoved()

object Game extends JFrame

{

val Height = 30

val Width = 30

val Scale = 10

def main(args: Array[String]) {

//launch the program; start the timer, etc.

val gameBoard = new SnakeDataModel

val view = new View(gameBoard)

add(view)

val controller = new Controller(gameBoard)

addKeyListener(controller)

setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE)

setVisible(true)

this.setSize(Width \* Scale + Scale / 2, Height \* Scale + Scale / 2)

controller.start

}

}

class SnakeDataModel extends Actor {

type Point = (Int, Int)

type Vector = (Int, Int)

type Snake = List[Point]

private var handlers : List[Actor] = Nil

var snake: Snake = List()

var apple: Point = (0,0)

//move the segment as specified by dir

private def translate(seg: Point, dir: Vector) : Point = {

(seg.\_1 + dir.\_1, seg.\_2 + dir.\_2)

}

private def randomInt(i:Int): Int = (Math.random \* i).toInt

def isGameOver = snake.isEmpty

def isAppleInSnake = snake.contains(apple)

//set the snake back to its start position

def resetSnake {

snake = (for (i <- List.range(1,10)) yield (i, 2)) reverse

}

//move the apple to a random spot on the board, not occupied by the snake

def resetApple {

do apple = (randomInt(Game.Width - 4) + 1, randomInt(Game.Height - 4) + 1)

while (snake.intersect(List(apple)) != List.empty)

}

//code to handle listeners to SnakeMoved event

def act = {

loop {

react {

case Listen(actor) => handlers = actor :: handlers

}

}

}

//advance one unit in the specified direction; grow forward if the

//apple is eaten.

def move(dir: Vector) {

type sCont = (Snake => Snake)

val newSnake = translate(snake.head, dir) :: snake

def buildSnake(snake: Snake, k: sCont) : Snake = {

val break = k;

def iter(snake: Snake, k: sCont) : Snake = {

(snake: @unchecked) match {

//loser

case h :: t if t.contains(h) => break(List())

//ate the apple

case h :: t if h == apple =>

iter(t, s => k(translate(h, dir) :: h :: s))

//truncate the tail

case h :: List() => k(List())

//rest of body remains constant

case h :: t => iter(t, s => k(h :: s))

}

}

iter(snake, s => s)

}

snake = buildSnake(newSnake, s => s)

handlers foreach (\_ ! SnakeMoved)

}

//initialize game state

resetSnake

resetApple

}

class Controller(model: SnakeDataModel) extends ActionListener with KeyListener {

import java.awt.event.KeyEvent.\_

val timer = new Timer(100, this)

var dir = (1,0)

override def actionPerformed(e: ActionEvent) {

if (model.isAppleInSnake)

model.resetApple

if (model.isGameOver) {

dir = (1,0)

model.resetSnake

model.resetApple

}

model.move(dir)

}

override def keyPressed(e: KeyEvent) {

//change direction

dir = e.getKeyCode match {

case VK\_LEFT => (-1, 0)

case VK\_RIGHT => (1, 0)

case VK\_UP => (0, -1)

case VK\_DOWN => (0, 1)

case \_ => dir

}

}

override def keyReleased(e: KeyEvent) {}

override def keyTyped(e: KeyEvent){}

def start {

timer.start

model.start

}

}

class View(model: SnakeDataModel) extends JPanel with Actor

{

override def paintComponent(g : Graphics) {

super.paintComponent(g)

val scale = Game.Scale

val apple = model.apple

//background

g.setColor(java.awt.Color.LIGHT\_GRAY)

g.fillRect(Game.Width \* scale, Game.Height \* scale, 0, 0)

//draw apple

g.setColor(java.awt.Color.RED)

g.fillRect(apple.\_1 \* scale, apple.\_2 \* scale, scale, scale)

//draw snake

g.setColor(java.awt.Color.GREEN)

model.snake foreach (seg =>

g.fillRect(seg.\_1 \* scale, seg.\_2 \* scale, scale, scale)

)

}

def act = {

loop {

react {

case SnakeMoved => repaint()

}

}

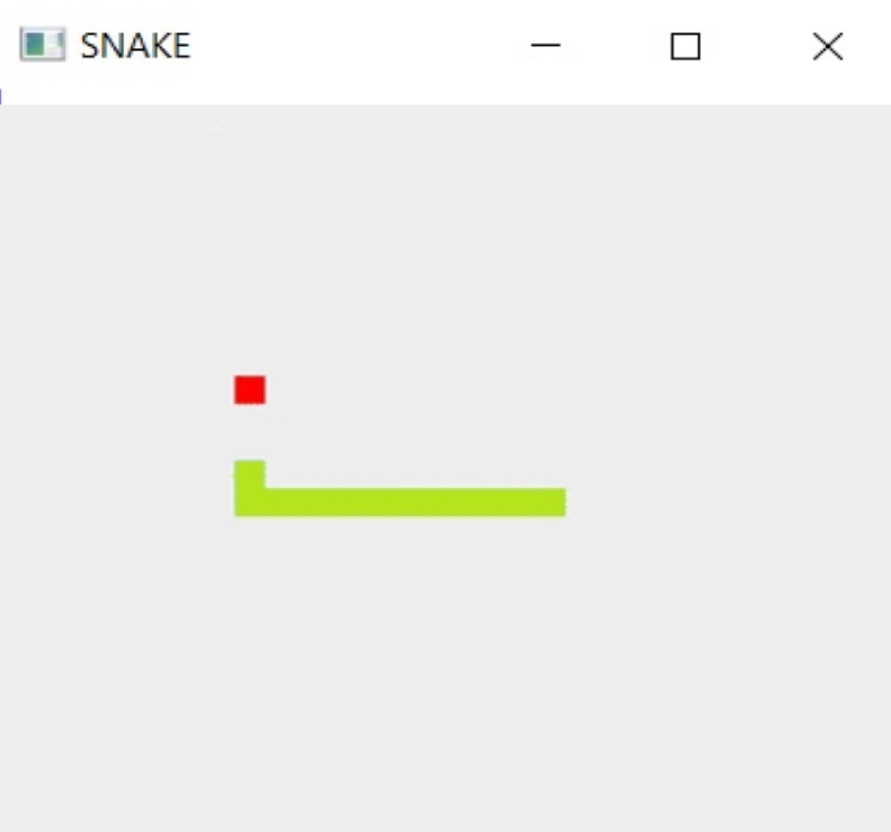
}

model ! Listen(this)

start

}

# Скриншоты



# Вывод

В результате проделанной курсовой работы были улучшены навыки программирования на функциональном, игра змея была сделана с помощью языка скала.