Санкт-Петербургский Политехнический Университет Петра Великого

Институт компьютерных наук и технологий

Высшая школа интеллектуальных систем и суперкомпьютерных технологий

КУРСОВОЙ ПРОЕКТ

Дисциплина: Алгоритмы и структуры данных

Тема: Разработка GUI приложения «Игра в "даты"»

Выполнил

студент гр. 3530903/80003 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Цаплин И.А.

(подпись)

Руководитель \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Глухих М. И.

(подпись)

«\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2019 г.

Санкт-Петербург

2019

Оглавление

[Техническое задание 3](#_Toc27565557)

[Общая архитектура программы 4](#_Toc27565558)

[Подробное описание правил и логики игры, а также способ формирования "карты" ходов для бота. 5](#_Toc27565559)

[Содержание игрового меню 7](#_Toc27565560)

[Сценарий игры 8](#_Toc27565561)

[Листинг программы 9](#_Toc27565562)

[Main.java 9](#_Toc27565563)

[Bot.kt 10](#_Toc27565564)

[ControllerN.kt 11](#_Toc27565565)

[GameAnd 14](#_Toc27565566)

[GameBarN 15](#_Toc27565567)

[GameButton 16](#_Toc27565568)

[GameLabel 16](#_Toc27565569)

[Тесты 17](#_Toc27565570)

[Скриншоты программы 19](#_Toc27565571)

# Техническое задание

Задается какая-то исходная дата ( день и месяц). Игра идет с роботом. Каждый игрок на своем ходе задает более позднюю дату, по сдедующему правилу: увеличивая на 1 или 2 либо день в месяце, либо месяц (но не то и другое сразу). При этом сочетание дня и месяца должно оставаться реальной датой. Игрок, задавший 31.12 проигрывает. Примеры:

Исходная дата: 29.12

игрок 30.12

робот 31.12 - Проиграл

Исходная дата: 28.10

игрок 28.12

робот 30.12

игрок 31.12 - Проиграл

# Общая архитектура программы

Программа разделена на пакеты model, view и controller в соответствии с паттерном MVC.

В пакете model содержаться класс GameDate, отвечающий за описание такой сущности как "дата". В этом классе реализованы такие функции, как: установка конкретного дня или месяца, определения следующих возможных дат, проверка даты на "реальность" и повышение дня или месяца.

В пакете view содержаться классы:

* GameAnd - класс, расширяющий класс Stage библиотеки JavaFX и описывающий то, как должно выглядеть окно, появляющееся при окончании игры.
* GameBarN - класс, расширяющий класс Bar библиотеки JavaFX и описывающий то, как должно выглядеть "меню" игры и какие пункты в себя включать.
* GameButton - класс, расширяющий класс Button библиотеки JavaFX и описывающий то, как должны выглядеть кнопки "повышения" дня и месяца в игровом окне.
* GameLabel - класс, расширяющий класс Label библиотеки JavaFX и описывающий то, как должно выглядеть отображение "текущего" дня и месяца.

В пакете controller содержаться классы:

* Bot - класс, описывающий сущьность бота, т.е. "интелекта" игры. В этом классе реализованы функции формирования "карты" ходов и самого хода непосредственно.
* ControllerN - класс, который "связывает" все остальные и содержит в себе реализации таких функция как: открытие и закрытие конкретных окон, завершение и перезапуск самой игры, добавление на окна кнопок и надписей, а также обновление их состояний.

# Подробное описание правил и логики игры, а также способ формирования "карты" ходов для бота.

Игрок, как и бот(далее по тексту бот будет считаться игроком, как и пользователь), может добавлять к месяцу или дню текущей игровой даты 1 или 2 соответственно, главное, чтоб эта игровая дата оставалась "реально существующей". Это означает, что игрок не сможет получить такие нереальные даты как "40.11", "15.81" и даже "31.11".

Если игрок увеличивает число, и переходит "границу" существующих чисел для месяца, то игровая дата переходит на следующий месяц и соответствующее число. Так, например, если добавить к дню даты "30.11" два, то игровая дата установиться на число "2.11"

Как только один из игроков после своего хода получает число "31.12" (то есть конец года), он проигрывает. Выигрывает соответственно его оппонент.

Чтобы бот мог "играть" и ходил не просто случайно, для него формируется "карта" ходов. Она представляет из себя два списка: один содержит даты, при которых бот может проиграть, а другой содержит даты из которых есть беспроигрышный ход. Назовём их "красный" и "зелёный" соответственно для простоты восприятия. Пока наши списки пустые.

Заполняем их в соответствии следующим пунктам:

1. Возьмем число "31.12" и добавим его в зелёный список, следуя логике, что если мы получили число "31.12", то противнику пришлось сделать ход, который завершил игру в нашу пользу. Ход при этом числе можно задать любой, т.к. он нам уже не понадобиться.
2. Уменьшаем число на один календарный день*, в нашем случае получаем "30.12".*
3. Посмотрим куда мы можем "прийти", т.е. какую/какие даты мы можем получить после нашего хода. *В нашем случае это единственная дата "31.12".* Если все эти даты находятся в зелёном списке, то добавляем нынешнюю игровую дату в красный список (т.к. после нашего хода, каким бы он ни был, оппонент получит дату с которой есть беспроигрышный ход). Если же хотя бы одна из этих дат находиться в красном списке, то определяем какой ход нужно сделать для того чтоб получить эту "красную" дату и добавляем нынешнюю игровую дату в зелёный список вместе с соответствующим ходом.
4. Повторяем пункты 2 и 3 пока не получим дату "1.1"

В итоге, мы имеем "карту" ходов для бота, т.к. он определит по нашим двум спискам, какой ход ему следует сделать.

Весь этот процесс совершается при создании экземпляра класса Bot.

# Содержание игрового меню

Игровое меню содержит в себе всего два пункта, в каждом из которых по два подпункта. Итого мы можем совершить четыре действия.

Меню содержит:

1. Пункт меню ***"Help",*** который содержит в себе:
   1. Подпункт ***"Rules"*,** который открывает окно с кратким описанием правил игры.
   2. Подпункт ***"MyGitHub"*,** который открывает в браузере по умолчанию страницу репозитория проекта на GitHub.
2. Пункт меню ***"Process"*,** который содержит в себе:
   1. Подпункт ***"Restart"*,** который начинает игру сначала.
   2. Подпункт ***"Close"*,** который закрывает прриложение.

# Сценарий игры

После запуска приложения, открывается окно выбора начальной даты. Дата может быть введена вручную игроком в соответствующую строку, а может быть выбрана им в календаре, который открывается нажатием на специальный символ возле строки ввода.

После выбора игроком даты, он нажимает на кнопку ***"ок"*** и окно с выбором даты закрывается и открывается главное игровое окно, содержащее игровое меню, описанное выше и игровое поле.

Описание игрового поля: Игровое поле представляет из себя два столбца, для дня и месяца соответственно. В каждом из столбцов находятся две кнопки: "+1" и "+2", естественно они добавляют к значению 1 и 2. Также в каждом из столбцов находится цифра отражающая текущее значение.

Игроки (пользователь и бот) начинают ходить по очереди. Первый ход всегда делает пользователь.

Когда один из игроков после своего хода получает число 31.12, он проигрывает, а это означает, что открывается новое окно с сообщением о том, кто победит и двумя кнопками ***"Restart"*** и ***"Close"***.

# Листинг программы

## Main.java

package model;  
  
import controller.ControllerN;  
import javafx.application.Application;  
import javafx.stage.Stage;  
  
public class Main extends Application {  
  
 @Override  
 public void start(Stage primaryStage) throws Exception{  
 ControllerN controller = new ControllerN(primaryStage);  
 controller.openChoice();  
 if (controller.getOk()) {  
 controller.openPrStage();  
 }  
 }  
  
 public static void main(String[] args) {  
 *launch*(args);  
 }  
}

## Bot.kt

package controller  
  
import model.GameDate  
  
class Bot(private val gameDate: GameDate) {  
 //red - даты, при которых проигрываем  
 //green - даты, при которых выигрываем  
 private val red = *mutableListOf*<Pair<Int, Int>>()//месяц день  
 private val green = *mutableMapOf*<Pair<Int, Int>, Pair<Int, Int>>() // месяц-день, добавить к месяцу-добавить к дню  
  
 init {  
 val mainDate = GameDate(11,31)//временный оъект, для построения "карты" ходов бота  
 green[11 *to* 31] = 0 *to* 0  
 do {  
 val pairNow = mainDate.month *to* mainDate.day  
 if (!red.contains(pairNow) && !green.containsKey(pairNow)) {  
 val daysPossible = mainDate.nextDatesPossible()  
 var addInRed = true  
 var step = 0 *to* 0  
 daysPossible.*forEach* **{** if (red.contains(**it**.key.month *to* **it**.key.day)) {  
 addInRed = false  
 step = **it**.value  
 }  
 **}** if (addInRed) red.add(mainDate.month *to* mainDate.day)  
 else green[pairNow] = step  
 }  
 } while (mainDate.minusDay())  
 }  
  
 fun step() {  
// println("ухожу с ${gameDate.month to gameDate.day}")  
 val pairNow = gameDate.month *to* gameDate.day  
 if (green.containsKey(pairNow)) {  
 //println("${green[pairNow]}")  
 gameDate.plusDay(green[pairNow]!!.second)  
 gameDate.plusMouth(green[pairNow]!!.first)  
// println("c клетки зелёной $pairNow Бот сходил ${green[pairNow]}")  
 }else if (red.contains(gameDate.month *to* gameDate.day)) {  
 var rnd = (1..2).*random*()  
 while (!gameDate.dayPlusPossible(rnd)) {  
 rnd = (1..2).*random*()  
 }  
 gameDate.plusDay(rnd)  
// println("c клетки красной $pairNow Бот сходил (0, $rnd)")  
 }  
 }  
}

## ControllerN.kt

package controller  
  
import javafx.geometry.Pos  
import javafx.scene.Scene  
import javafx.scene.control.Button  
import javafx.scene.control.DatePicker  
import javafx.scene.control.Label  
import javafx.scene.layout.BorderPane  
import javafx.scene.layout.HBox  
import javafx.scene.layout.VBox  
import javafx.stage.Stage  
import model.GameDate  
import view.GameAnd  
import view.GameBarN  
import view.GameButton  
import view.GameLabel  
import java.awt.Desktop  
import java.net.URI  
import java.time.LocalDate  
  
  
class ControllerN(private val primaryStage: Stage) {  
 private val dateN = GameDate(0, 1) // та дата, с которое "играем"  
 private val bot = Bot(dateN)  
 private val btns = *mutableListOf*<GameButton>()  
 var ok: Boolean = false  
  
 var dayPlusOnePossible = dateN.dayPlusPossible(1)  
 var dayPlusTwoPossible = dateN.dayPlusPossible(2)  
 var monthPlusOnePossible = dateN.monthPlusPossible(1)  
 var monthPlusTwoPossible = dateN.monthPlusPossible(2)  
  
  
 var labelL = GameLabel()  
 var labelR = GameLabel()  
 val bar = GameBarN(this)  
  
 var choiceIsOpen = false  
  
 var winner = "Player"  
  
 fun openChoice() {  
 val lbl = Label("Choice Date: ")  
 val choiceDate = Button("Ok!")  
 val choice = Stage()  
 val vPane = VBox(20.0)  
 val picker = DatePicker(LocalDate.of(2019, 1, 1))  
 vPane.*children*.addAll(lbl, picker, choiceDate)  
 vPane.*alignment* = Pos.CENTER  
 val testScene = Scene(vPane, 200.0, 200.0)  
 choice.*scene* = testScene  
 choiceDate.setOnAction **{** ok = true  
 choice.close()  
 dateN.setDate(picker.*value*.*monthValue* - 1, picker.*value*.*dayOfMonth*)  
 **}** choice.*isResizable* = false  
 choiceIsOpen = true  
 choice.showAndWait()  
 choiceIsOpen = false  
 }  
  
 fun openPrStage() {  
 labelL.*text* = "" + dateN.day  
 labelR.*text* = "" + (dateN.month + 1)  
 val vLabelL = labelL  
 val vLabelR = labelR  
 val topLabelL = Label("day:")  
 val topLabelR = Label("month:")  
  
 addBtn(1, 0)  
 addBtn(2, 0)  
 addBtn(0, 1)  
 addBtn(0, 2)  
  
 val vBoxL = VBox(20.0)  
 val topVBoxL = VBox(1.0)  
 topVBoxL.*alignment* = Pos.CENTER  
 topVBoxL.*children*.addAll(topLabelL, btns[1])  
 vBoxL.*children*.addAll(topVBoxL, vLabelL, btns[0])  
 vBoxL.*alignment* = Pos.CENTER  
  
 val vBoxR = VBox(20.0)  
 val topVBoxR = VBox(1.0)  
 topVBoxR.*alignment* = Pos.CENTER  
 topVBoxR.*children*.addAll(topLabelR, btns[3])  
 vBoxR.*children*.addAll(topVBoxR, vLabelR, btns[2])  
 vBoxR.*alignment* = Pos.CENTER  
  
 val hBox = HBox(30.0)  
 hBox.*children*.addAll(vBoxL, vBoxR)  
 hBox.*alignment* = Pos.CENTER  
  
 val root = BorderPane()  
 root.*center* = hBox  
 root.*top* = bar  
 bar.*prefWidth* = root.*width* val scene = Scene(root)  
 primaryStage.*scene* = scene  
 primaryStage.*isResizable* = false  
 resetBtns()  
 primaryStage.show()  
 winner = "Player"  
 checkWin()  
 }  
  
 private fun addBtn(plusD: Int, plusM: Int) {  
 val newBtn = GameButton(plusD, plusM)  
 newBtn.setOnAction **{** dateN.plusDay(plusD)  
 dateN.plusMouth(plusM)  
 afterStep()  
 if (!dateN.victoryNow()) {  
 bot.step()  
 afterStep()  
 }  
 **}** resetBtn(newBtn)  
 btns.add(newBtn)  
 }  
  
 private fun checkWin() {  
 if (dateN.victoryNow()) {  
 primaryStage.close()  
 GameAnd(this)  
 }  
 }  
  
 private fun resetBtn(el: GameButton) {  
 val plusD = el.plusD  
 val plusM = el.plusM  
 val temp = when {  
 plusD == 1 -> dayPlusOnePossible  
 plusD == 2 -> dayPlusTwoPossible  
 plusM == 1 -> monthPlusOnePossible  
 plusM == 2 -> monthPlusTwoPossible  
 else -> false  
 }  
 el.*isDisable* = !temp  
 }  
  
 private fun resetBtns() {  
 dayPlusOnePossible = dateN.dayPlusPossible(1)  
 dayPlusTwoPossible = dateN.dayPlusPossible(2)  
 monthPlusOnePossible = dateN.monthPlusPossible(1)  
 monthPlusTwoPossible = dateN.monthPlusPossible(2)  
 for (el in btns)  
 resetBtn(el)  
 }  
  
 private fun afterStep() {  
 resetBtns()  
 winner = when (winner) {  
 "Bot" -> "Player"  
 else -> "Bot"  
 }  
// println(winner)  
 checkWin()  
 labelL.*text* = "" + dateN.day  
 labelR.*text* = "" + (dateN.month + 1)  
 }  
  
 fun goToGH() {  
 Desktop.getDesktop().browse(URI("https://github.com/TsaplinIA"))  
 }  
  
 fun restartGame() {  
 ok = false  
 primaryStage.close()  
 openChoice()  
 openPrStage()  
 }  
  
 fun closeGame() {  
 primaryStage.close()  
 }  
  
}

## GameAnd

package view  
  
import controller.ControllerN  
import javafx.geometry.Pos  
import javafx.scene.Scene  
import javafx.scene.control.Button  
import javafx.scene.control.Label  
import javafx.scene.layout.VBox  
import javafx.scene.text.Font  
import javafx.stage.Stage  
  
class GameAnd(val controller: ControllerN): Stage() {  
 val vBox = VBox()  
 val root = Scene(vBox)  
 val labelW = Label("${controller.winner} WIN!!!")  
 val label = Label("Game Over")  
 val btn = Button("Close")  
 val btn2 = Button("Restart")  
 init {  
 btn.setOnAction **{** this.close()  
 **}** btn2.setOnAction **{** this.close()  
 controller.restartGame()  
 **}** vBox.*children*.addAll(labelW, label,btn,btn2)  
 vBox.*alignment* = Pos.CENTER  
 this.*scene* = root  
 val font = Font(50.0)  
 label.*font* = font  
 labelW.*font* = font  
 this.show()  
 }  
}

## GameBarN

package view  
  
import controller.ControllerN  
import javafx.scene.control.Alert  
import javafx.scene.control.Menu  
import javafx.scene.control.MenuBar  
import javafx.scene.control.MenuItem  
  
class GameBarN(val cont: ControllerN): MenuBar() {  
 val item1 = MenuItem("Rules")  
 val item2 = MenuItem("My GitHub")  
 val item3 = MenuItem("Restart")  
 val item4 = MenuItem("Close")  
 val menu = Menu("Help")  
 val menu2 = Menu("Process")  
 init {  
 item1.setOnAction **{** val rule = Alert(Alert.AlertType.INFORMATION)  
 rule.*contentText* = "[RU]\n" +  
 "Задается какая-то исходная дата ( день и месяц)." +  
 " Игра идет с роботом." +  
 " Каждый игрок на своем ходе задает более позднюю дату," +  
 " по сдедующему правилу: увеличивая на 1 или 2 либо день в месяце," +  
 " либо месяц (но не то и другое сразу)." +  
 " При этом сочетание дня и месяца должно оставаться реальной датой." +  
 " Игрок, задавший 31.12 проигрывает.\n\n\n" +  
 "[EN]\n" +  
 "Some initial date (day and month) is set." +  
 " The game is with a robot. Each player at his turn sets a later date," +  
 " according to the following rule:" +  
 " increasing by 1 or 2 either a day in a month or a month (but not both at once)." +  
 " In this case, the combination of day and month should remain a real date." +  
 " The player who asked 31.12 loses."  
 rule.*headerText* = "Rules"  
 rule.*title* = "Rules"  
 rule.show()  
 **}** item2.setOnAction **{** cont.goToGH()  
 **}** item3.setOnAction **{** cont.restartGame()  
 **}** item4.setOnAction **{** cont.closeGame()  
 **}** menu.*items*.addAll(item1, item2)  
 menu2.*items*.addAll(item3,item4)  
 this.*menus*.addAll(menu,menu2)  
 }  
}

## GameButton

package view  
  
import javafx.scene.control.Button  
  
class GameButton(val plusD: Int, val plusM: Int): Button() {  
 init {  
 *text* = "+${plusD + plusM}"  
 setPrefSize(60.0,30.0)  
 }  
}

## GameLabel

package view  
  
import javafx.scene.control.Label  
import javafx.scene.text.Font  
  
class GameLabel: Label() {  
 init {  
 this.*font* = Font(50.0)  
 }  
}

# Тесты

**GameDateTests.kt**

package model  
  
import org.junit.Test  
  
import org.junit.Assert.\*  
  
class GameDateTest {  
  
 @Test  
 fun plusMouth() {  
 val date = GameDate(0,1)  
 date.plusMouth(2)  
 assertEquals(2,date.month)  
 date.plusMouth(1)  
 assertEquals(3,date.month)  
 }  
  
 @Test  
 fun plusDay() {  
 val date = GameDate(0,1)  
 date.plusDay(2)  
 assertEquals(3,date.day)  
 date.plusDay(1)  
 assertEquals(4,date.day)  
 }  
  
 @Test  
 fun minusDay() {  
 val date = GameDate(11,30)  
 date.minusDay()  
 assertEquals(29, date.day)  
 assertEquals(11, date.month)  
 date.setDate(11,1)  
 date.minusDay()  
 assertEquals(30, date.day)  
 assertEquals(10, date.month)  
 }  
  
 @Test  
 fun nextDatesPossible() {  
 val date = GameDate(0,1)  
 assertEquals(4, date.nextDatesPossible().size)  
 date.setDate(11,31)  
 assertEquals(0, date.nextDatesPossible().size)  
 date.setDate(10,30)  
 assertEquals(3,date.nextDatesPossible().size)  
 }  
  
 @Test  
 fun dayPlusPossible() {  
 val date = GameDate(0, 1)  
 assertTrue(date.dayPlusPossible(1))  
 assertTrue(date.dayPlusPossible(2))  
 date.setDate(10, 29)  
 assertTrue(date.dayPlusPossible(1))  
 assertTrue(date.dayPlusPossible(2))  
 date.setDate(11,30)  
 assertTrue(date.dayPlusPossible(1))  
 assertFalse(date.dayPlusPossible(2))  
 }  
  
 @Test  
 fun monthPlusPossible() {  
 val date = GameDate(0, 1)  
 assertTrue(date.monthPlusPossible(1))  
 assertTrue(date.monthPlusPossible(2))  
 date.setDate(10,29)  
 assertTrue(date.monthPlusPossible(1))  
 assertFalse(date.monthPlusPossible(2))  
 date.setDate(9, 31)  
 assertFalse(date.monthPlusPossible(1))  
 assertTrue(date.monthPlusPossible(2))  
 }  
}

# Choice.PNGСкриншоты программы

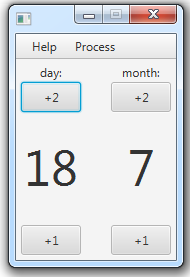
Рис.1.Окно выбора даты

Рис.2.Главное игровое окно

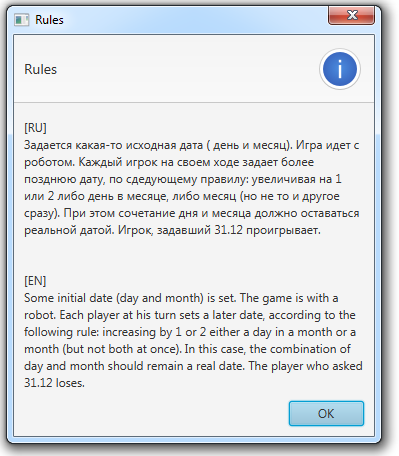
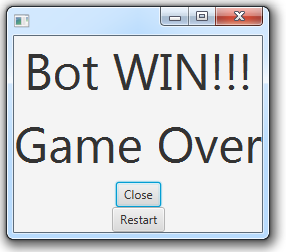
Рис.3.Окно с правилами игры

Рис.4. Окно окончания игры