ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ

ВЫСШЕГО ОБРАЗОВАНИЯ

***«*САНКТ-ПЕТЕРБУРГСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ ПЕТРА ВЕЛИКОГО»**

Институт компьютерных наук и технологий

**Отчет о прохождении учебной практики**

|  |
| --- |
| Никифоров Тимофей Алексеевич |

*(Ф.И.О. обучающегося)*

|  |
| --- |
| 2 курс, гр. 3530901/80003 |

|  |
| --- |
| 09.03.01 «Информатика и вычислительная техника» |

*(Направление подготовки (код и наименование)*

|  |
| --- |
| **Место прохождения практики**: Высшая школа информационных систем и суперкомпьютерных технологий (ВШИСиСТ) ИКНТ ФГАОУ ВО «СПбПУ» с использованием электронного обучения и дистанционных образовательных технологий. |

|  |
| --- |
| **Сроки практики:** с 22 июня по 18 июля 2020 г. |

|  |
| --- |
| **Руководитель практики:** |

|  |
| --- |
| Жвариков В. А., к.т.н., доцент ВШИСиСТ ИКНТ |

*(Ф.И.О., уч. степень, должность)*

|  |
| --- |
| **Оценка (зачет): \_\_\_\_\_\_\_\_\_\_\_** |

|  |
| --- |
|  |

|  |
| --- |
| Руководитель практики : Жвариков В. А. |

|  |
| --- |
| Обучающийся: Никифоров Т. А. |

|  |
| --- |
| Дата: |

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ

ВЫСШЕГО ОБРАЗОВАНИЯ

***«*САНКТ-ПЕТЕРБУРГСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ ПЕТРА ВЕЛИКОГО»**

Институт компьютерных наук и технологий

**ИНДИВИДУАЛЬНЫЙ ПЛАН (ЗАДАНИЕ И ГРАФИК)**

**ПРОВЕДЕНИЯ ПРАКТИКИ**

|  |
| --- |
| Ф.И.О. обучающегося Никифоров Тимофей Алексеевич |

|  |  |
| --- | --- |
| **Направление подготовки** (код/наименование): 09.03.01, «Информатика и вычислительная техника» | |
| **Профиль** (код/наименование): 09.03.01 | |
| **Вид практики:** учебная | |
| **Тип практики:** ознакомительная | |
| **Место прохождения практики**: Высшая школа информационных систем и суперкомпьютерных технологий (ВШИСиСТ) ИКНТ ФГАОУ ВО «СПбПУ» с использованием электронного обучения и дистанционных образовательных технологий. | |
| |  | | --- | |  | | Руководитель практики: Жвариков В. А., к.т.н., доцент ВШИСиСТ ИКНТ | | *(Ф.И.О., уч.степень, должность)* | |

**Рабочий график проведения учебной практики**

Сроки практики: с **22.06.2020** г. по **18.07.2020** г.

|  |  |  |  |
| --- | --- | --- | --- |
| № п/п | Этапы (периоды) практики | Вид работ | Сроки прохождения этапа (периода) практики |
| 1 | Организационный этап | Установочная лекция (вебинар) для разъяснения целей, задач, содержания и порядка прохождения практики, выдача сопроводительных документов по практике  Основная тема: написание игры Твистер для Андроид на языке Python | 22.06.2020 г. |
| 2 | Основной  этап | Изучение сторонних библиотек для создания программного обеспечения и основ объектно-ориентированного программирования в Python.  Сбор информации, обработка, систематизация и анализ фактического и теоретического материала.  Содержание практики: разработка мобильного приложения для операционной системы Андроид.  Планируемые результаты прохождения практики: получение навыков создания мобильных программ и взаимодействия с пользовательским устройством. | 23.06.2020 - 15.07.2020 г. |
| 3 | Заключительный этап | Подготовка отчета | 16.07.- 17.07.2020 г. |
| Защита отчета по практике (сдача зачета) | 17.07 - 18.07.2020 г |

Обучающийся \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Никифоров Т. А.

Руководитель практики \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Жвариков В. А.

Оглавление

[Техническое задание 4](#_Toc45274919)

[Метод решения 4](#_Toc45274920)

[Скриншоты программы 6](#_Toc45274921)

[Список использованных источников 10](#_Toc45274922)

[Приложение 10](#_Toc45274923)

# Техническое задание

Цель работы: получить начальные навыки создания программ для Андроид и взаимодействия с пользовательским устройством (в т. ч. использование технологии multi-touch); освоить применение сторонних библиотек для создания программного обеспечения на языке программирования Python; изучить основы объектно-ориентированного программирования в языке программирования Python.

В ходе летней практики планировалось создание мобильного приложения для операционной системы Android – игры Твистер – с использованием фреймворка Kivy, который предоставляет широкие возможности для кроссплатформенной разработки, в том числе для операционной системы Android.

GitHub-репозиторий: https://github.com/nikiforovta/AndroidTwister

# Метод решения

Разрабатываемая игра должна быть похожа на оригинальную игру Твистер: на игровом поле располагаются специальные секции (4 ряда по 6 кругов) разного цвета, на одну из которых один из игроков, играющих на одном устройстве, в результате прокрутки рулетки должен за ограниченное время поставить палец и удерживать до окончания игры или смены положения в результате очередной прокрутки рулетки.

На главном экране игры располагается игровое поле, описанное выше, рулетка, отвечающая за определение позиции хода игрока, номер игрока, который ходит в данный момент, и таймер, отображающий время, оставшееся для изменения положения игрока на поле.

В меню настройки игры есть возможность выбора количества игроков, пальцев и времени на смену положения пальца игрока на поле за ход. Также есть возможность ознакомиться с описанием приложения и правилами игры.

Рассмотрим методы, приведенные в классе FieldWidget, отвечающем за отрисовку игрового поля и подсчет параметров игры:

1. \_\_init\_\_ – конструктор, производящий начальную отрисовку игрового поля;
2. update\_field – обновление поля в результате нажатия;
3. on\_touch\_down/on\_touch\_up – переопределенные стандартные методы для виджетов, отвечающие за обработку взаимодействия с экраном; используется для отслеживания ходов игроков.

Рассмотрим методы, приведенные в классе RouletteWidget, отображающий рулетку на игровом поле:

1. \_\_init\_\_ - конструктор, производящий начальную отрисовку рулетки;
2. change\_color – смена цветов рулетки.

Рассмотрим методы, приведенные в классе MoveTimer, описывающем таймер хода на игровом поле:

1. start – начало анимации таймера:
   1. find\_next – поиск следующего невыбывшего игрока;
   2. finish\_callback – переопределённый метод, отвечающий за действия после окончания хода игрока;
2. on\_a – переопределенный метод, отвечающий за отображение таймера на экране.

В классе TwisterApp, отвечающем за отрисовку главного экрана приложения, есть метод build, реализующий эту самую отрисовку.

Помимо приведенных выше, есть методы, не относящиеся к какому-либо классу, но отвечающие за корректное функционирование приложения:

1. define\_circle – определение секции игрового поля с помощью передаваемых координат;
2. show\_info – показ всплывающего окна информации об игре;
3. show\_settings – пока всплывающего окна настроек игры:
   1. on\_slider\_value\_change – отображение значения ползунка в текстовом поле рядом с ползунком;
   2. change\_values – изменение значение игровых параметров после их задания в окне настроек.

Полный код программы с подробными комментариями приведен в Приложении.

# Скриншоты программы

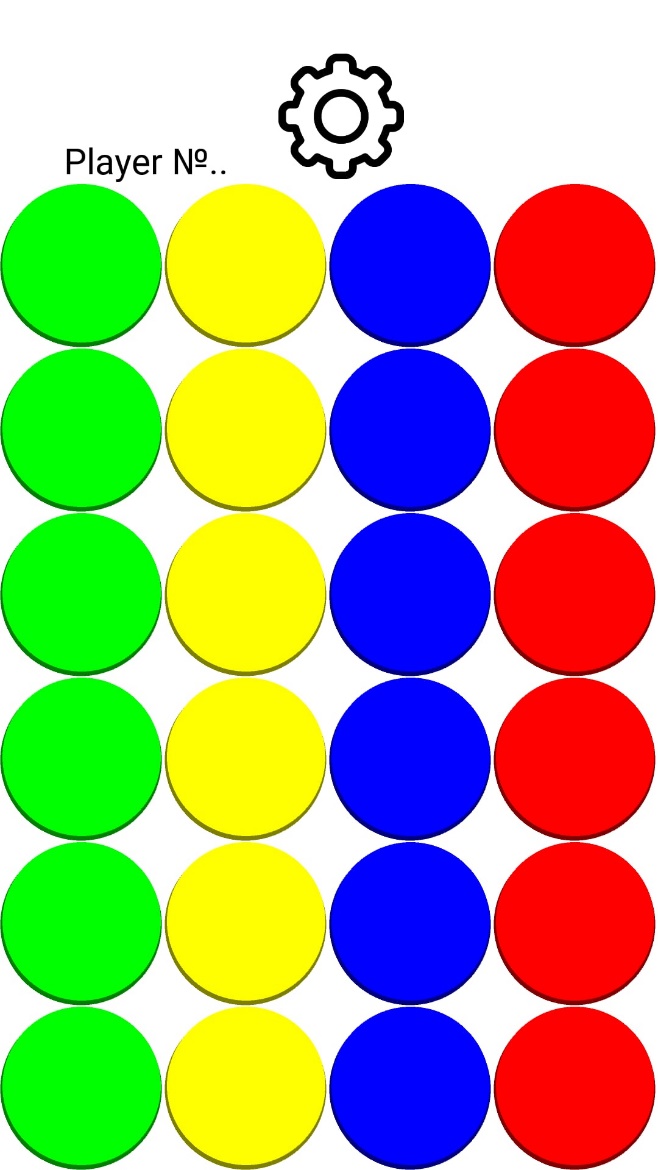


Рис. 1. Начальный экран приложения

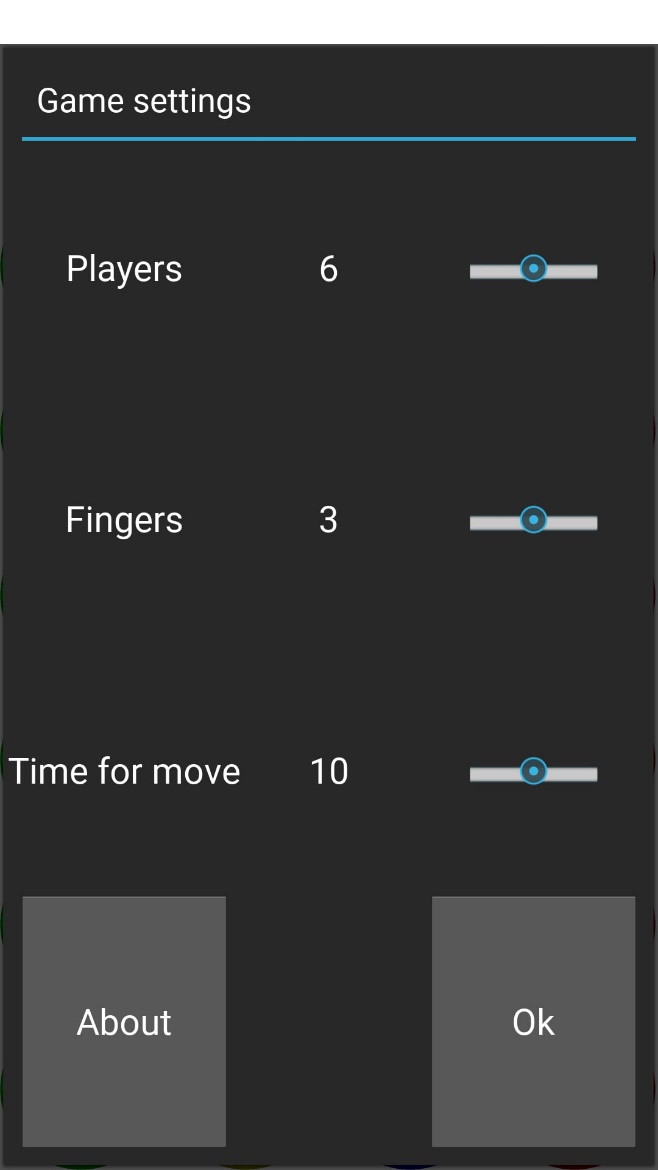


Рис. 2. Окно настроек игры

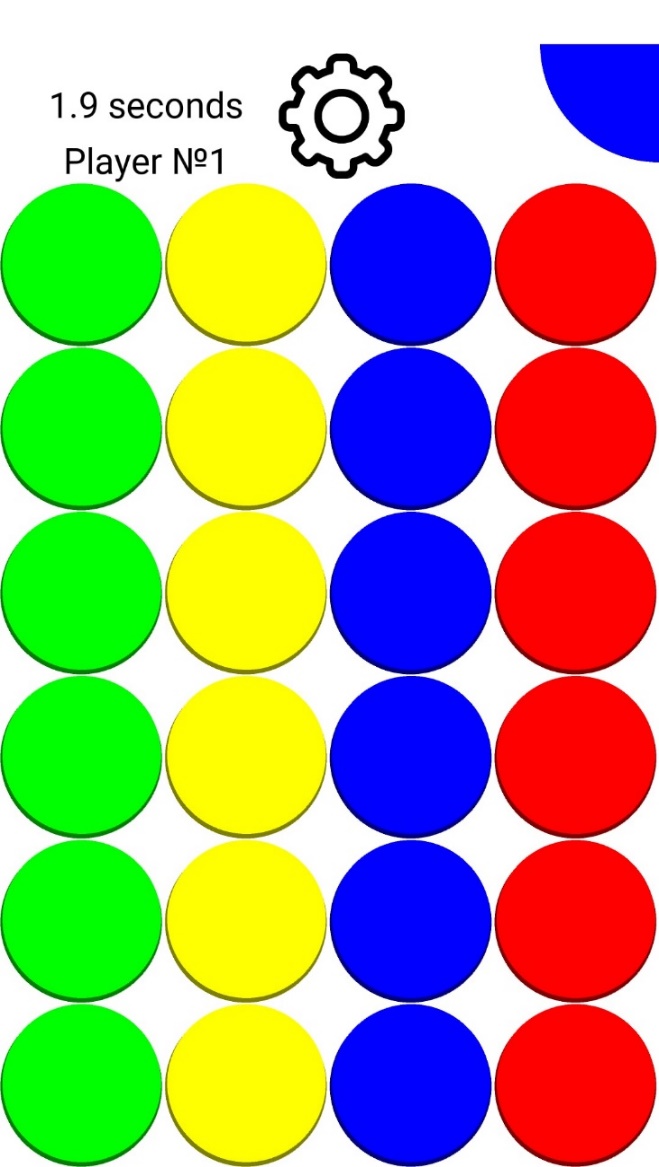


Рис. 3. Главный экран во время игры

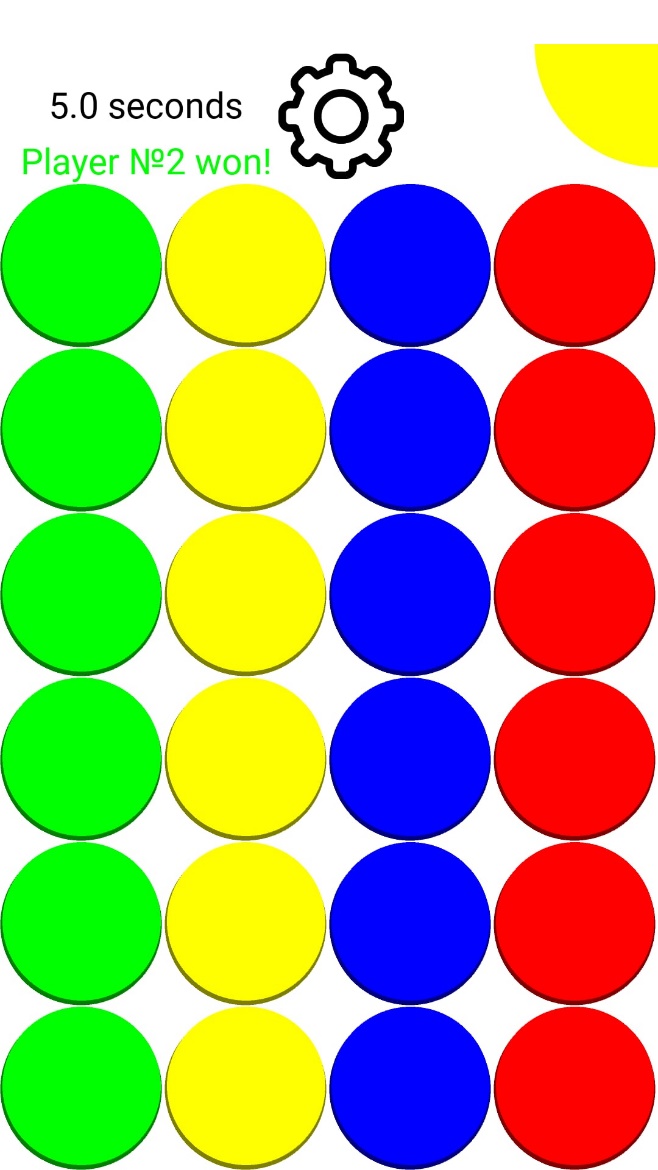


Рис. 4. Объявление победителя

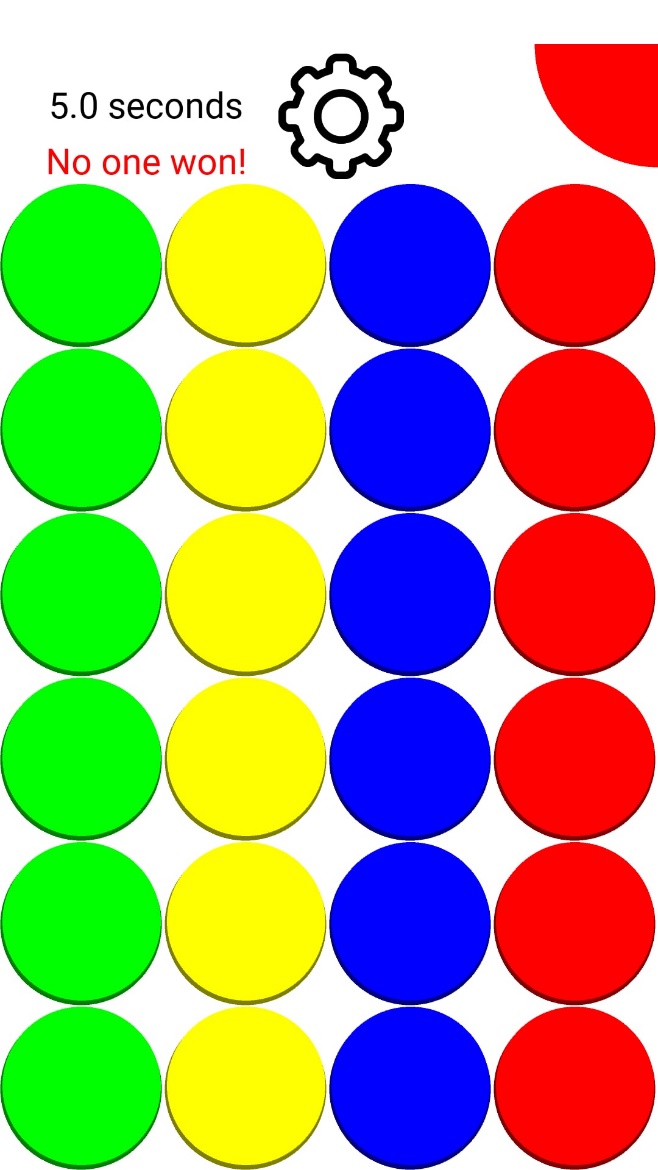


Рис. 5. Отсутствие победителей

# Список использованных источников

1. kivy.org – официальный сайт фреймворка, на котором можно найти инструкции по установке и использованию при разработке;
2. stackoverflow.com – поиск ответов на возникающие вопросы.

# Приложение

main.py

import random  
  
from kivy.animation import Animation  
from kivy.app import App  
from kivy.core.window import Window  
from kivy.graphics import Color, Ellipse  
from kivy.properties import NumericProperty  
from kivy.uix.boxlayout import BoxLayout  
from kivy.uix.button import Button  
from kivy.uix.gridlayout import GridLayout  
from kivy.uix.label import Label  
from kivy.uix.popup import Popup  
from kivy.uix.slider import Slider  
from kivy.uix.widget import Widget  
  
# Определение размера для размещения секций игрового поля  
step = Window.size[0] // 4  
  
  
# Метод для определения секции (ряд и цвет), с которой произошло взаимодействие  
def define\_circle(pos):  
 res = []  
 # Определение ряда  
 if 0 <= pos[1] <= step:  
 res.append("1")  
 elif step <= pos[1] <= 2 \* step:  
 res.append("2")  
 elif 2 \* step <= pos[1] <= 3 \* step:  
 res.append("3")  
 elif 3 \* step <= pos[1] <= 4 \* step:  
 res.append("4")  
 elif 4 \* step <= pos[1] <= 5 \* step:  
 res.append("5")  
 elif 5 \* step <= pos[1] <= 6 \* step:  
 res.append("6")  
 # Определение цвета  
 if 0 <= pos[0] <= step:  
 res.append("G")  
 elif step <= pos[0] <= 2 \* step:  
 res.append("Y")  
 elif 2 \* step <= pos[0] <= 3 \* step:  
 res.append("B")  
 elif 3 \* step <= pos[0] <= Window.size[0]:  
 res.append("R")  
 return ''.join(res)  
  
  
# Класс, отвечающей за отрисовку игрового поля и подсчет параметров игры (число оставшихся игроков, позиция хода,  
# позиции, занятые игроками, количество свободных секций каждого отдельного цвета, которое понадобится для рулетки)  
class FieldWidget(Widget):  
 # Позиции игроков на поле  
 gamer\_position = [[]]  
 # Позиция данного хода игрока  
 move\_position = ''  
 # Счетчик невыбывших игроков  
 gamer\_count = 0  
 # Счетчики занятых полей отдельных цветов, необходимые для определения следующего хода  
 r\_count = 0  
 g\_count = 0  
 y\_count = 0  
 b\_count = 0  
  
 # Начальная отрисовка игрового поля  
 def \_\_init\_\_(self, \*\*kwargs):  
 super(FieldWidget, self).\_\_init\_\_(\*\*kwargs)  
 with self.canvas:  
 for y in range(0, 6 \* step, step):  
 # Зеленый круг  
 Color(0, 0.5, 0)  
 Ellipse(pos=(0.5, y + 0.5), size\_hint=(None, None), size=(step - 5, step - 5))  
 Color(0, 1, 0)  
 Ellipse(pos=(4, y + 8), size\_hint=(None, None), size=(step - 10, step - 10))  
 # Желтый круг  
 Color(0.5, 0.5, 0)  
 Ellipse(pos=(step + 0.5, y + 0.5), size\_hint=(None, None), size=(step - 5, step - 5))  
 Color(1, 1, 0)  
 Ellipse(pos=(step + 4, y + 8), size\_hint=(None, None), size=(step - 10, step - 10))  
 # Синий круг  
 Color(0, 0, 0.5)  
 Ellipse(pos=(2 \* step + 0.5, y + 0.5), size\_hint=(None, None), size=(step - 5, step - 5))  
 Color(0, 0, 1)  
 Ellipse(pos=(2 \* step + 4, y + 8), size\_hint=(None, None), size=(step - 10, step - 10))  
 # Красный круг  
 Color(0.5, 0, 0)  
 Ellipse(pos=(3 \* step + 0.5, y + 0.5), size\_hint=(None, None), size=(step - 5, step - 5))  
 Color(1, 0, 0)  
 Ellipse(pos=(3 \* step + 4, y + 8), size\_hint=(None, None), size=(step - 10, step - 10))  
  
 # Обновление поля в результате нажатия, отрисовка зажатых секций  
 def update\_field(self, position, upd):  
 with self.canvas.after:  
 if len(position) == 2:  
 if position[1] == "R":  
 if upd:  
 Color(1, 1, 1)  
 Ellipse(pos=(3 \* step + 4, step \* (int(position[0]) - 1) + 8), size\_hint=(None, None),  
 size=(step - 10, step - 10))  
 Color(1, 0, 0)  
 Ellipse(pos=(3 \* step + 0.5, step \* (int(position[0]) - 1) + 0.5), size\_hint=(None, None),  
 size=(step - 5, step - 5))  
 else:  
 Color(0.5, 0, 0)  
 Ellipse(pos=(3 \* step + 0.5, step \* (int(position[0]) - 1) + 0.5), size\_hint=(None, None),  
 size=(step - 5, step - 5))  
 Color(1, 0, 0)  
 Ellipse(pos=(3 \* step + 4, step \* (int(position[0]) - 1) + 8), size\_hint=(None, None),  
 size=(step - 10, step - 10))  
 if position[1] == "G":  
 if upd:  
 Color(1, 1, 1)  
 Ellipse(pos=(4, step \* (int(position[0]) - 1) + 8), size\_hint=(None, None),  
 size=(step - 10, step - 10))  
 Color(0, 1, 0)  
 Ellipse(pos=(0.5, step \* (int(position[0]) - 1) + 0.5), size\_hint=(None, None),  
 size=(step - 5, step - 5))  
 else:  
 Color(0, 0.5, 0)  
 Ellipse(pos=(0.5, step \* (int(position[0]) - 1) + 0.5), size\_hint=(None, None),  
 size=(step - 5, step - 5))  
 Color(0, 1, 0)  
 Ellipse(pos=(4, step \* (int(position[0]) - 1) + 8), size\_hint=(None, None),  
 size=(step - 10, step - 10))  
 if position[1] == "B":  
 if upd:  
 Color(1, 1, 1)  
 Ellipse(pos=(2 \* step + 4, step \* (int(position[0]) - 1) + 8), size\_hint=(None, None),  
 size=(step - 10, step - 10))  
 Color(0, 0, 1)  
 Ellipse(pos=(2 \* step + 0.5, step \* (int(position[0]) - 1) + 0.5), size\_hint=(None, None),  
 size=(step - 5, step - 5))  
 else:  
 Color(0, 0, 0.5)  
 Ellipse(pos=(2 \* step + 0.5, step \* (int(position[0]) - 1) + 0.5), size\_hint=(None, None),  
 size=(step - 5, step - 5))  
 Color(0, 0, 1)  
 Ellipse(pos=(2 \* step + 4, step \* (int(position[0]) - 1) + 8), size\_hint=(None, None),  
 size=(step - 10, step - 10))  
 if position[1] == "Y":  
 if upd:  
 Color(1, 1, 1)  
 Ellipse(pos=(step + 4, step \* (int(position[0]) - 1) + 8), size\_hint=(None, None),  
 size=(step - 10, step - 10))  
 Color(1, 1, 0)  
 Ellipse(pos=(step + 0.5, step \* (int(position[0]) - 1) + 0.5), size\_hint=(None, None),  
 size=(step - 5, step - 5))  
 else:  
 Color(0.5, 0.5, 0)  
 Ellipse(pos=(step + 0.5, step \* (int(position[0]) - 1) + 0.5), size\_hint=(None, None),  
 size=(step - 5, step - 5))  
 Color(1, 1, 0)  
 Ellipse(pos=(step + 4, step \* (int(position[0]) - 1) + 8), size\_hint=(None, None),  
 size=(step - 10, step - 10))  
  
 # Обработка нажатия на экран (определение положения, игрока и возможного выбывания игрока по причинам, описываемым  
 # в правилах)  
 def on\_touch\_down(self, touch):  
 # Определение положения нажатия  
 self.move\_position = define\_circle(touch.pos)  
 # Определение, идет ли в данный момент игра  
 if player\_move.text[-1] != "." and player\_move.text[-1] != "!":  
 # Определение номера игрока, который ходит в данный момент  
 i = int(player\_move.text[-1])  
 # Если игрок не выбыл из игры  
 if self.gamer\_position[i - 1] != "LOSER":  
 # Если игрок разместил не все пальцы на игровом поле, его ход записывается в соответствующую ячейку  
 # массива  
 if '' in self.gamer\_position[i - 1]:  
 self.gamer\_position[i - 1][self.gamer\_position[i - 1].index('')] = self.move\_position  
 else:  
 self.gamer\_position[i - 1][0] = "LOSER"  
 self.gamer\_count -= 1  
 # Определение цвета занятой в результате нажатия ячейки игрового поля  
 if len(self.move\_position) == 2:  
 if self.move\_position[1] == "R":  
 self.r\_count += 1  
 if self.move\_position[1] == "G":  
 self.g\_count += 1  
 if self.move\_position[1] == "B":  
 self.b\_count += 1  
 if self.move\_position[1] == "Y":  
 self.y\_count += 1  
 # Обновление игрового поля после нажатия  
 self.update\_field(self.move\_position, True)  
  
 # Обработка отпускания пальца (определение положения, игрока и возможного выбывания игрока по причинам,  
 # описываемым в правилах игры)  
 def on\_touch\_up(self, touch):  
 # Определение положения отпускания  
 remove\_position = define\_circle(touch.pos)  
 # Определение, идет ли в данный момент игра  
 if player\_move.text[-1] != "." and player\_move.text[-1] != "!":  
 if len(remove\_position) == 2:  
 for player in self.gamer\_position:  
 # Определение игрока, отпустившего палец по положению отпускания  
 if remove\_position in player:  
 # Если игрок на момент отпускания разместил не все пальцы на поле и при этом позиция  
 # отпускания не соответствует позиции нажатия или же в данный момент ходит не этот игрок,  
 # то игрок выбывает из игры  
 if "" in player and remove\_position != self.move\_position or (  
 self.gamer\_position.index(player) + 1) != \  
 int(player\_move.text[-1]):  
 player[0] = "LOSER"  
 self.gamer\_count -= 1  
 # В противном случае удаляем из массива ходов игрока запись о ходе в данной позиции  
 else:  
 player[player.index(remove\_position)] = ''  
 # Определение цвета освобожденной в результате отпускания ячейки игрового поля  
 if remove\_position[1] == "R":  
 self.r\_count -= 1  
 if remove\_position[1] == "G":  
 self.g\_count -= 1  
 if remove\_position[1] == "B":  
 self.b\_count -= 1  
 if remove\_position[1] == "Y":  
 self.y\_count -= 1  
 # Обновление игрового поля после отпускания  
 self.update\_field(remove\_position, False)  
  
  
# Класс, отображающий рулетку на игровом поле  
class RouletteWidget(Widget):  
 # Цвет рулетки  
 roulette\_color = Color(1, 1, 1, 1)  
  
 def \_\_init\_\_(self, \*\*kwargs):  
 super(RouletteWidget, self).\_\_init\_\_(\*\*kwargs)  
 with self.canvas:  
 # Перед началом игры цвет рулетки белый  
 Color(1, 1, 1, 1)  
 # Рулетка представляет собой окружность, расположенную в правом верхнем углу экрана  
 self.roulette = Ellipse(  
 pos=(Window.size[0] - 0.75 \* step, Window.size[1] - 0.75 \* step),  
 size=(1.5 \* step, 1.5 \* step))  
  
 # Метод, отвечающий за смену цветов рулетки  
 def change\_color(self):  
 c = self.canvas  
 c.clear()  
 with c:  
 self.roulette\_color = random.choice([(1, 0, 0, 1), (0, 1, 0, 1), (0, 0, 1, 1), (1, 1, 0, 1)])  
 while True:  
 # Определяем, есть ли у игрока возможность поставить палец на секцию указанного цвета (свободна ли  
 # ячейка указанного цвета)  
 if self.roulette\_color == (1, 0, 0, 1) and field.r\_count != 6 or self.roulette\_color == (  
 1, 1, 0, 1) and field.y\_count != 6 or self.roulette\_color == (  
 0, 1, 0, 1) and field.g\_count != 6 or self.roulette\_color == (  
 0, 0, 1, 1) and field.b\_count != 6:  
 break  
 # Если все ячейки указанного цвета заняты, то выбираем другой цвет  
 else:  
 self.roulette\_color = random.choice([(1, 0, 0, 1), (0, 1, 0, 1), (0, 0, 1, 1), (1, 1, 0, 1)])  
 Color(rgba=self.roulette\_color)  
 # Перерисовываем окружность на той же позиции, но другого цвета  
 self.roulette = Ellipse(  
 pos=(Window.size[0] - 0.75 \* step, Window.size[1] - 0.75 \* step),  
 size=(1.5 \* step, 1.5 \* step))  
  
  
# Класс, описывающий таймер хода на игровом поле  
class MoveTimer(Label):  
 # Длительность хода, задаваемая в настройках игры (изначально используется значение по умолчанию)  
 a = NumericProperty(10)  
 # Число игроков (изначально используется значение по умолчанию)  
 players = NumericProperty(3)  
  
 # Метод, описывающей начало анимации таймера  
 def start(self):  
 Animation.cancel\_all(self)  
 # Анимация таймера состоит из двух последовательных частей (отсчет времени, равного длительности хода,  
 # и возвращение к значению, равному длительности следующего хода)  
 self.anim = Animation(a=0, duration=self.a) + Animation(a=self.a, duration=0)  
  
 # Метод, отвечающий за поиск следующего невыбывшего игрока  
 def find\_next(i):  
 while True:  
 i += 1  
 if i > self.players:  
 i = 1  
 # Если игрок не выбыл, то он будет ходить следующим  
 if field.gamer\_position[i - 1][0] != "LOSER":  
 break  
 return i  
  
 # Метод, отвечающий за действия после окончания хода (определение выбывания игрока, смена цвета рулетки,  
 # поиск следующего игрока)  
 def finish\_callback(\*args):  
 # Определение номера игрока, сыгравшего данный ход  
 i = int(player\_move.text[-1])  
 # Если игрок нажал на окружность цвета, отличного от указанного на рулетке, то он выбывает из игры  
 if len(field.move\_position) == 2 and (not (  
 field.move\_position[1] == "B" and roulette.roulette\_color == (0, 0, 1, 1) or field.move\_position[  
 1] == "R" and roulette.roulette\_color == (1, 0, 0, 1) or field.move\_position[  
 1] == "Y" and roulette.roulette\_color == (  
 1, 1, 0, 1) or field.move\_position[1] == "G" and roulette.roulette\_color == (  
 0, 1, 0, 1)) or field.move\_position not in field.gamer\_position[i - 1]):  
 field.gamer\_position[i - 1][0] = "LOSER"  
 field.gamer\_count -= 1  
 # Если в данный момент идет игра и игрок не нажал на игровое поле во время своего хода, то он выбывает  
 if len(field.move\_position) < 2 and i != 0:  
 field.gamer\_position[i - 1][0] = "LOSER"  
 field.gamer\_count -= 1  
 # Если осталось меньше двух невыбывших игроков  
 if field.gamer\_count < 2:  
 # Если остался один невыбывших игрок, определяется его номер, отображается сообщение о том,  
 # что этот игрок победил, и игра останавливается  
 if field.gamer\_count == 1:  
 for player in field.gamer\_position:  
 # Если данный игрок не выбыл из игры  
 if "LOSER" not in player:  
 # Вывод сообщения о победе игрока в текстовом поле, содержащем информацию о состоянии  
 # игры, зеленым цветом и остановка таймера игры  
 player\_move.text = f"Player №{field.gamer\_position.index(player) + 1} won!"  
 player\_move.color = (0, 1, 0, 1)  
 timer.anim.stop(timer)  
 # Если на поле не осталось невыбывших игроков, отображается сообщение о том, что победителей нет, и игра  
 # останавливается  
 if field.gamer\_count < 1:  
 # Вывод сообщения о поражении всех игроков в текстовом поле, содержащем информацию о состоянии  
 # игры, красным цветом и остановка таймера игры  
 player\_move.text = "No one won!"  
 player\_move.color = (1, 0, 0, 1)  
 timer.anim.stop(timer)  
 # Если невыбывших игроков больше одного, то цвет рулетки изменяется и отображается сообщение с номером  
 # игрока, который ходит следующим (при этом счетчик заново отсчитывает время хода игрока)  
 else:  
 roulette.change\_color()  
 player\_move.text = f"Player №{find\_next(i)}"  
  
 # Задание повторения последовательной анимации  
 self.anim.repeat = True  
 self.anim.bind(on\_start=finish\_callback)  
 self.anim.start(self)  
  
 # Метод, отвечающий за отображение таймера на игровом поле  
 def on\_a(self, instance, value):  
 self.text = f"{round(value, 1)} seconds"  
  
  
# Создание игрового поля  
field = FieldWidget(size\_hint\_y=0.875)  
# Создание таймера  
timer = MoveTimer(color=(0, 0, 0, 1))  
# Создание рулетки  
roulette = RouletteWidget()  
# Добавление текстового поля с сообщениями о состоянии игры  
player\_move = Label(text="Player №..", color=(0, 0, 0, 1))  
  
  
# Класс, отвечающий за отрисовку главного экрана приложения  
class TwisterApp(App):  
 def build(self):  
 # Фон приложения - белый  
 Window.clearcolor = (1, 1, 1, 1)  
 # Размещение виджетов по вертикали  
 screen = BoxLayout(orientation="vertical")  
 # Отступ для лучшего отображения информационных виджетов  
 screen.add\_widget(Widget(size\_hint\_y=0.03))  
 # Создание горизонтальной панели информационных виджетов  
 info = BoxLayout(orientation="horizontal", size\_hint\_y=0.1)  
 # Левая часть информационной панели состоит из 2 вертикально расположенных виджетов: таймера хода и  
 # текстового поля с сообщениями о состоянии игры  
 move = BoxLayout(orientation="vertical")  
 # Таймер хода  
 move.add\_widget(timer)  
 # Текстовое поле с сообщениями о состоянии игры  
 move.add\_widget(player\_move)  
 # Отступ для лучшего отображения текстового поля  
 info.add\_widget(Widget(size\_hint\_x=0.1))  
 # Добавление панели информационных виджетов к горизонтальной панели в верхней части экрана  
 info.add\_widget(move)  
 # Создание кнопки включения настроек, среди параметров которой изображение кнопки в двух состояниях (обычном  
 # и нажатом), размеры кнопки и метод, вызываемый при нажатии  
 settings = Button(background\_normal="settings.png", background\_down="settingsdown.png", size\_hint=(None, None),  
 size=(step - 30, step - 30), on\_press=lambda x: show\_settings())  
 # Добавление кнопки к горизонтальной панели в верхней части экрана  
 info.add\_widget(settings)  
 # Добавление рулетки к горизонтальной панели в верхней части экрана  
 info.add\_widget(roulette)  
 # Добавление панели информационных виджетов в верхнюю часть экрана приложения  
 screen.add\_widget(info)  
 # Добавление игрового поля к главному экрану приложения  
 screen.add\_widget(field)  
 # Отображение главного экрана приложения  
 return screen  
  
  
# Метод, отвечающий за отображение всплывающего окна информации об игре  
def show\_info():  
 # Содержимое всплывающего окна расположено в таблице из двух вертикально расположенных ячеек  
 content = GridLayout(rows=2)  
 # Создание текстового поля с информацией об игре, среди параметров которого размеры виджета, размеры текста,  
 # использование разметки и размер шрифта  
 about = Label(  
 text="This is a Twister game for Android.\nThe rules of the game are similar to the rules of the original "  
 "[i]Twister[/i]. On the playing field there are special sections (4 rows of 6 circles) of different "  
 "colors, "  
 "on one of which the player puts his finger for a limited time at the direction of the roulette and "  
 "holds it until the end of the game or a change of position as a result of the next scrolling of the "  
 "roulette.\nThe game is played by 2 to 10 players, each of whom can put on the field from 1 to 5 fingers "  
 "of one hand, the turn time is set from 5 to 15 seconds (these game settings are located in a separate "  
 "menu). The winner is the last player remaining in the field.\n[b]A player is eliminated from the game if "  
 "he[/b]: "  
 "does not put his finger in the specified place during his turn, removes his finger not in his turn, "  
 "regardless of the move removes his finger, provided that not all fingers are on the field.",  
 size\_hint=(0.9, 0.9), text\_size=(Window.size[0] - 15, None), markup=True, font\_size="13sp")  
 # Создание кнопки, отвечающая за закрытие всплывающего окна  
 ok = Button(text="Ok", size\_hint=(1, 0.1))  
 # Добавление текстового поля  
 content.add\_widget(about)  
 # Добавление кнопки  
 content.add\_widget(ok)  
 # Создание всплывающего окна, среди параметров которого название окна и содержимое, созданное выше  
 popup = Popup(title="Game info", content=content, auto\_dismiss=False)  
 # Добавление к кнопке опции закрытия всплывающего окна  
 ok.bind(on\_press=popup.dismiss)  
 # Открытие всплывающего окна  
 popup.open()  
  
  
# Метод, отвечающий за отображение всплывающего окна настроек игры  
def show\_settings():  
 # Содержимое всплывающего окна расположено в таблице, содержащей три колонки  
 content = GridLayout(cols=3)  
 # Добавление регулируемых настроек игрового процесса: количества игроков, количества пальцев у каждого игрока и  
 # количество времени на каждый ход  
 content.add\_widget(Label(text="Players"))  
 # Значение количества игроков по умолчанию  
 players\_label = Label(text="6")  
 # Создание ползунка для настройки количества игроков с возможностью регулировки от 2 до 10  
 players\_slider = Slider(min=2, max=10, value=6, step=1, cursor\_size=(75, 75))  
 content.add\_widget(players\_label)  
 content.add\_widget(players\_slider)  
 content.add\_widget(Label(text="Fingers"))  
 # Значение количества пальцев по умолчанию  
 fingers\_label = Label(text="3")  
 # Создание ползунка для настройки количества пальцев с возможностью регулировки от 1 до 5  
 fingers\_slider = Slider(min=1, max=5, value=3, step=1, cursor\_size=(75, 75))  
 content.add\_widget(fingers\_label)  
 content.add\_widget(fingers\_slider)  
 content.add\_widget(Label(text="Time for move"))  
 # Значение времени на один ход по умолчанию  
 time\_label = Label(text="10")  
 # Создание ползунка для настройки времени на один ход с возможностью переключения между 5, 10 и 15 секундами  
 time\_slider = Slider(min=5, max=15, value=10, step=5, cursor\_size=(75, 75))  
 content.add\_widget(time\_label)  
 content.add\_widget(time\_slider)  
  
 # Методы, отвечающие за отображение значения ползунка в текстовом поле слева от ползунка  
 def on\_p\_slider\_value\_change(instance, value):  
 players\_label.text = str(value)  
  
 def on\_f\_slider\_value\_change(instance, value):  
 fingers\_label.text = str(value)  
  
 def on\_t\_slider\_value\_change(instance, value):  
 time\_label.text = str(value)  
  
 players\_slider.bind(value=on\_p\_slider\_value\_change)  
 fingers\_slider.bind(value=on\_f\_slider\_value\_change)  
 time\_slider.bind(value=on\_t\_slider\_value\_change)  
  
 # Добавление кнопки, отвечающей за открытие всплывающего окна с информацией об игре  
 content.add\_widget(Button(text="About", on\_press=lambda x: show\_info()))  
 content.add\_widget(Widget())  
 # Создание кнопки, отвечающей за закрытие всплывающего окна настройки игры  
 ok = Button(text="Ok")  
 content.add\_widget(ok)  
 # Создание всплывающего окна, среди параметров которого название окна и содержимое, созданное выше  
 popup = Popup(title="Game settings", content=content, auto\_dismiss=False)  
 # Добавление к кнопке опции закрытия всплывающего окна  
 ok.bind(on\_press=popup.dismiss)  
  
 # Изменение значений параметров после их задания во всплывающем меню настроек  
 def change\_values(\*args):  
 # Задание нового значения для таймера хода  
 timer.a = time\_slider.value  
 # Задание нового значения количества игроков  
 timer.players = players\_slider.value  
 # Изменение цвета рулетки  
 roulette.change\_color()  
 # Инициализация массива ходов игроков размерности (количество игроков x количество пальцев)  
 field.gamer\_position = [["" for x in range(fingers\_slider.value)] for y in range(players\_slider.value)]  
 # Изменение текстового поля с сообщением о состоянии игры, которое изменится сразу после начала игры  
 player\_move.text = "Player №0"  
 # Изменение цвета текстового поля  
 player\_move.color = (0, 0, 0, 1)  
 # Задание значения количества невыбывших игроков  
 field.gamer\_count = players\_slider.value  
 # Задание значения свободных ячеек различных цветов на момент начала игры  
 field.r\_count = 0  
 field.g\_count = 0  
 field.y\_count = 0  
 field.b\_count = 0  
 # Начало игры  
 timer.start()  
  
 # Добавление к всплывающему окну метода, приведенного выше, который будет вызываться при закрытии всплывающего окна  
 popup.bind(on\_dismiss=change\_values)  
 # Открытие всплывающего окна  
 popup.open()  
  
  
# Запуск приложения  
if \_\_name\_\_ == "\_\_main\_\_":  
 TwisterApp().run()