Министерство науки и высшего образования Российской Федерации

Федеральное государственное автономное образовательное учреждение

высшего образования

«Северо-Восточный Федеральный университет имени М.К.Аммосова»

Институт математики и информатики

Кафедра математической экономики и прикладной информатики

РАЗРАБОТКА ИГРЫ НА PYTHON С ИСПОЛЬЗОВАНИЕМ ФРЕЙМВОРКА KIVY

Направление подготовки 09.04.03 Прикладная информатика

Профиль: Прикладная информатика в юриспруденции

Выполнил: студент 2 курса

группы М-ПИЮ-20 ИМИ СВФУ Сыбахов Марк Николаевич

г. Якутск, 2021 г.

ВВЕДЕНИЕ

Python - высокоуровневый язык программирования общего назначения с динамической строгой типизацией и автоматическим управлением памятью, ориентированный на повышение производительности разработчика, читаемости кода и его качества, а также на обеспечение переносимости написанных на нём программ. Язык является полностью объектно-ориентированным в том плане, что всё является объектами. Необычной особенностью языка является выделение блоков кода пробельными отступами. Синтаксис ядра языка минималистичен, за счёт чего на практике редко возникает необходимость обращаться к документации. Сам же язык известен как интерпретируемый и используется в том числе для написания скриптов. Недостатками языка являются зачастую более низкая скорость работы и более высокое потребление памяти написанных на нём программ по сравнению с аналогичным кодом, написанным на компилируемых языках, таких как Си или C++.

Фреймворк Kivy разрабатывается с 2011 года. С момента релиза на его основе было построено много проектов под платформу Андирод.

Если сравнивать Kivy с другими библиотеками языка Питон по набору функций, то среди крупных конкурентов можно выделить PyQT. Все прочие библиотеки явно будут уступать по функционалу.

УСТАНОВКА KIVY

Самый простой способ установить Kiv - это установить через Kivy pip с помощью предварительно скомпилированного колеса, если оно доступно, или из исходного кода.

Kivy предоставляет предварительно скомпилированные колеса для поддерживаемых версий Python в Windows, OS X, Linux и RPi. В качестве альтернативы установка из исходного кода требуется для более новых версий Python, не указанных выше, или если колеса не работают или не работают должным образом.

Запускаем терминал (запуск от имени администратора)

1. python -m pip install --upgrade pip setuptools virtualenv

это функция для установки pip

1. python -m virtualenv kivy\_venv

создаем виртуальную среду kivy\_venv

1. kivy\_venv\Scripts\activate

активируем виртуальную среду

Затем создаем папку под ваш проект. На этом папке создаем файл main.py (основной код проекта) и pong.kv(графический (интерфейс) код проекта).

Для запуска проекта вводим в терминале: python main.py

СОЗДАНИЕ ПРОЕКТА

Наш проект называется Пинг-понг – это игра для двух игроков разработанный на языке программирования python kivy.

В файле main.py пишем код проекта:

Kivy это фреймворк pythonа поэтому первыми кодами будут импорты библиотек kivy

1. **from** kivy.app **import** App ##
2. **from** kivy.uix.widget **import** Widget
3. **from** kivy.properties **import** (
4. NumericProperty, ReferenceListProperty, ObjectProperty)
5. **from** kivy.vector **import** Vector
6. **from** kivy.clock **import** Clock
7. **from** random **import** randint

Создаем главный интерфейс игры Class PongPaddle

1. **class** PongPaddle(Widget):
2. score = NumericProperty(0) *## очки игрока*
4. *## Отскок мячика при коллизии с панелькой игрока*
5. **def** bounce\_ball(self, ball):
6. **if** self.collide\_widget(ball):
7. vx, vy = ball.velocity
8. offset = (ball.center\_y - self.center\_y) / (self.height / 2)
9. bounced = Vector(-1 \* vx, vy)
10. vel = bounced \* 1.1
12. ball.velocity = vel.x, vel.y + offset

Создаем интерфейс шарика Class PongBall

1. **class** PongBall(Widget):
3. *# Скорость движения нашего шарика по двум осям*
4. velocity\_x = NumericProperty(0)
5. velocity\_y = NumericProperty(0)
7. *# Создаем условный вектор*
8. velocity = ReferenceListProperty(velocity\_x, velocity\_y)
10. *# Заставим шарик двигаться*
11. **def** move(self):
12. self.pos = Vector(\*self.velocity) + self.pos

Создаем логику игры Class PongGame

1. **class** PongGame(Widget):
2. ball = ObjectProperty(None) *# это будет наша связь с объектом шарика*
3. player1 = ObjectProperty(None) *# Игрок 1*
4. player2 = ObjectProperty(None) *# Игрок 2*
6. **def** serve\_ball(self, vel=(4, 0)):
7. self.ball.center = self.center
8. self.ball.velocity = Vector(vel[0], vel[1]).rotate(randint(0, 360))
10. **def** update(self, dt):
11. self.ball.move() *# двигаем шарик в каждом обновлении экрана*
13. *# проверка отскока шарика от панелек игроков*
14. self.player1.bounce\_ball(self.ball)
15. self.player2.bounce\_ball(self.ball)
17. *# отскок шарика по оси Y*
18. **if**(self.ball.y < 0) **or** (self.ball.top > self.height):
19. self.ball.velocity\_y \*= -1 *# инверсируем текущую скорость по оси Y*
21. *# отскок шарика по оси X*
22. *# тут если шарик смог уйти за панельку игрока, то есть игрок не успел отбить шарик*
23. *# то это значит что он проиграл и мы добавим +1 очко противнику*
24. **if** self.ball.x < self.x:
25. *# Первый игрок проиграл, добавляем 1 очко второму игроку*
26. self.player2.score += 1
27. self.serve\_ball(vel=(4,0)) *# заново спавним шарик в центре*
29. **if** self.ball.x > self.width:
30. *# Второй игрок проиграл, добавляем 1 очко первому игроку*
31. self.player1.score += 1
32. self.serve\_ball(vel=(-4,0)) *# заново спавним шарик в центре*
34. *# Событие прикосновения к экрану*
35. **def** on\_touch\_move(self, touch):
36. *# первый игрок может касаться только своей части экрана (левой)*
37. **if** touch.x < self.width / 3:
38. self.player1.center\_y = touch.y
40. *# второй игрок может касаться только своей части экрана (правой)*
41. **if** touch.x > self.width - self.width / 3:
42. self.player2.center\_y = touch.y
44. **class** PongApp(App):
45. **def** build(self):
46. game = PongGame()
47. game.serve\_ball()
48. Clock.schedule\_interval(game.update, 1.0/60) *# 60 FPS*
49. **return** game
51. **if** \_\_name\_\_ == '\_\_main\_\_':
52. PongApp().run()

В файле pong.kv пишем код графического интерфейса:

1. #:kivy 1.11.1

1. # Тут нарисуем наш шарик, называем этот объект точно также как и класс в коде
2. <PongBall>:
   1. size: 50, 50
4. canvas:
   1. Ellipse:
      1. pos: self.pos
      2. size: self.size

1. # Тут нарисуем панельку игрока, называем этот объект точно также как и класс в коде
2. <PongPaddle>:
   1. size: 25, 200
   2. canvas:
      1. Rectangle:
         1. pos: self.pos
         2. size: self.size
3. # А это игровое поле =)
4. <PongGame>:
5. ## тут привязываем шарик к свойству ball
   1. ball: pong\_ball
   2. player1: player\_left
   3. player2: player\_right

1. canvas:
   1. Rectangle:
   2. ## Бордюр посередине
      1. pos: self.center\_x - 5, 0
      2. size: 10, self.height

1. Label:
   1. ## Очки игрока слева
   2. font\_size: 70
   3. center\_x: root.width / 4
   4. top: root.top - 50
   5. text: str(root.player1.score)
3. Label:
   1. ## Очки игрока справа
   2. font\_size: 70
   3. center\_x: root.width \* 3 / 4
   4. top: root.top - 50
   5. text: str(root.player2.score)
5. ## а тут создаем экземпляр нашего шарика в игровом поле
6. PongBall:
   1. id: pong\_ball
   2. center: self.parent.center
8. ## создаем игрока 1
9. PongPaddle:
   1. id: player\_left
   2. x: root.x
   3. center\_y: root.center\_y
11. ## создаем игрока 2
12. PongPaddle:
    1. id: player\_right
    2. x: root.width - self.width
    3. center\_y: root.center\_y

Скрин проекта:

