Теория

В данной лабораторной работе предлагается ознакомиться с библиотекой Pygame для 2d графики, анимации, игр и т.п. Начнем с примера. Рассмотрим следующее простенькое приложение на Pygame:

```
import sys
import pygame

pygame.init()

width = 500
height = 500

screen = pygame.display.set_mode((width, height))
pygame.display.set_caption('YAHOOOO')
clock = pygame.time.Clock()

x = 30
y = 30
vx = 50
vy = 50

while True:
    dt = clock.tick(50) / 1000.0

    for event in pygame.event.get():
        if event.type == pygame.QUIT or event.type == pygame.KEYDOWN:
            sys.exit()

    x += vx * dt
    y += vy * dt

    screen.fill((0, 0, 0))
    pygame.draw.circle(screen, (150, 10, 50), (int(x), int(y)), 20)

    pygame.display.flip()
```

Если запустить этот код, мы увидим кружок, двигающийся с постоянной скоростью.

Разберемся что здесь происходит. Импортировали модуль pygame. Что бы работать с этой библиотекой нужно позвать pygame.init() в начале. Далее инициализируем окно. Обратите внимание, функции pygame.disaplay.set_mode() в качестве параметра передается кортеж из ширины и высоты окна. Создали часы (clock), этот объект поможет нам считать время между кадрами анимации и контролировать FPS.

Далее идет вечный цикл, в котором мы будем отрисовывать наши кадры и обрабатывать события от пользователя (нажатия кнопок, например). Что же мы делаем в цикле?

В первую очередь, зовем метод tick() наших часов. Возвращаемое значение - время в миллисекундах, прошедшее от предыдущего кадра (т.е., на самом деле от предыдущего вызова .tick()). Параметр - это ограничение FPS (frames per second - количество кадров в секунду). Т.е. с таким параметром метода tick(), часы позаботятся о том, чтобы у нас не было больше 50 кадров в секунду. На самом деле, если вы снова позовете .tick() раньше положенного времени (1/50 секунды в данном случае), то она просто зависнет, пока не пройдет нужное время. Таким образом, итерации нашего цикла будут выполняться не друг за другом а через паузу. Это необдимо, т.к. цикл вида

```
while True:
    for event in pygame.event.get():
        if event.type == pygame.QUIT or event.type == pygame.KEYDOWN:
            sys.exit()
```

сожрет 100% cpu.

В приложениях на Pygame используется событийная модель: программа крутится в цикле и обрабатывает поступающие события (нажатия кнопок, срабатывание таймера и т.п.). Список событий нам возвращает метод `pygame.event.get()`. События при этом удаляются из очереди в Pygame, т.е. если позвать этот метод дважды, то во второй раз мы не получим события, который получили в первый. Итак, в нашем примере, мы всего лишь проверяем, нажал ли пользователь любую клавишу, или может закрыл окно (событие QUIT, например если нажать на крестик в заголовке окна или Alt+F4), и если да, завершаем выполнение программы (только для этого мы и импортировали модуль sys).

Далее, обновляем координаты (no comments).

Далее, две функции рисования. `screen.fill(цвет)` красит весь экран (т.е. все наше окошко), `pygame.draw.circle(screen, цвет, координаты, радиус)`. Что характерно:

1. В обоих случаях нам нужен объект `screen`, который мы получили в начале программы, это наше окошко, собственно, где нужно рисовать.
2. Координаты в Pygame представлены кортежем целых чисел (x, y). Ось x направлена вправо, y вниз. Точка (0,0) находится в левом верхнем углу экрана. Заметьте, расчеты координат в примере ведутся в дробных числах, т.к. нам нужно точность. Но для рисования мы должны преобразовать координаты к типу `int`, т.к. для Pygame координаты - это номера пикселей на экране.
3. Цвет задается кортежем трех целих чисел: (red, green, blue). Каждая составляющая цвета изменяется от 0 до 255. Никогда, пусть вы и не художник. не используйте прогерские цвета, вроде (255, 0, 0) или (0, 255, 255), будьте чуть более оригинальны.

И последнее. В Pygame все функции рисования не рисуют сразу на экране. Они рисуют в некоем скрытом буфере. И только вызов `pygame.display.flip()` обновляет экран и отображает все. Без вызова `pygame.display.flip()` мы ничего не увидим на экране.

Справка по Pygame

Часы

pygame.time.Clock()	возвращает объект часов
clock.tick(fps)	устанавливает желаемый FPS и возвращает время прошедшее с прошлого кадра

События

<table>
<tr><td>pygame.event.get()</td><td colspan="3">возвращает список новых событий</td></tr>
<tr><td rowspan="8">event.type</td><td colspan="3">тип события, например:</td></tr>
<tr><td>pygame.QUIT</td><td colspan="2">попытка закрыть окно</td></tr>
<tr><td rowspan="6">pygame.KEYDOWN</td><td colspan="2">нажатие клавиши. При этом поле event.key будет соответствовать нажатой клавише:</td></tr>
<tr><td>pygame.K_ESCAPE</td><td>эскейп =)</td></tr>
<tr><td>pygame.K_SPACE</td><td>пробел</td></tr>
<tr><td>pygame.K_ENTER</td><td>энтер</td></tr>
<tr><td>pygame.K_0</td><td>ноль</td></tr>
<tr><td>pygame.K_a</td><td>A</td></tr>
<tr><td colspan="2">остальные тут</td></tr>
</table>

	pygame.KEYUP	отпускание клавиши. Аналогично.	
	pygame.MOUSEBUTTONUP	отпускание кнопки мыши. При этом поле event.button будет соответствовать клавише:	
		1	левая кнопка мыши
		2	средняя
		3	правая
		4	колесико вверх
		5	колесико вниз

Также можно получить информацию о состояниях кнопок и не обрабатывая события:

pygame.key.get_pressed()	Список состояний клавишь клавиатуры. `True` - нажата, `False` - нет. Например, чтобы проверить, нажата ли клавиша A, можно написать
	`if pygame.key.get_pressed()[pygame.K_a]:`
	`    ...`

pygame.mouse.get_pressed()	Аналогично, список состояний клавишь мыши. Наример,
	`if pygame.mouse.get_pressed()[0]:`
	`    ...`
	• нажата ли левая кнопка мыши (здесь кнопки номеруются с нуля, в отличае от событий мыши).

Рисование

pygame.draw.circle(screen, цвет, координаты, радиус, width=0)	рисует круг
pygame.draw.rect(screen, цвет, Rect(x, y, ширина, высота), width=0)	рисует прямоугольник, со сторонами параллельными границам окна. Rect(...) создает необходимый тут объект прямоугольника, который надо передать как параметр
line(screen, цвет, (x1, y1), (x2, y2), width=1)	рисует прямую линию от одной точки до другой
screen.fill(цвет)	заливка цветом всего окна
pygame.display.flip()	отрисовка всего

Необязательный параметр `width` в некоторых функциях задает толщину линии.

Практика

Упражнение №1

Научите шарик отскакивать от стенок. Постарайтесь также сделать, чтоб шарик не залетал за края экрана (самым простым, нафизичным способом).

Упражнение №2

Добавим управление: пусть при нажатой клавише-стрелке, у шарика появляется ускорение в соответствующую сторону. Испульзуйте список `pygame.key.get_pressed()`.

Упражнение №3

Добавим трение об воздух. Бесконечно ускорять шарик - не очень естественно. Напомним, что сила трения о воздух (а значит и соответствующее ускорение) пропорционально скорости и прортивонаправлено ей.

Упражнение №4

Цвет шарика. Пусть он зависит от скорости.

Упражнение №5

Добавляем второй шарик. И пишем соударение шаров. Соударение шаров рассчитывается так: нужно разложить движение по двум осям: одна - это нормаль контакта, т.к. перпендикуляр к поверхности в точке контакта (в нашем случае, это будет прямая, проходящая через центры шаров), вторая ось - перпендикуляр к первой. Так вот, при упругом соударении, движение по первой оси изменится также, как если это былобы лобовое соударение шаров, а по второй - не изменится.

Упражнение №6

Добавление шаров по нажатию кнопки мыши (добавить в том месте, где находится курсор)