PyJawan: Python Gaming Library

Anirudh Sathiya Narayanan, Luv Singhal, Vrishab Krishna

Contents

1 Introduction **3**

2 Acknowledgements **3**

3 PyJawan **4**

3.1 Project Requirements 4

3.1.1 Hardware requirements 4

3.1.2 Hardware requirements 4

3.2 4

4 Source Code **4**

1 Introduction

PyJawan is a set of Python modules designed for writing video games. Using OpenCVs image manipulation and rendering capabilities, our aim was to develop a library more efficient and easier to use than the PyGame library.

The library consists of a window and game rendering module, a drawing module for showing shapes and images, UI/Input modules, and engine for sprites.

To demonstrate the power and simplicity of this library, we have developed an example game of FlappyBird. This example tests most of the aspects of our library as a proof-of-concept and validation that each of the major functions provided and constitutes a significantly lower number of lines of code than that required for the PyGame equivalent.

2 Acknowledgements

We would like to express a deep sense of gratitude towards my teachers Ms Suguna Ganesh, and the school admin, Mahesh Bhaiya. They have always evinced keen interest in our work. Their constructive advice and endless support was instrumental in the success of the project.

My sincere thanks goes to Ms Shanta Chandran, our principal, for her coordination and extending every possible support for the completion of this project.

Finally, I would like to convey my thanks to the Central Board of Secondary Education (CBSE) for giving me the opportunity to undertake this project.

3 PyJawan

3.1 Project Requirements

3.1.1 Hardware requirements

The project requires the ability to render images at a fast pace. Hence, we have the following basic hardware requirements:

- Gigahertz or faster 32 bit or 64 bit processor
- Disk space `>= 10 MB`
- A working keyboard and monitor

3.1.2 Hardware requirements

In terms of software, the library will function with the following basic requirements:

- Operating Systems: macOS and Linux
- Python Version: `>=3.7.x`
- OpenCV Version: `>=4.2.x`

3.2

4 Source Code

```
> PyJawan/core/draw.py
import os
from utils.constants import HorizontalAlignment, VerticalAlignment
import cv2 as cv
import numpy as np

from math import factorial
from PIL import Image, ImageDraw, ImageFont
from numpy.core.fromnumeric import shape
from numpy.core.numeric import zeros_like

from utils.color import Color
from core.rect import Rect
# from core.surface import Surface
```

```
BASE_DIR = os.path.join(os.path.abspath(os.pardir), "utils", "fonts
    ")


class Drawer:
    def __init__(self):
        self.fonts = {}
        self.images = {}
    # SECTION Helper Functions

    def _bernstein_poly(self, i, n, t):
        return factorial(n) / (factorial(i) * factorial(n - i)) * (t
            **(n - i)) * (1 - t)**i

    def _bezier_curve(self, points, resol=1000):

        n = len(points)
        x, y = np.array(points).T

        t = np.linspace(0.0, 1.0, resol)

        # TODO: Fix poly_mat
        # poly_mat = np.fromfunction(lambda i, j: self.
            _bernstein_poly(j, n, t[i]) , (resol, n))
        poly_mat = np.array([self._bernstein_poly(i, n - 1, t)
                             for i in range(0, n)])

        xvals = np.dot(x, poly_mat).astype(np.int32)
        yvals = np.dot(y, poly_mat).astype(np.int32)

        return np.vstack([xvals, yvals]).T

    # !SECTION

    def load_font(self, name, path):
        self.fonts.set(name, ImageFont.load_path(path))

    def rect(self, surf, x: int, y: int, w: int, h: int, color=Color
        .Black, thickness=1, fill=False):
        cv.rectangle(surf.img, (x, y), (x + w, y + h),
                     color.bgr, -1 if fill else thickness)

    def fill(self, surf, color=Color.Black):
        surf.img[:, :] = color.bgr
```

```
def line(self, surf, x1: int, y1: int, x2: int, y2: int, color=
    Color.Black, thickness=1):
    cv.line(surf.img, (x1, y1), (x2, y2), color.bgr, thickness)

def circle(self, surf, x: int, y: int, r: int, color=Color.Black
    , thickness=1, fill=False):
    cv.ellipse(surf.img, (x, y), (r, r),
               360, 0, 360, color.bgr, thickness=-1 if fill else
                   thickness)

def ellipse(self, surf, x: int, y: int, w: int, h: int, color=
    Color.Black, angle=0, thickness=1, fill=False):
    cv.ellipse(surf.img, (x, y), (w // 2, h // 2),
               angle, 0, 360, color.bgr, thickness=-1 if fill else
                    thickness)

def arc(self, surf, start_pt: int, stop_pt: int, start_angle=0,
    stop_angle=90, color=Color.Black, thickness=0):
    # TODO
    pass

def curve(self, surf, points: list, color=Color.Black, thickness
    =0, fill=False, closed=False):
    points = self._bezier_curve(points)
    if fill:
        cv.fillPoly(surf.img, [np.array(
            points).reshape((-1, 1, 2))], color.bgr)
    else:
        cv.polylines(surf.img, [np.array(points).reshape(
            (-1, 1, 2))], closed, color.bgr, thickness=thickness)

def polygon(self, surf, vertices: list, color=Color.Black,
    thickness=0, fill=False):
    if fill:
        cv.fillPoly(surf.img, [np.array(
            vertices).reshape((-1, 1, 2))], color.bgr)
    else:
        cv.polylines(surf.img, [np.array(vertices).reshape(
            (-1, 1, 2))], True, color.bgr, thickness=thickness)

def gradient(self, surf, x: int, y: int, w: int, h: int, color1=
    Color.Black, color2=Color.Black):
    c1 = np.full((1, h, 3), color1.bgr, dtype=np.uint8)
    c2 = np.full((1, h, 3), color2.bgr, dtype=np.uint8)
```

```
    base = np.concatenate([c1, c2], axis=0)
    grad = cv.resize(base, (w, h), cv.INTER_LINEAR)
    surf.img[y:y + h, x:x + w] = grad

def text(
    self, surf, text: str, x: int, y: int,
    size=10, font_name="sans-serif", font_path="", color=Color.
        Black,
    h_align=HorizontalAlignment.Left,
    v_align=VerticalAlignment.Center
):
    if font_name in ("monospace", "serif", "sans-serif"):
        font = ImageFont.truetype(os.path.join(BASE_DIR, f"{
            font_name}.ttf"), size)
    else:
        font = ImageFont.load(font_path, size)

    img = Image.fromarray(surf.img)
    draw = ImageDraw.Draw(img)
    (w, h) = draw.textsize(text, font=font)

    if h_align == HorizontalAlignment.Center:
        x -= w // 2
    elif h_align == HorizontalAlignment.Right:
        x -= w

    if v_align == VerticalAlignment.Center:
        y -= h // 2
    elif v_align == VerticalAlignment.Bottom:
        y -= h

    draw.text((x, y), text, font=font, fill=color.bgr)
    surf.img = np.array(img)

def surface(self, surf, to_draw, x: int, y: int,):
    surf.img[y:y + to_draw.height, x:x +
             to_draw.width] = to_draw.img.copy()

def image(self, surf, path: str, rect: Rect):
    im = self.images.get(path)
    if im is None:
        im = cv.imread(path, cv.IMREAD_UNCHANGED)
        self.images[path] = im

    im = cv.resize(im, (rect.w, rect.h))
```

```
        try:
            mask = 255 * np.zeros((*im.shape[:2], 3), dtype=np.uint8)
            mask[im[:, :, 3] == 0] = 255
            surf.img[rect.y:rect.y + rect.h, rect.x:rect.x + rect.w]
                = (
                im[:, :, :-1] | surf.img[rect.y:rect.y + rect.h, rect.
                    x: rect.x + rect.w] & mask)

        except:
            surf.img[rect.y: rect.y + rect.h, rect.x: rect.x +
                     rect.w] = im
```

```
> PyJawan/core/event.py
from pynput import mouse, keyboard


class Event:
    pass


class KeyDownEvent(Event):
    def __init__(self, key: keyboard.Key):
        self.key = key

    @property
    def char(self):
        try:
            return self.key.char
        except AttributeError:
            return None


class KeyUpEvent(Event):
    def __init__(self, key: keyboard.Key):
        self.key = key

    @property
    def char(self):
        try:
            return self.key.char
        except AttributeError:
            return None


class MouseMoveEvent(Event):
```

```
    def __init__(self, mouseX: int, mouseY: int, prev_event):
        self.x = mouseX
        self.y = mouseY
        if prev_event != None:
            self.prev_x = prev_event.x
            self.prev_y = prev_event.y
        else:
            self.prev_x = None
            self.prev_y = None

    @property
    def dx(self):
        if self.prev_x != None:
            return self.x - self.prev_x

    @property
    def dy(self):
        if self.prev_y != None:
            return self.y - self.prev_y

    @property
    def distance(self):
        if self.prev_x != None and self.prev_y != None:
            return (self.x*self.x + self.y*self.y) ** 0.5


class MouseClickEvent(Event):
    def __init__(self, mouseX: int, mouseY: int, button: mouse.
        Button, pressed: bool):
        self.x = mouseX
        self.y = mouseY
        self.button = button
        self.pressed = pressed

# TODO see onscroll
```

```
> PyJawan/core/__init__.py
from core.surface import Surface
from core.draw import Drawer
from core.event import KeyDownEvent, KeyUpEvent, MouseClickEvent,
    MouseMoveEvent
from core.window import Window
from core.rect import Rect
from pynput.keyboard import Key
```

```
> PyJawan/core/rect.py
class Rect:
    def __init__(self, x: int, y: int, w: int, h: int):
        if w < 0:
            w = -w
            x = x - w

        if h < 0:
            h = -h
            y = y - h

        self.x = x
        self.y = y
        self.w = w
        self.h = h

    def area(self) -> int:
        return self.w * self.h

    def collides(self, rect) -> bool:
        return (
            (self.x + self.w >= rect.x) and
            (self.x <= rect.x + rect.w) and
            (self.y + self.h >= rect.y) and
            (self.y <= rect.y + rect.h)
        )

    def top_left(self):
        return (self.x, self.y)

    def top_right(self):
        return (self.x + self.w, self.y)

    def bottom_left(self):
        return (self.x, self.y + self.w)

    def bottom_right(self):
        return (self.x + self.w, self.y + self.h)

    def center(self):
        return (self.x + self.w // 2, self.y + self.h // 2)

    def __repr__(self):
        return f'Rect({self.x}, {self.y}, {self.w}, {self.h})'
```

```
    def __str__(self):
        return f'Rect({self.x}, {self.y}, {self.w}, {self.h})'

> PyJawan/core/surface.py
from core.draw import Drawer
from core.rect import Rect
from utils import Color, VerticalAlignment, HorizontalAlignment
import numpy as np
import cv2 as cv
from utils.constants import EventType
from types import FunctionType


class Surface:
    def __init__(self, width: int, height: int):
        self.img = np.zeros((height, width, 3), dtype=np.uint8)
        self.drawer = Drawer()

    @property
    def height(self):
        return self.img.shape[0]

    @property
    def width(self):
        return self.img.shape[1]

    @property
    def size(self):
        return self.img.shape[1::-1]

    def fill(self, color=Color.Black):
        self.drawer.fill(self, color)

    def draw_rect(self, x: int, y: int, w: int, h: int, color=Color.
        Black, thickness=1, fill=False):
        self.drawer.rect(self, x, y, w, h, color, thickness, fill)

    def draw_line(self, x1: int, y1: int, x2: int, y2: int, color=
        Color.Black, thickness=1):
        self.drawer.line(self, x1, y1, x2, y2, color, thickness)

    def draw_circle(self, x: int, y: int, r: int, color=Color.Black,
         thickness=1, fill=False):
        self.drawer.circle(self, x, y, r, color, thickness, fill)
```

```
    def draw_ellipse(self, x: int, y: int, w: int, h: int, color=
        Color.Black, angle=0, thickness=1, fill=False):
        self.drawer.ellipse(self, x, y, w, h, color, angle, thickness
            , fill)

    def draw_curve(self, points: list, color=Color.Black, thickness
        =0, fill=False, closed=False):
        self.drawer.curve(self, points, color, thickness, fill,
            closed)

    def draw_polygon(self, vertices: list, color=Color.Black,
        thickness=0, fill=False):
        self.drawer.polygon(self, vertices, color, thickness, fill)

    def draw_gradient(self, x: int, y: int, w: int, h: int, color1=
        Color.Black, color2=Color.Black):
        self.drawer.gradient(self, x, y, w, h, color1, color2)

    def draw_text(self, text: str, x: int, y: int, size=10,
        font_name="sans-serif", font_path="", color=Color.Black,
        h_align=HorizontalAlignment.Left, v_align=VerticalAlignment.
        Top):
        self.drawer.text(self, text, x, y, size, font_name,
                         font_path, color, h_align, v_align)

    def draw_surface(self, to_draw, x: int, y: int):
        self.drawer.surface(self, to_draw, x, y)

    def draw_image(self, path: str, rect: Rect):
        self.drawer.image(self, path, rect)
```

```
> PyJawan/core/window.py
import numpy as np
import cv2 as cv
from utils.constants import EventType
from types import FunctionType
import time
from core.event import *
from pynput import mouse, keyboard
from core.surface import Surface


class Window(Surface):
    def __init__(self, width: int, height: int, name="PyJawan"):
        super().__init__(width, height)
```

```
        self.name = name
        self.window = cv.namedWindow(
            name, cv.WINDOW_NORMAL | cv.WINDOW_GUI_NORMAL)
        cv.resizeWindow(name, width, height)
        self.handlers = {
            EventType.KeyUp: {},
            EventType.KeyDown: {},
            EventType.MouseClick: {},
            EventType.MouseMove: {}
        }
        self.mouseEvent = None
        self._registerListener()
        self.stop = False

    @property
    def height(self):
        return self.img.shape[0]

    @property
    def width(self):
        return self.img.shape[1]

    @property
    def size(self):
        return self.img.shape[1::-1]

    def render(self):
        if cv.getWindowProperty(self.name, 4) == 0:
            return False

        cv.imshow(self.name, self.img)

        return True

    def loop(self, main: FunctionType, delay=50):
        while self.render():
            current_time = time.time()
            key = cv.waitKey(delay) & 0xFF
            dt = time.time() - current_time
            main(dt)
            dt = time.time() - current_time
            if dt < delay:
                time.sleep((delay - dt) / 1000)
            if self.stop:
                break
```

```
def quit(self):
    self.stop = True

def close(self):
    cv.destroyAllWindows()

def on(self, event_type: EventType, handler_id: str, fn:
    FunctionType):
    self.handlers[event_type][handler_id] = fn

def off(self, event_type: EventType, handler_id: str):
    self.handlers[event_type].pop(handler_id)

def _callHandler(self, event_type: EventType, event: Event):
    if event_type == EventType.MouseMove or event_type ==
        EventType.MouseClick:
        win_rect = cv.getWindowImageRect(self.name)
        event.x -= win_rect[0]
        event.y -= win_rect[1]

        event.x *= self.width / win_rect[2]
        event.y *= self.height / win_rect[3]

        # Out of bounds
        if event.x < 0 or event.x >= self.width or event.y < 0 or
             event.y >= self.height:
            return

    if event_type == EventType.MouseMove:
        self.mouseEvent = event

    for handler in self.handlers[event_type].values():
        handler(event)

def _registerListener(self):
    mouse_listener = mouse.Listener(
        on_move=lambda x, y: self._callHandler(
            EventType.MouseMove, MouseMoveEvent(x, y, self.
                mouseEvent)),
        on_click=lambda x, y, btn, pressed: self._callHandler(
            EventType.MouseClick, MouseClickEvent(x, y, btn,
                pressed))
    )
```

```
        mouse_listener.start()

        key_listener = keyboard.Listener(
            on_press=lambda key: self._callHandler(
                EventType.KeyDown, KeyDownEvent(key)),
            on_release=lambda key: self._callHandler(
                EventType.KeyUp, KeyUpEvent(key))
        )

        key_listener.start()
```

```
> PyJawan/engine/__init__.py
from engine.sprite import Sprite
```

```
> PyJawan/engine/sprite.py
from core import Surface, Rect


class Sprite:
    def __init__(self, x, y, w, h):
        self.rect = Rect(x, y, w, h)
        self.alive = True

    def render(self, surf: Surface):
        pass

    def update(self, delta: int):
        pass

    def kill(self):
        print('Life is soup, I am fork -', str(self))
        self.alive = False

    def collides(self, sprite) -> bool:
        return self.rect.collides(sprite.rect)
```

```
> PyJawan/ui/button.py
from core import Rect, Surface, Drawer, Window
from utils.constants import EventType, HorizontalAlignment,
    VerticalAlignment
from types import FunctionType
from utils import Color


class Button:
```

```
def __init__(self, x: int, y: int, w: int, h: int, text='',
             text_col=Color.Black, hov_text_col=Color.Black,
             font_size=16,
             bg_col=Color.LightGray, hov_bg_col=Color.Gray,
             border_radius=0, is_visible=True):
    self.x = x
    self.y = y
    self.w = h
    self.h = h
    self.text = text
    self.text_col = text_col
    self.hov_text_col = hov_text_col
    self.font_size = font_size
    self.bg_col = bg_col
    self.hov_bg_col = hov_bg_col
    self.border_radius = border_radius
    self.is_visible = is_visible
    self.id = 'button-' + str(id(self))
    self.is_hovering = False
    self.click_handler = None

def _is_hovering(self, e):
    print(e.x, e.y)
    self.is_hovering = self.x < e.x < (
        self.x + self.w) and self.y < e.y < (self.y + self.h)

def on_click(self, f: FunctionType):
    self.click_handler = f

def register(self, window: Window):
    window.on(EventType.MouseMove, self.id, self._is_hovering)
    window.on(EventType.MouseClick, self.id, lambda e: self.
        click_handler(
        e) if self.click_handler else None)

def destory(self, window: Window):
    window.off(EventType.MouseMove, self.id)
    window.off(EventType.MouseClick, self.id)

def draw(self, surf: Surface):
    text_col = self.hov_text_col if self.is_hovering else self.
        text_col
    bg_col = self.hov_bg_col if self.is_hovering else self.bg_col

    if self.border_radius == 0:
```

```
            surf.draw_rect(self.x, self.y, self.w,
                           self.h, bg_col, fill=True)
        else:
            surf.draw_circle(self.x + self.border_radius, self.y +
                             self.border_radius, self.border_radius,
                                 color=bg_col, fill=True)

            surf.draw_circle(self.x + self.border_radius, self.y +
                self.h -
                             self.border_radius, self.border_radius,
                                 color=bg_col, fill=True)

            surf.draw_circle(self.x + self.w - self.border_radius,
                self.y +
                             self.border_radius, self.border_radius,
                                 color=bg_col, fill=True)

            surf.draw_circle(self.x + self.w - self.border_radius,
                self.y + self.h -
                             self.border_radius, self.border_radius,
                                 color=bg_col, fill=True)

            surf.draw_rect(self.x + self.border_radius, self.y, self.
                w - 2 *
                           self.border_radius, self.border_radius,
                               color=bg_col, fill=True)

            surf.draw_rect(self.x, self.y + self.border_radius, self.
                w,
                           self.h - 2 * self.border_radius, color=
                               bg_col, fill=True)

            surf.draw_rect(self.x + self.border_radius, self.y + self
                .h - self.border_radius, self.w - 2 * self.
                border_radius,
                           self.border_radius, color=bg_col, fill=True)

        if self.text and len(self.text):
            surf.draw_text(self.text, self.x + self.w // 2, self.y +
                self.h / 2, self.font_size,
                           color=text_col, h_align=HorizontalAlignment.
                               Center, v_align=VerticalAlignment.Center
                               )
```

> PyJawan/ui/__init__.py

```
from ui.button import Button

> PyJawan/utils/color.py
from enum import Enum, unique


class _Color:
    def __init__(self, col):
        r = (col >> 16) & 0xff
        g = (col >> 8) & 0xff
        b = (col >> 0) & 0xff

        self.bgr = (b, g, r)


class Color(_Color):
    def _hexFromString(self, string):
        origstr = string[::-1]
        string = origstr.upper()
        val = 0
        for i in range(len(string)):
            if string[i].isdigit():
                val += 16**i * int(string[i])
            elif 65 <= ord(string[i]) <= 70:
                val += 16**i * (ord(string[i]) - 55)
            else:
                raise ValueError("Unknown character " + origstr[i])

        return val

    def _colorFromList(self, lst):
        for i in lst:
            if not isinstance(i, int):
                raise TypeError(f"{i} must be of type int")
        return tuple(lst[::-1])

    # Color(r, g, b)
    # Color((r, g, b))
    # Color("#rrggbb")
    # Color("#rgb")
    # Color(0xRRGGBB)

    def __init__(self, r, g=None, b=None):
        if g == None and b == None:
            if isinstance(r, int):
```

```
            red = (r >> 16) & 0xff
            green = (r >> 8) & 0xff
            blue = (r >> 0) & 0xff

            self.bgr = (blue, green, red)
        elif isinstance(r, (list, tuple)):
            if len(r) != 3:
                raise ValueError("Colors should consist of only 3
                    values.")
            self.bgr = self._colorFromList(r)

        elif isinstance(r, str):
            if r[0] != "#" or (len(r) != 7 and len(r) != 4):
                raise ValueError(
                    "Colors should be in the form '#rrggbb' or '#
                        rgb'\nIf in the form '#rgb' it will
                        automaticallty repeat the r, g, b values.\
                        neg: #345 => #334455")
            if len(r) == 4:
                red = self._hexFromString(r[1]*2)
                green = self._hexFromString(r[2]*2)
                blue = self._hexFromString(r[3]*2)
            else:
                red = self._hexFromString(r[1:3])
                green = self._hexFromString(r[3:5])
                blue = self._hexFromString(r[5:7])

            self.bgr = (blue, green, red)
        else:
            raise TypeError("Unknwon Type " + str(type(r)))

    elif g == None or b == None:
        raise ValueError(
            "This function only accepts either one or three
                arguments")

    else:
        self.bgr = self._colorFromList([r, g, b])

AliceBlue = _Color(0xF0F8FF)
AntiqueWhite = _Color(0xFAEBD7)
Aquamarine = _Color(0x7FFFD4)
Azure = _Color(0xF0FFFF)
Beige = _Color(0xF5F5DC)
Bisque = _Color(0xFFE4C4)
```

```
Black = _Color(0x000000)
BlanchedAlmond = _Color(0xFFEBCD)
Blue = _Color(0x0000FF)
BlueViolet = _Color(0x8A2BE2)
Brown = _Color(0xA52A2A)
BurlyWood = _Color(0xDEB887)
CadetBlue = _Color(0x5F9EA0)
Chartreuse = _Color(0x7FFF00)
Chocolate = _Color(0xD2691E)
Coral = _Color(0xFF7F50)
CornflowerBlue = _Color(0x6495ED)
Cornsilk = _Color(0xFFF8DC)
Crimson = _Color(0xDC143C)
Cyan = _Color(0x00FFFF)
DarkBlue = _Color(0x00008B)
DarkCyan = _Color(0x008B8B)
DarkGoldenRod = _Color(0xB8860B)
DarkGray = _Color(0xA9A9A9)
DarkGreen = _Color(0x006400)
DarkKhaki = _Color(0xBDB76B)
DarkMagenta = _Color(0x8B008B)
DarkOliveGreen = _Color(0x556B2F)
DarkOrange = _Color(0xFF8C00)
DarkOrchid = _Color(0x9932CC)
DarkRed = _Color(0x8B0000)
DarkSalmon = _Color(0xE9967A)
DarkSeaGreen = _Color(0x8FBC8F)
DarkSlateBlue = _Color(0x483D8B)
DarkSlateGray = _Color(0x2F4F4F)
DarkTurquoise = _Color(0x00CED1)
DarkViolet = _Color(0x9400D3)
DeepPink = _Color(0xFF1493)
DeepSkyBlue = _Color(0x00BFFF)
DimGray = _Color(0x696969)
DodgerBlue = _Color(0x1E90FF)
FireBrick = _Color(0xB22222)
FloralWhite = _Color(0xFFFAF0)
ForestGreen = _Color(0x228B22)
Fuchsia = _Color(0xFF00FF)
Gainsboro = _Color(0xDCDCDC)
GhostWhite = _Color(0xF8F8FF)
Gold = _Color(0xFFD700)
GoldenRod = _Color(0xDAA520)
Gray = _Color(0x808080)
Green = _Color(0x008000)
```

```
GreenYellow = _Color(0xADFF2F)
HoneyDew = _Color(0xF0FFF0)
HotPink = _Color(0xFF69B4)
IndianRed = _Color(0xCD5C5C)
Indigo = _Color(0x4B0082)
Ivory = _Color(0xFFFFF0)
Khaki = _Color(0xF0E68C)
Lavender = _Color(0xE6E6FA)
LavenderBlush = _Color(0xFFF0F5)
LawnGreen = _Color(0x7CFC00)
LemonChiffon = _Color(0xFFFACD)
LightBlue = _Color(0xADD8E6)
LightCoral = _Color(0xF08080)
LightCyan = _Color(0xE0FFFF)
LightGoldenRodYellow = _Color(0xFAFAD2)
LightGray = _Color(0xD3D3D3)
LightGreen = _Color(0x90EE90)
LightPink = _Color(0xFFB6C1)
LightSalmon = _Color(0xFFA07A)
LightSeaGreen = _Color(0x20B2AA)
LightSkyBlue = _Color(0x87CEFA)
LightSlateGray = _Color(0x778899)
LightSteelBlue = _Color(0xB0C4DE)
LightYellow = _Color(0xFFFFE0)
Lime = _Color(0x00FF00)
LimeGreen = _Color(0x32CD32)
Linen = _Color(0xFAF0E6)
Maroon = _Color(0x800000)
MediumAquaMarine = _Color(0x66CDAA)
MediumBlue = _Color(0x0000CD)
MediumOrchid = _Color(0xBA55D3)
MediumPurple = _Color(0x9370DB)
MediumSeaGreen = _Color(0x3CB371)
MediumSlateBlue = _Color(0x7B68EE)
MediumSpringGreen = _Color(0x00FA9A)
MediumTurquoise = _Color(0x48D1CC)
MediumVioletRed = _Color(0xC71585)
MidnightBlue = _Color(0x191970)
MintCream = _Color(0xF5FFFA)
MistyRose = _Color(0xFFE4E1)
Moccasin = _Color(0xFFE4B5)
NavajoWhite = _Color(0xFFDEAD)
Navy = _Color(0x000080)
OldLace = _Color(0xFDF5E6)
Olive = _Color(0x808000)
```

```
    OliveDrab = _Color(0x6B8E23)
    Orange = _Color(0xFFA500)
    OrangeRed = _Color(0xFF4500)
    Orchid = _Color(0xDA70D6)
    PaleGoldenRod = _Color(0xEEE8AA)
    PaleGreen = _Color(0x98FB98)
    PaleTurquoise = _Color(0xAFEEEE)
    PaleVioletRed = _Color(0xDB7093)
    PapayaWhip = _Color(0xFFEFD5)
    PeachPuff = _Color(0xFFDAB9)
    Peru = _Color(0xCD853F)
    Pink = _Color(0xFFC0CB)
    Plum = _Color(0xDDA0DD)
    PowderBlue = _Color(0xB0E0E6)
    Purple = _Color(0x800080)
    RebeccaPurple = _Color(0x663399)
    Red = _Color(0xFF0000)
    RosyBrown = _Color(0xBC8F8F)
    RoyalBlue = _Color(0x4169E1)
    SaddleBrown = _Color(0x8B4513)
    Salmon = _Color(0xFA8072)
    SandyBrown = _Color(0xF4A460)
    SeaGreen = _Color(0x2E8B57)
    SeaShell = _Color(0xFFF5EE)
    Sienna = _Color(0xA0522D)
    Silver = _Color(0xC0C0C0)
    SkyBlue = _Color(0x87CEEB)
    SlateBlue = _Color(0x6A5ACD)
    SlateGray = _Color(0x708090)
    Snow = _Color(0xFFFAFA)
    SpringGreen = _Color(0x00FF7F)
    SteelBlue = _Color(0x4682B4)
    Tan = _Color(0xD2B48C)
    Teal = _Color(0x008080)
    Thistle = _Color(0xD8BFD8)
    Tomato = _Color(0xFF6347)
    Turquoise = _Color(0x40E0D0)
    Violet = _Color(0xEE82EE)
    Wheat = _Color(0xF5DEB3)
    White = _Color(0xFFFFFF)
    WhiteSmoke = _Color(0xF5F5F5)
    Yellow = _Color(0xFFFF00)
    YellowGreen = _Color(0x9ACD32)
```

> PyJawan/utils/constants.py

```
from enum import unique, Enum, auto

FRAME_RATE = 50


@unique
class EventType(Enum):
    KeyUp = auto()
    KeyDown = auto()
    MouseMove = auto()
    MouseClick = auto()


@unique
class HorizontalAlignment(Enum):
    Left = auto()
    Right = auto()
    Center = auto()


@unique
class VerticalAlignment(Enum):
    Top = auto()
    Center = auto()
    Bottom = auto()

> PyJawan/utils/__init__.py
from utils.color import Color
from utils.constants import *

> PyJawan/examples/flappybird.py
import sys
sys.path.append("../")
from core import Window, Drawer, Surface, Rect, KeyDownEvent, Key
from engine import Sprite
from utils import FRAME_RATE, EventType, Color, HorizontalAlignment,
     VerticalAlignment
import os
from random import randint
from typing import Tuple, List


# SECTION Constants

MAX_VERTICAL_VEL = -15
```

```
MIN_VERTICAL_VEL = 15
SPEED = 10
SLIT_WIDTH = 200
PIPE_SEP = 250
UP_ACCEL = -50
DOWN_ACCEL = 50
GRAVITY = 3

# !SECTION Constants

# SECTION Helper functions


def random_height(win_height) -> Tuple[int, int]:
    top = randint(0, win_height - SLIT_WIDTH - 1)
    bottom = randint(0, win_height - top - SLIT_WIDTH)
    return top, bottom

# !SECTION

# SECTION Classes


class Pipe(Sprite):
    def __init__(self, start_x: int, top: int, bottom: int, speed:
        int, win_height: int):
        super().__init__(start_x, 0, 100, 100)
        self.top = top
        self.width = 10
        self.bottom = bottom
        self.speed = speed
        self.win_height = win_height

    def render(self, surf: Surface):
        if not self.alive:
            return

        if surf.height < (self.top + self.bottom):
            raise RuntimeError(
                f"Pipe must have opening: surface height: {surf.height
                    }, pipe height: {self.top}, {self.bottom}")

        surf.draw_rect(self.rect.x, 0, self.width, self.top,
                       color=Color.Black, fill=True)
```

```
        surf.draw_rect(self.rect.x, surf.height - self.bottom, self.
            width, self.bottom,
                       color=Color.Black, fill=True)

    def update(self) -> int:
        self.rect.x -= self.speed
        if self.rect.x <= 0:
            self.rect.x = win_rect.w
            return 1
        else:
            return 0

    def collides(self, sprite: Sprite) -> bool:
        return sprite.rect.collides(
            Rect(self.rect.x, 0, self.width, self.top)
        ) or sprite.rect.collides(
            Rect(self.rect.x, self.win_height -
                 self.bottom, self.width, self.bottom)
        )


class Bird(Sprite):
    def __init__(self, x, y, up_im: str, down_im: str, accel=GRAVITY
        ):
        super().__init__(x, y, 40, 40)
        self.accel = accel
        self.init_pos = (x, y)
        self.up_im = up_im
        self.down_im = down_im
        self.vertical_velocity = 0
        self.alive = False

    def jump(self):
        self.accel = UP_ACCEL

    def reset(self):
        self.alive = True
        self.rect.x = self.init_pos[0]
        self.rect.y = self.init_pos[1]

    def render(self, surf: Surface):
        if not self.alive:
            return
        if self.vertical_velocity > 0:
            surf.drawer.image(surf, self.down_im, self.rect)
```

```
        else:
            surf.drawer.image(surf, self.up_im, self.rect)

    def update(self, win_height: int):
        if not self.alive:
            return

        self.vertical_velocity += self.accel
        self.vertical_velocity = max(self.vertical_velocity,
            MAX_VERTICAL_VEL)
        self.vertical_velocity = min(self.vertical_velocity,
            MIN_VERTICAL_VEL)
        self.rect.y += self.vertical_velocity

        if self.rect.y < 0 or self.rect.y >= win_height - self.rect.h
            :
            self.kill()
        self.accel = GRAVITY

# !SECTION

# SECTION Main functionality


win = Window(1280, 720, "Flappy Bird - PyJawan")
win_rect = Rect(0, 0, 1280, 720)

heights = [random_height(win.height) for i in range(win_rect.w //
    PIPE_SEP)]
pipes = [Pipe(win_rect.w - i * PIPE_SEP + 100, h[0], h[1], SPEED,
    win.height)
        for i, h in enumerate(heights)]

bird = Bird(100, 360, "./assets/bird_up.png",
            "./assets/bird_down.png",)


def reset():
    global bird, pipes, score
    bird.reset()
    pipes = [Pipe(win_rect.w - i * PIPE_SEP + 100, h[0], h[1],
                SPEED, win.height) for i, h in enumerate(heights)]
    score = 0
```

```
def handle_key(e: KeyDownEvent):
    if e.key == Key.space:
        if bird.alive:
            bird.jump()
    elif e.char == 'r':
        reset()
    elif e.char == 'q':
        win.quit()


win.on(EventType.KeyDown, 'key-handler', handle_key)

score = 0


def main(_):
    global score
    if bird.alive:
        win.draw_image("./assets/flappy-bird-1.png", win_rect)
        bird.render(win)

        for pipe in pipes:
            if pipe.collides(bird):
                bird.kill()
            pipe.render(win)
            score += pipe.update()

        bird.update(win.height)
        win.draw_text(f"Score: {score}", (win.width - 10), 10, size
            =20, font_name="monospace",
                      color=Color.AliceBlue, h_align=
                          HorizontalAlignment.Right)
    else:
        win.draw_text("Please press <R> to start", win.width // 2,
            win.height // 2, size=40, font_name="monospace",
                      color=Color.AliceBlue, h_align=
                          HorizontalAlignment.Center, v_align=
                          VerticalAlignment.Center)


win.loop(main, 10)
win.off(EventType.KeyDown, 'key-handler')
win.close()
```