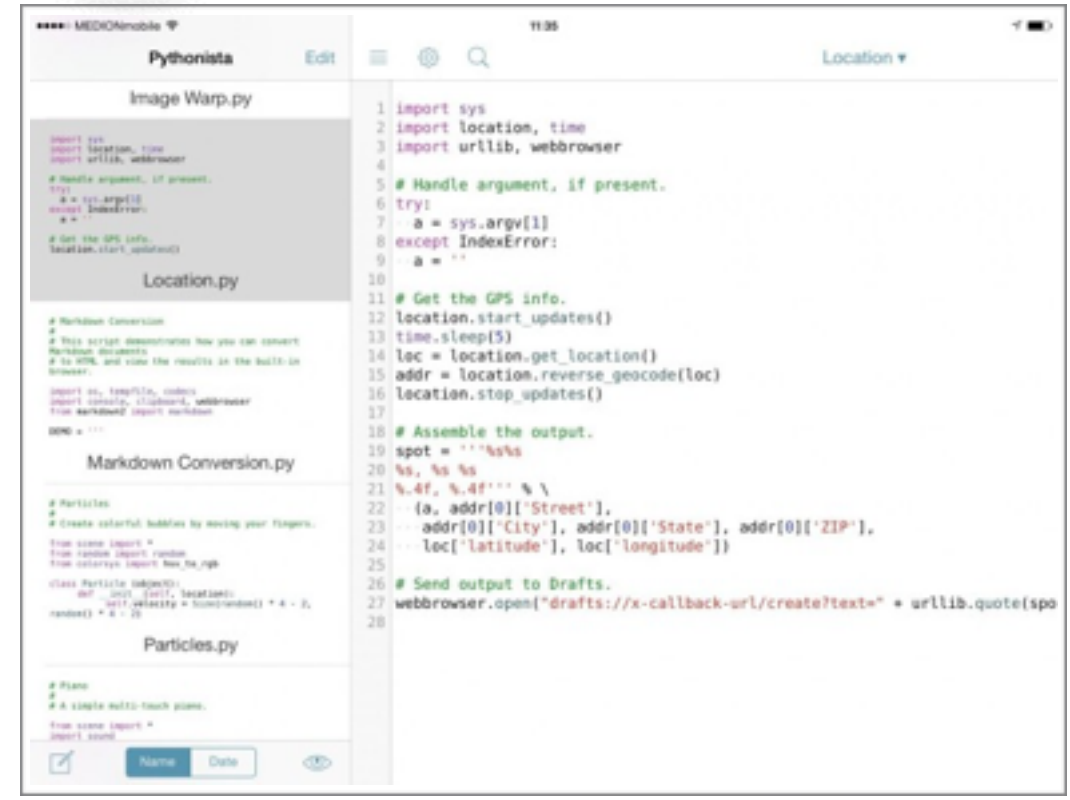
「PythonによるiOSアプリ制作 ~ゲームプログラミング入門~」
簡単にオリジナルゲームを作ろう！！



今日のスケジュール

1. Pythonについて
2. Pythonistaについて
3. ゲーム制作基礎（なんちゃってシューティングゲーム）
    - lesson 1　プレイヤーを動かしてみよう
    - lesson 2　インベーダーを並べてみよう
    - lesson 3　弾を飛ばしてみよう
    - lesson 4　弾に当たったインベーダーを消してみよう
    - lesson 5　点数を付けよう
    - lesson 6　時間制限を付けよう
    - lesson 7　クリア画面、ゲームオーバー画面を作ろう

4. オリジナルゲームを作ってみよう
5. 発表会

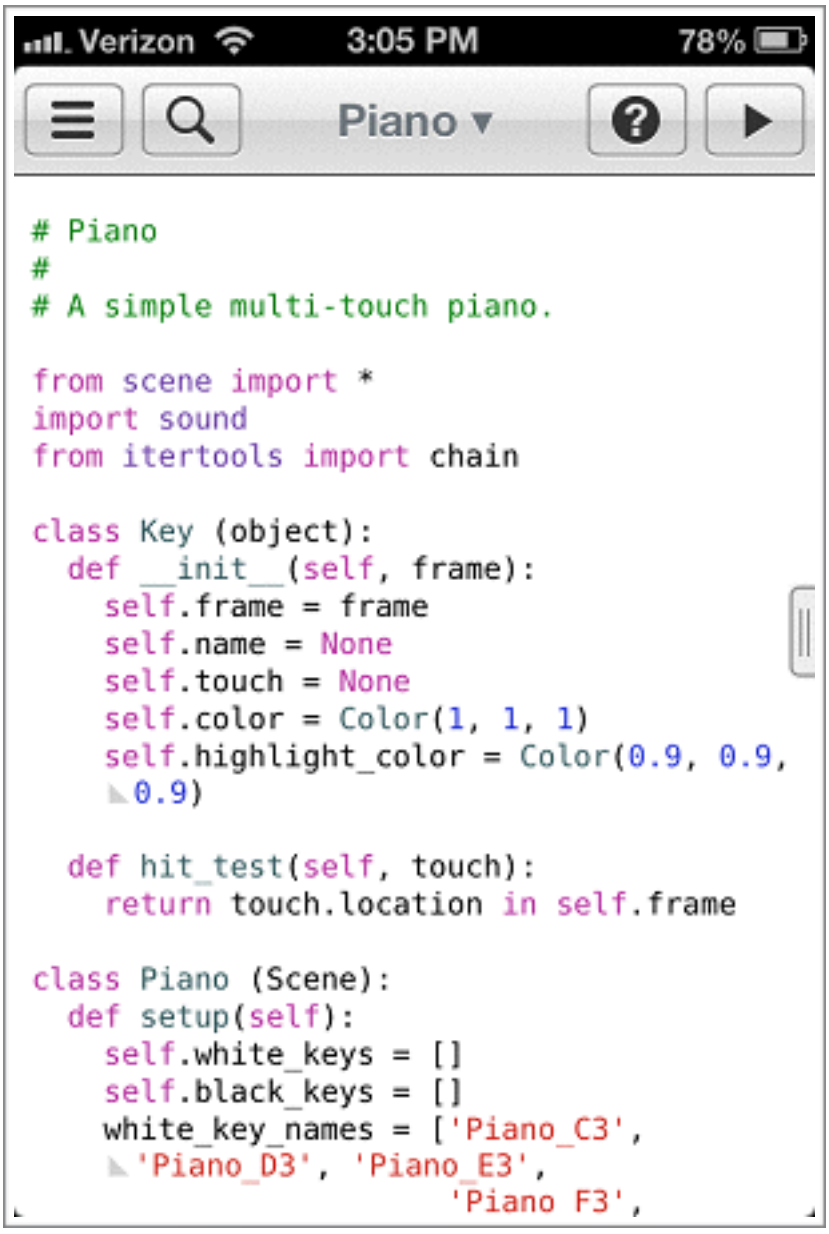
1．Pythonについて

●なんて読むの？：
パイソンと読みます。

●なんでPythonなのか？
・簡単にプログラミングできる
・幅広い分野で採用されている
・海外で人気。日本国内はこれから

●どんなところで使われている？
・インターネットのサービス
・ラズベリーパイ
（小型のコンピュータ）
・Pepper
（流行りのロボット）

●特徴
・インデント（段落）で処理の固まりを表現している（{}を使わない）



２．Pythonistaについて

本日は、**Pythonista**（パイソニスタ）というiPadのアプリケーション使用してプログラミングします（ちなみにパイソニスタとは、Pythonが好きな人のことを言います）。

操作方法は、別紙「Pythonistaの基本」を確認してください。

3．ゲーム制作基礎

今回は、最初に「なんちゃってシューティングゲーム」を作成します。

ゲーム内容：

プレイヤーを操作して、弾をインベーダーに当てて全て倒しましょう
時間制限以内に倒せないとゲームオーバーです。

操作方法：

iPadを横に傾けるとプレイヤーが移動します。
画面をタップすると弾が飛びます。

lesson1 プレイヤーを動かしてみる

下のコードを打ってプレイヤーを動かしてみましょう。完成したら、▶ボタンを押して実行してみましょう。____の箇所は「Tab」を押してください。

```
# -*- coding: utf-8 -*-
from scene import *

class GameScene (Scene):

____def setup(self):
________# 基礎のレイヤーを作成
        self.root_layer = Layer(self.bounds)
        center = self.bounds.center()

        # ゲーム用のレイヤーを作成
        self.game_layer = Layer(self.bounds)
        self.game_layer.background = Color(1, 30, 100)
        self.root_layer.add_layer(self.game_layer)

        # プレイヤーを描画
        self.x = center.x - 64
        self.y = 20
        image('Coffee', self.x, self.y, 100, 100)
        self.player = Rect( self.x, self.y, 100, 100 )

    def draw(self):

        center = self.bounds.center()
        background(0, 0, 0)

        #  重力値から移動量を算出
        g = gravity ()
        self.x += g.x * 100
        self.y = 20
        self.x = min ( self.size.w - 100, max( 0, self.x ))
        self.y = min ( self.size.h - 100, max( 0, self.y ))

        # プレイヤーを描画
        image('Coffee', self.x, self.y, 100, 100)
        self.player = Rect( self.x, self.y, 100, 100 )


run(GameScene())
```

lesson1の解説

• 変数

値や文字を入れておく入れ物のことです。以下の場合、”xx” が1を入れておく変数になります。
例：xx = 1

• クラスとメソッド

```
class GameScene (Scene): # これがクラス

     def setup(self): # これがメソッド
          ・・・
     def draw(self):
          ・・・
run(GameScene()) # クラスを実行します
```

クラスは**処理の固まり**、メソッドは**処理**を示します。

- GameSceneクラスは、今回のゲームの基本となるクラスです。
- GameSceneクラスには、**setup()**メソッドと**draw()**メソッドが必ずあります。
- setup()メソッドは、**最初に一度だけ**実行され、最初に一度だけ行う処理を書きます。
- draw()メソッドは、**毎秒に60回実行さ**れ、画面の表示が繰り返し行われます。

• Layer（レイヤー）とRect（レクト）

```
    def setup(self):
          # 基礎のレイヤーを作成
          self.root_layer = Layer(self.bounds)
          center = self.bounds.center()

          # ゲーム用のレイヤーを作成
          self.game_layer = Layer(self.bounds)
          self.game_layer.background = Color(1, 30, 100)
          self.root_layer.add_layer(self.game_layer)

          # プレイヤーを描画
          self.x = center.x - 64
          self.y = 20
          image('Coffee', self.x, self.y, 100, 100)
          self.player = Rect( self.x, self.y, 100, 100 )
```

レイヤーは、**画面で表示する面**を示します。元となるレイヤー（root_layer）の上に
実際にゲームで使用するレイヤー（game_layer）が**被さっている**イメージです（「図 Layerと
Rectの関係」を参照）。

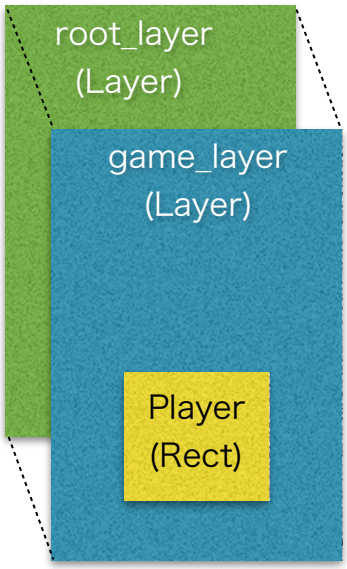


図 LayerとRectの関係

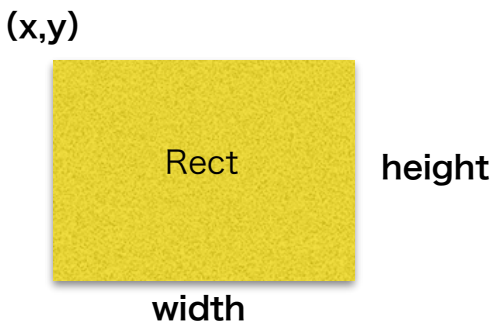


図 座標の考え方

上のコードで、レイヤーの生成とプレイヤーの初期表示を行っています。

Rectだけでは、画像の表示ができないので、同じ座標に**imageクラス**を使って画像を表示します。ここでは、コーヒーカップの画像を使用するので、'Coffee'を指定しています。

・Rectクラス

Rectクラスの書式は以下の書式となります。

`Rect(x座標, y座標, 幅, 高さ)`

・drawメソッド

このメソッドは、毎秒に60回実行され、画面の表示が繰り返し行われます。

```
    def draw(self):
        center = self.bounds.center()
        background(0, 0, 0)

        #  重力値から移動量を算出
        g = gravity ()
        self.x += g.x * 100
        self.y = 20
        self.x = min ( self.size.w - 100, max( 0, self.x ))
        self.y = min ( self.size.h - 100, max( 0, self.y ))

        # プレイヤーを描画
        image('Coffee', self.x, self.y, 100, 100)
        self.player = Rect( self.x, self.y, 100, 100 )
```

上のコードで、iPadの傾きによる重力値を**gravityクラス**で取得して、x,y座標としプレイヤーの再表示を繰り返し行っています。同じ画像がずっと表示しているように見えますが、繰り返し再表示がこのメソッドで行われています。

lesson2 インベーダーを並べてみよう

次にインベーターを並べてみましょう。

```
# -*- coding: utf-8 -*-
from scene import *

class GameScene (Scene):

     def setup(self):
          # 基礎のレイヤーを作成
          self.root_layer = Layer(self.bounds)
          center = self.bounds.center()
          .
          .
          # インベーダーデータを作成
          w = 110
          self.s_rect = []
          while(self.size.w > w):
               h = 110
               while(self.size.h > h ):
                    self.s_rect.append(Rect( w, h, 40, 40 ))
                    h += 160
               else:
                    w += 160

     def draw(self):
          center = self.bounds.center()
          background(0, 0, 0)
          .
          .
          # 障害物を描画
          for s in self.s_rect:
               image('Alien_Monster', s.x, s.y, s.w, s.h );


run(GameScene())
```

lesson2の解説

まず、setupメソッド内の以下の処理で、インベータのRectを**配列**にまとめています。配列は、まとめてデータを扱うのに適した方法です。

[0] Rect()
[1] Rect()
[2] Rect()
・
・
[n] Rect()

図 配列の考え方

```
# インベーダーデータを作成
w = 110
self.s_rect = []
while(self.size.w > w):
    h = 110
    while(self.size.h > h ):
        self.s_rect.append(Rect( w, h, 40, 40 ))
        h += 160
    else:
        w += 160
```

・for文

配列にまとめたものを1件づつ取り出して処理したい場合などに使用します。
以下の書式となります。

for 取り出した値 in 配列：

今回は、**drawメソッド**内で、先程作成した配列のデータを順に、取り出して画像を表示しています。

```
# 障害物を描画
for s in self.s_rect:
    image('Alien_Monster', s.x, s.y, s.w, s.h );
```

lesson3 弾を飛ばしてみよう

次に弾を飛ばしてみます。

```
# -*- coding: utf-8 -*-
from scene import *
import sound

class GameScene (Scene):
     BALL_INIT_Y = 40
     BALL_MAX_X = 1024
     BALL_MAX_Y = 748

    def setup(self):
        self.is_pushed = False # ボタン押していない

        # 基礎のレイヤーを作成
        self.root_layer = Layer(self.bounds)
        center = self.bounds.center()
         .
         .
        # 弾の座標
        self.ball_x = self.player.center().x - 16
        self.ball_y = self.BALL_INIT_Y

    def draw(self):
         .
         .
        # プレイヤーを描画
        image('Coffee', self.x, self.y, 100, 100)
        self.player = Rect( self.x, self.y, 100, 100 )

        # 弾を描画
        if self.is_pushed: # ボタン押し中
            # プレイヤーの位置から弾のX位置を再度計算
            self.ball_x = self.player.center().x - 16
            self.ball_y += 10
            image('Chestnut', self.ball_x, self.ball_y, 32, 32)
            self.ball = Rect(self.ball_x, self.ball_y, 32, 32)

            if self.ball_y >= self.BALL_MAX_Y: # 画面外への移動
                self.is_pushed = False # ボタン　押下　可
                self.ball_y = self.BALL_INIT_Y
```

```
    def touch_began(self, touch):
        center = self.bounds.center()

        # タッチした座標を保持
        x, y = touch.location.x, touch.location.y

        if not self.is_pushed: # ボタン押し中ではない
            self.is_pushed = True
            self.ball_y = self.BALL_INIT_Y #  弾を初期位置へ
            image('Chestnut', self.ball_x, self.ball_y, 32, 32)
            self.ball = Rect(self.ball_x, self.ball_y, 32, 32)
            sound.play_effect('Shot')

run(GameScene())
```

lesson3の解説

まず、setupメソッド内で弾の情報を初期化しています。
弾の発射位置は、プレイヤーの中心から発射されるように中心座標から弾の画像の半分のサイズずらして描画します。

`self.is_pushed`は、Booleanと言い、True（真）かFalse（偽）のいずれかの値が入ります。ここでは、**ボタンの押し状態**をこれで表現しています。

```
    def setup(self):
        self.is_pushed = False # ボタン押していない
        # 弾の座標
        self.ball_x = self.player.center().x - 16
        self.ball_y = self.BALL_INIT_Y
```

・touch_beganメソッド

画面をタッチした際のイベント（事象）を拾います。
タッチした座標を保持して、ボタンを押していない状態の場合は、弾の表示をします。
それと一緒に**ボタンを押した状態**にself.is_pushedの値を変更しています。

```
    def touch_began(self, touch):
        center = self.bounds.center()

        # タッチした座標を保持
        x, y = touch.location.x, touch.location.y

        if not self.is_pushed: # ボタン押し中ではない
            self.is_pushed = True
            self.ball_y = self.BALL_INIT_Y #  弾を初期位置へ
            image('Chestnut', self.ball_x, self.ball_y, 32, 32)
            self.ball = Rect(self.ball_x, self.ball_y, 32, 32)
            sound.play_effect('Shot')

run(GameScene())
```

・if文

条件によって処理を分岐させたいときにif文を使います。
以下の書式となります。

```
if 値が真の場合：
  処理・・
else: # ifにあてはまらない場合
  処理・・
```

lesson4 弾に当たったインベーダーを消してみよう

次に弾に当たったインベーダーを消してみましょう。

```
# -*- coding: utf-8 -*-
from scene import *
import sound

class GameScene (Scene):
        BALL_INIT_Y = 40
        BALL_MAX_X = 1024
        BALL_MAX_Y = 748

        def setup(self):
                self.is_pushed = False
                self.ball = None

                # 基礎のレイヤーを作成
                self.root_layer = Layer(self.bounds)
                center = self.bounds.center()
                 .
                 .

        def draw(self):
                center = self.bounds.center()
                background(0, 0, 0)

                # インベーダーを描画
                for s in self.s_rect:
                        image('Alien_Monster', s.x, s.y, s.w, s.h );
                 .
                 .
                # 弾が作成されていない場合はなにもしない
                if not self.ball: return

                # 弾とのあたり判定
                for i, s in enumerate( self.s_rect ):
                        if self.ball.intersects( s ):
                                # あたった四角を消す
                                del self.s_rect[i]
                                # 音をだす
                                sound.play_effect('Crashing')
```

lesson4の解説

・intersectsメソッド

intersectsメソッドを使用して、プレイヤーとインベーダーが当たったかを判定します。
当たっている場合は、True（真）となります。
今回は、lesson2で作成した**self.s_rect**をfor文を使用して、作成インベーダーがプレイヤーと当っているかを判定しています。
当っていた場合は、**del文**を使用して作成した配列から削除しています。

```
            # 弾とのあたり判定
            for i, s in enumerate( self.s_rect ):
                if self.ball.intersects( s ):
                    # あたった四角を消す
                    del self.s_rect[i]
                    # 音をだす
                    sound.play_effect('Crashing')
                    self.is_pushed = False # ボタン　押下　可
```

・del文

del文を使用して、弾の当たったインベーダー情報を配列(**self.s_rect**)から削除しています。
削除後の配列のイメージは以下のようになります。

例：Rect_2が弾と当たった場合

[0] Rect_0
[1] Rect_1
[2] Rect_2
・
・
[9] Rect_9

[0] Rect_0
[1] Rect_1
[2] Rect_3
・
・
[8] Rect_9

lesson5 点数を付けよう

弾がインベーダーに当たったら点数を付けてみましょう。

```
# -*- coding: utf-8 -*-
from scene import *
import sound

class GameScene (Scene):
     BALL_INIT_Y = 40
     BALL_MAX_X = 1024
     BALL_MAX_Y = 748

     # 文字の表示
     def text_display(self):
          w, h = self.size.w, self.size.h
          font = 'GillSans'
          h_x_pos = l_x_pos = w * 0.5

          h_text = u"スコア    :" + str(self.score)
          h_size = 40
          h_y_pos = h - 50

          # 点数を表示
          tint(1.0, 1.0, 1.0)
          text(h_text, font, h_size, h_x_pos, h_y_pos)

     def setup(self):
          self.is_pushed = False
          self.ball = None
          self.score = 0 # 点数を初期化

     def draw(self):
          center = self.bounds.center()
          background(0, 0, 0)

          # 点数を表示
          self.text_display()
          .
          .
          # 弾とのあたり判定
          if not self.ball: return
          for i, s in enumerate( self.s_rect ):
               if self.ball.intersects( s ):
                    .
                    .
                    self.score += 10 # 点数を加算
```

・textメソッド

text_displayメソッド内で、**textメソッド**を使用して点数表示をしています。点数は、**self.score**を使用して加算しています。

```
    def text_display(self):
        .
        .
        # 点数を表示
        tint(1.0, 1.0, 1.0)
        w, h = self.size.w, self.size.h
        text(u"スコア　："+ str(self.score), 'GillSans', 40, w * 0.5, h - 50)
```

点数の加算は、10点づつ行っています。
self.score += 10 は、self.score = self.score + 10と同様の意味です。

```
        # 弾とのあたり判定
        if not self.ball: return
        for i, s in enumerate( self.s_rect ):
            if self.ball.intersects( s ):
                # あたった四角を消す
                del self.s_rect[i]
                # 音をだす
                sound.play_effect('Crashing')
                self.is_pushed = False # ボタン　押下　可

                self.score += 10 # 点数を加算
```

lesson6 時間制限を付けよう

ゲーム要素を増やすために時間制限を付けてみましょう。

```
# -*- coding: utf-8 -*-
from scene import *
from threading import Timer
import sound

class GameScene (Scene):
    BALL_INIT_Y = 40
    BALL_MAX_X = 1024
    BALL_MAX_Y = 748

    # 文字の表示
    def text_display(self):
        .
        .
        h_text = u"スコア     :" + str(self.score)
        h_size = 40
        h_y_pos = h - 50

        l_text = u"残り：" + str(self.gameover_tot) + u"秒    "
        l_size = 40
        l_y_pos = h - 100

        # 点数を表示
        tint(1.0, 1.0, 1.0)
        text(h_text, font, h_size, h_x_pos, h_y_pos)

        # 残時間を表示
        tint(100.0, 100.0, 100.0)
        text(l_text, font, l_size, l_x_pos, l_y_pos)

    def setup(self):
        self.is_pushed = False
        self.ball = None
        self.score = 0
        self.gameover_tot = 20
        .
        .
```

```
        # インベーダーデータを作成
        w = 110
        self.s_rect = []
        while(self.size.w > w):
            h = 110
            while(self.size.h > h ):
                self.s_rect.append(Rect( w, h, 40, 40 ))
                h += 160
            else:
                w += 160

        # 1秒後に、self.callback_timer()を実行する
        self.timer = Timer(1.0, self.callback_timer)
        self.timer.start() # タイマー開始


# タイムアウトした際に呼ばれる関数
def callback_timer(self):
     self.gameover_tot = self.gameover_tot - 1
     if self.gameover_tot >= 0: # タイムアウト前
          self.timer = Timer(1.0, self.callback_timer)
          self.timer.start()
```

・Timerクラス

Timerクラスをすると、n秒経過後に、x処理を行うという処理を実行することができます。
以下の書式となります。
Timer(**秒**, **処理**)

今回は、1秒経過後に、self.callback_timer()の処理で、制限時間(self.gameover_tot)を減らしています。
まだ、self.gameover_totが0になっていない場合は、再度、1秒間タイマーを動作させて、処理を繰り返します。

lesson7 クリア画面、ゲームオーバー画面を作ろう

```
# -*- coding: utf-8 -*-
from scene import *
from threading import Timer
import sound

class GameScene (Scene):
    BALL_INIT_Y = 40
    BALL_MAX_X = 1024
    BALL_MAX_Y = 748

    # 文字の表示
    def text_display(self):
        w, h = self.size.w, self.size.h
        font = 'GillSans'
        h_x_pos = l_x_pos = w * 0.5
        if self.gamestatus == 1:
            h_text = u"スコア    :" + str(self.score)
            h_size = 40
            h_y_pos = h - 50

            l_text = u"残り：" + str(self.gameover_tot) + u"秒  "
            l_size = 40
            l_y_pos = h - 100
        else:
            if self.gamestatus == 2:
                h_text = u"ゲームクリアー \ ^o^  /"
            elif self.gamestatus == 3:
                h_text = u"ゲームオーバー orz..."
        h_size = 100
        h_y_pos = h - 300

        l_text = u"スコア    :"+ str(self.score)
        l_size = 40
        l_y_pos = h - 50

    # 点数を表示
    tint(1.0, 1.0, 1.0)
    text(h_text, font, h_size, h_x_pos, h_y_pos)
    # 残時間を表示
    tint(100.0, 100.0, 100.0)
    text(l_text, font, l_size, l_x_pos, l_y_pos)
```

```
    # ゲーム終了画面作成
    def create_gameend_layer(self):
        self.root_layer.remove_layer(self.game_layer)
        self.gameend_layer = Layer(self.bounds)

        if self.gamestatus ==2:  # ゲームクリアー
            self.gameend_layer.background = Color(200, 0, 100)
        elif self.gamestatus == 3: # ゲームオーバー
            self.gameend_layer.background = Color(200, 200, 0)
        self.root_layer.add_layer(self.gameend_layer)

    def setup(self):
        self.gamestatus = 1 # ゲーム中
        .
        .
    def draw(self):

        center = self.bounds.center()
        background(0, 0, 0)

        if self.gamestatus == 2: # ゲームクリア
            self.timer.cancel()
            self.gameend_layer.update(self.dt)
            self.gameend_layer.draw()
            self.text_display()
            return # 処理完了
        elif self.gamestatus == 3: # ゲームオーバー
            self.gameend_layer.update(self.dt)
            self.gameend_layer.draw()
            self.text_display()
            return # 処理完了

        # 点数と残り時間表示
        self.text_display()
        .
        .
        # 全部破壊したか
        if len(self.s_rect) == 0:
            self.timer.cancel()
            self.gamestatus = 2 # ゲームクリア
            self.create_gameend_layer()
```

```
    def touch_began(self, touch):
        if self.gamestatus != 1: return #ゲーム中でなければ何もしない
        .
        .
    def callback_timer(self):
        self.gameover_tot = self.gameover_tot - 1
        if self.gameover_tot >= 0: # タイムアウト前
            self.timer = Timer(1.0, self.callback_timer)
            self.timer.start()
        else:
            self.gamestatus = 3 # ゲームオーバー
            self.create_gameend_layer()
```

・ゲームの状態を管理する

ゲームの状態をself.gamestatusという変数で管理するようにします。
状態によって、レイヤーの切り替えを行います。

```
self.gamestatus = 1 # ゲーム中
self.gamestatus = 2 # ゲームクリア
self.gamestatus = 3 # ゲームオーバー
```

・ゲームクリアの条件

ゲームクリアの条件は、「画面上からインベーダーが全て消えた場合」としています。具体的な処理として、インベーダーの情報が入っている配列(self.s_rect)の長さが0であったらとしています。配列の長さを調べるのには、len()関数を使用します。

```
# 全部破壊したか
if len(self.s_rect) == 0:
      self.root_layer.remove_layer(self.game_layer)
      self.gameend_layer = Layer(self.bounds)
      self.gameend_layer.background = Color(200, 0, 100)

      self.timer.cancel()
      self.root_layer.add_layer()
      self.gamestatus = 2 # ゲームクリア
```

・ゲームオーバーの条件

ゲームオーバーの条件は、**「制限時間(self.gameover_tot)が終了した場合」としています。**
具体的な処理として、タイムアウト処理の中で、制限時間が、0未満になった場合としています。

```
    def callback_timer(self):
        self.gameover_tot = self.gameover_tot - 1
        if self.gameover_tot >= 0: # タイムアウト前
            self.timer = Timer(1.0, self.callback_timer)
            self.timer.start()
        else:
            self.root_layer.remove_layer(self.game_layer)
            self.gameend_layer = Layer(self.bounds)
            self.gameend_layer.background = Color(200, 200, 0)
            self.root_layer.add_layer(self.gameend_layer)

            self.gamestatus = 3 # ゲームオーバー
```

4. オリジナルゲームを作ってみよう

ゲーム作りの必要な要素は、サンプルゲームで紹介した機能で作成しました。
これらを組み合わせて、オリジナルゲームを作ってみましょう！！。