創建一個新的應用程序類

對於本教程的第一部分，我們希望 App 通過創建一個名為從標準類繼承其基本行為的**全新**應用程序類 MyApp 來自定義標準 App 類的行為。
將以下代碼段粘貼到該 myapp = ...行之前：

```
class SpaceGame(App):
    """
    Tutorial4 space game example.
    """
    def __init__(self):
        super().__init__()
```

然後**剪切**創建背景的四條線**並將**它們**粘貼**到 super()...線下方並縮進它們以匹配。最後，改變說零件 myapp.width 和 myapp.height 是 self.width 和 self.height 分別。現在，您的新代碼應該如下所示：

```
class SpaceGame(App):
    """
    Tutorial4 space game example.
    """
    def __init__(self):
        super().__init__()
        # Background
        black = Color(0, 1)
        noline = LineStyle(0, black)
        bg_asset = RectangleAsset(self.width, self.height, noline, black)
        bg = Sprite(bg_asset, (0,0))
```

是的，你可以從一開始就粘貼它，但我希望你能清楚這段代碼的*來源*。

有關變化的注意事項：

- class SpaceGame(App):定義一個名為的新*類*，SpaceGame 它 *繼承*了標準 App 類的所有功能。
- 下一行定義__init__了類的方法。在這種情況下，它不期望任何參數。
- 該 super().__init__(...行強制新的 SpaceGame 類__init__在開始自己的初始化之前調用標準 App 類的函數。如果您希望新類完全繼承父類的行為，請始終執行此操作。
- 最後，由於此代碼*初始化*遊戲，因此__init__在遊戲類的方法中放置用於創建黑色背景的代碼是有意義的。

目前，您的計劃已經破產。為了使新的 SpaceGame 類生效，我們必須*實例化*它而不是實例化 App 類。將程序的下一行更改 `myapp = App()`為 `myapp = SpaceGame()`。
嘗試運行該程序。你應該看到黑色背景。

創建一個新的 Sprite 類

在 SpaceGame 類定義的上方，粘貼這個新代碼：

```
class SpaceShip(Sprite):
    """
    Animated space ship
    """
    asset = ImageAsset("images/four_spaceship_by_albertov_with_thrust.png",
        Frame(227,0,65,125), 4, 'vertical')

    def __init__(self, position):
        super().__init__(SpaceShip.asset, position)
```

運行你的程序。它不應該做任何與以前不同的事情。創建一個新的 Sprite *類*實際上並不創建任何 sprite。它所做的就是創造一個製作精靈的*藍圖*。

通過在 SpaceGame `__init__`方法的末尾添加以下行來添加單個 SpaceShip 精靈（當然，正確縮進）：

```
        SpaceShip((100,100))
```

現在運行你的代碼。涼。嘗試在 SpaceGame `__init__`方法的末尾添加一些 SpaceShip 實例：

```
        SpaceShip((150,150))
        SpaceShip((200,50))
```

看起來你正在建造一支艦隊！

我們添加的代碼非常簡單，但有一行需要一些解釋：

```
    asset = ImageAsset("images/four_spaceship_by_albertov_with_thrust.png",
        Frame(227,0,65,125), 4, 'vertical')
```

該 `asset` 變量被創建*的類中*，但在任何方法以外。這使它成為一個*類*屬性，可供該類的所有*實例*使用。我們用它來調用父類雪碧`__init__`使用語法的方法：`SpaceShip.asset`。這種方法允許我們根據需要創建盡可能多的 SpaceShip 實例，但不創建多個資產。有**一個**物體代表宇宙飛船圖像，但有**多個**物體代表精靈。
這個創建一個 `ImageAsset` 參數的調用比我們在上一個教程中使用的參數多。這是他們的意思：

- 所述 `Frame(227,0,65,125)`參數指定的矩形截面*內*的圖像文件。如果你看一下 Github 中的圖像文件，你會發現它實際上由*十六個*不同的航天器圖像組成，一些有火箭推力，有些沒有。幀參數指的是我們想要的子圖像左上角的水平和垂直位置（227 和 0 像素），後面是它的寬度和高度（65 和 125 像素）。
- 該 4 論證意味著該資產實際上將包括與第一個相同大小的**四個**子圖像，並且……
- 該`'vertical'`參數意味著這四個圖像垂直排列在圖像文件中。回到 github 存儲庫，*看看*這個圖像文件，看看我的意思是"四個圖像……垂直排列"。

所有這些附加信息意味著此資產已準備好**動畫**！它由一個沒有推力的宇宙飛船和三個包含爆炸推力的宇宙飛船圖像組成。通過選擇我們想要在任何給定時間顯示哪些*幀*，我們可以在精靈本身內給出運動的外觀。

動畫一步！

首先，讓我們為 SpaceShip 類添加一些屬性。在 SpaceShip 類`__init__`方法的末尾添加以下行：

```
        self .vx =  1
        self .vy =  1
        self .vr =  0.01
```

這些將設置初始水平，垂直和旋轉速度。

然後，`step`向 SpaceShip 類添加一個方法。這應該出現在 SpaceShip 類`__init__`方法之後（但在它們之間留一個空格）：

```
    def step(self):
        self.x += self.vx
        self.y += self.vy
        self.rotation += self.vr
```

這將剛才添加的速度下精靈的位置屬性`x`，`y`和`rotation`，它們是被自動 Sprite 類的內置屬性*繼承*由飛船類。
如果您不確定粘貼這些代碼段的位置，請查看本頁末尾的完整列表。

不幸的是，只是向 `step`sprite 類添加一個方法並不意味著它將被調用。所以我們必須為 `step` 應用程序本身添加一個方法。`__init__`在 SpaceGame 類的方法下面添加以下代碼（但在它們之間留一個空格）：

```
    def step(self):
        for ship in self.getSpritesbyClass(SpaceShip):
            ship.step()
```

您**需要**為 `step` 自己的標準 App 類自定義添加方法。這種 `step` 方法**是**自動與遊戲中的每個視頻幀更新調用。

此方法體使用 `for` 循環來訪問 SpaceShip 類的每個*實例*，然後調用其 `step`method(`ship.step()`)。由於 `step` 每次視頻幀更新都會調用 SpaceGame 函數，這意味著每次更新視頻幀時 `step` 都會調用每個 SpaceShip 函數。

更改精靈圖像

現在來看動畫細節。當用戶按下空格鍵時，我們希望推力圖像具有動畫效果。所以這是我們要做的事情：

- 聽當空間按鈕被按下了 *下來*。
- *釋放*空格按鈕時監聽。
- 使用該 `step` 方法更改精靈圖像，具體取決於空間是否已關閉或已釋放。

首先，讓我們添加用於管理推進*狀態*的代碼，並聽取相應的密鑰。在 SpaceShip `__init__`方法的末尾添加以下內容：

```
        self .thrust =  0
        self .thrustframe =  1
        SpaceGame.listenKeyEvent（“ keydown ”，“ space ”，self .thrustOn）
        SpaceGame.listenKeyEvent（“ keyup ”，“ space ”，self .thrustOff）
```

然後將 `thrustOn`and 和 `thrustOff` 方法添加到 SpaceShip 類中。在 SpaceShip `step` 方法之後立即添加以下內容：

```
    def  thrustOn（self，event）：
         self .thrust =  1

    def  thrustOff（self，event）：
         self .thrust =  0
```

這些簡單的功能將跟踪空格鍵是否向下（推力為 1）或向上（推力為 0）。

最後，在 SpaceShip `step` 方法的末尾添加以下內容：

```
        # manage thrust animation
        if self.thrust == 1:
            self.setImage(self.thrustframe)
            self.thrustframe += 1
            if self.thrustframe == 4:
                self.thrustframe = 1
        else:
            self.setImage(0)
```

如果 `self.thrust` 設置為 1，則表示按下空格按鈕，精靈圖像設置為任何 `self.thrustframe` 值（記住我們在類`__init__`方法中將其初始化為 1 ）。圖像編號 0 是第一個圖像，1 是第二個圖像，依此類推。接下來的三行 *增加了*推力幀屬性，檢查它是否超出了我們的圖像列表的末尾（只有三個），並在必要時將其設置為 1。
最後，如果 `self.thrust` 設置為 0，則意味著空間按鈕被釋放，我們應該只顯示無推力的太空船圖像 `self.setImage(0)`。

最後細節

我們的遊戲還有很多改進，但這些改進留給學生練習！

你可能已經註意到宇宙飛船的精靈以一種非常奇怪的方式旋轉。這是因為精靈的默認"中心"實際上是它的左上角。您可以通過設置改變中心 `fxcenter` 和 `fycenter` 雪碧的屬性（或在我們的情況下，飛船）類。通過將此最後一行添加到 SpaceShip `__init__`方法來執行此操作：

```
        self .fxcenter =  self .fycenter =  0.5
```

再次運行你的程序，享受它的精彩！

問題

1. 擴展教程以使用`'left arrow'`和`'right arrow'`鍵左右旋轉船隻。
2. 擴展教程以使用 AWSD 鍵來控制船舶運動。
3. 高級：擴展教程以創建 `Blast` 使用文件夾中 `blast.png` 包含的圖像的精靈`/images`。使用該`'enter'`鍵在屏幕上隨機位置創建 Blast 精靈（或從太空飛船精靈"射擊"）。
4. 高級：擴展教程以動畫 `Blast` 精靈。

參考文獻:
https://github.com/HHS-IntroProgramming/Space-Shooter