PYTHON

Credit: https://github.com/bashkirtsevich-llc/PyBlackJack

Update and improve old blackjack code

Updating old projects can be fun and educational. **Andrew Smith** ensures that your cards are dealt correctly at any resolution.

OUR EXPERT

Andrew Smith is a software developer at NHS Digital, and has qualifications in software engineering and computer networks.

We're going to take a look at an old *Blackjack* project and see how we can update and modify it. *Blackjack* is a card game where the aim is to get a value total of 21 (or closest to 21) across the cards you're given, to win a round of the game.

The original project contains some interactive features that we can experiment with, such as buttons and events, along with some pleasing visual resources that we'll look at in this tutorial. The original project that this tutorial is based on was created by Allan Lavell, and the original source code can be retrieved from **https://github.com/bashkirtsevich-llc/PyBlackJack**.

During this tutorial we'll cover how the game program works and what resources are included for the project to work. We'll then look at positioning elements on the screen in relation to the screen resolution that the program is running in. We'll also add a button to the program, which will be used to exit the game.

Shuffling your cards

For this tutorial we'll install and set up the latest version of Python (3.10). For those that have Python/PyGame already installed, Python 3.8+ should be fine. Type the following code to install Python 3.10 and PyGame.

```
sudo apt-get install python3.10
sudo apt-get install python3-pip
python3.10 -m pip install pygame
```

Check both the Python and PyGame versions. Next, `git clone` from repository.

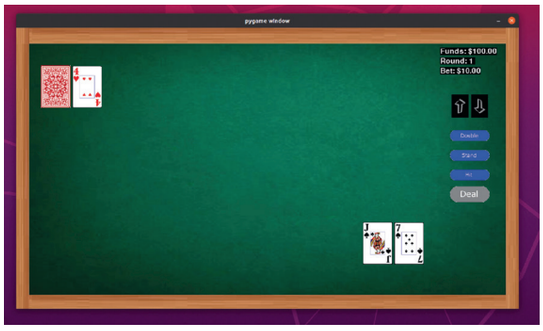
```
git clone https://github.com/asmith1979/lxf289_blackjack/
```

The project has been put into a folder called **PythonProjects**, which was created before downloading the project. Alternatively the source code and project can be retrieved from the **LXF289** archives at **https://linuxformat.com/archives**.

This tutorial will focus on the source code located in the **lxf289_blackjack** folder. Type `cd lxf289_blackjack` to open the folder and gain access to the Python source code. You'll see two Python source code files: **blackjack.py** and **lxf289ans.py**. The Python file **blackjack.py** is the Python script file that you'll be editing and **lxf289ans.py** contains the full tutorial code.

QUICK TIP

When choosing names for classes, functions or variables, make them meaningful so that someone else reading the code can see what they're intended for



A new game of Blackjack. The dealer's hand is shown at the top half of the screen, while the player's hand is at the bottom half of the screen.

To edit and view the source code you can either use your distro's default text editor or more specific programs such as *Notepad++*, *PyCharm* or *VS Code*. We'll be using *gedit* to view and edit the source files. If and when using this method to view/edit source files, it may be helpful to open two console windows. One for editing and viewing source files, and the other terminal window for executing the PyGame code.

Project overview

This project requires a lot of graphical resources for the program to work. For example, you'll find the full 52 deck of cards in the **/images/cards** folder. Navigate to this location on your machine to have a look at the card images that have been stored. You'll see that there's an image for each card in the deck as a well as a back-facing card image which is used in our *Blackjack* program. Card images have been created for each suite: Hearts, Diamonds, Clubs and Spades.

In the folder that's above this one – **images** – you'll see all the other image resources that are used in the game program. This includes the background image for the game and images of various buttons that the player can interact with during gameplay.

This Python project also uses sound to enhance the player experience. The sound effect file that we'll be using is located in the **sounds** folder. Inside this is the file **click2.wav**.

The code for the project consists of independent functions and object orientated programming techniques (classes). It's not the aim of this tutorial to cover all the code in-depth, but we will provide an overview of the main important parts that make the game program work and will bear some relevance with what's required later in this tutorial.

Because there are a lot of images used for this project, you may or may not have caught on to the fact that a lot of image loading takes place initially so that all image resources are ready for the game program to use. The function to used to load all images for the project is called `imageLoad`, which can be seen near the top of **blackjack.py**. Its code looks like the following:

```
def imageLoad(name, card):

    if card == 1:
        fullname = os.path.join("images/cards/", name)
    else:
        fullname = os.path.join('images', name)

    image = pygame.image.load(fullname)
    image = image.convert()

    return image, image.get_rect()
```

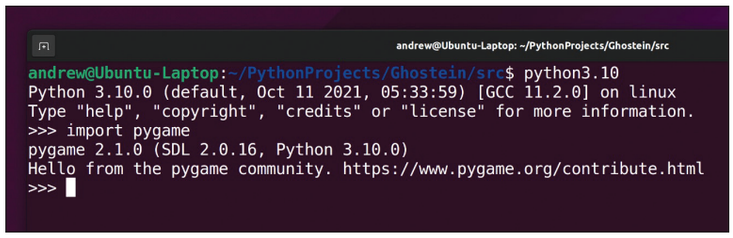
Overall, this function takes in two arguments: one is the file name and the other is an argument to identify whether or not a card image needs to be loaded. After loading the image into memory, the function returns both the image as an object as well as a rectangle surrounding the image. However, the function doesn't account for different screen resolutions that the program might be run in. This would result in, for example, the background image looking out of proportion with the rest of the program's visual elements. To correct this, we can make some changes to our program as well as to the `imageLoad` function as described below.

First, near the top of the program, after where the Python libraries are included, type the following code:

```
BACKGROUND_IMAGE = 0
CARD_IMAGE = 1
BUTTON_IMAGE = 2

SCREEN_WIDTH = 800
SCREEN_WIDTH = 600

# SCREEN_WIDTH = 1280
# SCREEN_HEIGHT = 718
```



The screenshot shows the setup and configuration of Python and PyGame after being correctly installed.

We now have constant values for the background image, card images and button images, which saves us having to use numeric values such as 0, 1 and 2. We can refer to them by name instead.

The next step is to rename the second argument passed into `imageLoad` from card to `imageTypeIn` because now we're trying to differentiate between three types of image – the background image, card image and button image – the details of which can be passed into the `imageLoad` function. Values for the screen dimensions have also been added here, where one set of screen dimensions have been added but commented out. This enables us to experiment with different screen resolutions for the program to run in.

Not too far down (or up) from this, is a line that reads `screen = pygame.display.set_mode((1280, 718))`. Replace `1280` with `SCREEN_WIDTH` and `718` with `SCREEN_HEIGHT` (or commonly known as substituting).

The next step is to add the following code into the `imageLoad` function itself after the line `image = pygame.image.load(fullname)`:

```
if imageTypeIn == BACKGROUND_IMAGE:
    image = pygame.transform.scale(image, (SCREEN_WIDTH, SCREEN_HEIGHT))
```

This code will expand any image loaded in as a `BACKGROUND_IMAGE` to the screen dimensions that the program has been told to run at. To test this, write the following code just after where you specified `BACKGROUND_IMAGE, CARD_IMAGE`, etc. The function now should look something like the following:

```
def imageLoad(name, imageTypeIn):
    global SCREEN_WIDTH
```

QUICK TIP

It's useful to look at existing examples of classes and functions as a guide as to how write your own.

» OBJECT ORIENTATED PROGRAMMING (OOP)

There have been quite a few object orientated programming techniques used in the project. For example, classes have been used to create all the buttons we interact with in the game program: Hit Button, Deal Button, Stand Button... etc. Each button class contains various properties and functions for the necessary operations to take place. Below we look at the dealButton class as an example. Only the main parts of the class are shown to demonstrate the main concepts and ideas.

```
class dealButton(pygame.sprite.Sprite):
    def __init__(self):
        pygame.sprite.Sprite.__init__(self)
        self.image, self.rect = imageLoad("deal.png", 0)
        self.position = (1155, 425)

    def update(self, mX, mY, deck, deadDeck, roundEnd, cardSprite, cards, playerHand, dealerHand, dCardPos, pCardPos, displayFont, playerCards, click, handsPlayed):
        # Get rid of the in between-hands chatter
        textFont = pygame.font.Font(None, 28)
        ....
```

A class is first declared using the class keyword as demonstrated above, `class dealButton` in this case. The next step is to write out the class constructor identified by `def __init__(self)`, which is used to initialise properties (or variables) to certain values when an instance of the class is created. In the above example, the class constructor is used to load in an image of the deal button and also set the position of the button. Also in this class is an update method (or function) that's been implemented to update positions during gameplay.

```
    global SCREEN_HEIGHT

    if imageTypeIn == CARD_IMAGE:
        fullname = os.path.join("images/cards/", name)
    else:
        fullname = os.path.join('images', name)

    image = pygame.image.load(fullname)

    if imageTypeIn == BACKGROUND_IMAGE:
        image = pygame.transform.scale(image, (SCREEN_
WIDTH, SCREEN_HEIGHT))

    image = image.convert()

    return image, image.get_rect()
```

Because `SCREEN_WIDTH and SCREEN_HEIGHT` are global variables, we include them into this function by using the `global` keyword. To test that our code works, run the program with the existing screen settings and then run the program again, commenting out the existing `SCREEN_WIDTH` and `SCREEN_HEIGHT` and removing comments from the other `SCREEN_WIDTH` and `SCREEN_HEIGHT` before the program is run.

What you're looking for in the program output is that the background image loaded in looks the same as in each screen settings. The buttons may appear funny and out of position when running at a different screen resolution, but we'll fix this later. To exit the program you'll have to press Ctrl+C because there's no way to formally exit the game. Later in this tutorial we'll be adding a button that will enable you to quit the game.

Next we'll add code that will position the buttons correctly relative to the screen resolution that's being used. However, before we add this code we could do with amending some of the existing code to prepare for this. Each of the buttons that have been created each have a class of their own (an object orientated programming concept), which you'll see declared in the program as `hitButton`, `standButton`, `dealButton`, `doubleButton`, `betUpButton` and `betDownButton`. Each of these classes contain three calls to the image load function. As an example, let's briefly look at the class `betDownButton`.

```
class betButtonUp(pygame.sprite.Sprite):

    def __init__(self, positionXIn, positionYIn):
        pygame.sprite.Sprite.__init__(self)
        self.image, self.rect = imageLoad("up.png", 
BUTTON_IMAGE)

....

    def update(self, mX, mY, bet, funds, click, roundEnd):
        if roundEnd == 1: self.image, self.rect = 
imageLoad("up.png", BUTTON_IMAGE)
        else: self.image, self.rect = imageLoad("up-grey.
png", BUTTON_IMAGE)
```

You should see something like the following when you view the code in your chosen IDE. Where `BUTTON_IMAGE` is, there should be a number 0, so replace this with `BUTTON_IMAGE` as in the example just shown. Do the same for the other classes described above (`hitButton`, `standButton`, `dealButton`, etc). After you've done this, save and run the program and you should see that buttons are now to scale (but could be positioned off-screen depending on screen resolution you are using). Even though the scaling issue with the button images is now fixed, the position of the buttons is not, because they'll only appear correct if using the screen resolution 1,280x718. We'll do something about this next.

We're now going to implement a class of our own called `buttonPositioningValues`. The purpose of this class or (data structure) is to position all the buttons relative to the screen resolution that's been set. This means regardless of what screen resolution is used, the buttons will always appear in a constant position.

Near the top of **blackjack.py** after the constant variables have been defined (`SCREEN_WIDTH`, etc.), write out the following code:

```
# UI Button Positioning variables
class buttonPositioningValues:

    # Class Constructor
    def __init__(self):
        self.myvalue = 1

    # Get the screen dimensions
    screenWidth = SCREEN_WIDTH
    screenHeight = SCREEN_HEIGHT

    # Set Deal Button Position
    dealButtonPosition_X = (screenWidth-125)
    dealButtonPosition_Y = (screenHeight-193)

    # Set Hit Button Position
    hitButtonPosition_X = (screenWidth-125)
    hitButtonPosition_Y = (screenHeight-243)

    # Set Stand Button Position
    standButtonPosition_X = (screenWidth-125)
    standButtonPosition_Y = (screenHeight-280)

    # Set Double Button Position
    doubleButtonPosition_X = (screenWidth-125)
    doubleButtonPosition_Y = (screenHeight-317)

    # Set Up Arrow Position
    upArrowButtonPosition_X = (screenWidth-150)
    upArrowButtonPosition_Y = (screenHeight-370)
```

Here, the dealer has won a round at Blackjack and that the player has lost and has also lost $10. Once a round is over, the player can only deal.

We've added a Quit button to the game so the player can leave at any time. This saves the player having to press Ctrl+C to exit the program.

```
# Set Down Arrow Position
downArrowButtonPosition_X = (screenWidth-100)
downArrowButtonPosition_Y = (screenHeight-370)

# Quit Button Position
quitButtonPosition_X = 100
quitButtonPosition_Y = (screenHeight-100)
```

You'll notice from the code we've just written that the X and Y position values are relative to the screen width and screen height, and are offset by a certain value to position them on the screen. Now that our class has been written out we now need to declare what's known as an instance of a class, so that the properties of the class can be accessed. Before we do this, scroll down to just under where there's a comment that says INITIALIZATION BEGINS and type the following code:

```
uiButtonPosition = buttonPositioningValues()
```

The next step from this is to alter the existing class constructors of the buttons used in the game program. The below is example of what you'll need to do to each button class.

```
bbU = betButtonUp(uiButtonPosition.
upArrowButtonPosition_X, uiButtonPosition.
upArrowButtonPosition_Y)
bbD = betButtonDown(uiButtonPosition.
downArrowButtonPosition_X, uiButtonPosition.
downArrowButtonPosition_Y)
```

As can be seen from this code, the class constructors of `betButtonUp` and `betButtonDown` have been altered to accept X and Y positions for the relevant buttons declared in our button-positioning class. Before continuing with the tutorial, do the same for each of the other button classes. The button classes themselves also need to be altered for each button featured in the game program. Below is an example of how it's done for the `dealButton` class:

```
class dealButton
...

def __init__(self, positionXIn, positionYIn):
    pygame.sprite.Sprite.__init__(self)
    self.image, self.rect = imageLoad("deal.png",
BUTTON_IMAGE)
    self.position = (positionXIn, positionYIn)

    # 2. Set the position values as part of class
    self.xPos = positionXIn
    self.yPos = positionYIn

def update(....)
    # 3. Pass the position values into the position
    self.position = (self.xPos, self.yPos)
```

The first step in altering the class is to change the constructor of the class as above to take in `positionXIn` and `positionYIn`. The rest of the changes are identified by comments in the above code example. Before continuing with the tutorial, do the same to the other button classes for the game program.

When you've done all the changes to each class, save your changes and execute the program. You'll see that all the buttons have been positioned relative to the screen resolution. Because there's quite a bit of code to alter, check the python file **lxf289ans.py**. The answer source code is written in there to see if your code is consistent with the answer... no cheating though! The text that's displayed in the game program is positioned relative to the button buttons (see **lxf289ans.py**).

Finally, we need to add a button to quit the game program rather than just pressing Ctrl+C. The image resource for a quit button has already been created in the image folder location. In the answer Python file, **lxf289ans.py**, there's a class created called `quitButton` that's used for the display and operation of the quit button in the game. Going over what we've covered in this tutorial, see if you can work out how it's been written, before looking at **lxf289ans.py**. To help you, have a look at the existing button classes that have been written to see if it'll provide you with any clues.

Instead of scrolling through code in an IDE, use the search facilities available to go direction to the function or variable directly. LXF

» GAMEPLAY FUNCTIONS

Even though object orientated programming has been implemented in this project, these individual functions have been used to help with the gameplay of *Blackjack*:

- `imageLoad` for loading an image resource.
- `soundLoad` used to load a sound resource.
- `display` this displays text on the screen.
- `playClick` for playing a sound resource once loaded.
- `gameOver` used to display the game-over screen.
- `shuffle` use this to shuffle the deck of cards using the Fisher-Yates algorithm, which helps eliminate the element of predictability.
- `createDeck` this creates the logical structure of deck of cards.
- `returnFromDead` when the main deck of cards has been emptied.
- `deckDeal` used to shuffle the deck.
- `hit` for taking cards from the deck.
- `checkValue` this checks the value of the hand of player and dealer.
- `blackJack` for seeing if blackjack has been achieved either by the player or dealer.
- `bust` used when player is bust.
- `endRound` determines what happens with the cards at the end of a round.
- `compareHands` this is used at the end and beginning of a round to compare hands of player and dealer.