

Drawing and moving stuff



Admin



- Project Questions?
- Quiz

So far what have we drawn?



- This semester what have we drawn?
 - Lucky Volunteer?

So far what have we drawn?



- This semester what have we drawn?
 - Shapes, that we can use for lots of useful things like graphs
 - And use to draw pictures.

Let's move past basic shapes.



- In week 4/Project3 we drew shapes
 - But it would be more useful to draw pictures
 - You can get a lot more detail
 - And outsource the art to the art people

Image file formats



- There are lots of types of image files
 - Like what?

Image file formats



- There are lots of types of image files
 - Like what?
 - png, gif, webp
 - Bmp, jpeg, tiff, svg
 - And what is the difference between the first and second lines?

Image file formats



- There are lots of types of image files
 - Like what?
 - png, gif, webp
 - Bmp, jpeg, tiff, svg
 - And what is the difference between the first and second lines?
 - Transparent background
 - I'll push you toward png since it works with dearpygui on all platforms.

First setup the window



- Let's first draw a window with a blue background
 - Given what we know now, how do we do that?
 - Lucky volunteer

First setup the window



- Let's first draw a window with a blue background
 - Given what we know now, how do we do that?
- Put up a window and draw a blue rectangle over the whole thing right?
- Is there anything here that looks new or unfamiliar?
- Let's try it.

```
import dearpygui.dearpygui as dpg
import comp151Colors
```

```
dpg.create_context()
WINDOW_SIZE = 800
with dpg.window(label="Tutorial"):
    with dpg.drawlist(width=WINDOW_SIZE,
height=WINDOW_SIZE):
        dpg.draw_rectangle((0,0), (WINDOW_SIZE, WINDOW_SIZE),
fill=comp151Colors.LIGHT_BLUE)
```

```
dpg.create_viewport(title='Drawing Demo',
width=WINDOW_SIZE, height=WINDOW_SIZE)
dpg.setup_dearpygui()
dpg.show_viewport()
dpg.start_dearpygui()
dpg.destroy_context()
```

Now let's add an image



- As we add the images,
 - Grab the ship and gold pile images from the resources page of the website
 - Put them in the project as before.
 - In the main project folder for this project

Now let's add an image



- To put an image on a dearpygui window, we need three parts
 - Load the image
 - Add the image to the 'texture registry' that dearpygui uses
 - Actually draw the image in the draw list.

Load the image



- To load the image
 - Use `dearpygui.load_image`
 - Takes one parameter – the name of the file
 - Returns four things
 - Width of the image
 - Height of the image
 - Channels (color information)
 - The actual image in a format that dearpygui likes.

- Examples of loading image

```
width, height, channels, dpg_shippict =  
dpg.load_image('ship.png')
```

```
pile_width, pile_heigh, pile_channels, pile_data =  
dpg.load_image('gold-pile.png')
```

- —

Add to the texture registry



- You will add all of your images to a single texture registry.
 - Holds all of the images for display on this window
 - Use `dearpygui.add_static_texture` (there is also an `add_dynamic_texture`)
 - Use four parameters, three positional and one by keyword
 - Positional: width and height and picture in dearpygui format
 - Keyword `tag='name'`
 - Name is what you will use to draw this later.
- Example

```
with dpg.texture_registry():  
    dpg.add_static_texture(width, height, dpg.shippict, tag="ship_pict")  
    dpg.add_static_texture(pile_width, pile_heigh, pile_data, tag="gold_pile")
```

Drawing the picture



- To draw, in your drawlist add a call to `draw_image`
 - Takes three positional arguments
 - The name of the texture tag/image to draw
 - The upper left corner to draw the image
 - The lower right of the image to draw
 - This lets us scale the image easily.

- Example

```
dpg.draw_image("ship_pict", (xloc, yloc), (xloc+width, yloc+height))
```

Let's draw this ship



- Code – new code since last time highlighted. (try scaling ship)

```
import dearpygui.dearpygui as dpg
import comp151Colors
dpg.create_context()
WINDOW_SIZE = 800
xloc = 100
yloc = 100
width, height, channels, dpg_shippict = dpg.load_image('ship.png') # 0: width, 1: height, 2: channels, 3: data
pile_width, pile_height, pile_channels, pile_data = dpg.load_image('gold-pile.png')
with dpg.texture_registry():
    dpg.add_static_texture(width, height, dpg_shippict, tag="ship_pict")
    dpg.add_static_texture(pile_width, pile_height, pile_data, tag="gold_pile")
with dpg.window(label="Tutorial"):
    with dpg.drawlist(width=WINDOW_SIZE, height=WINDOW_SIZE):
        dpg.draw_rectangle((0,0), (WINDOW_SIZE, WINDOW_SIZE), fill=comp151Colors.LIGHT_BLUE)
        dpg.draw_image("ship_pict", (xloc, yloc), (xloc+width, yloc+height))
dpg.create_viewport(title='Custom Title', width=800, height=600)
dpg.setup_dearpygui()
dpg.show_viewport()
dpg.start_dearpygui()
dpg.destroy_context()
```


Now let's move the ship



- First pass of moving the ship
 - Add a tag to the draw image

```
dpg.draw_image("ship_pict", (xloc, yloc), (xloc+width, yloc+height), tag="ship_update")
```

- Then after the show_viewport do this

```
while dpg.is_dearpygui_running():  
    xloc += 0.5  
    dpg.configure_item('ship_update', pmin=(xloc, yloc), pmax=(xloc + width, yloc + height))  
    dpg.render_dearpygui_frame()
```

- So lets try it and see.

Now let's move the ship



- First pass of moving the ship
 - Add a tag to the draw image

```
dpg.draw_image("ship_pict", (xloc, yloc), (xloc+width, yloc+height), tag="ship_update")
```

- Then after the show_viewport do this

```
while dpg.is_dearpygui_running():  
    xloc += 0.5  
    dpg.configure_item('ship_update', pmin=(xloc, yloc), pmax=(xloc + width, yloc + height))  
    dpg.render_dearpygui_frame()
```

- So lets try it and see.
 - Hmmmmmm, how do we fix that?
 - Shall we bounce it or reset it to the left?

Animation vs a program



- So far this is pretty much an animation
 - Like a movie
- What do we need to add to make this a proper program?
 - Lucky volunteer?

Animation vs a program



- So far this is pretty much an animation
 - Like a movie
- What do we need to add to make this a proper program?
 - Some sort of input
 - And what could we use for this kind of program?
 - Volunteer?

Animation vs a program



- So far this is pretty much an animation
 - Like a movie
- What do we need to add to make this a proper program?
 - Some sort of input
 - And what could we use for this kind of program?
 - Either mouse or keyboard.

Keyboard input



- In Dearpygui,
 - Keyboard input needs two things
 - Register the keyhandler
 - The key handler itself

Keyboard input



- In Dearpygui, Keyboard input needs two things

- Register the keyhandler

Put this near the texture registry

```
with dpkg.handler_registry():  
    dpkg.add_key_press_handler(callback=move_ship)
```

The **name** of the function to handle the key press

- The key handler itself For example

```
def move_ship(sender, app_data):
```

```
    key = app_data
```

```
    global xloc, yloc
```

```
    if key == dpkg.mvKey_Left:
```

```
        xloc -= speed
```

```
    elif key == dpkg.mvKey_Right:
```

```
        xloc += speed
```

```
    elif key == dpkg.mvKey_Up:
```

```
        yloc -= speed
```

```
    elif key == dpkg.mvKey_Down:
```

```
        yloc += speed
```

```
    with dpkg.mutex():
```

```
        dpkg.configure_item("ship_update", pmin=(xloc, yloc), pmax=(xloc + width, yloc + height))
```

You can name your function anything that makes sense but these two params are required
The app_data for key callbacks is the key pressed

There are a bunch of key constants mvKey_XXX we can check against

dpkg.mutex() protects against updating the image in two places at once. configure_item updates the image loc

Putting it all together



- Followed by

```
import dearpygui.dearpygui as dpg
import comp151Colors
```

```
dpg.create_context()
WINDOW_SIZE = 800
```

```
xloc = 100
```

```
yloc = 100
```

```
speed = 3
```

```
def move_ship(sender, app_data):
```

```
    key = app_data
```

```
    global xloc, yloc
```

```
    if key == dpg.mvKey_Left:
```

```
        xloc -= speed
```

```
    elif key == dpg.mvKey_Right:
```

```
        xloc += speed
```

```
    elif key == dpg.mvKey_Up:
```

```
        yloc -= speed
```

```
    elif key == dpg.mvKey_Down:
```

```
        yloc += speed
```

```
    with dpg.mutex():
```

```
        dpg.configure_item("ship_update", pmin=(xloc, yloc), pmax=(xloc + width, yloc + height))
```

- ```
width, height, channels, dpg_shippict = dpg.load_image('ship.png') # 0: width, 1: height, 2: channels, 3: data
pile_width, pile_heigh, pile_channels, pile_data = dpg.load_image('gold-pile.png')
dpg.create_viewport(title='Custom Title', width=800, height=600)

with dpg.texture_registry():
 ship_id = dpg.add_dynamic_texture(width, height, dpg_shippict, tag="ship_pict")
 dpg.add_static_texture(pile_width, pile_heigh, pile_data, tag="gold_pile")
with dpg.handler_registry():
 dpg.add_key_press_handler(callback=move_ship)
dpg.set_frame_callback(dpg.get_frame_count()+1, move_ship)
with dpg.window(label="Tutorial"):
 with dpg.drawlist(width=WINDOW_SIZE, height=WINDOW_SIZE):
 dpg.draw_rectangle((0,0), (WINDOW_SIZE, WINDOW_SIZE),
 fill=comp151Colors.LIGHT_BLUE)
 dpg.draw_image("ship_pict", (xloc, yloc), (xloc+width, yloc+height),
 tag="ship_update")

dpg.setup_dearpygui()
dpg.show_viewport()
dpg.start_dearpygui()
dpg.destroy_context()
```



## Now let's scatter some gold piles around



- To put them around – need to generate 'random' numbers

# Random Numbers



- Random vs 'pseudo-random' numbers
- Python random package is in standard library
- Several useful functions
  - `random.randint(a, b)`
    - Returns a random integer from a to b
  - `random.choice(seq)`
    - Returns a randomly chosen element in the sequence seq (usually a list)
  - `random.uniform(a, b)`
    - Return a random floating point number N from a to b

## Placing the gold piles



- Let's place the gold piles randomly around
  - Make a list of the gold piles
  - How will we represent these gold piles?

# Placing the gold piles



- Let's place the gold piles randomly around
  - Make a list of the gold piles
  - How will we represent these gold piles?
  - Possible
    - A list of tuples (representing upper left hand corners)
    - A list of dictionaries with corner location and more
      - Value of gold piles?
    - Lets use the tuples approach for now.



```
def create_gold_piles(number):
 piles = []
 for i in range(number):
 pile_location = (random.randint(1, WINDOW_SIZE),
random.randint(1, WINDOW_SIZE))
 piles.append(pile_location)
 return piles
```

```
gold_piles = create_gold_piles(5)
```

I put this code near the top of the file below the WINDOW\_SIZE constant

- Then in the with drawlist

```
count = 0
for pile_location in gold_piles:
 dpd.draw_image("gold_pile", (pile_location[0],
pile_location[1]), (pile_location[0]+pile_width,
pile_location[1]+pile_height), tag=f"gold_pile{count}")
 count += 1
```

The highlighted section is all one line.

# Count the gold piles you hit



- We might want to remove gold piles when 'collected'
  - Put a little more complicated
  - And not needed for project 7
  - So lets just count the ones we hit
- The next thing we need in order to count number hit is?
  - Lucky volunteer

# Count the gold piles you hit



- We might want to remove gold piles when 'collected'
  - Put a little more complicated
  - And not needed for project 7
  - So lets just count the ones we hit
- The next thing we need in order to count number hit is?
  - Draw text on the screen
  - Check for collisions
- Both are reasonable next steps

- Let's do the text first.
- First we will create a new variable near top of file

```
number_of_piles_collected = 0
```

- Now we need to draw the text.
    - In the drawlist add something like
- ```
dpg.draw_text((20, 20),  
f"You've collected {number_of_piles_collected} gold piles",  
color=comp151Colors.DARK_RED, size=22)
```
- We will update later

Check overlap



- We will check for overlap of the two rectangles for the ship and gold pile images
 - Use a simple check from
 - <https://www.geeksforgeeks.org/dsa/find-two-rectangles-overlap/>
 - We will adjust the example there to work with tuples since we haven't done classes.
- Note that the original assumed the y-origin was at the bottom, I have adjusted that here
 - Go ahead and grab this wholesale

```
def do_overlap(l1, r1, l2, r2):  
    # If one rectangle is to the left of the other  
    if l1[0] > r2[0] or l2[0] > r1[0]:  
        return False  
  
    # If one rectangle is above the other  
    if r1[1] < l2[1] or r2[1] < l1[1]:  
        return False  
  
    return True
```


One last bit



- To finish our example, let's add a 'horizon'
 - Place where the sea meets the evening sky.
 - So we will shrink our blue rectangle and add a light orange one.
 - And for the fun of it I added a setting sun
- At the top of the file near the ship locations
- Update the blue rectangle code to

```
dpg.draw_rectangle((0, 0), (WINDOW_SIZE, horizon_yloc),  
fill=comp151Colors.LIGHT_ORANGE)
```

```
dpg.draw_circle((WINDOW_SIZE / 2, horizon_yloc), 70,  
color=comp151Colors.RED, fill=comp151Colors.ORANGE)
```

```
dpg.draw_rectangle((0, horizon_yloc), (WINDOW_SIZE,  
WINDOW_SIZE), fill=comp151Colors.LIGHT_BLUE)
```

```
horizon_yloc = 150
```

Sailing off into the sunset



- Now let's tell the user they are sailing off into the sunset if the ship enters the sky area
- To do this, first draw some blank text,
 - put it right under the gold count text

```
dpg.draw_text((200, 100), "",  
color=comp151Colors.DARK_RED, size=20,  
tag="message_update")
```

- Then let's add a function to return text based on where the ship is (no text for sea)

```
def get_message():  
    if yloc > horizon_yloc:  
        return ""  
    else: # the player has travelled up into the sky area  
        return "You've sailed off into the sunset"
```

- Finally, in our move player

```
good_bye_text = get_message()
```

- Then in the with mutex section

```
dpg.configure_item("message_update", text=good_bye_text)
```

- Let's try it

Questions?

