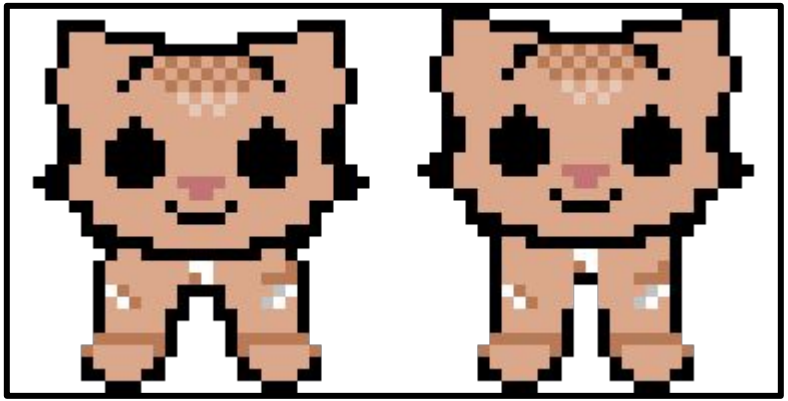
Philipps Universität Marburg

Animating Sprites

What are sprite sheets?

Sprite sheets

- On the right is one image file.
- The file represents two different sprite images.
- We refer to this as a “Sprite sheet.”

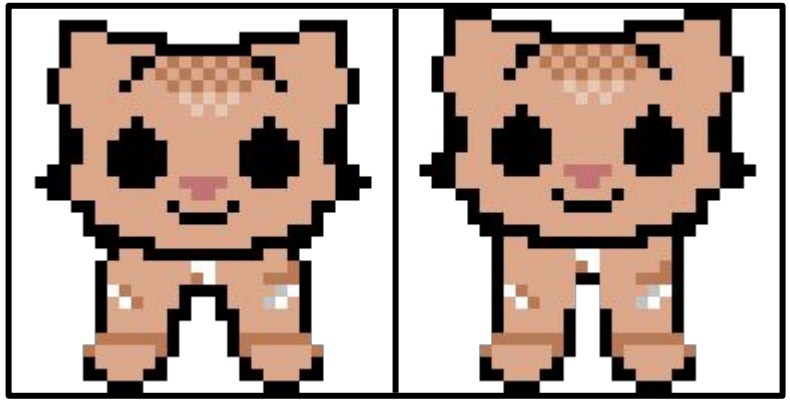


Sprite sheets: sprite images

- We know that the sprite sheet has two images separated by columns.
- The dimensions of each sprite image are:

Width = (image file width) / #columns

Height = image file height

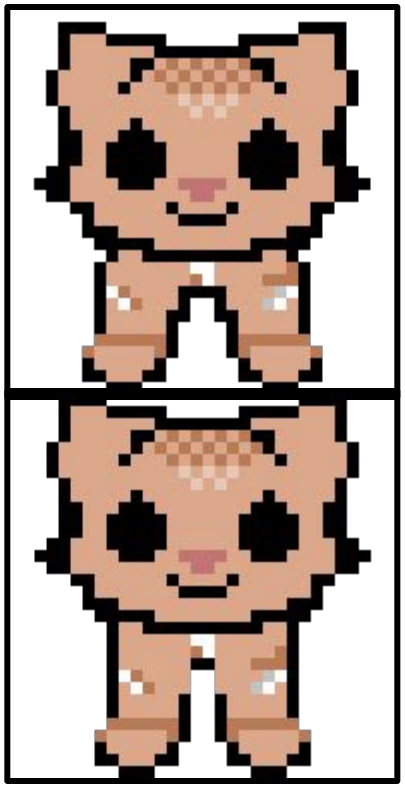


Sprite sheets: sprite images

- Alternatively, they could be separated by rows
- The dimensions of each sprite image are:

Width = image file width

Height = (image file height) / (#rows)

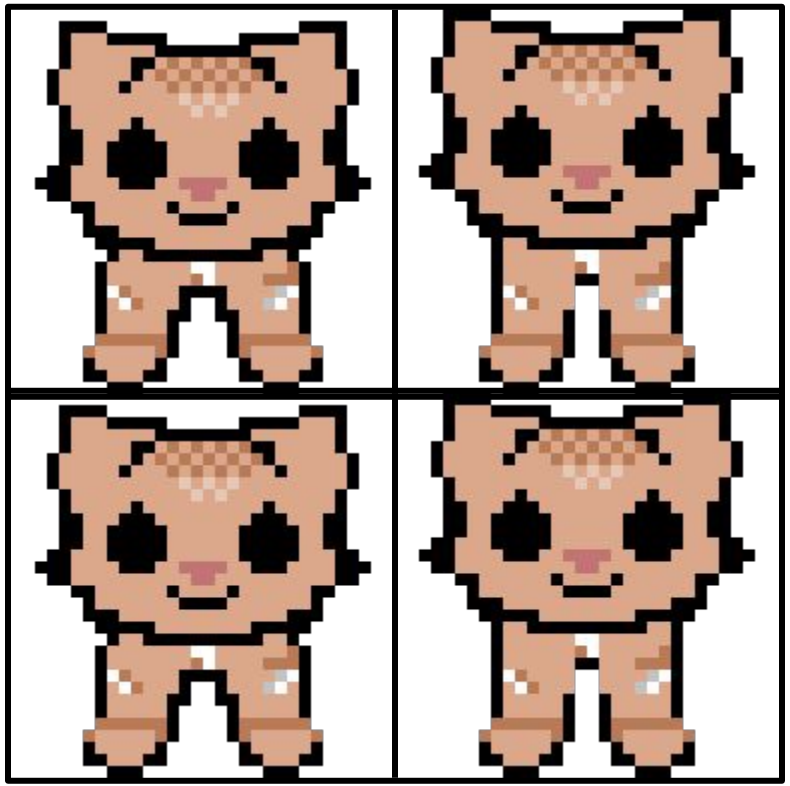


Sprite sheets: sprite images

- There can also be rows *and* columns
- The dimensions of each sprite image are:

Width = (image file width) / (#columns)

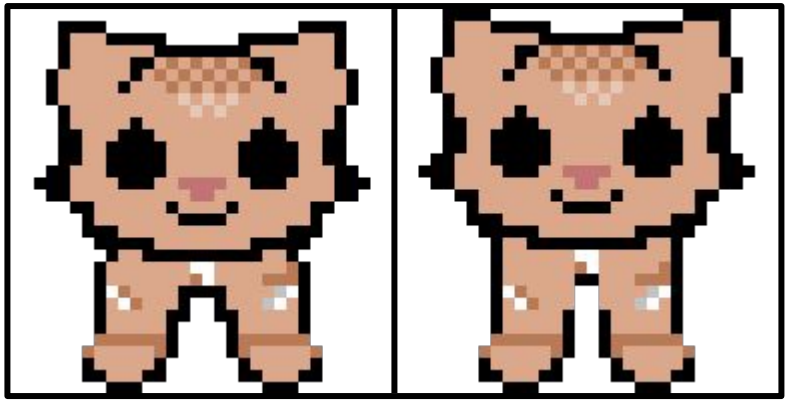
Height = (image file height) / (#rows)



How can we get just one sprite image of a sprite sheet?

Our sprite sheet

- For now, this is our sprite sheet
- The name of the file is “kitty-spritesheet.png”



Surfaces

- An image is a pygame.Surface object

```
""" In tutorial_notebook.ipynb """

import pygame

spritesheet = pygame.image.load("imgs/kitty-spritesheet.png")
```

- We can get the surface's rectangle

```
import pygame

spritesheet = pygame.image.load("imgs/kitty-spritesheet.png")
surface_rect = spritesheet.get_rect()
print(surface_rect) # prints <rect(0, 0, 64, 32)>
```

Surface Rects

- We can get the dimensions of each sprite image using what we know about pygame.Rect objects and the sprite sheet

```
import pygame
surface = pygame.image.load("imgs/kitty-spritesheet.png")
surface_rect = surface.get_rect()
sprite_width = surface_rect.width / 2
sprite_rect = pygame.Rect(0, 0, sprite_width, surface_rect.height)
print(surface_rect) # prints <rect(0, 0, 64, 32)>
print(sprite_rect) # prints <rect(0, 0, 32, 32)>
```

Subsurfaces

- We can get a subsurface by making a pygame.Rect that represents the location of the subsurface relative to the larger surface, and the width and height of the subsurface
- The first image starts at (x,y) = (0,0) with the height and width we just computed.
- Then we use the Surface.subsurface() method and pass in that rect.

```
""" continues from previous code """
first_image_rect = sprite_rect
first_image = spritesheet.subsurface(first_image_rect)
```

Subsurfaces

- If we save the image, we can observe the results ourselves

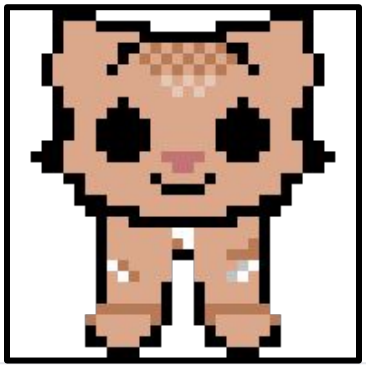
```
""" continues from previous code """
first_image_rect = sprite_rect
first_image = spritesheet.subsurface(first_image_rect)
pygame.image.save(first_image, "imgs/first_image.png")
```

imgs/first_image.png

Subsurfaces

- The second image begins at the pixel after the first image ends

```
""" continues from previous code """
second_image_rect = pygame.Rect(sprite_width, 0, sprite_width, surface_rect.height)
second_image = spritesheet.subsurface(second_image_rect)
pygame.image.save(second_image, "imgs/second_image.png")
```



imgs/second_image.png

Changing the sprite image in the Game Loop

0-immersive-kitty.py

- Take a look at the file “0-immersive-kitty.py”
- Let’s go through the parts of the code there

Imports, initialization, constants

- This is all familiar to you

```
import pygame, sys
from pygame.locals import *
import utils

pygame.init()
pygame.display.set_caption('Sprite sheets')
icon = utils.load_image("imgs/kitty.png", size=(32,32))
pygame.display.set_icon(icon)
SCREENRECT = pygame.Rect(0, 0, 640, 480)
WHITE = (225,225,225)
```

Function to load sprite sheet

- Here we have a function to load the sprite sheet so that each subsurface is an item in the list **‘images’**

```
def load_spritesheet(file_path, cols=1, scale=None, size=None):
    surface = utils.load_image(file_path, scale=scale, size=size)
    sheet_rect = surface.get_rect()
    image_width = int(sheet_rect.width / cols)

    images = []
    for i in range(0, sheet_rect.width, image_width):
        rect = pygame.Rect(i, 0, image_width, sheet_rect.height)
        image = surface.subsurface(rect)
        images.append(image)
    return images
```

Player class

We have a **Player class** that looks a little different than before

```
class Player(pygame.sprite.Sprite):
    speed = 10
    def __init__(self, sprite_sheet) -> None:
        super().__init__()
        self.sprite_sheet = sprite_sheet
        self.current_image = 0
        self.image = self.sprite_sheet[self.current_image]
        self.rect = self.image.get_rect()

    def update(self):
        self.current_image = 1 if self.current_image == 0 else 0
        self.image = self.sprite_sheet[self.current_image]

    def move(self, keystates, boundary):
        dx = self.speed * (keystates[K_RIGHT] - keystates[K_LEFT])
        self.rect.move_ip(dx, 0)
        self.rect.clamp_ip(boundary)
```

Game loop in main function

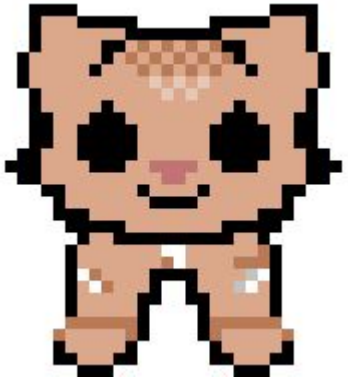
- The pieces of the main function and Game Loop are all familiar to you
- But if you want some additional clarification, do not hesitate to ask a tutor or a peer :)
- Take a look at the end of the loop to see what we do with the clock tick

```
        """
        - clock.tick(FPS) returns milliseconds since last tick
        - to get seconds, we can divide by 1000
        """
        dt = clock.tick(FPS) / 1000
        print(dt)
```

Run the code

- The FPS is 2 frames per second.
- Therefore, we should see the sprite image changing from the first image to the second image every 1 second.
- Because we are printing **dt**, you will see 0.5 (approximately) printing in the terminal

animated_sprites_tutorial/imgs/animated_cat.gif



Moving the sprite

- When we use the right and left arrows to move the sprite left to right, the sprite updates its position at the same time it updates the image.
- This makes the animation look like the cat is stepping from side to side
- We want the animation to not be synced with the position changes to make it look more independent from stepping

Animation based on actual time, not frames per second

1-immersive-kitty.py

- Take a look at the file “1-immersive-kitty.py”
- Let’s go through the parts that are different than 0-immersive-kitty.py

Main function

- **FPS = 10** instead of 2
- **sprites.update(dt)**: we pass dt into update
  - *Note: When we have more sprites, it would be better to have the player's own update function named differently, otherwise we would have to override the other sprites' update method to include dt.*

Player class

- **self.time_since_update = 0**: We added a member variable called time_since_update which is set to 0 when a Player object is created
- **Player.update method**:
  - We add the parameter ‘dt’
  - **self.time_since_update += dt**: we increase the value of **time_since_update** by **dt**
  - **if self.time_since_update >= 0.5**: When dt is 0.5 seconds or more, we update the sprite image, and reset **self.time_since_update** to **0**

Run the code

- Now, the sprite updates its image and position at a different rates

animated_sprites_tutorial/imgs/update-sprite-image.gif

Incorporate what you learned into your own game

Ideas

The next part is up to you! Here are some ideas for applying what you learned.

Animate the cat in the cat wants peppers game

You could add the cat animation we just programmed to the pepper catching game from earlier this week.

You will have to think about how to handle the Player and Pepper sprite updates differently. Could this mean putting them in different sprite groups? Or naming Player.update differently and calling it separately in the Game loop?

There are many possible options you could take in your code. Try to make some code that makes sense to you logically. Then, discuss with one of your peers or a tutor about other possible approaches, and make any changes you think would improve the code.

Animate the cat when it catches a pepper

What if you want the cat to only stretch up and go back down at the moment the cat catches the pepper?

Think about a *conditional* update to the player's sprite image.

There are many possible options you could take in your code. Try to make some code that makes sense to you logically. Then, discuss with one of your peers or a tutor about other possible approaches, and make any changes you think would improve the code.

Design your own game with animations!

- Have your own idea? Try it out!
- Remember the three parts of the game loop. It helps to write what event will trigger the update, what needs to be updated in the update function, and when (relative to the main game loop) and where precisely you expect the object to be drawn on your screen.
- Discussing with your peers or a tutor can really help you think about the logic and plan your code.
- Want to go even further? Design your own sprite sheets! But make sure you have to code them in your program so you can show them off ;) You can use online tools like https://www.pixilart.com/ or https://www.piskelapp.com/ for designing your sprite sheets.