Procedurally Generated Monsters!

A complete example of Python game development

Esther Alter

https://subalterngames.itch.io/procemon

@subalterngames
@esther_alter@mastodon.social
subalterngames.com

I am:

- Esther Alter (she/her)
- Professional Python + Unity C# dev at MIT
  - Boring!
- Indie game developer for 13 years
- Released three Python games
  - Neocolonialism (2011)
  - AZAZEL (2019)
  - Procemon (2022)

@subalterngames
@esther_alter@mastodon.social
subalterngames.com


PROĆEMON
You must catch them!

I guess I should include an outline of the talk here

- What is Procemon?
- Why Python?
- How is the code structured?
- Cool animation example
- Deployment
- What I'd do differently

I guess I should enumerate what this talk is *not* about

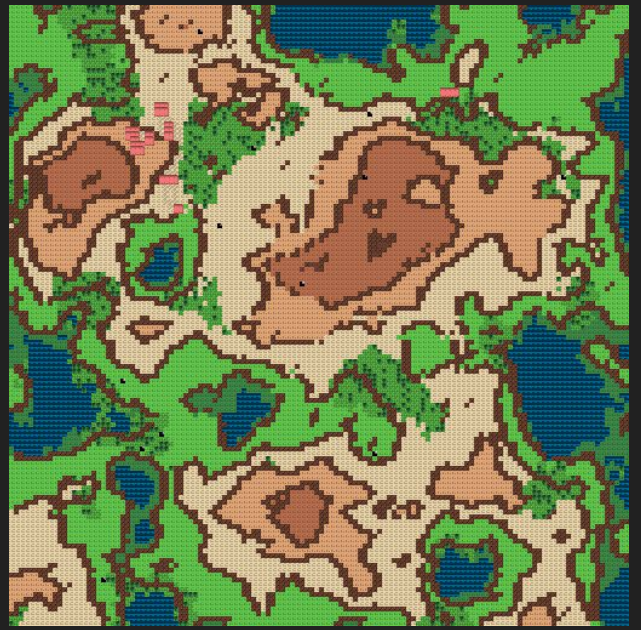
- Audio
- Game design
- Business

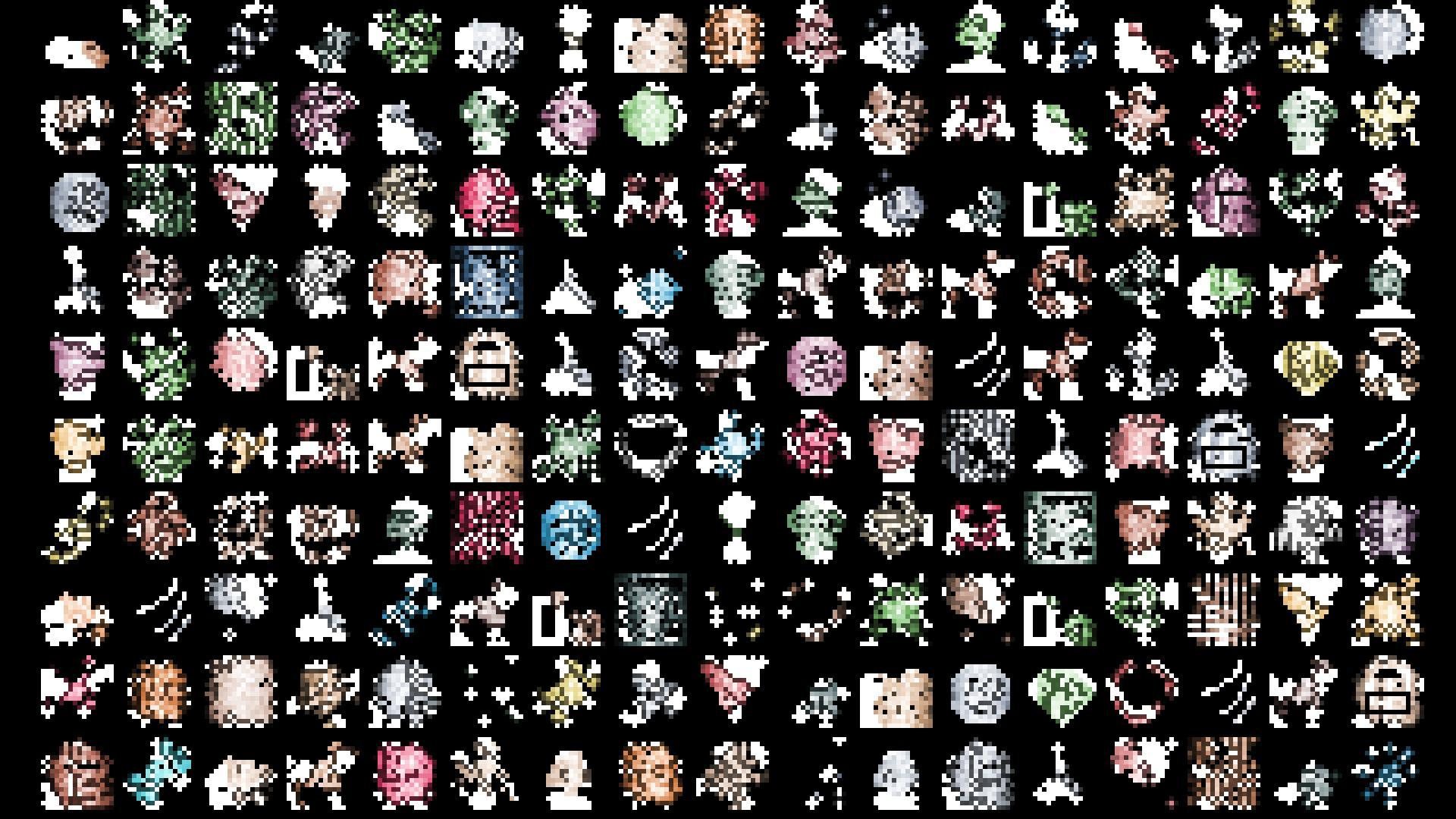
Procemon: You Must Catch Them

- Acquire cool monsters
- Explore the overworld
- Fight enemies in simple turn based battles
- Procedural generation

What is procedural generation?

- A random result that is bounded by an algorithm
- Example: world map generation
- Very common technique in games
- Downside: 10,000 bowls of oatmeal





Procemon: You Must Catch Them

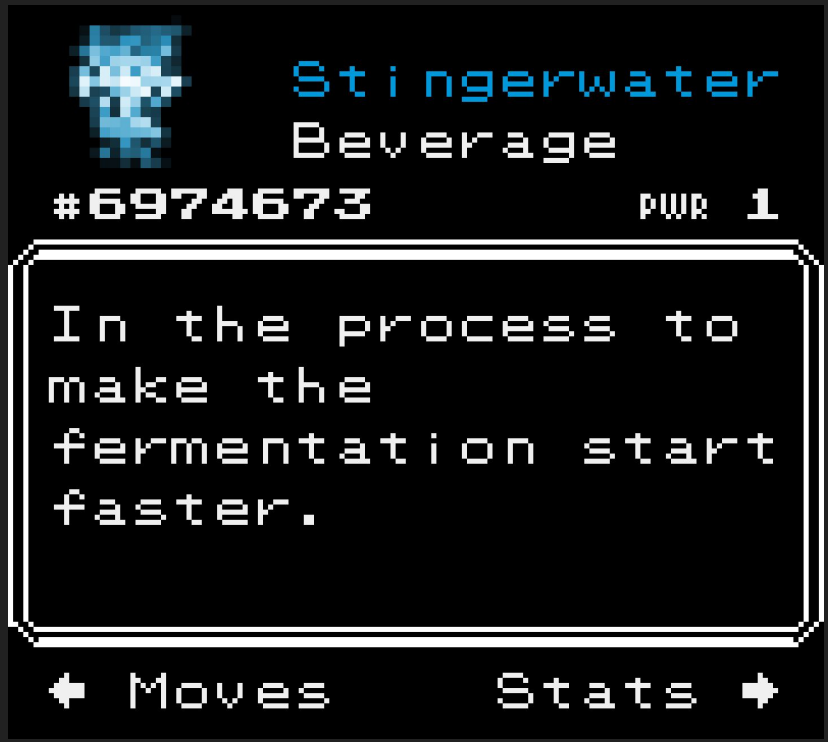
- Acquire cool monsters
- Explore the overworld
- Fight enemies in simple turn based battles
- Procedural generation
- No AI

Procedurally Generated...

- Monsters
  - Sprites

Procedurally Generated...

- Monsters
  - Sprites
  - Flavor text



Procedurally Generated...

- Monsters
  - Sprites
  - Flavor text
  - Stats

Procedurally Generated...

- Monsters
  - Sprites
  - Flavor text
  - Stats
- Moves
  - Name
  - Stats
  - Status effect

Procedurally Generated...

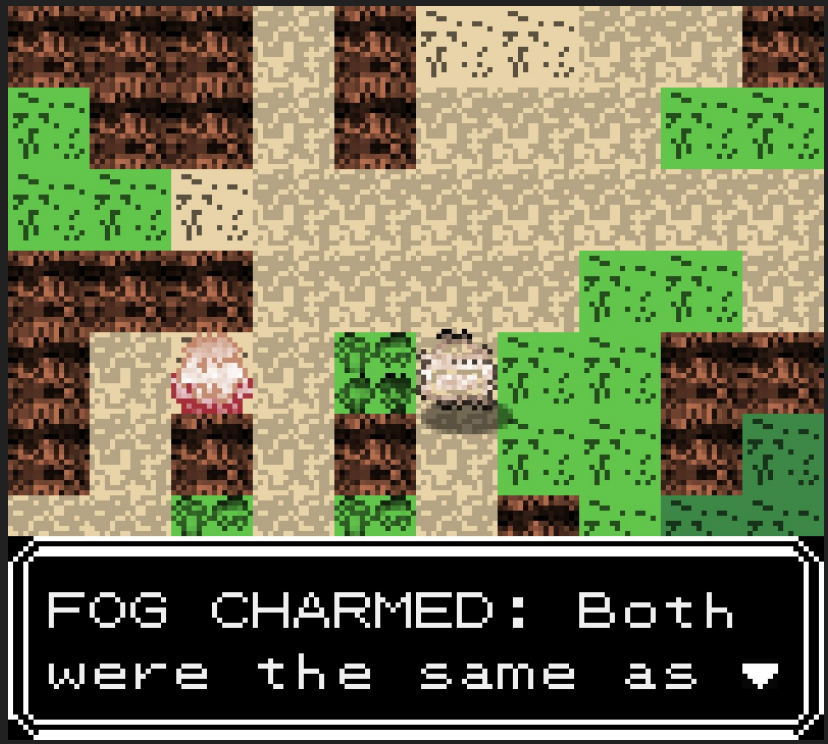
- Monsters
  - Sprites
  - Flavor text
  - Stats
- Moves
  - Name
  - Stats
  - Status effect
  - Animations

Procedurally Generated...

- Monsters
  - Sprites
  - Flavor text
  - Stats
- Moves
  - Name
  - Stats
  - Status effect
  - Animations
- Worlds
  - Map
  - Tile sprites

Procedurally Generated...

- Monsters
  - Sprites
  - Flavor text
  - Stats
- Moves
  - Name
  - Stats
  - Status effect
  - Animations
- Worlds
  - Map
  - Tile sprites
  - NPC sprites
  - NPC dialogue



Procedurally Generated...

- Monsters
  - Sprites
  - Flavor text
  - Stats
  - Fusion stats
  - Fusion sprites
  - Fusion animations
- Moves
  - Name
  - Stats
  - Status effect
  - Animations
  - Mood names
  - Mood stats
  - Mood effects
- Worlds
  - Map
  - Tile sprites
  - NPC sprites
  - NPC names
  - NPC dialogue
  - Warp animation
  - Battle entry animation
  - Seed names
- Audio
  - Splash music
  - Infinity Corporation music
  - World music
  - Battle music
  - Prof. Tree music glitch effect
  - UI sounds
  - Attack sounds
  - Fusion audio
- Visual
  - Splash animation
  - Badges
  - Prof. Tree glitch effect
  - Shareholder sprites
  - Shareholder dialogue glitch effect
  - Warp zone tiles
  - Discord bot sprite
  - Player name “font”
  - Glitch zone background

Development time and costs

- 9 months

Development time and costs

- 9 months
- $100

How I finished a game in nine months:

- I used Python
- I planned ahead

Why Python?

- FAST and EASY prototyping

```
from procemon_rpg.visual.splash_renderer import SplashRenderer


class Game:
    def __init__(self):
        # Initialize the game.
        r = SplashRenderer()
        while True:
            # Do a renderer. Get a new renderer.
            r = r.do()



if __name__ == "__main__":
    g = Game()
```

Why Python?

- FAST and EASY prototyping

r is always a **Renderer**

Main game loop

do() something, get a **Renderer**

```
from procemon_rpg.visual.splash_renderer import SplashRenderer


class Game:
    def __init__(self):
        # Initialize the game.
        r = SplashRenderer()
        while True:
            # Do a renderer. Get a new renderer.
            r = r.do()


if __name__ == "__main__":
    g = Game()
```

Why Python?

- FAST and EASY prototyping

Test script →

```
import numpy as np
from procemon_rpg.game_data import GameData
from procemon_rpg.music.sfx import SFX
from procemon_rpg.visual.renderer import Renderer
from procemon_rpg.visual.infinity_corporation.army.army_renderer
import ArmyRenderer
from procemon_rpg.macros.init import init

game = GameData.create()
game.create_world(power=1, thaw=True)
for dex_id in game.dex.monsters:
   game.player.monsters.append(
       game.dex.monsters[dex_id].create_monster(level=1))
game.player.roster = (0, 1, 2)
q = Renderer()
q.communicate(init(dex=game.dex))
r = ArmyRenderer(game_data=game,
                 sfx=SFX(rng=np.random.RandomState()))
r.do()
```

Why Python?

- FAST and EASY prototyping

Test script →
Init game data

Instantiate an
ArmyRenderer
r.do()

```
import numpy as np
from procemon_rpg.game_data import GameData
from procemon_rpg.music.sfx import SFX
from procemon_rpg.visual.renderer import Renderer
from procemon_rpg.visual.infinity_corporation.army.army_renderer
import ArmyRenderer
from procemon_rpg.macros.init import init

game = GameData.create()
game.create_world(power=1, thaw=True)
for dex_id in game.dex.monsters:
    game.player.monsters.append(
        game.dex.monsters[dex_id].create_monster(level=1))
game.player.roster = (0, 1, 2)
q = Renderer()
q.communicate(init(dex=game.dex))
r = ArmyRenderer(game_data=game,
                 sfx=SFX(rng=np.random.RandomState()))
r.do()
```

Why Python?

- FAST and EASY prototyping
  - Game loop was one of the first decisions I made
  - Easy in Python because it’s not a compiled language

Why Python?

- FAST and EASY prototyping
  - Game loop was one of the first decisions I made
  - Easy in Python because it’s not a compiled language
- Slow? 2D? Who cares?
  - It’s fast enough
  - **Constraints are good, actually**

Why Python?

- FAST and EASY prototyping
  - Game loop was one of the first decisions I made
  - Easy in Python because it’s not a compiled language
- Slow? 2D? Who cares?
  - It’s fast enough
  - **Constraints are good, actually**
- Huge toolbox with mature libraries
  - pygame-ce
  - numpy
  - PIL
  - h5py
  - etc.

A few words about pygame-ce...

- You should use pygame-ce instead of pygame
- This game was made prior to the fork

Why use pygame-ce?

- Easy and intuitive to use
- Interesting constraints
- Pixel-perfect
- Convert to and from numpy

How is pygame-ce used in Procemon?

- User input (keyboard)
- Play audio
- Render sprites
- Commands

```python
from typing import Tuple
import pygame
import pygame.freetype
from procemon_rpg.commands.command import Command
from procemon_rpg.visual.renderer import Renderer


class DrawRectangle(Command):
    def __init__(self, position: Tuple[int, int], size: Tuple[int, int],
                 color: Tuple[int, int, int] = None):
        self._rect: pygame.Rect = pygame.Rect(position[0], position[1],
                                              size[0], size[1])
        if color is None:
            self._color: Tuple[int, int, int] = (0, 0, 0)
        else:
            self._color = color

    def do(self) -> None:
        surface = pygame.Surface((self._rect.w, self._rect.h))
        surface.fill(self._color)
        Renderer.BLIT_SEQUENCE.append((surface, self._rect))
```

How is pygame-ce used in Procemon?

- User input (keyboard)
- Play audio
- Render sprites
- Commands

Command subclass

Size, position, and color

Do command
Create surface
Prepare to draw

```python
from typing import Tuple
import pygame
import pygame.freetype
from procemon_rpg.commands.command import Command
from procemon_rpg.visual.renderer import Renderer


class DrawRectangle(Command):
    def __init__(self, position: Tuple[int, int], size: Tuple[int, int],
                 color: Tuple[int, int, int] = None):
        self._rect: pygame.Rect = pygame.Rect(position[0], position[1],
                                              size[0], size[1])
        if color is None:
            self._color: Tuple[int, int, int] = (0, 0, 0)
        else:
            self._color = color

    def do(self) -> None:
        surface = pygame.Surface((self._rect.w, self._rect.h))
        surface.fill(self._color)
        Renderer.BLIT_SEQUENCE.append((surface, self._rect))
```

How is pygame-ce used in Procemon?

- User input (keyboard)
- Play audio
- Render sprites
- Commands

Command subclass

Size, position, and color

Do command
Create surface
Prepare to draw

```python
from typing import Tuple
import pygame
import pygame.freetype
from procemon_rpg.commands.command import Command
from procemon_rpg.visual.renderer import Renderer

class DrawRectangle(Command):
    def __init__(self, position: Tuple[int, int], size: Tuple[int, int],
                 color: Tuple[int, int, int] = None):
        self._rect: pygame.Rect = pygame.Rect(position[0], position[1],
                                              size[0], size[1])
        if color is None:
            self._color: Tuple[int, int, int] = (0, 0, 0)
        else:
            self._color = color

    def do(self) -> None:
        surface = pygame.Surface((self._rect.w, self._rect.h))
        surface.fill(self._color)
        Renderer.BLIT_SEQUENCE.append((surface, self._rect))
```

Result

How is pygame-ce used in Procemon?

- User input (keyboard)
- Play audio
- Render sprites
- Commands

```
# Do each render command.
while len(commands) > 0:
    command = commands.pop(0)
    command.do()

# More code here.

# Blit each surface.
if len(Renderer.BLIT_SEQUENCE) > 0:
    rects = pygame.display.get_surface().blits(
        Renderer.BLIT_SEQUENCE, True)

    Renderer.BLIT_SEQUENCE.clear()

    pygame.display.update(rects)
```

How is pygame-ce *not* used in Procemon:

- Sprite generation
- Animation generation
- Instead, use **numpy**

Why use numpy?

- Very fast
- Extensive use of direct-draw glitch effects
- “Shader” -ish
- You can use it with pygame-ce

Why use numpy?

- Very fast
- Extensive use of direct-draw glitch effects
- “Shader” -ish
- You can use it with pygame-ce

Why use numpy?

- Very fast
- Extensive use of direct-draw glitch effects
- “Shader” -ish
- You can use it with pygame-ce

Get random numbers that sum to 1.0

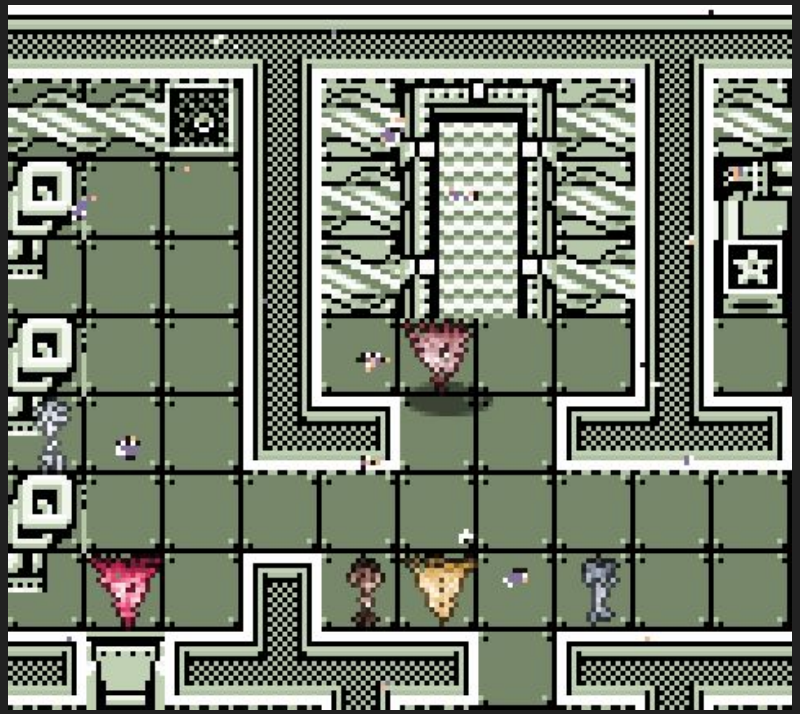
Get a copy of screen

Set pixels of each frame

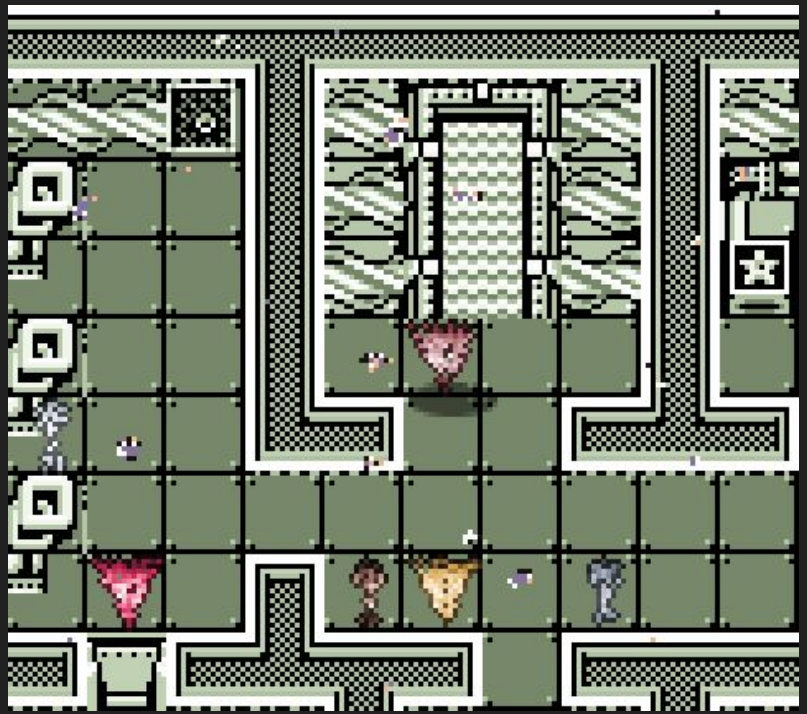
Move pixels by a sine function

```
random_arr: np.ndarray = (self._rng.dirichlet(alpha=np.ones(2), size=NUM_FRAMES) *
                          self.power + 1)
scaled_screen_arr = pygame.surfarray.array3d(pygame.transform.scale(
    pygame.display.get_surface().copy(), (160, 144)))
a = 0
roll = self._rng.random()
for i in range(NUM_FRAMES // 2):
    frame = scaled_screen_arr.copy()
    for j in range(frame.shape[0]):
        frame[j] = np.roll(frame[j], int(np.sin(
            j / frame.shape[0] * a) * random_arr[i][1] * 2), axis=0)
    a += 4 * random_arr[i][0]
```

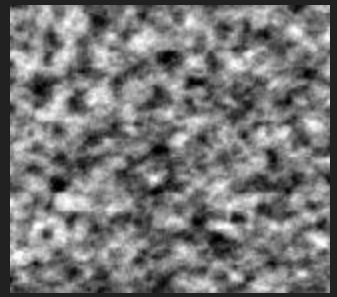
Look! A cool animation!



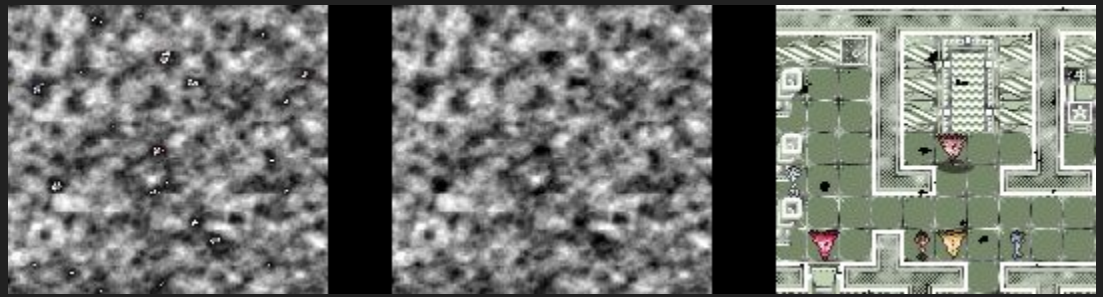
Look! A cool animation!



Look! A cool animation's frames!



noise



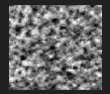
glitch world display

Look! A cool animation's code!

```
world_arr = pygame.surfarray.array3d(surface)
display_arr = pygame.surfarray.array3d(pygame.display.get_surface())
mask_increment = 256 // WarpToWorld._NUM_FRAMES
mask_value = mask_increment
power = rng.randint(2, 11)
n = noise.get_uint8_array(random_seed=self._random_seed, power=power)
noise_arr = np.zeros(shape=(Renderer.WIDTH, Renderer.HEIGHT))
for y in range(0, Renderer.SCALE):
    for x in range(0, Renderer.SCALE):
        noise_arr[y:noise_arr.shape[0]:Renderer.SCALE,
                  x:noise_arr.shape[1]:Renderer.SCALE] = n
while mask_value <= 256:
    glitch_arr: np.array = np.zeros(shape=(160, 144, 3), dtype=np.uint8)
    random_values = rng.random(size=23040).reshape((160, 144))
    for c in GLITCH_COLORS:
        glitch_arr[random_values > c] = GLITCH_COLORS[c]
    glitch_arr = glitch_arr.repeat(Renderer.SCALE, axis=0).repeat(Renderer.SCALE, axis=1)
    frame = np.zeros(shape=display_arr.shape, dtype=np.uint8)
    frame[noise_arr < mask_value] = glitch_arr[noise_arr < mask_value]
    frame[noise_arr < mask_value - mask_increment * 6] = (
        world_arr[noise_arr < mask_value - mask_increment * 6]
    frame[noise_arr >= mask_value] = display_arr[noise_arr >= mask_value]
    mask_value += mask_increment
```

Look! A cool animation's code!

World surface and display surface to numpy arrays

Create a noise array 

Generate each frame

Generate "glitch" array 

Per-pixel:

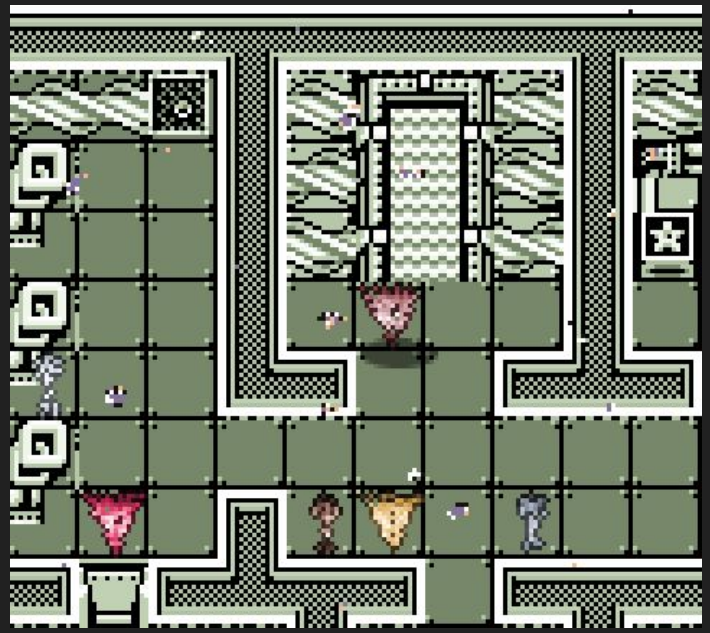
- **`noise < mask_value`** → *glitch*
- **`noise < mask_value-mask_increment*6`** → *world*
- **`noise >= mask_value`** → *display*

Increment **`mask_value`**

```
world_arr = pygame.surfarray.array3d(surface)
display_arr = pygame.surfarray.array3d(pygame.display.get_surface())
mask_increment = 256 // WarpToWorld._NUM_FRAMES
mask_value = mask_increment
power = rng.randint(2, 11)
n = noise.get_uint8_array(random_seed=self._random_seed, power=power)
noise_arr = np.zeros(shape=(Renderer.WIDTH, Renderer.HEIGHT))
for y in range(0, Renderer.SCALE):
    for x in range(0, Renderer.SCALE):
        noise_arr[y:noise_arr.shape[0]:Renderer.SCALE,
                  x:noise_arr.shape[1]:Renderer.SCALE] = n
while mask_value <= 256:
    glitch_arr: np.array = np.zeros(shape=(160, 144, 3), dtype=np.uint8)
    random_values = rng.random(size=23040).reshape((160, 144))
    for c in GLITCH_COLORS:
        glitch_arr[random_values > c] = GLITCH_COLORS[c]
    glitch_arr = glitch_arr.repeat(Renderer.SCALE, axis=0).repeat(Renderer.SCALE, axis=1)
    frame = np.zeros(shape=display_arr.shape, dtype=np.uint8)
    frame[noise_arr < mask_value] = glitch_arr[noise_arr < mask_value]
    frame[noise_arr < mask_value - mask_increment * 6] = (
        world_arr[noise_arr < mask_value - mask_increment * 6]
    frame[noise_arr >= mask_value] = display_arr[noise_arr >= mask_value]
    mask_value += mask_increment
```

Look! A cool animation!

- Noise generated in tcod
- Animation frames generated in numpy
- Rendered in pygame-ce



Deployment

- pyinstaller
  - Won’t protect your code
  - **Cannot cross-compile**
- GitHub workflow
  - Windows, MacOS, Linux
  - itch.io, Steam

Upload to itch.io

- Download and install Butler
- Upload

Upload to Steam

- Create archive
- FTP the archive to my own website
- Every minute, check if all archives are on my website
- Download the archives to my local computer
- Upload archives to Steam from my local computer
- Manually publish release

What I’d do differently

- Not much
- Have an actual business plan

I recommend:

- Make games in Python
- Have fun with constraints

Procedurally Generated Monsters!

A complete example of Python game development

Esther Alter

https://subalterngames.itch.io/procemon

@subalterngames
@esther_alter@mastodon.social
subalterngames.com