# Creating Flappy Bird Game

Rules :

Keep it alive !

Do not hit pipes/ground

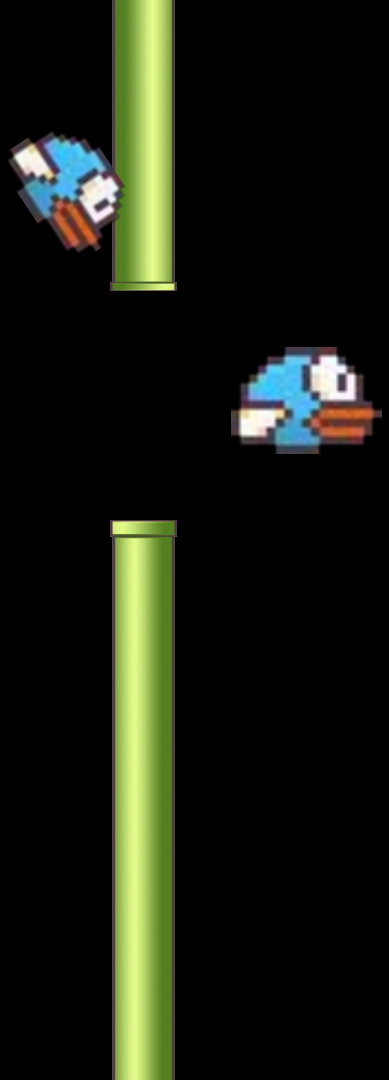
Flap !

Flap !

Flap !

## Main elements:

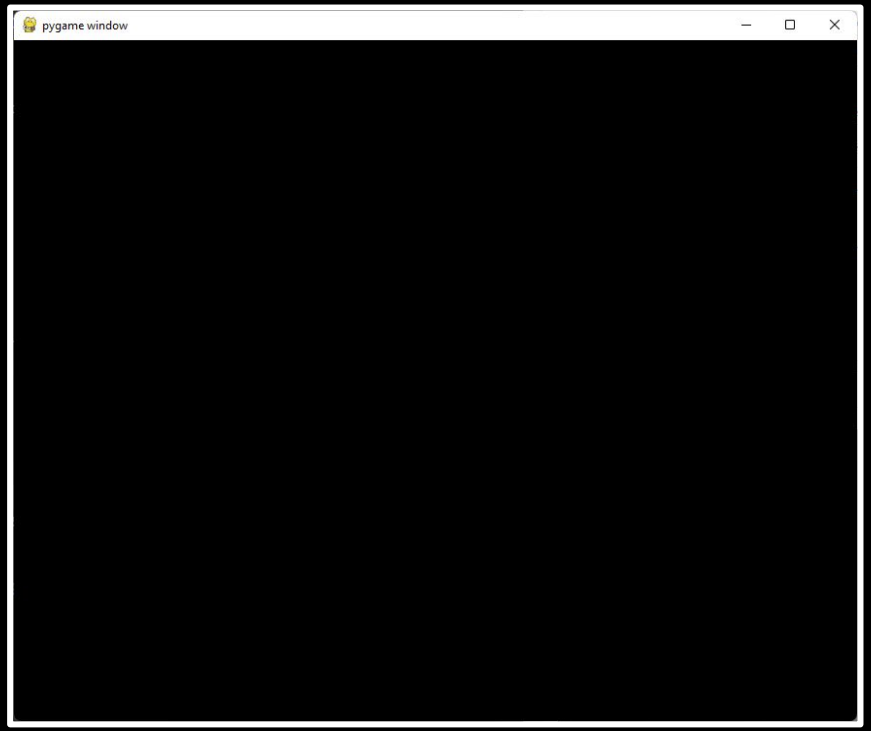
1. Rolling ground
2. Pipes
3. Flappy Bird
4. Collision



score: + 1 !!

# 1: Setting Up Scene

```python
import pygame
from sys import exit

# Initialize Pygame
pygame.init()

# Set up the display
size = WIDTH, HEIGHT = 550, 720
screen = pygame.display.set_mode((WIDTH, HEIGHT))
pygame.display.set_caption('Flappy bird')

# Main game loop
running = True
while running:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            running = False

    # Game logic and rendering goes here

    pygame.display.flip()  # Update the display

# Quit Pygame
pygame.quit()
```



# 1: Rolling ground

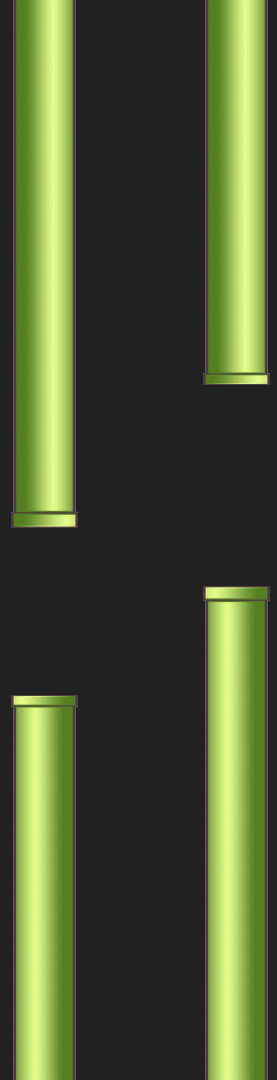
```python
class Ground(pygame.sprite.Sprite):
    def __init__(self, x, y):
        # initialize the clss
        pygame.sprite.Sprite.__init__(self)
        self.image = ground
        self.rect = self.image.get_rect()
        self.rect.x, self.rect.y = x, y

    def update(self):
        # Move Ground
        self.rect.x -= scroll_speed_1
        if self.rect.x <= -551:
            self.kill()
```

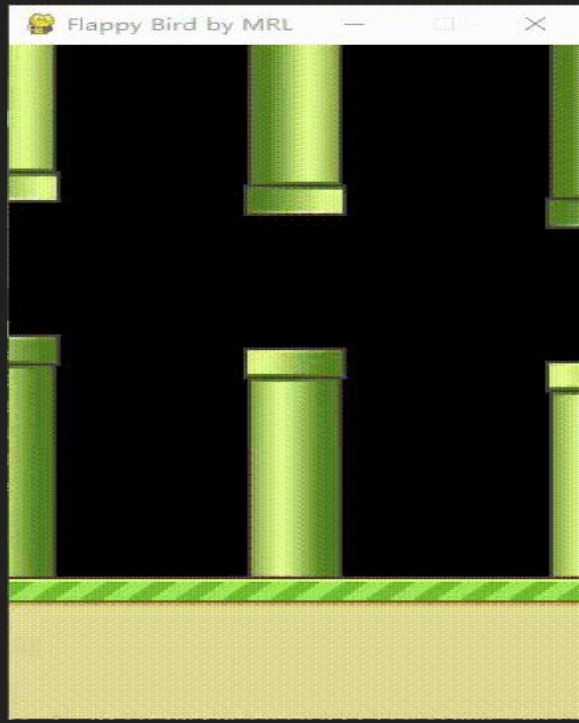
```python
# draw background setting - pipes, ground and bird
the_ground.draw(screen)
# Spawn Ground
if len(the_ground) <= 2:
    the_ground.add(Ground(WIDTH, y_position_ground))
```

# 2. Pipes -- Rolling

```python
class Pipes(pygame.sprite.Sprite):
    def __init__(self, x, y, image, pipe_type):
        pygame.sprite.Sprite.__init__(self)
        self.image = image
        self.rect = self.image.get_rect()
        self.rect.x = x
        self.rect.y = y
        self.enter = False
        self.exit = False
        self.passed = False
        self.pipe_type = pipe_type

    def update(self):
        # Move Pipe
        self.rect.x -= scroll_speed_1
        if self.rect.x <= -WIDTH:
            self.kill()
```

```python
# Spawn Pipes
if timer <= 0 and bird.sprite.alive:
    x_top, x_bottom = 550, 550
    y_top = random.randint(-600, -488)
    y_bottom = y_top + random.randint(105, 145) + pipe_bottom.get_height()
    pipes.add(Pipes(x_bottom, y_bottom, pipe_bottom, pipe_type: 'top'))
    pipes.add(Pipes(x_top, y_top, pipe_top, pipe_type: 'bottom'))
    timer = random.randint(200, 250)
timer -= 1
```

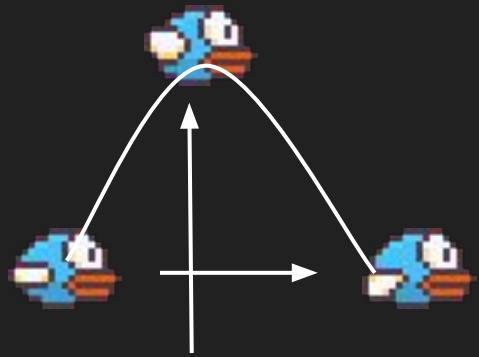




# 3. Bird

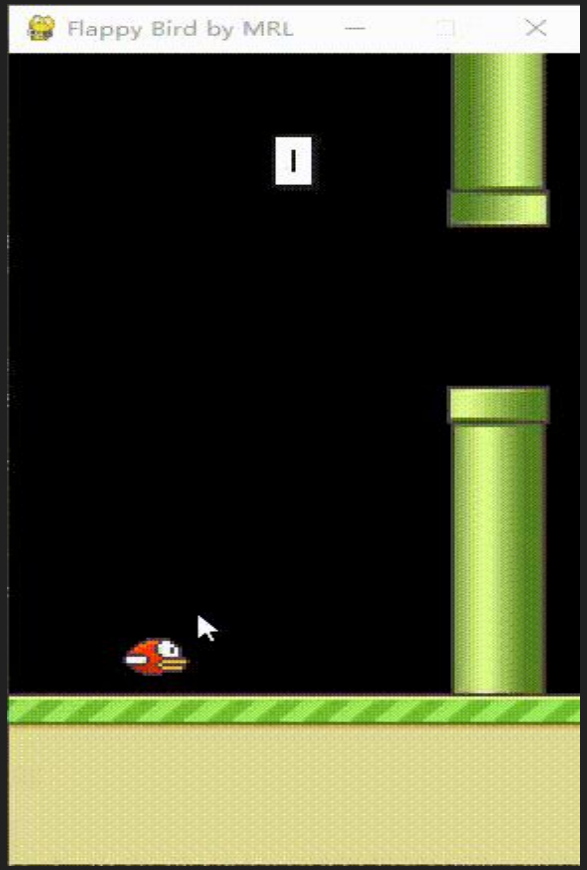
```
def update(self, user_input):
    # Animate Bird
    if self.alive:
        self.image_index += 1
    if self.image_index >= 30:
        self.image_index = 0
    self.image = bird_images[self.image_index // 10]
```

```
self.gravity += 0.5
if self.gravity > 7:
    self.gravity = 7
if self.rect.y < 500:
    self.rect.y += int(self.gravity)
if self.gravity == 0:
    self.flap = False

self.image = pygame.transform.rotate(self.image, self.gravity * -7)
```



## 3. Bird

```
if user_input[pygame.K_SPACE] and not self.flap and self.rect.y > 0 and self.alive:
    self.flap = True
    self.gravity = -7
    flap_sound.play()
```



# 4. Collision

```
# Collision Detection
global test
collision_pipes = pygame.sprite.spritecollide(bird.sprites()[0], pipes, dokill: False)
collision_ground = pygame.sprite.spritecollide(bird.sprites()[0], the_ground, dokill: False)
if collision_pipes or collision_ground:
    bird.sprite.alive = False
    screen.blit(game_over, dest: [180, 300])
    if test == 0:
        hit_sound.play()
        test = 1
    if user_input[pygame.K_r]:
        score = 0
        level = 0
        test = 0
        main()
```

```
  global score
  if self.pipe_type == 'bottom':
      if bird_star_position[0] > self.rect.topleft[0] and not self.passed:
          self.enter = True
      if bird_star_position[0] > self.rect.topright[0] and not self.passed:
          self.exit = True
      if self.enter and self.exit and not self.passed:
          self.passed = True
          score += 1
          point_sound.play()
```

# LEVEL UP!

```
# Show level
if 10 > score >= 5:
    level = 1
if score >= 10:
    level = (score // 10) + 1
if score == 5:
    level_up_sound.play()
if score / 10 in [1, 2, 3, 4, 5]:
    level_up_sound.play()

level_text = font_2.render('LEVEL ' + str(level), antialias: True, pygame.Color(255, 255, 255))
screen.blit(level_text, dest: (200, 90))
```

## pipes: more pipes coming in

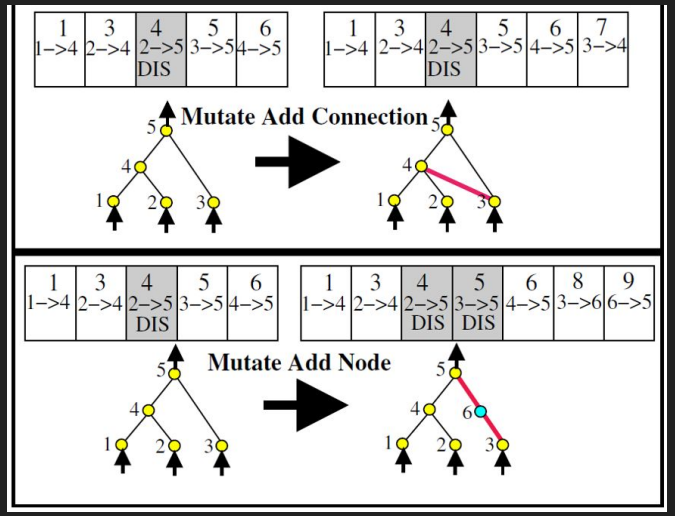
```
if level == 1:
    timer = random.randint(160, 200)
if level == 2:
    timer = random.randint(80, 120)
if level == 3:
    timer = random.randint(60, 80)
if level >= 4:
    timer = random.randint(40, 60)
```

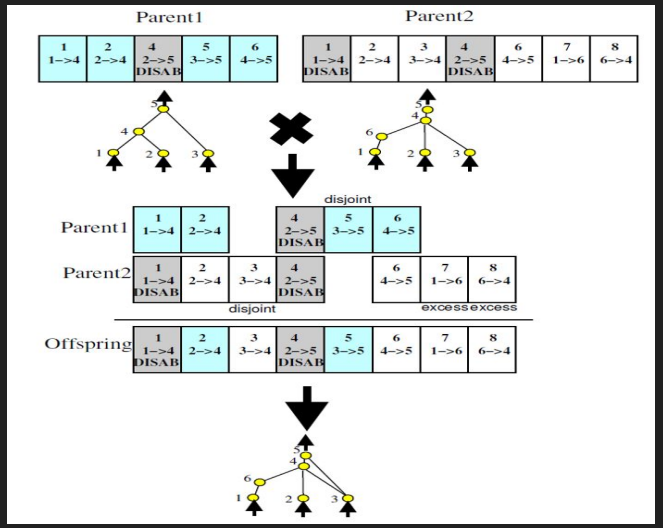
## Ground: speeding up

```
def update(self):
    # Move Ground
    self.rect.x -= scroll_speed_1
    if level == 2:
        self.rect.x -= 0.6 * scroll_speed_1
    if level >= 3:
        self.rect.x -= 1.2 * scroll_speed_1
    if self.rect.x <= -551:
        self.kill()
```

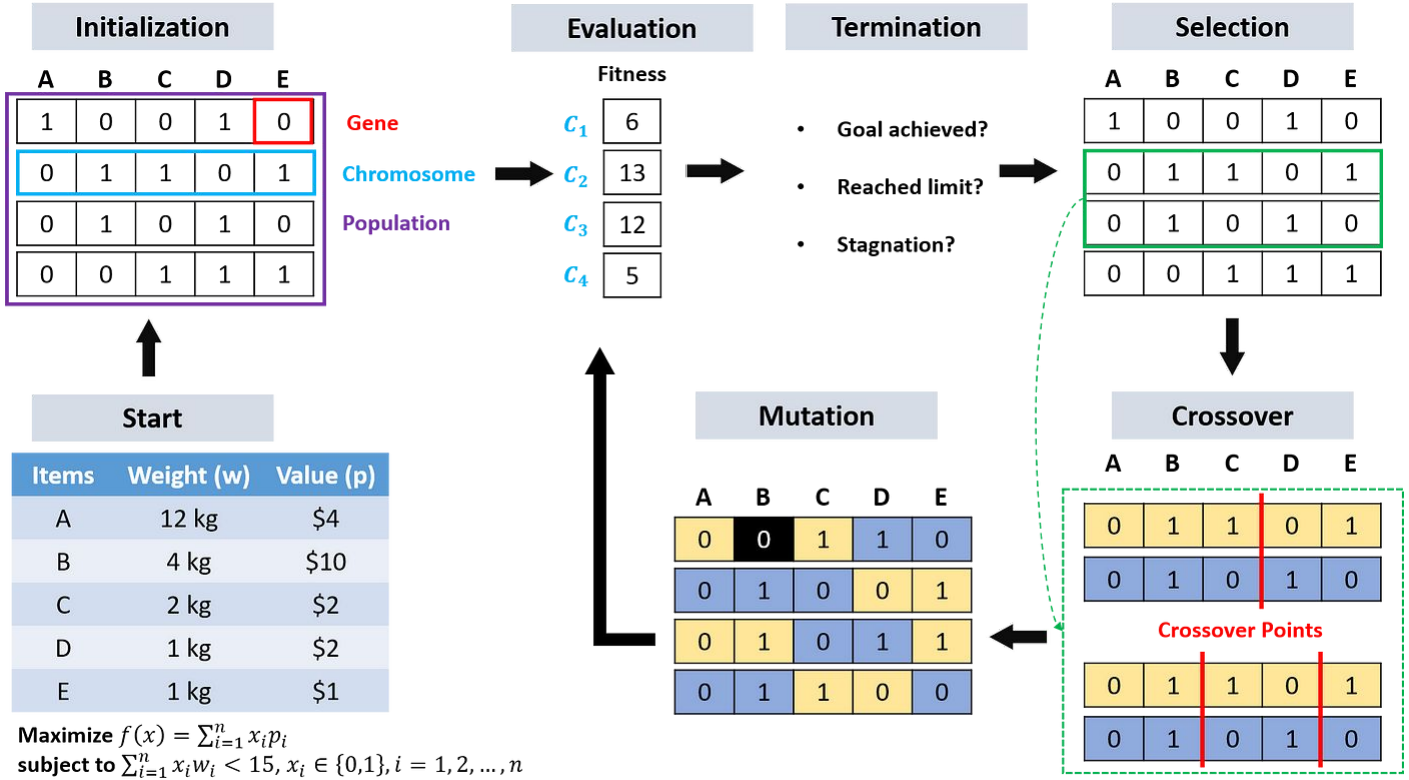
# AI -- N.E.A.T

NEAT stands for NeuroEvolution of Augmenting Topologies, which is a genetic algorithm designed to efficiently evolve artificial neural network topologies.




moduler: https://neat-python.readthedocs.io/en/latest/
paper: https://nn.cs.utexas.edu/downloads/papers/stanley.cec02.pdf




https://www.youtube.com/watch?v=ihX3-WDua2I