

# Simulacija logističkog rasta populacije

Logistički rast populacije, za razliku od eksponencijalnog koji pokazuje isključivo rast populacije bez obzira na ostale uvijete, prikazuje malo realniju sliku. Naime populacija će rasti sve dok okoliš i njegov kapacitet to dopuštaju. Navedena se pojava može opisati jednačinom:

$$\frac{dN}{dT} = r_{max}N \frac{(K - N)}{K}$$

N- broj jedinki

T- vrijeme

K- kapacitet okoliša

r- stopa rasta

Ova jednačina ima brojne varijacije, ja sam zbog jednostavnosti korištenja odabrao ovu:

$$N(t) = \frac{K}{1 + (\frac{K - N_0}{N_0})e^{-rt}}$$

N(t)- broj jedinki u vremenu

K- kapacitet okoliša

$N_0$ - početni broj jedinki

r- stopa rasta

Svoj sam zadatak odlučio realizirati i vizualno prikazati na dva načina:

## **1) Grafičkim prikazom**

-nakon odabira parametara prikazuje se grafički prikaz logističkog rasta populacije s obzirom na odabrane uvjete (najčešće bi trebao biti u obliku slova „S“)

## **2) Simulacijom (uz sve navedene parametre sadrži i parametar za brzinu)**

-simulacija prikazuje rast broja jedinki s obzirom na navedene parametre

-kada broj jedinki dosegne kapacitet okoliša (približno) krenut će se sporije stvarati i na taj način reprezentirati logistički rast

## Najvažnije klase i funkcije:

1. Klasa za stvaranje gumba, omogućuje pregledno i jasno strukturirano korisničko sučelje. Zahvaljujući njoj kreiranje gumba i smještanje na ekran postaje vrlo lagano.

```
def main_menu(clicked):
    global state, graph

    btn_size = (70, 100)

    MENU_MOUSE_POS = pygame.mouse.get_pos()

    MENU_TEXT = pygame.font.SysFont("arial", 70).render("LOGISTIČKI RAST POPULACIJE", (0, 0, 0), True)
    MENU_RECT = MENU_TEXT.get_rect(center=(640, 200))

    GRAF_BUTTON = Gumb(image=None, pos=(220, 580), size=(120, 70),
        text_input="GRAF. PRIKAZ", font=font_menu, base_color="adffcd", hovering_color="white")
    SIMULACIJA_BUTTON = Gumb(image=None, pos=(440, 580), size=(120, 70),
        text_input="SIMULACIJA", font=font_menu, base_color="adffcd", hovering_color="white")
    QUIT_BUTTON = Gumb(image=None, pos=(660, 580), size=(120, 70),
        text_input="QUIT", font=font_menu, base_color="adffcd", hovering_color="white")

    SCREEN.blit(MENU_TEXT, MENU_RECT)

    for button in [GRAF_BUTTON, SIMULACIJA_BUTTON, QUIT_BUTTON]:
        button.changeColor(MENU_MOUSE_POS)
        button.update(SCREEN)

    if clicked:
        if GRAF_BUTTON.checkForInput(MENU_MOUSE_POS):
            state = "PLAY"
            graph = get_logistic_graph(parameters["r"], parameters["K"], parameters["N"])
            SCREEN.blit(graph, GRAPH_POS)
        if SIMULACIJA_BUTTON.checkForInput(MENU_MOUSE_POS):
            state = "OPTIONS"
        if QUIT_BUTTON.checkForInput(MENU_MOUSE_POS):
            pygame.quit()
            sys.exit()
```

3. Klasa slider koja omogućuje učinkovit odabir vrijednosti parametara i olakšava korisniku korištenje programa.

```
class Gumb:
    def __init__(self, image, pos, size, text_input, font, base_color, hovering_color):
        self.image = image
        self.x_pos = pos[0]
        self.y_pos = pos[1]
        self.width = size[0]
        self.height = size[1]
        self.font = font
        self.base_color = base_color
        self.hovering_color = hovering_color
        self.text_input = text_input
        self.text = self.font.render(self.text_input, True, self.base_color)
        if self.image:
            self.rect = self.image.get_rect(center = (self.x_pos, self.y_pos))
        else:
            self.rect = pygame.Rect(self.x_pos - self.width // 2, self.y_pos - self.height // 2, self.width, self.height)
            self.text_rect = self.text.get_rect(center = (self.x_pos, self.y_pos))

    def changeText(self, text):
        self.text_input = text
        self.text = self.font.render(self.text_input, True, self.base_color)
        self.text_rect = self.text.get_rect(center = (self.x_pos, self.y_pos))

    def update(self, screen):
        if self.image is not None:
            screen.blit(self.image, self.rect)
        pygame.draw.rect(screen, "black", [self.x_pos - self.width // 2, self.y_pos - self.height // 2, self.width, self.height])
        screen.blit(self.text, self.text_rect)

    def checkForInput(self, position):
        if position[0] in range(self.rect.left, self.rect.right) and position[1] in range(self.rect.top, self.rect.bottom):
            return True
        return False

    def changeColor(self, position):
        if position[0] in range(self.rect.left, self.rect.right) and position[1] in range(self.rect.top, self.rect.bottom):
            self.text = self.font.render(self.text_input, True, self.hovering_color)
```

2. Funkcija main() koja je zapravo baza i temelj svega, pomoću nje se povezuje main menu sa funkcijama za crtanje grafa i projekciju simulacije koje su također prilično važne, ali i prevelike da bi ih stavio u ovaj dokument.

```
class Slider:
    def __init__(self, cx, cy, w, h, min, max, value=0.5):
        self.cx = cx
        self.cy = cy
        self.w = w
        self.h = h
        self.min = min
        self.max = max
        self.value = value
        self.colors = (100, 100, 100)
        self.color = (60, 60, 60)

        self.scaled_value = (value - min) / (max - min)
        self.scaled_pos = self.scaled_value * self.w

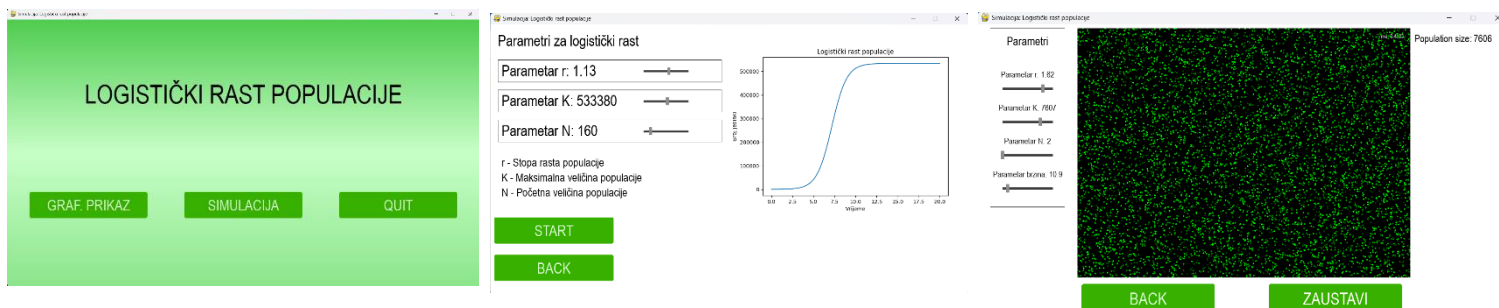
        self.x = self.scaled_pos + self.cx - self.w // 2
        self.y = self.cy
        self.held = False

    def update(self, clicked, screen, disabled=False):
        if not disabled:
            if not pygame.mouse.get_pressed()[0]:
                self.held = False
            if self.get_rect().collidepoint(pygame.mouse.get_pos()) and clicked:
                self.held = True
            if self.held:
                self.x = min(max(pygame.mouse.get_pos()[0], self.x - self.w // 2), self.x + self.w // 2)
                self.x = min(max(pygame.mouse.get_pos()[0], self.x - self.w // 2), self.x + self.w // 2)
            if not self.held and clicked and pygame.Rect(self.x - self.w // 2, self.y - 4, self.w, 8).collidepoint(pygame.mouse.get_pos()):
                self.x = pygame.mouse.get_pos()[0]

        pygame.draw.line(screen, self.color, (self.x - self.w // 2, self.y), (self.x + self.w // 2, self.y), 2)
        pygame.draw.rect(screen, self.color2, self.get_rect())

    def get_rect(self):
        return pygame.Rect(self.x - 4, self.y - 10, 8, 20)
```

## Izgled samog programa:



Priredio: Roko Vrdoljak, 4.d  
V. gimnazija, Zagreb 2024.

