ТЕХНОЛОГИЧНО УЧИЛИЩЕ ЕЛЕКТРОННИ СИСТЕМИ
към ТЕХНИЧЕСКИ УНИВЕРСИТЕТ - СОФИЯ

КУРСОВ ПРОЕКТ

Предмет: Въведение в скриптовите езици
Тема: Симулатор на екосистема

Ученик:
Габриел Мартинов

Научен ръководител:
Даниел Димитров

СОФИЯ
2023 г.

Използвани съкращения

GUI	Graphic User Interface **(**Графичен Потребителски Интерфейс)
IDE	Integrated Development Environment (Интегрирана Среда за Разработка)
ООП	Object – oriented programming (Обектно – ориентирано програмиране)
CMD	Command Prompt

Увод

Екосистемите са сложни и динамични, обхващайки множество видове, които взаимодействат помежду си и с околната среда. През последните десетилетия, учените са осъзнали растящата необходимост от по-добро разбиране на тези взаимодействия, особено в контекста на бързо променящите се условия на околната среда и нарастващото въздействие на човешката дейност.

С развитието на компютърните технологии и повишаването на мощността на обработката на данни, учените започват да разработват математически модели и симулации, които да им помогнат да предскажат и анализират поведението на екосистемите. Такива симулатори имат потенциала да предоставят ценни инсайти и информация, която може да се използва за опазване на биологичното разнообразие, управление на природните ресурси и адаптация към измененията в климата.

Глава 1
Взаимодействие между биоразнообразието в екосистемите

Взаимовръзката между животните и растенията в екосистемите е от критично значение за поддържането на баланс и устойчивост в тях. Тази връзка се основава на взаимна зависимост и взаимна полза между двете групи организми.

Растенията са основен източник на храна за животните в екосистемите. Те синтезират собствената си храна чрез процеса на фотосинтеза, като превръщат слънчевата енергия, вода и въглероден диоксид в глюкоза и кислород. Животните се хранят с тези растения и други хранителни вещества, използвайки глюкозата като източник на енергия.

Освен като източник на храна, растенията предоставят и други ползи за животните. Много животни полагат гнезда, създават убежища и намират защита в растителните структури. Растенията предоставят също така жизненонеобходимия кислород за животните и абсорбират вредните газове от околната среда.

От своя страна, животните играят важна роля в разпространението на растенията. Някои видове животни, като птици и насекоми, са отговорни за опрашването на цветовете и разпространението на полен от едно растение на друго. Това е важен процес за размножаването и разнообразието на растителния вид.

Животните играят роля при разсейването на семената и разпространението на плодове и плодови тела на растенията. Животните разпръскват семената на плодовете на различни места в екосистемата. Това помага за разпространението на растенията в нови райони и насърчава разнообразието на растителния вид.

Взаимовръзката между животните и растенията е сложна и неотменна част от екосистемите. Тя е от съществено значение за поддържането на баланс и функционирането на екосистемите, като осигурява храна, защита, разнообразие и разпространение на видовете. Без тази връзка, екосистемите биха били неустойчиви и неспособни да съществуват и да се развиват.

Глава 2

Подбор и оценка на използваните технологии

2.1 Функционални изисквания към разработката

Текущият проект цели реализирането на екосистемен симулатор на програмния език Python, както и приложение за визуализиране и навигация през симулацията, реализирано на същия език посредством библиотеката Pygame. Проектът трябва да е написан на скриптов език, да използва минимум една библиотека или фреймуърк и кода да е повече от 100 реда.

2.1.1 Изисквания към симулацията

За осъществяване на целта на проекта е необходимо:

- Възможност за параметърно управление на:
    - Брой растения, които да се нарисуват на екрана
    - Брой тревопасни животни, които да се нарисуват на екрана
    - Брой хищници, които да се нарисуват на екрана
    - Околната температура
    - Влажността на въздуха
- Алгоритъмът, трябва да:
    - Движи животните свободно по екрана
    - Реализира взаимодействие между тревопасните животни и растенията, като например консумация на растения за енергия
    - Реализира взаимодействие между хищниците и тревопасните животни, като например нападение и консумация на тревопасни животни за енергия
    - Визуализира в реално време промените в екосистемата, като например нарастване или намаляване на популацията на животните и растенията
    - Предоставя статистически данни за брой животни и растения
    - Реализира природно бедствие – торнадо

2.1.2 Изисквания към приложението

За удобство при работа с разработения симулатор е необходимо, да се осигурят следните функционалности:

- Приложението трябва да разполага с ясно видим прозорец, посредством който може да се наблюдава симулацията
- Навигация през симулацията, посредством бутони и текст боксове
- Езикът, на който е реалзирана разработката трябва да е Python, а конкретната библиотека – Pygame

2.2 Подбор на алгоритъм

За постигане на целта на проекта е разработен авторски алгоритъм., чрез който се осъществява динамиката на опростена екосистема, състояща се от три основни вида организми, които я обитават – растения, растителноядни животни и хищници.

2.3 Подбор на програмен език

Текущият проект е разработен на езика за програмиране Python, който отговаря на изискването за използване на скриптов език. Визуализиране и навигация през симулацията е постигнато чрез използване на библиотека Pygame.

2.3.1 Предимства на езика

Предимства на програмния език Python са, че той е със сравнително лесен синтаксис и е независим от операционната система, което позволява, проектът да бъде стартиран на всяка компютърна система. Наред с това съществуват множество библиотеки, чрез които проекта може да бъде реализиран по най-добрия начин.

2.3.2 Недостатъци на езика

Езикът Python има известни недостатъци при правенето на симулации. Това, че езикът е интерпретиран прави производителността му по-ниска,сравнено с тази на C++. Използваната от Python памет е по-голяма, за разлика от тази на C++.

2.4 Подбор на технологии за разработка на приложението

Приложението е написано отново на Python, посредством библиотеката за разработка на игри – Pygame. Библиотеката не е особено подходяща за направа на GUI (Graphic User Interface), но е по-добър избор от Tkinter, защото предоставя възможност за реализацията на по-сложни проекти.

2.5 Среди за разработка

2.5.1 Текстов редактор

Настоящият проект бе разработен на редактора Visual Studio Code IDE (Integrated Development Environment).

2.5.2 Среда за разработка на документация

Настоящата документация бе разработена на Word.

Глава 3

Реализация на проекта

3.1 Архитектура и основен сценарий

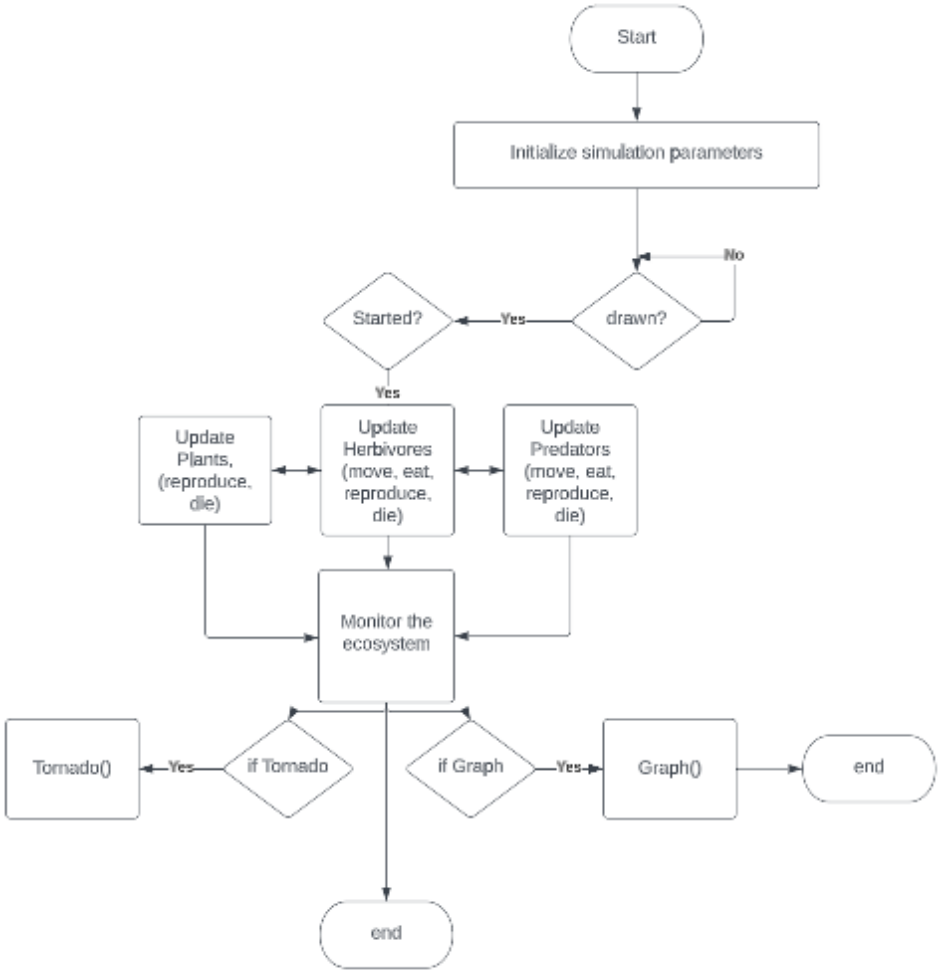
Проектът се състои от две основни части:

- Алгоритъм за симулиране на екосистема;
- Приложение, посредством което може да се параметризира симулираната екосистема, както и да се симулира природно бедствие - торнадо.

Симулацията генерира различен брой растения, тревопасни и/или хищни животни, които се визуализират на основния екран на приложението и се обновяват постоянно на база алгоритъма (фиг. 3.1). Посредством бутони може да се стартира или прекратява симулацията, както и да се прегледа статистическа диаграма с популацията на

биоразнообразието през периода на симулацията. Посредством различни текст боксове може да се регулира симулацията, като:

- се променят стойностите на първоначално показаните животни и/или растения
- се променят параметрите на околната среда – температура и влажност на въздуха
- се симулира природно бедствие - торнадо



Фигура 3.1: Блок – схема на алгоритъма

3.2 Алгоритми за симулиране на екосистема

За разработката на проекта съм използвал ООП. Има 3 класа – Растения, Тревопасни животни и Хищни животни. Всеки от тях разполага с функции, които да симулират основните процеси – хранене, размножаване, ходене и смърт.

3.2.1 Алгоритъм за визуализация

Функцията draw взима 2 аргумента, които са цветовете на растението (Фиг. 3.2). На база енергията на растението, радиусът се променя, като това ще ни осигури растежа на растенията по-късно,като той става чрез функцията grow(). При нея умножаваме енергията на растението и я събираме с нея, като гледаме да не надвишава 100. Следващите редове рисуват самите растения по екрана под формата на кръгчета, като са използвани pygame и pygame.gfxdraw, за да може кръговете да са максимално детайлни и не деформирани.

```
def draw(self, screen, PLANT_COLOR, PLANT_COLOR2):
    radius = max(2, min(4, int(self.energy / 20)))
    pygame.gfxdraw.filled_circle(screen, int(self.x), int(self.y), radius, PLANT_COLOR)
    pygame.gfxdraw.aacircle(screen, int(self.x), int(self.y), radius, PLANT_COLOR)
    pygame.gfxdraw.aacircle(screen, int(self.x), int(self.y), radius - 2 , PLANT_COLOR2)
    pygame.gfxdraw.filled_circle(screen, int(self.x), int(self.y), radius - 2, PLANT_COLOR2)
```

Фигура 3.2: Функция за визуализация на растения

По подобен начин работи и визуализацията на другите два класа – Тревопасни (Фиг 3.3) и Хищници (Фиг 3.4), където имаме разлики в цвета и размера при мъжките и женските видове. На случаен принцип, чрез библиотеката random се избират дали даденото животно да е мъжко или женско и съответно след това се визуализират чрез функцията draw() (Фиг. 3.5а и Фиг. 3.5б). Координатите за появяване се избират чрез библиотеката random, като максималните и минималните стойности, от които може да се избира, са границите на симулатора.

```python
    def __init__(self, x, y, energy, color=HERBIVORE_COLOR):
        self.x = x
        self.y = y
        self.energy = energy
        self.angle = random.uniform(0, 2 * math.pi)
        self.gender = random.choice(["male", "female"])
        self.color = self.adjust_color_by_gender(color)

    def adjust_color_by_gender(self, color):
        if self.gender == "male":
            return (30,144,255)  # Darker shade
        else:
            return (173, 216, 230)  # Lighter shade

    def draw(self, screen):
        MALE_SIZE = 9
        FEMALE_SIZE = 6

        size = MALE_SIZE if self.gender == 'male' else FEMALE_SIZE

        pygame.draw.circle(screen, self.color, (int(self.x), int(self.y)), size)
```

Фигура 3.3: Функции за визуализация на тревопасни животни

```python
class Predator:
    def __init__(self, x, y, energy, color=PREDATOR_COLOR):
        self.x = x
        self.y = y
        self.energy = energy
        self.angle = random.uniform(0, 2 * math.pi)
        self.gender = random.choice(["male", "female"])
        self.color = self.adjust_color_by_gender(color)

    def adjust_color_by_gender(self, color):
        if self.gender == "male":
            return (128, 0, 0)  # Darker shade
        else:
            return (250, 128, 114)  # Lighter shade

    def draw(self, screen):
        MALE_SIZE = 10
        FEMALE_SIZE = 7

        size = MALE_SIZE if self.gender == 'male' else FEMALE_SIZE

        pygame.draw.circle(screen, self.color, (int(self.x), int(self.y)), size)
```

Фигура 3.4: Функции за визуализация на хищни животни

Фигура 3.5а: Разлика между женски и мъжки хищник

Фигура 3.5б: Разлика между женски и мъжки тревопасен

3.2.2 Алгоритъм за движение

Задаваме случайно число чрез библиотеката random, като стойността на това число варира от 0 до 2* π, което е от 0 до 360. Запазва се във променлива с име angle. Функцията move() използва математическите функции синус и косинус за изчисляване на новата позиция на обекта спрямо скоростта на движение и стойността на параметъра angle (Фиг. 3.6), като ги умножава едно с друго. Получените стойности се запазват в променливи dx и dy, като тях събираме с текущите си кординати, за да получим новите кординати – new_x и new_y. След това проверяваме дали са в рамките на симулационният прозорец. Накрая обновяваме кординатите на обекта с новите стойнсоти и намаляме енергията на животното, чрез константа зададена в алгоритъма.

```
        self.angle = random.uniform(0, 2 * math.pi)
```

```
    def move(self, HERBIVORE_SPEED, ENERGY_DEPLETION_FACTOR):
        dx = HERBIVORE_SPEED * math.cos(self.angle)
        dy = HERBIVORE_SPEED * math.sin(self.angle)
        new_x = self.x + dx
        new_y = self.y + dy
        if new_x > SCREEN_WIDTH:
            new_x = SCREEN_WIDTH
        elif new_x < 390:
            new_x = 390
        if new_y > SCREEN_HEIGHT - 10:
            new_y = SCREEN_HEIGHT - 10
        elif new_y < SCREEN_HEIGHT - 590:
            new_y = SCREEN_HEIGHT - 590
        self.x = new_x
        self.y = new_y
        self.energy *= ENERGY_DEPLETION_FACTOR
```

```
    def move(self, PREDATOR_SPEED, ENERGY_DEPLETION_FACTOR):
        dx = PREDATOR_SPEED * math.cos(self.angle)
        dy = PREDATOR_SPEED * math.sin(self.angle)
        new_x = self.x + dx
        new_y = self.y + dy
        if new_x > SCREEN_WIDTH:
            new_x = SCREEN_WIDTH
        elif new_x < 390:
            new_x = 390
        if new_y > SCREEN_HEIGHT - 10:
            new_y = SCREEN_HEIGHT - 10
        elif new_y < SCREEN_HEIGHT - 590:
            new_y = SCREEN_HEIGHT - 590
        elif new_y < 0:
            new_y = 0
        self.x = new_x
        self.y = new_y
        self.energy *= ENERGY_DEPLETION_FACTOR
```

Фигура 3.6: Функции за движение на животните

3.2.3 Алгоритъм за движение към храна

Функцията ни започва с даване на стойности на две променливи – nearest_herbivore/plant и min_distance (Фиг. 3.7), като на първата слагаме стойност None, но после ще ни послужи да запазим информацията за най – близкия организъм, който е по – надолу в хранителната верига, а втората променлива ще ни послужи да запазим разстоянието от текущото животно и по – низшия организъм. След това чрез for цикъл минаваме през всички организми на долния разред и пресмятаме дистанцията чрез питагоровата теорема. Проверяваме чрез if дали текущото разстояние е по-малко от предходното и дали влиза в зоната на видимост на хищника/тревопасния. Накрая в нашата променлива min_distance остава най – малкото разстояние, а в nearest_herbivore/plant – съответстващото животно/растение. Ако nearest_herbivore/plant не е None, то тогава пресмятаме ъгълът между двата организъма чрез math.atan2 функцията и ъпдейтваме angle променливата с резултата. Ако не е намерено най-близкото тревопасно/растение, то тогава извикваме функцията turn(), която завърта текущото животно на случайна посока и отново променяме стойността на променливата self.angle.

```
def turn(self):
    self.angle += random.uniform(-math.pi / 16, math.pi / 16)

def move_towards_food(self, herbivores):
    nearest_herbivore = None
    min_distance = float('inf')

    for herbivore in herbivores:
        distance = math.sqrt((self.x - herbivore.x) ** 2 + (self.y - herbivore.y) ** 2)
        if distance < min_distance and distance <= PREDATOR_SIGHT_RANGE:
            min_distance = distance
            nearest_herbivore = herbivore

    if nearest_herbivore is not None:
        self.angle = math.atan2(nearest_herbivore.y - self.y, nearest_herbivore.x - self.x)
    else:
        self.turn()
```

```python
def turn(self):
    self.angle += random.uniform(-math.pi / 16, math.pi / 16)

def move_towards_food(self, plants):
    nearest_plant = None
    min_distance = float('inf')

    for plant in plants:
        distance = math.sqrt((self.x - plant.x) ** 2 + (self.y - plant.y) ** 2)
        if distance < min_distance and distance <= HERBIVORE_SIGHT_RANGE:
            min_distance = distance
            nearest_plant = plant

    if nearest_plant is not None:
        self.angle = math.atan2(nearest_plant.y - self.y, nearest_plant.x - self.x)
    else:
        self.turn()
```

Фигура 3.7: Функции за придвижване към храна

3.2.4 Алгоритъм за хранене

Започваме с пресмятането на разстоянието, отново, чрез формулата за разстояние между 2 точки в равнината (Фиг. 3.8). Резултатът се записва в променливата distance. След това гледаме дали резултатът е по-малък от зададеното във функцията число и ако това е така, връщаме True и покачваме енергията на текущото животно с тази на „изяденото“, а ако не е – False. След това се пренасяме в основния цикъл на симулатора, където гледаме дали резултатът е истина или неистина. Ако е истина животното от по-нисш клас в хранителната верига бива премахнато.

```python
def eat(self, herbivore):
    distance = math.sqrt((self.x - herbivore.x) ** 2 + (self.y - herbivore.y) ** 2)
    if distance <= 20:
        self.energy += herbivore.energy
        return True
    else:
        return False
```

```python
for herbivore in herbivores:
    if not 90 <= int(humidity) <= 100:
        if predator.eat(herbivore):
            herbivores.remove(herbivore)
            herbivores_count -= 1
```

```python
for plant in plants:
    if not 90 <= int(humidity) <= 100:
        if herbivore.eat(plant):
            plants.remove(plant)
            plants_count -= 1
```

```python
def eat(self, plant):
    distance = math.sqrt((self.x - plant.x) ** 2 + (self.y - plant.y) ** 2)
    if distance <= 15:
        self.energy += plant.energy
        return True
    else:
        return False
```

Фигура 3.8: Функции за хранене

3.2.5 Алгоритъм за репродукция

В основния цикъл извикваме функцията reproduce() на променливата new_herbivores/predators (Фиг 3.9). Тя проверява дали текущото ни животно е женско и дали енергията му отговаря на изискванията за репордукция. След това се енергията на родителя се намалява със зададената константа в алгоритъма REPRODUCTION_ENERGY_COST и функцията връща на променливата ново животно от същия клас с нови атрибути –

координатите на родителя и нова, по-малка енергия. След това в основния цикъл добавяме новото животно в списъка herbivores/predators_count.

```
new_herbivore = herbivore.reproduce()
if new_herbivore is not None:
    herbivores.append(new_herbivore)
    herbivores_count += 1
```

```
def reproduce(self):
    if self.gender == "female":
        if self.energy >= HERBIVORE_FOOD_THRESHOLD + REPRODUCTION_ENERGY_COST:
            self.energy -= REPRODUCTION_ENERGY_COST
            return Herbivore(self.x, self.y, self.energy / 2)
        else:
            return None
```

```
new_predator = predator.reproduce()
if new_predator is not None:
    predators.append(new_predator)
    predators_count += 1
```

```
def reproduce(self):
    if self.gender == "female":
        if self.energy >= PREDATOR_FOOD_THRESHOLD + REPRODUCTION_ENERGY_COST:
            self.energy -= REPRODUCTION_ENERGY_COST
            return Predator(self.x, self.y, self.energy / 2)
        else:
            return None
```

Фигура 3.9: Функции за репродукция на животни – хищници и тревопасни

Размножаването при растенията става по подобен начин чрез функцията generate_new_plant() (Фиг. 3.10), където взимаме нови координати в близост до „майчиното“ растение и проверяваме дали те не превишават границите на симулационният прозорец. Функцията връща новото растение, ако изпълва изискванията и се рисува на екрана, като се добавя нова бройка в променливата plants_count, която следи броя на растенията в симулацията.

```
    for plant in plants:
        plant.grow()

        if random.random() < PLANT_REPRODUCTION_RATE / 5:
            new_plant = plant.generate_new_plant()
            if new_plant is not None:
                plants.append(new_plant)
                plants_count += 1
```

```
def generate_new_plant(plant):
    new_x = plant.x + random.randint(-200, 200)
    new_y = plant.y + random.randint(-200, 200)
    if 390 <= new_x <= SCREEN_WIDTH and SCREEN_HEIGHT - 590 <= new_y <= SCREEN_HEIGHT - 10:
        return Plant(new_x, new_y, plant.energy / 200)
    else:
        return None
```

Фигура 3.10: Функция за репродукция на растения

3.2.6 Алгоритъм за симулация на торнадо

Той се състои от 3 функции, които са за трите класа. Започваме от мейн луупа, където имаме три листа – plant_angles, herbivore_angles и predator_angles (Фиг 3.11), който съдържа произволна избрани стойности от 0 до 360. Ако координатите на мишката изпълняват условието на if – а, тогава извикваме една от функциите – tornado_plants(), tornado_herbivores() или tornado_predators() (Фиг 3.12), зависимост от класа. Функцията взима няколко параметъра – лист с конкретния клас, координатите на торнадото, радиуса му, скоростта, с която да завърта попадналите организми, скоростта на привличане и списък с ъглите на конкретния клас. Минаваме през всички единици от конкретния клас и пресмятаме разстоянието между торнадото и животното/растението и взимаме ъгълът на конкретното животно от листа. Проверяваме дали разстоянието е по – малко от радиуса на торнадото и ако е така проверяваме дали произволно избраното число от 0 до 1 е по-малко от променливата death_probability и ако е така премахваме съответното животно/растение. Ако числото е по-голямо от променливата death_probability обновяваме ъгъла и разстоянието между торнадото и животното/растението на база силата на привличане и обновяваме координатите с новите, както и ъгълът за конкретния организъм.

```
plant_angles = [random.uniform(0, 2 * math.pi) for _ in range(int(plants_count + 1000))]
herbivore_angles = [random.uniform(0, 2 * math.pi) for _ in range(int(herbivores_count + 1000))]
predator_angles = [random.uniform(0, 2 * math.pi) for _ in range(int(predators_count + 1000))]

if 390 <= image1_x <= SCREEN_WIDTH and SCREEN_HEIGHT - 590 <= image1_y <= SCREEN_HEIGHT - 10:
    tornado_plants(plants, image1_x, image1_y, radius, speed, attraction_speed, plant_angles,)
    plants_count = len(plants)
    tornado_herbivores(herbivores, image1_x, image1_y, radius, speed, attraction_speed, herbivore_angles)
    herbivores_count = len(herbivores)
    tornado_predators(predators, image1_x, image1_y, radius, speed, attraction_speed, predator_angles)
    predators_count = len(predators)
```

Фигура 3.11: Листове с ъгли и извикване на функция

```
death_probability = 0.1

def tornado_plants(plants, mouse_x, mouse_y, radius, speed, attraction_speed, plant_angles):
    for i, plant in enumerate(plants):
        dx = (mouse_x - plant.x)
        dy = (mouse_y - plant.y)
        distance = math.sqrt(dx**2 + dy**2)
        angle = plant_angles[i]

        if distance <= radius:
            if random.random() < death_probability:
                plants.remove(plant)
            new_angle = angle + speed
            new_distance = distance - attraction_speed
            new_x = mouse_x + math.cos(new_angle) * new_distance
            new_y = mouse_y + math.sin(new_angle) * new_distance

            new_x = max(390, min(new_x, SCREEN_WIDTH))
            new_y = max(SCREEN_HEIGHT - 590, min(new_y, 590))

            plant.x, plant.y = new_x, new_y
            plant_angles[i] = new_angle
```

```python
def tornado_herbivores(herbivores, mouse_x, mouse_y, radius, speed, attraction_speed, herbivore_angles):
    for i, herbivore in enumerate(herbivores):
        dx = (mouse_x - herbivore.x)
        dy = (mouse_y - herbivore.y)
        distance = math.sqrt(dx**2 + dy**2)
        angle = herbivore_angles[i]

        if distance <= radius:
            if random.random() < death_probability:
                herbivores.remove(herbivore)
            new_angle = angle + speed
            new_distance = distance - attraction_speed
            new_x = mouse_x + math.cos(new_angle) * new_distance
            new_y = mouse_y + math.sin(new_angle) * new_distance

            new_x = max(390, min(new_x, SCREEN_WIDTH))
            new_y = max(SCREEN_HEIGHT - 590, min(new_y, 590))

            herbivore.x, herbivore.y = new_x, new_y
            herbivore_angles[i] = new_angle
```

```python
def tornado_predators(predators, mouse_x, mouse_y, radius, speed, attraction_speed, predator_angles):
    for i, predator in enumerate(predators):
        dx = (mouse_x - predator.x)
        dy = (mouse_y - predator.y)
        distance = math.sqrt(dx**2 + dy**2)
        angle = predator_angles[i]

        if distance <= radius:
            if random.random() < death_probability:
                predators.remove(predator)
            new_angle = angle + speed
            new_distance = distance - attraction_speed
            new_x = mouse_x + math.cos(new_angle) * new_distance
            new_y = mouse_y + math.sin(new_angle) * new_distance

            new_x = max(390, min(new_x, SCREEN_WIDTH))
            new_y = max(SCREEN_HEIGHT - 590, min(new_y, 590))

            predator.x, predator.y = new_x, new_y
            predator_angles[i] = new_angle
```

Фигура 3.12: Функции за трите класа, симулиращи торнадо

3.2.7 Алгоритъм за визуализация на графика

Стойностите на променливите plants_count, herbivores_count и predators_count се записват в списъците – plants_counts, herbivores_counts и predators_counts. При извикване на функцията plot_results() използваме Mathplotlib (Фиг 3.13) и като параметри задаваме 3те списъка. След като графиката се нарисува, наблюдателят може да види изменението на живите организми през различни периоди.

```python
def plot_results(herbivores_counts, predators_counts, plants_counts):
    import matplotlib.pyplot as plt

    time_steps = list(range(len(herbivores_counts)))

    plt.plot(time_steps, herbivores_counts, label="Herbivores", color="blue")
    plt.plot(time_steps, predators_counts, label="Predators", color="red")
    plt.plot(time_steps, plants_counts, label="Plants", color="green")

    plt.xlabel("Time Steps")
    plt.ylabel("Count")
    plt.title("Population Dynamics")
    plt.legend()

    plt.show()

    print(herbivores_counts)
    print(predators_counts)
    print(plants_counts)
```

Фигура 3.13: Функция за визуализация на графика

3.2.8 Направа на приложение

Приложението, чрез което се контролира и наблюдава симулацията също е направено с Pygame. За бутоните са използвани правоъгълници, за натискането на които седят if statement-и. Същото важи и за текст боксовете през, които се въвеждат стойностите на параметрите.

Глава 4
Ръководство на потребителя

4.1 Инсталиране на приложението

4.1.1 Необходими налични технологии на локалната машина (Dependencies)

За да може да се снабдите с приложението, ще са Ви необходими следните инсталирани технологии:

- Python
- Pygame
- Matplotlib

4.1.2 Инсталиране на Python

За да стартирате приложението, то на локалната машина трябва да
има инсталиран Python. За целта трябва да последвате инструкциите за инсталиране от официалния сайт на езика: https://www.python.org/downloads/)

След това отворете вашето CMD (Command Prompt) или Terminal и напишете следните команди:

- **pip install pygame**
- **pip install matplotlib**

4.1.2 Клониране на проекта от GitHub

Приложението може да бъде клонирано от GitHub хранилището на проекта, което се намира на
https://github.com/MartinovG/VSE_9klas_ecosystem_simulator чрез командата:

- **git clone https://github.com/MartinovG/VSE_9klas_ecosystem_simulator.git**

4.2 Изпълняване на приложението

След като вече имате инсталирани всички необходими технологии, можете да пуснете приложението като навигирате до src директорията, като напишете в терминала:

- **cd src**

и след това стартирате приложението чрез командата:

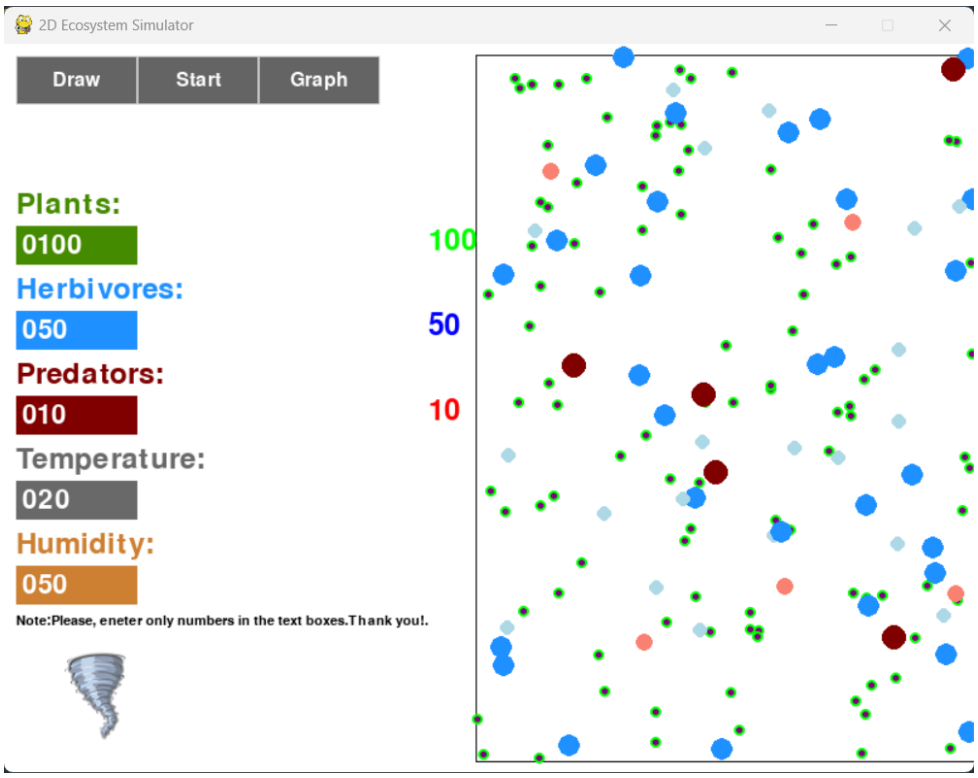
- **py main.py**

или аналогичното:

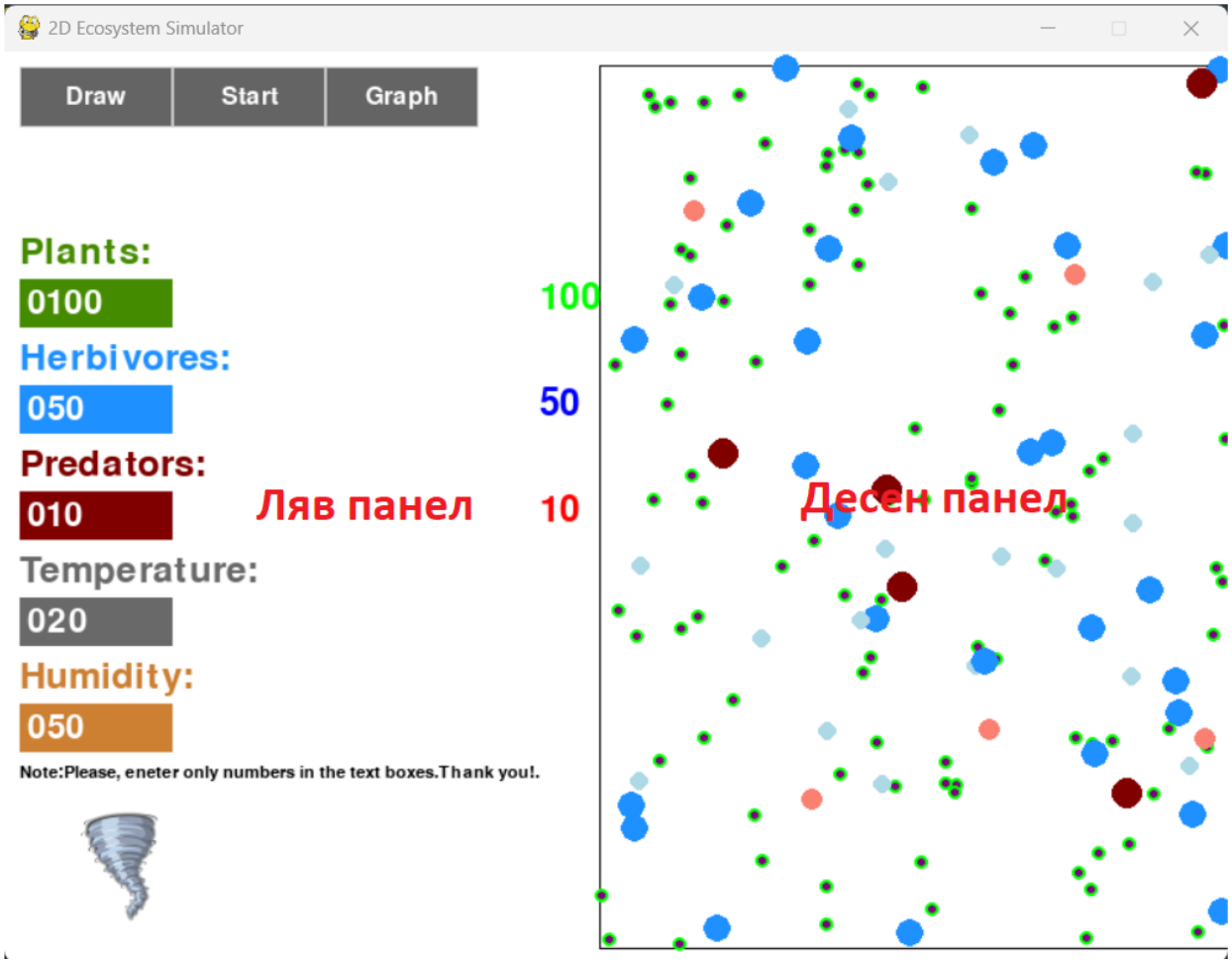
- **python main.py**

4.3 Основно оформление (Layout) на приложението

След въвеждане на горните команди, приложението ще се стартира и ще се покаже неговият основен екран (Фиг. 4.1). Той се състои от два основни панела: ляв и десен (Фиг. 4.2). В левия седи менюто за параметризиране на симулацията, а в десния е прозореца, през който можем да наблюдаваме симулацията.



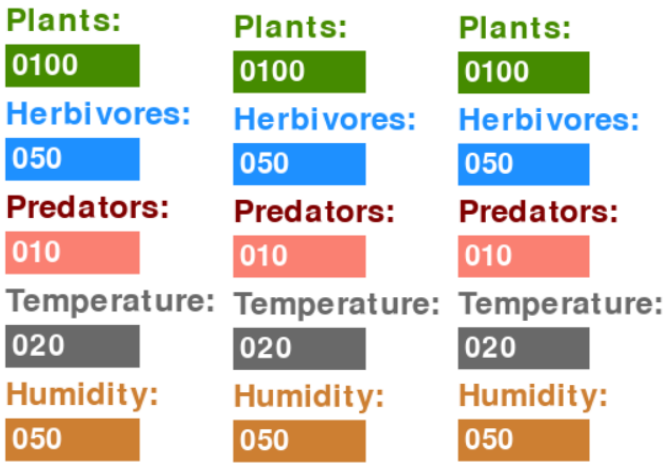
Фигура 4.1: Цялостен изглед на приложението



Фигура 4.2: Панели на главния екран

4.4 Настройване на параметри

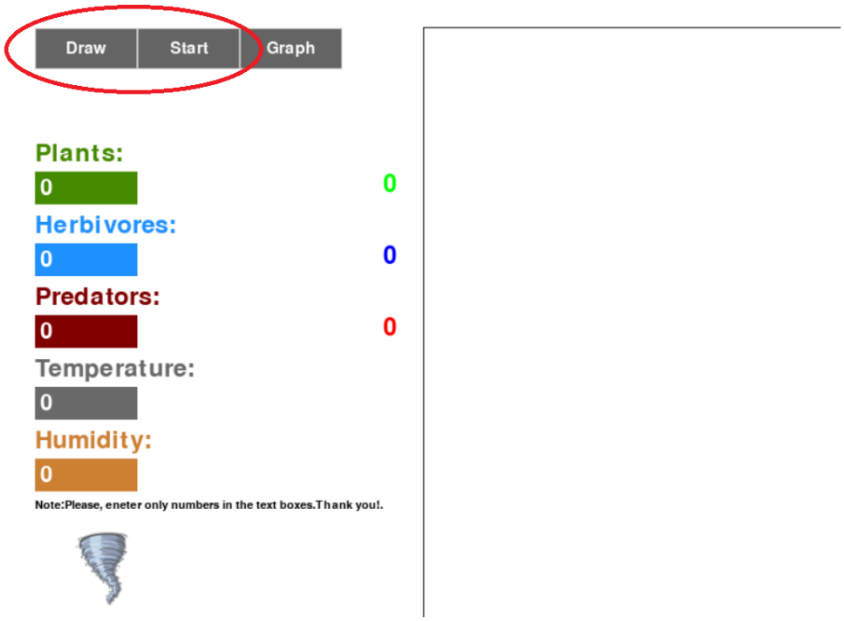
В левият панел виждаме 5 бутона в различни цветове, които ще светнат в различен цвят, когато потребителят натисне върху тях за да покаже, че те са активни (Фиг 4.3). Когато те са активни могат да бъдат въведени стойности, като ако потребителят не желае да добавя даден клас животно, то той ще трябва да остави съответния текст бокс на 0, а не празен.



Фигура 4.3: Активни и не активни текст боксове

4.5 Стартиране на симулацията

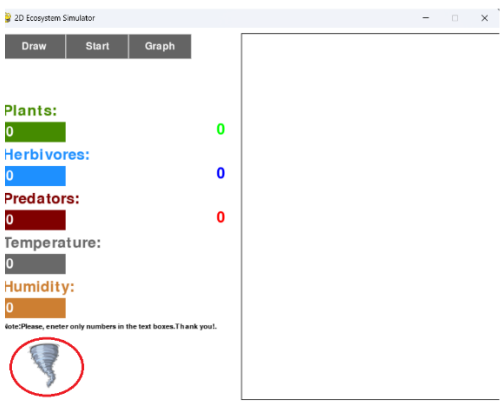
За да може потребителят да започне симулацията, то той трябва да зададе параметри на симулацията и след това да натисне бутона „Draw“ (Фиг 4.4), който ще нарисува животните, които ще бъдат симулирани. В този момент тя още не е стартирана. За да стане това, потребителят трябва да натисне и бутона „Start“, чрез който по всяко време може да паузира и стартира симулацията.

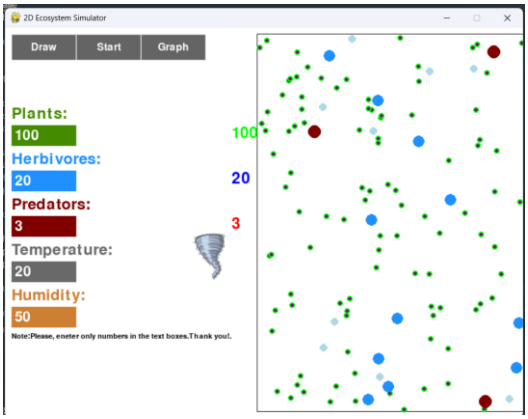


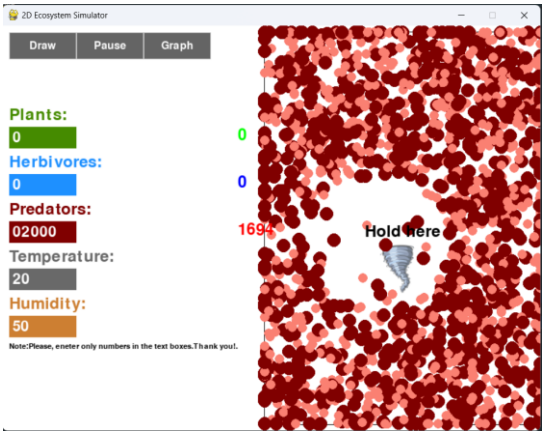
Фигура 4.4: Бутоните „Draw“ и „Start/Pause“

4.6 Торнадо

С торнадото можем да симулираме природното бедствие. То работи, само когато симулацията е пусната и нанася поражение на биоразнообразието (Фиг 4.5). Неговата позиция е в долния десен ъгъл и чрез привлачването му върху симулационния прозорец виждаме щетите, които би нанесло в реалния живот.







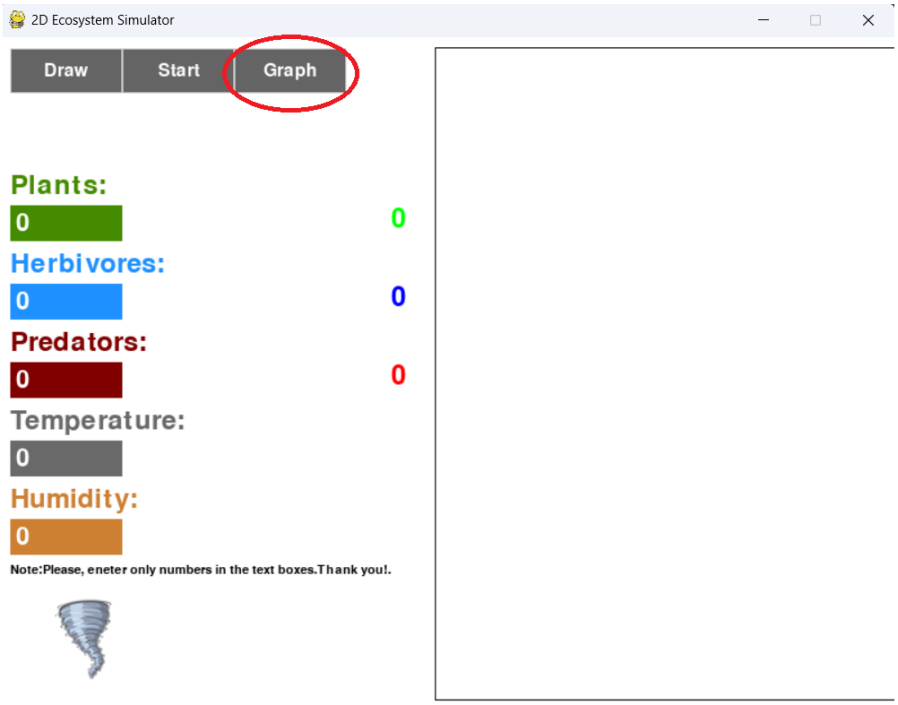
Фигура 4.5а: Иконата на торнадото

Фигура 4.5б: Приближаване на торнадото към прозореца

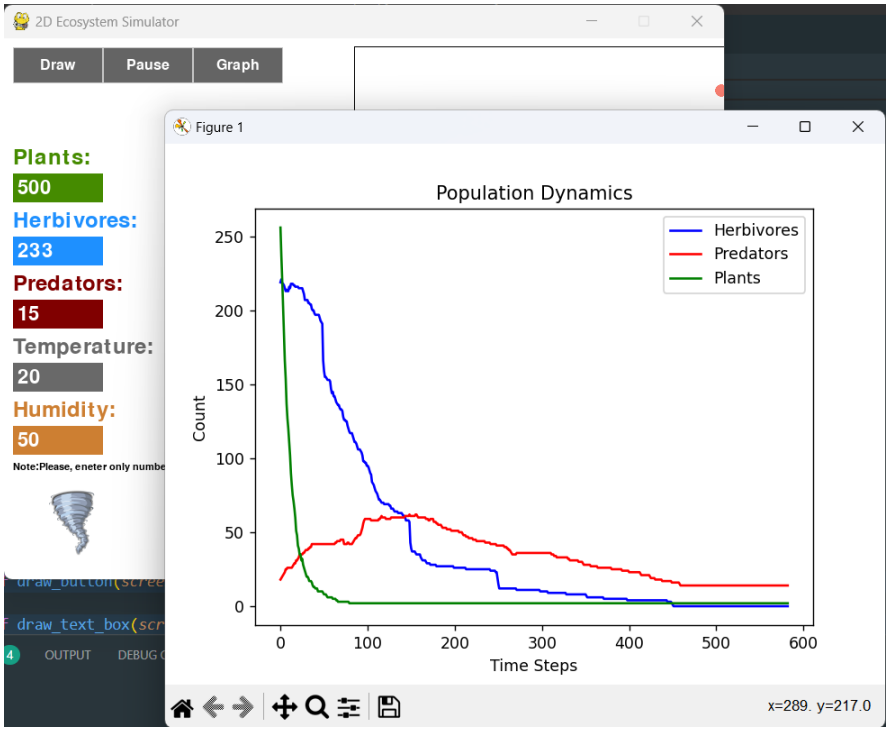
Фигура 4.5в: Вредата, която нанася на екосистемата

4.7 Резултатна графика

Потребителят може да достъпи до графика, която сравнява популациите на трите класа през единица време. Това става чрез бутона „Graph“ (Фиг 4.6), който след като бъде натиснат се отваря графиката (Фиг 4.7), симулаторът спира да бъде използваем, тъй като графиката ни е краен резултат, тоест когато я отворим считаме симулацията за приключена. При затваряне на графиката и приложението със симулатора също се затваря.



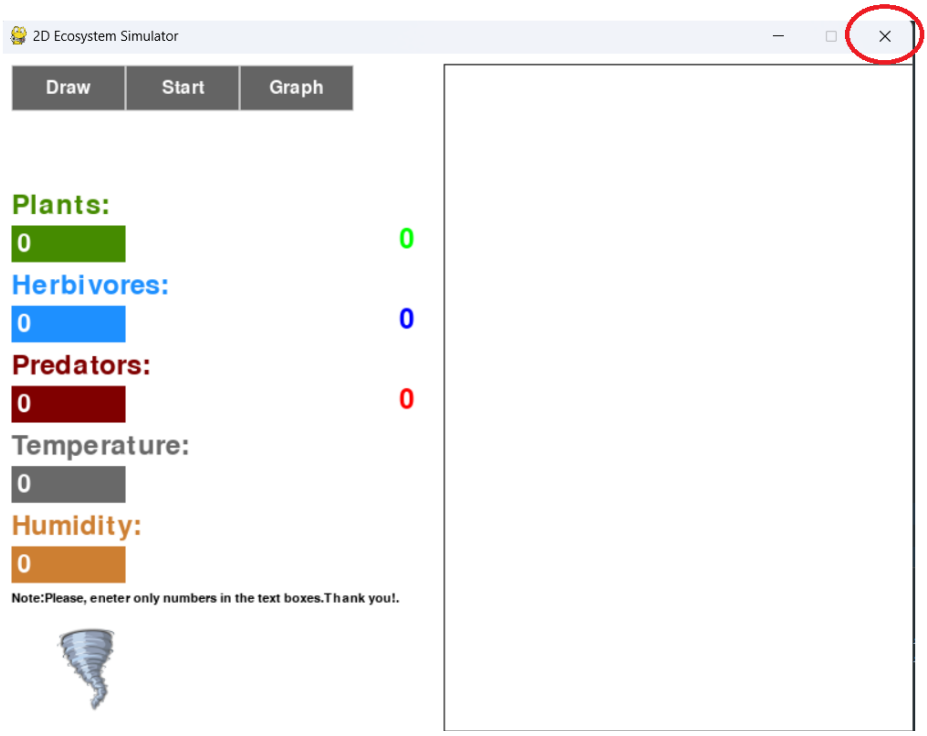
Фигура 4.6: Бутонът "Graph"



Фигура 4.7: Графика на популацията на животните и растенията

4.8 Излизане от приложението

За да се излезе от приложението трябва само да се натисне хиксът в горния ляв ъгъл (Фиг 4.8).



Фигура 4.8: Бутон за излизане от приложението

Заключение

Настоящия курсов проект реализира опростен симулатор на екосистема, който използва авторски алгоритъм. В текущата разработка бе реализирано и приложение на Pygame, чрез което се моделира и наблюдава симулацията.

Вариантите за подобряване на разработката са много: по – сложни и детайлни алгоритми за самата симулация, подобряване на GUI – я, добавяне на повече видове животни, имплементиране на човешка дейност, реализиране на още природни бедствия, по – голям контрол над симулацията и много други.

Библиография

- Stack Overflow - https://stackoverflow.com/
- Geeks for geeks - https://www.geeksforgeeks.org/
- Python documentation - https://docs.python.org/3/
- Pygame documentation - https://www.pygame.org/docs/
- Matplotlib documentation - https://matplotlib.org/stable/index.html
- Smith, J. (2020). Ecosystems: Complexity and Interactions. 2nd Edition. Nature Press
- Ecosystem Dynamics - https://www.youtube.com/watch?v=5ZU1mqYqajU

Съдържание