



**WYDZIAŁ
ELEKTROTECHNIKI
I INFORMATYKI**
POLITECHNIKI RZESZOWSKIEJ

Adam Szałański 164018

Łucja Pałkus 160787

Projekt z przedmiotu
Programowanie w języku Python

Gra symulacyjna Wyspa Wilków

Rzeszów, 2022

Spis treści

1. Cel i założenia projektu	3
1.1. Cel projektu	3
1.2. Założenia projektu	3
2. Struktura projektu	4
2.1. main.py	5
2.2. Settings.py	6
2.3. Level.py	6
2.4. Field.py	7
2.5. Animals.py	8
2.5.1. klasa Tile	9
2.5.2. klasa Animal	9
2.5.3. klasa WolfFemale	10
2.5.4. klasa WolfMale	11
2.5.5. klasa RabbitFemale	12
2.5.6. klasa RabbitMale	13
3. Prezentacja działania programu	14

1. Cel i założenia projektu

1.1. Cel projektu

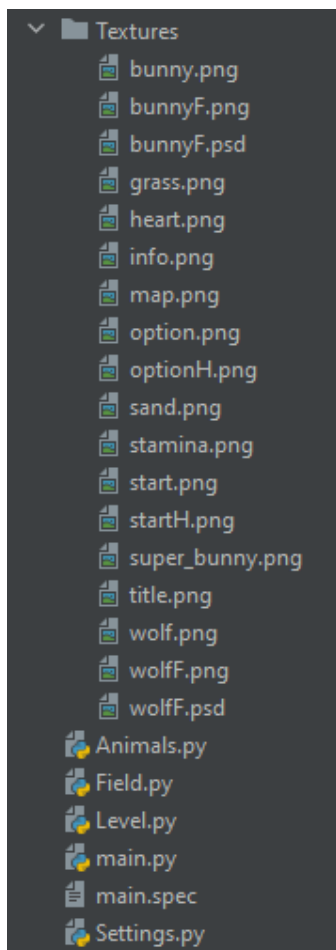
Celem projektu było utworzenie prostej gry symulacyjnej wykorzystując język programowania Python.

1.2. Założenia projektu

Gra przedstawia sytuację w której na planszy o rozmiarze zdefiniowanym przez użytkownika, znajdują się wilki oraz króliki. Zwierzęta te mogą być zarówno płci męskiej jak i żeńskiej. O ich początkowej ilości decyduje użytkownik w podobny sposób, jak o rozmiarach planszy. W momencie kiedy zwierzęta powiększają swoje stado, wybór pomiędzy płcią nowego członka stada rozwiązywany jest przy pomocy generatora liczb losowych. Każde ze zwierząt ma swój osobny poziom tłuszczy, od którego zależy jakie akcje mogą wykonywać oraz która z możliwych akcji będzie miała najwyższy priorytet. Poziom tłuszczy spada przy każdym kroku zwierzęcia, a także w momencie kiedy zwierzęta się rozmnażają lub walczą ze sobą. Aby uzupełnić swój tłuszczy, zwierze musi coś zjeść. W przypadku wilków pokarmem są króliki, natomiast w przypadku królików jedzeniem jest trawa. Trawa rośnie na polach planszy. Kiedy królik zje trawę, pole z którego trawę zjedzono zmienia się w pole piaszczyste. Pola pustynne po jakimś czasie ponownie zarastają trawą tak, aby króliki nie wyginęły z głodu.

2. Struktura projektu

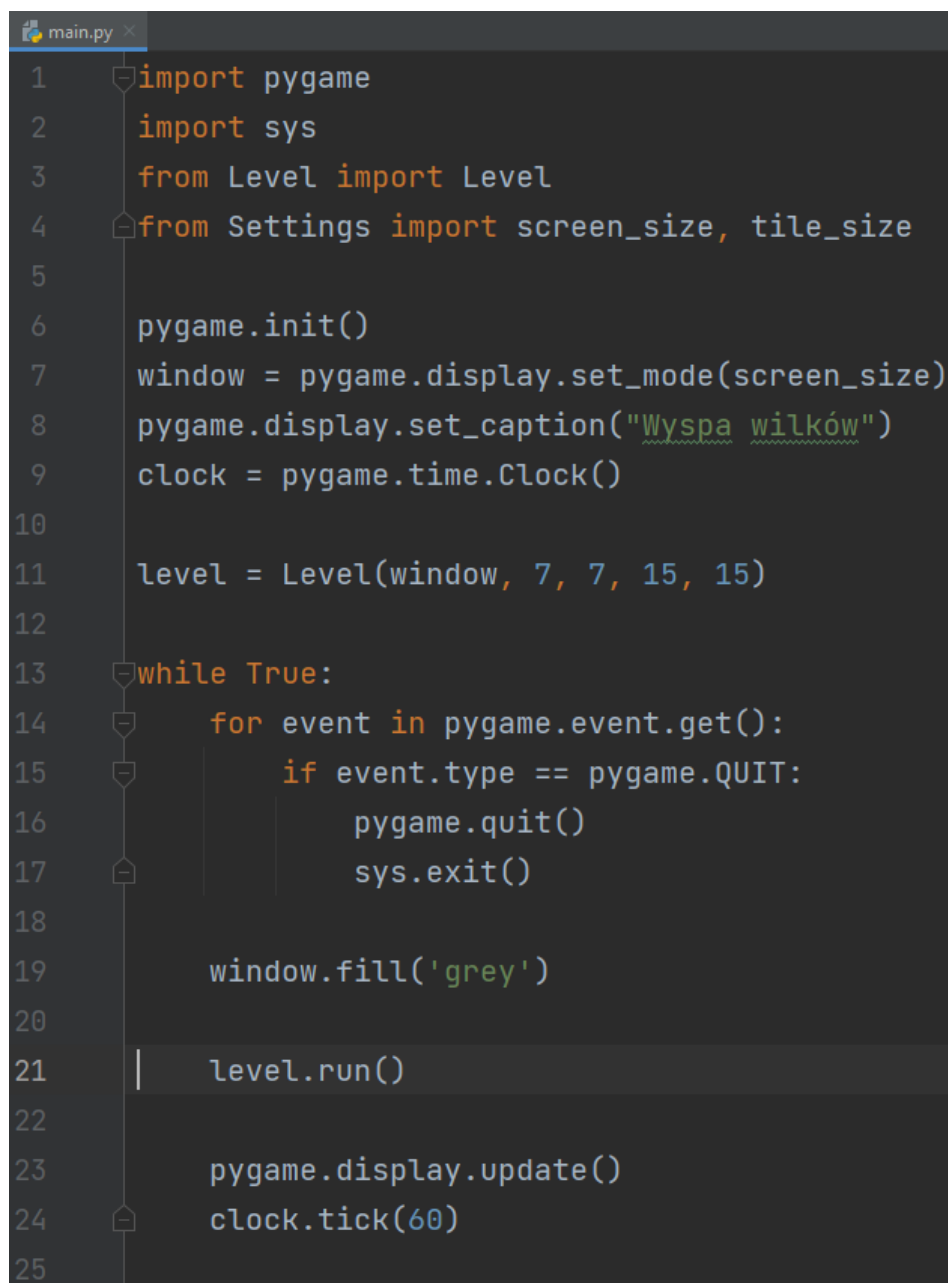
Projekt składa się z 5 skryptów napisanych w języku programowania Python oraz z 16 obrazów będących teksturami wykorzystywanymi w programie (rys: 2.1).



Rysunek 2.1: Struktura plików projektu

2.1. main.py

Plik main.py jest głównym skryptem sterującym grą (rys: 2.2). Odbywają się w nim wszystkie inicjalizacje konieczne do działania gry oraz główna pętla sterująca, w której odbywa się symulacja. Na rysunku 2.2 widoczna jest deklaracja początkowej ilości wilków, wilczyc, królików i królic poprzez konstruktor klasy Level. Ich początkowe wartości to odpowiednio 7, 7, 15, 15.

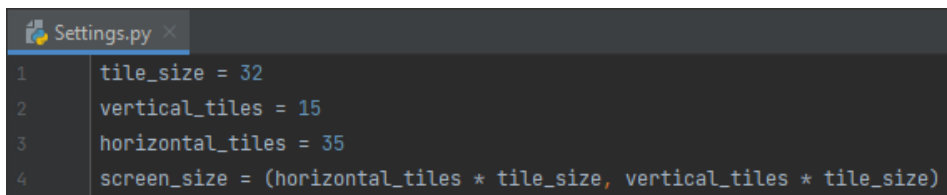


```
1 import pygame
2 import sys
3 from Level import Level
4 from Settings import screen_size, tile_size
5
6 pygame.init()
7 window = pygame.display.set_mode(screen_size)
8 pygame.display.set_caption("Wyspa wilków")
9 clock = pygame.time.Clock()
10
11 level = Level(window, 7, 7, 15, 15)
12
13 while True:
14     for event in pygame.event.get():
15         if event.type == pygame.QUIT:
16             pygame.quit()
17             sys.exit()
18
19     window.fill('grey')
20
21     level.run()
22
23     pygame.display.update()
24     clock.tick(60)
25
```

Rysunek 2.2: Kod pliku main.py

2.2. Settings.py

W pliku Settings.py ustalane są wymiary planszy.

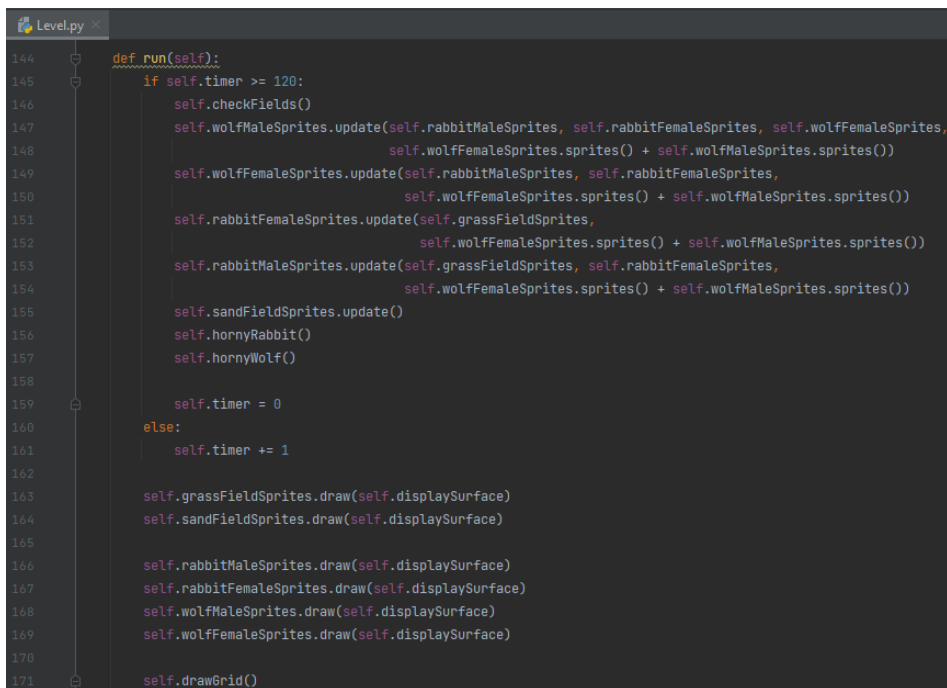


```
1 tile_size = 32
2 vertical_tiles = 15
3 horizontal_tiles = 35
4 screen_size = (horizontal_tiles * tile_size, vertical_tiles * tile_size)
```

Rysunek 2.3: Zawartość pliku Settings.py

2.3. Level.py

W klasie Level znajdują się wszystkie grupy obiektów występujących w symulacji oraz funkcje wykorzystywane do wykonywania akcji w każdej turze.



```
144 def run(self):
145     if self.timer >= 120:
146         self.checkFields()
147         self.wolfMaleSprites.update(self.rabbitMaleSprites, self.rabbitFemaleSprites, self.wolfFemaleSprites,
148                                     self.wolfFemaleSprites.sprites() + self.wolfMaleSprites.sprites())
149         self.wolfFemaleSprites.update(self.rabbitMaleSprites, self.rabbitFemaleSprites,
150                                     self.wolfFemaleSprites.sprites() + self.wolfMaleSprites.sprites())
151         self.rabbitFemaleSprites.update(self.grassFieldSprites,
152                                         self.wolfFemaleSprites.sprites() + self.wolfMaleSprites.sprites())
153         self.rabbitMaleSprites.update(self.grassFieldSprites, self.rabbitFemaleSprites,
154                                     self.wolfFemaleSprites.sprites() + self.wolfMaleSprites.sprites())
155         self.sandFieldSprites.update()
156         self.hornyRabbit()
157         self.hornyWolf()
158
159     self.timer = 0
160     else:
161         self.timer += 1
162
163     self.grassFieldSprites.draw(self.displaySurface)
164     self.sandFieldSprites.draw(self.displaySurface)
165
166     self.rabbitMaleSprites.draw(self.displaySurface)
167     self.rabbitFemaleSprites.draw(self.displaySurface)
168     self.wolfMaleSprites.draw(self.displaySurface)
169     self.wolfFemaleSprites.draw(self.displaySurface)
170
171     self.drawGrid()
```

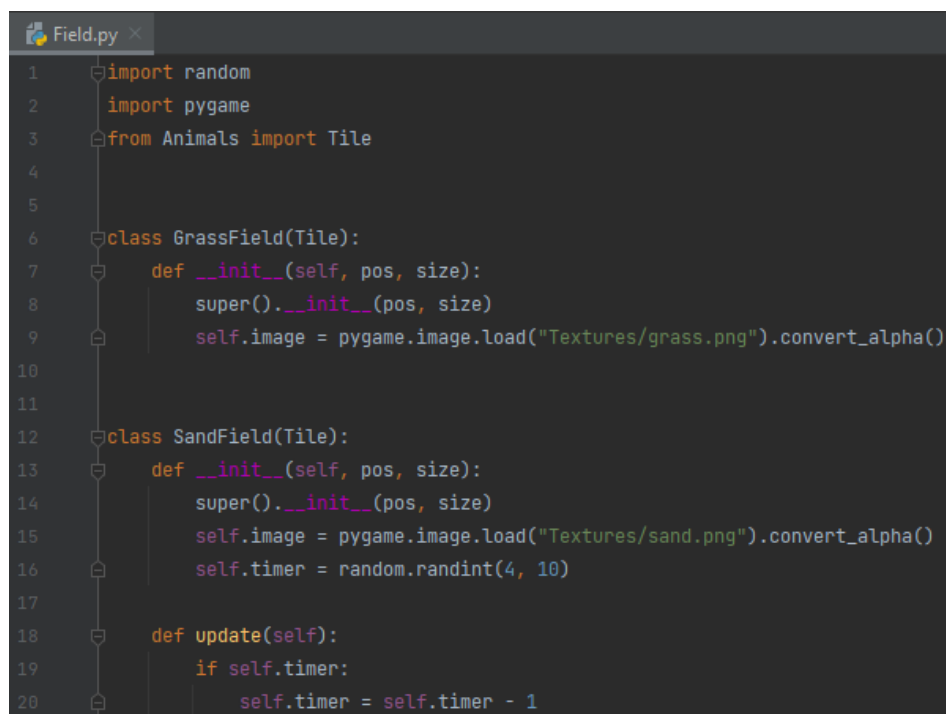
Rysunek 2.4: Fragment kodu w pliku Level.py odpowiedzialny za wywoływanie akcji w kolejnej turze symulacji

2.4. Field.py

W pliku Field.py znajdują się dwie klasy:

- GrassField
- SandField

Odpowiadają one odpowiednio polu trawiastemu i polu piaszczystemu. Dodatkowo piaszczyste pole posiada losowo ustawiany licznik, który odlicza czas do odrośnięcia trawy na tym polu.



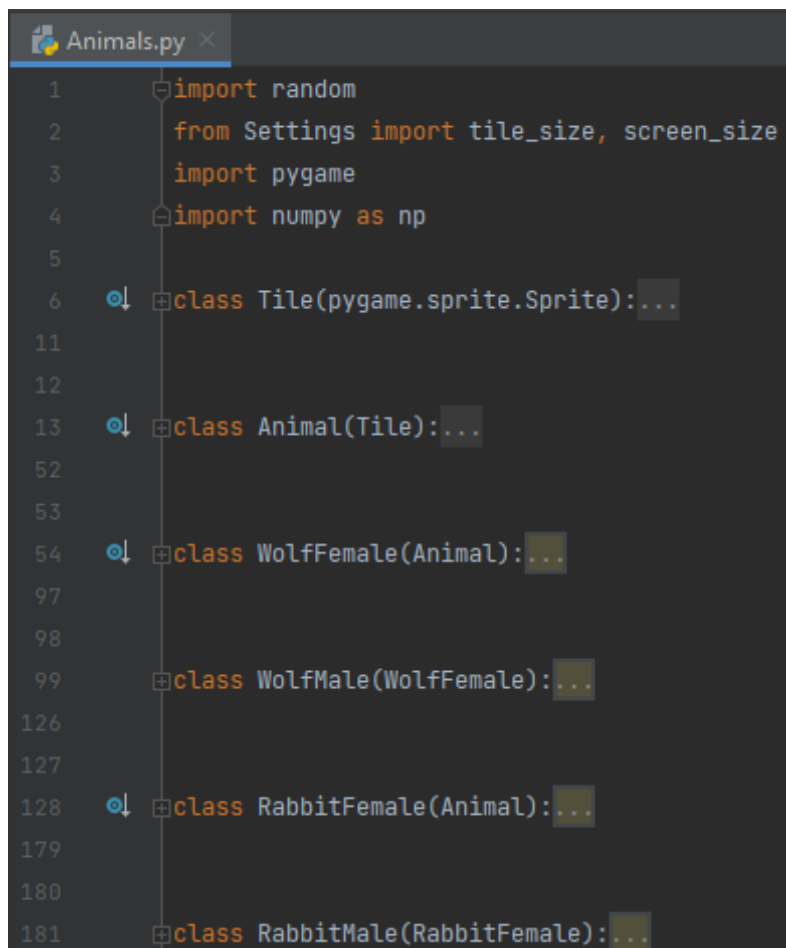
```
1  import random
2  import pygame
3  from Animals import Tile
4
5
6  class GrassField(Tile):
7      def __init__(self, pos, size):
8          super().__init__(pos, size)
9          self.image = pygame.image.load("Textures/grass.png").convert_alpha()
10
11
12  class SandField(Tile):
13      def __init__(self, pos, size):
14          super().__init__(pos, size)
15          self.image = pygame.image.load("Textures/sand.png").convert_alpha()
16          self.timer = random.randint(4, 10)
17
18      def update(self):
19          if self.timer:
20              self.timer = self.timer - 1
```

Rysunek 2.5: Zawartość pliku Field.py

2.5. Animals.py

Plik Animals.py zawiera 6 klas:

- Tile
- Animal
- WolfFemale
- WolfMale
- RabbitFemale
- RabbitMale



```
1  import random
2  from Settings import tile_size, screen_size
3  import pygame
4  import numpy as np
5
6  class Tile(pygame.sprite.Sprite):...
11
12
13  class Animal(Tile):...
52
53
54  class WolfFemale(Animal):...
97
98
99  class WolfMale(WolfFemale):...
126
127
128  class RabbitFemale(Animal):...
179
180
181  class RabbitMale(RabbitFemale):...
```

Rysunek 2.6: Zawartość pliku Animals.py

2.5.1. klasa Tile

Klasa Tile odpowiedzialna jest za wywołanie funkcji inicjalizacyjnych PyGame koniecznych do wyświetlania tekstur dla obiektu danej klasy w oknie symulacji. W związku z tym jest ona dziedziczona przez pozostałe klasy z pliku Animals.py.

```
6 class Tile(pygame.sprite.Sprite):  
7     def __init__(self, pos, size):  
8         super().__init__()   
9         self.image = pygame.Surface((size, size))  
10        self.rect = self.image.get_rect(topleft=pos)
```

Rysunek 2.7: Klasa Tile

2.5.2. klasa Animal

Klasa ta jest podstawą dla każdego zwierzęcia. Definiuje ona podstawowe funkcje niezależne od gatunku zwierzęcia, takie jak wykonywanie kroku lub sprawdzanie, czy zwierze nie umarło z głodu.

```
13 class Animal(Tile):  
14     def __init__(self, pos, size):  
15         super().__init__(pos, size)  
16         self.fat = 20  
17         self.lastPosition = None  
18  
19     def update(self, availablePositions=[]):  
20         if len(availablePositions) == 0:  
21             availablePositions = self.randomStep()  
22         availablePositions = list(dict.fromkeys(availablePositions))  
23         step = None  
24         while step == self.lastPosition or step is None:  
25             step = random.choice(availablePositions)  
26             if (len(availablePositions) == 1):
```

Rysunek 2.8: Fragment klasy Animal

2.5.3. klasa WolfFemale

WolfFemale jest to klasa wilczycy, która dziedziczy po klasie Animal. W klasie znajdują się funkcje definiujące podstawowe zachowania każdego wilka, takie jak np. pogoń za królikiem gdy wilczyca jest głodna lub ucieczka przed partnerem gdy wilczyca ma zbyt niski poziom tłuszczu.

```
54 class WolfFemale(Animal):  
55     def __init__(self, pos, size):  
56         super().__init__(pos, size)  
57         self.image = pygame.image.load("Textures/wolfF.png").convert_alpha  
58         self.fat = 50  
59  
60     def update(self, rabbitMaleSprites, rabbitFemaleSprites, wolfSprites,  
61               availablePositions):  
62         if len(availablePositions) == 0:  
63             availablePositions = self.rabbitStep(rabbitMaleSprites.sprites)  
64             safePositions = self.wolfRunAway(wolfSprites, availablePositions)  
65             availablePositions = safePositions.copy()  
66             super().update(availablePositions)  
67  
68     def rabbitStep(self, rabbitSprites):
```

Rysunek 2.9: Fragment klasy WolfFemale



Rysunek 2.10: Tekstura wilczycy

2.5.4. klasa WolfMale

WolfMale dziedziczy po klasie WolfFemale. Klasa ta jest rozszerzona o funkcję pogoni za partnerką gdy poziom tłuszczu jest dostatecznie wysoki.

```
99 class WolfMale(WolfFemale):
100     def __init__(self, pos, size):
101         super().__init__(pos, size)
102         self.image = pygame.image.load("Textures/wolf.png").convert_alpha()
103
104     def update(self, rabbitMaleSprites, rabbitFemaleSprites, femaleWolfSprites):
105         availablePositions = []
106         if self.fat > 7:
107             availablePositions = self.wolfStep(femaleWolfSprites)
108         super().update(rabbitMaleSprites, rabbitFemaleSprites, femaleWolfSprites)
109
110     def wolfStep(self, wolfFemaleSprites):
111         xr = self.rect.x
112         yr = self.rect.y
```

Rysunek 2.11: Fragment klasy WolfMale



Rysunek 2.12: Tekstura wilka

2.5.5. klasa RabbitFemale

RabbitFemale jest to klasa królicy, która dziedziczy po klasie Animal. W klasie znajdują się funkcje definiujące podstawowe zachowania każdego królika, takie jak np. zjadanie trawy, gdy króliczka jest głodna lub ucieczka przed wilkiem, gdy ten znajduje się w pobliżu.

```
128 class RabbitFemale(Animal):
129     def __init__(self, pos, size):
130         super().__init__(pos, size)
131         self.inDanger = False
132         self.fat = 10
133         self.image = pygame.image.load("Textures/bunnyF
134
135     def update(self, grassSprites, wolfSprites, availab
136         if len(availablePositions) == 0:
137             availablePositions = self.grassStep(grassSp
138             safePositions = self.wolfRunAway(wolfSprites, a
139             if len(safePositions) == len(availablePositions
140                 self.inDanger = False
141             availablePositions = safePositions.copy()
```

Rysunek 2.13: Fragment klasy RabbitFemale



Rysunek 2.14: Tekstura królicy

2.5.6. klasa RabbitMale

RabbitMale dziedziczy po klasie RabbitFemale. Klasa ta jest rozszerzona o funkcję pogoni za partnerką gdy poziom tłuszczy jest dostatecznie wysoki.

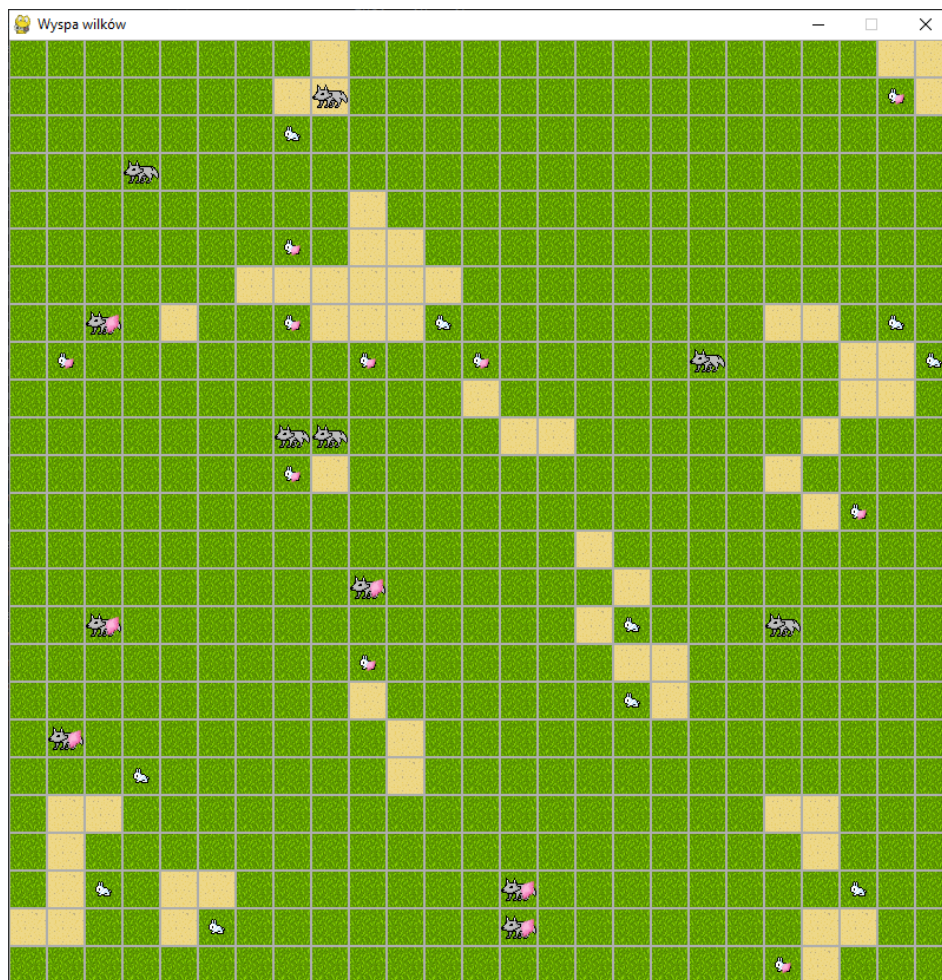
```
181 class RabbitMale(RabbitFemale):|
182     def __init__(self, pos, size):
183         super().__init__(pos, size)
184         self.image = pygame.image.load("Textures/bunny.png")
185
186     def update(self, grassSprites, femaleRabbitSprites, wolfSprites, availablePositions):
187         availablePositions = []
188         if self.fat > 7:
189             availablePositions = self.rabbitStep(femaleRabbitSprites)
190         super().update(grassSprites, wolfSprites, availablePositions)
191
192     def rabbitStep(self, femaleRabbitSprites):
```

Rysunek 2.15: Fragment klasy RabbitMale



Rysunek 2.16: Tekstura królika

3. Prezentacja działania programu



Rysunek 3.17: Zrzut ekranu z symulacji