Symulacja Systemów i Modelowanie
Raport 2
Wariant Flocking Algorithm zakładający omijanie losowo generowanych przeszkód

Adam Staniszewski
Dariusz Królicki

4 maja 2024

Spis treści

1 Zmiany kosmetyczne względem poprzedniej wersji projektu

Wraz z nową iteracją projektu przygotowane zostały dwie zmiany, które nie są znaczące dla przygotowywanych funkcjonalności, jednak zwiększają czytelność i ogólną estetykę kodu.

Po pierwsze, klasa **Obstacle** używa teraz dekoratora **dataclass** - od teraz służy ona jedynie do przechowywania danych i nie posiada własnych metod.

```
@dataclass
class Obstacle:
    """
    Attributes:
        x: x-coordinate of the obstacle's center
        y: y-coordinate of the obstacle's center
    """
    def __init__(self, x, y, radius, screen):
        self.position = (x, y)
        self.screen = screen
        self.radius = radius
```

Po drugie, do projektu został dodany moduł **pylint_runner**, który pomaga w dostosowywaniu kodu do standardu PEP 8.

```
def run_pylint_on_folder(folder_path):
    for root, _, files in os.walk(folder_path):
        for file in files:
            if file.endswith(".py"):
                file_path = os.path.join(root, file)
                print(f"Running pylint on: {file_path}")
                os.system(f"pylint {file_path}")
```

2 Nowe funkcjonalności

2.1 Sposób generowania przeszkód

Przeszkody są od teraz generowane losowo. Współrzędne są wybierane losowo przy użyciu rozkładu jednostajnego, z zakresu obejmującego rozdzielczość ekranu symulacji.

```
min_x = 0
max_x = 1366
min_y = 0
max_y = 768
random_positions = [
(random.randint(min_x, max_x),
random.randint(min_y, max_y))  for _ in range(10)
]
```

2.2 Kontrola symulacji

Symulacja może być teraz resetowana przy użyciu odpowiedniego klawisza - pomaga to przy testowaniu.

```
def check_keys(self):
    self.keys = pygame.key.get_pressed()
    if self.keys[pygame.K_r]:
        current_time = pygame.time.get_ticks()
        if current_time - self.reset_cooldown > self.last_reset_time:
            self.reset_simulation()
            self.last_reset_time = current_time


def reset_simulation(self):
    self.flock.clear()
    self.obstacles.clear()

    self.start()
```

Zmienne **reset_cooldown** i **last_reset_time** należą do klasy **Environment**. Pierwsza z nich określa, kiedy ostatnio wykonany był reset symulacji, druga to czas między kolejnymi możliwymi resetami (bazowo 1 sekunda). Zapobiega to sytuacji, w której jedno naciśnięcie klawisza wielokrotnie zresetuje symulację.

2.3 Interakcje między obiektami

Prace nad nową wersją projektu skupiły się na interkacji ptaków z przeszkodami. Z tego względu paramtetr **perception_radius** został rozdzielony na dwa. Jeden z nich określa zasięg widzenia innych ptaków, a drugi przeszkód.

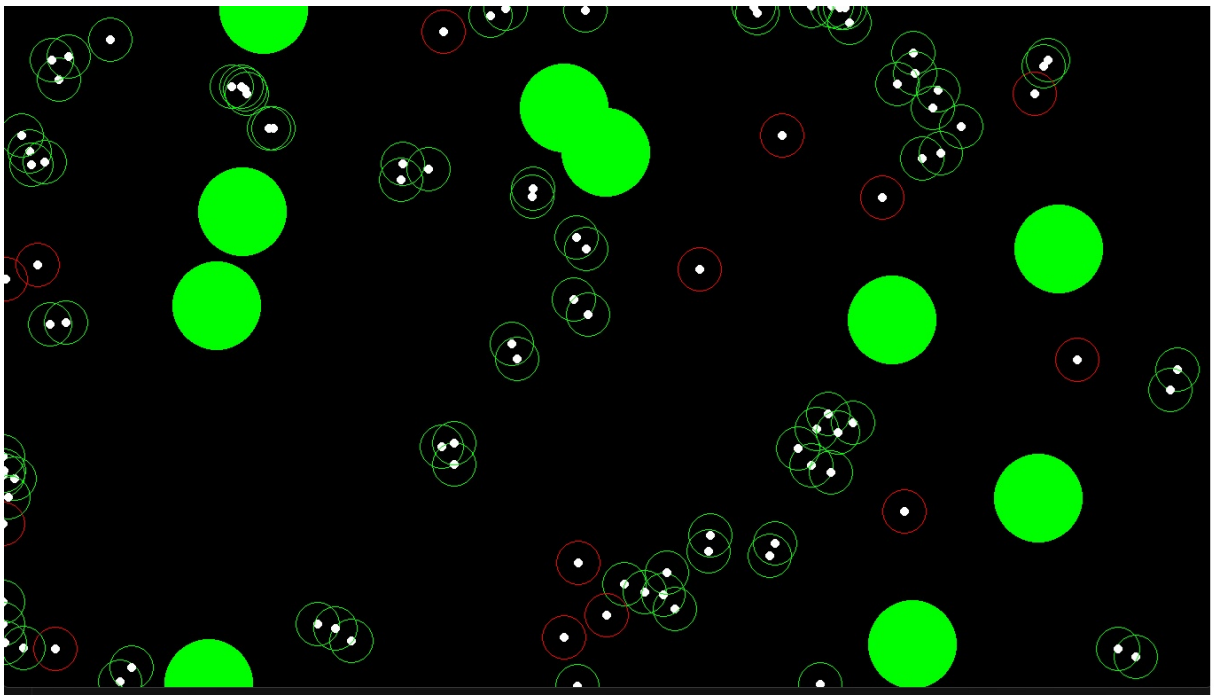
```
self.perception_radius_flock: int = perception_radius[0]
self.perception_radius_obstacle: int = perception_radius[1]
```

Perception radius jest teraz przekazywany do konstruktora klasy **Bird** jako lista.
Klasa **Bird** została rozszerzona o cztery nowe wartości:

1. **has_neighbors** - bool;
2. **approaching_obstacle** - bool;
3. **last_projection_time** - int;
4. **projection_cooldown** - int;

```
self.has_neighbors: bool = False
self.approaching_obstacle: bool = False
self.last_projection_time = 0
self.projection_cooldown: float = 0.1
```

Flaga **has_neighbors** służy do sygnalizowania, że w perception radius danego osobnika znajduje się inny. Ustawienie wartości na **True** powoduje zmianę koloru obrysu perception radius.



Rysunek 1: Osobniki z czerwonym okręgiem wokół nie posiadają sąsiadów.

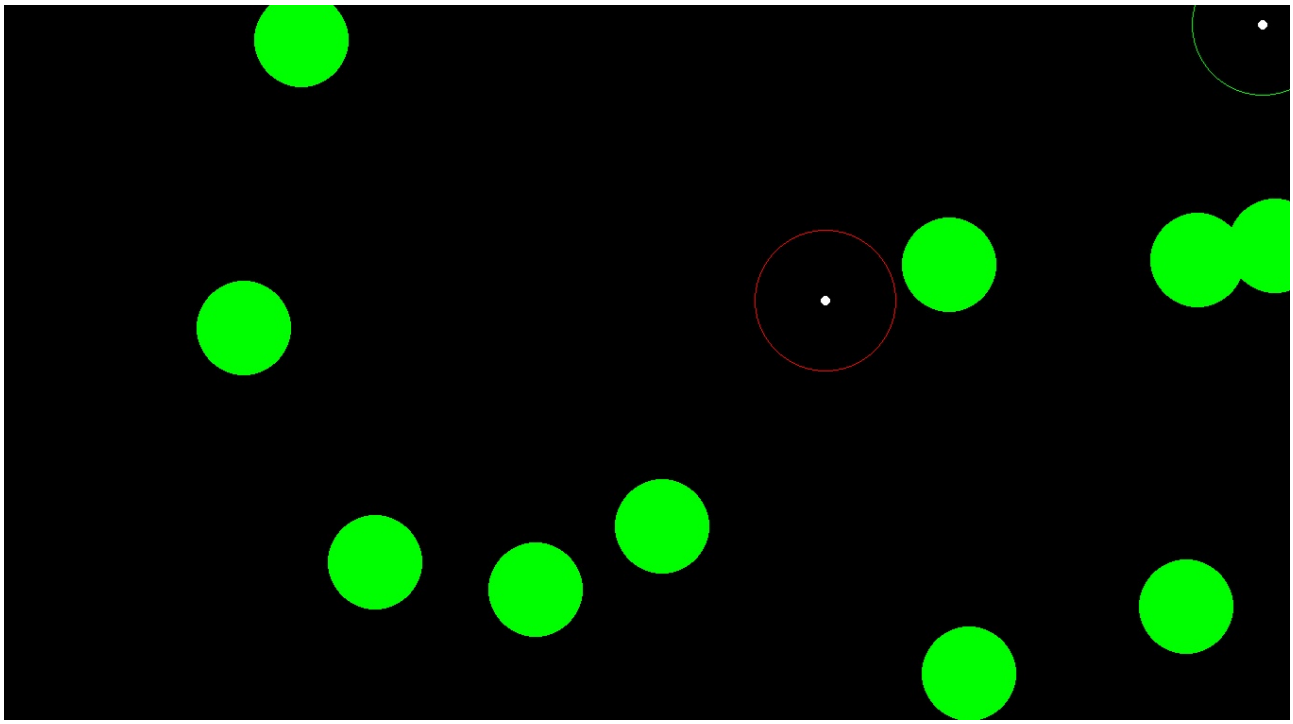
Pozostałe parametry są wykorzystywane przez nową metodę - **dodge_obstacles**.

```
def dodge_obstacles(self, obstacles: List[Obstacle]) -> None:
    """
    Adjust velocity vector if obstacle is within perception radius
    """
    current_time = time.time()

    if current_time - self.last_projection_time >= self.projection_cooldown:
        for obstacle in obstacles:
            dx = obstacle.position[0] - self.position[0]
            dy = obstacle.position[1] - self.position[1]
            distance_to_obstacle = math.sqrt(dx ** 2 + dy ** 2) - obstacle.radius
            if distance_to_obstacle <= self.perception_radius_obstacle:
                if not self.approaching_obstacle:
                    self.approaching_obstacle = True

                if self.approaching_obstacle:
                    to_obstacle = pygame.Vector2(dx, dy)
                    to_obstacle.normalize_ip()
                    projection = self.velocity.dot(to_obstacle)
                    if projection > 0:
                        self.velocity -= 2 * projection * to_obstacle
                        self.approaching_obstacle = False
                        self.last_projection_time = current_time
```

Rysunek 2: Metoda **dodge_obstacles**, Sposób formatowania kodu wymusił wstawienie screenshota.

Metoda sprawdza odległość między osobnikiem, a przeszkodami. Jeśli odległość od przeszkody

jest mniejsza niż odpowiedni perception radius, to obliczony zostanie **projection**, który wskaże kierunek ruchu osobnika względem przeszkody. Jeśli osobnik zmierza ku przeszkodzie, to wektor jego ruchu jest modyfikowany tak, aby uniknąć kolizji. Wyjątek stanowi sytuacji, w której od ostatniej transformacji wektora minęło za mało czasu - gdyby nie to zabezpieczenie, to na granicy perception radius osobnik zatrzymałbym się, ciągle korygując tor lotu. W przypadku zblizającej się kolizji, perception_radius_obstacle zmienia kolor z zielonego na czerwony.



Rysunek 3: Osobniki z czerwonym okręgiem wokół są niebezpiecznie blisko przeszkody.

3 Kolejne planowane etapy rozwoju projektu

Na myśli przychodzą dwie oczywiste modyfikacje. Po pierwsze można zastasować sposób obliczania odległości niewykorzystujący relatywnie kosztownego pierwiastkowania, Po drugie, można rozpatrzyć zmodyfikowanie funkcji odpowiedzialnej za ominajnie przeszkód, tak aby proces był bardziej płyny, dzięki czemu wyeliminowana zostanie potrzeba korzystania ze zmiennych **last_projection_time** i **projection_cooldown**.

Kolejne iteracje projektu można śledzić pod adresem:
`https://github.com/StaniszewskiA/SSIM-Proj`