Spike: 13
Title: Tactical Steering – Hiding Behvaiour in Hunter-Prey Simulation

Author: Marco Giacoppo, 104071453

Goals / deliverables:
The goal of this spike was to implement tactical hiding behavior in a 2D agent-based simulation. A prey agent should be able to detect a pursuing hunter and dynamically seek appropriate hiding spots using objects in the environment (simple circular obstacles). This spike includes:
- Custom Pyglet simulation in Python.
- Prey agent that selects hiding spots based on hunter location.
- Visual 'X' indicators for all hiding locations.
- Green marker on selected hiding spot.
- Reset feature to reshuffle the obstacles.
- Panic behaviour and active hunter pursuit logic.

Technologies, Tools, and Resources used: -
Python 3.11
- Pyglet (for game visualisation)
- Lab 12 code for foundation
- Custom steering implementations (seek, flee, pursuit, wander)

Tasks undertaken:

1. Project Initialization & Setup

- Reused Lab 12 code to confirm agent movement with wander() and seek().

- Set up a World object containing agents and obstacles.

2. Obstacle and Hiding Spot Logic

- Created a get_hiding_spot() function for obstacles to calculate positions behind them, relative to the hunter.

```
    ) -> Any

def get_hiding_spot(self, hunter_pos, offset=10):
    to_obj = self.pos - hunter_pos
    to_obj.normalise()
    return self.pos + to_obj * (self.radius + offset)
```

- Added obstacles to the world and ensured correct rendering.

```
class Game():
    def __init__(self):
        self.world = World(window.size[0], window.size[1])
        # add one agent
        self.world.obstacles = [
            Obstacle(self.world),
            Obstacle(self.world, x=430., y=120., radius=20.),
            Obstacle(self.world, x=580., y=450., radius=20.),
            Obstacle(self.world, x=510., y=590., radius=30.),
            Obstacle(self.world, x=760., y=870., radius=40.)]
        # unpause the world ready for movement
        self.world.paused = False
```

3. Tactical Hiding in Prey

- In Prey.calculate(), computed hiding spots for each obstacle.
- Selected the best spot based on distance from the hunter.
- If no valid hiding spot was available, the prey stood still.

```python
            self.vehicle.color = COLOUR_NAMES[self.color]

        else:
            self.mode = "hide"

            # Clear old markers
            for marker in self.hiding_spot_markers: ⋯
            self.hiding_spot_markers.clear()

            # Draw red X markers for all hiding spots
            for obstacle in self.world.obstacles: ⋯

            best_obstacle = self.best_obstacle_to_go()

            if best_obstacle:
                target_pos = best_obstacle.get_hiding_spot(self.world.hunter.pos, offset=10)

                # Add green X for the chosen spot
                green1 = pyglet.shapes.Line(target_pos.x - 7, target_pos.y - 7, target_pos.x + 7, target_pos.y + 7,
                                            color=(0, 255, 0), batch=window.get_batch("info"))
                green2 = pyglet.shapes.Line(target_pos.x - 7, target_pos.y + 7, target_pos.x + 7, target_pos.y - 7,
                                            color=(0, 255, 0), batch=window.get_batch("info"))
                self.hiding_spot_markers.extend([green1, green2])

                for obstacle in self.world.obstacles:
                    obstacle.target.color = COLOUR_NAMES['INVISIBLE']
                    obstacle.circle_emphasise.color = COLOUR_NAMES['INVISIBLE']

                best_obstacle.target.color = COLOUR_NAMES[self.color]
                best_obstacle.circle_emphasise.color = COLOUR_NAMES[self.color]
                best_obstacle.target.x = target_pos.x
                best_obstacle.target.y = target_pos.y
            else:
                target_pos = self.pos

        # Move toward the chosen spot
        accel = self.seek(target_pos)

        nearby_obstacle = self.nearby_obstacle()
        if nearby_obstacle:
            accel = self.flee(nearby_obstacle)

        self.accel = accel
        return accel or Vector2D(0, 0)
```

4. Visual Feedback

- Drew red "X" markers at every possible hiding spot (one per obstacle).
- Highlighted the chosen hiding spot with a larger green "X".
- Updated markers every frame for real-time feedback.

5. Hunter Pursuit Behavior

- Replaced wander() with a pursuit() method to chase the predicted future position of the prey.

```python
def pursuit(self, evader):
    # Calculate to target
    to_evader = evader.pos - self.pos
    relative_heading = self.heading.dot(evader.heading)

    # If evader is ahead and facing us, do a simple seek
    if to_evader.dot(self.heading) > 0 and relative_heading < -0.95:
        return self.seek(evader.pos)

    # Otherwise, predict future position
    look_ahead_time = to_evader.length() / (self.max_speed + evader.speed())
    future_pos = evader.pos + evader.vel * look_ahead_time
    return self.seek(future_pos)
```

- Adjusted max_speed to balance realism and challenge.

6. Reset Functionality

- Pressing R triggers a reset of obstacle positions using randrange() across the screen.
- Allows for randomized replay/testing scenarios.

What we found out:

- Tactical hiding works well. The prey evaluates all candidate obstacles and adapts its behavior depending on the hunter's proximity.
- Visualization improves comprehension. Seeing all hiding spots via red "X" and the selected spot with green "X" makes the logic intuitive.
- Pursuit adds realism. The hunter becomes a true threat, making the simulation more immersive and game-like.
- Fleeing behavior adds tension. The panic response when the hunter is close creates a dramatic and responsive moment in the simulation.
- Reset functionality is efficient. Pressing R to randomize obstacle positions avoids restarting and improves testing flow.
- Something was wrong with how the prey agent was moving. Thanks to OpenAI, turns out my math was wrong. Is_safe = distance < safe_distance was supposed to be Is_safe = distance > safe_distance.