CITS4403
Computational Modelling

Week 8 Lecture: Agent-based Modelling II

Dr. Siwen Luo

Semester 2, 2024
School of Computer Science,
University of Western Australia

0 LECTURE PLAN

THE UNIVERSITY OF WESTERN AUSTRALIA

W8 Lecture: Agent-based Modelling II

1. **Recap**
2. Traffic Jam
3. Boids Model

1 Recap…

Agent-based Model

Agent-based Model (ABM) is a computational model for simulating the actions and interactions of autonomous agents (both individual or collective entities such as organisations or groups) to understand the behaviour of a system and what governs its outcomes.

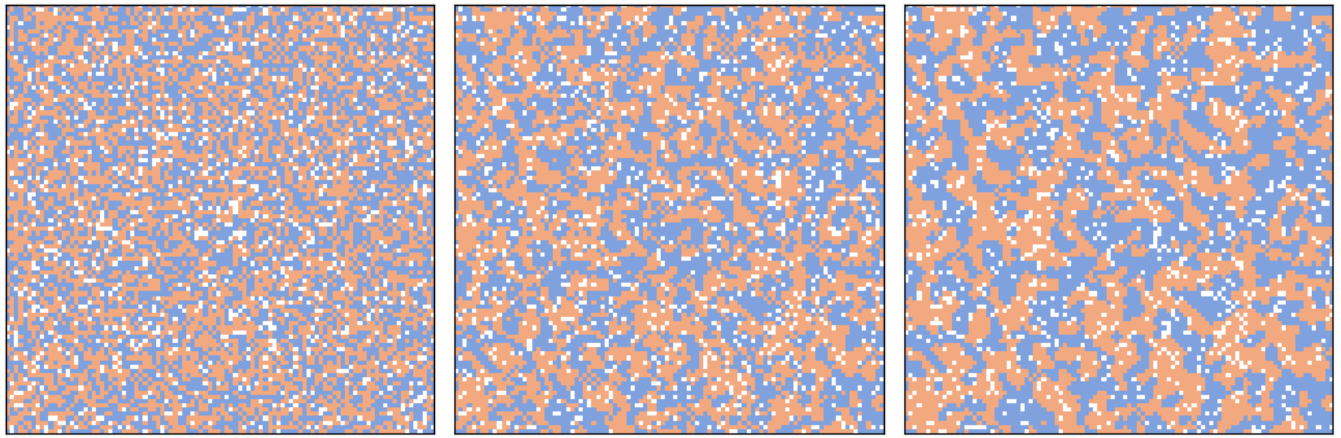
ABMs are stochastic models that include randomness.

Three elements in ABM:

- Agents
- Environment
- Interactions

THE UNIVERSITY OF WESTERN AUSTRALIA

Schelling's Model of Segregation

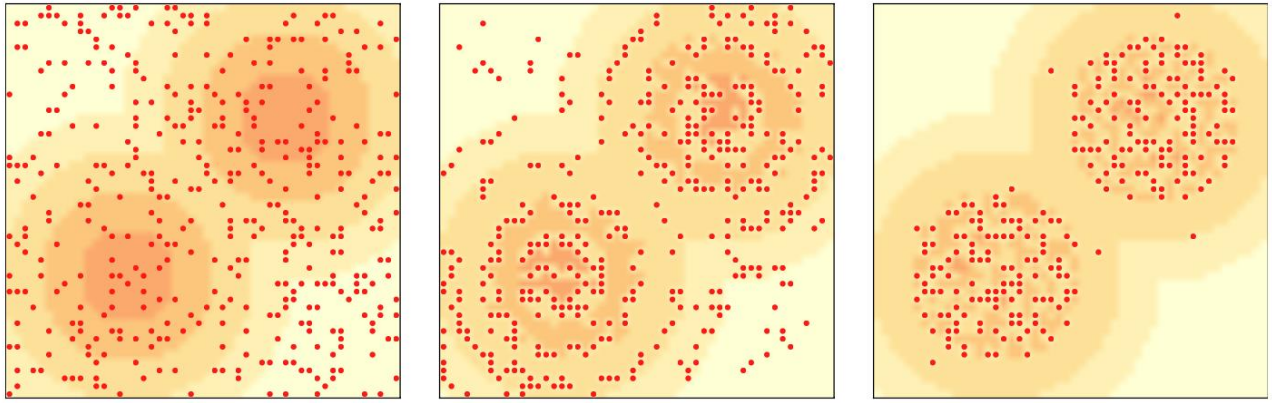


Initial configuration | After 2 steps | After 10 steps

Recap…

Sugarscape Model of Wealth Distribution

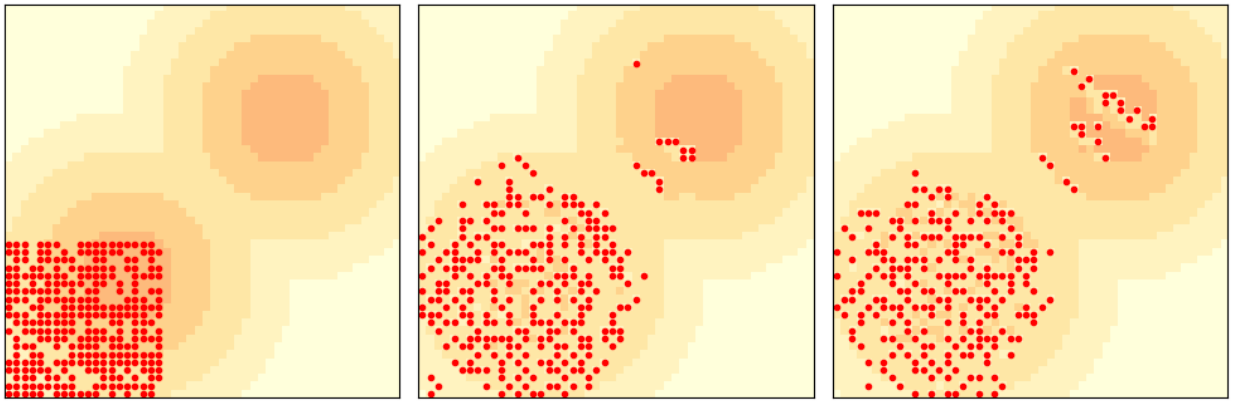
Agents move around on a 2-D grid, harvesting and accumulating “sugar”, which represents economic wealth. Some parts of the grid produce more sugar than others, and some agents are better at finding it than others.



Initial configuration | After 2 steps | After 10 steps

Sugarscape Model of Wealth Distribution

Agents move around on a 2-D grid, harvesting and accumulating “sugar”, which represents economic wealth. Some parts of the grid produce more sugar than others, and some agents are better at finding it than others.



Initial configuration | After 6 steps | After 12 steps

Different initial configuration results in different outcome and findings.

ABM in Continuous Space

In Schelling's Model of Segregation and Sugarscape model of wealth distribution:

- agents occupy **discrete** locations in space.

ABMs are not limited to discrete space. The space can also be continuous.

ABM in Continuous Space

In Schelling's Model of Segregation and Sugarscape model of wealth distribution:

- agents occupy **discrete** locations in space.

ABMs are not limited to discrete space. The space can also be continuous.

How can we alter the Schelling's model to continuous space?

Houses in discrete cells → Agents might build a house where sufficient space exists

What about Sugarscape model?

Compass direction of vision and movement → Radial in any direction

Traffic jams

Build an agent-based model for highway simulation.

- **Space:**
  - 1-D if we restrict the motion to a single lane.
  - Possible world properties: distances between cars, speed limit for cars.

- **Agents**:
  - Drivers in the cars
  - Possible agent properties: acceleration rate (decision made by driver)

Traffic jams

What pieces of machinery do we need?

- Initialize 1D array with equal distance between cars
- Determine the speed limit on this highway
- Determine the next car for each car
- Setup the initial speed for each car
- Cars will move in order on this highway
- For the movement of each car:
  - Determine the distance that each car can move
  - Choose the acceleration rate for each driver
  - Compute the speed of each car (with randomness involved)
  - Update the new location for each car
- If speed > distance from next car, crash happens, and the car will stop with speed set to zero

Traffic jams

Randomness in this model:

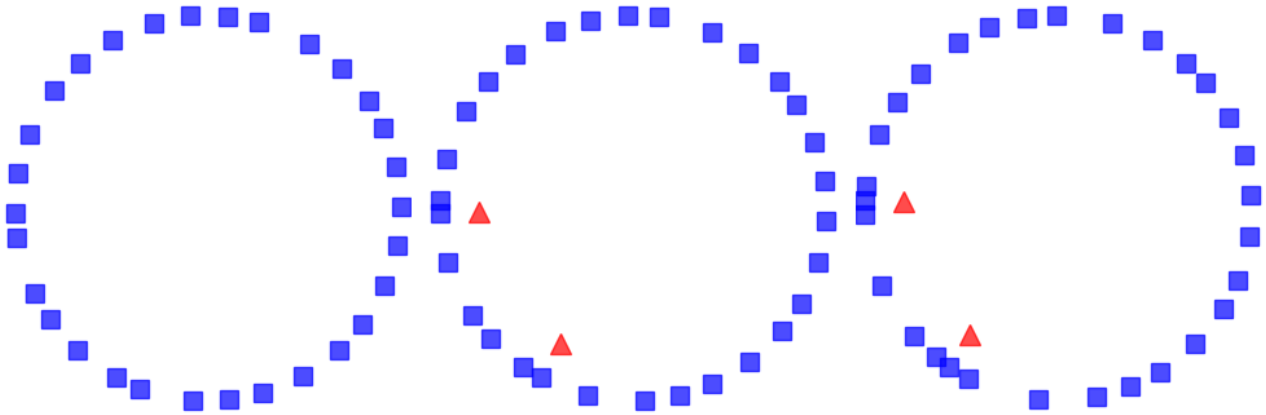
```
# add random noise to speed
speed *= np.random.uniform(1-self.eps, 1+self.eps)
```

A random noise is added to the speed computation at each step of driver's movement to reflect the imperfect speed up in real world.

With this random noise, even if every driver's acceleration rate is the same, the eventual speed would be different, and crash might occur, causing some cars to stop.

Traffic jams

Simulation outcome with 30 cars and random noise is 0.02.

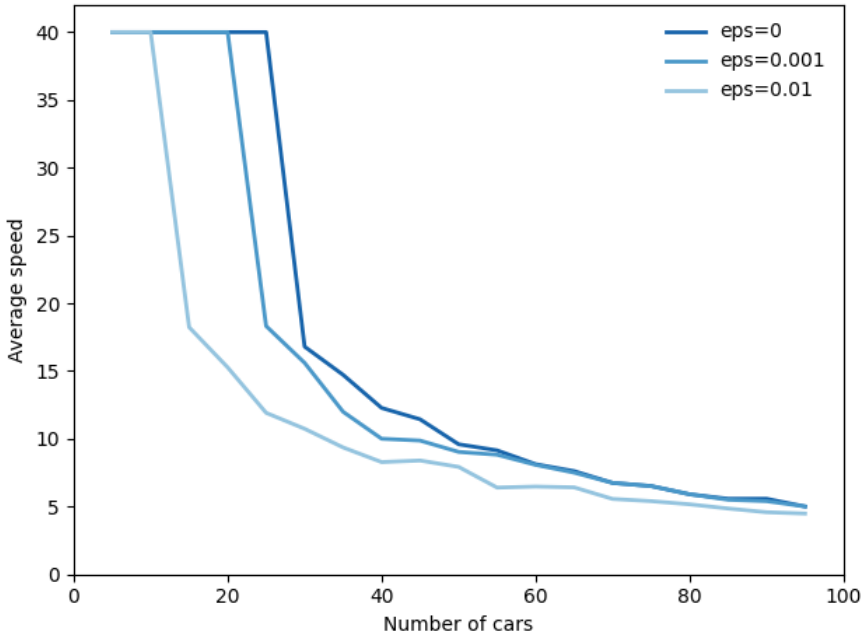


After 16 steps After 17 steps After 18 steps

Because of the added random noise, some cars are going faster than others, and the spacing has become uneven.

2 Traffic iams

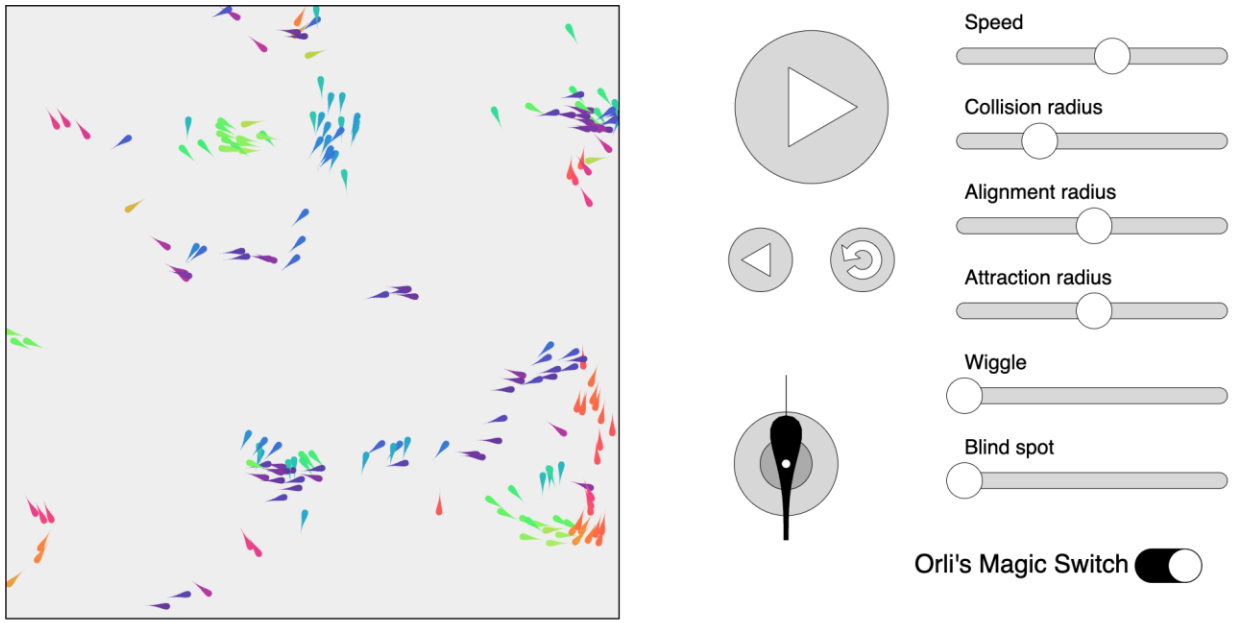
To quantify the effect of driving behavior:



Calculate average speed of cars for different cars and different noise value in simulation.

Boids: Flock of Birds

Collective behavior and swarming



based on Complexity Explorable:

*" **Flock'n Roll -** Collective Behavior - Schooling Fish & Flocking Birds"*

https://www.complexity-explorables.org/slides/flockn-roll/

Boids: Flock of Birds

Build an agent-based model for herd behavior:

Agents in this model are called “Boids”, which is both a contraction of “birdoid” and an accented pronunciation of “bird” (although Boids are also used to model fish and herding land animals).

Each agent has three behaviors:

- **Flock centering:** move toward the center of the flocks.
- **Collision avoidance:** Avoid obstacles, including other boids.
- **Velocity matching:** align velocity (speed and direction) with neighboring boids.

Boids: Flock of Birds

What pieces of machinery do we need?

- Initialize number of boids with random velocity and random positions.
- Boids movement rules:
  - **Find center**: Finds other Boids within range (radius and angle of the field of view) and computes a vector toward their centroid.
  - **Avoid collision**: Finds objects, including other Boids (and carrots), within a given range, and computes a vector that points away from their centroid.
  - **Align velocity**: finds other Boids within range and computes the average of their headings.
  - **Love (find carrot)**: computes a vector that points toward the carrot.

Boids: Flock of Birds

Only boids within the range (in the neighbourhood) will be considered for the movement.

```
def get_neighbors(self, boids, radius, angle):
    """Return a list of neighbors within a field of view.

    boids: list of boids
    radius: field of view radius
    angle: field of view angle in radians

    returns: list of Boid
    """
    neighbors = []
    for boid in boids:
        if boid is self:
            continue
        offset = boid.pos - self.pos

        # if not in range, skip it
        if offset.mag > radius:
            continue

        # if not within viewing angle, skip it
        diff = self.vel.diff_angle(offset)
        if abs(diff) > angle:
            continue

        # otherwise add it to the list
        neighbors.append(boid)

    return neighbors
```

Boids: Flock of Birds

For each movement rule: find center, avoid collision, align velocity, and love (find carrot), compute the vector pointing to the new direction for each Boid in the movement.

```
def center(self, boids, radius=1, angle=1):
    """Find the center of mass of other boids in range and
    return a vector pointing toward it."""
    neighbors = self.get_neighbors(boids, radius, angle)
    vecs = [boid.pos for boid in neighbors]
    return self.vector_toward_center(vecs)
```

```
def avoid(self, boids, carrot, radius=0.3, angle=np.pi):
    """Find the center of mass of all objects in range and
    return a vector in the opposite direction, with magnitude
    proportional to the inverse of the distance (up to a limit)."""
    objects = boids + [carrot]
    neighbors = self.get_neighbors(objects, radius, angle)
    vecs = [boid.pos for boid in neighbors]
    return -self.vector_toward_center(vecs)
```

Boids: Flock of Birds

For each movement rule: find center, avoid collision, align velocity, and love (find carrot), compute the vector pointing to the new direction for each Boid in the movement.

```
def align(self, boids, radius=0.5, angle=1):
    """Return the average heading of other boids in range.

    boids: list of Boids
    """
    neighbors = self.get_neighbors(boids, radius, angle)
    vecs = [boid.vel for boid in neighbors]
    return self.vector_toward_center(vecs)
```

```
def love(self, carrot):
    """Returns a vector pointing toward the carrot."""
    toward = carrot.pos - self.pos
    return limit_vector(toward)
```

Boids: Flock of Birds

A weighted sum is computed for each movement rule, so different factors have different impacts on the final movement decision.

```
def set_goal(self, boids, carrot):
    """Sets the goal to be the weighted sum of the goal vectors."""

    # weights for various rules
    w_avoid = 10
    w_center = 3
    w_align = 1
    w_love = 10

    self.goal = (w_center * self.center(boids) +
                 w_avoid * self.avoid(boids, carrot) +
                 w_align * self.align(boids) +
                 w_love * self.love(carrot))
    self.goal.mag = 1
```

Boids: Flock of Birds

A weighted sum is computed for each movement rule, so different factors have different impacts on the final movement decision.

```
def set_goal(self, boids, carrot):
    """Sets the goal to be the weighted sum of the goal vectors."""

    # weights for various rules
    w_avoid = 10
    w_center = 3
    w_align = 1
    w_love = 10

    self.goal = (w_center * self.center(boids) +
                 w_avoid * self.avoid(boids, carrot) +
                 w_align * self.align(boids) +
                 w_love * self.love(carrot))
    self.goal.mag = 1
```

Boids: Flock of Birds

Update the final velocity and position to complete the movement for every Boid.

```
def move(self, mu=0.1, dt=0.1):
    """Update the velocity, position and axis vectors.

    mu: how fast the boids can turn (maneuverability).
    dt: time step
    """

    self.vel = (1-mu) * self.vel + mu * self.goal
    self.vel.mag = 1
    self.pos += dt * self.vel
    self.axis = self.length * self.vel
```

The parameter *mu* determines how quickly the birds can change speed and direction.

Boids: Flock of Birds

Many parameters influence flock behavior, including the **radius, angle** and **weight for each behavior**, as well as **maneuverability**, *mu*. These parameters determine the ability of the Boids to form and maintain a flock, and the patterns of motion and organization within the flock.

With different settings, the movement can resemble a flock of birds, school of fish or cloud of flying insects.

Preview…

W10 Lecture: Evolution

Simulation the theory of evolution by natural selection also by agent-based modelling.