MATH089 Project 7 - Agent-based models: flocking, schooling

Posted: 11/22/21

Due: 12/01/21, 11:55PM

1 Introduction

1.1 Environment setup

- Add the Julia `Agents` package.
- Add the Julia `InteractiveDynamics` package
- Add the Julia `CairoMakie` package

2 Methods

2.1 Game of Life

2.1.1 Loading the pre-defined agent model

Conway's game of Life is a predefined model within the `Agents` package.

```
∴ using Agents
```

```
∴
```

It can be loaded through the following instruction

```
∴ model, agent_step!, mode_step! =
    Models.game_of_life(; rules = (2, 3, 3, 3), dims = (100, 100),
                          metric = :chebyshev)
```

```
(AgentBasedModel with 10000 agents of type Cell
 space: GridSpace with size (100, 100), metric=chebyshev, periodic=true
 scheduler: fastest
 properties: Dict(:rules => (2, 3, 3, 3)), Agents.dummystep,
Agents.Models.game_of_life_model_step!)
```

```
∴
```

A random initial state is defined through

```
∴ for i in 1:nagents(model)
      if rand(model.rng) < 0.2
          model.agents[i].status = true
      end
  end
```

```
∴
```

Agent-based models are of special relevance to time-varying phenomena, and results are best presented as animations instead of static plots. The `InteractiveDynamics` and `CairoMakie` package has support for such visualizations.

```
∴ using InteractiveDynamics
```

```
∴ import CairoMakie
```

```
∴
```

Define functions to visualize the state of a cell

```
∴ ac(x) = x.status == true ? :black : :white
```

```
ac
```

```
∴ am(x) = x.status == true ? '■' : '□'
```

```
am
```

```
∴
```

Invoking the `abm_video` function executes successive steps of the Game of Life and saves the states to a video file

```
∴ abm_video(
      "/home/student/courses/MATH089/GameOfLife.mp4",
      model,
      dummystep,
      mode_step!;
      title = "Game␣of␣Life",
      ac,
      as = 12,
      am,
      framerate = 5,
      scatterkwargs = (strokewidth = 0,),
  )
```

```
∴
```

2.1.2 Step-by-step definition of the model

Define the Game of Life rules and agent structure

```
∴ rules = (2, 3, 3, 3) # (D, S, R, O);
```

```
∴ mutable struct Cell <: AbstractAgent
      id::Int
      pos::Dims{2}
      status::Bool
  end
```

```
∴
```

Define a function to build the model

```
∴ function build_model(; rules::Tuple, dims = (50, 50), metric =
  :chebyshev, seed = 120)
      space = GridSpace(dims; metric)
      properties = Dict(:rules => rules)
      model = ABM(Cell, space; properties)
      idx = 1
      for x in 1:dims[1]
          for y in 1:dims[2]
              add_agent_pos!(Cell(idx, (x, y), false), model)
              idx += 1
          end
      end
      return model
  end;
```

```
∴ model=build_model(; rules)
```

```
AgentBasedModel with 2500 agents of type Cell
 space: GridSpace with size (50, 50), metric=chebyshev, periodic=true
 scheduler: fastest
 properties: Dict(:rules => (2, 3, 3, 3))
```

Define the initial state

```
∴ for i in 1:nagents(model)
      if rand(model.rng) < 0.2
          model.agents[i].status = true
      end
  end
```

```
∴
```

Run the model

```
∴ abm_video(
      "/home/student/courses/MATH089/SmallGameOfLife.mp4",
      model,
      dummystep,
      mode_step!;
      title = "Game of Life",
      ac,
      as = 12,
      am,
      framerate = 5,
      scatterkwargs = (strokewidth = 0,),
  )
```

```
∴
```

3 Results

4 Discussion