Agent-based modelling of complex systems

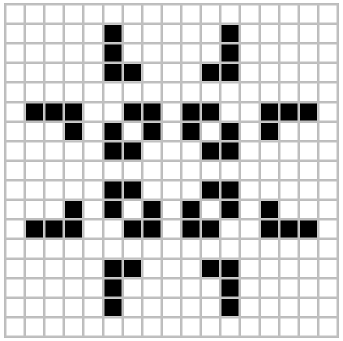
Janusz Szwabiński

Lecture #3: Simple ABMs

Outlook

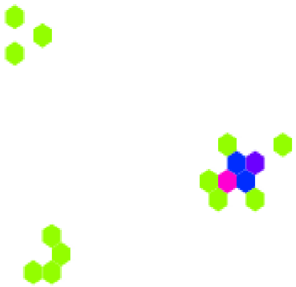
- Game of Life
- Heroes and Cowards
- Simple economy

Game of Life



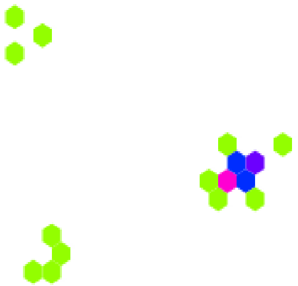
- proposed by the British mathematician John Horton Conway (1970)
- toy model (thought experiment)
- motivated by von Neumann’s problem of finding a hypothetical machine that had the ability to create copies of itself
- Conway’s model **simplifies** von Neumann’s ideas
- a 2D **cellular automaton**
- a **zero-player game** (its evolution determined by its initial state, no further input required)
- **emergence and self-organization**

Digression – a cellular automaton



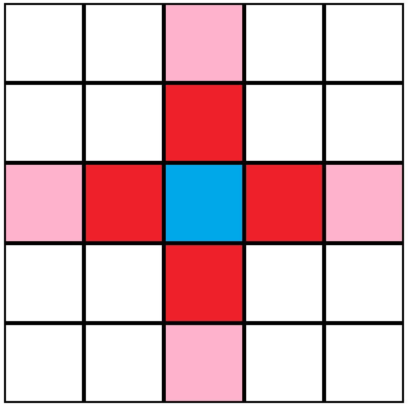
- discovered by S. Ulam and J. von Neumann in 1940
- got popularized by Conway’s Game of Life (1970) and the work of Wolfram (1980s)
- a discrete model studied in computability theory, mathematics, physics, complexity science, theoretical biology and microstructure modeling
- a regular **grid of cells**, each in **one of a finite number of states**
- any **finite** number of dimensions

Digression – a cellular automaton

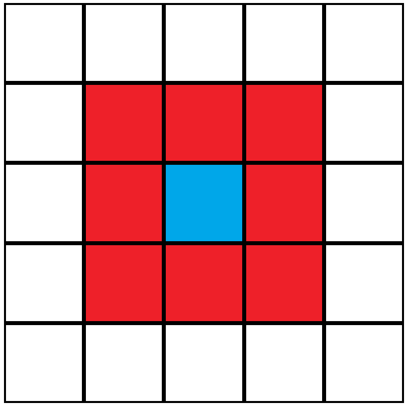


- for each cell, a set of cells called its **neighborhood** is defined
- an initial state (time t = 0) is selected by assigning a **state for each cell**
- a new generation is created (advancing t by 1) **according to some fixed rule** (a mathematical function) that determines the new state of each cell in terms of the current states of the cell and its neighborhood
- the rule is usually **the same** for each cell and **constant** in time
- the rule is applied to the whole grid **simultaneously**

Digression – a cellular automaton



Von Neumann neighborhood



Moore neighborhood

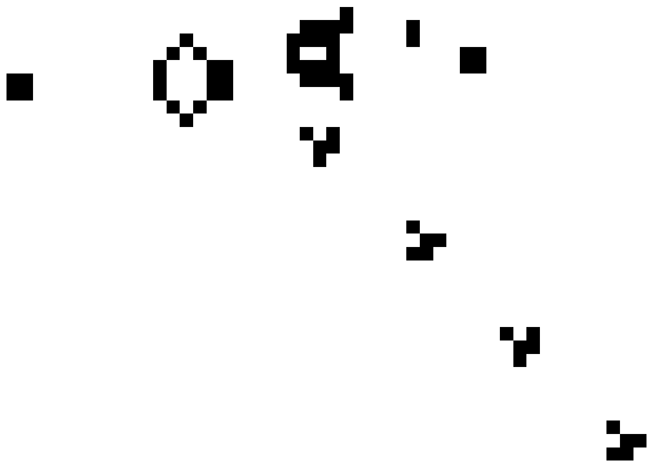
Digression – a cellular automaton

- applications:
  - biology → patterns of some seashells (Conus, Cymbiola) are generated by natural cellular automata
  - chemistry → Belousov-Zhabotinsky reaction
  - cryptography → random number generation, one-way function in public key cryptography
  - complex systems → modeling emergence and self-organization
  - IT → building error correction codes

Game of Life

- it has the power of **universal Turing machine** → anything that can be computed algorithmically can be computed within Game of Life
- new field of mathematical research → **simulation games**
- applications in computer science, physics, biology, biochemistry, economics, mathematics, philosophy and generative sciences

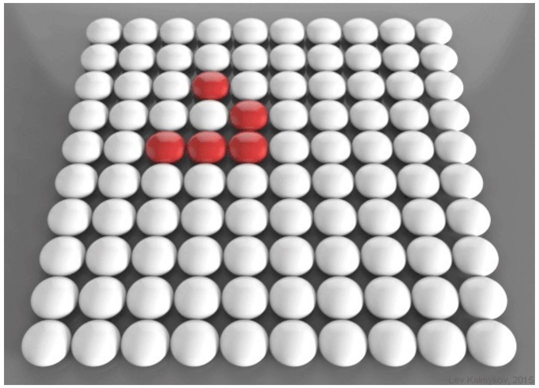
Game of Life



Conway’s criteria for the rules:

1. There should be no explosive growth
2. There should exist small initial patterns with chaotic, unpredictable outcomes
3. There should be potential for von Neumann universal constructors
4. The rules should be as simple as possible, whilst adhering to the above constraints

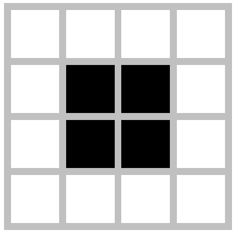
Game of Life



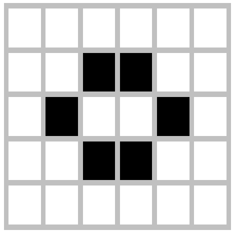
- universe – an infinite 2D square grid
- two possible states – dead/alive (or populated/unpopulated)
- Moore neighborhood
- rules:
  1. Any live cell with fewer than two live neighbours dies, as if caused by **underpopulation**
  2. Any live cell with two or three live neighbours lives on to the next generation
  3. Any live cell with more than three live neighbours dies, as if by **overpopulation**
  4. Any dead cell with exactly three live neighbours becomes a live cell, as if by **reproduction**

Game of Life

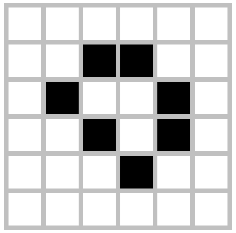
Still life - stable patterns that do not change in time



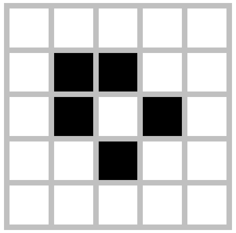
Block



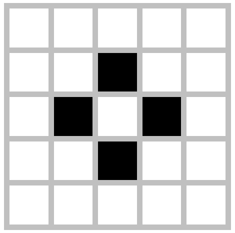
Beehive



Loaf



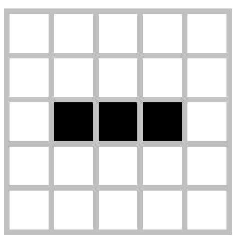
Boat



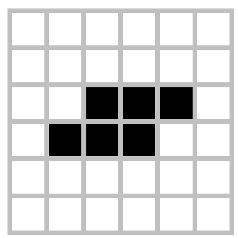
Tube

Game of Life

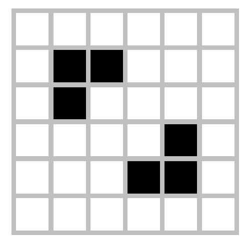
Oscillators – these patterns change over a specific number of ticks



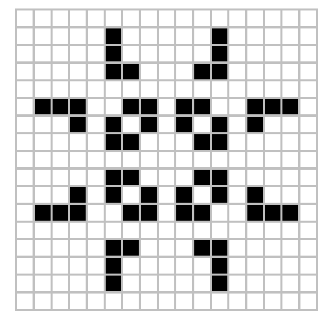
Blinker (period 2)



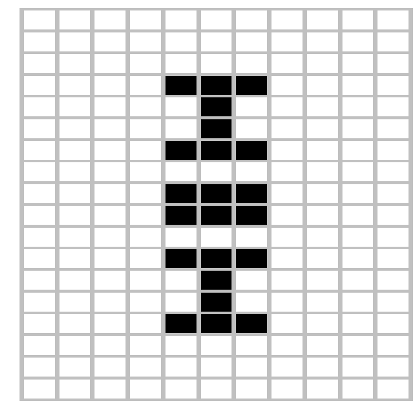
Toad (2)



Beacon (2)



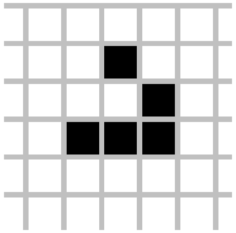
Pulsar (3)



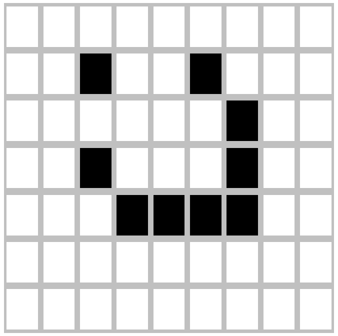
Pentadecathlon (15)

Game of Life

Gliders and spaceships – patterns that move, returning to the same configuration but shifted after a finite number of generations



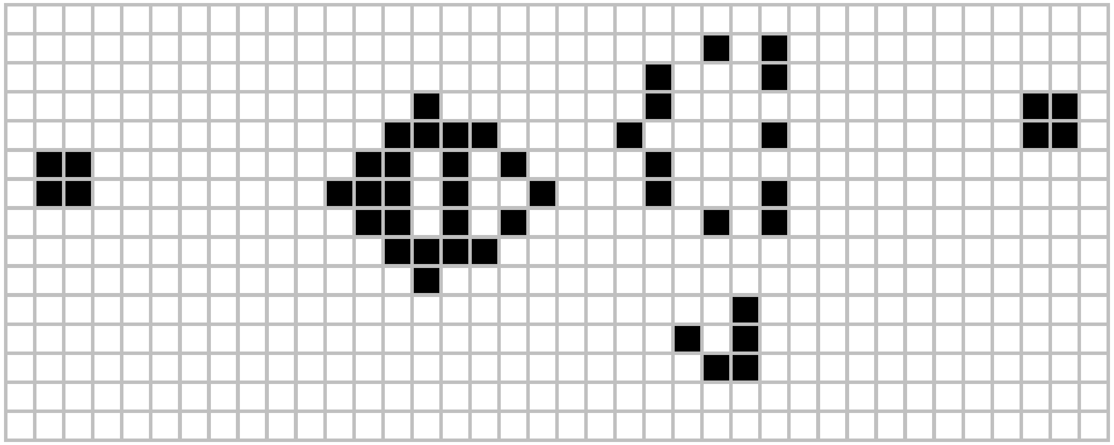
Glider



Spaceship

Game of Life

Guns – repeating patterns which produce a spaceship after a finite number of generations



Gosper glider gun

Game of Life

Puffers – moving patterns, their creation leaves a stable or oscillating debris behind at regular intervals.

Puffer train

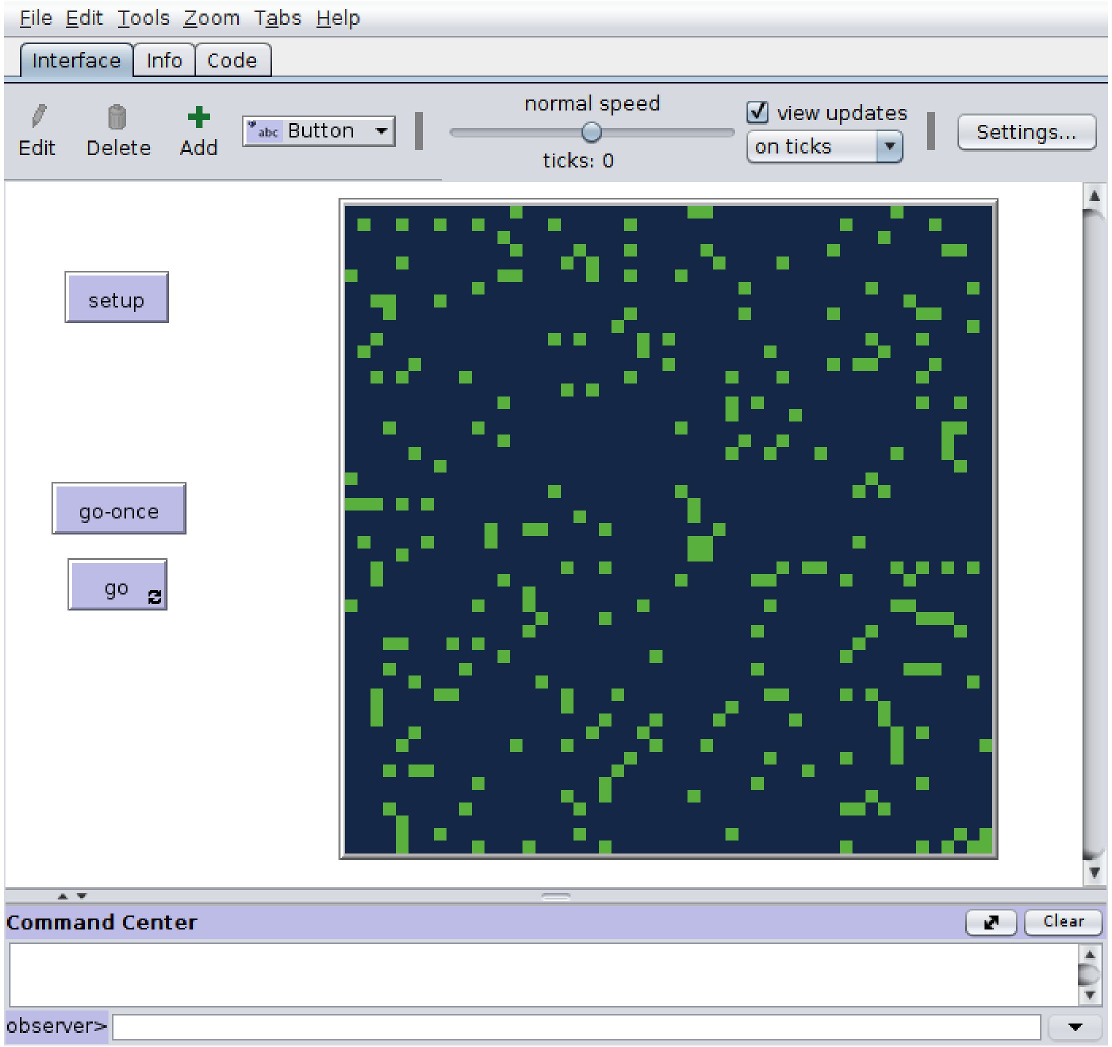
Game of Life

Rakes – moving patterns that emit spaceships at regular intervals as they move.

Game of Life

Breeder – oscillating patterns which leave behind guns at regular intervals. Unlike guns, puffers, and rakes, each with a linear growth rate, breeders have a quadratic growth rate

Game of Life

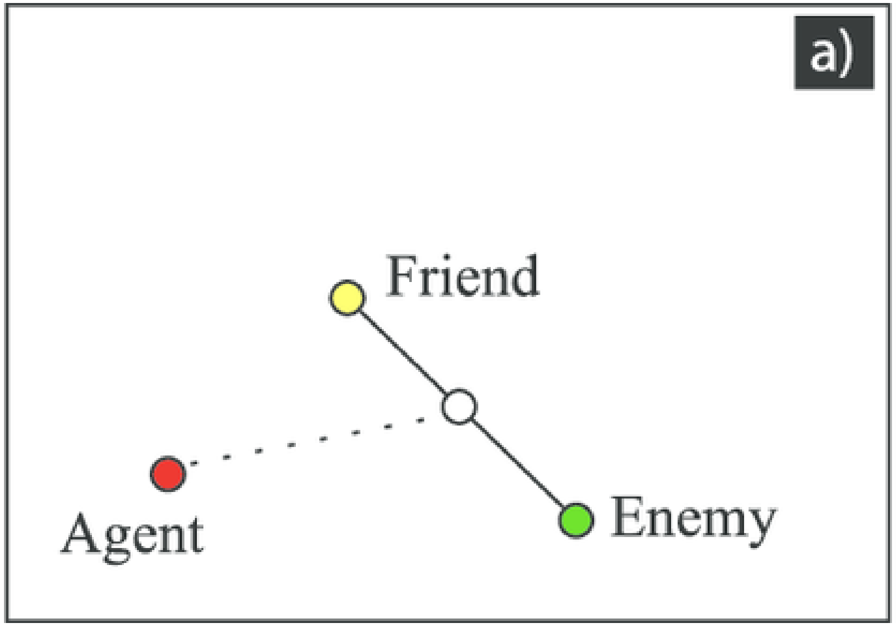


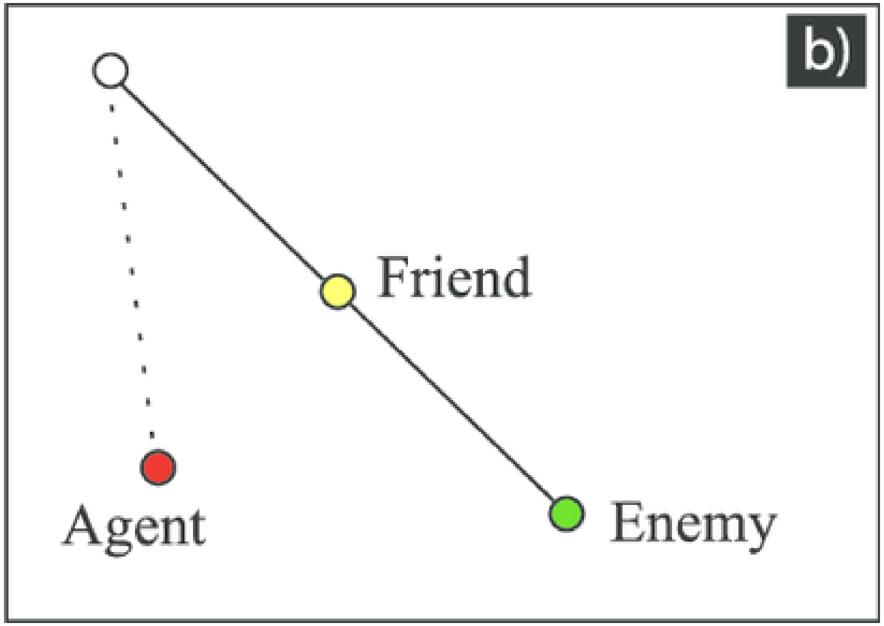
File Edit Tools Zoom Tabs Help
Interface Info Code
Edit Delete Add
Button
normal speed
ticks: 0
view updates
on ticks
Settings...
setup
go-once
go
Command Center
Clear
observer>

Heroes and Cowards

- also called “Friends and Enemies” or “Aggressors and Defenders” game
- it dates back to the Fratelli Theater Group at the 1999 Embracing Complexity conference
- in the human version of this game, each person arbitrarily chooses someone else in the room to be their perceived friend, and someone to be their perceived enemy. They don't tell anyone who they have chosen, but they all move to position themselves either such that
  a) they are between their friend and their enemy
     (BRAVE/DEFENDING), or
  b) such that they are behind their friend relative to their enemy
     (COWARDLY/FLEEING)

Heroes and Cowards

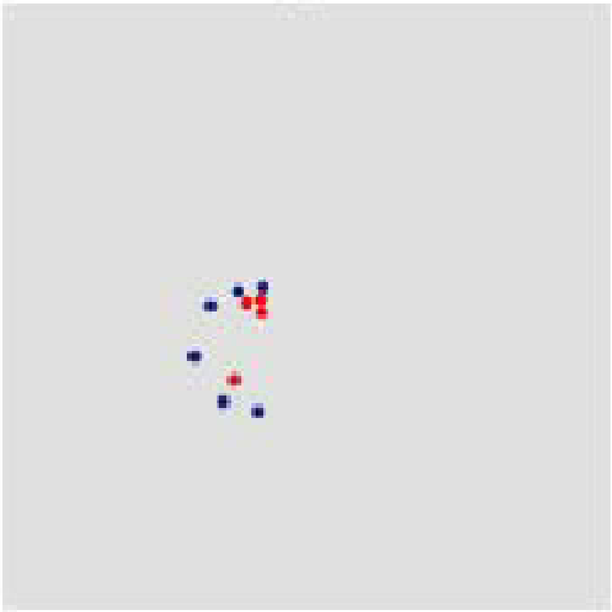




defending

fleeing

Heroes and Cowards

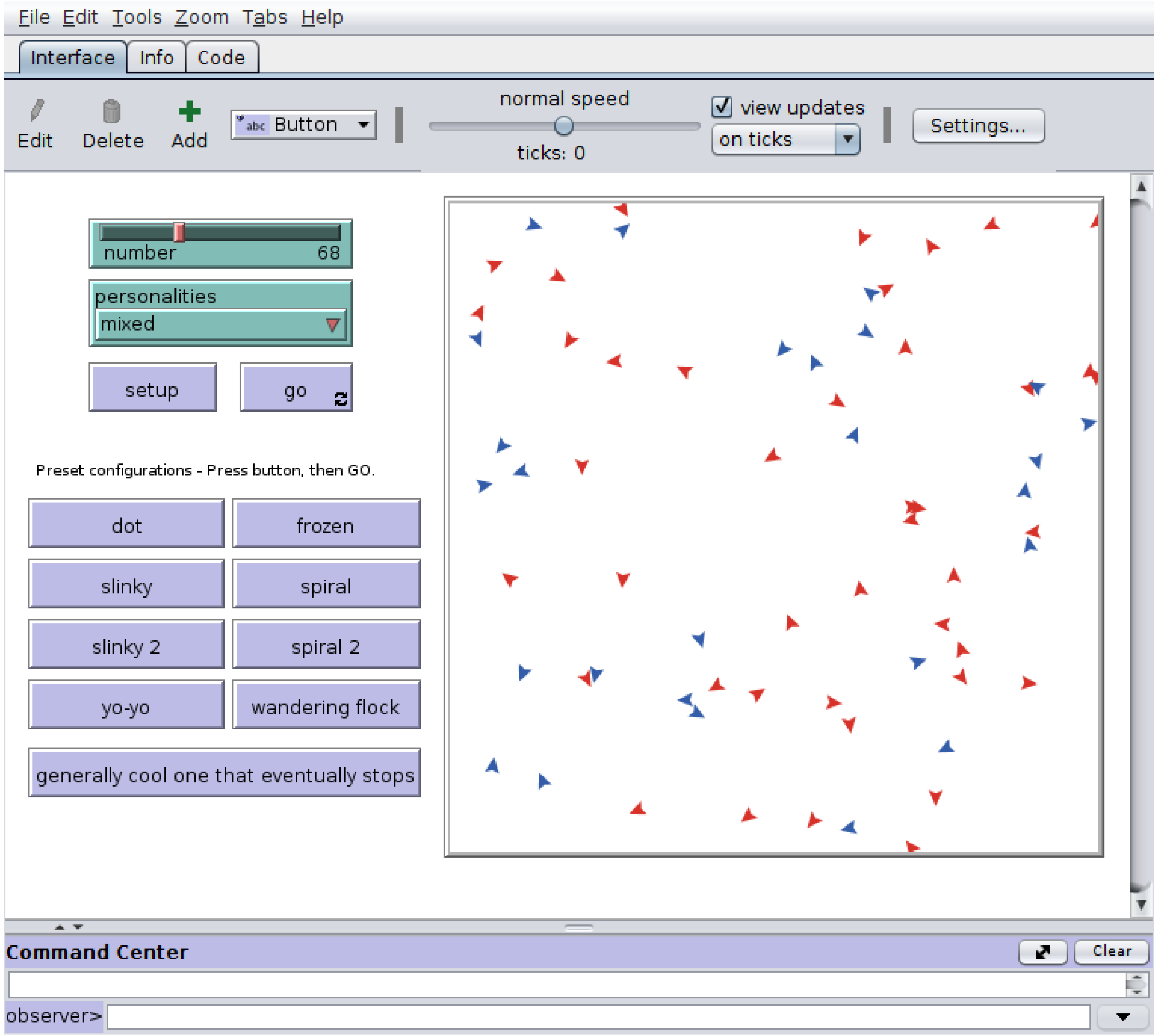


Heroes and Cowards

- universe – a plane
- agents – people being in one of two states: brave or cowardly
- initial state:
  - all brave
  - all cowards
  - mixed population
  - random positions
- rules:
  - if brave, move toward the midpoint of your friend and enemy
  - if a coward, put your friend between you and your enemy
- time evolution – in every tick check the state of the agent and act accordingly

Can you predict what will happen?

Heroes and Cowards



Heroes and Cowards

- use pseudocode for an explicit model description!:

```
Initialize:
    Create NUMBER agents
    Move each agent to a random location
    If “hero” personality chosen, each agent turns blue
    If “coward” personality chosen, each agent turns red
    If “mixed” personality chosen, color each agent red or blue at random
    Each agent picks one other agent as friend
    Each agent picks one other agent as enemy
    Start the clock

At each tick:
    Each blue agent moves a step towards a location between his
    friend and its enemy
    Each red agent moves a step towards a location that puts his friend
    between him and his enemy
```

Digression – random number generators

- agent-based models often need to make use of **randomness** (agents’ behavior is often best modeled as a random process)
- most of the random number generators available for programmers are actually **pseudo-random**
- while the numbers appear random, they are generated deterministically
  - starting with the same seed will always generate the same result
- true random numbers can be based on an essentially random physical phenomenon
  - sources of entropy: radioactive decay, thermal noise, shot noise, avalanche noise in Zener diodes, clock drift, the timing of actual movements of a hard disk read/write head, and radio noise
- modern RNGs are almost indistinguishable from truly random numbers
- they are more desirable in scientific modeling because of reproducibility
- usually, if the seed of RNG is not explicitly given, it is based on the current date and time (there is no way to figure out the seed and repeat the calculations)

Digression – RNGs in Python

- **Mersenne Twister** as the core generator
  - 53-bit precision floats
  - a period of 2^{19937}-1
  - implemented in C
  - fast and threadsafe
  - one of the most extensively tested RNGs
  - completely deterministic → not suitable for all purposes (do not use it in cryptography)
- **`os.urandom()`** (may be accessed via random.SystemRandom)
  - generates numbers from sources provided by the operating system
  - suitable for cryptographic use (however, its quality depends on the OS implementation)

Simple economy

- economy is one of the areas receiving increasing attention from ABM community
- there is a natural mapping between ABM methods and economics, because the latter consists of heterogeneous actors (e.g. buyers and sellers)
- **SugarScape** – one of the most famous models for artificially intelligent agent-based social simulation proposed by Joshua M. Epstein & Robert Axtell (1996)
  - a world populated by economic agents characterized by a limited vision
  - spatially distributed resource available (sugar and spice)
  - agents look around, find sugar, move, metabolize, leave pollution etc
  - each version of the model explores some of the conditions and dynamics
  - ABM turned out to be well suited as a methodology for behavior-based economics

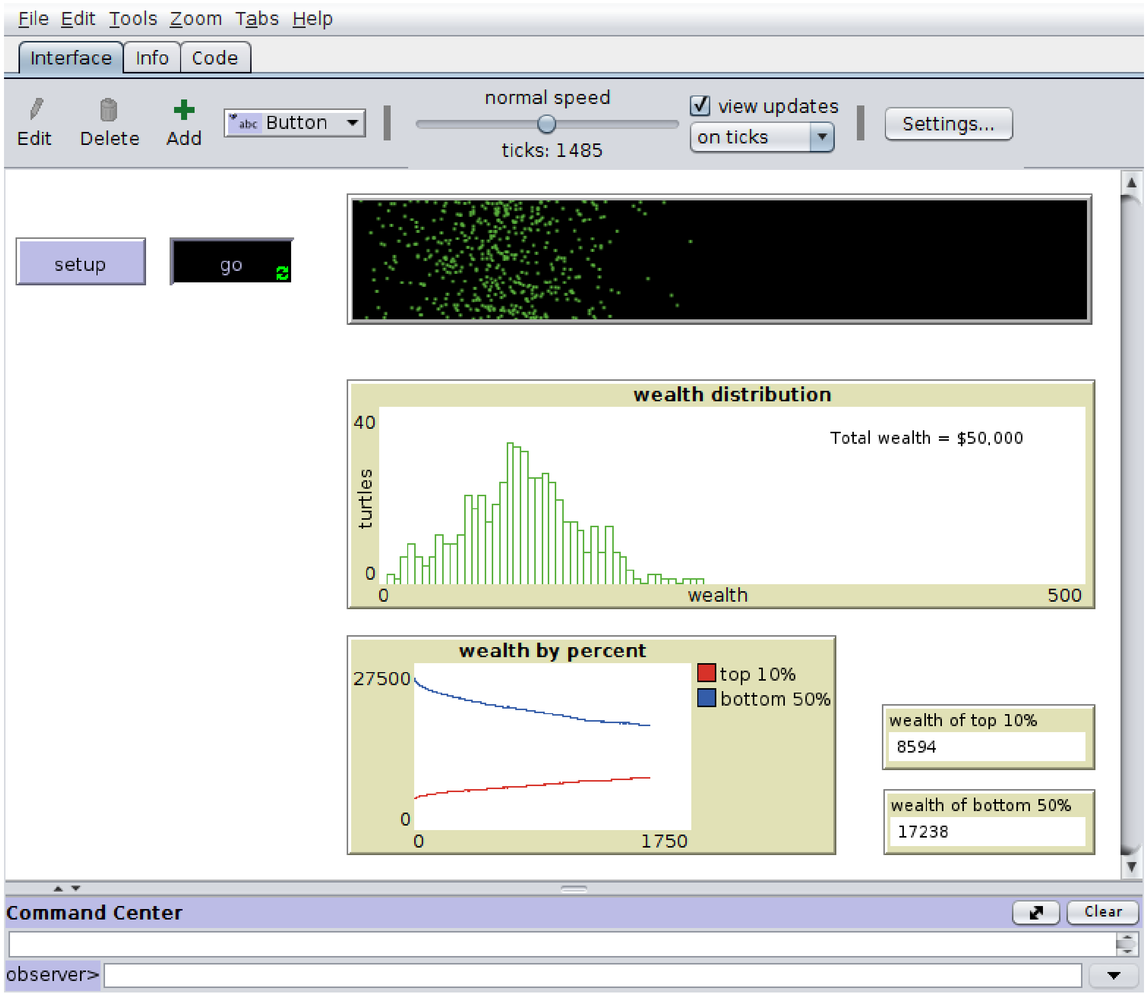
Simple economy

- consider a fixed number of people, e.g. 500
- each of them starts with the same amount of money, e.g. $100
- at every tick, each person gives one of his dollars to any other person at random
- total amount of money is fixed → no one can have less than zero money

What will happen to the distribution of money?

Is there a stable limiting distribution of the money?

Simple economy



Simple economy

- most people have an intuition that the distribution will stay relatively flat
- however, it is not flat!
  - it has been shown that the distribution converges to an **exponential** one
  - there is a great inequality in monetary wealth!
  - the key condition is the conservation of money
  - it is an instance of the Boltzmann-Gibbs law from statistical mechanics
- ABM can be seen as extending the perspective of statistical physics (from microscopic particles to macroscopic patterns) to domains beyond physics