Golang UA, Kyiv, May 18 2018

Go as an Agent-Based Model simulation tool

Ivan Danyliuk

Status.im

@idanyliuk

What is ABM?

Agent Based Modelling (ABM)

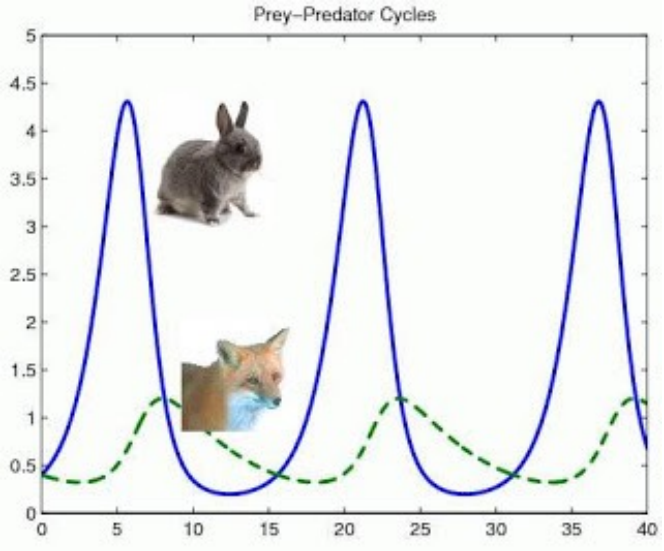
- Technique to explore complex phenomenas
- Define behavior rules for **agents**
- Run them concurrently many times
- Observe emerging dynamics on macro-level

Agent Based Modelling (ABM)

- Used in economics, social sciences, urban planning, biomedicine etc.
- Examples:
    - flock of birds
    - fish school
    - humans crowd
    - immune responses

Agent Based Modelling (ABM)

- Opposite to equation based modelling
- Examples: Lotka–Volterra equations – predator-prey system
- Agent-based model reproduce the same oscillation pattern

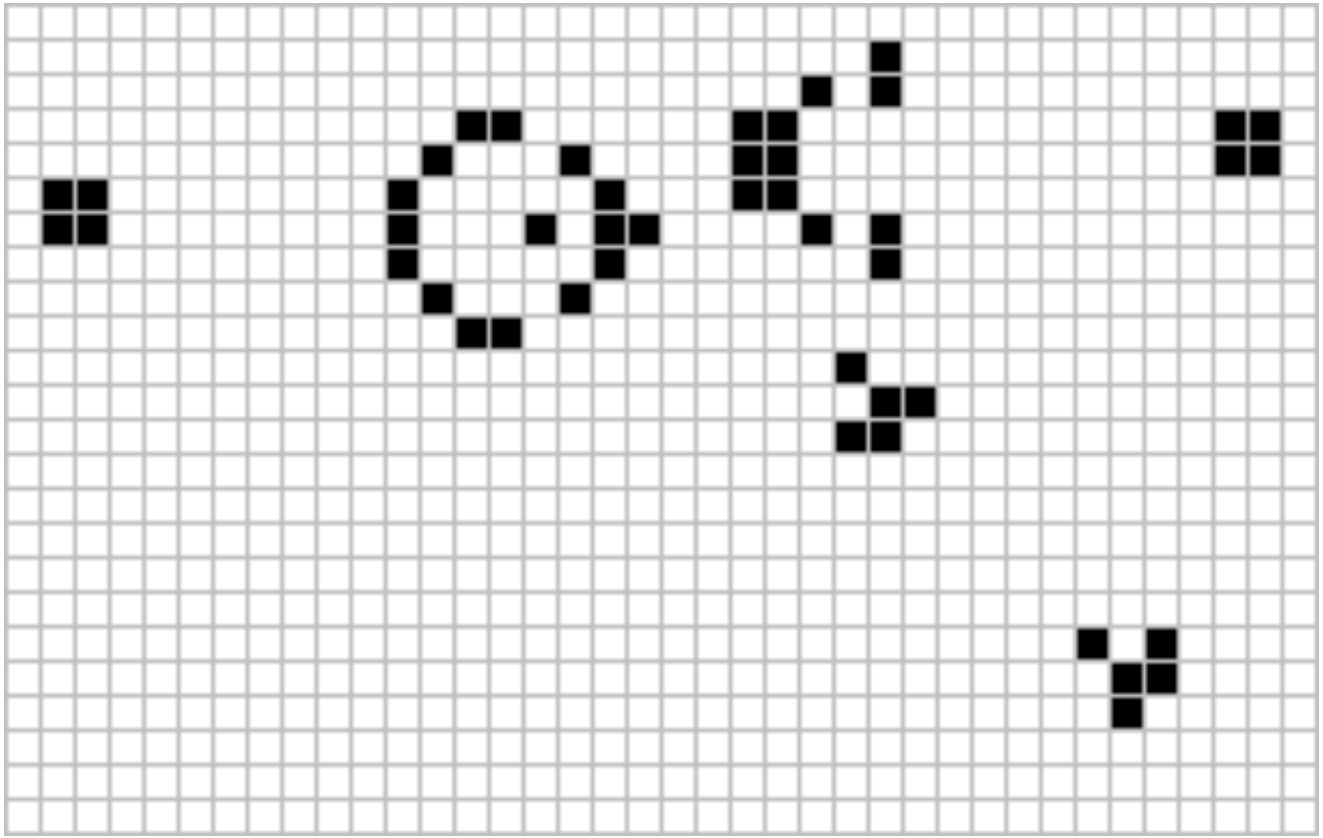


History of ABM

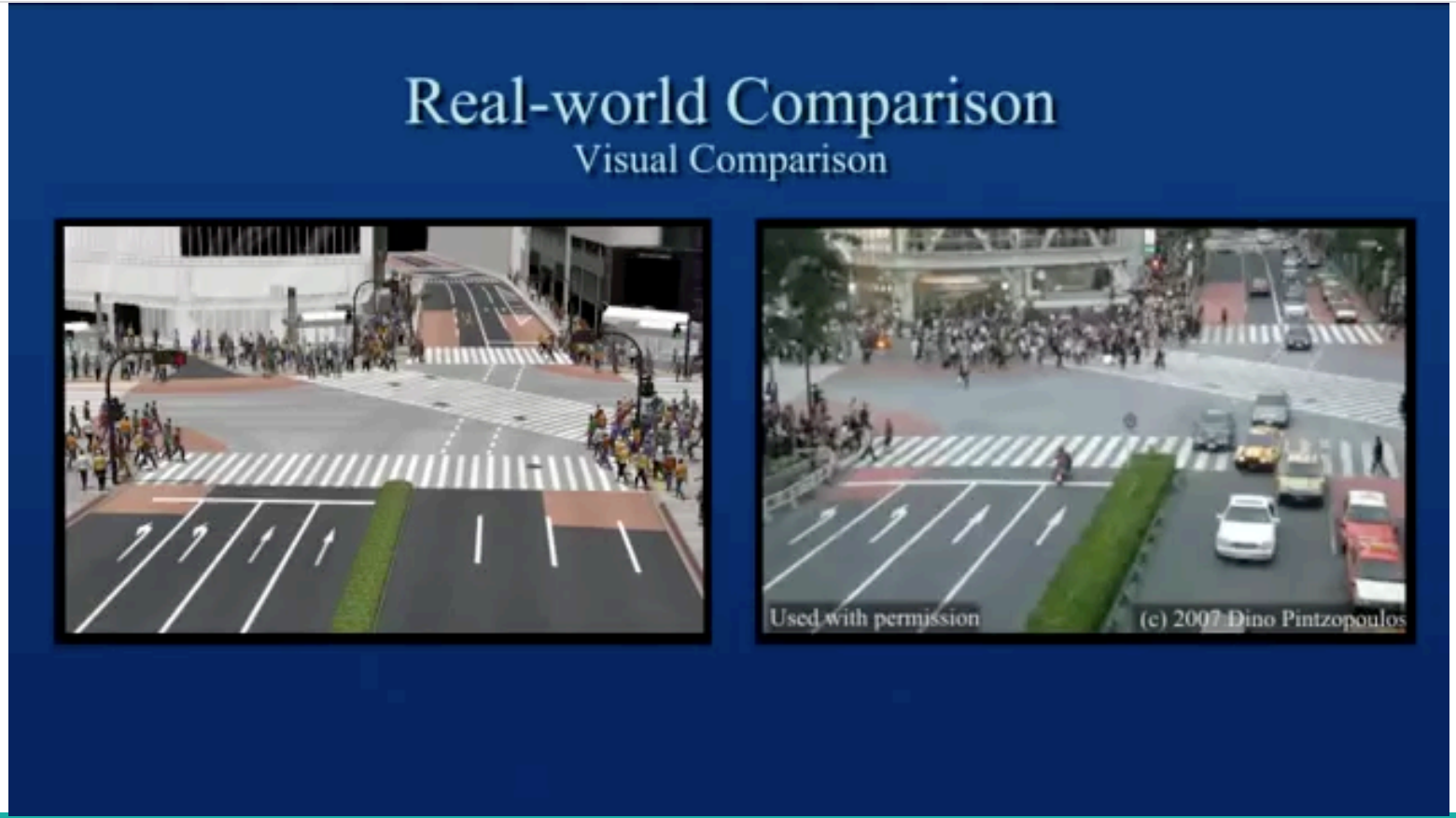
- **Cellular Automata** by Stanislaw Ulam and John von Neumann (1940s)
- **Game of Life** by John Horton Conway (1970s)
- **Distributed Prisoner's Dilemma** by Robert Axelrod (1980s)
- 1990-2000: Expansion of ABM software

Examples

Game of Life (Conway, 1970)

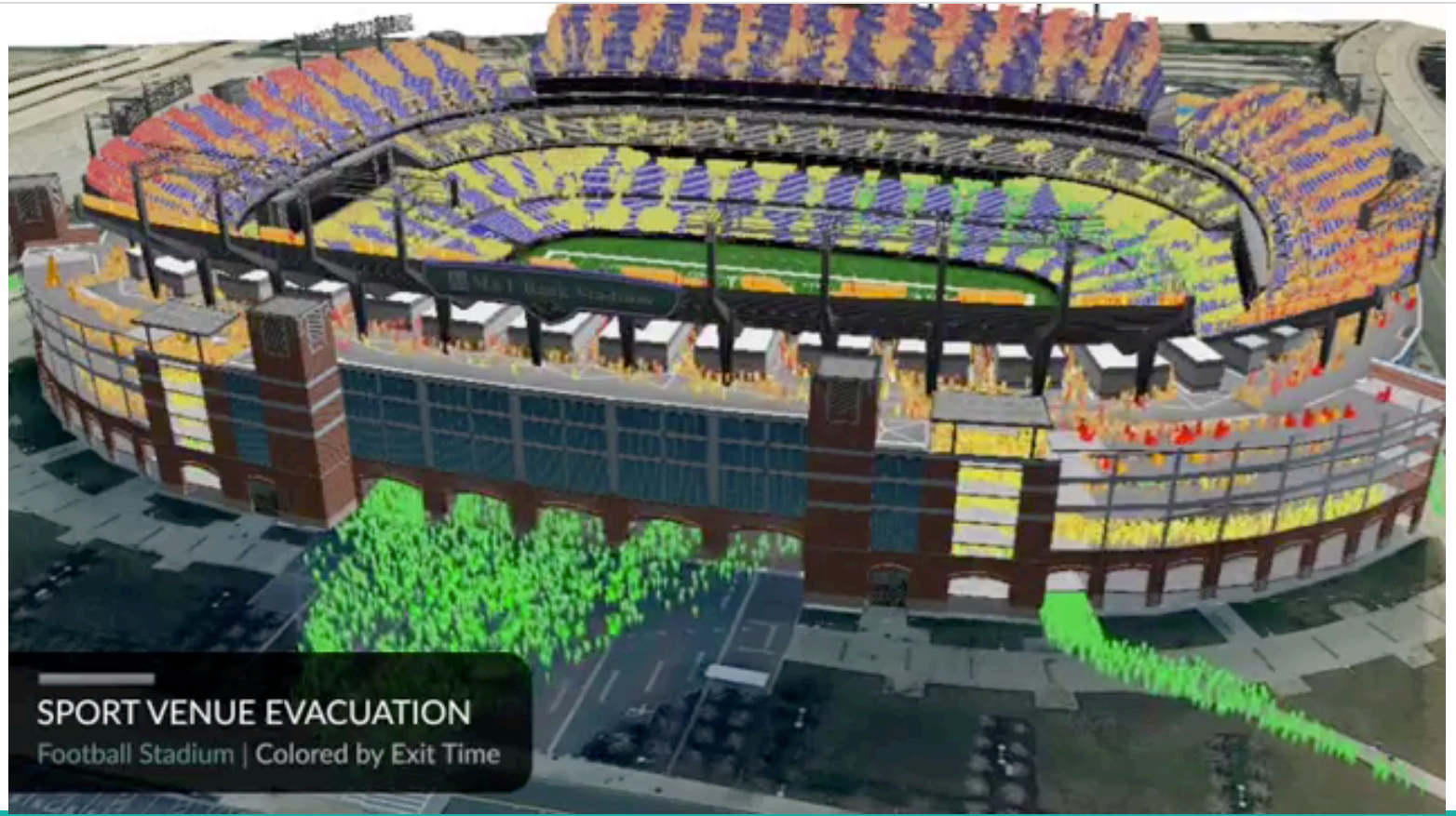


Crowd simulation



Source: https://www.youtube.com/watch?v=hpYdjHzHTkY

Facilities evacuation simulations



Source: https://www.youtube.com/watch?v=bTp1DRfULII

Traffic simulations

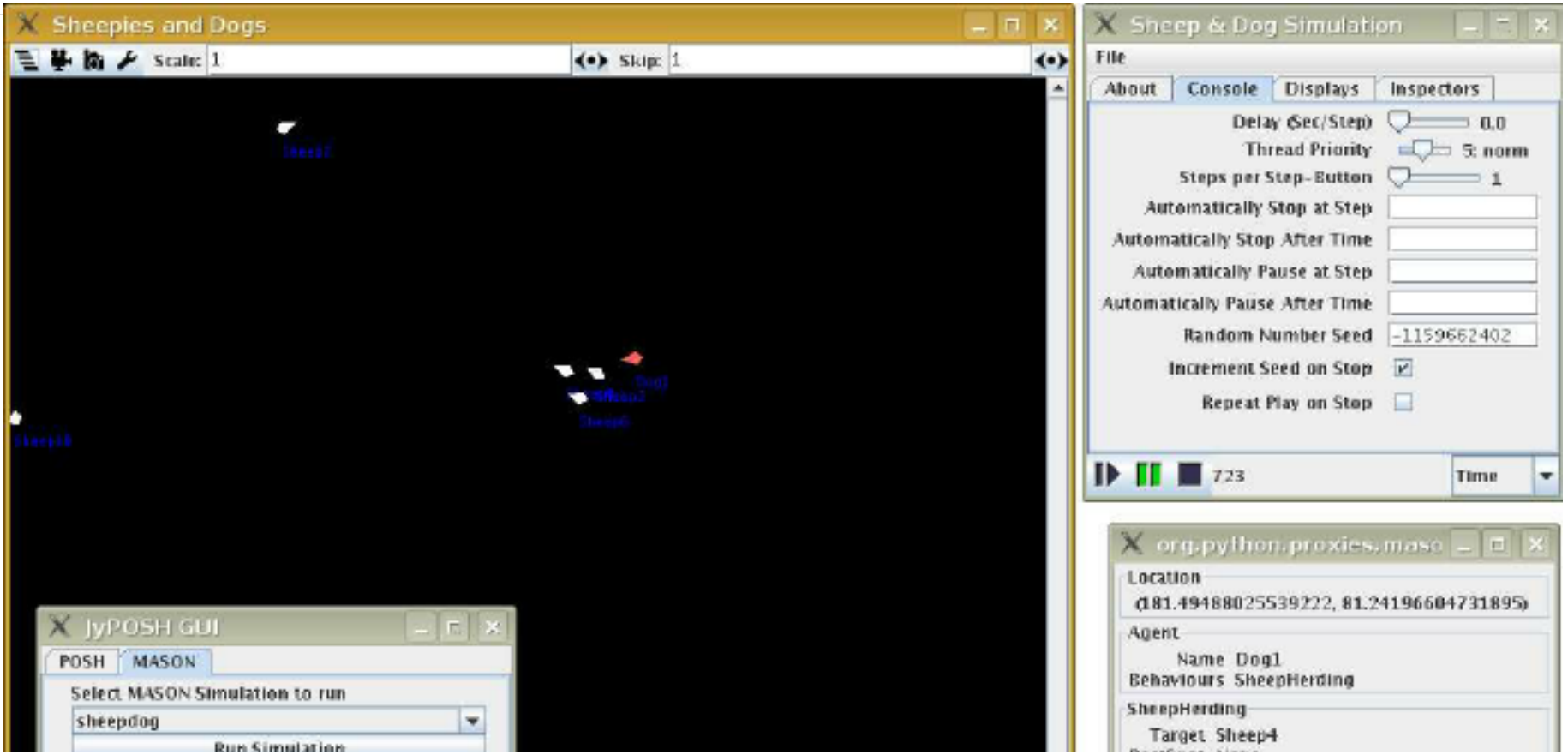
Source: https://www.youtube.com/watch?v=h3HLzMJ-ac8

ABM Software

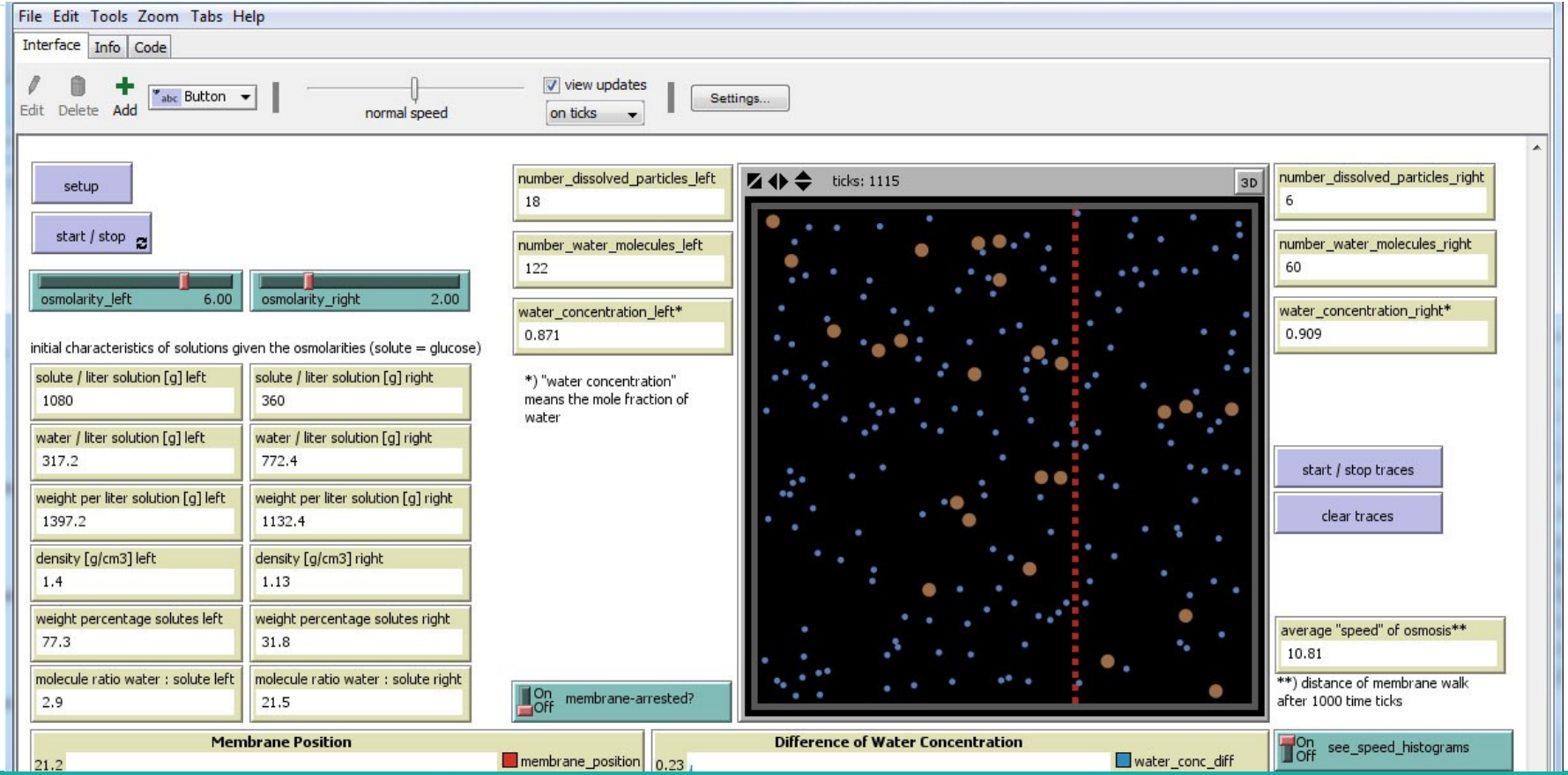
ABM Software

- Most of the ABM software is:
  - Hard to use
  - Complex
  - Ugly
  - Slow
  - Java

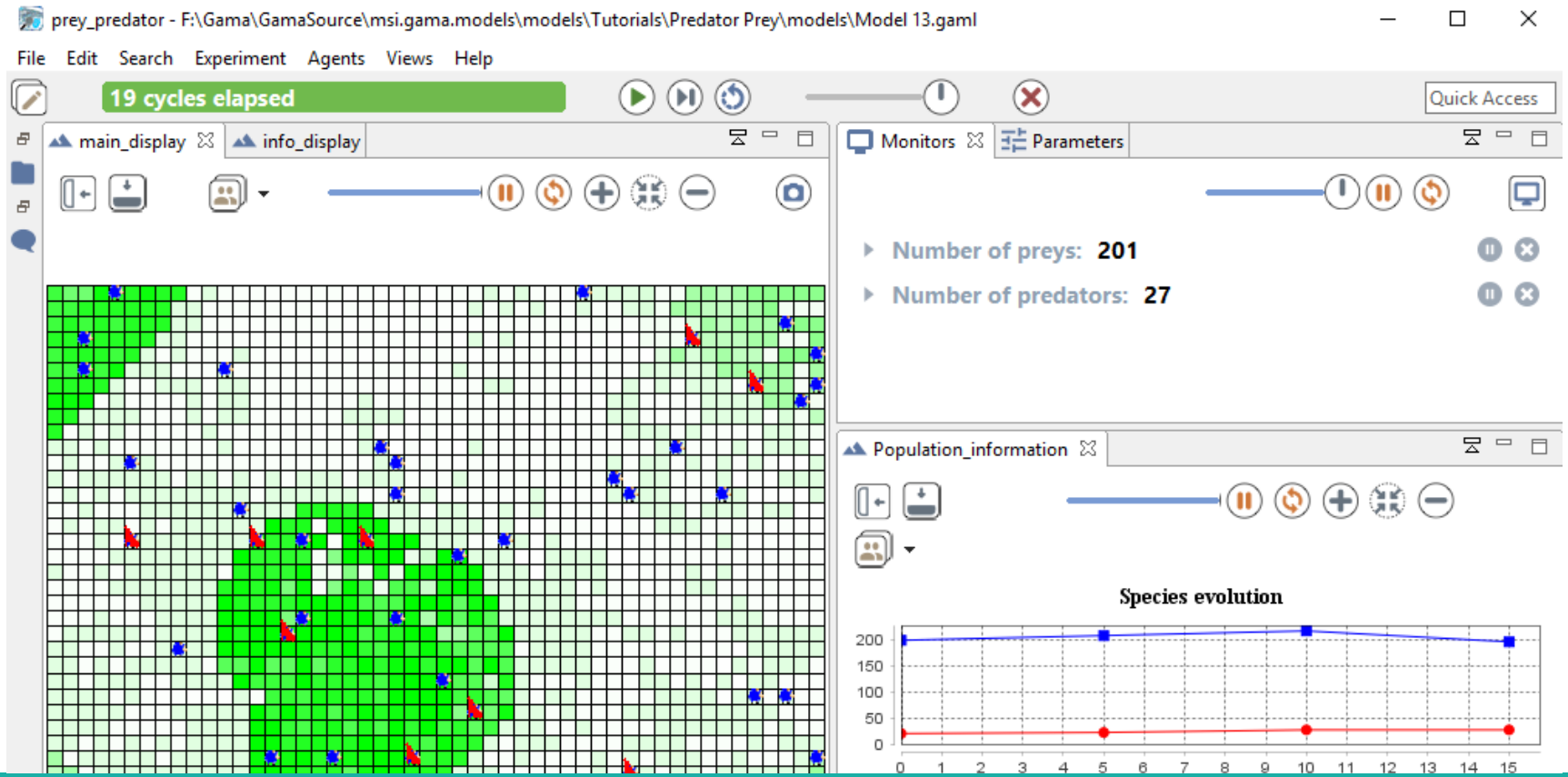
Mason


Sheepies and Dogs
Scale: 1
Skip: 1
Sheep2
Dog1
Sheep4
Sheep3
JyPOSH GUI
POSH
MASON
Select MASON Simulation to run
sheepdog
Run Simulation
Sheep & Dog Simulation
File
About
Console
Displays
Inspectors
Delay (Sec/Step)
0.0
Thread Priority
5: norm
Steps per Step-Button
1
Automatically Stop at Step
Automatically Stop After Time
Automatically Pause at Step
Automatically Pause After Time
Random Number Seed
-1159662402
Increment Seed on Stop
Repeat Play on Stop
723
Time
org.python.proxies.maso
Location
(181.49488025539222, 81.24196604731895)
Agent
Name Dog1
Behaviours SheepHerding
SheepHerding
Target Sheep4

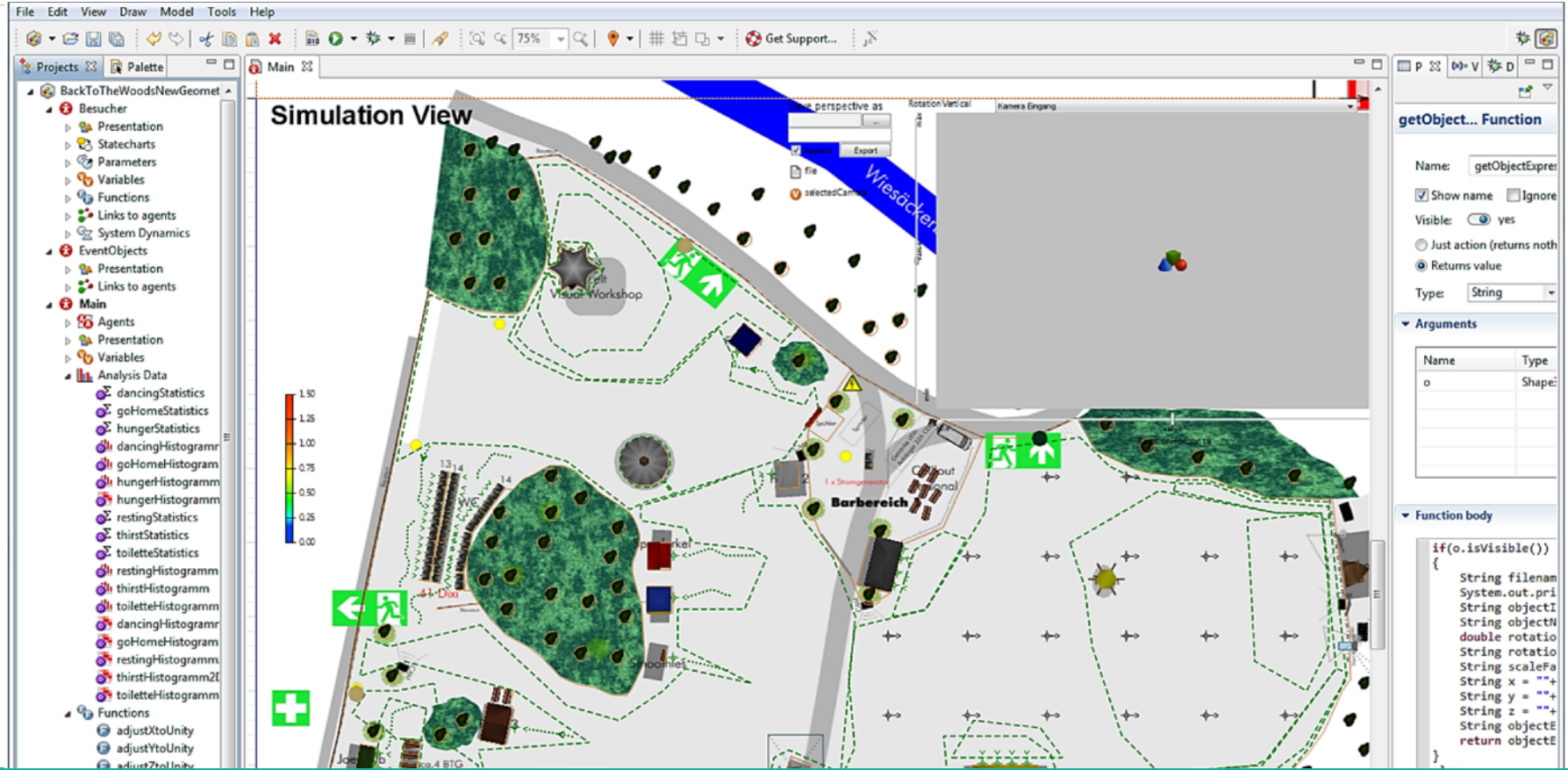
NetLogo


File Edit Tools Zoom Tabs Help
Interface Info Code
Edit Delete Add Button normal speed view updates on ticks Settings...
setup
start / stop
osmolarity_left 6.00
osmolarity_right 2.00
initial characteristics of solutions given the osmolarities (solute = glucose)
solute / liter solution [g] left 1080
solute / liter solution [g] right 360
water / liter solution [g] left 317.2
water / liter solution [g] right 772.4
weight per liter solution [g] left 1397.2
weight per liter solution [g] right 1132.4
density [g/cm3] left 1.4
density [g/cm3] right 1.13
weight percentage solutes left 77.3
weight percentage solutes right 31.8
molecule ratio water : solute left 2.9
molecule ratio water : solute right 21.5
number_dissolved_particles_left 18
number_water_molecules_left 122
water_concentration_left* 0.871
*) "water concentration" means the mole fraction of water
On Off membrane-arrested?
ticks: 1115
3D
number_dissolved_particles_right 6
number_water_molecules_right 60
water_concentration_right* 0.909
start / stop traces
clear traces
average "speed" of osmosis** 10.81
**) distance of membrane walk after 1000 time ticks
On Off see_speed_histograms
Membrane Position
21.2
membrane_position
Difference of Water Concentration
0.23
water_conc_diff

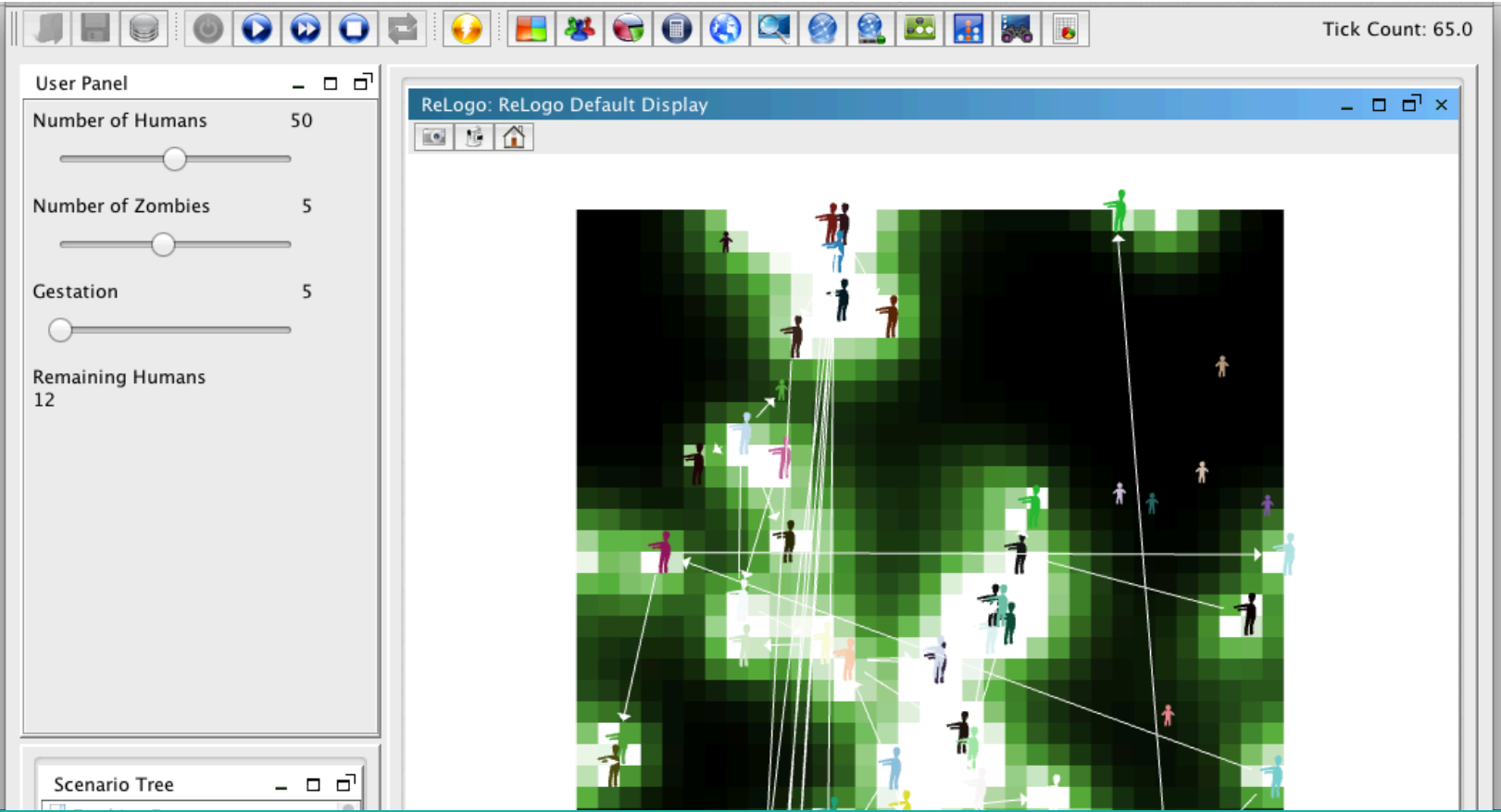
GAMA


prey_predator - F:\Gama\GamaSource\msi.gama.models\models\Tutorials\Predator Prey\models\Model 13.gaml
File Edit Search Experiment Agents Views Help
19 cycles elapsed
Quick Access
main_display
info_display
Monitors
Parameters
Number of preys: 201
Number of predators: 27
Population_information
Species evolution
200
150
100
50
0
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

AnyLogic


File Edit View Draw Model Tools Help
Get Support...
Projects
Palette
Main
Simulation View
BackToTheWoodsNewGeomet
Besucher
Presentation
Statecharts
Parameters
Variables
Functions
Links to agents
System Dynamics
EventObjects
Presentation
Links to agents
Main
Agents
Presentation
Variables
Analysis Data
dancingStatistics
goHomeStatistics
hungerStatistics
dancingHistogramm
goHomeHistogram
hungerHistogramm
hungerHistogramm
restingStatistics
thirstStatistics
toiletteStatistics
restingHistogramm
thirstHistogramm
toiletteHistogramm
dancingHistogramm
goHomeHistogram
restingHistogramm
thirstHistogramm2
toiletteHistogramm
Functions
adjustXtoUnity
adjustYtoUnity
Wiesdecken
Visual Workshop
Barbereich
getObject... Function
Name:
Show name
Ignore
Visible:
yes
Just action (returns noth
Returns value
Type:
String
Arguments
Name
Type
o
Function body
if(o.isVisible())
{
String filenam
System.out.pri
String objectI
String objectN
double rotatio
String rotatio
String scaleFa
String x = ""+
String y = ""+
String z = ""+
String objectE
return objectE
}

RePast


Tick Count: 65.0
User Panel
Number of Humans 50
Number of Zombies 5
Gestation 5
Remaining Humans
12
ReLogo: ReLogo Default Display
Scenario Tree

ABM Software Issues

Hard to use

- New programming language
- UI / Interface
- Performance / scalability
- Installation issues

Good luck trying to play with even simple model on your weekend.

What exactly ABM software does?

ABM Software

- Program model in high-level simple language
- Run many models concurrently
- Compute interactions/world/state
- Collect/analyze world/model
- Present result via some UI

ABM Software

- Program model in high-level **simple language** 
- Run many models concurrently
- Compute interactions/world/state
- Collect/analyze world/model
- Present result via some UI

ABM Software

- Program model in high-level **simple language** ✓
- Run many models **concurrently** ✓
- Compute interactions/world/state
- Collect/analyze world/model
- Present result via some UI

ABM Software

- Program model in high-level **simple language** ✓
- Run many models **concurrently** ✓
- **Compute** interactions/world/state ✓
- Collect/analyze world/model
- Present result via some UI

ABM Software

- Program model in high-level **simple language** ✓
- Run many models **concurrently** ✓
- **Compute** interactions/world/state ✓
- **Collect/analyze** world/model ✓
- Present result via some UI

ABM Software

- Program model in high-level **simple language** ✓
- Run many models **concurrently** ✓
- **Compute** interactions/world/state ✓
- **Collect/analyze** world/model ✓
- Present result via some **UI** ?

“Sounds like a plan!”

Gopher

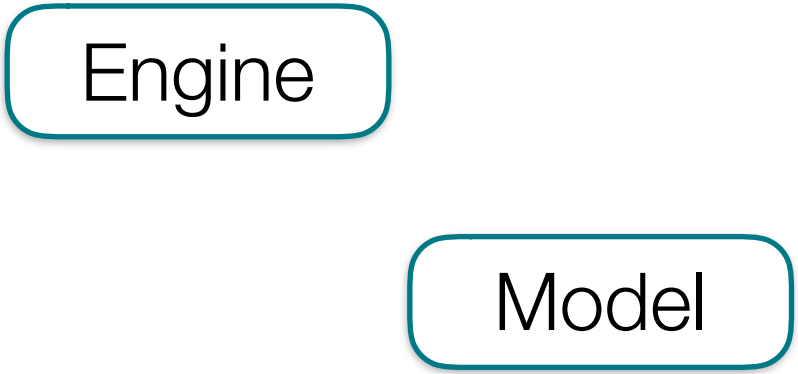
GoABM

github.com/divan/goabm

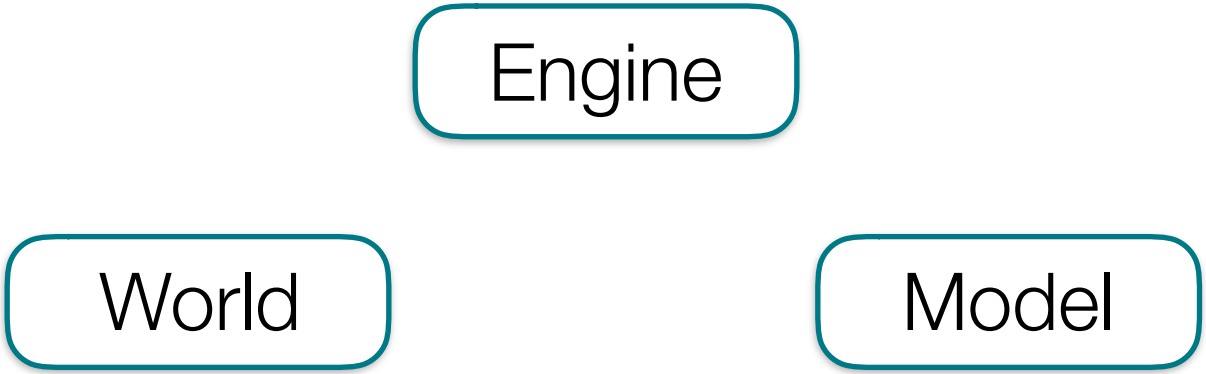
What's important for ABM framework?

What's important for ABM framework?

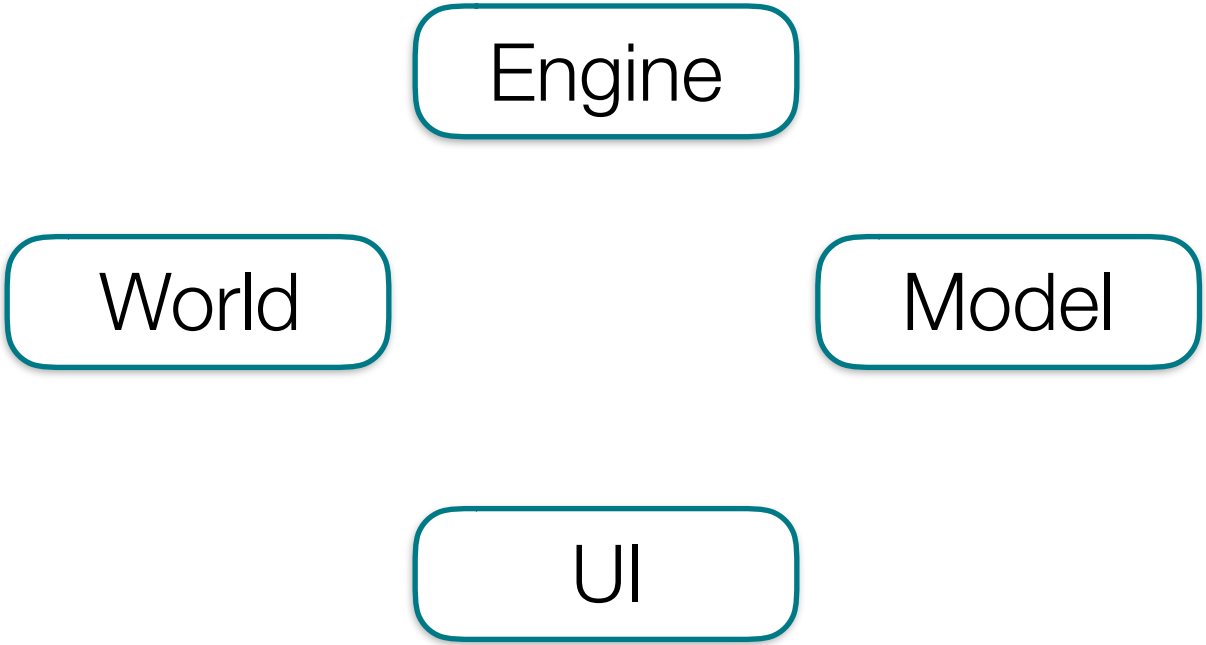
What's important for ABM framework?



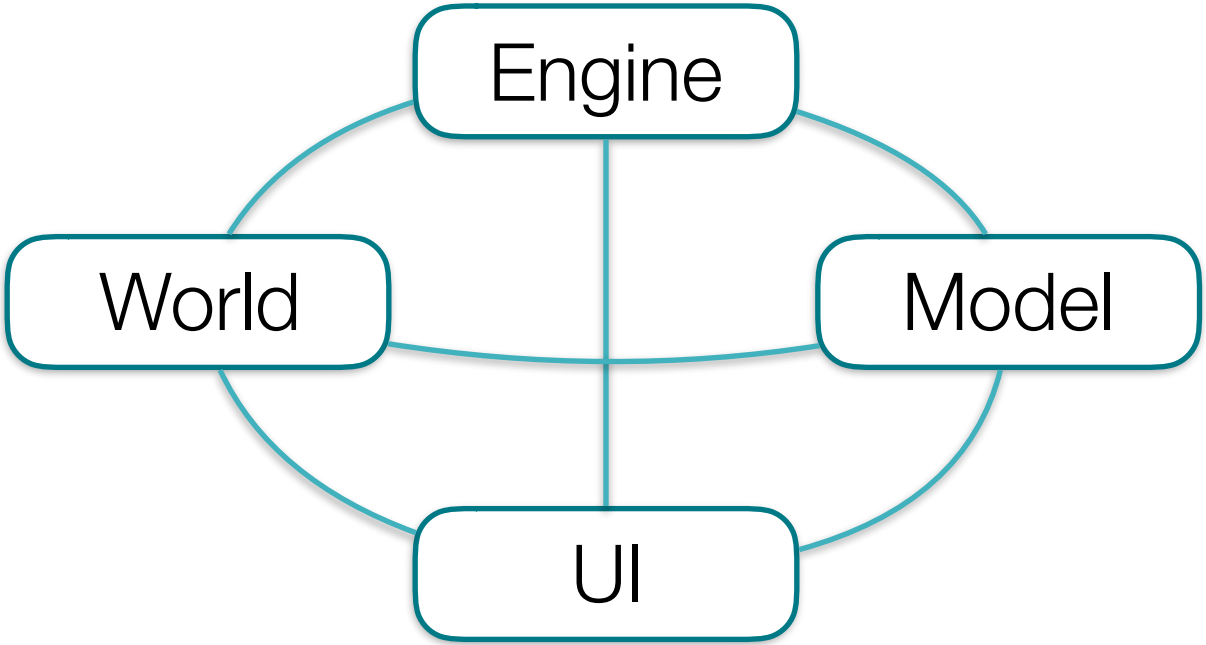
What's important for ABM framework?



What's important for ABM framework?



What's important for ABM framework?



Engine

GoABM

```
// ABM is an "engine" of the simulation framework.
// It connects the Agents, Worlds and UI and runs the actual
// simulation.
type ABM struct {
    ... // private fields
}

func (*ABM) AddAgent(a Agent) {
    ...
}

func (*ABM) StartSimulation() {
    ...
}
```

GoABM

```
// ABM is an "engine" of the simulation framework.
// It connects the Agents, Worlds and UI and runs the actual
// simulation.
type ABM struct {
    ... // private fields
}

func (*ABM) AddAgent(a Agent) {
    ...
}

func (*ABM) StartSimulation() {
    ...
}
```

GoABM

```
// ABM is an "engine" of the simulation framework.
// It connects the Agents, Worlds and UI and runs the actual
// simulation.
type ABM struct {
    ... // private fields
}

func (*ABM) AddAgent(a Agent) {
    ...
}

func (*ABM) StartSimulation() {
    ...
}
```

GoABM

```
// ABM is an "engine" of the simulation framework.
// It connects the Agents, Worlds and UI and runs the actual
// simulation.
type ABM struct {
    ... // private fields
}

func (*ABM) AddAgent(a Agent) {
    ...
}

func (*ABM) StartSimulation() {
    ...
}
```

GoABM

```
        for i := 0; i < a.Limit(); i++ {
                if a.World() != nil {
                        a.World().Tick()
                }
                var wg sync.WaitGroup
                for j := 0; j < a.AgentsCount(); j++ {
                        wg.Add(1)
                        go func(wg *sync.WaitGroup,, j int) {
                                a.agents[j].Run()
                                wg.Done()
                        }(&wg, j)
                }
                wg.Wait()
                if a.reportFunc != nil {
                        a.reportFunc(a)
                }
        }
```

Agent

```
// Agent  defines a model of
independent agent's behavior.
type Agent interface {
  Run()
}
```

```
// Agent  defines a model of
independent agent's behavior.
type Agent interface {
   Run()
}
```

```
// Human implements Agent for human that can age.
type Human struct {
    age int
    dead bool // zero values must be useful :)
}
```

```
// Human implements Agent for human that can age.
type Human struct {
    age int
    dead bool // zero values must be useful :)
}

func (h *Human) Run() {
    If h.dead { return }
    h.age++
    if h.age == AvgDeathAge {
        h.Die()
    }
}

func (h *Human) Die() {
    h.dead = true
}
```

```
// Human implements Agent for human that can age.
type Human struct {
    age int
    dead bool // zero values must be useful :)
}

func (h *Human) Run() {
    If h.dead { return }
    h.age++
    if h.age == AvgDeathAge {
        h.Die()
    }
}

func (h *Human) Die() {
    h.dead = true
}
```

Agent model

- You can create any type of actions
- Use random generators for different probability distributions
  - github.com/atgjack/prob
  - http://www.gonum.org

World

World

- World is where agent interact and have "location"
  - Continious
  - Discrete (grids)
  - Graphs
  - GIS data (geography)
- Non-spatial worlds
  - Belief space
  - XOR-distance based P2P networks

```
// World represents space in which agents dwell and
// interact. It updates the state on each Tick().
type World interface {
    Tick() // mark the beginning of the next time period
}
```

```
// World represents space in which agents dwell and
// interact. It updates the state on each Tick().
type World interface {
    Tick() // mark the beginning of the next time period
}
```

UI

```
// UI defines the minimal user interface type.
type UI interface {
        Stop()
        Loop()
}

type Charts interface {
        AddChart(name string, values <-chan float64)
}

type Grid interface {
        AddGrid(<-chan [][]interface{})
}

type Grid3D interface {
        AddGrid3D(<-chan []interface{})
}
```

```
// UI defines the minimal user interface type.
type UI interface {
        Stop()
        Loop()
}

type Charts interface {
        AddChart(name string, values <-chan float64)
}

type Grid interface {
        AddGrid(<-chan [][]interface{})
}

type Grid3D interface {
        AddGrid3D(<-chan []interface{})
}
```

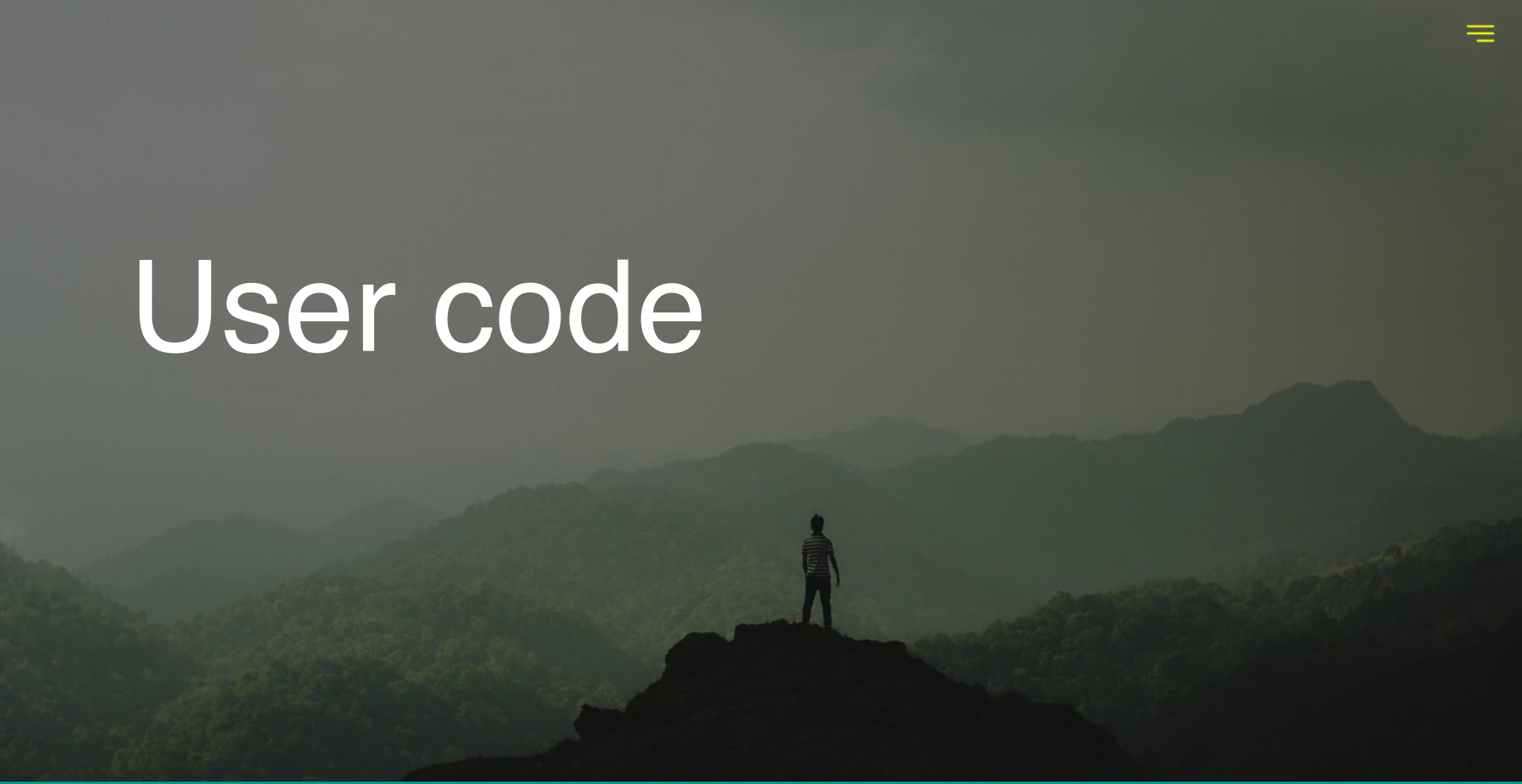
```
// UI defines the minimal user interface type.
type UI interface {
        Stop()
        Loop()
}

type Charts interface {
        AddChart(name string, values <-chan float64)
}

type Grid interface {
        AddGrid(<-chan [][]interface{})
}

type Grid3D interface {
        AddGrid3D(<-chan []interface{})
}
```

```
// UI defines the minimal user interface type.
type UI interface {
        Stop()
        Loop()
}

type Charts interface {
        AddChart(name string, values <-chan float64)
}

type Grid interface {
        AddGrid(<-chan [][]interface{})
}

type Grid3D interface {
        AddGrid3D(<-chan []interface{})
}
```

UI

- Console / terminal
- Native Desktop (x/exp/shiny)
- Web-based / WebGL

User code



```
func main() {
        a := abm.New()
        a.SetWorld(grid.New(100, 100))
        a.AddAgent(&Human{})

        ch := make(chan int)
        a.SetReportFunc(func(a *abm.ABM) {
                ch <- a.Count(func(agent abm.Agent) bool {
                        return agent(*human.Human).IsAlive()
                })
        })
        go a.StartSimulation()

        ui := term.NewUI()
        defer ui.Stop()
        ui.AddChart("Humans Alive", ch)
        ui.Loop()
}
```

Demo time

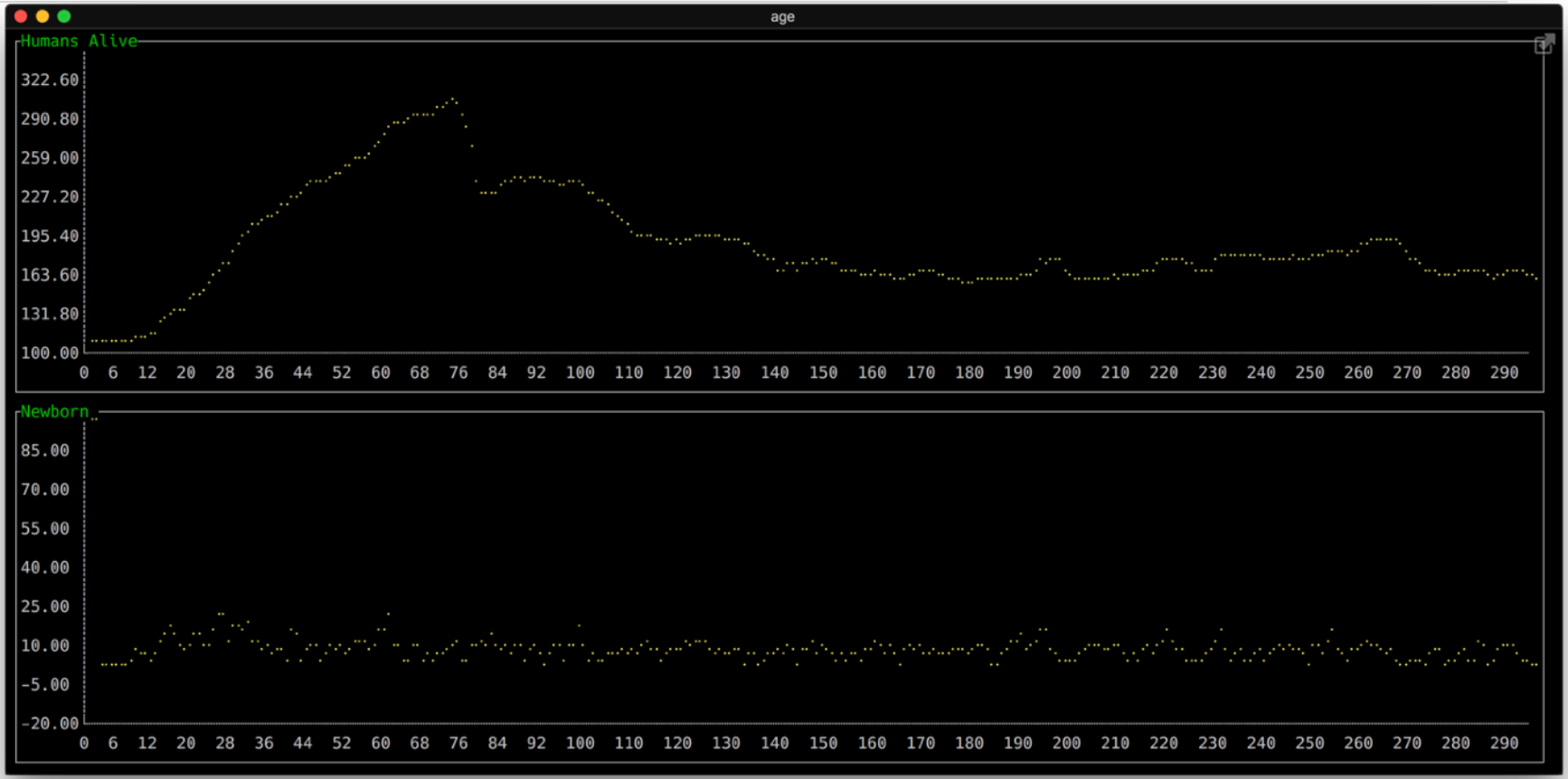
UI implementing ui.Charts interface

```
package main

import (
  "github.com/divan/goabm/abm"
  "github.com/divan/goabm/models/human"
  "github.com/divan/goabm/ui/term"
)

func main() {
  a := abm.New()

  for i := 0; i < 100; i++ {
     a.AddAgent(human.New(a))
  }

  a.LimitIterations(200)

  alivesCh := make(chan float64)
  a.SetReportFunc(func(a *abm.ABM) {
     alive := a.Count(func(agent abm.Agent) bool {
        h := agent.(*human.Human)
        return h.IsAlive()
     })
     alivesCh <- float64(alive)
  })

  go a.StartSimulation()

  ui := term.NewUI()
  defer ui.Stop()

  ui.AddChart("Humans Alive", alivesCh)
  ui.Loop()
}
```

UI implementing ui.Charts interface

```
package main

import (
  "github.com/divan/goabm/abm"
  "github.com/divan/goabm/models/human"
  "github.com/divan/goabm/ui/term"
)

func main() {
  a := abm.New()

  for i := 0; i < 100; i++ {
     a.AddAgent(human.New(a))
  }

  a.LimitIterations(500)

  alivesCh, newbornsCh := make(chan float64), make(chan
float64)
  a.SetReportFunc(func(a *abm.ABM) {
     alive := a.Count(func(agent abm.Agent) bool {
        h := agent.(*human.Human)
        return h.IsAlive()
     })
     alivesCh <- float64(alive)
     newborns := a.Count(func(agent abm.Agent) bool {
        h := agent.(*human.Human)
        return h.Age() < 3
     })
     newbornsCh <- float64(newborns)
  })

  go a.StartSimulation()

  ui := term.NewUI()
  defer ui.Stop()

  ui.AddChart("Humans Alive", alivesCh)
  ui.AddChart("Newborns", newbornsCh)

  ui.Loop()
}
```

UI implementing ui.Charts interface



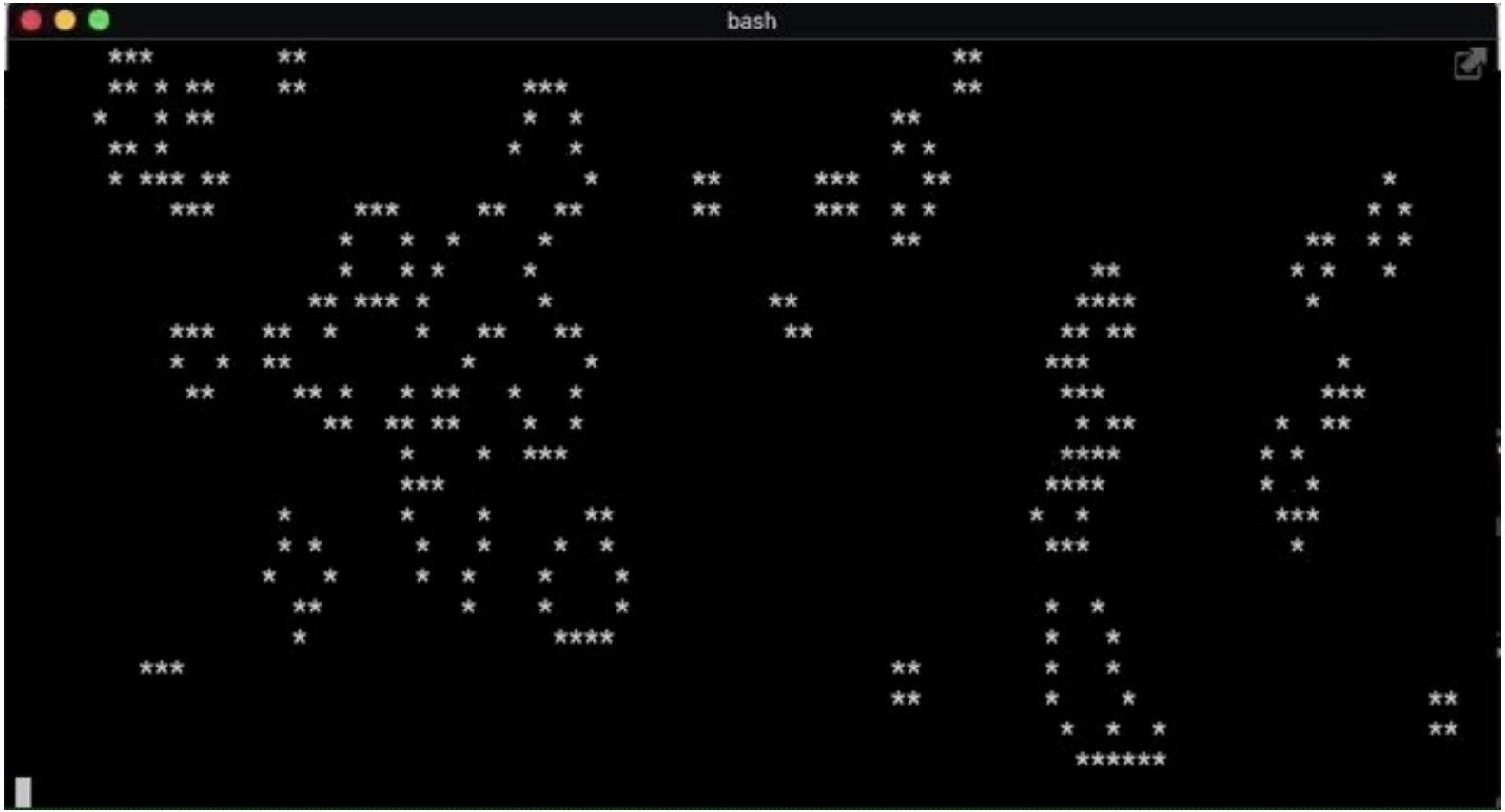
Game of Life example

```
// Run runs single iteration over cell.
//
// Any live cell with fewer than two live neighbors dies, as if caused by under population.
// Any live cell with two or three live neighbors lives on to the next generation.
// Any live cell with more than three live neighbors dies, as if by overpopulation.
// Any dead cell with exactly three live neighbors becomes a live cell, as if by reproduction.
func (c *Cell) Run() {
  neighbors := c.CountNeighbors()
  if c.IsAlive() {
     if neighbors < 2 || neighbors > 3 {
        c.Die()
     }
  } else {
     if neighbors == 3 {
        c.Reborn()
     }
  }
}
```

Game of Life example

```
package main

import (
  "math/rand"
  "time"

  "github.com/divan/goabm/abm"
  "github.com/divan/goabm/models/conway_life"
  "github.com/divan/goabm/ui/term_grid"
  "github.com/divan/goabm/worlds/grid2d"
)

func main() {
  rand.Seed(time.Now().UnixNano())
  a := abm.New()
  w, h := termgrid.TermSize()
  g := grid.New(w, h)
  a.SetWorld(g)

  // populate grid randomly
  for x := 0; x < w; x++ {
     for y := 0; y < h; y++ {
        alive := rand.Float64() > 0.5
        cell := life.New(a, x, y, alive)
        a.AddAgent(cell)
        g.SetCell(x, y, cell)
     }
  }

  ch := make(chan [][]interface{})
  a.SetReportFunc(func(a *abm.ABM) {
     ch <- g.Dump(life.IsAlive)
     time.Sleep(10 * time.Millisecond)
  })

  go a.StartSimulation()

  ui := termgrid.New()
  defer ui.Stop()
  ui.AddGrid(ch)
  ui.Loop()
}
```

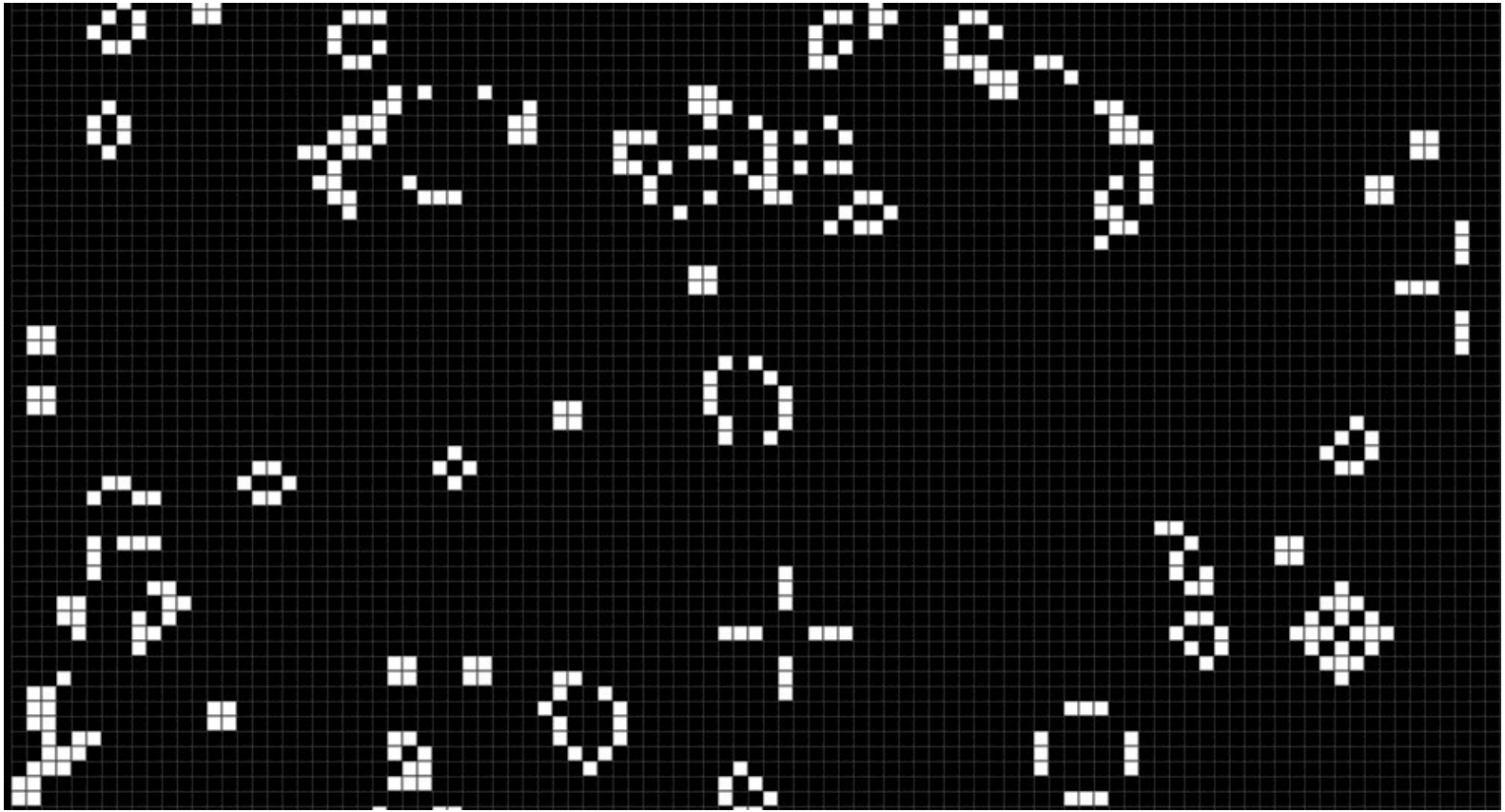
Game of Life example



Game of Life example

```
package main

import (
  "math/rand"
  "time"

  "github.com/divan/goabm/abm"
  "github.com/divan/goabm/models/conway_life"
  "github.com/divan/goabm/ui/shiny_grid"
  "github.com/divan/goabm/worlds/grid2d"
)

func main() {
  rand.Seed(time.Now().UnixNano())
  a := abm.New()
  w, h := 300, 200
  g := grid.New(w, h)
  a.SetWorld(g)

  // populate grid randomly
  for x := 0; x < w; x++ {
     for y := 0; y < h; y++ {
        alive := rand.Float64() > 0.5
        cell := life.New(a, x, y, alive)
        a.AddAgent(cell)
        g.SetCell(x, y, cell)
     }
  }

  ch := make(chan [][]interface{})
  a.SetReportFunc(func(a *abm.ABM) {
     ch <- g.Dump(life.IsAlive)
     time.Sleep(10 * time.Millisecond)
  })

  go a.StartSimulation()

  ui := shiny.New()
  defer ui.Stop()
  ui.AddGrid(ch)
  ui.Loop()
}
```

Game of Life example



Random Walk

Main page
Contents
Featured content
Current events
Random article
Donate to Wikipedia
Wikipedia store

Interaction
Help
About Wikipedia
Community portal
Recent changes
Contact page

Tools
What links here
Related changes
Upload file
Special pages
Permanent link
Page information
Wikidata item
Cite this page

Print/export
Create a book
Download as PDF
Printable version

In other projects
Wikimedia Commons

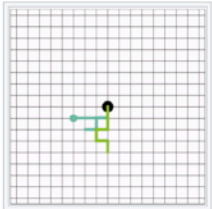
Languages
العربية
Български
Čeština
Deutsch

Random walk

From Wikipedia, the free encyclopedia

For the Lawrence Block novel, see Random Walk.

A **random walk** is a mathematical object, known as a stochastic or random process, that describes a path that consists of a succession of random steps on some mathematical space such as the integers. An elementary example of a random walk is the random walk on the integer number line, $\mathbb{Z}$, which starts at 0 and at each step moves +1 or −1 with equal probability. Other examples include the path traced by a molecule as it travels in a liquid or a gas, the search path of a foraging animal, the price of a fluctuating stock and the financial status of a gambler can all be approximated by random walk models, even though they may not be truly random in reality. As illustrated by those examples, random walks have applications to many scientific fields including ecology, psychology, computer science, physics, chemistry, biology as well as economics. Random walks explain the observed behaviors of many processes in these fields, and thus serve as a fundamental model for the recorded stochastic activity. As a more mathematical application, the value of pi can be approximated by the usage of random walk in agent-based modelling environment.[1][2] The term *random walk* was first introduced by Karl Pearson in 1905.[3]

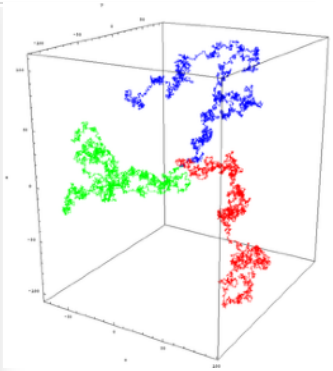


Example of five random walk simulations in a 2-dimensional integer lattice. Produced by the Random Walk Simulator.

Various types of random walks are of interest, which can differ in several ways. The term itself most often refers to a special category of Markov chains or Markov processes, but many time-dependent processes are referred to as random walks, with a modifier indicating their specific properties. Random walks (Markov or not) can also take place on a variety of spaces: commonly studied ones include graphs, others on the integers or the real line, in the plane or in higher-dimensional vector spaces, on curved surfaces or higher-dimensional Riemannian manifolds, and also on groups finite, finitely generated or Lie. The time parameter can also be manipulated. In the simplest context the walk is in discrete time, that is a sequence of random variables $(X_t) = (X_1, X_2, \ldots)$ indexed by the natural numbers. However, it is also possible to define random walks which take their steps at random times, and in that case the position X_t has to be defined for all times $t \in [0, +\infty)$. Specific cases or limits of random walks include the Lévy flight and diffusion models such as Brownian motion.

Random walks are a fundamental topic in discussions of Markov processes. Their mathematical study has been extensive. Several properties, including dispersal distributions, first-passage or hitting times, encounter rates, recurrence or transience, have been introduced to quantify their behaviour.

Contents [hide]



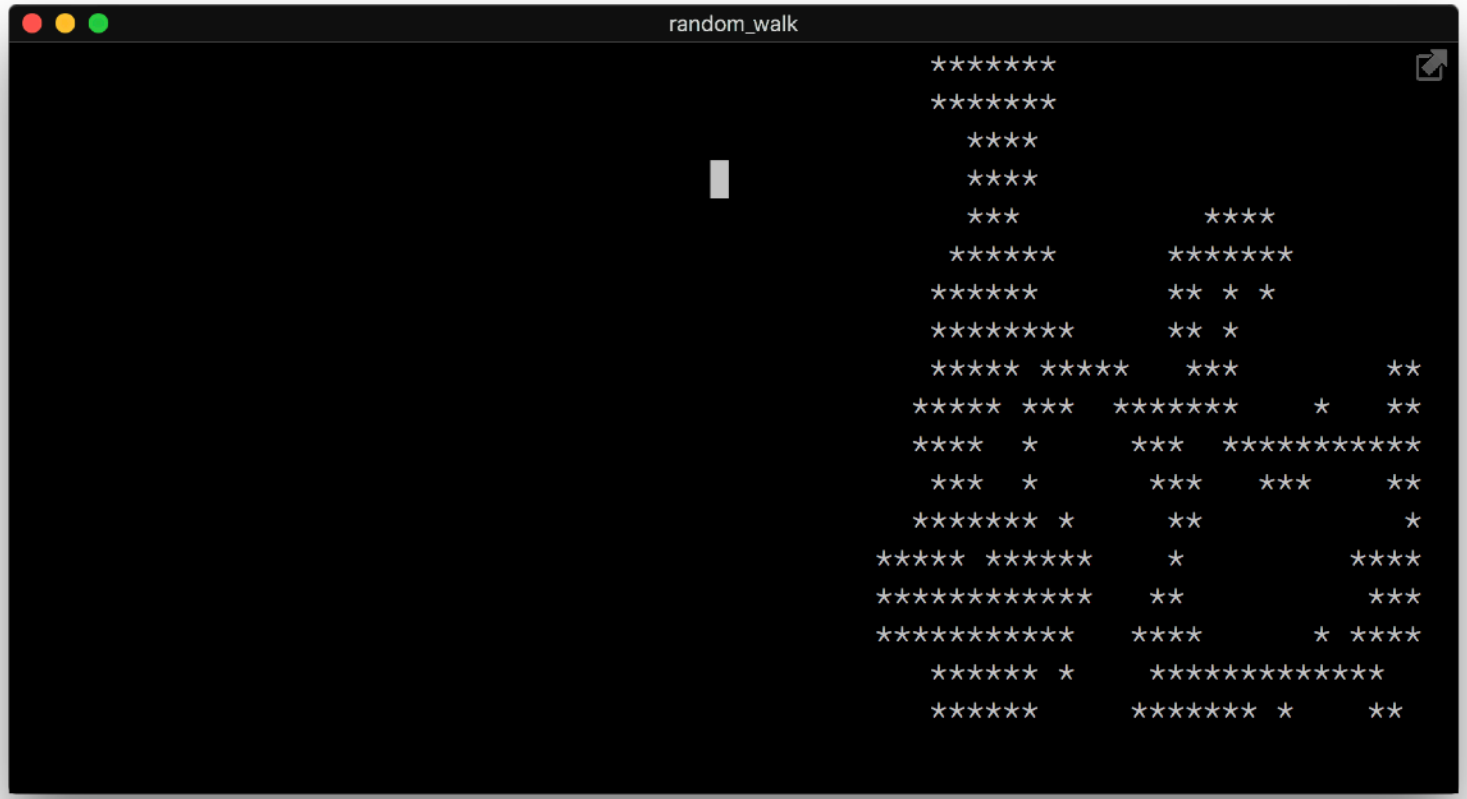
Random Walk

```
func (w *Walker) Run() {
  rx := rand.Intn(4)
  oldx, oldy := w.x, w.y
  switch rx {
  case 0:
     w.x++
  case 1:
     w.y++
  case 2:
     w.x--
  case 3:
     w.y--
  }

  var err error
  if w.trail {
     err = w.grid.Copy(oldx, oldy, w.x, w.y)
  } else {
     err = w.grid.Move(oldx, oldy, w.x, w.y)
  }

  if err != nil {
     w.x, w.y = oldx, oldy
  }
}
```

Random Walk

```
package main

import (
  "log"
  "math/rand"
  "time"

  "github.com/divan/goabm/abm"
  "github.com/divan/goabm/models/random_walker"
  "github.com/divan/goabm/ui/term_grid"
  "github.com/divan/goabm/worlds/grid2d"
)

func main() {
  rand.Seed(time.Now().UnixNano())
  a := abm.New()
  w, h := termgrid.TermSize()
  grid2D := grid.New(w, h)
  a.SetWorld(grid2D)

  cell, err := walker.New(a, rand.Intn(w-1),
rand.Intn(h-1), true)
  if err != nil {
    log.Fatal(err)
  }
  a.AddAgent(cell)
  grid2D.SetCell(cell.X(), cell.Y(), cell)

  ch := make(chan [][]interface{})
  a.SetReportFunc(func(a *abm.ABM) {
    ch <- grid2D.Dump(func(a abm.Agent) bool
{ return a != nil })
  })

  go func() {
    a.StartSimulation()
    close(ch)
  }()

  ui := termgrid.New()
  defer ui.Stop()
  ui.AddGrid(ch)
  ui.Loop()
}
```

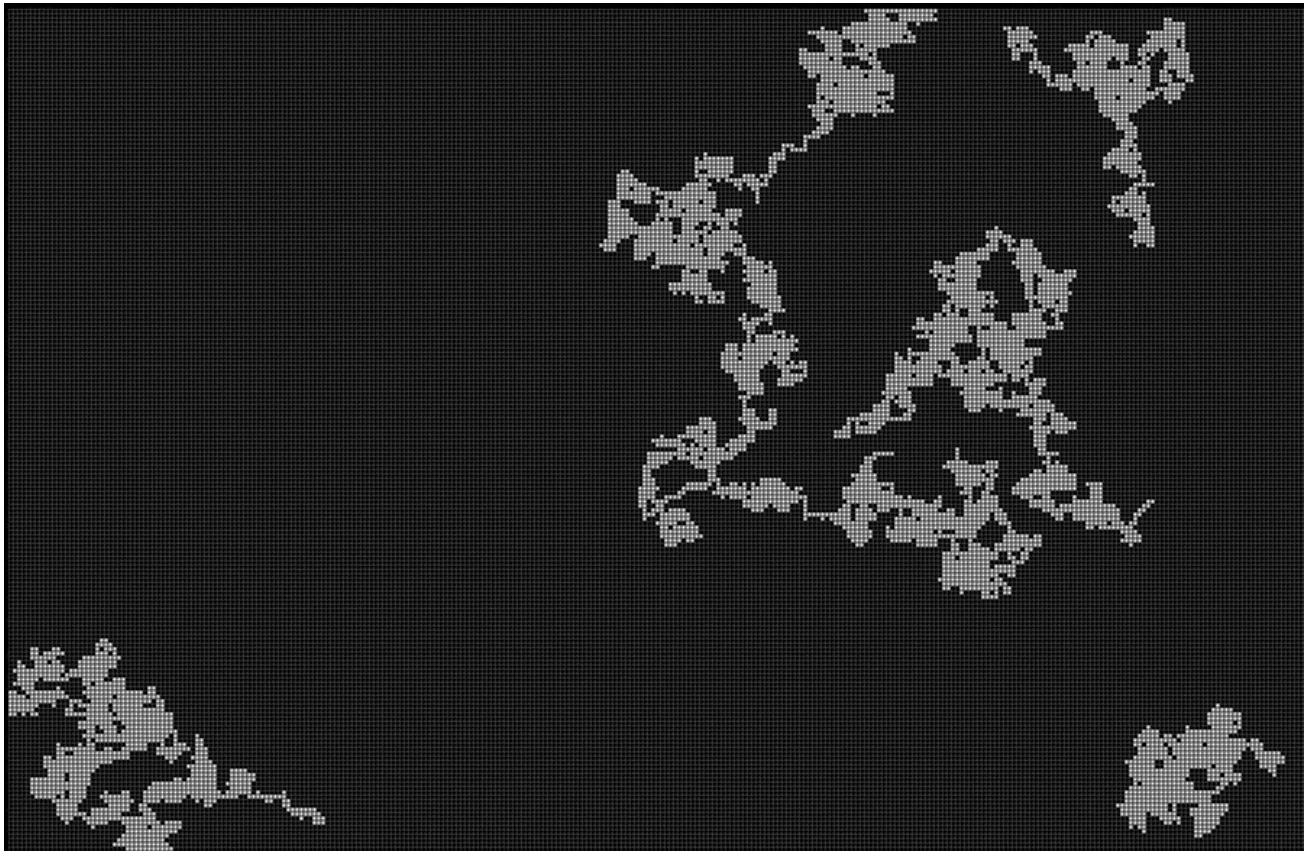
Random Walk



Random Walk

```
package main

import (
  "log"
  "math/rand"
  "time"

  "github.com/divan/goabm/abm"
  "github.com/divan/goabm/models/random_walker"
  "github.com/divan/goabm/ui/term_grid"
  "github.com/divan/goabm/worlds/grid2d"
)

func main() {
  rand.Seed(time.Now().UnixNano())
  a := abm.New()
  w, h := 300, 200
  grid2D := grid.New(w, h)
  a.SetWorld(grid2D)

  cell, err := walker.New(a, rand.Intn(w-1),
rand.Intn(h-1), true)
  if err != nil {
    log.Fatal(err)
  }
  a.AddAgent(cell)
  grid2D.SetCell(cell.X(), cell.Y(), cell)

  ch := make(chan [][]interface{})
  a.SetReportFunc(func(a *abm.ABM) {
    ch <- grid2D.Dump(func(a abm.Agent) bool
{ return a != nil })
  })

  go func() {
    a.StartSimulation()
    close(ch)
  }()

  ui := shiny.New(w, h)
  defer ui.Stop()
  ui.AddGrid(ch)
  ui.Loop()
}
```

Random Walk



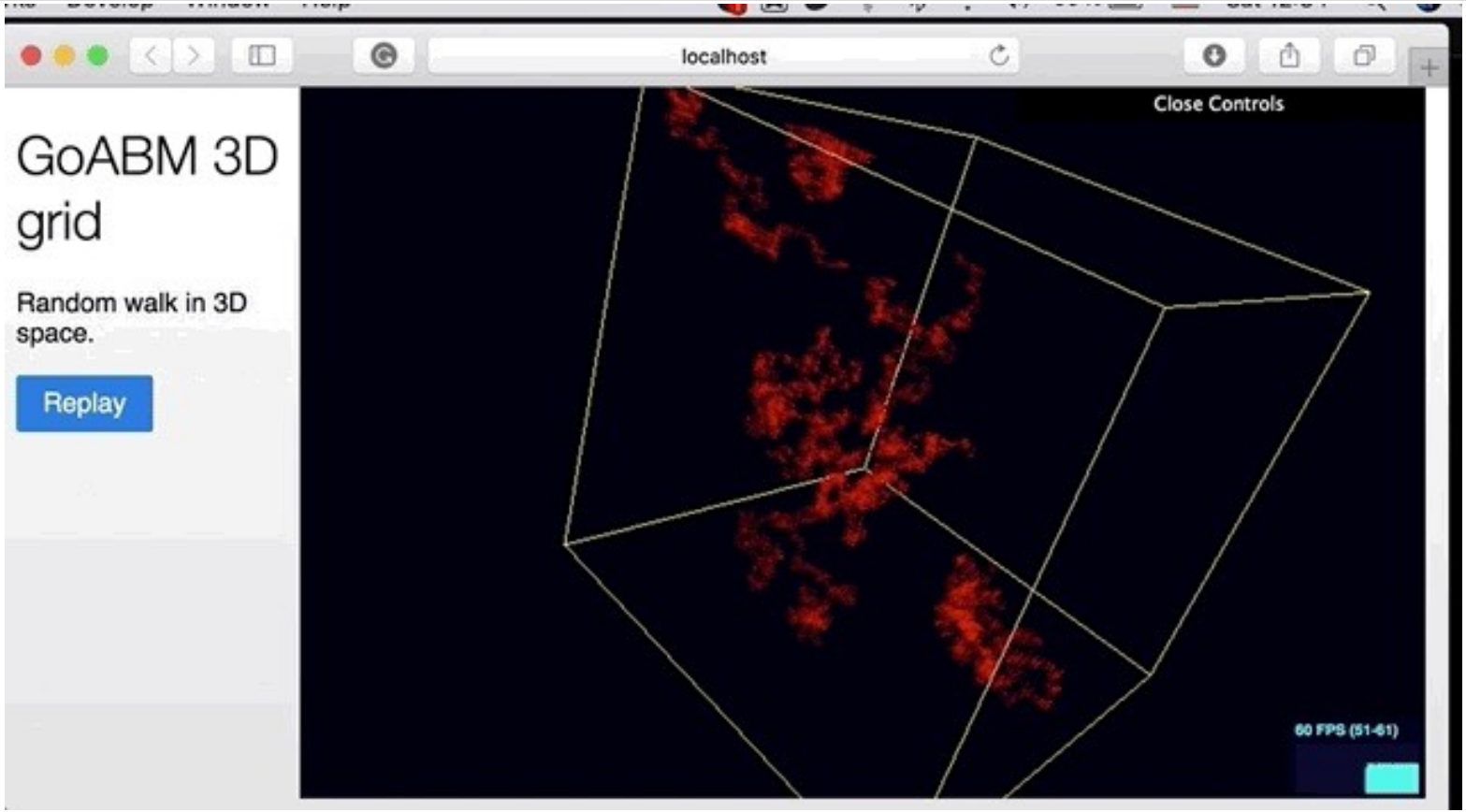
Random Walk

```
func (w *Walker) Run() {
  rx := rand.Intn(6)
  oldx, oldy, oldz := w.x, w.y, w.z
  switch rx {
  case 0:
     w.x++
  case 1:
     w.y++
  case 2:
     w.x--
  case 3:
     w.y--
  case 4:
     w.z++
  case 5:
     w.z--
  }
  err := w.abm.World().(*grid.Grid).Copy(oldx, oldy, oldz, w.x, w.y, w.z)
  if err != nil {
     w.x, w.y, w.z = oldx, oldy, oldz
  }
}
```

Random Walk

```
func main() {
 rand.Seed(time.Now().UnixNano())
 a := abm.New()
 w, h, d := 100, 100, 100
 g := grid.New(w, h, d)
 a.SetWorld(g)

 for i := 0; i < 10; i++ {
    cell := NewWalker(a, rand.Intn(w),
rand.Intn(h), rand.Intn(d))
    a.AddAgent(cell)
    g.SetCell(cell.x, cell.y, cell.z,
cell)
 }

 a.LimitIterations(10000)

 ch := make(chan []interface{})
 a.SetReportFunc(func(a *abm.ABM) {
    ch <- g.Dump(func(a abm.Agent) bool {
return a != nil })
 })

 go func() {
    time.Sleep(1 * time.Second)
    a.StartSimulation()
    close(ch)
 }()

 ui3d := ui.New(w, h, d)
 defer ui3d.Stop()
 ui3d.AddGrid3D(ch)
 ui3d.Loop()
}
```

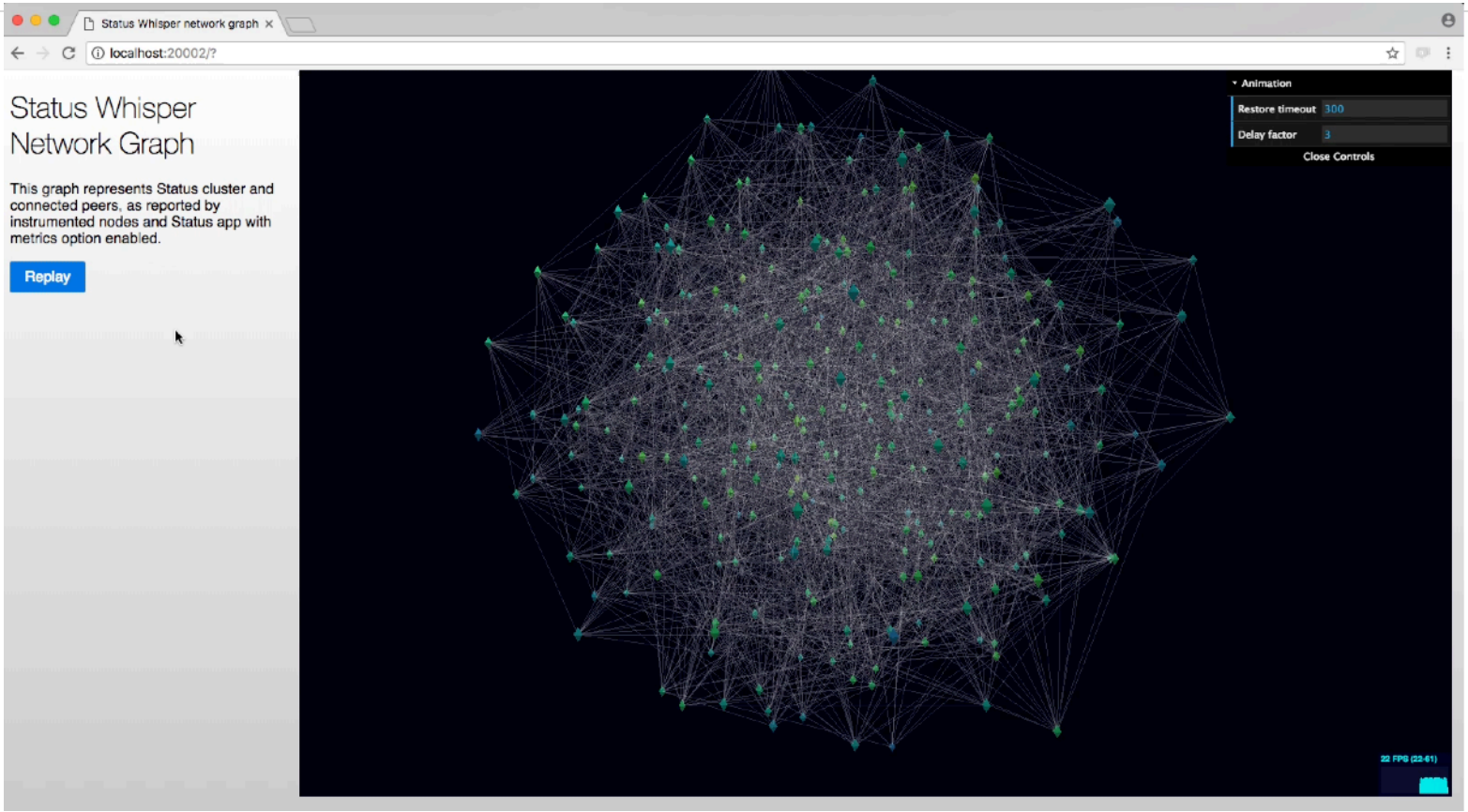
Random Walk


localhost
GoABM 3D grid
Random walk in 3D space.
Replay
Close Controls
60 FPS (51-61)

Bonus: wrap existing Go software into Agent interface.

Example: simulation of Whisper P2P messaging protocol using go-ethereum codebase

Whisper simulation



Source: https://www.youtube.com/watch?v=lt54a1ELUwE

Conclusions

Conclusions

Model Language

- A good fit for models language (simple, readable, easy to learn)
- Allows reusing existing software (e.g. networking simulations)
- Testable (you can write state-of-the-art tests for your models)

Conclusions

Performance

- Nothing on CPU can beat GPU-optimized simulations
- Currently not optimized at all and gives decent performance on classic models
- Huge space for further optimization
- Good flexibility in what can be optimized on framework side vs user model side

Conclusions

UI

- No out of the box high-level UI solutions
- But many options available (console, native desktop, Canvas, WebGL based)
- Framework can provide UIs for most common cases
- Truly awesome flexibility with UIs (switch between console to WebGL with one line of code)

goabm is pretty much work-in-progress/proof-of-concept pet project

may be a good fit to develop models before running large-scale simulation

will be open sourced soon

Thank you

@idanyliuk