Swarm Sugar Scape as a Starting Point

Paul E. Johnson

Dept of Political Science

University of Kansas

Prepared for Swarmfest 2005
Torino, Italy

Plan for SSS Tutorial

- Modified Suzuki method:
  - Run/Study examples
  - Tinker with examples
  - Hope theory/framework will percolate up
- Goal is to develop appreciation for model building from the “bottom up”.

Presentation linked to Handouts

- file-line numbered handouts are cited in this presentation

sss-2.2-20050604-Handouts.tar.gz

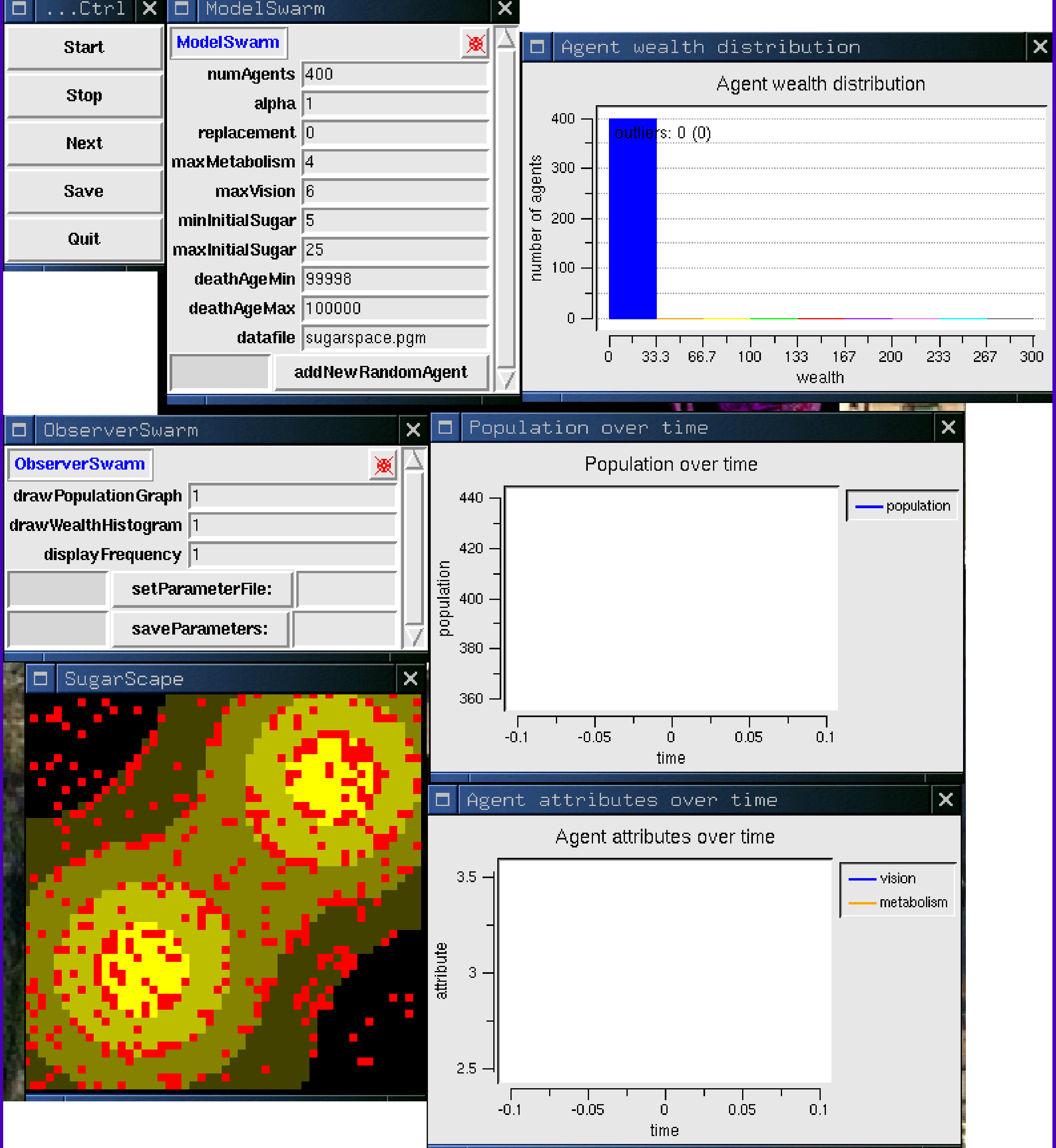
- Original Sugarscape model presented in famous book by Epstein & Axtell, *Growing Artificial Societies* (MIT Press, 1996)

Begin with Swarm Sugarscape

- Step by step [0-sss-shell1.txt]
  - Download a “tarball” [0-001]
  - Unpack it with tar and gzip [0-013]
  - Compile with: [0-046]

    make
  - Run with:

    ./sss [0-127]
  - see “screenshots” sss-1.png, sss-2.png

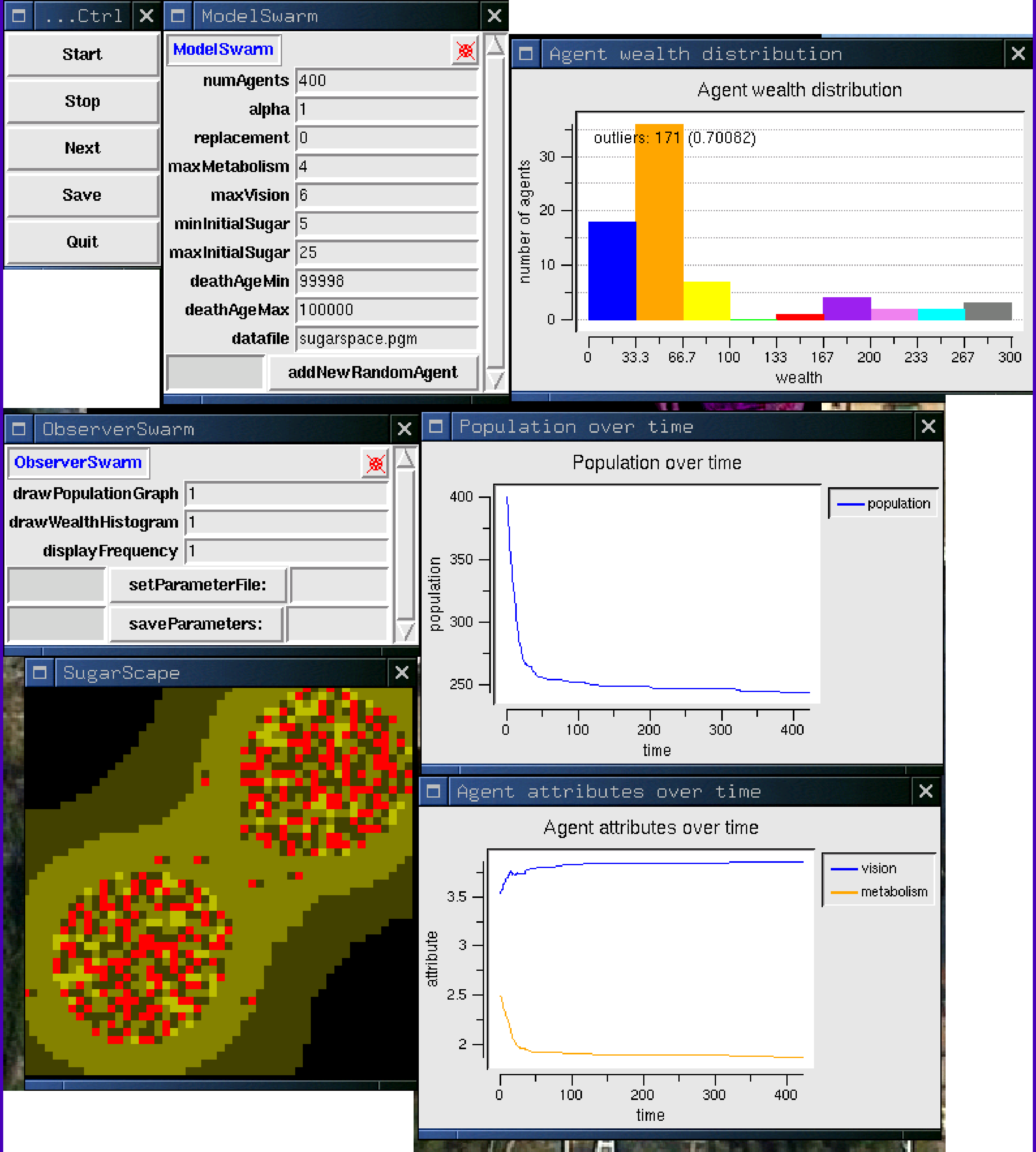


ModelSwarm

numAgents	400
alpha	1
replacement	0
maxMetabolism	4
maxVision	6
minInitialSugar	5
maxInitialSugar	25
deathAgeMin	99998
deathAgeMax	100000
datafile	sugarspace.pgm

ObserverSwarm

drawPopulationGraph	1
drawWealthHistogram	1
displayFrequency	1



ModelSwarm

numAgents	400
alpha	1
replacement	0
maxMetabolism	4
maxVision	6
minInitialSugar	5
maxInitialSugar	25
deathAgeMin	99998
deathAgeMax	100000
datafile	sugarspace.pgm

addNewRandomAgent

ObserverSwarm

drawPopulationGraph	1
drawWealthHistogram	1
displayFrequency	1

setParameterFile:

saveParameters:

Study File Layout

- Paired “*h” and “*m” files
- “h” is a “header file” containing:
  - A class declaration
  - A list of *variables*

    (aka: instance variables, IVARS)
  - A list of *methods*
- “m” is the “implementation file”, where the methods are fully written out.

Study File Layout #2

- Makefile [11]
  - used by “make” program to manage compiling (compile=convert text into an executable program)
- README [1]
  - comments from authors

Study File Layout #3

- main.m [2]
  - only file actually required in order to have a program because it has the “main” function in it.
  - The function “main” [2-011] is the one that the system runs when you start the program. It orchestrates everything else.
  - // means “comment”
    - same as /* comment */
  - If this were written in C, it would be “main.c”

What does main do?

- initSwarm() [2-015]
  - a big, multi-purpose function called from the Swarm library. Does much work behind the scenes.
- create observerSwarm [2-017]
- which is then told to: [2-019]
  - buildObjects (create “things” for the simulation)
  - buildActions (scheduling framework)
  - activateIn (places observer's schedule into context)

Deciphering sss: What is your mission?

When studying a model, remember that every model must have

- agents who can:
  - do “stuff”
  - remember information
  - “find” each other and/or environment and place self in the “environment”
- a way to observe/measure events in the model

sss easy to decipher

- SugarAgent class: individual agents are *instances* of this class
- Examine SugarAgent.h to see what kinds of messages the SugarAgent can respond to:
  - move about
  - live & die: take sugar, metabolize sugar
  - “get” info on status (for observational purposes)
  - drawSelf on the indicated “raster”

Little Wrinkles in Sugar Agent

- x,y declared as public [3-025]
  - allows other objects who are in contact with a sugar agent to "directly read" the agent's location with this syntax:

```
x_coord = agent->x;
y_coord = agent->y;
```

  - relatively rarely used in Swarm models because it ignores "encapsulation"
- could instead add getX and getY methods for SugarAgents

Little Wrinkles in Sugar Agent #2

- Note SugarAgents don't directly kill themselves [4-034]
- They ask the modelSwarm to kill them
- That's
  - not intuitive
  - method to avoid runtime crashes
  - allowing "recycling" of objects

SugarSpace

- SugarSpace is a “family” of grids
  - agentGrid: lattice of “hangers” where agents place themselves [5-033,6-082]
  - sugar: a lattice of integer values [5-026,6-032]
  - sugarMax: a lattice of maximum allowed sugar values[5-030,6-040]
- Agents repeatedly ask SugarSpace to tell them if (x,y) is
  - occupied [5-060]
  - full of sugar [5-051]

Sugar Agents don't directly interact

- - step { };   [4-015]
  - find best open spot
  - go there, take the sugar
  - calculate metabolism
  - consider dying (or not!)
- Interaction is indirect, via
  - search for open spaces and
  - values in the sugar grid.

Little Wrinkles #1

- The world is a flat square
- Agents should have to worry about stepping off edge [0, xsize-1] x [0, ysize-1]
- Agents don't worry, however.
- The SugarSpace worries for them. It translates all requests for information about (x,y) to be “in bounds”.
- Work done by “xnorm:” and “ynorm:”[5-070]

Little Wrinkles #2

- How to initialize sugar values?
  - text file: sugarspace.pgm [6-038]
  - hdf5 file: sss.hdf [6-050]
- As README explains, user can choose which format by a C compiler flag [1-084]

Little Wrinkles #3

- See how agents move in the SugarSpace?
- Agent tells the space it wants to move to (1,1)

  [sugarSpace move: self toX: 1 Y: 1];
- Watch what the “moveAgent:toX:Y:” method in SugarSpace does: [6-177]
  - figures out where agent is now
  - puts “nil” on agent's current position
  - adds agent at desired position

Hierarchy

- Swarm conceptualized as a “bottom up” modeling system
- Agents are lowest level, most “autonomous” elements
- ModelSwarm is “intermediate level”
  - causes agents to be created
  - causes environment to be created
  - makes agents aware of environment
  - schedules agent & environment actions

Frequently used method names

Optional but recommended:

- buildObjects;
- buildActions;

Mandatory!

- activateIn:

Special Items Worth Noting

- Agents are created and stored into a "linked list" object:

```
agentList = [List create: self];
```

- Could create & add agents:

```
agent = [self addNewRandomAgent];
[agentList addLast: agent];
```

- -addNewRandomAgent creates agents and puts them into the agentGrid in SugarSpace

Wrinkle: Overwrite Warnings

- Swarm's Grid2d can hold one object per cell.
- If one tries to add a second object to a cell, the cell “loses” the first and issues a warning to the programmer
- addNewRandomAgent turns off warnings to place agents
- harmless?

buildActions

- ActionGroup: things that should happen in a particular order
- Schedule: object that can link future times with collections and messages (abstract enough?)
- sss has “modelActions”, an ActionGroup
- Put modelActions into the modelSchedule
- activateIn: method ties modelSchedule into “global time sequence”, meshing with observer.

About Selectors

- Difficult concept!
- Its a “symbolic handle” for a method that an agent can carry out
- Needed in Swarm because of Activity framework.
- Associate objects with selectors to schedule future events.
- Integral part of “run-time” (dynamic) binding

Observer Swarm

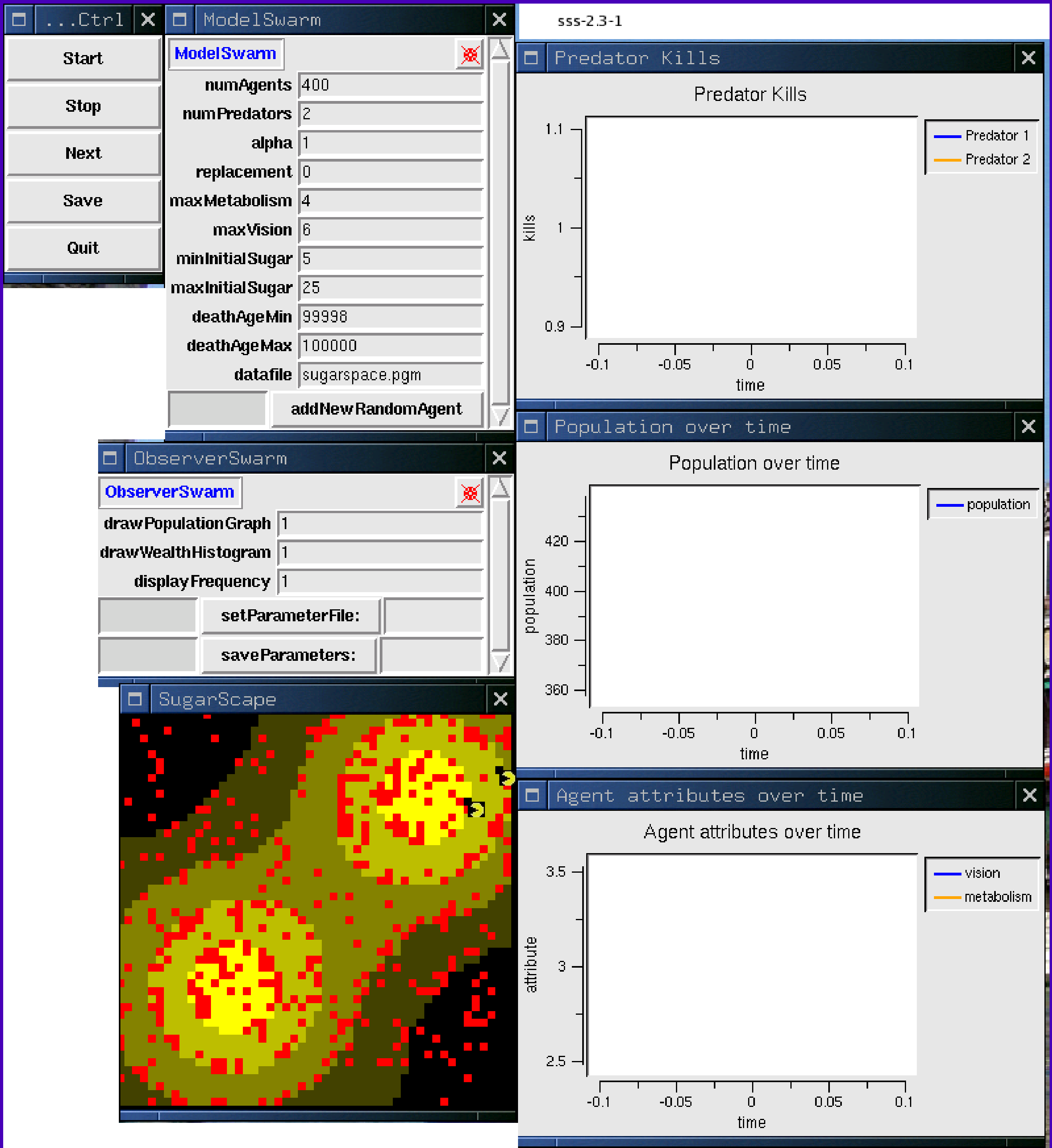
- Controls the graphical interface
- Creates & advances displays
- Raster: grid of dots
  - agentDisplay uses Object2dDisplay tools to collect info from agents
  - setDisplayWidget: tells agentDisplay that, when it "displays", it should do so on the Raster
  - Raster does not show on screen until "drawSelf" is called.

Graphs: 3 step sequence

- EZGraph class can create graph window
- User must add sequences to be graphed
  - createSequence:withFeedFrom:andSelector:
  - createAverageSequence:...
  - createMovingAverageSequence:...
- Schedule must include a “step” command to update the graph

Integrate a Predator

- SugarAgents may be “killed” by agents from a Predator class
- Handout: Transition-2.2-to-2.3.txt
  - Output from diff program
  - + new lines
  - ! edited lines

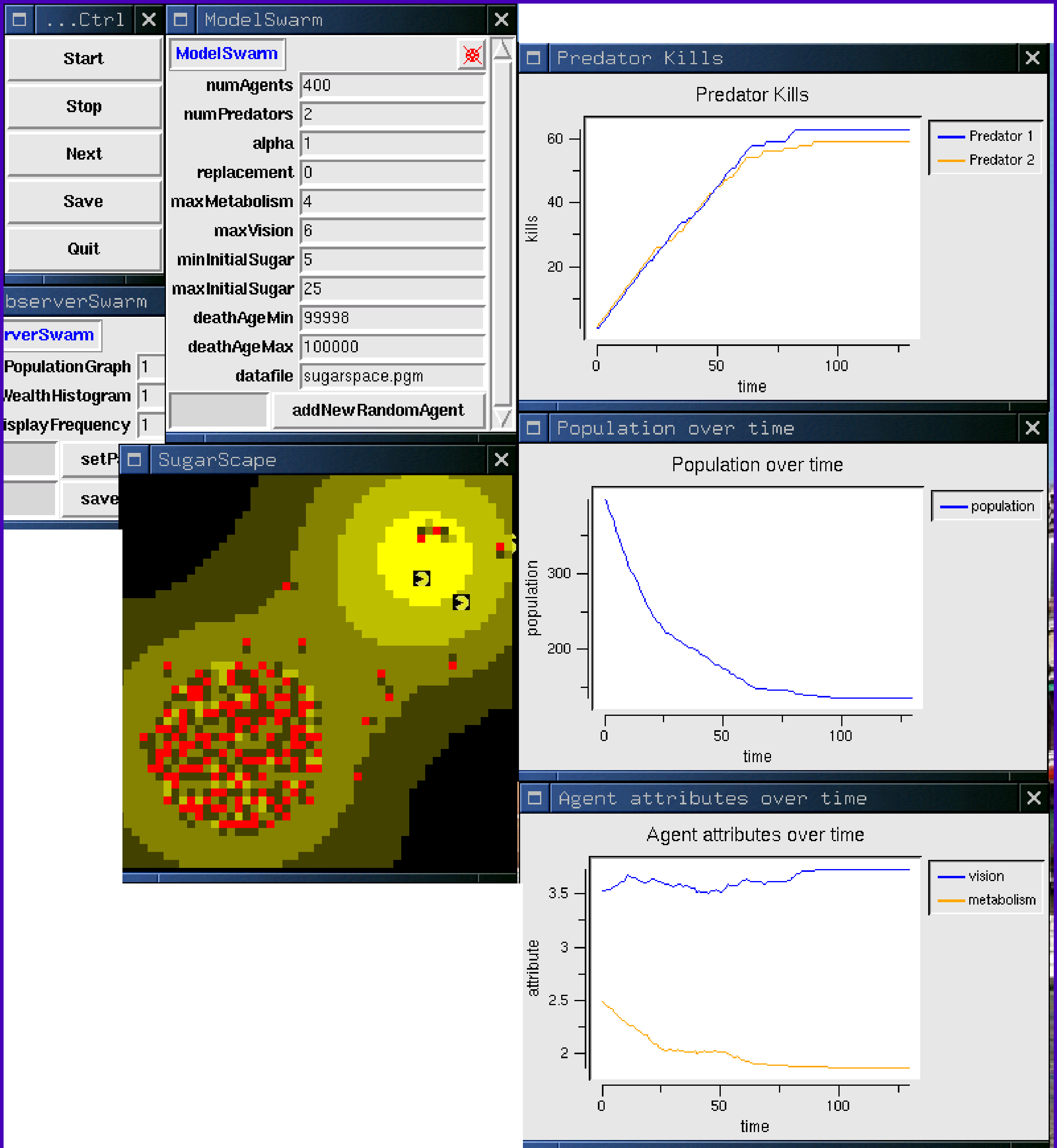


ModelSwarm

numAgents	400
numPredators	2
alpha	1
replacement	0
maxMetabolism	4
maxVision	6
minInitialSugar	5
maxInitialSugar	25
deathAgeMin	99998
deathAgeMax	100000
datafile	sugarspace.pgm

ObserverSwarm

drawPopulationGraph	1
drawWealthHistogram	1
displayFrequency	1



ModelSwarm

numAgents	400
numPredators	2
alpha	1
replacement	0
maxMetabolism	4
maxVision	6
minInitialSugar	5
maxInitialSugar	25
deathAgeMin	99998
deathAgeMax	100000
datafile	sugarspace.pgm

PopulationGraph	1
WealthHistogram	1
isplayFrequency	1

Easy (?) steps :)

- Add Predator.o to Makefile
- cp SugarAgent.[hm] Predator.[hm]
- Edit Predator.h
- Edit Predator.m
- Edit ModelSwarm to create Predators and schedule their actions.
- Edit SugarSpace to create predator grid
- Edit ObserverSwarm to draw predators

Predator Step method [12-132]

- rename moveToBestOpenSpot to “move”
- returns a “targetAgent” [12-166]
- Take that agent's sugar [12-139]
- Tell ModelSwarm and SugarSpace to slate that targetAgent for death [12-149]

Hunting SugarAgents

- Predator is able to search in the agentGrid by asking SugarSpace for agents
- -move method scans “up” and “down”, never diagonal [12-184]
- agent with highest sugar value is taken.
- Caution: Predators move carelessly, possibly stepping on each other in predatorGrid.

Model Swarm

- new IVARS:
  (int)numPredators;   [12-446]
  id <List> predatorList   [12-464
- buildObjects adds new for loop creating predators   [12-538]
- new method called to create Predators
  -addPredator;   [12-575]

ObserverSwarm

- predatorDisplay: tracks positions of predators [12-666]
- Note RASTER showing predator positions with “pixmaps”
- Right/Left button selects agent-types
- New killGraph of predators [12-702]

Laments

- No BatchSwarm class
- No command-line option processing
- No data output that would support Batch Swarm runs
- Separate Parameters class would make work much easier