11.4 Modeling an "Able" Invader—the "Cane" Toad

AgentSheets Quick Review Questions

Introduction to Computational Science:
Modeling and Simulation for the Sciences, 2nd Edition
Angela B. Shiflet and George W. Shiflet
Wofford College

***Compose all the following answers in AgentSheets*:**

Phase 0: Initialization

Quick Review Question 1 Write the method *createToads*.

Quick Review Question 2 Write the *WHEN-CREATING-NEW-AGENT* method for initialization of a *Toad* agent.

Quick Review Question 3 Suppose *PERCENT_AWPS* is 0.3, *PERCENT_AWPS_FENCED* is 25, and *PERCENT_MOIST_AREAS* is 0.1 and the grid is 100-by-40 cells. On the average, after the initialization phase how many of the following would we expect on the grid:

- **a.** *Awp* agents before initialization of *FencedAwp* agents
- **b.** *FencedAwp* agents
- **c.** *Awp* agents after initialization of *FencedAwp* agents
- **d.** *MoistArea* agents
- **e.** If there are 5 *Awp*, 2 *FencedAwp*, and 3 *MoistArea* agents, none of which are next to a border or each other, how many *AwpAdjacent* agents are there?

Quick Review Question 4 Write the following *Desert* methods for completion of the landscape:

- **a.** *placeAwps*
- **b.** *placeFencedAwps*
- **c.** *initAwp*
- **d.** *initAwp2*

Phase 1: Consumption

Quick Review Question 5 Suppose *AMT_EAT* = 0.01 and *FRACTION_WATER* = 0.6. Assume a toad is on top of a desert cell. Give the values of a toad's *energy* and

water and a desert cell's *food* after execution of *eat* and *updateFood* for each of the following situations:

a. *energy* = 0.9, *water* = 0.8, and *availableFood* = 0.03
b. *energy* = 0.9, *water* = 0.8, and *availableFood* = 0.005
c. *energy* = 0.999, *water* = 0.8, and *availableFood* = 0.03
d. *energy* = 0.9, *water* = 0.999, and *availableFood* = 0.03

Quick Review Question 6 Write the following consumption methods:

a. *Toad* method *toadMayEat*
b. *Toad* method *eat*
c. *Desert* method *updateFood*
d. *Toad* method *toadMayDrink*
e. *Toad* method *drink*

Phase 2: Movement

Quick Review Question 7 Write the toad method *toadMove*.

Quick Review Question 8 Write the following functions related to movement for moisture:

a. *thirsty*
b. *stayHere* and *here*
c. *lookForMoisture*
d. *moveW*
e. *useWaterEnergyHopping*

Quick Review Question 9 Write the *Toad* method *lookForFood*.

Quick Review Question 10 Write the following *Toad* methods related to hopping for fun:

a. *hopForFun*
b. *hopHere*

Phase 3: Complete Cycle

Quick Review Question 11 Write the following functions:

a. *changeCounts*
b. *checkTerminate*

Answers to Quick Review Questions

1.

ON (createToads)

Create toad with probability INIT_PERCENT_TOADS/100

Number of rules: 1

If

```
SEE (▪ , ▪), and
%-CHANCE (@INIT_PERCENT_TOADS)
```

Then

```
NEW (▪ , ■)
```

2.

▾WHEN-CREATING-NEW-AGENT ()

'food' and 'moisture' set to -1 to avoid more than 1 toad in same position

Number of rules: 1

If

```
SEE (▪ , ■)
```

Then

```
SET (@numAlive , to , @numAlive + 1), and
SET (energy , to , @AMT_MIN_INIT + random(@INIT_RANGE)), and
SET (water , to , @AMT_MIN_INIT + random(@INIT_RANGE)), and
SET (prevDir , to , 0), and
SET (numTimeSteps , to , 1), and
SET (availableFood , to , -1), and
SET (availableMoisture , to , -1), and
SET (food , to , -1), and
SET (moisture , to , -1)
```

3.

a. 12 = (0.003)(100)(40)

b. 3 = (0.25)(12), where 12 is obtained from Part a

c. 9 = 12 - 3

d. A little less than 4 = (0.001)(100)(40), because immediately before initialization of moist areas, some of the (100)(40) = 4000 *Desert* agents have likely been converted to *Awp* and/or *FencedAwp* agents

e. 56 = (8)(5 + 2) because each *Awp* and *FencedAwp* agent is surrounded by 8 *AwpAdjacent* agents.

4. a.

▾ON (placeAwps)

Place AWPs, fenced AWPs, and moist areas on desert. Chance of changing a Desert agent to an Awp agent is PERCENT_AWPS/100. Chance of changing a Desert agent to a MoistArea agent is PERCENT_MOIST_AREAS/100.

Number of rules: 2

If
SEE (▪ , ▒), and
SEE (➡ , ▒), and
%-CHANCE (@PERCENT_AWPS)
Then
ERASE (▪), and
NEW (▪ , ■)

If
SEE (▪ , ▒), and
SEE (➡ , ▒), and
%-CHANCE (@PERCENT_MOIST_AREAS)
Then
NEW (▪ , ▒), and
ERASE (▪)

4. b.

▾ON (placeFencedAwps)

Change of changing an Awp agent to a FencedAwp is PERCENT_AWPS_FENCED/100.

Number of rules: 1

If
%-CHANCE (@PERCENT_AWPS_FENCED), and
SEE (▪ , ■)
Then
ERASE (▪), and
NEW (▪ , ■)

4. c.

ON (initAwp)

If current agent is a Desert agent that is staced below another desert agent (such as an Awp agent), erase the current agent. If The current agent is a Desert agent that is adjacent to an Awp or a FencedAwp agent, change the current agent to an AwpAdjacent agent.

Number of rules: 3

If
SEE (,), **and**
STACKED-A (immediately below , a , Desert)
Then
ERASE ( )

If
SEE (,), **and**
NEXT-TO (> , 0 ,)
Then
ERASE ( ), **and**
NEW (,)

If
SEE (,), **and**
NEXT-TO (> , 0 ,)
Then
ERASE ( ), **and**
NEW (,)

4. d.

ON (initAwp2)

If The current agent is a Desert agent that is adjacent to an AwpAdjacent agent, change the current agent to an AwpOver2 agent.

Number of rules: 1

If
SEE (,), **and**
NEXT-TO (> , 0 ,)
Then
ERASE ( ), **and**
NEW (,)

5. a. *energy* = 0.91, *water* = 0.806, and *food* = 0.02 because *amtEat* = 0.01, so *energy* = 0.9 + 0.01, *water* = 0.8 + 0.6*0.01, and *food* = 0.03 - 0.01

b. *energy* = 0.905, *water* = 0.803, and *food* = 0.0 because *amtEat* = *availableFood* = 0.005, so *energy* = 0.9 + 0.005, *water* = 0.8 + 0.6*0.005, and *food*= 0.005 - 0.005

c. *energy* = 1.0, *water* = 0.8006, and *food* = 0.029 because *amtEat* = 1 - *energy* = 0.001, so *energy* = 0.9 + 0.001, *water* = 0.8 + 0.6*0.001, and *food* = 0.03 - 0.001

d. *energy* = 0.91, *water* = 1.0, and *food* = 0.02 because *amtEat* = 0.01, so *energy* = 0.9 + 0.01, *water* = the minimum of 0.999 + 0.6*0.01 = 1.005 and 1.0, and *food* = 0.03 - 0.01

6. a.

ON (toadMayEat)

Toad behavior regarding eating

Number of rules: 3

If
```
STACKED-A (somewhere above , a , Desert), and
IS (energy , < , @WOULD_LIKE_EAT)
```
Then
```
MAKE ( • , eat)
```

If
```
STACKED-A (somewhere above , a , Desert), and
IS (water , < , @WOULD_LIKE_DRINK), and
IS (random(1.0) , < , @MAY_EAT)
```
Then
```
MAKE ( • , eat)
```

If
```
no condition
```
Then
```
SET (amtEat , to , 0)
```

6. b.

ON (eat)

Function to update a toad's energy and water after it eats

Number of rules: 1

If
```
no condition
```
Then
```
MAKE ( • , calcAmtEat), and
SET (energy , to , energy + amtEat), and
MAKE ( • , waterFromFood)
```

ON (calcAmtEat)

Calculate amtEat the minimum of @AMT_EAT, food, and 1 - energy.

Number of rules: 1

If
```
no condition
```
Then
```
SET (oneMinusEnergy , to , 1-energy), and
SET (minAMTEATAvail , to , min(@AMT_EAT,availableFood)), and
SET (amtEat , to , min(oneMinusEnergy,minAMTEATAvail))
```

ON (waterFromFood)

Update toad's water obtained from food

Number of rules: 2

If
IS (water + @FRACTION_WATER * amtEat , < , 1)
Then
SET (water , to , water + @FRACTION_WATER * amtEat)

If
no condition
Then
SET (water , to , 1)

6. c.

ON (updateFood)

Update the food for a desert agent after a toad on top of it has eaten.

Number of rules: 4

If
SEE (,), **and**
STACKED-A (immediately below , a , Toad)
Then
SET (food , to , food - amtEat[top]), **and**
MAP (food , to Color , between , 15263976 , for , 0 , and , 7368816 , for , @FOOD_CELL)

If
SEE (,), **and**
STACKED-A (immediately above , a , Toad)
Then
SET (food , to , food - amtEat[top]), **and**
MAP (food , to Color , between , 15263976 , for , 0 , and , 7368816 , for , @FOOD_CELL)

If
SEE-A (, Desert), **and**
STACKED-A (immediately below , a , Toad)
Then
SET (food , to , food - amtEat[top])

If
SEE-A (, Desert), **and**
STACKED-A (immediately above , a , Toad)
Then
SET (food , to , food - amtEat[top])

6. d.

ON (toadMayDrink)

Toad behavior regarding drinlomg

Number of rules: 1

If
IS (water , < , @WOULD_LIKE_DRINK), **and**
STACKED (somewhere above ,)
Then
MAKE (, drink)

6. **e.**

ON (drink)

Function to update a toad's water after it drinks

Number of rules: 2

If
IS (water + @AMT_DRINK , <= , 1)
Then
SET (water , to , water + @AMT_DRINK)

If
no condition
Then
SET (water , to , 1)

7.

ON (toadMove)

Possibly move the toad

Number of rules: 4

If
IS (water , < , @WOULD_LIKE_DRINK)
Then
MAKE (• , thirsty)

If
IS (energy , < , @WOULD_LIKE_EAT)
Then
MAKE (• , lookForFood)

If
IS (random(1.0) , < , @MAY_HOP)
Then
MAKE (• , hopForFun)

If
no condition
Then
MAKE (• , stayHere)

8. a.

ON (thirsty)

Function to change the position of a very thirsty toad

Number of rules: 4

If
STACKED (somewhere above , ■)
Then
MAKE (▪ , stayHere)

If
STACKED-A (somewhere above , a , Desert)
Then
SET (maxMoisture , to , max(moisture[up],max(moisture[right],max(moisture[down],moisture[left])))), **and**
MAKE (▪ , lookForMoisture)

If
STACKED (somewhere above , ▨), **and**
SEE-A (⬅ , Desert)
Then
MAKE (▪ , moveW)

If
STACKED (somewhere above , ▨)
Then
MAKE (▪ , stayHere)

8. b.

ON (stayHere)

Procedure for toad to stay in current location

Number of rules: 1

If
no condition
Then
MAKE (▪ , here), **and**
MAKE (▪ , useWaterEnergySitting)

ON (here)

Function to update a toad's state when staying in the current location

Number of rules: 1

If
no condition
Then
SET (availableFood , to , food[bottom]), **and**
SET (availableMoisture , to , moisture[bottom])

8. c.

ON (lookForMoisture)

Function to move toad towards maximum moisture when toad was at current location at last time step. Called by lookForMoisture.

Number of rules: 2

If
IS (maxMoisture , < , 0)
Then
MAKE (▪ , stayHere)

If
no condition
Then
HILLCLIMB (moisture , in , Four Directions (Von Neumann neighborhood)), **and**
MAKE (▪ , here), **and**
MAKE (▪ , useWaterEnergyHopping)

8. d.

ON (moveW)

Function to update a toad's state when moving to the west

Number of rules: 1

If
no condition
Then
MOVE (←), **and**
MAKE (▪ , here), **and**
MAKE (▪ , useWaterEnergyHopping)

8. e.

ON (useWaterEnergyHopping)

Procedure to reduce a toad's energy and water after hopping

Number of rules: 3

If
STACKED (immediately above , ▩)
Then
SET (energy , to , energy - @ENERGY_HOPPING), **and**
SET (water , to , water - @WATER_HOPPING)

If
STACKED (immediately above , ■)
Then
SET (energy , to , energy - @ENERGY_HOPPING), **and**
SET (water , to , water - @WATER_HOPPING)

If
no condition
Then
SET (energy , to , energy - @ENERGY_HOPPING)

9.

ON (lookForFood)

Function to have a toad go to/remain at a vacant neighborhood location with the maximum amount of food

Number of rules: 3

If
STACKED (somewhere above , ), **and**
SEE-A ( , Desert)
Then
MAKE ( , moveW)

If
STACKED (somewhere above , )
Then
MAKE ( , stayHere)

If
no condition
Then
SET (maxFood , to , max(availableFood,max(food[up],max(food[right],max(food[down],food[left]))))), **and**
MAKE ( , goToFood)

ON (goToFood)

Function to update a toad's location to hop in a random "legal" direction with the most amount of food

Number of rules: 2

If
IS (availableFood , = , maxFood)
Then
MAKE ( , stayHere)

If
no condition
Then
HILLCLIMB (food , in , Four Directions (Von Neumann neighborhood)), **and**
MAKE ( , here), **and**
MAKE ( , useWaterEnergyHopping)

10. a.

ON (hopForFun)

Function to update a toad's location to hop in a random "legal" direction if possible

Number of rules: 4

If
STACKED (somewhere above , ), **and**
SEE-A (, Desert)
Then
MAKE ( , moveW)

If
STACKED (somewhere above , )
Then
MAKE ( , stayHere)

If
NEXT-TO (> , 0 , )
Then
MOVE-RANDOM-ON (), **and**
MAKE ( , hopHere)

If
no condition
Then
MAKE ( , stayHere)

10. b.

ON (hopHere)

Procedure when toad hops to current location

Number of rules: 1

If
no condition
Then
MAKE ( , here), **and**
MAKE (, useWaterEnergyHopping)

11. a.

ON (changeCounts)

Method to eliminate a toad that should be dead or migrated

Number of rules: 3

If
IS (water , < , @DESICCATE)
Then
ERASE ( ), **and**
SET (@numAlive , to , @numAlive - 1), **and**
SET (@numDead , to , @numDead + 1)

If
IS (energy , < , @STARVE)
Then
ERASE ( ), **and**
SET (@numAlive , to , @numAlive - 1), **and**
SET (@numDead , to , @numDead + 1)

If
STACKED (somewhere above ,)
Then
ERASE ( ), **and**
SET (@numAlive , to , @numAlive - 1), **and**
SET (@numMigrated , to , @numMigrated + 1)

11. b.

ON (checkTerminate)

Check if the simulation should terminate because no more toads exist on the grid

Number of rules: 1

If
IS (@numAlive , = , 0)
Then
STOP-SIMULATION ()