**Appendix**

**Learning Hunting behavior in cubs through Observation, Imitation and Reinforcement Learning**

Pragadheeshwaran Thirumurthi Prarthana G Alevoor

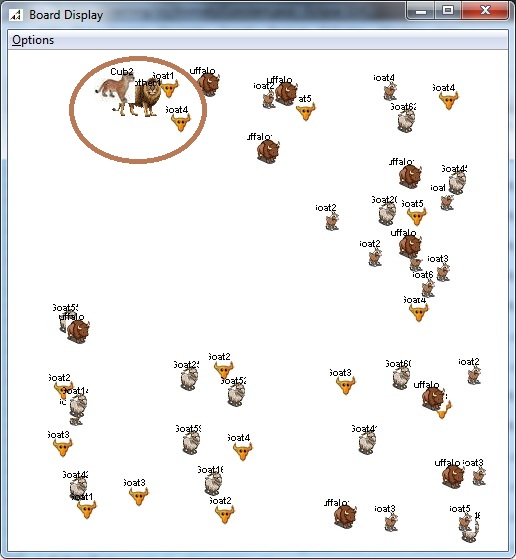
Department of Computer Science

{pragadh, prar1986}@cs.ucla.edu

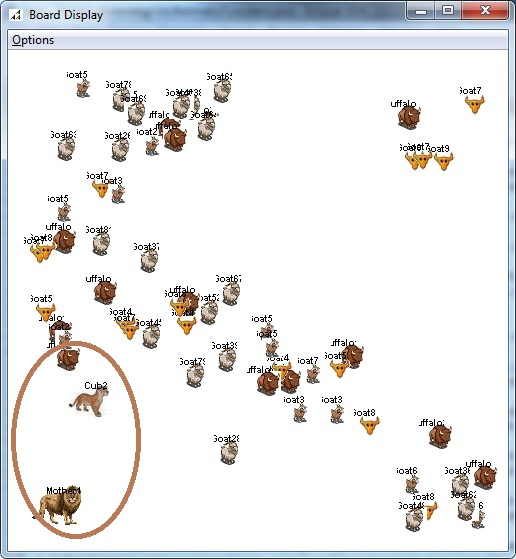
Professor: Michael Dyer {dyer@cs.ucla.edu}

**A. Results / Screenshots:**

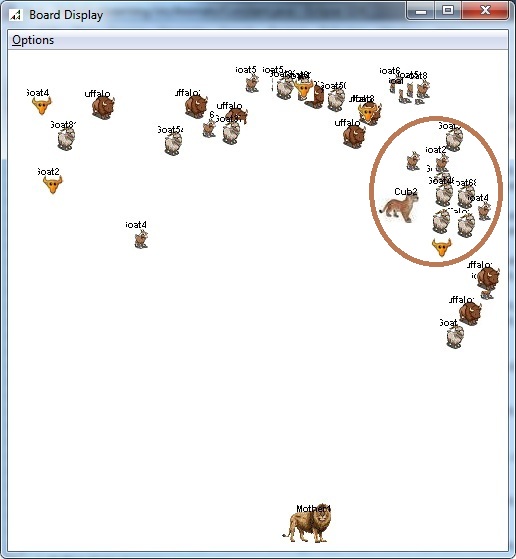
The following section contains screen shots of various scenarios that are encountered during our predator hunting.

**Phase 1: Cub observer the mother’s hunting behavior and builds its neural network**

**Phase 2: Mother observes the cub performing the hunt and gives reinforcement to the cub**



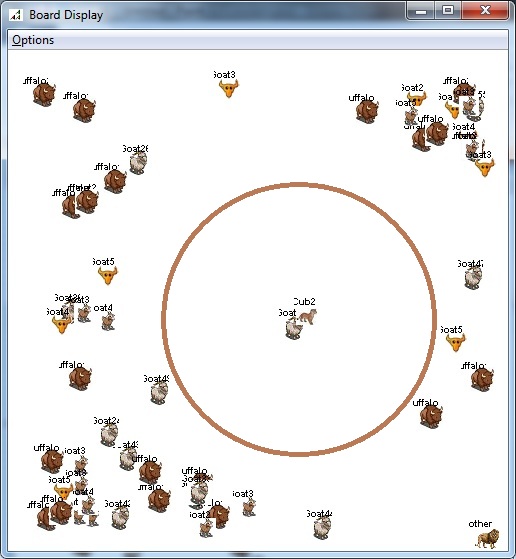
**Phase 3: Hunting Group of Prey – Self Learning**



If the prey is present in a group, cub drawbacks from hunting.

**Sense Range:**

As each prey has a sense range, they tend to move away from the cub’s sense range. This is observed in the following screenshot.



**B. Rules and Actions:**

**ACTIONS**

**CONDITIONS**

75 >= Distance >50

Size == Large

Distance >= 50

50 >= Distance >= 25

Size == Medium

25 > Distance >10

PrevAction<= Snarl

10 > Distance

Energy > 10

10 > Distance

Energy <= 10

If Mother Lion

&& Preys Around

If Cub && Preys Around

Action (Roar)

Action (Claw)

Action (Run)

Action (Snarl)

Action (Pounce)

Action (Drawback)

Action (Growl)

Action (Eat)

**C. Sample Output Run:**

**Case 1:**

In the following output snippet, the Cub performs the following actions for successful hunt of a Goat

* + Search – 0.0
  + Pounce – 0.4
  + ClawChase – 0.5
  + Claw – 0.6
  + Eat – 0.7

Animat to Be Hunt: Goat; Distance: 0.12666352012619664; Energy: 100; Size: 15; Previous Action: 0.0; Action To Be Taken0.0

Animat to Be Hunt: Goat; Distance: 0.12007815063537368; Energy: 100; Size: 15; Previous Action: 0.4; Action To Be Taken0.4

Animat to Be Hunt: Goat; Distance: 0.12007815063537368; Energy: 100; Size: 15; Previous Action: 0.5; Action To Be Taken0.5

Animat to Be Hunt: Goat; Distance: 0.07763814586933462; Energy: 55; Size: 15; Previous Action: 0.6; Action To Be Taken0.6

Animat to Be Hunt: Goat; Distance: 0.09063297913161555; Energy: 10; Size: 15; Previous Action: 0.7; Action To Be Taken0.7

**Case 2:**

In the following output snippet, the Cub performs the following actions for successful hunt of a Buffalo

* + Search – 0.0
  + Claw – 0.6
  + Eat – 0.7

Animat to Be Hunt: Buffalo; Distance: 0.12135639555005293; Energy: 100; Size: 15; Previous Action: 0.0; Action To Be Taken0.0

Animat to Be Hunt: Buffalo; Distance: 0.08340815083658593; Energy: 100; Size: 15; Previous Action: 0.6; Action To Be Taken0.6

Animat to Be Hunt: Buffalo; Distance: 0.07447633568272734; Energy: 55; Size: 15; Previous Action: 0.6; Action To Be Taken0.6

Animat to Be Hunt: Buffalo; Distance: 0.08340815083658593; Energy: 10; Size: 15; Previous Action: 0.7; Action To Be Taken0.7

**Case 3:**

In the following output snippet, the Cub performs the following actions for successful hunt of a Buffalo

* + Search – 0.0
  + Pounce – 0.4
  + ClawChase – 0.5
  + Claw – 0.6
  + Eat – 0.7

Animat to Be Hunt: Buffalo; Distance: 0.12450454844770316; Energy: 100; Size: 15; Previous Action: 0.0; Action To Be Taken0.0

Animat to Be Hunt: Buffalo; Distance: 0.10590615112585099; Energy: 100; Size: 15; Previous Action: 0.4; Action To Be Taken0.4

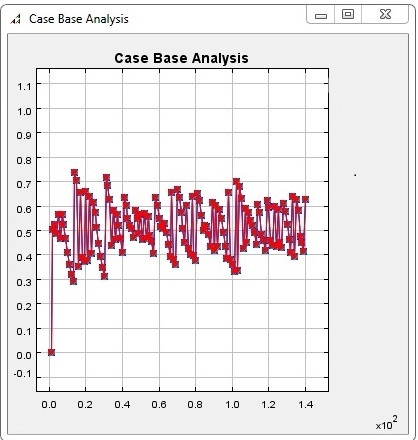
Animat to Be Hunt: Buffalo; Distance: 0.10590615112585099; Energy: 100; Size: 15; Previous Action: 0.5; Action To Be Taken0.5

Animat to Be Hunt: Buffalo; Distance: 0.064632855578925; Energy: 55; Size: 15; Previous Action: 0.6; Action To Be Taken0.6

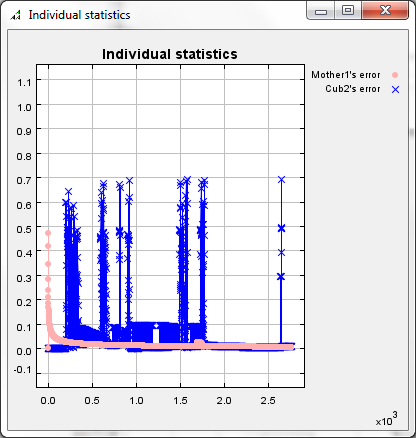
Animat to Be Hunt: Buffalo; Distance: 0.07659657174517254; Energy: 10; Size: 15; Previous Action: 0.7; Action To Be Taken0.7

**D. Experiments:**

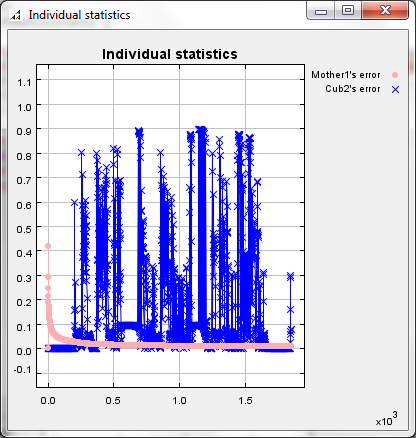
1. **Error rate analysis of the Cub based on Case-Base Model**



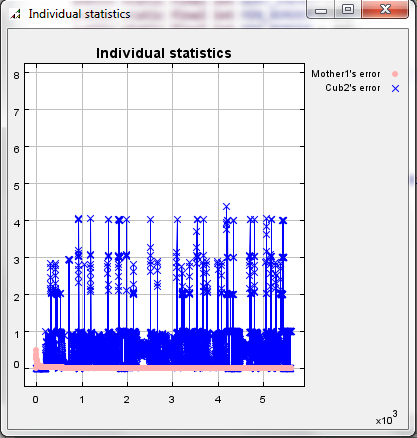
1. **Imitation Learning for Cub**



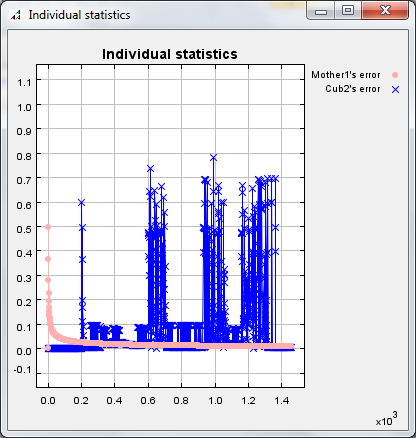
1. **Error Estimate for Cub**



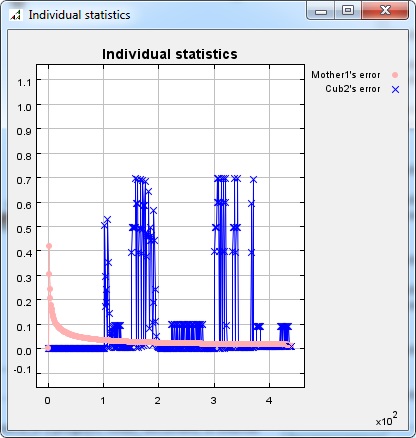
1. **Cub Imitation Learning**



1. **Better Cub Statistics**



1. **Overall Learning of the Cub**



As we could see the error probability of the cub finally converges to zero.

**E. Source Code:**

The source for various key functionalities are shown below.

**Algorithm:**

The algorithm to determine to direction of movement in a prey/predator is show below.

If Animat is a Prey,

If Predator is present in vicinity.

Find the location of the predator

Find directions away from the predator

If directions away from predator are available

Move in a direction away from the predator

If no direction is available (in case of surrounding walls/border)

Move in a random available direction

Move in a random available direction

Scan for nearby predators and continue the process

If Animat is a Predator,

If Prey is in vicinity

Find the direction in which the prey is located

Move in that direction approaching the prey

If no Prey is present, move in a random direction

Scan for nearby preys and continue the process

**Prey Movement:**

//If Animat is a Prey

if(objAnimatVisionSensorVisionSensor.animatType ==GOAT || objAnimatVisionSensorVisionSensor.animatType==BUFFALO)

{

//Scan for enemies in surrounding area

ArrayList<Animat> enemyList = isEnemyPresent(objAnimatVisionSensor);

// If no enemies are present, find a random direction to move and move in that direction

if(enemyList.isEmpty())

{

int direction=findRandomDirection(objAnimatVisionSensor);

moveAnimat(objAnimatVisionSensor, direction, Constant.SPEED);

}

//If Predator is present in vicinity

else

{

// Find available directions away from the predator and move in that direction

ArrayList<Integer> direction = findMoveDirection(objAnimatVisionSensor, enemyList);

int x=0,dir=0;

x = Random.uniform.nextIntFromTo(0,direction.size()-1);

direction = direction.get(x);

moveAnimat(objAnimatVisionSensor, dir, Constant.SPEED+1);

}

enemyList.clear();

}

**Predator Movement:**

// If animat is a predator

if(objAnimatVisionSensor.animatType==LION || objAnimatVisionSensor.animatType==CUB)

{

// Check if enemy animat is present

ArrayList<Animat> enemyList = isEnemyPresent(objAnimatVisionSensor);

// If no enemy is present move in a random direction

if(enemyList.isEmpty())

{

int direction=findRandomDirection(objAnimatVisionSensor);

moveAnimat(objAnimatVisionSensor, direction, Constant.SPEED);

}

// If prey present, choose direction of prey and move in that direction

else

{

ArrayList<Integer> direction = findMoveDirection(objAnimatVisionSensor,al);

int dir=0,x=0;

do

{

x = Random.uniform.nextIntFromTo(0,direction.size()-1);

direction = direction.get(x);

}while(dir==5);

moveAnimat(objAnimatVisionSensor, direction, Constant.SPEED+3);

}

enemyList.clear();

}

**Action Selection:**

// This function is used to determine the action a predator should perform based on various conditions

public double determineAction(Animat animatToBHunt, Animat animatHunting)

{

// The default action that would be performed is Search for Preys

double actionToBTaken=ActionConstants.SEARCH;

double distance =0;

// If any prey is present nearby

if(animatToBHunt!=null)

{

// Obtain distance of the prey from the predator

distance = getNormalizeddistanceance(animatHunting, animatToBHunt);

// The cub will drawback if there are more preys located closely

if(animatHunting.animatType == 3 && Lion.DRAWBACK)

actionToBTaken = ActionConstants.DRAWBACK;

// If the prey is large sized and not closer to the predator, the cub performs Roar action

else if(distance>.50 && distance<=.75 && animatToBHunt.size == Constant.largeSize) actionToBTaken = ActionConstants.ROAR;

// If the prey is not closer to the predator, the predator Runs towards the prey

else if(distance>=.50) actionToBTaken = ActionConstants.RUN;

// If the prey is medium sized and closer to the predator, the cub performs Snarl action

else if(distance<=.50 && distance>=.25 && animatToBHunt.size == Constant.mediumSize) actionToBTaken = ActionConstants.SNARL;

// If the prey is closer to the predator and if the previous action performed is Snarl, then the predator Pounces on the prey

else if(distance<.25 && distance>.10 && prevAction <= ActionConstants.SNARL) actionToBTaken = ActionConstants.POUNCE;

// If the energy of the energy of the prey is very less, the predator eats the prey

else if(distance<.10 && animatToBHunt.energy<=10) actionToBTaken = ActionConstants.EAT;

// If the prey is not weak enough, the predator claws the predator to decrease its energy level

else if(distance<.10 && animatToBHunt.energy>10) actionToBTaken = ActionConstants.CLAW;

if(animatHunting.animatType==4 && Lion.DRAWBACK)

actionToBTaken = ActionConstants.GROWL;

prevAction = actionToBTaken;

}

else

actionToBTaken = ActionConstants.SEARCH;

return actionToBTaken;

}

// The predator runs towards prey at a greater speed

// This decreases the energy of the prey considerably

public void run(Animat predator, Animat prey)

{

predator.energy -=4;

board.moveTowards(predator,prey);

}

// The predator moves in random to find available preys

public void search(Animat predator)

{

board.moveFarther(predator);

}

// Snarl action reduces the energy of the prey and also the predator closes on the prey

public void snarl(Animat predator, Animat prey)

{

predator.energy -=1;

prey.energy -= predator.energy/5;

board.moveTowards(predator, prey);

}

// Roar action reduces the energy of the prey and the predator moves towards the prey

public void roar(Animat predator, Animat prey)

{

predator.energy -=2;

prey.energy -=predator.energy/4;

board.moveTowards(predator, prey);

}

// Claw action considerably decreases the energy of the prey.

public void claw(Animat predator, Animat prey)

{

predator.energy -=2;

prey.energy -= predator.energy/3;

}

// Pounce actions moves the predator closer to the prey

public void pounce(Animat predator, Animat prey)

{

predator.energy -=3;

prey.energy -= predator.energy/2;

board.moveTowards(predator,prey);

}

// Eat action kills the prey and the prey is removed from the board. Energy of the prey is gained by they predator

public void eat(Animat predator, Animat prey)

{

prey.energy = 0;

if(prey.animatType == 1)

{

((Buffalo)prey).killBuffalo();

predator.energy += Constant.BUFFALO\_ENERGY;

}

else if(prey.animatType == 2)

{

((Goat)prey).killGoats();

predator.energy += Constant.GOAT\_ENERGY;

}

board.emptyAnimat();

}

// The cub will drawback and move far from the prey if there are more preys located closeby

public void drawback(Animat predator, Animat prey)

{

predator.energy -=5;

board.moveFarther(predator);

}

**Scan function:**

// Scan area for Lion

// Scan for nearby prey within its Range

public ArrayList<Animat> scanArea(int RANGE)

{

// HashMap is used to sort the preys based on distances and energy

// Closest prey with lowest energy would be choosen

HashMap<Double, Animat> hmRange = new HashMap<Double, Animat>();

ArrayList<Animat> alBuffalo = new ArrayList<Animat>();

ArrayList<Animat> alGoats = new ArrayList<Animat>();

if(!alRange.isEmpty())

alRange.clear();

// Go through the available goats to find the nearby goats

for (Iterator iter = board.getGoats().iterator(); iter.hasNext();)

{

Animat animat = (Animat) iter.next();

if(animat.getX()>=this.getX()-RANGE && animat.getX()<=this.getX()+ RANGE)

{

if(animat.getY()>=this.getY()-RANGE && animat.getY()<=this.getY()+ RANGE)

{

hmRange.put(getMetric(animat.getEnergy(), board.getDistance(this, animat)), animat);

alGoats.add(animat);

}

}

}

// Go through the available buffalo to find the nearby goats

for (Iterator iter = board.getBuffalos().iterator(); iter.hasNext();)

{

Animat animat = (Animat) iter.next();

if(animat.getX()>=this.getX()-RANGE && animat.getX()<=this.getX()+ RANGE)

{

if(animat.getY()>=this.getY()-RANGE && animat.getY()<=this.getY()+ RANGE)

{

hmRange.put(getMetric(animat.getEnergy(), board.getDistance(this, animat)), animat);

alBuffalo.add(animat);

}

}

}

// The values added to the hashmap are sorted based on the key (Distance & Energy)

// This gives the nearest prey with least energy

TreeSet<Double> keys = new TreeSet<Double>(hmRange.keySet());

for(double key: keys)

{

alRange.add(hmRange.get(key));

}

//To Check if the Prey are in a Group

if(this.animatType==3 && alGoats.size() >3)

{

DRAWBACK = true;

alRange.clear();

}

//To Check if the Prey are in a Group

if(this.animatType==3 && alBuffalo.size() >3)

{

DRAWBACK = true;

alRange.clear();

}

return alRange;

}

**Scan for animals near Buffalo**

public ArrayList<Animat> scanArea(int RANGE)

{

if(!alRange.isEmpty())

alRange.clear();

// Get nearby buffalos in range

for (Iterator iter = board.getBuffalos().iterator(); iter.hasNext();)

{

Buffalo buffalo= (Buffalo) iter.next();

if(buffalo.getX()>=this.getX()-RANGE && buffalo.getX()<=this.getX()+ RANGE)

{

if(buffalo.getY()>=this.getY()-RANGE && buffalo.getY()<=this.getY()+ RANGE)

alRange.add(buffalo);

}

}

//If more number of buffalos are present nearby, the buffolos would not be bothered about the presence of lion

if(alRange.size() >=5)

{

alRange.clear();

return alRange;

}

// Get nearby goats in range

for (Iterator iter = board.getLions().iterator(); iter.hasNext();)

{

Lion lion= (Lion) iter.next();

if(lion.getX()>=this.getX()-RANGE && lion.getX()<=this.getX()+ RANGE)

{

if(lion.getY()>=this.getY()-RANGE && lion.getY()<=this.getY()+ RANGE)

alRange.add(lion);

}

}

return alRange;

}

**Neural Network:**

// This method is trains the Neural network

public synchronized void train() throws NeuralException

{

// Get the animat that has to be Hunt

animatToBHunt = board.getAnimatToHunt(getCurrentLion());

this.net.getNet().getMonitor().setLearningRate(.8);

this.net.getNet().getMonitor().setMomentum(0.8);

// Determine the action that the cub should take based on various input parameters

if(animatToBHunt != null)

actionShouldveBeen = board.determineAction(animatToBHunt, getCurrentLion());

else

actionShouldveBeen = ActionConstants.SEARCH;

// Compute the network's error

System.out.println("Action Should Have Been: "+actionShouldveBeen+"; Retrieved Value: "+retrievedValue);

Lion animal = getCurrentLion();

double err = Math.abs(actionShouldveBeen - retrievedValue);

animal.setError(err);

// Get the object that watches over the training

Monitor monitor = net.getNet().getMonitor();

// Set the monitor parameters

monitor.setTrainingPatterns(1);

monitor.setTotCicles(1);

// Setup the inputs for the next round of training

this.desiredNetworkOutput

.setInputArray(new double[][] { { actionShouldveBeen } });

this.inputForTraining.setInputArray(properInput());

// Now actually train the network

try {

this.net.train(inputForTraining);

} catch (NeuralException ex) {

SimUtilities.showError("Error training neural network for agent \""

+ "\"", ex);

throw ex;

}

}

// The age of the Cub is considered for the cub to start training its neural network

public void preStep() throws NeuralException

{

// If the cub has attained certain age, the cub starts self learning

if(this.age > Constant.CUBAGE)

self = true;

if(self)

train();

// reset for this step

this.wasScolded = false;

}

@Override

// The step methods gets called repeatedly

public void step() throws NeuralException{

if(this.age > Constant.CUBAGE)

{

self = true;

board.getCub().setRange(Constant.LIONRANGE);

}

try {

this.retrievedValue = retrieve();

setActionPerformed();

} catch (NeuralException ex) {

SimUtilities.showError(

"Error computing the next action to perform. \n"

+ "Agent \"" + "\".", ex);

throw ex;

}

}

@Override

public void postStep() throws Exception{

// The age of the cub is determined by the tickCount of the Repast model

this.age = (int)NeuralModel.model.getTickCount();

if(self){

//move cub based on action taken

actionPerformed = (int) getActionPerformed();

// Positive reinforcement is given to the cub if it performs the action that should be performed

if (Math.abs(actionShouldveBeen - actionPerformed)<=epsilon)

{

praise();

doAction();

}

// Negative reinforcement is given to the cub if it does not perform the action that should be performed

else

{

scold();

animatToBHunt = board.getAnimatToHunt(getCurrentLion());

if(animatToBHunt != null)

actionShouldveBeen = board.determineAction(animatToBHunt, getCurrentLion());

else

actionShouldveBeen = ActionConstants.SEARCH;

doAction();

}

}

//Observation learning (Following the mother) is carried out for the cub if it has not reached the age

else{

Mother mother = (Mother) board.getMother();

double x = mother.getX();

double y = mother.getY();

this.setX(x - 10);

this.setY(y - 10);

this.animatX = x-10;

this.animatY = y-10;

actionPerformed = (int) getActionPerformed();

animatToBHunt = board.getAnimatToHunt(getCurrentLion());

if(animatToBHunt != null)

actionShouldveBeen = board.determineAction(animatToBHunt, getCurrentLion());

else

actionShouldveBeen = ActionConstants.SEARCH;

doAction();

}

}

// Cub starts performing the hunt only when it has reached a certain age.

private Lion getCurrentLion(){

if(this.age > Constant.CUBAGE){

return board.getCub();

}else return board.getMother();

}

// The function returns the inputs that has to be fed into the neural network

protected double[][] properInput(){

double nSize[][] = new double[1][6];

Animat lion = getCurrentLion();

if(animatToBHunt != null)

{

nSize[0][0] = animatToBHunt.getPosX()+animatToBHunt.getPosY();

nSize[0][1] = animatToBHunt.getSize();

nSize[0][2] = animatToBHunt.getAnimatType();

nSize[0][3] = animatToBHunt.getEnergy();

}

else

{

nSize[0][0] = 0.0;

nSize[0][1] = 0.0;

nSize[0][2] = 0.0;

nSize[0][3] = 0.0;

}

nSize[0][4] = lion.getPosX()+ lion.getPosY();

nSize[0][5] = lion.getEnergy();

return nSize;

}

// This function is used for the cub's step function in determining the action that should be performed

protected double[] getInputs(){

double nSize[] = new double[6];

if(animatToBHunt!=null)

{

nSize[0] = animatToBHunt.getPosX()+animatToBHunt.getPosY();

nSize[1] = animatToBHunt.getSize();

nSize[2] = animatToBHunt.getAnimatType();

nSize[3] = animatToBHunt.getEnergy();

}

else

{

nSize[0] = 0.0;

nSize[1] = 0.0;

nSize[2] = 0.0;

nSize[3] = 0.0;

}

Animat cub = getCurrentLion();

nSize[4] = cub.getPosX()+ cub.getPosY();

nSize[5] = cub.getEnergy();

return nSize;

}

//Called only by cub during its imitationNN build

public Animat getAnimatToHunt(Lion cub){

if(cub.animatToHunt == null)

cub.doHunt(cub);

return cub.animatToHunt;

}

**Case Base:**

// Function to build case bases for learning the

public void createCaseBases(Animat animatToHunt, Animat animatHunting)

{

AgentInputs av1 = new AgentInputs();

SensoryItem f1, f2;

f1 = new SensoryItem("AnimatHunting",animatHunting);

f2 = new SensoryItem("AnimatToHunt",animatToHunt);

av1.addSensoryItem(f1);

av1.addSensoryItem(f2);

List<AgentAction> act1 = new ArrayList<AgentAction>();

double actionToBTaken=board.determineAction(animatToHunt, animatHunting);

AgentAction a1 = new AgentAction(actionToBTaken);

act1.add(a1);

Case c1 = new Case(av1,act1);

this.caseBase.addCase(c1);

}

public CaseBase getCaseBase()

{

return this.caseBase;

}