All for one one for all

Student Name: Bamshad Shirmohammadi

Supervisor: Professor Arend Hintze

Field of study: Micro data analysis with emphasis on Business Intelligence

Dalarna university

**Abstract:**

Evolvement is the mystical characteristic of all creatures. A bird wants to fly higher, a tiger will live longer, if it run faster and the best robot is the one who serve us better. Yes, in the Artificial Intelligence we have the magic of evolvement and more interestingly we we can discuss the definition of concepts such as dominance and submissive behavior, sacrifice and team working. In this research we are going to check if the robots or agents through the process of evolvement will change their behavior (based on the situation) to go for all for one, one for all and average methods. All for one means the best genetic is the one which their worst member has the highest score among all the genes. One for all is the opposite and therefore, everyone should help to just one agent get a very high score and finally, average is between these two extreme approaches. In our world the agents are Markov brains and their task is to collect resources.

Table of Contents

[Introduction 2](#__RefHeading___Toc1298_74506550)

[Background and related work 2](#__RefHeading___Toc1300_74506550)

[Method and implementation 2](#__RefHeading___Toc1641_2061819211)

[Components: 3](#__RefHeading___Toc1643_2061819211)

[1. Area 3](#__RefHeading___Toc1645_2061819211)

[2. Agents 4](#__RefHeading___Toc1647_2061819211)

[3. Evaluation 4](#__RefHeading___Toc1649_2061819211)

[One example 5](#__RefHeading___Toc1291_74506550)

[References 8](#__RefHeading___Toc1302_74506550)

# Introduction

**Research question:**

Is it possible to find specific scenarios that force agents (which use Markov brain) to go for the following approaches?

a) All for one (Everyone should be fine): They should cooperate in a way that their lower score is still a good one. On the other words, the winner group is the one that has highest score among lowest scores of each genetic.

b) One for all (We should have a legend): All of them works so one of them get a very high score

c) Average: They should have a good average score.

# Background and related work

1) All-for-one or One-for-all - how Differential Reward Schemes affect the Evolution of Cooperative Behavior

2) Evolution of autonomous hierarchy formation and maintenance

# Method and implementation

The whole process is explained in the following figure:

1. Creating the

area

2. Adding the

Agents and

Making evaluation

process

3. Running the

Experiments

4. Analyzing the

Results (scores)

## Components:

### 1. Area

**General explanation:**

It is the place where the experiments will be done. This area is divided into tiles and each tile can be set as empty, grass and wall. The agents can only move in empty and grass tiles. The agents collects the grasses which means if they go into a grass tile, it will be converted to an empty tile.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

Figure 1. An area example

In the above figure, the gray tiles are walls and the green tiles are grass.

**Technical explanation:**

The area is an object of the Area class which is created in ExampleWorld.hpp (world/ExampleWorld folder). The Area class is defined in area.cpp (core folder). We have set the area properties in the ExampleWorld.cpp (world/ExampleWorld folder).

The Area class has a constructor which has three arguments that are x, y (as the dimensions) and startTile which is for selecting the initial filling type (such as grass) of the tiles in the area. To specify the tiles type individually we can use set() function in the Area class. X value and Y value are the first two arguments to select the tile position and the third argument sets the tile’s type.

The following table explains how set the different attributes of the area:

|  |  |
| --- | --- |
| Function name | Description |
| Area::Area(int \_xDim,int \_yDim,int startTile) | It is the constructor of area class that set the area size and initial tiles type. \_xDim and \_yDim can be a positive number and the startTile is defined as follow:  0 = empty  1 = grass  2 = agent  3 = wall |
| Area::set(int x,int y, int what) | Using this function we can set each tile’s type individually. x and y identify the tile position and what it’s type (grass,empty, wall and agent) |

Table 1. The main functions to set the area attributes

### 2. Agents

**General explanation:**

Agents are the people of world (area) and their task is to collect grasses. They can turn left, right, move forward/backward or do nothing in each turn. As soon as they move to a new tile which is filled with grass, they will collect it and will gain one score. So, eventually we will assess each agent based on it’s total score by the end of the turns.

**Technical explanation:**

Agents are objects of Organism class which is decelerated and defined in Organism.hpp and Organism.cpp respectively.

The agents has several modules that are shown in the next figure:

Brain

ID

bornInGeneration

genome

DataMap

Initial position

Figure 2. Agents modules

The agents make simple actions. In this scenario they can turn left, right or move forward.

This class uses three other classes that are Brain, Genome and dataCollector. Brain is in the brain folder and the two other are in the core folder.

In this project we have created them in ExampleWorld::evaluateSolo() (which is in world/ExampleWorld/ExampleWorld.cpp).

### 3. Evaluation

The evaluation is based on scores which is number of grasses that the agents collect during the experiments.

Condition 1) Groups of individuals collaborate together in the world, everyone receives only what they harvest (a control)

Condition 2) An organism is picked and cloned 4 times, the four clones are put in the environment and their individual performance recorded, the agents now receive:

a) what their worst performer received (all of one)

b) what their best performer received (one of all)

c) what they got on average (also a control)

**Technical explanation:**

We use ExampleWorld::evaluateSolo() and ExampleWorld::evaluate() for this process (world/ExampleWorld/ExampleWorld.cpp).

## One example

In this section I show how we are going to set the scenario fields.

I) World/ExampleWorld/ExampleWorld.cpp

As the first step we should build our area.

ExampleWorld::ExampleWorld(){

//initialize the area container to size 15x15 and all the tiles as empty

area=std::make\_shared<Area>(15,15,0);

1. Setting the dimension of area and the initial

tiles type. 15\*15 and empty

//adding all walls

for(int i=0;i<15;i++){

2. Setting the walls in the area

area->set(0,i,3);

area->set(14,i,3);

area->set(i,0,3);

area->set(i,14,3);

}

//adding some grass

for(int y<=10;y<=12;y++){

3. Setting the grass positions in the area

for(int x<=2;x<=12;x++){

area->set(x,y,1);

}

}

area->setAsStartConfig(); // we mark this state as the start configuration, makes resetting easier later

}

So, it is a 15\*15 pixels area which the round pixels are filled with wall. There are 3\*11 tiles for grass.

After creating the area, we add the agents and evaluate their performance (each individual) at once as follow:

void ExampleWorld::evaluateSolo(std::shared\_ptr<Organism> organism,bool analyze,bool visualize){

double score=0.0;

4. Setting the initial score of the agents

auto brain=organism->brain; //retrieve the brain from the organism

int xPos,yPos,dir; // define three variables defining the position and facing direction of the agent

brain->resetBrain(); //the brain is reset at "birth"

area->resetToStartConfig();

5. Setting the initial position and direction of the agents

xPos=Random::getInt(1, 14); //random xPos, lower and upper limit are INCLUSIVE!

yPos=Random::getInt(1, 14); //random yPos, lower and upper limit are INCLUSIVE!

dir=Random::getInt(0,3); //four direction 0-3 = N,E,S,W

area->set(xPos,yPos,\_agent); //marking the start location of the agent

6. Setting the number of turns (200) that each agent has to gather scores

for(int t=0;t<200;t++){

int inFront=area->getFromDirDist(xPos, yPos, dir, 1); //this gives you the tile in front of the agent

brain->setInput(0, inFront&1); //bit magic to set the inputs

brain->setInput(1, (inFront>>1)&1); //bit magic to set the inputs

//now you need to allow the brain a single update "it is thinking now"

brain->update();

//now the actions from the brain are querried and translated into a single "action" number

int action=(bit(brain->getOutput(0))<<1)+bit(brain->getOutput(1));

//clear the tile the agent stands on

It is the place that agents inform us their decision

(it is Not one of setting steps)

area->set(xPos,yPos,\_empty);

//executing the actions

switch(action){

case 0:

//do nothing

7. Implementation of the brain decisions.

For example, for case zero do nothing and turn

right in case 2.

Is it what we decide?

is it our decision to set how to deal with each case?

For example I can say what the agent should do

if the action variable’s value is zero or

these actions are constant?

Also, why we have cases (for grass, wall, …) only for the front movement

and not for turn left and right?

I think here we should have all the directions

and all the four cases for each direction.

break;

case 1:

//turn left

dir=(dir-1)&3;

break;

case 2:

//turn right

dir=(dir+1)&3;

break;

case 3:

//move forward

//that depends on what tile was inFront

switch(inFront){

case \_gras:

score+=1.0; //gras was eaten

// NO break -> to run into the next case which moves the agent

case \_empty:

//move forward

xPos+=area->xm[dir];

yPos+=area->ym[dir];

break;

case \_agent:

case \_wall:

//agent tried to move into a wall or an agent (which should not exist)

//do nothing

break;

}

break;

}

//update the area where we stand

area->set(xPos,yPos,\_agent);

//add a time tick to the agea, so it knows that time progressed

area->tick();

}

organism->dataCollector->setValue("score",score);

}

II) Main.cpp

8. what is 100?

auto population=Population::createStartPopulation(

Parameters::getInt(std::string("Core-numberOfStartOrganisms"), 100),

BioGenome::getTemplate(), // default genome

MarkovBrain::getTemplate(), //standard Markov Brain

world->nrOfSensors(), world->nrOfMotors());

9. what is 50000?

int nrOfGenerationsToEvolve=Parameters::getInt(std::string("CorenrOfGenerationsToEvolve"),50000);

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

Figure 2. The illustration of the example environment

# References