Implementation of Neural Network Architectures in FlatWorld II Simulation Environment

Veera V S B Nunna, Meghana Savitri Dasigi

Dept. of Computer Science, University of New Mexico

E-mail: bhargavn6@unm.edu; meghanadasigi@unm.edu;

Abstract:

This project aims at building different architectures that increase in the complexity, to train a simulated neural network into learning to move, to eat and then observe the Flat World for noise and move in the direction of food. The main aim of the project is to make the agent live for a longer time. All this is achieved in a series of architectures. The successive architectures are built from the defects of the existing architecture.

1. Introduction:

The project at hand is developing a agent with the help of neural networks, imitating the real world biological neurons, to perform some tasks as it is being developed. The project starts at a simple point where the agent is just created like a new born baby. It doesn't have knowledge what is happening around or what is its purpose. As it is created and it doesn't have knowledge of anything it tends to die eventually because of various reasons that are defined by us. We program the agent to be able to identify its food and eat and try to sustain longer in its world. There are many architectural designs and testing to develop this. We try to program the agent as natural as possible. The agent is upgraded with eyes, ears and a head. The eyes and ears are used to sense the world around it and help get food and boost energy. All the changes are not done at once. The project plan is to move strategically with some architectural milestones. That is, in the very basic the agent is introduced, then in the next architecture the agent starts moving around, then it starts eating all it gets in contact with furthermore it develops eyes, then it starts seeing around it for food. Its food consists of healthy, neutral and dangerous foods that give energy, do not do anything and deplete the energy levels of the agent, respectively. As the agent develops eyes it learns to look around and learn what are foods and what are not and then classify among

them and try eating only healthy foods. Similar is the case with the ears. Therefore finally the agent must be able to coordinate its eyes, ears and eat simultaneously and move around the world trying to sustain as long as possible.

2. Approach

The approach followed in designing the architectures is really simple. We start from scratch where the agent just living. Eventually we developed the movement, the senses and then analysis of the paths and foods it finds in the world. We have gone through all these changes from architecture 0 through 7, where the agent develops gradually. The problems of the existing architectures are found out and analyzed to create the next architecture and implement it. Each architecture is then analyzed and evaluated. The evaluation is done by plotting histograms and comparing the lifespan of the agent in comparison to previous architectures.

3. Details of Architectural Designs

3.1 Architecture 0: No movement, measure the lifetime.

In this architecture, the agent is created and we measure the lifetime of the agent as it just lives. I.e the lifetime of the agent is measured when it has no activity and no food to make it live. This can be considered as the stand by time of the agent. So the lifetime is measured as function of time as it is created through the death of the agent. Also it can be observed that the speed of the agent is null as it doesn't involve in movement or any activity.

3.2 Architecture 1: Movement, measure lifetime as function of speed.

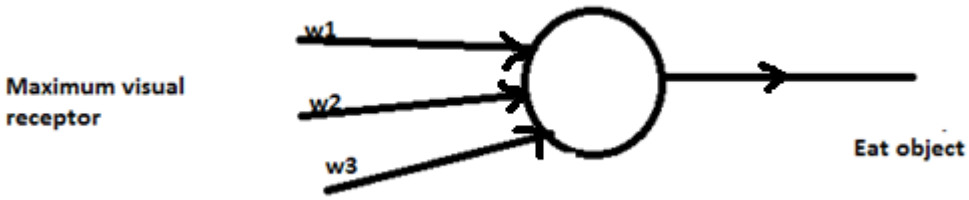
Here the agent is created and allowed to move in the world. As the agent is created, it starts moving and consumes energy to move and dies eventually because of lack of energy. Hence in this architecture we measured the lifetime, as the agent is born and starts moving and dying. Also, as the agent moves it comprises of certain speed. It was observed that as the agent started moving the rate of depletion raises and dies quickly when compared to the basic architecture.

3.3 Architecture 2: Movement, eat-all-on-contact.

In this architecture the agent starts eating the objects in the World as it comes into contact with it. So here the function of speed is not null. The characteristics of agent are: it is alive, it moves and eats the objects on its path. Hence it can be deduced that the lifetime of the agent is not predictable as it eats combinations of healthy, neutral and bad food. Upon multiple experiments, it was seen that the lifetimes do not follow a particular pattern. This can be considered as the base architecture for the further architectures.

3.4 Architecture 3: Movement, eat-all-on contact. Classify the objects with a single delta rule neuron

In this architecture, the agent behaves in the same way as in the Architecture 2. But here it can see the object. It learns to differentiate the object that is just consumed. We took the eyelet with the maximum visual as the input to the single delta rule neuron. The change in the energy is used as the desired value for the learning.



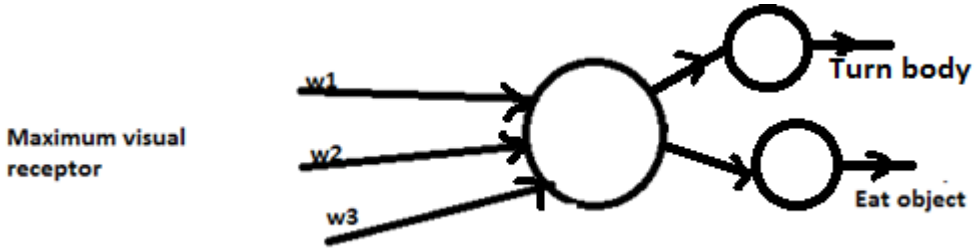
3.4 Architecture 4:

Although the agent inherits all the features from the previous architectures, the agent doesn't use any knowledge from the previous architectures. Instead, it just keeps going in a straight line. It eats everything in contact. It regenerates at a random point. This is a trial and error method to analyze the lifespan if the agent keeps moving in path without any detours and eating everything.

3.5 Architecture 5:

There is a little development in the agent from the previous architecture. It still has the same features as from the previous architecture. In this the agent still keeps moving a

straight line but eats selectively. It eats only green food which is the healthy food and absolutely necessary for the agent to survive.

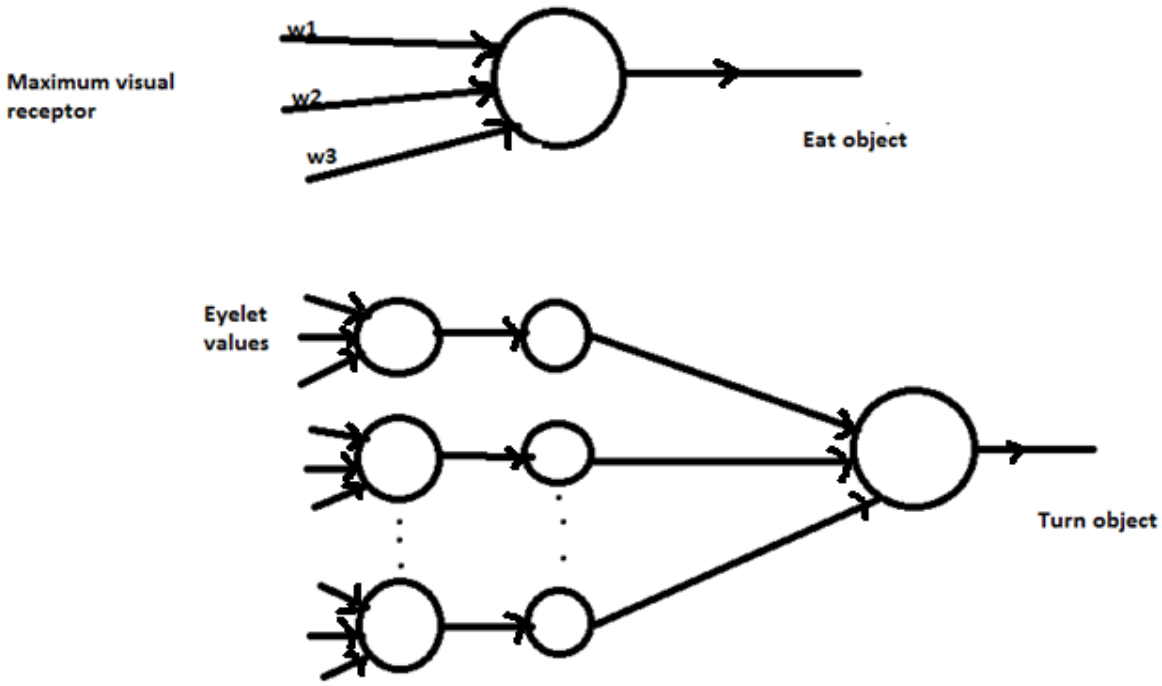


3.6 Architecture 6:

In this architecture, the agent can analyze the brightest object visible through its eyelets, if it is green or not. The agent moves towards the object if it is green. Otherwise it changes its direction.

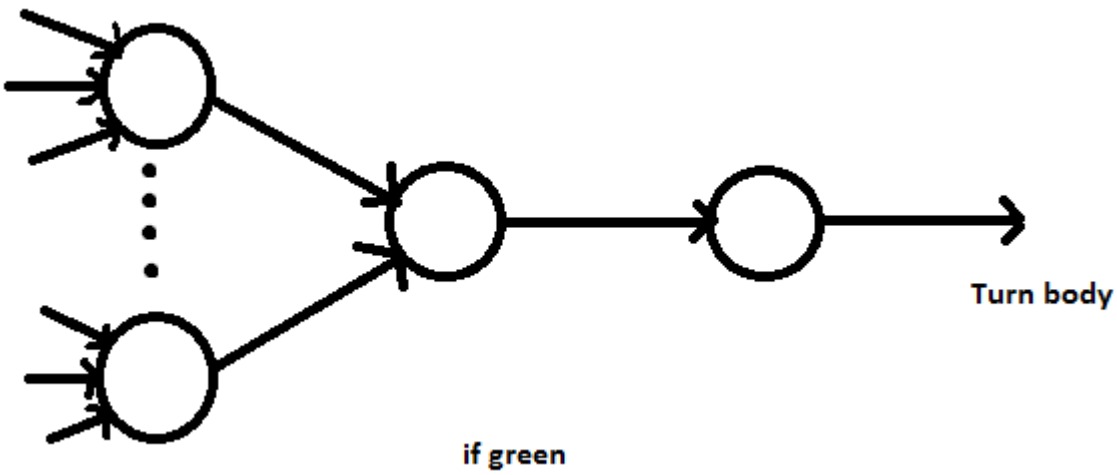
3.7 Architecture 7:

In this architecture the agent doesn't move if the metabolic charge is above a certain threshold value't'. The agent move only if the metabolic charge falls below the threshold value. Also the agent moves towards the brightest object visible in its eyes. But it doesn't consume if the object is not green. It changes its direction by and starts moving. Also is



3.8 Architecture 8:

The agent still eats and stops, until its charge is lesser than the threshold metabolic charge. The agent sees the objects with its eyes and analyzes it. It moves towards the object only if it is green. Suppose, it finds a red object emitting the brightest light and also one of the eyelets records green light but less bright than the red object. It still chooses the green object over the red object. This is implemented by using Winner takes all strategy.



4. Learning Approach

The basic goal of all architectures is to increase the learning capabilities at each level. The most basic learning is that the agent learns how to classify objects. As the world is contained of healthy, neutral and dangerous objects, the agent learns on training, which food does more good to it and helps in sustaining in the world longer. Then it leanrs and improves by learning to eat only green food.Therefore, as it classifies it also learns on what food to prioritize as it moves towards two objects equidistance from its present position. From the consumption, the energy levels differ in the agent. This change is then measured for various foods and combinations of food consumed.

- In architecture 0, the agent just lives
- Next it starts moving randomly until its metabolic charge reaches 0 and dies.
- It then starts eating every in contact.
- The agent then starts using its eyelets. It eats everything on contact but is able to differentiate between objects.
- The agent then moves only in straight line.
- Moves in straight line, eats selectively.
- Looks at the brightest object near it, moves towards it if it's green, otherwise change direction.
- Differentiates between the brightest objects and moves only towards green object.

5. Evaluation Approach

Every new architecture goal is to improve the life span from its previous architecture. We are going to implement the neural network architectures programmatically in c. We will create a .c file and call the functions it in 'controller.c'. All the actions of the agent will be based on the outputs of the neural network. For every architecture we are going to collect the data such as number of trails, misclassifications in the training, number of objects consumed in each life, also the varying the speeds for each architecture. We are going to run 120 trails for each experiment to get good statistical data. We pick the minimum, maximum and medium of each architecture.

6. Analysis/Presentation Approach

We are going to compare the lifetime for each of the architectures to see the improvement in the life time for each of the architectures. Also vary the speed in each of architecture and observe the life time as the speed changes. We will plot the lifespan for each of architecture. And also mean of the lifetimes obtained in trials.

7. Results:

7.1 Comparison of all architectures

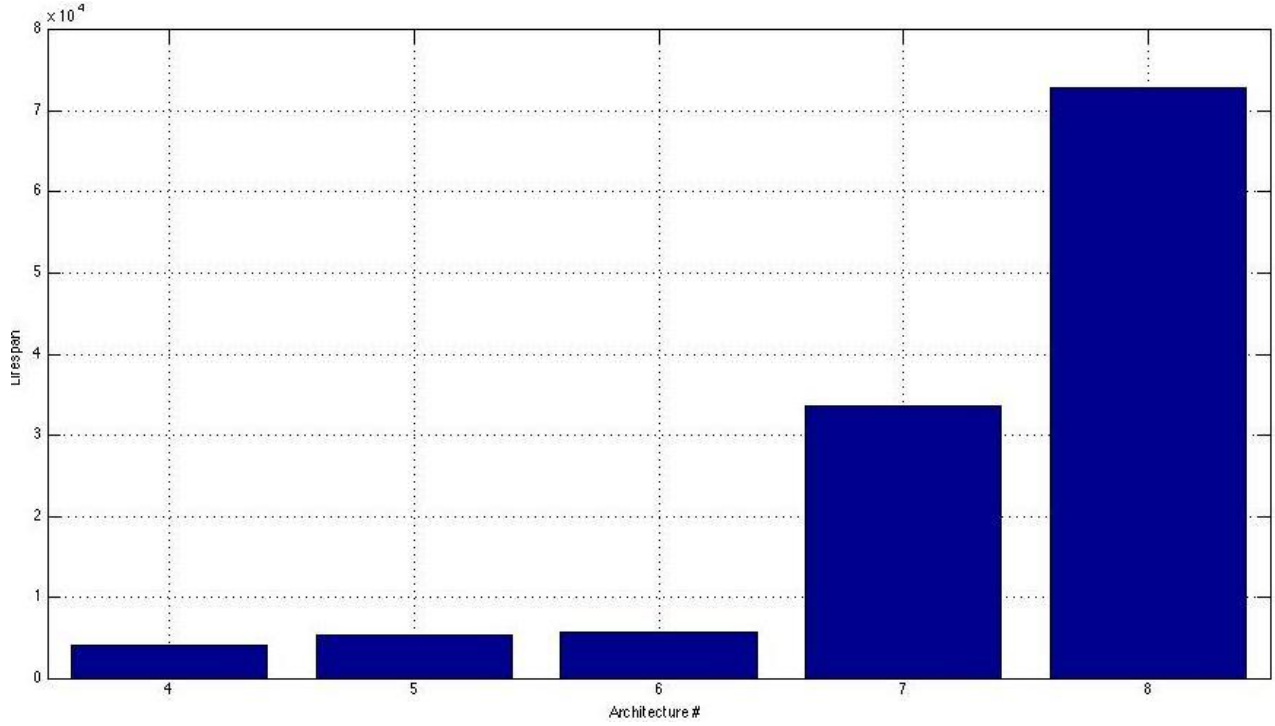


Figure 1 – Average lifespan of each architecture.

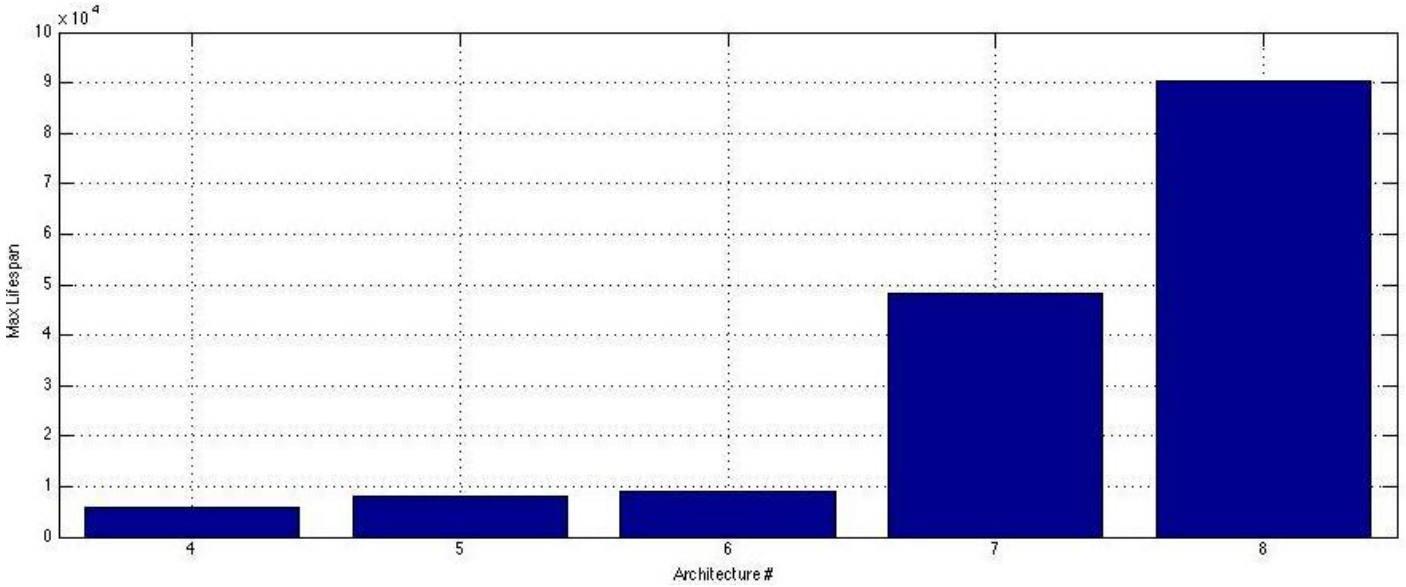


Figure 2– Maximum lifespan of each architecture

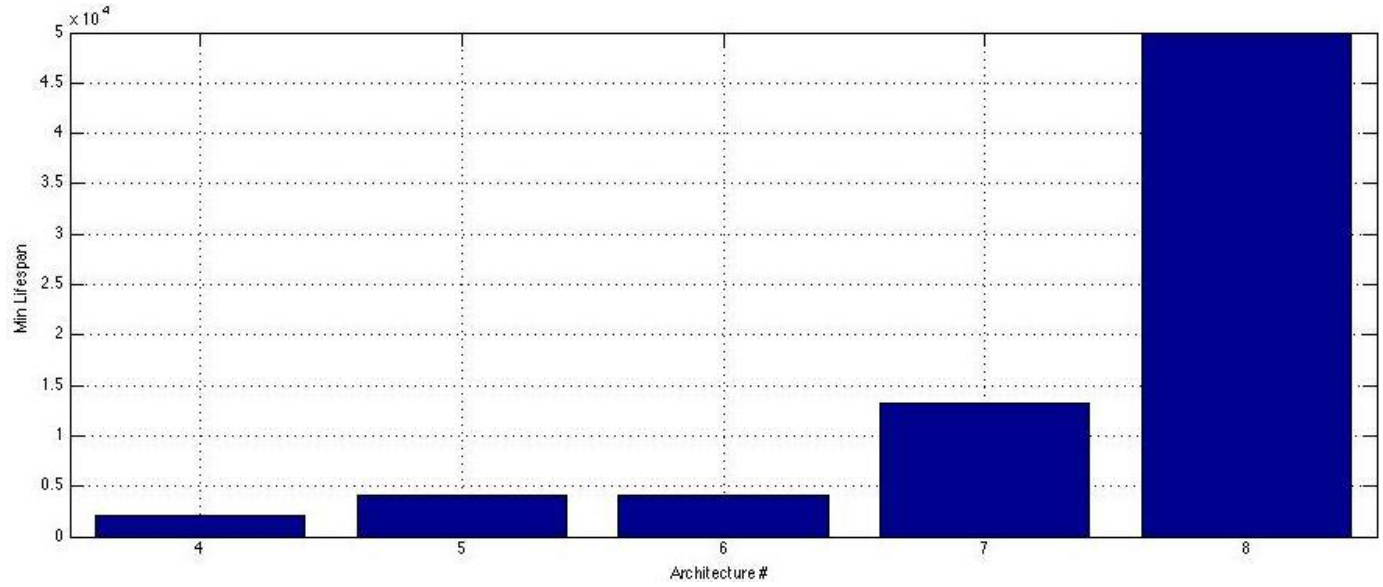


Figure 3- Minimum lifespan for each architecture

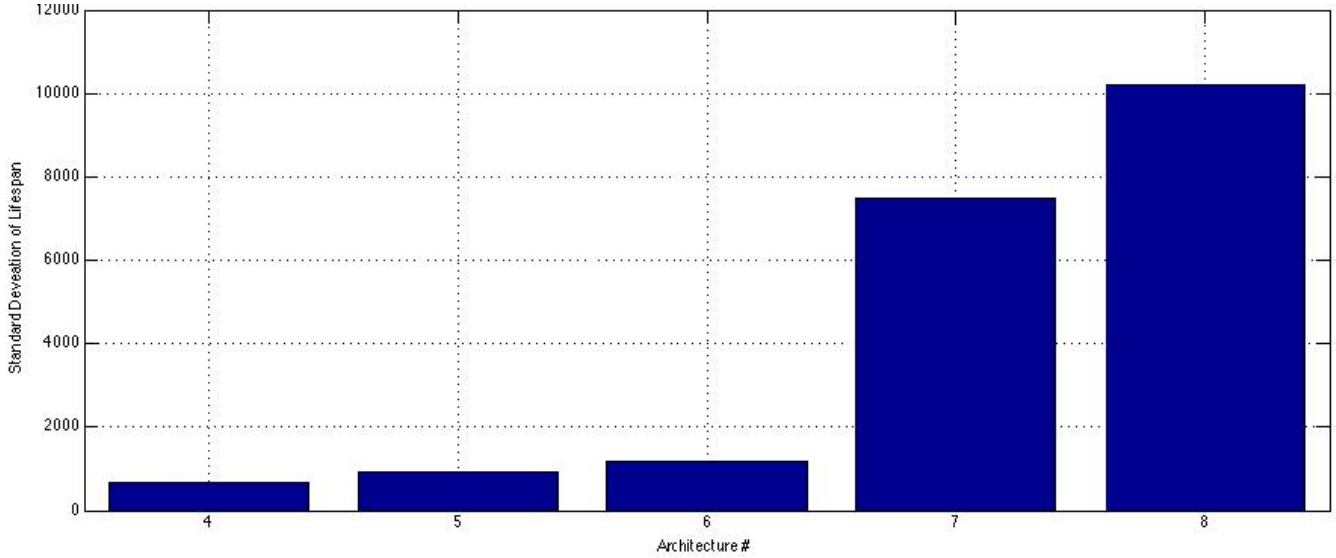


Figure 3- standard deviation vs Architecture number.

7.2 Results of architecture 4: the maximum lifespan is 6007. This better than the previous architecture because it goes in a straightline but doesn't eat anything.

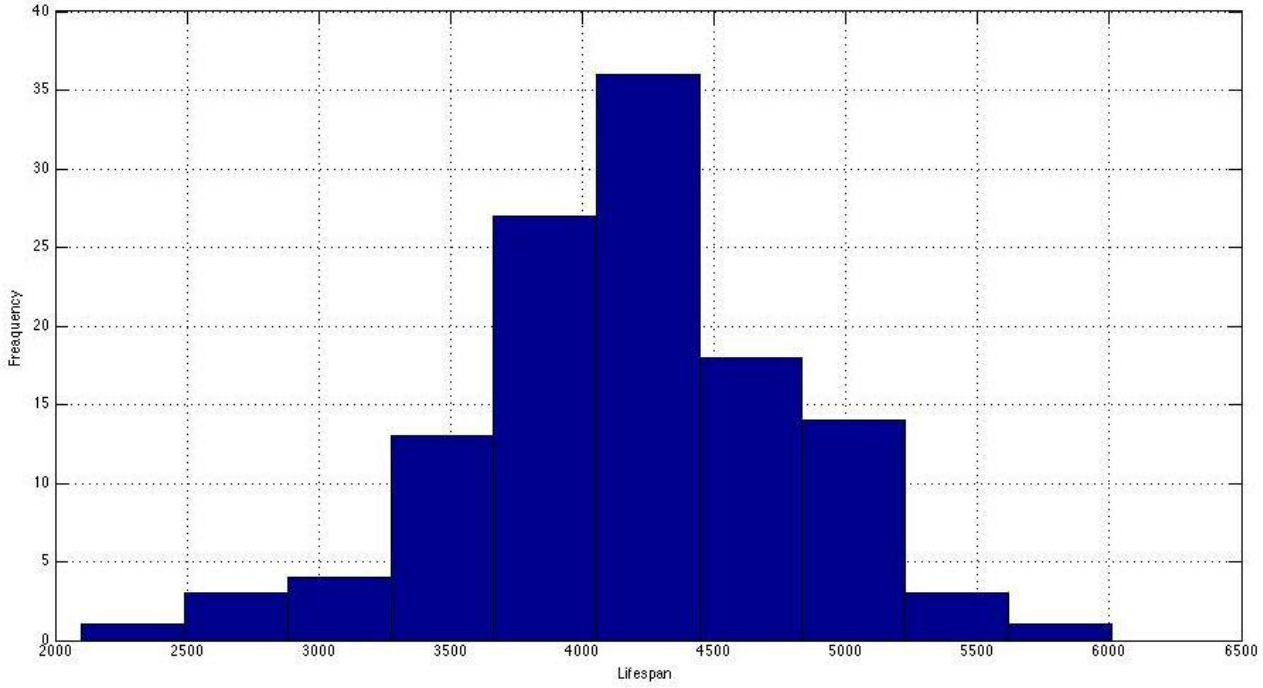


Figure 5 – Histogram of lifespan for Architecture 4

7.3 Result of architecture 5:

The maximum lifespan is 8148. This is because the agent moves in a straight line.

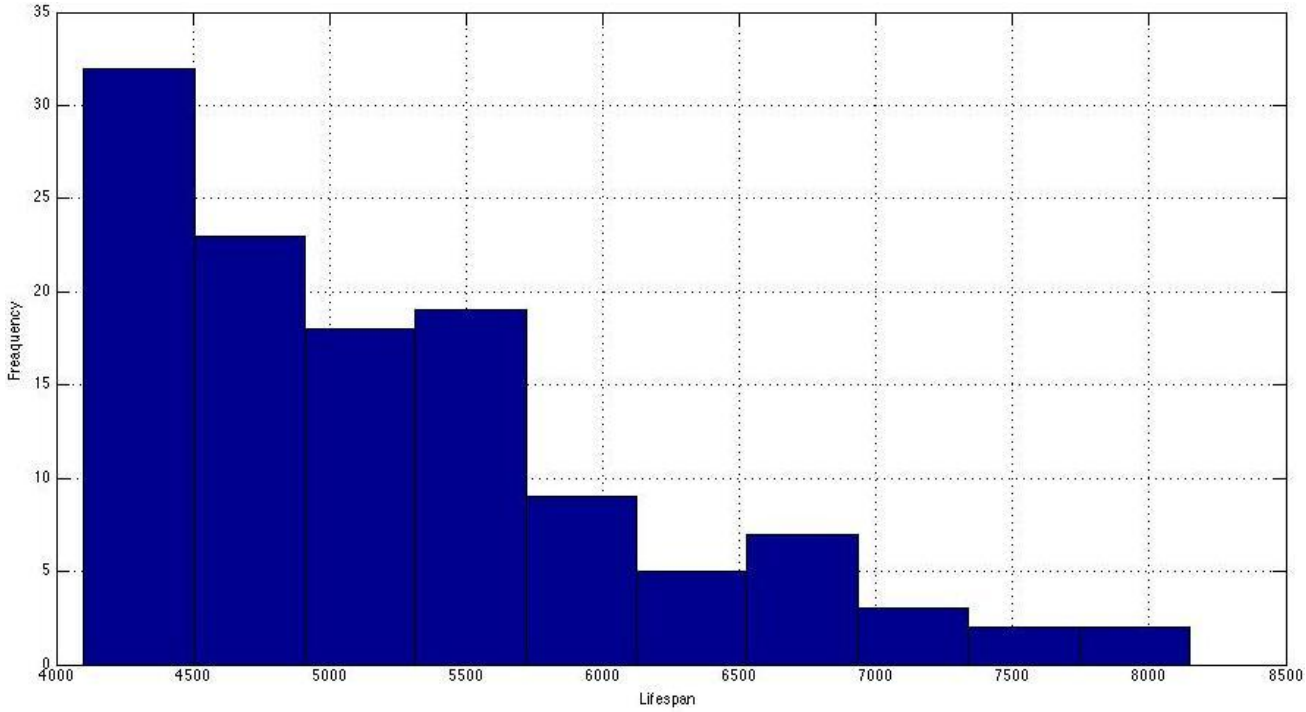


Figure 6– Histogram of lifespan for Architecture 5

7.4 Result of Architecture 6:

The maximum lifespan of this architecture is 8965. We ran the architecture for 120 epochs. This is a better time than the previous because it moves in straight line and eats only selectively green food.

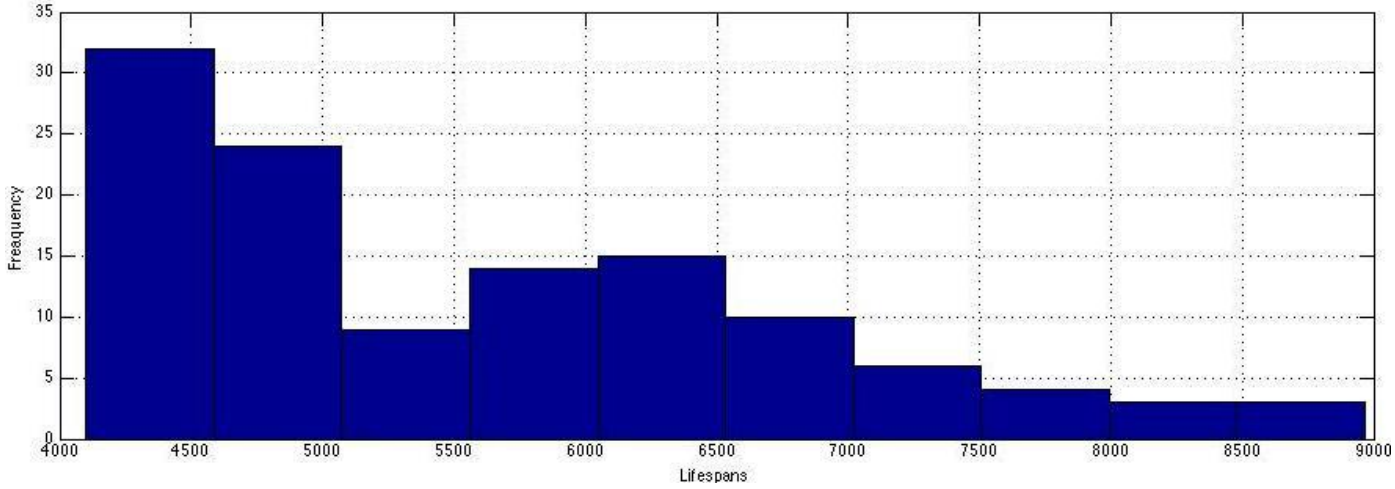


Figure 7- Histogram of Lifespan of Architecture 6

7.5 Result of Architecture 7:

We got better results in architecture 7 than architectures 4,5,6. The maximum lifespan achieved is 33,693 units. We ran the architecture for 120 epochs. We also measured the average lifespan as a function of threshold value

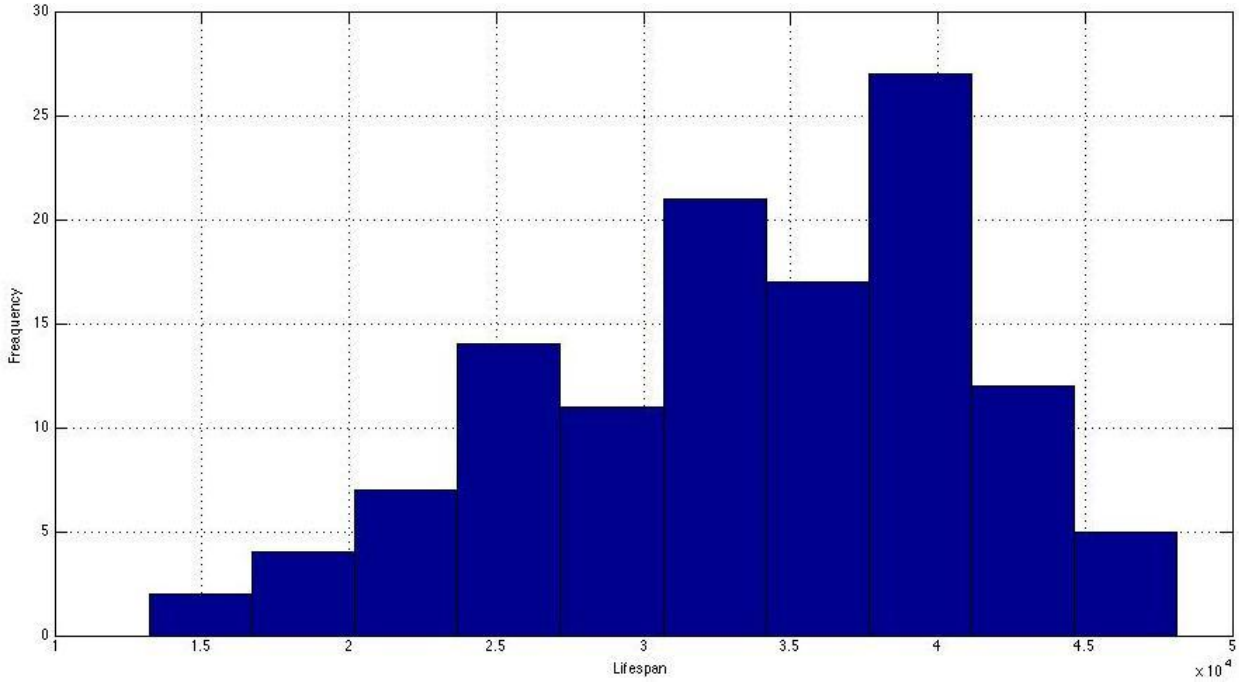


Figure 8 –Histogram of Architecture 7

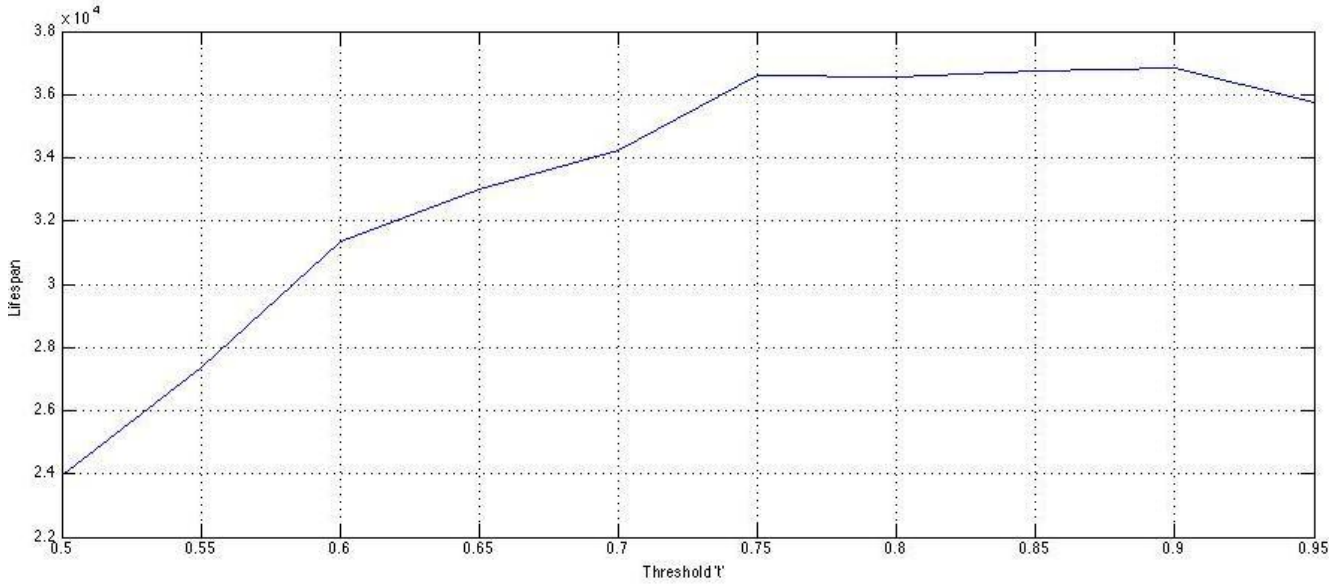


Figure 9– Lifespan vs Threshold for architecture 7

7.6 Result of Architecture 8: We got the maximum lifespan as 90392 which is better than the 7th architecture. In this architecture the agent always picks green food, which in turn helps increase the lifespan. It avoids going towards other objects, which saves the metabolic charge. Therefore resulting in best lifespan by far. We also measured the average lifespan as a function of threshold value

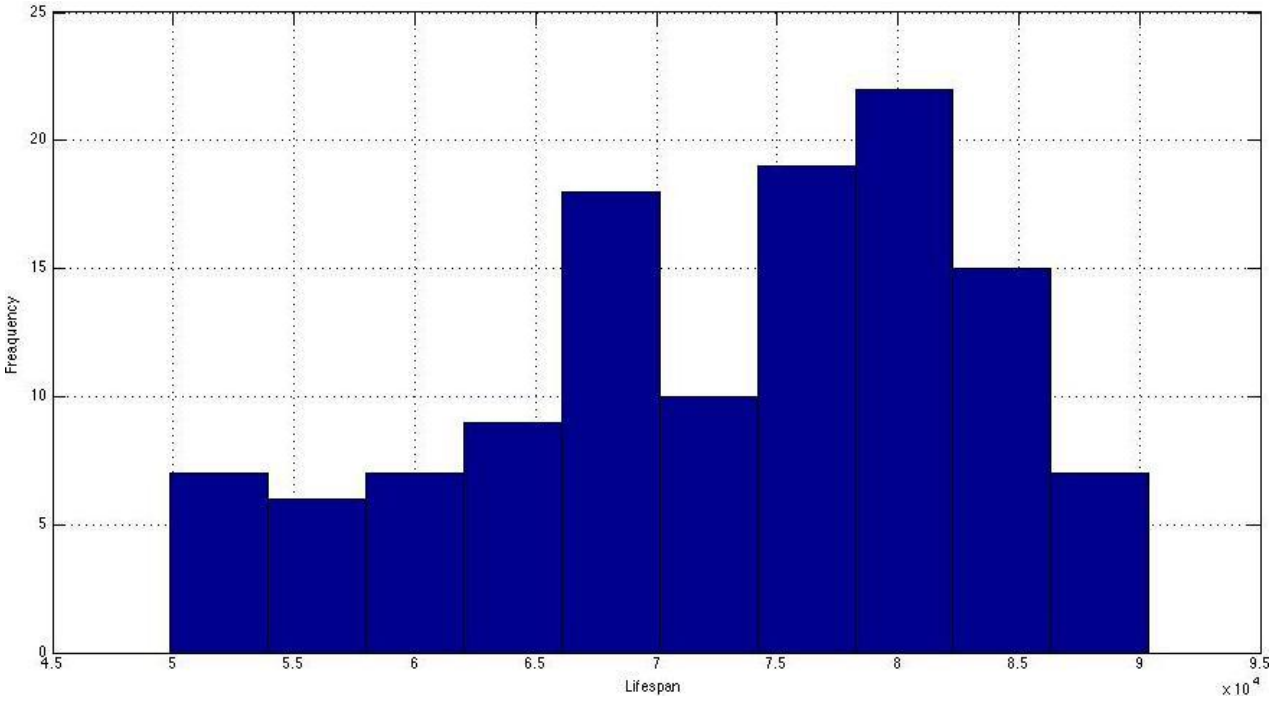


Figure 10- Histogram of lifespan of Architecture 8

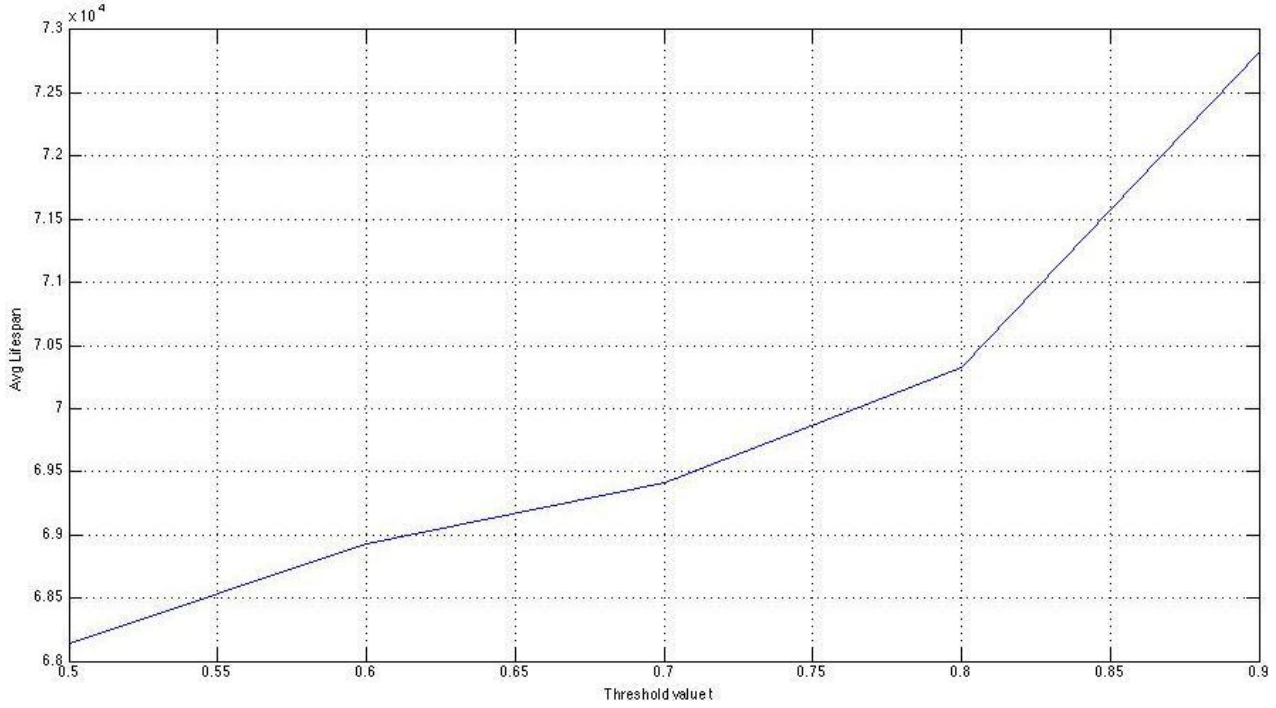


Figure 11- Lifespan vs Threshold of Architecture 8

8. Discussion:

In the document, there are 600 foods, 200 of each type (blue, green and red). The energy of the agent decreases by 0.1 if eats red food. The energy increases by 0.1 if the agent consumes green food. There wouldn't be any reaction if the agent consumed blue food.

In our project, we concentrated mainly on the agent eating the green food. In architecture 8, the agent eats the maximum number of green foods, thus increasing the lifespan to the maximum which is 90392. Thus architecture 8 achieves, by far the best lifespan. So, analyzing the lifespan achieved, compared to the 100,000 units of lifespan we have achieved 90% of the lifespan.

9. Summary and Conclusion:

- The summary of the project is that maximizing the consumption of green food and minimizing the movement of the agent to the required amount, results in the maximum lifespan of the agent. By minimizing the movement, we mean, moving of the agent only when it reaches the threshold metabolic charge,t. Following this strategy, we could achieve a very high lifespan of 90%.

10. Acknowledgements:

We profusely thank Professor Caudell for discussing the project repeatedly without tire in the class and clarifying our doubts. We are also thankful for the CS department labs for being able to use anytime.

11. References:

[1]Simon Haykin, Neural Networks textbook

[2] Professor Thomas Caudell notes.

Appendix:

```
/*
 *  Controller.c
 *  For the UNM Neural Networks class, this should be the only
file you will need to modify.
 *   World and agent initialization code are located in the
main().  An
 *  example of a non-neural controller is included here.
 *   Note that most all of the functions called here can be
found in the
 *  file FlatworldIICore.c
 *
 *
 *  Created by Thomas Caudell on 9/15/09.
 *  Modified by Thomas Caudell on 9/30/2010
 *  Modified by Thomas Caudell on 9/13/2012
 *  Modified by Thomas Caudel on 9/10/14
 *  Copyright 2009 UNM. All rights reserved.
 *
 */


void agents_controller( WORLD_TYPE *w )
{ /* Adhoc function to test agents, to be replaced with NN
controller. tpc */

     FILE *arch;
arch=fopen("arch9.txt","a+");

  AGENT_TYPE *a ;
  int collision_flag=0 ;
  int i,k ;
  int maxvisualreceptor = -1 ;
```

```
  int nsomareceptors ;
  int nacousticfrequencies ;
  float delta_energy ;
  float dfb , drl, dth, dh ;
  float headth ;
  float forwardspeed ;
  float maxvisualreceptordirection ;
  float bodyx, bodyy, bodyth ;
  float x, y, h ;
  float **eyevalues, **ear0values, **ear1values, **skinvalues
;
  float ear0mag=0.0, ear1mag=0.0 ;
  time_t now ;
  struct tm *date ;
  char timestamp[30] ;

  int cflag=0;
  float xg,xb,xr,n_error;
  float rms=0;

  //AGENT_TYPE *a ;

  /* Initialize */
  forwardspeed = 1 ;
     a = w->agents[0] ; /* get agent pointer */


     /* test if agent is alive. if so, process sensors and
actuators.  if not, report death and
        reset agent & world */
     if( a->instate->metabolic_charge>0.0 )
     {
          /* get current motor rates and body/head angles */
          read_actuators_agent( a, &dfb, &drl, &dth, &dh ) ;
```

```
          read_agent_body_position( a, &bodyx, &bodyy, &bodyth
) ;
          read_agent_head_angle( a, &headth ) ;

          /* read somatic(touch) sensor for collision */
          collision_flag = read_soma_sensor(w, a) ;
    skinvalues = extract_soma_receptor_values_pointer( a ) ;
    nsomareceptors = get_number_of_soma_receptors( a ) ;
    for( k=0 ; k<nsomareceptors ; k++ )
    {
      if( (k==0 || k==1 || k==7 ) && skinvalues[k][0]>0.0 )
      {
        n_error=arch3(xg,xr,xb,delta_energy,w1,w2,w3,bias);
        if(n_error > 0.5)
        {
        //printf("Eating the object\n");
        delta_energy = eat_colliding_object( w, a, k) ;
        }
       cflag=1;

        maxvisualreceptor = intensity_winner_takes_all( a ) ;
           xg=eyevalues[maxvisualreceptor][1];
         xr=eyevalues[maxvisualreceptor][0];
         xb=eyevalues[maxvisualreceptor][2];
         //printf("Eye|G: %f  R: %f  B:  %f\n",xg,xr,xb);
        // printf("Delta: %f \n",delta_energy);
         //n_error=arch3(xg,xr,xb,delta_energy,w1,w2,w3,bias);
         /*Updating weights.
         w1=w1+(0.1*n_error*xg);
         w2=w2+(0.1*n_error*xr);
         w3=w3+(0.1*n_error*xb);
         bias=bias+(0.1*n_error);
         *///End updating weights
         //counter_b++;
```

```
          //err2+=pow(n_error,2);
          //rms=sqrt(err2/counter_b);
         // printf("Counter: %d \n",counter_b);
         // printf("--\n");
           //printf("%f,%f,%f,%f \n",w1,w2,w3,bias);
          //fprintf(arch,"\n%d\t%f",counter_b,rms);




      }
    }
      /*Changes here
      if(cflag==1)
      {

         cflag=0;

       }*/

    /* read hearing sensors and load spectra for each ear, and
compute integrated sound magnitudes */
    read_acoustic_sensor( w, a) ;
    ear0values = extract_sound_receptor_values_pointer( a, 0 )
;
    ear1values = extract_sound_receptor_values_pointer( a, 1 )
;
    nacousticfrequencies = get_number_of_acoustic_receptors( a
) ;
    for( i=0 ; i<nacousticfrequencies ; i++ )
    {
      ear0mag += ear0values[i][0] ;
      ear1mag += ear1values[i][0] ;
    }
```

```
    //printf("simtime:     %d     ear0mag:     %f     ear1mag:
%f\n",simtime,ear0mag,ear1mag) ;

          /*  read  visual  sensor  to  get  R,  G,  B  intensity
values */
          read_visual_sensor( w, a) ;
          eyevalues  =  extract_visual_receptor_values_pointer(
a, 0 ) ;

    /* find brights object in visual field */
    //maxvisualreceptor = intensity_winner_takes_all( a ) ;
    maxvisualreceptor = pick_green_objects( a, eyevalues ) ;
          if( maxvisualreceptor >= 0 )
          {
            xg=eyevalues[maxvisualreceptor][1];
            xr=eyevalues[maxvisualreceptor][0];
            xb=eyevalues[maxvisualreceptor][2];
            /* use brightest visual receptor to determine how
to turn body to center it in the field of view */
               maxvisualreceptordirection                    =
visual_receptor_position(                a->instate->eyes[0],
maxvisualreceptor ) ;
               /* rotate body to face brightes object */


               set_agent_body_angle(      a,      bodyth      +
maxvisualreceptordirection ) ;



    }
    else
    {
```

```
      //printf("agents_controller-      No   visible   object,
simtime: %d, changing direction.\n",simtime) ;
      read_agent_body_position( a, &bodyx, &bodyy, &bodyth ) ;
               set_agent_body_angle( a, bodyth + 45.0 ) ;
    }

    /* move the agents body */

    if( a->instate->metabolic_charge>0.8)
    {
     set_forward_speed_agent(a,0.0);
    }
    else
    {
     set_forward_speed_agent( a, forwardspeed ) ;
          move_body_agent( a ) ;
    }
    //set_forward_speed_agent( a, forwardspeed ) ;
          //move_body_agent( a ) ;

          /*  decrement  metabolic  charge  by  basil  metabolism
rate.  DO NOT REMOVE THIS CALL */
          basal_metabolism_agent( a ) ;
          simtime++ ;
          //printf("Eye|G: %f  R: %f  B:  %f\n",xg,xr,xb);

     } /* end agent alive condition */
     else
     {

    /* Example of agent is dead condition */
          printf("agent_controller- Agent has died, eating %d
objects. simtime: %d\n",a->instate->itemp[0], simtime ) ;
    fprintf(arch,"%d\n",simtime);
```

```
          now = time(NULL) ;
          date = localtime( &now ) ;
          strftime(timestamp, 30, "%y/%m/%d H: %H M: %M S:
%S",date) ;
          printf("Death time: %s\n",timestamp) ;

          /* Example as to how to restore the world and agent
after it dies. */
          restore_objects_to_world( Flatworld ) ;  /* restore
all of the objects back into the world */
          reset_agent_charge( a ) ;               /* recharge
the agent's battery to full */
          a->instate->itemp[0] = 0 ;              /* zero the
number of object's eaten accumulator */
          x    =    distributions_uniform(    Flatworld->xmin,
Flatworld->xmax ) ; /* pick random starting position and
heading */
          y    =    distributions_uniform(    Flatworld->ymin,
Flatworld->ymax ) ;
          h = distributions_uniform( -179.0, 179.0) ;
          printf("\nagent_controller- new coordinates after
restoration:  x: %f y: %f h: %f\n",x,y,h) ;
          set_agent_body_position( a, x, y, h ) ;     /* set
new position and heading of agent */

          /* Accumulate lifetime statistices */
          avelifetime += (float)simtime ;
          simtime = 0 ;
    nlifetimes++ ;
          if( nlifetimes >= maxnlifetimes )
          {
               avelifetime /= (float)maxnlifetimes ;
               printf("\nAverage  lifetime:  %f\n",avelifetime)
;
```

```
                exit(0) ;
            }



     } /* end agent dead condition */

  fclose(arch);

float   arch3(float   x1,float   x2,float   x3,float   type,float
w1,float w2,float w3,float bias)
{
     float d,y,error;
 if(type < 0)
 {
   d=0.0;

 }
 else if(type==0)
 {
      d=1.0;
  }
  else if(type > 0)
  {
      d=1.0;
  }
  //printf("Desired value: %f \n",d);
 //Activation function

 y=(w1*x1)+(w2*x2)+(w3*x3)+bias;
 error=y;
 //printf("Error: %f\n",error);
```

```
 return error;

}

}
```