Table of contents

1. Intro

This simulation begins by randomizing the chromosomes once the creature is initialized. At the end of each generations, arena mode (modified tournament mode) is used to pick 2 parents and splice the chromosomes). Grid size and generation counts are not too important.

The simulation is 50 X 50 grid to ensure there's a good sample size to breed the AI.

The number of turns per simulation is the default 100, this turn count was chosen because it was the default, it's a nice and easy number to work with during the build. In the first generation, 100 turns per generation is double the max energy count, which shows which creature has been eating or not. Anything survived passed 100 turns can eat and dodge quite consistently.

5000 generation was chosen to find the local minimum and what could probably be the global minimum. It is for data collection.

The simulation is designed so if the tester wishes to change any trait, for example monsters into food and food to monsters or even make the creature cannibalize, the simulation will still breed the adapted creature that avoids food and eats monsters.

2. The Creature

2.1 Percepts:

Percept format 2 was chosen because it was much easier to work with and made more sense. On the contrary, format 1 was too limited in terms of the information given; format 3 cannot tell if a monster is also on the food square. Percept format 2 gives the AI information about each world objections so the AI can learn how to react accordingly.

2.2 Chromosomes:

Each creature has 17 chromosomes, 14 are crucial to the creature's survival and 3 are just bad traits introduced for showcase purposes; it is designed so someone can get a set of chromosomes and predict how it will react:

C_0 - C_7 is the 8 surrounding squares while C_8 is center square, values 0 to 8 is randomized on to these chromosomes. This is used to teach the creature its surrounds.

C_9 and C_{10} is the weight for the first and second 9 squares for the percepts.

C_{11} and C_{12} is the weight for the last 9 squares, with 11 being the red strawberry and 12 being the green.

C15 is the Greediness of the creature, more on this in the **Action** section.

The rest of the chromosomes are just weird little traits added for entertainment, they are weights that are obsolete after 100 generations of breeding.

C13 is the weight for randomness, it maps straight to the last action.

C14 is the weight of laziness, if there's nothing around the creature, this may or may not causes the creature to stand still whether the value is positive or negative.

C16 is just the creature's favorite direction, it adds 0.01 to the direction but after some generations of breeding, they all have the same favorite directions, but it's far better than watching them move diagonally up always.

By looking at the chromosomes, we can tell what the creature likes and dislikes, here's a set of C9 to C12 from one of the creatures:

```
-0.42938066 -0.7236593 0.79620767 0.0037996173
```

The traits modifier e.g. likes strawberries, greediness, laziness etc. is on a -1 to 1 scale. -1 it will avoid and 1 it will go towards

Positive number means that the creature will go towards or have the trait and the negative means avoid or don't have the trait, further explanation is in the **Action** section.

This tells us that this creature avoids monsters in fear of dying, avoids other creatures since they are just blockade, loves strawberries and will only eat green strawberries when there's no red nearby, they are in a safe environment or desperate.

2.3 Action:

Actions 0 to 8:

Since the creature doesn't know how to read the percept array, chromosomes 0 to 7 is randomized to tell the creature at what position should the creature read info about monster, creature and food from.

```
private float directional(int[] percepts,int i){
    int pos;
    float food;
    pos = (int)chromosomes[i];
    if(percepts[pos+18]<=1){
        food=percepts[pos+18]*chromosomes[12];
    }else{
        food=chromosomes[11];
    }
    return percepts[pos]*(chromosomes[9])+percepts[pos+9]
                    *(chromosomes[10])+food;

}
```

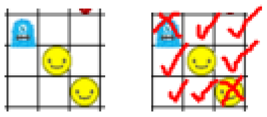
We can simplify this to Monsters + Creature + food, each block is calculated as such

Monsters = monster percept * monster modifier

All three sets of percepts are used in this calculation because they all interact with the creature, monster kills, creature blocks and food nourish or stuns. To survive the monster must know how to react accordingly.

Food is in an if statement because food is the only square that can have 0, 1 or 2 as the value, red strawberries was normalized from 2 to 1 so the creature wont favor the red strawberries.

However, with that calculation it is still flawed since the creature is still not fully aware of its surroundings

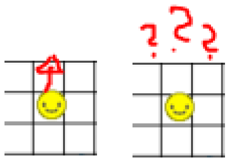


In this picture, the creature in the middle (let's say it hates monsters and other creatures) will avoid going towards the other creature and the monster, but if the creature goes up or left, it could risk getting eaten, if it tries to go right or down, the creature might end up wasting a move since the other creature have occupied it next turn and only one can be in one square.

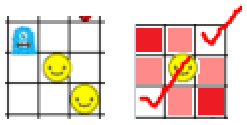
2.3.1 cDirect

To avoid this, cDirect – Centric direction is implemented

```
private float cDirect(int[] percepts,int i){
    int pos = i;
    return directional(percepts,(int)chromosomes[pos])+
            (0.25f)*directional(percepts,(int)chromosomes[(pos-1)==-1?7:pos])+
            (0.25f)*directional(percepts,(int)chromosomes[(pos+1)==8?0:pos]);
}
```



To calculate whether to move up or not, the monster will take the neighboring squares from both side into account.



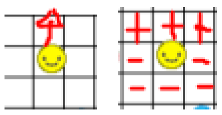
With cDirect, this now translate to go top right or top left.

2.3.2 oDirect:

oDirect (Omni-Direction) is the improved version of cDirect.

```
private float oDirect(int[] percepts, int i){
    int pos = i;
    float returnV=cDirect(percepts,i);
    pos+=1;
    for(int j = 0; j<4; j++){
        pos++;
        if(pos>=8){
            pos = 0;
        }
        returnV -=
            (0.25f)*directional(percepts,(int)chromosomes[pos]);
    }
    return returnV;

}
```



It takes the cDirect of the direction the creature is moving and minus all the other directions for a full spatial awareness. This heavily rely on the creature to accurately guess its surroundings (that requires perfect gene where c0 to c8 has number 1 to 9 with no repeat numbers) or it will confuse the creature, which means that the fittest will have an even better awareness

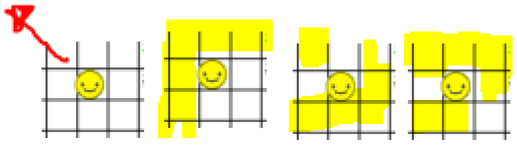
```
2.0 3.0 1.0 9.0 4.0 6.0 3.0 1.0 5.0
```

Example of c0 to 8 and c13 to c20.

In a chromosome, it is highly likely for a few directions to be double up, which serves as a confusion for creatures oDirect and cDirect calculations.

How does cDirect counters duplicated directions?

Each creature has a favorite direction (by default it moves top left if there's nothing around).



In that case, this creature only needs to know how to correctly detects move to the yellow squares to survive, since the blind spots (numbers that are missing from the gene, for example: 7 and 8 from the genes above) are never considered, the monster will never move to the blind spot.

Action 4:

```
actions[4]=chromosomes[14];
if(plantMode){
    actions[4]=chromosomes[14];
}
```

Just for amusement value, this is modified so that the laziness trait can affect this, if there's nothing around, creature with greater than 0 chromosome 22 will chose to stay put while negative chromosome 22 is active.

Action 4 is also used for the Plant mode, more on this in the **Plant Mode** section.

Action 9:

```
int eatPos;
eatPos=(int)chromosomes[8];
actions[9]=(float)(percepts[eatPos]*(chromosomes[9])+
            percepts[eatPos+9]*(chromosomes[10]));

if(percepts[eatPos+18]<=1){
    actions[9]+=(float)(percepts[eatPos+18]*(chromosomes[12]));
}else{
    actions[9]+=(float)((chromosomes[11]));
}
actions[9]*=(chromosomes[15]+(50-energy)/50);
```

This calculation allows the creature to choose what to eat, eatPos establish which tile the creature is on. It is similar to directional where it takes Monster, Creatures and food into account.

The if statement is to separate green and red strawberries since the creature can choose to like or dislike them separately, and since red is detected as 2, it needs to be normalized to 1 so it doesn't prefer red over the others.

```
actions[9]*=(chromosomes[15]+(50-energy)/50);
```

This here is the greedy meter of the creatures; it tells the creature whether it should eat depending on how hungry or greedy they are. This is because if the creature is breed based on staying alive or eating, it will eat regardless of how hungry they are and leave others to starve to death. It also means that if there's a food but there's monsters nearby, the creature will see if it values its life or get energy, without taking hunger into account. This algorithm allows the creature to make decision about whether it should eat or not based on its hunger, its greediness and its surrounds.

Random actions:

```
actions[10]=chromosomes[13];
if(!plantMode){
    actions[(int)chromosomes[16]]+=0.001f;
}
```

Actions 10 takes chromosomes 21 as the modifier, it is uninteresting since it is usually this trait is gone (negative) by generation 100 so it is never used, but teaches us that randomness is a bad trait if you want to survive.

If plant mode is not activated, the creature will take on a favorite direction, again, not too interesting since by generation 100, they all move the same.

3. Arena Selection:

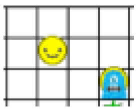
3.1 Elitism – survivors gets to go to next round; the rest is a generation made from arena selection.

Arena selection is a selection method modified from tournament selection.

What is the difference between arena and tournament selection?

Tournament selection, like tournaments has a set rule. This often means that anyone that does not qualify will never be selected for breeding.

The world is too harsh for the creature since it's vision is a 3 x 3 grid based, with limited information this could led to death by pure luck.

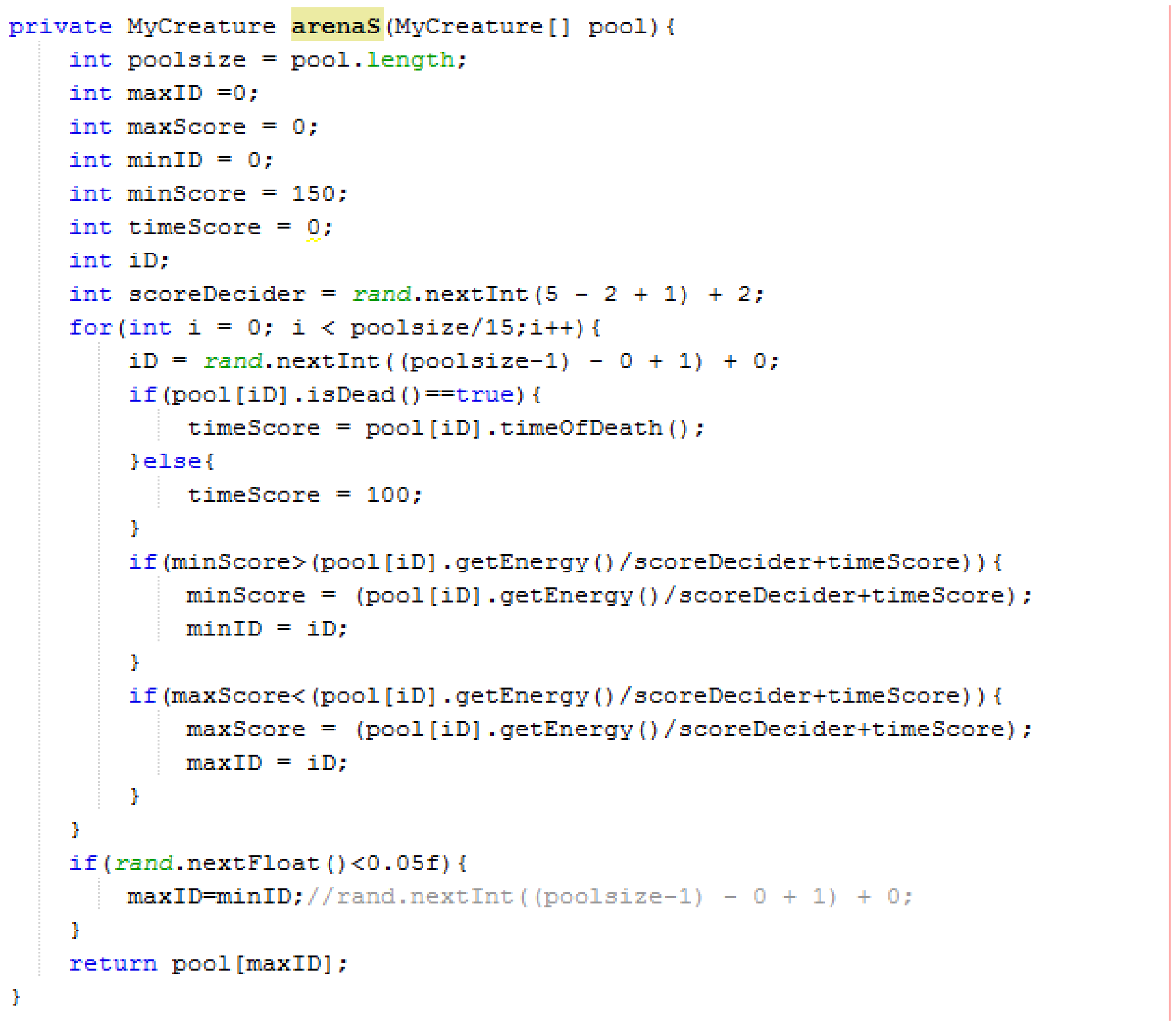

Because this creature cannot see this monster, death by this monster is unfair and it is out of this creature's control.

This means that with the tournament set rule, a creature could have the perfect chromosomes, die due to unfortunate events and never share its perfect genes (this is the worst sentence if taken out of context)

(skip this if you don't care why it is called Arena selection)

Arena is like a fighting pit where rules are more seen as "guidelines". You could be a world champion in boxing, but if the other gladiators walk in with a sword, you might as well be unarmed. And if by sheer luck, if the audience throws a gun into the arena to the weakest, skinniest person, suddenly tides turn and the weak person is the winner.

(name explanation over)



```java
private MyCreature arenaS(MyCreature[] pool){
    int poolsize = pool.length;
    int maxID =0;
    int maxScore = 0;
    int minID = 0;
    int minScore = 150;
    int timeScore = 0;
    int iD;
    int scoreDecider = rand.nextInt(5 - 2 + 1) + 2;
    for(int i = 0; i < poolsize/15;i++){
        iD = rand.nextInt((poolsize-1) - 0 + 1) + 0;
        if(pool[iD].isDead()==true){
            timeScore = pool[iD].timeOfDeath();
        }else{
            timeScore = 100;
        }
        if(minScore>(pool[iD].getEnergy()/scoreDecider+timeScore)){
            minScore = (pool[iD].getEnergy()/scoreDecider+timeScore);
            minID = iD;
        }
        if(maxScore<(pool[iD].getEnergy()/scoreDecider+timeScore)){
            maxScore = (pool[iD].getEnergy()/scoreDecider+timeScore);
            maxID = iD;
        }
    }
    if(rand.nextFloat()<0.05f){
        maxID=minID;//rand.nextInt((poolsize-1) - 0 + 1) + 0;
    }
    return pool[maxID];
}
```

The scoreDecider is used to determine how important energy is compared to the time score, since the biggest part of this simulation is survival and energy management comes after. The energy score is either 5 to 2 times less important to mix up the rules of the strict tournament selection. scoreDecider is randomized to make sure it's always the best at surviving all the time, if the creature score is high it can breed.

The size set for the tournament is set to n = poolsize/15 so it is scalable.

There is a 5% for the loser to win to counter the fact that tournament selection has 0 chance of picking the unfit or the unfortunate fit.

3.2 Chromosomes splicing:

Chromosome splicing is mostly randomized where the daddy parent has 60% of giving the chromosome code except for the 4 important traits of:

Monster modifier

Creature modifier

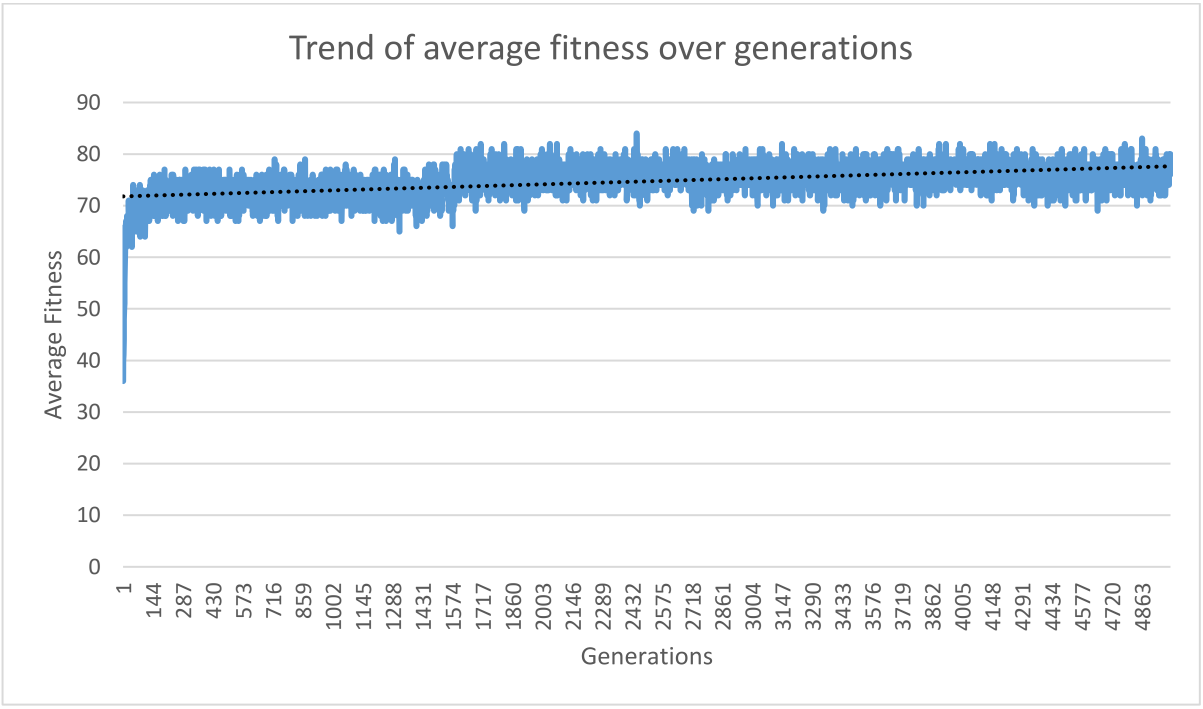
Red strawberries modifier

Eating square

The code will look at the time of death for both parents, the one survive the longest is the daddy and other is mommy, the baby will take monster and creature modifier from daddy and eating square and red strawberries modifier is from mommy. In short, the daddy teaches the kid how to dodge and what to dodge from, and mommy teaches the baby how to forage and eat.

3.3 Mutation – every cell of the chromosomes has a 1% chance of mutating; this is small enough that a mutation would not completely cripple the baby and big enough that it can introduce a positive chance that could increase the survival of the simulation. Mutation works by re-randomize the cell.

4. Conclusion - graph:



In this graph, y-axis is the average fitness and x-axis is the generations. With each generation, the graph overall trends upwards due to a fitter chromosome with higher score is found, then it stays at around the same level until mutation miraculously mutate a baby with a superior chromosome that can score a higher score which further improve the overall fitness from. The overall fitness sometimes drops when the survivors of last round has poor genes and the one with the best genes dies due to bad luck, therefore the gene pool of better genes is now incorporating poor genes, but it easily goes back up after 100 generation breeding. Fitness is calculated by turn count + energy/5

5. Misc.

5.1 GenGraph:

GenGraph is a graphic mod adapted from Robert Young's EvolutionGraph to show the trend, it creates a generation map so you can watch as the generation performance rise and fall. To make the scrollbar appear simply resize the window.

5.2 Plant mode:

```
boolean plantMode = false;
```

To activate plant mode, simply change the Boolean to true at line 18 of MyCreature.java.

Once it is activated, instead of randomizing the chromosomes, all chromosomes are set to 1 and the lazy chromosome 14 is set to 9, since action 4 is mapped to chromosome 14, action 4 would have the largest weight which cause the creatures to stand still and react to nothing (like a plant).

This function is to see how fast can the creatures evolve from plants to intelligent creatures, and to show that nothing in this simulation is hard coded, except for the selection.

5.3 Engine:

Netbeans was used to code everything. I haven't change anything other than adding the GenGraph so it should just run fine like every other Netbeans project files.