Machine Learning Algorithm for Flappy Bird using Neural Network and Genetic Algorithm

INFO6205_233

Name	NUID
Jarvis(Jiawei) Yao	001858168
Boxuan Lu	001833213

Directory

ProblemDiscription

Flappy Bird used to be a prevailing mobile game through the past few years. In this game, players will be required to navigate their bird to pass columns of green pipes without hitting them. Players will get awarded every time the bird surpasses a column of pipe. But in fact, frequent death also tortures players after their impulsive tap spoils everything. Genetic algorithm is a search-based optimization technique inspired by the process of natural selection and genetics. It uses the same combination of selection, crossover and mutation to evolve initial random population. In this case, we will assembly massive bird generations to explore the path towards the end. In addition, birds are encouraged to eat every available fruit randomly appearing in their path.

Overview

The bird class conveys genetic perspective of our project. Core attribute of bird is its gene whose length depends on the moves towards the end. Our birds will lay down their life to find the path and classic pioneers' memory will be carved into genes of descendant. After effort of infinite generations, birds finally figures out the most efficient path and their genes will maintain stable.

GeneExpression

Gene of each bird describes a specific move at every unit step. A bird can choose to flap up, flap down, hover barely or make a move and also find fruit at current position. These **six** independent moves are defined in gene class as **Enum** variable. An array is used to store seperate genes which is stochastically generated and combined in sequence.

Genotype	phenotype
00	Flap Up

Genotype	phenotype
01	Hover
02	Flap Down
03	Flap Up
04	Flap Up
05	Flap Up

Fitness

Fitness strategy is decided to give award to outstanding individuals and do punishment on those with poor performance. In our game, birds will be labeled by scores after it finish game once:

```
public int fitness() {
        // Basic Point
        this.score = getStage().getHerEvilLifStopHere()*20;

        while(!getStage().getEventTriggered().isEmpty()) {
                birdCase bc = getStage().getEventTriggered().remove();
                this.setWholeLife(this.getWholeLife() + " -> "+bc);
                switch(bc) {
                    case Eating: this.score += 66;// Eat fruit will get points
                    case TouchEnd: this.score = this.score*2;break;// Touch end will be rewarded
                    case TouchBoundary: this.score = this.score==0?0:this.score/2;break;// Touch bo
                    case TouchPipe: this.score = this.score==0?0:this.score/5;break;// Touch pipe i
                    case Empty_handed: this.score = this.score<5?0:this.score-5;break;//Bird should
                    default: break;
                }
        }
        // Pass the column will add extra points
        while(!getStage().getDis2edge().isEmpty()) {
                this.score += getStage().getDis2edge().remove()*15;
        }
        return score;
}
```

Evolution

Evolution in natural world means eliminating bad individuals and preserve good individuals. In our game, evolution machanism is built by sort/select/eliminate functions.

Sort

PriorityBlockingQueue is introduced to store bird generation because of its useful features. We have bird implemented Comparable interface and they can be sorted automatically at the same time they are added to this container.

```
PriorityBlockingQueue<bird> pq = new PriorityBlockingQueue<>();
```

Select

We implement two different selection method: Roulette Wheel Selection and Rank Selection.(exact priciple will not be discussed here)
According to the requiredment, we select **half of generation**, which means 500, to be survived and bleed. Bird with higher score is more possible to be selected to survive and reproduce the offsprings.

Eliminate

A **delay elimination** mechanism is implemented inspired by how Java JVM do **Garbage Cleaning**. We don't want to kill those bad individuals instantly. They are allowed to live and contribute their gene for another generation. We set a flag called **qualification flag** on each individual. This flag can be set to four values: newborn, good(to be survived), pending(last for one more generation), kill(will be eliminated after this turn). This is like what JVM has done on those unreferenced instances. Instances will be labeled 'grey' at first scan if it is found unlinked with other one , but they will only be swept out after two consecutive fail inspections.

Crossover

Father and mother will reproduce a pair of offspring after crossover. We will judge on past performance of both gene suppliers and create two different newborns. On each bit of gene, newborns are more likely to inherit from the one with higher scores.

```
public gene[] crossover(bird father, bird mother, boolean father_first) {
        Random r = new Random();
        gene[] x_genes = new gene[generation.stage_length];
        double genetic_prob = (double)father.score/(father.score+mother.score);
        if(father_first) {
                for(int i=0;i<generation.stage_length;i++) {
                        if(i<father.getStage().getHerEvilLifStopHere())
                                x_genes[i] = (r.nextDouble()<genetic_prob?father.genes[i]:mother.ge
                        else x_genes[i] = gene.getRandGene();
                }
        }
        else {
                for(int i=0;i<generation.stage_length;i++) {
                        if(i<mother.getStage().getHerEvilLifStopHere())
                                x_genes[i] = (r.nextDouble()<genetic_prob?father.genes[i]:mother.ge
                        else x_genes[i] = gene.getRandGene();
                }
        }
        return x_genes;
}
```

PS: set **father_first** flag to true/false to breed a pair of offsprings

Mutate

After crossover, newborns still have chance to mutate its gene and the mutate ratio is set as 0.01%. Apart from routine mutate, a large-scale mutation is prepared to help birds jump out from **local optimum** and achieve the **global maximum** scores. The large mutation will be triggered when the deviation of whole generation is small but best score hasn't reached the hreshold we expect for the best score.

Reproduce

Newborns/Alive parents/Random new indeviduals make up the next generation by proportion 5:3:2

Log

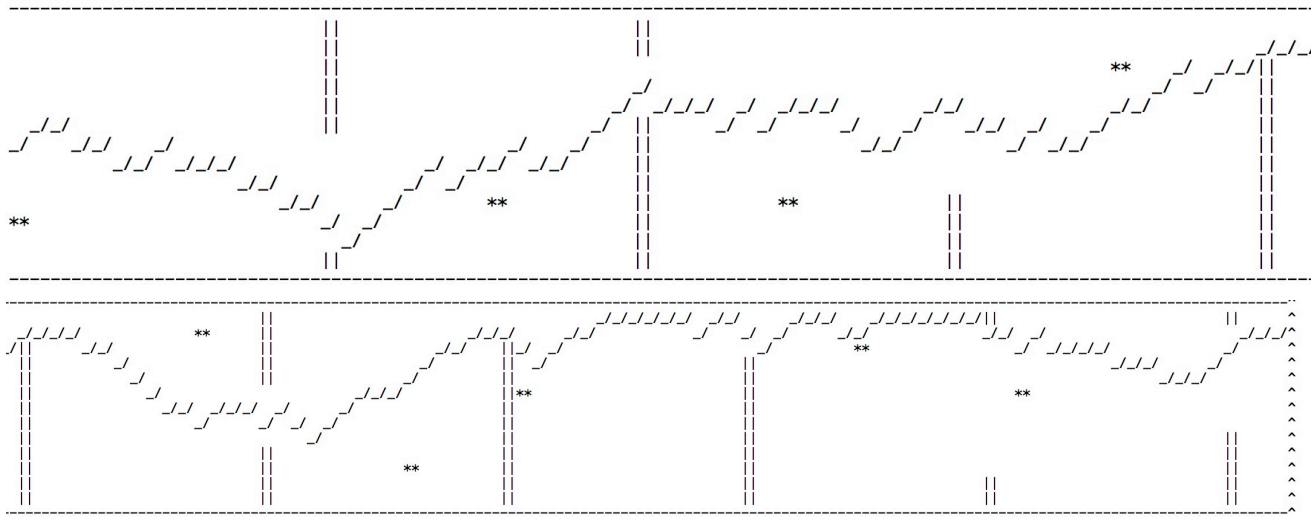
Log files are stored on local path so if you want to run it please make a change anyway.
Logs will be generated and stored in txt format when a higher socore occurs. You can find all the events triggered by this bird throughout his life in our record. Gene sequence is recorded as well.
Log will be something like this:

```
Name: primitive_g0_Thread-1
Score: 481
Whole life: Start -> Flying -> Flying -> Empty_handed -> Flying -> Empty_handed -> Flying ->
Empty_handed -> Flying -> Flying -> Empty_handed -> Flying -> TouchBoundary -> Flying -> Empty_handed ->
```

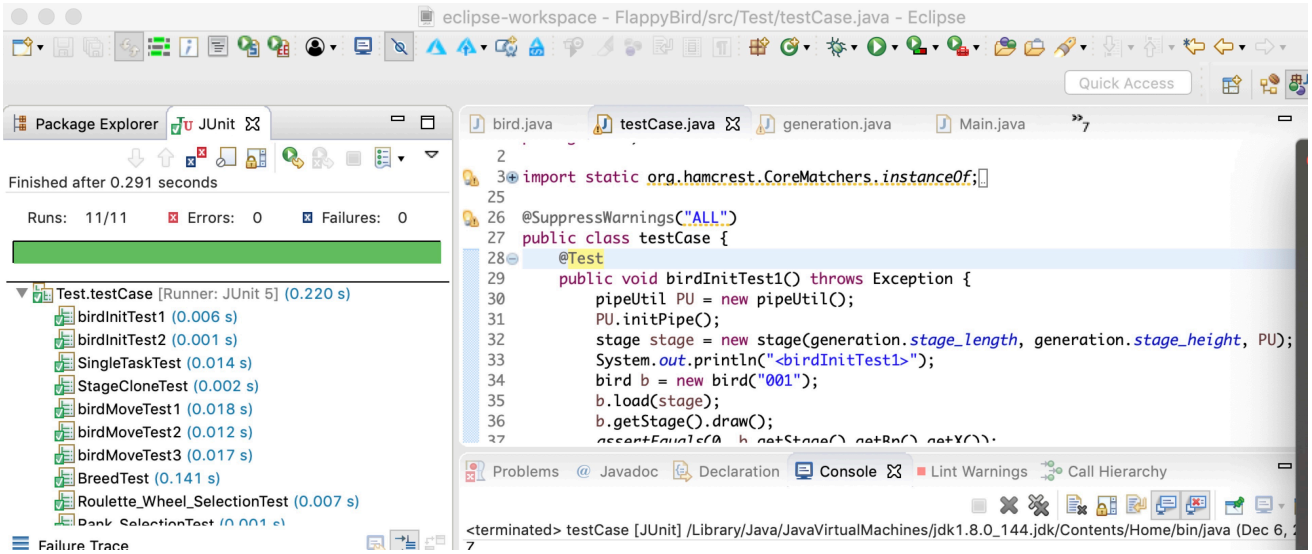
```
Flying -> Empty_handed -> Flying -> TouchBoundary -> Flying -> Flying -> Flying -> Empty_handed ->
Flying -> CrossSlit -> Empty_handed -> Flying -> Flying -> Flying -> Empty_handed -> Flying -> Flying ->
Flying -> Empty_handed -> Flying -> Empty_handed -> Flying -> Flying -> Flying -> Empty_handed -> Flying
-> Flying -> Flying -> Empty_handed -> Flying -> Flying -> CrossSlit -> Flying -> Empty_handed -> Flying
-> Flying -> Flying -> Empty_handed -> Flying -> Flying -> Empty_handed -> Flying -> Flying ->
Empty_handed -> Flying -> Flying -> Flying -> Empty_handed -> Flying -> Flying -> Flying -> Empty_handed
-> Flying -> CrossSlit -> Empty_handed -> Flying -> Empty_handed -> Flying -> Empty_handed -> Flying ->
Empty_handed -> Flying -> Flying -> Empty_handed -> Flying -> Empty_handed -> Flying -> Empty_handed ->
Flying -> Flying -> Flying -> Flying -> Empty_handed -> Flying -> Flying -> Flying -> TouchBoundary ->
Empty_handed -> Flying -> CrossSlit -> Empty_handed -> Flying -> Flying -> Flying -> Empty_handed ->
Flying -> TouchBoundary -> Flying -> Empty_handed -> Flying -> Empty_handed -> Flying -> Flying ->
Flying -> Flying -> Empty_handed -> Flying -> Flying -> Empty_handed -> Flying -> Flying -> Flying ->
CrossSlit -> Empty_handed -> Flying -> Flying -> Flying -> Flying -> Flying -> Flying -> Flying ->
Flying -> Flying -> Empty_handed -> Flying -> Flying -> Flying -> Flying -> Flying -> Flying ->
TouchPipe -> Died
Gene sequence: -> FlapUp -> Hover -> FlapUp_Eat -> FlapUp_Eat -> FlapUp_Eat -> FlapUp -> FlapUp_Eat ->
FlapUp -> Hover_Eat -> Hover_Eat -> FlapUp -> FlapDown -> FlapDown -> Hover_Eat -> FlapUp_Eat -> Hover -
> FlapDown -> Hover_Eat -> FlapUp -> FlapDown -> FlapDown_Eat -> Hover_Eat -> FlapUp -> Hover ->
FlapUp_Eat -> Hover -> FlapDown -> FlapDown_Eat -> Hover -> Hover -> FlapUp_Eat -> FlapUp -> FlapUp ->
FlapDown_Eat -> FlapDown -> FlapUp_Eat -> FlapUp -> FlapDown_Eat -> FlapDown -> Hover -> Hover_Eat ->
Hover -> FlapUp -> Hover_Eat -> Hover_Eat -> Hover_Eat -> Hover_Eat -> Hover_Eat -> Hover ->
FlapDown_Eat -> FlapDown_Eat -> FlapUp_Eat -> Hover -> FlapDown -> FlapUp -> Hover_Eat -> FlapUp ->
FlapUp -> FlapUp_Eat -> Hover_Eat -> Hover -> FlapDown -> FlapUp_Eat -> FlapUp -> FlapDown_Eat ->
Hover_Eat -> FlapDown -> Hover -> Hover -> FlapDown_Eat -> Hover -> FlapDown_Eat -> Hover -> FlapDown ->
FlapDown_Eat -> FlapUp -> FlapDown -> FlapUp -> FlapDown -> FlapDown -> FlapDown -> FlapUp -> FlapUp ->
Hover_Eat -> FlapDown -> FlapDown -> Hover -> FlapDown -> FlapUp -> Hover -> Hover -> Hover -> Hover ->
FlapDown_Eat -> FlapDown -> FlapDown_Eat -> FlapUp -> Hover_Eat -> FlapUp_Eat -> Hover_Eat
```

Meanwhile, log will trace all the process and make a document of the local best one:(in /history/vicissitude)

```
hybrid_g8061_Thread-4 : 6140
```



UnitTest

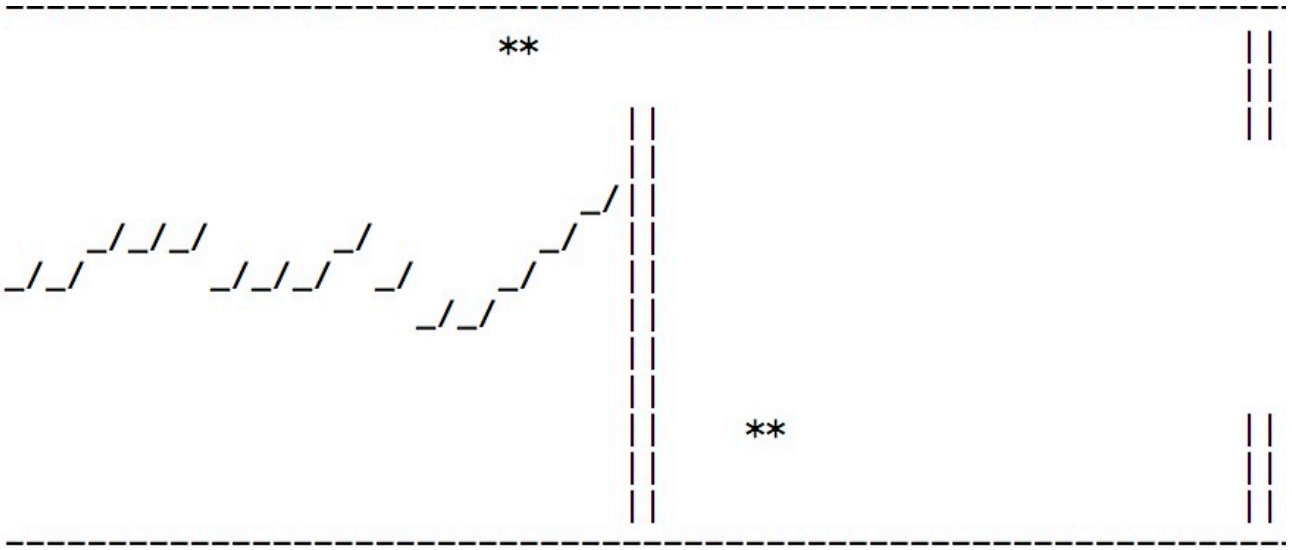


Demo

Parameters:

- stage_length:140
- stage_height:15
- interval between columns of pipes:15
- generation population:1000
- max iteration steps:10000
- mutation ratio:0.01% So there will be 9 columns in total.

- [x] stage 1
  Initially, Initially all the birds would do the "same" wrong thing. So they would all die out quickly.

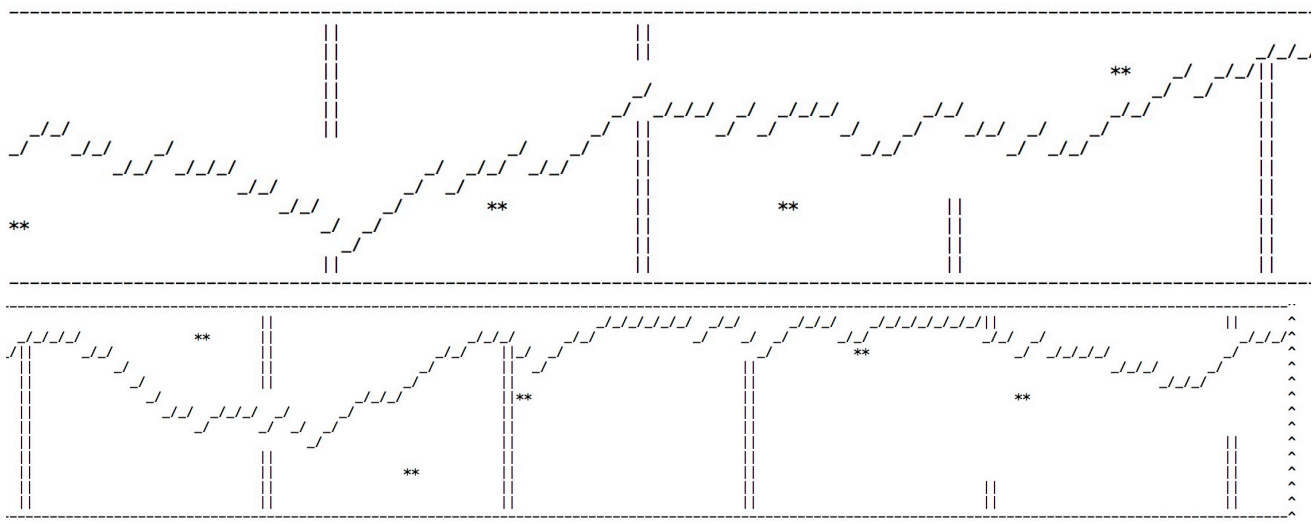


- [x] stage 2
  After some time however, they would start to show more variation, but still perform the wrong moves. This gives us a spread of flappy birds throughout the screen (lengthwise).

- [x] stage 3
  After a bit of training (>**100** times) the spread decreases and is more concentrated at the height where there is a hole in the pipes. They start performing a lot better since they now understand when to flap and when not to.

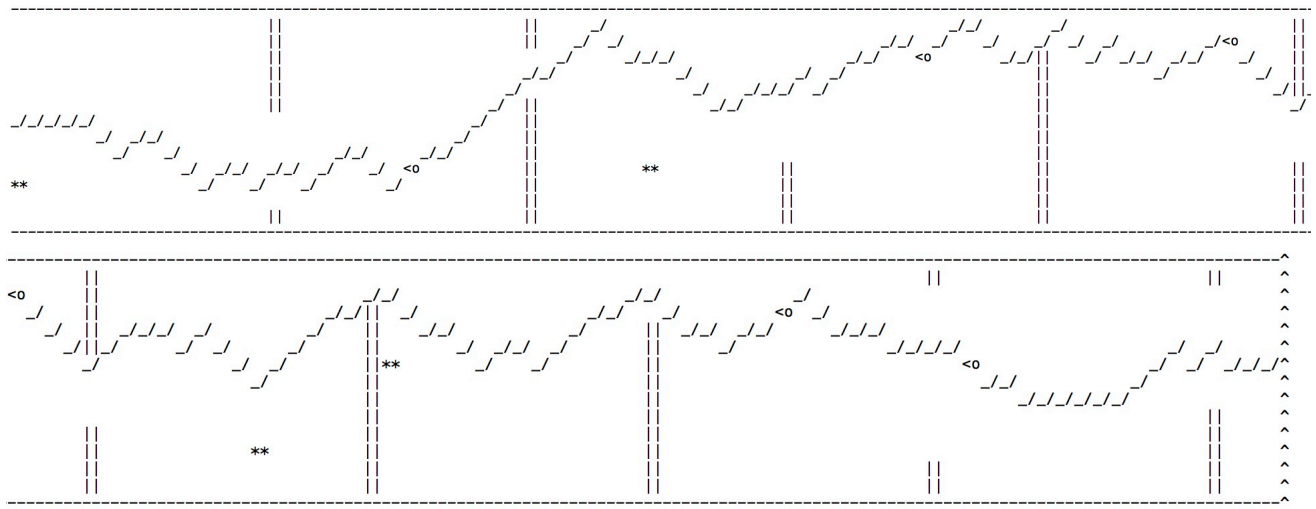
- [x] stage 4

  As time passed by(>=**1091** times), model reach a local optimum, which means bird reach the end but still can improve their scores by eating more fruits and pass the holes more smoothly.

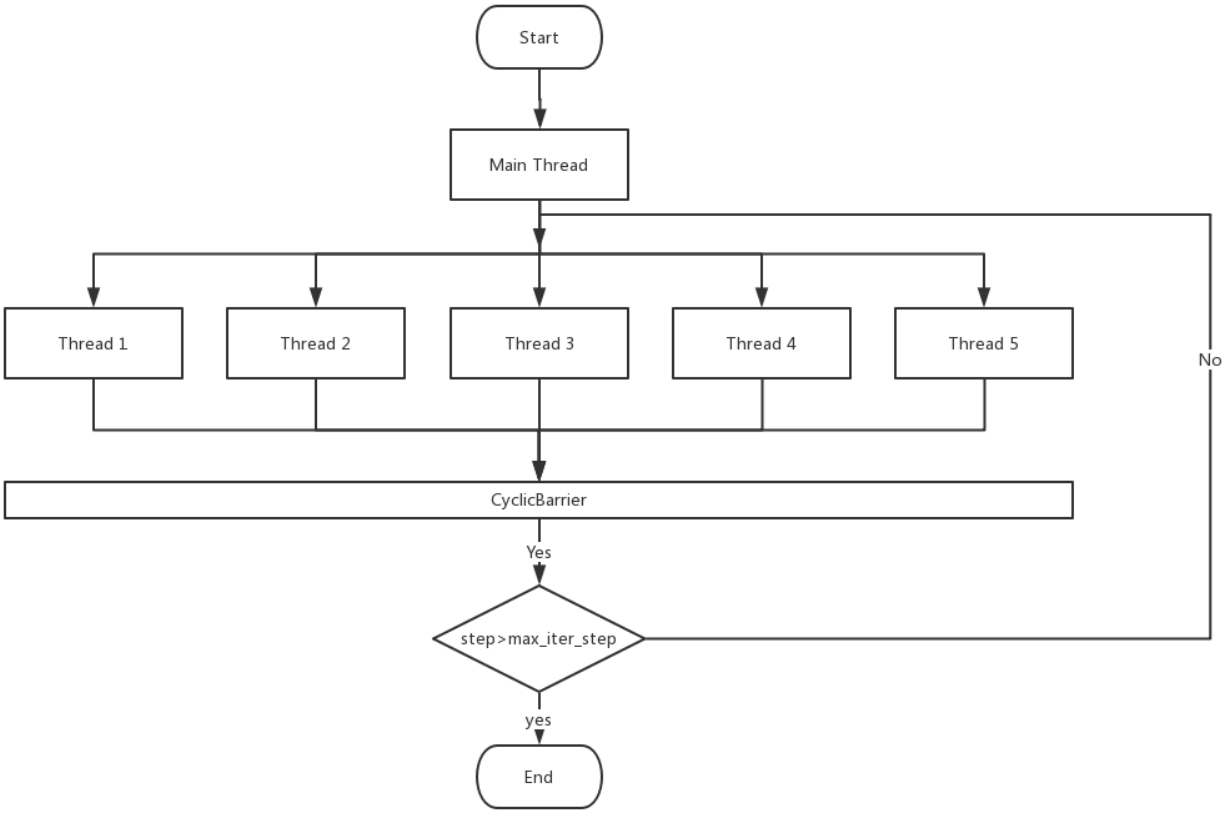


- [x] final but not best stage

  Finally(==**8061** generation), birds have learnt to eat some fruits and be more prudent when passing holes. In this pattern, they can learn to go pass every fixed stage and perform better than what average human can do.



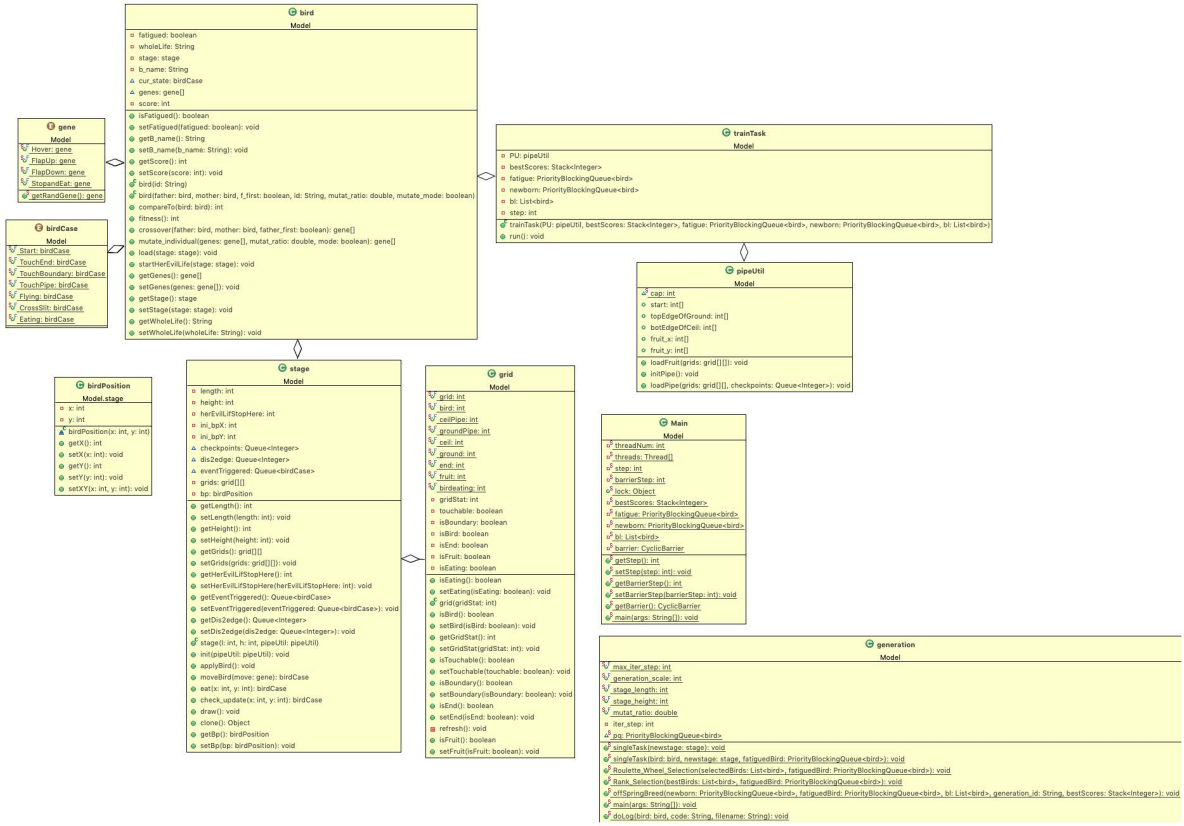
ParallelComputationMechanism



At first, the main thread achieves system initialization, and then creates five threads. For each generation, the population is split into n buckets and each thread performs the fitness evaluation, and selection for all the organisms of one bucket in parallel with all other buckets. We used a CyclicBarrier Object to make threads blocking when the thread finish running task of one generation. Until all buckets are done, the population of each bucket merges back into one sorted solution, generates new generation and releases all threads. When the generation reaches maximum number of generations. All the threads include main thread end.

ClassDefinition

ClassDiagram



grid

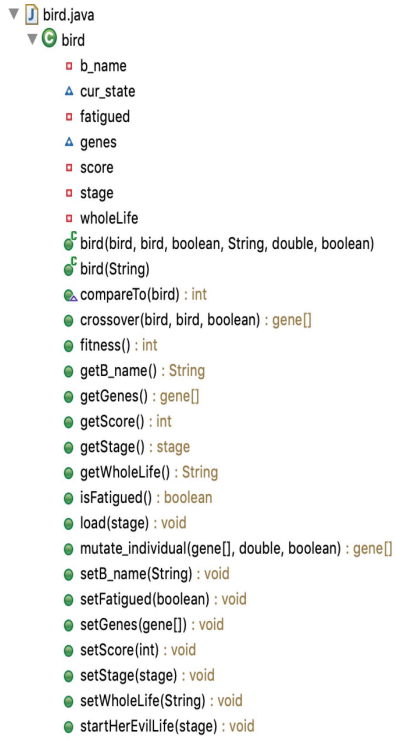
This class difine the inherent property of a single grid which will be used to build the whole stage.

> 00 grid
>
> > 01 bird
> >
> > > 02 ceilPipe
> > >
> > > > 03 groundPipe
> > > >
> > > > > 04 ceil
> > > > >
> > > > > > 05 ground
> > > > > >
> > > > > > > 06 end
> > > > > > >
> > > > > > > > 07 fruit
> > > > > > > >
> > > > > > > > > 08 birdeating

bird

Bird class implemnts **Comparable** interface and override compareTo() method so as to make automatical sort possible in PriorityBlockingQueue. You can also check **mutate_individual()**, **crossover()** function definition here.

fatigued attribute demonstrates whether or not this bird has explored the stage at once.
wholeLife stores all the events triggered throughout last trip
load(stage stage) transmits our birds into constant stage.
startHerEvilLIfe() urges birds to set foot on their trip.



stage

Stage class taks the role of visualization and updating bird position in map. It implemnts **Clonable** interface. Different Grid Mark:

```
public void draw() {
        for(int h=this.height-1;h>=0;h--) {
                for(int l=0;l<this.length;l++) {
                        switch(grids[l][h].getGridStat()) {
                                case 0: System.out.print("  ");break;
                                case 1: System.out.print("_/");break;
                                case 2: System.out.print("||");break;
                                case 3: System.out.print("||");break;
                                case 4: System.out.print("--");break;
                                case 5: System.out.print("--");break;
                                case 6: System.out.print("^");break;
                                case 7: System.out.print("**");break;
                                case 8: System.out.print("<o");break;
                                default: System.out.print("  ");break;
                        }
                }
                System.out.println();
        }
}
```

pipeUtil

Obviously **pipeUtil** is a utility class which defines functions to assist in parameter initialization in stage.

gene

Gene class is actually an enum class which defines gene compositions.

birdCase

BirdCase class is actually an enum class. Stage will return the bird case everytime it moves the bird forward.

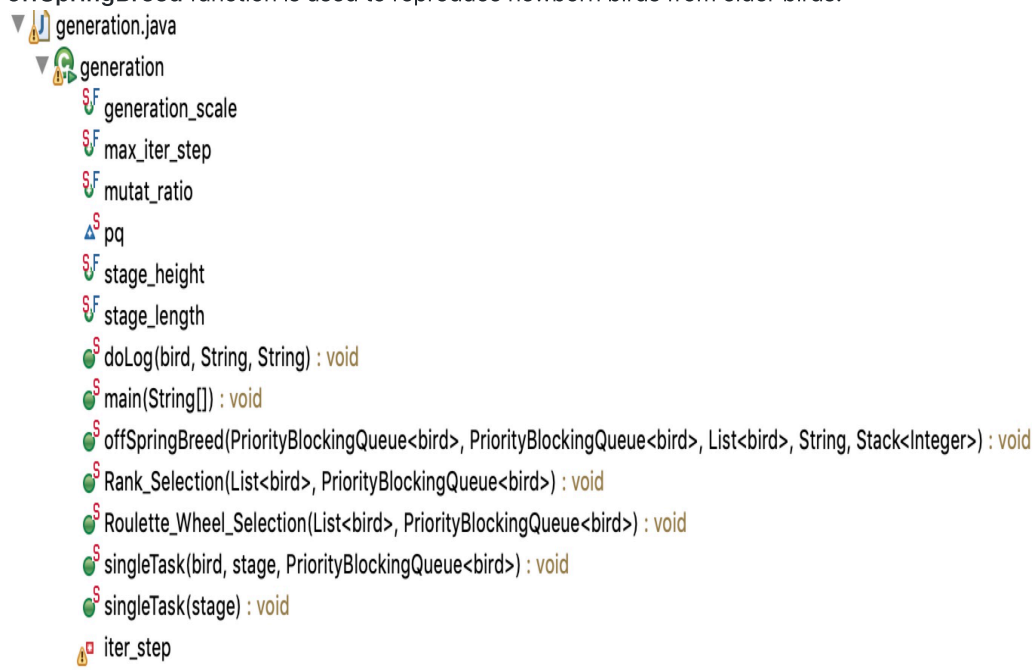
00	01	02	03	04	05	06
Start	TouchEnd	TouchBoundary	TouchPipe	Flying	CrossSlit	Eating

generation

Generation class defines global hyper parameters used in generic algorithm:

```
public static final int max_iter_step = 10000;
public static final int generation_scale = 1000;
public static final int stage_length = 250;
public static final int stage_height= 15;
public static final double mutat_ratio = 0.0001;
```

Also, we implements Roulette Wheel Selection and Rank Selection in this class to pick up genes with highest adaptability. **offSpringBreed** function is used to reproduce newborn birds from elder birds.



trainTask

TrainTask class makes multi-thread available in our project. As we have discussed above, five threads are drafted to run a batch of tasks of one whole generation and then combine the results at barrier.

Main

Main is the dominating controller to run a demo using in concurrent mode.