Info 6205 Data Structure and Algorithm Final Project Report

Genetic Algorithm Implementation

Yiru Cang 001814293 Jiaming Duan 001814402

1. Project Goal:

Choose a sufficiently complex problem, use the idea of genetic algorithm to solve specific problem, and find the best solution to the problem.

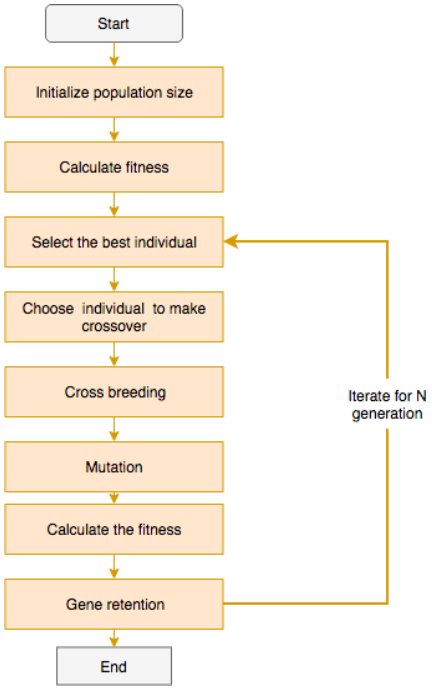
2. Topic and background:

Each player has 3,000 gold coins in the starting state. In the game, each player has 100 kinds of equipment to choose, each of the equipment has different level of price (In our project, the price always flows between 1-100). Players can choose to purchase any number of equipment. Each kind of equipment will make effect of attacking other players with the corresponding damage value (In our project, the damage value of all the equipment is between 1-1000) after purchase.

However, more equipment for one player won't lead to a better effect. In the actual game, too much game equipment will cause the player to walk slower, which directly affects the probability of winning or losing the game.

3. Genetic Algorithm concept & Process:

The genetic algorithm is a computational model that simulates the natural selection and genetic mechanism of Darwin's biological evolution. It is a method to search for the optimal solution by simulating the natural evolutionary process.



Program Work Flow

Generation analysis:

- Genotype: It is the set of replicable and heritable information contained within the cells of an organism [1]. Coding for the 100 kinds of equipment using {0,1} method, 0 stands for not buying the corresponding equipment and 1 stands for buying the equipment, in theory, the genotype has 2^{100} types, i.e. 1.268E30, almost 10^{30} which is already a pretty huge number.
  e.g. For any equipment, $Code_{equip}$: {010010100011110101101011….}
- Phenotype: It is the properties of the organism which are either observable or in some way interact with the environment. In our project, it stands for one of the the player finishes buying the equipment, how many damage value it has and how much money it cost matches the phenotype.
  e.g. For the damage value: [1-100*1000] = [1-1,000,000]
  .For the price: [1-100*100]
- Expression: It stands the "mapping" of genotype to phenotype which one kind of combination can match several kinds of genotype in our project. For example, for the first time, the player buys the equipment with damage value E_1 and E_3 with the price P_1 and P_3, while for the second time, the player buys E_2 with the Price P_2, that might be coincidentally $(E_1 + E_3) = E_2$ and $(P_1 + P_3) = P_2$, with this case, the gene apply is {1,0,1} and {0,1,0} but they have same phenotype.
- Environment: it stands for the natural environment in which an organism list[2], in our project, the overall property the player has (3000 coins) is a limitation, so in the rating process (explain later) will make much effect on the situation that the player spends more than 3000 which just like the filter of the environment.
- Fitness: It defines how well the candidate solution solves the problem, in our project, the higher damage value the player can reach within the limit of the money, the better result it will be. For high-quality individuals, the value should be higher; for low-quality individuals, the value should be lower. In this way, we can judge the merits of individuals through the value of fitness function.

4. Basic API:

Function	Explanation
Generation.init()	Initialize the first generation''
GenerationList.clone(generation g)	Clone a same generation g with new array
GenerationList.mutation(generation p1)	Make mutation to the genes with p1
rating(generation g, Event e)	Fitness function for the rating of children
getchance(Event event)	Change the fitness list into a Monotone increasing List.
pickone(double[] ch)	Implement roulette method to choose parents
evolve(int popSize, Event event)	Make children with the chosen parents
crossover	Make new cross breeding
createCSVFile()	Make the output into csv file

5. Implementation Details:

1* Initialize the gene sequence

```
public void init() {
    for(int i=0;i<equipcondition.length;i++) {

        double r= Math.random();
        if(r>0.5)
        equipcondition[i]=true;
    }
}
```

The code above shows the initialization of the gene sequence, 'equipcondition' is an array with Boolean type, use the random to generate the initial gene with random 0,1. In our project, we define each generation has 1000 children to make the evolve process.

2* Fitness function choosing

```
public double rating(generation g, Event e) {
    int value = 0;
    int weight = 0;
    for (int i = 0; i < 50; i++) {
        if (g.equipcondition[i]) {
            value += e.eventvalue[i];
            weight += e.eventplace[i];
        }
    }
    double score =0.1* value +5*(3000-weight);
    if (min > score)
        min = score;
    if (max < score)
        max = score;
    return score;

}
```

Rating() returns the score for each children of the generation 'value' stands for the damage value and weight stands for the price weight to the overall amount of money. Since the player only has 3000 coins to buy the equipment, all the situation that spends over 3000 will be cause a severe result for the damage value, compared with the price, it should also be considered carefully but it weights less than the price and since the denominations are different so we gave the coefficient of 0.1 when testing, the final program use coefficient of 1 with 7 for(3000-weight).

3* Individual selection

The basic idea in our project to choose the better parents is using the method of roulette, it always controlled by accumulating probabilities, it is usually more likely to be selected with higher fitness. This probability reflects the proportion of individual i's fitness in the sum of the individual's individual fitness. The higher the personal fitness level has， the higher the probability of selection. After calculating the selection probabilities for each individual in the population, multiple rounds of selection are required to select to generate new individuals.

```
public generation pickone(double[] ch) {
    double target = Math.random() * ch[999];
    int lo = 0;
    int hi = 999;
    while (hi > lo) {
        int mid = lo + (hi - lo) / 2;
        if (target < ch[mid])
            hi = mid;
        else if (target > ch[mid])
            lo = mid + 1;
    }
    return parentgenerations.get(lo);
}
```

Each round produces a uniform random number between [0,1] and uses the random number as a selection pointer to determine the selected individual. After selecting individuals, they can be randomly paired (implement with the method evolve) for subsequent crossover operations. Use dichotomy to choose the target value.

For example:

The initial value:	2	5	7	9
Origin-(Max-Min):	0	3	4	7
Probability weight:	0	3	7	17
Corresponding interval:	[0,1.5]	[1.5,5.5]	[5.5,12]	[12,17]
Corresponding Probability:	2/23	5/23	7/23	9/23

4* Cross breeding

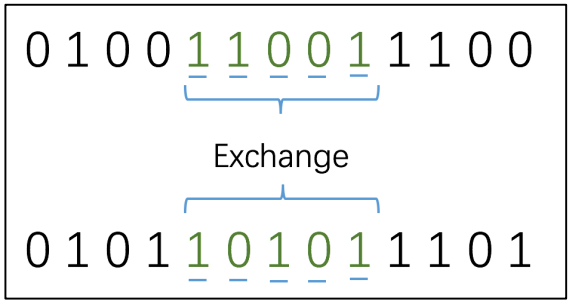
The crossover operator randomly exchanges two genes in the population according to the crossover rate to generate new gene combinations and expects to combine the beneficial genes together [3]. When finish choosing the parent, make new children with the selected parents, in our project, the parents can be the same genotype when belongs to self-breeding.

```
    public double crossover(generation p1, generation p2, ArrayList<generation> child, Event event)
        generation g1 = clone(p1);
        generation g2 = clone(p2);
        int size = 50;
        int a = ((int) (Math.random() * size)) % size;
        int b = ((int) (Math.random() * size)) % size;
        int min = a > b ? b : a;
        int max = a > b ? a : b;
        for (int i = min; i <= max; i++) {
            boolean t = g1.equipcondition[i];
            g1.equipcondition[i] = g2.equipcondition[i];
            g2.equipcondition[i] = t;
        }
        child.add(g1);
        child.add(g2);
        return rating(g1, event) > rating(g2, event) ? rating(g1, event) : rating(g2, event);

    }

}
```

Cross breeding is actually a process about exchanging the gene fragment in the same position of the parents, with the unpredictable probability of the exchanging process, use random function to cut part of the fragment and make exchanging between the parents. After the breeding process, add the child to the next generation.

For example:

With g.length = 13, cross rate = 5/13:



Crossover Example

5* Mutation Process

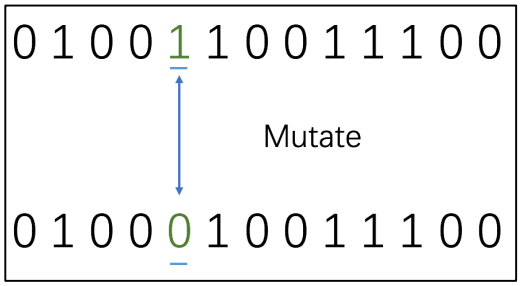
In the generating process, similar to the real situation in the actual environment, the part of gene has very small probability to mutate, which might lead to a new phenotype. The basic idea of the mutation process id one of the gene bit changes in to another type.

```
public void mutationcild(Event event) {
    for(generation g:childgenerations) {
        for(Boolean b:g.equipcondition) {
            double k=Math.random();
            if(k<=0.01) {
                if (b)
                    b = false;
                else
                    b = true;
            }
        }
        if(rating(g,event)>max) {
            max=rating(g,event);
        }
    }
}
```

As the code shows above, for each of the generation g in the childgenration, use random to get a random number to choose if it will mutate, set the mutation probability as 0.01 which is 1%, reverse the corresponding gene to implement mutation process.

Mutation Example:

Suppose g[4] is chosen, the random gives a result of 0.005, so the original gen 1 is mutated in to 0.



Mutation Example

6. Project debugging:

- The chosen of the fitness (rating) function

  In the repeated experiments we found that different fitness definition greatly affects the program results, especially the speed of the curve convergence, that is, find the offspring of the optimal solution which is the number of the generation.

  Since the 'value'(rating result) is nearly 10 times to the price consumption, in order to make them into the same scale for convenient, we add a smaller number 1 and the amplified the scale of the price. We found that the big effect the term (3000-weight) makes, the faster the stable status the result will reach, so we modified each of the coefficient for several times to get a suitable performance.

- The probability of the mutation

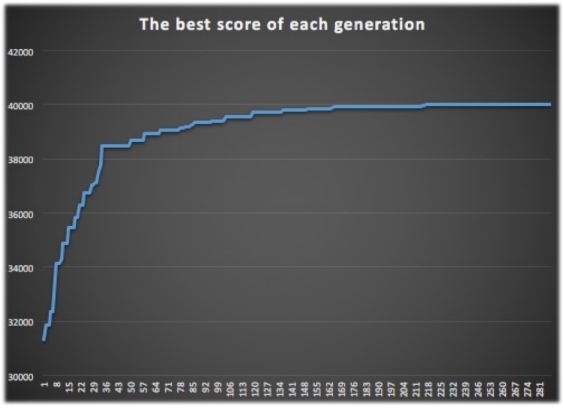
  Mutation probability of a simple gene is important, a big probability of mutation will lead to disorder when making generation, we test and set it to be 0.01.

7. Sample Output:

```
40059.0
40059.0
40059.0
40059.0
The best gene sequence:1,1,0,1,1,1,0,1,1,1,1,0,1,1,0,1,1,1,1,0,0,1,1,1
The money spend is:2908.0
The attack value is:39341.0
```

The figure above shows the sample output result of the beat gene sequence it found and the managing method, 2908 is very like to approach 3000 and it attack value reaches 39341 which seems good. Also, it will automatically generate a csv file to the path parallel with project with the corresponding rating value for convenient.

8. Sample Result:



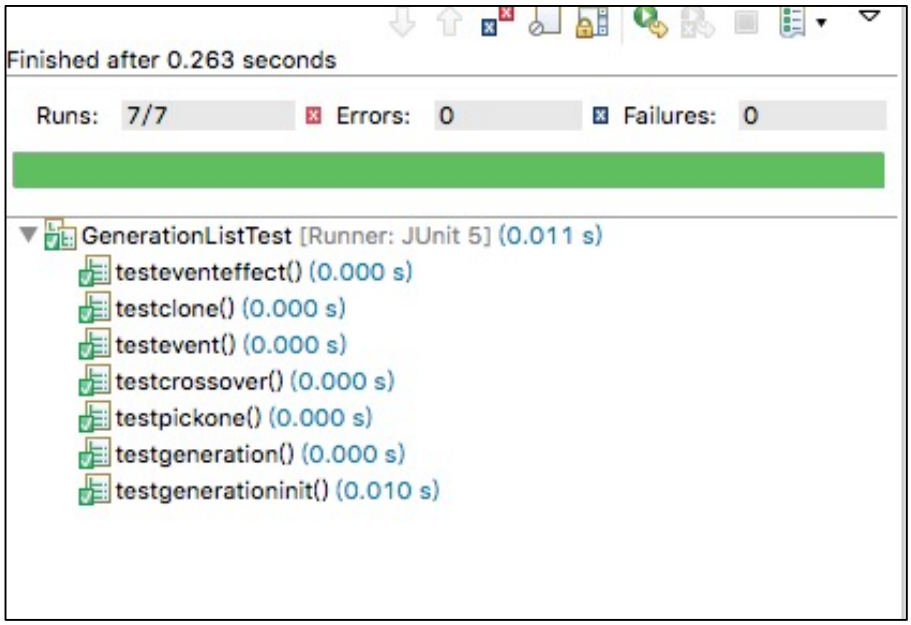
Result of generation selection with Y-axis(rating) & x-axis (generation number)

As the result figure above shows, the original random parents has a lower rating value, it is easy to understand that the very first generations have bad performance with either over the price the player has or less damage value the equipment reach.
With the fitness function and the roulette method, the better parents are more likely to have children with the corresponding high probability to be chosen, and so on for the generations continue, each generation will have a better performance compared with the previous one [4].

However, the result can not be consistently going up, the slop of the rating increasing become smaller and smaller, when it reaches the the beat solution it can find, the rating value will no longer increase making the line as a horizontal straight level.
In out project, the bottleneck of the increasing reaches at about No.268 generation, then the data maintain stable in the later generation.

In conclusion, the Genetic Algorithm can solve this N-dimensional problem, with 100 variable of the equipment, also with the consideration of the money that the player has, also for the damage value that will affect the game result. Each generation has 2^{100} kinds of candidates which is really big set, the initial guess is also important which reflects the relationship between the damage value and the price in our project, a bad initial guess will lead to a bad result that we won't get the desired solution to the problem.

9. JUnit Test Result



We create 7 kinds of unit tests to test the function we have and all of these run smoothly.

a. testeventeffect() test if generate random data successfully.
b. testclone() test if the clone went well
c. testevent() test if create correct number of event
d. testcrossover() test if the crossover create correct number of children
e. testpickone() test the if roulette range is correct
f. testgeneration() test the number of first generate
g. testgenerationinit() test the if the data is successfully been randomly created.

10. Reference

[1] https://en.wikipedia.org/wiki/Genotype

[2] ***Allaby, Michael, ed. (2009). A dictionary of zoology (3rd ed.). Oxford: Oxford University Press. ISBN 9780199233410. OCLC 260204631.***

[3] **https://baike.baidu.com/item/遗传算法/838140?fr=aladdin**

[4] Adaptive Particle Swarm Optimization . IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics. 2009, 39 (6): 1362–1381. 2009-04-07