CS613 - Advanced Concepts in Object-Oriented Programming

Maynooth University
National University of Ireland Maynooth

Genetic Algorithm Project

Supervisor : Robert Cleary

Team:

- Danish Zafar – 19251534 (DANISH.ZAFAR.2020@mumail.ie)
- Yunus Emre Cicen – 20250781 (YUNUS.CICEN.2021@MUMAIL.IE)
- Eyerusalem Hagos YTBAREK– (EYERUSALEM.YTBAREK.2021@MUMAIL.IE)

Project zipped folder “GA_Danish_Yunus_Eyerusalem.rar” includes README.pdf and \doc folder that is generated by Javadoc tool)

You can find the \doc folder for the documentation and the methods where we explained with Javadoc tool.

Github link: https://github.com/yecicen/geneticAlgorithm

Project Outline:

Our Genetic algorithm works by trying to generate “maynooth” by using basic genetic techniques. We have taken an abstract factory of letters/alphabets which produces letter in different languages using specific language factories. For example, our Abstract letter factory creates 'maynooth' using letters and our language factory is responsible for creating letters in EnglishLetterFactory (where EnglishFactory produces letters like ‘m, ’a’, ’y’ etc.) or RussianLetterFactory and IrishLetterFactory etc. using factory method pattern.

The LetterFactory, generates a set of WORDS (chromosomes) with Letters (genes), the collection of Words is population. We have restricted the Letter space to "a, b, c, d, e, f, m, y, n, o, t, h" to reduce the time complexity. We took the population size as 8. Our algorithm evolves by checking fitness as per the match of letters between population and target ('maynooth'). We then sort the population (sorting is according to more matching with the target which we called as our fitness evaluator function) and applied the Strategy pattern to perform Selection, Crossover and Mutation. We have strategy of type Random (), Balanced (), etc. where we Select, Crossover & Mutate either randomly or balanced way. For simplification we just showed random () strategy but we can easily extend and implement any other strategy we want.

Finally after some generations (i.e. After continuous evolving of algorithm by selection, crossover and mutation) we get the target result.

Team Work:

We co-ordinated the whole project by brainstorming in Lab and then implementing the idea. We started the project by researching the links and videos at our own end mentioned in references. Later, we met and finalise the design of it and divided the work. Danish (Id-19251534) did the abstract factory and factory method pattern implementation of letters. Yunus (Id-20250781) and Eyerusalem worked on the GA base development, Singleton pattern and Strategy pattern implementation. We worked together on repl.it and later published results in Github and by helping each other we finished the project.

We have taken care of the marking scheme mentioned in the guidelines:

- Data abstraction used appropriately throughout

When we use the abstract factory pattern for creating letters, we abstracted the information on base class and we implemented on EnglishLetterFactory. We used interfaces to define strategy and the method were implemented in concrete classes.

- Information hiding present and appropriate throughout

We have presented the behavioural and implementation details separate for Strategy and Factory pattern. We presented two types of strategy Random () and balanced () in the interface which were used for on high level for Selection, Crossover and Mutation. Its implementation is done in the respective classes of Selection, Crossover and Mutation.

In factory pattern, we made the Abstract class of LetterFactory which is responsible for creating English letters and Other Language Letters and the factory method pattern is responsible for creating the letters as per the clients language decision. Their implementation is done in the respective English Letter factory and other Language factory class.

- Multiple-file or multiple-module programming solution submitted - classes, subclasses (class hierarchies/inheritance), generics and polymorphism used appropriately to reuse existing design and code throughout.

We have used package (Letters), multiple classes along with few child/ sub-classes to emphasize on inheritance, code reuse. We have shown generics also by making chromosome class accept only type <letter> to be more type safe. Each class has single responsibility with multiple classes, sub-classes and interfaces.

- Programming solution submitted creates data structures that can be objectively judged to be cohesive (facilitate maintenance), facilitate reuse/extensibility, adhere to the principle of substitutability and the open-closed principle – and reuse existing libraries where appropriate.

The code is structured well with comments throughout for better understanding of other person. We have taken care of single responsibility by a class principle and try to re-use the code. For example, Strategy Interface is open of extension of kinds of Selection, mutation and crossover methods we require to add in future. Moreover, out factory can also be extended with more languages without class explosion.

- Singleton pattern correctly applied to allow a single instance of a class to be created only.

We have applied singleton pattern design also to instantiate Genetic algorithm as it should have only one instance of it. We used the eager initialization as it is thread safe and client needs to use it for sure so memory is not wasted.

- Strategy pattern correctly applied to allow dynamic change of behaviour at runtime

We applied strategy pattern to Selection, Mutation and Crossover. We made a Strategy interface for the same which has two methods random () and balanced () used by selection, mutation and crossover depending on the context.

- Factory pattern correctly applied to handle instantiation of GA operator objects.

We applied Abstract factory pattern to generate alphabet letters. The abstract factory of Letter has EnglishLetterFactory and RussianLetterFactory etc. which produces various alphabets/Letter in different language using this factory method pattern to let different factory decide how to create letters. Eg, EnglishLetterFactory generates English letters and similarly, other language factories to create those language letters. We showed RussianLetterFactory just for conceptual understanding.

REFERENCES:

1. https://www.youtube.com/watch?v=UcVJsV-tqlo&t=3s
2. https://youtu.be/uQj5UNhCPuo
3. https://www.screencast.com/t/dVuRVqJmwa
4. https://www.screencast.com/t/jZ6QDJWKn0
5. https://www.screencast.com/t/BlJKzVgwSF
6. https://www.23andme.com/en-eu/gen101/genes/
7. https://www.obitko.com/tutorials/genetic-algorithms/
8. Eric Freeman, Elisabeth Freeman, Kathy Sierra, Bert Bates-Head First Design Patterns - OReilly (2008).pdf