Middle East Technical University Department of Computer Engineering

CENG 443

Introduction to Object-Oriented Programming Languages and Systems

Fall 2020 - 2021

Homework 1 - Gold Wars
version 1.0

Due date: 27 November 2020, 23:59

1 Introduction

In this assignment, you will build an animation about countries buying/selling gold and intelligence agencies intercepting countries' buy/sell orders. When building the animation, you will employ the object-oriented design principles you have learned during the classes. To see the big picture of the animation, a sample implementation is provided.

2 Animation Entities

You will build the animation with the following classes. Do not add new classes or rename/remove the existing ones. You will complete the parts where **TODO** comments are.

1. **SimulationRunner**: Contains the main method. Already filled for you.
2. **Display**: Represents the display on which animation entities, i.e. countries, orders, and agencies, are repeatedly drawn.
3. **Common**: Brings together animation parameters, instances of entities, and other utility fields/methods that are required for running the application.
4. **Entity**: Represents an animation entity. Already filled for you.
5. **Position**: Represents a position (x and y coordinates). Already filled for you.
6. **GoldPrice**: Represents the live gold price that changes at random. Already filled for you.

7. **OrderFactory**: Base class for **BuyOrderFactory** and **SellOrderFactory**.

8. **BuyOrderFactory**

9. **SellOrderFactory**

10. **Order**: Base class for **BuyOrder** and **SellOrder**. An order has a random amount (between 1 and 5), a random speed, and a random path. An order is a filled circle labelled with the amount information and initials of the originating country. While buy orders are green, sell orders are pink. When an order hits the horizontal line, just below the live gold price, the order is executed for the gold price recorded at the hit time.

11. **BuyOrder**

12. **SellOrder**

13. **Country**: Countries generate buy and sell orders at random. A country has a name, some gold, some cash, and a dynamic worth (cash + gold x current gold price).

14. **Agent**: Base class for **BasicAgent**.

15. **BasicAgent**: Represents intelligence agencies (IA). When an IA comes in contact with an order belonging to other countries, the IA steals it. In other words, the IA gains [order amount x current gold price] cash. If it is a buy order, order's country loses while IA's country gains the amount of cash described above. When it is a sell order, order's country loses while IA's country gains [order amount] gold.

16. **State**: Base class for **Rest**, **Shake**, **GotoXY**, and **ChaseClosest**. An IA has a state that it changes at random.

17. **Rest**: A state where the IA stays still.

18. **Shake**: A state where the IA makes random dispositions in horizontal and vertical directions.

19. **GotoXY**: A state where the IA picks a random position as its destination and moves there with a random speed.

20. **ChaseClosest**: A state where the IA picks the closest order as its destination and chases it with a random speed.

21. **AgentDecorator**: Base class for **Novice**, **Master**, and **Expert**.

22. **Novice**: The IA with stolen money > 2000 is decorated with a white badge (rectangle).

23. **Master**: The IA with stolen money > 4000 is decorated with a yellow badge.

24. **Expert**: The IA with stolen money > 6000 is decorated with a red badge.

3 UML Class Diagram

To properly document your code, you are required to draw a class diagram of your application exported as a PDF document. Try to keep your diagram as simple as possible by only including important fields, methods, and relations between the classes.

4 Design Issues

- You are expected to comment your code.
- You are expected to apply Factory and Abstract Factory patterns. Countries should not know what type of order they have generated.
- You are expected to apply Decorator pattern. The IA should not be aware of the fact that it is being decorated by other parts your application.
- An IA should not know what state it has got when it changes its state.
- Decorators, e.g. **Novice**, should not know each other other. In other words, they should not depend on each other.

  States, e.g. **GotoXY**, should not know each other. In other words, they should not depend on each other.
- Your are expected to read the provided images, e.g. country flags, from a directory called **images** so that we can run your code without modifications.

5 Submission

Submission will be done via ODTUClass. You will submit a zip file called **hw1.zip** that contains **uml.pdf** and all aforementioned classes except **Position**, **Entity**, **GoldPrice**, and **SimulationRunner**. A penalty of **5 x LateDay x LateDay** will be applied for submissions that are late at most 3 days.