Battle Arena v2 – The battle continues (Coding Hand in 2)

Introduction

Hello and welcome in our game development company the hippie hipster. You will work on a game named Battle Arena. This is the second version of the game and we want to clean up our code base to make our project more flexible and maintainable for the future. Version 1 of the game was more like a proof of concept and most of the architecture is not easy to extend. But before we talk about the refactoring let us start with the game and the ruleset.

Game Design

Two players playing against each other in a turn-based computer game. Both players start with one coin (coins are the in-game currency) and one hero. If the health points of the hero fall to zero or below zero, this player lose the fight. If both players run out of health the player with the higher health points wins the game.

There are three actions available in the game and in every round each player can do just one action. The list below shows the available actions.

Actions	**Description**
Hit the enemy	The hero tries to hit the enemy with his weapon.
Buy a goblin	The player gets a new goblin into his army.
Buy a leprechaun	The player gets a new leprechaun.

The players get every round new coins to buy something. One coin will be generated from the player himself and every leprechaun generates one coin per round.

Creatures:

Goblin: A goblin has the chance 3:10 to hit the hero of the other player. In every round every goblin tries to fight the enemy. There are three different kinds of goblins, which differ on price and strength.

Name	**Cost**	**Strength**	**Hit Chance**
Tiny Goblin	1 coin	1	3:10
Medium Goblin	3 coins	2	3:10
Strong Goblin	6 coins	3	3:10

- **Leprechaun**: Every leprechaun adds one more coin to the account of the player in every round.

Name	**Cost**	**Generate**
Leprechaun	2 coins	1 coin each round

Weapons

At the moment only the Cynrad Bow is implemented. We want to use the Lathar Sword from an older version of the game and therefor you can find your task in the **Adapter** chapter.

Weapons can be swapped without losing an action and for a better understanding, the stats are listed below:

Name	**Strength**	**Hit Chance**
CynradBow	10	2:10
LatharSword	5	4:10

Your mission

Refactor the already existing solution and clean up the code with the design patterns listed:

Singleton

Its time to implement a logging system into our game. The logging system help the developer to debug the game. The logging system should be available in the whole application and we want to have only one instance of it. The Singleton seems to be perfect for this task.

The Singleton should record the calling order of object instances. You can track the user and time of creation. Write the information to a file at the end of the application.

Try lazy and static initialization and describe your decision and think about the garbage collector and other problems coming apart of the singleton design pattern. The Singleton is also famous to be an anti-pattern.

- **1 Points:** implementation of lazy initialization
- **1 Points:** implementation of static initialization
- **2 Points:** description of the pattern / anti-pattern

Adapter

We want to use a sword from the old version of the game. The old sword class named LatharSword and should not be changed or modified, because it is still in use by the old version.

I've heard that there are class adapter and object adapter available. Find a perfect solution for our project and describe your decision.

- **2 Points:** implementation of the pattern
- **2 Points:** description of the difference between class and object adapter

Observer

To release our game on a current gaming console we need achievements. Maybe the observer pattern could help us out to reach our goal. The observer could watch both players and track some information. If something special occurs, let the player know.

- **2 Points:** implementation of the observer pattern

Bonus:

Feel free to play around with the game rules and settings, if you have any ideas to make the game more fun. The following design patterns can be implemented as a supplement and you can earn **2 points** for each pattern. Don’t forget, there are only up to 10 points available for the whole hand-in.

Pattern Name	**Possible location**
Prototype	It is good practice to save data into a database or into a configuration file. A json file or a sqlite database could be perfect for our goblins. To minimize the access to the external data the prototype pattern suits perfect.
Facade	We are using a console prompt for the user interaction. If we hide the current user interaction behind a facade, it is easy for us to update to a GUI without changing our logic code.
Bridge or Decorator	Our game could be a lot more fun if the weapons have additional items to equip. The bridge or the decorator could be the perfect fit to our problem.
State	There are two different states in our game: the fight menu and the goblin shop. Check if the state pattern would be relevant for our game.