Peer-Review 2: UML and Network Protocol

Andrea Sgobbi, Roberto Scardia, Jonatan Sciaky, Luca Simei
Group 5

9 maggio 2023

1 Positive Aspects

- Revisited Common Goals:
  Grouping common goals in larger categories prevents a lot of code reuse and leaves room for expansion. We think converting to this common goal model was a great idea.
- Splitting up the model:
  Using messages that convey just part of the model, instead of the whole thing, is less wasteful resource-wise and results in a cleaner code.
- Asynchronous communication:
  A positive feature of the communication protocol is that the messages are sent asynchronously, in compliance with the principles of *MVC* paradigm and event-driven programming.

2 Negative Aspects

- General lack of information:
  Both the UML and sequence diagrams lack a lot of vital information needed for someone without direct access to the code base to understand the architecture. Above all, it's fundamentally unclear how the whole architecture is distributed between client and server, given some of the interactions listed. For example, the notify() command which

on the UML takes no arguments while on the sequence diagram seems to send the Command to the controller. Another example is that the Command and Update classes in the class diagram are not connected to other classes and this makes it more difficult to analyze each use of those classes. This may seem like nitpicking, but without being able to read the code these inaccuracies make it extremely difficult to actually understand and evaluate the entire work. All documents are unclear in most parts, for example: in the sequence diagram of the *initial setup* it seems that the server starts the client in some way but this is against the project assignment; we do not understand which method the model uses to notify the view about a change; it is said in the comment that the controller inherits from Obsever<Command> but you haven't provided a custom implementation of the Observer class, for the implementation that was provided requires that the generic type extends from an enumeration type.

- Update methods:
  All the update methods are an implementation of an abstract class. Still, each implementation contains the same methods and attributes: we advise that the parent Update class contains all the interface definitions. Moreover, in the UML diagram we advise that you change the name of the abstract class Update for it can be confused (and often is) with the update() method. It is also unclear how the original class is inferred when receiving Update objects on the client, whether it is using Reflection or Instanceof or other techniques. If the original class is irrelevant because the client only accesses the handleUpdate() method using a command pattern, it's unclear how this would ever interact with the view having no reference to it. It's not clear why the Command class is not immutable: as a data packet, we advise that it should be. The Update class doesn't convey all the information that the client requires in order to implement a complete user interface: it doesn't notify the nicknames of the other players, it doesn't specify whose player is the bookshelf, every implementation of Update share the same exact internal fields inherited from the parent class but it unclear why a *DisconnectedPlayerUpdate* should have a Bookshelf or Board data. We suggest the use of a strategy pattern in the same fashion as the Command class.

3 Architecture comparison

On the surface, this architecture and ours have some similarities; both implement a sort of strategy pattern with a single entry point and multiple data structures with the same interface to convey different update actions in the MVC paradigm. However, it's not clear how you have implemented this feature: if you are not using reflection, your only alternative is to use a series of *switch -> case*. This solution, although it can come naturally to mind, is sub-optimal and against the principle of software engineering. We advise that you use a different approach, implementing a complete strategy pattern and ditching the use of the CommandType enumeration.

Yet there's also various differences: our controller (divided in *LobbyController* and *GameController*) is structured to accommodate the multi-games functionality, while we deduce from the fact that your server contains a *maxPlayer* field that this is not yet accounted for. We do not implement a command handler, instead reflection is used to select the correct method inside the controller's classes to handle the incoming messages.

Unfortunately, we lack a lot of the information needed to make further comparisons, though overall we consider this implementation in line with all the principles the project should follow.