Jonas Ege Carlsen

# TDT4240 Exercise 2 - Patterns

**TASK 1**
I chose to use the Pong game from the previous exercise.

**Task 2**
I struggled with figuring out where a Singleton could fit in. I ended up with creating a class that managed all of the different screens in the application. Even though my application is quite small and not in need of a Screen Manager, having one could be beneficial if the system were to grow later-on. The singleton can be found in the *ScreenManager* class.

**Task 3**
Described in Task 4B.

**TASK 4A)**
**For the patterns listing in Step3, which are architectural patterns, and which are design patterns? What are the relationships and differences of architectural patterns and design patterns?**

To understand the differences and relationships between architectural patterns and design patterns one first needs to understand what they are. One could define architectural patterns as tried and tested approaches to structuring an entire system. Such patterns oftentimes defines proposed subsystems and their responsibilities, as well as providing rules and guidelines for organizing the relationships between the subsystems. Design patterns operate on a smaller scale, where they provide schemes for refining the subsystems or the relationships between them.

Both types of patterns have been created in order to solve recurring problems in technology, although their purposes are different. As mentioned above, architectural patterns are mostly concerned with structuring an entire system, whilst design patterns focuses on the subsystems of that system. One could say that architectural patterns are used for solving technical problems with regards to how the entire system is going to work, whilst design patterns are used to solve technical problems within a subsystem.

The patterns can be categorized as such:
**Architectural Patterns:**
- Model View Controller
- Entity Component System
- Pipe and Filter

The reasoning for defining these patterns as architectural is simple; In order for any of them to work, the entire system must “work together”. MVC aims to separate UI from functionality. If this is only done in parts of the system, the purpose of the pattern is lost. The same goes for ECS and Pipe and Filter.

**Design Patterns:**

- Observer
- State
- Template Method
- Abstract Factory

The design patterns listed above have been defined as such due to their range. Even though patterns such as the Observer pattern can be utilized all throughout the system, it is not necessary to do so. One could utilize such a pattern in only a specific part of the system and it would still perform as intended.

# Task 4B

I chose to implement the Observer pattern. Classes that can be observed or clases that want to observe can implement two interfaces, *Observable* and *Observer*. In my case, *AiBall* and *Ball* are Observables, while *AiBall* and *AiPaddle* are Observers. The Observers are registered in *SoloPlayScreen*.

# Task 4C

In my case, adding the Observable pattern allowed me to remove some redundant code from *SoloPlayScreen* and into the classes where they belong. This improved both code cohesion and readability by moving ball- or paddle-related code into their respective classes.

The pattern also made the application more flexible by lowering the coupling within the game itself. Previously, the game-loop continuously checked if the ball collided with a paddle so that the AI ball could be launched. Once launched, the AI paddle continuously checked if the AI ball had reached its final position, so that the paddle could move towards that position. With this new solution, the ball fires a notification whenever it bounces off a paddle. The AI ball receives this notification and launches itself. Once the AI ball reaches its final position it fires a notification which the AI paddle receives.

The downsides of using this pattern is that it could possibly add more complexity to the application. Even if the pattern is implemented perfectly, the code can become harder to follow, as notifications are raised and received in different parts of the code base.

Raising notifications all the time could also possibly lead to a reduction in system performance, although this won't be a big issue in a tiny application such as this one.

Another disadvantage to be wary of when using observer patterns is race conditions. I did not take this into account for my application as I did not see the need, but for a larger and/or more complex system whom are prone to race conditions, one should ensure that the pattern is implemented correctly.