Design Pattern Discussions

**Builder Pattern**: Builder pattern seperates the construction of an object from its representation. This allows us to change the structure of an object without creating it. Also, if we wanted to change one part for every object, we can do so easily by changing the builder. We use this principle in places where we create complex objects such as Frame objects or monster objects. Likewise, we use pattern in places where we create aggregate objects, such as Grid aggregating Tile’s.

**Observer Pattern**: Observer pattern generates a subscription list, so that if an event occurs, every “observer” within this list is informed of this change. Using the model-view principle, observer pattern ensures that only observers have access to the viewing functionality, and not the models that are affected. For each subject, we either create a new observer object or register one of the existing observer objects. We use this pattern to better seperate UI and the underlying model, by ensuring that the UI gets the information via an observer and vice versa.

**Low coupling**: Low coupling aims to have a software in which dependecy between classes are kept minimal, i.e. one part of the code can work seperately from others. We achieve this by having different classes that handle different part of the software, such as UI and System parts. We only use high coupled classes where we need to, e.g. when we use JavaFX components, or when it makes sense design-wise, e.g. a spesific enchantment is a subclass of the abstract Enchantment class.

**Facade pattern**: In this pattern, only one object exists to receive the system operation, that comes from UI layer. This object is called the overall system or the root object. We will use this pattern to minimize the dependencies between system and the UI. Instead of an object directly accessing the information sent by the UI, which can be confusing since it can be hard to track the changes, we will use a facade controller.