Authors: Pavel Detvan & Martin Popov

Part 1:

1. Factory pattern – allows to easily add new types of blocks without changing existing code; Decorator – only for block classes, this would allow to easily add new features (radioactivity, chrome plated, …) without changing the existing code. The responsibility of creating new types of blocks will be given to the BlockFactory class and the client code won't have to know what the concrete implementation of these blocks is.
2. We need Open-Closed Principle for this problem, this will make all the software entities open for extension but closed for modification. That is why I would choose the Strategy pattern. This enables selecting an algorithm at runtime. Instead of implementing a single algorithm directly, code receives run-time instructions as to which in a family of algorithms to use. Combining Open-Closed principle and Strategy pattern would create a flexible system for representing a robot that can be upgraded with new abilities/features during the gameplay. (It is notable to mention that a Decorator pattern can also be used to solve this problem and it would provide a similar result as the Decorator works on the basis of wrapping an object with another one which can be the robot being wrapped with upgrades)
3. Observer Pattern - defines a one-to-many dependency between objects so that when one object changes state, all its dependents are notified and updated automatically. Having an Observer and Subject interfaces will provide decoupling as the Subject object can notify any observer that implements the Observer interface, without knowing anything about the specific observer's implementation.
4. Strategy pattern would be good for this as it works on the basis of encapsulating different algorithms (in this case filters) in objects which are interchangeable during runtime. The client can provide their desired strategy (filter) which will then be used on each file.