Assignment 1 Design Rationale: CL_AppliedSession10_Group1

Requirement 1 – Spawn logic of new non-player actor classes

The design goal of this requirement is to extend the non-player actor classes, with the creation of an additional two new spawnable classes, 'SuspiciousAstronaut' and 'Alienbug'. We used dependency injection upon the crater class through a 'spawner' parameter upon initialisation. Each actor that can be spawned implements the 'ActorFactory' interface which handles creation of an actor. ActorSpawner class takes Actorfactory creation and handles spawn logic. Abstract monster class describes some non-player actor class that uses the wandering behaviour, interacting with environment around it. Two new spawnable classes extend and implement individual functionality in singular classes, more on this in req 2 design rationale

Open closed principle: Factory pattern interface and abstract base class used to implement creation and spawn logic of actors to abstract away from hard-coding spawn logic to the crater class. This was done as to respect the notion that classes should be 'open for extension but closed for modification.

Singular responsibility principle: New actor classes, 'Alienbug' and 'SuspiciousAstronaut' implement individual logic. This principle highlights the idea that we should gather things that change for the same reason and separate those that change for different reasons. The abstraction to a monster class abstracts a non-player actor, however the individual inheritance of these into individual classes is necessary to implement independent variables of 'Alienbug' and 'SuspiciousAstronaut'. More on this in req 2 design rationale.

An advantageous feature of this design implementation is through the use of abstract. By abstracting logic into a monster class and through a factory class we are future-proofing our code robustness and reusability for future implementations. This would out-perform alternative designs of this design goal such as each new non-player actor class extending from the player class.

As it currently stands, by relying on dependency injection without the use of a list of spawnables, the crater class can only spawn 1 form of non-playable actor. This negates the dependency inversion principle which stresses that code bundles should depend on abstract class and methods instead of dependencies relying on injection from concrete classes from one into another. It also could be argued that this implementation violates OCP, considering the assumption that one crater cannot spawn two 'monsters' at the same time, or that a crater may generate more than one type of spawnable. An alternative design may justify the the use of an Array list of spawnable's and cycling through these to handle spawn logic.