Requirement 2 Design Rationale

Bed Of Chaos implementation

The BedOfChaos class extends from the NonPlayerCharacter abstract class, using existing code to implement the boss without modification of the existing code (adhering to OCP). This also means that BedOfChaos can be treated as any other NPC and have the general NPC methods called (*playturn* and *allowableActions*), adhering to LSP.

To implement the BedOfChaos's attacking tendencies, the BedOfChaos implements HostileAttacker and uses the *canAttackTarget* method to define whether it can attack the nearby actor (if there is a nearby actor, and if it is the player or HOSTILE_TO_ENEMY). The BedOfChaos also adds the AttackBehaviour to its behaviour list, which leverages the existing NPC attacking framework. This adheres to OCP (zero modification of existing code required to accommodate this, just extension), SRP (the BedOfChaos's attacking logic is handled within the methods provided by HostileAttacker, and AttackBehaviour) and LSP (BedOfChaos is accepted in AttackBehaviour as a HostileAttacker). The BedOfChaos also uses the Chaos as its weapon, which expands from InstrinsicWeapon (OCP) to allow it to do damage.

Disadvantages and Justification

- BedOfChaos's implementation of HostileAttacker is a violation of ISP as BedOfChaos's use of *canAttackTarget* is redundant – AttackBehaviour already checks for HOSTILE_TO_ENEMY, and the Player class is already HOSTILE_TO_ENEMY. There is no real special condition for BedOfChaos to attack players. However, to allow for polymorphism and without modification of the existing classes, this must be implemented. It is also possible in the future other HostileAttacker may have their own unique attack conditions (similar to the existing Guts) so the current approach is chosen for extensibility. A NonConditionalHostileAttacker interface could also be made and implemented, however this is considered overengineering given the currently small number of hostile NPCs in the game.
- BedOfChaos directly inherits from NonPlayableCharacter instead of potentially inheriting from a more desirable common Boss abstract class. The current approach was chosen because the assumption is made that any future Bosses will not share many common behaviours. However, if there are groups of bosses that act similarly, potential abstract classes could be made for these groups.

Growth Mechanics

To represent entities that can grow, the Growable interface is implemented. To allow for the growth to fire every turn of the game, the GrowthBehaviour class is created and implements the Behaviour interface, leveraging the already existing Behaviour system (adherence to LSP as the GrowBehaviour is treated as any other Behaviour in the game engine). Finally, the GrowAction extends Action to be treated as any other

Action in the game engine (also adherence to LSP). GrowAction takes in a Growable as an argument and calls *grow* on the Growable, within its own *execute* method to ensure the grow logic fires every turn. This approach adheres to DIP – GrowAction depends on the Growable interface to trigger the growth logic rather than the concrete classes that have growth logic. Overall, Growable, GrowthBehaviour and GrowAction are all responsible for different steps of allowing growth to trigger (SRP), and if any new entities can grow in the future, they only need to implement the Growable interface and the unique logic for their growth – everything else is already handled by the existing classes (OCP).

The Growable interface's *grow* method allows for BedOfChaos to meet the growth mechanic requirements through implementation of this.

Disadvantages and Justification

- Both GrowAction and GrowBehaviour must take in a Growable as an argument and store it, creating an association. This is Connascence of Type (and name) as both these classes must take in the same Growable interface and same *grow* method name. This coupling is considered acceptable, as it allows for adherence to DIP, keeping any concrete implementations decoupled from each other. Growable is also unlikely to change often if at all, so the maintenance cost from this Connascene is low.
- GrowBehaviour's implementation may be a code smell (Middle Man) as it essentially passes the growth trigger to GrowAction which then calls *grow* – not doing much work itself. However, this is necessary as it follows the mechanics of the game engine, where the Behaviour's *getAction* method is designed to allow for NPCs to have actions be triggered that are associated with Behaviours.

Branch, Leaf and BedOfChaos Parts

The BedOfChaosPart interface is created with the *getAttackPower* and *triggerSpecialAbility* methods so every BedOfChaosPart that extends from the BedOfChaos can have their abilities triggered every turn, and so they can contribute to the total attack power of the BedOfChaos. Leaf implements BedOfChaosPart and uses *triggerSpecialAbility* to implement its heal logic. Other new types of parts for the BedOfChaos can implement this and perform any type of effect without modification of existing code as the BedOfChaos is passed through as an argument (adherence to OCP). Additionally, the concerns of getting attack power and the special ability are common amongst all current and future BedOfChaosParts (adherence to DRY and ISP – all methods in the interface needed and used) – and are separated from the BedOfChaos itself (adherence to SRP).

To represent BedOfChaosParts that may grow – the GrowablePart interface extends from BedOfChaosPart and Growable to have these common behaviours – reusing existing code (OCP). Branch implements this interface as it can grow new Branches or Leaves, and still acts as a BedOfChaosPart (has attackPower associated with it). Furthermore, it needs the canGrow method to implement its inability to grow if it has a Leaf attached (adherence to ISP).

Disadvantages and Justification

- Currently the BedOfChaos and Branch may exhibit the God Class code smell (very large *grow* methods that juggle lots of responsibility – pick whether to grow Branch or Leaf, also trigger special abilities and update Chaos attackPower) – if more parts are implemented, a potential GrowthLogic interface could compartmentalise this growth choosing, however currently it would be speculative generality. This is also the only method that gets triggered each turn of the game in BedOfChaos – could move some of the SRP violating functions such as *triggerSpecialAbility*, but this would require a new behaviour, and new sets of coupling – overengineering for this degree of code smell.
- Connascence of Algorithim in BedOfChaos and Branch – they both implement the same 50/50 logic that either grows a Branch or Leaf. It may be possible in future however, that new parts can only grow off the Boss, or the Branch, instead of both – meaning that keeping them separate may be better for extensibility. Additionally, to wrap this logic into one common interface implementation (GrowthLogic as mentioned above) would be speculative generality and overengineering.