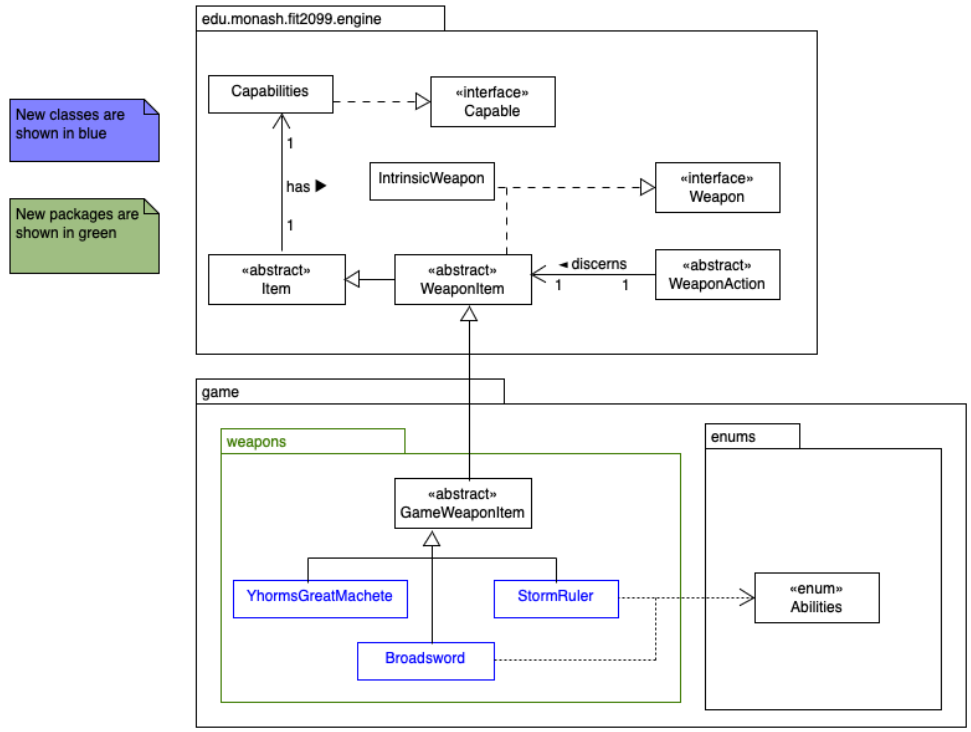
Design Rationale

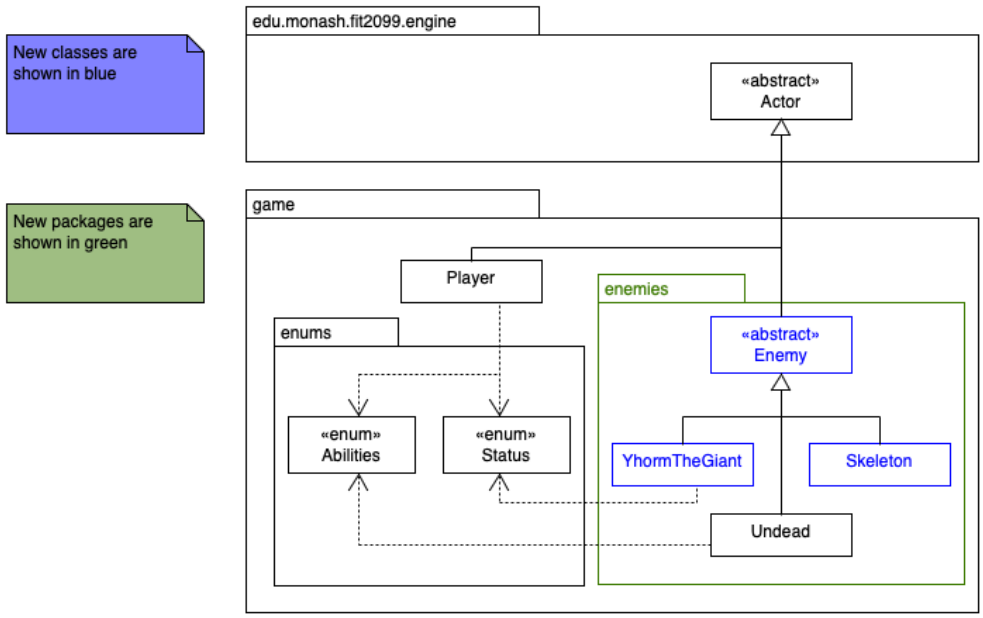
Matthew Crick
Joshua Nung

Weapons



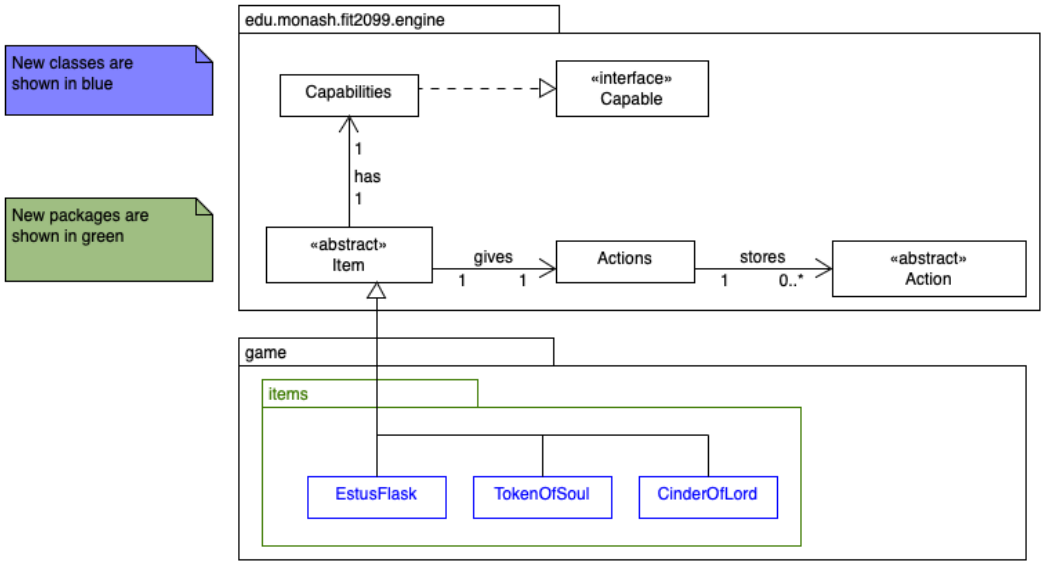
- We will use GameWeaponItem as an abstract class that extends the engine's WeaponItem class, to model the different weapons in the game.
    - This allows us to modify properties common to all the weapons in the game in a single class without repetition. For example, we can override the getDropAction() method to ensure that no weapons can be dropped through an action in the game by the player or enemies. It also means that it is convenient to add new weapons that inherit these common properties.
- To model the passive abilities of the Broadsword and Storm Ruler, the Abilities enum and the Capable interface (to check that a Weapon has a passive ability using hasCapability()) can be used as it represents permanent properties of entities. The Abilities enum is flexible in that different abilities can be added to the Abilities enum, and a given ability can be reused for more than one weapon.
- The WeaponAction class provides active abilities for WeaponItems.
- The IntrinsicWeapon class can be used for the Undead which do not carry weapons.

Actors



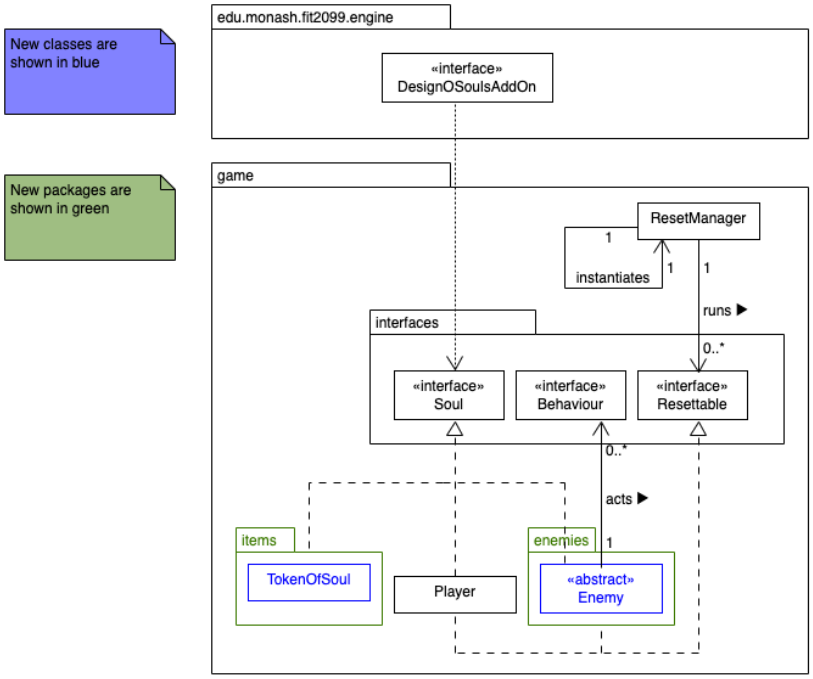
- The actors are the player and the enemies. An abstract Enemy class can be made to be used for subclasses representing enemies. This can provide common properties to all enemies, for example that they are hostile only to the player and not to each other.
    - A future consideration is the possibility of having a LordOfCinder abstract class that extends the Enemy class, to represent different bosses/lords. This may be beneficial if there are common properties that should be implemented for all the future bosses together.
- The Status enum can be used for representing temporary states, such as the player being disarmed while charging Storm Ruler, or Yhorm The Giant being unable to do an action for a single turn after being stunned by the Storm Ruler. The Abilities enum is also useful for implementing certain permanent properties of actors, such as the player being hostile to enemies, and the Undead having a 10% chance to automatically die each turn. The capabilities in these Status and Abilities enums can be reused for different or new enemies with similar properties.

Items



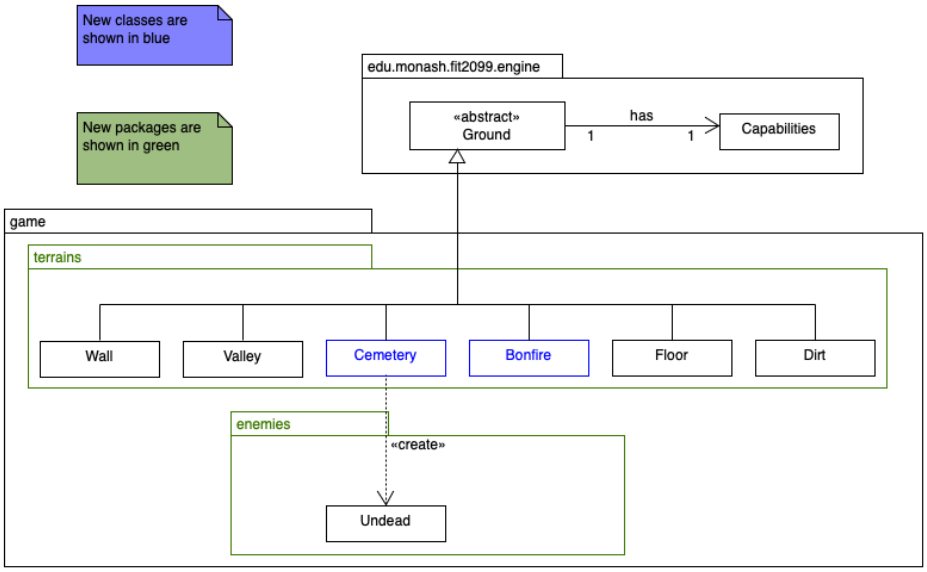
- We extend Item to create a package of items that inherit Item's capabilities and actions as properties. The rationale behind this is to reuse what's provided as an item, using it as an abstract class and extending it to create our own classes of EstusFlask, TokenOfSoul and CinderOfLord.
- The Player Actor interacts with the EstusFlask; though in design these two have been made separate as the idea behind EstusFlask as an item is no different to that of any other item – this may prove to be beneficial when it comes to implementation as the absence of a dependency between EstusFlask and Player allows for changes to each of these classes to be irrespective of the other; endeavouring to improve future re-usability and maintenance.

Interfaces



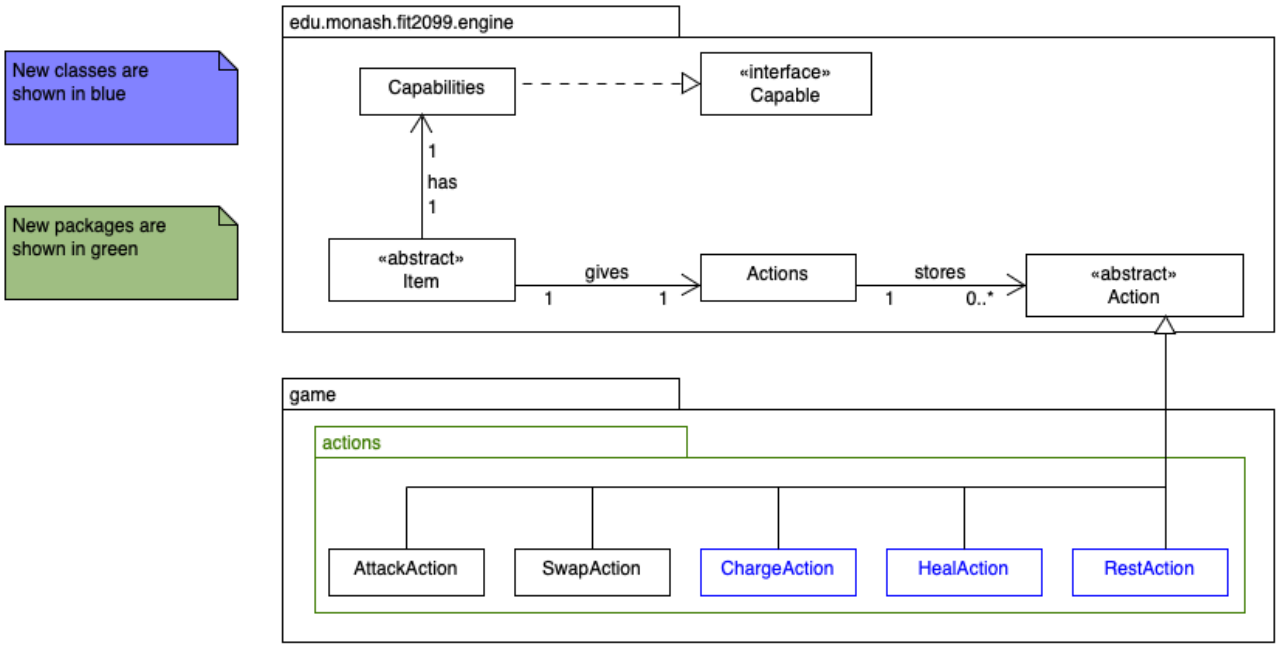
- To implement Souls, the Soul interface which has the methods transferSouls(), addSouls(), and subtractSouls() can be used. It will be relevant for the player, enemies, and token of soul item as these entities will have some amount of souls (represented by an attribute). The reusable methods provided by the Soul interface will be convenient to use within action logic to transfer souls to the player or remove souls from the player.
- The abstract Enemy class can have an (initially empty) attribute for a collection of behaviours that the different enemy classes can inherit. These behaviours are established by the Behaviour interface, whereby classes representing behaviours implement the interface, and can then be reused by different enemies.
- The player and enemies will need to implement the Resettable interface so that when the ResetManager runs, it will be able to appropriately loop through the player and enemy entities and execute their resetting code. This allows each type of actor to provide its own unique logic for resetting.

Terrains



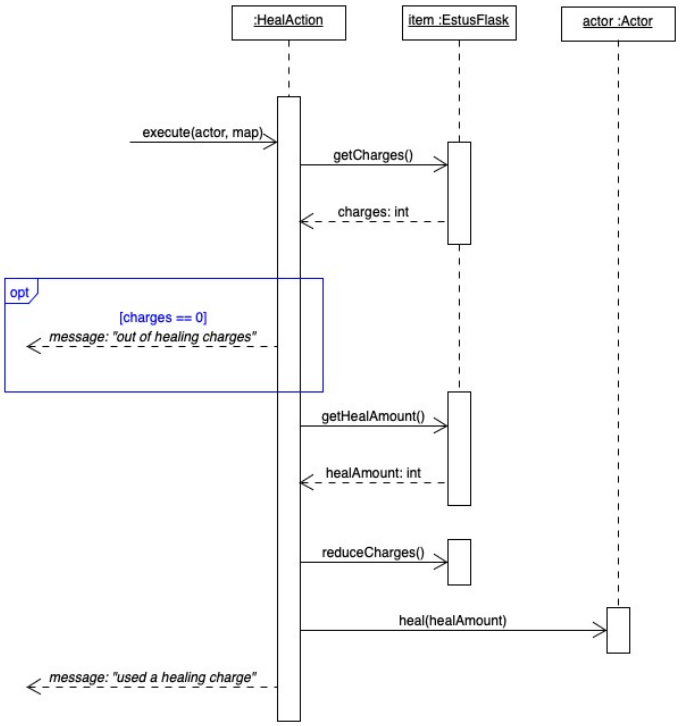
- We will create a Cemetery class and a Bonfire class by extending the abstract class 'ground' to inherit its traits. This is in endeavour to not repeat characteristics that are common. Within Cemetery will be unique properties relevant to only the Cemetery class; its ability to spawn an Undead suggesting a dependency relationship. Similarly, Bonfire is to have its own unique capabilities. Cemetery and Bonfire are to be included in the Terrains package that encloses all sub-classes of ground.
- We wish to adapt our game such that stepping onto a Valley square instantly kills the player; future consideration would be to adapt its method 'canActorEnter' to true such that it allows for a player to enter and upon doing so inherits the 'dying' feature of the game; this feature is to be shared by falling from a valley, killed by an enemy or burned by flames and is possible for valley because it inherits from the abstract class ground who has capabilities; the enum abilities could be adapted such that when a player enters a valley it executes this dying feature.
- While possibly changing 'canActorEnter' to true to allow an actor to step onto a valley; the case of a non-player stepping onto a valley would have to be considered - this is because both Player and enemies inherit from Actor and because of their inheritance relationship both Player and enemies are considered Actors. Thus because we wish for allies or enemies to be unable to enter and have only the Player be able to enter, an alternative consideration would be to utilise a status attached to an actor such that upon entering a valley a ‘Status.FALL’ allows the player to fall by way of a capability check on the players next turn that assesses the status.

Actions



- New Action classes have been made by extending the engine's Action class, namely ChargeAction (for the Storm Ruler), HealAction (for the Estus Flask), and RestAction (for resting at the Bonfire). These actions are fairly general, so that if for example new potions, new weapons that can charge, or new bonfires are implemented in the future, ChargeAction, HealAction, and RestAction can be re-used for them.
- Something to consider is how to implement the Stun action of Storm Ruler and retrieving souls from picking up a token of soul. Rather than creating entirely new action classes for these which may be overly specific and create unnecessary dependencies with the items (Storm Ruler and token of soul), it may be beneficial for now to design these actions by modifying or overriding the existing AttackAction and PickUpItemAction (from the engine) classes.

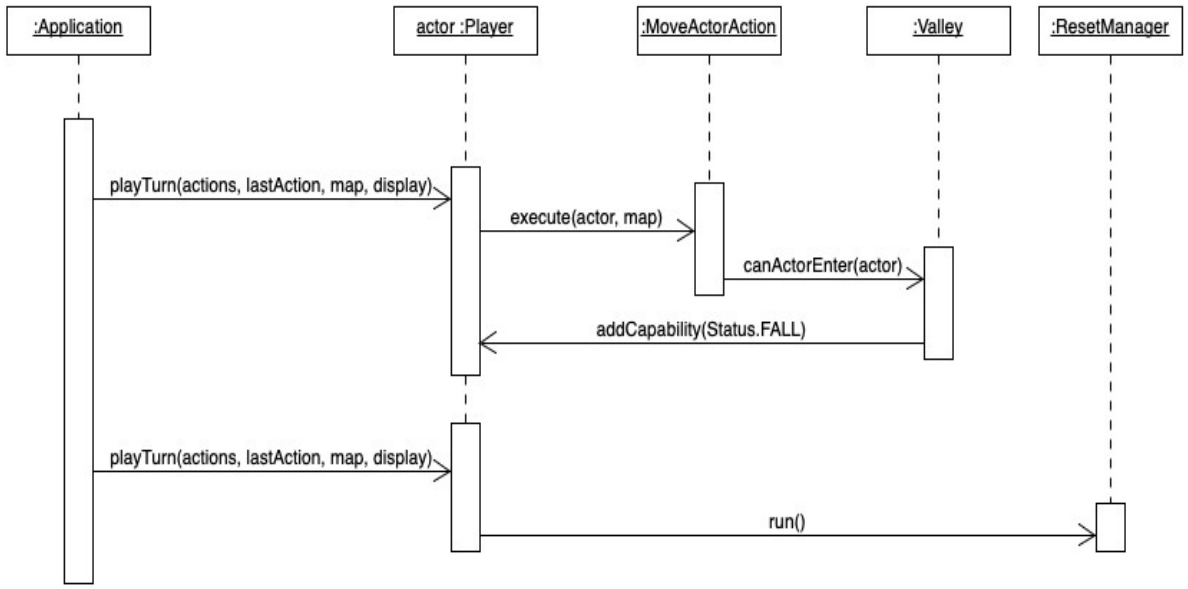
HealAction



- The HealAction sequence diagram makes use of the existing execute and heal methods invoked from within the engine. The sequence follows such that execution of HealAction makes a call to getCharges to return the number of charges from EstusFlask, if equal to 0, output its own message. Then the sequence continues to obtain a heal amount from the EstusFlask, reduces its charges as appropriate and invokes the heal method from within the engine to perform the HealAction on the Actor and produce an output message.

- Attributes such as ‘charges: int’ and ‘healAmount: int’ are to be new variables associated to the EstusFlask class that allow for HealAction to be performed in conjunction with an Actor. Additionally, methods ‘getCharges(), getHealAmount() and reduceCharges() are to be new methods implemented and later invoked for use by the HealAction class.

Valley



- The valley sequence follows such that once a Turn is made for a Player to makes a moveActorAction onto a Valley square, its status is changed to consider this state as per status.FALL. Because Player inherits from Actor and inherits its method hasCapability we will be able to assess this status on the next playTurn for Player in endeavour to return True ultimately allowing the ResetManager to execute its methods to reset the game and effectively kill the player.

- The main design choice lies behind the addition of addcapability(status.FALL) to Player which acts as a switch that turns on the True check on the next turn to run the Reset Manager. The switch allows Valley to remain seperate to all other forms of terrain which are otherwise safe in the sense that they are killing the player.