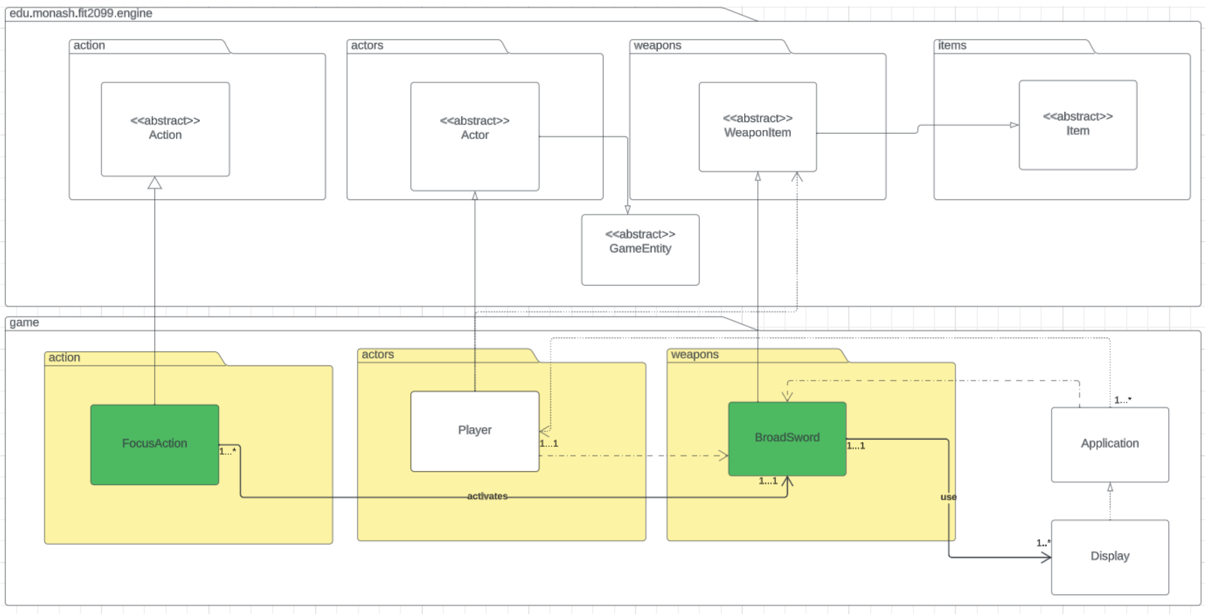
FIT2099: Design Rationale Assignment 1

Requirement 1:



Player Class:
Player was an inherent class that is made by game, it was provided by the teaching team in order to accommodate a playable character in the text-based game “design borne”. The player class utilises a displaychar() to display “@” on the map, with the accommodating base actors attributes to indicate health, stamina, (mana in the future).

The role of Player class is to interact with the environment that the player is in (The gameMap,Item,OtherActors etc), its responsibility is to perform specific actions using a menu-based character input. This allows the user to play as a player, and perform intended actions based on user input. The player is extended from actor which has gamentities, as part of capability set, this means the player can add different capabilities depending on implementation, which can be used to indicate specific status/ability to determine preconditions.

FocusAction Class: FocusAction is a special skill that is provided by the weapon. It is a type of action that utilises 40 stamina per turn, through a selected action that is provided by the menu.

The Roles and responsibility of FocusAction class is to inherit actions from actionList, in order to interact with a weapon, this is a specific skill that can be activated using new actionList, that will inevitably be activated by the tick() function in the game. This FocusAction per requirement is to increase damage multiplier by 5 ticks, that can be activated again to add on damage multiplier to increase damage output.

This is an interaction with current Weapon Item that will be in the actor’s inventory, through the weaponItem it can be identified as the ability of attacking through enum, therefore it is identified as a weapon item that can be held by the actor using weaponItem. It also interacts with item drop and pickup action, as well as GameMap location(x,y) to appear on the current given map, this is because in order for the player to activate the special skill, the actor has to go through the inventory to find the broadsword, in order to get a new return focusAction, otherwise the focusAction will not be available to the actor, as the actor does not have the weapon item and the ability to do FocusAction.

BroadSword Class: BroadSword is an item that is used by the player, this item has a specific enum attached to identify, and grant the ability of attacking. The Broadsword has inherent attributes attached to it, it has specific damage output to the other actor, hit rate as well as verb for the user to be notified of the performing action.

IntrinsicWeapon Class: This class is the only option current player can perform, This is a class we will use to return a Broadsword as an attack option to the other actor, this condition is satisfied only by

These classes work together to satisfy requirement 1, as an interaction between the player, the broadsword, the focusAction. The player first gets to pickup an item that has enumerated type "CAN_ATTACK" , This item utilises the engine class of dropAction to drop the Broadsword at any given time The Broadsword use the enumerated type gives a label to the item in order to identify that specific item has an attack ability. Only the items with "CAN_ATTACK" will return a new attack action utilising a new intrinsic weapon that is overridden in game, that is provided in the engine. Once the item has been dropped, it removes the capability of the item CAN_ATTACK, so the method will know that the actor no longer has the weapon, and thus the access to that specific attack action. This is further utilised by the FocusAction, as FocusAction allows the player to get access as a new action when the actor has the BroadSword, only if the broadsword is accessible by Player, the Focus Action will be added as an option for user input, the FocusAction increases the multiplier of damage by 5 turns, prior to 5 turns the action can be activated again to add on top of the damage multiplier.

Everything is then called by the application, and displayed to indicate user feedback of all actions performed.

Design Principles:
Pros:

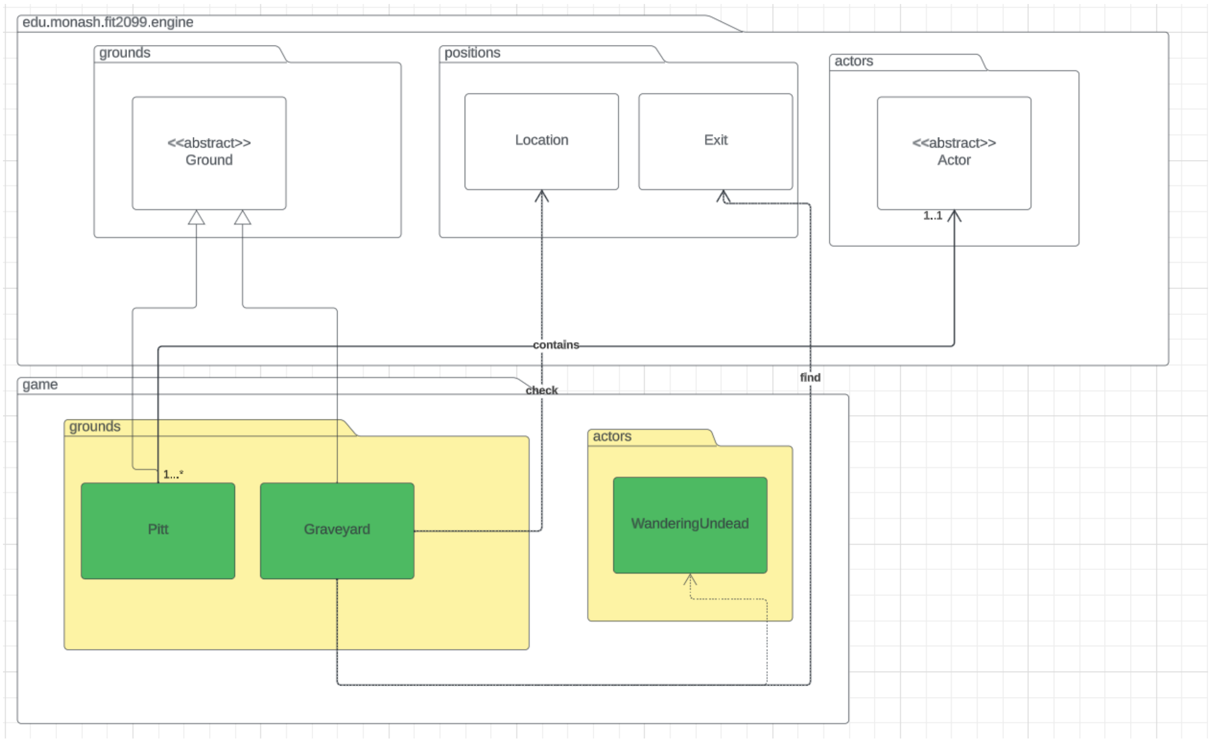
SRP, I have designed every single class to interact with the system using that specific class. This is to keep separation of concern, for example FocusAction can only implement the special focus action, this will make the code more maintainable for the future development.

LSP: The focusAction can be substituted for the abstract action, this is same for the broadsword. This makes code more understandable/readable. This increases consistency and reduces bugs for future development/extension.

Cons:
FocusAction has association with broadsword. One concrete class depends on another, this is a bad design from an OOP standpoint. This breaks DIP, as I haven't even implemented any interface.

FIT2099: Design Rationale Assignment 1

Requirement 2:



Pitt Class: The Pitt class is a type of ground that will instantly kill any actor or otheractor upon stepping into its location, the Pitt will kill the actors instantly as their max health will be depleted as a consequence.

The role of Pitt class is a forbidden area that an actor or otheractor can enter, but will kill upon entering. By adding Pitt, we can identify a new type of ground in our application, so we can use that as a way for the player to enter new ground.

Graveyard Class: The Graveyard is also a type of ground that will spawn other factors depending on a probability on gamestick, every tick will give a 25% chance to raise an otheractor (wandering undead) by adding valid locations that is spawnable, where the wandering undead can enter. .

Wandering Undead Class:

Wandering undead is a type of other actor that is displayed as “t”, wandering undead has a wandering behaviour that allows the wandering undead to go around a new location every time a tick is passed. This wandering undead also has an attack behaviour that is implemented in requirement 2, for the otheractor to attack the player by knowing if the player has a status “HOSTILE_TO_ENEMY” or not. The wandering undead will not attack other actors that do not have the status.

All of the classes above work to satisfy requirement 2, firstly by allowing the pitt to extend from ground and be added to the application, so “+” can be identified as pitt, this is done through overriding the method of displaychar(). The Pitt will ascertain the maximum base actors attribute which is extended to the player as well as other actor, to determine their health point and hurt the passed in parameter, this will indicate that the maximum base Attribute in health will be depleted and set to 0 upon entering, which enacts a is unconscious method, this will remove the actor from the map for the other actor, or display a message “YOU DIED” to the player.

For the spawning requirement, the graveyard has a probability counter, which means for every tick the probability is set to 25. This means there is a 25% chance of spawning otheractor to that specific location after checking if the location is empty or not. (Explained more in depth in coding interview)

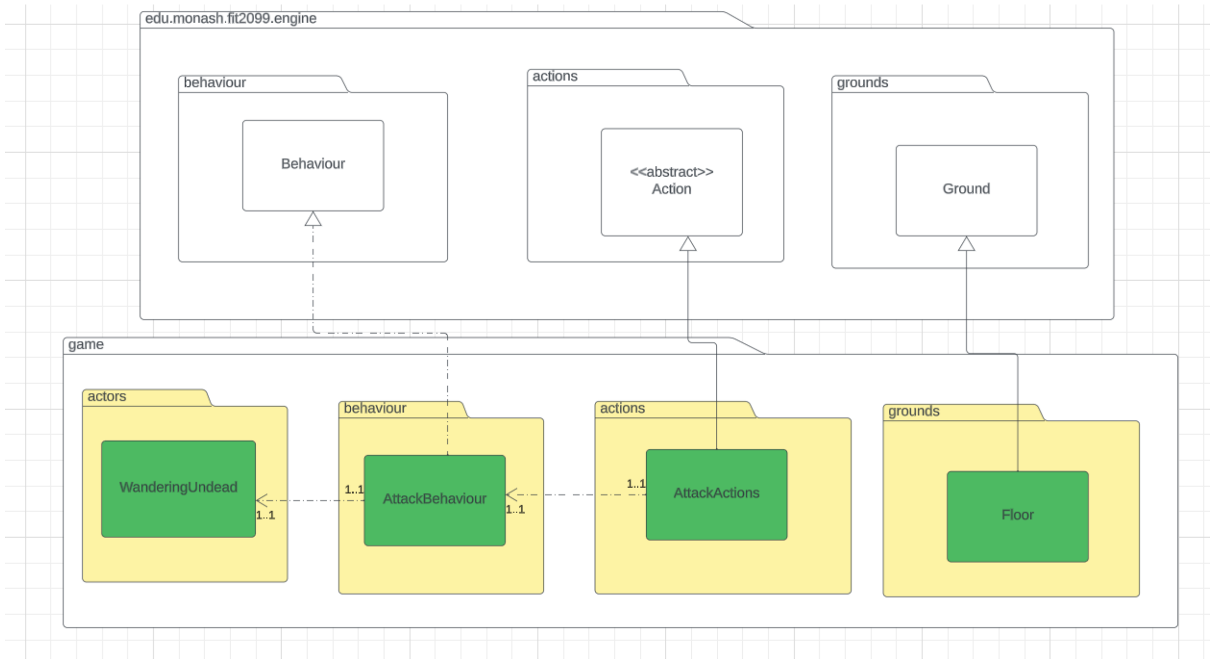
Design Principles:
Pros:

SRP, I have designed every single class to interact with the system using that specific class. This is to keep separation of concern, for example Pitt can only act as the ground, this will make the code more maintainable for the future development.

LSP: The Pitt and Graveyard can be substituted for the abstract ground. This makes code more understandable/readable. This increases consistency and reduces bugs for future development/extension.

Cons:
Graveyard has association with Wandering undead, One concrete class depends on another, this is a bad design from an OOP standpoint. This breaks DIP, as I haven’t even implemented any interface.

Requirement 3:



WanderingUndead Class:

FIT2099: Design Rationale Assignment 1

The WanderingUndead now has the ability to attack a player, by detecting if the player is near the otheractor, The wandering undead has a base weapon that deals 30 damage upon attacking. The wandering undead has the ability to if there is nearby actors in its surrounding, this is used to iterate through the exits that is given by the engine, and if the location contains an actor that has the capability of HOSTILE_TO_ENEMY, wandering undead will attack using its base weapon and hurt the actor with the capability. The wandering undead will not attack another wandering undead, as the other actor does not have the capability of HOSTILE_TO_ENEMY.

AttackBehaviour Class: The attack behaviour class is utilised by the wandering undead to detect the player by getting the actor's location, and then hurt the actor using its base weapon.

AttackActions Class:The attack action class uses target, direction and location which is utilised, to ascertain where the actors are with the specified parameter.

Floor Class:
The wandering undead cannot enter floors, which is presented as a safe space for the actor. The wandering undead is not able to enter floors, as the enemy has an enumerated type that returns false when they try to enter floors.

Design Principles:
Pros:
SRP: all the logic pertaining to attack behaviour which extends from behaviour is contained into one class, which indicates one method for one class is called rather than one class with multiple methods. This makes it easier to extend in the long run, while reducing more complexity in one particular class.

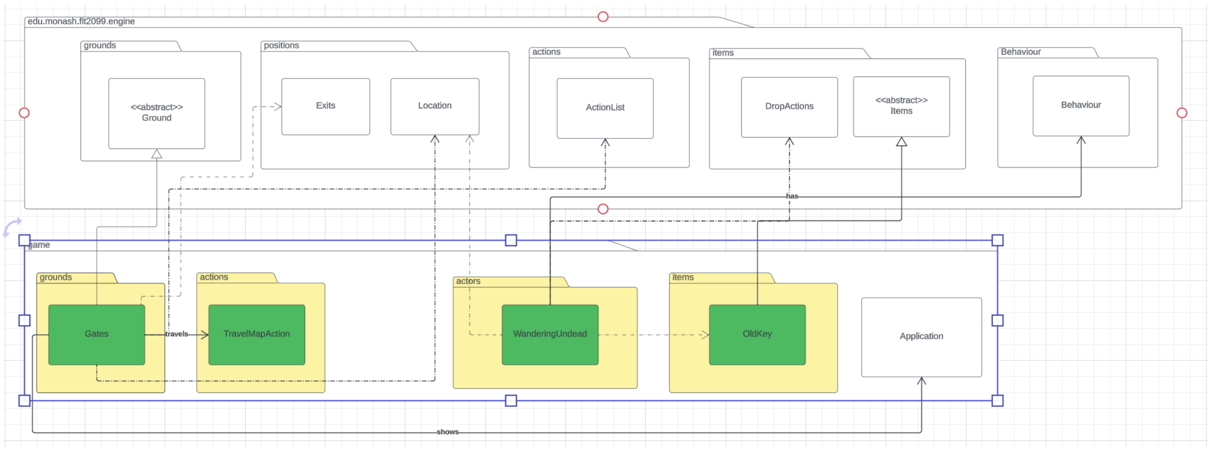
Cons:
This design is really complex, there's too many dependencies around all classes and therefore, it is considered less optimal when we want to make new features or integrate existing classes with current classes in the future.

Requirement 4:

Design Principles:
Pros:
Open close principles, in the travelMapAction, I check the burial ground capability of each ground that is in the map, rather than using if conditions to check display character. Which makes the program open to extension close for modification. This supports the open close principle because I am able to add more functionalities extending the existing system instead of modifying existing code.

Cons:
There's no interface and uses zero polymorphism, so it is hardcoded for distinguishing between new items as we have to downcast.

FIT2099: Design Rationale Assignment 1

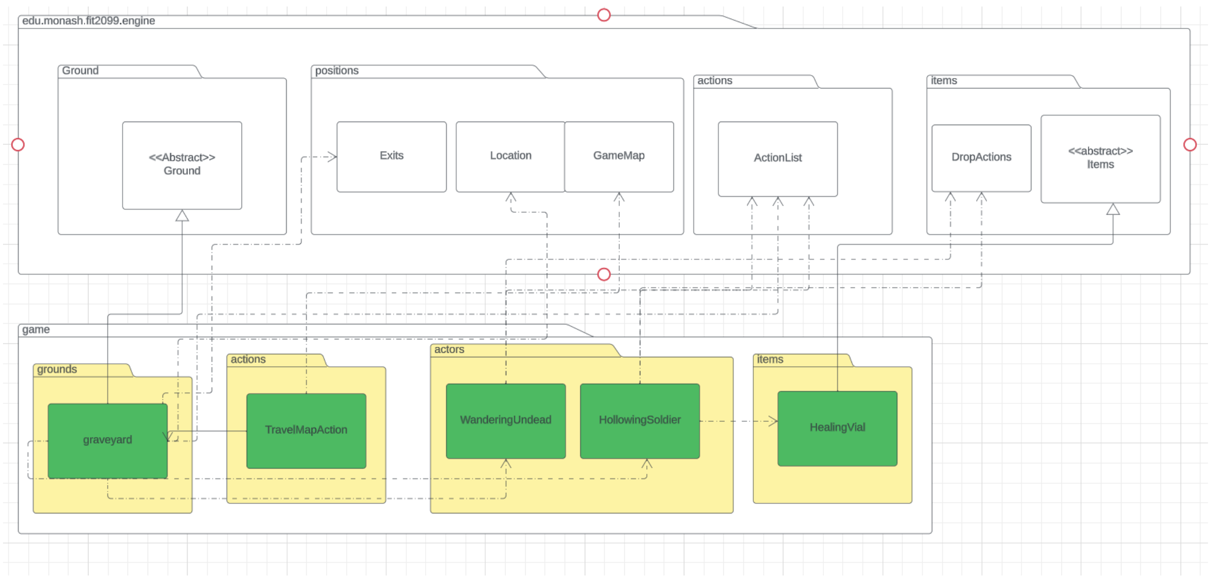


Gates class: Gates class acts as a route to a new map, the gates will extract the new map which is called burial map, and calls the two parameters to setGround, the actor will then travel to a new map after unlocking the gate, as gate has a default capability of IS_LOCKED.
TravelMapAction class: This is the action to moveActors from one map to another.
Oldkey class: Oldkey is used to unlock/remove the capability of IS_LOCKED.
Wandering Undead class: Wandering undead will drop old keys, and this drop can then be used to unlock the gates.

The role of gates is to act as a bridge between the new map and the old, as well as keeping the actors out without an old key that has the capability of unlocking/removing the lock on the gates.

The role of TravelMapAction is to give the users an option to change the map from the initial map to the burial map.

Requirement 5:



HollowingSoldier Class: This is a new class that is made similarly to the wandering undead
HealingVial Class: Healing vial is an item that is consumable.

FIT2099: Design Rationale Assignment 1

Design Principles:
Pros:
DIP: Every enemy whether hollowing soldier or wandering undead has an actionlist, if they want to add more actions they add it in the actor class, instead of depending on actions they can depend on the actionlist to return the list of actions they want to perform.
They depend on high level classes rather than concrete classes.

Cons:
This design is really complex, there’s too many dependencies around all classes and therefore, it is considered less optimal when we want to make new features or integrate existing classes with current classes in the future.