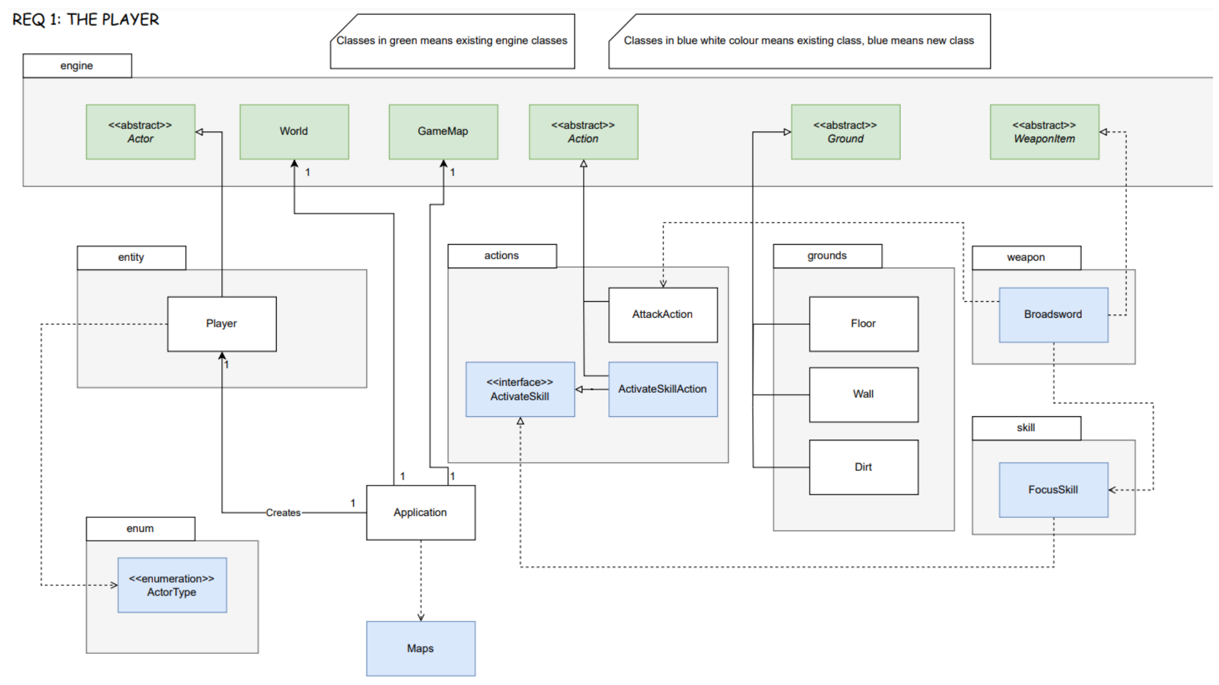
FIT 2099 Assignment 1: Design Rationale

Name: Shannon Jian Hong Chia
Student ID: 32837372

Table Of Contents

Req 1



Brief

The UML diagram represents an object oriented system for Requirement 1. It has 5 new concrete classes, and 1 new interface, with all related classes in the game package categorised into its respective packages, alongside the existing game engine.

grounds

The Ground abstract class plays a crucial role in defining the game's map and terrain. As this provides the fundamental principle and design pattern for us to create any ground related terrain. Classes such as Floor, Wall, and Dirt have extended to this Ground abstract class.

By extending the "Ground" class, it promotes code reuse and ensures that common behaviours are encapsulated within the base class. Additionally, each ground type encapsulates specific behaviours through methods like "canActorEnter". These methods define how each actor can interact with the terrain, establishing rules for movement and access.

To further add on, the use of extending to the abstract class encourages polymorphic behaviour for future implementation. For instance, the creation of "Void" (Req 2), can be easily implemented with the required methods. Hence fostering extensibility in the ground related mechanism.

More importantly, this design adheres to essential object-oriented principles, including abstraction (abstract class), inheritance (class extension), and polymorphism (method overriding), which allows for a cleaner, modular, and maintainable code.

enum

ActorType

The “ActorType” enumeration categorises various types of actors within the game. It assigns a distinct label to each actor, such as “PLAYABLE” for player-controlled character and “ENEMY_WANDERING_UNDEAD” for hostile creatures.

The purpose is to distinguish between actors based on their roles and attributes within the game. By using descriptive labels, the enumeration enhances code readability, as it allows others to understand the role and behaviour of actors by examining their associated “ActorType”.

This class serves as a classification system, focusing on only one aspect of the game’s design (SRP). Additionally, it allows for new additional enum values as additional actor types might be introduced into the game, which doesn't require modifying existing code, ensuring that the system remains closed for modification but open for extension (OCP).

entity

Player

The addition of the “PLAYABLE” actor type specifically for the player, allows players to be distinct from other actors, such as enemies or neutral non-player characters. This is crucial in ensuring that certain interactions within the game world are available exclusively to the player character. This permits the game engine to differentiate between what actions and interactions are permissible for the player and what is not.

This design follows the dependency inversion principle (DIP), by allowing the game engine to depend on high-level actor types, rather than concrete player character classes, allowing for better flexibility and maintainability.

Weapon

Broadsword

The Broadsword is designed as a subclass of the “WeaponItem” class, adhering to the inheritance principle. This hierarchy allows for the specialisation of weapon attributes while inheriting common weapon functionality. Moreover, the Broadsword contains a AttackAction and ActivateSkillAction (FocusSkill), that only allows the wielder to have access when it is picked up and is adjacent to a valid target (OCP).

Actions

AttackAction, ActivateSkillAction extends the abstract Action class to enforce common methods, allowing for polymorphism through abstractions (DIP).

AttackAction

Performs a fixed amount of damage on the enemy, with the same accuracy (DRY).

ActivateSkillAction

The ActivateSkillAction class represents an action that allows an actor to activate or reactivate the implementer of the ActivateSkill interface (ISP). It extends the Action class and provides methods for executing the action and describing in in the game menu. This allows ActivateSkillAction to invoke the ActiveSkill interface's method when executed, adhering to SRP in implementing classes.

When this action is invoked, it will perform the selected skill buff on the player.

skill

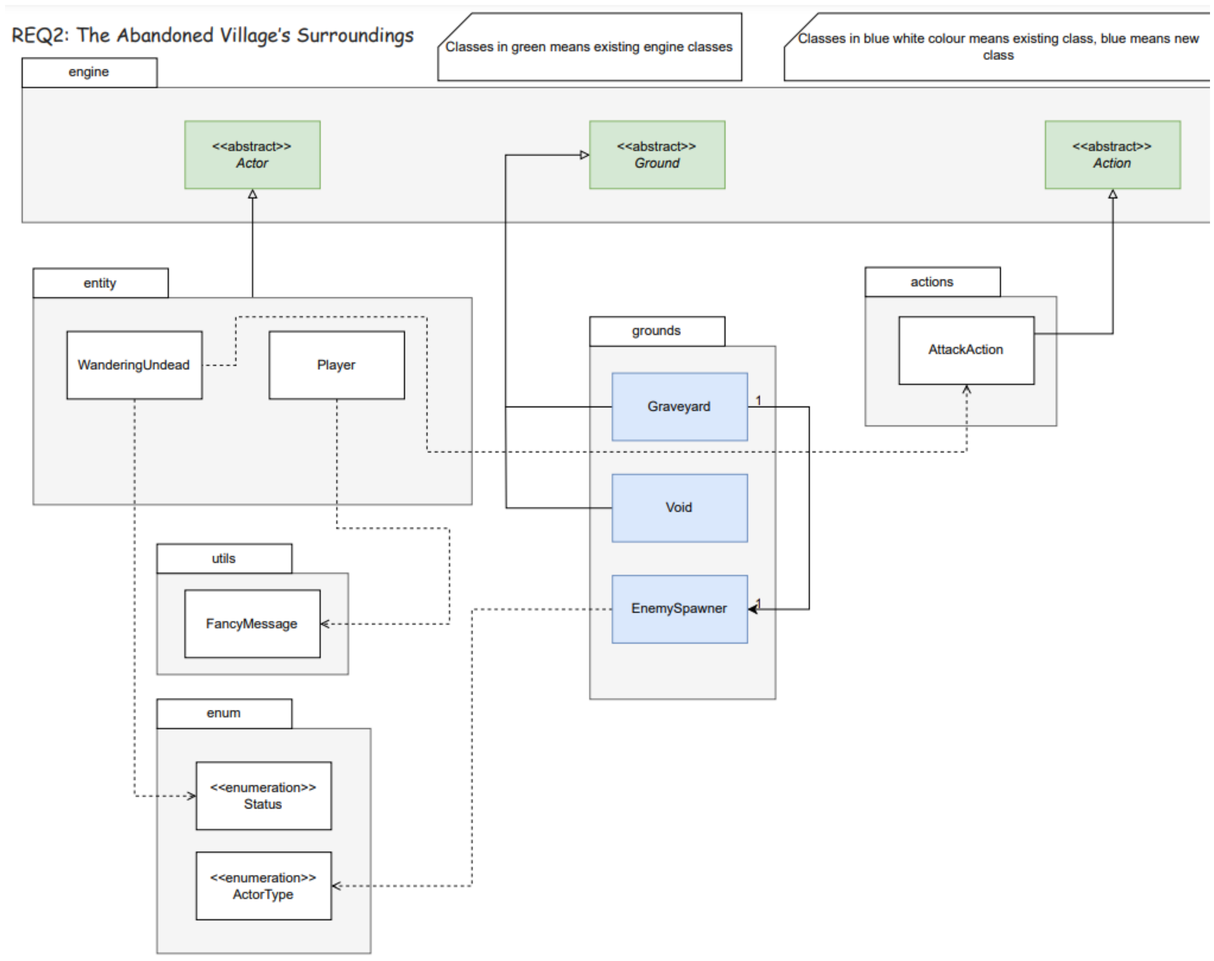
FocusSkill

The FocusSkill class represents a skill that can be activated by an actor, providing enhancements to a Broadsword weapon. It implements the ActivateSkill interface and allows the actor to focus their attacks with the Broadsword. This design adheres to the SRP by focusing on the skill-related functionality.

The skill requires certain parameters to be provided, as they are integral to the skill's behaviour and provide flexibility for adjusting the weapon attributes This design adheres to OCP by allowing for extension without modification.

Additionally, The skill activate requires a deduction of 20% of the actor's stamina, which is encapsulated within the "activateSkill" method (SRP). The duration of the FocusSkill will be kept tracked within the Broadsword class.

Req 2



Brief

The diagram represents a UML class diagram for requirement 2. It has 3 new concrete classes. All this classes are closely related to the Wandering Undead 💀.

entity

WanderingUndead

The WanderingUndead represents an enemy actor within the game world. This class extends the "Actor" abstract class, and is responsible for defining the actions, behaviours (Explained in REQ 3), and attributes associated with the wandering undead.

The addition of capabilities such as ("Status.HOSTILE_TO_ENEMY and ActorType.ENEMY_WANDERING_UNDEAD") enables other game entities and mechanics to identify this actor's hostility and type.

The playTurn method delegates the action selection process to the individual behaviour (SRP), and the usage of behaviours follows OCP to allow for easy addition of new behaviours without modifying the existing class.

grounds

Similarly to REQ1, the following classes will extend to the Ground abstract class (OCP).

Void

The Void class represents a deep mystery hole that can be entered by any actor but results in their own demise. It extends the Ground class and defines the behaviours for actors that enter it.

The “tick” method is responsible for handling the consequences of an actor entering the Void ground (SRP). When an actor enters the Void, a message is displayed indicating that the actor has fallen into the void, the actor is removed from the game map, effectively causing their death, and lastly a “YOU DIED” fancy message is displayed if the actor is a player.

EnemySpawner

The EnemySpawner class provides a utility for spawning enemy actors at a given location (In this context, its at the Graveyard 🪦). It randomly spawns actors based on the specified probabilities.

This design adheres to SRP by focusing solely on the task of spawning enemy actors, ensuring that this responsibility is well-defined and encapsulated.

The spawning of enemies is done by using a switch statement for Actor Types, handling each Actor Type to be spawned individually, allowing for easy adjustments to spawn rates in the future.

One compromise in this design includes the hard-coded probabilities for enemy spawning, which limits the configurability for future uses.

Graveyard

The Graveyard class represents a special type of ground that can spawn enemy actors based on the specified actor type (Req 2: Wandering Undead, Req 4: Hollow Soldier). It extends the Ground class and uses an EnemySpawner to create and spawn enemy actors within the graveyard.

The Graveyard incorporates an “ActorType”, which specifies the type of enemy actor to spawn in the graveyard. The “tick” method is overridden to perform the enemy spawning action during each game tick, which follows the OCP.

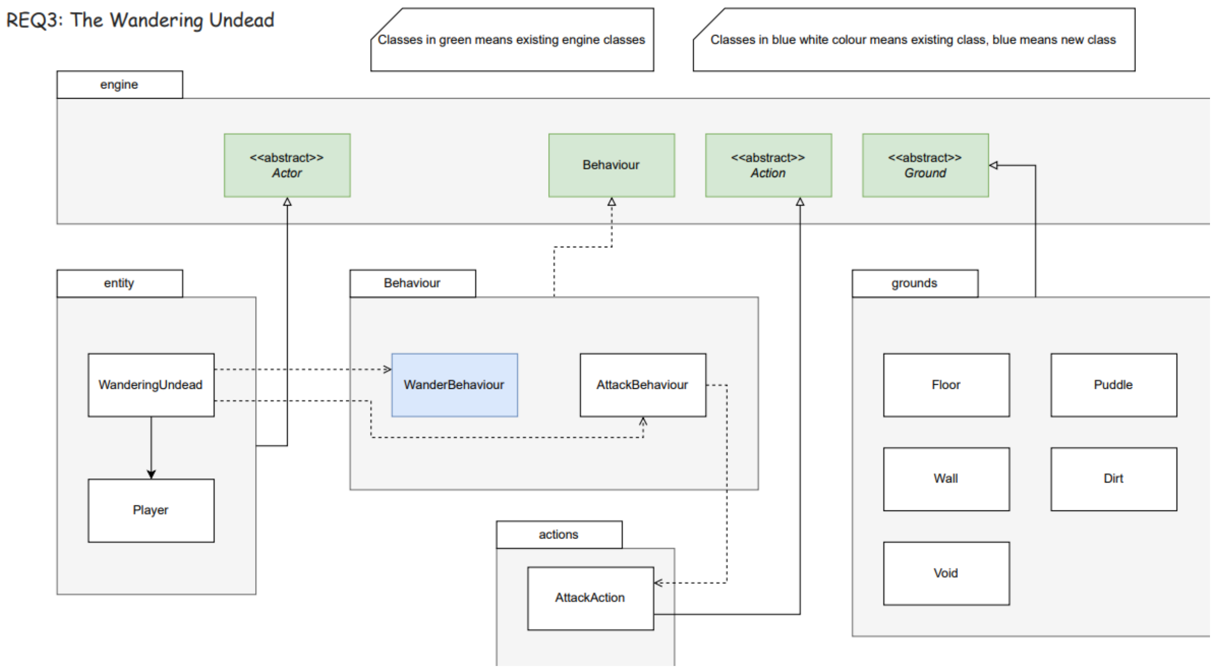
actions

Similarly to REQ1, AttackAction extends the abstract Action class to enforce common methods, allowing for polymorphism through abstractions (DIP).

AttackAction

Performs a fixed amount of damage on the enemy, with the same accuracy (DRY). In REQ2, the Wandering Undead would perform an attack on the player.

Req 3



<u>Brief</u>

The diagram represents a UML diagram for requirement 3. It has 1 new class, with all classes being in the game package, and organised into their appropriate subpackages.

Behaviour

All behaviours implement the Behaviour interface, which ensures the implementation of the main method getAction(). This abstraction allows for polymorphism and code maintainability. Moreover, this ensures that any other class such as WanderingUndead adheres to OCP, as the code in the WanderingUndead wouldn't be modified when new behaviours are introduced. This also allows for future implementations, where maybe the followingBehaviour (Mobs follow player) can be implemented!!

AttackBehaviour

The AttackBehaviour represents a behaviour where an actor (In this context, WanderingUndead) attempts to attack nearby actors of a specified target type (Player). It searches for adjacent locations and attacks the first actor found with the target type.

WanderBehaviour

The WanderingBehaviour class is designed to define the wandering behaviour of actors within the game. It is responsible for selecting a random location to move to.

grounds

Similarly to REQ1, the following classes will extend to the Ground abstract class (OCP).

Floor, Puddle, Wall, Dirt and Void. These classes “canActorEnter” methods have been adjusted for each ground class (LSP and SRP), allowing for diverse and interactive environments where some grounds are passable, while others are not for the Wandering Undead.

entity

WanderingUndead

Similar to Req 2. To add on, the Wandering Undead now has a collection of behaviours that dictate its actions during gameplay. Different behaviours are associated with specific priorities and purpose, which are wandering behaviour and attack behaviour. When a player is within the wandering undead adjacent, the undead will try to attack the player, otherwise it will just wander around the map.

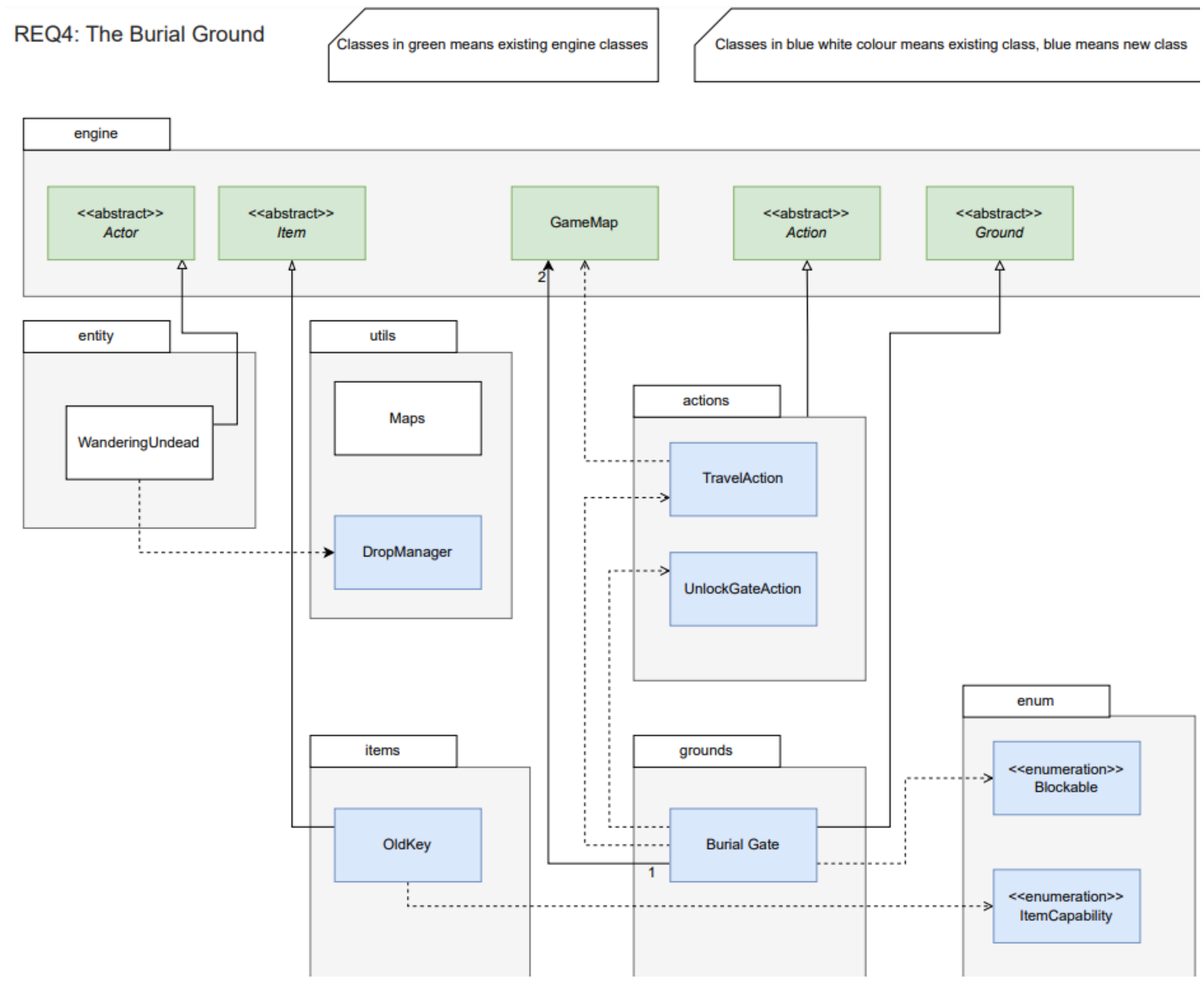
Moreover, the wandering undead intrinsic weapon “limb” with specific damage and hit rate attribute is provided. This allows encapsulation of the weapon properties within the class, allowing for easy modification if needed.

actions

AttackAction

Similar to Req 2. To add on, the AttackBehaviour will now depend on the AttackAction. As mentioned before, if there’s a player within the Wandering Undead vicinity, it will now attack using the intrinsic weapon.

Req 4



Brief

The diagram represents a UML diagram for requirement 4. There are 7 new classes that are related to the new burial ground, and a new item.

grounds

Similarly to REQ1, the following classes will extend to the Ground abstract class (OCP).

Burial Gate

The BurialGate class represents a special ground type in the game, which acts as a gate to connect two different GameMaps. It extends the Ground class and provides the functionality for locking and unlocking the gate, as well as allowing actors to enter it.

The class encapsulates the behaviour of the gate, including its accessibility rules and allowable actions (SRP). Additionally, the customisable destination allows actors to travel through map, allowing for future expansion (OCP). Subsequently, the “canActorEnter” demonstrates which actor can enter the gate based on its locked and unlocked state, allowing derived classes to be substitutable for their base class (LSP). Lastly, the allowable actions such as unlocking with an old key signifies the usage of SRP by encapsulating actor interactions.

Some limitations of this include the Single-location gates, where the gate does not accommodate for scenarios where gates can appear at different locations. Additionally, it is limited to an Item-Based unlocking, which doesn’t support more complex unlock conditions.

enum

Blockable

Represent the state of being”blocked”, by providing a clear and structured way to denote whether certain entities or objects are blocked or unblocked. This enum adheres to the SRP by focusing only on defining the “blocked” state.

ItemCapability

Representing different capabilities associated wit items in the game, this provides a structured and type-safe way to categorise and identify specific item capabilities. The enum follows the SRP by solely focusing on dev=fining various item capabilities, such as an “old key” or a “healing” capability. Moreover, this enum supports future extensibility by allowing the addition of new item capabilities without modifying existing code (OCP).

Items

OldKey

The OldKey class is created to represent a specific item in the game, it encapsulates the properties and behaviours associated with this item. To inherit the common item-related properties and behaviours, the OldKey class extends the abstract “Item” class, allowing it to leverage the existing infrastructure provided by the “Item” class. Additionally, the class adheres to SRP by focusing the sole functionality of the old key item. The class utilised the aforementioned ItemCapability enum, which is essential for distinguishing the old key from other items in the game.

Utils

DropManager

The DropManager class provides a utility for randomly dropping items at a specified location based on specified drop chance. It contains a static method for dropping items at a location with the associated drop probabilities.

This class adheres to the SRP by focusing solely on the task of item dropping, and facilitates the placement of items. The “dropItem” static method, which is the primary interface for dropping items, promotes simplicity and ease of use since it does not require the class to be instantiated. Moreover, this design accepts arrays of items and associated drop chances, allowing for flexibility in specifying the items and their drop probabilities, suitable for handling various in-game scenarios.

Limitation of the class is that it solely focuses on item dropping, and does not handle other aspects of game item management. Additionally, the design assumes that item drop probabilities are specified accurately.

actions

UnlockGateAction

The UnlockGateAction represents an action where an actor can attempt to unlock a gate. It allows one actor to unlock a gate located at a specified location, using an item with a certain capability (In this context, the Old Key).

The class adheres to SRP, as its primary responsibility is to execute the unlocking action and provide a descriptive menu message. The class implements the “execute” method, which checks if the gate is locked (“BLOCKED” capability), whether the actor has the required item capability, and only unlocks the gate if conditions are met. This design also allows for better code reuse and versatility in handling different gates and items.

One limitation is that the gate presumes all gates rely on the possession of a specific item capability, and does not consider a more complex unlocking mechanism.

TravelAction

The TravelAction class represents an action where an actor can travel from one GameMap to another, allowing an actor to move to a different GameMap at a specified coordinates. Similarly to the UnlockGateAction, it adheres to SRP by only handling the travelling between GameMaps. The execution of this action is done by using the “map.moveActor” method, to transfer the actor from one GameMap to another.

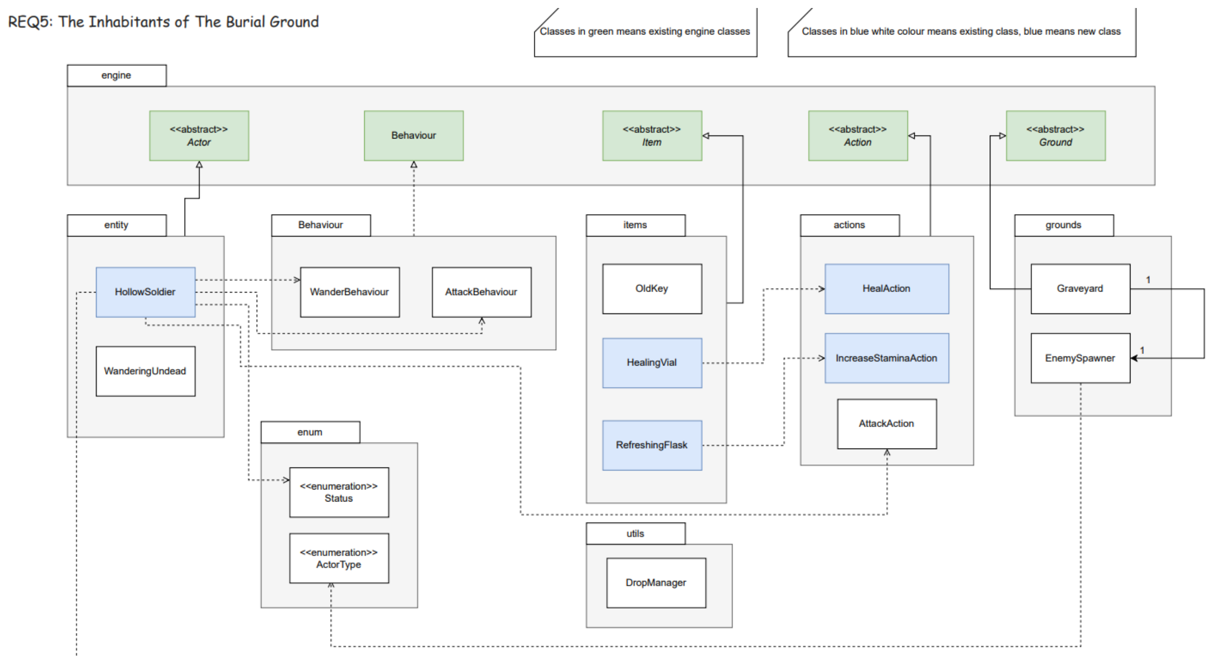
Parameters such as “from”, “to”, “x-coordinate”, “y-coordinates” are necessary, allowing for flexibility in specifying the source and destination GameMap instance and the target coordinates, promoting versatility in handling various travel scenarios.

entity

WanderingUndead

Similarly to Req 3, now the WanderingUndead is able to drop items upon death. The “Unconscious” method is overridden to handle the behaviour when the Wandering Undead becomes unconscious. Upon unconsciousness, the WanderingUndead drops an “OldKey” at its location with a specified drop chance via the DropManager (OCP), rewarding the player for defeating enemies.

Req 5



Brief

The UML for this requirement is similar to the requirement before, as it involves the creation of a new enemy (Hollow Soldier), which is similar to the Wandering Undead, and the creation of some consumable item. There are 5 new concrete classes, with 1 being the HollowSoldier, 2 being the new item, and 2 new actions for the item.

actions

Extends abstract action class (LSP), relies on abstraction (Actor and gameMap) (DIP)

HealAction

The HealAction class represents an action where an actor can be healed by an item. The design of this action is clear, as it only focuses on healing back the actor (SRP). This action effectively extends the abstract Action class (LSP), and can be used interchangeably with other actions in the game. Moreover, the healing amount is dynamic, it is calculated based on a percentage of the actor's maximum health.

IncreaseStaminaAction

The IncreaseStaminaAction represents an action where an actor can increase their maximum stamina. It allows one actor to increase their stamina by a certain amount. The stamina is increased by using the preexisting game engine, by modifying a specific attribute maximum amount.

Items

Items abstract Item class (OCP) and interacts with abstractions (Action and Actor) (DIP)

HealingVial

The Healing Vial extends the abstract “Item” class (DIP), making it easy to integrate healing vials into the game. It leverages the existing capabilities and action framework. The design of this item is meant to be a single-use item (Implementation of a “used” flag), which also provides a strategic depth to the gameplay, as players must now decide when to use their healing vials. The HealingVial uses the HealingAction to increase the player's health

RefreshingFlask

The Refreshing Flask has a clear responsibility of representing a flask item that increases the player's stamina. Similarly to the Healing Vial, this item is also a single-use item. The RefreshingFlask uses the IncreaseStaminaAction to permanently increase the players maximum stamina.

entity

WanderingUndead

The wandering undead will now have a chance to drop another item upon death, which is the Healing Vial

HollowSoldier

The whole structure is similar to the WanderingUndead, except that the Hollow Soldier has different stats for its intrinsic weapon, and has a chance of dropping a Healing Vial, or a Refreshing Flask upon death.