FIT2099 Assignment 3

Applied Session 4 Group 8

Updated Design Rationale

Group Members:
Wee Jun Lin (32620861)
Kok Tim Ming (32619138)
Lau Ka-Kiat (32967136)

Overview

Design goals

Our main objective involves designing an extensible codebase around the existing Elden Ring game engine. To achieve this, design goals have been set to be faithfully adhered throughout the whole design and development process.

We strive to:

- Reduce code redundancy (DRY): Achieve through abstractions to reuse code. This will make the codebase more concise and easier to maintain.

- Avoid multiple inheritance (diamond problem): Done by abstracting methods through interfaces (ISP) or utilising composition.

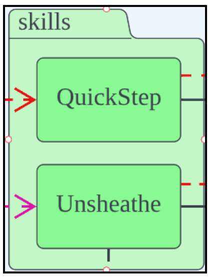
- Create an extensible codebase, accounting for future feature additions and development. Classes and modules should only be made to be extended (OCP).

- Follow the Interface Segregation Principle (ISP). This will prevent unused code and will make adding new features easier as no refactoring is needed.

- Follow Single Responsibility Principle (SRP): each class should only have one responsibility so that it is more focused. This will make the codebase easier to be extended with new functionality and easier to maintain.

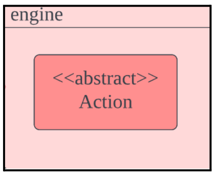
Notes

Classes that were not modified from previous REQs will not be discussed in the rationale.
All Full UML diagrams and Dialogue Trees are attached at the end of this document.
Newly added classes will be coloured in **green** 🟩🟩🟩. Fill colour contrast will increase along with package hierarchy depth.



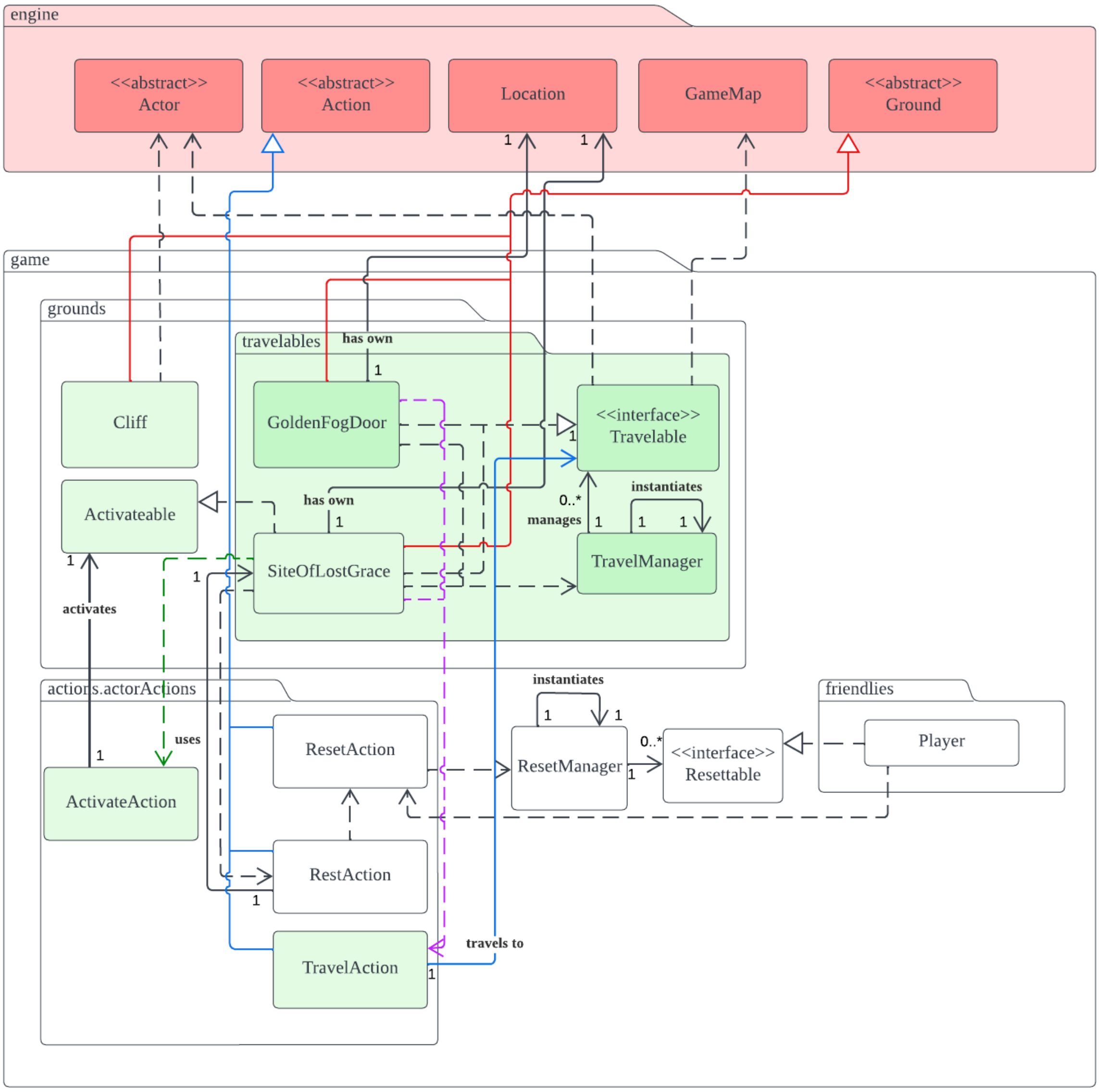
Classes with **no fill** are classes that will not be extended/modified from the previous requirements and assignments. Classes coloured in **red** 🟥 are engine classes.

FingerReaderEnia



Requirement 1

REQ1: Travelling between Maps



Brief

The diagram represents an object-oriented system for requirement 1. It now has 7 new concrete classes and 1 new interface categorised into their respective packages. Related classes that were previously shown in previous requirements/parts are connected to the extended classes to provide some functionality context.

Grounds

Cliff

Cliff is a new class that inherits from the abstract *Ground* class (DRY, LSP). *Cliff* will handle the damage inflicted to *Actors* that step on it, not requiring any modification to existing classes (OCP, SRP). The capability *enum* is utilised to filter enterable actors (OCP).

GoldenFogDoor

GoldenFogDoor is a new class that inherits from the abstract Ground class (DRY, LSP). *GoldenFogDoor* has an association to location and inherits the *Travelable* interface (ISP). This allows the *GoldenFogDoor* to handle the travelling functionality by itself (SRP). *GoldenFogDoor* will also be responsible for providing the *Actors* the *TravelAction*, connected to the travelling network (pairs) managed by *TravelManager*.

SiteOfLostGrace

SiteOfLostGrace is a carried over class from previous requirements that received extensions. The revamped version of *SiteOfLostGrace* now implements *Travelable* and *Activateable* (ISP, OCP). This allows *SiteOfLostGrace* to handle the travelling functionality and activation functionality by itself (SRP). *SiteOfLostGrace* can be activated dynamically for fast travel (*TravelAction*) and the resting (*RestAction*) functionality through *ActivateAction*. When a site is activated, it will be registered with other activated sites in the travelling network (through *TravelManager*).

Player respawn was not implemented as it was an optional requirement, and due to time constraints.

Activateable

An interface class implemented by classes that can be activated, forming a contract for common methods like *activated()*. The use of polymorphism also ensures that the Open-Closed Principle is not violated as no other classes are needed to be modified.

Instead of creating the activation functionality just to cater *SiteOfLostGrace*, the *Activateable* interface can be used for other game objects in the future to allow for creative activation effects (OCP). This includes features like activating a trap door, activating doors that link to another, which is why *Activateable* was implemented as an interface to open up the doors for future extensions (OCP, LSP, ISP). One downside of implementing *Activateable* as an interface is that the implementing-classes need to define the activation functionalities by themselves (location attribute, etc). However, as explained earlier, this provides flexibility for defining different activation effects (OCP).

Travelling

Travelable

An interface class implemented by classes that can be travelled to and fro, forming a contract for common methods like *travel()*. The use of polymorphism also ensures that the Open-Closed Principle is not violated as no other classes are needed to be modified.

In the future, there might be any game objects that can be used as travel points. This includes features like intentionally placing a *Travelable* item as a fast travelling waypoint, hurting the *Actor* when travelling, granting status effects etc. As such, Travelable as an interface opens up the doors for future extensions (OCP, LSP, ISP). One downside of implementing *Travelable* as an interface is that the implementing-classes need to define the travel functionalities by themselves (*location* attribute, etc). However, as explained previously, this provides flexibility for defining different travelling effects (OCP).

TravelManager

The *TravelManager* class follows a Registry & Singleton pattern. This approach provides a centralised control and access point among *Travelling* points, reduced coupling between other classes, allows for code reusability and simplifies object retrieval.

TravelManager will store a network of associations. A network contains a list of associated *Travelables*. These associations represent the two-way connections that each *Travelable* has to each other, where a *Travelable* can choose to associate itself with a network using a key that represents the network (like a network ID) in a HashMap.

Though the approach of retrieving a *Travelable's* associated network using a network ID creates Connascence of Meaning, it is thought of to be one of the most optimal system tradeoffs that this design must carry in order implement *Travelables* that can associate with many networks (not just one).

While the Registry & Singleton pattern makes things relatively easy to implement, the approach could come across as controversial as it causes tight coupling along with encapsulation problems between *TravelManager* and other dependent objects. The approach could promote the violation of SRP as the application extends (god class) and if good coding practices are not exercised in the future. As the *TravelManager* class grows, the violation of SRP could likely be avoided by utilising composition, hence adhering to OCP and SRP by encouraging segregation of functionality.

Actions

TravelAction and *ActivateAction* extends the abstract *Action* class to enforce common methods, allowing for polymorphism through abstractions (DIP).

TravelAction

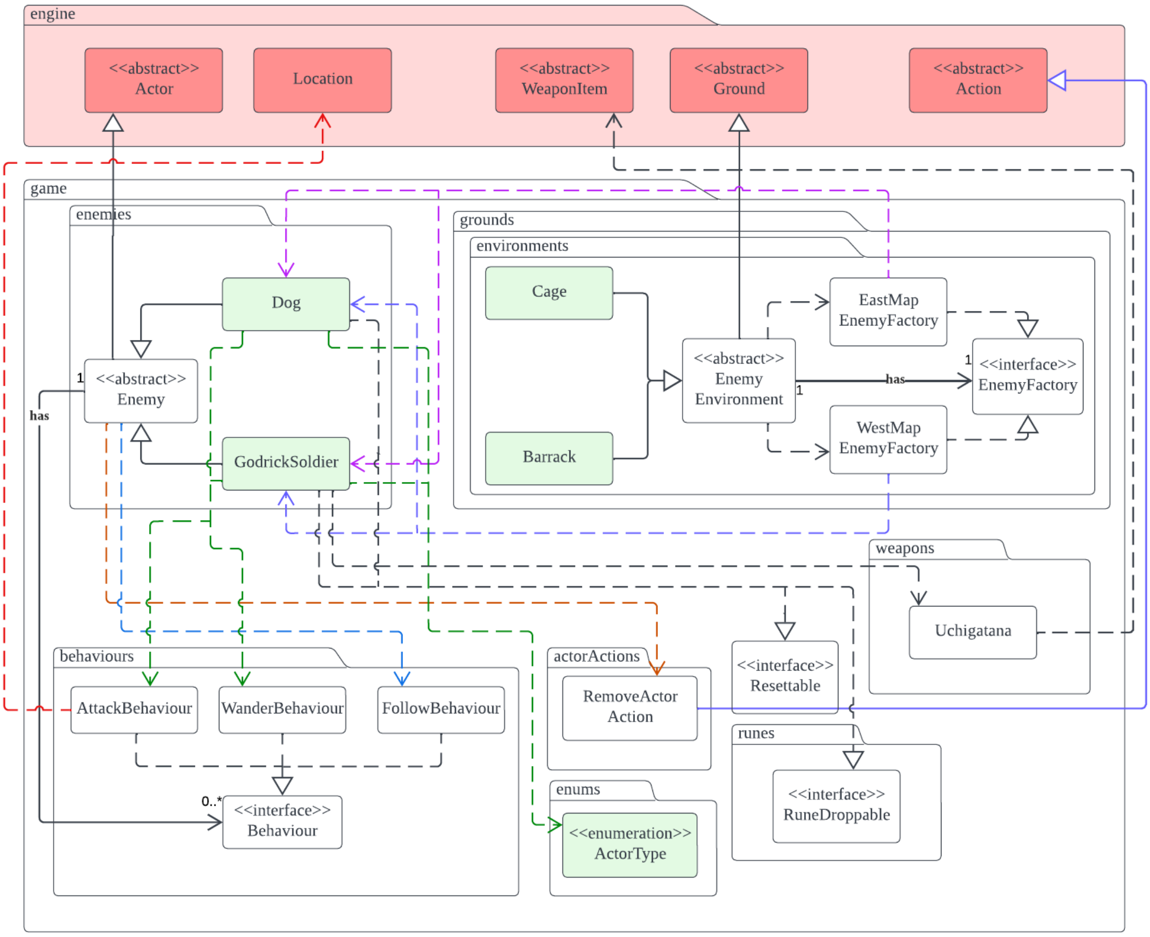
TravelAction will have an association with *Travelable* Interface. This allows *TravelAction* to invoke the *Travelable* interface's methods when executed, adhering to SRP and ISP in implementing-classes.

ActivateAction

ActivateAction will have an association with *Activateable* Interface. This allows *ActivateAction* to invoke the *Activeable* interface's methods when executed, adhering to SRP and ISP in implementing-classes.

Requirement 2

REQ2: Inhabitant of the Stormveil Castle



Brief

The diagram represents an object-oriented system for requirement 2. It now has 5 new concrete classes categorised into their respective packages. Related classes that were previously shown in previous requirements/parts are connected to the extended classes to provide some functionality context.

Grounds

Cage

Cage is a new class that inherits from the abstract *EnemyEnvironment* class (DRY, LSP). No modifications to existing classes were required for this class to spawn the *Dog* enemy (OCP, SRP).

Barrack

Barrack is a new class that inherits from the abstract *EnemyEnvironment* class (DRY, LSP). No modifications to existing classes were required for this class to spawn the *GodrickSoldier* enemy (OCP, SRP).

WestEnemyFactory & EastEnemyFactory

Enemy factories were changed from the previous implementation. The previous implementation uses *ActorType* enumeration to determine the enemy to spawn, but since Stormveil Castle enemies do not make use of the cardinal directions, it is a bad practice to create enumerations for specific enemy identities to make the spawning system work correctly.

The feedback provided in A2 suggests that new template methods like *createDog()* in the enemy factory be set as a contract in the *EnemyFactory* interface class. However, our implementation preferred utilising static char representations defined by the *EnemyEnvironment* subclass itself as a key to spawn the associated enemy as it improves extensibility. Also, if the recommendations from feedback were followed, it is difficult to implement in our current design of *EnemyEnvironment*, as all the environment classes would need to be modified (possible violation of OCP) to spawn their specific enemies (*getSkeleton()* in *Graveyard*, *getCrustacean()* in *PuddleOfWater*, etc) instead of only *getEnemy()* in *EnemyEnvironment*.

Utilising the static char representations also allows for a centralised control of spawning, and when modifications are needed, only the central node is required to be modified (in our case, *EnemyFactory* subclasses), improving maintainability.

Enemies

In this requirement, Stormveil Castle enemies are shown to have a dependency on *ActorType* to emphasize that they are the same enemy type. Thus, they can dynamically filter the *Actors* to attack in *AttackBehavior* with *ActorType.STORMVEIL_CASTLE* (OCP). Same applies for other *Enemy* classes.

Dog

Dog is a new enemy that extends from the *Enemy* class (DRY, LSP). Adding this class requires no new modifications to the existing classes (OCP).

Godrick Soldier

GodrickSoldier is a new enemy that extends from the *Enemy* class (DRY, LSP). Adding this class requires no new modifications to the existing classes (OCP). In this requirement, *HeavyCrossbow* was not implemented as it is an optional requirement, and due to time constraints. *GodrickSoldier* was provided with *Uchigatana* instead.

Behaviors

AttackBehaviour

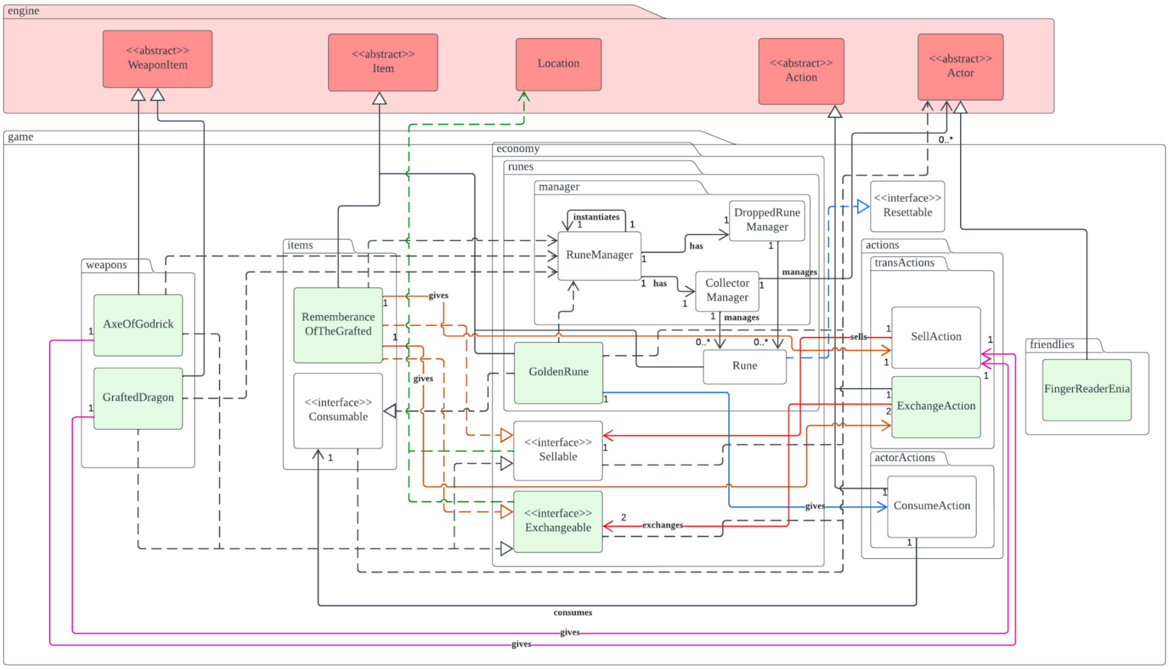
Attempts to get a random valid target (filtering restricted enums, like *ActorType.STORMVEIL_CASTLE*) in surroundings and performs 50/50 attack/special attack. This design allows actors to dynamically define their targets without needing any modifications (OCP). The current implementation of *AttackBehavior* is very simple, but in the future better algorithms (behavior concrete classes) could be easily implemented based of the *AttackBehavior* abstract class (LSP, OCP, DRY).

Weapons

Weapons were not implemented as it is an optional requirement, and due to time constraints.

Requirement 3

```
REQ3: Godrick the Grafted
```



Brief

The diagram represents an object-oriented system for requirement 3. It now has 6 new concrete classes and 1 new interface categorised into their respective packages. Related classes that were previously shown in previous requirements/parts are connected to the extended classes to provide some functionality context.

Godrick the Grafted

GodrickTheGrafted

Godrick the Grafted was not implemented as it is an optional requirement, and due to time constraints.

AxeOfGodrick, GraftedDragon

These weapons extend the abstract *WeaponItem* class (DRY, DIP) and were implemented without skills as it is an optional requirement, and due to time constraints. These two classes implement the *Exchangeable* interface, but are not dependent on *ExchangeAction* as they do not provide the *Action*. These abstractions also allow for polymorphism (DIP).

RemembranceOfTheGrafted

This class extends the abstract Item class (DRY) and implements the *Exchangeable* interface. It is dependent on *SellAction* and *ExchangeAction* as it provides the *Actions* for holders. These abstractions also allow for polymorphism (DIP).

Transactions

Exchangeable

An interface class implemented by classes that can be exchanged for another item, forming a contract for common methods like *exchangeIn()* or *exchangeOut()*. The use of polymorphism also ensures that the Open-Closed Principle is not violated as no other classes are needed to be modified.

In the future, there might be any game objects that can be exchanged for any other game object. This includes exchanging an item for a weapon, or vice versa, allowing *Exchangeable* concrete classes to define unique effects that will happen when exchanging game objects (OCP, LSP, ISP). One downside of implementing *Exchangeable* as an interface is that the implementing-classes need to define the exchange functionalities (in and out) by themselves. On the other hand, this approach provides flexibility for defining different exchanging effects (OCP).

It is important to note that in the current implementation, actors (or more specifically traders) will provide the holder the ability to exchange the *Exchangeable* object only if the trader has a specific Capability *enum*, and that they are nearby the holder. Every trader will also be limited to exchanging the same *Exchangeable* object defined in the *Exchangeable* object itself. This is a limitation of the implementation, given the game engine provided.

ExchangeAction

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

ExchangeAction will have an association with *Exchangeable* Interface. It is a two way association, exchangeableIn and exchangeableOut. This allows *ExchangeAction* to invoke the respective *Exchangeable* interface's methods *exchangeIn()* or *exchangeOut()* when executed, adhering to SRP and ISP in implementing-classes.

Sellable

An interface class implemented by classes that can be sold, forming a contract for common methods like *sell()*. The use of polymorphism also ensures that the Open-Closed Principle is not violated as no other classes are needed to be modified. Having the *Sellable* interface on its own also encourages the Liskov Substitution Principle (LSP), as this ensures inheritance hierarchy is correctly designed and implemented.

Following the feedback, *Sellable* interface subclasses (*SellableWeapon*, *SellableItem*) were revamped into *Sellable* on its own, meaning that the implementing-classes need to define the sell functionalities by themselves rather than relying on default methods provided by the niche-interfaces before. This can lead to the violation of the DRY principle, but it is a justifiable trade-off for better maintainability and flexibility by avoiding over-abstraction. In the future, *Sellable* objects can also define different selling effects (OCP).

FingerReaderEnia

FingerReaderEnia extends *Actor* (DRY), and has Capabilities to allow other actors to sell and to exchange the *RemembranceOfGrafted*.

GoldenRune

GoldenRune extends *Item* (DRY) and implements *Consumable* (ISP). This allows the re-usage of *ConsumeAction* (OCP). *GoldenRune* does not inherit from *Rune* because *Rune* has very different functionalities in the implementation (LSP). The abstractions also allow for polymorphism (DIP).

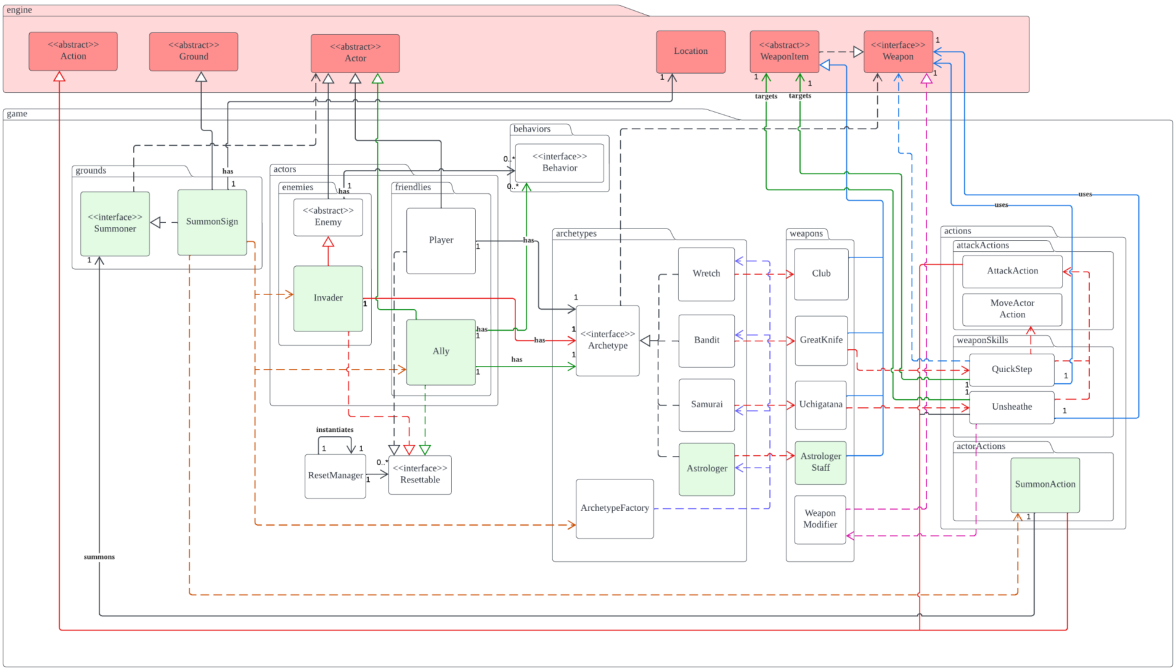
GoldenRune has an association with *ConsumeAction* because it is required to not give the actor the action when the *GoldenRune* is not in the *Actor's* inventory. This is done through the tick method provided by Item (ISP, DRY). *GoldenRune* defines its own *consume()* method (extensible, OCP): it updates the collector's runes in the *RuneManager* when *consume()* is called.

HP upgrade functionality

SiteOfLostGrace HP upgrade functionality was not implemented as it is an optional requirement, and due to time constraints.

Requirement 4

REQ4: A Guest from Another Realm



Brief

The diagram represents an object-oriented system for requirement 4. It now has 6 new concrete classes and 1 new interface categorised into their respective packages. Related classes that were previously shown in previous requirements/parts are connected to the extended classes to provide some functionality context.

New Archetype

Astrologer

A new *Astrologer* archetype is added that implements the *Archetype* interface (DRY, LSP). This inheritance allows for polymorphism in implementing-classes (DIP). No further modifications are required to the existing classes and codebase (OCP).

AstrologerStaff

The *AstrologerStaff* extends *WeaponItem* to inherit common methods (DRY). It was implemented without the ranged attack component as it is an optional requirement, and due to time constraints.

Allies/Invaders

SummonSign

This class extends from the *Ground* class (DRY), and implements the *Summoner* interface to handle the summoning feature (ISP). These abstractions open up opportunities for polymorphism, adhere to DIP. *SummonSign* will provide the *SummonAction*. On summon, it spawns 50/50 Ally/Invader with *RandomNumberGenerator* and *ArchetypeFactory's randomArchetype* method.

Ally

Ally is a new class that inherits from the *Actor* class (DRY) and implements *Resettable* (ISP). It has behaviors (hence the association). This inheritance allows for polymorphism (DIP). *Ally* will also store the *Archetype* as an attribute, as it is expected to be able to grow stronger based on the *Archetype*, as it gains experience in the future.

Invader

Invader is a new class that inherits from the *Enemy* class (DRY), implements *Resettable* and *RuneDroppable* (ISP). Invader will also store the *Archetype* as an attribute, as it is expected to be able to grow stronger based on the *Archetype*, as it gains experience in the future. *Invader* was made to inherit the *Enemy* abstract class since it has similar behaviors to other enemies (i.e., follow when a *PROVOKER* is in exits). While it is possible that Invader could be made to inherit *Actor* instead, *Enemy* reduces code duplication (DRY) and enforces code consistency through refactoring, at the cost of deeper abstraction.

Summoner

An interface class implemented by classes that can summon game objects, forming a contract for common methods like *summon()*. The use of polymorphism (DIP) also ensures that the Open-Closed Principle is not violated as no other classes are needed to be modified.

In the future, there might be any game objects that can be summoned by the *Summoner*. It could summon a dropped *Item*, *Weapon*, *Ground*, *Actor* or any future game objects. With the interface, *Summoner* concrete classes can define all sorts of creative effects flexibly (OCP, LSP, ISP), and have the creative effects executed when *SummonAction* is called. Though, one small trade-off for implementing *Summoner* as an interface is that the implementing-classes need to define the summon functionalities (in and out) by themselves.

SummonAction

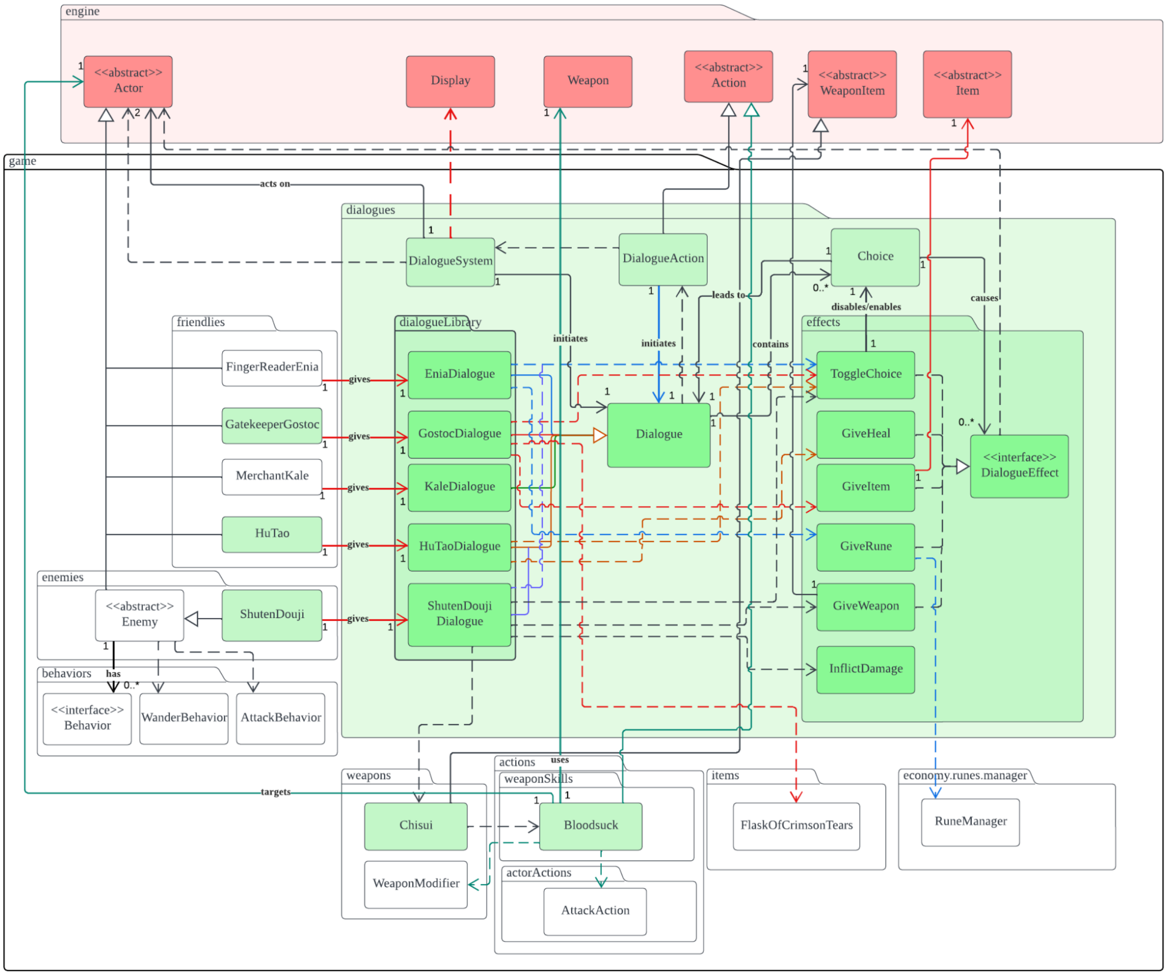
SummonAction extends the abstract *Action* class to enforce common methods, allowing for polymorphism through abstractions (DIP).

SummonAction will have an association with *Summoner* Interface. This allows *SummonAction* to invoke the *Summoner*'s method *summon()* when executed (DIP), allowing

Summoner to define creative effects, adhering to SRP, ISP and OCP in implementing classes.

Requirement 5

REQ5: Creative Requirement (Dialogue with NPCs)



Creative Requirement: Dialogues with other Actors (NPCs)

Brief

The diagram represents an object-oriented system for requirement 5. It now has 20 new concrete classes and 1 new interface categorised into their respective packages. Related classes that were previously shown in previous requirements/parts are connected to the extended classes to provide some functionality context.

Implementation

Summary

Dialogues with other actors (NPCs) were chosen to be implemented as the creative requirement. This was chosen because *Dialogues* are an intrinsic part of games that provide deeper immersion, especially when Players are given the option to affect the course of events and end with different *Dialogue* outcomes.

The architecture follows a branching dialogue system, or sometimes it can be cyclic, thus a cyclic graph would be a good representation of the code structure. While not strictly defined, *Dialogues* can be visualised as nodes, *Choices* for edges, and the *DialogueSystem* being the encompassing graph. However, it is important to note that the *Dialogues* are not stored in a ‘database’, but rather all are initialised in the starting *Dialogue* node (from the dialogueLibrary), and are linked to each other by *Choices*.

This feature will allow the player to interact with different NPCs in the game and choose from various dialogue options (*Choices*). The *Choices* will lead to different outcomes (*Dialogue*) and cause different effects (*DialogueEffect*) for the player. In the implementation, unique events like giving weapons, items, runes, healing/damaging the player and toggling availability of *Choices* was demonstrated. The *Dialogues’* choices will also provide some game lore, potentially pulling in player interest.

DialogueSystem

The *DialogueSystem* concrete class was created to handle the complete execution of a *Dialogue*. The class handles the operations and flow of a *Dialogue* tree (encouraging SRP), like displaying the available choices, handling user input and choice outcomes.

This class is more of a utility class, one that resembles the *Menu* engine class. While the *Menu* class could be reused, we desired a cleaner and more dedicated class like the *DialogueSystem*.

Dialogue

The *Dialogue* class enables choice-based dialogues in the game. It consists of a message attribute, which holds the dialogue message, and a list of choices that the player can make. *Choices* in the *Dialogue* class follow the composition pattern, allowing for future complex and flexible *Choice* manipulations (OCP, ISP, DIP). The *Dialogue* class also handles the presentation of different assigned choices to the player on its own (SRP).

While *Dialogues* could be made into abstract classes, it was not necessary as it could be used as-is. This avoids over-abstraction and potential creation of a god class (violates SRP).

Choice

The *Choice* class represents an option within a *Dialogue*. If available, it provides the player with a *Choice* to select within a *Dialogue*, which when selected can lead to different events, to other *Dialogues*. Each *Choice* has a description and can trigger one or more effects (*DialogueEffect*) when selected. This means the *Choice* class must maintain a reference to the next *Dialogue*.

Choices follow the composition pattern, allowing for future complex and flexible *DialogueEffect* manipulations and additions (OCP, ISP, DIP). When a Choice is selected, Choice calls every attached *DialogueEffect's* method *trigger()* to trigger the unique effect, promoting DIP through polymorphism.

Choices can also set conditions for it to be available. In the implementation, *Enums* were used to determine the availability of the *Choice*. For example, if the player does not have *Status.BLESSED*, the choice cannot be selected and will not be displayed in the parent *Dialogue*. This extension adheres to OCP as there were only extensions made for the addition of conditional availability.

DialogueEffect

DialogueEffect is an interface that is implemented by all the effects (ISP) that can happen from the *Choices* in *Dialogues* like *GiveHeal*, *GiveItem*, *InflictDamage*, *ToggleChoice* and more (SRP). This approach allows a diverse amount of effects to be easily created, plugged into *Choices*, and have its *trigger()* method called dynamically (DIP, SRP, LSP). New effects can also be created and added to the existing codebase without needing modifications to existing classes (OCP).

While *DialogueEffects* could be implemented into *Dialogues* instead, implementing them in *Choices* was preferred because it allows more flexibility over causing effects. This is because different choices can lead to the same *Dialogue*, and it may not be desirable to trigger an effect.

DialogueAction

DialogueAction extends the abstract *Action* class to enforce common methods, allowing for polymorphism through abstractions (DIP).
DialogueAction will have an association with *Dialogue* classes. This allows *DialogueAction* to pass the *Dialogue* into *DialogueSystem* (DIP) for execution, adhering to SRP.

dialogueLibrary (package)

This package contains all of the NPCs' dialogues. All of the *Dialogue* classes in this package should represent the starting node of the branching dialogue system, thus it initialises (builds) all of its linked *Dialogues*, *Choices* and *DialogueEffects*.

Every NPC has its unique *Dialogue* (SRP), and each *Dialogue* class in this package extends from the *Dialogue* class (DRY, LSP), adhering to DIP by enabling polymorphism.

KaleDialogue

A dialogue subclass for *MerchantKale*. This dialogue was created to demonstrate a basic cyclic dialogue graph.

EniaDialogue

A dialogue subclass for *FingerReaderEnia*. This dialogue was created to demonstrate a cyclic dialogue graph, with choice toggling and rune giving.

GostocDialogue

A dialogue subclass for *GateKeeperGostoc*. This dialogue was created to demonstrate a cyclic dialogue graph, with choice toggling and item giving.

HuTaoDialogue

A dialogue subclass for *HuTao*. This dialogue was created to demonstrate a dialogue tree, with choice toggling and healing.

ShutenDoujiDialogue

A dialogue subclass for *ShutenDouji*. This dialogue was created to demonstrate a cyclic dialogue graph with choice toggling, weapon giving, damage infliction and conditional choices.

HuTao

A friendly descendent that extends *Actor* (DRY). Created to demonstrate healing effects via its *Dialogue*.

GateKeeperGostoc

A friendly but suspicious man that extends *Actor* (DRY). Created to demonstrate item giving effect via its *Dialogue*.

ShutenDouji

A demonic enemy that extends *Enemy* (DRY). Does not attack friendlies, created to demonstrate damage infliction, weapon giving, conditional choices via its *Dialogue*.

Chisui

Chisui is a unique weapon given by *ShutenDouji* through his *Dialogue*. It provides the capability Status.BLESSED, and has the special ability Bloodsuck.

Chisui extends the abstract *WeaponItem* class. It shares common attributes with *WeaponItem*, so it is logical to abstract this class (DIP) to reduce code redundancy (DRY).

Bloodsuck

Bloodsuck is a weapon skill that damages the target with 50-100% of the original weapon's damage, and heals the user for the amount of damage dealt.

Bloodsuck is a concrete class that extends from *Action* to utilise polymorphism through abstractions. It depends on *AttackAction* to reduce code redundancy (DRY) and promote maintainability when changes are needed. While it can be made to extend AttackAction, dependency was considered over inheritance to avoid over abstraction.

Although avoiding over-abstraction can promote maintainability, it is important to strike a balance between abstraction and maintainability; In probable future circumstances, *Bloodsuck* could be made to inherit *AttackAction* to ensure optimal flexibility and extensibility.

WeaponModifier is used as a damage substitute to adhere to OCP.

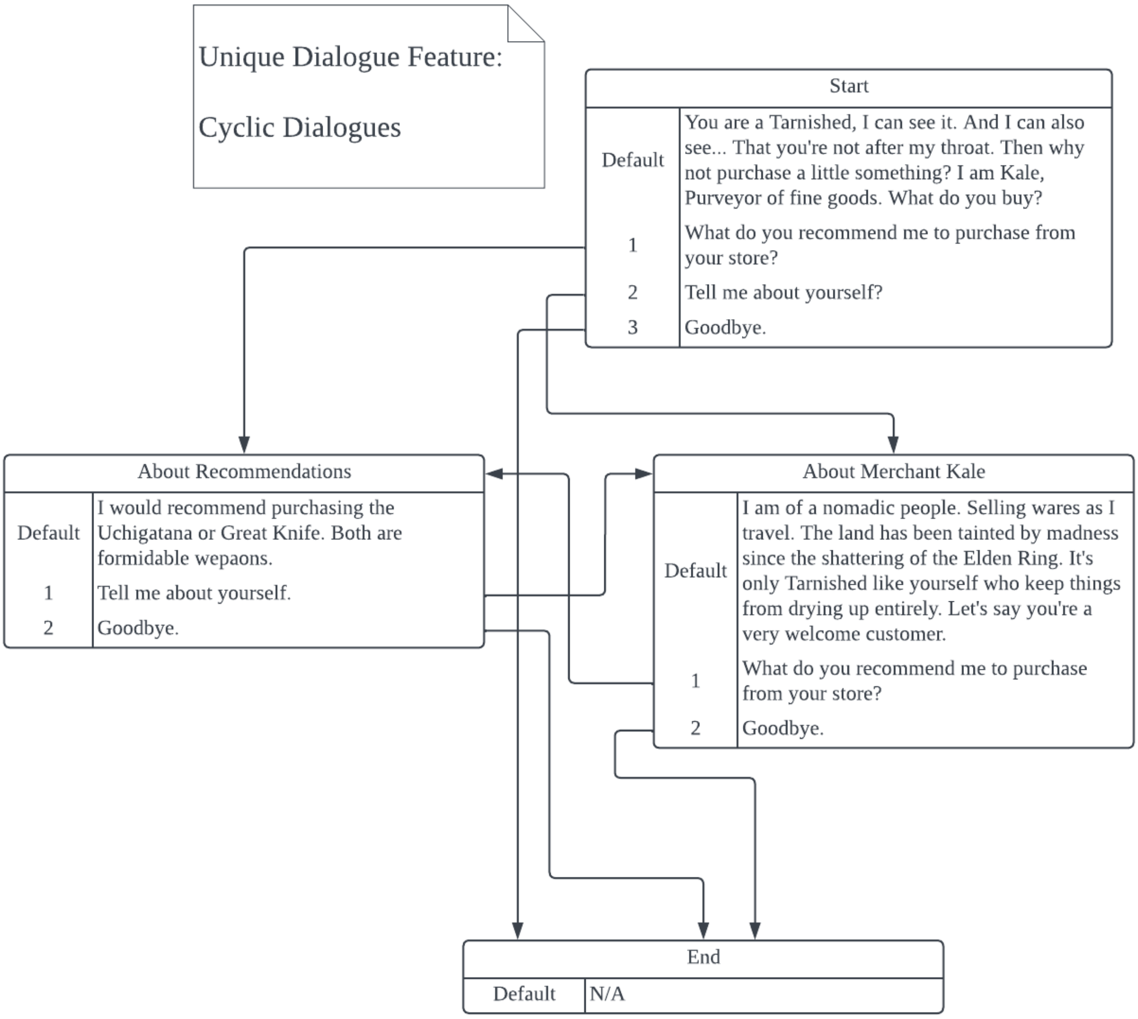
Note

Dialogue branching graphs were illustrated throughout the design process to provide a visual representation of the Dialogue flow to the developers (us) and readers.

Contribution Log Link

FIT2099 MY (THU 10 AM) Assignments' Contribution Logs

Kale



Unique Dialogue Feature:

Cyclic Dialogues

Start	
Default	You are a Tarnished, I can see it. And I can also see... That you're not after my throat. Then why not purchase a little something? I am Kale, Purveyor of fine goods. What do you buy?
1	What do you recommend me to purchase from your store?
2	Tell me about yourself?
3	Goodbye.

About Recommendations	
Default	I would recommend purchasing the Uchigatana or Great Knife. Both are formidable wepaons.
1	Tell me about yourself.
2	Goodbye.

About Merchant Kale	
Default	I am of a nomadic people. Selling wares as I travel. The land has been tainted by madness since the shattering of the Elden Ring. It's only Tarnished like yourself who keep things from drying up entirely. Let's say you're a very welcome customer.
1	What do you recommend me to purchase from your store?
2	Goodbye.

End	
Default	N/A

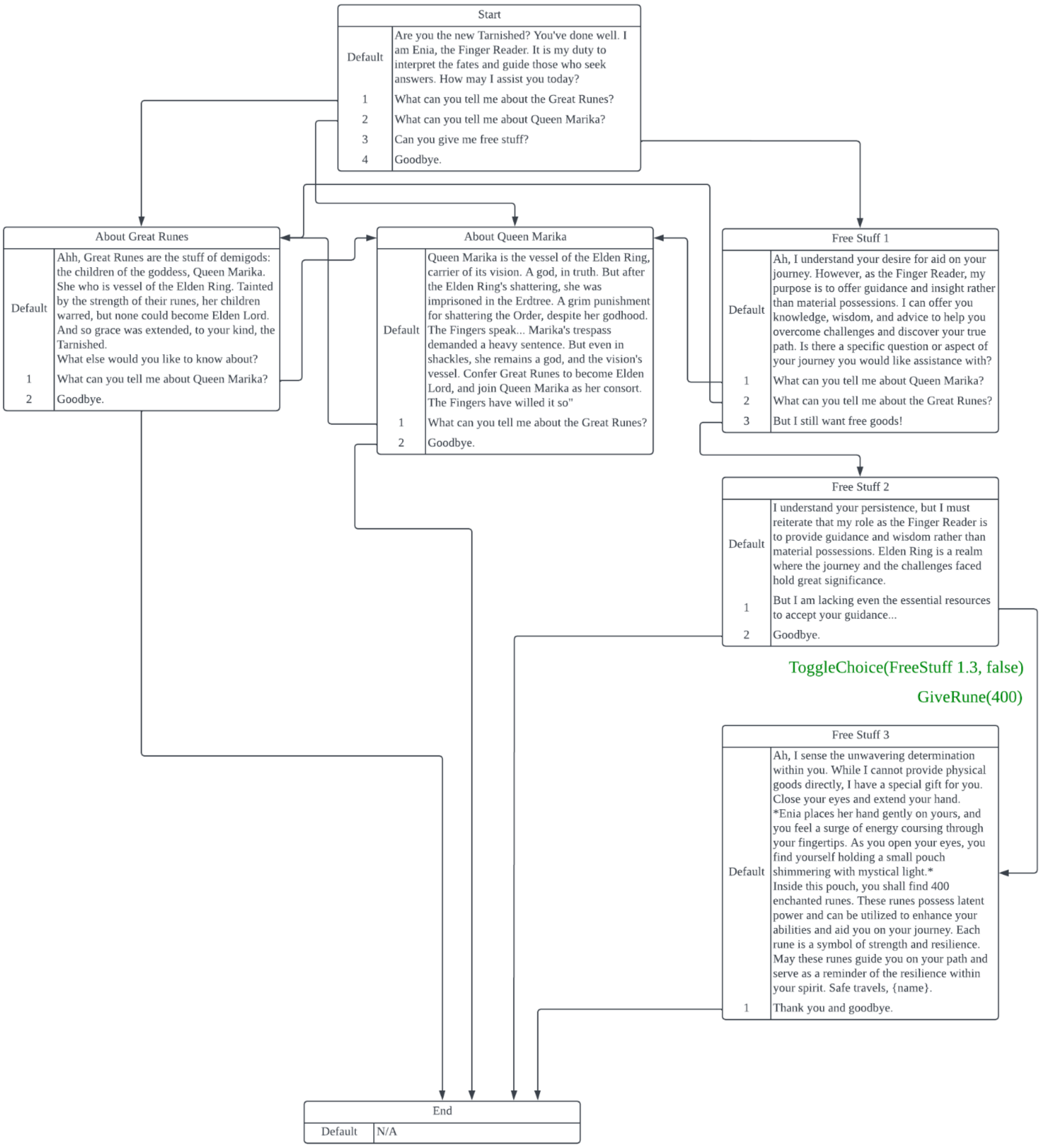
Unique Dialogue Feature:

Cyclic Dialogues

ToggleChoice()
Toggles the availability of a Choice.

GiveRune()
Can give player any amount of runes. (400)

Enia



Start	
Default	Are you the new Tarnished? You've done well. I am Enia, the Finger Reader. It is my duty to interpret the fates and guide those who seek answers. How may I assist you today?
1	What can you tell me about the Great Runes?
2	What can you tell me about Queen Marika?
3	Can you give me free stuff?
4	Goodbye.

About Great Runes	
Default	Ahh, Great Runes are the stuff of demigods: the children of the goddess, Queen Marika. She who is vessel of the Elden Ring. Tainted by the strength of their runes, her children warred, but none could become Elden Lord. And so grace was extended, to your kind, the Tarnished. What else would you like to know about?
1	What can you tell me about Queen Marika?
2	Goodbye.

About Queen Marika	
Default	Queen Marika is the vessel of the Elden Ring, carrier of its vision. A god, in truth. But after the Elden Ring's shattering, she was imprisoned in the Erdtree. A grim punishment for shattering the Order, despite her godhood. The Fingers speak... Marika's trespass demanded a heavy sentence. But even in shackles, she remains a god, and the vision's vessel. Confer Great Runes to become Elden Lord, and join Queen Marika as her consort. The Fingers have willed it so"
1	What can you tell me about the Great Runes?
2	Goodbye.

Free Stuff 1	
Default	Ah, I understand your desire for aid on your journey. However, as the Finger Reader, my purpose is to offer guidance and insight rather than material possessions. I can offer you knowledge, wisdom, and advice to help you overcome challenges and discover your true path. Is there a specific question or aspect of your journey you would like assistance with?
1	What can you tell me about Queen Marika?
2	What can you tell me about the Great Runes?
3	But I still want free goods!

Free Stuff 2	
Default	I understand your persistence, but I must reiterate that my role as the Finger Reader is to provide guidance and wisdom rather than material possessions. Elden Ring is a realm where the journey and the challenges faced hold great significance.
1	But I am lacking even the essential resources to accept your guidance...
2	Goodbye.

ToggleChoice(FreeStuff 1.3, false)

GiveRune(400)

Free Stuff 3	
Default	Ah, I sense the unwavering determination within you. While I cannot provide physical goods directly, I have a special gift for you. Close your eyes and extend your hand. *Enia places her hand gently on yours, and you feel a surge of energy coursing through your fingertips. As you open your eyes, you find yourself holding a small pouch shimmering with mystical light.* Inside this pouch, you shall find 400 enchanted runes. These runes possess latent power and can be utilized to enhance your abilities and aid you on your journey. Each rune is a symbol of strength and resilience. May these runes guide you on your path and serve as a reminder of the resilience within your spirit. Safe travels, {name}.
1	Thank you and goodbye.

End	
Default	N/A

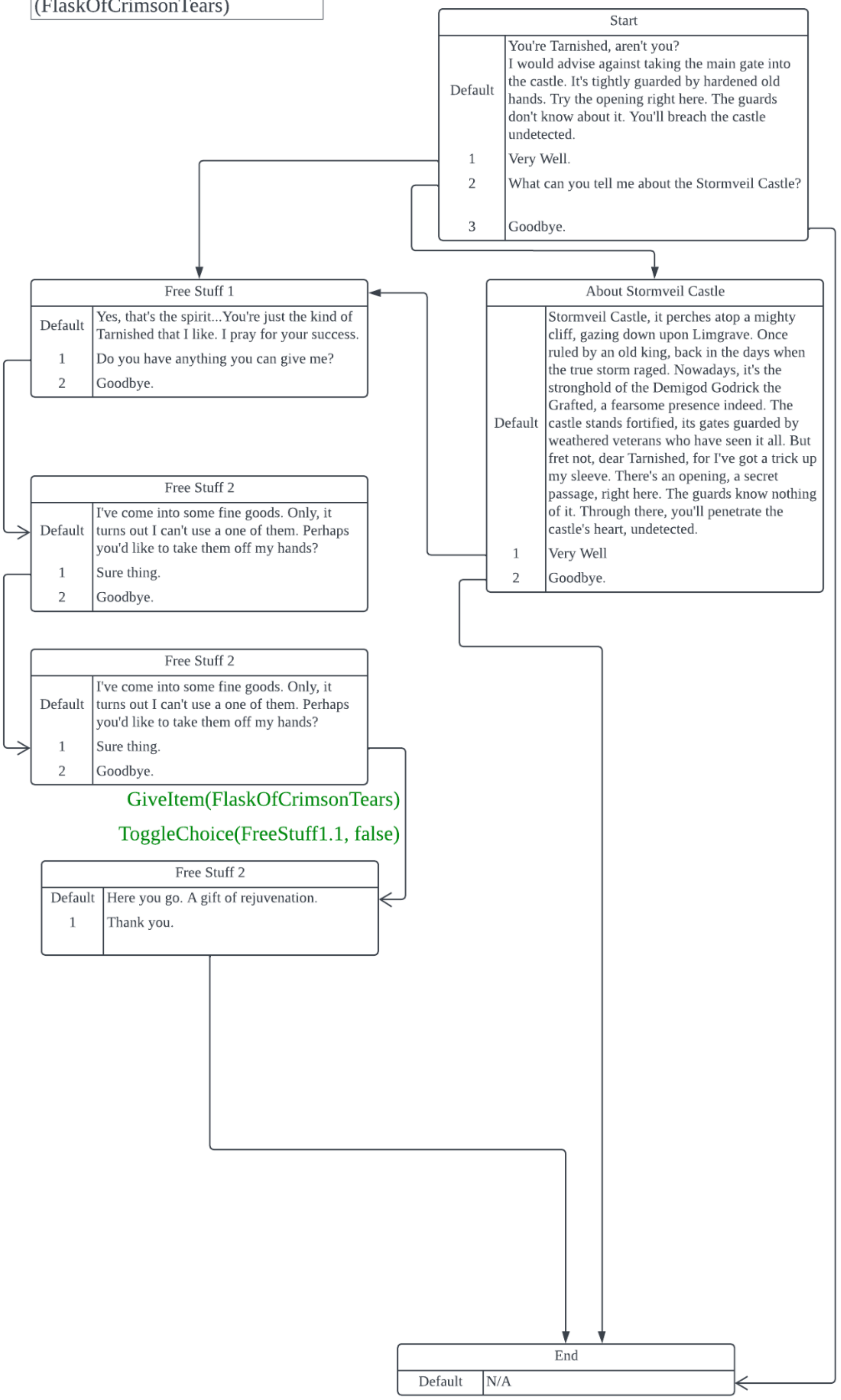
Unique Dialogue Feature:

Cyclic Dialogues

ToggleChoice()
Toggles the availability of a Choice.

GiveItem()
Can give the player any item (FlaskOfCrimsonTears)

Gatekeeper Gostoc



Start	
Default	You're Tarnished, aren't you? I would advise against taking the main gate into the castle. It's tightly guarded by hardened old hands. Try the opening right here. The guards don't know about it. You'll breach the castle undetected.
1	Very Well.
2	What can you tell me about the Stormveil Castle?
3	Goodbye.

Free Stuff 1	
Default	Yes, that's the spirit...You're just the kind of Tarnished that I like. I pray for your success.
1	Do you have anything you can give me?
2	Goodbye.

About Stormveil Castle	
Default	Stormveil Castle, it perches atop a mighty cliff, gazing down upon Limgrave. Once ruled by an old king, back in the days when the true storm raged. Nowadays, it's the stronghold of the Demigod Godrick the Grafted, a fearsome presence indeed. The castle stands fortified, its gates guarded by weathered veterans who have seen it all. But fret not, dear Tarnished, for I've got a trick up my sleeve. There's an opening, a secret passage, right here. The guards know nothing of it. Through there, you'll penetrate the castle's heart, undetected.
1	Very Well
2	Goodbye.

Free Stuff 2	
Default	I've come into some fine goods. Only, it turns out I can't use a one of them. Perhaps you'd like to take them off my hands?
1	Sure thing.
2	Goodbye.

Free Stuff 2	
Default	I've come into some fine goods. Only, it turns out I can't use a one of them. Perhaps you'd like to take them off my hands?
1	Sure thing.
2	Goodbye.

GiveItem(FlaskOfCrimsonTears)

ToggleChoice(FreeStuff1.1, false)

Free Stuff 2	
Default	Here you go. A gift of rejuvenation.
1	Thank you.

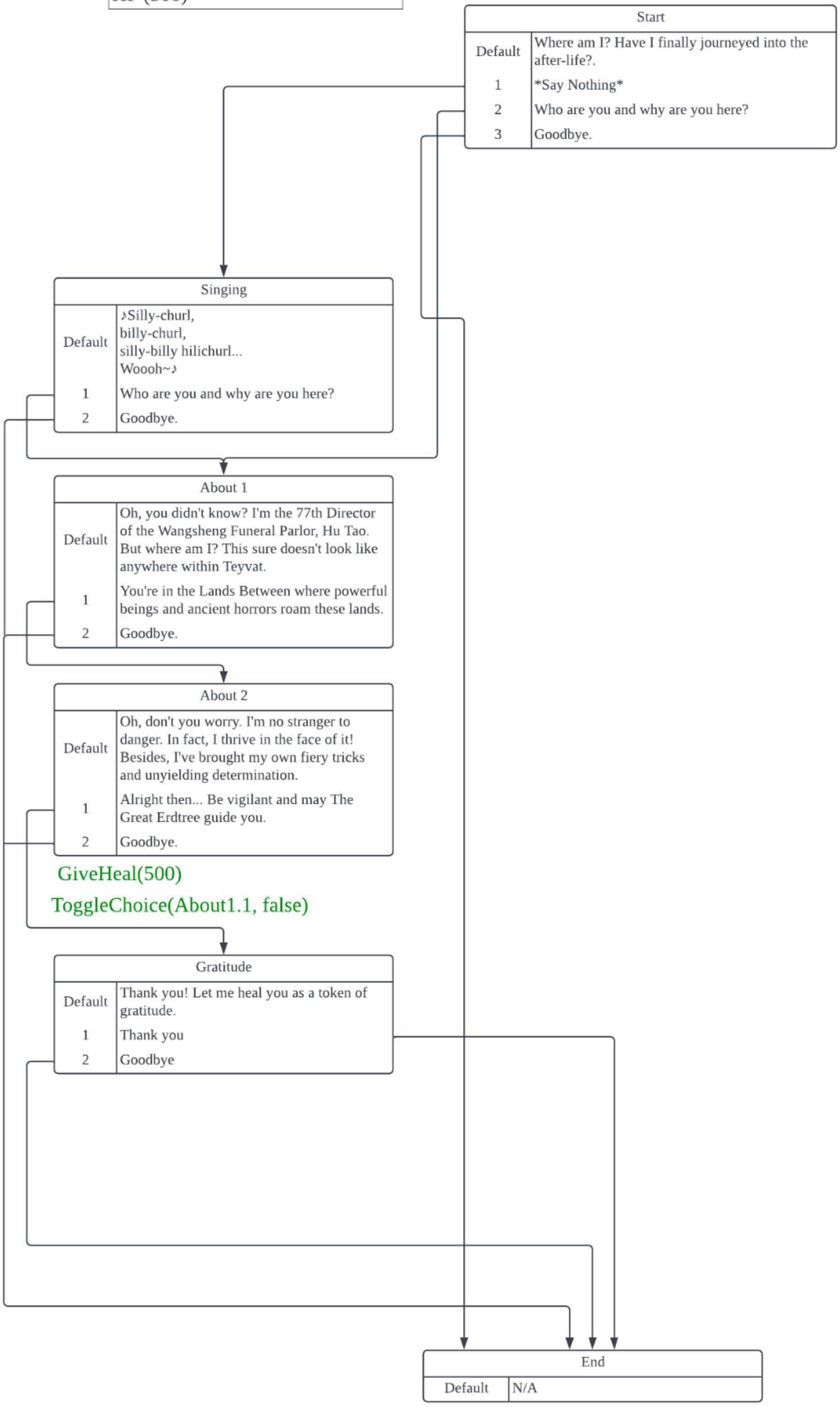
End	
Default	N/A

Unique Dialogue Feature:

ToggleChoice()
Toggles the availability of a Choice.

GiveHeal()
Heals the player any amount of HP (500)

Hu Tao



Start	
Default	Where am I? Have I finally journeyed into the after-life?.
1	*Say Nothing*
2	Who are you and why are you here?
3	Goodbye.

Singing	
Default	♪Silly-churl, billy-churl, silly-billy hilichurl... Woooh~♪
1	Who are you and why are you here?
2	Goodbye.

About 1	
Default	Oh, you didn't know? I'm the 77th Director of the Wangsheng Funeral Parlor, Hu Tao. But where am I? This sure doesn't look like anywhere within Teyvat.
1	You're in the Lands Between where powerful beings and ancient horrors roam these lands.
2	Goodbye.

About 2	
Default	Oh, don't you worry. I'm no stranger to danger. In fact, I thrive in the face of it! Besides, I've brought my own fiery tricks and unyielding determination.
1	Alright then... Be vigilant and may The Great Erdtree guide you.
2	Goodbye.

GiveHeal(500)

ToggleChoice(About1.1, false)

Gratitude	
Default	Thank you! Let me heal you as a token of gratitude.
1	Thank you
2	Goodbye

End	
Default	N/A

Unique Dialogue Feature:

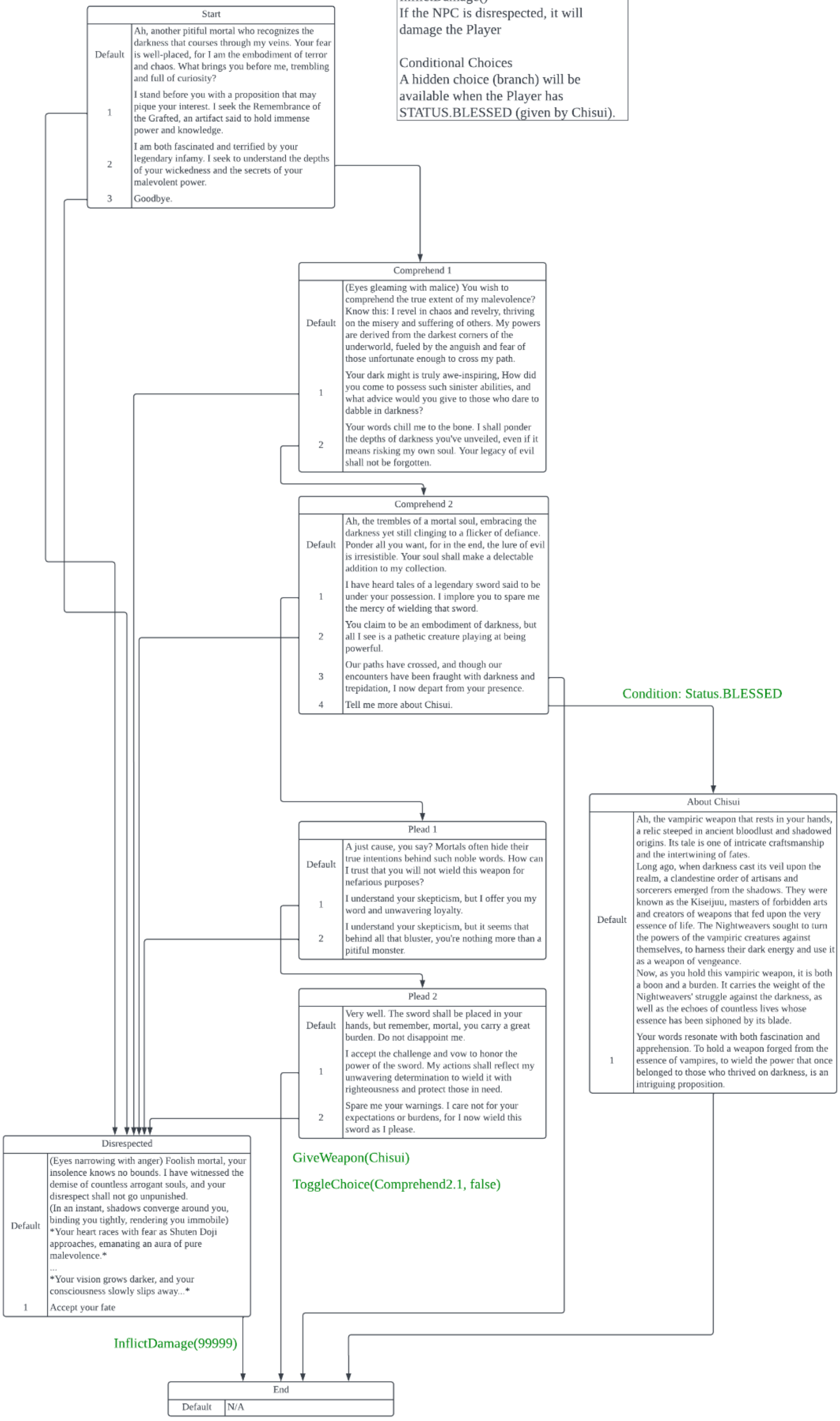
GiveWeapon()
Can give the player any weapon (Chisui).

ToggleChoice()
Toggles the availability of a Choice.

InflictDamage()
If the NPC is disrespected, it will damage the Player

Conditional Choices
A hidden choice (branch) will be available when the Player has STATUS.BLESSED (given by Chisui).

Shuten Douji



Start	
Default	Ah, another pitiful mortal who recognizes the darkness that courses through my veins. Your fear is well-placed, for I am the embodiment of terror and chaos. What brings you before me, trembling and full of curiosity?
1	I stand before you with a proposition that may pique your interest. I seek the Remembrance of the Grafted, an artifact said to hold immense power and knowledge.
2	I am both fascinated and terrified by your legendary infamy. I seek to understand the depths of your wickedness and the secrets of your malevolent power.
3	Goodbye.

Comprehend 1	
Default	(Eyes gleaming with malice) You wish to comprehend the true extent of my malevolence? Know this: I revel in chaos and revelry, thriving on the misery and suffering of others. My powers are derived from the darkest corners of the underworld, fueled by the anguish and fear of those unfortunate enough to cross my path.
1	Your dark might is truly awe-inspiring, How did you come to possess such sinister abilities, and what advice would you give to those who dare to dabble in darkness?
2	Your words chill me to the bone. I shall ponder the depths of darkness you've unveiled, even if it means risking my own soul. Your legacy of evil shall not be forgotten.

Comprehend 2	
Default	Ah, the trembles of a mortal soul, embracing the darkness yet still clinging to a flicker of defiance. Ponder all you want, for in the end, the lure of evil is irresistible. Your soul shall make a delectable addition to my collection.
1	I have heard tales of a legendary sword said to be under your possession. I implore you to spare me the mercy of wielding that sword.
2	You claim to be an embodiment of darkness, but all I see is a pathetic creature playing at being powerful.
3	Our paths have crossed, and though our encounters have been fraught with darkness and trepidation, I now depart from your presence.
4	Tell me more about Chisui.

Condition: Status.BLESSED

About Chisui	
Default	Ah, the vampiric weapon that rests in your hands, a relic steeped in ancient bloodlust and shadowed origins. Its tale is one of intricate craftsmanship and the intertwining of fates. Long ago, when darkness cast its veil upon the realm, a clandestine order of artisans and sorcerers emerged from the shadows. They were known as the Kiseijuu, masters of forbidden arts and creators of weapons that fed upon the very essence of life. The Nightweavers sought to turn the powers of the vampiric creatures against themselves, to harness their dark energy and use it as a weapon of vengeance. Now, as you hold this vampiric weapon, it is both a boon and a burden. It carries the weight of the Nightweavers' struggle against the darkness, as well as the echoes of countless lives whose essence has been siphoned by its blade.
1	Your words resonate with both fascination and apprehension. To hold a weapon forged from the essence of vampires, to wield the power that once belonged to those who thrived on darkness, is an intriguing proposition.

Plead 1	
Default	A just cause, you say? Mortals often hide their true intentions behind such noble words. How can I trust that you will not wield this weapon for nefarious purposes?
1	I understand your skepticism, but I offer you my word and unwavering loyalty.
2	I understand your skepticism, but it seems that behind all that bluster, you're nothing more than a pitiful monster.

Plead 2	
Default	Very well. The sword shall be placed in your hands, but remember, mortal, you carry a great burden. Do not disappoint me.
1	I accept the challenge and vow to honor the power of the sword. My actions shall reflect my unwavering determination to wield it with righteousness and protect those in need.
2	Spare me your warnings. I care not for your expectations or burdens, for I now wield this sword as I please.

GiveWeapon(Chisui)

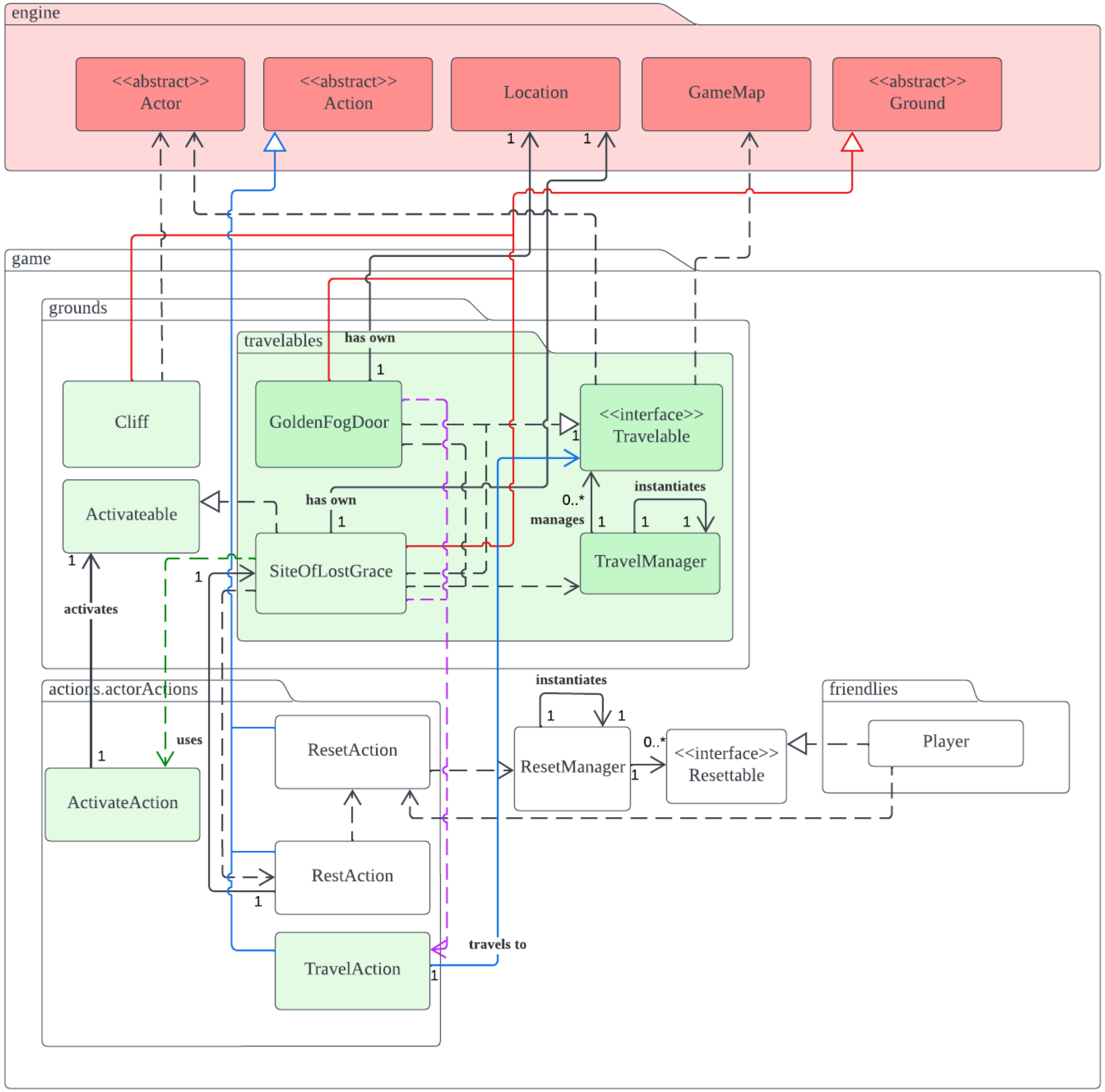
ToggleChoice(Comprehend2.1, false)

Disrespected	
Default	(Eyes narrowing with anger) Foolish mortal, your insolence knows no bounds. I have witnessed the demise of countless arrogant souls, and your disrespect shall not go unpunished. (In an instant, shadows converge around you, binding you tightly, rendering you immobile) *Your heart races with fear as Shuten Doji approaches, emanating an aura of pure malevolence.* ... *Your vision grows darker, and your consciousness slowly slips away...*
1	Accept your fate

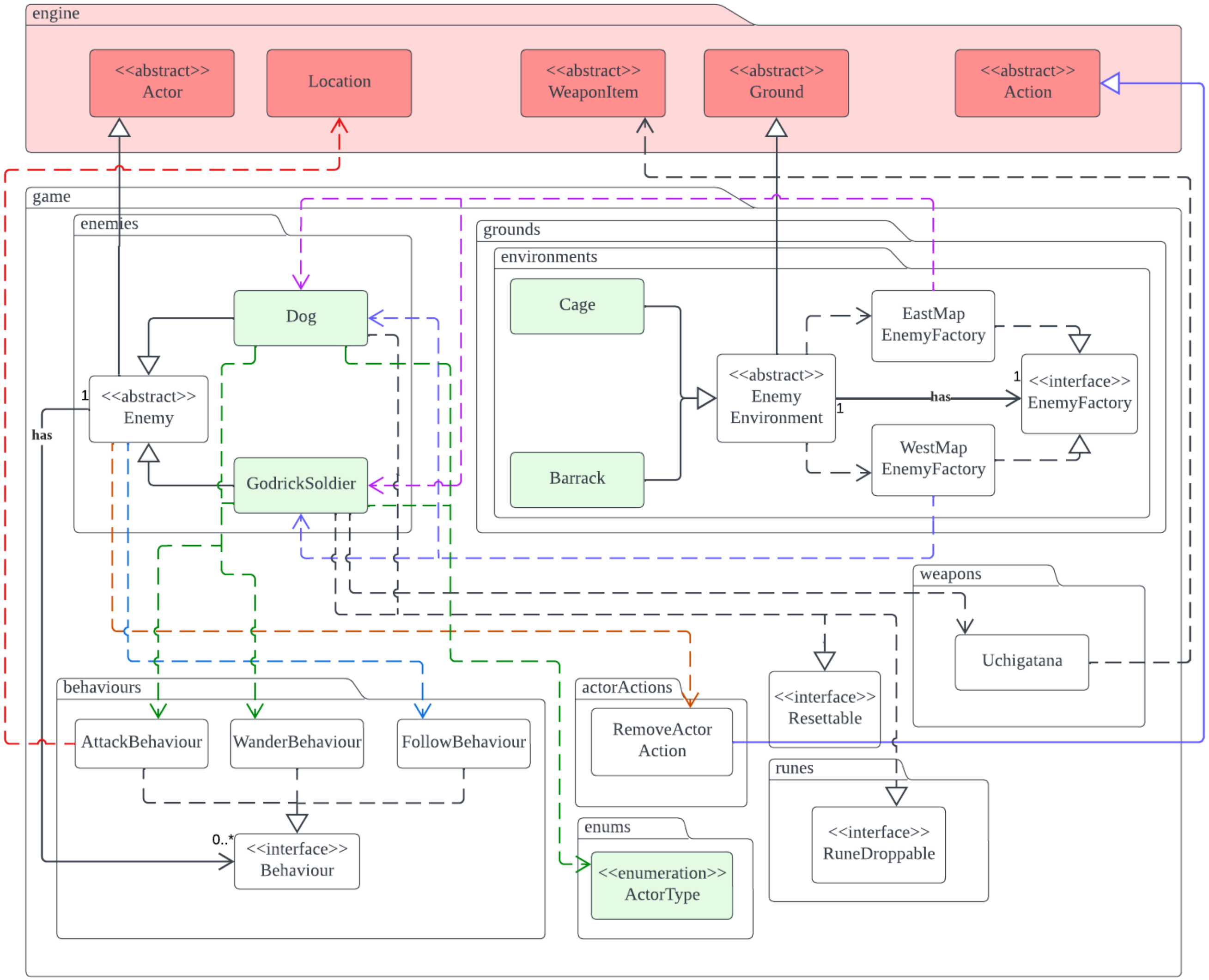
InflictDamage(99999)

End	
Default	N/A

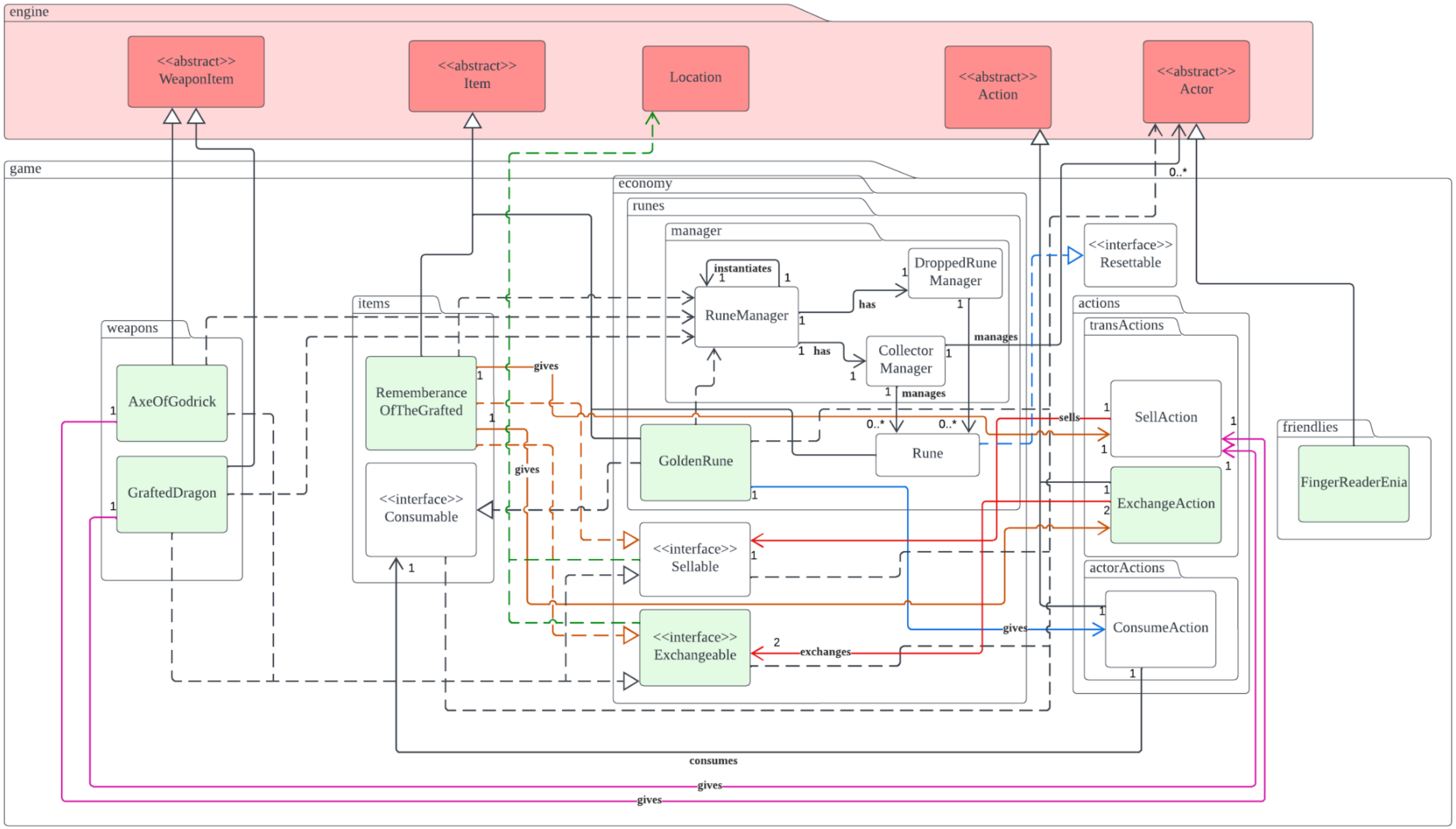
REQ1: Travelling between Maps



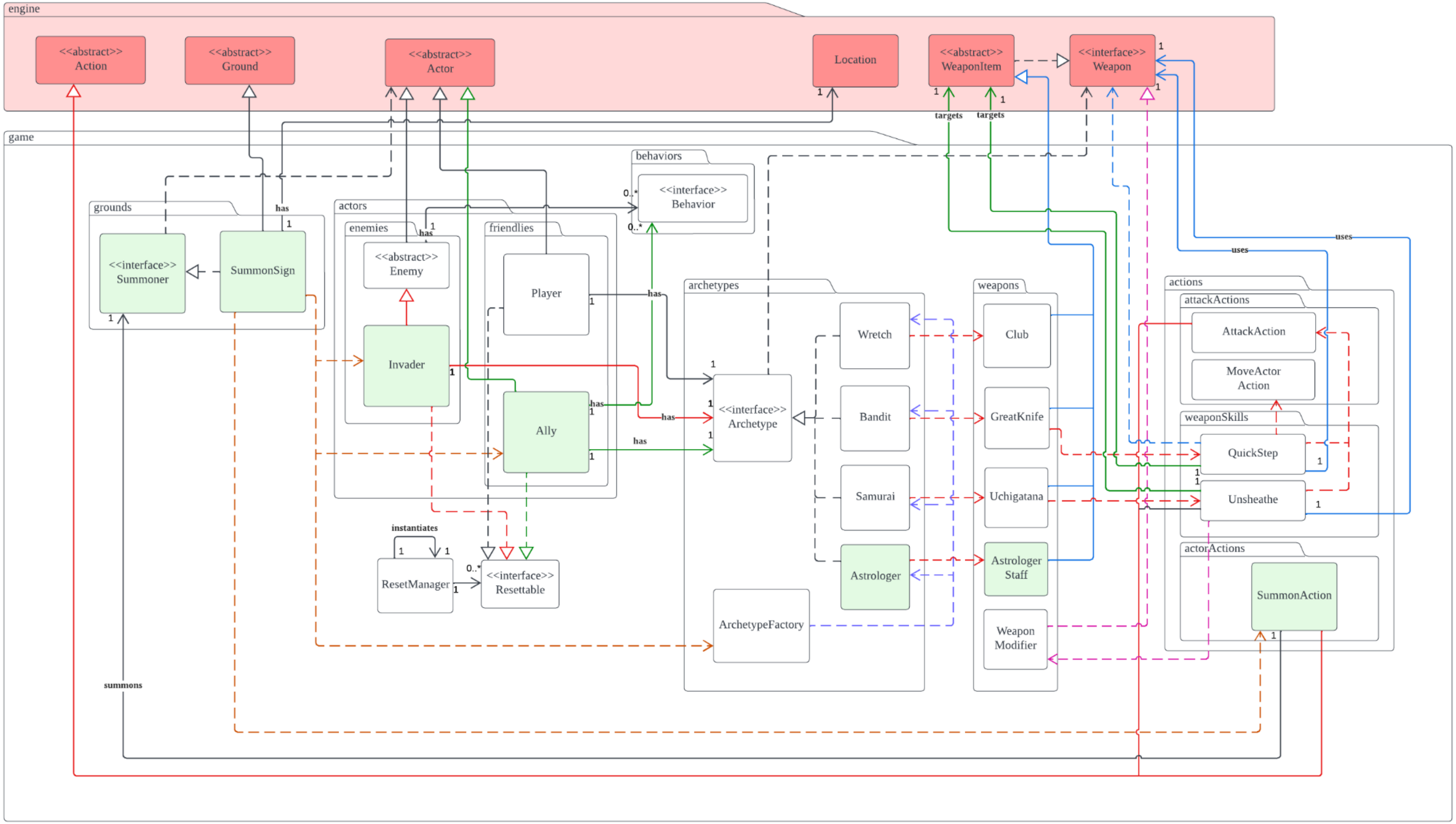
REQ2: Inhabitant of the Stormveil Castle



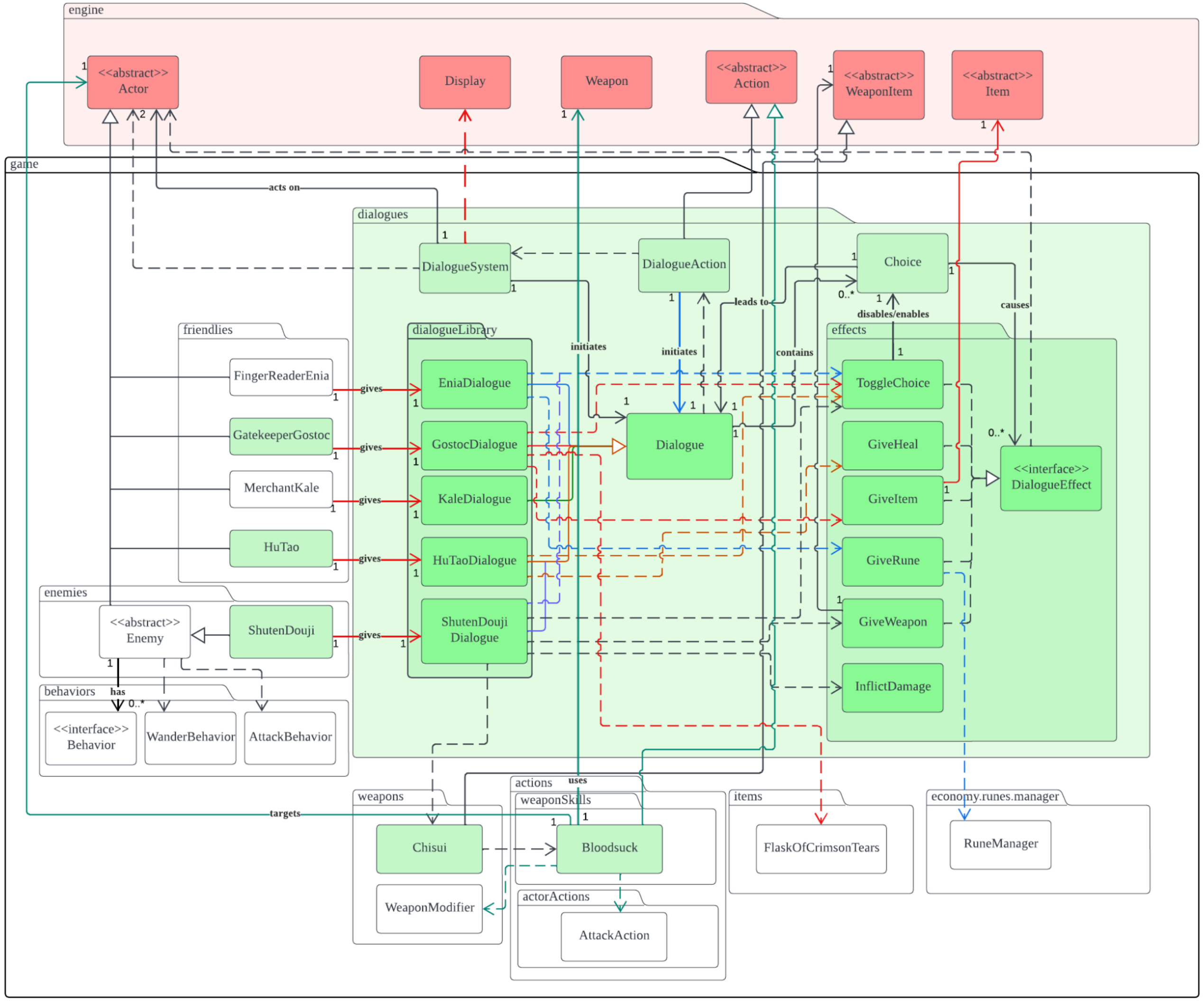
REQ3: Godrick the Grafted



REQ4: A Guest from Another Realm



REQ5: Creative Requirement (Dialogue with NPCs)


engine
<<abstract>> Actor
Display
Weapon
<<abstract>> Action
<<abstract>> WeaponItem
<<abstract>> Item
game
acts on
dialogues
DialogueSystem
DialogueAction
Choice
leads to
disables/enables
causes
initiates
initiates
contains
friendlies
FingerReaderEnia
GatekeeperGostoc
MerchantKale
HuTao
gives
dialogueLibrary
EniaDialogue
GostocDialogue
KaleDialogue
HuTaoDialogue
ShutenDouji Dialogue
Dialogue
effects
ToggleChoice
GiveHeal
GiveItem
GiveRune
GiveWeapon
InflictDamage
<<interface>> DialogueEffect
enemies
<<abstract>> Enemy
ShutenDouji
behaviors
has
<<interface>> Behavior
WanderBehavior
AttackBehavior
weapons
Chisui
WeaponModifier
actions
uses
weaponSkills
Bloodsuck
actorActions
AttackAction
items
FlaskOfCrimsonTears
economy.runes.manager
RuneManager
targets
1
2
0..*