Student name: Ngu Trien Luan

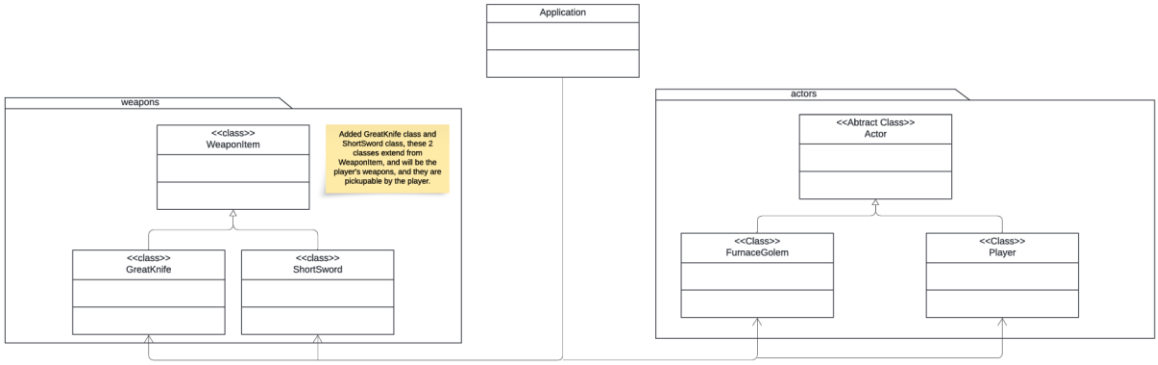
Student ID: 34347860

Design Rationale

Desgin Objective:

The design focuses on dividing the system into smaller, manageable classes to encourage modularity and encapsulation. Each class will have a specific, well-defined role, making the system easier to understand and maintain. By following the SOLID and DRY principles, the design should facilitate easy extension and support reusability. This involves recognizing shared functionality, abstracting it into reusable components, and anticipating changes to improve code flexibility and adaptability for future modifications and enhancements.

REQ 1:

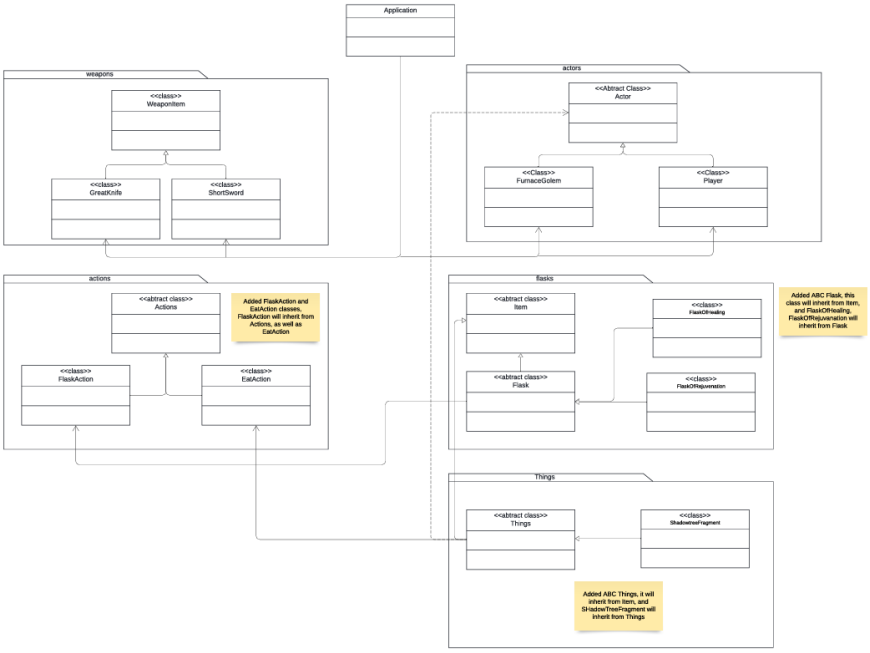


Classes Modified/Created	Responsibilities	Rationale
GreatKnife Extends WeaponItem	Class respresents a Great Knife (weapon) Relationships: - It extends the WeaponItem class to achieve portable functionality.	Reason for decision: - **Single Responsibility Principle (SRP):** The GreatKnife and Short Sword class are responsible only for the behavior and properties of the Great Knife weapon and Short Sword weapon. They do not handle any logic outside of this domain, such as player stats or combat mechanics. This clear separation of responsibilities keeps the class focused and manageable. - **Open-Closed Principle (OCP)**: The classes can be extended
ShortSword Extends WeaponItem	Class respresents a Short Sword (weapon) Relationships: It extends the WeaponItem class to achieve portable functionality.	

		without modifying its core structure. For instance, the attack method can be overridden to provide custom attack behavior for the Great Knife, Short Sowrd, without changing the parent WeaponItem class, keeping it closed for modification but open for extension. - **Liskov Substitution Principle (LSP)**: The GreatKnife class and Short Sword can be used in place of its parent class, WeaponItem, without breaking the game's logic. Since it inherits and follows the expected behavior of a WeaponItem, objects of GreatKnife and Short Sword can substitute WeaponItem objects and still function correctly. - **Interface Segregation Principle (ISP):** The GreatKnife ans Short Sword class does not implement unnecessary methods. It only focuses on what is relevant to a weapon, like attack actions and pickup rules, ensuring it doesn't carry methods that are irrelevant to its role. - **Dependency Inversion Principle(DIP):** Keeps the system flexible and allows different types of weapons to be easily integrated or swapped out. - The GreatKnife and Short Sword class avoids code duplication by inheriting from the WeaponItem class. Shared behaviors, such as attacking, are handled by the parent class, allowing reuse without duplicating attack logic. The attack method calls the superclass implementation rather than writing new logic, adhering to DRY by reusing common code. - The class design is generally strong, balancing flexibility and maintainability, but the abstraction and inheritance can lead to increased complexity and

		potential pitfalls if not carefully managed.

REQ 2:

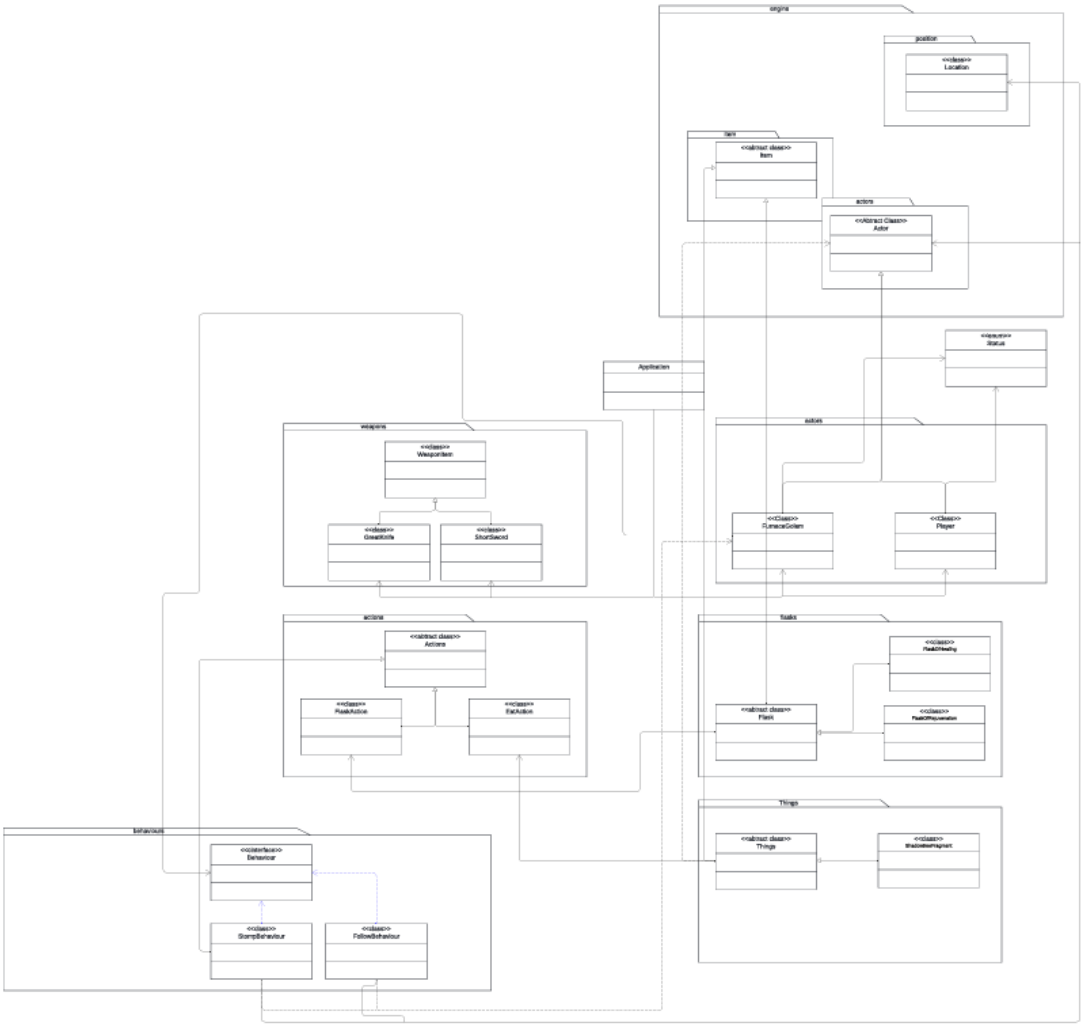


Classes Modified/Created	Responsibilities	Rationale
EatAction Extends Actions	Class respresents eating action Relationships: - It extends the Actions class - Associated with Things class	Reason for decision: - **Single Responsibility Principle (SRP)**: The 2 classes, Eat and Flask Action classes, both perform only for the behavior and properties of their actions. They are not responsible for other classes or behaviours. This clear separation of responsibilities keeps the class focused and manageable. - **Open-Closed Principle (OCP)**: The classes and be extended without changing any structure of the class. We can simply override the function in the class to get the class do as we want, for instance eat or flask.
FlaskAction Extends Actions	Class respresents using flask action Relationships: - It extends the Action class - Associated with FlaskAction class	

		- **Liskov Substitution Principle (LSP)**: Since the 2 Actions inherit from Actions abc, we can simply use them instead of using Actions class, and still follows the structure. - **Interface Segregation Principle (ISP):** The 2 classes don't implement unnecessary methods. It only override Actions methods, ensuring it doesn't carry methods that are irrelevant to its role. - **Dependency Inversion Principle(DIP):** Keeps the system flexible, can call the Action multiple times, without any redundant code
Flask (Abstract class) Extend from Item	Class representing a flask Relationship: - Extends the Item class	Reason for decision: - **Single Responsibility Principle (SRP)**: Flask only handles the core properties and behavior of an item. Specific flask effects are delegated to subclasses. - **Open-Closed Principle (OCP)**: The class is open for extension via subclasses like FlaskOfHealing, allowing specific effects without changing the core class. - **Liskov Substitution Principle (LSP)**: Since it extends Item, any Flask object can be used wherever an Item is expected, preserving expected behavior. - **DRY Principle**: Common properties and methods are centralized in the Flask class, and specific flask types only need to implement the unique aspects.
FlaskOfHealing extends Flask	Represents a specific flask type that heals players. **Relationships**: - Inherits from Flask and extends its functionality to provide healing when used.	Reason for decision: - **Single Responsibility Principle (SRP)**: This class is only responsible for implementing the healing effect, with the general item behavior being inherited. - **Open-Closed Principle (OCP)**: The class can be further extended if more specific healing types are needed (e.g., greater healing flask).

		- **Liskov Substitution Principle (LSP)**: Since FlaskOfHealing extends Flask, it can be used wherever a Flask or Item is expected. - **DRY Principle**: The class reuses logic from Flask and focuses only on the additional functionality of healing.
ShadowtreeFragment extends Item	Represents Shadowtree's fragment. **Relationships**: - Extends the Item class, inheriting common item functionality.	- **Single Responsibility Principle (SRP)**: The class focuses solely on defining the properties of the Shadowtree fragment, following the item's general behavior. - **Open-Closed Principle (OCP)**: If needed, additional behavior can be added to the fragment without altering the base class. - **Liskov Substitution Principle (LSP)**: The class can replace any Item in the game and will behave as expected. - **DRY Principle**: Reuses general Item functionality, avoiding duplicating code for item interactions.
Things Extends Item	Represents general utility methods or constants for in-game items that player can pickup **Relationships**: - Extend from Item	- **Single Responsibility Principle (SRP):** This class centralizes utility logic that doesn't belong in any one class, such as shared methods or constants. - **Open-Closed Principle (OCP):** The utility methods can be extended without modifying existing ones. - **DRY Principle:** By providing shared functionality, the class reduces code duplication across different item-related classes.

REQ 3:



Classes Modified/Created	Responsibilities	Rationale
FollowBehaviour Extends Behaviour	This class represents a behavior in which an actor follows a target if they are within a certain range. **Relationships**: - Implements Behaviour, which defines a behavior that can be performed by an actor. - Interacts with Player, Actor, and Location to determine whether the target is in range	Reasson for Decision: - **Single Responsibility Principle (SRP)**: The class focuses on a singular task: handling the follow behavior of an actor, separating it from other behavior types. - **Open-Closed Principle (OCP)**: If different follow behaviors are needed, this class can be extended without modifying the core follow logic (e.g., a conditional follow behavior).

	and if the actor should move towards the target.	- **Liskov Substitution Principle (LSP)**: As it implements Behaviour, FollowBehaviour can be used interchangeably with other behavior types in any class that expects a Behaviour instance. - **DRY Principle**: The class avoids redundant logic by reusing components from the GameMap and Location classes to calculate actor positions and determine the next move.
StompBehaviour Extends Behaviour	Represents a stomp action that an actor can perform to damage a player **Relationships**: - Implements Behaviour to provide stomping functionality. - Interacts with Actor, Player, GameMap, and Location for determining stomp range, target position, and potential obstacles.	- **Single Responsibility Principle (SRP)**: The class is responsible for implementing the stomp action, separating it from other attack behaviors. - **Open-Closed Principle (OCP)**: The stomp behavior can be extended to add additional effects (e.g., stronger damage or area-of-effect attacks) without modifying existing functionality. - **Liskov Substitution Principle (LSP)**: It can be substituted for any other Behaviour without breaking the system. - **DRY Principle**: Common functionalities like distance calculations and map checks are reused via the Location and GameMap classes, reducing code duplication.
Floor Extends Ground	Represents a type of ground (floor) inside a building that can be interacted with by actors. **Relationships**: - Extends the Ground class, inheriting basic ground functionality. - Uses Actor to determine if an actor can enter the floor based on its status.	- **Single Responsibility Principle (SRP):** The class is responsible solely for defining the behavior of a floor, including the rules for actor interaction. - **Open-Closed Principle (OCP):** The class is open for extension; additional functionalities (such as reactions to other statuses) can be added without altering the core floor logic. - **Liskov Substitution Principle (LSP):** As it extends Ground, Floor can be used wherever a Ground object is expected, ensuring substitutability without breaking existing logic.

		- **DRY Principle:** Common logic for checking whether an actor can enter the ground is inherited from the Ground class, avoiding code duplication**.**

REQ 4:

Classes Modified/Created	Responsibilities	Rationale
StompBehaviour Extends Behaviour	Represents a behavior where an actor can stomp and damage another actor or interact with the ground. **Relationships**: - Implements the Behaviour interface and interacts with Actor, GameMap, and Location to execute	- **Single Responsibility Principle (SRP)**: The class focuses solely on the logic for executing a stomp action. - **Open-Closed Principle (OCP)**: New variations of the stomp behavior can be added without modifying the existing logic. - **Liskov Substitution Principle (LSP)**: The class can be used as any other behavior within the

	the stomp action based on distance.	game since it implements Behaviour. - **DRY Principle**: The class avoids code duplication by utilizing existing game mechanics such as GameMap and Location for determining range and target interactions.
Fire Extends Ground	Represents a fire on the ground, capable of damaging actors with the BURN status. **Relationships**: - Inherits from the Ground class and interacts with Actor and Location to inflict damage on actors with the BURN status. - Fire extinguishes itself after 5 game ticks.	- **Single Responsibility Principle (SRP)**: The class handles the behavior of fire, including damage infliction and self-extinguishing. - **Open-Closed Principle (OCP)**: New behaviors, such as more complex fire mechanics, can be added without modifying existing functionality. - **Liskov Substitution Principle (LSP)**: Since it extends `Ground`, it can replace any other ground types without breaking the game. - **DRY Principle**: It leverages the base `Ground` functionality for position and interaction logic while adding specific fire-related behavior.
Puddle Extends Ground	Represents a puddle of water on the ground within the game. **Relationships**: - Inherits from the Ground class to provide functionality for actors interacting with ground elements.	- **Single Responsibility Principle (SRP):** The class is only responsible for representing a puddle and its behavior in the game. - **Open-Closed Principle (OCP):** The class can be extended if additional behaviors related to puddles are required without modifying the core logic. - **Liskov Substitution Principle (LSP):** It can be used wherever a Ground object is expected,

		ensuring compatibility with other ground types. - **DRY Principle:** The core logic for ground interaction is inherited from the Ground class, preventing redundancy.