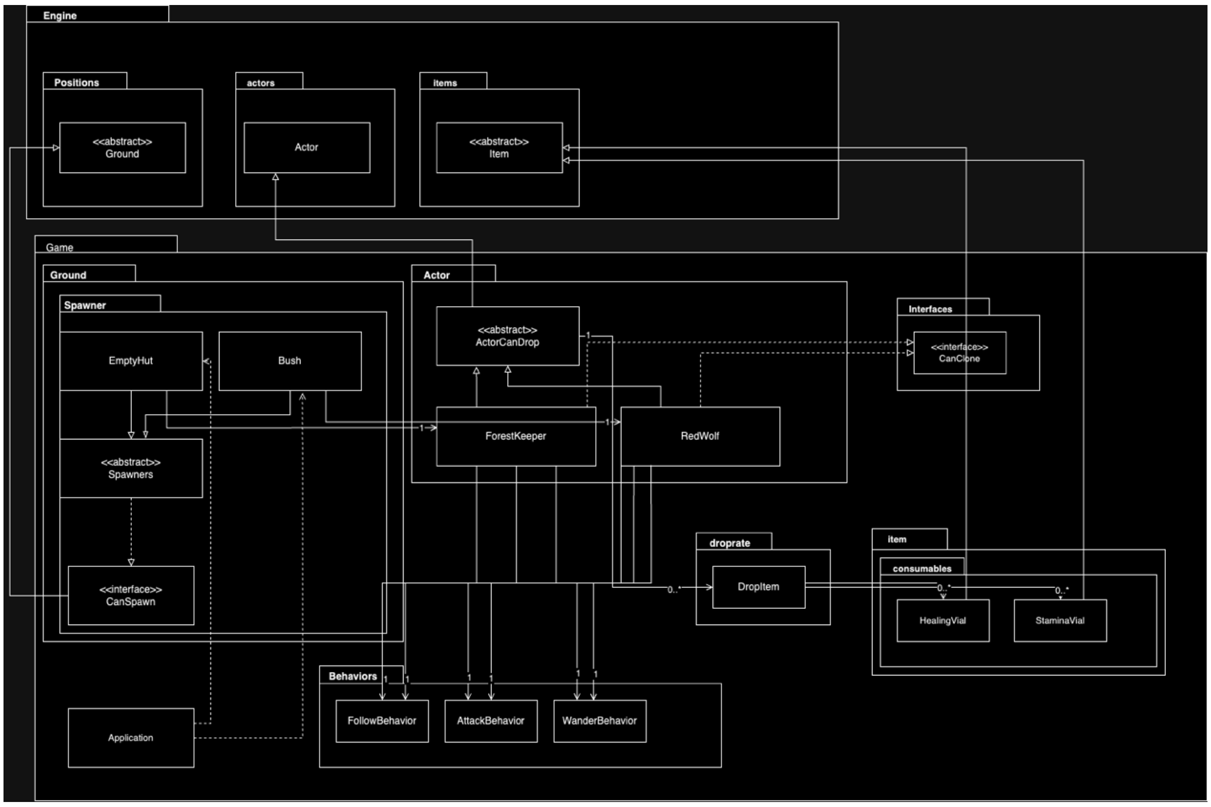
Requirement 1 Design Rationale



The code adheres to the DRY principle by using inheritance to create common behavior in classes like "Bush" and "EmptyHut." Both inherit from the "Spawners" class, which encapsulates shared functionality for spawning and interactions with the game world. This avoids code duplication and promotes code reusability. Adding on, the abstract class ActorCanDrop also provides all the functionality for any actors that can drop item (e.g. HealingVial & StaminaFlask) once they meet their demise.

- **Advantage**: This approach reduces redundancy and makes maintenance easier.

Each class adheres to the Single Responsibility Principle. For instance, the "Bush" class is responsible for representing the terrain and defining its default properties, while the "Spawners" class handles spawning mechanics, and "RedWolf" represents a specific game character. Same goes for EmptyHut class and ForestKeeper class.

Furthermore, The design supports OCP by allowing easy extension. If a new type of terrain or character needs to be added, one can create a new subclass of "Spawners" or "Actor/ActorCanDrop" without modifying existing code.

Lastly, my code achieves Dependency Inversion Principle. For example, The "Bush" class depends on the "Spawners" class, adhering to the DIP. High-level modules (e.g., "Bush") depend on abstractions (e.g., "Spawners") rather than low-level details.

- **Advantage**: This promotes loose coupling and allows for flexibility in the implementation of "Spawners."

Disadvantages:

- The code doesn't address error handling or exception management, which is important for robust game development.
- Documentation and comments should be improved to enhance code readability and understandability.