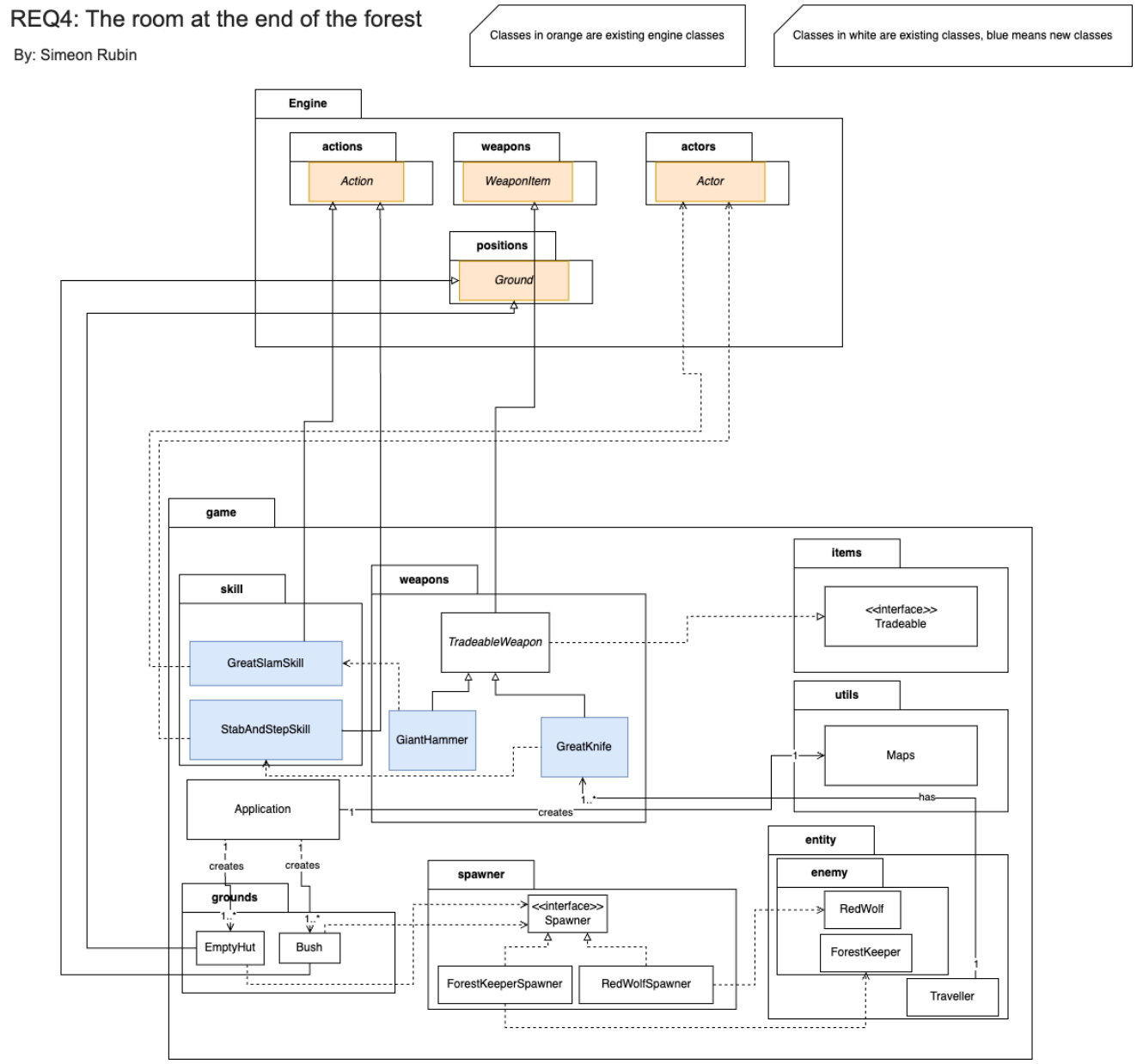
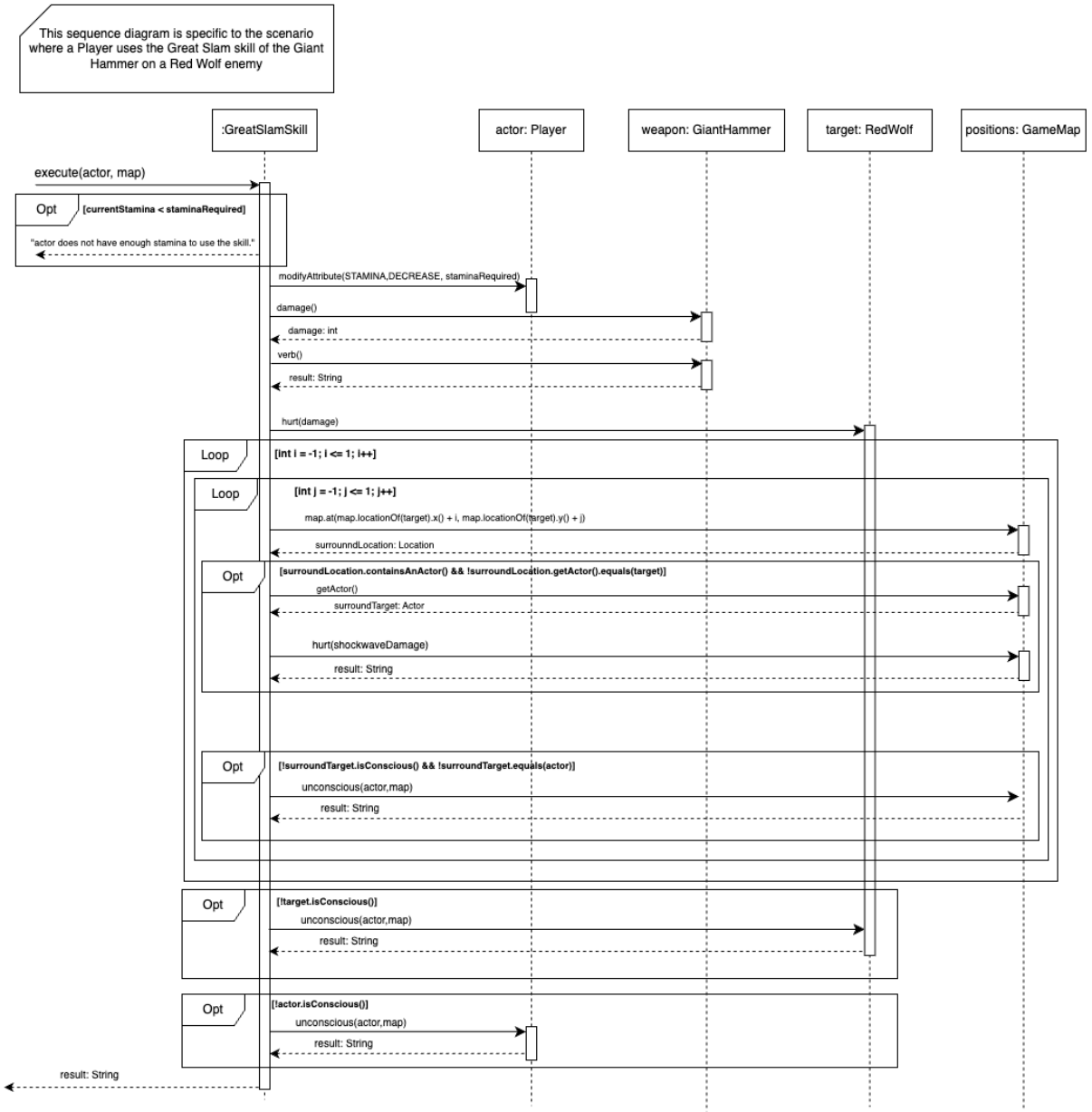
REQ4 Rationale



REQ4: Sequence Diagram

By: Simeon Rubin



In order to prepare the design for this assignment, minimal changes were made to the areas of functionality relevant for this requirement.
The previous design has weapon skill classes implementing the ActivateSkill interface, however, this design has them extend the Action abstract class. The reason for this change is that the ActivateSkill is more appropriate for weapon's whose ability is a stat buff i.e. FocusSkill. The skills in this requirement are more like a special action type and for that reason they extend the Action classes' methods.

For the implementation of this requirement I kept the design as simple as possible, ensuring I added only the necessary classes to achieve the functionality. The GreatKnife weapon item class was added to the system which extends TradeableWeapon and has a dependancy with the StabAndStepSkill class. This class handles the functionality of this weapon's special ability.This was added to the Traveller inventory and thus, the Traveller class has an association with the GreatKnife class.
A similar implementation was followed for the GiantHammer which also extends the TradeableWeapon class and has a dependency on the GreatSlamSkill which handles it's great slam ability.
This design adheres to the Single-Responsibility Principle as both of the weapon classes are solely responsible for their respective weapon objects and their specifications. All additional ability such as AttackAction or special abilities such as StabAndStepSkill and GreatSlamSkill are handled by separate classes which execute the more complicated logic of these respective abilities. This also adheres to the Open-Closed Principle as the addition of new weapon classes into the system does not require one to change existing code by simply to add the new weapon classes into the system and have them inherit the TradeableWeapon class. Another SOLID principle which is considered in the design for this requirement is that of the Dependency Inversion Principle. All non-abstract classes extend an abstract classes' features in this design; the weapon classes extend TradeableWeapon and the skill classes for the weapons extend the abstract Action class from the engine. This creates loose coupling in the design, ensuring the high level classes are independent of the functionality of the low-level classes.

Smaller modifications include the addition of a new map and the integration of previous components of the design to add new EmptyHut and Bush Ground types which spawn ForestKeeper and RedWolf enemy types respectively.
The reuse of previous class logic in adding new components to the system limits repeated code and keeps the system simpler and more efficient.