SWEN30006 Project 1

Workshop 03, Team 11

Name1: Siyi Zhou

Name2: Ziheng Cao

Name3: Kang Pi

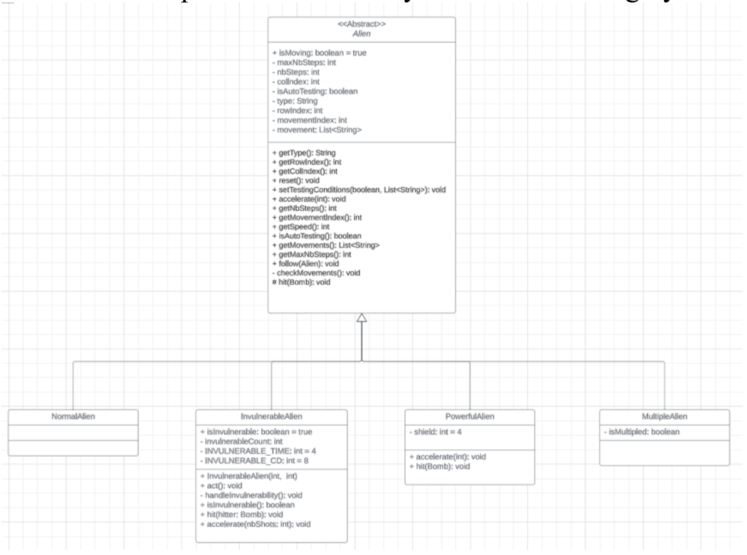
(3 member team)

In our journey to enhance the SpaceInvader game's design, we recognized the significance of maintaining the original framework while accommodating innovative features. Our objective was to uphold high cohesion, reduce class coupling, and ensure modularity in software design following the GRASP pattern(General Responsibility Assignment Software Patterns). To achieve this goal, we introduced pivotal modules such as GameVersion, Explosion class, Explodable interface and Accelerable interface, and specialized Alien subclasses, all smoothly integrated into the existing structure.

When entering the design level, we observed that the SpaceInvader class obtained too many responsibilities and thus needed more cohesion. To increase the cohesion for the SpaceInvader class, we created an artificial abstract class, GameVersion, as an intermediary between the SpaceInvader class and all other actors' classes. Then the SpaceInvader class created an instance of GameVersion based on the type and delegated its previous responsibility of operating the game to the GameVersion class. After modification, the SpaceInvader owned only two responsibilities: one was creating the game version, and the other was creating the log result output.

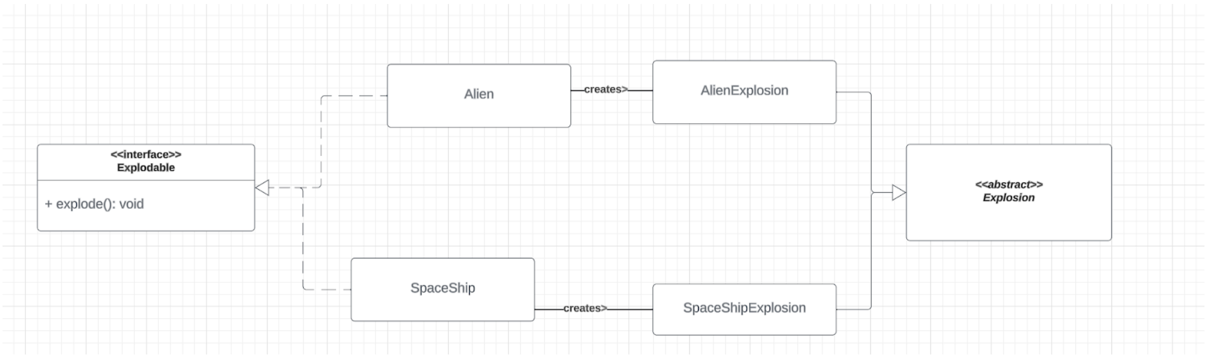
The GameVersion class was created as a game controller to handle the game's logic based on the different versions, "Simple" or "Plus". We can see that "Pure fabrication" and "Polymorphism" principles have been applied. The creation of GameVersion is an example of pure fabrication, and it allows the GameVersion class to achieve high cohesion by focusing on responsibilities related to operating the game, including setting up the environment and determining the game progress. Furthermore, we used the "polymorphism" principle to create Simple and Plus classes extended from GameVersion. This design allows the GameVersion superclass to delegate different game logic to its corresponding child classes. Besides, this design also helps to encapsulate game logic and behaviour per version, which decreases the coupling between modulus. For example, suppose the game company wants to remove the Simple version in the future. In that case, it can be done by deleting the Simple class without modifying functionality from other version classes. Hence, the creation of the GameVersion class and polymorphic development for the GameVersion class embody the "protected variation" principle, demonstrated by its open-for- extension and closed-by modification characteristics.

The design of the Alien serves as an abstract class, creating a template that enforces a consistent interface while allowing for subclass-specific implementations that adhere closely to GRASP to ensure high-quality object-oriented design. By localising specialised behaviours within subclasses (InvulnerableAlien, MultipleAlien, NormalAlien, PowerfulAlien), the structure achieves High Cohesion, allowing each class to manage its responsibilities without becoming an 'Anti-Pattern.' Using overridden methods like act() and hit() in the various Alien subclasses minimises coupling and maximises adaptability, making the structure more resilient to future changes. Additionally, these overrides serve as a mechanism for Polymorphism, which allows various types of aliens to display unique behaviours even when managed as generic Alien objects in the game grid. For instance, the InvulnerableAlien class overrides the hit() method to check its invulnerability status before deciding whether to take damage. This contrasts with the behaviour in NormalAlien, which would take damage when hit. Similarly, PowerfulAlien uses its version of hit() to deplete its shields before taking any actual damage. These specialised behaviours are encapsulated within each subclass. However, they are triggered via the same method name (hit() or act()), allowing the parent Alien class or game engine to interact with all aliens through a standard interface. The Creator pattern is realised as the Alien class, acting as a centralised factory for generating various alien instances. The InvulnerableAlien class employs Pure Fabrication for its unique invulnerability logic, which does not naturally fit within the general Alien class but is essential for that subclass. Information Expert is applied in every specialised subclass, where each class contains the logic most pertinent to its specialised behaviour; for instance, PowerfulAlien knows precisely how to manage its shield strength. Lastly, Protected Variations are employed in encapsulating the variability of features like shields and invulnerability, thus insulating the more extensive system from potential changes in these aspects. These GRASP patterns collectively contribute to a highly modular, flexible, and maintainable design.

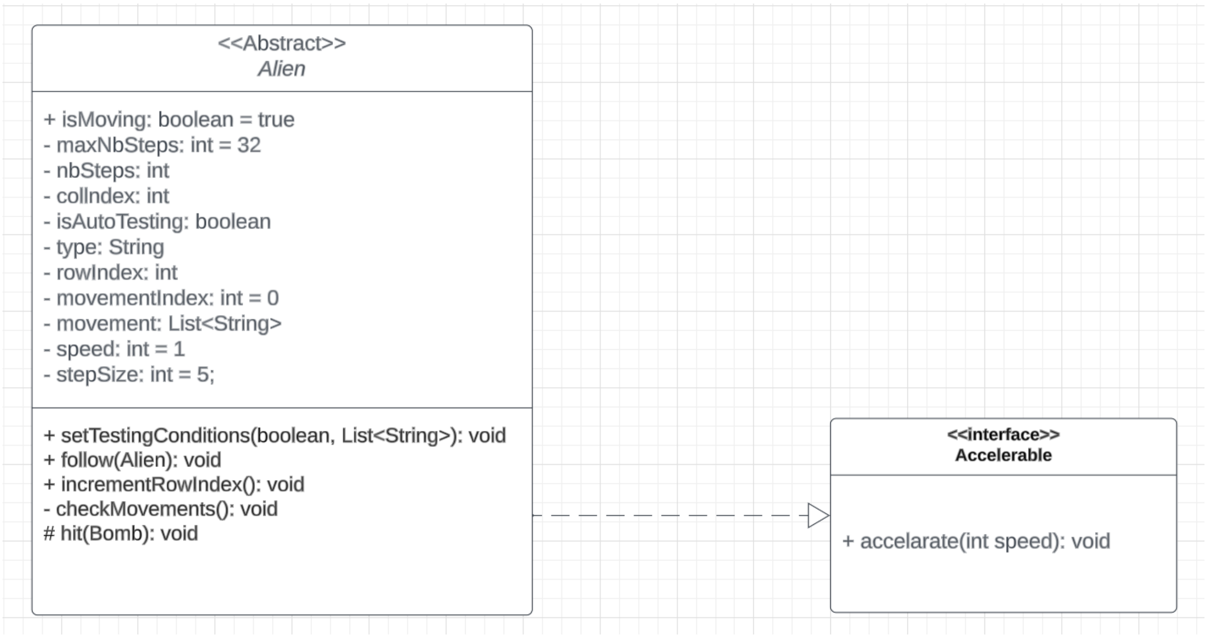


The original design of the Explosion() method, which created an explosion image from the 'Bomb' class, presents a concern regarding the principle of Single Responsibility and cohesion. The Bomb class should focus on hitting behaviour, while exploding should be performed by Alien and Spaceship when they are hit by bombs. Therefore, we employed the "Creator" principle, to determine that Explosion should be created by Alien and SpaceShip classes.

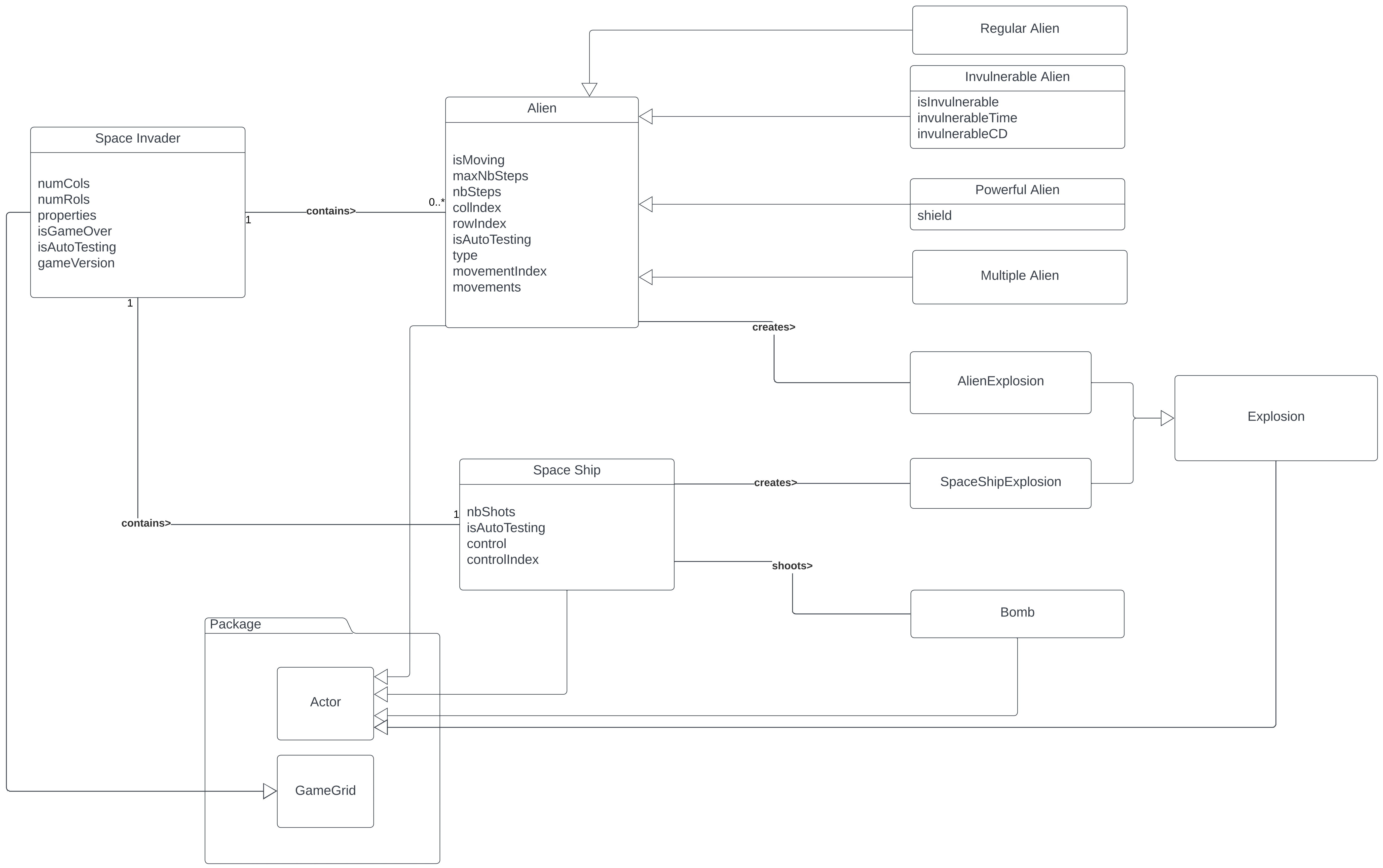
To achieve the explosion feature for Alien and SpaceShip while minimising the coupling between modulus, we employed the "Polymorphism" principle by modifying the Explosion class as an abstract class and creating the Explodable interface. The Explosion class is represented as an actor and extended from Actor class, to indicate that every explosion has its image and rendering time to the UI when the Alien is dead, or SpaceShip is hit. The Explosion class delegated its responsibility for the explosion to child classes (AlienExplosion and SpaceShipExplosion) when a particular explosion happened. The design that each child class owns its own features of explosion embodies the concept of high cohesion. To ensure that both Alien and SpaceShip classes will create an explosion actor after they are hit by bombs, we created the Explodable interface. This interface contains the explode() method and should be implemented by all actor classes that obtain exploding behaviour. This change of delegating exploding behaviour to an Explodable interface decreases the coupling between classes. For example, those classes can easily remove exploding behaviour by swapping out implementations from the interface without affecting the entire class. The explosion classes work with the Explodable interface, increasing the extensibility for more explosion features and achieving the decoupling for classes that own explosion features.

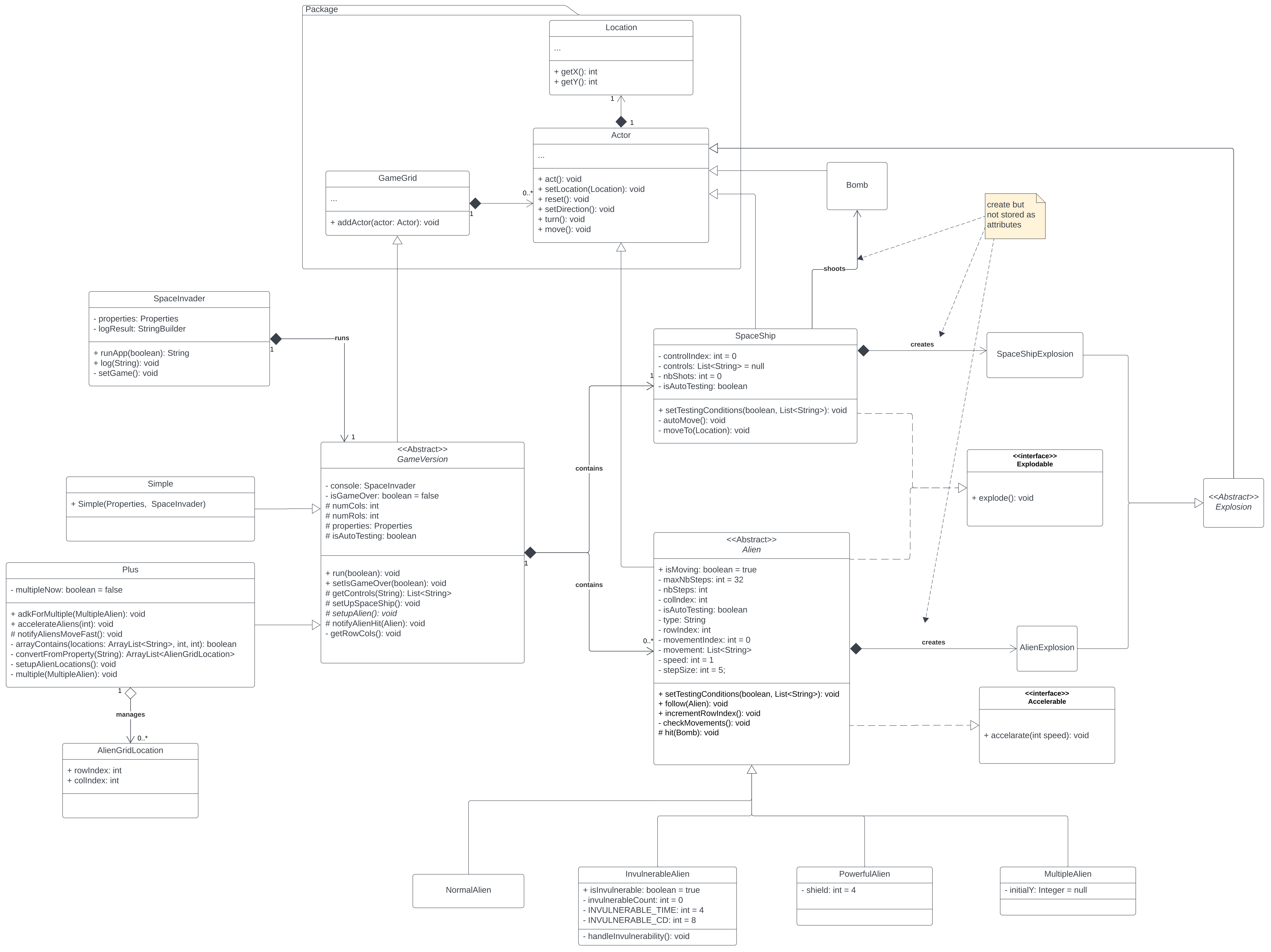


Incorporating the Accelerable interface embodies the essence of the "Pure Fabrication" and "Low Coupling" principles, serving as a modular and standardised contract for game entities that can accelerate. The interface achieves high cohesion by focusing solely on the acceleration mechanism encapsulated in the accelerate() method. It concentrates on a singular responsibility, ensuring that all logic related to this functionality is localised. At the same time, it enforces low coupling by abstracting the accelerate() functionality from the consuming classes, thus allowing for internal changes without affecting dependent code. Additionally, the interface facilitates polymorphism, enabling various implementing classes to provide their unique accelerated versions (). For example, while a Vehicle class might implement accelerate() to increase speed, a DataProcessor class might optimise algorithms to expedite data processing. Additionally, the interface provides an unmatched level of extensibility, enabling the seamless integration of diverse acceleration algorithms, such as LinearAcceleration or ExponentialAcceleration. This flexibility ensures that the game can quickly adapt to changing requirements without necessitating a complete overhaul of the existing codebase. Encapsulating acceleration behaviour via the interface also contributes to "High Cohesion". The Acceleration interface focuses on all acceleration-related features in the game entity. For example, consider a bug that suddenly makes a multiple Alien or other Alien entity accelerate unpredictably during gameplay. Since Alien implements the Accelerable interface, the developer can confidently isolate the issue within how the class overrides the accelerate() method. Lastly, using an interface promotes polymorphism, aligning with GRASP principles by reducing class dependencies so that Game entities can focus on their core responsibilities, fostering a modular and highly maintainable design. Thus, the Accelerable interface serves as a cornerstone for creating a resilient and adaptive game structure, ensuring that the design is in harmony with the core tenets of GRASP.

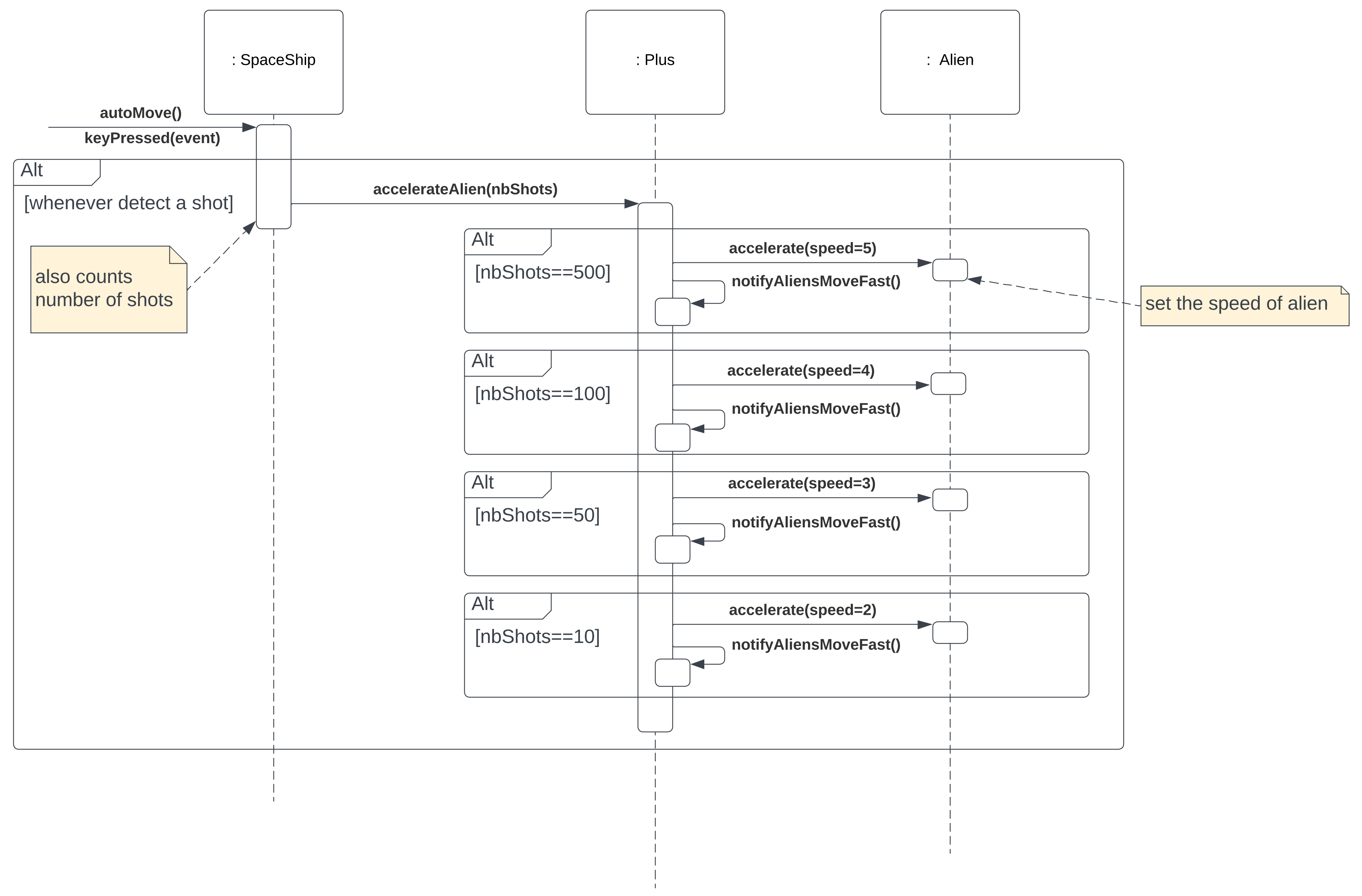


Overall, we delegated the game operation responsibility from the SpaceInvader class to the GameVersion class, incorporating polymorphic development for the GameVersion class to achieve high cohesions for those classes. Furthermore, we utilised two interfaces, Explodable and Accelerable, to achieve exploding and accelerating behaviours. We also changed alien and explosion classes to abstract and created child classes to showcase different types of aliens and explosions. The usage of polymorphisms in interface and abstract classes enhances the level of extensibility and achieves lower decoupling compared to the original design.

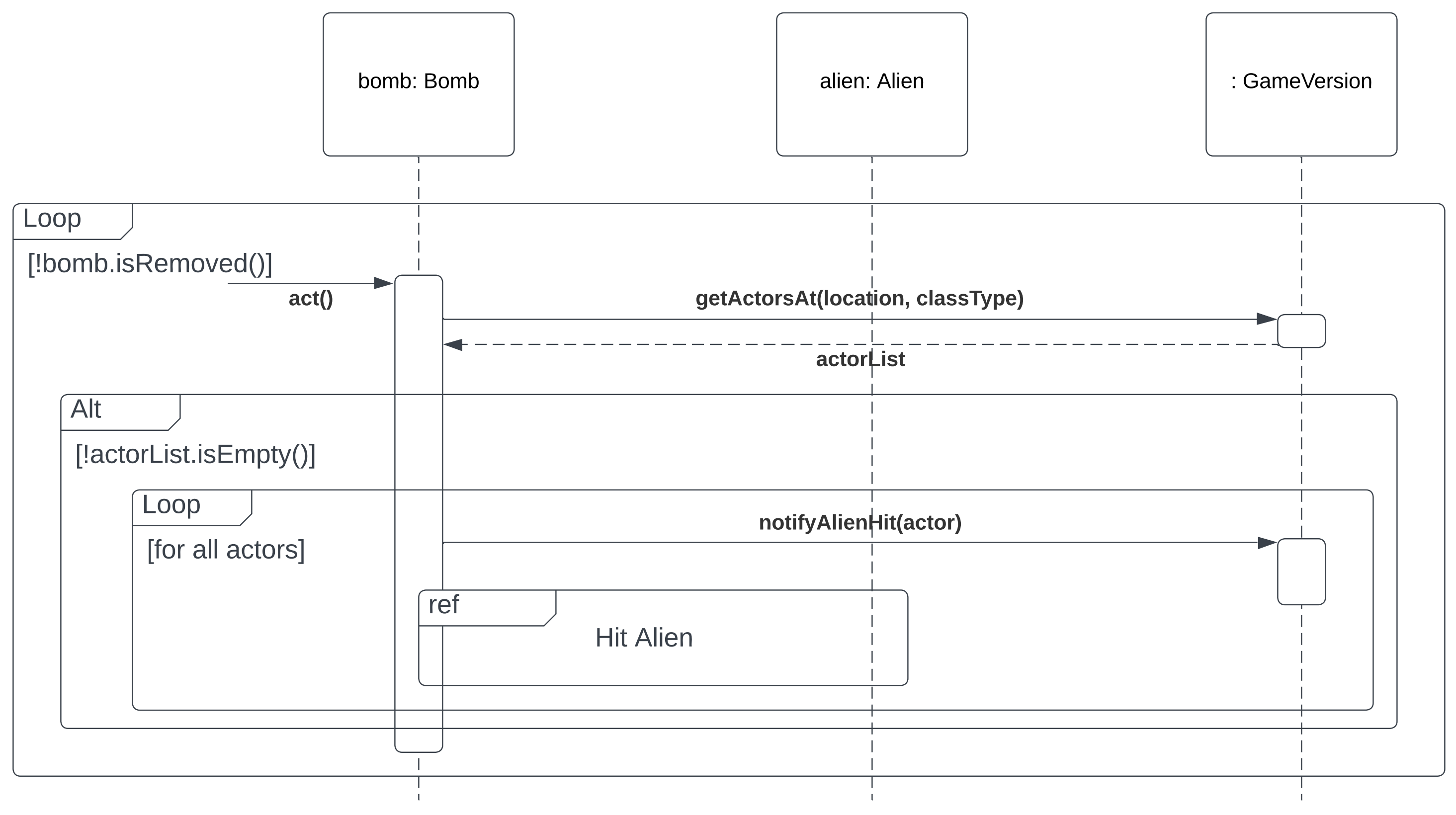

Regular Alien
Invulnerable Alien
isInvulnerable
invulnerableTime
invulnerableCD
Alien
isMoving
maxNbSteps
nbSteps
colIndex
rowIndex
isAutoTesting
type
movementIndex
movements
Space Invader
numCols
numRols
properties
isGameOver
isAutoTesting
gameVersion
contains>
1
0..*
Powerful Alien
shield
Multiple Alien
creates>
AlienExplosion
Explosion
Space Ship
nbShots
isAutoTesting
control
controlIndex
creates>
SpaceShipExplosion
contains>
1
1
shoots>
Bomb
Package
Actor
GameGrid

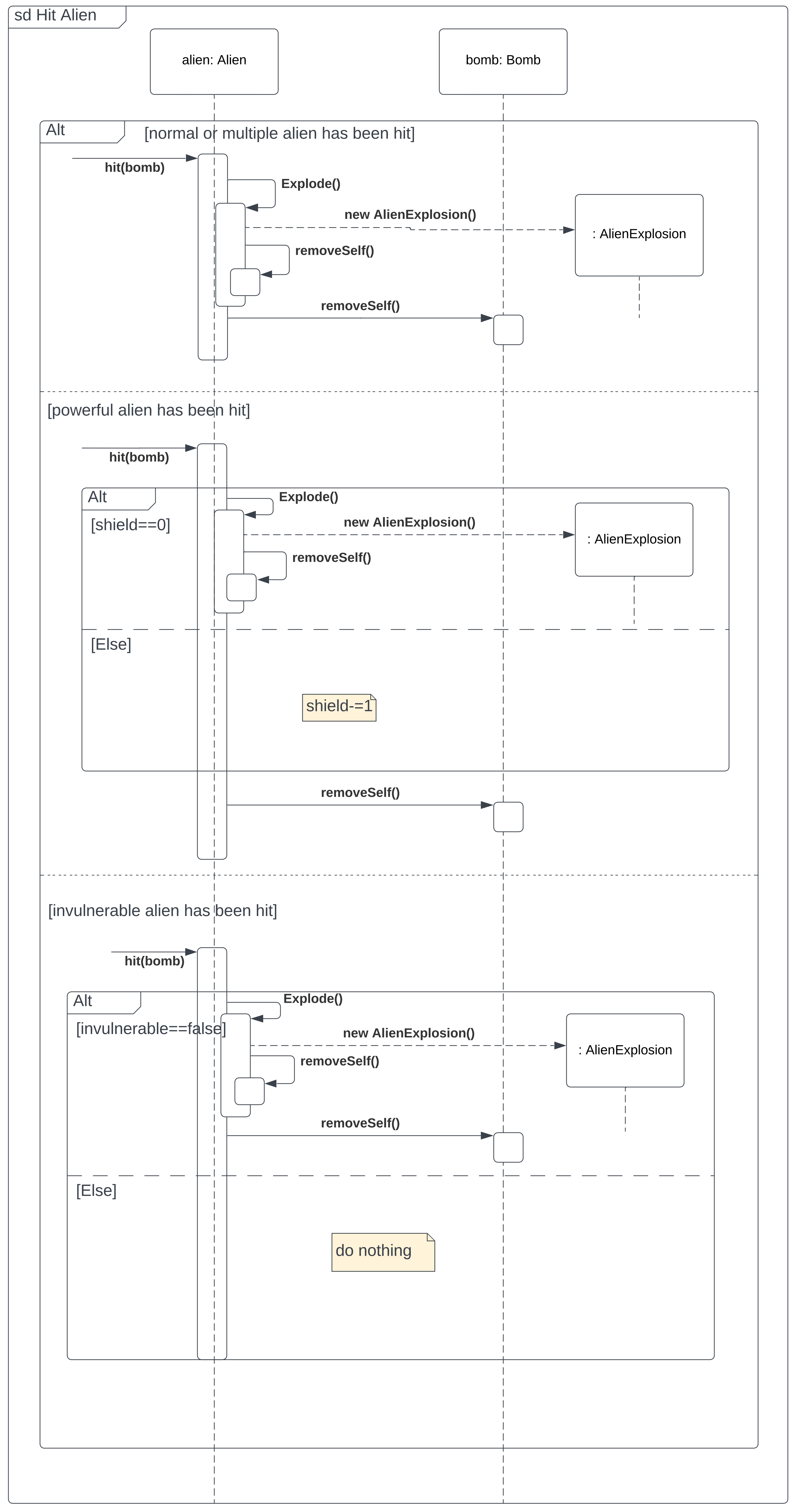

Package
Location
...
+ getX(): int
+ getY(): int
1
1
Actor
...
+ act(): void
+ setLocation(Location): void
+ reset(): void
+ setDirection(): void
+ turn(): void
+ move(): void
GameGrid
...
+ addActor(actor: Actor): void
1
0..*
Bomb
create but not stored as attributes
shoots
SpaceInvader
- properties: Properties
- logResult: StringBuilder
+ runApp(boolean): String
+ log(String): void
- setGame(): void
runs
1
1
SpaceShip
- controlIndex: int = 0
- controls: List<String> = null
- nbShots: int = 0
- isAutoTesting: boolean
+ setTestingConditions(boolean, List<String>): void
- autoMove(): void
- moveTo(Location): void
creates
SpaceShipExplosion
1
<<Abstract>>
GameVersion
- console: SpaceInvader
- isGameOver: boolean = false
# numCols: int
# numRols: int
# properties: Properties
# isAutoTesting: boolean
+ run(boolean): void
+ setIsGameOver(boolean): void
# getControls(String): List<String>
# setUpSpaceShip(): void
# setupAlien(): void
# notifyAlienHit(Alien): void
- getRowCols(): void
contains
contains
1
0..*
Simple
+ Simple(Properties, SpaceInvader)
<<interface>>
Explodable
+ explode(): void
<<Abstract>>
Explosion
Plus
- multipleNow: boolean = false
+ adkForMultiple(MultipleAlien): void
+ accelerateAliens(int): void
# notifyAliensMoveFast(): void
- arrayContains(locations: ArrayList<String>, int, int): boolean
- convertFromProperty(String): ArrayList<AlienGridLocation>
- setupAlienLocations(): void
- multiple(MultipleAlien): void
1
manages
0..*
AlienGridLocation
+ rowIndex: int
+ colIndex: int
<<Abstract>>
Alien
+ isMoving: boolean = true
- maxNbSteps: int = 32
- nbSteps: int
- colIndex: int
- isAutoTesting: boolean
- type: String
- rowIndex: int
- movementIndex: int = 0
- movement: List<String>
- speed: int = 1
- stepSize: int = 5;
+ setTestingConditions(boolean, List<String>): void
+ follow(Alien): void
+ incrementRowIndex(): void
- checkMovements(): void
# hit(Bomb): void
creates
AlienExplosion
<<interface>>
Accelerable
+ accelarate(int speed): void
NormalAlien
InvulnerableAlien
+ isInvulnerable: boolean = true
- invulnerableCount: int = 0
- INVULNERABLE_TIME: int = 4
- INVULNERABLE_CD: int = 8
- handleInvulnerability(): void
PowerfulAlien
- shield: int = 4
MultipleAlien
- initialY: Integer = null

Accelerating alien status

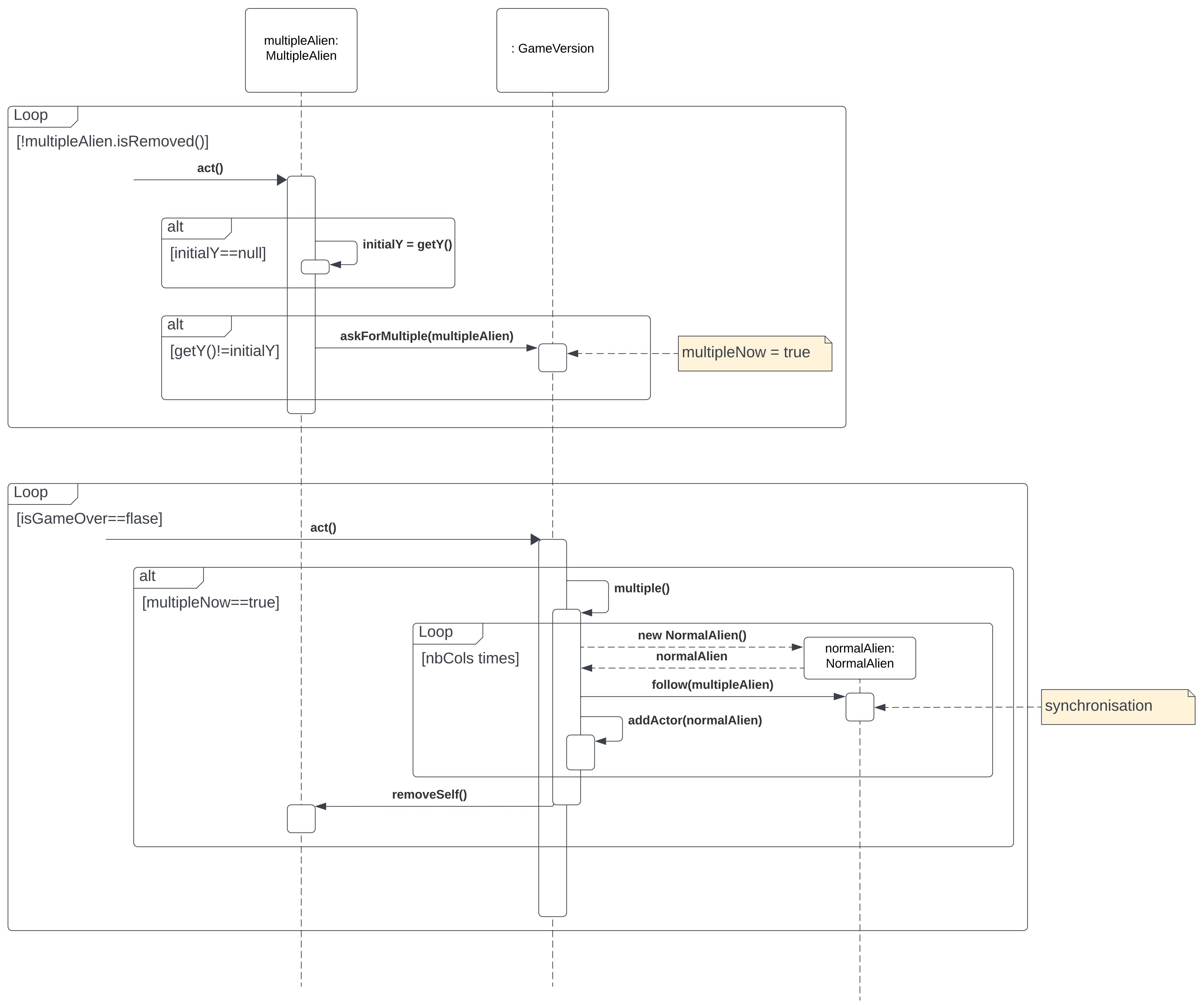


Alien has been hit status





Multiple alien functioning status


multipleAlien: MultipleAlien
: GameVersion
Loop
[!multipleAlien.isRemoved()]
act()
alt
[initialY==null]
initialY = getY()
alt
[getY()!=initialY]
askForMultiple(multipleAlien)
multipleNow = true
Loop
[isGameOver==flase]
act()
alt
[multipleNow==true]
multiple()
Loop
[nbCols times]
new NormalAlien()
normalAlien: NormalAlien
normalAlien
follow(multipleAlien)
addActor(normalAlien)
synchronisation
removeSelf()