SWEN30006 Project 1 Report

Workshop: Thursday 2.15pm
Team 4: Maheen Abdul Khaliq Khan (1193813), Mahamithra Sivagnanam (1225270), Alleena Haider Waseem (1204035)

Introduction

This report provides a comprehensive analysis of the design changes we made to the skeleton code and includes a detailed analysis of the existing and new design. This report aims to identify the design decisions originally made in the code based on GRASP principles, as well as aims to justify the changes we incorporated in alignment with design principles. It consists of three main sections:

- ***Analysis of Current Design (P1)***: This section assesses the existing design based on GRASP principles, highlighting concerns that we felt may hinder the extension of new features.

- ***Proposed New Design of the Simple Version (P2)***: This section describes and justifies the refactoring of the simple version to achieve a better design and facilitate future extensions.

- ***Proposed Design of the Extended Version (P3)***: This section outlines the design for the new features added to the game, discussing the rationale behind design choices and how they contribute to an enhanced architecture.

Section 1: Analysis of Current Design (P1)

1.1 Concerns with the Current Design:

1.1.1 Unnecessary Dependencies
The original design exhibited a significant issue with unnecessary dependencies between classes. Notably, the existence of the `AlienGridLocation` class seemed redundant and could have hindered future extensions by introducing interdependence between classes.

1.1.2 Lack of Polymorphism
Polymorphism is a fundamental principle, it is not initially evident in the code provided, but we later introduced it. We felt that the initial absence of polymorphism would complicate the efficient and maintainable handling of different alien types. This is because all types of aliens share quite a few methods, attributes, and behaviors in common, and the lack of polymorphism behavior in the code would have made handling them a lot more difficult.

1.1.3 High Cohesion and Low Coupling
The `Alien` and `SpaceShip` classes maintain high cohesion, whilst the `SpaceInvader` class demonstrates lower cohesion due to its multifaceted responsibilities. The coupling between `Alien` and `SpaceShip` classes is low, but the `SpaceInvader` class tightly couples with both `Alien` and `SpaceShip` classes, potentially affecting maintainability.

1.1.4 AlienGridLocation Class
The purpose and necessity of the `AlienGridLocation` class is questionable, as it appears to serve as pure fabrication, solely to represent alien locations on the grid.

1.2 Positives of the Current Design:

The `SpaceInvader` class takes on multiple responsibilities, including managing aliens, the spaceship, and game state, which raises concerns about its high cohesion. This class is a great example of a Controller as the `SpaceInvader` class controls the flow of actions and decisions within the game.

Section 2: Proposed New Design of the Simple Version (P2)

2.1 Refactoring for Better Design

2.1.1 Relocating Alien Grid Location Management
In our effort to enhance cohesion and encapsulation, we suggest relocating the management of `alienGridLocation` directly into the `Alien` class itself. This move not only simplifies the handling of future expansions related to aliens but also aligns with the fundamental principle of maintaining entity-specific functionality within their respective classes.

Our rationale for this decision stems from the goal of strengthening the overall cohesion of the `Alien` class. We believe that consolidating all elements related to aliens within this class is essential. Leaving `alienGridLocation` as a separate class could lead to increased dependencies in future extensions, potentially causing complications. Additionally, maintaining separate `Alien` and `alienGridLocation` classes will result in a high degree of coupling between them. This tight coupling means that changes made to one class could negatively impact the other, which we aim to avoid.

2.1.2 Enhanced Controller Responsibilities
While the current code already assigns significant responsibilities to the `SpaceInvader` class, including control and management across multiple other classes, we believe that further enhancing this class's role in managing the game and its elements would be beneficial for future extensions.

2.2 Elements we did not refactor: Existing Indirection

The `SpaceInvader` class's existing role in indirection was well set up, in that it was able to manage various game elements and settings, as well as establish a clear separation between game behavior and configuration parameters. This separation enables effortless adjustments to game settings without direct code modifications. An example of this is how the `SpaceInvader` class sets up the game by accessing the properties file as well as establishing/managing the other elements of the game.

2.3 Future areas for improvement

2.3.1 Reduce the coupling between SpaceInvader and SpaceShip
The initial code we received illustrates a tightly coupled relationship between the `SpaceInvader` class and the `Spaceship` class. In this arrangement, the `Spaceship` class frequently relies on the `SpaceInvader` class to make changes to the game's visual elements and even handle end-of-game outcomes, whether it's a victory or loss scenario. We acknowledge that this design presents certain shortcomings, and regrettably, due to time constraints, we couldn't address them fully.

We consider this design problematic because these two classes should ideally maintain a looser coupling, given their distinct purposes and responsibilities. The high degree of interdependence between them could potentially lead to issues when expanding the game, such as adding new levels. Managing screen displays exclusively within the `SpaceShip` class may pose challenges, and we recognize the need for a more modular and organized approach in handling such aspects.

2.3.2 Reduce the coupling between SpaceInvader and Bomb, increase Bomb’s cohesion

The original skeleton code also exhibits a tightly coupled relationship between the `SpaceInvader` class and the `Bomb` class. Regrettably, due to time constraints, we were unable to address this issue. The `Bomb` class's lack of cohesion resulted in it handling certain tasks that weren't ideally within its scope, such as removing dead aliens from the game grid. The high level of coupling between these classes has the potential to complicate future game extensions. Furthermore, the lack of cohesion within the `Bomb` class has led to suboptimal code design, making it challenging to comprehend, modify, or reuse. This lack of a clear, singular purpose within the `Bomb` class hampers code quality and overall maintainability.

Section 3: Proposed Design of the Extended Version (P3)

3.1 Design for New Features

3.1.1: Extending the Alien Class

In the extended version, we introduced child classes to the `Alien` class (as shown in **Fig.1, Fig.2)**, in order to represent the distinct alien types. This approach promotes code reusability and maintainability by encapsulating common functionalities in the parent `Alien` class and allowing each child class to focus on its unique behaviors.

3.1.2: Alien Grid Location Continuity

We maintained the practice of managing the alien grid location within the `Alien` class (as shown in **Fig.2)**, emphasizing cohesion and ensuring that all alien-related components are consolidated within a single class. We felt that having `alienGridLocation` as a separate class was redundant and hence included it in the constructor of the `Alien` class.

3.1.3: Expanding SpaceInvader's Role

The `SpaceInvader` class serves as a central hub for orchestrating various aspects of the game, adhering to the Controller Principle, managing the flow of game behavior, and ensuring each class within the system has a distinct role (as shown in **Fig.1, Fig.2)**.

3.1.4: Dynamic Gameplay Element

We introduced a dynamic element to the game by enabling the `Spaceship` class to track the number of shots it makes (as shown in **Fig.1, Fig.2)**, triggering alterations in the movement patterns of all aliens on the grid upon reaching a predefined threshold. This is so that we can increase the speed of the aliens in the "plus" version of the game. The way we want/ expect this element to work is shown in **Fig.3.**

3.1.5: Alien Life Counts

We assigned specific life counts to each alien type (as shown in **Fig.1)**, prompting players to consider their targeting priorities based on the life values of the aliens they encounter. The way we want/ expect this feature to work is shown in **Fig.3.**

3.1.6: Polymorphism Implementation

Polymorphism is implemented by overriding the act method in the child alien classes, allowing each alien type to exhibit unique behaviors whilst benefiting from the shared methods and functionality encapsulated by the parent `Alien` class (as shown in **Fig.1, Fig.2).**

3.1.7: Dedicated Child Classes for Alien Types

Dedicated child classes, such as `multiplyAlien`, `powerfulAlien`, and `invulnerableAlien`, are introduced to maintain code organization while encapsulating the distinct characteristics and behaviors of each alien type (as shown in **Fig.1, Fig.2)**. The distinct behaviors for each of the child classes is shown in **Fig.3.**

Conclusion

In this comprehensive design analysis, we systematically evaluated the original and new project designs, applying GRASP principles to enhance adaptability, cohesion, and maintainability. Our analysis identified key areas for improvement, which we addressed in our proposed design. We optimized class dependencies, introduced polymorphism, and restructured classes for a more elegant and organized system (as shown in **Fig.1, Fig.2, Fig.3)**.

In the simple version, we improved cohesion by consolidating alienGridLocation management within the Alien class. In the extended version, we expanded functionality, introducing distinct alien types and adhering to the Controller Principle in the SpaceInvader class. We also added dynamic gameplay elements like shot tracking and alien life counts for strategic depth.

Our emphasis on GRASP principles led to implementing polymorphism and dedicated child classes for extended alien types, promoting code reusability and adaptability. In summary, our redesign efforts have created a more flexible, cohesive, and maintainable codebase, ensuring the project's long-term success.

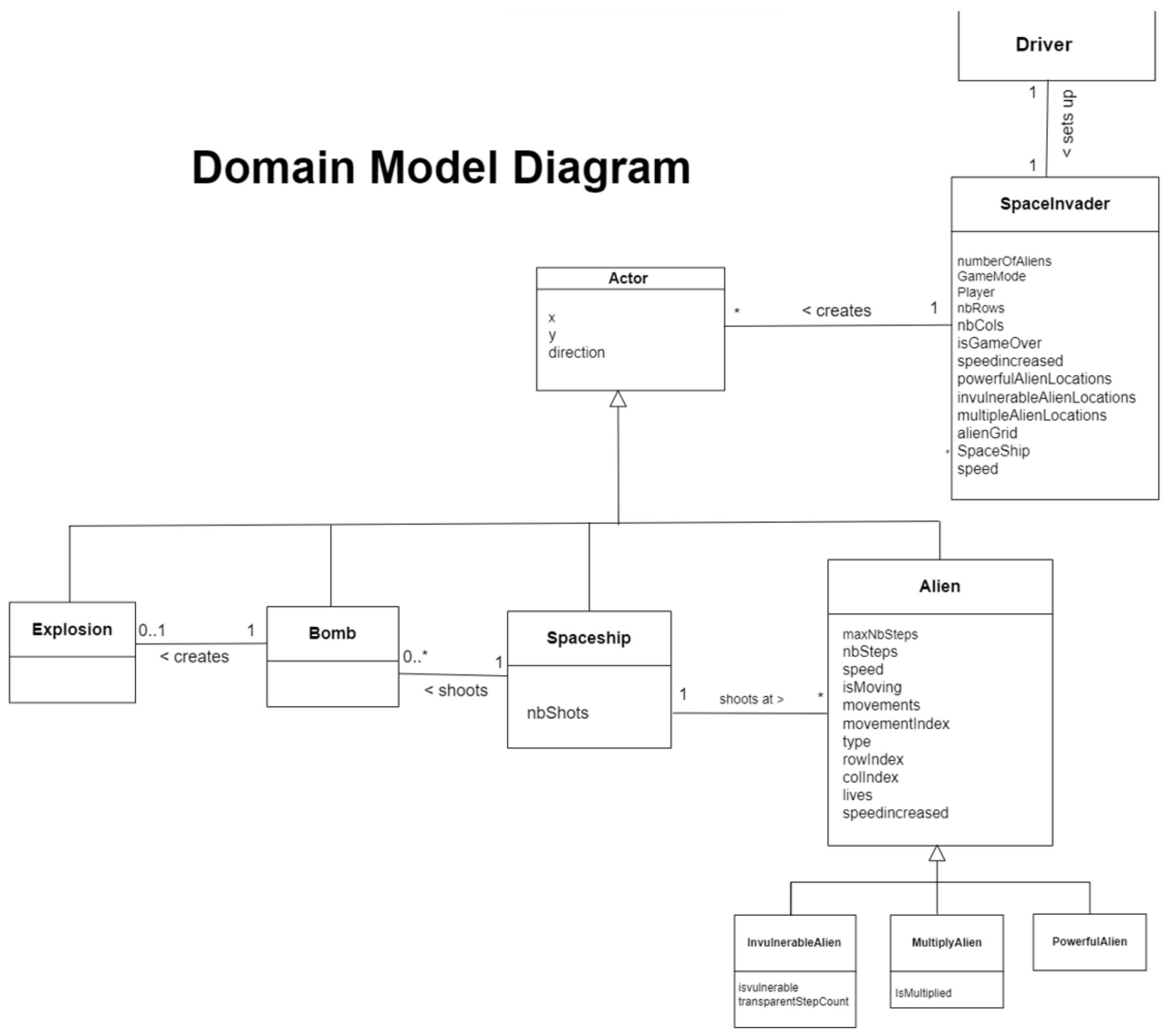


Figure 1: Domain Class Diagram capturing the noteworthy aspects of the game and its extensions.

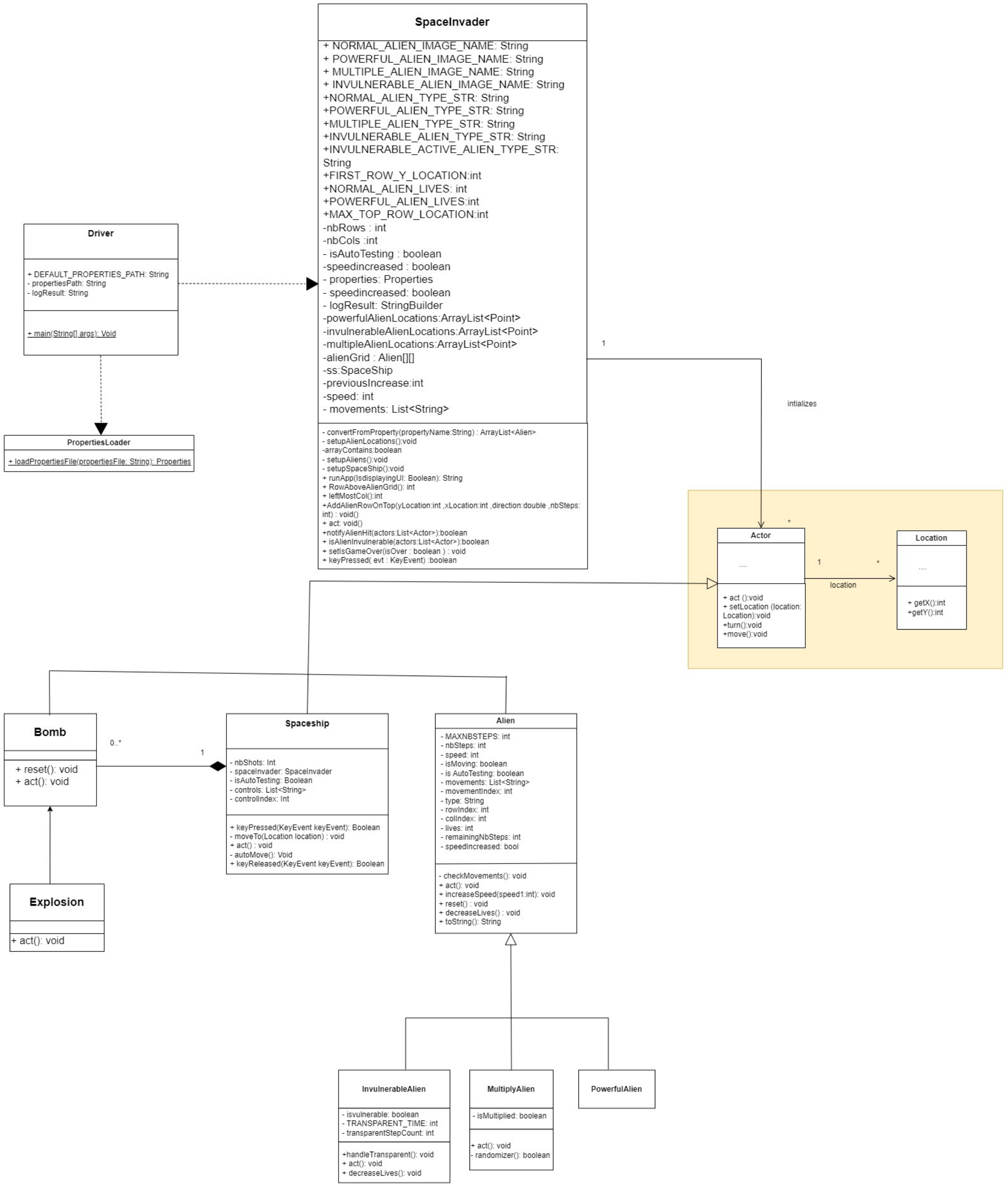
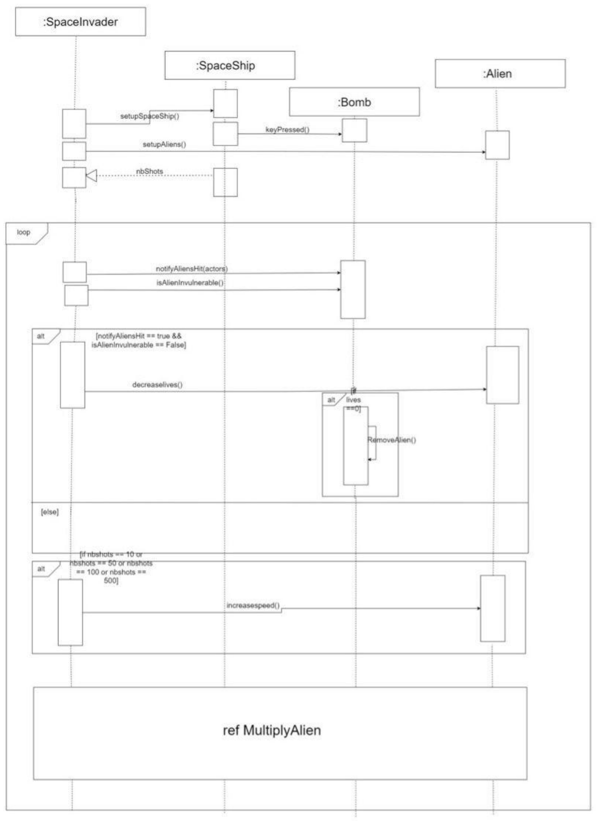

SpaceInvader
+ NORMAL_ALIEN_IMAGE_NAME: String
+ POWERFUL_ALIEN_IMAGE_NAME: String
+ MULTIPLE_ALIEN_IMAGE_NAME: String
+ INVULNERABLE_ALIEN_IMAGE_NAME: String
+NORMAL_ALIEN_TYPE_STR: String
+POWERFUL_ALIEN_TYPE_STR: String
+MULTIPLE_ALIEN_TYPE_STR: String
+INVULNERABLE_ALIEN_TYPE_STR: String
+INVULNERABLE_ACTIVE_ALIEN_TYPE_STR: String
+FIRST_ROW_Y_LOCATION:int
+NORMAL_ALIEN_LIVES: int
+POWERFUL_ALIEN_LIVES:int
+MAX_TOP_ROW_LOCATION:int
-nbRows : int
-nbCols :int
- isAutoTesting : boolean
-speedincreased : boolean
- properties: Properties
- speedincreased: boolean
- logResult: StringBuilder
-powerfulAlienLocations:ArrayList<Point>
-invulnerableAlienLocations:ArrayList<Point>
-multipleAlienLocations:ArrayList<Point>
-alienGrid : Alien[][]
-ss:SpaceShip
-previousIncrease:int
-speed: int
- movements: List<String>
- convertFromProperty(propertyName:String) : ArrayList<Alien>
- setupAlienLocations():void
-arrayContains:boolean
- setupAliens():void
- setupSpaceShip():void
+ runApp(IsdisplayingUI: Boolean): String
+ RowAboveAlienGrid(): int
+ leftMostCol():int
+AddAlienRowOnTop(yLocation:int ,xLocation:int ,direction:double ,nbSteps: int) : void()
+ act: void()
+notifyAlienHit(actors:List<Actor>):boolean
+ isAlienInvulnerable(actors:List<Actor>):boolean
+ setIsGameOver(isOver : boolean) : void
+ keyPressed(evt : KeyEvent) :boolean
Driver
+ DEFAULT_PROPERTIES_PATH: String
- propertiesPath: String
- logResult: String
+ main(String[] args): Void
PropertiesLoader
+ loadPropertiesFile(propertiesFile: String): Properties
1
intializes
*
Actor
....
+ act ():void
+ setLocation (location: Location):void
+turn():void
+move():void
1
*
location
Location
....
+ getX():int
+getY():int
Bomb
+ reset(): void
+ act(): void
0..*
1
Spaceship
- nbShots: Int
- spaceInvader: SpaceInvader
- isAutoTesting: Boolean
- controls: List<String>
- controlIndex: Int
+ keyPressed(KeyEvent keyEvent): Boolean
- moveTo(Location location) : void
+ act() : void
- autoMove(): Void
+ keyReleased(KeyEvent keyEvent): Boolean
Alien
- MAXNBSTEPS: int
- nbSteps: int
- speed: int
- isMoving: boolean
- is AutoTesting: boolean
- movements: List<String>
- movementIndex: int
- type: String
- rowIndex: int
- colIndex: int
- lives: int
- remainingNbSteps: int
- speedIncreased: bool
- checkMovements(): void
+ act(): void
+ increaseSpeed(speed1:int): void
+ reset() : void
+ decreaseLives() : void
+ toString(): String
Explosion
+ act(): void
InvulnerableAlien
- isvulnerable: boolean
- TRANSPARENT_TIME: int
- transparentStepCount: int
+handleTransparent(): void
+ act(): void
+ decreaseLives(): void
MultiplyAlien
- isMultiplied: boolean
+ act(): void
- randomizer(): boolean
PowerfulAlien

Figure 2: Design class diagram documenting the new design for Space Invaders Plus including extensions.

Sequence Diagram



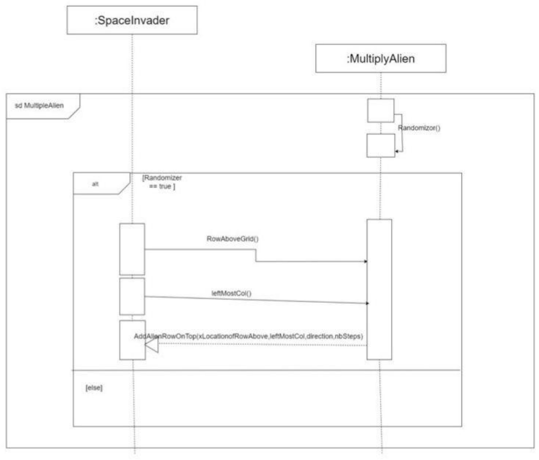


Figure 3: Dynamic design model documenting the dynamic behavior of aliens in terms of their different statuses regarding effects of the game stage and the number of shots. The left hand shows the sequence of events for normal aliens, Invulnerable, and powerful aliens, while the right hand side shows the extension to this that is used for aliens that multiply.