Assignment 1 Report: Space Invaders Plus, ABC Game™

SWEN30006 SEMESTER 2 2023

Evan Liapakis, Toby Guan, Claire Tosolini

September 8, 2023

Contents

1 Introduction

The objective of this report is to outline the process undertaken to develop the new version of *Space Invaders*, ***Space Invaders Plus***. This involved a full analysis of the requirements as outlined by ABC Games™, as well as identifying the limitations of the existing implementation. Static and dynamic models were created to clearly outline our proposed implementation.

2 Requirements Analysis

Our initial focus was to ensure we had a thorough understanding of the business requirements. We constructed a partial domain model of our own (Figure 1), from both the main success scenario of the Use Case provided, and the proposed extensions. This was constructed with the intent of providing us a baseline point of comparison with any existing documentation and infrastructure, as well as direction for our implementation.

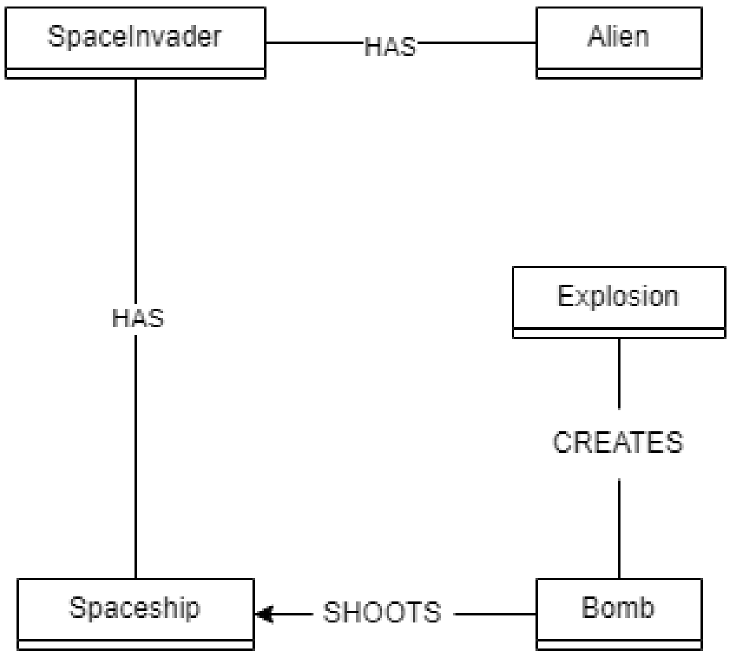

Figure 1: Primary Draft Domain Model

3 Current Implementation Analysis

An analysis, with respect to GRASP principles, of the simple version, *Space Invaders*, was undertaken. Upon careful examination of the given partial Class Diagram, a number of principle violations were discovered and outlined below.

3.1 The *Alien* class

In the current implementation, all aliens were crowded into a single *Alien* class, with differentiation achieved through assigning a String 'type' attribute. This design violates the GRASP principle of **polymorphism**, and would make extending the game more complex, as it would promote heavy code duplication in applying different *Alien* logic, as well as require significant conditional logic based on testing object types. These points are undesirable in a software system in general, but particularly so given the purpose of the new implementation is to *extend* the existing one, as **Polymorphism** involves creating conditional behavioural variation in an extensible way.

3.2 The *SpaceInvader* class

Another issue we observed related to the interaction between the classes *SpaceInvader* and *Explosion.* The responsibility of creating instances of the latter was assigned to the former, clearly violating the GRASP principle of the **Creator Pattern**. According to this principle, SpaceInvader is not the best choice for this responsibility as it does not aggregate, record, closely use, or contain the initialising data for an instance of *Explosion.*

Furthermore, this design also violates the GRASP principle of **High Cohesion**, as it involves assigning further, unrelated responsibility to a class that is already quite complex. Too much responsibility assigned will result in a class that is less comprehensible and more difficult to maintain.

4 Proposed Implementation: Simple Version

Given the significant existing code base, we aimed to implement with minimal adjustments, providing our improvements whilst preserving functionality and utilising infrastructure. Refactoring the simple version involved changes directly influenced by the issues outlined in Section 3 above. We determined that since an Explosion only occurs when a Bomb collides with an Alien, Bomb closely uses Explosion. Assigning the responsibility of instantiating *Explosion* objects to the *Bomb* class therefore satisfied the **Creator Pattern** and promoted **High Cohesion** in our design.

In order to produce a more **Polymorphic** and therefore extensible solution, we made the *Alien* class abstract, then created the concrete subclass *Normal Alien* extending from it.

5 Proposed Implementation: Extended Version

The proposed design of the extended version followed on from the changes we had been making. Since the abstract *Alien* class had already been created, implementing the new alien types was much more straightforward. Using the principle of **Polymorphism**, the new types were effectuated through extending the abstract class, creating three additional concrete subclasses.

6 Domain Model Creation

Referring to our own domain model and the partial class diagram provided, we constructed a Domain Model (Figure 2) to accurately reflect the proposed implementation as per the changes outlined in sections 4 and 5.

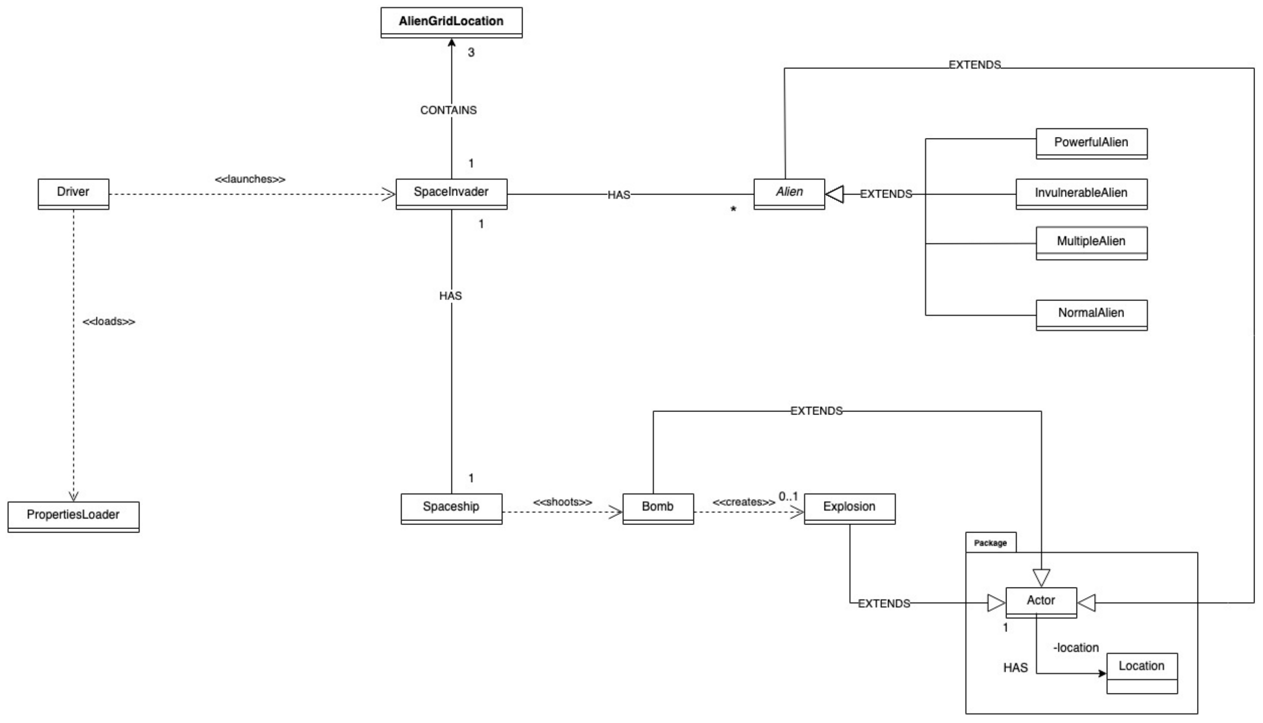

Figure 2: Domain Model

7 Design Class Diagram Creation

The Design Class Diagram below provides a more detailed description of our proposed implementations outlined above. Justifications for specific methods not already mentioned can be found in section 8.

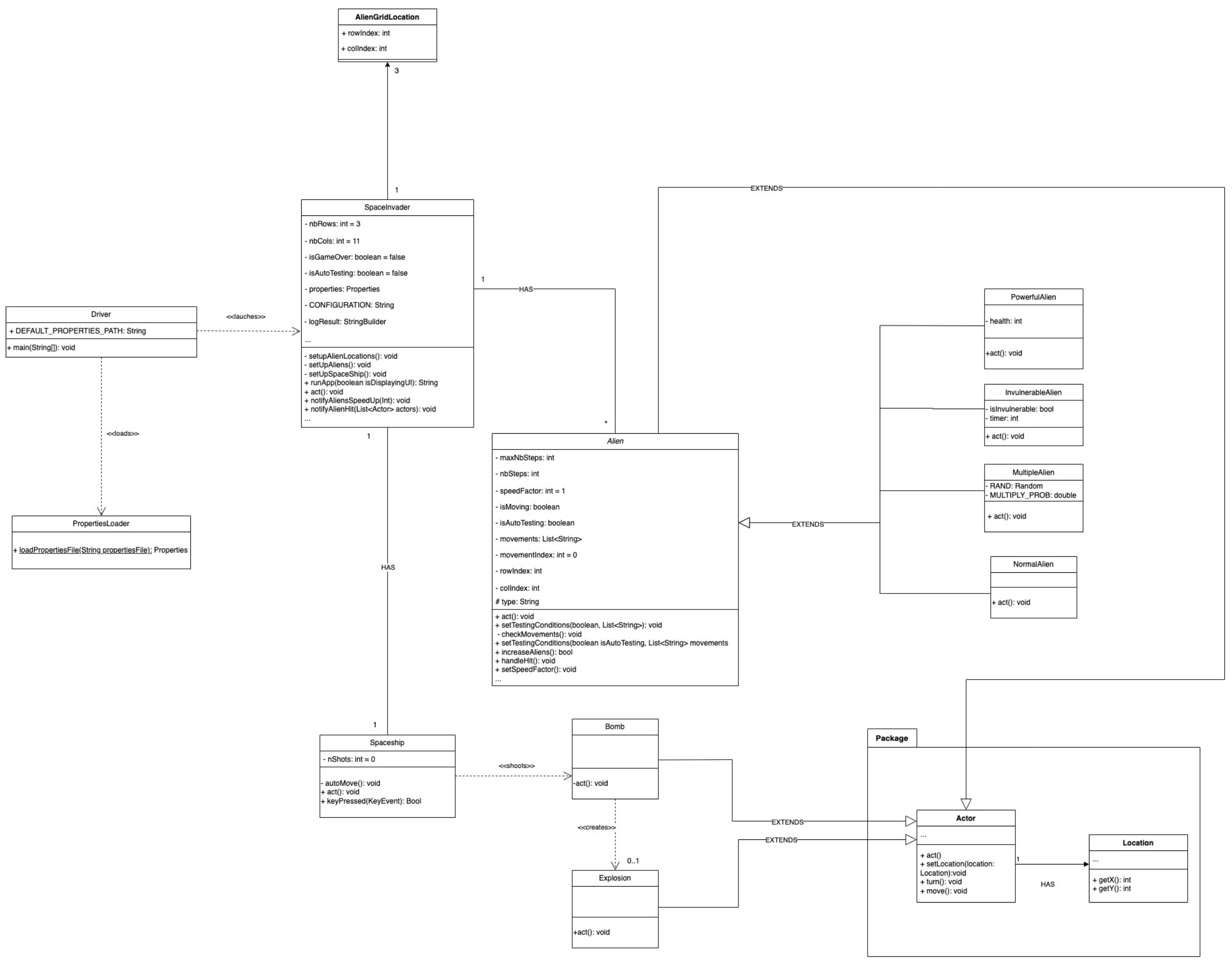

Figure 3: Design Class Diagram

8 Design Sequence Diagrams (DSD) Creation

After gaining a clearer understanding by laying out the different classes and familiarising ourselves with the JGameGrid library, we identified the following significant system events, and captured them in design sequence diagrams below

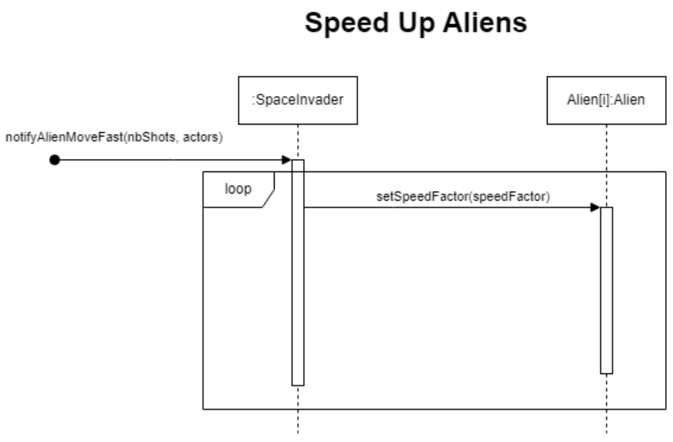
Speed Up Aliens
:SpaceInvader
Alien[i]:Alien
notifyAlienMoveFast(nbShots, actors)
loop
setSpeedFactor(speedFactor)

Figure 4: DSD: Alien Speed Increase

In the extended version, a new feature to increase speed is being introduced. Since the *SpaceInvader* class already holds all the details about every Alien in the game, it makes sense to assign the responsibility of adjusting the speed for each *Alien* on the game grid to the *SpaceInvader* class, based on the principle of **Information Expert**.

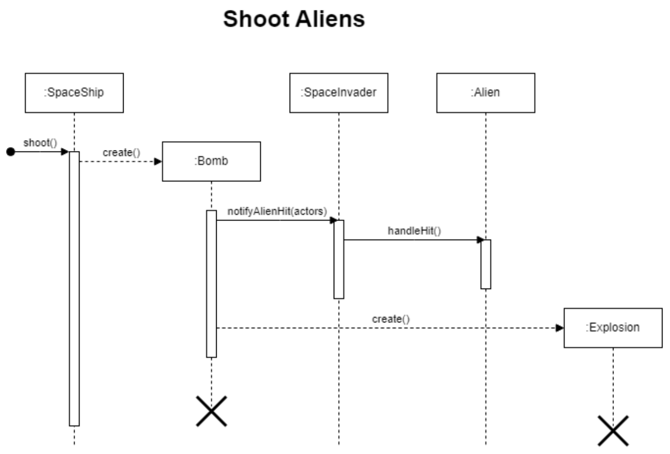

Figure 5: DSD: Alien Collision with Bomb

Different from the original design, we made the shooting Alien process more streamlined and **less coupled**. By reducing interaction between classes, again the pattern of **Information Expert** can be observed as each class is designed to only handle its primary responsibility, as previously discussed in 3.2 and 4.

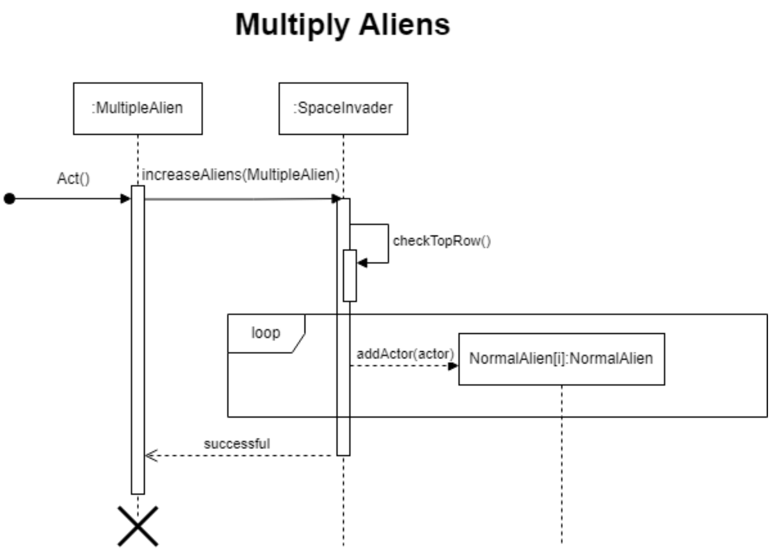

Figure 6: DSD: Alien Multiplying

The behaviour of *MultipleAliens* are designed to be randomly initiated by its own *act()* function, consistent with the proposed new feature. With this in mind, when *SpaceInvader* detects a flag raised by a **MultipleAlien**, it triggers a validation process to determine if new Aliens can be introduced. If conditions permit, Aliens are then spawned. To ensure they are synchronised with the current Aliens in the grid, this design approach resonates with the **Creator** and **Information Expert** pattern from the GRASP principles, whereby the *SpaceInvader* is vested with the responsibility of instantiating new Aliens, fulfilling the feature's objectives.

List of Figures