Contents

1. Introduction:

This report delves into the design and development of the Space Invaders game, which has been transformed to educate primary students about celestial bodies and space exploration. The objective is to provide an engaging platform for learning science concepts related to planets, asteroids, comets, and other celestial phenomena. Through this report, we explore the game's mechanics, functionalities, and software architecture, shedding light on how it facilitates interactive learning experiences for young learners. Furthermore, we examine the integration of design patterns such as the Observer pattern and the Factory pattern, highlighting their roles in structuring the game and enhancing its educational value.

2. Game Overview:

- Description of the Space Invaders game.

Celestial Bodies is an educational arcade-style game that aims to teach primary students about the solar system and celestial phenomena. Players control a spacecraft tasked with protecting Earth from incoming celestial bodies such as planets, asteroids, and comets. The game features dynamic gameplay where players must shoot projectiles to destroy the descending celestial bodies while learning about their characteristics and properties through a quiz.

- Features and Functionalities

The game features several key elements:

1. Player Spaceship:

Players manoeuvre a spacecraft horizontally along the bottom of the screen using keyboard or controller inputs. The spacecraft can shoot projectiles vertically to eliminate descending celestial bodies.2. Celestial Bodies:

The primary objective is to hit descending celestial bodies, each representing a different element of the solar system. Celestial bodies move vertically towards Earth, with increasing speed and complexity as the game progresses.

3. Scoring System:

Players earn points by successfully hitting and destroying celestial bodies while simultaneously answering a quiz question regarding the planets.

4. Educational Content:

Celestial Bodies integrates educational content about each celestial body encountered in the game. Players can learn about the characteristics, composition, and significance of planets, asteroids, comets, and other celestial phenomena.

- Gameplay Mechanics

1. Movement:

Players can move the spacecraft horizontally across the screen to align with incoming celestial bodies using intuitive keyboard or controller controls.

2. Shooting:

The spacecraft can shoot projectiles vertically to intercept and destroy descending celestial bodies. Players must aim accurately to hit their targets and prevent collisions with Earth.

3. Collision Detection:

Collision detection mechanisms determine whether projectiles fired by the spacecraft collide with celestial bodies. Upon collision, the affected celestial body is destroyed, and the player earns points.

4. Game Over:

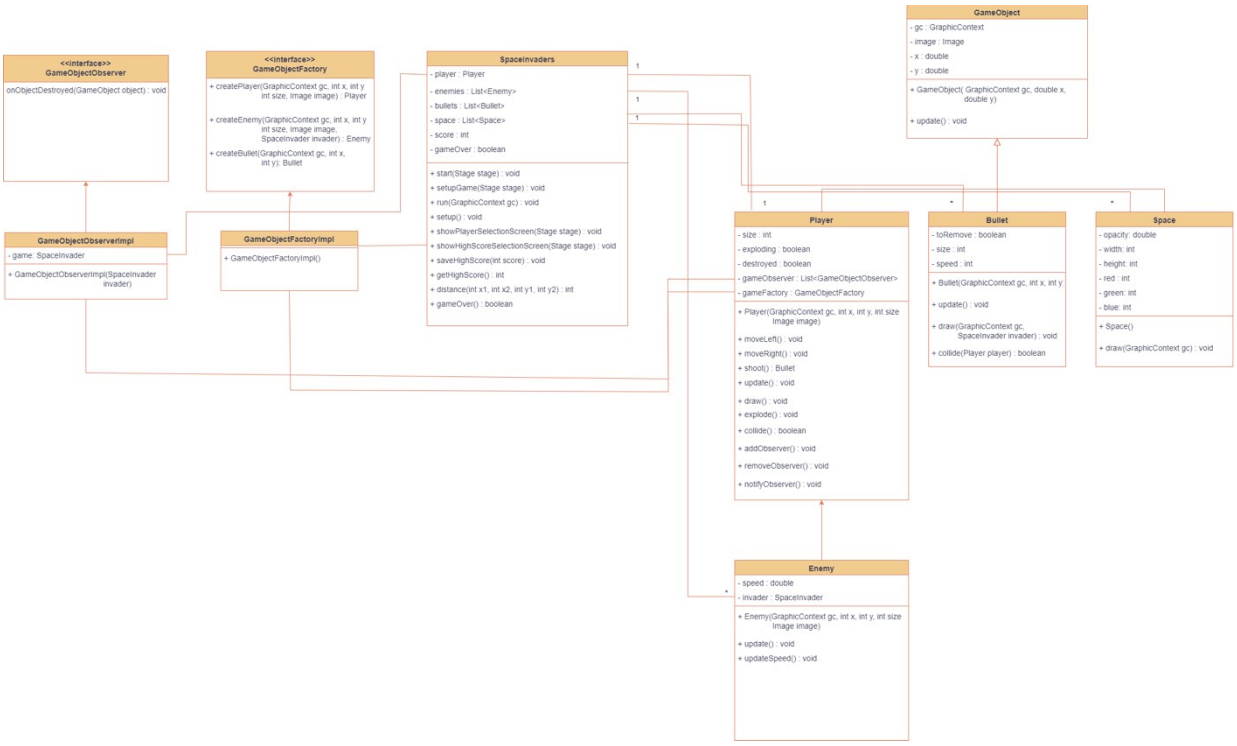
The game ends if a celestial body collides with Earth or if the player fails to prevent the destruction of Earth. A game over screen displays the player's final score and provides options to replay the game or access educational content.

Overall, the game offers an engaging and educational gameplay experience, combining arcade-style mechanics with informative content about the solar system and celestial bodies. Players can enjoy dynamic gameplay while expanding their knowledge of astronomy and space exploration.

3. Software Design:

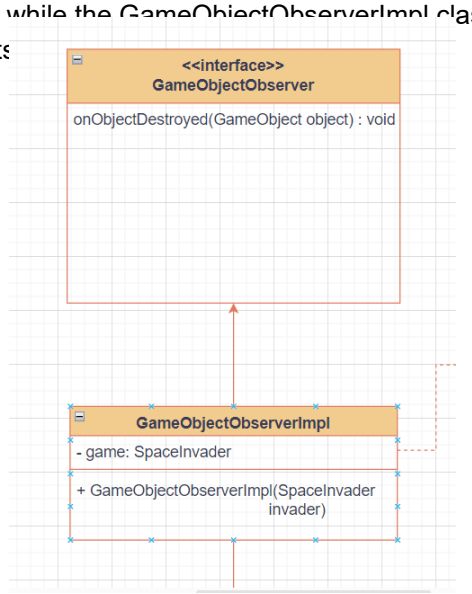
- Class Diagram:

The class diagram provides a visual representation of the structure and relationships within the Celestial Bodies game. It illustrates how various classes such as Player, Enemy, Bullet, Space, and GameObject interact and collaborate to facilitate gameplay.



- Observer Pattern:

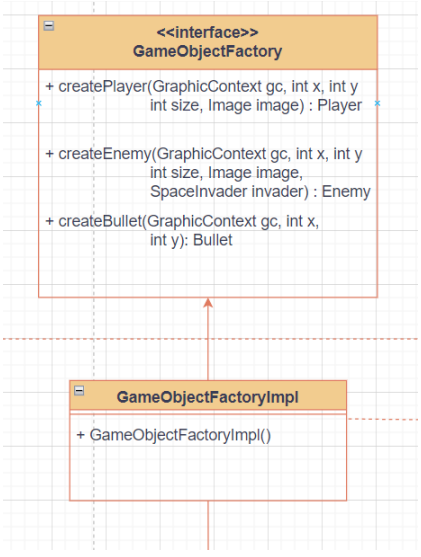
In Celestial Bodies, the Observer pattern is implemented to facilitate communication between GameObjects and GameObjectObservers. This pattern enables GameObjects to notify their observers when specific events occur, such as object destruction or state changes. The GameObjectObserver interface defines methods for observing GameObject events, while the GameObjectObserverImpl class implements these methods to handle object destruction events



Factory Pattern:

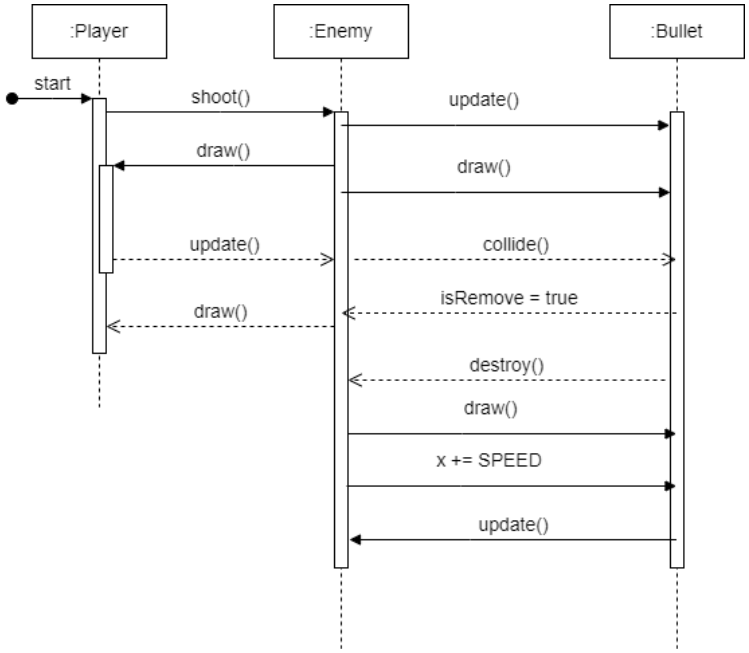
The Factory pattern is employed in Celestial Bodies to create instances of game objects such as Player, Enemy, and Bullet. The GameObjectFactory interface defines factory methods for creating these objects, while the GameObjectFactoryImpl class implements these methods to instantiate concrete objects based on specific parameters. This pattern promotes loose coupling and flexibility in object creation, hiding the instantiation logic from the client code.

By using the Factory pattern, the game achieves flexibility in object creation, allowing for easier maintenance and extension of the codebase. It also enables dependency injection, making the game more testable and adaptable to changes in requirements.



- Sequence Diagram

The sequence diagram illustrates the interaction between Player, Enemy and Bullet during gameplay. When a GameObject, such as an Enemy, is destroyed, it notifies all registered observers by calling their respective **onObjectDestroyed** method. This allows observers, such as GameObjectObserverImpl, to respond accordingly, such as updating the game state or displaying animations.



- Use Case Diagram

The Use Case Diagram illustrates the various interactions between the player (actor) and the Celestial Bodies game system. It outlines the essential actions and functionalities available to the player during gameplay. Here's a description of each component:

Player: The primary actor in the game, representing the user who interacts with the Celestial Bodies system.

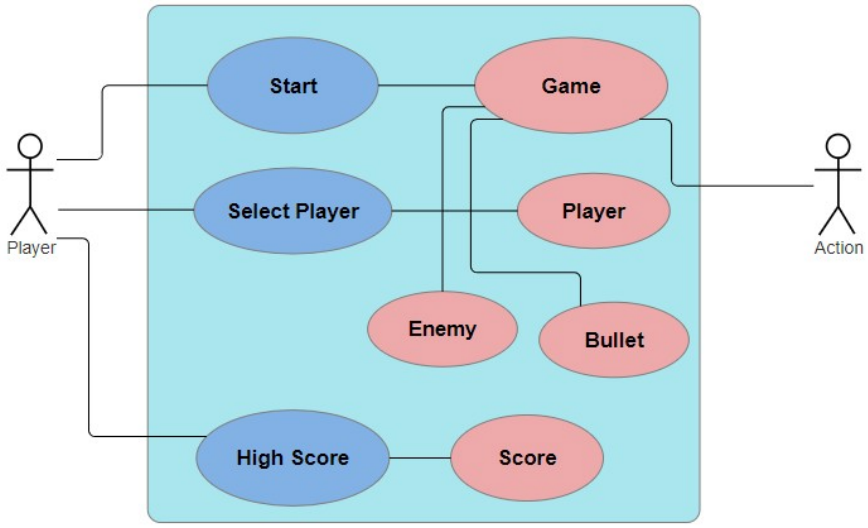
Use Cases:

- Move Spaceship: This use case allows the player to manoeuvre their spaceship horizontally along the bottom of the screen. It enables the player to avoid incoming celestial bodies and position the spaceship strategically.
- Shoot: With this use case, the player can shoot projectiles from their spaceship to destroy celestial bodies. It forms the core mechanic of the game, empowering the player to defend against the incoming threats.
- Destroy Celestial Body: When a celestial body is hit by the player's projectiles, it triggers this use case, resulting in the destruction of the targeted body. It contributes to the player's progression in the game and earns them points.

- View High Score: This use case allows the player to view the highest score achieved in the game. It provides a benchmark for the player's performance and encourages competition to surpass previous records.
- Select Player: Before starting the game, the player can choose their preferred spaceship design using this use case. It offers customization options and allows players to personalize their gaming experience.
- View Celestial Body Information: When a celestial body is hit during gameplay, this use case displays relevant information about the specific body. It serves an educational purpose, providing players with insights into different celestial objects and enhancing their learning experience.

Relationships:

- The player (actor) initiates and interacts with all the use cases, driving the gameplay experience.
- The core actions of moving the spaceship, shooting, and destroying celestial bodies are fundamental to gameplay and are initiated by the player.
- The ability to view the high score allows players to monitor their progress and strive for improvement.
- Selecting a player enables customization and personalization, enhancing player engagement.
- Viewing celestial body information enriches the gaming experience by providing educational content related to the game's theme of celestial bodies and space exploration.



4. Application of Design Patterns:

The Space Invaders game leverages design patterns to enhance its architecture, promote code maintainability, and facilitate extensibility. Specifically, the Observer pattern is employed for managing object interactions, while the Factory pattern is utilized for efficient object creation.

In the Space Invaders game, two design patterns, namely Observer and Factory, are applied to manage object interactions and object creation efficiently.

Observer Pattern:

- Implementation:

In Space Invaders, the Observer pattern facilitates communication between game objects and their observers. GameObjects such as Player, Enemy, and Bullet act as subjects, while the GameObjectObserver serves as an observer. Each GameObject maintains a list of observers and

notifies them when the object is destroyed. The GameObjectObserverImpl class implements the GameObjectObserver interface, defining methods to handle destruction events.

- Benefits:

  - Loose Coupling: The Observer pattern promotes loose coupling between game objects and their observers. This allows for easier maintenance and extensibility of the codebase.

  - Scalability: Dynamic registration and removal of observers enable easy addition of new observer types without modifying existing code.

  - Reusability: By decoupling observing logic from observed objects, the pattern promotes code reuse and modular design.

- Modifications: No significant modifications are made to the Observer pattern for Space Invaders. However, the GameObjectObserverImpl class may include additional logic to handle specific events triggered by object destruction, such as updating the game score or initiating animations.

Factory Pattern:

- Implementation:

The Factory pattern centralizes object creation logic in Space Invaders. The GameObjectFactory interface declares methods for creating Player, Enemy, and Bullet objects. The GameObjectFactoryImpl class provides concrete implementations of these methods, encapsulating the object creation process.

- Benefits:

  - Encapsulation: The Factory pattern encapsulates object creation logic, shielding clients from the details of instantiation.

  - Abstraction: Defining a common interface abstracts the object creation process, facilitating easy switching between implementations and extension with new object types.

  - Testability: The pattern enhances testability by enabling clients to inject mock implementations of the factory interface for isolated testing.

- Modifications: Space Invaders does not require significant modifications to the Factory pattern. However, the GameObjectFactoryImpl class may be extended to support additional object creation functionalities or handle variations in object instantiation based on game conditions.

Overall, the application of the Observer and Factory patterns in Space Invaders enhances the game's maintainability, scalability, and testability, contributing to a more robust and adaptable codebase.

5. Key Elements of the Code:

The core components of the game code include classes for Player, Enemy, Bullet, Space, and GameObject, each responsible for specific game functionalities. The Player class manages player input and movement, while the Enemy class controls enemy behavior and AI. The Bullet class represents projectile objects fired by the player and enemies, while the Space class generates background space elements for visual effect. The GameObject class serves as the base class for all game objects, providing common functionality such as position tracking and rendering.

In the Space Invaders game codebase, several classes play crucial roles in defining the game's functionality and interactions. Here's an overview of some important classes and their functionalities:

1. SpaceInvaders: This class serves as the entry point for the game, managing the game loop, user input, and scene rendering. It orchestrates the creation of game objects, such as Player, Enemy, and Bullet, and handles collision detection and game state updates.

2. Player: The Player class represents the player-controlled spaceship in the game. It handles player movement, shooting mechanics, and collision detection with enemies and bullets. The class also incorporates the Observer pattern to notify observers (such as GameObjectObserverImpl) when the player is destroyed.

3. Enemy: Instances of the Enemy class represent the celestial bodies such as planets, asteroids, and comets that descend upon the player's spaceship. The class manages enemy movement, shooting behavior, and collision detection with the player's ship and bullets. Additionally, it utilizes the Factory pattern through the GameObjectFactoryImpl class to create new enemy instances.

4. Bullet: The Bullet class encapsulates the behavior and attributes of projectiles fired by both the player and enemies. It manages bullet movement, collision detection with other game objects, and rendering on the game canvas. The class incorporates variations in bullet behavior based on game conditions, such as power-ups or enemy difficulty levels.

5. GameObjectFactoryImpl: This class implements the GameObjectFactory interface and serves as a centralized factory for creating instances of game objects, including Player, Enemy, and Bullet. It abstracts the object creation process, promoting code reuse and encapsulation.

6. GameObjectObserverImpl: Instances of this class act as observers for game objects, receiving notifications when observed objects are destroyed. It implements the GameObjectObserver interface and defines the `onObjectDestroyed` method to handle destruction events, such as updating the game score or initiating animations.

6. Conclusion:

In conclusion, the Space Invaders game represents a successful implementation of classic arcade gameplay mechanics using modern Java and JavaFX technologies. Through the application of design patterns such as Observer and Factory, the codebase demonstrates principles of modularity, abstraction, and encapsulation, contributing to a maintainable and extensible game architecture.

- Reflections on the software design process.

The software design process for Space Invaders involved careful consideration of game mechanics, object interactions, and user experience. Iterative development cycles, frequent code reviews, and continuous integration practices were essential in ensuring the quality and stability of the codebase. Moreover, the adoption of design patterns facilitated the implementation of robust and scalable solutions, promoting code reuse and maintainability.

- Future improvements or enhancements for the game.

Looking ahead, several areas for future improvement and enhancement of the game can be identified. These include:

1. Power-Ups and Upgrades: Introducing power-ups and ship upgrades to add depth and variety to gameplay, providing players with additional abilities and challenges.

2. Multiplayer Mode: Implementing multiplayer functionality to allow players to compete or cooperate with each other in real-time, adding a social dimension to the game experience.

3. Level Design and Progression: Designing a curated progression system with distinct levels, enemy waves, and boss fights to provide players with a sense of achievement and challenge as they advance through the game.

By focusing on these areas of improvement and continuing to iterate on the game design and implementation, Space Invaders can evolve into a more engaging and compelling gaming experience for players of all ages.