**Project Documentation: Tower Defense Game in Unity 3D with Design Patterns**

**GitHub:** [**tincenma/TowerDefenseProject**](https://github.com/tincenma/TowerDefenseProject)

**Full Name: Temirlan Ilyasov**

**Group: SE-2312**

**Overview of System Architecture**

The Tower Defense game is structured around a Model-View-Controller (MVC) architecture to ensure modularity, maintainability, and ease of future updates. Each of the core elements of the game—models, views, and controllers—are separated logically and physically, promoting clean interactions between different layers.

* **Models**: Contain all the essential data and logic behind the game entities such as enemies, waves, turrets, and player statistics.
* **Views**: Handle the UI elements that the player interacts with, including the pause menu, win screen, game over screen, and the display of rounds survived.
* **Controllers**: Manage the interaction between the models and views, handling user input and game state updates. Key controllers include the GameManager, WaveSpawner, BuildManager, and state management classes.

The game state is managed through a state machine that allows the system to transition smoothly between different states, such as playing, game over, and win states. Each of these states is represented by a different class, which helps in managing the game flow and user experience.

**Implemented Design Patterns**

The project incorporates multiple design patterns to provide a clean and scalable codebase. Below are the descriptions of the implemented patterns, including their purpose in the project:

**1. Creational Patterns**

* **Singleton**:
  + Used in classes like GameManager, BuildManager, and WaveSpawner to ensure that there is only one instance of each manager throughout the game lifecycle.
  + This pattern helps manage shared resources such as game state, building processes, and wave spawning efficiently.
* **Factory Method**:
  + Implemented in the turret building process within BuildManager. The turret blueprint and creation process are abstracted away, allowing for different types of turrets to be built without modifying the core logic of the game.
  + This pattern allows the easy addition of new turret types without requiring extensive changes to the existing code.

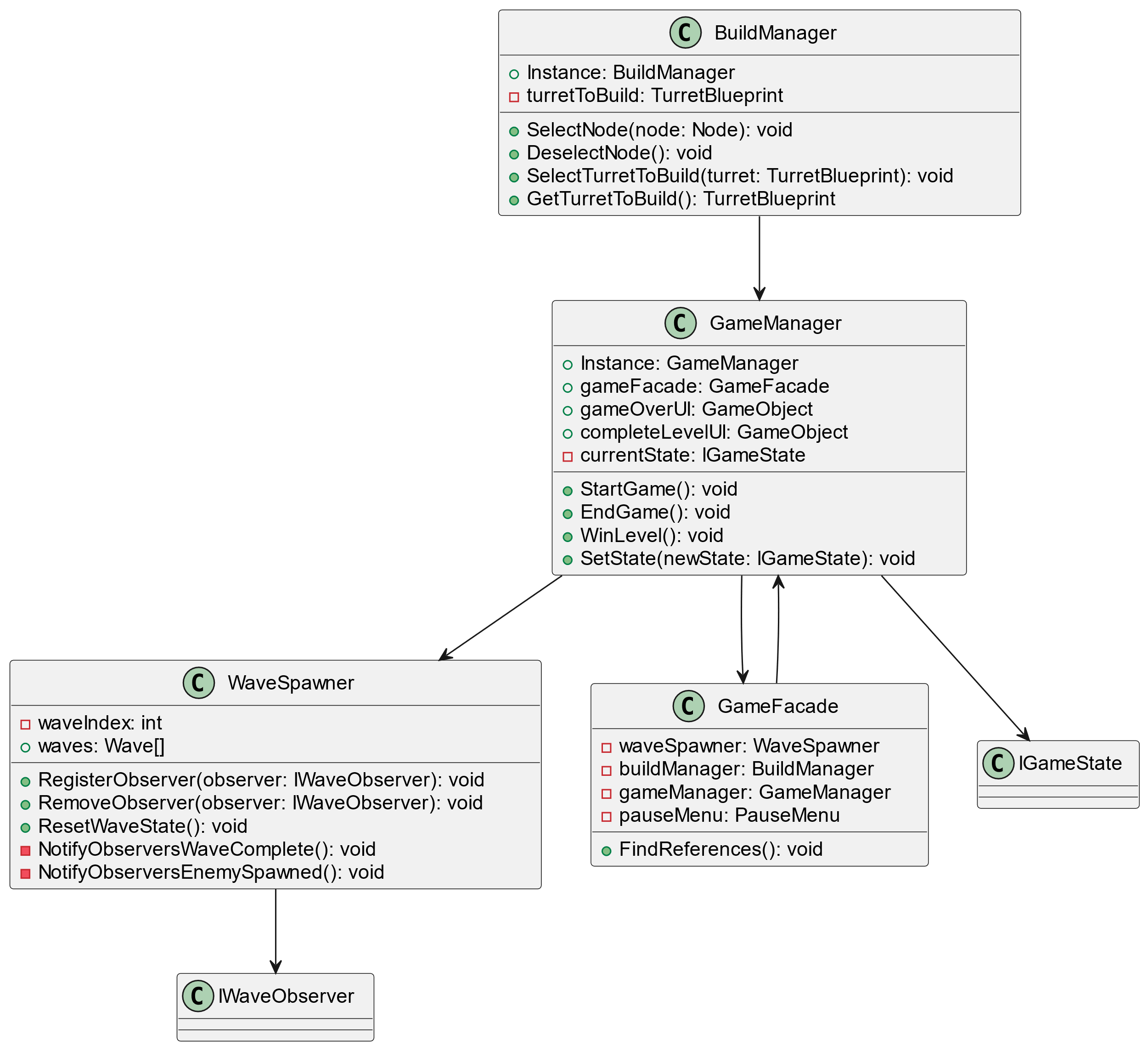
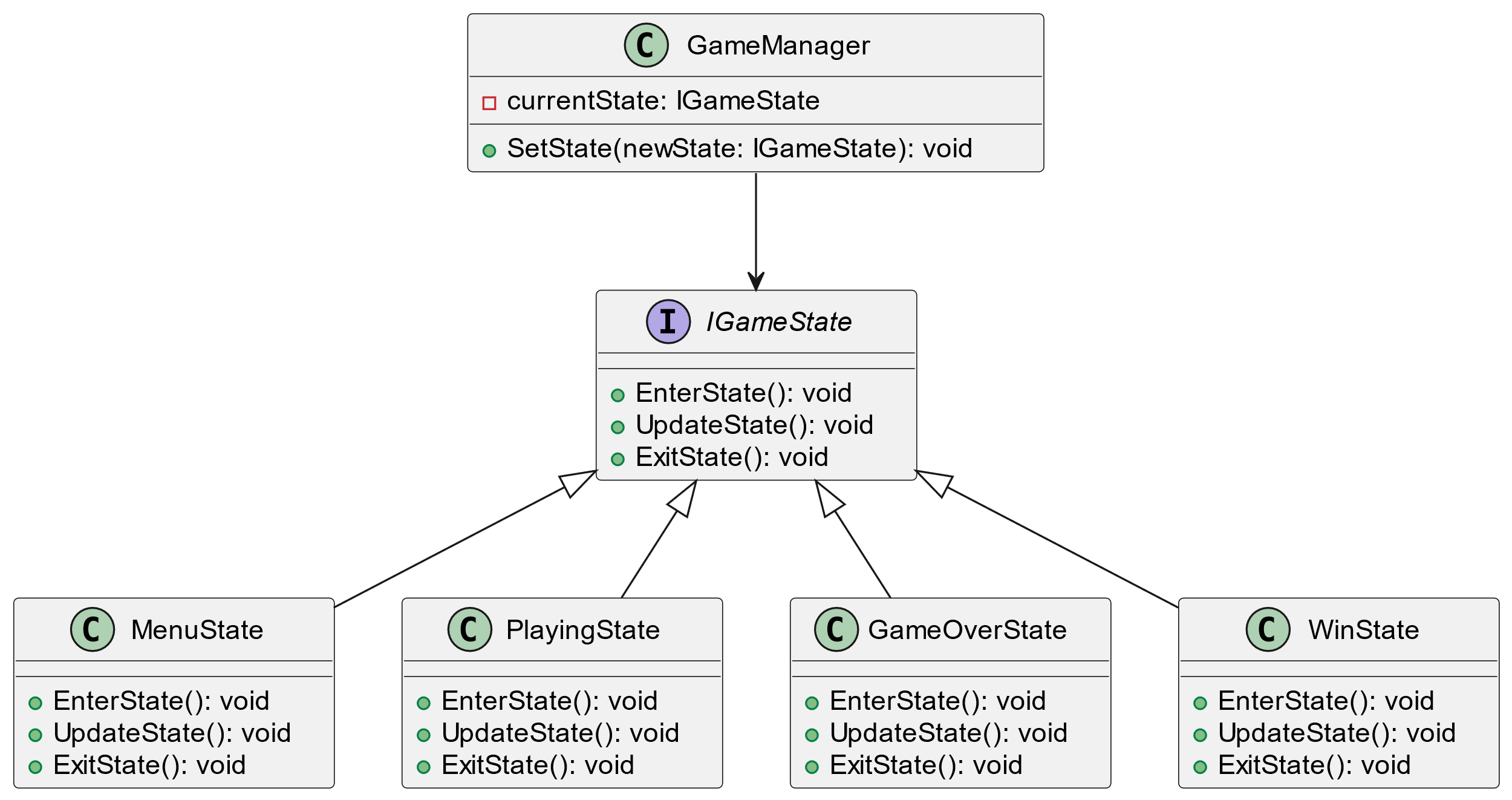
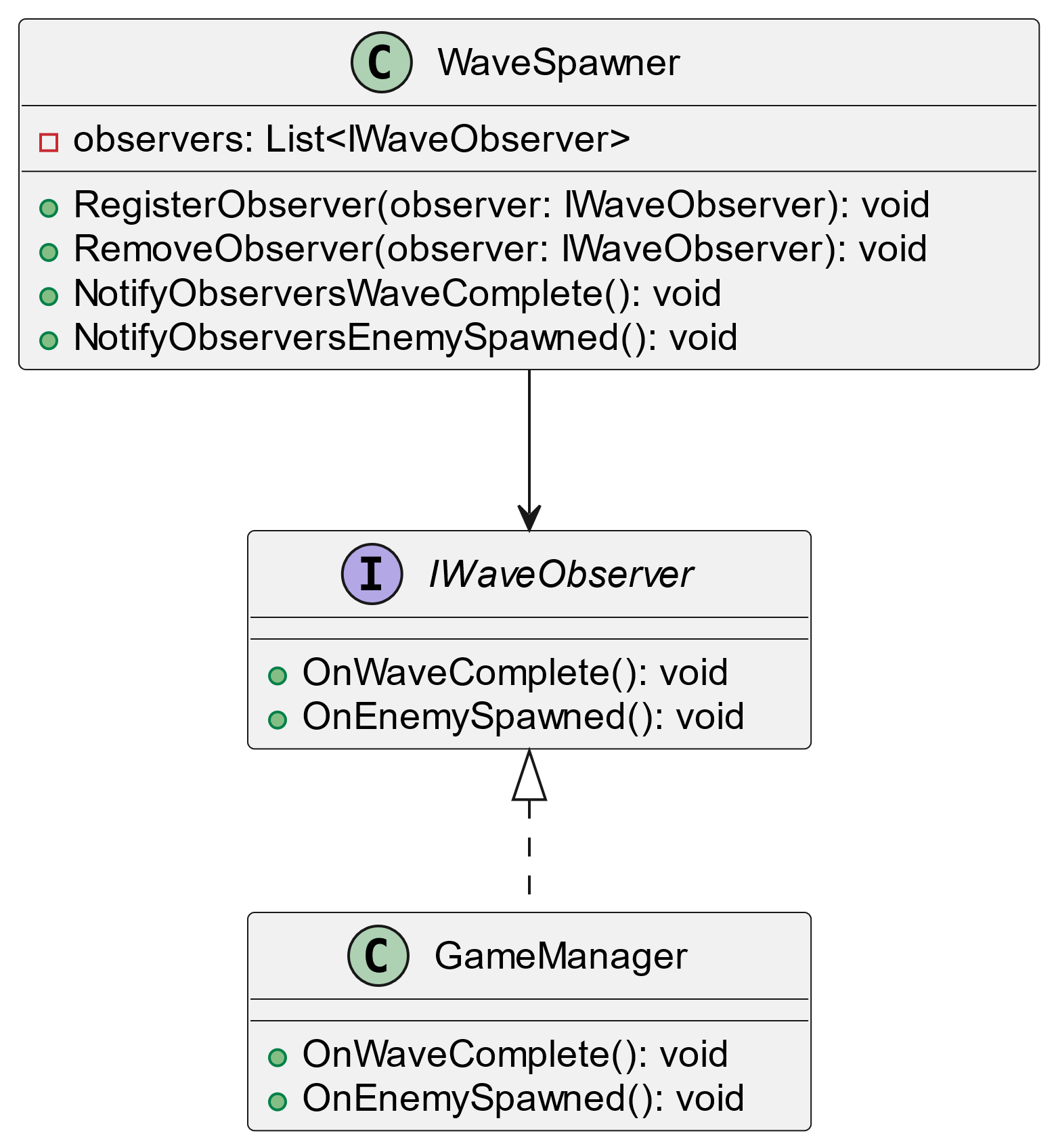
**2. Structural Patterns**

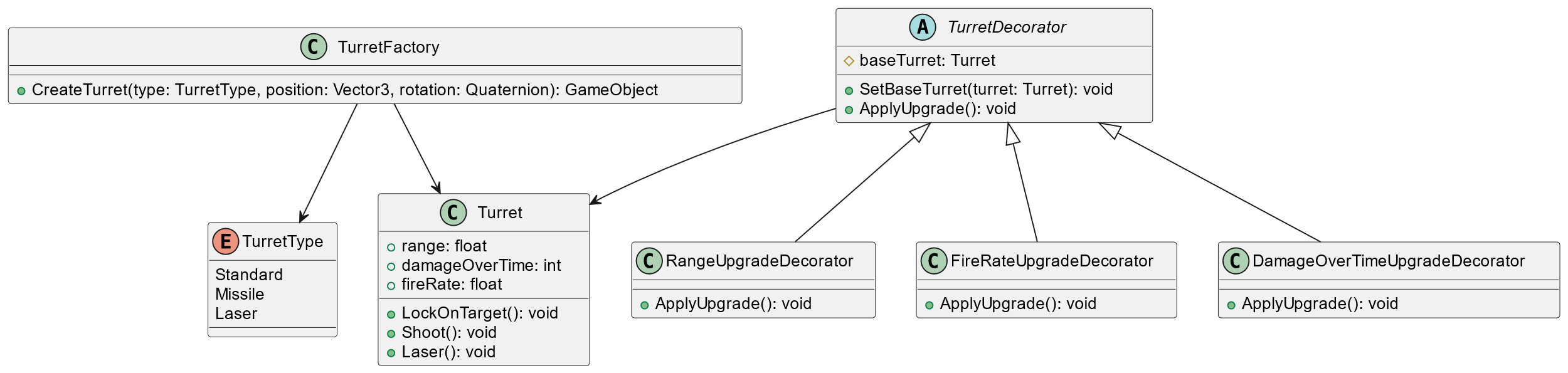
* **Decorator**:
  + Used to enhance the functionality of game entities, such as adding special effects to turrets or providing additional capabilities to enemies.
  + The decorator pattern ensures flexibility in applying modifications without changing the core structure of the objects.
* **Facade**:
  + The GameFacade class provides a unified interface to interact with multiple subsystems like WaveSpawner, BuildManager, PauseMenu, and more.
  + This pattern simplifies the complex interactions by providing a single point of access, thus reducing dependencies and making the code more readable.

**3. Behavioral Patterns**

* **Observer**:
  + Implemented between WaveSpawner and GameManager to notify the GameManager about specific events, such as when a wave is completed or when enemies are spawned.
  + This pattern decouples the subject (WaveSpawner) from its observers (GameManager), promoting better modularity and scalability.
* **State**:
  + The state pattern is used to manage game states like MenuState, PlayingState, GameOverState, and WinState.
  + Each state is encapsulated in its own class, and the transitions are managed by the GameManager. This makes it easier to modify or add new states without impacting the rest of the code.

**UML Diagrams**

** ** 



**Usage Instructions**

**1. Setting Up the Project**

* Import the project into Unity (tested on Unity version 2022.3 or later).
* Open the Level01 scene to start editing or running the game.
* Ensure all prefabs (turrets, enemies, etc.) are correctly assigned in the Inspector under GameManager, BuildManager, and WaveSpawner.

**2. Playing the Game**

* Start the game from the MainMenu scene.
* Click "Start" to enter the gameplay mode.
* Use the mouse to build turrets by selecting nodes on the map.
* Use WASD or arrow keys to navigate the camera.
* Press Esc to pause the game or access the menu.
* Survive through multiple waves of enemies to win the level.

**3. Adding New Waves or Turrets**

* **New Waves**: Modify the waves array in the WaveSpawner component in the Unity Inspector. Each element represents a different wave configuration (enemy type, count, and rate).
* **New Turrets**: Create a new turret prefab and add it to the BuildManager using the Factory Method for easy integration.

**Assumptions and Limitations**

**Assumptions**

* The player can freely navigate the camera throughout the level and build turrets on pre-designated nodes.
* The player starts with a set amount of money, which is used to build turrets and gain more by defeating enemies.
* Each enemy wave has a defined set of enemies that spawn at a given rate.

**Limitations**

* The game currently supports only one active level at a time. Adding multiple levels requires duplicating the Level01 scene and adjusting wave configurations accordingly.
* The camera controls and wave spawning mechanisms are simplistic and may need enhancement for larger, more complex maps.
* The current implementation of the state pattern may need further expansion to support additional game states like a tutorial or multiplayer mode.

**Version Control**

* The project uses Git for version control.
* Each new feature or bug fix should be documented using clear and descriptive commit messages.
* Activity of the current repository:
  + Initial commit of Unity project files

created [main](https://github.com/tincenma/TowerDefenseProject/activity?ref=main)

* + Added README.md file with project description

pushed 1 commit to [main](https://github.com/tincenma/TowerDefenseProject/activity?ref=main)

* + Updated README.md

pushed 1 commit to [main](https://github.com/tincenma/TowerDefenseProject/activity?ref=main)

* + Merge branch 'main' of <https://github.com/tincenma/TowerDefenseProject>

pushed 2 commits to [main](https://github.com/tincenma/TowerDefenseProject/activity?ref=main)

**Future Enhancements**

* **Enhanced AI for Enemies**: Introduce behaviors like pathfinding and target prioritization to make enemy movements less predictable.
* **Level Editor**: Implement a level editor to allow users to create and save their own maps and wave configurations.
* **Multiplayer Mode**: Introduce a cooperative multiplayer mode where players can defend the map together.

**Conclusion**

This Tower Defense game project showcases the effective use of design patterns to create a scalable, modular, and maintainable codebase. By leveraging Singleton, Factory Method, Facade, Decorator, Observer, and State patterns, the project achieves clear separation of concerns, ease of future expansion, and a solid foundation for further enhancements.