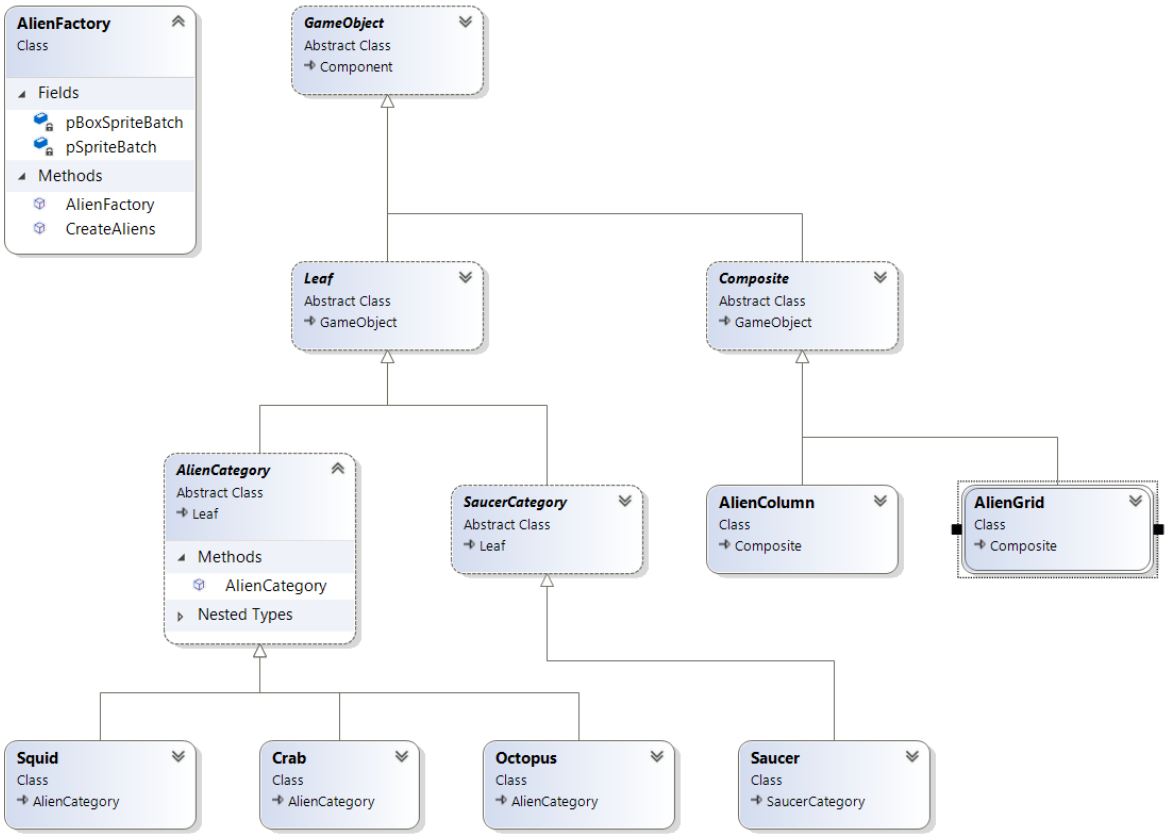
SE 456 Space Invaders Design Document

This document is to introduce all the design pattern used in Space Invaders game. It will briefly discuss the purpose and benefit of design patterns and how each design patten implemented in the code with UML diagrams explanation.

1. Factory:

UML Diagram:



The Alien Factory class and create instances of Squid, Crab, Octopus, Saucer, Alien Column, and Alien Grid.

Pattern description:

Factory pattern provides an interface for creating objects, and meanwhile it allows subclasses to alter the type of objects that will be created. It means that client can create objects without knowing specifically internal logic.

Pattern benefits:

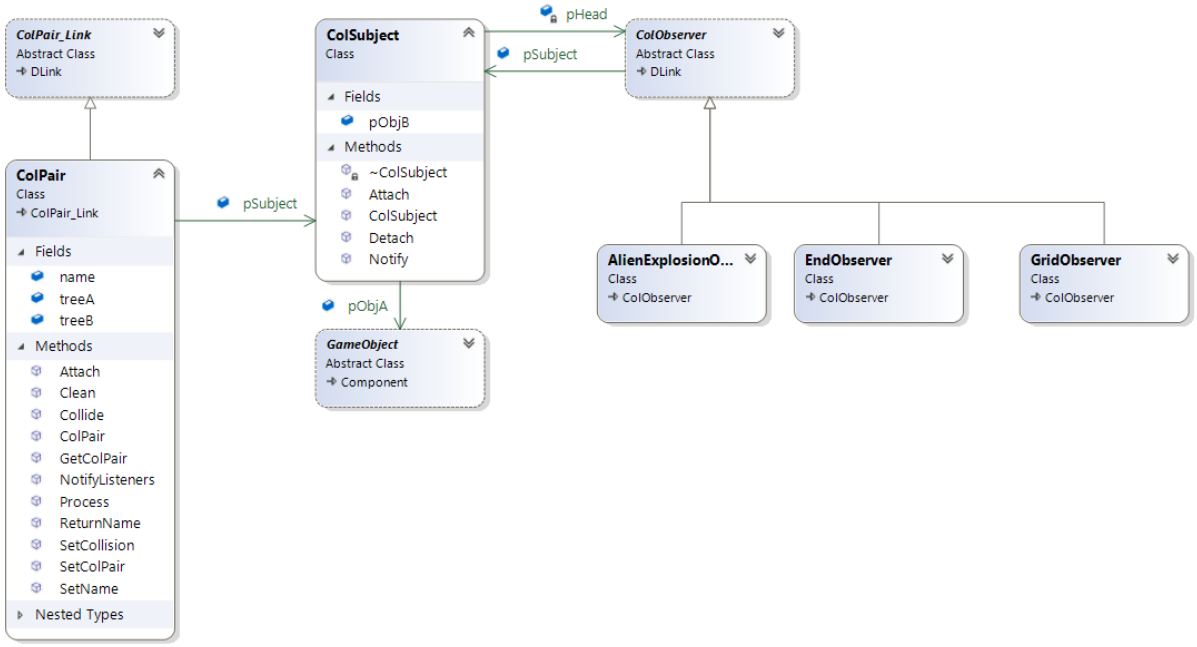
1. This pattern allows client to create multiple objects conveniently and without knowing their actual construction implementation.
2. It is easy to add new objects to create in factory.

Object Oriented Mechanics:

The Alien Factory class has a Create() method. This method will take the type of alien as an input parameter. This function uses a switch case statement to construct different objects with different input parameter of object names. In my code, I need to construct multiple aliens and shield bricks that I use the factory pattern to construct them by calling Factory.Create(). If I want to add new aliens or brick type I can easily modify the factory class and add relative implementation code. While other people do not need to know how to create the alien, they could just use this interface. It is a good example of decoupling.

2. Observer:

UML Diagram:



Collision Subject has a pointer to Collision Observer Class which is an abstract class. The concrete observer class will implement the Notify() method to tell client to do corresponding tasks.

Pattern description:

Observer pattern is a software design pattern in which a subject maintaining a list of its observers. Normally, it is a one-to-many dependencies. When any kind of state changes happen, subject will notify observers automatically.

Pattern benefits:

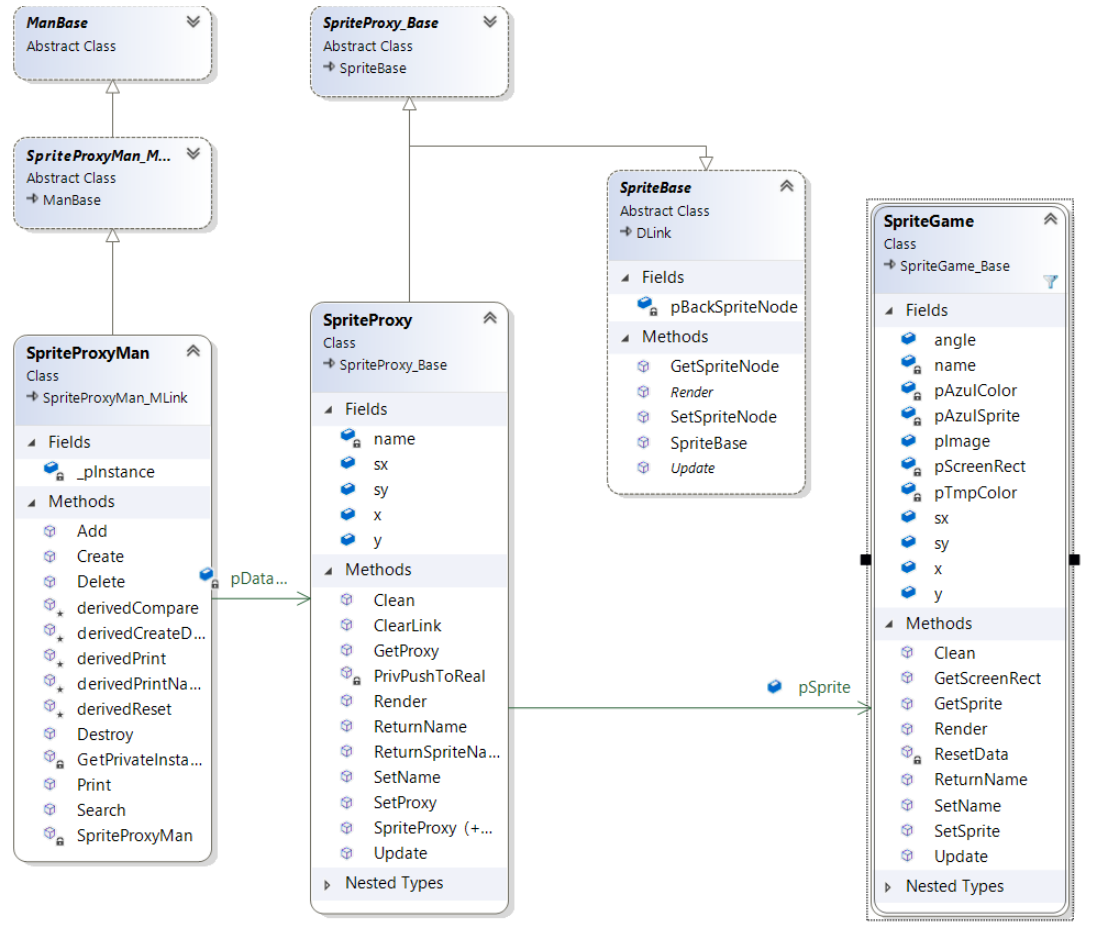
1. It makes coupling loose, the observers can register and unregister any time and meanwhile the subject have no awareness of the existence of observers. It also allows reusability.
2. By having a watchdog object, you don't have to constantly checking the occurrence or the change of a specialized activity or the state of the system. It saves the processing power and memory.

Object Oriented Mechanics:

When an event happens, the client executes the corresponding action by observer notifying. This pattern is used in the Collision System to apply the list of responses associated with the collision event happened between Game Objects. In the code, the Collision Pair Manager will attach the observers to the Subject. When the Collision Subject changes, it will notify all the related observers by calling the Notify() method in each object. The Alien Grid is keep moving horizontally, when they reach the left wall or right wall the observer catch that state and call the Notify() to tell them change the movement direction.

3. Proxy:

UML Diagram:



The Sprite Proxy class has the same type of variable of position x & y as real Sprite Game does. It also has a pointer to point at real Sprite Game object.

Pattern description:

A design pattern that allows for the creation of a surrogate or placeholder object that acts as a substitute for another object. The proxy object is used to control access to the original object,

providing a way to add additional functionality or behavior to the original object without modifying its code. It adds a level of indirection between the client and the real object, allowing the proxy to intercept requests and perform additional operations before or after forwarding them to the real object.

Pattern benefits:

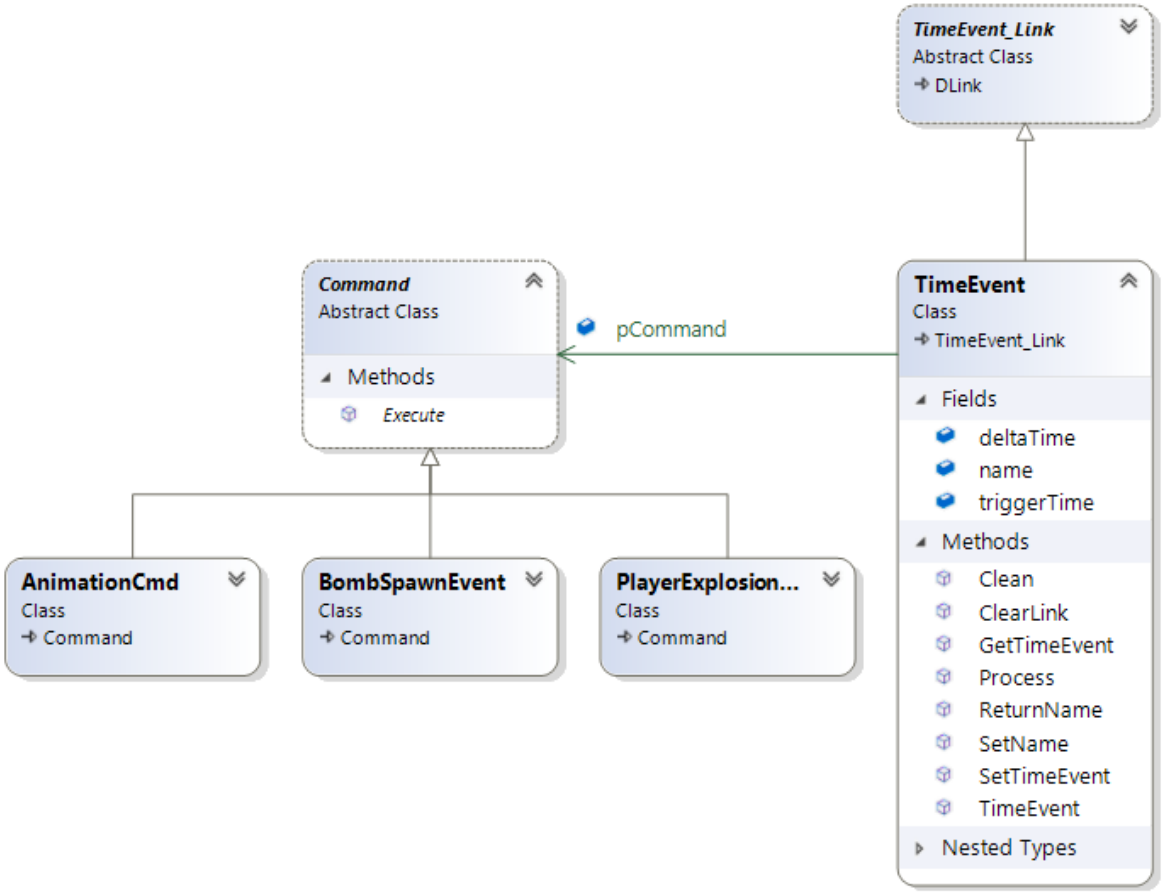
1. The proxy object hides the details of the original object's implementation from the client.
2. Clients can interact with the original expensive object indirectly, allowing for additional functionality or behavior to be added without modifying the original object's code.

Object Oriented Mechanics:

A client object needed to invoke specific attributes/behaviors of a heavy object. Accordingly, the client will use the real object’s interface which is the proxy class, then the proxy will execute the requested changes through the actual object. In the code, to create real aliens objects, especially for image flipping behavior and position changing, are heavy as each alien has numerous data values, which will result in an unnecessary extreme memory load while executing the aliens’ animation. Hence, we use proxy to take place the real construction of sprite.

4. Command:

UML Diagram:



Time Event Class has a pointer to Command class which is an abstract class. Its inherited class will implement the Execute function when timer event trigger.

Pattern description:

A behavioral design pattern which usually constructs an object to encapsulate all information needed to perform an action or trigger an event later. This information includes the method name, the object that owns the method and values for the method parameters. It encapsulates

commands in objects allowing requests triggering without knowing the requested operation or the requesting object. Additionally, it provides the options to queue commands, undo/redo actions and other manipulations.

Pattern benefits:

1. For decoupling, it allows the invoker to remain unaware of the receiver's identity, making the code more flexible and easier to maintain.
2. The receiver can be modified or replaced without affecting the invoker, as long as it still responds to the same interface.

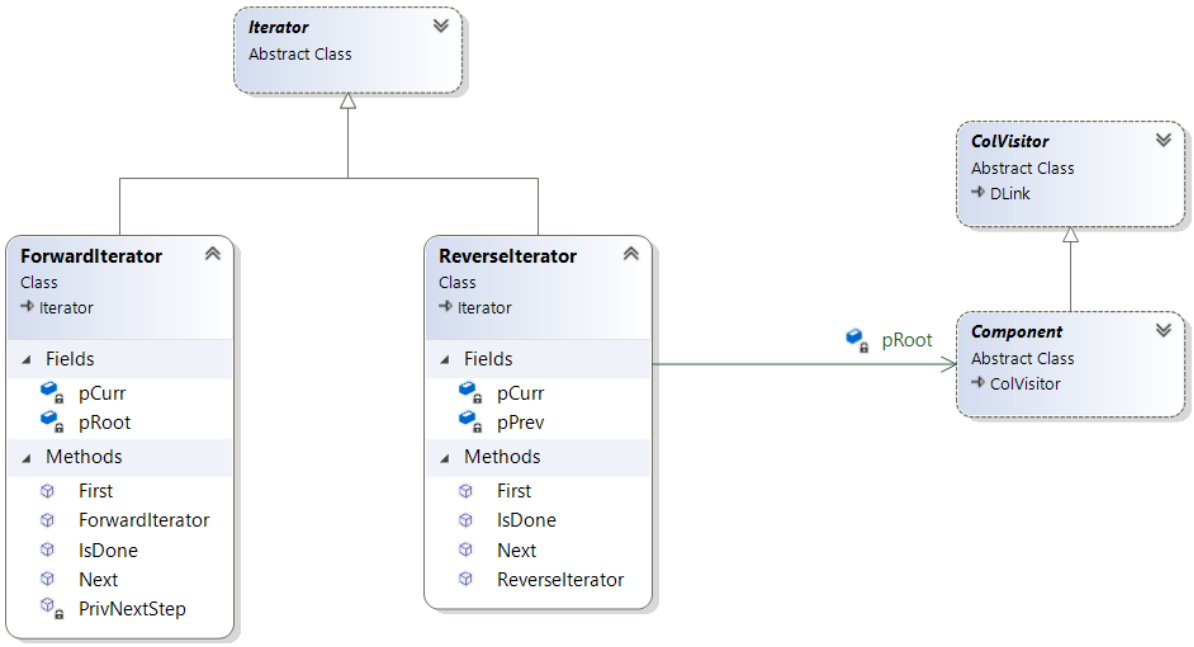
Object Oriented mechanics:

There are two main components of command pattern, receiver and invoker. The receiver is a command object and it does the work when the Execute() method in command is called. An invoker object knows how to execute a command, and optionally does bookkeeping about the command execution. The invoker does not know anything about a concrete command, it knows only about the command interface. The client decides which commands to execute at which points. To execute a command, it passes the command object to the invoker object.

In the code, the command design pattern is used with the timer system. The client needs to trigger a group of timer actions which are the commands and change that timespan constantly. For instance, the grid has to drop a bomb from the bottom alien randomly within a particular time period. Therefore, by applying this pattern can walk through different timed events and execute them when the time matches.

5. Iterator:

UML Diagram:



The Forward Iterator and Reverse Iterator inherit the base Iterator class. They both have a pointer to component and current node. The reverse iterator has a previous node pointer.

Pattern description:

A design pattern in which an iterator is used to traverse a container and access the container's elements. An iterator delivers a way to sequential access to the objects in a collection without exposing its underlying representation.

Pattern benefits:

1. You can clean up the client code and the collections by extracting bulky traversal algorithms into separate classes.

2. You can implement new types of collections and iterators and pass them to existing code without breaking anything.

3. You can iterate over the same collection in parallel because each iterator object contains its own iteration state.

4. You can delay an iteration and continue it when needed.

Object Oriented mechanics:

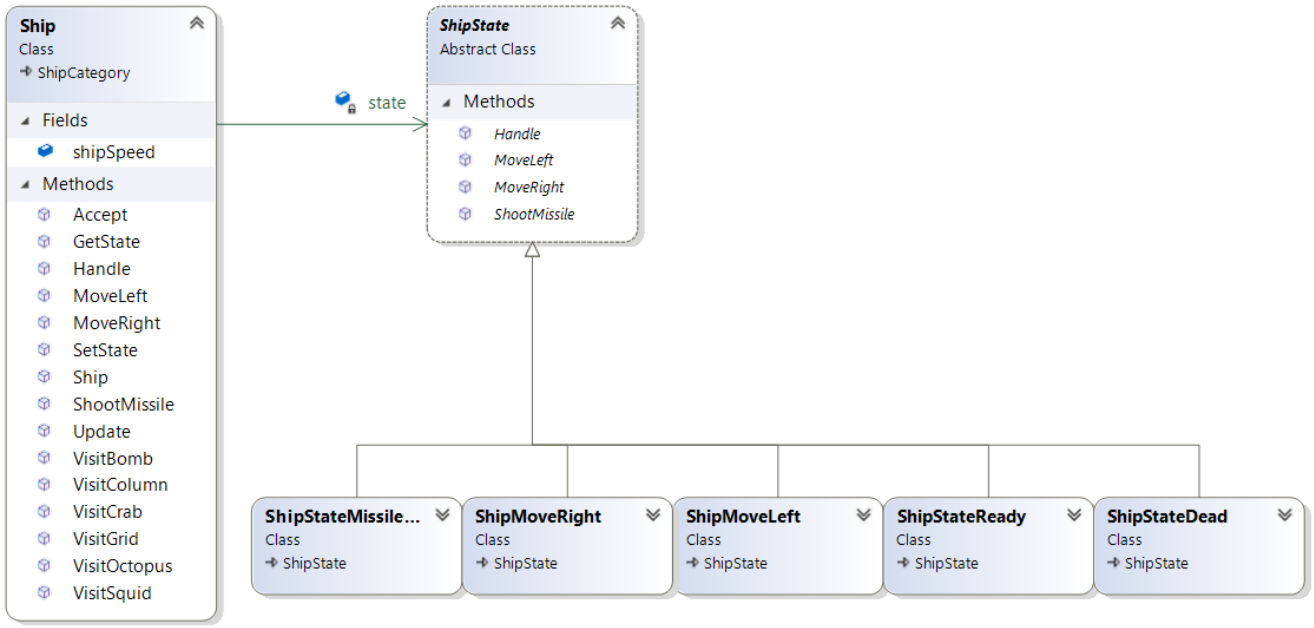
Any iterator always has three operations. It behaves like a for loop eg.

for (xxx.First(); xxx.Next(); !xxx.isDone())

First() returns the first element of data structure, Next() return the next element in case you can define you own search way to access your next element. IsDone() returns whether is it the last element of data structure. To traverse the Game Object Node tree, using this pattern to walk through each item either forward or reverse depending on the cost. In the code, the Aliens Grid movement and animation apply this forward iterator. The elimination of collision box use the reverse iterator.

6. State:

UML Diagram:



Ship class has a pointer to Ship State and the concrete State classes have a method name Handle().

The ship will call Handle() while internal state changes and handle the corresponding activities.

Pattern description:

State is a behavioral design pattern that allows an object to change its behavior after its internal state changes. The current state of objects determines object's behavior. It appears as if the object changed its class, and it is represented by a separate object that implements a common interface.

The object's state can change based on the events. Meanwhile the user input can also triggers a transition to a new state.

Pattern benefits:

1. It organizes the code related to particular states into separate classes.

2. It introduces new states without changing existing state classes or the context.

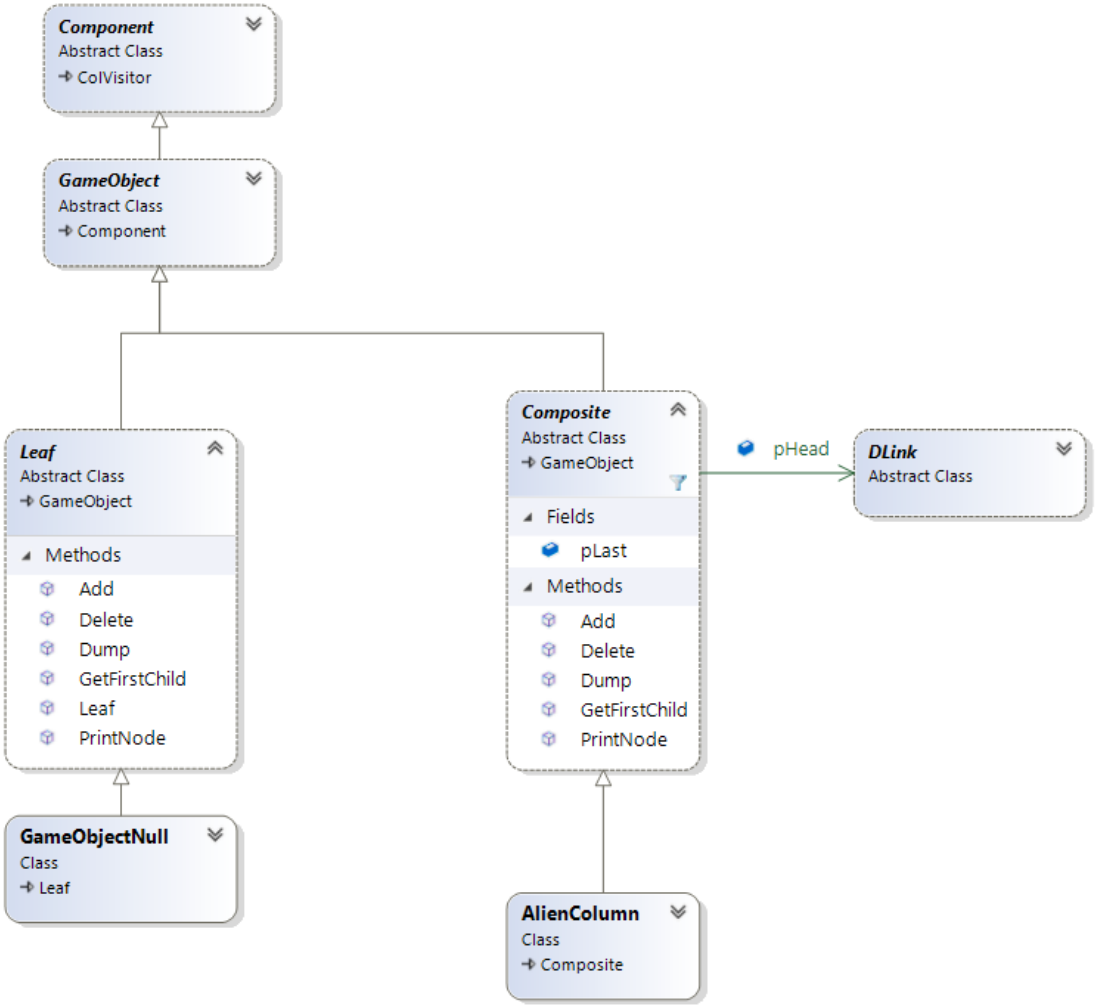
3. It simplifies the code of the context by eliminating bulky state machine conditionals.

Object Oriented mechanics:

The context changes its state internally giving some specific conditions. Each state handles the same actions differently. The game requires to control the ship’s behaviors including the ready state, shooting missile state, and dead state. Those states are encapsulated in Ship class. While the ship is in the shooting missile state, it can’t send another one. Therefore, there are no multiple missiles existing simultaneously. The missile should hit one of the game objects before the player can launch another missile. This is an exemplification of a same object under different states to act different behaviors.

7. Composite:

UML Diagram:



Pattern description:

The Composite pattern is a structural design pattern that allows you to treat a group of objects in the same way as a single instance of an object. It involves creating a class hierarchy where some classes

represent individual objects, and other classes represent collections of objects that can be composed of individual objects or other collections.

Pattern benefits:

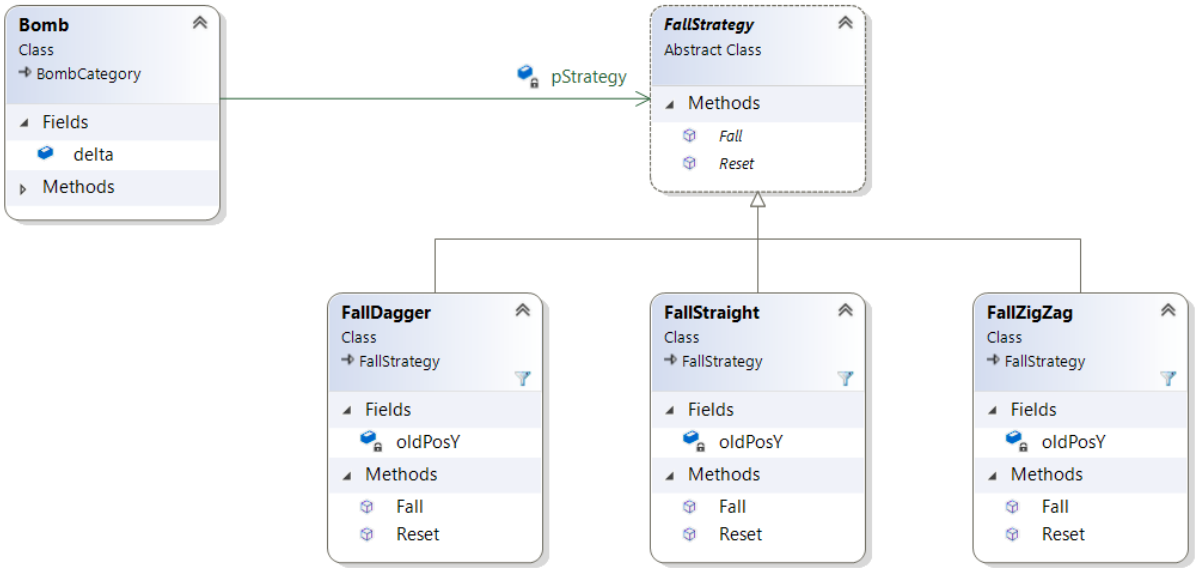
1. It provides a flexible structure that can be easily extended and modified to accommodate new types of objects or collections.
2. It simplifies client code by providing a uniform interface for working with individual objects and collections.
3. It guarantees the consistency of code behavior.

Object Oriented mechanics:

This pattern allows clients to interact with individual objects and collections in a unit way without the necessity to know the specific difference between them. Therefore, this pattern will simplify client code and make it easier to add new types of leaf or composite objects to the hierarchy. In the space invader code, the Alien factory uses this pattern. Not only the alien factory class can construct the leaf objects which are the real aliens object such as squid and crab, it can also construct an alien grid or alien column which are the collections of the aliens. This will allow the aliens can move as an entirety.

8. Strategy

UML Diagram:



The Fall Strategy class 3 derived classes because the aliens randomly drop 3 types of bombs. The derived class will implement the abstract methods Fall() and Reset().

Pattern description:

A behavioral design pattern that allows you to define a family of interchangeable algorithms, encapsulate each of them into a separate but interchangeable class object.

Pattern benefits:

1. It can swap algorithms which used inside an object at runtime easily.

2. It encapsulate the algorithm into separate single objects which will provide a high degree of flexibility and extensibility.

9. Visitor

UML Diagram: