Structural Design Pattern

Hung Tran

Fpt software

November 26, 2021

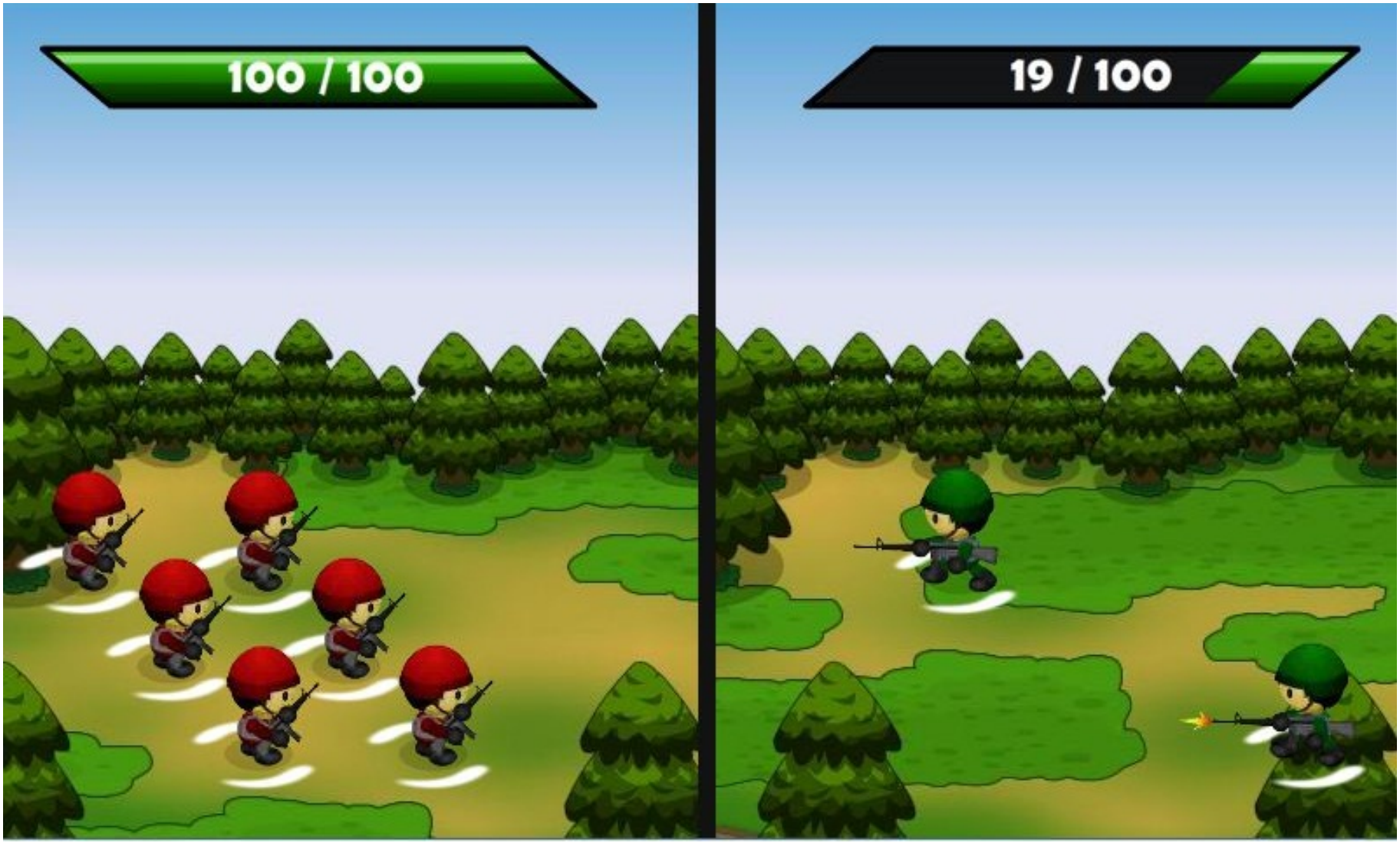
Outline

Structural Pattern Overview

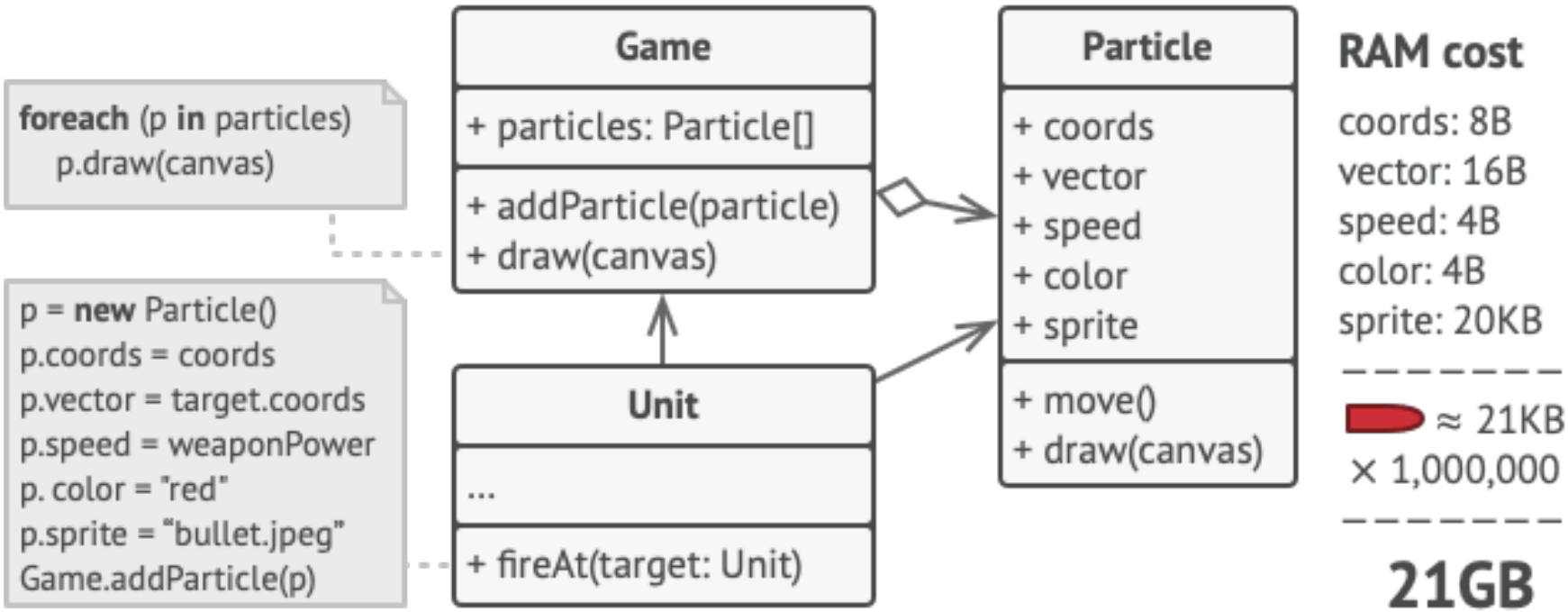
How classes and objects are composed fo form larger structure.

- **Adapter**: Convert the interface of a class into another interface.
- **Bridge**: Decouple an abstraction from its implementation.
- **Composite**: Compose objects into tree structure.
- **Decorator**: Attach additional responsibilities to an object dynamically.
- **Façade**: Provide a unified interface to a set of interfaces.
- **Flyweight**: Use sharing to support large numbers of fine-grained objects efficiently.
- **Proxy**: Provide a surrogate or placeholder for another object to control access to it.

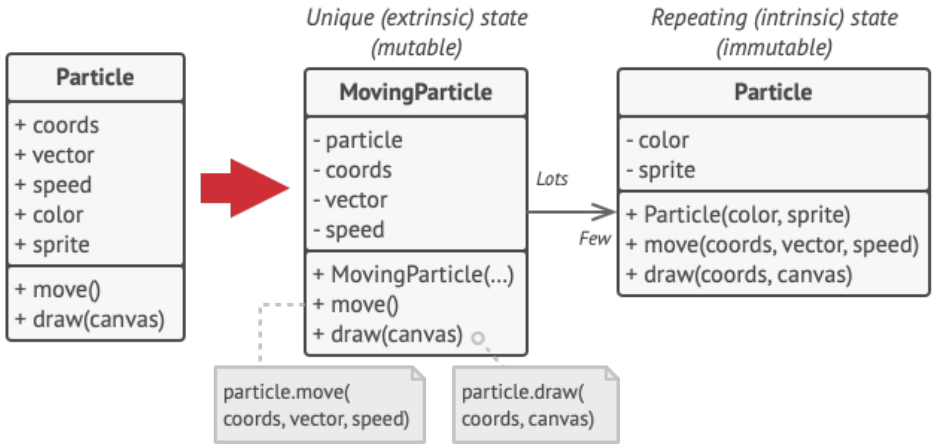
Problem Statement



Problem Statement



Problem: Particle class

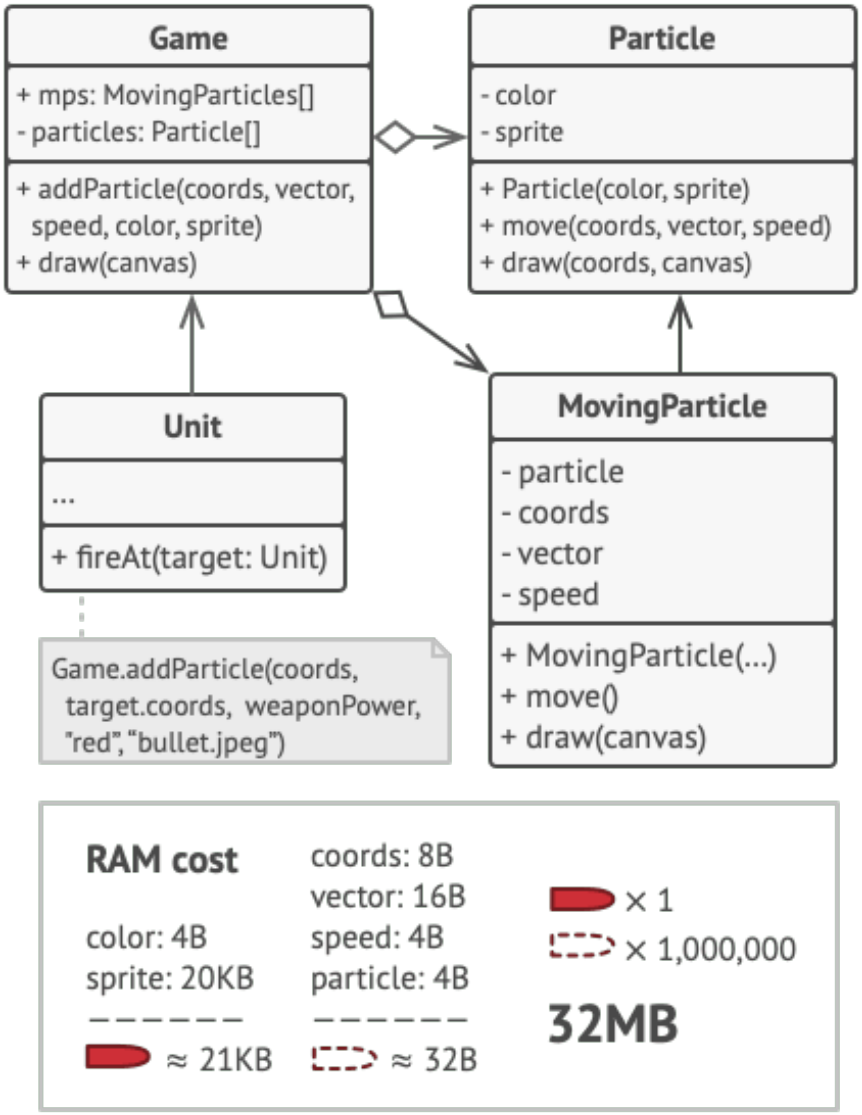


- The color and sprite fields consume a lot more memory than other fields.
- These two fields store almost identical data across all particles.
- For example, all bullets have the same color and sprite.
- Coordinates, movement vector and speed, are unique to each particle

Problem: Particle class

- The intrinsic state: The constant data of an object, the other objects only can read.
- The extrinsic state: The other objects can alter these states.

Solution: Flyweight pattern

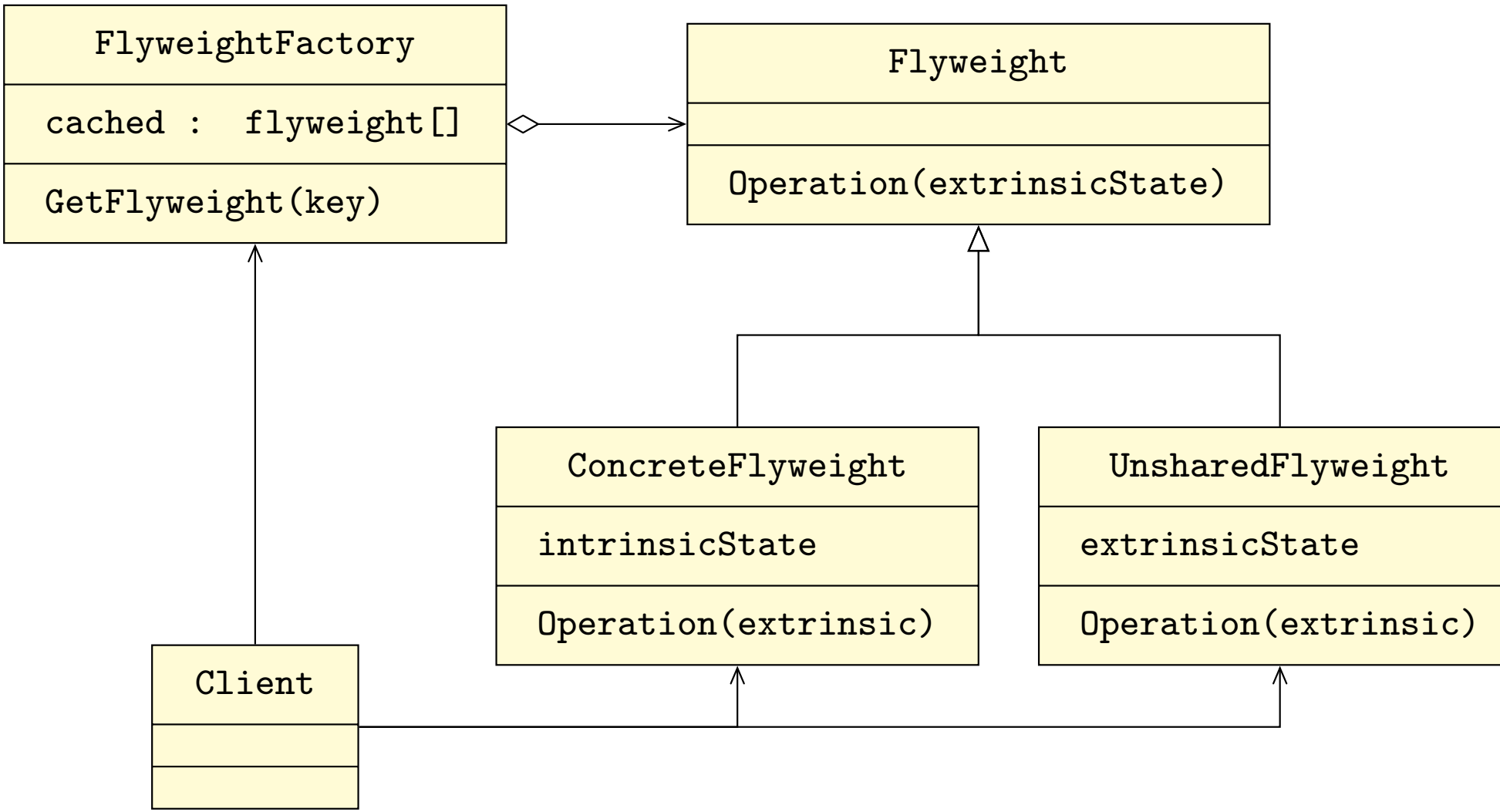


- Stop storing the extrinsic state inside the object
- Pass this state to specific methods which rely on it.
- Only the intrinsic state stays within the object, letting you reuse it in different contexts
- Only three different objects would suffice to represent all particles in the game: a bullet, a missile, and a piece of shrapnel.

The Intent of Flyweight Design Pattern

Flyweight is a structural design pattern that lets you fit more objects into the available amount of RAM by sharing common parts of state between multiple objects instead of keeping all of the data in each object.

Structure of Flyweight Pattern



Applicability

- Use the Flyweight pattern only when your program must support a huge number of objects which barely fit into available RAM.
- The benefit of applying the pattern depends heavily on how and where it's used. It's most useful when:
- An application needs to spawn a huge number of similar objects
- This drains all available RAM on a target device
- The objects contain duplicate states which can be extracted and shared between multiple objects

How to Implement

- Divide fields of a class that will become a flyweight into two parts: the intrinsic state and the extrinsic state
- Leave the fields that represent the intrinsic state in the class, but make sure they're immutable. They should take their initial values only inside the constructor.
- Go over methods that use fields of the extrinsic state. For each field used in the method, introduce a new parameter and use it instead of the field.
- Optionally, create a factory class to manage the pool of flyweights. It should check for an existing flyweight before creating a new one. Once the factory is in place, clients must only request flyweights through it. They should describe the desired flyweight by passing its intrinsic state to the factory.

Pros and Cons

- You can save lots of RAM, assuming your program has tons of similar objects.
- You might be trading RAM over CPU cycles when some of the context data needs to be recalculated each time somebody calls a flyweight method.
- The code becomes much more complicated. New team members will always be wondering why the state of an entity was separated in such a way.

Relations with Other Patterns

- You can implement shared leaf nodes of the Composite tree as Flyweights to save some RAM.
- Flyweight shows how to make lots of little objects, whereas Facade shows how to make a single object that represents an entire subsystem.
- Flyweight would resemble Singleton if you somehow managed to reduce all shared states of the objects to just one flyweight object. But there are two fundamental differences between these patterns:
- There should be only one Singleton instance, whereas a Flyweight class can have multiple instances with different intrinsic states.
- The Singleton object can be mutable. Flyweight objects are immutable.

Thank You!