Useful Game Design Patterns

How are Design Patterns useful?

So we've done a little exploration into design patterns, and maybe you're excited by the possibilities they offer to solve problems more easily. Some questions might still linger, though.

- What does it really matter how my code is organized?
- I can't see how this solution could possibly apply to my problem
- What is a prototype and how do I use it?

I'm going to try to address some of these questions with an example and thought experiment.

So You're Going to Tell Me How To Solve It

No, I won't. I want you to get used to the idea of thinking about how to solve certain design challenges by introducing some concepts you might not have considered and some resources you might not have seen.

I'll try and describe how the pieces fit together and give you some ideas and resources about how to make them fit, but I'm not going to give code that can be used in future assignments. That said, I think design patterns are really cool and extremely useful if you can find the right way to apply them.

The Challenge

Let’s think about this in the context of an example: a game. Little games are fairly popular in this track because people enjoy them and they can be fun to craft (in small at least).

Let’s say we want to make a little RPG. Nothing too fancy.

- Three races
- Three character classes
- Four types of monsters
- Consumable items
- Special attacks

Design Process

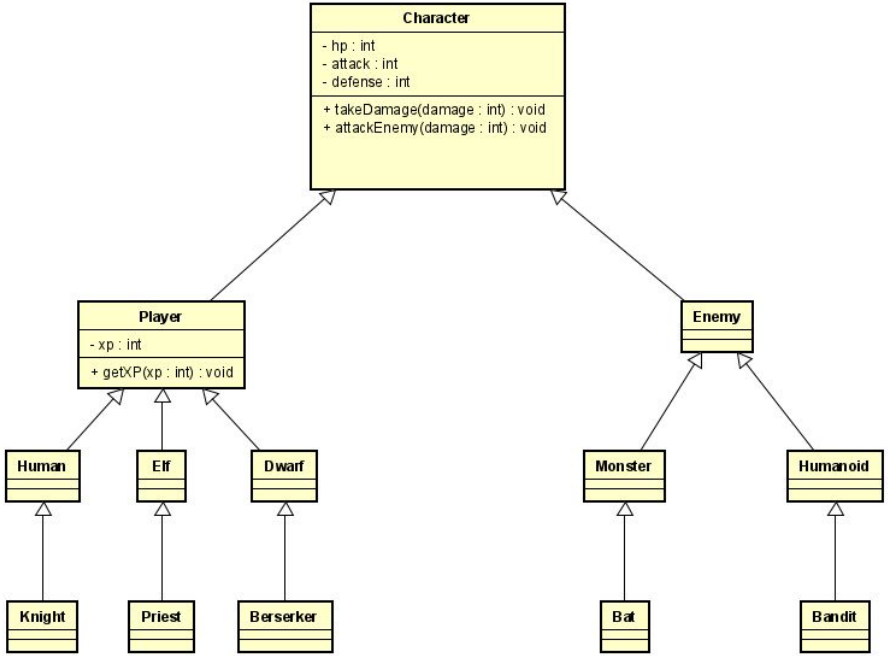
How would we start to design a system like this with object-oriented design?

It's all classes and inheritance, so… let's break it all into subclasses. That makes sense, right? The shared attributes will be on the base level class and then the specialization classes will add more functionality as it goes down. Ok, let's go with that.

- Overarching class for both players and monsters
- Subclass for player and enemy types
- Subclass for races and monsters
- Subclass for the player's class

Class Diagram

A (simplified) class diagram might look like this:



The Problem

So we have our plan and we just have to implement it.

- We declare the top level object and create separate files for every subclass
- There are a lot of files in here
- Every time I make a new one I have to keep calling the super constructor
- How do the entities interact with each other?
- How am I going to use a special ability?
- How do the enemies know what to do?

The Mediator Pattern

This seems like a sticky problem. We want some kind of dynamic interplay between a character and a monster.

- How is damage calculated?
- Who hits? Who misses?
- Who is in control of combat?

Enter the **mediator pattern**. Creating a mediator forces objects to interact through it, which means the mediator can decide things like who hits, how much damage is taken, etc. But what do we pass in? The characters themselves?

The Command Pattern

Say a character wants to do an attack. What does it need? We can kind of brainstorm what we need to resolve an attack

- The enemy target
- What kind of attack it is (Elemental damage? Piercing? Blunt?)
- The hit success rate
- How much damage it does

What if we bundle all these together into a single object? The **command** pattern allows us to bundle relevant information into a single command so that another entity can deal with it.

Next Steps

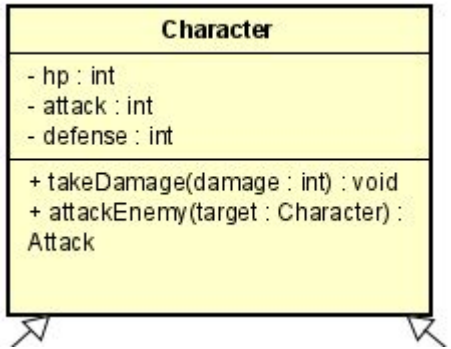
Ok, so now we have an idea of how this might work. A character decides to attack, which creates an Attack command. It has:

- A reference to the attacking character
- A reference to the defending character
- How much damage it does
- Perhaps how likely it is to hit
- The type of damage

And it’s done in a way that’s mostly decoupled from the character itself. The attack goes to the mediator which processes the attack and determines the result.

Subclass Explosion

With the basic attacking mechanics down, let's think about how our classes fit into it. Every character needs to be able to generate an attack, because characters attack. We'll put it in the Character class.



Now Player needs to implement it, and every subclass of Player too. So does Monster and every subclass of Monster. This could get complicated….

Subclass Explosion

Not only are we locked into an Attack object for every one of our eight subclasses, it will be the same for every kind of attribute we try to add. Not every one of these subclasses might need these attributes, so let's put them only on some of the subclasses. Right?

Well, there are some challenges with that.

- Generic objects are better when they're interacting
- Expecting specialized objects means you have a decision for every interaction

Subclass Explosion

Is there a way to be able to extend the functionality of an object without changing its class? To be able to bolt on and change pieces of its behavior without changing its structure?

YES!

It’s called the **Component Pattern**

- Behavior broken down into component attributes
- The component can modify the base object and extend its behavior
- The component contains a reference to its parent object
- The type of actor is modified by its components

What Kind of Components?

Is that really it? What kind of things can we even do with components?

- Race
- Class
- Level
- Equipment
- Spellcasting

Every time we want to modify the behavior of an entity or add a new system we make a new component. The components modify the base object so that the most generic functions stay accessible, but become more specialized as the components are attached.

Is It Really More Flexible?

The flexibility and power is even stronger when you consider that the components can be changed even during gameplay. If you want the character to be able to change their race or class, you just need to change the component holding that information.

This approach makes any game project immensely easier to deal with, and it can be extended to other domains as well. Components are a huge part of frontend web programming, where the pages are sliced into smaller and smaller components that can share their state with the parent, and in design tools like Figma and Adobe XD.

Further Reading

I’m really excited about how powerful some of these design patterns can be. If you’re interested in more, check out

- Refactoring.guru My favorite site for understanding a new pattern and getting an example (with code in a bunch of different languages)
- Game Programming Patterns an entire *free* book discussing the most useful patterns in game design and how to implement them