Game Programming Patterns – Revisited – Command

Command
- is a reified method call
  - is a method call wrapped in an object
  - an OO replacement for callbacks

ex. Configuring Input
- * taking in input (ex 'A' press) & translating it to a meaningful action in the game, typically called 1x frame by the game loop
- * many games let users configure button mapping

① Base class representing a triggerable game command

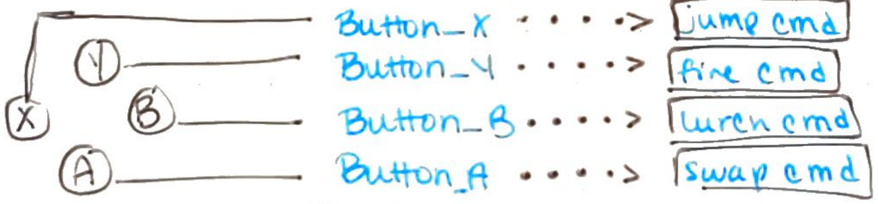
"when you have an interface w/ a single method that doesn't return anything there's a good chance it's the command pattern

② Create subclass for each game action

③ in our input handler, store a pointer to a command for each btn.

④ when btn pressed call execute()

* can define cmd class whose execute() does nothing & assign btn handler to that object. This pattern is Null Object



ex. Directions for Actors
- * The assumed coupling of the previous example of a top-level function limits its usefulness

"Instead of calling functions that find the commanded object themselves, we'll pass in the object that we want to order around."

```
virtual void execute(Game-Actor& actor) = 0;
```

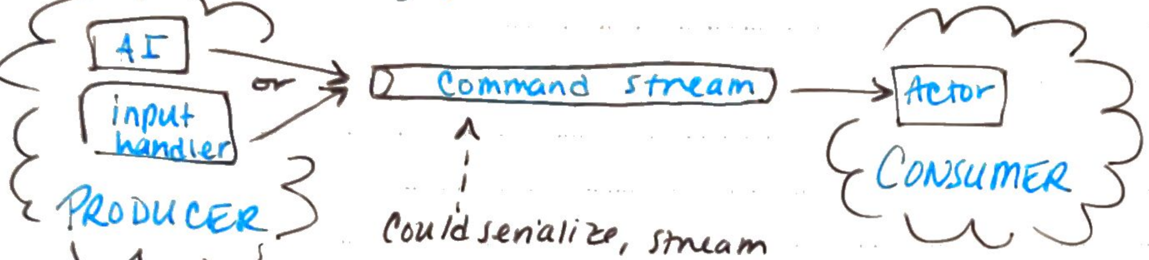
* Game Actor is our game obj. that represents a character in the gameworld, pass it into execute() & do actions on the actor

④ change handleInput() to return commands
- we can delay when the call is executed

⑤ take returned command & pass in actor to do action on

"we can let the player control any actor in game by changing the actor we exec. cmds on."

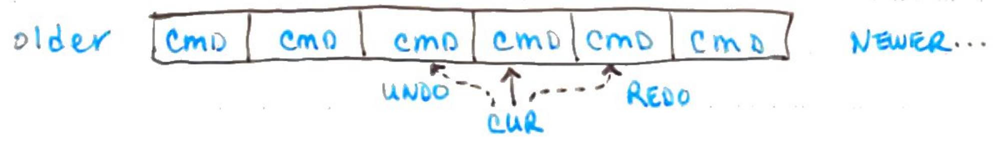
*can now use the same pattern for AI controlling the other players
"By making the commands that control an actor first-class objs, we've removed the tight coupling of a direct method call."



Could serialize, stream over network & then replay for making a networked ∞ player game

ex Undo + Redo

* previous examples a command is a reusable object that represents a thing that can be done.
* Now they represent a think that can be done at a specific point in time
→ add state to the class of where the unit was previously
NOTE: you could also use a persistent data structure where you store a ref. to the object before any changes. Restore the object if you undo
> or if you want ∞ undo/redo, add action to a list w/ "current" designation on the most recent action & move down/up the list based on undo/redo



*if they choose a new cmd after undoing some, everything in the list after the current command is discarded
*if you can use closures, DO IT. If it makes sense, sometimes closures can be so automatic it's hard to see what state they're holding
"For me, the usefulness of the Command pattern really shows how effective the fxnal paradigm is for many problems."
SEE ALSO: Subclass Sandbox, Chain of Responsibility, Flyweight
"friends don't let friends create singletons"?