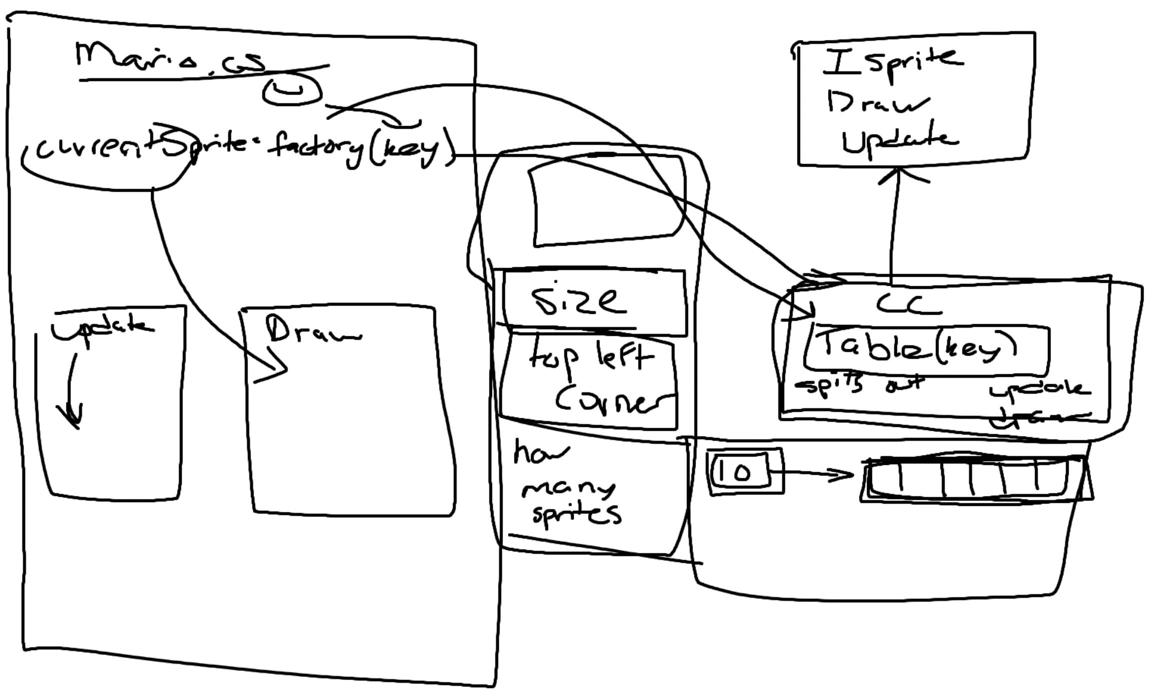
Basically, we will have a sprite factory that uses a map to determine what sprite needs to be returned. We can map keys (most likely integers) to a new object that Will and I will create called something along the lines of “SpriteInformation”. This SpriteInformation object would be instantiated with all of the information needed to rip from a sprite sheet and animate the sprite itself: things like the starting x and y positions, and the constants to iterate by.

If we set up a state machine for all of Mario’s actions, then part of the transitions (which will be initiated by commands) will be calling our sprite factory with a key that gets the SpriteInformation needed for that state’s sprite.

We will have an ISprite interface with a single implementing concrete class that has the map hard coded into it. This way all we need to do to add a new returnable sprite is add another key and SpriteInformation object to the map. If this works as we want it to, then during Mario’s state changes he could call the sprite factory that would return the sprite for his next state. This sprite object would be set to a private variable in the Mario.cs object called _currentSprite, which would be an instance of the ConcreteSpriteClass that implements ISprite (where the map is hardcoded).



Table

key(1)

✓

Keys / SF	
1	new SpriteInfo(10, 2, 3)
2	
.	
.	
.	
.	
n	