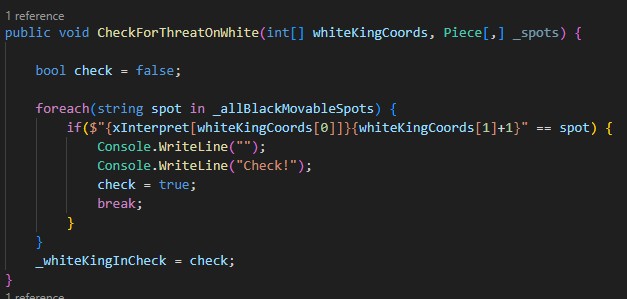
1. **Abstraction**

**Definition:** Abstraction is a practice used in programming to simplify complex blocks of code into an easy-to-use method.

**How It is used in my program:** My program has lots of abstracted methods, because most of them require lots of lines of code. One specifically, needs to be called every iteration to check if the King is threatened by a piece:



Instead of writing those lines I just made it into a single method, that can be called in an abstract way:



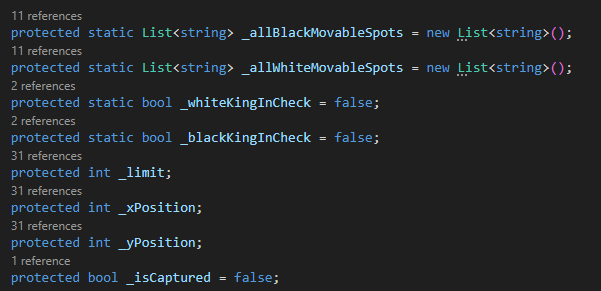
My program has abstracted methods for all of the classes.

**How did using these principles help my project to become more flexible:** Whenever I want to change the behavior of my methods, I can do it in a few lines of code, I can easily remove and add more lines to the methods, and whenever I can the function, it will still be simple.

1. **Encapsulation**

**Definition:** Encapsulation is the process of manipulating the access of the attributes and methods of a program, so you have control over which class has access to what. This is used so you always know where to find a problem, because you know exactly who has access to what.

**How it is used in my program:** My program uses access modifiers like ‘protected’ and ‘private’ to the attributes, and they are only accessible in the class or child classes. If someone wants to access values from those attributes, there are many public methods to access them:





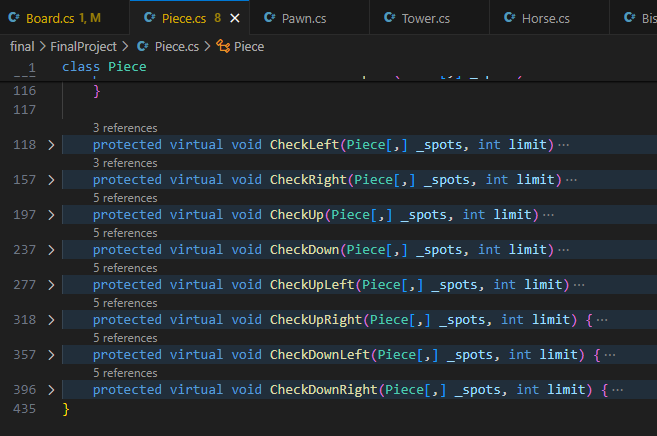
**How did using these principles help my project to become more flexible:**

Whenever an error comes when developing, I always know where to fix, because only the classes have direct access to the attributes. When implementing stuff, it is very easy to add new variables and methods, because they’re all connected in the same class, and I can keep their access private for encapsulation.

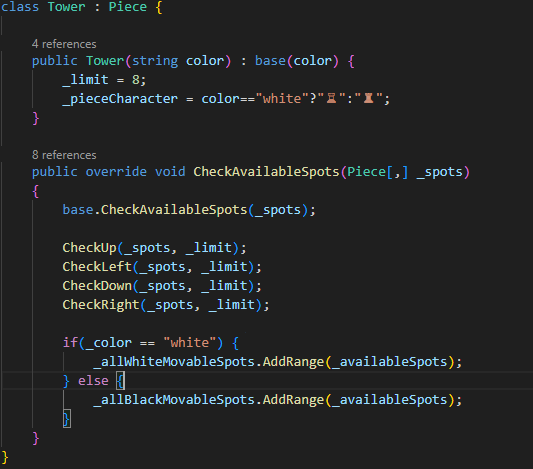
1. **Inheritance**

**Definition:** Inheritance is when a class exists as a parent, and is able to pass its characteristics and behavior to other classes, that will be its child classes. This practice is used to create many classes that have shared behavior, but are not necessarily the same.

**How it is used in my program:** This technique made my program possible, because there are many entities that had to be created that generally are the same thing, but had different behaviors. I’m talking about the Chess Pieces, they all move, they all have to know where they’re supposed to move, they all have an assigned color, but it’s not the same. Because of that, I created a Piece class to hold all of the general characteristics:



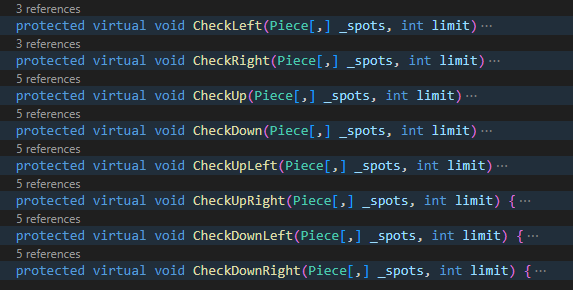
And created the other child classes that inherit its behavior, like the Tower class:



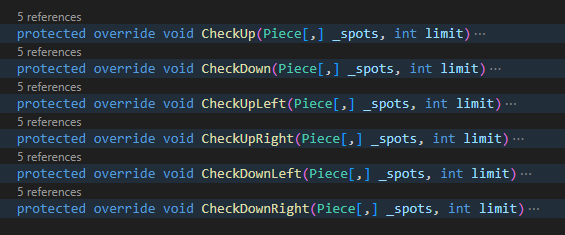
**How did using these principles help my project to become more flexible:** It made my program a lot more flexible because if I ever need to add additional behavior to my classes, I can do it in one place, and if I want to add behavior to a child class, like animations, it also becomes very simple.

1. **Polymorphism**

**Definition:** Polymorphism is a technique in program that allows a piece of code to have different behavior, depending on the context. It is used with Inheritance, when a child class inherits a method from its parent, but doesn’t necessarily need the exact same behavior.

**How it is used in my program:** There was one very useful way that Polymorphism was used in my program. In the Piece Class, there are many methods to check different directions in which a piece can go. 

I decided to write it all on the parent class because pretty much all of the child classes share the same behavior, except for the Pawn class. Because the Pawn can’t capture another piece vertically, unlike all of the other pieces, I had to override some of those checking functions in the Pawn class:



This way, the pawn works correctly, thanks to polymorphism.

**How did using these principles help my project to become more flexible:** It made my program more flexible because I don’t have to be stuck to the parent’s class behavior, I can just override unwanted behavior and keep writing. This will be useful to add more complex behavior in the future, specially for the Pawn class, that has more rules that were not implemented in the game, and the King class, that also needs more implementations to work just like a web game.