Video Explanation:

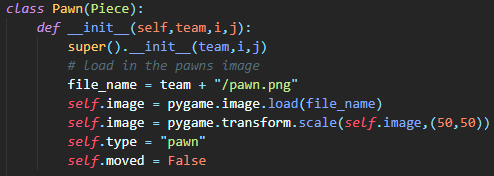
<https://youtu.be/N0fZL68oJ-4>

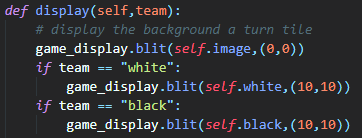
OOP concepts:

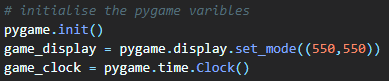
I used abstract classes for pieces. It allows me to have methods that are the same for all the different pieces while also having their own move methods. Inheritance as also used for all of the getters and setters because any of the pieces might need to use these methods.

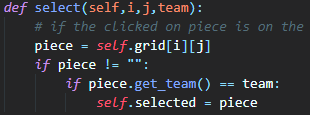
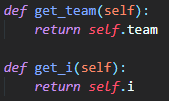
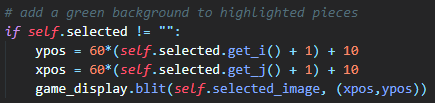
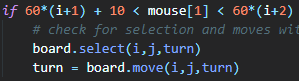
Overall the abstraction is redundant because a class only ever inherits from the piece class.

The super() method was used to initialise the subclasses of Piece. This is because they all share the same variables i, j and team but each needs a separate image and name.

The piece class could be made more useful if the piece name was passed up and all opening and resizing of images could be done by the super class which would reduce the amount of repeated code.



The pygame module uses class for its display and clock functionality. The classes are created at the start of the program and in this case they are global variables. They can be made as local variables but that can make it easy to duplicate the screen and break the pygame module. After the objects are created they have their own methods that can be called like any other methods with their arguments.

Simple methods and attributes were used to give the classes memory and functionality. The methods are public. The attributes were also public but used as if they were private because python does not have inbuilt private attributes.

Getters were used to retrieve attributes from outside of the scope of an object. They return simple values that are used in calculation and comparisons.

Methods are called from both outside of the classes using their objects but also from within a class calling its self-function. The self-function is also used for all attributes.