CS-500 Module Six Journal

Isaac Raymond

CS-500-11462-M01

Professor Ronak Nouri

Object-Oriented Programming in Game Design.

Object oriented programming allows us to model the real world in our program much more easily. Each item in a real world setting can be an object in our code that has, for example, in a game programming context, its own health, movement speed, etc. The properties of an object can easily be reused in another similar object in the game, can easily be modified when code is broken or not behaving correctly, it is easy to understand. These properties make object oriented programming almost essential in today's game programming world.

A video game might have many different types of human non-playable characters. They might all inherit some of the same type of code, as each human needs health, speed, perhaps weapons properties, etc.   There might be a class for all enemies, there might be a class for all friendly players, and we can also create a class for any human controlled players. Object oriented programming allows us to create properties inherited by each of these types of game objects so that we know what to expect when we modify them with their code. This makes the code much more maintainable in the long run. For example: A friendly NPC class might be

```
class FriendlyNPC:
        __init__(self, name, health, power):
              self.name = name
              self.health = health
              self.power = power

        def conversation(self):
```

```
        print(f"{self.name} has started talking to the main character.  Their current health
    is {self.power}.")
```

This object has name, health, and power as attributes. The method "conversations" can be edited to include custom conversations for each NPC. We can create an instance of FriendlyNPC:

```
npc1 = FriendlyNPC("Jeff", 100, 10)
npc1.conversation()
```

Object oriented programming allows for the idea of inheritance, where a class can get attributes and methods from another class. this can save code by reusing previous written, but also allows for easier readability because we know what to expect from an object that inherits another class. Characters that share attributes and methods can inherit from the same parent class.

Object oriented programming allows for encapsulation, as well, which can help to control the state of an object. Encapsulation ensures that the values of an object are only changed changed through predefined methods. This can reduce code writing errors and can help maintain the integrity of data in our running program. We can do this using double underscore to make one or all of the attributes private:

```
class FriendlyNPC:
    __init__(self, name, health, power):
        self.__name = name
        self.health = health
    self.power = power
```

The attribute "name" can now only be modified through provided methods and cannot be directly modified through other code.

Object oriented programming allows for polymorphism, a concept regarding different classes defining different methods with the same name but different behavior. If there is an NPC class, and FriendlyNPC inherits from that class, we can modify the inherited method "conversations" to make it unique to the FriendlyNPC but still keep the other inherited attributes and methods from the parent NPC class.

Object oriented programming leads to easier-to-edit code as it is organized in a more logical fashion. If the previously mentioned NPC class seems to work in some instances but not in FriendlyNPC objects, then we quickly know what part of the code needs modification, the part that's unique to FriendlyNPC. Inheritance allows for code reusability as well. Polymorphism allows for the unique edits that need to be made for each class that is inheriting from another class. There's a slight increase to performance overhead because of object oriented programming (code could be written to perform the same functions of object oriented programming, without using object oriented programming, and it would technically be less resource-intensive), but with today's machines, such overhead has minimal impact. Object oriented programming makes the creation of a program much faster by reusing code, making code easier to understand, and thus, easier to build on or bug-fix.