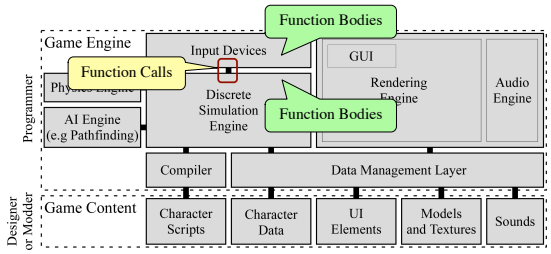
Computer Game Development

Credits: Planetfall (1983)

Steve Meretzky

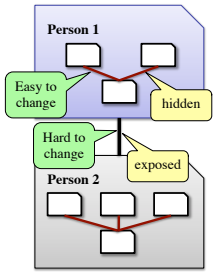
Credits: Portal (2007)

Challenge: Breaking Up Software



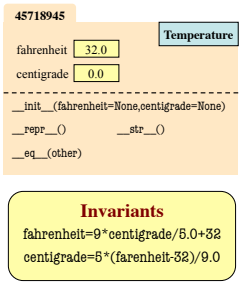
Encapsulation: Reducing Dependencies

- Development is iterative
  - You are always making changes (to improve your software)
- Coordination hurts iteration
  - Others are calling your functions
  - If you change how functions work, their code may no longer work
  - **Example**: Our test code in A1
- **Encapsulation**: limit what the other programmers can access in your code
  - If cannot access, changes are okay



Encapsulation is the Primary Purpose of Object Oriented Programming

- Applies to both code and data!
  - Turtles have a lot of data that you never, ever saw
  - Did you need to see it
  - Would it have been a good idea if you could have seen it?
- Encapsulation in Python
  - Make all data private
  - Force data access through the properties (getters/setters)
  - Or through the methods (see Assignment 6)



Encapsulation is the Primary Purpose of Object Oriented Programming

- Applies to both code and data!
  - Turtles have a lot of data that you never, ever saw
  - Did you need to see it
  - Would it have been a good idea if you could have seen it?
- Encapsulation in Python
  - Make all data private
  - Force data access through the properties (getters/setters)
  - Or through the methods (see Assignment 6)

```
class Temperature(object):
    _fahrenheit = 32.0
    # _centigrade = 0.0  NOT NEEDED!

    @property
    def centigrade(self):
        """Temp value in centigrade"""
        return 5*(self._fahrenheit-32)/9.0

    @centigrade.setter
    def centigrade(self,value):
        # Change fahrenheit instead
        self._fahrenheit=9*value/5.0+32
```


Interface vs. Implementation

Interface

- Unhidden methods/properties
- Specifications of the above

```
@property
def centigrade(self):
    """Temp value in centigrade"""
    return 5*(self._fahrenheit-32)/9.0
```

Difficult to change!

Implementation

- Hidden fields and methods
- Bodies of methods/properties

```
@property
def centigrade(self):
    """Temp value in centigrade"""
    return 5*(self._fahrenheit-32)/9.0
```

Easy to change

The Challenge of Making Software

```
def vignette(self):
    """Simulate antique lenses.

    Antique lenses had vignetting or corner
    darkening. This method darkens each pixel
    in the image by the factor
                (d / hfD)^2
    where d is the distance from the pixel to
    the center of the image and hfD (for half
    diagonal) is the distance from the center of
    the image to the corners."""
    rows = self.current.rows
    cols = self.current.cols
    # FINISH ME
```

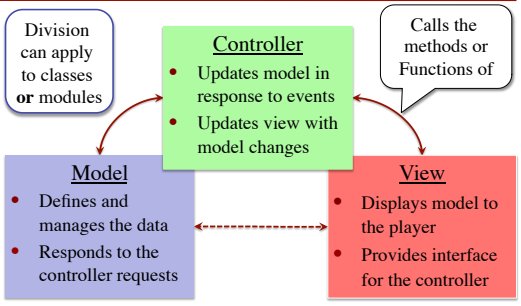
- We do a lot for you
  - Classes made ahead of time
  - Detailed specifications
  - You just "fill in blanks"
- The "Real World"
  - Vague specifications
  - Unknown # of classes
  - Everything from scratch
- Where do you start?

Software Patterns

- **Pattern**: reusable solution to a common problem
  - Template, not a single program
  - Tells you how to design your code
  - Made by someone who ran into problem first
- In many cases, a pattern gives you the interface
  - List of headers for the public methods
  - Specification for these public methods
  - Only thing missing is the implementation

Just like this course!

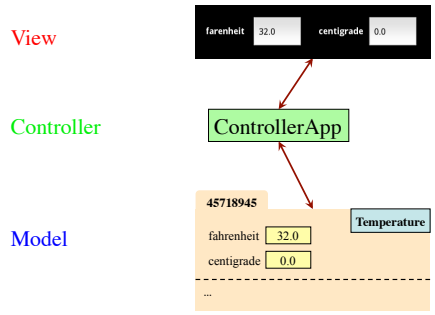
Model-View-Controller Pattern



TemperatureConverter Example

- Model: (`Temperature` in `model.py`)
  - Stores one value: fahrenheit
  - But the methods present two values
- View: (`TemperaturePanel` in `view.py`)
  - Constructor creates GUI components
  - Recieves user input but does not "do anything"
- Controller: (`ConverterApp` in `controller.py`)
  - **Main class**: instantiates all of the objects
  - "Communicates" between model and view

TemperatureConverter Example



MVC and Assignment 6