**Big Apples**

For bigger apples, we need to make several changes. First we need to tell the apple its size, so it can draw itself bigger. We also need to check for collisions with the worm anywhere in the apple. Finally, we need to place new apples so that they are entirely on the screen, and not on top of the worm.

**Tell the Apple Its Size**

When we create an apple, we’ll tell it how big it is. For this example, we’ll make apples that are 2 cells wide and 3 cells high.

In WormData.py, we’ll tell the apple its size. This is in the newGame() method. Change the line that creates the apple to add the dimensions to the Apple creation.

```python
self.apple = Apple(-1, -1, 2, 3)
```

We need the apple to expect these new parameters. This is in the Apple.py file. We change the __init__ method to receive these parameters by adding to the parameter list. Then we add 2 lines to store the width and height as part of the apple object.

```python
def __init__(self, x, y, width, height):
    self.width = width
    self.height = height
```

**Drawing the Apple**

To draw the apple with its new size, we need to change the draw() method in Apple.py. The apple is drawn as a rectangle. Previously it was drawn as a 1 cell by 1 cell rectangle. We’ll change the width and height by multiplying the width and height of the rectangle by the apple’s dimensions. Change the rectangle creation to look like this:

```python
rect = pygame.Rect(self.x*cell_size, self.y*cell_size, self.width*cell_size, self.height*cell_size)
```

**Collision Detection**

Now that the apple has multiple cell locations, we need to detect collisions for all locations. This requires the creation of a method in Apple.py. Add this method anywhere in the file, as long as it is not in the middle of any other method.

```python
def hit(self, position):
    x, y = position
    if ((self.x <= x and self.x + self.width > x) and
         (self.y <= y and self.y + self.height > y)):
        return True
    else:
        return False
```

This checks if the position (of the worm head) is anywhere inside the apple.

Now we need to use this method to check for collision with the worm head. In WormData.py’s evolve() method, find the collision detection. It looks like this:

```python
if self.player.getHead() == self.apple.getPosition():
```

Replace the line with this:

```python
if self.apple.hit(self.player.getHead()):
```

Notice the call to our new collision method.

**Placing Apples on Screen**
Previously, we only had to worry about the apple’s one position being on the screen when we place it. Now, we need to worry about all parts of the apple being on screen.

In Apple.py, we will add these two methods to make life easier:

```python
def getWidth(self):
    return self.width

def getHeight(self):
    return self.height
```

Now, in WormData.py, we need to make some changes to the apple placement. Find the placeApple() method. Make it look like this. We’ve added several lines, moved one line and changed several lines. Pay attention to all of the details.

```python
def placeApple(self):
    apple_width = self.apple.getWidth()
    apple_height = self.apple.getHeight()
    while True:
        x = random.randint(0, self.width-apple_width)
        y = random.randint(0, self.height-apple_height)
        self.apple.setPosition(x, y)
        # don't place the apple on the worm's body
        hit_worm = False
        for b in self.player.getBody():
            if self.apple.hit(b):
                hit_worm = True
        if not hit_worm:
            break
    return
```

Other

You figure out the actual apple size that makes the best game. I only made the example with 2x3 to demonstrate the difference between width and height. You might want a square apple. How would you do that? What if you are drawing the apple as an image? How do you make the collision and drawing code work together to give a consistent message to the user?