Robots Part 2

This part of the code was taught to Roger on September 26th.

Rocks and Robots

The first thing we want to do is add one robot and one rock to make the game world feel less lonely and work on the rules of the game. If the game works with one, it can work with more. This is induction!

Adding Rocks

After the code which creates the map in init() add this:

<Code:Python|Adding Rocks>

1. Add rocks.

self.gameMap[7][7] = '*' </Code>

There, we have added a rock to the map.

If you want to get fancy you can add it randomly (or add a second, random rock!) as follows:

<Code:Python|Alternate version>

1. Add rocks.

self.gameMap[7][7] = '*'

```
      # Add a rock randomly
      rx = random.randint(1,self.gameW)
      ry = random.randint(1,self.gameH)
      self.gameMap[ry][rx] = '*'
```

</Code>

Next, we need to make sure that if the player hits the rock, he will die.

To do this we add an elif to movePlayer(). Add the following elif to the if condition in movePlayer:

<Code:Python|rock collision in movePlayer()>

```
      if self.gameMap[to_y][to_x] == '#':
          return
      elif self.gameMap[to_y][to_x] == '*':
```

```
            self.killPlayer()
```

```
        self.px = to_x
        self.py = to_y
```

</Code>

For reference, here is a version that uses if-elif-else:

<Code:Python|version with elif-else>

```
        if self.gameMap[to_y][to_x] == '#':
            pass
        elif self.gameMap[to_y][to_x] == '*':
            self.killPlayer()
        else:
            self.px = to_x
            self.py = to_y
```

</Code>

It doesn't matter which version you use, pick the one which feels right to you.

In any case you will notice the killPlayer() function. This is new. Here it is:

<Code:Python|killPlayer()>

```
  def killPlayer(self):
      print("You have died! ¯\_(ツ)_/¯", end="")
      quit()
```

</Code>

There it is. Simple idea, simple execution. We just want something functional for now.

In one version I used GAME OVER instead of a shrug smiley, but I thought this way was a bit funnier.

Adding Robots

Adding robots is a bit trickier because they have to move intelligently on their own. But overall it is also a simple idea.

First let's add a robot. Here is what it looks like: Place this code after adding the rocks:

<Code:Python|adding a robot>

1. Add Robot

self.gameMap[15][12] = 'r' </Code>

This code looks deceptively simple but the idea behind it is very powerful. Consider briefly we could

also place the player on the map in this way as well. Why don't we do this? It could be a good question for after the game is finished. A) Keep special track off-map, or B) keep track of it on the map itself. Which method is better in which situations?

Moving the Robots

Moving the robots is easy. The robots always move after the player moves. Therefore, we modify checkEvents() like this:

<Code:Python|modify checkEvents()>

```
            if event.key == pygame.K_LEFT:
                self.movePlayer(self.px-1, self.py)
                self.moveRobots()
```

```
            elif event.key == pygame.K_RIGHT:
                self.movePlayer(self.px+1, self.py)
                self.moveRobots()
```

```
            elif event.key == pygame.K_UP:
                self.movePlayer(self.px, self.py-1)
                self.moveRobots()
```

```
            elif event.key == pygame.K_DOWN:
                self.movePlayer(self.px, self.py+1)
                self.moveRobots()
```

</Code>

As you can see, all we did is “move the robots after the player moves”. This is a simple idea in programming. When you want to do something, break it down in to a simple statement and write a function name which does it. Worry about writing the function later!

Okay so now let's write moveRobots()!

moveRobots()

Since we are keeping track of the robots on the map, we have to find the robots on the map and move them. Watch carefully how this is done:

<Code:Python|moveRobots()>

```
  def moveRobots(self):
      # 1. Find and move every robot.
      for x in range(self.gameW):
          for y in range(self.gameH):
              if self.gameMap[y][x] == 'r':
                  self.moveRobot(x,y)
```

</Code>

That seems simple enough! Find a robot on the map, and move it!

<Code:Python|moveRobot()>

```
  def moveRobot(self, x, y):
      # The robot wants to move towards the player.
      dx = self.px - x
      dy = self.py - y
```

```
      if dx != 0 and dy != 0:
          xory = random.randint(1,2)
          if xory == 1:
              dx = 0
          else:
              dy = 0
```

```
      if dx > 0:
          self.gameMap[y][x] = ' '
          self.gameMap[y][x+1] = 'r'
```

```
      if dx < 0:
          self.gameMap[y][x] = ' '
          self.gameMap[y][x-1] = 'r'
```

```
      if dy > 0:
          self.gameMap[y][x] = ' '
          self.gameMap[y+1][x] = 'r'
```

```
      if dy < 0:
          self.gameMap[y][x] = ' '
          self.gameMap[y-1][x] = 'r'
```

</Code>

The logic is simple. Find the distance between the robot and the player. But since the player can only move up, down, left and right we want to make it so the robots cannot move diagonally. The meaning of “xory” is x OR y. Meaning, if there is both dx and dy we zero one of them randomly so the robot only moves up, down, left or right in a single move.

The logic of moving it on the map is simple too. All we do is erase the old robot and draw a new one on the map at the new position.

But there is a big problem with this!

If we move the robot down (or to the right) then after the robot moves, the for loops in moveRobots() will detect the *new* robot at the *new* position and move him again!

This is a serious problem, but the solution is very simple. Just write a large R to the map for the new robot's position. Since the moveRobots() method looks for a small r, it will not move the large R. Then we can fix the problem later. Here is the new code:

<Code:Python|large R method>

```
  def moveRobot(self, x, y):
      # The robot wants to move towards the player.
      dx = self.px - x
      dy = self.py - y
```

```
      if dx != 0 and dy != 0:
          xory = random.randint(1,2)
          if xory == 1:
              dx = 0
          else:
              dy = 0
```

```
      if dx > 0:
          self.gameMap[y][x] = ' '
          self.gameMap[y][x+1] = 'R'
```

```
      if dx < 0:
          self.gameMap[y][x] = ' '
          self.gameMap[y][x-1] = 'R'
```

```
      if dy > 0:
          self.gameMap[y][x] = ' '
          self.gameMap[y+1][x] = 'R'
```

```
      if dy < 0:
          self.gameMap[y][x] = ' '
          self.gameMap[y-1][x] = 'R'
```

</Code>

There. Now the robots will only be moved once. We now have to repair this after we have moved all the robots:

<Code:Python|fixed moveRobots()>

```
  def moveRobots(self):
      # 1. Find and move every robot.
      for x in range(self.gameW):
          for y in range(self.gameH):
              if self.gameMap[y][x] == 'r':
                  self.moveRobot(x,y)
```

1. 2. repair the map

for x in range(self.gameW):

```
          for y in range(self.gameH):
              if self.gameMap[y][x] == 'R':
```

```
                    self.gameMap[y][x] = 'r'
```

</Code>

There! Now, after the robots have been moved, all the large R is changed back to small R. The idea here is to flag a robot in a special way. Since we cannot add a 'moved' variable, as the robots are not being stored in a class, or in a list which we could run through, this is a good way to 'flag' a robot as being moved.

Robots dying to rocks

There has to be a way to escape the robots. Therefore let us make the robots die when they hit a rock, just like the player.

We will change moveRobots() like this:

<Code:Python|robots dying to rocks>

```
      if dx > 0:
          self.gameMap[y][x] = ' '
          if self.gameMap[y][x+1] != '*':
              self.gameMap[y][x+1] = 'R'
```

```
      if dx < 0:
          self.gameMap[y][x] = ' '
          if self.gameMap[y][x-1] != '*':
              self.gameMap[y][x-1] = 'R'
```

```
      if dy > 0:
          self.gameMap[y][x] = ' '
          if self.gameMap[y+1][x] != '*':
              self.gameMap[y+1][x] = 'R'
```

```
      if dy < 0:
          self.gameMap[y][x] = ' '
          if self.gameMap[y-1][x] != '*':
              self.gameMap[y-1][x] = 'R'
```

</Code>

Now, if there is a rock at the position which the robot wants to move to, it will 'die' on the heap and add it's junk to that heap. Thus the player has a means of fighting the robots by exploiting their own stupidity.

Finally, since the player has a way to fight the robots, there should also be a danger; if the player touches a robot, he dies. Here, add this kill condition at the start of moveRobot():

<Code:Python|robots win>

```
  def moveRobot(self, x, y):
```

```
        # The robot wants to move towards the player.
        dx = self.px - x
        dy = self.py - y
```

1. If a robot touches a player, the player dies.

if dx == 0 and dy == 0:

```
            self.killPlayer()
```

1. (The rest of the code goes here)

</Code>

Now, we have a game! The robot can die, the player can die. In part 3 we will define a win condition and polish up the game a bit by adding some more special features!

Next

- Part 3