Robots Part 3

Goals

Here are our goals for today:

1. Finish the game!

This means we need a list of features we want to implement, and a list of bugs we need to fix. Then, we 'freeze' these lists and only work on adding the named features and fixing the named bugs.

Features

- Next-level feature
- Teleport Command

Bugs

- No win condition
- Robots not killing player
- Insta-leveling
- Teleport Bug

Bug #1: Robots not killing player

There is some problem where the robots move on top of a player. Also, robots should turn to stone if they hit each other. For this let's re-work the logic of moveRobot().

<Code:Python>

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

```
      if dx > 0:      dx = 1
      if dx < 0:      dx = -1
      if dy > 0:      dy = 1
      if dy < 0:      dy = -1
```

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

```
            dy = 0
```

```
    at = self.gameMap[y+dy][x+dx]
```

```
    if at == ' ':
        at = 'R'
```

```
    elif at == 'R' or 'r':
        at = '*'
```

```
    elif at == '@':
        self.killPlayer()
```

```
    self.gameMap[y][x] = ' '
    self.gameMap[y+dy][x+dx] = at
```

</Code>

Not shown is sn elif which says if a robot moves on to a * it becomes a *.

This logic is a bit cleaner and allows for more expansion later.

Bug #2: Insta-Leveling

When leveling, the robots would all instantly die. We traced this to a bug where in the initialization of the map, a '' (nothing) was being added instead of a ' ' (space). We could fix this in the moveRobots logic, but, it really should be fixed in the initialization. The map area must 'exist', and rules must be applied and relied on. It's not the function of moveRobot() to check map sanity.

Bug #3: No Win Condition

This means that when we were searching the map, we were doing it in the wrong place. Here is the new code:

<Code:Python>

```
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'
```

```
        if self.countRobots() == 0:
            self.level = self.level + 1
            self.makeLevel(self.level)
    def countRobots(self):
        robots = 0
```

```
        for x in range(self.gameW):
            for y in range(self.gameH):
                if self.gameMap[y][x] == 'r':
                    robots = robots + 1
        return robots
```

</Code>

As you can see, moveRobots() has been updated to be a bit more logical, and an explicit check for robots has been designed. This could be made more efficient but it would require mixing the logic of countRobots into something unrelated, which would make the code more difficult to understand. It's not a good practice so we will avoid refactoring this code prematurely.

Bug #4: The "Teleport Bug"

Before we deal with the teleport bug, let's discuss the new code for leveling and then the teleport command itself. If the bug is not obvious by the end of this, we'll explain it then.

Feature #1: The New Level function

<Code:Python> class Game:

```
    def __init__(self, window):
        self.window = window
        self.screen = window.screen
        self.logo = window.logo
        self.font = window.font
```

1. Clear the screen.

self.screen.fill[1)]

1. Set up game variables

self.running = True

1. Set up level 1.

self.level = 1

```
      self.makeLevel(self.level)
```

```
  def makeLevel(self, level):
      # create map
      self.gameW = 60
      self.gameH = 21
      self.gameMap = [[' ' for _ in range(self.gameW)] for _ in
range(self.gameH)]
```

1. add walls

for x in range(self.gameW):

```
          self.gameMap[0][x] = '#'
          self.gameMap[self.gameH-1][x] = '#'
```

```
      for y in range(self.gameH):
          self.gameMap[y][0] = '#'
          self.gameMap[y][self.gameW-1] = '#'
```

1. put player in a random place.

self.px = int(self.gameW / 2) # random.randint(1, self.gameW-2)

```
      self.py = int(self.gameH / 2)        # random.randint(1, self.gameH-2)
```

1. Add rocks.

numRocks = int(level / 2) + 1

```
      for x in range(numRocks):
          rx = random.randint(1, self.gameW-2)
          ry = random.randint(1, self.gameH-2)
          self.gameMap[ry][rx] = '*'
```

1. Add Robots

numRobots = level

```
      for x in range(numRobots):
          rx = random.randint(1, self.gameW-2)
          ry = random.randint(1, self.gameH-2)
          self.gameMap[ry][rx] = 'r'
```

</Code>

So, essentially, all of the “board setup” and map initialization features are moved into a newLevel() function. Simple concept. This is called after the check in moveRobots().

Bug #4: The Teleport Bug (Again)

What's the teleport bug? The teleport bug might have shown up earlier as a rock appearing in the middle of a wall. Then, a -2 would be added ex. (self.gameW-2) as a bound for randomly determining the rock's position. But the teleport bug is especially onerous as it could cause the player to teleport directly onto a stone or a robot. Thus we must make a new function which finds a free spot on the board, and use this instead of blindly picking a random map space.

First add this to the keyboard handler checkEvents(): <Code:Python>

```
            elif event.key == pygame.K_t:
                self.teleportPlayer()
```

```
            elif event.key == pygame.K_q:
                print("Game quit on level " + str(self.level))
                quit()
```

</Code>

We threw in a quit command for free. Now we'll add the teleport function as an example of what to do.

<Code:Python>

```
  def teleportPlayer(self):
      (self.px, self.py) = self.findFreeSpace()
```

</Code>

Actually, as you can see, the code is more compact and probably a bit more orderly or easy to read than getting two random numbers. **In every case where we must add something randomly to the map, we must use a call to findFreeSpace() or there is a chance it will be randomly placed on a pre-existing addition.**

Let's look at findFreeSpace() now:

<Code:Python>

1. Note that this function can lock if there are no free spaces!
    1. We don't check for this because long before this could happen
    2. a situation will be created where the player will always die after
    3. his first move.

def findFreeSpace(self):

```
      ok = False
      while ok == False:
          rx = random.randint(1, self.gameW-2) # 1 and -2 are wall bounds
          ry = random.randint(1, self.gameH-2) # so we dont land on a wall
          if self.gameMap[ry][rx] == ' ':
```

```
                ok = True

      return (rx, ry)
```

</Code>

At this point the game is essentially finished. There is no point in a save or load game function, athough one could theoretically be added. This will be left as an exercise for the reader, if interested. In such a case one must decide between only saving the level or saving the entire map state as well.

1)

0, 0, 0