Aula 4: Oficina de Programação e Robótica

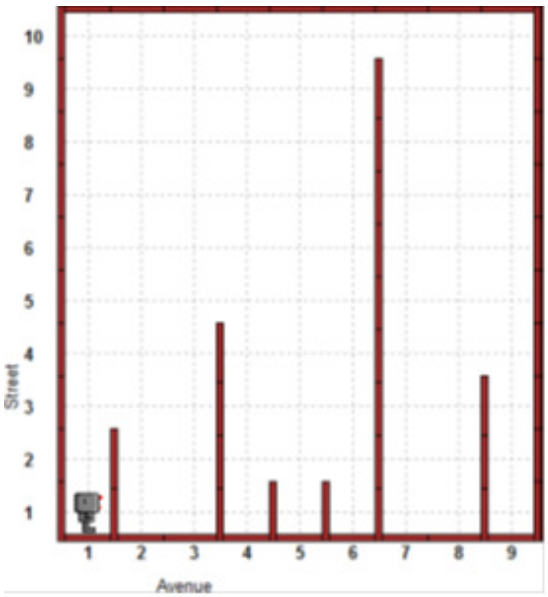
- O que vimos na última aula?
  - Indentação
    - São vários níveis. Geralmente contamos de zero, a partir da esquerda.
    - Ressalta a hierarquia entre os elementos do código.
    - Para algumas linguagens de programação, é apenas uma forma de organizar o código. No nosso caso, ela é essencial para que nossos programas funcionem corretamente.
    - Nos ajuda a identificar os blocos de comando.
  - Bloco de comando
    - É um conjunto de comandos afins, ou seja, que têm algo em comum.
    - Pode conter sub-blocos (blocos de comando aninhados).
    - Todas as linhas dentro do mesmo bloco possuem nível igual ou maior de indentação, nunca menor.
    - Começa com um cabeçalho.
  - Cabeçalho
    - No nosso caso, é uma linha que termina com ":".
  - Decomposição de problemas complexos em problemas mais simples.
    - Importância da criação de métodos.
    - Importância do código ser o mais genérico possível.
    - Importância de se usar termos de fácil compreensão.

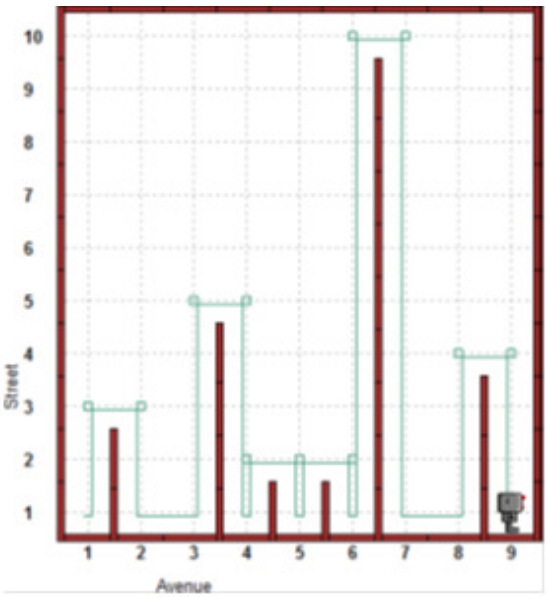
- Vocês já perceberam que, além do computador ser capaz de entender seu programa, também é importante que as pessoas sejam capazes de compreender o que você escreveu? Mais uma coisa que facilita isso é colocar ***comentários*** ao longo do seu código, explicando trechos mais complexos, ou o que cada método faz...
- Existem 2 tipos de comentários: ***comentários de linha*** e ***comentários de bloco***. No mundo do Reeborg temos apenas comentários de linha, com a seguinte sintaxe (regras de construção de uma frase e da sua disposição num discurso):

  # *esse é um comentário*

- Ou seja, o comentário vai do caracter # até o final da linha (por isso se chama comentário de linha). Eles podem estar numa linha isolada, ou logo depois de um comando do código.
- Comentários são ignorados pelo computador, só fazem diferença para pessoas.
- É recomendável colocar comentários logo no início de todos os seus programas, dizendo o nome do arquivo, o nome do programador, a data, e uma pequena explicação do que o programa faz. O quanto será explicado, e com que profundidade, você é quem decide. Mas lembre-se de que um comentário tem de ser útil para qualquer pessoa que venha a lê-lo!

- Exercício: Escalando Paredes (escala_paredes.wld).
  - O mundo tem sempre 10 avenidas, o que permite haver no máximo 9 paredes.
  - A altura das paredes é variável.
  - O programa deve funcionar para qualquer quantidade de paredes, de 0 a 9.
  - Veja o antes e o depois de um exemplo (com 9 em vez de 10 avenidas) nas figuras a seguir.





- O programa básico:

```
for i in range(9):
    if front_is_clear():
        move()
    else:
        escala_parede()
turn_off()
```

- Mas o que é o método ***escala_parede()***? O Reeborg já sabia escalar paredes? Na verdade, ele ainda não sabe. Nós é que vamos escrever o método ***escala_parede()*** daqui a pouco, mas já estamos supondo que ele existe para facilitar o entendimento e a construção da solução. Observe que o nome escolhido foi ***escala_parede()***, e não xyz(). Esse é um exemplo prático de como é bom que os elementos dos nossos códigos tenham nomes significativos, ou seja, que transmitam sua intenção.

- Vamos escrever então o método ***escala_parede()***...

```
def escala_parede():
    sobe_parede()
    desce_parede()
```

- Aqui fizemos o mesmo: supomos a existência de métodos que ainda serão escritos para facilitar nosso trabalho.

- Agora vamos escrever os métodos sobe_parede() e desce_parede()

```
def sobe_parede():
    turn_left()
    while not right_is_clear():
        move()
    virar_a_direita()

def desce_parede():
    move()
    virar_a_direita()
    while front_is_clear():
        move()
    turn_left()
```

- O método **virar_a_direita()** já é nosso conhecido:

```
def virar_a_direita():
    for i in range (3):
        turn_left()
```

- E, por fim, temos o programa completo:

```
def virar_a_direita():
    for i in range (3):
        turn_left()

def escala_parede():
    sobe_parede()
    desce_parede()

def sobe_parede():
    turn_left()
    while not right_is_clear():
        move()
    virar_a_direita()

def desce_parede():
    move()
    virar_a_direita()
    while front_is_clear():
        move()
    turn_left()

for i in range(9):
    if front_is_clear():
        move()
    else:
        escala_parede()
turn_off()
```

Mas o que esse código tem de diferente do meu???

Muitos de vocês, se não todos, conseguiram fazer esse exercício em aula, mas de uma forma em que o código só funciona para exatamente 9 paredes. O enunciado, porém, pede que ele funcione para qualquer quantidade de paredes, de 0 a 9. Essa versão que estamos enviando funciona da forma correta. **Queremos que vocês pensem sobre isso, para discutirmos na próxima aula: o que faz com que o código de vocês não funcione exatamente como foi pedido?**

Por que o meu robô fica dando voltas no mundo pra sempre, em vez de parar na esquina da rua 1 com a avenida 10???

Percebemos esse comportamento nos programas de alguns de vocês. Quem teve esse problema, conseguiu resolvê-lo? **Também queremos que vocês nos digam, na próxima aula, o que estava acontecendo no código para gerar este comportamento no robô.**

Independentemente das particularidades do código de cada um, esse problema é muito típico em laços de repetição (loops). Vocês se lembram de que, quando usamos o ***for***, determinamos explicitamente a quantidade de repetições? E quando usamos o ***while***, o que diz para o computador que já é hora de sair daquele loop? Isso é indicado pela **condição** avaliada no ***while***: enquanto ela for verdadeira, seu bloco de comandos é repetido; quando for falsa, o loop para e o programa segue a diante, executando o que está logo abaixo do bloco. E se a condição nunca se tornar falsa? O loop nunca acabará, certo? Isso é o que chamamos de **loop infinito** (ou **laço infinito**).