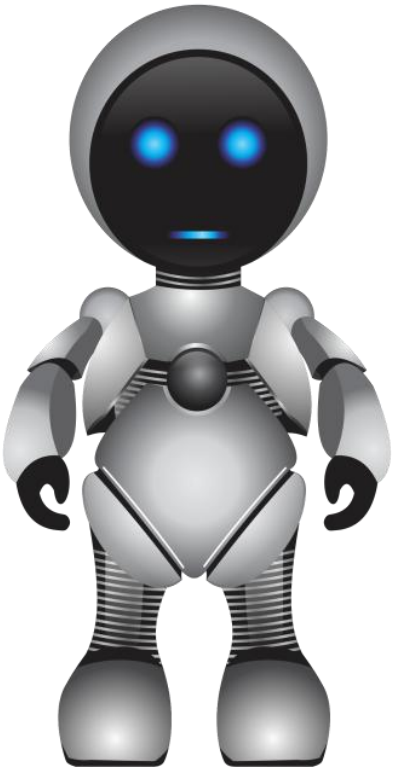
Documentação PROTO



Luigi da Silva Garcia mr.garcialuigi@gmail.com

Data: 11/2/2013

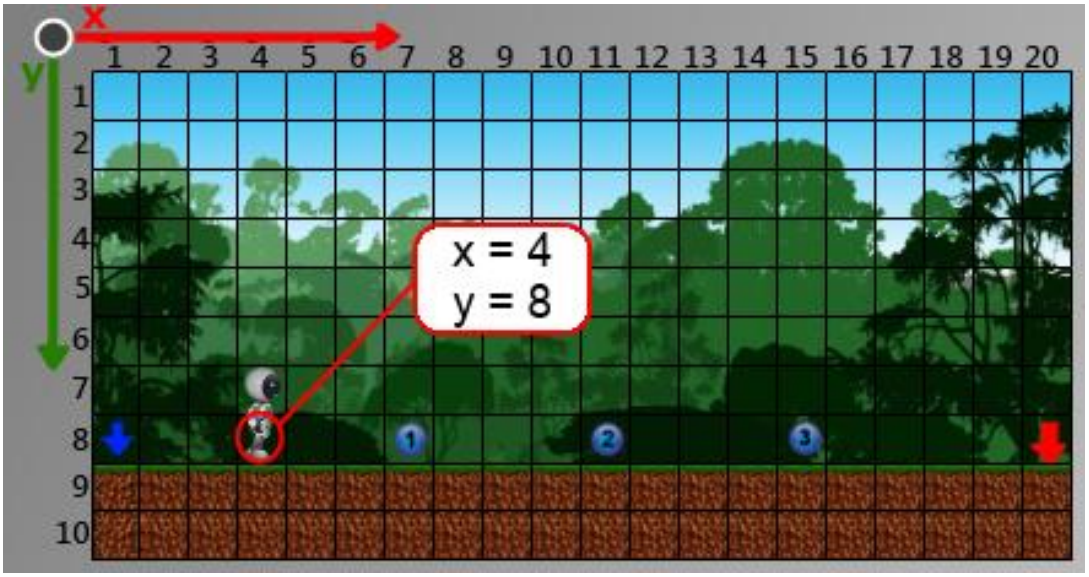
Classe Proto

A classe Proto possui todos os comandos e atributos para manipulação do robô.

Proto.x e Proto.y

Retornam um valor inteiro das coordenadas do robô, tanto x quanto y. Não tem efeito atribuir valores a estes atributos, são apenas de leitura.

A referência das coordenadas é sempre a metade de baixo do robô, já que ele ocupa 2 quadros.

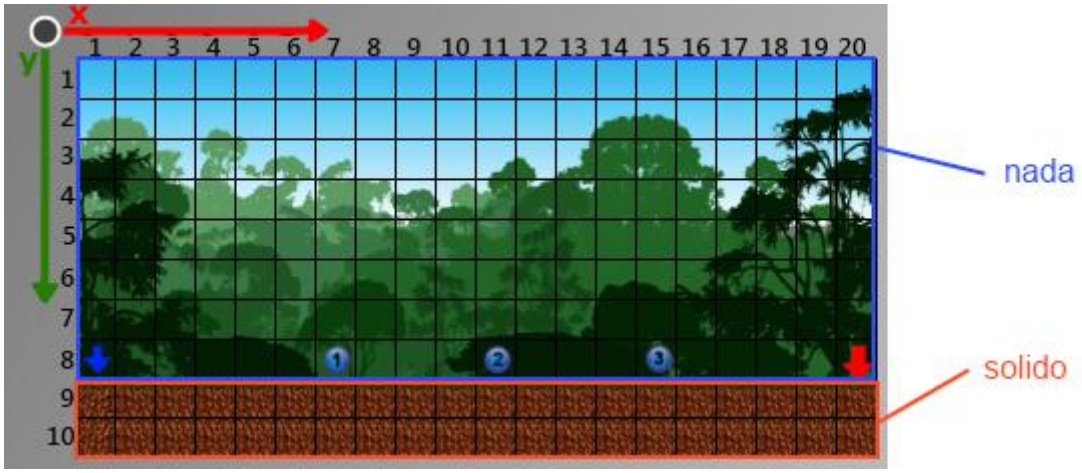


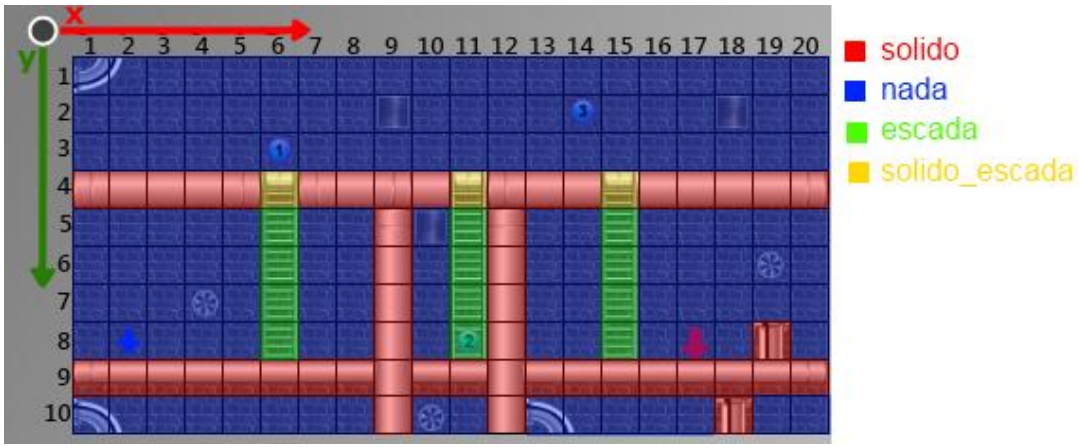
Proto.escanear(int x, int y)

Esse é o principal e mais usado comando de Proto. O robô tem acesso a um mapa do cenário através desse comando, que retorna o que tem posicionado em uma dada coordenada.

Os retornos do comando podem ser: nada, solido, escada, solido_escada.

Exemplos:





Observação: o “solido_escada” é usado tanto como um solido(andar, pular, colidir) como uma escada. Normalmente o robô identifica o “solido_escada” para usar o comando de descer escadas.

O comando Proto.escanear(int x, int y) normalmente vem dentro de um bloco if, para fazer testes e saber o que tem no cenário.

Exemplo:

```
If(Proto.escanear(5, 8) == solido){
    // ação aqui
}
```

Isso pode ser usado com referência as posições do robô, para criar um campo de visão.

O código a baixo verifica sempre se existe algo um quadro a frente e em baixo do robô, para assim poder andar.

```
If(Proto.escanear(Proto.x +1, Proto.y + 1) == solido){
    Proto.moverParaDireita();
}
```

Você verá mais sobre os comandos do robô como o de mover nos próximos tópicos.

Proto.moverDireita()

Esse comando faz com que Proto caminhe 1 quadro para a direita, ou seja, se a posição dele estiver em x=4 e y=8, ele caminha até x = 5 e y = 8.

Proto só pode caminhar em lugares sólidos, e caso contrário ele cai.

Proto.moverEsquerda()

Esse comando faz com que Proto caminhe 1 quadro para a esquerda, ou seja, se a posição dele estiver em x=4 e y=8, ele caminha até x = 3 e y = 8.

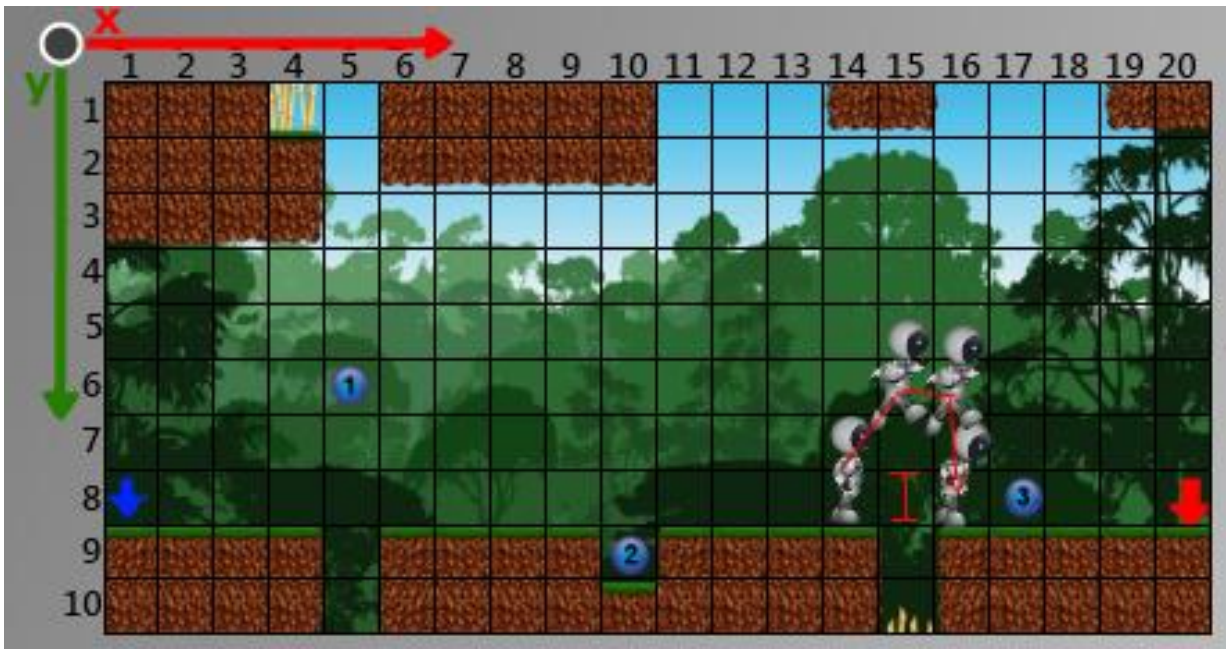
Proto.pularDireita()

Ao executar esse comando, Proto pula 2 quadros pra frente, e atinge a altura suficiente para pular um quadrado de altura. Não é interessante que o robô bata em uma parede ao pular.

Algo sólido a baixo do pulo de Proto pode antecipar sua queda.

O pulo é uma técnica um pouco mais avançada, pode ser usada em diversas ocasiões, mas deve ser treinada.

A imagem a baixo mostra a trajetória do pulo do robô.

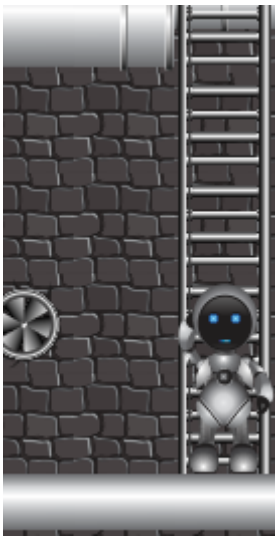


Proto.pularEsquerda()

Comando para o Proto pular para esquerda. Tem o mesmo funcionamento do pulo para a direita, mas para o outro lado.

Proto.subir()

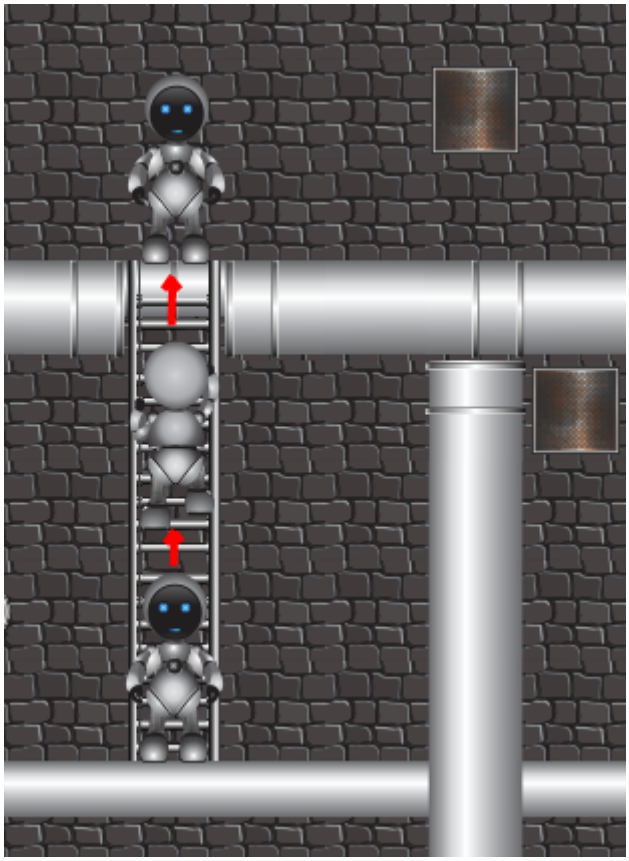
Esse comando faz com que Proto suba escadas. Elas não são sólidas, ou seja, o robô pode passar caminhando normalmente por onde tem uma escada posicionada, caso ele não queira subir. Para o Proto subir a escada, ele deve estar posicionado onde está o início da escada, como mostra a imagem.



Ou seja, se nas coordenadas x e y atuais de Proto tiver uma escada, então pode subir.

Ao ser executado o comando subir, Proto vai até o final da escada, e se posiciona em um sólido que tem no final dela. Esse sólido é uma mescla de sólido com escada.

A imagem a baixo mostra o trajeto que o robô percorre ao receber uma instrução para subir escada.

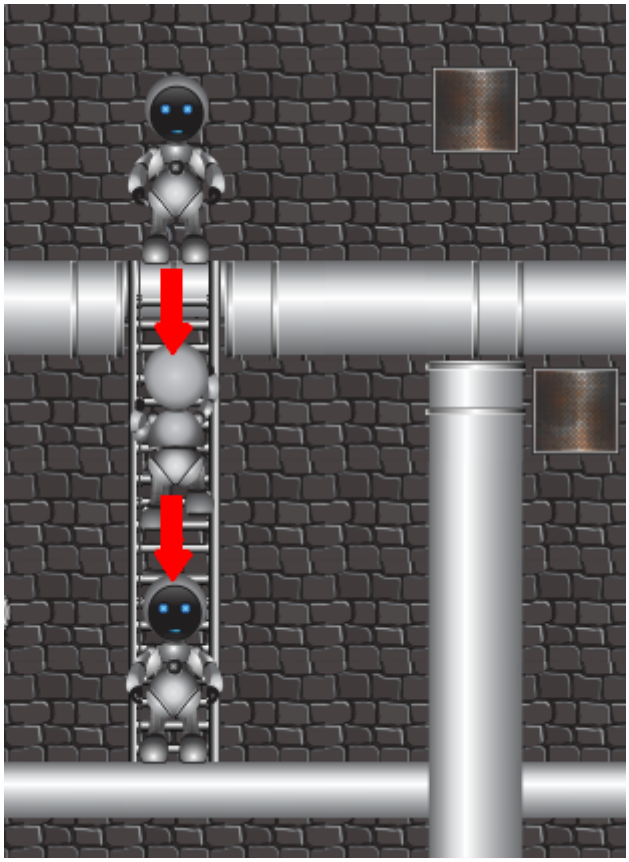


Ao tentar subir em um lugar que não tenha escada, o robô não esboça reação.

Proto.descer();

Para descer escadas, processo é parecido. Porém, a escada deve estar no mesmo x do Proto, mas 1 quadro de y a baixo. Essa escada é uma mescla de sólido e escada, onde ele pode tanto andar, quanto descer.

A trajetória é a mesma, ele desce até o final da escada.



Observação: É possível perceber que ao subir ou descer uma escada, o x do robô fica no mesmo lugar, o que pode resultar em o robô ficar subindo e descendo sem parar. Uma boa técnica para fugir disso seria uma flag, alguma maneira de identificar que ele acabou de subir/descer a escada, e então ativar uma fuga no próximo quadro, saindo para um dos lados. O algoritmo dessa sugestão seria o seguinte:

```
//-------------------------------------------------

se (flag == verdadeira){

        mover direita;

        flag = falsa;

}


senão se (escada) {
```

```
        subir();

        flag = verdadeira;

}

//--------------------------------------------------
```

Isso é apenas uma sugestão, você pode criar outras maneiras de resolver esse problema.

Conceitual: Sentido

Uma outra forma de programar a inteligência de Proto é utilizando o sentido. Esse sentido pode ser para direita ou para esquerda.

Esses sentidos podem facilitar, dependendo do algoritmo a ser utilizado. Por exemplo, se o sentido do robô estiver configurado para a direita, o comando **Proto.moverParaSentido()** vai fazer o robô andar para a direita(mesmo efeito do comando "Proto.moverParaDireita()"). Da mesma maneira, se o sentido tiver configurado para a esquerda, o comando **Proto.moverParaSentido()** fará ele andar para esquerda.

O sentido pode ser setado conscientemente, invertido ou até sorteado.

Proto.setarSentidoParaEsquerda()

Configura o sentido do robô para esquerda.

Proto.setarSentidoParaDireita()

Configura o sentido do robô para direita.

Proto.inverterSentido()

Esse comando pode ser bem útil, pois troca o sentido do robô. Se o sentido estiver para a esquerda, ele seta para a direita. Se estiver para a direita, ele seta para a esquerda.

Em uma varredura de cenário, é uma ótima técnica.

Proto.sortearSentido()

Esse comando sorteia um dos 2 sentidos possíveis, e seta ao robô. Também pode ser muito útil para alguns algoritmos.

Proto.moverParaSentido()

Esse comando tem o mesmo efeito dos outros de mover, porém ele age encima do sentido

Exemplo: Se estivar para a esquerda, ele funciona como o **Proto.moverParaEsquerda().**

Proto.pularParaSentido()

Funciona como os outros comandos que agem encima do sentido do robô.

Exemplo: Se estiver para a esquerda, ele funciona como o **Proto.pularParaEsquerda().**

Uma ação de cada vez

O robô vai executar o código a cada quadro. Esse código deve "decidir" o que o robô vai fazer em determinada situação. Ou seja, o código não são passos a serem seguidos, e sim comparações e testes que resultam em o robô tomar a decisão correta em cada situação. Exemplo: Se tiver sólido a frente, mover, se não, pular.

Caso o código leve á duas ações em apenas um quadro(o robô chegou no comando mover e no comando pular no mesmo quadro) ele vai sempre executar o último.

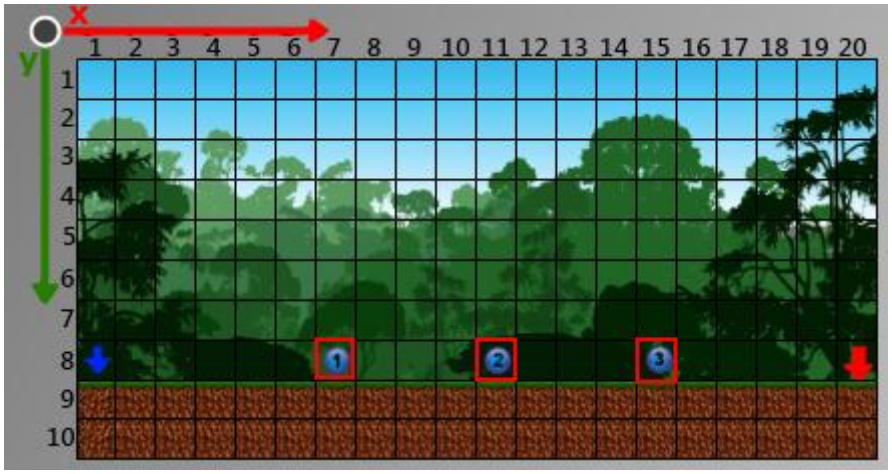
Um robô bem programado, tem a inteligência de tomar UMA decisão certa a cada quadro.

Classes Esfera1, Esfera2 e Esfera3

Essa classes tem as coordenadas das posições de cada esfera na fase. Todas as fases tem 3 esferas.

X e Y

É importante para o jogador saber o posicionamento das esferas no cenário.



Por exemplo:

O comando **Esfera1.x** retorna o valor 7.

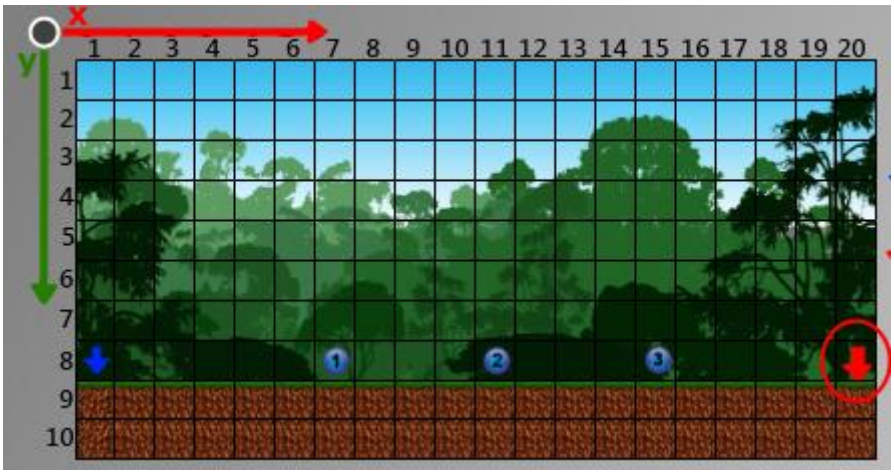
O comando **Esfera2.y** retorna o valor 8.

Classe Objetivo

A classe objetivo representa o final da fase, onde o robô deve chegar. Esta classe possui as coordenadas x e y da posição do objetivo no cenário.

X e Y

Objetivo.x e Objetivo.y retornam as coordenadas da localização no cenário.



O comando **Objetivo.x** retorna o valor 20.

O comando **Objetivo.y** retorna o valor 8.

Criação de variáveis

A interface de edição do código do robô no jogo possui um ambiente de criação de variáveis. Isso porque o robô executa aquele código por quadro, impossibilitando criação e reutilização de variáveis ali.

Então, essas variáveis criadas neste ambiente, ficam acessíveis dentro do código a qualquer momento.

Basicamente o que acontece ao executar o robô na fase, são as variáveis serem criadas antes, e depois o robô faz um loop infinito daquele código(executando o código a cada quadro).

A imagem a baixo mostra o ambiente de criação e exclusão de variáveis.