



**INSTITUTO FEDERAL DE EDUCAÇÃO,
CIÊNCIA E TECNOLOGIA GOIANO**
Campus URUTAÍ
NÚCLEO DE INFORMÁTICA
PROJETO DE INICIAÇÃO CIENTÍFICA



CAROLINNA ÉLIDA GOMES DE MORAIS

Programação Orientada a blocos: aprendendo a programar com o mBot

Orientador(a):

Cristiane de Fátima dos Santos Cardoso

Urutaí, 31 de julho de 2020



**INSTITUTO FEDERAL DE EDUCAÇÃO,
CIÊNCIA E TECNOLOGIA GOIANO**
Campus URUTAÍ
NÚCLEO DE INFORMÁTICA
PROJETO DE INICIAÇÃO CIENTÍFICA



CAROLINNA ÉLIDA GOMES DE MORAIS

Programação Orientada a blocos: aprendendo a programar com o mBot

Documento apresentado como parte integrante de projeto de iniciação científica do Instituto Federal de Educação, Ciência e Tecnologia Goiano - Campus Urutaí.

Orientador(a):

Cristiane de Fátima dos Santos Cardoso

Urutaí, 28 de fevereiro 2021

FOLHA DE APROVAÇÃO

CAROLINNA ÉLIDA GOMES DE MORAIS

Programação Orientada a blocos: aprendendo a programar com o mBot

Documento apresentado como parte integrante de projeto de iniciação científica do Instituto Federal de Educação, Ciência e Tecnologia Goiano - Campus Urutaí.

Aprovado em: 31/07/2021

BANCA EXAMINADORA



Profª. Dra. Cristiane de Fátima dos Santos Cardoso
Orientador(a)

Agradecimentos

Agradecemos ao IF Goiano pelo incentivo e auxílio na forma de bolsa PIBIC-júnior.

Resumo

A intensa necessidade de automação e até mesmo a curiosidade humana tem tornado crescente o uso de robôs e o estudo em robótica. Assim, a robótica tem conseguido um espaço cada vez maior na sociedade permitindo um contato bastante precoce: as pessoas têm tido interações com ela cada vez mais jovens. Muitos materiais estão disponíveis para auxiliar o estudante em seus primeiros passos, no entanto, a maioria destes materiais fazem uso de linguagens de médio/baixo nível, dificultando o aprendizado e limitando a criatividade. Assim, este material apresenta o uso de software para robô seguidor de linha, com uso de linguagem de programação orientada a blocos, mostrando aspectos do dispositivo e do software, ao passo que alguns experimentos. Para isso, são utilizados o kit educacional mBot, Makeblock e software mBlock.

CAPÍTULO 1: MATERIAL E AMBIENTE DE PROGRAMAÇÃO

1.1 Equipamento

O MBot é um robô educacional para iniciantes, que torna o ensino e a aprendizagem de programação de robôs simples e divertidos. Com apenas uma chave de fenda, é possível construir um robô a partir do zero e iniciar o aprendizado em programação, sendo que a proposta do mBot é a programação baseada em blocos. À medida que avançam, os estudantes podem aprender sobre uma variedade de máquinas robóticas e peças eletrônicas, e desenvolverão suas habilidades lógicas de raciocínio e design.

Figura 1: Imagem do robô Mbot.



O mBot, com seus grandes olhos adoráveis e carinha sorridente, encantou mais de 4,5 milhões de crianças em todo o mundo, o que o classifica como uma excelente ferramenta para brincadeira educacional e uma grande ajuda para professores nas suas aulas (STEAM), ele incentiva as crianças a exercitarem suas habilidades interdisciplinares e, ao mesmo tempo, proporciona diversão e prazer em criar. Além deste perfil pedagógico, o mBot também é um concorrente pesado frequente em grandes competições de robôs, como o MakeX.

1.2 Quais são as partes do mBot?

O mBot possui 4 portas de expansão e pode conectar-se a mais de 100 tipos de módulos eletrônicos. As crianças podem usar uma variedade de pacotes de complementos e mais de 500 peças na Plataforma Maker para criar uma variedade de formas imaginativas.

Como pode ser visto na Figura 2, o robô, basicamente, é dividido em cinco partes: as quatro portas de expansão; o Mcore Main Control Board que contém o sensor de luz e infravermelho; o Sensor Ultrassônico que detecta a distância e evita obstáculos; o Sensor seguidor de linha que segue as linhas automaticamente; o Bluetooth que facilita a conexão sem fio.

Figura 2: Imagem do robô Mbot e suas divisões.

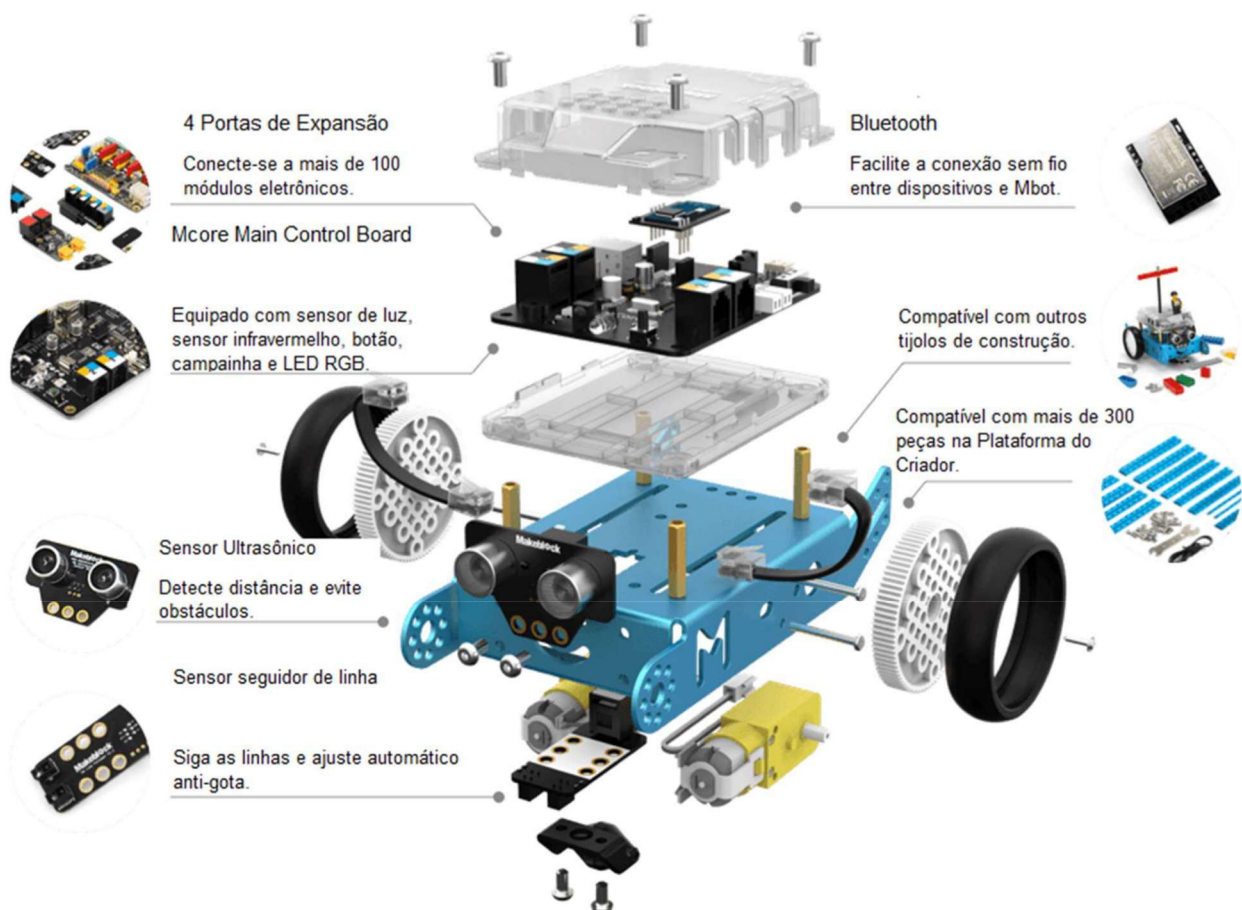
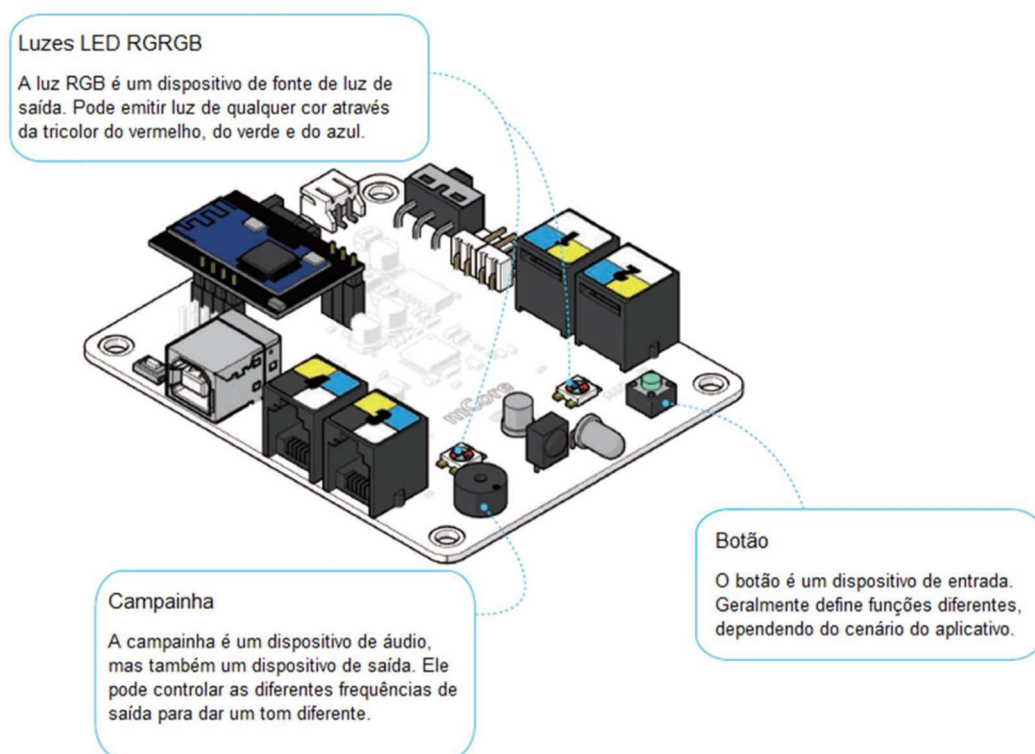


Figura 3: Imagem do Arduino.



O Arduino é uma plataforma de prototipagem eletrônica open-source que se baseia em hardware e software flexíveis e fáceis de usar. É destinado a artistas, designers, hobbistas e qualquer pessoa interessada em criar objetos ou ambientes interativos. O microcontrolador na placa é programado com a linguagem de programação Arduino, baseada na linguagem Wiring, que é mais complicada para a compreensão de crianças.

1.3 O software mBlock

O mBlock 5 é uma ferramenta de software versátil para o ensino, é inspirado no Scratch 3.0 e suporta linguagens de programação gráfica e textual. Até agora, quase 5,5 milhões de pessoas usaram o software como uma ferramenta para estudar, criar e compartilhar. Com o mBlock 5, as crianças são capazes de projetar histórias, jogos e animações atraentes e programar dispositivos de hardware como robôs Makeblock

e microbit. Além disso, o mBlock 5 suporta a linguagem Python, sendo possível alternar para o modo Python simplesmente com um clique.

Figura 4: Ícone do mBlock.



O mBlock 5 é equipado com recursos de Inteligência Artificial (IA) e Internet das coisas (Internet of Things – IoT) que permitem que as crianças se divirtam com algumas tecnologias de ponta.

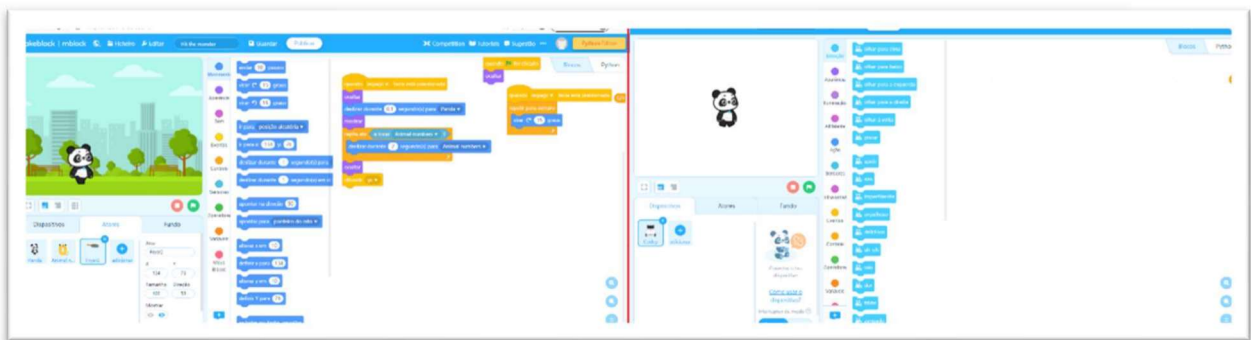
Links para download do mBlock 5:

- Para PC: <http://www.mblock.cc/software/>
- Para uso nos navegadores da Web: <https://ide.makeblock.com^#/>
- Para uso nos navegadores da Web: <https://ide.makeblock.com^#/>
- Para Android e iOS: pesquise “mBlock” em qualquer loja de aplicativos para fazer o download.

CAPÍTULO 2: INICIANDO NO AMBIENTE DE PROGRAMAÇÃO

O mBlock tem um aspecto divertido e bastante intuitivo, a Figura 5 mostra as versões de navegador e aplicativo para PC lado a lado. As cores e o próprio personagem apresentado inicialmente reforçam seu caráter educacional. O personagem é chamado de Sprite.

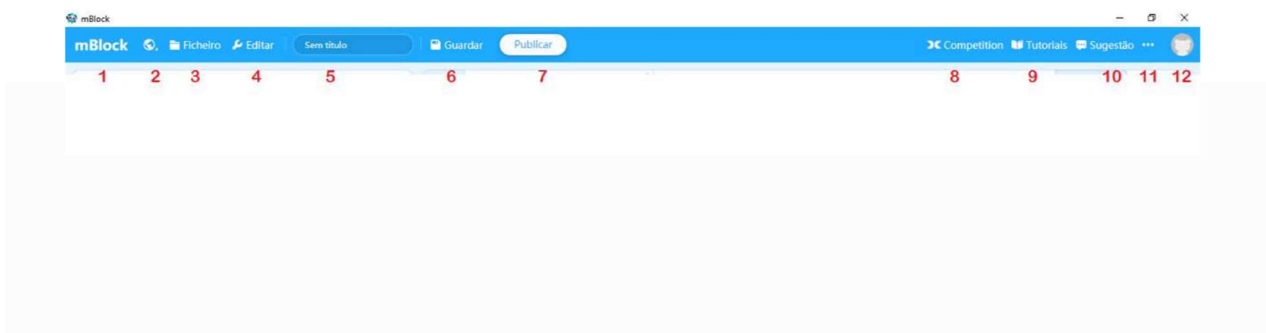
Figura 5: MBlock web e aplicativo para PC lado a lado.



2.1 Elementos do ambiente

Uma das primeiras coisas com a qual o usuário irá se deparar, é a barra de ferramentas na parte superior da tela. A seguir descrevemos cada um dos itens observados na Figura 6.

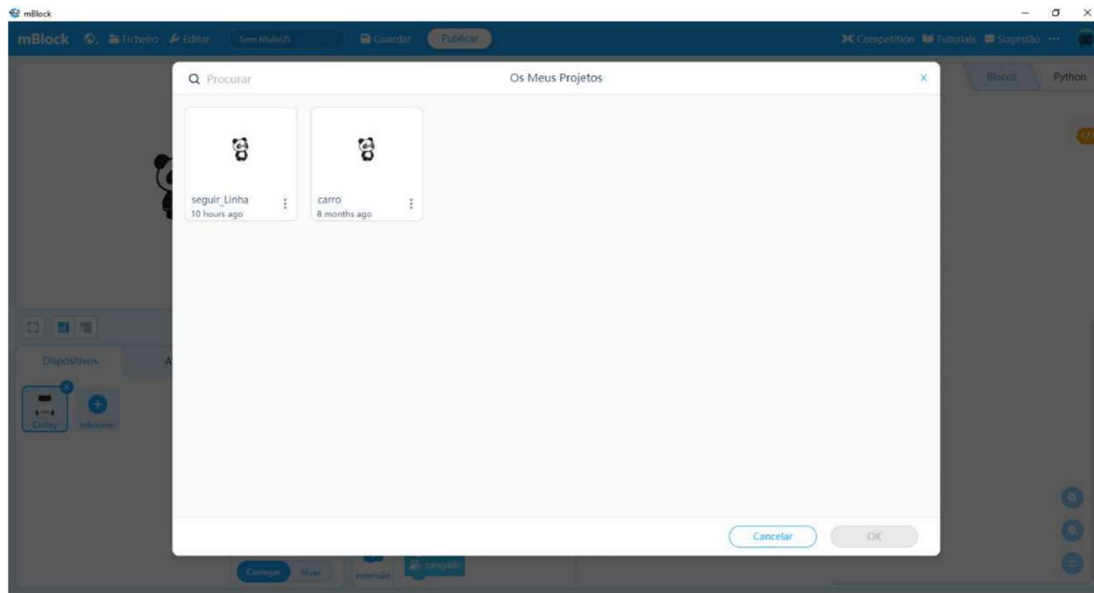
Figura 6: Barra de ferramentas.



1. Link para a plataforma do mBlock: este item quando clicado leva para o site <https://www.mblock.cc/en-us/>.
2. Idioma: você pode alterar o idioma do mBlock 5 aqui.
3. Ficheiro: a partir daqui você pode criar um novo projeto, compartilhar ou abrir um projeto que esteja salvo na plataforma do mBlock, sendo necessário possuir uma conta registrada no site para estas opções. Também é possível importar e salvar no computador.
4. Editar: neste item você pode acionar o modo turbo ou esconder/exibir o palco.
5. Título: clique para alterar o título do seu projeto de trabalho.
6. Salvar: clique para salvar seu projeto. Você pode salvar diretamente na sua conta mBlock registrada no site ou salvar no seu computador.
7. Publicar: publica o seu programa na plataforma makeblock <https://planet.mblock.cc>
8. Competition: vai para o link de competição de robôs <https://www.makex.io/en>
9. Tutoriais: exibe um guia do usuário e programas de exemplos, em que uma variedade de programas de amostra é fornecida aqui.
10. Sugestão: permite comunicar erros.
11. Opções adicionais (três pontinhos): permite procurar atualizações, ver informações sobre o mBlock e realizar cooperação com centros de treinamento.
12. Menu da conta logada: permite ver seus projetos, o perfil, as definições da conta, o código de auth na nuvem e sair, que é como se fosse uma senha criptografada da conta.

Para visualizar seus projetos clique no ícone da conta, e em seguida escolha o item “Os Meus Projetos” do menu, será aberta uma janela, conforme a Figura 7.

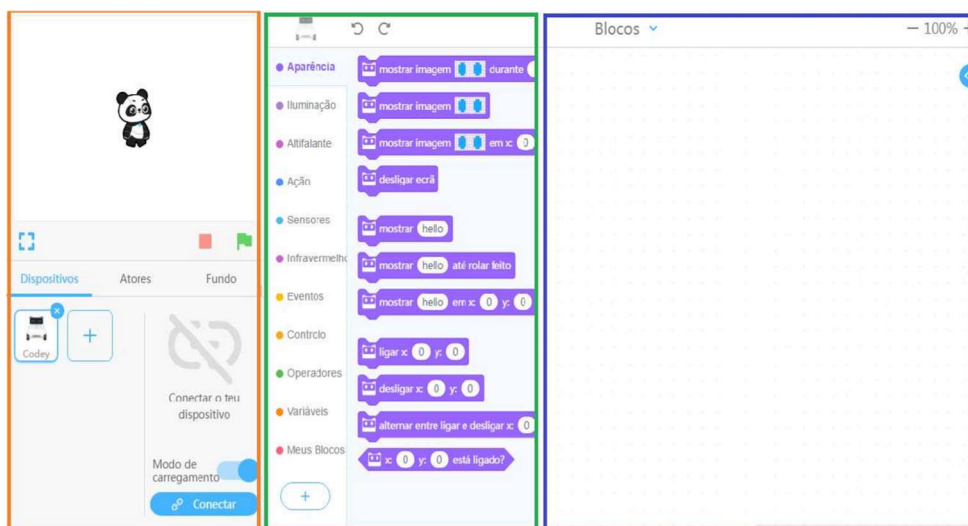
Figura 7: Página “Meus Projetos.”



2.2 A divisão do ambiente

O ambiente do MBlock 5 é dividido em três partes, que são o palco, a área dos blocos e a área de scripts.

Figura 8: Ambiente do MBlock 5 e suas partes.



Laranja: Área do palco
Verde: Área dos blocos
Azul: Área dos scripts

- Área do palco: aqui você pode visualizar a execução do seu projeto, conectar dispositivos, definir seus sprites e planos de fundo.
- Área de Blocos: aqui você pode encontrar os blocos necessários por categoria e cor para construir seus programas. As principais categorias são: aparência, eventos, operadores, ação, emoção, sensores etc.
- Área de scripts: aqui você pode programar simplesmente arrastando blocos da área de blocos para esta área.

2.3 Atores, dispositivos e fundo

Os Sprites, ou atores, são os personagens que executarão as ações. No mBlock é possível utilizar personagens como animais, pessoas, plantas, adereços, fantasias, comidas, edifícios etc. Um ator pode ser adicionado clicando na aba “Atores”, abaixo do palco, em seguida “adicionar”, a Figura 9 mostra a aba “Atores” e o botão adicionar. Feito isto, será aberta a janela da Figura 10, com as diversas categorias de atores. Após a escolha do personagem, deve ser construído um script para o mesmo, este script deverá conter ações, decisão, uso de sensores etc. Normalmente, um ator possui mais de uma “fantasia”, uma fantasia é uma variação no personagem, usada para simular movimentos de uma dança ou andar ou situações diversas, o Panda possui duas fantasias para simular que está andando, Figura 13(b) mas para produzir este efeito é preciso criar um script com o bloco que alterne de fantasia, logo falaremos disso.

Figura 9: Adicionando atores.

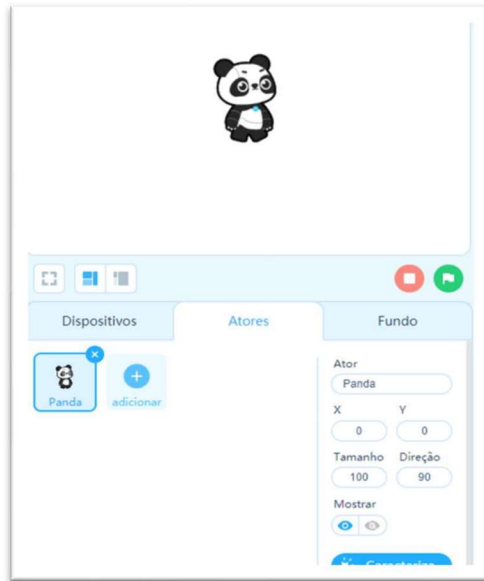
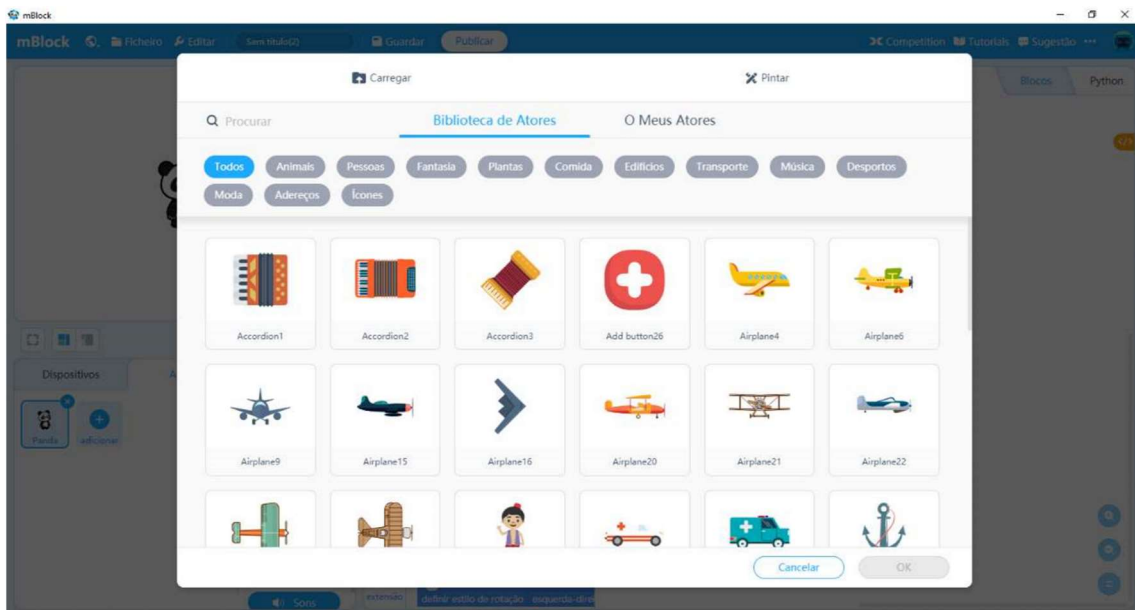


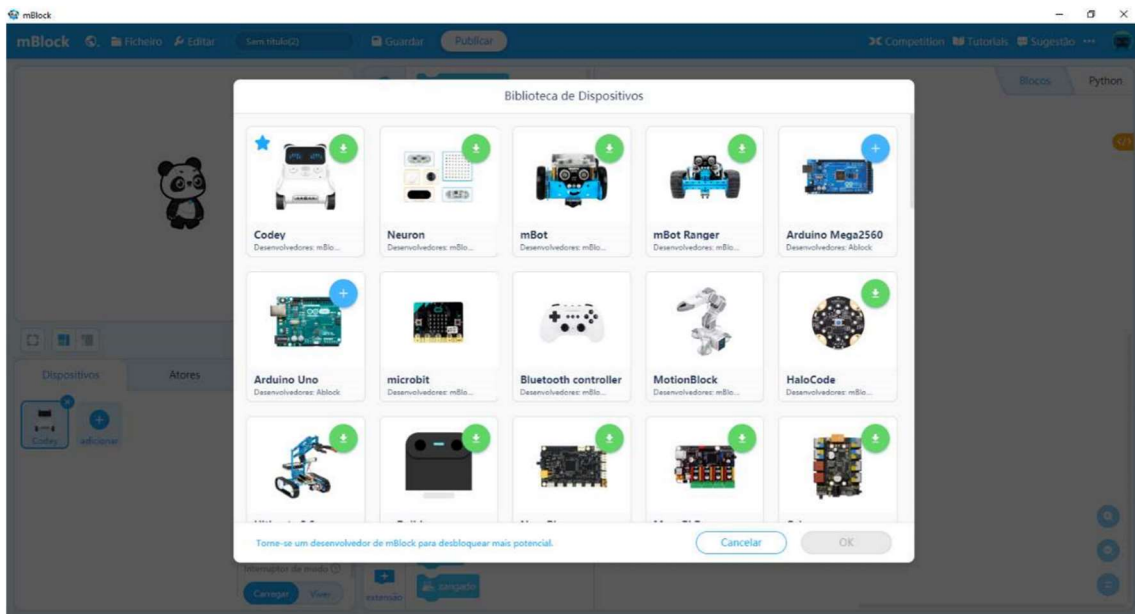
Figura 10: Escolhendo atores.



Alternativamente, podemos utilizar dispositivos e não atores, neste caso é preciso conectar o dispositivo. Além do mBot, é possível programar o Codey, Neuron, mBot Ranger, Arduino, microbit, bluetooth, MotionBlock, HaloCode, Raspberry Pi, dentre outros. A Figura 11 mostra a janela de adição de dispositivos.

Quando usamos um Sprite, o programa se restringe a exibição das ações e interações dos personagens 2D na tela do dispositivo que está rodando o mBlock. Já quando utilizamos um dispositivo, como o mBot, é preciso conectar ele ao computador por meio de um cabo USB para que o código possa ser enviado ao dispositivo.

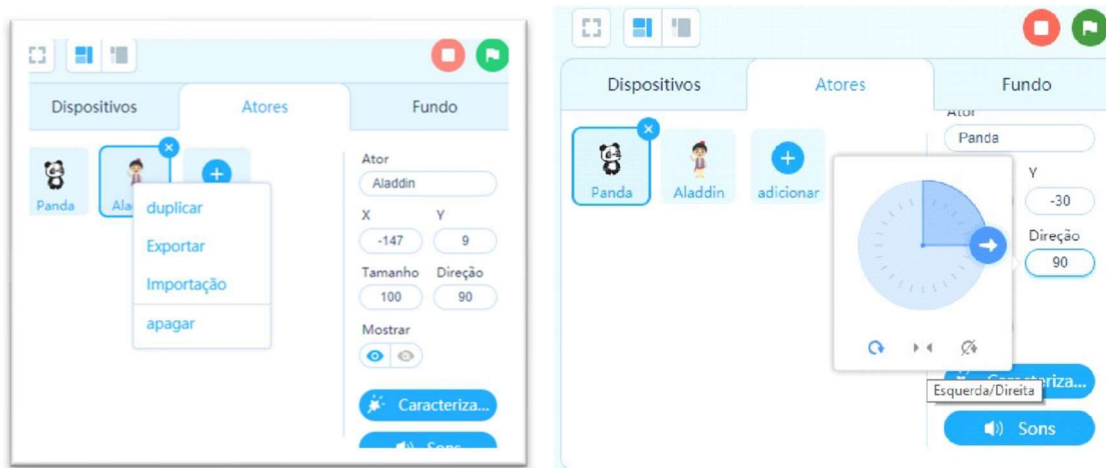
Figura 11: Adicionar dispositivos.



A figura 12 mostra as configurações de um ator, em que, clicando com o botão direito do mouse temos as opções de duplicar, exportar, importar ou apagar, o mesmo é observado para os dispositivos. Além disso o ator possui um nome, um posicionamento do palco, dado pelas coordenadas X e Y (o centro do palco é $X=0$ e $Y=0$), um tamanho (percentual da figura original), e a direção (ângulo). Ao clicar na direção aparece um círculo mostrando uma seta sobre o círculo, a qual pode ser arrastada para direcionar o personagem, Figura 12(b), assim, é possível escolher a direção dele mesmo sem conhecer ângulo e trigonometria. Outra funcionalidade muito importante nessa janela, a movimentação do personagem da esquerda para a direita (as setas embaixo do círculo de direção, Figura 12(b)), por meio dessas setas habilita-se/desabilita-se o movimento da esquerda para direita e ainda pode-se habilitar ou não o giro do personagem (seta com rotação em baixo do círculo de direção da Figura 12(b)).

Também é possível mostrar, ocultar, atribuir sons e mudar as características do ator. Ao clicar no botão “caracterização” é aberto um editor de imagens, exibido na Figura 13, em que podemos desenhar sobre a figura original que representa o ator.

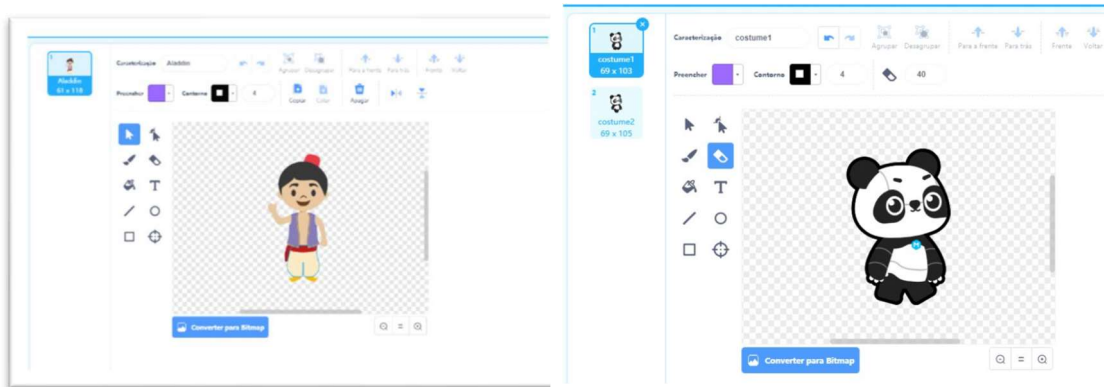
Figura 12: Editando as informações de Atores.



(a) Ações do ator

(b) Mudando a direção do ator

Figura 13: Mudando a caracterização do ator.

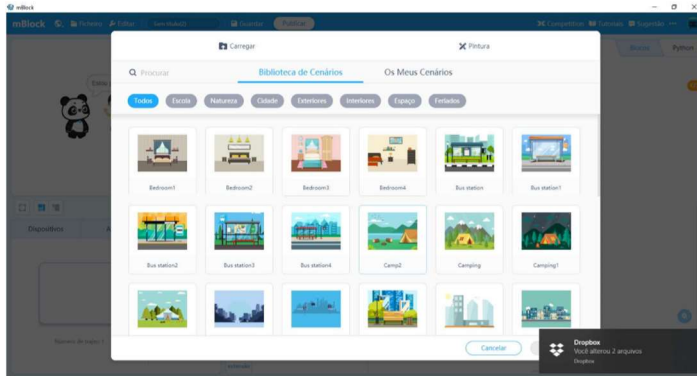


(a) Alladin

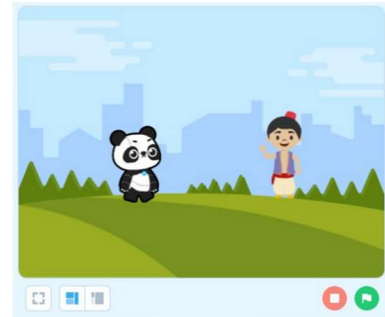
(b) Panda

O fundo é o desenho que vai aparecer atrás dos atores, ao clicar na aba “Fundo” e no botão adicionar, vão aparecer as opções disponíveis para o fundo, Figura 14 (a).

Figura 14: Inserindo o fundo no palco.



(a) Escolha do fundo.



(b) Fundo inserido no palco.

2.4 Blocos

Os blocos são divididos por categoria. A seguir mostramos os blocos de cada uma das categorias destinadas ao mBot e personagens, lembrando que as categorias mudam conforme o tipo de dispositivo.

2.4.1 Aparência

A categoria aparência contém blocos que alteram a aparência do ator ou do dispositivo, a Figura 15(a) mostra os blocos de Aparência para um dispositivo e a Figura 15(b) mostra os blocos de Aparência para um ator. Inicialmente deve-se clicar sobre o dispositivo ou ator e em seguida, clicar-se na categoria Aparência para ver todos os blocos disponíveis. Conforme o tipo de personagem, será exibida uma das duas listas de blocos.

O dispositivo é limitado quanto ao aspecto de aparência, são poucos os blocos e nem sempre o dispositivo contará com o recurso, por exemplo, o mBot não possui o visor de led para os olhos.

Figura 15: Blocos relacionados a “Aparência.”



(a) MBot

(b) Ator



Na Figura 16 vemos um programa inicial para o personagem Panda, o primeiro item que inserimos foi o evento  para que o programa saiba quando o usuário quer iniciar a ação do personagem (sempre deve haver um evento que inicie). Quando o usuário clicar na bandeira abaixo do palco (apontado pela seta vermelha na Figura 17), o programa inicia sua execução, que no caso fará com que o Panda diga “Olá” por dois segundos, pois usamos o bloco  conforme pode ser visto na Figura 17. O botão vermelho interrompe a ação dos personagens.

Figura 16: Primeiro programa – mudando a aparência do personagem.

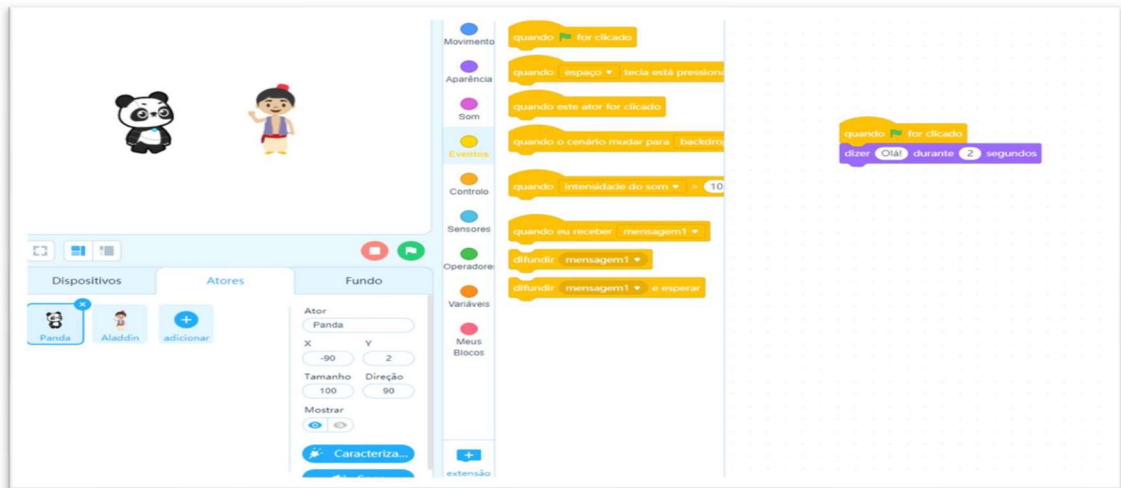
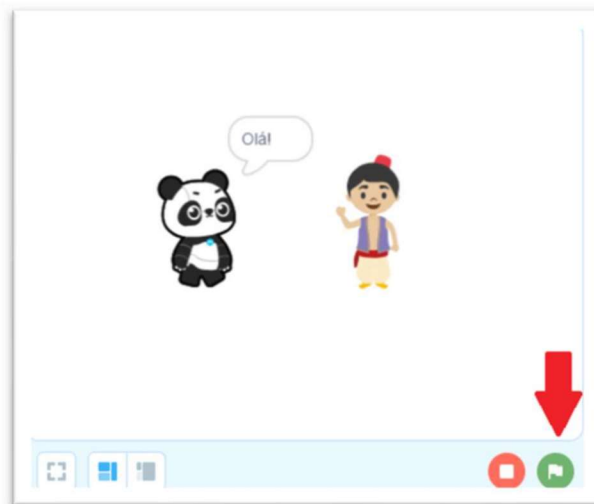


Figura 17: Executando o programa.



2.4.2 Mostrar

Essa categoria existe apenas para o mBot e outros dispositivos. Ela permite colocar os LEDs de uma determinada cor durante segundos e tocar determinada nota durante algum tempo ou tocar o som em uma determinada frequência por um determinado tempo. Vamos mostrar o uso destes blocos no Capítulo de experimentos.

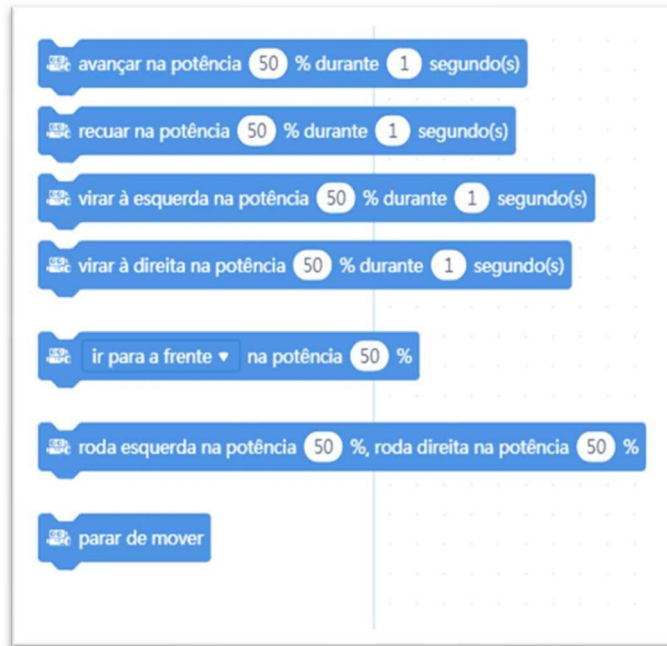
Figura 18: Blocos relacionados a “Mostrar”.



2.4.3 Ação

Essa categoria existe apenas para dispositivos. Ela determina o movimento do mbot, podendo avançar ou recuar, virar à esquerda ou à direita, ir para frente e parar de mover, andar mais rápido ou devagar, isso é feito por meio do ajuste da potência usada no movimento. É semelhante a categoria movimento dos atores, contudo, mais limitada pois o personagem do tipo ator possui mais liberdade de movimentos. Vamos mostrar o uso destes blocos no Capítulo de experimentos.

Figura 19: Blocos relacionados a “Ação”.

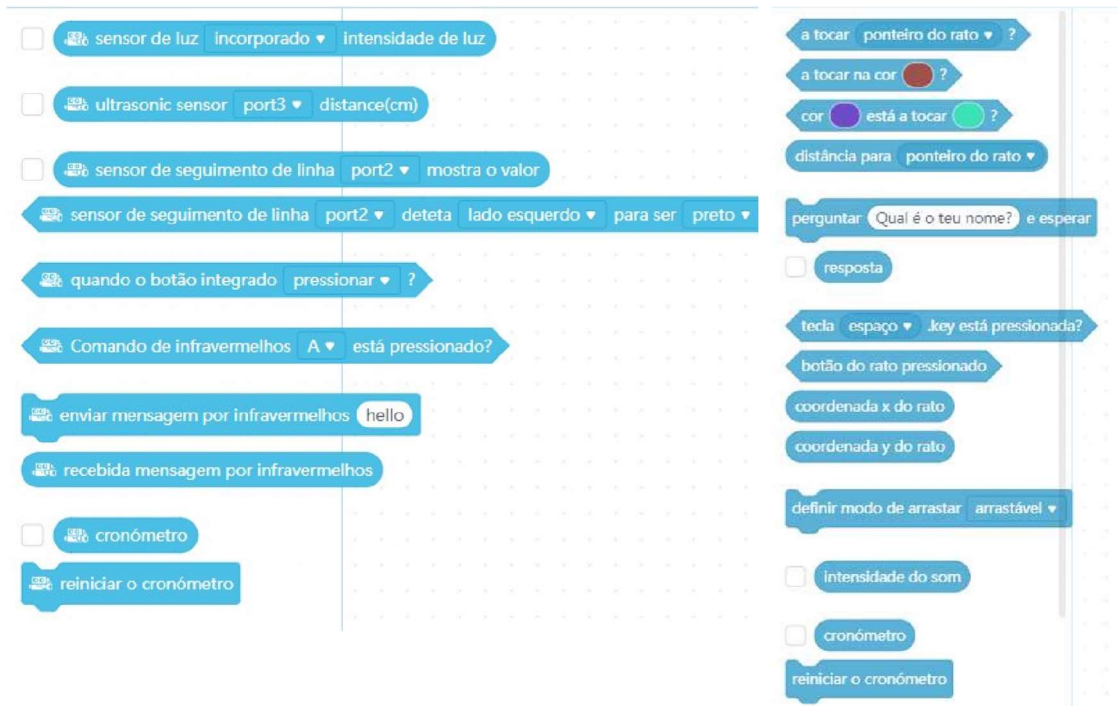


2.4.4 Sensores

Essa categoria permite ao mBot verificar os LEDs, inspecionar o sensor ultrassônico, verificar o valor do sensor de seguir linha, verificar se o botão integrado foi pressionado, enviar e receber mensagens por infravermelho. Já quando é um ator, é possível testar várias situações relativas ao ambiente: quando o personagem tocar o ponteiro do mouse ou o limite do palco, ou uma cor, ou mesmo um outro personagem; existe também a possibilidade enviar e receber mensagens, verificar teclas pressionadas, usar um cronometro etc.

Os blocos da categoria Sensores sempre são utilizados em conjunto com blocos da categoria Controlo, pois é necessário testar as diferentes situações e reagir de maneira diferente conforme observado no ambiente. Por exemplo, um personagem deve caminhar até encontrar o fim do palco.

Figura 20: Blocos relacionados a “Sensores”.



(a) MBot

(b) Ator

2.4.5 Eventos

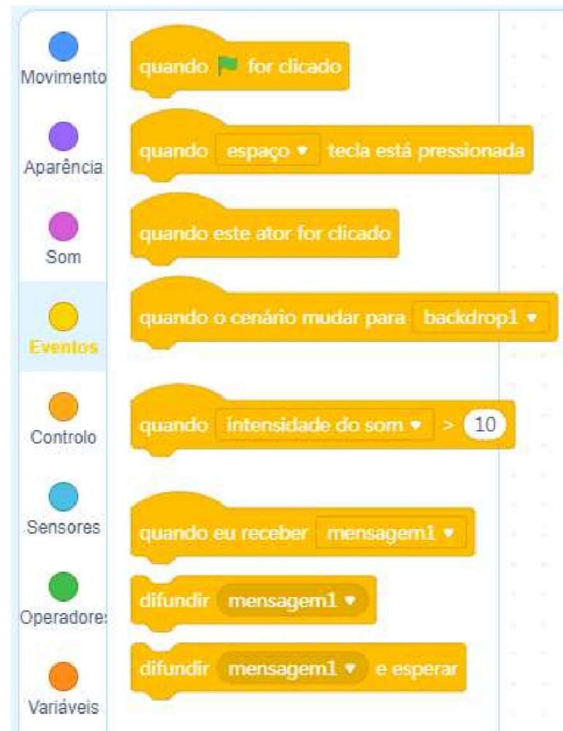
Essa categoria contém blocos para que o personagem ou mBot perceba a ocorrência de alguma coisa no ambiente, por exemplo, quando a bandeira verde for clicada, ou a tecla de espaço for pressionada, ou o botão integrado for pressionado, ou mesmo quando o personagem receber mensagens. As duas categorias são parecidas para o mBot e para o ator, conforme pode ser observado na Figura 21. Geralmente estes eventos iniciam a ação do dispositivo ou ator.

Na Figura 22, o Panda diz “Olá” por 2 segundos, altera seu tamanho em 10% e em seguida pensa “Hummm” durante dois segundos, o ator Alladin tem seu próprio script e tem ações diferentes do Panda.

Figura 21: Blocos relacionados a “Eventos”.

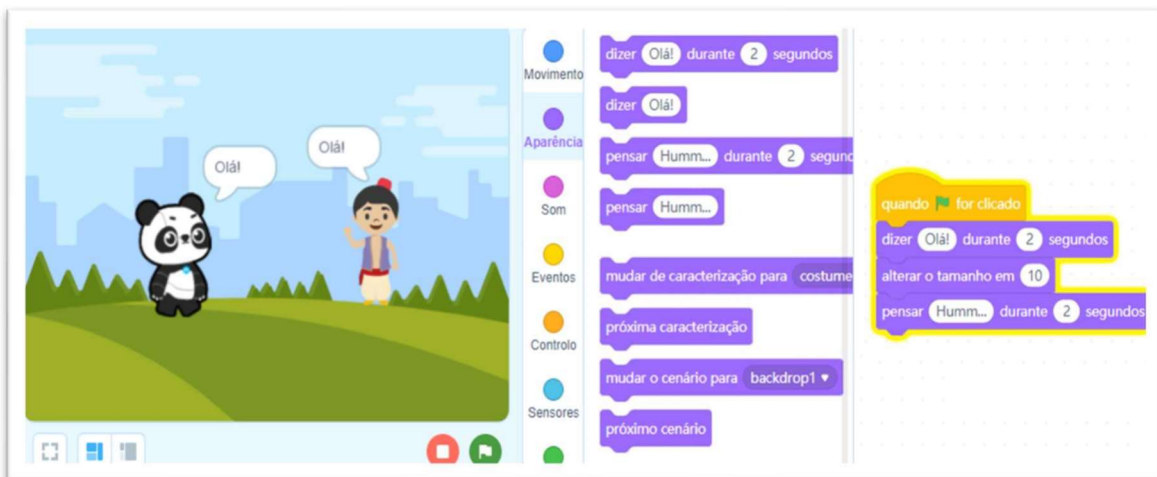


(a) MBot



(b) Ator

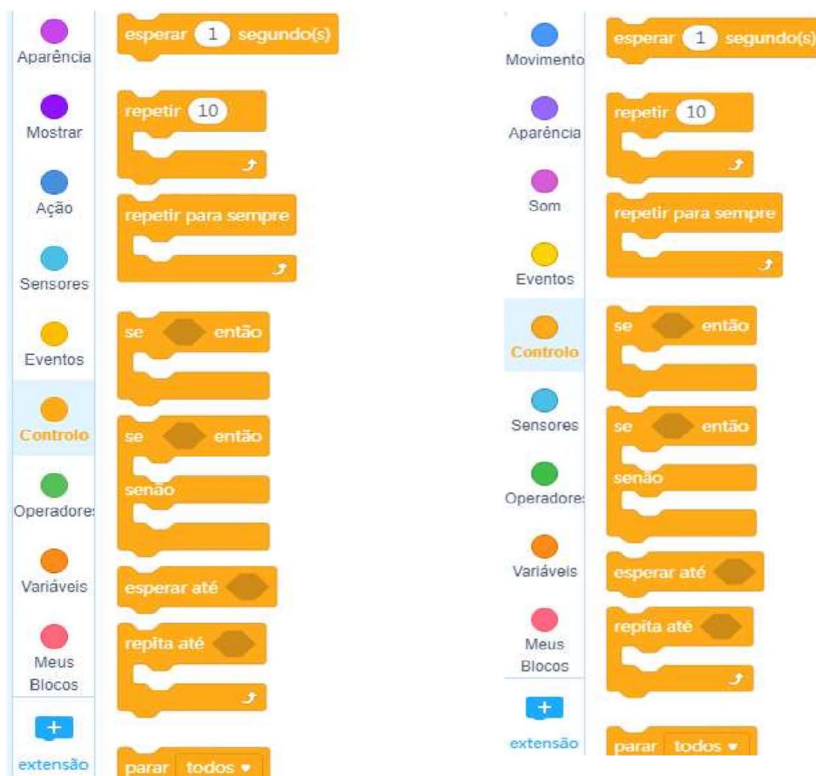
Figura 22: Iniciando com o evento de clicar na bandeira.



2.4.6 Controlo

Essa categoria é uma das mais importantes pois contém estruturas de decisão e controle de fluxo. As estruturas de decisão consistem do teste de uma determinada condição e execução de um bloco de comando ou de outro mediante o resultado do teste. Também há estruturas de repetição para que uma determinada condição no conjunto de blocos de comandos seja repetido, também é possível ter um laço de repetição infinito usando o bloco “repetir para sempre”, este bloco é particularmente útil pois força o ator/mBot a ter uma rotina. Há, ainda, blocos para fazer com que se espere por determinado tempo ou parar.

Figura 23: Blocos relacionados a “Controlo”.



(a) MBot

(b) Ator

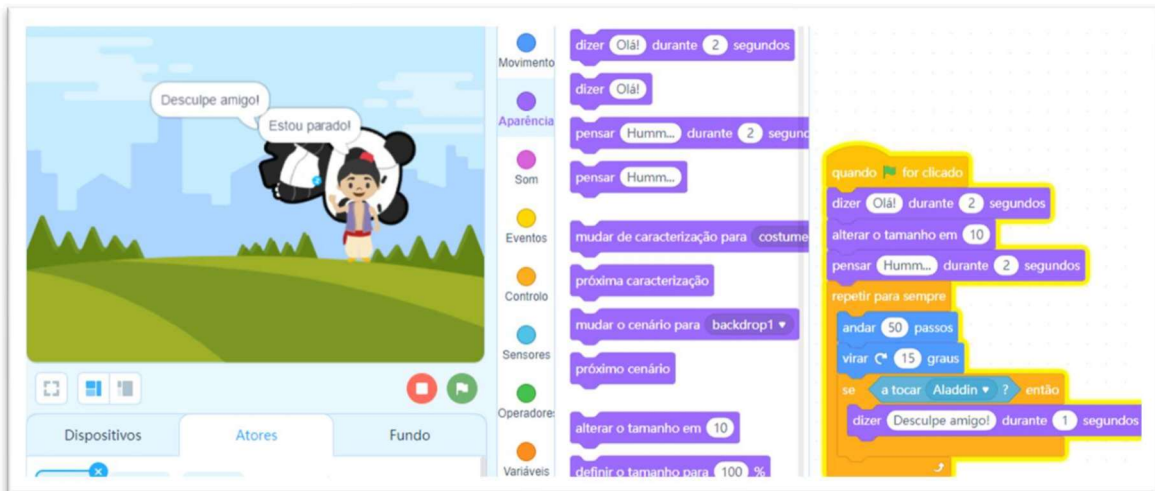
Na Figura 24, fizemos com que o Panda, após fazer o cumprimento, passasse a andar 50 passos, virando 15° em seguida, e estas duas ações são repetidas infinitamente ou até que o usuário clique no botão vermelho de interromper a execução, logo abaixo do palco. É importante observar que o personagem não caminha dando passos, mas apenas alterando sua posição ao longo do palco, isso associado a virada de 15° faz com que o Panda rodopie na tela.

Figura 24: Usando “repetir para sempre”.



Na Figura 25 usamos o bloco de comando se - senão em que o personagem panda anda 50 passos, vira 15° e em seguida verifica se tocou no Aladdin, caso isso ocorra, o Panda emitirá uma mensagem “Desculpe amigo”.

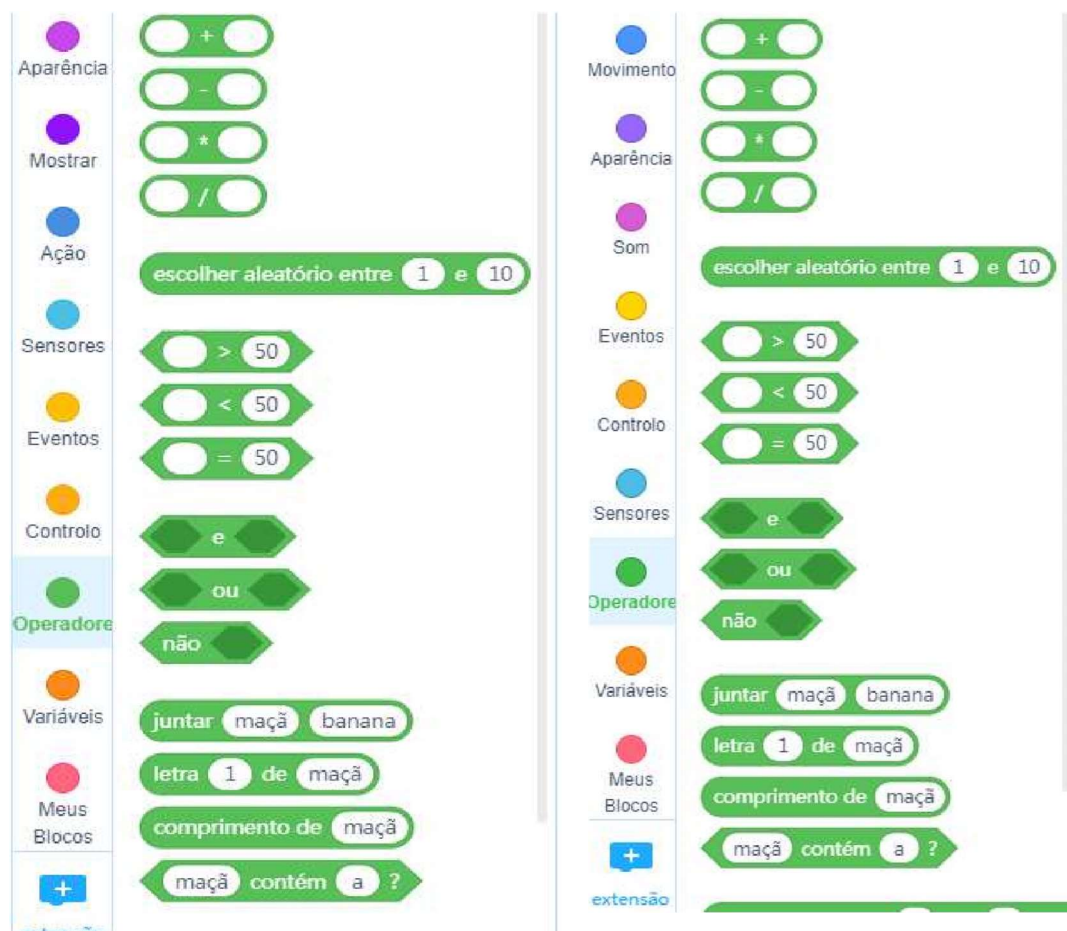
Figura 25: Usando “se - senão” e sensor.



2.4.7 Operadores

Essa categoria consiste de operadores lógicos (e, ou, não), aritméticos (+, -, *, /) e relacionais (>, <, =), além de funções de que podem ser aplicadas a palavras (juntar palavras, selecionar uma letra dentro de uma palavra, verificar se a palavra contém uma determinada letra, etc. Há ainda o resto da divisão, arredondamento, valor absoluto e escolha de números aleatórios. Essa categoria é idêntica para atores e mBot.

Figura 26: Blocos relacionados a “Operadores”.

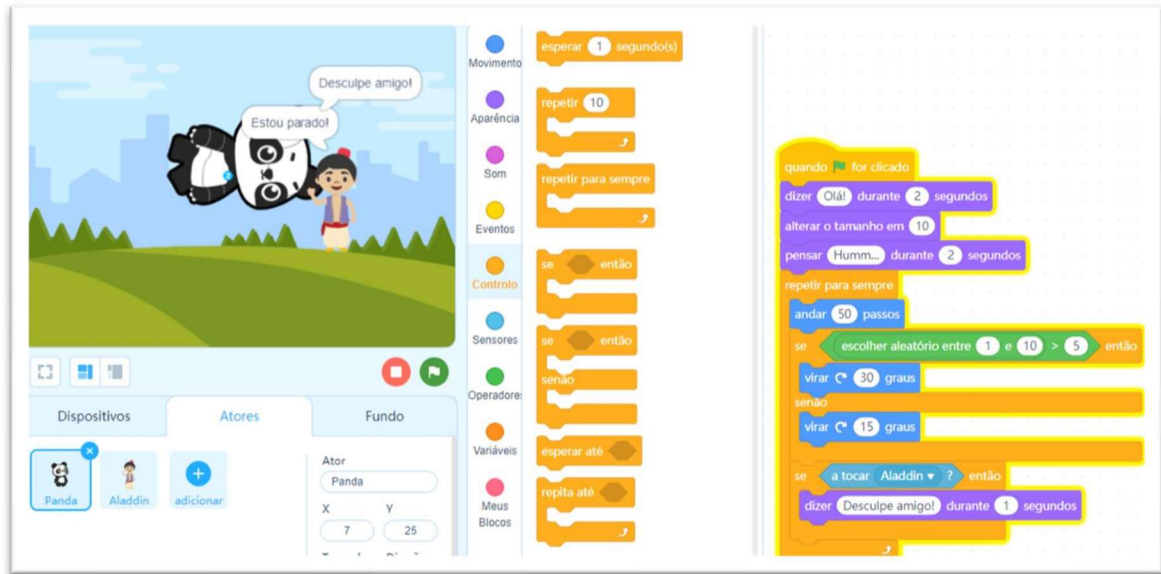


(a) MBot

(b) Ator

Na Figura 27 vemos um exemplo de uso do operador “>”. O operador precisa ser encaixado dentro de um teste condicional. No caso, após o panda andar 50 passos, se o número aleatório (entre 1 e 10) gerado for maior que 5, o Panda vira 30°, senão, o Panda vira 15°. Todo operador possui operandos, alguns são binários pois recebem dois operandos, é o caso do operador > em que nós precisamos de dois números para saber se um é maior que o outro, neste exemplo usamos o número gerado aleatoriamente (ao acaso) pelo mBlock e o valor 5.

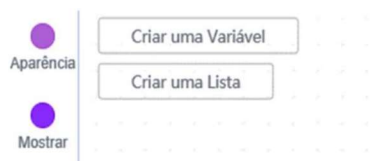
Figura 27: Usando operador “>” e número aleatório.



2.4.8 Variáveis

Essa categoria permite criar uma variável e criar uma lista. Uma variável é uma informação que vai ficar guardada na memória durante a execução do programa, podendo esta informação ser um número ou texto. Muitas vezes queremos registrar o número de vezes que um personagem deve executar uma ação, neste caso podemos usar a variável.

Figura 28: Blocos relacionados a “Variáveis”.



(a) MBot



(b) Ator

2.4.9 Meus Blocos

Essa categoria permite criar seu próprio bloco. Quando você cria seu próprio bloco, pode reutilizá-lo quantas vezes precisar, por exemplo, um personagem operado por meio das setas de navegação do teclado é algo muito comum e em programas é usual que se use esse recurso, assim é mais útil produzir a sequência de blocos que geram a movimentação e reutilizar.

Figura 29: Blocos relacionados a “Meus blocos”.

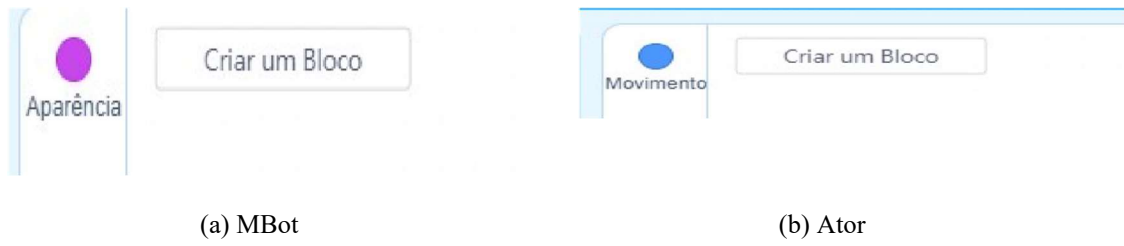
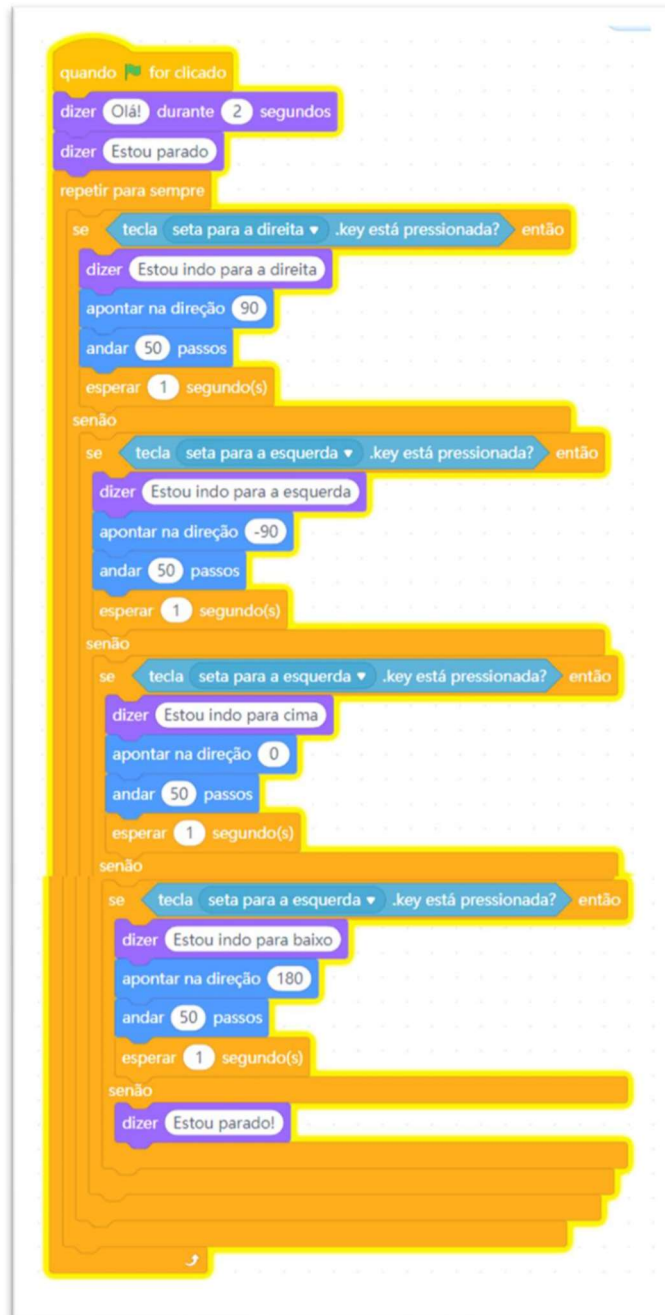


Figura 30: Programa para movimentar um personagem por meio do teclado.



2.4.10 Som

Essa categoria existe apenas para atores. Ela permite tocar, iniciar e parar o som, alterar o efeito, definir e limpar todos efeitos, alterar o volume.

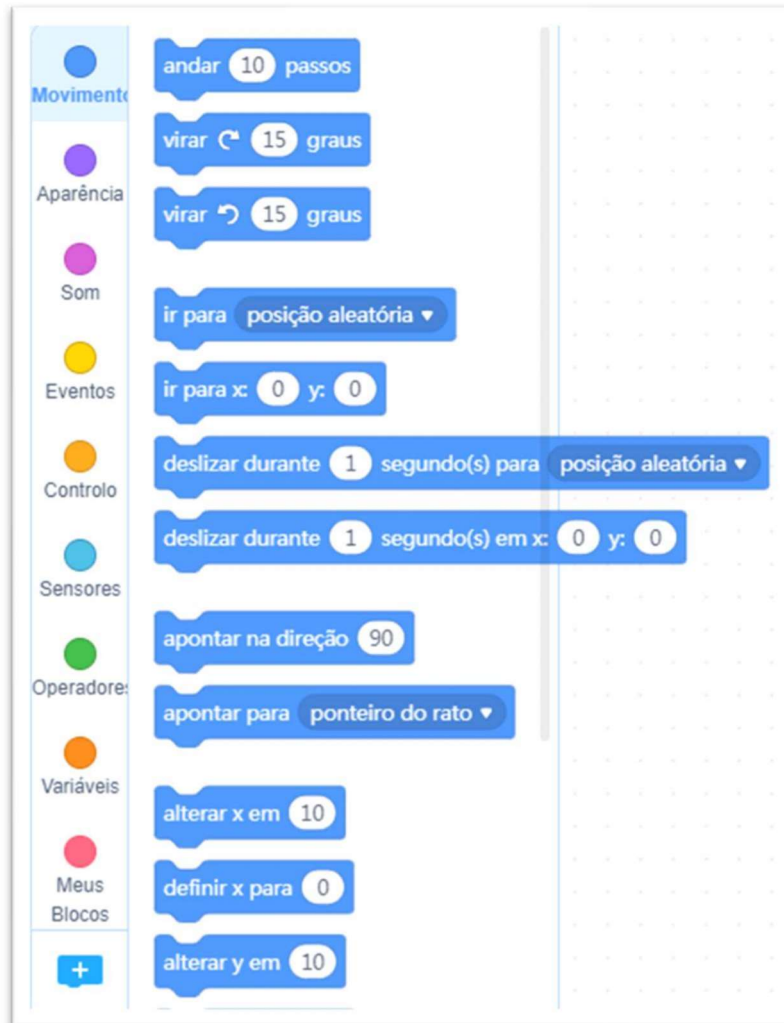
Figura 31: Blocos relacionados a “Som”.



2.4.11 Movimento

Essa categoria existe apenas para atores. Ela permite andar determinados passos, virar determinados graus, ir para posição aleatória, deslizar durante segundos, apontar em tal direção.

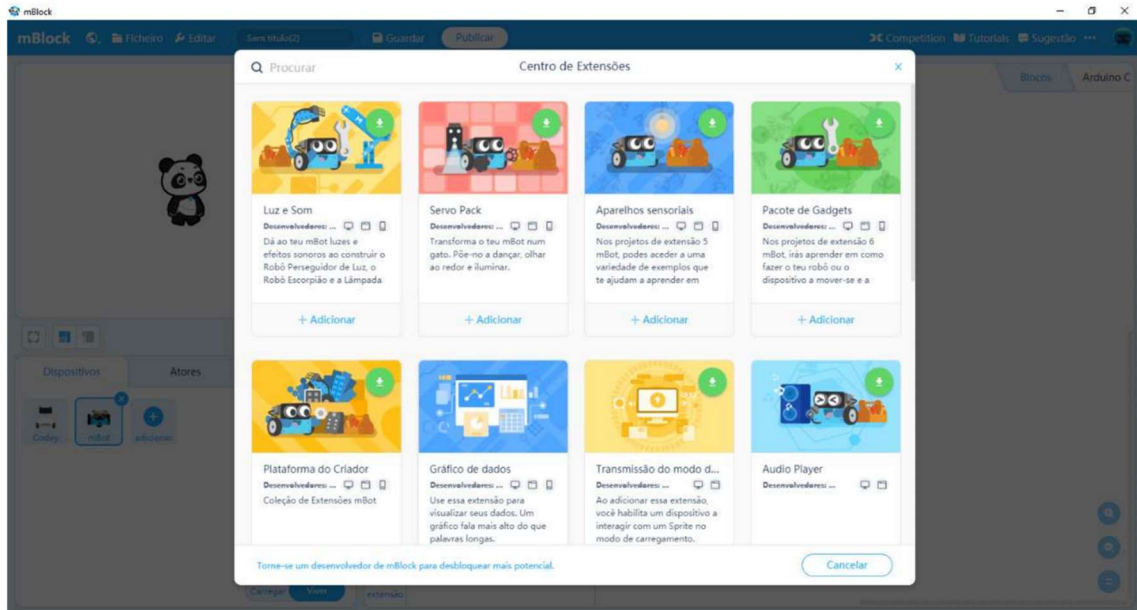
Figura 32: Blocos relacionados a “Movimento”.



2.5 Extensões

O mBlock permite o uso de extensões por meio do botão extensões, que fica na parte de baixo da área de blocos. Ao clicar neste botão, será aberto o centro de extensões.

Figura 33: Centro de extensões.



CAPÍTULO 3: EXPERIMENTOS COM MBOT

3.1 1º Experimento:

Foi desenvolvido um programa para o mBot com o objetivo de mudar a cor dos leds, utilizando o auxílio de tutoriais da internet.

Página 34: Programa em blocos para leds.

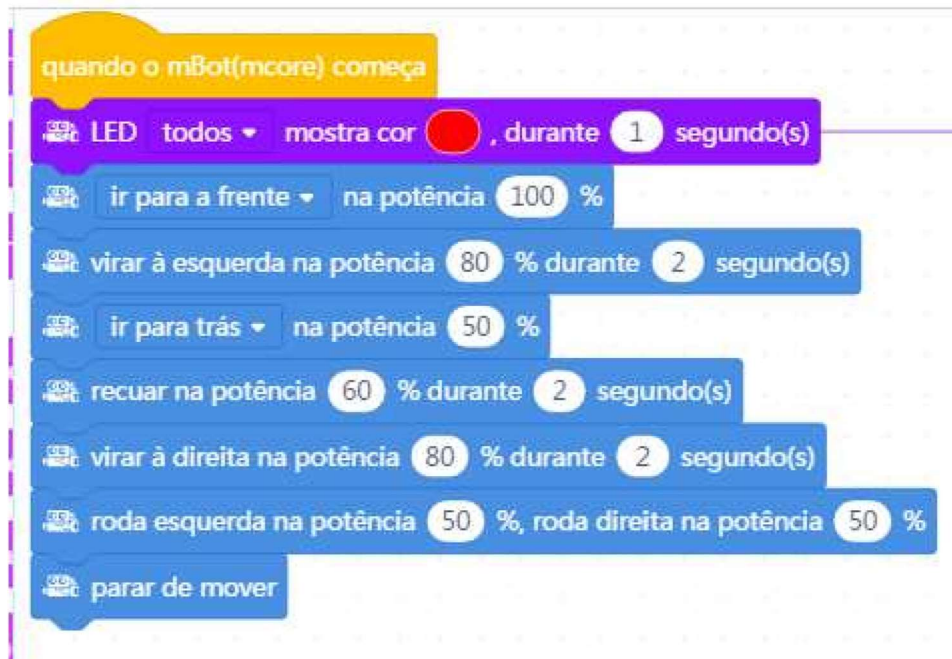


- Bloco 1: ponto de partida do programa;
- Bloco 2: mostrar todos os leds na cor vermelha durante 1 segundo;
- Bloco 3: mostrar os leds do lado esquerdo na cor verde durante 1 segundo;
- Bloco 4: mostrar os leds do lado direito na cor azul durante 1 segundo;
- Bloco 5: mostrar todos os leds na cor roxa durante 1 segundo.

3.2 2º Experimento:

Foi desenvolvido outro programa para o mBot com o objetivo de fazer uma “dança”. Este programa conta com movimentos sincronizados e alteração dos leds.

Figura 35: Programa em blocos.



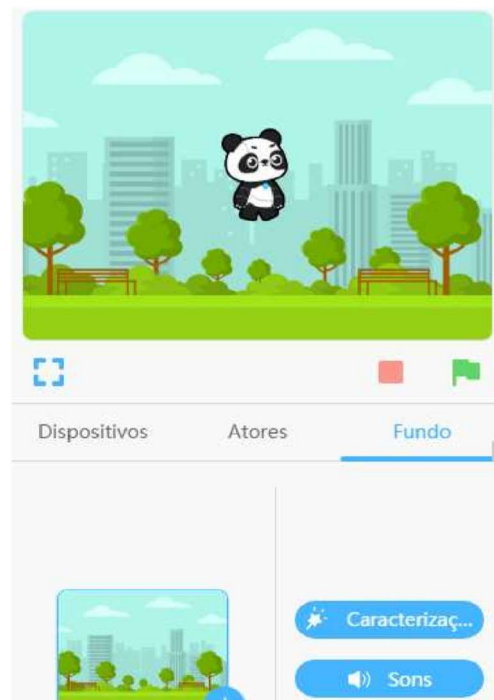
- Bloco 1: ponto de partida do programa;
- Bloco 2: mostrar todos os leds na cor vermelha durante 1 segundo;
- Bloco 3: ir para a frente na potência de 100%;
- Bloco 4: virar à esquerda na potência de 80% durante 2 segundos;
- Bloco 5: ir para trás na potência de 50%;
- Bloco 6: recuar na potência de 60% durante 2 segundos;
- Bloco 7: virar à direita na potência de 80% durante 2 segundos;
- Bloco 8: roda esquerda e direita do robô na potência de 50%;
- Bloco 9: parar de mover.

3.3 3º Experimento:

Foi desenvolvido um jogo para o ator com o objetivo de pegar as maçãs do céu, utilizando um cesto.

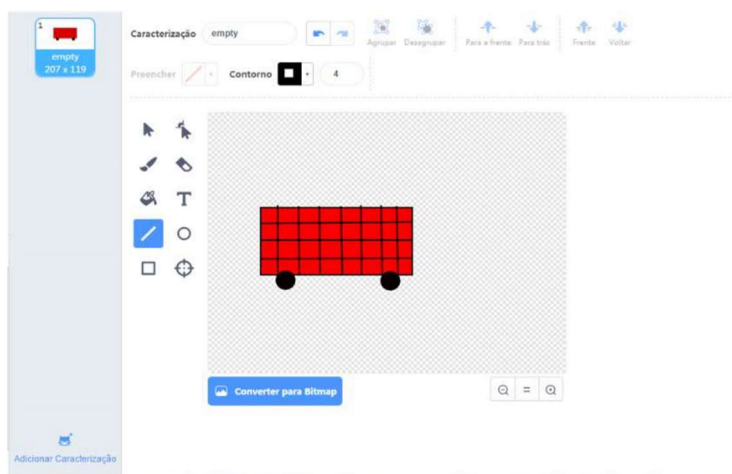
- Inicialmente, foi escolhido um cenário para o jogo.

Figura 36: Cenário do jogo.



- Em seguida, foi desenhado um novo ator que seria o cesto.

Figura 37: Desenho do cesto.



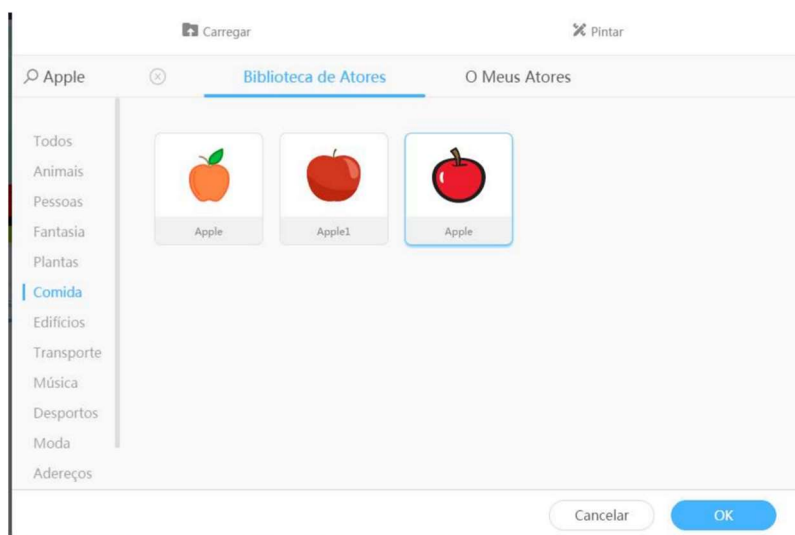
- Depois foi desenvolvido um código em blocos para esse novo ator.

Figura 38: Código do ator cesto.



- Bloco 1: ponto de partida na bandeira verde;
 - Bloco 2: ir para $x = 3$ e $y = -135$;
 - Bloco 3: inicia o bloco de repetição → se a tecla de seta para a direita está pressionada, então o ator vai andar 20 passos → se a tecla de seta para a esquerda está pressionada, então o ator vai andar 20 passos para trás.
- Logo, foi escolhido um ator na biblioteca de atores para a maçã.

Figura 39: Ator maçã.



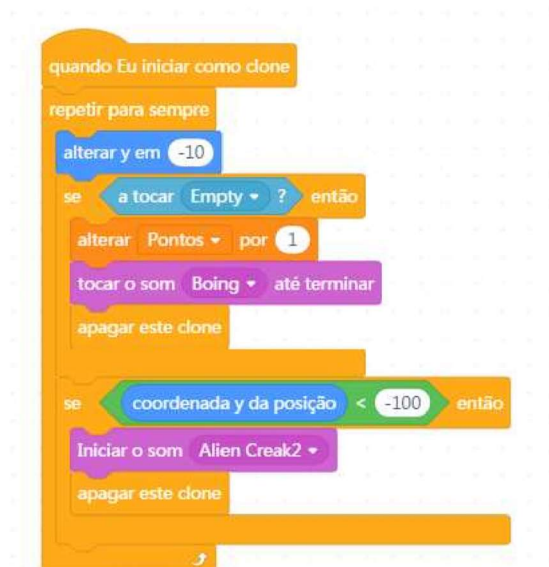
- Em seguida, foi desenvolvido dois códigos em blocos para esse outro ator.

Figura 40: Código 1 para o ator maçã.



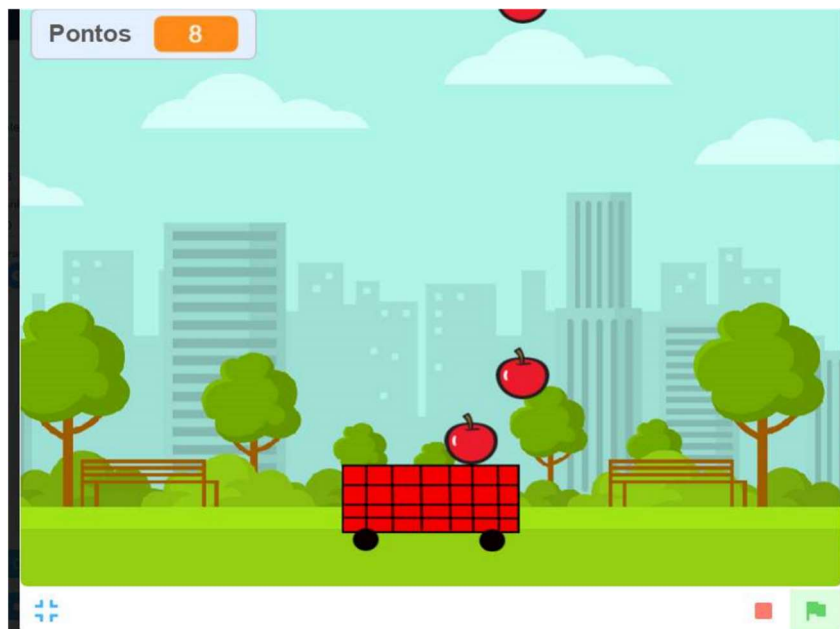
- Bloco 1: ponto de partida na bandeira verde;
- Bloco 2: definir os pontos para 0;
- Bloco 3: mostrar os pontos;
- Bloco 4: inicia o bloco de repetição → ir para x com o valor aleatório entre -210 e 210 e y = 210 → criar um clone do ator maçã → esperar durante 0,1 e 1,5 segundos escolhidos aleatoriamente.

Figura 41: Código 2 para o ator maçã.



- Bloco 1: ponto de partida do clone;
- Bloco 2: inicia o bloco de repetição → alterar $y = -10$ → se tocar a música “empty”, então vai alterar pontos por 1 → tocar o som “boing” até terminar e apagar o clone → se a coordenada y estiver na posição menor que -100, então vai iniciar o som “alien creak2” e apagar o clone.

Figura 42: Jogo pronto!

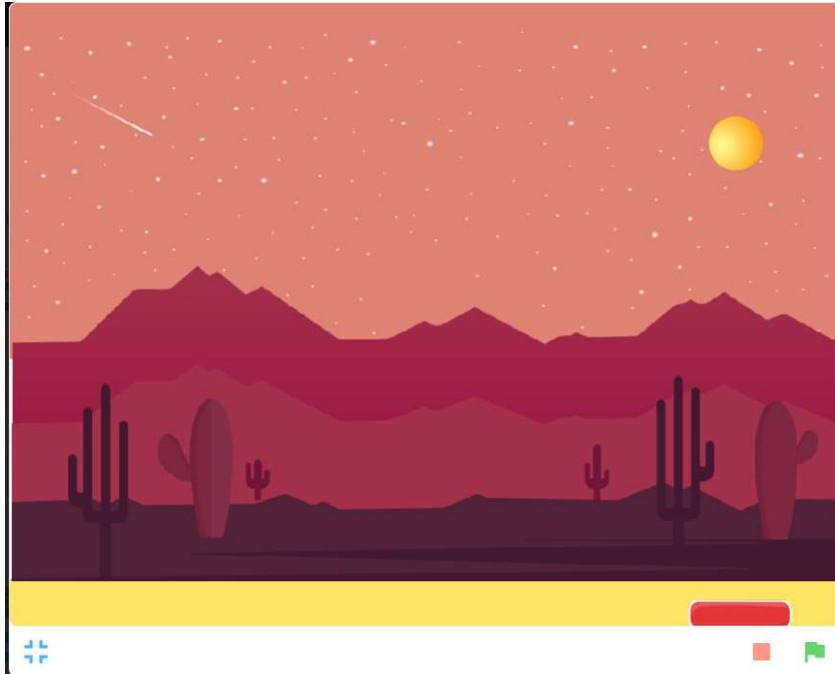


3.4.4º Experimento:

Foi desenvolvido um jogo parecido com Ping Pong para o ator, com o objetivo de pegar a bola usando uma barra e não deixar essa bola encostar no chão.

- Inicialmente foi escolhido um cenário para o jogo.

Figura 43: Cenário do jogo.



- Em seguida, foi escolhido o ator “bola” na aba de atores.

Figura 44: Ator “bola”.



- Depois, foi desenhado os atores “base” (barra usada para pegar a bola) e “fim” (chão do cenário).

Figura 45: Ator “base”.

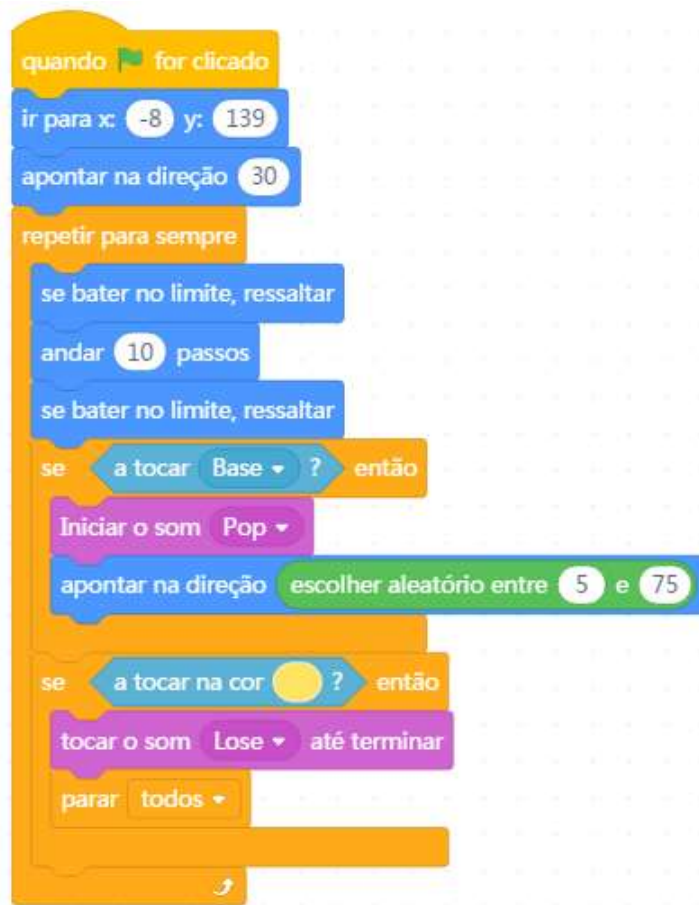


Figura 46: Ator “fim”.



- Logo, foi criado um código para o ator “bola”, definindo as suas funções no jogo.

Figura 47: Código do ator “bola”.



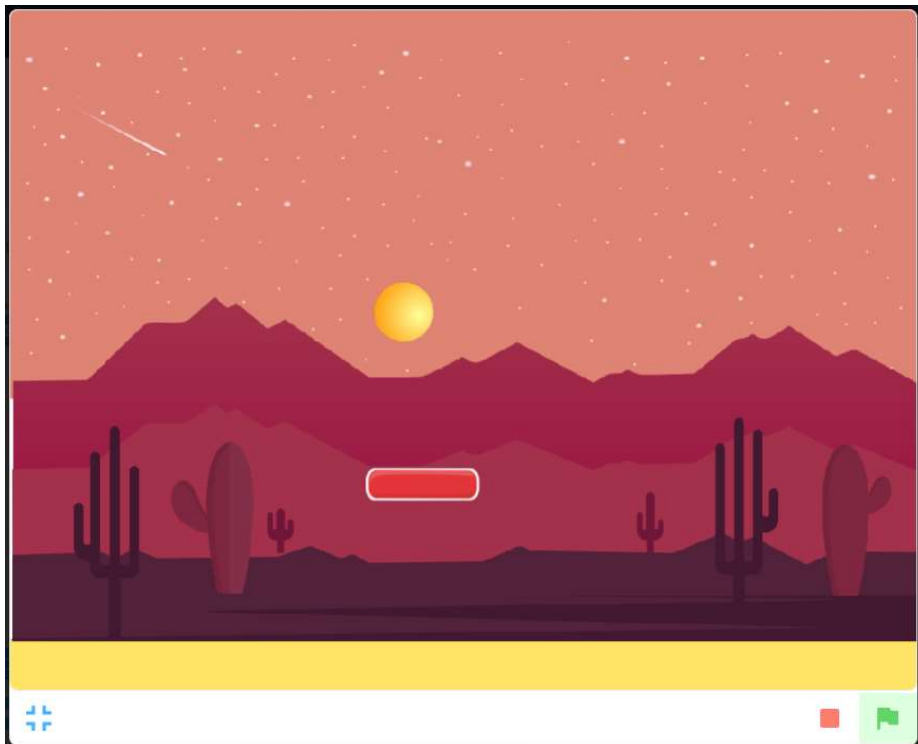
- Bloco 1: ponto de partida na bandeira verde;
- Bloco 2: ir para a posição $x = -8$ e posição $y = 139$;
- Bloco 3: apontar na direção = 30;
- Bloco 4: inicia o bloco de repetição → se o ator “bola” bater no limite, ressaltar → andar 10 passos → novamente, se o ator “bola” bater no limite, ressaltar. Depois, inicia o bloco de condição → se tocar o som “Base”, então vai iniciar o som “Pop” e apontar na direção escolhida de forma aleatória entre 5 e 75. Por fim, inicia outro bloco de condição → se tocar na cor amarela, então tocar o som “Lose” até terminar e parar todos.

Figura 48: Código para o ator “base”.



- Bloco 1: ponto de partida na bandeira verde;
- Bloco 2: ir para a posição x = -8 e posição y = -105;
- Bloco 3: esperar durante 0.5 segundos;
- Bloco 4: inicia o bloco de repetição → ir para o ponteiro do rato.

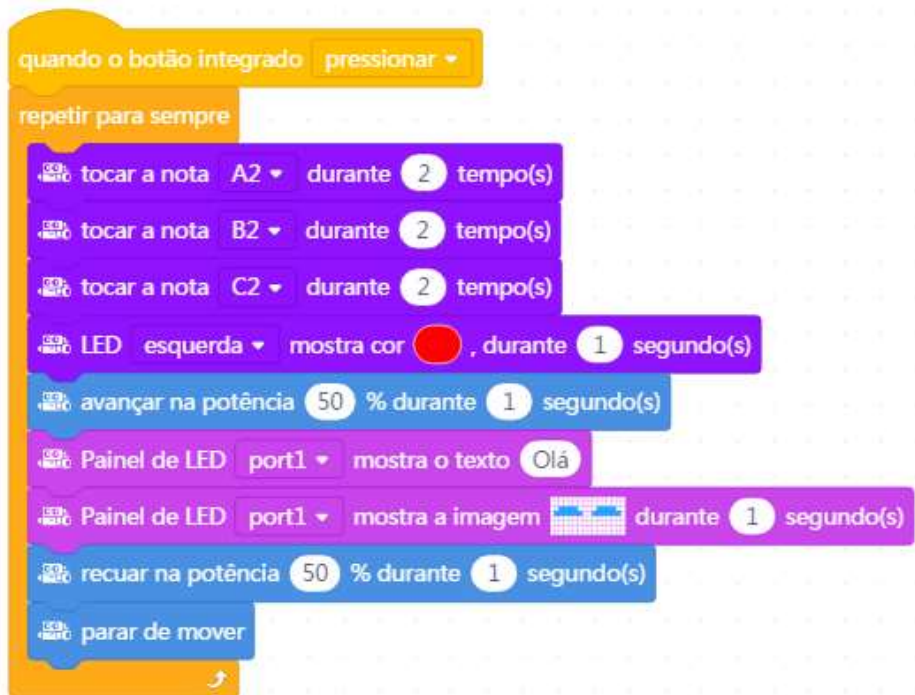
Figura 49: Jogo pronto!



3.3 5º Experimento:

Foi desenvolvido um programa em blocos para o mBot com o objetivo de fazer uma apresentação de cumprimento.

Figura 50: Programa em blocos.



- Bloco 1: ponto de partida ao pressionar o botão integrado;
- Bloco 2: inicia o bloco de repetição → tocar a nota “A2” durante 2 tempos; tocar a nota “B2” durante 2 tempos; tocar a nota “C2” durante 2 tempos; mostrar a led na cor vermelha do lado esquerdo durante 1 segundo → avançar na potência de 50% durante 1 segundo; mostrar o texto “Olá” no painel de led na porta 1; mostrar a imagem definida no painel de led na porta 1 durante 1 segundo → recuar na potência de 50% durante 1 segundo e parar de mover.

CAPÍTULO 4: EXPERIMENTOS COM SEGUIE LINHA

Para realizar experimentos, e modificar a maneira como o robô segue linha é necessário modificar o firmware do mBot, isso é feito em Arduino. O quadro a seguir apresenta o código do firmware. Em vermelho foi destacada a parte relativa ao código que resulta em seguir a linha, com pode ser observado, é chamada uma função chamada pinMode.

Quadro 1: firmware do mBot.

<pre>***** ***** * File Name : mbot_firmware.ino * Author : Ander, Mark Yan * Updated : Ander, Mark Yan * Version : V06.01.106 * Date : 07/06/2016 * Description : Firmware for Makeblock Electronic modules with Scratch. * License : CC-BY-SA 3.0 * Copyright (C) 2013 - 2016 Maker Works Technology Co., Ltd. All right reserved.</pre>	<pre>* http://www.makeblock.cc/ ***** *****/ #include <Wire.h> #include <MeMCore.h> Servo servos[8]; MeDCMotor dc; MeTemperature ts; MeRGBLed led(0,30); MeUltrasonicSensor us; Me7SegmentDisplay seg;</pre>
---	---

```

MePort generalDevice;
MeLEDMatrix ledMx;
MeBuzzer buzzer;
MeIR ir;
MeGyro gyro;
MeJoystick joystick;
MeCompass Compass;
MeHumiture humiture;
MeFlameSensor FlameSensor;
MeGasSensor GasSensor;
MeTouchSensor touchSensor;
Me4Button buttonSensor;

typedef struct MeModule
{
    int device; int port; int slot;
    int pin; int index; float
    values[3];
} MeModule;

union{
    byte byteVal[4]; float floatVal;
    long longVal;
} val;

union{ byte byteVal[8]; double
doubleVal;
} valDouble;

union{ byte byteVal[2]; short
shortVal;
} valShort;

#if defined(__AVR_ATmega32U4__) const int analogs[12]
PROGMEM =
{A0,A1,A2,A3,A4,A5,A6,A7,A8,A9,A10,A11};
#else const int analogs[8] PROGMEM =
{A0,A1,A2,A3,A4,A5,A6,A7};
#endif
String mVersion = "06.01.106";
boolean isAvailable = false;

int len = 52; char buffer[52]; byte index = 0;
byte dataLen; byte modulesLen=0; boolean
isStart = false; char serialRead;
uint8_t command_index = 0;
#define VERSION 0
#define ULTRASONIC_SENSOR 1
#define TEMPERATURE_SENSOR 2
#define LIGHT_SENSOR 3
#define POTENTIOMETER 4
#define JOYSTICK 5
#define GYRO 6
#define SOUND_SENSOR 7
#define RGBLED 8 #define SEVSEG 9

```

```

#define MOTOR 10
#define SERVO 11
#define ENCODER 12
#define IR 13
#define IRREMOTE 14
#define PIRMOTION 15
#define INFRARED 16
#define LINEFOLLOWER 17
#define IRREMOTECODE 18
#define SHUTTER 20
#define LIMITSWITCH 21
#define BUTTON 22
#define HUMITURE 23
#define FLAMESENSOR 24
#define GASSENSOR 25
#define COMPASS 26
#define DIGITAL 30
#define ANALOG 31
#define PWM 32
#define SERVO_PIN 33
#define TONE 34
#define BUTTON_INNER 35
#define LEDMATRIX 41
#define TIMER 50
#define TOUCH_SENSOR 51

#define GET 1
#define RUN 2
#define RESET 4 #define START 5 float
angleServo = 90.0;
int servo_pins[8]={0,0,0,0,0,0,0,0}; unsigned char
prevc=0; boolean buttonPressed = false;
uint8_t keyPressed = KEY_NULL;

void readButtonInner(uint8_t pin, int8_t s)
{
    pin = pgm_read_byte(&analogs[pin]);
    pinMode(pin,INPUT); boolean currentPressed =
    !(analogRead(pin)>10);

    if(buttonPressed == currentPressed){ return;
    }
    buttonPressed = currentPressed;
    writeHead(); writeSerial(0x80);
    sendByte(currentPressed);
    writeEnd();
}

void setup(){ pinMode(13,OUTPUT);
digitalWrite(13,HIGH); delay(300);
digitalWrite(13,LOW);
Serial.begin(115200); delay(500);
buzzer.tone(500,50); delay(50);
buzzerOff(); ir.begin(); led.setpin(13);
led.setColor(0,0,0); led.show();

```

```

ro.begin();
Serial.print("Version: "); Serial.println(mVersion);
ledMx.setBrightness(6);
ledMx.setColorIndex(1);
}
int irDelay = 0; int irIndex = 0; char
irRead = 0;
boolean irReady = false; String irBuffer = "";
double lastTime = 0.0; double currentTime =
0.0;
double lastIRTime = 0.0;

void loop(){
readButtonInner(7,0); keyPressed = buttonSensor.pressed();
currentTime = millis()/1000.0-lastTime; if(ir.decode())
{
irRead = ((ir.value>>8)>>8)&0xff; lastIRTime =
millis()/1000.0;
if(irRead==0xa||irRead==0xd){
irIndex = 0; irReady = true;
}else{
irBuffer+=irRead; irIndex++;
if(irIndex>64){ irIndex = 0;
irBuffer = "";
}
}
irDelay = 0;
}else{ irDelay++; if(irRead>0){
if(irDelay>5000){ irRead = 0;
irDelay = 0;
}
}
}
readSerial(); if(isAvailable){
unsigned char c = serialRead&0xff;
if(c==0x55&&isStart==false){
if(prevc==0xff){ index=1;
isStart = true;
}
}else{ prevc = c; if(isStart){
if(index==2){ dataLen = c; }else
if(index>2){ dataLen--;
}
writeBuffer(index,c);
}
}
index++; if(index>51){ index=0;
isStart=false;

```

```

}
if(isStart&&dataLen==0&&index>3){ isStart =
false; parseData();
index=0;
}
}
}
void buzzerOn(){
buzzer.tone(500,1000);
}
void buzzerOff(){
buzzer.noTone();
}
}
unsigned char readBuffer(int index){
return buffer[index];
}
void writeBuffer(int index,unsigned char c){ buffer[index]=c;
}
void writeHead(){ writeSerial(0xff);
writeSerial(0x55);
}
void writeEnd(){
Serial.println();
}
}
void writeSerial(unsigned char c){
Serial.write(c);
}
}
void readSerial(){ isAvailable = false;
if(Serial.available()>0){ isAvailable =
true;
serialRead = Serial.read();
}
}
}
/*
ff 55 len idx action device port slot data a
0 1 2 3 4 5 6 7 8
*/
void parseData(){ isStart = false; int idx =
readBuffer(3); command_index = (uint8_t)idx;
int action = readBuffer(4); int device =
readBuffer(5); switch(action){ case GET:{
if(device != ULTRASONIC_SENSOR){
writeHead();
writeSerial(idx);
}
readSensor(device);
writeEnd();
}
break; case RUN:{
runModule(device);
callOK();
}
break;
case RESET:{
//reset dc.reset(M1);
dc.run(0); dc.reset(M2);

```

```

    dc.run(0);    buzzerOff();
    callOK();
}
break;
case START:{
    //start
    callOK();
}
break;
}
}
void callOK(){    writeSerial(0xff);
writeSerial(0x55);    writeEnd();
}
void sendByte(char c){
    writeSerial(1);
    writeSerial(c);
}
void sendString(String s){
    int l = s.length();    writeSerial(4);
writeSerial(1);    for(int i=0;i<l;i++){
writeSerial(s.charAt(i));
}
}
//1 byte 2 float 3 short 4 len+string 5 double void sendFloat(float
value){
    writeSerial(2);    val.floatVal = value;
writeSerial(val.byteVal[0]);
writeSerial(val.byteVal[1]);
writeSerial(val.byteVal[2]);
writeSerial(val.byteVal[3]);
}
void sendShort(double value){    writeSerial(3);
    valShort.shortVal = value;    writeSerial(valShort.byteVal[0]);
writeSerial(valShort.byteVal[1]);
writeSerial(valShort.byteVal[2]);
writeSerial(valShort.byteVal[3]);
}
void sendDouble(double value){    writeSerial(5);
    valDouble.doubleVal = value;
writeSerial(valDouble.byteVal[0]);
writeSerial(valDouble.byteVal[1]);
writeSerial(valDouble.byteVal[2]);
writeSerial(valDouble.byteVal[3]);
writeSerial(valDouble.byteVal[4]);
writeSerial(valDouble.byteVal[5]);
writeSerial(valDouble.byteVal[6]);
writeSerial(valDouble.byteVal[7]);
}
short readShort(int idx){    valShort.byteVal[0] =
readBuffer(idx);    valShort.byteVal[1] = readBuffer(idx+1);
return valShort.shortVal;
}
float readFloat(int idx){    val.byteVal[0] =
readBuffer(idx);    val.byteVal[1] = readBuffer(idx+1);
val.byteVal[2] = readBuffer(idx+2);

```

```

    val.byteVal[3] = readBuffer(idx+3);
    return val.floatVal;
}
char _receiveStr[20] = {}; uint8_t
_receiveUInt8[16] = {}; char* readString(int
idx,int len){    for(int i=0;i<len;i++){
    _receiveStr[i]=readBuffer(idx+i);
}
    _receiveStr[len] = '\0';
    return _receiveStr;
}
uint8_t* readUInt8(int idx,int len){
    for(int i=0;i<len;i++){
        if(i>15){
            break;
        }
        _receiveUInt8[i] = readBuffer(idx+i);
    }
    return _receiveUInt8;
}
void runModule(int device){
    //0xff 0x55 0x6 0x0 0x2 0x22 0x9 0x0 0x0
0xa
    int port = readBuffer(6);
    int pin = port;    switch(device){
case MOTOR:{
        int speed = readShort(7);    if(dc.getPort()!=port){
dc.reset(port);
        }
        dc.run(speed);
    }
    break;
case JOYSTICK:{
        int leftSpeed = readShort(6);    dc.reset(M1);
dc.run(leftSpeed);
        int rightSpeed = readShort(8);    dc.reset(M2);
dc.run(rightSpeed);
    }
    break;
case RGBLED:{
        int slot = readBuffer(7);    int idx =
readBuffer(8);    int r = readBuffer(9);
int g = readBuffer(10);    int b =
readBuffer(11);    led.reset(port,slot);
if(idx>0)
        {
            led.setColorAt(idx-1,r,g,b);
        }
        else
        {
            led.setColor(r,g,b);
        }
        led.show();
    }
    break;
case SERVO:{
        int slot = readBuffer(7);
        pin =
slot==1?mePort[port].s1:mePort[port].s2;    int v =
readBuffer(8);

```

```

Servo sv = servos[searchServoPin(pin)];    if(v >= 0 && v <=
180)
{
    if(!sv.attached())
    {
        sv.attach(pin);
    }
    sv.write(v);
}
}
break; case SEVSEG: {
    if(seg.getPort()!=port){    seg.reset(port);
    }
    float v = readFloat(7);    seg.display(v);
}
break;
case LEDMATRIX: {
    if(ledMx.getPort()!=port){    ledMx.reset(port);
    }
    int action = readBuffer(7);    if(action==1){
int px = buffer[8];    int py = buffer[9];    int
len = readBuffer(10);    char *s =
readString(11,len);    ledMx.drawStr(px,py,s);
    }else if(action==2){    int px = readBuffer(8);
int py = readBuffer(9);    uint8_t *ss =
readUInt8(10,16);    ledMx.drawBitmap(px,py,16,ss);
    }else if(action==3){    int point =
readBuffer(8);    int hours = readBuffer(9);
    int minutes = readBuffer(10);

ledMx.showClock(hours,minutes,point);
    }else if(action == 4){
    ledMx.showNum(readFloat(8),3);
    }
}
break;
case LIGHT_SENSOR: {
    if(generalDevice.getPort()!=port){
generalDevice.reset(port);
    }
    int v = readBuffer(7);
    generalDevice.dWrite1(v);
}
break; case IR: {
    String Str_data;    int len = readBuffer(2)-3;
for(int i=0;i<len;i++){
    Str_data+=(char)readBuffer(6+i);
    }
    ir.sendString(Str_data);
    Str_data = "";
}
break; case SHUTTER: {
    if(generalDevice.getPort()!=port){
generalDevice.reset(port);

```

```

}
int v = readBuffer(7);
if(v<2){
    generalDevice.dWrite1(v);
}else{
    generalDevice.dWrite2(v-2);
}
}
break; case DIGITAL: {
pinMode(pin,OUTPUT);    int v =
readBuffer(7);
    digitalWrite(pin,v);
}
break; case PWM: {
    pinMode(pin,OUTPUT);    int v =
readBuffer(7);
    analogWrite(pin,v);
}
break; case TONE: {    int hz = readShort(6);
int tone_time = readShort(8);    if(hz>0){
    buzzer.tone(hz,tone_time);
}else{
    buzzer.noTone();
}
}
break;
case SERVO_PIN: {
    int v = readBuffer(7);
    Servo sv = servos[searchServoPin(pin)];    if(v >= 0 && v
<= 180)
    {
        if(!sv.attached())
        {
            sv.attach(pin);
        }
        sv.write(v);
    }
}
break; case TIMER: {
    lastTime = millis()/1000.0;
}
break;
}
}

int searchServoPin(int pin){    for(int
i=0;i<8;i++){    if(servo_pins[i] == pin){
    return i;
    }
    if(servo_pins[i]==0){    servo_pins[i] = pin;
    return i;
    }
}
return 0;
}

void readSensor(int device){
/******
ff 55    len idx action device port slot data a

```



```

0 1 2 3 4 5 6 7 8
0xff 0x55 0x4 0x3 0x1 0x1 0x1 0xa

*****/
float value=0.0; int port,slot,pin; port = readBuffer(6);
pin = port;
switch(device){
case ULTRASONIC_SENSOR:{
if(us.getPort()!=port){ us.reset(port);
}
value = (float)us.distanceCm(); writeHead();
writeSerial(command_index);
sendFloat(value);
}
break;
case TEMPERATURE_SENSOR:{
slot = readBuffer(7);
if(ts.getPort()!=port||ts.getSlot()!=slot){ ts.reset(port,slot);
}
value = ts.temperature(); sendFloat(value);
}
break; case LIGHT_SENSOR: case
SOUND_SENSOR: case
POTENTIOMETER:{
if(generalDevice.getPort()!=port){
generalDevice.reset(port);
pinMode(generalDevice.pin2(),INPUT);
}
value = generalDevice.aRead2();
sendFloat(value);
}
break;
case JOYSTICK:{ slot = readBuffer(7);
if(joystick.getPort() != port){
joystick.reset(port);
}
value = joystick.read(slot); sendFloat(value);
}
break; case IR:{ if(irReady){
sendString(irBuffer); irReady = false;
irBuffer = "";
}
}
break;
case IRREMOTE:{
unsigned char r = readBuffer(7); if(millis()/1000.0-
lastIRTime>0.2){ sendByte(0);
}else{
sendByte(irRead==r);
}
//irRead = 0;
irIndex = 0;
}
break;

```

```

case IRREMOTECODE:{
if(irRead<0xff){
sendByte(irRead);
}
irRead = 0;
irIndex = 0;
}
break;
case PIRMOTION:{
if(generalDevice.getPort()!=port){
generalDevice.reset(port);
pinMode(generalDevice.pin2(),INPUT);
}
value = generalDevice.dRead2();
sendFloat(value);
}
break;
case LINEFOLLOWER:{
if(generalDevice.getPort()!=port){
generalDevice.reset(port);
pinMode(generalDevice.pin1(),INPUT);
pinMode(generalDevice.pin2(),INPUT);
}
value =
generalDevice.dRead1()*2+generalDevice.dRead2();
sendFloat(value);
}
break;
case LIMITSWITCH:{ slot =
readBuffer(7);

if(generalDevice.getPort()!=port||generalDevic
e.getSlot()!=slot){ generalDevice.reset(port,slot);
}
if(slot==1){

pinMode(generalDevice.pin1(),INPUT_PULL UP);
value = !generalDevice.dRead1();
}else{
pinMode(generalDevice.pin2(),INPUT_PULL UP);
value = !generalDevice.dRead2();
}
sendFloat(value);
}
break;
case BUTTON_INNER:{
//pin = analogs[pin];
pin = pgm_read_byte(&analog[pin]);
char s = readBuffer(7); pinMode(pin,INPUT);
boolean currentPressed = !(analogRead(pin)>10);
sendByte(s^(currentPressed?1:0)); buttonPressed =
currentPressed;
}
break;
case COMPASS:{
if(Compass.getPort()!=port){
Compass.reset(port);

Compass.setpin(Compass.pin1(),Compass.pi n2());

```

```

    }
    sendFloat(Compass.getAngle());
}
break;
case HUMITURE: {
    uint8_t index = readBuffer(7);    if(humiture.getPort()!=port){
        humiture.reset(port);
    }
    uint8_t HumitureData;    humiture.update();
    HumitureData = humiture.getValue(index);
sendByte(HumitureData);
}
break;
case FLAMESENSOR: {
    if(FlameSensor.getPort()!=port){
        FlameSensor.reset(port);

FlameSensor.setpin(FlameSensor.pin2(),FlameSensor.pin1());
    }
    int16_t FlameData;
    FlameData = FlameSensor.readAnalog();
sendShort(FlameData);
}
break;
case GASSENSOR: {
    if(GasSensor.getPort()!=port){
        GasSensor.reset(port);

GasSensor.setpin(GasSensor.pin2(),GasSensor.pin1());
    }
    int16_t GasData;
    GasData = GasSensor.readAnalog();    sendShort(GasData);
}
break; case GYRO: {
    int axis = readBuffer(7);    gyro.update();

```

```

    value = gyro.getAngle(axis);    sendFloat(value);
}
break; case VERSION: {
    sendString(mVersion);
}
break; case DIGITAL: {
pinMode(pin,INPUT);
sendFloat(digitalRead(pin));
}
break;
case ANALOG: {
    //pin = analogs[pin];
    pin = pgm_read_byte(&analogs[pin]);
pinMode(pin,INPUT);    sendFloat(analogRead(pin));
}
break; case TIMER: {
    sendFloat(currentTime);
}
break; case TOUCH_SENSOR:
{
    if(touchSensor.getPort() != port){
touchSensor.reset(port);
    }
    sendByte(touchSensor.touched());
}
break; case BUTTON:
{
    if(buttonSensor.getPort() != port){
buttonSensor.reset(port);
    }
    sendByte(keyPressed == readBuffer(7));
}
break;
}
}
}

```

4.1 A Função pinMode

Essa função configura o pino especificado para funcionar como uma entrada ou saída. Adicionalmente, o modo INPUT explicitamente desativa os resistores pull-up internos.

Sintaxe: pinMode (pino, modo).

Parâmetros:

- pino: o número do pino do Arduino no qual se quer configurar o modo.
- modo: o modo do pino. Este pode ser INPUT, OUTPUT ou INPUT_PULLUP; que correspondem respectivamente a entrada, saída e entrada com pull-up ativado.
- □ Retorna: Nada.

Código de Exemplo:

O código configura o pino digital 13 como OUTPUT e troca seu estado entre HIGH e LOW.

```
void setup() { pinMode(13, OUTPUT); // configura o pino digital 13
              como saída
}

void loop() { digitalWrite(13, HIGH); // ativa o pino digital 13
              delay(1000); // espera por um segundo
              digitalWrite(13, LOW); // desativa o pino digital 13
              delay(1000); // espera por um segundo
}
```

4.2 Compreendendo o código do segue linha

O código a seguir está dentro da função readSensor, que faz a leitura dos sensores do mBot. As informações são obtidas a partir de um buffer com 8 elementos, sendo que a porta é o elemento de número 6, ela é armazenada na variável port. O trecho destacado primeiro verifica se a porta

obtida do dispositivo é diferente da porta recebida, e se for reinicia a porta e coloca dois pinos em modo de entrada (pin1 e pin2), em seguida realiza o cálculo do valor a ser enviado com base na leitura usando a função dRead1 e dRead2 essas funções fazem parte do mBot por meio da biblioteca MePort.

```
case LINEFOLLOWER:{
  if(generalDevice.getPort()!=port){    generalDevice.reset(port);
    pinMode(generalDevice.pin1(),INPUT);
    pinMode(generalDevice.pin2(),INPUT);
  }
  value = generalDevice.dRead1()*2+generalDevice.dRead2();    sendFloat(value);
}
```

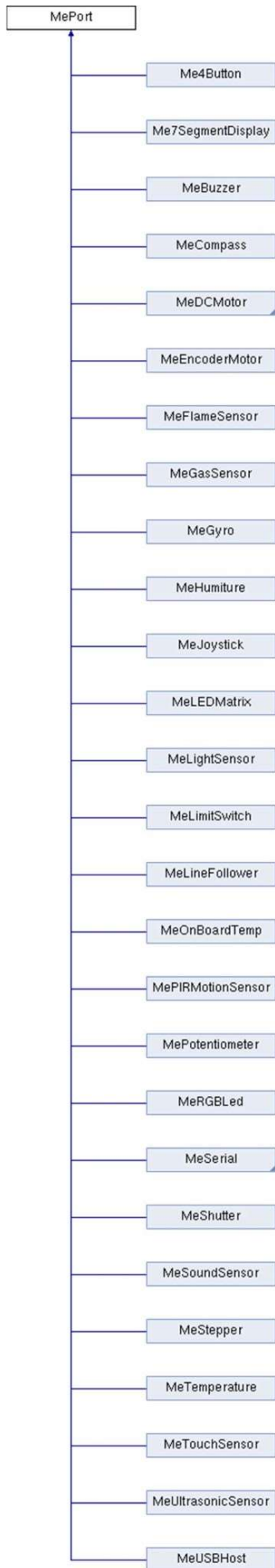
O dRead1 lê um valor digital no slot1 da porta RJ25 do objeto corrente. Como parâmetro recebe o modo (INPUT, INPUT_PULLUP). Ela retorna um valor de entrada. Já dRead2 lê o slot 2.

4.2 Alterando o código fonte do robô

4.2.1 Acendendo um led quando encontra um obstáculo

Para isso, foram adicionadas duas linhas ao sensor ultrassônico.

```
case ULTRASONIC_SENSOR:{
  if(us.getPort()!=port){    us.reset(port);
  }
  value = (float)us.distanceCm();    writeHead();
  writeSerial(command_index);
  led.setColor(1,0,0);
  led.show();
  sendFloat(value);
}
break;
```

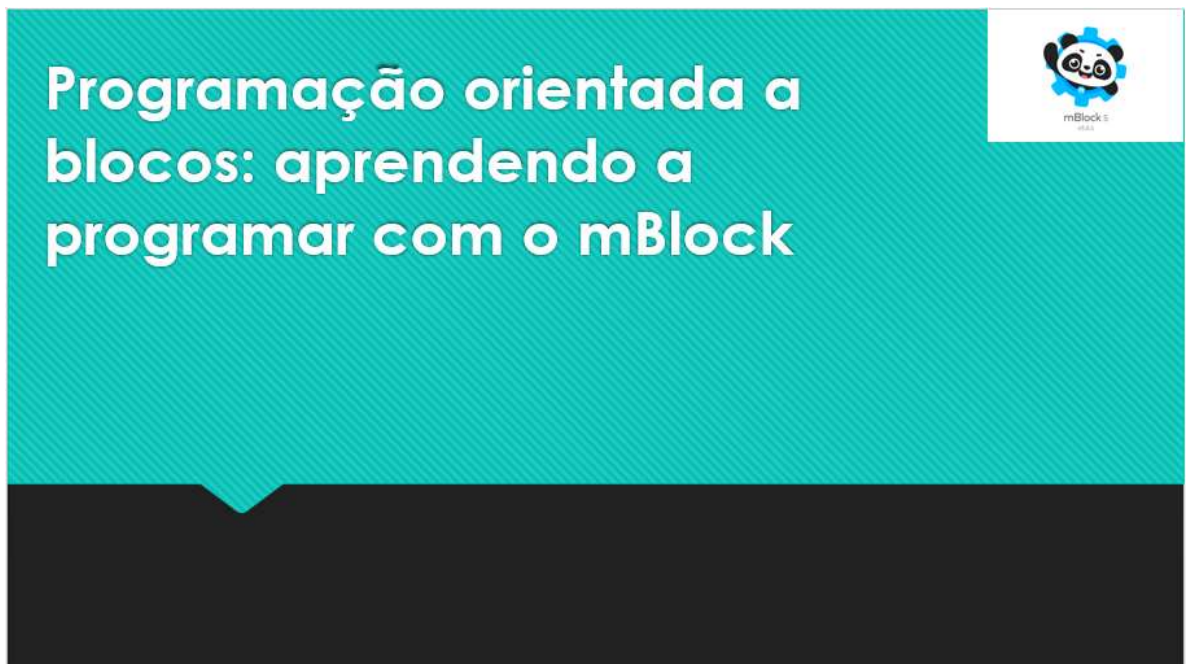


CAPÍTULO 5: APRESENTANDO A PROGRAMAÇÃO ORIENTADA A BLOCOS PARA ESTUDANTES DO ENSINO FUNDAMENTAL

Foi desenvolvida uma aula com slides sobre o mBlock e através do Google Meet essa aula foi ministrada para estudantes de no mínimo 14 anos

- Primeiramente, foi feito slides explicativos sobre o funcionamento do mBlock.

Figura 51: Slide inicial 1 da aula.



- Em seguida a aula foi ministrada, a duração foi de 100 minutos. Foram explicados conceitos tais como o mBot, o mBlock e as categorias de blocos foram apresentadas, conforme o presente material.

Figura 52: Slide inicial 2 da aula.

O que é mBlock?



- É uma ferramenta de software versátil para o ensino.
- Com o mBlock, vocês podem projetar histórias, jogos e animações atraentes e programar alguns dispositivos como o mBot.



Imagem do robô mBot

- Na aula estavam presentes duas crianças: uma de 13 anos e uma de 14 anos. Houve a participação de ambas.
- Ao final da aula foi proposto um pequeno desafio.

Figura 53: Desafio proposto.

Agora é a sua vez!



Escolha um cenário bem bonito e faça um joguinho para o ator "panda" pegar a bola ou qualquer objeto da sua escolha.

5.1 Conclusão sobre o experimento

O desafio proposto foi avaliado da seguinte forma: 10 pontos para o uso do bloco “se”, 20 pontos para o uso do bloco “repetir para sempre” e 30 pontos para o uso do bloco “criar uma variável”, totalizando 60 pontos.

As duas estudantes foram avaliadas com a nota final 30. Assim, pode-se concluir que há uma dificuldade maior com o uso do bloco “criar uma variável”.