შეხვედრა 12: შეგრძნებიდან მოქმედებამდე — ავტონომიური სისტემები

გამარჯობა! ჩვენ უკვე ვისწავლეთ, თუ როგორ შევაგროვოთ ინფორმაცია სენსორებით და როგორ ვმართოთ აქტუატორები. დღეს კი ამ ცოდნას გავაერთიანებთ და შევქმნით პროექტს, სადაც არდუინო დამოუკიდებლად რეაგირებს გარემოზე. ეს არის მომენტი, როდესაც ჩვენი პროექტი უბრალოდ ბრძანებების შემსრულებლიდან ნამდვილ „ჭკვიან“ მოწყობილობად იქცევა.

1. სისტემის სრული ციკლი

1.1. მონაცემთა ნაკადის გააზრება: სენსორი -> არდუინო (კოდი) -> აქტუატორი

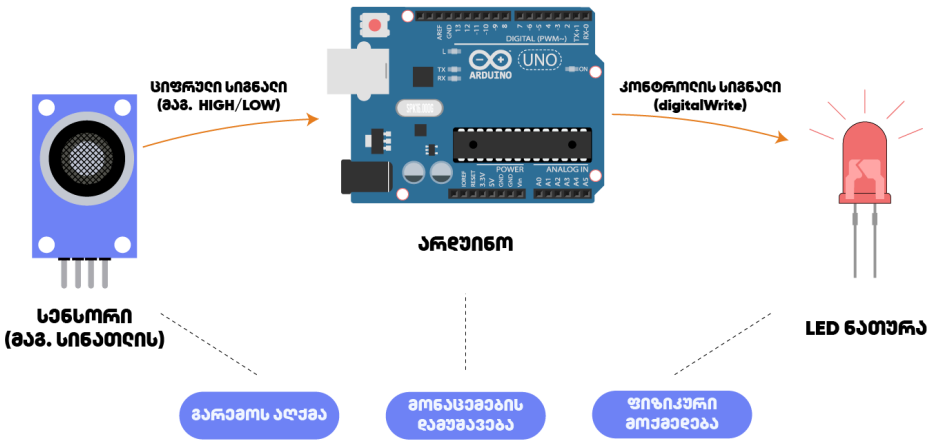
როგორ ფიქრობს არდუინო? წარმოიდგინე, როგორ მუშაობს ადამიანის რეფლექსი:

1. **შეგრძნება (სენსორი):** შენი თვალი ხედავს, რომ ბურთი შენსკენ მოფრინავს.
2. **დამუშავება (არდუინო):** შენი ტვინი ამუშავებს ინფორმაციას და ხვდება, რომ ბურთი შენ მოგხვდება.
3. **მოქმედება (აქტუატორი):** შენი ტვინი უგზავნის ბრძანებას ხელს, რომ ბურთი აიცილოს.

ჩვენი პროექტი ზუსტად იგივე პრინციპით მუშაობს. ეს არის სრული ციკლი:

- **სენსორი** აგროვებს ინფორმაციას გარემოდან.
- **არდუინო** (ჩვენი კოდის საშუალებით) ამუშავებს ამ ინფორმაციას და იღებს გადაწყვეტილებას.
- **აქტუატორი** ასრულებს ფიზიკურ მოქმედებას ამ გადაწყვეტილების საფუძველზე.

  ეს არის ყველა რობოტისა და ჭკვიანი მოწყობილობის მუშაობის ფუნდამენტური პრინციპი.



სავარჯიშო:

შემდეგი მოქმედებები შეუსაბამე სწორ კომპონენტს:

ცხელი ობიექტის შეხება და ცხელების შეგრძნება

ტვინში ინფორმაციის მიღება: "ობიექტი ცხელია!"

ტვინის გადაწყვეტილება: "ხელი უნდა მოვაშორო!"

ყურით ძლიერი ხმაურის გაგონება

ტვინში ინფორმაციის დამუშავება: "ხმაური ძალიან ძლიერია!"

ტვინის გადაწყვეტილება: "ყურები უნდა დავიცვა!"

ტვინში ინფორმაციის დამუშავება: "სინათლე ძალიან ინტენსიურია!"

ტვინის გადაწყვეტილება: "უნდა დავიცვა თვალები!"

თვალების დახამხამება

1.2. რეალურ დროში რეაგირების კონცეფცია

რატომ არის ეს ციკლი ასეთი მნიშვნელოვანი? იმიტომ, რომ ის **რეალურ დროში** ხდება. არდუინო არ ფიქრობს წუთობით. ის კითხულობს სენსორს, ამუშავებს მონაცემს და ააქტიურებს აქტუატორს წამის მცირე მონაკვეთში.

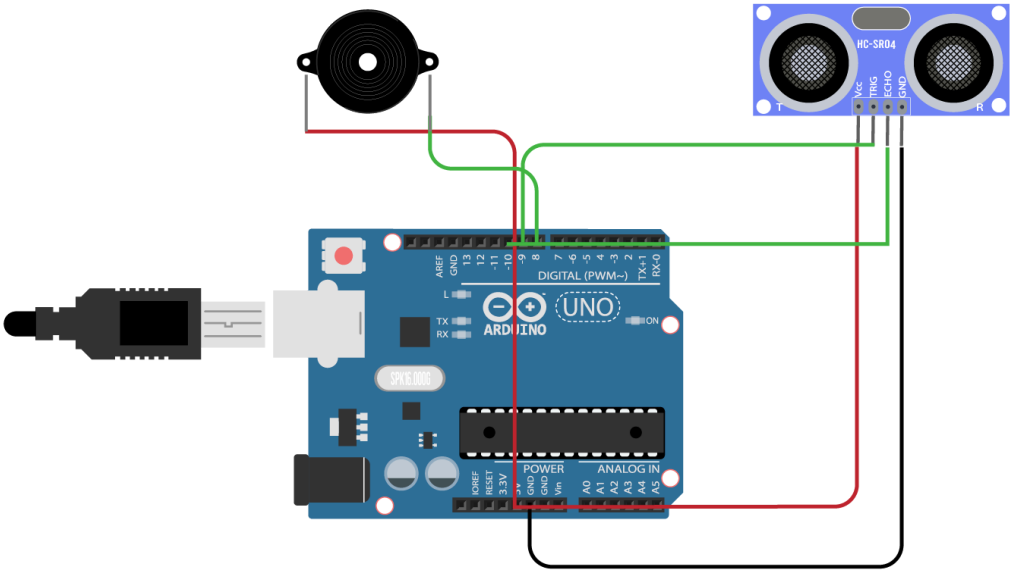
სწორედ ეს მყისიერი რეაგირება ხდის ჩვენს პროექტებს ინტერაქტიულს. მანქანის პარკირების ასისტენტი, რომელიც დაბრკოლებაზე 2 წამის დაგვიანებით მოგვცემდა სიგნალს, სრულიად უსარგებლო იქნებოდა. რეალურ დროში რეაგირება არის ის, რაც განასხვავებს ჭკვიან მოწყობილობას ჩვეულებრივისგან.

1.3. სენსორისა და აქტუატორის ერთ სქემაში გაერთიანება

AnITa-ს პლატფორმაზე ჩვენი სამუშაო გამარტივებულია – საჭირო სენსორები და აქტუატორები უკვე დაკავშირებულია არდუინოს შესაბამის პინებთან.

მნიშვნელოვანია გვესმოდეს, რომ ისინი ერთიან სისტემას ქმნიან. მაგალითად, დღევანდელ პროექტში:

- **მანძილის სენსორი** (HC-SR04) დაკავშირებულია ციფრულ პინებთან (მაგ. 9 და 10).
- **პიეზო ზუმერი** დაკავშირებულია სხვა ციფრულ პინთან (მაგ. 8).
- ისინი ერთად მუშაობენ ერთი „ტვინის“, ჩვენი არდუინოს, მეთვალყურეობის ქვეშ.



2. პირობითი ლოგიკის გამოყენება აქტუატორის სამართავად

2.1. `if/else` პირობების გამოყენება აქტუატორის ჩართვა/გამორთვისთვის

ჩვენ უკვე ვიცით, როგორ გამოვიყენოთ `if/else` სერიულ მონიტორზე ტექსტის დასაბეჭდად. ახლა ჩვენ იმავე ლოგიკას გამოვიყენებთ, ოღონდ ტექსტის ნაცვლად, აქტუატორს ვმართავთ.

ლოგიკა ძალიან მარტივია:

```
if (პირობა სრულდება) {
  // ჩართე აქტუატორი (მაგ. tone(8, 500);)
} else {
  // გამორთე აქტუატორი (მაგ. noTone(8);)
}
```

2.2. პროექტის ალგორითმის დაგეგმვა: რა შემთხვევაში რა მოქმედება უნდა შესრულდეს

სანამ კოდის წერას დავიწყებთ, უმჯობესია ყოველთვის შევიმუშაოთ ჩვენი პროექტის ალგორითმი და შევადგინოთ ფსევდო კოდი. ალგორითმის შემუშავება დავიწყოთ ორი მარტივი კითხვით:

1. **რა იქნება მოქმედების გამომწვევი ფაქტორი (Trigger)?** ანუ, რა პირობა უნდა შესრულდეს, რომ მოქმედება დაიწყოს? (მაგალითად: „მანძილი 20 სმ-ზე ნაკლებია“).
2. **რა მოქმედება უნდა შესრულდეს (Action)?** ანუ, რა უნდა გააკეთოს აქტუატორმა, როცა ეს პირობა სრულდება? (მაგალითად: „ზუმერმა უნდა გამოსცეს ხმა“).

როცა ამ ორ კითხვაზე პასუხი გვაქვს, კოდის დაწერა ბევრად მარტივია.

2.3. კოდის შემუშავება Input-Output ლოგიკის გათვალისწინებით

კოდის სტრუქტურა `loop()` ფუნქციაში, როგორც წესი, ასეთია:

```
void loop() {
  // 1. წაიკითხე მონაცემი სენსორიდან.
  // 2. შეასრულე საჭირო გამოთვლები (მაგ. მანძილის გამოთვლა).
  // 3. გამოიყენე if/else, რათა შეადარო შედეგი ზღვარს.
  // 4. if ბლოკში ჩაწერე აქტუატორის ჩართვის ბრძანება.
  // 5. else ბლოკში ჩაწერე აქტუატორის გამორთვის ბრძანება.
}
```

3. პრაქტიკული მაგალითები

მოდი, განვიხილოთ რამდენიმე ისეთი იდეა, სადაც ეს ციკლი მუშაობს.

3.1. ავტომობილის ფარების ავტომატური ჩამრთველი: თუ ბნელა, აანთე შუქდიოდი.

სენსორი: ფოტორეზისტორი (სინათლის სენსორი).

აქტუატორი: შუქდიოდი.

ალგორითმი:

```
int lightValue = analogRead(A0);
if (lightValue > 700) { // თუ ბნელა
  digitalWrite(13, HIGH); // აანთე შუქდიოდი
} else {
  digitalWrite(13, LOW); // გამორთე შუქდიოდი
}
```

3.2. განგაშის სისტემის ალგორითმი: თუ ობიექტი ახლოსაა, გამოსცეს ხმა ზუმერით

სენსორი: ულტრაბგერითი მანძილის სენსორი.

აქტუატორი: პიეზო ზუმერი.

ალგორითმი:

```
float distance = calculateDistance(); // მანძილის გამოთვლის ფუნქცია
if (distance < 20) { // თუ ობიექტი 20 სმ-ზე ახლოსაა
  tone(8, 1000); // ჩართე ზუმერი
} else {
  noTone(8); // გამორთე ზუმერი
}
```

ეს არის ზუსტად ის ალგორითმი, რომელსაც ჩვენს დღევანდელ პრაქტიკულ დავალებაში გამოვიყენებთ.

3.3. ტემპერატურის კონტროლის ალგორითმი: თუ ცხელა, ჩართე ძრავის გამაგრილებელი (ვენტილატორი)

სენსორი: TMP36 (ტემპერატურის სენსორი).

აქტუატორი: DC ძრავა (ვენტილატორი).

ალგორითმი:

```
float temperature = calculateTemperature(); // ტემპერატურის გამოთვლის ფუნქცია

if (temperature > 28) { // თუ 28 გრადუსზე ცხელა
  digitalWrite(7, HIGH); // ჩართე ძრავა
} else {
  digitalWrite(7, LOW); // გამორთე ძრავა
}
```

ხშირად არის შემთხვევა, როდესაც პროექტი ერთზე მეტ აქტუატორს იყენებს. ამ შემთხვევაში გვჭირდება ჩვენთვის კარგად ცნობილი ლოგიკური ოპერატორები. მაგალითად, თუ მოცემული გვაქვს ფოტორეზისტორი და ტემპერატურის სენსორი, მაშინ მისთვის შედარების ოპერატორი დაიწერება შემდეგნაირად:

```
if (lightValue > lightThreshold && temperatureC > tempThreshold) {
```

რომელ ოპერატორს გამოვიყენებთ დამოკიდებულია იმაზე, თუ რა ლოგიკა აქვს ჩვენს კოდს.

დავალება 12: „პარკინგის ასისტენტი“

შენი ამოცანაა, დაწერო პროგრამა, რომელიც მუდმივად ზომავს მანძილს ობიექტამდე. თუ ობიექტი 20 სანტიმეტრზე ახლოს მოვა, ზუმერმა უნდა გამოსცეს გამაფრთხილებელი ხმოვანი სიგნალი. თუ ობიექტი შორსაა, ზუმერი ჩუმად უნდა იყოს.

დავალების სიმულაცია tinkercadში:

https://drive.google.com/drive/folders/1pzRu7t3LTUaN4cxNjfQZ7rOiu0-UBz_S //12.1

დავალება 12.1: შეცდომის პოვნა და გასწორება

პროგრამისტმა კოდის წერისას შეცდომა დაუშვა. if პირობაში მან შედარების ოპერატორის (<) ნაცვლად მნიშვნელობის მინიჭების ოპერატორი (=) გამოიყენა, რის გამოც პროგრამის ლოგიკა არასწორად მუშაობს. შენი ამოცანაა, იპოვო და გაასწორო ეს შეცდომა.

მინიშნება: გაიხსენე, რომელი სიმბოლო გამოიყენება ორი მნიშვნელობის შესადარებლად და რომელი – ცვლადისთვის მნიშვნელობის მისანიჭებლად.

```
int trigPin = 9;
int echoPin = 10;
```

```
int buzzerPin = 8;

void setup() {
  pinMode(trigPin, OUTPUT);
  pinMode(echoPin, INPUT);
  pinMode(buzzerPin, OUTPUT);
}

void loop() {
  digitalWrite(trigPin, LOW);
  delayMicroseconds(2);
  digitalWrite(trigPin, HIGH);
  delayMicroseconds(10);
  digitalWrite(trigPin, LOW);

  long duration = pulseIn(echoPin, HIGH);
  float distance = (duration * 0.0343) / 2;

  // იპოვე შეცდომა ამ პირობაში
  if (distance = 20) {
    tone(buzzerPin, 1000);
  } else {
    noTone(buzzerPin);
  }

  delay(100);
}
```

დავალება 12.2: კოდის დასრულება

პროგრამის ძირითადი ნაწილი უკვე აგებულია, კოდი ზომავს მანძილს. თუმცა, მთავარი ნაწილი – `if/else` კონსტრუქცია, რომელიც მანძილს ამოწმებს და ზუმერს რთავს/თიშავს – გამორჩენილია. შენი ჯერია! დაამატე `if/else` ბლოკი, რათა პროგრამა დასრულდეს.

```
int trigPin = 9;
int echoPin = 10;
int buzzerPin = 8;

void setup() {
  pinMode(trigPin, OUTPUT);
  pinMode(echoPin, INPUT);
  pinMode(buzzerPin, OUTPUT);
```

```
}

void loop() {
  digitalWrite(trigPin, LOW);
  delayMicroseconds(2);
  digitalWrite(trigPin, HIGH);
  delayMicroseconds(10);
  digitalWrite(trigPin, LOW);

  long duration = pulseIn(echoPin, HIGH);
  float distance = (duration * 0.0343) / 2;

  // --- ჩაამატე if/else ბლოკი აქ ---
  // 1. შეამოწმე, თუ distance ნაკლებია 20-ზე.
  // 2. თუ პირობა სრულდება, ჩართე ზუმერი (tone).
  // 3. სხვა შემთხვევაში, გამორთე ზუმერი (noTone).

  delay(100);
}
```

დავალება 12.3: შეიმუშავე პროგრამული კოდი
დაწერე პროგრამული კოდი, შექმენი პროგრამა, რომელიც შეასრულებს შემდეგ სამუშაოს:

1. განსაზღვრავს Trig, Echo და Buzzer პინებს.
2. setup ფუნქციაში მოამზადებს პინების რეჟიმებს.
3. loop ფუნქციაში გაზომავს მანძილს.
4. if/else პირობით შეამოწმებს, არის თუ არა მანძილი 20-ზე ნაკლები და, შესაბამისად, ჩართავს ან გამორთავს ზუმერს.

```
void setup() {
  // დაწერე setup ფუნქციის კოდი აქ.
}

void loop() {
  // დაწერე loop ფუნქციის კოდი აქ.
}
```

სწორი პასუხი (პროგრამული კოდი სრულად):

```
// პინების განსაზღვრა
```

```
int trigPin = 9;
int echoPin = 10;
int buzzerPin = 8;

void setup() {
  // პინების რეჟიმების განსაზღვრა
  pinMode(trigPin, OUTPUT);
  pinMode(echoPin, INPUT);
  pinMode(buzzerPin, OUTPUT);
}

void loop() {
  // 1. გააგზავნე ულტრაბგერითი იმპულსი.
  digitalWrite(trigPin, LOW);
  delayMicroseconds(2);
  digitalWrite(trigPin, HIGH);
  delayMicroseconds(10);
  digitalWrite(trigPin, LOW);

  // 2. გაზომე ექოს დაბრუნების დრო და გამოთვალე მანძილი სანტიმეტრებში.
  long duration = pulseIn(echoPin, HIGH);
  float distance = (duration * 0.0343) / 2;

  // 3. გამოიყენე if/else კონსტრუქცია.
  //    თუ მანძილი 20-ზე ნაკლებია...
  if (distance < 20) {
    // ...ჩართე ზუმერი (გამოიყენე tone() ფუნქცია).
    tone(buzzerPin, 1000);
  } else {
    // ...სხვა შემთხვევაში, გამორთე ზუმერი (გამოიყენე noTone() ფუნქცია).
    noTone(buzzerPin);
  }

  delay(100); // მცირე პაუზა გაზომვებს შორის
}
```

JSON

```
{
  "version": 1,
```

```
  "board": "arduino:avr:uno",
  "steps": [
    {
      "type": "compile"
    },
    {
      "type": "static",
      "rules": [
        {
          "id": "pin_trig_output",
          "name": "TRIG pin mode",
          "kind": "require_regex",
          "pattern":
"pinMode\\s*\\(\\s*(?:\\btrigPin\\b|\\bTRIG(?:_PIN)?\\b|9(?!\\d))
\\s*,\\s*OUTPUT\\s*\\)\\s*;?",
          "flags": "iu",
          "must_pass": true,
          "source": "stripped",
          "msg_fail": "trigPin (პინი 9) უნდა იყოს OUTPUT რეჟიმში:
pinMode(trigPin, OUTPUT);",
          "msg_pass": "TRIG პინი სწორად არის OUTPUT რეჟიმში."
        },
        {
          "id": "pin_echo_input",
          "name": "ECHO pin mode",
          "kind": "require_regex",
          "pattern":
"pinMode\\s*\\(\\s*(?:\\bechoPin\\b|\\bECHO(?:_PIN)?\\b|10(?!\\d)
)\\s*,\\s*INPUT\\s*\\)\\s*;?",
          "flags": "iu",
          "must_pass": true,
          "source": "stripped",
          "msg_fail": "echoPin (პინი 10) უნდა იყოს INPUT რეჟიმში:
pinMode(echoPin, INPUT);",
          "msg_pass": "ECHO პინი სწორად არის INPUT რეჟიმში."
        },
        {
          "id": "pin_buzzer_output",
          "name": "Buzzer pin mode",
          "kind": "require_regex",
```

```
          "pattern":
"pinMode\\s*\\(\\s*(?:\\bbuzzerPin\\b|\\bBUZZER(?:_PIN)?\\b|8(?!\
\d))\\s*,\\s*OUTPUT\\s*\\)\\s*;?",
          "flags": "iu",
          "must_pass": true,
          "source": "stripped",
          "msg_fail": "buzzerPin (პინი 8) უნდა იყოს OUTPUT
რეჟიმში: pinMode(buzzerPin, OUTPUT);",
          "msg_pass": "Buzzer პინი სწორად არის OUTPUT რეჟიმში."
        },
        {
          "id": "trig_pulse_seq",
          "name": "TRIG pulse sequence",
          "kind": "require_ordered_regex",
          "patterns": [

"digitalWrite\\s*\\(\\s*(?:\\btrigPin\\b|9(?!\\d))\\s*,\\s*LOW\\s
*\\)",
            "delayMicroseconds\\s*\\(\\s*\\d+\\s*\\)",

"digitalWrite\\s*\\(\\s*(?:\\btrigPin\\b|9(?!\\d))\\s*,\\s*HIGH\\
s*\\)",
            "delayMicroseconds\\s*\\(\\s*\\d+\\s*\\)",

"digitalWrite\\s*\\(\\s*(?:\\btrigPin\\b|9(?!\\d))\\s*,\\s*LOW\\s
*\\)"
          ],
          "flags": "iu",
          "must_pass": true,
          "source": "stripped",
          "msg_fail": "TRIG-ზე უნდა გაგზავნოთ იმპულსი: LOW →
delay → HIGH → delay → LOW.",
          "msg_pass": "TRIG იმპულსის თანმიმდევრობა სწორია."
        },
        {
          "id": "pulsein",
          "name": "Measure echo with pulseIn",
          "kind": "require_regex",
          "pattern":
"pulseIn\\s*\\(\\s*(?:\\bechoPin\\b|10(?!\\d))\\s*,\\s*HIGH\\s*\\
)",
```

```
          "flags": "iu",
          "must_pass": true,
          "source": "stripped",
          "msg_fail": "echoPin-ის პულსი უნდა გაიზომოს:
pulseIn(echoPin, HIGH);",
          "msg_pass": "echoPin-ის გაზომვა გამოყენებულია
pulseIn()-ით."
        },
        {
          "id": "distance_formula",
          "name": "Distance formula",
          "kind": "require_regex",
          "pattern":
"\\b(distance|dist)\\b\\s*=\\s*[^;]*duration[^;]*0\\.034\\d*[^;]*
/\\s*2\\b",
          "flags": "iu",
          "must_pass": true,
          "source": "stripped",
          "msg_fail": "მანძილი გამოთვალეთ სწორად: distance =
(duration * 0.0343) / 2;",
          "msg_pass": "მანძილის ფორმულა სწორადაა."
        },
        {
          "id": "threshold_20",
          "name": "Threshold check for 20cm",
          "kind": "require_regex",
          "pattern":
"if\\s*\\(\\s*distance\\s*<\\s*20(?:\\.0+)?\\s*\\)",
          "flags": "iu",
          "must_pass": true,
          "source": "stripped",
          "msg_fail": "შედარება უნდა იყოს: if (distance < 20)",
          "msg_pass": "20 სმ-ის ზღვარი სწორადაა გამოყენებული."
        },
        {
          "id": "else_structure",
          "name": "else statement exists",
          "kind": "require_regex",
          "pattern": "\\}\\s*else\\s*\\{",
          "flags": "iu",
          "must_pass": true,
```

```
          "source": "stripped",
          "msg_fail": "გამოიყენეთ else ბლოკი buzzer-ის
გასამორთად.",
          "msg_pass": "else ბლოკი არსებობს."
        },
        {
          "id": "loop_sequence",
          "name": "Correct loop sequence",
          "kind": "require_ordered_regex",
          "patterns": [

"digitalWrite\\s*\\(\\s*(?:\\btrigPin\\b|9(?!\\d))\\s*,\\s*LOW\\s
*\\)",

"pulseIn\\s*\\(\\s*(?:\\bechoPin\\b|10(?!\\d))\\s*,\\s*HIGH\\s*\\
)",
            "if\\s*\\(\\s*distance\\s*<\\s*20(?:\\.0+)?\\s*\\)"
          ],
          "flags": "iu",
          "must_pass": true,
          "source": "stripped",
          "msg_fail": "loop()-ში თანმიმდევრობა: TRIG პულსი →
pulseIn → if შედარება.",
          "msg_pass": "loop() თანმიმდევრობა სწორია."
        },
        {
          "id": "if_then_tone",
          "name": "tone() in if block",
          "kind": "require_ordered_regex",
          "patterns": [
            "if\\s*\\(\\s*distance\\s*<\\s*20(?:\\.0+)?\\s*\\)",

"tone\\s*\\(\\s*(?:\\bbuzzerPin\\b|8(?!\\d))\\s*,\\s*\\d+\\s*\\)"
          ],
          "flags": "iu",
          "must_pass": true,
          "source": "stripped",
          "msg_fail": "როცა distance < 20, if ბლოკში უნდა იყოს
tone(buzzerPin, ...);",
          "msg_pass": "if ბლოკში buzzer სწორად ირთვება
tone()-ით."
```

```
        },
        {
          "id": "else_then_notone",
          "name": "noTone() in else block",
          "kind": "require_ordered_regex",
          "patterns": [
            "\\}\\s*else\\s*\\{",

"noTone\\s*\\(\\s*(?:\\bbuzzerPin\\b|8(?!\\d))\\s*\\)"
          ],
          "flags": "iu",
          "must_pass": true,
          "source": "stripped",
          "msg_fail": "else ბლოკში უნდა იყოს noTone(buzzerPin);",
          "msg_pass": "else ბლოკში buzzer სწორად ირთვება
noTone()-ით."
        }
      ]
    }
  ]
}
```

Sim Elements HTML

```
<div style="display: flex; gap: 30px; margin-bottom: 20px; align-items:
center; justify-content:
 center;">
```

```
    <div style="display: flex; flex-direction: column; align-items:
center; gap: 10px;">
        <wokwi-hc-sr04 id="ultrasonic" distance="100"></wokwi-hc-sr04>
        <label style="font-weight: bold;">Ultrasonic Sensor
(HC-SR04)</label>
    </div>

    <div style="display: flex; flex-direction: column; align-items:
center; gap: 10px;">
        <wokwi-buzzer id="buzzer-8" pin="8"></wokwi-buzzer>
    </div>
 </div>

 <div class="distance-control">
    <label>Distance (cm): <span
data-ultrasonic-display></span></label>
    <input
        type="range"
        data-ultrasonic-distance
        min="2"
        max="400"
        class="distance-slider"
    />
    <div class="distance-labels">
        <span>2 cm (Very Close)</span>
        <span>400 cm (Far)</span>
    </div>
 </div>
```

Sim Elements js

```
return {
    onInit: function(runner, sensorValues, ultrasonicDistance, buzzer)
{
```

```
        // Initialize distance if not set
        if (ultrasonicDistance.value === undefined) {
            ultrasonicDistance.value = 50;
        }

        // Set up automatic buzzer detection (monitors Timer1/Timer2
for tone() calls)
        buzzer.setupBuzzerSimulation();
    }
};
```