CZU: 371.3: 004.8

FUNDAMENTELE STRATEGICE PENTRU DEZVOLTAREA CONCEPTULUI STEAM ÎN CADRUL LABORATORULUI INTELIGENȚA ARTIFICIALĂ CREATIVĂ

Dorin AFANAS, dr., conf. univ.
Universitatea de Stat din Tiraspol, Moldova

Rezumat. În prezentul articol sunt evidențiate unele probleme care persistă în învățământul național din Republica Moldova. Sunt propuse unele soluții pentru diminuarea lor. Una din aceste soluții ar fi integrarea învățământului STEAM prin intermediul lucrărilor de laborator, având la bază proiectul Arduino.

Abstract. This article highlights some problems that persist in national education in the Republic of Moldova. Some solutions are proposed to reduce them. One of these solutions would be the integration of STEAM education through laboratory work, based on the Arduino project.

Cuvinte cheie: Arduino, circuit electric, codul programului, argument.
Keywords: Arduino, electrical circuit, program code, argument.

1. Introducere

La momentul actual există un șir de exemple când tinerii abandonează specialități aferente disciplinelor matematica, fizica, chimia, etc. argumentând că ele sunt prea complicate și necaptivante.

O soluție pentru a spori atractivitatea față de disciplinele menționate ar putea fi introducerea treptată a învățământului STEAM, mai ales că asemenea experiențe reușite există deja în alte țări dezvoltate. Pentru creșterea atractivității față de așa discipline ca matematica, fizica, chimia, biologia, geografia și nu în ultimul rând față de informatică, ne poate veni în ajutor proiectul Arduino. Acest proiect, pe lângă costul redus ia în considerație și dificultatea învățării, adică proiectul dat nu necesită mari investiții financiare și nu necesită cunoștințe profunde de programare pentru crearea diferitor lucrări în funcție de creativitatea studenților.

Arduino este o companie open-source care produce atât plăcuțe de dezvoltare bazate pe microcontrolere, cât și partea de software destinată funcționării și programării acestora. Pe lângă acestea include și o comunitate uriașă care se ocupă cu creația și distribuirea de proiecte care au ca scop crearea de dispozitive care pot sesiza și controla diverse activități sau procese în lumea reală [1].

Proiectul este bazat pe designul plăcilor cu microcontroler produse de câțiva furnizori, folosind diverse tipuri de microcontrolere. Aceste plăci pun la dispoziția utilizatorului pini digitali și analogici, care pot fi interfațați cu o gamă largă de plăcuțe numite scuturi (shield-uri) și/sau cu alte circuite. Plăcile au interfețe de comunicații seriale, inclusiv USB pe unele modele, pentru a încărca programe din calculatorele personale. Pentru programarea microcontrolerelor, Arduino vine cu un mediu de dezvoltare integrat (IDE) bazat pe proiectul Processing, care include suport pentru limbaje de programare ca C și C++.

Primul Arduino a fost lansat în 2005, având ca țintă asigurarea unei soluții ieftine și simple pentru începători și profesioniști spre a crea dispozitive capabile să interacționeze cu mediul, folosind senzori și sisteme de acționare. Cele mai comune exemple sunt dispozitivele pentru utilizatorii începători precum: roboții simpli, termostatele și/sau detectoarele de mișcare.

Plăcuțele Arduino sunt disponibile comercial sub formă preasamblată sau sub forma unor kituri de asamblat acasă (do-it-yourself). Specificațiile schemelor sunt disponibile pentru orice utilizator, permițând oricui să fabrice plăcuțe Arduino. Adafruit Industries estimase la mijlocul anului 2011 că peste 300.000 de plăcuțe oficiale Arduino au fost produse [2], iar în anul 2013 peste 700.000 de plăcuțe oficiale erau în posesia utilizatorilor [3].

Prezentăm în continuare un model pentru lucrare de laborator prin intermediul căruia studentul are posibilitatea să facă cunoștință cu unele noțiuni aferente disciplinelor fizica, informatica și matematica, noțiuni care pot fi însușite și conștientizate fără mari dificultăți.

2. Crearea circuitului electric și a programului respectiv la calculator

În cadrul acestui proiect vom realiza un circuit ce conține:

- fire pentru conexiune;
- trei rezistoare de 220 Ω;
- un rezistor de 10 kΩ;
- un buton (întrerupător);
- două LED-uri roșii;
- un LED verde.

La finele acestei activități vom cunoaște cum putem crea un program simplu pentru circuitul electric construit.

Obiectivele acestei activități sunt:

- conștientizarea intrărilor digitale;
- conștientizarea ieșirilor digitale;
- formarea competenței de a construi un circuit electric;
- conștientizarea tipurilor de prezentare ale circuitelor electrice;
- conștientizarea necesității reprezentării circuitelor electrice prin două moduri;
- conștientizarea instrucțiunilor utilizate;
- conștientizarea variabilelor utilizate;
- formarea competenței de a crea programul la calculator aferent circuitului electric construit.

Se recomandă de realizat activitatea după următoarea schemă/următorul algoritm:

2.1. Noțiuni fundamentale. În acest proiect, vom construi ceva care mai mult se poate asemăna cu o interfață a navei spațiale într-un film științifico-fantastic din anii 1970. Vom construi un circuit cu mai multe LED-uri care luminează atunci când apăsăm pe un

comutator/buton. Un LED verde va fi aprins, până când apăsăm un buton. Când Arduino primește un semnal de la buton, lumina verde se va stinge și alte 2 lumini roșii vor începe să clipească.

Pinii digitali Arduino pot citi doar două stări, atunci când există tensiune pe un pin de intrare și când nu există. Acest tip de intrare este numit în mod normal digital (sau uneori binar, pentru două stări). Aceste stări sunt denumite în mod obișnuit HIGH și LOW. HIGH este același lucru dacă am spune „există tensiune aici” și LOW înseamnă că nu există tensiune pe acest pin. Când utilizăm un pin OUTPUT HIGH vom folosi o comandă numită **digitalWrite()** pentru a-l porni. Valoarea tensiunii dintre pin și masă este de 5 V. Când dorim să oprim un pin OUTPUT folosim comanda LOW. Pinii digitali Arduino pot acționa atât ca intrări, cât și ca ieșiri în codul creat la calculator. Vom putea configura pinii în funcție de ceea ce dorim să obținem. Când pinii sunt ieșiri, putem porni componente precum LED-uri. Dacă configurăm pinii ca intrări, atunci putem verifica dacă un comutator este sau nu apăsat. Deoarece pinii 0 și 1 sunt utilizați pentru comunicarea cu computerul, se recomandă de început cu pinul 2.

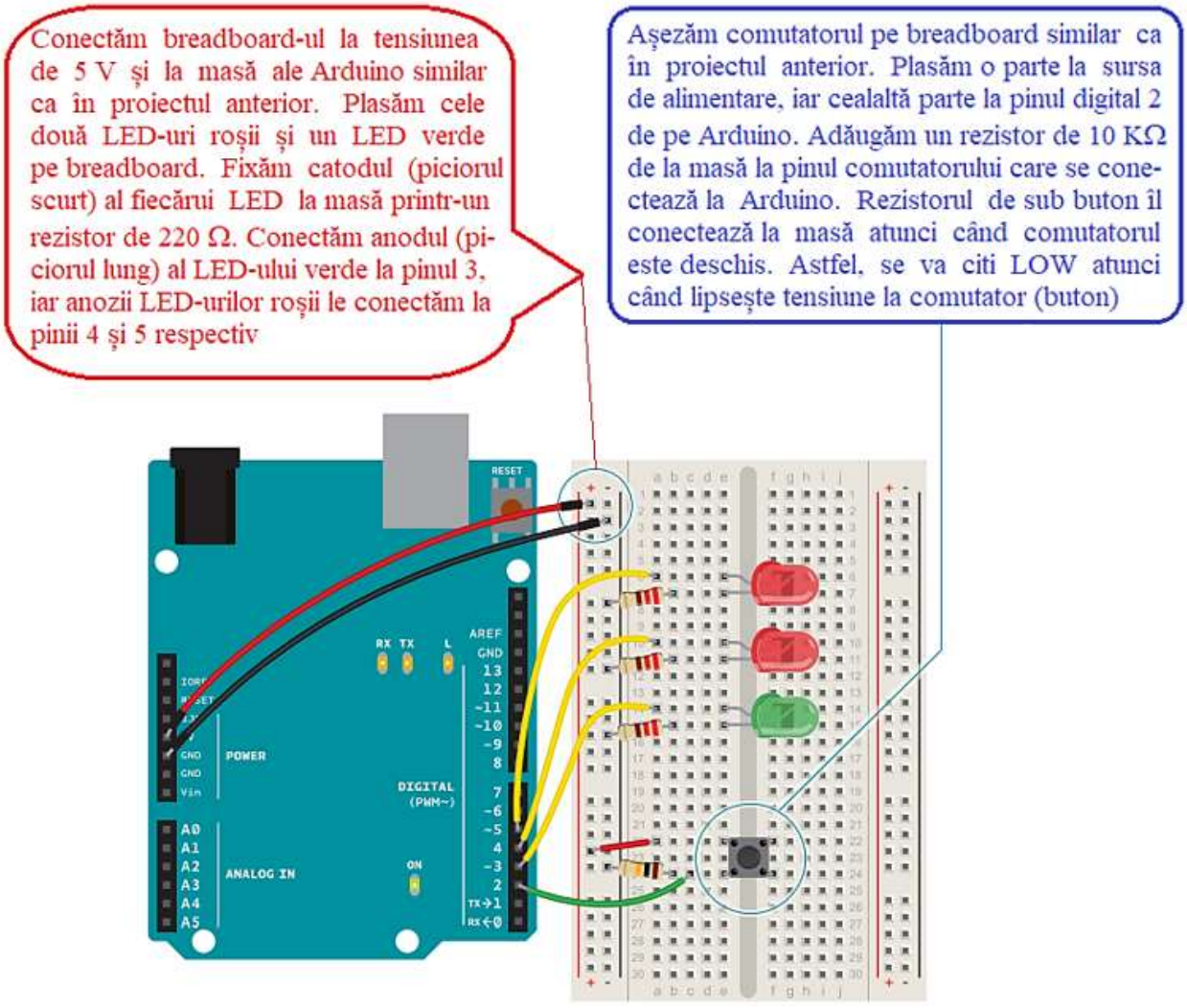


Figura 1. Ilustrația circuitului

2.2. Prezentarea circuitului. Se recomandă de prezentat circuitul în două moduri: o vizualizare pe panou (fig. 1) numită *ilustrația circuitului*, așa cum arată componentele real și altul schematic (fig. 2) numită *prezentare schematică*, care este un mod mai abstract de a arăta relațiile dintre componente într-un circuit. Schemele nu arată întotdeauna unde componentele sunt plasate în realitate, ci arată mai mult cum sunt conectate între ele.

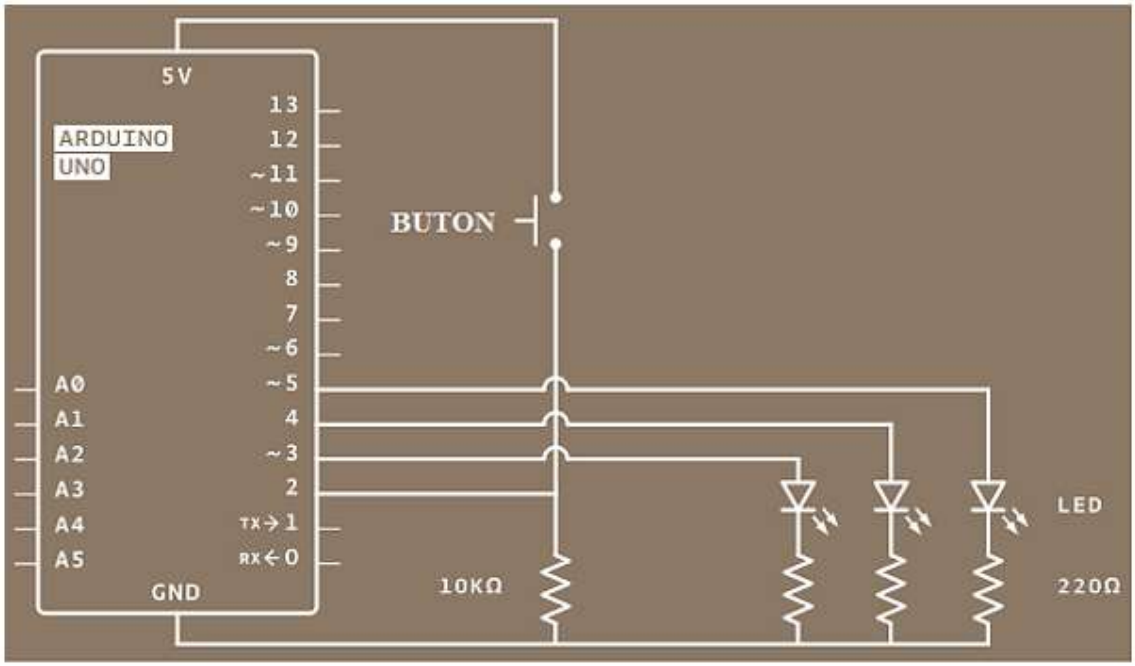
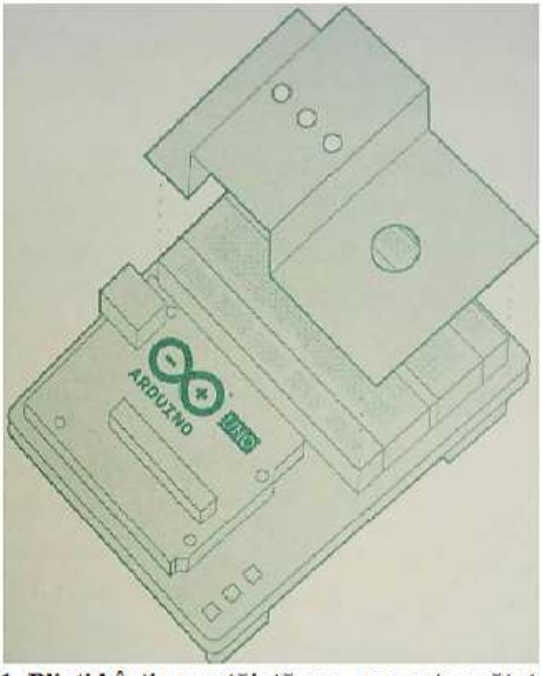
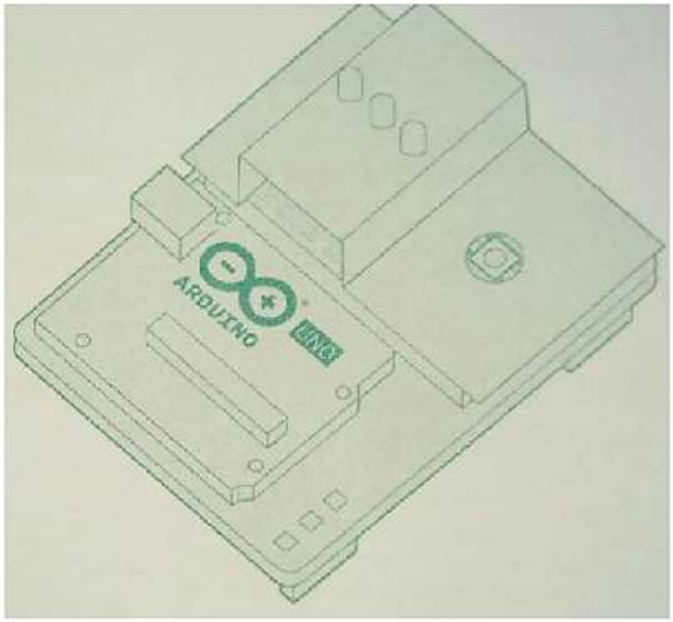


Figura 2. Prezentarea schematică a circuitului

Putem acoperi breadboard-ul sau îl putem decora pentru a crea propriul sistem de lansare (fig. 3). Luminile aprinse și oprite nu înseamnă nimic de la sine, dar când le amplasăm într-un panou de control și le dăm etichete, ele capătă sens. Ce dorim să indice LED-ul verde? Ce înseamnă LED-urile roșii intermitente? Aici noi decidem!



1. Pliați hârtia pre-tăiată așa cum este arătat



2. Așezăm pliatul peste breadboard. Led-urile și butonul vor ajuta la menținerea pliatului în poziția dorită

Figura 3.

2.3. Codul programului. Fiecare schiță (sketch) Arduino are două funcții principale. Funcțiile sunt părți ale unui program de computer care rulează comenzi specifice. Funcțiile au nume unice și sunt „numite” când este nevoie. Funcțiile necesare dintr-o schiță (sketch) Arduino se numesc **setup()** și l**oop()**. Aceste funcții trebuie declarate, ceea ce înseamnă că trebuie să-i indicăm lui Arduino ce vor face cu ele. Operatorii **setup()** și **loop()** sunt declarați așa cum este indicat mai jos:

```
void setup(){
}
void loop(){
}
```

Orice cod pe care îl scriem în parantezele figurate (acolade) va fi executat atunci când funcția este apelată.

În acest program, vom crea o variabilă înainte de a intra în partea principală a lui. Variabilele sunt nume pe care le vom da locurilor din memoria Arduino, astfel încât să putem urmări ceea ce se întâmplă. Aceste valori se pot schimba în funcție de instrucțiunile programului creat. Numele variabilelor ar trebui să fie descriptive indiferent de valoarea pe care o stochează. De exemplu, o variabilă numită **switchState** va spune ce stochează: starea unui switch. Pe de altă parte, variabila numită "*x*" nu va spune prea multe despre ce stochează.

Pentru a crea o variabilă, trebuie să declarăm tipul ei. Tipul de date **int** va conține un număr întreg (numit și întreg). Când declarăm o variabilă, de obicei îi acordăm și o valoare inițială. Declarația variabilei ca fiecare instrucțiune trebuie să se încheie cu punct și virgulă (;) așa cum este prezentat mai jos:

```
1   int switchState = 0;
```

Operatorul **setup()** rulează o dată, când Arduino este pornit pentru prima oară. Aici configurăm pinii digitali pentru a fi intrări sau ieșiri utilizând o funcție numită **pinMode()**. Pinii conectați la LED-uri vor fi ieșiri (OUTPUTs), iar pinul de comutare va fi intrare (INPUT). Trebuie să acordăm o atenție deosebită literelor majuscule și celor minuscule din codul nostru. De exemplu, **pinMode()** este numele unei comenzi, iar **pinmode()** va genera o eroare. Astfel, codul în continuare va fi:

```
2   void  setup(){
3   pinMode(3, OUTPUT);
4   pinMode(4, OUTPUT);
5   pinMode(5, OUTPUT);
6   pinMode(2, INPUT);
7   }
```

Operatorul **loop()** rulează continuu după finalizarea configurării **setup()**. Operatorul **loop()** este locul în care vom verifica tensiunea la intrări și vom activa/dezactiva ieșirile.

Pentru a verifica nivelul de tensiune pe o intrare digitală, vom utiliza funcția **digitalRead()** care verifică tensiunea pin-ului ales. Pentru a cunoaște ce pin să verificăm, funcției **digitalRead()** îi atribuim în mod obligatoriu un argument. Argumentele sunt informații pe care le transmitem funcțiilor, spunându-le cum ar trebui să funcționeze. De exemplu, **digitalRead()** are nevoie de un argument: ce pin trebuie să verifice. În programul nostru **digitalRead()** va verifica starea pin-ului 2 și se va stoca valoarea în variabila **switchState**. Dacă există tensiune pe pin când se apelează **digitalRead()**, variabila **switchState** va obține valoarea HIGH (sau 1). Dacă nu există tensiune pe pin, **switchState** va obține valoarea LOW (sau 0).

Dacă vom dori să includem careva comentarii în limbajul natural în programul nostru, atunci o putem realiza. Comentariile sunt note pe care le lăsăm pentru noi și pe care computerul le ignoră. Pentru a scrie un comentariu, este suficient să adăugăm două bare oblice //. Computerul va ignora orice inscripție după cele două bare oblice și programul în continuare poate fi:

```
8   void  loop(){
9   switchState = digitalRead (2);
10  // acesta este un comentariu
```

Vom folosi cuvântul ”if” pentru a verifica starea căruiva dispozitiv (de exemplu, poziția butonului (comutatorului)). O afirmație **if()** din programare compară două lucruri și determină dacă comparația este adevărată sau falsă. Apoi efectuează acțiunile pe care i le-am spus să le facă. Când comparăm două lucruri în programare, utilizăm două semne egale ”==”. Dacă utilizăm un singur semn, atunci vom seta o valoare în loc să o comparăm. Pașii 11 și 12 ai codului sunt arătați mai jos:

```
11  if (switchState == LOW) {
12  // butonul nu este apăsat
```

Uneori, poate fi util să scriem fluxul programului în pseudocod: un mod de a descrie ceea ce dorim să facă programul într-un limbaj simplu, dar structurat într-un mod care face mai ușoară scrierea unui program real din acesta. În asemenea cazuri vom determina dacă **switchState** este HIGH (adică butonul este apăsat) sau nu. Dacă comutatorul este apăsat, vom opri LED-ul verde și cel roșu aprins. În pseudocod, declarația ar putea arăta astfel: dacă **switchState** este LOW: porniți LED-ul verde, dezactivați LED-urile roșii dacă **switchState** este HIGH: opriți LED-ul verde, activați LED-urile roșii, etc.

Comanda **digitalWrite()** ne permite să trimitem 5V sau 0V la un pin de ieșire. Această comandă poate lua două argumente: ce pin să controlăm și ce valoare să setăm acelui pin: HIGH sau LOW. Dacă dorim să pornim LED-urile roșii și LED-ul verde în interiorul declarației **if()**, atunci codul ar arăta astfel:

```
13  digitalWrite(3, HIGH); // LED-ul verde
14  digitalWrite(4, LOW); // LED-ul roșu
15  digitalWrite(5, LOW); // LED-ul roșu
16  }
```

După ce am scris codul pentru Arduino când comutatorul (butonul) este deschis, vom defini ce se întâmplă când comutatorul este închis. Instrucțiunea **if()** are o componentă opțională **else** care permite să se întâmple ceva dacă condiția inițială nu este îndeplinită. În acest caz, după ce am verificat dacă comutatorul a fost LOW, scriem codul pentru condiția HIGH după instrucțiunea **else.** Pentru ca LED-urile roșii să clipească când butonul este apăsat, va trebui să oprim și să aprindem luminile în declarația **else**. Pentru a realiza acest lucru, vom schimba codul pentru a arăta astfel:

```
17  else {    // butonul este apăsat
18  digitalWrite(3, LOW);
19  digitalWrite(4, LOW);
20  digitalWrite(5, HIGH);
```

Dacă rulăm programul acum, atunci luminile se vor schimba când apăsăm butonul (comutatorul). Pentru o ieșire mai interesantă se recomandă de adăugat ceva mai multă complexitate programului.

După ce am setat LED-urile într-o anumită stare, vom dori ca Arduino să se întrerupă pe un moment, înainte de a le schimba înapoi. Dacă nu facem schimbări în cod (nu dorim să așteptăm), luminile se vor aprinde și se vor stinge atât de repede încât ochiul nostru va sesiza numai niște lumini puțin slabe, dar nu aprinse și oprite. Acest lucru se datorează faptului că Arduino trece prin **loop()** de mii de ori în fiecare secundă, iar LED-ul va fi aprins și oprit mai repede decât putem percepe. Funcția **delay()** ne permite să oprim Arduino de la executarea unei comenzi pentru o perioadă de timp. Funcția **delay()** ia un argument care determină numărul de milisecunde înainte de a executa următorul set de cod. Se cunoaște că o secundă conține 1000 de milisecunde. Prin urmare, dacă scriem **delay(250)**, atunci vom avea o întârziere de un sfert de secundă, adică pentru a se executa următorul set de cod va trebui să așteptăm un sfert de secundă. Codul va arăta astfel:

```
21  delay(250);    // așteptăm un sfert de secundă
22  // comutăm LED-urile
23  digitalWrite(4, HIGH);
24  digitalWrite(5, LOW);
25  delay(250);    // așteptăm un sfert de secundă
26  }
27  } // revenim la începutul buclei (loop)
```

Dacă rulăm acum programul, atunci LED-urile roșii vor clipi când butonul va fi apăsat.

Odată ce Arduino este programat, ar trebui să vedem lumina verde aprinsă. Când apăsăm comutatorul, luminile roșii vor începe să clipească, iar lumina verde se va stinge. Încercăm să modificăm timpul celor două funcții **delay()** și realizăm observări ce se întâmplă cu luminile și cum se schimbă răspunsul sistemului în funcție de viteza intermitentă. Când apelăm **delay()** (întârziere) în programul nostru, acesta oprește toate celelalte funcționalități. Nici o citire a senzorului nu va avea loc până când nu va trece perioada respectivă. Deși întârzierile sunt deseori utile, se recomandă ca atunci când vom crea propriile proiecte să ne asigurăm dacă acestea nu interferează inutil cu interfața noastră.

Sketch-ul Arduino integral este prezentat în figura 4.

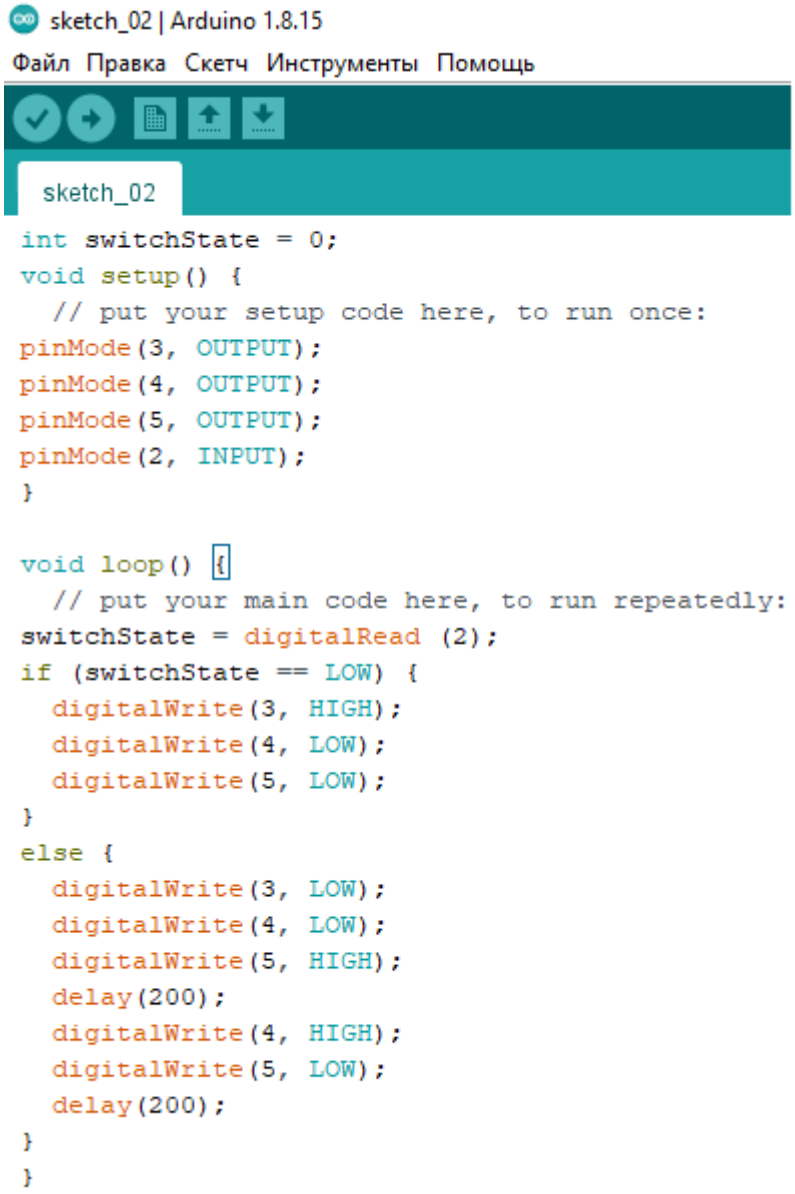


Figura 4. Sketch-ul Arduino integral

În continuare pot fi realizate activități care pot răspunde la întrebările:

– Cum am face ca LED-urile roșii să clipească când începe programul ?

– Cum am putea face o interfață mai mare sau mai complexă pentru proiectul nostru cu LED-uri și comutatoare ?

Se recomandă să luăm în considerație așteptările studenților/cursanților, atunci când începem să creăm o interfață pentru proiectul dorit:

– Când vor apăsa un buton, vor dori feedback imediat sau nu ?
– Ar trebui să existe o întârziere între acțiunea lor și ceea ce face Arduino ? Încercați și plasați-vă în locul altui utilizator în timp ce proiectați și vedeți dacă așteptările cadrului didactic se potrivesc cu realitatea proiectului creat.

3. Recapitularea comenzilor utilizate

Constante utilizate sunt patru:

– **HIGH** (5V sau 1);
– **LOW** (0V sau 0);
– **INPUT** (INTRARE);
– **OUTPUT** (IEȘIRE).

Variabile utilizate sunt două:

– Pinii digitali Arduino pot citi doar două stări, atunci când există tensiune pe un pin de intrare și când nu există. Din aceste considerente ei pot fi considerați, în sens matematic, ca variabile ce pot lua două valori: **HIGH** și **LOW**;
– **int switchState** poate lua numai valori întregi **HIGH** (sau 1) și **LOW** (sau zero). În informatică însuși **int** poartă numele *tip de date*.

Funcții utilizate sunt șase:

– **digitalWrite()** ne permite să trimitem 5V sau 0V la un pin de ieșire; este o funcție de două variabile, pe care în sens matematic le putem nota cu *x* și *y*; *x* indică ce pin controlăm (2 – 13), iar *y* – ce tensiune putem trimite acelui pin: 5V (**HIGH**) sau 0V (**LOW**); domeniul de definiție al funcției **digitalWrite()** conține 24 de elemente, iar domeniul valorilor admisibile conține numai două elemente: 5V (**HIGH** sau ”există tensiune”) și 0V (**LOW** sau ”nu există tensiune”);
– **pinMode()** ne permite să configurăm pinii digitali ca intrări sau ieșiri; este o funcție de două variabile pe care simbolic le notăm cu *x* și *y*; *x* indică numărul pinului digital (2 – 13), iar *y* indică starea lui (INPUT sau OUTPUT); domeniul de definiție al funcției **pinMode()** conține 24 de elemente, iar domeniul valorilor admisibile va conține numai două elemente: INPUT (INTRARE) și OUTPUT (IEȘIRE);
– **digitalRead()** ne permite să verificăm nivelul tensiunii la un pin de intrare; este o funcție de o singură variabilă (numărul pinului); domeniul de definiție al funcției **digitalRead()** conține 12 elemente; domeniul valorilor admisibile conține două elemente: ”există tensiune”, ”nu există tensiune”;

- **delay()** ne permite să oprim executarea unei comenzi pentru o perioadă de timp (măsurat în milisecunde); este o funcție de o singură variabilă; domeniul de definiție cât și domeniul valorilor admisibile reprezintă mulțimea numerelor naturale (evident mărginită superior); este o funcție bijectivă;
- **setup()** este o funcție de mai multe variabile. În informatică poartă denumirea de *operator*.
- **loop()** este o funcție de mai multe variabile (este o buclă); în programare, putem folosi o instrucțiune specială care repetă o parte din cod; spunem că această construcție este o buclă și că programul efectuează un ciclu repetitiv. În informatică poartă denumirea de *operator*.

Concluzii

În rezultatul unor asemenea activități pot fi rezolvate următoarele probleme:

- formarea competențelor tineretului studios în domeniul STEAM;
- mărirea interesului față de științele exacte, mai ales față de fizică, informatică și matematică;
- diminuarea costului ridicat al utilajelor care este folosit în învățământul STEAM;
- posibilitatea susținerii financiare a unui asemenea învățământ de către stat;
- diminuarea abandonării de către tineret a specialităților aferente disciplinelor fizica, informatica și matematica.

Acest articol a fost elaborat în cadrul proiectului de cercetări științifice „Metodologia implementării TIC în procesul de studiere a științelor reale în sistemul de educație din Republica Moldova din perspectiva inter/transdisciplinarității (concept STEAM)”, Programul „Program de stat” (2020- 2023), Prioritatea IV: Provocări societale, cifrul 20.80009.0807.20.

BIBLIOGRAFIE

1. „Arduino - Introduction”. arduino.cc.
2. „How many Arduinos are "in the wild?" About 300,000”. Adafruit Industries. 15 mai 2011.
3. „Arduino FAQ – With David Cuartielles”. Malmö University. 5 aprilie 2013.