Numele proiectului : Tetris

Autor: Hrincă Robert-Mateiaș

Scopul jocului: Tetris este un joc tip puzzle care are ca scop completarea liniilor orizontale de la baza tablei de joc. Liniile trebuie completate fără spații libere pentru a câștiga puncte. Odată completată o linie în acest fel, aceasta dispare de pe tabla de joc. Cu ajutorul unor forme geometrice se pot completa liniile, fiind permisă schimbarea orientării acestora. Formele seamănă cu unele litere precum J, S, L, O, Z. Jocul se termină atunci când nu mai exista loc în partea superioara a tablei de joc pentru piese, deci atunci când jucătorul nu a mai putut completa linii.

Descrierea proiectului: Jucătorul va interacționa cu o matrice de LED-uri 16x32 (care de fapt sunt două matrici de 8x16) ce va arăta jocul în sine, o matrice de LED-uri 8x8 care va arăta următoarea piesă ce va fi pusă în joc, un display care va arăta scorul jucătorului, un buzzer care va emite un sunet de fiecare dată când o piesă de joc își schimbă orientarea și atunci când scorul este mărit, deci când o linie este completată și 4 butoane care permit controlarea piesei de joc. Prin tasta W se schimbă orientarea piesei de joc, prin tasta A și D se mută piesa în stânga respectiv dreapta iar prin S piesa se va muta în partea inferioară a matricii mai repede.

Componente: 1x Arduino MEGA

1x Buzzer

5x LED 8x8 Dot Matrix

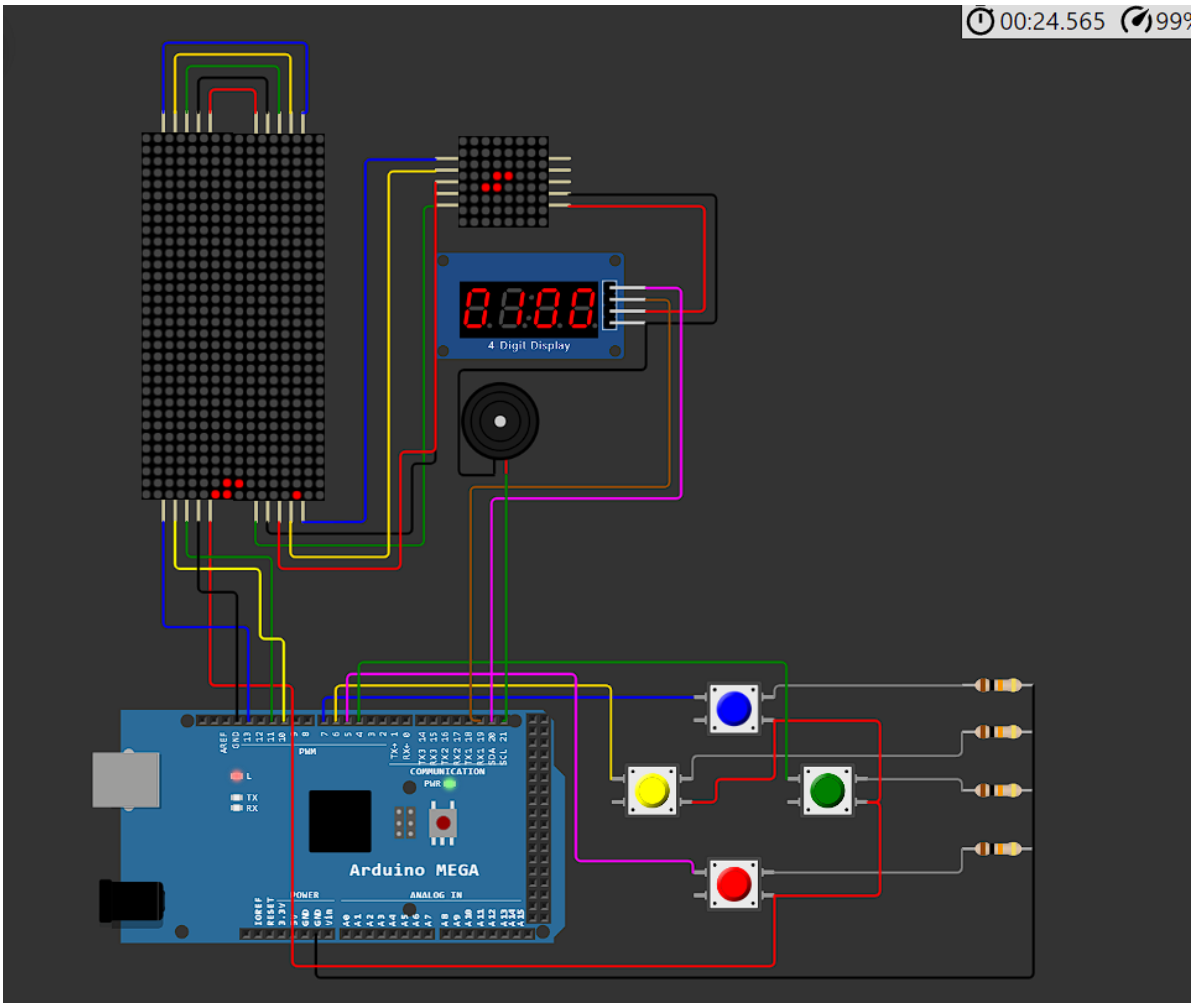
1x Seven segment LED display

4x Push bottons

4x Resistors

Conexiuni: Arudino MEGA – sunt folosiți pinii : GND, 13, 11 ,10 pentru prima matrice de 8x16

- prima matrice se conectează la a doua matrice prin pinii acesteia de V+, GND , DIN, CS, CLK
- matricea de 8x8 se conectează la a doua matrice de 8x16 prin pinii V+, GND , DIN, CS, CLK
- Seven segment LED display este conectat la: matricea de 8x8 prin GND și VCC ale matricei și la Arduino MEGA prin pinii 19,20
- Buzzerul este conectat la pin-ul 21 al Arudino MEGA si GND al Seven segment LED display
- Butoanele se conectează la terminalii rezistorilor, la GND sau alimentare
- Rezistoarele sunt conectate la GND în serie



Cod: varianta completă este în arhivă, voi explica funcțiile care nu se repetă

struct-ul pentru Punct are rolul de a ține coordonatele a mai multor componente ale formei geometrice, iar Piesa definește prin Punct centrul formei geometrice, offset-ul este folosit pentru coordonatele LED-urilor care se vor aprinde pentru a forma simbolul geometric, rotatie este folosit pentru a determina rotația formei și tip-ul memorează forma geometrică (S, I, J, Z....) a actualei piese de joc

```
struct Punct {
  int x, y;
};

struct Piesa {
  Punct centru;
  Punct offset[4];
  int rotatie;
  char tip;
};
```

assignPunct pune în thisPunct coordonatele altui punct

addPuncte returnează suma a două puncte

```
void assignPunct(Punct& thisPunct, const Punct& altul) {
  thisPunct.x = altul.x;
  thisPunct.y = altul.y;
}

Punct addPuncte(const Punct& punct1, const Punct& punct2) {
  Punct result;
  result.x = punct1.x + punct2.x;
  result.y = punct1.y + punct2.y;
  return result;
}
```

assignPiesa copiază centru-ul, offset-ul, rotația și tip-ul unei piese de joc în alta

assignTip copiază offset-ul unei piese în alta

```
void assignPiesa(Piesa& thisPiesa, const Piesa& alta) {
  thisPiesa.centru = alta.centru;
  for(int i = 0; i < 4; ++i) {
    thisPiesa.offset[i] = alta.offset[i];
  }
  thisPiesa.rotatie = alta.rotatie;
  thisPiesa.tip = alta.tip;
```

```
}

void assignTip(Piesa& thisPiesa, const Piesa& alta) {

  for(int i = 0; i < 4; ++i) {
    thisPiesa.offset[i] = alta.offset[i];
  }

}
```

rotestePiesa are rolul de a roti piesa de joc când se apasă W prin luarea din altă funcție a offset-ului corespunzător piesei rotite

```
Piesa rotestePiesa(Piesa thisPiesa) {
  Piesa rez = thisPiesa;
  rez.centru = thisPiesa.centru;
  if(thisPiesa.tip == 'I')
  {
    assignTip(rez,I_block(rez.rotatie));
  }
  if(thisPiesa.tip == 'J')
  {
    assignTip(rez,J_block(rez.rotatie));
  }

……..


  if(thisPiesa.rotatie == 3){
    rez.rotatie = 0;
  }

  rez.rotatie ++;
  thisPiesa.rotatie = rez.rotatie;

  return rez;
}
```

Spre exemplu pentru piesa I de aici se vor lua formele rotite ale acesteia

```
Piesa I_block(int x){

  static Piesa i_block_0 {{7,1}, {{-1, 0}, {0, 0}, {1, 0}, {2, 0}} };
  i_block_0.tip = 'I';
  static Piesa i_block_90 {{7,1} ,{{0, -1}, {0, 0}, {0, 1}, {0, 2}} };
  i_block_90.tip = 'I';
  static Piesa i_block_180 {{7,1}, {{-1, 0}, {0, 0}, {1, 0}, {2, 0}} };
  i_block_180.tip = 'I';
  static Piesa i_block_270 {{7,1} , {{0, -1}, {0, 0}, {0, 1}, {0, 2}} };
  i_block_270.tip = 'I';

  static Piesa piese[] = {i_block_90 , i_block_180, i_block_270, i_block_0};

  return piese[x];
}
```

Funcția show() iterează prin matriciile matrice_next_piesă (8x8) și coloane(16x32) pentru a trimite informația necesară parinderii LED-urilor

```
void show() {
  for (char index = 7; index >= 0; --index) {
    digitalWrite(SS, LOW);
    shiftOut(DIN, CLK, MSBFIRST, 8 - index);
    shiftOut(DIN, CLK, MSBFIRST, matrice_next_piesa[index]);

    for (char value = 0; value < Y_SEGMENTS; ++value) {
      shiftOut(DIN, CLK, MSBFIRST, index + 1);
      shiftOut(DIN, CLK, LSBFIRST, coloane[value][8 + index]);
    }
    for (char value = Y_SEGMENTS - 1; value >= 0; --value) {
      shiftOut(DIN, CLK, MSBFIRST, 8 - index);
      shiftOut(DIN, CLK, MSBFIRST, coloane[value][index]);
    }
    digitalWrite(SS, HIGH);
  }
}
```

Functia reset() are rolul de a pregăti matriciile pentru următoarea iterație a jocului

```
void reset() {
  for (int i = 0; i < 8; ++i) {
    matrice_next_piesa[i] = 0x00;
  }
  for (int i = 0; i < Y_SEGMENTS; ++i) {
    for (int j = 0; j < 8 * X_SEGMENTS; ++j) {
      coloane[i][j] = 0x00;
    }
  }
}
```

arataPiesa() monitorizează evoluția piesei de joc în matrice

arataNextPiesa() pregătește de afișare piesa următoare de joc

```
void arataPiesa() {
  for (int i = 0; i < 4; i++) {
    Punct pozitie;
    assignPunct(pozitie, addPuncte(piesa.centru, piesa.offset[i]));
    coloane[(31 - pozitie.y) / 8][pozitie.x] |= 1 << (pozitie.y % 8);
  }
}

void arataNextPiesa() {
  Punct centruModificat = {3, 3};
  for (int i = 0; i < 4; i++) {
    Punct pozitie;
    assignPunct(pozitie, addPuncte(centruModificat, nextPiesa.offset[i]));
    matrice_next_piesa[ pozitie.y] |= 1 << (7 - pozitie.x % 8);
  }
}
```

arataTabla() se ocupă cu starea curentă a jocului, apelează funcțiile de mai sus și în funcțiile de valorile găsite in matricea valori va da update la jocul de pe ecran matricea găsită in show(), coloane

```
void arataTabla() {
  reset();
  arataPiesa();
  arataNextPiesa();
  for (int i = 0; i < 32; ++i) {
    for (int j = 0; j < 16; ++j) {
      if (valori[i][j]) {
        coloane[(31 - i) / 8][j] |= 1 << (i % 8);
      }
    }
  }
}
```

piesaNoua() asignează random o formă piesei de joc ce urmează

```
Piesa piesaNoua() {
  static Piesa i_block {{7, 1} ,{{-1, 0}, {0, 0}, {1, 0}, {2, 0}}};
  i_block.tip = 'I';
  static Piesa j_block {{7, 1}, {{-1, 0}, {-1, 1}, {0, 1}, {1, 1}} };
  j_block.tip = 'J';
  static Piesa l_block {{7, 1} ,{{0, 1}, {-1, 1}, {1, 0}, {1, 1}}};
  l_block.tip = 'L';
  static Piesa o_block {{7, 1} ,{{0, 0}, {1, 0}, {0, 1}, {1, 1}}};
  o_block.tip = 'O';
  static Piesa s_block {{7, 1},{{0, 0}, {1, 0}, {-1, 1}, {0, 1}}};
  s_block.tip= 'S';
  static Piesa t_block {{7, 1},{{0, 0}, {-1, 1}, {0, 1}, {1, 1}}};
  t_block.tip = 'T';
  static Piesa z_block {{7, 1},{{0, 0}, {-1, 0}, {0, 1}, {1, 1}}};
  z_block.tip = 'Z';

  static Piesa piese[] = {j_block, l_block, o_block, s_block, t_block, z_block,
i_block};

  return piese[random(6)];
}
```

valideazaMiscare() se asigură că piesa nu depășește granițele matricii

```
int valideazaMiscare(Piesa const& piesa, int miscare) {

  for (int i = 0; i < 4; ++i) {

    Punct pozitie;
    assignPunct(pozitie, addPuncte(piesa.centru, piesa.offset[i]));

    if (miscare == 1 && (pozitie.y == 32 || valori[pozitie.y][pozitie.x])) {
      return 0;
    }

    if (miscare != 1 && (pozitie.y == 32 || valori[pozitie.y][pozitie.x])) {
      return -1;
    }

    if (pozitie.x < 0 || pozitie.x > 15 || pozitie.y < 0 || pozitie.y > 31) {
      return -1;
    }
  }
  return 1;
}
```

detecteazaButon() dă update la coordonatele piesei de joc în caz de este mutată

```
int detecteazaButon() {
  Piesa copie;
  int mutata = false;
  int miscare = 0;
  if (digitalRead(W)) {

    copie = rotestePiesa(piesa);
    tone(BP, 10, 1000);
    mutata = true;
  } else {

    assignPiesa(copie,piesa);
    if (digitalRead(D)) {
      copie.centru.x --;
      mutata = true;
    }
```

```
    if (digitalRead(A)) {
      copie.centru.x ++;
      mutata = true;
    }
    if (digitalRead(S)) {
      copie.centru.y ++;
      mutata = true;
      miscare = 1;
    }
  }

  int miscareValida = valideazaMiscare(copie, miscare);
  if (mutata && miscareValida > 0) {
    assignPiesa(piesa,copie);
  }

  return miscareValida;
}
```

setup() se specificp pinii de ieșire, se setează luminozitatea display-ului cu 7 segmente și se inițializează primele piese de joc

```
void setup() {
  Serial.begin(115200);

  display.setBrightness(7);
  randomSeed(analogRead(0));

  pinMode(CLK, OUTPUT);
  pinMode(DIN, OUTPUT);
  pinMode(SS, OUTPUT);
  pinMode(BP, OUTPUT);


  Piesa new_piesa = piesaNoua();
  new_piesa.rotatie = 0;
  assignPiesa(piesa, new_piesa);


  Piesa new_piesa1 = piesaNoua();
  new_piesa1.rotatie = 0 ;
```

```
  assignPiesa(nextPiesa, new_piesa1);

  for (int j = 0; j < 13; ++j){
    valori[31][j] = 1;
  }

}
```

verificaRanduri() verifică dacă o linie a fost completată, mărește scorul și face buzzer-ul sa emita un sunet în caz de linia se completează

```
void verificaRanduri() {
  for (int i = 31; i >= 0; i --) {
    int j;
    for (j = 0; j < 16; j ++) {
      if (!valori[i][j]) {
        break;
      }
    }
    if (j >= 16) {
      tone(BP, 30, 1000);
      scor += 100;
      for (int k = i; k > 0; k --) {
        for (j = 0; j < 16; j++) {
          valori[k][j] = valori[k - 1][j];
        }
      }
      i ++;
    }
  }
}
```

afiseazaScor() se ocupă de display-ul pe 7 segemente pentru afișarea scorului, ia fiecare cifră din scor și in funcție de digits[] afișează scorul pe display

```
void afiseazaScor() {
  static const char digits[] = {
    SEG_A | SEG_B | SEG_C | SEG_D | SEG_E | SEG_F,
    SEG_B | SEG_C,
    SEG_A | SEG_B | SEG_D | SEG_E | SEG_G,
    SEG_A | SEG_B | SEG_C | SEG_D | SEG_G,
    SEG_B | SEG_C | SEG_F,
    SEG_A | SEG_C | SEG_D | SEG_F,
    SEG_A | SEG_C | SEG_D | SEG_E | SEG_F | SEG_G,
    SEG_A | SEG_B | SEG_C,
    SEG_A | SEG_B | SEG_C | SEG_D | SEG_E | SEG_F | SEG_G,
    SEG_A | SEG_B | SEG_C | SEG_D | SEG_F | SEG_G
  };
  char numbers[4] = {digits[0], digits[0], digits[0], digits[0]};
  int cscor = scor;
  int i = 3;
  while (cscor) {
    numbers[i--]  = digits[cscor % 10];
    cscor /= 10;
  }
  display.setSegments(numbers, 4, 0);
}
```

loop() monitorizează dacă pieisa de joc stă pe loc sau a fost așezată în partea inferioara a matricii de joc și apelează funcțiile de mai sus pentru a ține matricea la curent cu ce LED-uri trebuie aprinse

```
void loop(){
  if (millis() - ultimaMutare >= 100) {
    int miscareValida = detecteazaButon();
    if (miscareValida == 0) {
      for (int i = 0; i < 4; i++) {
        Punct pozitie;
        assignPunct(pozitie, addPuncte(piesa.centru, piesa.offset[i]));
        valori[pozitie.y][pozitie.x] = 1;
      }
      assignPiesa(piesa, nextPiesa);
      Piesa new_piesa = piesaNoua();
      assignPiesa(nextPiesa, new_piesa);
    }
    ultimaMutare = millis();
  }
  if (millis() - ultimaCadere >= 500) {
    Piesa copie;
    assignPiesa(copie, piesa);
    copie.centru.y ++;
    if (valideazaMiscare(copie, 1) == 0) {
      for (int i = 0; i < 4; i++) {
        Punct pozitie;
        assignPunct(pozitie, addPuncte(piesa.centru, piesa.offset[i]));
        valori[pozitie.y][pozitie.x] = 1;
      }
      assignPiesa(piesa,nextPiesa);
      Piesa new_piesa = piesaNoua();
      assignPiesa(nextPiesa, new_piesa);
    } else {

      assignPiesa(piesa,copie);
    }
    ultimaCadere = millis();
  }
  verificaRanduri();
  afiseazaScor();
  arataTabla();
  show();
}
```

Resurse:

https://wokwi.com/projects/388240574525847553