ArduinoBoy

A “Modern” Retro Game Console

ДОКУМЕНТАЦИЯ

ИЗГОТВИЛИ:
Божидар Андонов
Петко Люцканов

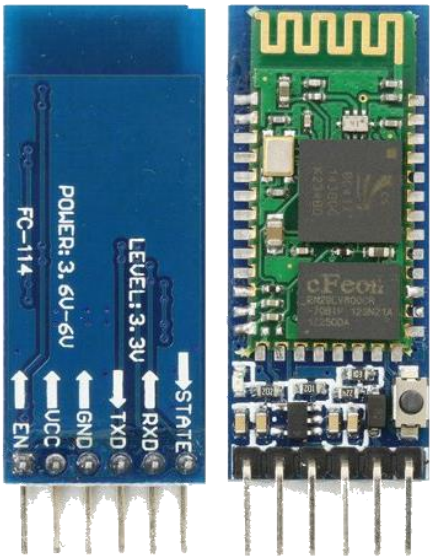
СЪДЪРЖАНИЕ

Списък от компоненти.................................. 3
Описание... 4
Електрическа схема................................... 6
Блок схема... 7
Описание на функционалността и сорс код.............. 8
Заключение.. 18

СПИСЪК ОТ КОМПОНЕНТИ

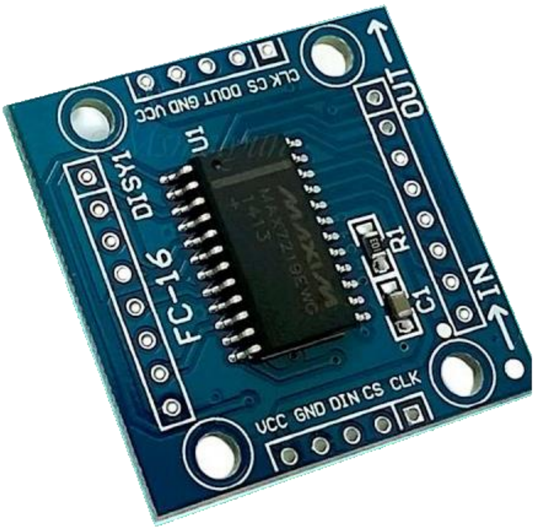
1. Arduino Uno
2. Bluetooth HC-05 модул
3. MAX7219 модули за контролиране на LED матриците(x2)
4. LED матрици 8x8(x2)

Фигура 1- Arduino Uno, микроконтрелора, който извършва всички операции



Фигура 2- Bluetooth HC-05 модул, предаваш информация към и от микроконтролера

Фигура 3- LED матрицата, чрез която визуално се представят игрите

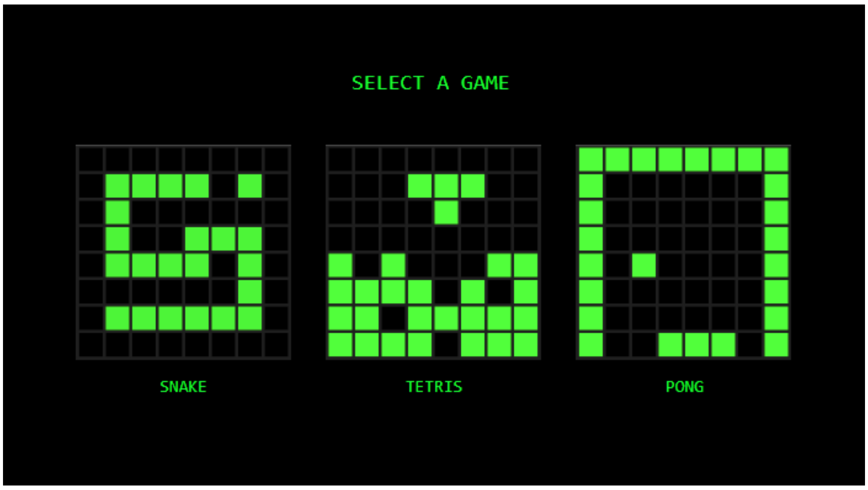


Фигура 4- MAX7219 модул, контролиращ LED матриците4

ОПИСАНИЕ

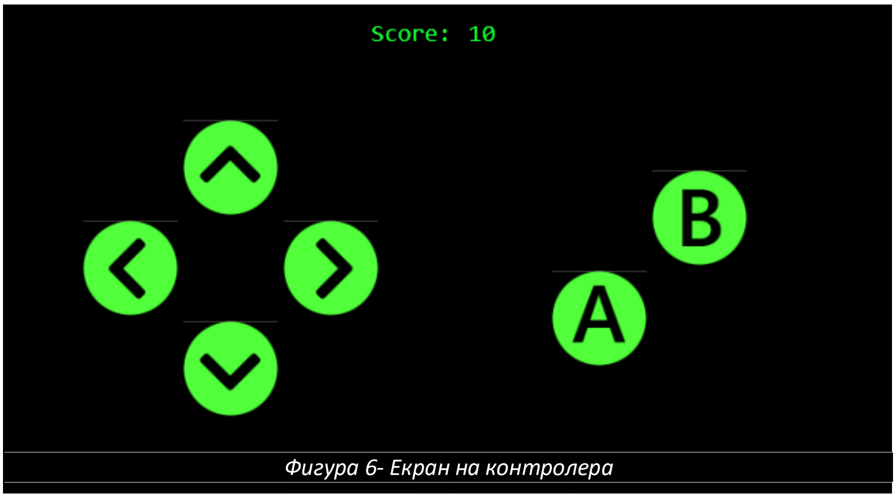
Нашият проект се казва **ArduinoBoy** и представлява мини игрова конзола с някои класически игри, а именно - Змията, Тетрис и Понг. Използваме **Arduino Uno**, за управление на целия проект. Изобразяването на игрите става чрез 2x 8x8 **LED матрици**, контролирани от **MAX7219 модули.**

Специалното на тази конзола е, че няма никакви физически бутони, а комуникацията между потребителя и вградената система се извършва чрез **HC-05 Bluetooth модула**, който трябва да се сдвои и свърже към смартфон с инсталирано специално разработено(с MitAppInventor) приложение, наречено **ArduinoBoyController** за този проект. Потребителят след това има право да избере една от трите игри и да я играе, като тя ще бъде изобразена върху LED матриците.



Фигура 5- Екран за избиране на игра

Контролерът представлява 4 бутона във всяка посока, както и A и B бутони, които имитират оформлението на **GameBoy** конзолата, която всъщност и е вдъхновението за този проект.



Фигура 6- Екран на контролера

Тетрисът представлява класически тетрис, т.е без гравитация и без наместваща ротация – точно както в класиката. Целта на играта е да не се стигне най-горният ред, защото нова фигурка няма да може да бъде поставена. Играчът може да движи и да върти тетроминотата, както по часовниковата стрелка, така и обратно. Колкото повече фигурки бъдат сложени, толкова по-бърза и трудна става играта.

Змията също е класическа ретро игра, която решихме да включим в проекта. Тя представлява змия, която се опитва да събере възможно най-много храна без да се "ухапе"(да се блъсне в себе си). Играчът може да контролира накъде се движи змията с четирите бутона за посока. В тази версия няма стени и змията продължава от другата страна на матрицата ако премине през ръба й. Тук също колкото повече храна събираш, толкова по-бързо се движи самата змия.

Понг играта представлява топче, което се подава от двама играчи, като всеки може да се движи само хоризонтално. Ако даден играч изпусне топчето, другият печели точка. Първият, достигнал 10 точки, побеждава. В ArduinoBoy версията на играта, играе само един играч, а A.I. представлява другия играч. За да са по-интересни(и по-забързани) рундовете, колкото повече пъти топчето бива ударено от даден играч, толкова неговата скорост нараства.

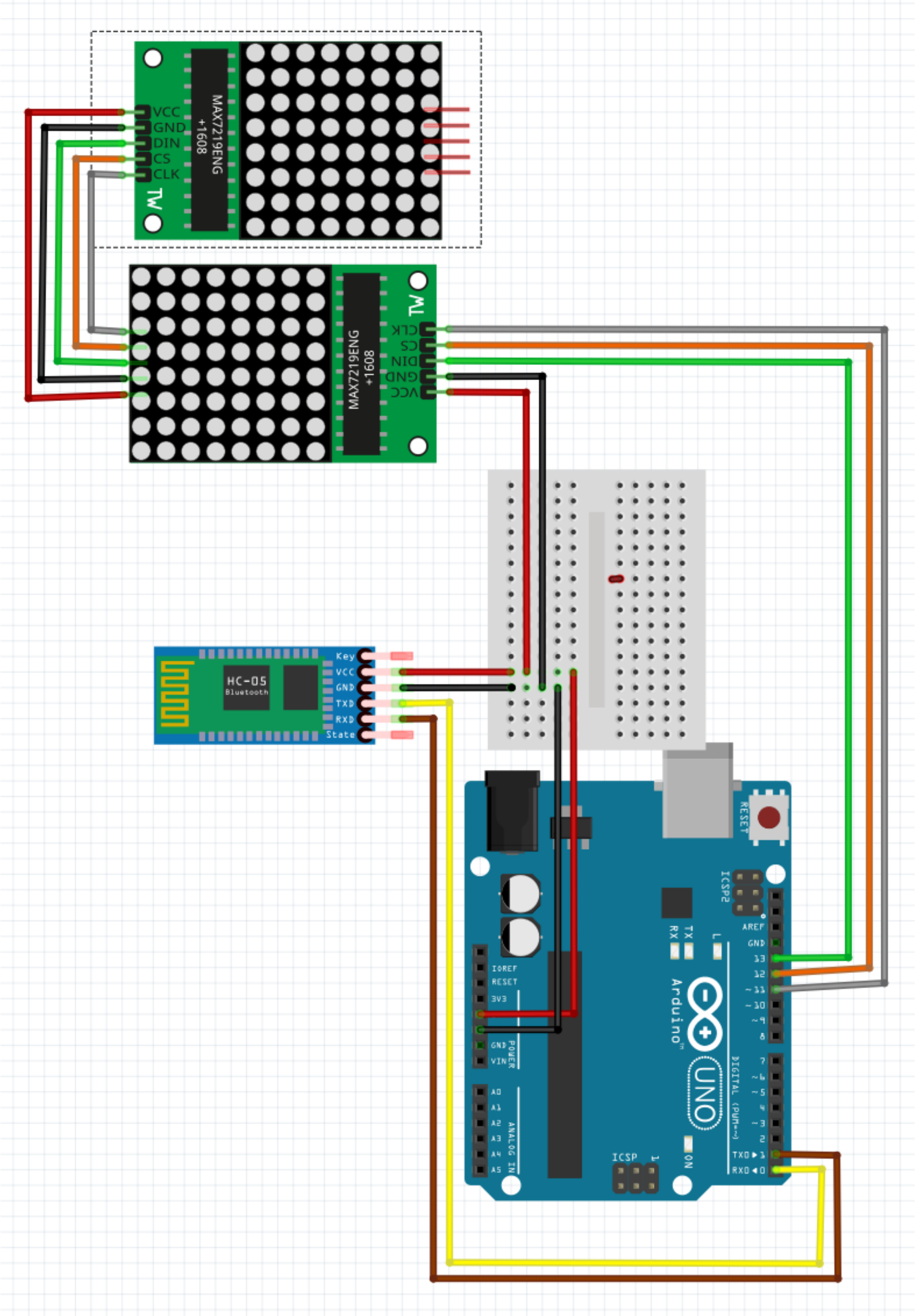
Фигура 7- Игра на тетрис

Фигура 8- Игра на змия

Фигура 9- Игра на понг

ЕЛЕКТРИЧЕСКА СХЕМА

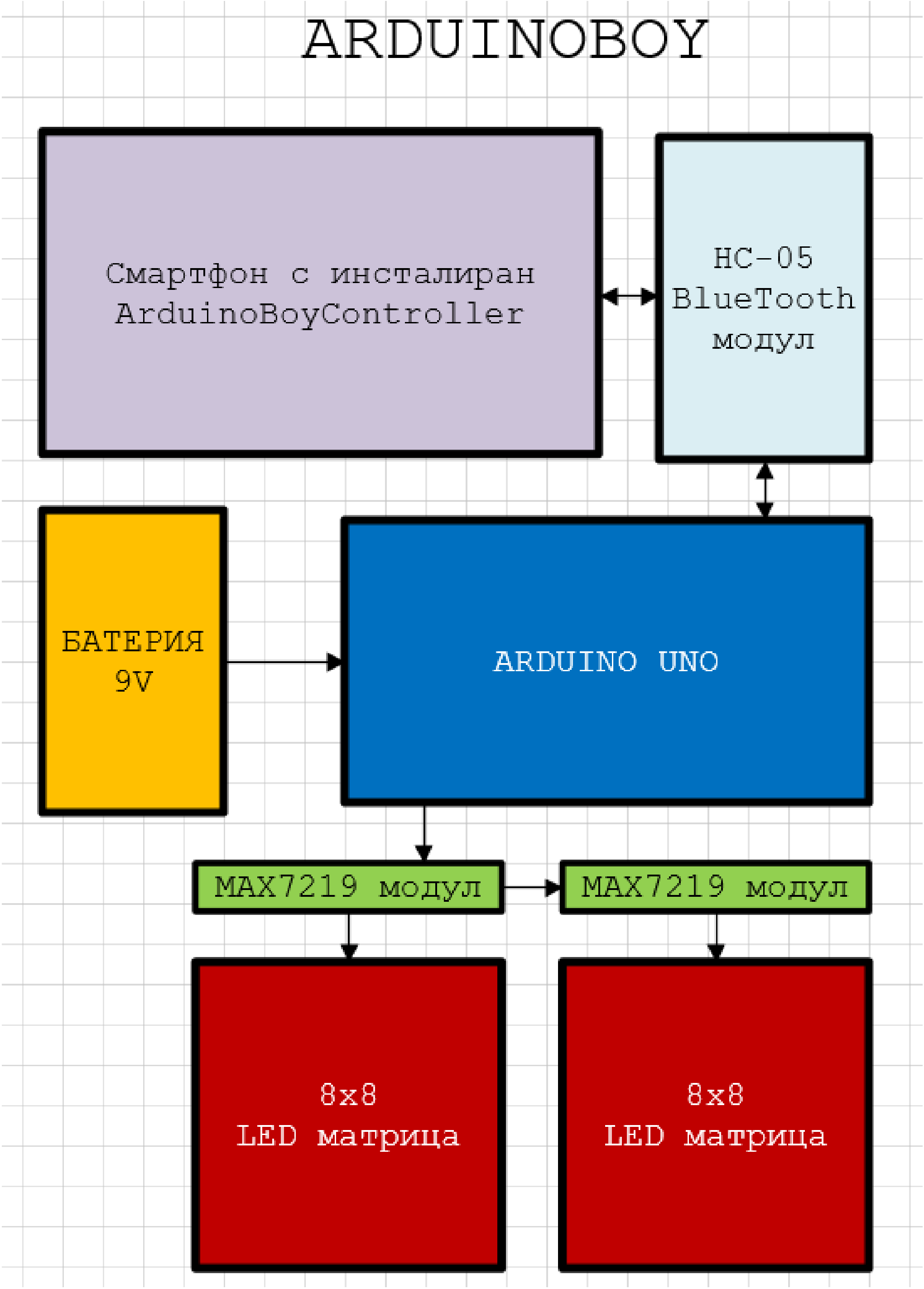
Това е електрическата схема на проекта ни, която се състои от компонентите включени горе. MAX7219 модулите изглеждат различно, но това е само визуално. Всички пинове са свързани аналогично на физическия проект.



Фигура 10 - Електрическа схема

БЛОК СХЕМА

Това е блок схемата на проекта ни. Батерията подава захранване на проекта. Телефонът се свързва към Arduino-то чрез BlueTooth модула и комуникацията йм протича през него. Когато потребителят прати някаква команда, Arduino-то изпраща информация към MAX7219 модулите, които от своя страна включват и изключват определени LED-ове на матриците.



Фигура 11- Блок схема

ОПИСАНИЕ НА ФУНКЦИОНАЛНОСТТА И СОРС КОД

Това е целият код използван за програмиране на Arduino Uno-то. Чрез коментарите и допълнително вмъкнатия текст ще разберете функционалността. Първо инициализираме основните променливи, които ще използваме по време на изпълнение на приложението. Определяме и sprite-ове, които предефинират какво да се покаже на дисплея когато бъдат извикани.

```
#include <ArduinoSTL.h>
#include <LedControl.h>
#include <vector>

#define DINPin 11  // DataIn pin for the MAX7219
module
#define CSPin 12   // Load pin for the MAX7219 module
#define CLKPin 13  // Clock pin for the MAX7219
module

// The following are all used structs within the
project
// Location is used to map certain game elements to
the LED matrix
struct Location
{
  byte x; // [0-15] the row index of the element
  byte y; // [0-7] the column index of the element
};

// This struct is specifically made for the tetris
game and is used to
// track where the tetromino is.
struct Tetromino
{
  // All of the blocks represent one location
element, so that the tetromino
  // can be displayed properly
  Location block1;
  Location block2;
  Location block3;
  Location block4;
  Location center; // The center is used to determine
the rotation point of the tetromino
};

// This struct is used to locate both the A.I. paddle
and the player controlled one in the Pong game
struct Paddle
{
  // It consists of two Location objects as well,
they determine where the paddle is located
  Location block1;
  Location block2;
};

// Direction is a struct, used in the pong game and
its purpose is to determine what
// direction the ball will move in.
struct Direction
{
  short x; // [-1...1] the row direction, -1 is
upwards, 1 is downwards
  short y; // [-1...1] the column direction, -1 is
left, 1 is right
};

// All of the following are sprites, which are used
to display generic information
// on the LED displays when an event is fired. All of
them are in binary, represents
// whether the light is on or off.

// The sprite used in the game over screen.
byte gameOverSprite[16] =
{
  B11110110,
  B10001001,
  B10111111,
  B01100100,
  …
  …
  …
```

Има много от тези спрайтове, затова ще ги пропуснем

```
  B00000000,
  B00000000
};

// The following variables are used by all of the
games
LedControl matrixController(DINPin, CLKPin, CSPin,
2); // The controller for the LED matrix.
unsigned long timer; // timer, which uses the
millis() function to determine when certain events
should be fired.
bool isGameOver; // Variable, which determines
whether the current game is over.
int playerScore; // Variable, used to store the
player's current score.

// The following variables are used by the Tetris
game
Tetromino tetromino; // The tetromino that the user
currently has control of
bool tetrisMatrix[16][8]; // The matrix, used to
determine which LEDs turn on
short fallingTetrominoDelay; // The delay used to
determine how long it will be until the tetromino
goes down one step.

// The following variables are used by the Snake game
char prevDirection; // The previous direction the
snake followed.
char direction; // The current direction the snake is
following
Location food; // The row and column coordinates of
the food.
bool foodState; // Determines what state the blinking
food is in (true -> LED on, false -> LED off)
short const scorePerFood = 10; // A constant used to
determine how many points each eaten food gives.
```

След декларирането на променливите, декларираме основните методи, с които работи ардуиното – setup() и loop(), в които нулираме основни променливи и чакаме потребителя да избере игра. Метод playSnake() е основният метод, контролиращ играта “Snake”, а setupSnake() е този, който я подготвя преди всеки неин пуск.

```
short snakeMovementDelay; // The delay used to
determine how long it will be until the snake follows
the current direction again
std::vector<Location> snake; // A vector, containing
all of the coordinates of the snake.

// Pong following variables are used by the Pong game

Paddle playerPaddle; // The paddle controlled by the
player
Paddle aiPaddle; // The paddle controlled by the A.I.
Location ball; // The current location of the ball.
Direction ballDirection; // The direction in which
the ball is going in
byte const pointsToWin = 10; // A constant, used to
determine how many points each side needs to win the
Pong game
short const aiMovementDelay = 220; // The constraint,
which makes it impossible for the A.I. to win every
game.
short ballDelay; // The delay used to determine how
long it will be until the ball changes its position
unsigned long aiTimer; // An additional timer, whose
purpose is to determine whether the A.I. can move its
paddle yet.
int numberOfHits; // Number of times the ball was hit
by either of the players. Each time it is hit, the
velocity of the ball increases.
int aiScore; // An additional score counter, used for
the A.I.

void setup()
{
  // Open the serial port for communication
  Serial.begin(9600);

  // Wake the LED boards up
  matrixController.shutdown(0, false);
  matrixController.shutdown(1, false);

  // Set the intensity of the display
  matrixController.setIntensity(0, 0);
  matrixController.setIntensity(1, 0);

  // Clear the display
  matrixController.clearDisplay(0);
  matrixController.clearDisplay(1);

  // Giving a random seed, so that the games spawn
random items in random locations. Uses
  // the noise the A0 port
  randomSeed(analogRead(A0));
}

void loop()
{
  // Wait for a signal from the smartphone app. Once
a game is selected
  // on the phone, the function responsible for the
game is called.
  if (Serial.available() >= 0)
  {
    char requestedGame = Serial.read();
    if (requestedGame == 'T')
    {
      playTetris();
    }
    else if (requestedGame == 'S')
    {
      playSnake();
    }
    else if (requestedGame == 'P')
    {
      playPong();
    }
  }
```

```
}

// The following methods are used to play the SNAKE
game.

// This method is the main one, responsible for
running the SNAKE game.
void playSnake()
{
  setupSnake();

  // Start the timer
  timer = millis();
  while (!isGameOver)
  {
    // When information is sent through the mobile
application, it is interpreted here.
    if (Serial.available() >= 0)
    {
      char input = Serial.read();
      // 'L' - Left; 'R' - Right; 'D' - Down; 'U' -
Up
      if (input == 'L' || input == 'R' || input ==
'D' || input == 'U')
      {
        char prevDirection = direction;
        char newDirection =  input;
        direction =
changeSnakeDirection(prevDirection, newDirection);
      }
    }
    // When a certain time passes, move the snake in
the selected direction
    if (millis() - timer >= snakeMovementDelay)
    {
      timer = millis();
      moveSnake();
    }
  }
  displayGameOverScreen();
}

// This method is used to setup the Snake game every
time before it is run.
void setupSnake()
{
  //Setting variables to their default starting
values
  isGameOver = false;
  playerScore = 0;
  foodState = true;
  snakeMovementDelay = 800;

  // If anything from the snake in the last game
remained, it's cleared.
  while (!snake.empty())
  {
    snake.pop_back();
  }

  // Set the starting location for the snake
  Location startingBlock;
  startingBlock.x = 7;
  startingBlock.y = 4;
  snake.push_back(startingBlock);
  startingBlock.x = 7;
  startingBlock.y = 3;
  snake.push_back(startingBlock);
  startingBlock.x = 7;
  startingBlock.y = 2;
  snake.push_back(startingBlock);
// Get a new food location and then display the game
on the LED matrices
  newFood();
  printSnakeGameBoard();
}
```

Тук можем да видим методите за смяна на посоката и как се генерира нова храна. Също така декларираме методи за показване на състоянието на игралното поле и змията. Започнат е и методът, който придвижва змията в 2D пространството.

```
// This method is used to change the direction of the
snake
char changeSnakeDirection(char prevDirection, char
newDirection)
{
  // If the previous direction is exactly the
opposite to the requested one,
  // ex. Left and Right, then don't change it (The
snake can't turn 180 degrees)
  switch (newDirection)
  {
    case 'R':
      if (prevDirection == 'L')
      {
        return 'L';
      }
      else return 'R';
      break;
    case 'L':
      if (prevDirection == 'R')
      {
        return 'R';
      }
      else return 'L';
      break;
    case 'D':
      if (prevDirection == 'U')
      {
        return 'U';
      }
      else return 'D';
      break;
    case 'U':
      if (prevDirection == 'D')
      {
        return 'D';
      }
      else return 'U';
      break;
  }
}

// This method is used to generate a new food when
the previous one has been collected by the snake.
void newFood()
{
  bool needNewLocation = true; // Used to determine
whether the generated location overlaps with the
snake.
  byte x, y; // The coordinates for the new food.
  while (needNewLocation)
  {
    x = random(0, 15);
    y = random(0, 7);

    // Checking whether the generated location
overlaps with the snake
    for (std::vector<Location>::iterator i =
snake.begin(); i != snake.end(); i++)
    {
      if (i -> x == x && i -> y == y)
      {
        needNewLocation = true;
        break;
      }
      else needNewLocation = false;
    }

    if (!needNewLocation)
    {
      food.x = x;
      food.y = y;
    }
  }
  // Display the new food alongside the snake
  printSnakeGameBoard();
}
```

```
// This method is used to display the Snake game
elements
void printSnakeGameBoard()
{
  // Clear the displays
  matrixController.clearDisplay(0);
  matrixController.clearDisplay(1);
  // Determine what state of the LED of the blinking
food must be in - On or off
  foodState = !foodState;
  matrixController.setLed(food.x / 8, food.x % 8,
food.y, foodState);
  // Light up the snake
  for (std::vector<Location>::iterator i =
snake.begin(); i != snake.end(); i++)
  {
    matrixController.setLed(i -> x / 8, i -> x % 8, i
-> y, true);
  }
}

// This method is used to turn off all LEDs
associated with the snake itself
void turnOffOldSnakeLocation()
{
  for (std::vector<Location>::iterator i =
snake.begin() + 1; i != snake.end(); i++)
  {
    matrixController.setLed(i -> x / 8, i -> x % 8, i
-> y, false);
  }
}

// This method is responsible for making the snake
move
void moveSnake()
{
  turnOffOldSnakeLocation();

  // Get the back and the front of the snake
  Location backLocation = snake.back();
  snake.pop_back();
  Location frontLocation = snake.front();

  // Get the new location of the front of the snake,
dependent on the
  // direction it moves in. If it goes outside the
matrix, it will return
  // from the other side.
  Location newFrontLocation;
  switch (direction)
  {
    case 'R':
    {
      newFrontLocation.x = frontLocation.x;
      newFrontLocation.y = (frontLocation.y + 1) % 8;
      break;
    }
    case 'L':
    {
      newFrontLocation.x = frontLocation.x;
      if (frontLocation.y == 0)
      {
        newFrontLocation.y = 7;
      }
      else newFrontLocation.y = frontLocation.y - 1;
      break;
    }
    case 'D':
    {
      newFrontLocation.x = (frontLocation.x + 1) %
16;
      newFrontLocation.y = frontLocation.y;
      break;
    }
    case 'U':
    {
      if (frontLocation.x == 0)
      {
```

Трябва да обърнем внимание на setNewSnakeSpeed() метода. Той се състои в това, че колкото по-голяма става змията, толкова по-бързо започва да се придвижва. Виждаме и основните методи за играненето на тетрис (playTetris() и setupTetris()), които са аналогични на тези на змията.

```
        newFrontLocation.x = 15;
      }
      else newFrontLocation.x = frontLocation.x - 1;
      newFrontLocation.y = frontLocation.y;
      break;
    }
  }

  // Insert the new front location into the snake
vector and collect food if the
  // snake overlaps with it
  snake.insert(snake.begin(), newFrontLocation);
  collectFood(backLocation);

  printSnakeGameBoard();
  checkIfSnakeGameOver();
}

// This method is responsible for enlarging the snake
when food is collected
// and adds points to the score counter for doing so.
void collectFood(Location backLocation)
{
  if (snake[0].x == food.x && snake[0].y == food.y)
  {
    snake.push_back(backLocation);
    addSnakeScore();
    newFood();
  }
}

// This method is used to add points to the score
counter and send that
// information to the mobile phone, which on its turn
displays it.
void addSnakeScore()
{
  playerScore += scorePerFood;
  String stringToPrint = "C" + String(playerScore);
  Serial.print(stringToPrint);
  setNewSnakeSpeed();
}

// This method manages how fast the snake is moving.
The more one progresses,
// the faster the snake becomes.
void setNewSnakeSpeed()
{
  if (snakeMovementDelay > 300)
  {
    snakeMovementDelay -= 10;
  }
}

// This method is responsible for determining whether
it is game over.
void checkIfSnakeGameOver()
{
  // Determine whether the front of the snake
collides with a part of its body.
  // If that's the case, then it is game over and the
game stops.
  Location front = snake.front();
  byte count = 0;
  for (std::vector<Location>::iterator i =
snake.begin(); i != snake.end(); i++)
  {
    if (front.x == i -> x && front.y == i -> y)
    {
      count++;
    }
  }
  if (count > 1)
  {
    isGameOver = true;
  }
}
```

```
// The following methods are used for playing the
TETRIS game

// This is the main method, used to play the tetris
game.
void playTetris()
{
  setupTetris();

  // Start the timer
  timer = millis();
  while (!isGameOver)
  {
    // This is triggered when it is time for the
tetromino to go down one step
    if (millis() - timer >= fallingTetrominoDelay)
    {
      timer = millis();
      shiftDown();
      String stringToPrint = "C" +
String(playerScore);
      Serial.print(stringToPrint);
    }
    // Managing user input.
    if (Serial.available() > 0)
    {
      char command = Serial.read();
      if (command == 'L')
      {
        shiftLeft();
      }
      else if (command == 'R')
      {
        shiftRight();
      }
      else if (command == 'A')
      {
        rotate(-1, 1);
      }
      else if (command == 'B')
      {
        rotate(1, -1);
      }
      else if (command == 'D')
      {
        fastForward();
      }
    }
  }
  displayGameOverScreen();
}

// This method is used to setup the tetris game
void setupTetris()
{
  // Assign default values to the variables.
  playerScore = 0;
  fallingTetrominoDelay = 1000;
  isGameOver = false;

  // Clear the tetris matrix if something remains
from the last game and
  // get the first tetromino
  for (int i = 0; i < 16; i++)
  {
    for (int j = 0; j < 8; j++)
    {
      tetrisMatrix[i][j] = false;
    }
  }
  getNewTetromino();
}

// This method is responsible for fast-forwarding the
placement
```

Тук са методите за по-бързото падане на тетроминото, както и алгоритъма за неговото обръщане в зависимост от това, дали потребителя иска да го върти по часовниковата стрелка или срещу нея

```
// of the tetromino when the Down button has been
pressed
void fastForward()
{
  while (tetrisMatrix[tetromino.block1.x +
1][tetromino.block1.y] != 1 &&
         tetrisMatrix[tetromino.block2.x +
1][tetromino.block2.y] != 1 &&
         tetrisMatrix[tetromino.block3.x +
1][tetromino.block3.y] != 1 &&
         tetrisMatrix[tetromino.block4.x +
1][tetromino.block4.y] != 1 &&
         tetromino.block1.x != 15 &&
tetromino.block2.x != 15 &&
         tetromino.block3.x != 15 &&
tetromino.block4.x != 15)
  {
    shiftDown();
    delay(fallingTetrominoDelay / 7);
  }
}

// This method is used to rotate the tetromino in the
best way possible
// in a clockwise or anti-clockwise direction.
void rotate(int rotationXIndex, int rotationYIndex)
{
  // The center is zero only when it is the 'O'
tetromino
  if (tetromino.center.x == 0 && tetromino.center.y
== 0)
  {
    return;
  }
  turnOffOldTetrominoLocation();

  // The numbers on those variables correspond to the
Tetromino struct ones.
  // This algorithm is used to determine where the
new blocks will be located
  // after the rotation.
  Location tempPosition1, tempPosition2,
tempPosition3, tempPosition4;
  Location relativePosition1, relativePosition2,
relativePosition3, relativePosition4;
  Location newLocation1, newLocation2, newLocation3,
newLocation4;

  tempPosition1.x = tetromino.block1.x -
tetromino.center.x;
  tempPosition1.y = tetromino.block1.y -
tetromino.center.y;
  relativePosition1.x = rotationXIndex *
tempPosition1.y;
  relativePosition1.y = rotationYIndex *
tempPosition1.x;
  newLocation1.x = tetromino.center.x +
relativePosition1.x;
  newLocation1.y = tetromino.center.y +
relativePosition1.y;
  …
  …
  …
```

Тук, всичко също е аналогично и го пропускаме

```
  newLocation4.x = tetromino.center.x +
relativePosition4.x;
  newLocation4.y = tetromino.center.y +
relativePosition4.y;

  // When the new location is overlapping with an
already placed
  // tetromino or is outside the bounds of the
screen, the rotation
  // is not completed.
  if (newLocation1.x < 0 || newLocation1.x > 15 ||
      newLocation1.y < 0 || newLocation1.y > 7 ||
      newLocation2.x < 0 || newLocation2.x > 15 ||
      newLocation2.y < 0 || newLocation2.y > 7 ||
      newLocation3.x < 0 || newLocation3.x > 15 ||
      newLocation3.y < 0 || newLocation3.y > 7 ||
      newLocation4.x < 0 || newLocation4.x > 15 ||
      newLocation4.y < 0 || newLocation4.y > 7)
  {
    turnOnNewTetrominoLocation();
    return;
  }
  else if
(tetrisMatrix[newLocation1.x][newLocation1.y] ==
false &&

tetrisMatrix[newLocation2.x][newLocation2.y] == false
&&

tetrisMatrix[newLocation3.x][newLocation3.y] == false
&&

tetrisMatrix[newLocation4.x][newLocation4.y] ==
false)
  {
    tetromino.block1 = newLocation1;
    tetromino.block2 = newLocation2;
    tetromino.block3 = newLocation3;
    tetromino.block4 = newLocation4;
  }

  turnOnNewTetrominoLocation();
}
```

Метод за придвижване надолу.

```
// This method is used to bring the tetromino one
step down
void shiftDown()
{
  turnOffOldTetrominoLocation();

  // Check whether the tetromino is already at the
bottom or whether it
  // is touching the matrix
  if (tetromino.block1.x == 15 || tetromino.block2.x
== 15 || tetromino.block3.x == 15 ||
tetromino.block4.x == 15)
  {

tetrisMatrix[tetromino.block1.x][tetromino.block1.y]
= 1;

tetrisMatrix[tetromino.block2.x][tetromino.block2.y]
= 1;

tetrisMatrix[tetromino.block3.x][tetromino.block3.y]
= 1;

tetrisMatrix[tetromino.block4.x][tetromino.block4.y]
= 1;
    addTetrisScore(5);
    getNewTetromino();
    return;
  }
  else if (tetrisMatrix[tetromino.block1.x +
1][tetromino.block1.y] == 1 ||
           tetrisMatrix[tetromino.block2.x +
1][tetromino.block2.y] == 1 ||
           tetrisMatrix[tetromino.block3.x +
1][tetromino.block3.y] == 1 ||
```

Тук са декларирани методите за преместване на тетроминото наляво и надясно, както и за генериране на ново такова, когато е докоснало някоя друга фигура. Можем да видим и проверка за край на играта.

```
          tetrisMatrix [tetromino.block4.x +
1][tetromino.block4.y] == 1)
  {

tetrisMatrix[tetromino.block1.x][tetromino.block1.y]
= 1;

tetrisMatrix[tetromino.block2.x][tetromino.block2.y]
= 1;

tetrisMatrix[tetromino.block3.x][tetromino.block3.y]
= 1;

tetrisMatrix[tetromino.block4.x][tetromino.block4.y]
= 1;
    addTetrisScore(5);
    getNewTetromino();
    return;
  }

  // Offset the tetromino by one to the bottom.
  tetromino.block1.x++;
  tetromino.block2.x++;
  tetromino.block3.x++;
  tetromino.block4.x++;
  tetromino.center.x++;
  turnOnNewTetrominoLocation();
}

// This method is used to shift the tetromino left
when the user requests it
void shiftLeft()
{
  // Check whether it is already at the leftmost
position or the space is already occupied
  if (tetromino.block1.y == 0 || tetromino.block2.y
== 0 || tetromino.block3.y == 0 || tetromino.block4.y
== 0)
  {
    return;
  }
  else if
(tetrisMatrix[tetromino.block1.x][tetromino.block1.y
- 1] == 1 ||

tetrisMatrix[tetromino.block2.x][tetromino.block2.y -
1] == 1 ||

tetrisMatrix[tetromino.block3.x][tetromino.block3.y -
1] == 1 ||

tetrisMatrix[tetromino.block4.x][tetromino.block4.y -
1] == 1)
  {
    return;
  }

  // Offset the tetromino by one to the left.
  turnOffOldTetrominoLocation();
  tetromino.block1.y--;
  tetromino.block2.y--;
  tetromino.block3.y--;
  tetromino.block4.y--;
  tetromino.center.y--;
  turnOnNewTetrominoLocation();
}

// This method is used to shift the tetromino right
when the user requests it
void shiftRight()
{
  // Check whether it is already at the rightmost
position or the space is already occupied
  if (tetromino.block1.y == 7 || tetromino.block2.y
== 7 || tetromino.block3.y == 7 || tetromino.block4.y
== 7)
  {
    return;
  }
  else if
(tetrisMatrix[tetromino.block1.x][tetromino.block1.y
+ 1] == 1 ||

tetrisMatrix[tetromino.block2.x][tetromino.block2.y +
1] == 1 ||

tetrisMatrix[tetromino.block3.x][tetromino.block3.y +
1] == 1 ||

tetrisMatrix[tetromino.block4.x][tetromino.block4.y +
1] == 1)
  {
    return;
  }

  // Offset the tetromino by one to the right.
  turnOffOldTetrominoLocation();
  tetromino.block1.y++;
  tetromino.block2.y++;
  tetromino.block3.y++;
  tetromino.block4.y++;
  tetromino.center.y++;
  turnOnNewTetrominoLocation();
}

// This method is used to generate a new tetromino
when the old one has been placed.
void getNewTetromino()
{
  clearFullRows();
  int tetrominoIndex = random(0, 7);
  switch (tetrominoIndex)
  {
    // Setting all of the locations manually,
depending on the chosen tetromino.
    case 0:
      tetromino.block1.x = 0;  //  - - o o o o - -
      tetromino.block1.y = 2;
      tetromino.block2.x = 0;
      tetromino.block2.y = 3;
      tetromino.block3.x = 0;
      tetromino.block3.y = 4;
      tetromino.block4.x = 0;
      tetromino.block4.y = 5;
      tetromino.center.x = 0;
      tetromino.center.y = 3;
      break;
    case 1:
      tetromino.block1.x = 0;  //  - - o - - - - -
      tetromino.block1.y = 2;  //  - - o o o - - -
      tetromino.block2.x = 1;
      …
      …
      …
```

Всички генерирания са сходни така че ще ги пропуснем.

```
      tetromino.block3.y = 3;
      tetromino.block4.x = 1;
      tetromino.block4.y = 4;
      tetromino.center.x = 1;
      tetromino.center.y = 3;
      break;
  }
  turnOnNewTetrominoLocation();
  checkIfTetrisGameOver();
}

// This method determines whether it is game over in
the game of Tetris
void checkIfTetrisGameOver()
{
  // If a tetromino overlaps with a location from the
matrix, then there is no space
```

Методите тук отговарят за изчистване на редовете когато се напълнят, както и за кои части от LED матрицата трябва да светнат. Тук отново виждаме, че колкото повече точки се трупат, толкова по-бързо падат тетроминотата.

```
  // and that means the game is over.
  if
(tetrisMatrix[tetromino.block1.x][tetromino.block1.y]
== true ||

tetrisMatrix[tetromino.block2.x][tetromino.block2.y]
== true ||

tetrisMatrix[tetromino.block3.x][tetromino.block3.y]
== true ||

tetrisMatrix[tetromino.block4.x][tetromino.block4.y]
== true)
  {
    matrixController.clearDisplay(0);
    matrixController.clearDisplay(1);
    isGameOver = true;
  }
}

// This method is responsible for clearing any full
rows that have
// formed due to placing a tetromino in a certain
location.
void clearFullRows()
{
  for (int i = 0; i < 16; i++)
  {
    // Find out whether the current row (i) is full.
    bool isRowFull = true;
    for (int j = 0; j < 8; j++)
    {
      if (tetrisMatrix[i][j] == false)
      {
        isRowFull = false;
        break;
      }
    }

    // If it is, clear it and move everything above
it one step lower.
    if (isRowFull)
    {
      for (int j = 0; j < 8 ; j++)
      {
        tetrisMatrix[i][j] = false;
      }
      lightUpTetrisMatrix();
      delay(200);

      for (int k = i; k > 0 ; k--)
      {
        for (int l = 0; l < 8; l++)
        {
          tetrisMatrix[k][l] = tetrisMatrix[k -
1][l];
          tetrisMatrix[k - 1][l] = false;

        }
      }
      lightUpTetrisMatrix();
      // Add score for clearing the line.
      addTetrisScore(15);
      i--;
    }
  }
}

// This method adds points to the score counter
void addTetrisScore(short scoreToAdd)
{
  playerScore += scoreToAdd;
  setTetrisSpeed();
}

// This method sets the speed at which the tetromino
falls. The longer
// the game goes on, the faster this speed is and the
harder the game becomes.
```

```
void setTetrisSpeed()
{
  if (fallingTetrominoDelay > 300)
  {
    fallingTetrominoDelay = 1000 - playerScore;
  }
}

// This method turns off the old tetromino location.
void turnOffOldTetrominoLocation()
{
  // Refresh the whole matrix first.
  lightUpTetrisMatrix();
  Location block = tetromino.block1;
  matrixController.setLed(block.x / 8, block.x % 8,
block.y, false);
  block = tetromino.block2;
  matrixController.setLed(block.x / 8, block.x % 8,
block.y, false);
  block = tetromino.block3;
  matrixController.setLed(block.x / 8, block.x % 8,
block.y, false);
  block = tetromino.block4;
  matrixController.setLed(block.x / 8, block.x % 8,
block.y, false);
}

// This method lights up the tetromino on the LED
matrices.
void turnOnNewTetrominoLocation()
{
  // Refresh the whole matrix first.
  lightUpTetrisMatrix();
  Location block = tetromino.block1;
  matrixController.setLed(block.x / 8, block.x % 8,
block.y, true);
  block = tetromino.block2;
  matrixController.setLed(block.x / 8, block.x % 8,
block.y, true);
  block = tetromino.block3;
  matrixController.setLed(block.x / 8, block.x % 8,
block.y, true);
  block = tetromino.block4;
  matrixController.setLed(block.x / 8, block.x % 8,
block.y, true);
}

// This method is used to light up the LEDs according
to the tetrisMatrix array.
void lightUpTetrisMatrix()
{
  for (int i = 0; i < 16; i++)
  {
    for (int j = 0; j < 8; j++)
    {
      matrixController.setLed(i / 8, i % 8, j,
tetrisMatrix[i][j]);
    }
  }
}

// This method is responsible for displaying the game
over screen on both the
// snake and tetris games.
void displayGameOverScreen()
{
  for (int i = 0; i < 16; i++)
  {
    matrixController.setRow(i / 8, i % 8,
gameOverSprite[i]);
  }
  while (Serial.available() == 0) {}
  Serial.write('V');
  matrixController.clearDisplay(0);
  matrixController.clearDisplay(1);
}

// The following methods are used for playing the
PONG game
```

Всичко оттук нататък е за играта Понг. Отново виждаме двата основни метода за стартиране на играта, както и част от метода за движение на топчето по игралното поле.

```
// This method is the main one, used to run the Pong
game properly.
void playPong()
{
  // Setup the game and start the timers.
  setupPong();
  timer = millis();
  aiTimer = millis();

  while (!isGameOver)
  {
    // When the ball needs to move, the method
responsible for that is called.
    if (millis() - timer > ballDelay)
    {
      moveBall();
      timer = millis();
    }
    // If the A.I. is allowed to make a move, it does
so
    if (millis() - aiTimer > aiMovementDelay)
    {
      movePongAi();
      aiTimer = millis();
    }
    // Interpret information sent by the player's
smartphone;
    if (Serial.available() > 0)
    {
      char dir = Serial.read();
      if (dir == 'R' || dir == 'L')
      {
        movePongPlayer(dir);
      }
    }
  }
  displayPongWinner();
}

// This method is used to setup the Pong game
void setupPong()
{
  // Set variables to their default values.
  String scoreToSend = "C0:0";
  Serial.print(scoreToSend);
  isGameOver = false;
  playerScore = 0;
  aiScore = 0;

  // Call a method to begin a new round.
  newPongRound();
}

// This method is responsible for creating new rounds
after a point has
// been scored.
void newPongRound()
{
  // Set variables to their default values
  ballDelay = 250;
  numberOfHits = 0;
  ball.x = random(1, 8);
  ball.y = random(1, 7);
  ballDirection.y = random(-1, 2);
  ballDirection.x = 1;
  playerPaddle.block1.x = 15;
  playerPaddle.block1.y = 3;
  playerPaddle.block2.x = 15;
  playerPaddle.block2.y = 4;
  aiPaddle.block1.x = 0;
  aiPaddle.block1.y = 3;
  aiPaddle.block2.x = 0;
  aiPaddle.block2.y = 4;

  // Light up the proper LEDs on the display.
  matrixController.clearDisplay(0);
  matrixController.clearDisplay(1);
  matrixController.setLed(0, 0, aiPaddle.block1.y,
true);
  matrixController.setLed(0, 0, aiPaddle.block2.y,
true);
  matrixController.setLed(1, 7,
playerPaddle.block1.y, true);
  matrixController.setLed(1, 7,
playerPaddle.block2.y, true);
  matrixController.setLed(ball.x / 8, ball.x % 8,
ball.y, true);

  // Wait one second so the player won't be caught
off-guard.
  delay(1000);
}

// This method is responsible for moving the ball in
a proper direction
void moveBall()
{
  // If Y is already set, we won't be able to change
it at a later point.
  bool isYSet = false;

  // Emulate bouncing off the walls.
  if (ball.y == 0)
  {
    if (ballDirection.y == -1)
    {
      ballDirection.y = 1;
      isYSet = true;
    }
  }
  else if (ball.y == 7)
  {
    if (ballDirection.y == 1)
    {
      ballDirection.y = -1;
      isYSet = true;
    }
  }

  // Emulate hitting the paddle or falling below it.
  if (ball.x == 1 && ballDirection.x == -1)
  {
    // When the ball is about to hit the paddle, do
this:
    if (aiPaddle.block1.y == ball.y + ballDirection.y
|| aiPaddle.block2.y == ball.y + ballDirection.y)
    {
      ballDirection.x = 1;
      numberOfHits++;
      if (!isYSet)
      {
        ballDirection.y = random(-1, 2);
      }
      setBallSpeed();
    }
    // else the other player gets a point.
    else
    {
      playerScore++;
      checkIfPongGameOver();
      return;
    }
  }
  else if (ball.x == 14 && ballDirection.x == 1)
  {
    if (playerPaddle.block1.y == ball.y +
ballDirection.y || playerPaddle.block2.y == ball.y +
ballDirection.y)
    {
      ballDirection.x = -1;
      numberOfHits++;
      if (!isYSet)
      {
        ballDirection.y = random(-1, 2);
      }
      setBallSpeed();
```

Тук, трябва да обърнем внимание на алгоритъма, с който A.I. се придвижва, както и на този, с който играчът се мести.

```
}
    else
    {
      aiScore++;
      checkIfPongGameOver();
      return;
    }
  }

  // Turning the old ball location off, setting the
new location, based on the
  // direction it has to go and lighting up the new
location.
  matrixController.setLed(ball.x / 8, ball.x % 8,
ball.y, false);
  ball.x += ballDirection.x;
  ball.y += ballDirection.y;
  matrixController.setLed(ball.x / 8, ball.x % 8,
ball.y, true);
}

// This method is used to determine whether it is
game over, if it is
// not yet, it displayes the current scores.
void checkIfPongGameOver()
{
  if (playerScore == pointsToWin || aiScore ==
pointsToWin)
  {
    isGameOver = true;
    return;
  }
  else
  {
    displayScore();
  }
}

// This method is used to set the ball velocity. The
more times the ball is hit,
// the faster it becomes.
void setBallSpeed()
{
  if (ballDelay >= 100)
  {
    ballDelay = 250 - numberOfHits * 10;
  }
}

// This method determines what direction the A.I.
will move in and whether
// it will move at all.
void movePongAi()
{
  // The A.I. can only move if the ball is heading
towards it.
  if (ballDirection.x == -1)
  {
    // Specific scenarios for the A.I.
    if (ball.y == 0 && ball.x == 1 &&
aiPaddle.block1.y == 0)
    {
      return;
    }
    if (ball.y == 7 && ball.x == 1 &&
aiPaddle.block2.y == 7)
    {
      return;
    }

    // Turning off the old location of the A.I.
paddle.
    matrixController.setLed(0, 0, aiPaddle.block1.y,
false);
    matrixController.setLed(0, 0, aiPaddle.block2.y,
false);

    // Determine what direction the paddle should go
in, based on several factors.
    if (ballDirection.y == 0)
    {
      if (ball.y < aiPaddle.block1.y)
      {
        aiPaddle.block1.y--;
        aiPaddle.block2.y++;
      }
    }
    else if (ballDirection.y == 1)
    {
      if (ball.y < aiPaddle.block1.y)
      {
        aiPaddle.block1.y--;
        aiPaddle.block2.y--;
      }
      else if (ball.y > aiPaddle.block1.y &&
aiPaddle.block2.y < 7)
      {
        aiPaddle.block1.y++;
        aiPaddle.block2.y++;
      }
    }
    else
    {
      if (ball.y < aiPaddle.block2.y &&
aiPaddle.block1.y > 0)
      {
        aiPaddle.block1.y--;
        aiPaddle.block2.y--;
      }
      else if (ball.y > aiPaddle.block2.y)
      {
        aiPaddle.block1.y++;
        aiPaddle.block2.y++;
      }
    }

    // Light up the new A.I. paddle location.
    matrixController.setLed(0, 0, aiPaddle.block1.y,
true);
    matrixController.setLed(0, 0, aiPaddle.block2.y,
true);
  }
}

// This method is used to change the position of the
player paddle
void movePongPlayer(char dir)
{
  // Clear the old paddle locations.
  matrixController.setLed(1, 7,
playerPaddle.block1.y, false);
  matrixController.setLed(1, 7,
playerPaddle.block2.y, false);

  // Depending on the direction the paddle needs to
be moved in, different actions occur.
  if (dir == 'R')
  {
    if (playerPaddle.block2.y < 7)
    {
      playerPaddle.block1.y++;
      playerPaddle.block2.y++;
    }
  }
  else if (dir == 'L')
  {
    if (playerPaddle.block1.y > 0)
    {
      playerPaddle.block1.y--;
      playerPaddle.block2.y--;
    }
  }

  // Light up the new paddle location.
  matrixController.setLed(1, 7,
playerPaddle.block1.y, true);
```

Метод displayScore() ползваме за да покажем резултатите между рундовете на Понг, както и дали играчът е победил или загубил.

```
  matrixController.setLed(1, 7,
playerPaddle.block2.y, true);
}

// This method is used to display the score after
either of the two
// players scores a point.
void displayScore()
{
  // Clear the displays
  matrixController.clearDisplay(0);
  matrixController.clearDisplay(1);

  // Send information, regarding the scores to the
mobile phone
  String scoresToSend = "C" + String(playerScore) +
":" + String(aiScore);
  Serial.print(scoresToSend);

  // Display the corresponding numbers on the LED
matrices.
  for (int i = 0; i < 8; i++)
  {
    matrixController.setRow(0, i,
numberSprites[aiScore][i]);
    matrixController.setRow(1, i,
numberSprites[playerScore][i]);
  }

  // Wait for user input before continuing and begin
a new round.
  while (Serial.available() == 0) {}
  Serial.read();
  newPongRound();
}

// This method is used to display the winner, once
the Pong game has been finished.
void displayPongWinner()
{
  if (playerScore == pointsToWin)
  {
    displayWin();
  }
  else if (aiScore == pointsToWin)
  {
    displayLoss();
  }

  // Wait for input from the user.
  while (Serial.available() == 0) {}
  Serial.write('V');
}

// This method uses the win sprite to display the
word 'WIN' on the LED matrices.
void displayWin()
{
  for (int i = 0; i < 16; i++)
  {
    matrixController.setRow(i / 8, i % 8,
winSprite[i]);
  }
}

// This method uses the loss sprite to display the
word 'LOSS' on the LED matrices.
void displayLoss()
{
  for (int i = 0; i < 16; i++)
  {
    matrixController.setRow(i / 8, i % 8,
lossSprite[i]);
  }
}
```

ЗАКЛЮЧЕНИЕ

Създадохме проект, който цели да пресъздаде една ретро конзола с помощта на Arduino, като съчетаем остарялото с модерното. Това чувство наистина се постига от ниската "резолюция" на екрана(16x8) и ползване на Bluetooth технологии и смартфон за управлението й. Името ArduinoBoy все пак е съчетание от ползвания сравнително нов микроконтролер и една остаряла, но класическа конзола, което също придава това усещане.

Доволни сме от постигнатото, но искаме да отбележим, че както всеки проект, така и този има някои недостатъци:

- Ниската резолюция наистина ограничава набора от игри, които могат да бъдат програмирани.
- Потребителят трябва да изтегли отделно приложение за да управлява случващото се на екрана.
- Не на всеки ще му се харесат подбраните игри и този "oldschool" вид на конзолата
- Проектът е обемист и др.

Въпреки тези недостатъци, смятаме, че проектът има и перспективи за развитие:

- Може да се добави функционалност за спиране на текущата игра и връщане към екрана за избиране на нова такава
- Може да се обогати преживяването, ако се включат и звуци към игрите и др.