Electronic Design Lab Project Report

Tilt Controlled Gaming Console

Group Members:
Prasad Thakur (06D07015), Abhishek Singh (06D07037), Pravin Mote (06D07024).

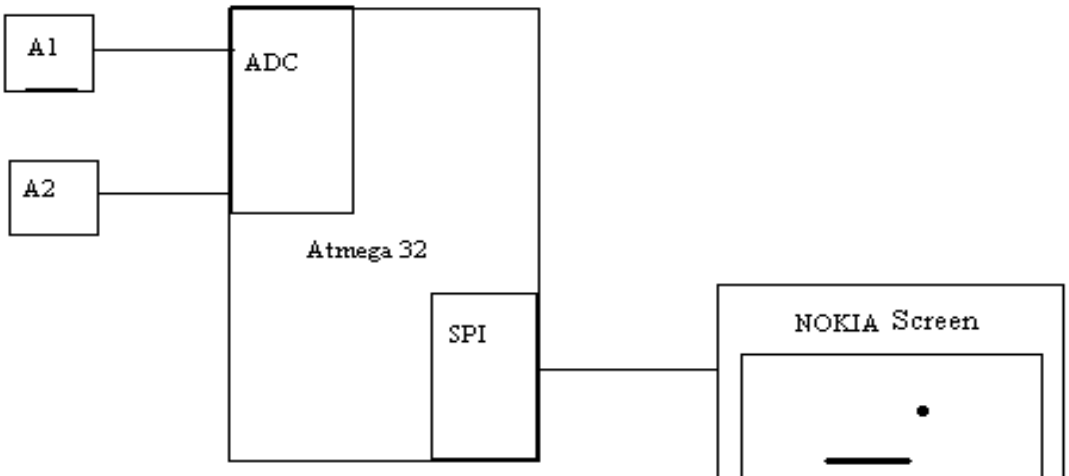
Introduction:

The unit is a portable gaming console in which the user can tilt the unit to navigate the way in a 2D plane. We created graphics using the Atmel Mega32.We also looked to implement the game in a manner that was completely easy and intuitive to play.

The general schematic is as follows: The Atmel Mega32 generates a signal to the screen showing the current position of the snake. The user sees the position and moves in a real space to modify this position within the 2D plane. The accelerometers picks up the values caused due to the tilt made by the user and passes it to the analog to digital converter on the Atmega 32, which then interprets the changes in acceleration and modifies the snakes' position accordingly. This continues until the person reaches the borders or the snake touches itself.

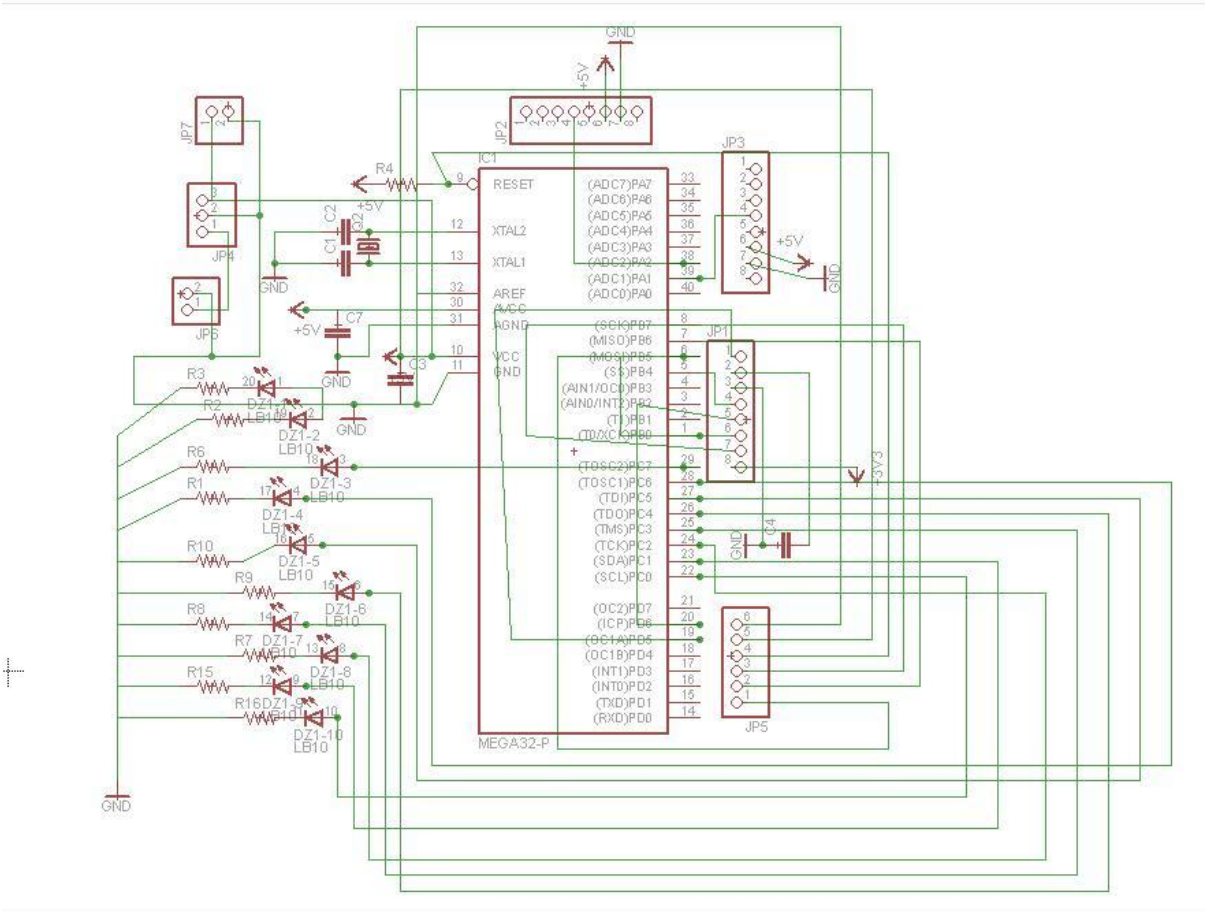
It was realized early on that there would be two main tasks involved in the project--creating the graphics and interfacing with accelerometers.

Block Diagram:



A1: Accelerometer (up-down detection)
A2: Accelerometer (right-left detection)
SPI: Serial Peripheral interface
ADC: Analog to Digital Converter.

Circuit Diagram:



2. Major Units of the Project:

2.1. Screen (Nokia 3310).

The Nokia 3310 screen is controlled by the Phillips PCD8544 CMOS LCD controller driver,designed to drive a graphic display of 48 rows and 84 columns. All necessary functions for the display are provided in a single chip, including on-chip generation of LCD supply and bias voltages, resulting in a minimum of external components and low power consumption.
The PCD8544 interfaces to microcontrollers through a serial bus interface.

The DDRAM of PCD8544 is a 48*84 bit static RAM which stores the display data. The RAM is divided into six banks of 84 bytes (6 * 8* 84 bits). During RAM access, data is transferred to the RAM through the serial interface. There is a direct correspondence between the X-address and the column output number. The timing generator of PCD 8544 produces the various signals required to drive the internal circuits. The addressing can be vertical or horizontal. Also, the display can be operated in normal or inverted mode.

Figure1: RAM format, addressing
.

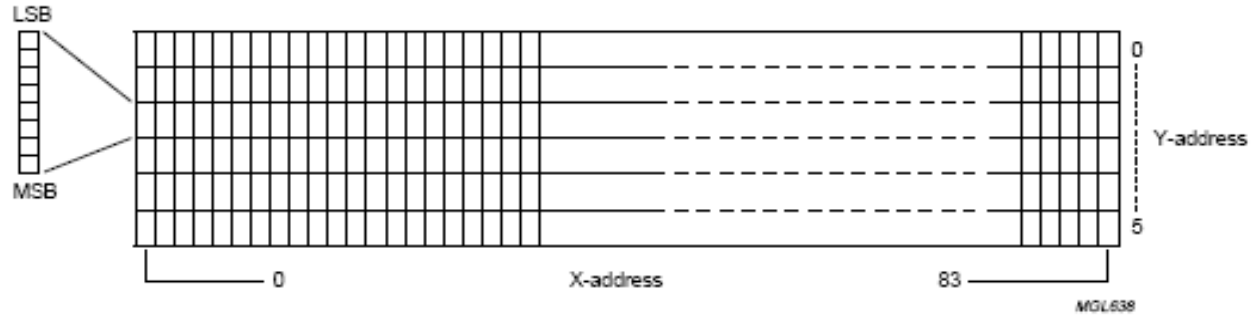
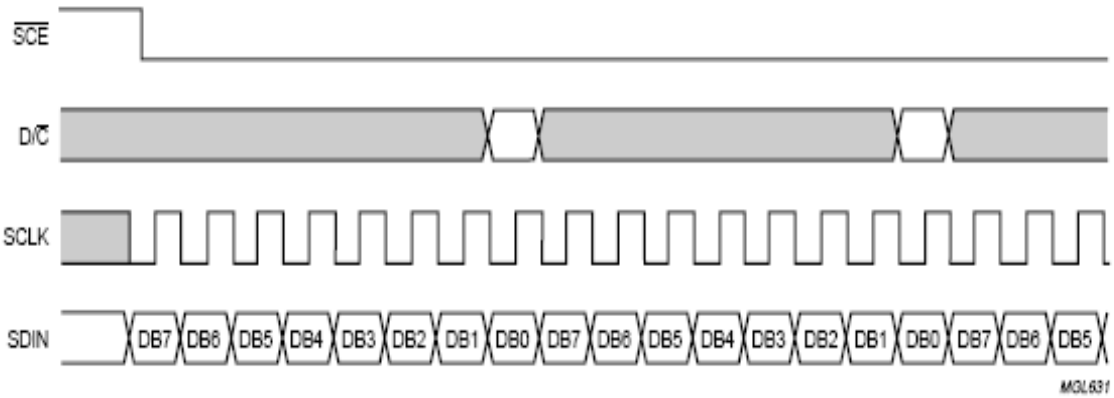


Figure 2. Serial bus protocol - transmission of several bytes.



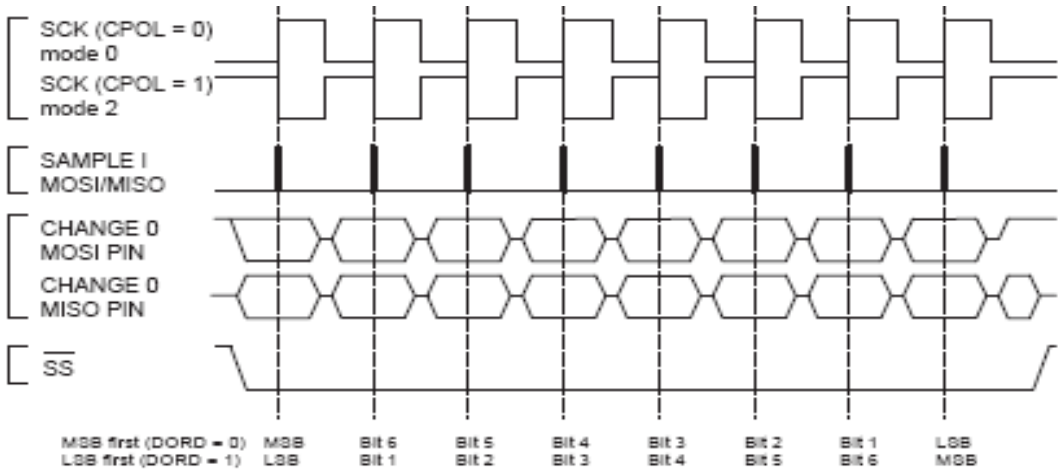
Initially we were not aware of the Serial Peripheral Interface available in Atmega 32.Hence, in order to achieve the above synchronus data transfer, we checked the feasibility of several options available in the Atmega 32 like inbuilt timers/clocks,USART etc.We also

tried out various supporting circits in order to drive the LCD screen. One such attempt included the use of a PCB whcih implemented a small resistive,capacitive circuit to provide optimum voltages and currents to the various inputs of the screen.However,after considerable part of the initial lab session in testing these tings,we concluded that these approaches were incorrect.

Later we realized that the timing diagram shown above is the same as that of the Serial Peripheral Interface of the Atmega 32.The Serial Peripheral Interface ensures proper values of the 'Setup Time' and 'Hold time'. Also, the Serial Peripheral Interface (SPI) allows high-speed synchronous data transfer between the ATmega32 and peripheral devices or between several AVR devices. The SPI ensures Full-duplex, Three-wire Synchronous Data Transfer. It can be used in Master or Slave Mode Operation. We have LSB First or MSB First Data Transfer option and seven programmable bit rates to choose from. When the transmission is finished the end of transmission Interrupt Flag is modified.

We operate the SPI in Master Output Slave Input mode (MISO). When configured as a Master, the SPI interface has no automatic control of the Slave Select line. This is handled by user software before communication can start. When this is done, writing a byte to the SPI Data Register starts the SPI clock generator, and the hardware shifts the eight bits into the Slave. After shifting one byte, the SPI clock generator stops, setting the end of Transmission Flag (SPIF). The Master may continue to shift the next byte by writing it into SPDR, or signal the end of packet by pulling high the Slave Select (SS) line. The last incoming byte is kept in the Buffer Register for later use.

Figure 3. : Mode 0 operation of the Serial Peripheral Interface



We use the SPI in Mode 0 as shown above.(when CPHA=0,CPOL=0).This results in the data transfer as required.

The Atmega32 has about 2KB of Data Memory and 1 KB of EEPROM memory.We define a array called number[84][6].Every byte stored in this array corresponds to a column(0 to 83) in a particular bank(0 to 5).As per the specifications(figure 1) of PCD8544,the MSB of the stored byte corresponds to the lowermost pixel in paricular column of a bank.

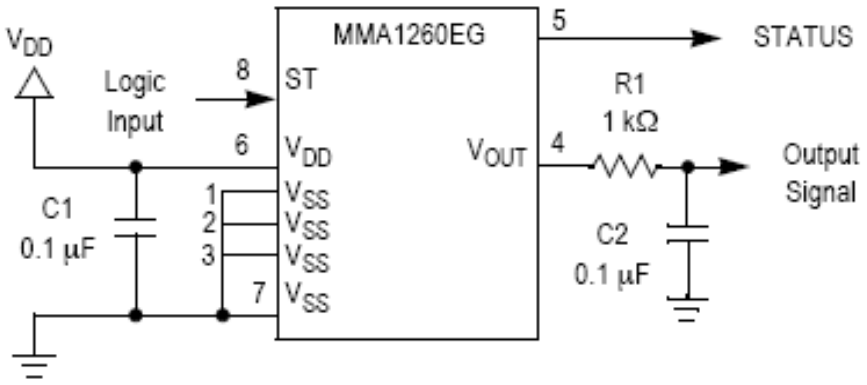


Figure4. SOIC Accelerometer with Recommended Connection Diagram

2.2 Accelerometers:

We make use of 2 sampled peices of Z axis sensitivity MMA1260EG Low G accelerometer for the detection of motion(tilt).When no 'g' is acting on the accelerometer,the output is 2.5 volts.The sensitivity of the unit is 1200mV/'g'.It also has a calibrated self test option. One of the accelerometer detects motion along an axis in the plane of the screen (left-right) while the other measures motion in the same plane but in along an axis perpendicular to the previous axis (up-down).

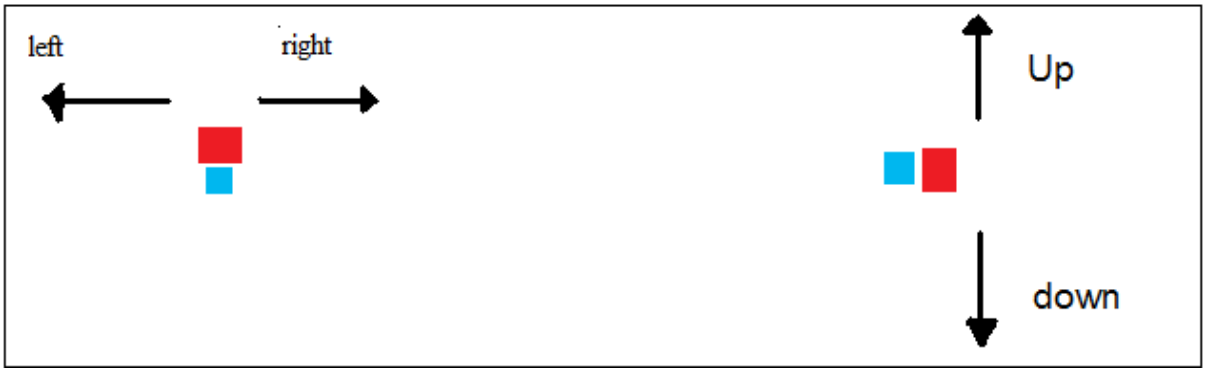
The accelerometers are used as tilt sensors. Whenever the device is tilted in a positive 'g' direction the accelerometer output varies from 2.5 volts to 5 volts. On the other hand, the output varies from 2.5 to 0 volts in case of negative 'g' inputs. This output value is proportional to the 'g' acting on the device.

This output is fed to the Analog to Digital converter of the Atmega32. The ATmega32 features a 10-bit successive approximation ADC. We use the prescaling circuit and feed (CK/128) to the ADC clock source. By using an appropriate control word we select the first 8 bits starting from the MSB and neglect the last 2 bits.

In order to avoid false inputs due to small deflections and other system noise, we have decided upon 2 threshold values. The threshold value has to be surpassed in order for the signal to be accepted as the input. Upper threshold value (3.2 volts) deals with the positive 'g' inputs to the accelerometer and lower threshold (1.8 volts) deals with the negative 'g' inputs.

As mentioned earlier, the output of each of the accelerometer is fed to one of the differential input channels of the ADC. Now, at any given moment, the snake has only 2 directions to go. For example, if the snake's 'head' and 'neck'(pixel immediately after the head) have the same column, then it can go either right or left.(we do not consider going straight as a direction because that is the expected path of the snake when no input is given). The 'head' is shown in red while the 'neck' in blue.

Figure: Big Block is the head while the smaller is the neck



Thus once we take an input from one of the accelerometers, we are must take input from the other accelerometer unit in the next turn (which measures motion in a direction perpendicular to the previous input). That is, we must not take consecutive inputs from the same accelerometer.

Hence we multiplex between the 2 accelerometers. This ensures that once we have taken input from one accelerometer, it is disabled until we get input from the other accelerometer unit.

The accelerometers are positioned in order to give the following outputs:

Table 1.

	Direction of turn when Upper Threshold crossed	Direction of turn when Threshold crossed
Input 1 of ADC	Right	Left
Input 2 of ADC	Up	Down

Thus the snake can be manoeuvred in the 2D plane.

2.3. Snake game Algorithm:

Our game algorithm tries to imitate the commercially available game. The Rules of the game are:

1.The snake should not touch the borders. If the co-ordinates of the head become equal to that of the borders, the game is terminated.

2. If the co-ordinates of the head become equal to the co-ordinates of any of part of the snakes' own body, the game is terminated.

3. When the co-ordinates of the head of the snake become equal to that of the food particle, the length of the snake increments by one.

4. When no input is given, the position of the snake increments in the direction of the previous input, until the borders are reached.

Working of the code:

We have defined an array which is responsible for the display of the body of the snake. We have predefined the maximum length of the snake to be 20.Initially, out of these, only 5 are

displayed and the rest are invisible. The invisible length is not considered when in the case when the snake's head collides with the invisible length. We define the initial position of the snake to at the centre of the screen with the head pointing to the right.

We have also defined a function called 'gameover' and 'newfood'. The gameover displays the characters 'END' whenever the snake touches the borders or itself. The display of above characters is done using the horizontal addressing mode of the PCD8544 screen.

The food particle is a 2*2 pixels image displayed on the screen. We have chosen this in order to make it more prominent on the screen. The position of this food particle is generated using a random function which is included in the 'newfood' function. Precaution is taken that the food particle is not generated on the border and on the entire length of the snake. When the snake consumes the food particle, the length of the snake is incremented by one (i.e. one more snake pixel is made visible) .

The 'turn' function is called whenever the input from the accelerometers exceeds the upper and lower threshold values. Whenever the an input is received, We have defined four variables called r, l, u, d which control the right, left, up, down turns of the snake in the plane of the screen. The snake turns in the direction corresponding to the variable which is set to 1.The function also takes care that the consecutive inputs are not taken from the same accelerometer.

The main function starts off by initializing the ADC. Port A is used as input for the accelerometer output. This is followed by the SPI(serial peripheral interface) initialization. The initial position is then set by defining the X and Y coordinates. The X co-ordinate is 'OR'ed with 80 to ensure that the MSB remains 1, which is as per the requirements of PCD8544.

The snake's entire body follows its head. Hence, the neck follows the head and pixel after the neck follows the neck. Thus, we start the process by copying coordinates from the tail (last pixel) of the snake to the head of the snake. This way of copying the coordinates ensures that no data is lost. After the entire body of the snake including the neck is updated, the position of the head is updated depending on which of the direction variables (l, r, u, d) is set to 1.

Immediately following power-on, the contents of all internal registers and of the RAM of the PCD 8544 are undefined. A ~RES(reset bar, active low) pulse must be applied. Attention should be paid to the possibility that the device may be damaged if not properly reset. All internal registers are reset by applying an external RES pulse (active LOW) within the specified time. However, the RAM contents are still undefined.). The ~RES input must be less than or equal to 0.3VDD when VDD reaches VDDmin (or higher) within a maximum time of 100 ms after VDD goes HIGH. This is implemented using the software code.

After each change in position of the snake, the game termination conditions are checked. The snake display code is put in an infinite loop and breaks out of this loop only when the termination conditions are met. At the end of each cycle, the Port A is checked for any new input from the accelerometer. Thus the game continues.

Adding new speed levels and maze structures:

The major problems we had to solve in the project involved storage space and programming complexity while attempting the 2*2 unit pixel to display the snake instead of the normal 1*1

unit pixel. Since we are programming on the Atmel Mega32, data space was a scarce resource. We managed to come up with a new compact way to represent the pixels on the Nokia screen that reduced the data memory required by a factor of 8. (Earlier we were using an [48]*[84] array, but now we are using [6]*[48] array to display things on the screen). Our new way of declaring the array proved to be more intuitive and optimum for the bank wise addressing scheme followed by the PCD8544.Also the new representation reduced bit rate between the SPI and screen considerably. This allowed us to achieve reasonable frame-rates, kept within the storage space and that was tenable to code within the allotted time frame.

In order to make the game as real as possible, we have included a variety of speed levels and maze patterns. As the user completes one level, a new maze is drawn to make the game more challenging. Also when 3 such levels are completed, the speed of the snake increases by a factor of 2. This speed increment is achieved by reducing the delay between 2 iterations.

Thus, the final version of the game is an unending sequence of levels, each more difficult to handle than the previous.

A noticeable addition to our project was the Atmega 32 programmer circuit that we built in order to avoid frequent removal of the Atmega32 from the bread board. We were lucky, in the sense that the programmer circuit worked correctly in the very first attempt.

Conclusion:

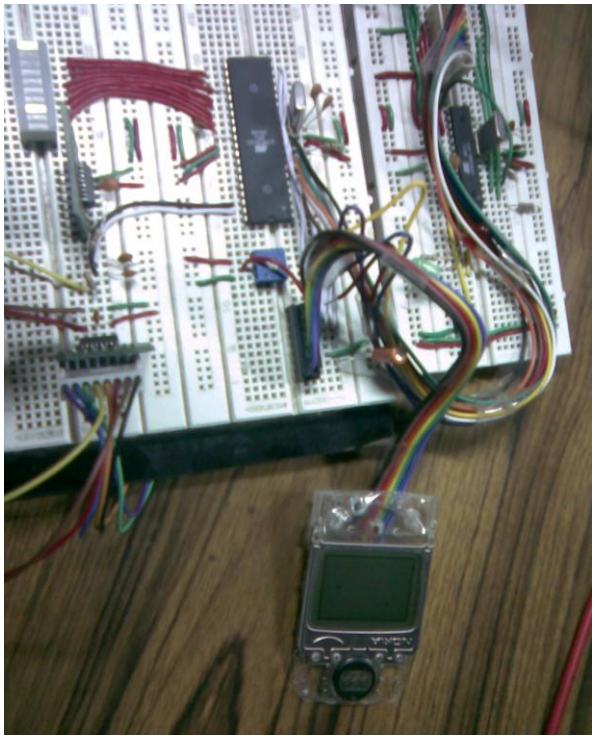
We are satisfied with the results of our project, as it had met most of our expectations. If we were to have more time, we would have made the product more polished such as menu screens, reset buttons and a more creative level structure for the mazes. But for all intents and purposes, the game in its current iteration is indeed very entertaining.

Component Details:

Quantity	Item
1	Atmel Mega32
2	MMA1260D Accelerometer(Sampled)
1	Nokia 3310 screen

Some Pics of the working project:

This picture shows the game in progress. The borders, snake and the food particle are visible.



The overall working circuit on breadboard.