Speech Enabled Alphabet Blocks

By

Will Lovett
Nathaniel Lee
Khomkrit Kaowthumrong

1. Introduction

We were inspired by the design of the Music Blocks toy by Neurosmith. In this paper we attempt to describe our prototype implementation of a system of alphabet blocks that contain embedded computation. Each block by itself is a self-contained sensor network that can detect its orientation and neighbors. The prototype that we present here is a simplified system of blocks that can be assembled to spell out words. The word is then relayed to a near-by computer and processed by a speech synthesizer. We hope that this prototype can be a basis for future research into the use of sensor networks in the field of child education.

In the following section we will present the overall design of our implementation. This paper then continues with an evaluation of the system and the overall educational value in section 3. Section 4 describes the computation involved in the system. Section 5 describes future work that may be done on the system to expand its capabilities and usage.

2. Design

Before finalizing our design, several ideas were considered. The first project idea involved musical notes using the beeping sounds of the programmable crickets. It would start with a base note upon which consecutive crickets would modify the note up or down by changing the pitch. Small blocks would be constructed to hold the crickets and give them orientation. By attaching gravity sensors to these blocks, the position of the blocks could then be determined. A block positioned with an arrow pointed up would raise the pitch, while a block positioned with an arrow pointed down would lower it. Building on this concept, letters were introduced to represent their corresponding musical note. For example, the letter ‘C’ would represent a cricket beep with a musical tone of ‘C’. Chaining these letters together could form a short musical score. Once the letters were placed next to one another, the parallels to language became obvious. If individual blocks could represent a musical note, then they could represent a letter as well. Placing letters in a certain order would produce words instead of musical notes. Hence, a language device was realized by connecting individual letter blocks to form simple words.

Once the language device was decided upon, the hardware extent of the project became the primary focus. The outer shell was constructed of interlocking plastic pieces approximately three inches in width. Each piece snapped into the adjoining pieces creating a cube frame. Since the connection between pieces was not permanent, modifications could be handled quickly and easily. The crickets were placed in the middle of the cubes using Velcro. After the crickets were in place, touch sensors detecting the presence of neighboring letter blocks were installed. The addition of gravity sensors enabled the variations in output. A block could then represent more than one letter based on whether the sensor was activated or not. One sensor idea was designed using wire, a tube, and a metal ball bearing. At one end of the plastic tube, two loops of wire
were placed so that they nearly touched. A small metal ball bearing was inserted into the tube.
When the ball rolled onto the two wires, the circuit would be completed thereby ascertaining the
orientation of the letter block. The gravity sensors were an interesting design, but in the end,
touch sensors were used due to their proven track record. To communicate between blocks, fiber
optics were placed from the cricket’s infrared output to the opposite end of the block. The fiber
optic output was lined up with the next cricket’s infrared input. The use of the fiber optics
directed the flow into a concentrated beam minimizing signal leakage and interference. Finally,
stick-on letters were placed on two sides of each block corresponding to both the letter and the
orientation of the block.

With the block’s body completed, software was written to bring them to life. For the sequence to
start, a beginning block would send out its signal if it "felt" an adjoining block.

The signal was composed of its numerical letter counterpart and a terminating character to
represent the end of the transmission. The next block awaiting the signal would then add its own
bits and a terminating character to represent the end of its contents. Once this cascading effect
reached the last block, the output was sent to a final cricket connected to a PC. Using speech
software, this numeric input of bits and terminating characters was converted into alphabetic
equivalents. Each letter’s sound was then produced, streaming together to form recognizable
words. Any word contained in the speech software’s database can be created with the use of the
"speaking blocks."

3. Evaluation & Education
Over the course of designing and building this project, we have run into numerous set-backs.
One of the first problems we ran into was getting the crickets to determine the orientation of the
blocks. The original gravity sensor idea (outlined in section 2) had the potential of working very
well. Unfortunately, due to poorly conducting materials, it was not consistent enough for our
robust prototype. After further evaluation and testing, it would be preferable to replace the
external touch sensors with some type of internal gravity switch, such as a mercury switch. This
would allow for a much more robust exterior as well as more consistent readings.

Another problem we came across when assembling the communicating blocks was related to the
other set of sensors. Our original idea called for micro-momentary switches on the side of a
block that would detect neighbor blocks. Unfortunately, because the blocks are so light, the
spring inside the switch was strong enough to push the neighboring block away. This was also a
problem for the switch on the bottom, but the addition of ballast weight was the remedy for this
instance. To solve the side switch problem, we decided to change the detection method. Our
solution was to place to sensor nodes on the side of the blocks, and then a conducting plate on
the next block. When the blocks were touched together, the sensor nodes contacted the
conducting plate, completing the circuit. This works reasonably well, but would not be an ideal
design for a production model due to the lack of durability.

The last major problem we encountered while building our prototype was the physical alignment
of the cricket IR transceivers. Our original design required the use of fiber optics to transmit
output data from one side of the cricket to the opposite face of the block. Since the fiber optic
wires were not stiff enough to maintain their position, we added paperclips as guides. Even with
the added control over the orientation of the fiber optic direction, we were unable to maintain consistent signal integrity. Our solution was to mount the fiber optic in the very center of a solid plate on the output side of our block. This ensured that no matter what orientation the parent block had, it would be transmitting its data to the center of the daughter block. This also meant that we had to align all the crickets so that the receiver ports were in the center of the block. This was accomplished by a combination of riser blocks and Velcro, for lateral control.

The concept of communicating blocks has great potential for the education of children. The most obvious application of these prototype blocks is in teaching how to spell. With the expansion of the blocks to include letters on all six sides as well as increasing the number of blocks that can be used, the complexity increases rapidly. This same concept could be reformatted for other ideas as well, such as word blocks to form sentences, math blocks that perform functions, or note blocks that form musical compositions. Beyond the basic levels of simple block combination, the blocks could be made to directly interact with computer programs on the host computer. For instance, a screen shows a cartoon-like bedroom that shows a cat that is the wrong color. The computer would ask the child, probably by voice, what had the wrong color-- the child would have to respond by spelling out cat. These types of multi-directional interaction could be expanded to many different applications including possibly college level math problems.

4. Computation
The computational aspect of our block system is embedded in the logic of the blocks. The block, through the use of touch sensors, can detect its orientation and neighbor. Each block in our prototype contains only one Cricket. In a production system this number may need to rise to six. This rise in the number of Crickets is due to the limited number of sensors that a Cricket can handle at any one time. The single prototype is responsible for detecting the orientation of the block. The pressure or lack of pressure on the touch sensor allows the Cricket to determine which side is exposed. A second sensor is needed to detect the presence of an adjacent block. On the full implementation, a Cricket may be needed for each side of the block. The control logic of this setup is yet to be fully determined and tested. The addition of customized hardware that can read from a larger number of sensors may be useful in the construction of the full production system.

IR is used in order to relay information back to the host computer where the bulk of the speech processing occurs. The Crickets in our prototype blocks emit a series of IR pulses that correspond to a predetermined set of message parameters. The current implementation uses a space in Latin Unicode to represent a start of message signal. (See Appendix A for Cricket Code) Upon receiving this signal, the receiving Cricket will begin recording the next set(s) of IR signals into memory. These intermediate signals are represented by a 2-byte code. In Unicode these 2 bytes represent the corresponding letter in the alphabet. The complete set of Unicode standards can be found at http://www.unicode.org. The termination code that signals the completion of the message is a “/”. This code again is sent in 2 bytes. Upon receiving the termination code of “/”, the receiving Cricket will then transmit the message that it has just received with its own identity code appended to it before the termination code. On the production version (6 sides) the transmission could be accomplished through the use of offset holes on the sides of the blocks. For example, each side of the block could have 4 holes placed at each corner. A set of holes
would also exist on the receive side of the adjacent block. As the block rotates the valid transmit hole would be the upper right hole of the side. The corresponding receiving hole would be the upper left on the receiving block. As the block is rotated the hole alignments would change thus providing different connections or transmit paths for the Crickets inside the block. The entire message is then relayed to the computer via the interface Cricket by the final block in the sequence.

A speech enable program was written to process all the IR information passed into the system via the interface Cricket. This program resides on the host machine that is connected to the interface Cricket. To collect all the bytes received by the interface Cricket, the serial port that the interface Cricket is attached to is opened and any bytes that are sent to the system are processed accordingly. An MSComm object was instantiated in Visual Basic 6.0 to receive any input from the interface Cricket. Only a message that starts with a space will be processed. This limitation protects the system from reading in messages that are incomplete or from processing message starting in the middle. Upon receiving the termination code of “/” the word that was sent to the system is passed to the speech synthesizer and spoken. The speech synthesizer used in this program is comprised of the Microsoft Speech API version 5.1. (See Appendix B for full code for speech program)

5. Future Direction

We plan to build additional blocks and program them for many possible uses. Additional blocks could be made for the existing system to expand the possible combinations of words that the system can spell out. This set of blocks could then be used to see if the addition of computation and speech enabled alphabet blocks provide an added benefit to children who are learning how to spell words. All six sides of the blocks could be made functional for this test. This would mean that for a small number of blocks, say 12 blocks, a very large number of words could be made.

Additionally, equate blocks and variable blocks could be made so that they could receive and store data on a variable basis, thus providing the ability to store known words in memory. This capability would allow users to spell out entire sentences using our blocks. This may provide children a hands-on approach to learning how to read and write.

Expanding on this idea of programmable blocks, an entire programming language could be written around the block construct. Each block in the language would represent a possible basic construct of the language. Again here variable blocks could be made to signify variables in the programming language. Flow control blocks could also be made to allow users to control the flow of execution of their program. All this information for each line or arrangement of code could then be sent back to the host computer for further processing.

Additionally, the coding scheme need not be 2 dimensional. The block could be arranged in a 3 dimensional configuration. The blocks could be stack on top of one another to build a program in 3 dimensions. The semantics of this new 3 dimensional arrangement still need to be explored further.
Appendix A, Cricket Code

Cricket 1:
global [value1]

to cricket1
loop
ifelse switcha
  setvalue1 66
][
  setvalue1 67
]
if switchb
  send 0
wait 1
send 32
wait 1
send 0
wait 1
send value1
wait 1
send 47
wait 1
]
]
end

Code for Cricket 2:
global [temp-ir]
global [value1]
global [value2]

to cricket2
loop
  waituntil [newir?]
  settemp-ir ir
  if temp-ir = 0
    waituntil [newir?]
    settemp-ir ir
    if temp-ir = 32
      waituntil [newir?]
      settemp-ir ir
      if temp-ir = 47
        send 0
        wait 1
        send 32
        wait 1
        send 0
        wait 1
        send value1
        wait 1
        send 47
        wait 1
      ]
    ]
  ]
end

Code for Cricket 3:
global [temp-ir]
global [value1]
global [value2]
global [value3]

to cricket2
loop
  waituntil [newir?]
  settemp-ir ir
  if temp-ir = 0
    waituntil [newir?]
    settemp-ir ir
    if temp-ir = 32
      waituntil [newir?]
      settemp-ir ir
      if temp-ir = 47
        send 0
        wait 1
        send 32
        wait 1
        send 0
        wait 1
        send value1
        wait 1
        send 47
        wait 1
      ]
    ]
  ]
end
send 0
wait 1
send 32 ; start code of space
wait 1
send 0
wait 1
send value1 ; the previous cricket's code
wait 1
send 0
wait 1
send value2 ; this cricket's code
wait 1
send 0
wait 1
send 47 ; termination code
wait 1
]
]
setvalue2 temp-ir ; since value2 was not a termination code get one more code
ifelse switcha[
    setvalue3 78 ; set the value of this cricket to N
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
]
Appendix B, Speech Processing Code

Option Explicit
Dim inputBuffer As String ' Input Buffer
Dim flag As Boolean ' Correct Start Flag

' If the speak button was pressed then speak what was in the textbox
Private Sub actionCmd_Click()
    spkSpeak.Speak txtText.Text
End Sub

Private Sub Form_Load()
    ' set the form's caption
    Caption = "TTS Server"

    ' open the comm. port
    MSComm1.PortOpen = True
End Sub

Private Sub Form1_Unload()
    ' close the comm. port
    MSComm1.PortOpen = False
End Sub

' Comm port control code
Private Sub MSComm1_OnComm()
    ' Event messages.
    If MSComm1.ComEvent = comEvReceive Then
        ' Put the input from the serial port into the input buffer
        ' This line also converts the 2 bytes into unicode
        inputBuffer = StrConv(MSComm1.Input, vbUnicode)

        ' select which unicode character was received
        Select Case inputBuffer
            Case ""
                txtText.Text = ""
                flag = True
            Case "/"
                ' speak the textbox text
                spkSpeak.Speak txtText.Text
                flag = False
            Case Else
                If flag = True Then
                    ' append the received character to the existing text characters
                    txtText.Text = txtText.Text + inputBuffer
                End If
        End Select
    End If
End If
End Sub