

# Close Encounters Project Report

Edith Hand  
James Smagala

## 1. Abstract

This automaton was inspired by the landing scene from the movie "Close Encounters of the Third Kind." The automaton consists of a simple scene displaying Devil's Tower and the alien landing site. The mother ship is behind Devil's tower and is controlled by a motor which can either hide or display the mother ship. There are also two smaller space ships next to the landing site which are controlled by a second motor. The main feature of this automaton is that the spaceships respond differently to different sets of musical tones. One cricket is used to control two different sets of tones to play. One set of tones is the five-note 'hello' sequence which should be familiar to anyone who has seen the movie. The second set of tones effectively creates a 'musical insult'. Which of the two sets of tones gets played is randomly selected. A second cricket responds to the tones played and takes a different course of action for each set of tones. This cricket controls the motor that displays or hides the mother ship and the motor that starts or stops the two smaller spaceships from spinning.

## 2. Design

### 2.1 Mechanisms

A Finite State Machine functions as the core computational mechanism for this automaton. Figure 1 shows the Finite State Diagram for this automaton.

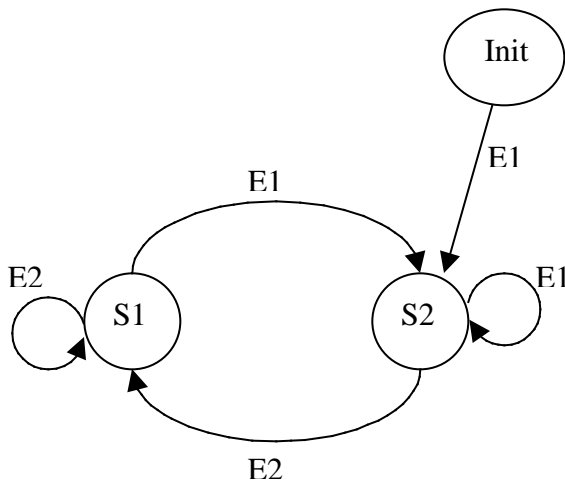


Figure 1: Finite State Diagram

S1 and S2 represent the two states that the automata can be in. S1 represents the state in which the mother ship is hidden and the small spaceships are still. S2 represents the state in which the mother ship is showing and the small spaceships are moving. E1 and E2 represent events. E1 represents the event where the singer sings the five-note 'hello' sequence. E2 represents the event where the singer sings an insult. In either possible

state, the decision of whether to initiate event E1 or event E2 is chosen randomly. The automaton begins in an initial state, designated 'Init' in Figure 1. From the initial state, event E1 is automatically initiated.

Mechanically, the mechanisms used are simplistic. The two small ships are mounted to a platform that is turned directly by a motor. The mother ship is attached to a long wooden shaft that lifts it from a through an arc of approximately 90 degrees in order to have it rise above or sink below the tower.

## **2.2 Materials and Construction Techniques**

The main materials used for this automaton are wood, pottery clay, and newspaper. The flat base is a sheet of particle board. The main support beneath Devil's Tower (providing its shape) is a large plastic cup. We used newspaper, duct tape, and masking tape liberally to build the underlying structure of the hills surrounding the landing site. We then used the modeling clay over all of this to create the final look of the landmark. The arm used to support the mother ship and the base for the other spaceships are made from basswood. There is also a small amount of Lego in the automaton. Both of the motors used to power the ships are Lego motors. Therefore, parts of the assemblies to attach the moving parts to the motors are made from Lego. All of the spaceships are made out of paper. The reflective black surface of the landing site is an unwanted PlayStation disc.

## **2.3 Design Process**

The initial design for the automaton was inspired by the landing scene from the movie. Several smaller ships are seen to be moving around the large landing pad at the foot of devils tower, then the mother ship appears from behind the tower, rising slowly. Communication takes place by a complex musical language, in which the well-recognized five note sequence seems to be a greeting. The use of these three events forms the core of our design. The initial design involved mechanisms that were significantly more complex than those that were actually implemented. Originally, to make the mother ship rise straight up, we planned to use a pile driver mechanism with a notched shaft and a gear missing several teeth. The ship would rise, and then fall as the gear turned to a section with no teeth. Mechanically, this did not work due to weight issues as well as an inability to stop the ship in an upright position, which would have prevented the finite state mechanism from being effective. Realizing that the ship would still appear impressive even if it did not rise from directly from behind devils tower, it was mounted directly to a shaft attached to the motor, rising or falling along an arc of approximately 90 degrees between its two states.

The mechanism for the smaller ships was always supposed to be a simple rotary motion, though the use of a direct attachment to a platform turned by the gear arose from a desire to simplify the motion. Originally, the ships were to be geared down to control the rate at which they turned, and the gearing mechanism hidden under the model. Again, for simplicity, the final design did not gear down the speed and the mechanism was mounted above the model.

The design of the cricket code to implement the finite state machine proceeded almost exactly as planned. The original design was a finite state machine that would have at least two states. We had originally planned to implement more, but were unable to due to time constraints. One cricket randomly selects a transition, plays a corresponding tune, and sends to the second cricket, which responds according to the signal and its current state. The first cricket was originally supposed to control additional lights and sounds, but these did not get implemented.

## **2.4 Influences**

Certainly, the automata we looked at earlier in the semester had an influence on the original design our automaton. However, two factors greatly reduced their influence on the final design. The first is that traditional automata artists attempt to implement complex actions and interactions mechanically. This works at cross purposes with the inclusion of crickets in the project. By adding computation, complex interactions can be achieved with simpler mechanisms. Additionally, the difficulty of implementing a reliable mechanical mechanism of the complexity displayed in a Ganson piece or Cabaret piece becomes rapidly apparent when seemingly simple mechanisms do not work as expected. In the interest of having a simple but working automaton, as opposed to a complex broken one, traditional mechanisms gradually crept out of the project and were replaced by less complex mechanisms and more complex computing.

## **3. Evaluation/Education**

### **3.1 The Automaton as a Learning Experience**

There is little that could be learned from the finished automaton directly. A far more enlightening way to interact with an automaton is to try to build one. There was some discussion at the beginning of the class that the automaton that left their mechanism visible provided a better learning experience than those that hid it. This is perhaps true in its limited way, but in retrospect, seeing a mechanism does not allow a very complete understanding of the way in which it actually works, while attempting to build one requires a learning interaction to take place.

### **3.2 Ease of Use**

As it currently stands, the crickets must be turned on in a specific order, which makes it slightly non-intuitive to operate. This could easily be simplified by implementing a short routine in the control cricket that poles the second cricket repeatedly until it gets a response. Past this point, the automaton operates itself as a finite state machine, which needs no intervention.

## **Summary**

The understanding of various mechanical or computational principles that underlie an automaton cannot be appreciated by observing the work of other artists, and cannot be fully grasped by assembling a kit that recreates an artist's work. The process of designing and attempting to implement one's own mechanism forces an understanding of the complexities underlying automata in general that are hidden in finished product of another artists work.

## Appendix A – Project Photos

Photos will be attached as an appendix to the appendix. We have several, but the software needed to retrieve them from the camera is at Edith's parents' house. We could not get there in time to include them with this report due to an inability to move either of our cars from amidst large piles of snow.

## Appendix B – Cricket Code

```
global [m-temp-ir] ; global var for mother ship
global [prev-ir] ; global var for previous ir
    ; val of mother ship
global [wat=len] ; length of time to wait
global [random-tine] ; determines which tune to sing
```

```

;
; Subroutine show
; This subroutine turns on motor b which
; spins the small spaceships and turns
; on motor a 'thisway' for 1.5 seconds
; which shows the mothership
;
to show
    ; check to see if the ship is hiding
    if prev-ir = 2 [
        b, setpower 1 ; slow it down
        b, on ; turn it on
        a, thisway
        a, setpower 2 ; slow it down
        a, onfor 15 ; on for 1.5 seconds
    ]
    ; otherwise, we're already showing,
    ; so don't move the motor

    send 3 ; say we're done showing
end
```

```

;
; Subroutine hide
; This subroutine turns off motor b
; and turns on motor b 'thatway' for
; 1.3 seconds which hides the mothership.
; Note 'thatway' must have a shorter
; duration than 'thisway' because 'thisway'
; has to fight against gravity but
; 'thatway' doesn't. Setting the
```

```
; durations the same results in much
; crashing!
;
to hide
  ; check to see if the ship is shown
  if prev-ir = 1 [
    b, off
    a, thatway
    a, setpower 2 ; slow it down
    a, onfor 13 ; 1.3 seconds
  ]
  ; otherwise, we're already hiding,
  ; so don't move the motor

  send 4 ; say done hiding
end
```

```
;
; Subroutine init-listen
; This subroutine is the first one
; called by the cricket controlling
; the spaceships. It initializes
; the prev-ir global variable to
; indicate that the mothership is
; hiding. It then calls the
; subroutine listen.
to init-listen
  setprev-ir 2 ; say the ship is hiding
  listen ; listen for the singer
end
```

```
;
; Subroutine listen
; This subroutine waits for new input
; received on the ir port. Once a new ir
; value is received, it decides whether
; to show or hide the mothership based on
; this value. It then updates the
; prev-ir global variable and recurses
; to do it all over again.
;
to listen
  waituntil [newir?] ; check for new ir byte
  setm-temp-ir ir ; sets m-temp-ir to ir
  if m-temp-ir = 1 [show]
  if m-temp-ir = 2 [hide]
```

```
; update previous ir val
setprev-ir m-temp-ir
```

```
listen ; do it again
end
```

```
#####
;### Begin subroutine definitions
;### for the cricket controlling
;### the spaceships
;#####
```

```
;
; Subroutine sing-a-note
; This subroutine takes as input
; the frequency of a note to play.
; It then plays the note for 0.5
; seconds and waits for 0.1 seconds.
```

```
;
to sing-a-note :value
  note :value 5
  wait 1
end
```

```
;
; Subroutine init-tune
; This subroutine is the first one
; called by the singing cricket.
; It sings a tune, then sends an ir
; signal to the mothership cricket
; telling it to show the ship. It
; then enters the recursive function
; talk-to-ship.
```

```
;
to init-tune
  sing-a-tune
  send 1 ; tell the ship to show itself
  talk-to-ship
end
```

```
;
; Subroutine sing-a-tune
; This subroutine plays the five-note
; 'hello' sequence from the movie
; 'Close Encounters of the Third Kind'
```

```
;
to sing-a-tune
  sing-a-note 40
  sing-a-note 35
  sing-a-note 45
  sing-a-note 89
  sing-a-note 59
end
```

```
;
; Subroutine sing-bad
; This subroutine plays a three-note
; sequence which we're calling the
; 'bad song'. The point of this song
; is to, in a sense, offend the mothership
; hence causing it hide.
; NOTE: If you are a musician and you have
; perfect pitch (or if you can figure out
; what notes the frequencies translate to),
; you might understand why the mothership
; would be offended :)
;
; Note that we don't call sing-a-note here
; because we actually want the pitches
; to play for a shorter duration.
```

```
;
to sing-bad
  note 89 2
  wait 1
  note 70 2
  wait 1
  note 79 2
  wait 1
end
```

```
;
; Subroutine talk-to-ship
; This subroutine first calculates some random
; numbers. wait-len is a pseudo random number
; between 0 and 64 that specifies how many tenths
; of a second that the singer should wait before
; singing again. random-tune is a psuedo-random
; number between 0 and 32 which determines which
; song the singer should sing. If the number is
; less than 16, then it sings the bad song,
; otherwise, it sings the 5-note 'hello' sequence.
```

```

; This subroutine is recursive and hence just
; keeps going and going...
;
to talk-to-ship
  ; get a random number to decide how long to wait
  setwait-len random
  ; now scale it down
  setwait-len wait-len / 1000
  setwait-len wait-len * 2
  ; now wait-len is roughly between 0 and 64

  ; get a random number to decide which
  ; tune to sing
  setrandom-tune random
  setrandom-tune random-tune / 1000
  ; now random-tune is roughly bet. 0 and 32

waituntil [newir?] ; wait for ship to 'talk'

ifelse random-tune < 16 [
  wait wait-len ; wait a 'random' amount of time
  sing-bad      ; sing a bad song :(
  send 2        ; tell the ship to hide
] [
  wait wait-len ; wait a 'random' amount of time
  sing-a-tune   ; sing pretty music :)
  send 1        ; tell the ship to show
]

talk-to-ship ; do it again
end

```