The purpose of this project has been to update and upgrade existing Soar 6 code for the Missionaries and Cannibals problem. In addition, a GUI for the problem has been created, and data obtained from three subjects using this interface. The data has been used to assess the extent to which the Soar model accurately reproduces human performance. The results show that the end result of the model does correlate with human data, but the way in which the model reaches that state (with regard to both time and number of moves taken) is largely unrepresentative.
INTRODUCTION

There are two main aims to this project, both of which involve studies of the “Missionaries and Cannibals” (M&C) problem. The first aim is to take an existing implementation of a solution to the problem in the Soar 6 language, and to update the code to Soar 7. As the M&C is a classic AI problem and is relatively simple, the code would be useful as a teaching aid in introducing students to the Soar language. The complexity of the language is a recognised problem in teaching students to program in Soar. The task would be made easier, if students could be made familiar with the language through use of simple example programs with which they are already familiar. Already this is beginning to happen, and a library of Soar implementations of similarly simple AI problems is beginning to be constructed. The 8-puzzle and Tower of Hanoi are other examples of existing Soar programs. However, a major problem with the current version of the code for the M&C problem is that it is almost entirely uncommented. It is very difficult for the novice Soar programmer to get to grips with a new piece of code without the aid of comments. In order to rectify this problem, comprehensive commenting will be added to the code.

The second objective is again aimed at making the Soar code more user-friendly: a Graphical User Interface (GUI) for the M&C problem will be designed and implemented in Tcl/TK. The existing Soar code will be integrated with the Tcl/TK code so that the GUI acts as the front-end to the Soar code. This will enable the user to make use of the Soar model, and to interpret its solutions without the need to extract data from complicated traces and make use of shell commands. The interface will then be used to collect real-user data from a number of subjects to allow the success of the model to be assessed by a comparison between the solution times of the model and real user.

The intention is that the GUI should log a number of different actions and time each move. This data should then be output to a file for later analysing in a spreadsheet. If the interface can be coded so that it can be operated either by a human user, or by the Soar model, and data can be automatically logged in both situations, then fair comparisons can be made.

Aside from making the Soar code more understandable, the main objective then is to evaluate the model in terms of the extent to which it models human techniques. The criteria by which the Soar model will be judged are:

i) Number of moves from beginning to end (including restarted attempts)

ii) Number of moves in the final successful attempt from start state to goal state (the solution path).
iii) Time taken to reach the goal (in both the complete run, which may include failed and restarted attempts, and in the subset of this that forms the successful solution path.

iv) In addition to these statistical comparisons, it may be useful to compare the traces from the Soar model and the human data and to make some observations about the type of moves made.

In order for the Soar model to be declared an accurate representation of human methods for solving the M&C problem, its data must correlate with human data on the majority of these criteria.
IMPLEMENTATION

The original Soar 6 code has been upgraded and commented as intended. The new version of the code is included in Appendix A, and on the enclosed floppy disk. All of comments by this author have been enclosed in square parentheses to distinguish them from the original. In addition, the three monitor operators have been modified so that each time they are applied, they output the current time to the trace screen. This allows execution time data to be collected simply by executing the model.

Explanation of GUI

The GUI has been implemented in Tcl/TK version 8.0a2 (Wish 4) but is designed to be backwardly compatible with version 7. The code for the interface is also included on the floppy disk and in Appendix A where it is line numbered for easy referencing. The GUI half way through a solution is shown in Figure 1a (running on Windows 95) and in Figure 1b (on a PowerMac), below before an explanation of its main features.

The interface shows 3 representations of the problem space the ones at the left and right side display the start state and goal state respectively. All buttons in these two displays are disabled as they are there merely for the benefit of the users to remind them of what they began with and what they are aiming for. The central display is the...
current state and is the only representation that changes during the problem. Obviously, the $M$ buttons represent missionaries, and the $C$'s cannibals. The radio buttons at the bottom of each display indicate on which side of the river (the blue rectangle) the boat resides. These radio buttons update automatically, and cannot be changed by the user. The button at the top-centre of the display labelled Go! is used to send the selected Missionaries and Cannibals across the river. The Go! Button is disabled (shaded over and cannot be pressed) when no Missionaries or Cannibals are selected. When a Missionary or Cannibal is clicked on the same side as the boat, the button sticks in the pressed position (sunken) until either the user clicks Go! or the user changes their mind about which piece to move and clicks the button again, in which case it will pop back out. If the user clicks a button on the opposite side to the boat, the action is ignored as pieces cannot be moved without the boat. A maximum of two pieces can be selected (as the capacity of the boat is two), if any further pieces are clicked, they are ignored – a selected piece must first be deselected. The Go! button can be pressed when one or two pieces are selected on the same side as the boat. This causes the selected pieces to be moved across the river, the Last Action display (bottom right) to be updated, and the Go! button to be disabled (until another piece is selected).

The bottom row of buttons has the following function: Auto issues the run command to the Soar model, to automatically solve the problem. The GUI updates to reflect the current state of the solution. Manual informs the program that the problem will be solved by a human user and writes the current time to the data file (see below). Stop should be pressed by the subject as soon as the goal state is achieved as this writes the time at which the solution was reached to the data file. The subject presses Reset if they reach an impasse (or believe to have reached an impasse), or if they move to an illegal state where the number of cannibals outnumber missionaries. The ? button causes an About window to be displayed showing information about the author etc.
Unfortunately, although the intention was to integrate the GUI with the Soar model, it has not been possible to complete this task in the time. This was necessarily the last part of the project to be attempted, – obviously the interface had to be coded first, before the Soar code could be combined with it. Because integration of the two components was planned, it is possible to explain how it would have been achieved. The only additional programming required would be to rewrite the three Soar monitor operators so that rather than just writing to the screen, each operator calls the GUI procedure goBtn{} and sets the value of the variables $btn1 and $btn2 which store the pathnames of the buttons to be swapped. This data is already processed within each monitor operator as can be seen from the Soar traces (which output the pieces being moved, and where they are being moved to and from), although it is in a slightly different format and would need to be built up into the full packet expected by the btnPres{btn} procedure. Calling of Tcl/TK functions has also been done previously where the three monitor operators have been changed slightly so that they output the current time each time the operator is applied. All of the necessary mechanisms to link the two aspects of the program are in place, and the skills required to achieve this have been demonstrated in other areas of the program. It is estimated that this could be achieved with an extra one or two days work – perhaps less with a more competent programmer! It may have been possible to have added this functionality to the system in the time available, but at the expense of being able to provide any results. Data for the Soar model can still be gathered from the model trace, and separate timing of the solution. It was deemed that the generation of some conclusions to the project was more important than this small aspect of the program.

**Data File**

When the GUI is first started, it opens a file in the current directory called “data.csv” When the file is initialised, a title is written to it, and the time at which the application was launched (the current time) is inserted. Following this, each time a piece is moved across the river, a line is appended to the file, to say what was moved, from and to which side, and the time at which the pieces were moved. If the Reset button is pressed, the attempt must have failed, so a message indicating that the problem has been restarted is added to the file along with the time. When the Manual button is pressed, the message “Begin” is inserted into the file before the time, this allows for the application to be loaded (and the time inserted) and for the task to then be explained to the subject. When the subject is ready to begin, they are instructed to first press Manual, in effect this resets the clock so that the time taken to explain the system is not added to the time taken by the subject to reach the solution. Each time that data is added to the file, it takes the form of a plain text string followed by the time, these two fields are separated by a comma to allow the file to be imported into a spreadsheet as Comma Separated Values (CSV) data. This enables the data to be preformatted into the appropriate columns when imported. A spreadsheet package such as Microsoft’s Excel 97 allows arithmetic operations to be performed directly on the time field as it appears as hh:mm:ss. This allows for statistical analysis of the data by computing the time taken to perform each move, and the cumulative time taken for complete solutions (or the point at which the attempt was restarted).
Limitations

The GUI has been implemented primarily as a mechanism to obtain human data, and as such has not been programmed with the kind of level of validation that would be expected of a completely finished application. However, a high degree of validation has been incorporated (e.g. the user can only move pieces on the same side as the boat, an unmanned boat cannot change sides, only two pieces can cross the river at once etc). Areas that could benefit from further system enforced rules are listed below:

- The user is not forced to press the Manual button, – failure to do this may result in an artificially inflated solution time.

- The system does not automatically detect illegal states (where cannibals outnumber missionaries), the experimenter and subject are relied upon to spot illegal states and hit Restart. However this would be relatively straightforward to implement – it simply involves counting the number of missionaries and cannibals at one side of the river after each press of Go! (the pieces on the other side can be determined if the total is known for just one side).

- The system does not automatically detect duplicate states (see previous point).

- The system does not automatically detect when the goal state is achieved. (See previous–but–one point).
Human data was collected from three subjects (including the author). A short explanation of the M&C problem, and a demonstration of the interface was given by the author to the other two subjects. The data from all three subjects was then imported into Excel, where a number of other statistics were generated from the times given, these comprise:

- Time per move (i.e. how long between the swapping of pieces)
- Cumulative time (i.e. the time starting from zero rather than whatever time the trial began – reset to zero each time the user restarts after moving to an illegal state)
- Move number (i.e. a count tally of how many moves have been made which is reset to zero each time the subject restarts after moving to an illegal state)
- Total moves (the total number of moves made from start to finish (regardless of restarts))
- Total time (the total number of moves made from start to finish (regardless of restarts))
- Solution time (the time taken to go from start to goal in one run – i.e. the time from the final restart to the goal state)

The Excel printout of this data is included at the end of Appendix A. Each of the three subjects, and the Soar model all eventually execute the optimum solution path which takes 11 moves. However, Soar takes considerably more moves to reach this solution as shown in Figure 2, below (note that subject 1 was the author and so may have had an unfair advantage).

![Figure 2: Comparison between the total number of moves to reach the goal](image-url)
The time taken to execute the solution path (i.e. not including restarted attempts) could not be gathered from the Soar model at this stage of the implementation, but Figure 3 shows this data for the three subjects, and may be useful in further study:

![Figure 3: Time taken (in seconds) to execute the solution path.](image)

The total solution time (Figure 4) is perhaps not entirely representative of how long it would normally take to reach the goal from start to finish for the following reasons. The data from the author (34 seconds) is perhaps too low because by the end of testing the model, the author new the M&C problem inside out and so is probably too familiar. Subject 2 may also have an unrepresentative time because the subject seemed to stumble upon the solution – Soar does not really model the effects of chance. The data from Subject 3 probably gives the best comparison, as this subject improved as the trial progressed, and reached the solution deliberately. These effects would be largely overcome with greater numbers of subjects.

![Figure 4: Total time from start to finish.](image)

Overall seems that Soar reaches the same state as the human subjects eventually – that of finding the optimal solution although it reaches this stage far quicker. Also, Soar takes well over twice as many moves to get to this state. Soar moves the pieces of the problem far too quickly. – Its time per move is 0.18 seconds compared with an average for the 3 subjects of 10.86 seconds. An obvious flaw to the comparison is that in the data collected, the Soar model does not make use of the interface. While this in some ways is a valid argument, the problem is that use of the interface may actually decrease the amount of time that it takes for the human subject to solve the problem. – The interface certainly does not slow down the user, as solving the problem by hand, or “in their head” would take even longer. One factor modelled well by Soar is that different subjects take different total number of moves to solve the problem. In the 3 traces included on the disk, Soar took 72, 78 and 59 moves (with chunks excised each time) to solve the problem. However, if chunks are not excised, Soar will remember the perfect solution path after just one run, and so will make just 11 moves on subsequent trials – human subjects fail to match this level of recall after only one trial.
In general, the project has been successful, and with a little extra work to tie together
the Soar and Tcl/Tk code could be complete. Soar itself has not been proved a
completely successful model of human performance for this task at least.
Comparisons similar to those made here but with different subject bases (such as
younger, or from different backgrounds (effect of IQ level?)) would be interesting,
and may provide data that better correlates with the Soar model. Obviously the
subject’s familiarity with windowed operating systems will also have an effect, as this
affects their control of the mouse etc. although all subjects used here were relatively
familiar with the operation of a GUI.

Modification could also be made to enhance the GUI, and mostly takes the form of
adding validation to force the user to do certain things that they are currently “trusted”
to do. The modifications have been discussed previously, in the Limitations of GUI
section.
missionaries.soar

The code for the Soar model of the file is provided below.

### FILE: missionaries.soar

### AUTHOR(2): Aladin.Akyurek [Soar 6.0.7]
### AUTHOR(1): Aladin.Akyurek [Soar 5.1.1]

### MODIFIED(3): [May 21, 99] Why does everyone work on this code in may?! :-(
### -- " -- : Comprehensive comments added.
### : All additions enclosed in square parentheses "[]"
### CREATED(2): May 10, 93
### CREATED(1): May 21, 91

### ABSTRACT. This file provides a Soar system to solve
### the missionaries and cannibals problem. 3 missionaries and
### 3 cannibals come to a river. There is a boat on their side of
### the river that can be used by either 1 or 2 persons at a time.
### Goal is to cross river (using only the boat, but to preserve the
### lives of the missionaries, cannibals must never outnumber missionaries.

package require Soar

### [This tells Soar where the default rules are                    
### [May need to set the variable if not done already in your system]
### [or you're not at Nottingham. Probably tidier to do that in the ]
### [mis-can.tcl file (GUI implementation) if used                ]
### source $default

### TOP GOAL:
### MISSIONARIES-AND-CANNIBALS
### [declare the top goal - top because "superstate nil"

sp {top-goal*elaborate*goal*missionaries-and-cannibals
   (state <g> ^superstate nil)
   -->
   (<g> ^name missionaries-and-cannibals
   ^superstate nil)
   (<g> ^problem-space <p>)
   (<p> ^name missionaries-and-cannibals
   ^default-state-copy yes
   ^one-level-attributes boat + &,
   location + &
   ^two-level-attributes mlist + &,
   clist + &,
   next + &,
   holds + &))

### MISSIONARIES-AND-CANNIBALS PROBLEM SPACE:
### INITIAL-STATE
### [in the initial state, mis/can/boat all on left, so ]
### [tell Soar this... ]
### [NB don't need to do this for goal state, as goal   ]
### [test (later) makes the goal explicit     ]

sp {mac*propose*state*initial-state
   (state <g> ^name missionaries-and-cannibals
   ^superstate nil)
   (<g> ^problem-space <p>)
   (<p> ^name missionaries-and-cannibals)
   -->
   (<g> ^mlist <m*> + &, <x*> + &
   ^clist <c*> + &, <y*> + &
   ^boat <b> ^location <leftbank> + &, <rightbank> + &
   ^holds <h1> + &, <h2> + &, <h3> + &,<h4> + &, <h5> + & )
   (<h1> ^mlist <m*> ^location <leftbank>)
   (<h2> ^clist <c*> ^location <leftbank>)
   (<h3> ^mlist <x*> ^location <rightbank>)
   (<h4> ^clist <y*> ^location <rightbank>)
   (<h5> ^boat <b> ^location <leftbank>)
   )

---

### TOP GOAL PROBLEM SPACE:
### MISSIONARIES-AND-CANNIBALS/MAC
### [describe the conditions of the problem to Soar    ]
### [eg. there's a boat, called boat                ]
### [and it can be at one side or the other - location ]
### [keep track of the missionaries and cannibals in ]
### [variables called mlist and clist                ]

sp {mac*propose*space*mac
   (state <g> ^name missionaries-and-cannibals
   ^superstate nil)
   -->
   (<g> ^problem-space <p>)
   (<p> ^name missionaries-and-cannibals
   ^default-state-copy yes
   ^one-level-attributes boat + &,
   location + &
   ^two-level-attributes mlist + &,
   clist + &,
   next + &,
   holds + &))

---

### TOP GOAL PROBLEM SPACE:
### MISSIONARIES-AND-CANNIBALS/MAC
### [describe the conditions of the problem to Soar    ]
### [eg. there's a boat, called boat                ]
### [and it can be at one side or the other - location ]
### [keep track of the missionaries and cannibals in ]
### [variables called mlist and clist                ]

sp {mac*propose*space*mac
   (state <g> ^name missionaries-and-cannibals
   ^superstate nil)
   -->
   (<g> ^problem-space <p>)
   (<p> ^name missionaries-and-cannibals
   ^default-state-copy yes
   ^one-level-attributes boat + &,
   location + &
   ^two-level-attributes mlist + &,
   clist + &,
   next + &,
   holds + &))

### MISSIONARIES-AND-CANNIBALS PROBLEM SPACE OPERATORS:
### MOVE1, MOVE2, AND MOVE11
### [treat left-right and right-left moves as same, then]
### [only 3 kinds of move are allowed:                  ]
### [move1: just 1 missionary, or just one cannibal     ]
### [move2: move 2 same - 2missionaries *or* 2cannibals ]
### [move11: move one of each -missionary *and* cannibal]
### [...so propose operators for each of these moves    ]

sp {mac*propose*operator*move1
  "Moves either a missionary or a cannibal."
  (state <g> ^problem-space <p> )
  (<p> ^name missionaries-and-cannibals)
  (<g> ^holds <h1> <h2>)
  (<h1> ^<x> <x*> ^location <f>)
  (<h2> ^boat <b> ^location <f>)
  (<x*> ^next <i> <> none)
  (<f> ^opposite-of <t>)    -->
  (<g> ^operator <o>)
  (<o> ^name move1
  ^done? no         ^inst <i>
  ^from <f>         ^to <t>)}

sp {mac*propose*operator*move2
  "Moves either two missionaries or two cannibals."  
  (state <g> ^problem-space <p> )
  (<p> ^name missionaries-and-cannibals)
  (<g> ^holds <h1> <h2>)
  (<h1> ^<xx> <xx*> ^location <f>)
  (<h2> ^boat <b> ^location <f>)
  (<xx*> ^next <i1> <> none)
  (<f> ^opposite-of <t>)    -->
  (<g> ^operator <o>)
  (<o> ^name move2
  ^done? no         ^inst <i1>
  ^with <i2>
  ^from <f>         ^to <b>)}

sp {mac*propose*operator*move11
  "Moves one missionary and one cannibal."
  (state <g> ^problem-space <p> )
  (<p> ^name missionaries-and-cannibals)
  (<g> ^holds <h1> <h2> <h3>)
  (<h1> ^mlist <m*> ^location <f>)
  (<h2> ^clist <c*> ^location <f>)
  (<h3> ^boat <b> ^location <f>)
  (<m*> ^next <i> <> none)    (<c*> ^next <j> <> none)
  (<f> ^opposite-of <t>)    -->
  (<g> ^operator <o>)
  (<o> ^name move11
  ^done? no         ^inst <i>
  ^with <j>
  ^from <f>         ^to <b>)

### MISSIONARIES-AND-CANNIBALS PROBLEM SPACE:
### OPERATOR IMPLEMENTATION
### [now write code to implement each of the 3 moves ]
### [just proposed above...    ]

sp {implement*move1
  (state <gp> ^operator <ox>)
  (state <gp> ^problem-space <p> )
  (<o> ^name move1
  ^done? no         ^inst <i>)}
*from f
*to t
  (g ^name missionaries-and-cannibals)
  (h ^holds h1 h2 h3)
  (h1 ^clist x)
  (h2 ^clist y)
  (h3 ^boat b)
  (x ^next c1)
  (y ^next c1)
  (c1 ^value v)
  (x ^next c2)
  (y ^next c2)
  (f ^opposite-of t)
  -->
  (x ^next c2 c1 - )
  (y ^next c1 c - )
  (h3 ^location c b - )
  (g ^done? yes no - )

sp (implement(move2
  (state g ^operator o)
  (state g ^problem-space p)
  (o ^name move2
   *done? no
   *with c2
   *from f
   *to t)
  (g ^name missionaries-and-cannibals)
  (h ^holds h1 h2 h3)
  (h1 ^clist x)
  (h2 ^clist y)
  (h3 ^boat b)
  (x ^next c1)
  (y ^next c1)
  (c1 ^value v)
  (x ^next c2)
  (y ^next c2)
  (f ^opposite-of t)
  -->
  (x ^next c2 c1 - )
  (y ^next c1 c - )
  (h3 ^location c b - )
  (g ^done? yes no - )

sp (implement(move11
  (state g ^operator o)
  (state g ^problem-space p)
  (o ^name move11
   *inst <i1> *with <i2> *from f *to t)
  (g ^name missionaries-and-cannibals)
  (h ^holds h1 h2 h3 h4 h5)
  (h1 ^mlist m1)
  (h2 ^clist c1)
  (h3 ^clist c2)
  (h4 ^clist c3)
  (h5 ^boat b)
  (x ^next i1)
  (y ^next n)
  (i1 ^value v)
  (i2 ^value v)
  (i3 ^value v)
  (f ^opposite-of t)
  -->
  (x ^next i2 i1 - )
  (y ^next n1 n - )
  (h3 ^location c f - )
  (g ^done? yes no - )

sp (mac'save*operator*applied*a
  (state g ^operator o)
  (state g ^problem-space p)
  (o ^name name)
  (p ^name missionaries-and-cannibals)
  (q ^applied)
  -->
  (g ^applied o q - )

sp (mac'save*operator*applied*b
  (state g ^operator o)
  (state g ^problem-space p)
  (o ^name name)
  (p ^name missionaries-and-cannibals)
  (q ^applied)
  -->
  (g ^applied o q - )
### MISSIONARIES-AND-CANNIBALS PROBLEM SPACE:
### OPERATOR TERMINATION
### [for each move proposed and implemented above, code ]
### [is needed to terminate the move, so when the move ]
### [has been made (*done? yes*) reconsidered it (@)    ]

```prolog
sp {terminate*move1
  (state <g> ^operator <o>)
  (state <g> ^problem-space <p>)
  (^name move1 ^done? yes)
  (^name missionaries-and-cannibals)
  -->
  (^operator <o> @ ) }

sp {terminate*move2
  (state <g> ^operator <o>)
  (state <g> ^problem-space <p>)
  (^name move2 ^done? yes)
  (^name missionaries-and-cannibals)
  -->
  (^operator <o> @ ) }

sp {terminate*move11
  (state <g> ^operator <o>)
  (state <g> ^problem-space <p>)
  (^name move11 ^done? yes)
  (^name missionaries-and-cannibals)
  -->
  (^operator <o> @ ) }
```

### MISSIONARIES-AND-CANNIBALS PROBLEM SPACE:
### STATE EVALUATION
### [Need a way to tell Soar what to do next            ]
### EVALUATION: STATE FAILURE
### [define the 3 cases in which the solution fails...  ]
### [1. if one(mis) VS >one(can) at same location       ]
### [2. two(mis) VS 3(can) at same location             ]
### [NB can *never* be 3(mis) VS >3(can), so only need  ]
### [to make two checks for this kind of failure        ]
### [3. finally, solution failed if it gets back to a   ]
### [state already tried                                ]

```prolog
sp {mac*evaluate*state*failure*one-against-more
  (state <g> ^desired <d>)
  (state <g> ^problem-space <p>)
  (^name missionaries-and-cannibals)
  (^mlist <m*> ^location <loc>)
  (^clist <c*> ^location <loc>)
  (^value missionary ^next none)
  (^value cannibal ^next <j2>)
  (^value cannibal ^next <j3>)
  -->
  (^failure <d>)}

sp {mac*evaluate*state*failure*two-against-more
  (state <g> ^desired <d>)
  (state <g> ^problem-space <p>)
  (^name missionaries-and-cannibals)
  (^mlist <m*> ^location <loc>)
  (^clist <c*> ^location <loc>)
  (^value missionary ^next <i2>)
  (^value missionary ^next <i1>)
  (^value cannibal ^next <j2>)
  (^value cannibal ^next <j3>)
  -->
  (^failure <d>)}

sp {mac*evaluate*state*failure*detect-state-duplicate
  (state <g2> ^problem-space <p2>)
  (^superstate <g1>)
  (state <g> ^desired <d>)
  (state <g> ^problem-space <p>)
  (^name missionaries-and-cannibals)
  (^mlist <m*> ^location <loc>)
  -->
  (^superstate <g>)
  (^name <name>)
  (^value missionaries-and-cannibals)
  (^next <j1>)
  (^location <loc>)
  (^failure <d>)
```

### [2. two(mis) VS 3(can) at same location         ]
### [NB can *never* be 3(mis) VS >3(can), so only need  ]
### [to make two checks for this kind of failure        ]
### [3. finally, solution failed if it gets back to a   ]
### [state already tried                                ]

```prolog
sp {mac*evaluate*state*failure*one-against-more
  (state <g> ^desired <d>)
  (state <g> ^problem-space <p>)
  (^name missionaries-and-cannibals)
  (^mlist <m*> ^location <loc>)
  (^clist <c*> ^location <loc>)
  (^value missionary ^next none)
  (^value cannibal ^next none)
  -->
  (^failure <d>)}

sp {mac*evaluate*state*failure*two-against-more
  (state <g> ^desired <d>)
  (state <g> ^problem-space <p>)
  (^name missionaries-and-cannibals)
  (^mlist <m*> ^location <loc>)
  (^clist <c*> ^location <loc>)
  (^value missionary ^next <i2>)
  (^value missionary ^next none)
  (^value cannibal ^next <j2>)
  (^value cannibal ^next <j3>)
  -->
  (^failure <d>)}

sp {mac*evaluate*state*failure*detect-state-duplicate
  (state <g2> ^problem-space <p2>)
  (^superstate <g1>)
  (state <g> ^desired <d>)
  (state <g> ^problem-space <p>)
  (^name missionaries-and-cannibals)
  (^mlist <m*> ^location <loc>)
  -->
  (^superstate <g>)
  (^name <name>)
  (^value missionaries-and-cannibals)
  (^next <j1>)
  (^location <loc>)
  (^failure <d>)
```

### [3. finally, solution failed if it gets back to a   ]
### [state already tried                                ]

```prolog
sp {mac*evaluate*state*failure*one-against-more
  (state <g> ^desired <d>)
  (state <g> ^problem-space <p>)
  (^name missionaries-and-cannibals)
  (^mlist <m*> ^location <loc>)
  -->
  (^failure <d>)}

sp {mac*evaluate*state*failure*two-against-more
  (state <g> ^desired <d>)
  (state <g> ^problem-space <p>)
  (^name missionaries-and-cannibals)
  (^mlist <m*> ^location <loc>)
  -->
  (^failure <d>)}

sp {mac*evaluate*state*failure*detect-state-duplicate
  (state <g2> ^problem-space <p2>)
  (^superstate <g1>)
  (state <g> ^desired <d>)
  (state <g> ^problem-space <p>)
  (^name missionaries-and-cannibals)
  (^mlist <m*> ^location <loc>)
  -->
  (^superstate <g>)
  (^name <name>)
  (^value missionaries-and-cannibals)
  (^next <j1>)
  (^location <loc>)
  (^failure <d>)
```

### [1. if one(mis) VS >one(can) at same location       ]
### [2. two(mis) VS 3(can) at same location             ]
### [NB can *never* be 3(mis) VS >3(can), so only need  ]
### [to make two checks for this kind of failure        ]
### [3. finally, solution failed if it gets back to a   ]
### [state already tried                                ]
### MISSIONARIES-AND-CANNIBALS PROBLEM SPACE:
### SEARCH CONTROL

## Create "reject preference" for inverse operator.
## Definition of Inverse Operator:
## apply(a[i],s[i]) = s[j],
## apply(a[j],s[j]) = s[i],
## where a[i] and a[j] are different instantiations
## of the operator A, and s stands for state. The
## instantiation a[i] is the inverse of a[j].
## Example: a[i] = walk(x,y), a[j] = walk(y,x).

sp {mac*operator*inverse*move1*reject
  "Reject inverse operator."
  (state <g> ^operator <o> + )
  (state <g> ^problem-space <p> )
  (<o> ^name move1 ^inst <i> ^from <loc> ^to <oloc> )
  (<p> ^name missionaries-and-cannibals)
  (<o> ^applied <q>)
  (<q> ^name move1 ^inst <j> ^from <oloc> ^to <loc> )
  (<i> ^value <value>)
  (<j> ^value <value>)
  -->
  ( <g> ^operator <o> - )}

sp {mac*operator*inverse*move2*reject
  "Reject inverse operator."
  (state <g> ^operator <o> + )
  (state <g> ^problem-space <p> )
  (<o> ^name move2 ^inst <i1> ^with <i2> ^from <loc> ^to <oloc> )
  (<p> ^name missionaries-and-cannibals)
  (<o> ^applied <q>)
  (<q> ^name move2 ^inst <j1> ^with <j2> ^from <oloc> ^to <loc> )
  (<i1> ^value <value>)
  (<i2> ^value <value>)
  (<j1> ^value <value>)
  (<j2> ^value <value>)
  -->
  ( <g> ^operator <o> - )}

sp {mac*operator*inverse*move11*reject
  "Reject inverse operator."
  (state <g> ^operator <o> + )
  (state <g> ^problem-space <p> )
  (state <g> ^operator <o> - )
  (state <g> ^problem-space <p> )

### MISSIONARIES-AND-CANNIBALS PROBLEM SPACE:
### SEARCH CONTROL

## Create "reject preference" for inverse operator.
## Definition of Inverse Operator:
## apply(a[i],s[i]) = s[j],
## apply(a[j],s[j]) = s[i],
## where a[i] and a[j] are different instantiations
## of the operator A, and s stands for state. The
## instantiation a[i] is the inverse of a[j].
## Example: a[i] = walk(x,y), a[j] = walk(y,x).

sp {mac*operator*inverse*move1*reject
  "Reject inverse operator."
  (state <g> ^operator <o> + )
  (state <g> ^problem-space <p> )
  (<o> ^name move1 ^inst <i> ^from <loc> ^to <oloc> )
  (<p> ^name missionaries-and-cannibals)
  (<o> ^applied <q>)
  (<q> ^name move1 ^inst <j> ^from <oloc> ^to <loc> )
  (<i> ^value <value>)
  (<j> ^value <value>)
  -->
  ( <g> ^operator <o> - )}

sp {mac*operator*inverse*move2*reject
  "Reject inverse operator."
  (state <g> ^operator <o> + )
  (state <g> ^problem-space <p> )
  (<o> ^name move2 ^inst <i1> ^with <i2> ^from <loc> ^to <oloc> )
  (<p> ^name missionaries-and-cannibals)
  (<o> ^applied <q>)
  (<q> ^name move2 ^inst <j1> ^with <j2> ^from <oloc> ^to <loc> )
  (<i1> ^value <value>)
  (<i2> ^value <value>)
  (<j1> ^value <value>)
  (<j2> ^value <value>)
  -->
  ( <g> ^operator <o> - )}

sp {mac*operator*inverse*move11*reject
  "Reject inverse operator."
  (state <g> ^operator <o> + )
  (state <g> ^operator <o> - )
  (state <g> ^problem-space <p> )
### MISSIONARIES-AND-CANNIBALS PROBLEM SPACE:

### MONITOR OPERATORS

### [Next 3 just print to stdout a representation of the]

### [3 moves - if GUI used, the Tcl script redefines them]

### [to inform interface of move, rather than screen ]

```tcl
sp {mac*[monitor*operator-application*move1
(state <g> ^operator <o>)
(<> ^name move1
  ^inst <>
  ^from <>
  ^to <>)
(<> ^name <fname>)
(<> ^name <tname>)

(.APPLIED|<o>:move1(|<v>           |
,|<fname>           |
,|<tname>           |
at,[clock format [clock seconds] -format %X]|)}}
```

```tcl
sp {mac*[monitor*operator-application*move2
(state <g> ^operator <o>)
(<> ^name move2
  ^inst <>
  ^with <>
  ^from <>
  ^to <>)
(<> ^name <fname>)
(<> ^name <tname>)

(.APPLIED|<o>:move2(|<v>           |
,|<fname>           |
,|<tname>           |
at,[clock format [clock seconds] -format %X]|)}}
```

```tcl
sp {mac*[monitor*operator-application*move11
(state <g> ^operator <o>)
(<> ^name move11
  ^inst <>
  ^with <>
  ^from <>
  ^to <>)
(<> ^name <fname>)
(<> ^name <tname>)

(.APPLIED|<o>:move11(|<ival>           |
,|<jval>           |
,|<fname>           |
,|<tname>           |
at,[clock format [clock seconds] -format %X]|)}}
```

### End-Of-File of missionaries.soar

**mis–can.tcl**

The Tcl/TK code for the interface:

```
# Missionaries and Cannibals interface
#
#------------------------------------------

proc makeGUI {} {
```
global curmis
global curcan
global time
global boatVar
global num
global goStatus
global w

toplevel $w
wm title $w "Missionaries and Cannibals"
wgeometry $w $255x300
wm iconname $w "M&C"

## Press this button to cross the river with the miss/can combo selected
button $w.goBtn -text "Go!" -width 20 -command "insertTime
  go;updateTime;goBtn" -state disabled
pack $w.goBtn -side top -pady 5

## Setup 3 diff representations as follows:
## ss = start state
## cs = current state
## gs = goal state
set s $w.stateFrame
makeState "Start state" $s.ssFrame disabled {1 1 1} {1 1 1}
$s.ssFrame.boatFr.boatLeft configure -state disabled -disabledforeground Black
$s.ssFrame.boatFr.boatRight configure -state disabled
$s.ssFrame.boatFr.boatLeft select

set curmis {1 1 1}
set curcan {1 1 1}

set c $w.stateFrame
class configure -state disabled -disabledforeground Black
$c.boatFr.boatLeft configure -state disabled -disabledforeground Black
$c.boatFr.boatRight configure -state disabled -disabledforeground Black
set boatVar L

## create missionaries
for {set j 0} {set j < 3} {incr j} {
  set xpos [expr $j*.33]
  set ypos [expr 0.0]
  set num [lindex $mis $j]
  if {num == 1} {
    set d $c.left.mis.$j$text$num
    button $d -relief raised -text $text -state $state
    -disabledforeground black -command "btnPres $d"
  } else {
    set d $c.right.mis.$j$text$num
  }
  place $d -relx $xpos -rely $ypos -relwidth .33 -relheight 1.0
}

set b $w.btnFrame
frame $b -borderwidth 2 -relief groove

## create frames etc
frame $s.ssFrame -borderwidth 2 -relief groove
frame $s.ssFrame.boatFr -borderwidth 2 -relief flat
frame $s.csFrame.boatFr -borderwidth 2 -relief flat
frame $s.gsFrame.boatFr -borderwidth 2 -relief flat
frame $s.ssFrame.boatFr.boatLeft
frame $s.ssFrame.boatFr.boatRight
frame $s.csFrame.boatFr.boatLeft
frame $s.csFrame.boatFr.boatRight
frame $s.gsFrame.boatFr.boatLeft
frame $s.gsFrame.boatFr.boatRight

pack $b -side top -padx 20
## create cannibals
set text "C"
for {set j 0}  {$j < 3} {incr j} {
    set xpos [expr $j*.33]
    set ypos [expr 0.0]
    set num [lindex $can $j]
    if {$num == 1} {
        set d $name.left.can.$j$text$num
    } else {
        set d $name.right.can.$j$text$num
    }
    button $d -relief raised -text $text -state $state -disabledforeground black -command "btnPres $d"
    place $d -relx $xpos -rely $ypos -relwidth .33 -relheight 1.0
}
frame $name.boatFr
radiobutton $name.boatFr.boatLeft -text "   Boat   " -variable boat$name -value L
radiobutton $name.boatFr.boatRight -text " " -variable boat$name -value R
pack $name.boatFr.boatLeft -side left
pack $name.boatFr.boatRight -side left
pack $name.boatFr -side bottom

## finally add everything to the gui
pack $name.msg -side top
pack $name.left.mis $name.left.can -side top
pack $name.right.mis $name.right.can -side top
pack $name.left $name.river $name.right -side left -pady 5
pack $name -side left -padx 2
}

proc goBtn {} {
    global goStatus
global qtyPres
global w
    if {$qtyPres > 0} {
        if {$goStatus == 1} {
            set goStatus 0
        } else {
            set goStatus 1
        }
        btnPres $w.goBtn
    }
}

proc about {} {
    global w
    set m "GUI for the Missionaries & Cannibals AI problem\n        (c)1999 T. A. D'Roza\n    All rights reserved\n    (The source code may be freely distributed providing\n    that the information in this window remains intact)"
    tk_messageBox -type ok -icon question -title About... -parent $w -message $m
}

proc setCS {newmis newcan} {
    global curmis
global curcan
global w
    ##first destroy the existing buttons (have to do this because I can't move them to another frame
    for {set i 0} {$i < 3} {incr i} {
        ##first get rid of missionaries
        set num [lindex $curmis $i]
        set type "M"
        if {$num == 1} {
            set d $w.stateFr.csFrame.left.mis.$i$type$num
        } else {
            set d $w.stateFr.csFrame.right.mis.$i$type$num
        }
        destroy $d
        ##now do same for cannibals
        set type "C"
        set num [lindex $curcan $i]
        if {$num == 1} {
            set d $w.stateFr.csFrame.left.can.$i$type$num
        } else {
            set d $w.stateFr.csFrame.right.can.$i$type$num
        }
        destroy $d
    }
    ##now recreate the buttons in the new positions
    for (set i 0) {$i < 3} {incr i} {
        ##first get rid of missionaries
        set num [lindex $newmis $i]
        set type "M"
        if {$num == 1} {
            set d $w.stateFr.csFrame.left.mis.$i$type$num
        } else {
            set d $w.stateFr.csFrame.right.mis.$i$type$num
        }
        destroy $d
    }
    ##now do same for cannibals
    set type "C"
    set num [lindex $newcan $i]
    if {$num == 1} {
        set d $w.stateFr.csFrame.left.can.$i$type$num
    } else {
        set d $w.stateFr.csFrame.right.can.$i$type$num
    }
    destroy $d
}
set xpos [expr $i * .33]

## first make missionaries
set num [lindex $newmis $i]
set type "M"
if {$num == 1} {
    set d $w.stateFr.csFrame.left.mis.$i$type$num
} else {
    set d $w.stateFr.csFrame.right.mis.$i$type$num
} button $d -relief raised -text $type -command "btnPres $d"
place $d -relx $xpos -rely 0.0 -relwidth .33 -relheight 1.0

## now do same for cannibals
set type "C"
set num [lindex $newcan $i]
if {$num == 1} {
    set d $w.stateFr.csFrame.left.can.$i$type$num
} else {
    set d $w.stateFr.csFrame.right.can.$i$type$num
} button $d -relief raised -text $type -command "btnPres $d"
place $d -relx $xpos -rely 0.0 -relwidth .33 -relheight 1.0

set curmis $newmis
set curcan $newcan

proc insertTime {text} {
    global chID
    global time
global goStatus
    global btn1
    global btn2
    if {$goStatus} {
        $w.goBtn configure -relief sunken
    } else {
        set type1 "M:"
        append text $type1
    }

    if {$text == "go"} {
        set text {}
    } else {
        set type2 "$type1"
    } append text $type2

    set time [clock format [clock seconds] -format %X]
    puts $chID "$text,$time"
}

proc updateTime {} {
    global w
    set t [clock format [clock seconds] -format %X]
    $w.btnFrame.timeMsg configure -text "Last action at, $t"
}

proc btnPres {btn} {
    global chID
global qtyPres
    global goStatus
    global btn1
    global btn2
    global boatVar
    if {$goStatus} {
        $w.goBtn configure -relief sunken
    } else {
        if { [string first "C" $btn1] != -1 & & $btn2 != {}} {
            set type2 "C:"
        } else {
            set type1 "M:"
        } append text $type2

        if { [string first "left" $btn1] != -1} {
            append text "left:right"
        } else {
            append text "right:left"
        }
    }
}

set curmis $newmis
set curcan $newcan

proc insertTime {text} {
    global chID
global time
    global boatVar
    global goStatus
    global qtyPres
    global btn1
    global btn2
    if {$goStatus} {
        $w.goBtn configure -relief sunken
    } else {
        if { [string first "C" $btn1] != -1 & & $btn2 != {}} {
            set type1 "C:"
        } else {
            set type2 "$type1"
        } append text $type2

        if { [string first "left" $btn1] != -1} {
            append text "left:right"
        } else {
            append text "right:left"
        }
    }
}

proc updateTime {} {
    global w
    set t [clock format [clock seconds] -format %X]
    $w.btnFrame.timeMsg configure -text "Last action at, $t"
}

proc btnPres {btn} {
    global chID
global qtyPres
    global goStatus
    global btn1
    global btn2
    global boatVar
    global goStatus
    global w
    if {$goStatus} {
        $w.goBtn configure -relief sunken
    } else {
        if { [string first "C" $btn1] != -1 & & $btn2 != {}} {
            set type1 "C:"
        } else {
            set type2 "$type1"
        } append text $type2

        if { [string first "left" $btn1] != -1} {
            append text "left:right"
        } else {
            append text "right:left"
        }
    }
}
```tcl
$w.goBtn configure -relief raised
}

## if the button pressed was not Go! then see which side of the river it's on
if {$btn != "$w.goBtn"} {
    if {[string first "left" $btn] != -1} {
        set side L
    } else {
        set side R
    }
}

## if the button pressed is not Go!, inc the pressed button count
if {$qtyPres < 2 && $btn != "$w.goBtn" && [$btn cget -relief] == "raised" && $side == $boatVar} {
    $btn configure -relief sunken
    if {$btn != "$w.goBtn"} {
        incr qtyPres
    } else {
        ## if a button (not Go!) has been deselected...
        ## has to be in an ELSE so that the button is not sunken then
        ## immediately raised
        if {$btn != "$w.goBtn" && [$btn cget -relief] == "sunken" && $side == $boatVar} {
            ## then raise the button, dec the count
            $btn configure -relief raised
            incr qtyPres -1
        }
    }
}

## if the button has just been raised, then erase the copy
if {[string first "$w.goBtn" $btn]} {
    if {$btn1 == $btn} {
        set btn1 $btn2
        set btn2 {}
    } else {
        set btn2 {}
    }
}

## if at least one button is depressed, and Go! is selected, then make the
## button cross the river
if {$qtyPres >= 1 && $goStatus} {
    if {[string first "left" "$w.goBtn1"] != -1} {
        set changeFr 1
        set changeTo 0
        set remove 0
        set newmis $curmis
        set newcan $curcan
    } else {
        set changeFr 0
        set changeTo 1
        set remove 0
        set newmis $curmis
        set newcan $curcan
    }
}
```

puts $chID "Timer data"
puts $chID "=========
puts $chID "Action, Total Time"
insertTime "GUI Loaded"

set goStatus 0
set btn1 {}
set btn2 {}
set chID ""
set qtyPres 0
initFile
set w .ms
makeGUI

Data file

An copy of the data file, from a short run of the interface by the author is included below.

Timer data

Action, Total Time
GUI Loaded, 14:20:05
Begin, 14:20:06
M:C:left:right, 14:20:06
M:right:left, 14:20:06
C:C:left:right, 14:20:06
C:right:left, 14:20:07
M:M:left:right, 14:20:07
C:right:left, 14:20:07
C:C:left:right, 14:20:07
C:right:left, 14:20:07
Goal achieved, 14:20:07

The following page contains a printout of the Excel spreadsheet created from the data from each of the three subjects. The spreadsheet created from the Soar trace is included in the three pages following that.
### APPENDIX B

The contents of the enclosed floppy disk are as follows:

<table>
<thead>
<tr>
<th>Filename</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>mc-mac.tcl</td>
<td>Source code for the GUI (Mac version)</td>
</tr>
<tr>
<td>mc-PC.tcl</td>
<td>Source code for the GUI (PC version)</td>
</tr>
<tr>
<td>data.csv</td>
<td>Example output file from the GUI</td>
</tr>
<tr>
<td>mc.soar</td>
<td>Source code of the Soar model</td>
</tr>
<tr>
<td>mis–trace.1</td>
<td>Trace of the Soar model (72 moves)</td>
</tr>
<tr>
<td>mis–trace.2</td>
<td>Trace of the Soar model (78 moves)</td>
</tr>
<tr>
<td>mis–trace.3</td>
<td>Trace of the Soar model (59 moves)</td>
</tr>
<tr>
<td>Report.doc</td>
<td>This file in Word 97 format</td>
</tr>
<tr>
<td>data.xls</td>
<td>The Excel spreadsheet in Excel 97 format</td>
</tr>
<tr>
<td>Soar.xls</td>
<td>Spreadsheet of a Soar trace (78 moves) in Excel 97 format</td>
</tr>
</tbody>
</table>