ヒューマンインタフェース'99 ＃3

久野　靖*

1999.4.23

はじめに

□ 今回はいよいよ GUI を扱います。ただしいきなり C などのプログラミング言語で書くと結構大変なので、その代りに tcl/tk と呼ばれるスクリプト言語を用いてとにかく動く GUI プログラムを作って見ましょう。

□ 続いて、GUI やマウス、ビットマップディスプレイなどを含めて取り扱える理論的な枠組みである Card らによる認知情報処理アプローチについて説明します。

□ 最後に、Card らのモデルがどれくらい合っているか、実験に基づいて確かめて見ようと思います。

1　tcl/tk 入門

1.1　tcl/tk とは?

□ tcl/tk とは… John Ousterhout が設計/開発した、スクリプト言語 (tcl) とその上で動く GUI ツールキット (tk)

□ C++などで延々と「このボタンはここ、押された時の動作はこう…」などと書き連ねて行くよりさくっと短く GUI が作れる

□ 速度は速くないが、GUI ではあまり問題にならない

1.2　ボタンとラベル

□ 例題その 1

```
#!/usr/local/bin/wish
button .b1 -text "Button 1" -command "exit"
pack .b1
```

□ 説明:

- 1 行目:　tcl/tk インタプリタを使って実行するという指定
- 2 行目:　ボタンを作る。ラベルと押された時の動作を指定 (ボタン等の部品はすべて「.」で始まる名前を持つ。
- 3 行目:　ボタンを画面に配置

□ 動かし方:

- 上記の内容を (たとえば)button1 というファイルに作る
- chmod +x button1
- button1 で実行

□ 例題その 2

```
#!/usr/local/bin/wish
label .l0 -text "ボタン押して!"
button .b1 -text "丸                                時
計" -command "exec oclock &"
button .b2 -text "計                                算
機" -command "exec xcalc &"
button .b3 -text "終了" -command "exit"
pack .l0 .b1 .b2 .b3 -fill x
```

□ (演習) 上の 2 つの例題を動かせ。

1.3　レイアウトマネージャ

□ レイアウトマネージャ:　部品を窓内に配置する (その結果見えるようになる) 機能。

- その際、部品の大きさや窓の大きさを調整する (場合がある)

□ pack:　部品を (上から順、右から順など) 順番に詰めるタイプのレイアウトマネージャ

□ grid:　部品を碁盤の目状に縦横に揃えるタイプのレイアウトマネージャ

□ place:　(x,y) 座標、幅、高さ

```
#!/usr/local/bin/wish
. config -width 300 -height 200
button .b1 -text "Button 1" -command "exit"
place .b1 -x 20 -y 60 -width 80 -height 40
```

*筑波大学大学院経営システム科学専攻

- 「.」というのは「wish のウィンドウそのもの」のこと。

□ (演習)place を使って 3 つのボタンを「コ」の字に配置せよ。

1.4 チェックボタン

□ チェックボタン: on/off のいずれかの状態に設定できるボタン

- on/off の状態を保持する「変数の名前を」指定しておく

```
#!/usr/local/bin/wish
. config -width 200 -height 150
checkbutton .c0 -text "透明" -var trans
place .c0 -x 10 -y 50
button .b1 -text "時計" -command {
  if { $trans } {
    exec oclock -transparent &
  } else {
    exec oclock &
  }
}
place .b1 -x 10 -y 100
button .b2 -text "終了" -command "exit"
place .b2 -x 100 -y 100
```

□ if 文: 2 方向への枝分かれ

- $名前: 変数の値を参照する (設定は「set 変数名 値」)

□ { ... } : ブロック

- 書き方に注意。中かっこ閉じが行末だとそこで文が終わってしまうのでまずいことがある。

1.5 ラジオボタン

□ ラジオボタン: 一群のボタンのうち、どれか 1 つだけが on になるボタン

- 値を入れる変数名が共通のボタンがグループになる
- どのボタンが on のときどの値になるかをボタン毎に指定する

```
#!/usr/local/bin/wish
. config -width 200 -height 150
radiobutton .r1 -text "
緑" -value green -var color
radiobutton .r2 -text "
赤" -value red   -var color
radiobutton .r3 -text "
青" -value blue  -var color
place .r1 -x 10 -y 40
place .r2 -x 70 -y 40
place .r3 -x 130 -y 40
button .b1 -text "時計" -command {
  exec oclock -bg $color &
}
place .b1 -x 10 -y 100
button .b2 -text "終了" -command "exit"
place .b2 -x 100 -y 100
```

1.6 スライダ

□ スライダ: 一定範囲の数値を設定

- 「現在の値」を入れる変数を指定できる

```
#!/usr/local/bin/wish
. config -width 200 -height 150
scale .s1 -label "大きさ" \
  -orient h -from 80 -to 200 -var size
place .s1 -x 10 -y 10
button .b1 -text "時計" -command {
  exec oclock -geom ${size}x${size} &
}
place .b1 -x 10 -y 100
button .b2 -text "終了" -command "exit"
place .b2 -x 100 -y 100
```

- 「${ 変数名 }」という書き方→他の英数字が隣接してるときに変数名の範囲を区切るのに使う

1.7 入力欄

□ 入力欄: テキストを打ち込む場所

```
#!/usr/local/bin/wish
. config -width 200 -height 150
entry .e0 -textvar ent
place .e0 -x 10 -y 40
bind .e0 <Key-Return> {
  exec oclock -geom ${ent}x${ent} &
  set ent ""
}
button .b2 -text "終了" -command "exit"
place .b2 -x 100 -y 100
```

□ bind: 部品上で「イベント」(入力操作等) が起きた時に実行する動作を指定できる。

1.8 メニューとメニューボタン

□ メニューボタン: メニューを出すためのボタン

□ メニュー: 内部に任意個数の command が入れられる

```
#!/usr/local/bin/wish
. config -width 200 -height 150
set color red
menubutton .b0 -textvar color -menu .b0.m0 -ind true \
  -relief raised
place .b0 -x 10 -y 40
```

```
  menu .b0.m0 -tearoff false
  .b0.m0 add command -label "
緑" -command "set color green"
  .b0.m0 add command -label "
赤" -command "set color red"
  .b0.m0 add command -label "
青" -command "set color blue"
  button .b1 -text "時計" -command {
    exec oclock -bg $color  &
  }
  place .b1 -x 10 -y 100
  button .b2 -text "終了" -command "exit"
  place .b2 -x 100 -y 100
```

1.9 演習問題

□ 「ランダムな時間待ってから、画面上に 0〜9 の数字が表示される」ものとする。その数字と同じ値をすばやく入力できるようなユーザインタフェースを設計し、スケッチせよ。

□ 上でスケッチした設計を tcl/tk で実現せよ。

1.10 計測実験

□ 上の実装に「ランダムに待って数字表示」と「入力完了時に所要時間を測って表示」という機能を追加する。入力欄を使った例:

```
#!/usr/local/bin/wish
. config -width 200 -height 150
set c1 {
  set time [clock clicks]
  set num [expr $time % 10]
  after 3000 $c2
}
set c2 {
  set num ""
  set delay [expr 3000 + ([clock clicks] % 5017)]
  after $delay $c1
}
after 100 $c2
label .l0 -textvar num -font "*-times-*-24-*" \
  -fg red -relief sunken
place .l0 -x 80 -y 10 -width 40 -height 40
entry .e0 -textvar ent
place .e0 -x 80 -y 60 -width 40
bind .e0 <Key-Return> {
  set now [clock clicks]
  puts "$ent [expr $now - $time]"
  set ent ""
}
button .b2 -text "終了" -command "exit"
place .b2 -x 100 -y 100
```

- 「after ミリ秒 動作」→指定時間後に動作を実行
- 「clock clicks」→実行開始時からの時間をマイクロ秒単位で返す
- 「expr 数式」→式の値を計算
- [...] → ... の実行結果 (出力) をその場所に埋め込む

□ (演習) 先の自分の作品に上と同様の機能を追加して所要時間を計測せよ。

2 認知情報処理アプローチ

2.1 認知情報処理アプローチとは

□ 人間を一種の情報処理系としてモデル化し、その performance を理解

□ 心理学者だけでなく、システム設計に関わるデザイナが活用できる (人間の performance を予測する/設計を評価するのに使える) ことをめざす

□ たとえば「画面に表示された文字と同じキーを打つ」人間内部の処理構造を次のように分解して考える。

- (入力) →「知覚系」→「認知系」→「運動系」→ (出力)

2.2 Model Human Processor

□ 人間内部の処理系: 知覚系、認知系、運動系の 3 つがある。

□ 各系の下位構造: プロセサとメモリ

- 知覚系→知覚プロセサ+視覚メモリ (VIS)+聴覚メモリ (AIS)
- 認知系→認知プロセサ+作業記憶 (WM ないし STM)+長期記憶 (LTM)

2.3 プロセサの処理時間

□ サイクルタイムが単位で、何サイクル、という形でモデル化

- τ p(知覚): 100[50-200]msec
- τ c(認知): 70[25-170]msec
- τ m(運動): 70[30-100]msec

2.4 メモリの特性、容量、減衰時間

□ 視覚メモリ: 物理的

- δ vis = 200[70-1000]msec
- μ vis = 17[7-17] 文字

□ 聴覚メモリ: 物理的

- δ ais = 1500[900-3500]msec
- μ ais = 5[4.4-6.2] 文字

□ 作業記憶: 音響的/視覚的、chunk(かたまり) 単位で保持可能

- δ wm (1chunk) = 73[73-226]sec
- δ wm (3chunk) = 7[5-34]sec
- μ wm = 3[2.5-4.1]chunks
- μ wm* = 7[5-9]chunks ←LTMの助けによる実効値

□ 長期記憶: 意味的

- δ ltm = ∞
- μ ltm = ∞

2.5 知覚システム (特に視覚)

□ 視覚では焦点にない情報はとりこめない

- 眼球運動: 飛躍+停留の反復 (およそ 230msec サイクル)
- つまり「見てない場所」に情報が現われたら取り込むのに 230[70-700]msec 必要
- 速読法→文字単位でなぞらず、なるべく大きな「かたまり」でなぞる

□ 知覚処理は刺激が累積されて閾値を超えると起こる→強い刺激ほど速く処理できる、暗いところでは時間が掛かる

2.6 認知システム

□ サイクルタイムは作業内容によって大幅に変化

- たとえば数字照合→ 33msec、3 次元図形→ 94msec など
- 努力や練習によって速くなる

□ WM はレジスタとして処理中の情報を保持

- リハーサルを行えば内容を長時間維持可能

□ LTM はいわゆる「記憶」

- 何が貯蔵されるかは情報の符合化操作により定まり、貯蔵された情報は検索手がかりを含む (符合化特異性原理)
- 検索の困難度は検索手がかりに関連づけられた項目の多さによる (弁別原理)
- LTM は書き込みに時間が掛かる

2.7 運動システム

□ 認知システムの「決定」を運動に変換する

- 実際の運動はフィードバックしつつ修正しながら行われる
- 最初の反応 (最少単位の時間) がτ m

2.8 ユーザパフォーマンスの予測

□ 単純反応: τ p + τ c + τ m

□ Fitts の法則: 距離 D にあるサイズ S の目標へのポジショニング時間

- Tpos = Im log2(D/S + 0.5)
- ただし Im = -(τ p+τ c+τ m)/log2 ε = 100[50-120]msec/bits

□ 比較照合: 物理照合 (τ c) →名称照合 (2 τ c) →カテゴリ照合 (3 τ c) と進むにつれ、そのレベルまで変換するためのτ c が追加される

□ 選択反応: 選択数 n の log に比例する時間が掛かる

- Tsel = Ic H
- ただし Ic = 150[0-157]msec/bits、H = log2(n+1)

□ 練習効果: タイプライタの例で 1000msec(初心者) → 60msec(プロ)

- 練習のべき法則: Tn = T1 n**(-α)
- ただしα = 0.4[0.2-0.6]

2.9 演習問題

□ MHP のモデルや Fitzz の法則を追試する実験を設計し、tcl/tk で実装せよ。

□ 何か役に立つユーザインタフェースを tcl/tk で実装し、その操作に要する時間を MHP のモデルで予測してみよ。また実際に計測を行ってその予測がどれくらい正しかったか分析せよ。

3　レポート課題 (その 1)

□ 授業中にできなかった実験ものの演習問題の中で面白いと思うものを選び (または自分独自に実験を考案してもよい)、実験を行ってその内容と結果をまとめなさい。

□ (ゴールデンウィークで 1 週休みなので課題を用意しました。最初の予告通り、授業に出ていれば最後の感想レポート程度で単位を差し上げますから、この課題は強制ではありません。欠席がある人は課題の内容に応じて補填するということです)