11.0 User Programming Tricks

- Important concepts
- Useful overall strategies
- When to use various user program segments
User Program Files

- A user program file contains **one or more** program segments (functions).
- Each user program file is associated with **one and only one** potential array.
- Each potential array can be associated with **one and only one** user program file.
- User program files **share the name** of their array (e.g. `einzeln.pa0 <> einzeln.prg`).
Legal Program Segments

Only one segment of each type can be in a user program file

- **Define_Data**
  - Defines Global variables and arrays

- **Init_P_Values**
  - Pre-sets fast adjust potentials (new)

- **Initialize**
  - Initializes ion’s starting parameters

- **Tstep_Adjust**
  - Controls time step size used

- **Fast_Adjust**
  - Fast adjusts potentials as ions fly

- **Efield_Adjust**
  - Adjusts electrostatic fields

- **Mfield_Adjust**
  - Adjusts magnetic fields

- **Accel_Adjust**
  - Adjusts ion accelerations

- **Other_Actions**
  - Post tstep control of ion’s parameters

- **Terminate**
  - Post flight analysis and rerun control
Program Segment Access Rules

- Each program segment is **ONLY** accessed at specified points in the trajectory computation cycle.

- The ion **MUST** be in an instance that **references** a potential array that has a user program file **containing** the program segment (**e.g. Initialize**). **The ONLY exception is the Init P Values segment.**
Program Segments and Calculations

Block Diagram of Trajectory Calculations and Where User Program Segments are Called

Program Segments
- Initialize
- Init_P_Values
- Tstep_Adjust
- Fast_Adjust
- Efield_Adjust
- Mfield_Adjust
- Accel_Adjust
- Other_Actions
- Terminate

The Idaho National Engineering and Environmental Laboratory
Segment Access Paradigm

- Reserved Variables used for communication
- The propose and modify paradigm:
  1. SIMION calculates: proposed time step
  2. SIMION calls Tstep_Adjust (as per rules) passing value in ion_time_step
  3. Tstep_Adjust examines and/or modifies the value in ion_time_step
- Reserved Variable r/w access is controlled
Reserved Variables

- Reserved Variables used for communication between SIMION and program segments

<table>
<thead>
<tr>
<th>Variable Name</th>
<th>Use</th>
<th>Units</th>
<th>Read Access</th>
<th>Write Access</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ion_Px_Abs_gu</td>
<td>Varies</td>
<td>Volts</td>
<td>Init_P_Values</td>
<td>Init_P_Values</td>
</tr>
<tr>
<td>Ion_Py_Abs_gu</td>
<td>Varies</td>
<td>Volts</td>
<td>Init_P_Values</td>
<td>Init_P_Values</td>
</tr>
<tr>
<td>Ion_Pz_Abs_gu</td>
<td>Varies</td>
<td>Volts</td>
<td>Init_P_Values</td>
<td>Init_P_Values</td>
</tr>
<tr>
<td>Ion_Px_mm</td>
<td>Ion's current grid</td>
<td>mm</td>
<td>Init_P_Values</td>
<td>Init_P_Values</td>
</tr>
<tr>
<td>Ion_Py_mm</td>
<td>Ion's current grid</td>
<td>mm</td>
<td>Init_P_Values</td>
<td>Init_P_Values</td>
</tr>
<tr>
<td>Ion_Pz_mm</td>
<td>Ion's current grid</td>
<td>mm</td>
<td>Init_P_Values</td>
<td>Init_P_Values</td>
</tr>
<tr>
<td>Ion_BfieldX_gu</td>
<td>Magnetic field</td>
<td>Gauss</td>
<td>Accel_Adjust</td>
<td>Accel_Adjust</td>
</tr>
<tr>
<td>Ion_BfieldY_gu</td>
<td>Magnetic field</td>
<td>Gauss</td>
<td>Accel_Adjust</td>
<td>Accel_Adjust</td>
</tr>
<tr>
<td>Ion_BfieldZ_gu</td>
<td>Magnetic field</td>
<td>Gauss</td>
<td>Accel_Adjust</td>
<td>Accel_Adjust</td>
</tr>
<tr>
<td>Ion_Charge</td>
<td>Ion's current charge</td>
<td>Volts/grid unit</td>
<td>Init_P_Values</td>
<td>Init_P_Values</td>
</tr>
<tr>
<td>Ion_Color</td>
<td>Color of ion</td>
<td>0-15</td>
<td>Init_P_Values</td>
<td>Init_P_Values</td>
</tr>
<tr>
<td>Ion_DvoltsX_gu</td>
<td>Voltage gradient</td>
<td>Volts/mm</td>
<td>Effi ped_Adjust</td>
<td>Effi ped_Adjust</td>
</tr>
<tr>
<td>Ion_DvoltsY_gu</td>
<td>Voltage gradient</td>
<td>Volts/mm</td>
<td>Effi ped_Adjust</td>
<td>Effi ped_Adjust</td>
</tr>
<tr>
<td>Ion_DvoltsZ_gu</td>
<td>Voltage gradient</td>
<td>Volts/mm</td>
<td>Effi ped_Adjust</td>
<td>Effi ped_Adjust</td>
</tr>
<tr>
<td>Ion_Instance</td>
<td>Current instance</td>
<td>1+max instance</td>
<td>Init_P_Values</td>
<td>None</td>
</tr>
<tr>
<td>Ion_Mass</td>
<td>Ion's current mass</td>
<td>amu</td>
<td>Init_P_Values</td>
<td>Init_P_Values</td>
</tr>
<tr>
<td>Ion_mm_Pot_Ground</td>
<td>Max Current Scaling</td>
<td>mm/gal unit</td>
<td>Init_P_Values</td>
<td>None</td>
</tr>
<tr>
<td>Ion_Number</td>
<td>Ion's Number</td>
<td>1+max ion</td>
<td>Init_P_Values</td>
<td>None</td>
</tr>
</tbody>
</table>

Variable Name | Use                | Units | Read Access | Write Access |
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Ion_Px_Abs_gu</td>
<td>Ion's current PA</td>
<td>grid units</td>
<td>Init_P_Values</td>
<td>Init_P_Values</td>
</tr>
<tr>
<td>Ion_Py_Abs_gu</td>
<td>Ion's current PA</td>
<td>grid units</td>
<td>Init_P_Values</td>
<td>Init_P_Values</td>
</tr>
<tr>
<td>Ion_Px_gu</td>
<td>Ion's current grid</td>
<td>mm</td>
<td>Init_P_Values</td>
<td>Init_P_Values</td>
</tr>
<tr>
<td>Ion_Py_gu</td>
<td>Ion's current grid</td>
<td>mm</td>
<td>Init_P_Values</td>
<td>Init_P_Values</td>
</tr>
<tr>
<td>Ion_Pz_gu</td>
<td>Ion's current grid</td>
<td>mm</td>
<td>Init_P_Values</td>
<td>Init_P_Values</td>
</tr>
<tr>
<td>Ion_Splat</td>
<td>Ion Status Flag</td>
<td>None</td>
<td>Init_P_Values</td>
<td>Init_P_Values</td>
</tr>
<tr>
<td>Ion_Time_of_Birth</td>
<td>Ion's Birth Time</td>
<td>micro seconds</td>
<td>Init_P_Values</td>
<td>Init_P_Values</td>
</tr>
<tr>
<td>Ion_Time_of_Flight</td>
<td>Ion's current TDP</td>
<td>micro seconds</td>
<td>Init_P_Values</td>
<td>Init_P_Values</td>
</tr>
<tr>
<td>Ion_Instance</td>
<td>Current instance</td>
<td>1+max instance</td>
<td>Init_P_Values</td>
<td>None</td>
</tr>
<tr>
<td>Ion_Volts</td>
<td>Electrostatic Pot</td>
<td>Volts</td>
<td>Effi ped_Adjust</td>
<td>Effi ped_Adjust</td>
</tr>
<tr>
<td>Ion_Vx_mm</td>
<td>Ion's current Velocity</td>
<td>mm/micro sec</td>
<td>Init_P_Values</td>
<td>Init_P_Values</td>
</tr>
<tr>
<td>Ion_Vy_mm</td>
<td>Ion's current Velocity</td>
<td>mm/micro sec</td>
<td>Init_P_Values</td>
<td>Init_P_Values</td>
</tr>
<tr>
<td>Ion_Vz_mm</td>
<td>Ion's current Velocity</td>
<td>mm/micro sec</td>
<td>Init_P_Values</td>
<td>Init_P_Values</td>
</tr>
<tr>
<td>Retrun_Flym</td>
<td>Return Flym Flag</td>
<td>NO = 0</td>
<td>YES != 1</td>
<td>Init_P_Values</td>
</tr>
<tr>
<td>Trajectory_Image_Control</td>
<td>Control Trajectory Image Viewing and Recording</td>
<td>Value</td>
<td>Yes</td>
<td>Init_P_Values</td>
</tr>
<tr>
<td>Return_Changed_Potentials</td>
<td>Control restoring of changed potentials at end of Fly</td>
<td>YES != 1</td>
<td>YES = 1</td>
<td>None</td>
</tr>
<tr>
<td>Update_PE_Surface</td>
<td>New PE Surface</td>
<td>YES = 0</td>
<td>None</td>
<td>Other_Actions</td>
</tr>
</tbody>
</table>
Temporary Variables

- Visibility: Visible only in user program segment that defines it
- Lifetime: Exists only during a reference to the program segment
- User Adjustability: No
- Conserves Changes: No
- Error Trapping: Yes (for sto without rcl)
Static Variables & Arrays

- Visibility: Globally visible to most user programs except: Initialize, Init_P_values, and Terminate.
- Lifetime: Reset before each ion or group flight.
- User Adjustability: No
- Conserves Changes: No
Adjustable Variables & Arrays

- Visibility: Globally visible to all user programs
- Lifetime: Values conserved throughout entire fly’m (including reruns).
- User Adjustable:
  - Before ion flying starts
  - During ion flying (via AdjV tabbed screen)
- Conserves Changes: YES (before fly’m)
Controlling Variable Visibility

- Adjustable Variables
  - Use leading underscore in names to see (e.g. `_Left_Electrode_Voltage`)
Controlling Variable Visibility

- Use Adjustable & Static Arrays for Flags

```
Adefa aflags 20 ;”defaults.dat” adjustable flags -- initialized
Adefs sflag 500 ; static array flag -- zeroed for each ion
1 arcl aflags ; simple recall (x = aflags[1])
0 1 sto aflags ; simple store (aflags[1] = 0)

defa flag 0 ; flag shown at start (unless _OK underscore)
rcl flag
1 sto flag
```

The Idaho National Engineering and Environmental Laboratory

Page 11-12
Creating Integrated Demos

1. Ke to Velocity (Test Demo)
Creating Integrated Demos

1. Ke to Velocity (Test Demo)
Creating Integrated Demos

1. Ke to Velocity (Test Demo)

seg other_actions

rcl ion_splat x=0 exit ;exit if ion hasn’t splatted
rcl ion_mass x>y goto ions ;jump if proton
message :Relativistic ke -> v Test
message :An electron with ke = its rest mass
message :relativistic velocity from
message :v = c * sqrt(0.75)
message :given c = 2.99792458e+005 mm/usec
rcl ion_vx_mm ; velocity at splat
message ;v = # mm/usec (SIMION)
message ;rest mass of electron
message ;and starting ke
rcl ion_mass
rcl ion_vx_mm
>ke
message ;ke = # eV (SIMION v >KE fn)
message ;equation for const velocity tof
message ;tof = 1000mm / v ;array and workbench 1000mm long
message ;tof assuming expected velocity
rcl ion_time_of_flight
message ;tof = # usec (SIMION)
message ;exit

lbl ions ;proton data output
message :Non-Relativistic ke -> v Test
message :A proton with ke = 1eV
message :v = sqrt(2 * ke/m)
message :v = sqrt(2.0 * 1.60217733e-19/1.6726231e-27)
message :where
message :1eV = 1.60217733e-19 J
message :proton’s mass = 1.6726231e-27 kg
message ;v = 1.38411203e+001 mm/usec (expected)
rcl ion_vx_mm ; velocity at splat
message ;v = # mm/usec (SIMION)
message ;defined initial ke
rcl ion_mass
rcl ion_vx_mm
>ke
message ;ke = # eV (SIMION v >KE fn)
message ;equation for const velocity tof
message ;tof = 1000mm / v ;array and workbench 1000mm long
message ;tof assuming expected velocity
rcl ion_time_of_flight
message ;tof = # usec (SIMION)
message ;exit
Creating Integrated Demos

- Recommended Strategies
  - Use auto-loading FLY (or ION) and REC files
  - Control displayed number format with REC file
  - Use self-evident message output if possible via Initialize, Other_Actions, Terminate, and/or data recording (REC)
  - Hide Adjustable Variables from user (e.g. _Voltage (shown), flag (hidden), or via arrays)
Randomizing Ions

- Use Initialize Segment to Randomize Ions

Initialize

Input Ions
Group of ions with identical starting parameters.

Adjustable Variables
Percent_Energy_Variation
Cone_Angle_Off_Vel_Axis
Randomizing Ions

**Initialize Segment Strategy**

- Get polar v components
- Compute KE
- Randomize KE
- Convert to v

- Get ion’s velocity components from SIMION
- Convert to 3D polar and save speed plus az and el angles
- get ion_mass, recall speed, convert to ke, and save ke
- use rand and percent_energy_variation to randomize ke
- get ion_mass, recall new ke, convert to v, and save speed

- Create random conical pattern in vertical
- Convert to 3D Rectangular
- Swing back to az = el = 0

- assume ion flying vertical
- randomize el off vertical by cone angle
- randomize az to +- 180 degrees
- convert to rectangular 3D coords
- rotate vector -90 in elevation (restores a starting point)

- Rotate entry el
- Rotate entry az

- rotate vector initial el angle
- rotate vector initial az angle

- Pass V Rect. to SIMION
- pass ion’s velocity components back to SIMION
Initialize for Random Ions

; this user program randomly changes the initial ke and direction of ions
; energy is randomly changed -> Percent_Energy_Variation * ke
; ions are emitted randomly within a cone of revolution around the
; ion's defined velocity direction axis
; the full angle of the cone is +- Cone_Angle_Off_Vel_Axis
; (e.g. 90.0 is full hemisphere, 180 is a full sphere)
--------------------------------------------------------------------
; you can use it with your own lenses without modification ----------
; (just rename user program file using your pa's name)
; Note: you can also modify the emission distributions as desired ------
defa Percent_Energy_Variation     50    ; (+- 50%) random energy variation
defa Cone_Angle_Off_Vel_Axis      90    ; (+- 90 deg) cone angle hemisphere
--------------------------------------------------------------------

seg initialize ; initialize ion's velocity and direction
;------------------- get ion's initial velocity components ------------
rcl ion_vz_mm       ; get ion's specified velocity components
rcl ionVy_mm
rcl ion_vx_mm
;------------------- convert to 3d polar coords ----------------
>p3d                ; convert to polar 3d
;------------------- save polar coord values ----------------
sto speed            ; store ion's speed
sto az_angle          ; store ion's az angle
sto el_angle          ; store ion's el angle
;------------------- make sure Percent_Energy_Variation is legal ------------
; force 0 <= Percent_Energy_Variation <= 100
rcl Percent_Energy_Variation abs
100 x>y rlup stp Percent_Energy_Variation
;------------------- make sure Cone_Angle_Off_Vel_Axis is legal ------------
; force 0 <= Cone_Angle_Off_Vel_Axis <= 180
rcl Cone_Angle_Off_Vel_Axis abs
180 x>y rlup stp Cone_Angle_Off_Vel_Axis
;------------------- calculate ion's defined ke ----------------
rcld ion_mass         ; get ion's mass
rcld speed            ; recall its total speed
>ke                   ; convert speed to kinetic energy
sto kinetic_energy    ; save ion's defined kinetic energy
;------------------- compute new randomized ke ----------------
rcld Percent_Energy_Variation 100 /
sto del_energy 2 * rand    ; fac = 2 * del_energy * rand
rcld del_energy * 1 +     ; fac += 1 - del_energy
rcld kinetic_energy *     ; new ke = fac * ke
;------------------- convert new ke to new speed ----------------
rcld ion_mass          ; recall ion mass
x><y                   ; swap x any y
>spd                   ; convert to speed
sto speed              ; save new speed
;------------------- compute randomized el angle change 90 +- Cone_Angle_Off_Vel_Axis -------
; we assume elevation of 90 degrees for mean
; so cone can be generated via rotating az +- 90 --------
2 rcl Cone_Angle_Off_Vel_Axis * rand   ; rand = - Cone_Angle_Off_Vel_Axis + 90
rcld Cone_Angle_Off_Vel_Axis - 90 +
;------------------- compute randomized az angle change ------------
; this gives 360 effective because of +- elevation angels ---
180 rand * 90 -        ; +- 90 randomized az
;------------------- recall new ion speed ----------------
rcld speed              ; recall new speed
;------------------- at this point x = speed, y = az, z = el ------------
;------------------- convert to rectangular velocity components -----------
>3d                    ; convert polar 3d to rect 3d
;------------------- el rotate back to from 90 vertical ------------
-90 >elr
;------------------- el rotate back to starting elevation -----------
rcld el_angle >elr
;------------------- az rotate back to starting azimuth -------------
rcld az_angle >az
;------------------- update ion's velocity components with new values ------
sto ion_vx_mm          ; return vx
rlup
sto ion_vy_mm          ; return vy
rlup
sto ion_vz_mm          ; return vz
rlup
;------------------- done -----------------------------------------
Another Randomizing Example

- Monte Carlo example
- Secondary ion energies (fitting)

Molecular Ion Energy Distribution

\[ y = 347.32x^5 - 866.99x^4 + 819.18x^3 - 358.17x^2 + 71.686x - 2.9558 + 0.0112 \]

\[ R^2 = 0.9997 \]

Relative Intensity vs. Stopping Energy (eV)

Molecular Ions
Electrons

Energy (eV)

Random number from 0-1
Another Randomizing Example

- Monte Carlo example cont.
- Secondary ion energies (coding)

```plaintext
;molecular energies
rand sto x
    ;get random number x (0-1)
0.0112
    ; +0.0112
2.9558 rcl x * - ; -2.9558x
71.686 rcl x rcl x * sto xpower * + ; +71.686x^2
358.17 rcl xpower rcl x * sto xpower * - ; -358.17x^3
819.18 rcl xpower rcl x * sto xpower * + ; +819.18x^4
866.99 rcl xpower rcl x * sto xpower * - ; -866.99x^5
347.22 rcl xpower rcl x * sto xpower * + ; +347.22x^6
abs
```
Controling Times and Time Steps

- Critical Timing Example
Controlling Times and Time Steps

- Strategy: Tstep_Adjust via Ion_Time_Step

    If TOF >= left then right test
    If TOF + TS < left then exit
    ion_time_step = left - TOF
    exit

    LBL right test
    If TOF >= right then exit
    If TS > MAX then TS = MAX
    If TOF + TS < right then exit
    ion_time_step = right - TOF
    exit
Binary Boundary Approach

- Use Tstep_Adjust and Other_Actions

Kill Radius = 25 mm
Binary Boundary Approach

Strategy

Other_Actions Control
If boundary crossed
  if not too close
  restore prior ion state
  save time_step
  set half step flag
else
  save ion’s state
  (Px,Py,Pz,
   Vx,Vy,Vz,TOF)

Tstep Control
If half step flag not set
  exit
else
  get saved time_step
  halve it
  save to ion_time_step
  clear half step flag
Fast Adjusting Potentials

- Strategy: Fast_Adjust or Init_P_Values
  - Time varying fields: Fast_Adjust (only)
    - Tstep_Adjust may be needed for step size control
  - Static fields: Fast_Adjust or Init_P_Values
    - Fast_Adjust for large arrays and smaller numbers of ions.
    - Init_P_Values for smaller arrays and larger numbers of ions.
  - Conduct time trials (very useful).
Specifying Known Fields

- Strategy: Mfield_Adjust or Efield_Adjust
  - Fixed Magnetic fields (Mfield_Adjust)
    - Use field-free magnetic array

- 0 sto BfieldZ_gu ;zero Bz
- sto BfieldY_gu ;zero By
- rcl _Bfield_gauss ;get Bx from Adj. Var.
- sto BfieldX_gu ;save user defined Bx
Specifying Known Fields

- **Strategy:** Mfield_Adjust or Efield_Adjust
  - Measured Magnetic fields (Mfield_Adjust)
    - Use field-free magnetic array
    - Adefa Bfield 10000 ;”mfield.dat” measured B
  
  - Get ion’s gu coords (abs or 3D)
    Use to get interpolated B fields at point
    Pass Bx, By, and Bz fields back to SIMION
    (see measured magnetic fields example)
Specifying Known Fields

- Strategy: Mfield_Adjust or Efield_Adjust
- Analytic Electrostatic fields (Efield_Adjust)

Selectively use:

```
seg efield_adjust
  3 rcl ion_number x<=y exit
; quadrupolar field
  ; use refined field <= 3
```

Compute and store potential:

```
rcl ion_px_mm entr *
rcl ion_py_mm entr * -
10 / sto ion_volts
rcl ion_px_mm 0.2 *
rcl ion_mm_per_grid_unit *
sto ion_dvoltsx_gu
```

Compute and store x gradient:

```
rcl ion_py_mm -0.2 *
rcl ion_mm_per_grid_unit *
sto ion_dvoltsy_gu
```

Zero z gradient:

```
0 sto ion_dvoltsz_gu
exit
```

Compute and store y gradient:

```
rcl ion_px_mm -0.2 *
rcl ion_mm_per_grid_unit *
sto ion_dvoltsy_gu
```

The Idaho National Engineering and Environmental Laboratory
Modifying Ion Accelerations

- **Strategy:** Use Accel_Adjust

```plaintext
seg accel_adjust  ; beginning of accel_adjust segment

rcl ion_time_step x=0 exit  ; exit if zero time step
rcl linear_damping x=0 exit  ; exit if damping set to zero
abs sto damping  ; force damping term to be positive
* sto tterm  ; compute and save number of time constants
chs e^x 1 x>-y -  ; (1 - e^(-(t * damping)))
rcl tterm / sto factor  ; factor = (1 - e^(-(t * damping)))/(t * damping)

rcl ion_ax_mm  ; recall ax acceleration
rcl ion_vx_mm  ; recall vx velocity
rcl damping * -  ; multiply times damping and sub from ax
rcl factor *  ; multiply times factor
sto ion_ax_mm  ; store as new ax acceleration

rcl ion_ay_mm  ; recall ay acceleration
rcl ion_vy_mm  ; recall vy velocity
rcl damping * -  ; multiply times damping and sub from ay
rcl factor *  ; multiply times factor
sto ion_ay_mm  ; store as new ay acceleration

rcl ion_az_mm  ; recall az acceleration
rcl ion_vz_mm  ; recall vz velocity
rcl damping * -  ; multiply times damping and sub from az
rcl factor *  ; multiply times factor
sto ion_az_mm  ; store as new az acceleration
```

Drag Demo
Arbitrary Ion Accelerations

- **Strategy:** Use Accel_Adjust
  - Can define arbitrary field forces (e.g. $1/r^5$)
  - Can define arbitrary field directions
    - Always normal to ion’s current velocity
    - Always parallel to ion’s current velocity
    - Moving points of attraction
    - Random points of attraction
    - Whatever you can imagine
Controlling Ion’s Fate

- Strategy: Other_Actions
  - Position
  - Velocity
  - Mass
  - Charge
  - Color
  - TOF
  - Splat

User Programming’s Master Controller

(Most Powerful Program Segment)
Controlling Ion’s Fate

- Strategy: Other_ACTIONS
  - Mass and/or charge
  - Energy and/or position
  - Boundary approaches
  - Collision simulations
  - Simulations involving jumps
  - Selective ion marking (e.g. via color & marks)
  - Selective ion killing
Controlling Ion’s Fate

- Non-ideal grid
- Resulting ion trajectories
- Other_Actions used to jump ions
Controlling Successive Runs

- Strategy: Terminate and Initialize
  - Via the Rerun_Flym reserved variable
Controlling Successive Runs

- Strategy: Terminate and Initialize

- **Initialize**
  - If first run
  - set starting potential
  - Clear Rerun_Flym flag

- **Fast_Adjust**
  - Init_P_Values
  - Set electrode potential

- **Terminate**
  - Get test results
  - If goal attained then exit
  - Compute new potential
  - Set Rerun_Flym flag
Complex Multi-Run Simulations

- Monte Carlo -- Self-Charge Stabilization
Afternoon User Programming Lab

- Self-directed lab. Do what interests you.

- Options:
  - Form groups to work on a common interest
  - Explore user program demos in reference
  - Continue to explore a previous lab
  - Attack the instructors