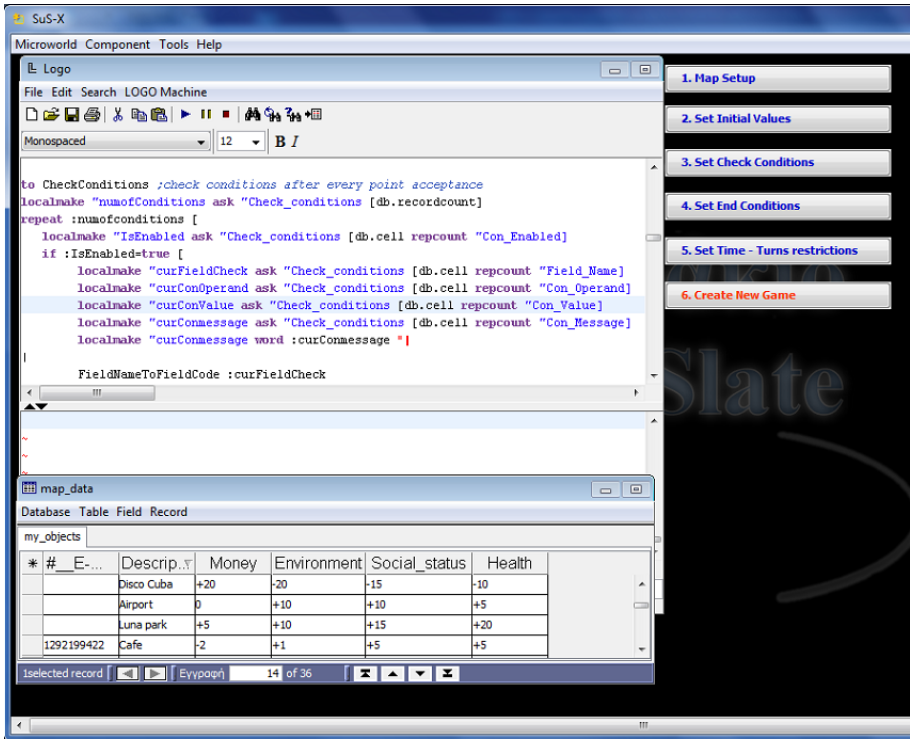
Educational Technology Lab

National and Kapodistrian University of Athens
School of Philosophy
Faculty of Philosophy, Pedagogy and Philosophy (P.P.P.), Department of Pedagogy

Director: Prof. C. Kynigos



Sus-X
Technical Manual

Ver.: 2.1

Table of contents

The purpose of the manual

The purpose of this manual is to provide appropriate information to every teacher, researcher or even to an advanced user who wants to modify the functionalities of the game generator Sus-X, by changing the existing features or by adding new ones.

For this purpose, the procedures that take place while playing the game is important to be understood, so as to make its deconstruction possible.

Designing a new game

During the "design phase" of the game, no Logo programming is needed, only simple display "Views" commands which are located within the “events” buttons.

“Views” is an inherent functionality of “E-Slate” authorial platform and are referred to stored viewing modes of components on the screen. “Views” allow us to store, display or hide components and also to save (with a name of our choice) the exact appearance of the game (or the microworlds’ appearance in general).

Playing the game

After the "design phase”, when the new game is ready, all the processes are taking place through a logo program which is scripted in the "Logo" component.

Start a game

At the start of the game (when “Start” button is pressed):

• The chronometer starts to measure the time
• Initial values are loading to the game properties
• Counting the number of map points is initialized
• "Select" button is displayed in order to select a point on the map
• 'Start' button, changes appearance
• Previous game elements are removed from the game data base
• The actions of the player (logging) are recording

Select a point on the map

When a point is selected (on the map) from the player ("Select” button):

• Any previous message from the text box (that displays information) is deleted
• The number of the selected points is increased
• A new record with all the elements - properties of the selected point is added to the Data Base of the game
•Logo checks if the maximum number of turns is completed
• Logo checks if any of the property values activates any of the termination criteria set by the designer
•If there is no reason for ending the game the "check conditions" set by the designer are checked and if any of them is valid, a posted message is received.
• The game keeps a record of players' actions (logrecord procedure).

Ending the game

If the game is ended by "internal" reasons, e.g because time or turns limit is reached or an ending condition set by the designer is activated:

- The chronometer stops
- Information about the game state at the time of termination is displayed
- The reason of game ending is appeared
- "Stop" button changes appearance
- "Select" button hides

If the game is terminated by the player ("Stop" button) the same actions as above are performed but in that case there is no message describing the reason for ending the game.

Brief description of the LOGO procedures

Starting or Ending a game

- Logo checks if the user asks for starting or ending the game (from the same button)
- For starting the game, the initial values are recovered from the DB (Game_properties)
- Previous game data are deleted from the users' DB (game_data)

```
to New_Game ; when start or stop game button is pressed
localmake "action ask "new [button.text]
if :action="|End Game| [
ask "chrono [stopchronometer]
ask "new [button.settext "|Start Game|]
ask "new [button.setfgcolor [0 150 0]]
ask "accept [hide]
ask "|Remaining| [hide]
ask "SpinEndGame [sbutton.setvalue -1]
stop
]
make "MaxTurns ask "Turns [sbutton.value]
make "TimeLimit ask "Available_time [sbutton.value]
make "EndReason "none
make "FieldNames ask "map_data [db.fieldnames]
make "FieldNames Butfirst :FieldNames
make "FieldNames Butfirst :FieldNames
make "FieldNames ExcludeFieldsFromList(:FieldNames)
make "FieldCount length :fieldnames
ask "chrono [resetchronometer]
ask "chrono [startchronometer]
ask "Available_time [hide]
Ask "Turns [hide]
ask "lbl_time [hide]
ask "lbl_turns [hide]
ask "TXT_end [area.settext "||]
ask "TXT_Check [area.settext "||]
ask "TXT_end [hide]
ask "TXT_check [hide]
Ask "Accept [restore]
ask "new [button.settext "|End Game|]
ask "new [button.setfgcolor [255 0 0]]
ask "|Remaining| [restore]
localmake "recordcount ask "game_data [(db.recordcount "game_table )]
repeat :recordcount [ask "game_data [(db.removerecord "game_table 1) ]]
ask "Game_data [(db.addrecord "game_table )]
ask "Game_data [(db.setcell "game_table 1 "Description  "|NEW GAME| )]
repeat :FieldCount [
        localmake "initValue ask "Game_properties [db.cell repcount "Initial_value]
    localmake "curFieldname ask "Game_properties [db.cell repcount "Field_name]
        ask "Game_data [(db.setcell "game_table 1 :curFieldname  :initValue )]
]
make "curPoint 0
make "TurnsRemain :Maxturns
ask "logtext [area.settext "||]
logrecord true
ask "SpinEndGame [sbutton.setvalue 0]
endend
```

Choose - accept the selected point on the map

- The values of the points from the map DB are retrieved and inputted on the users' DB (game_data)
- Logo checks if the maximum number of turns is reached–If this happens, the game will come to an end (GameEnded).
- Logo checks if the game should be ended due to selected conditions (CheckEndConditions)
- Logo checks the property values in order to display warning messages (CheckConditions)

```
to Accept_point ;get and save the chosen point
ask "TXT_check [area.settext "||]
make "curPoint :curPoint+1
make "messsageStr "||
ask "game_data [(db.addrecord "game_table )]
localmake "curMapRecord ask "map_data [db.activerecord]
ask "game_data [
localmake "fieldlist (db.fieldnames "game_table)
]
repeat length :fieldlist [
        localmake "curField item repcount :fieldlist
        localmake "curFieldValue ask "map_data [db.cell :curMapRecord :curField]
        localmake "curRec :curPoint+1
        ask "game_data [(db.setcell "game_table :curRec :curField :curFieldValue)]
]
updateRunningSum
If :curPoint = :MaxTurns [
        make "EndReason "byMaxTurns
    GameEnded
        ]

CheckEndConditions
if :EndReason="none [
  checkconditions
  GetAndShowValues "TXT_check
]
make "TurnsRemain :MaxTurns - :curPoint
logrecord false
end
```

Properties control to display warning messages

• Check conditions are retrieved from DB «Check_conditions»
• Properties are compared with check conditions and a cumulative message is appeared.

```
to CheckConditions ;check conditions after every point acceptance
localmake "numoflines ask "Check_conditions [db.recordcount]
print se "Total_check_conditions= :numoflines
repeat :numoflines [
    localmake "IsEnabled ask "Check_conditions [db.cell repcount "Con_Enabled]
    if :IsEnabled=true [
            localmake "con1 ask "Check_conditions [db.cell repcount "Condition1]
            localmake "con2 ask "Check_conditions [db.cell repcount "Condition2]
            localmake "con3 ask "Check_conditions [db.cell repcount "Condition3]
            localmake "con4 ask "Check_conditions [db.cell repcount "Condition4]
            localmake "totalcon 4
            Localmake "foundcon1 0
            Localmake "foundcon2 0
            Localmake "foundcon3 0
            Localmake "foundcon4 0
       localmake "info_cond se :con1 :con2
       localmake "info_cond se :info_cond :con3
       localmake "info_cond se :info_cond :con4
       print se "Check? :info_cond
            ask "Game_data [
              ifelse not emptyp :con1 [
              localmake "foundcon1 length (db.select "running_sum  :con1)]
         [ localmake "totalcon :totalcon - 1]
         ifelse not emptyp :con2 [
         localmake "foundcon2 length (db.select "running_sum  :con2)]
         [ localmake "totalcon :totalcon - 1]
         ifelse not emptyp :con3 [
         localmake "foundcon3 length (db.select "running_sum  :con3)]
         [ localmake "totalcon :totalcon - 1]
         ifelse not emptyp :con4 [
         localmake "foundcon4 length (db.select "running_sum  :con4)]
         [ localmake "totalcon :totalcon - 1]
       ]
            if :foundcon1 + :foundcon2 + :foundcon3 + :foundcon4 = :totalcon
       [
            localmake "curConmessage ask "Check_conditions [db.cell repcount "Con_Message ]
            localmake "curConmessage word :curConmessage "|
|
            ask "TXT_check [area.append :curConmessage] ]
    ]
]
ask "TXT_check [restore]
end
```

Checks if end conditions of the game are met

- End conditions are retrieved from DB «End_conditions»
- Properties are compared with end conditions

```
to CheckEndConditions ;check if the game is ended by end conditions
localmake "numofEndConditions ask "End_conditions [db.recordcount]
print se "Total_end_conditions= :numofEndConditions
repeat :numofendconditions [
  localmake "IsEnabled ask "End_conditions [db.cell repcount "End_Enabled]
  if :IsEnabled=true [
            localmake "con1 ask "End_conditions [db.cell repcount "End_Condition1]
            localmake "con2 ask "End_conditions [db.cell repcount "End_Condition2]
```

```
            localmake "con3 ask "End_conditions [db.cell repcount "End_Condition3]
            localmake "con4 ask "End_conditions [db.cell repcount "End_Condition4]
            localmake "totalcon 4
            localmake "foundcon1 0
            localmake "foundcon2 0
            localmake "foundcon3 0
            localmake "foundcon4 0
            localmake "info_end_cond se :con1 :con2
            localmake "info_end_cond se :info_end_cond :con3
            localmake "info_end_cond se :info_end_cond :con4
            print se "End? :info_end_cond
            ask "Game_data [
              ifelse not emptyp :con1 [
              localmake "foundcon1 length (db.select "running_sum  :con1)]
          [ localmake "totalcon :totalcon - 1]
          ifelse not emptyp :con2 [
          localmake "foundcon2 length (db.select "running_sum  :con2)]
          [ localmake "totalcon :totalcon - 1]
          ifelse not emptyp :con3 [
          localmake "foundcon3 length (db.select "running_sum  :con3)]
          [ localmake "totalcon :totalcon - 1]
          ifelse not emptyp :con4 [
          localmake "foundcon4 length (db.select "running_sum  :con4)]
          [ localmake "totalcon :totalcon - 1]
       ]
            if :foundcon1 + :foundcon2 + :foundcon3 + :foundcon4 = :totalcon
       [
            localmake "curEndMessage ask "End_conditions [db.cell repcount "End_Message ]
            localmake "curEndMessage word :curEndMessage "|
|
       ask "TXT_End [area.append :curEndMessage]
                                make "EndReason "byConditions
                                ]]
  ]
if :EndReason="byConditions [GameEnded]
end
```

Calculating and displaying properties (their values)

- Calculation of property values in the users' DB (game_data)
- Text composition and displaying providing all the information

```
to GetAndShowValues :TextAreaName ; Calculates and shows all field values
ask "game_data [
            localmake "fieldlist (db.fieldnames "running_sum)
            repeat length :fieldlist [
            localmake "curField item repcount :fieldlist
            localmake "curFieldValue (db.cell "running_sum 1 :curfield)
                                localmake "lineText word :curField "=
                                localmake "lineText word :lineText :curFieldValue
                                ask :TextAreaName [area.append :lineText]
                                ask :TextAreaName [area.append "|
|]
            ]
]
end
```

Checks the time out limit of the game

- Time calculation of the game
- Show the remaining game time

```
to CheckTimeLimit ; checks if time is up
localmake "curTime chronometerminutes
ifelse :curTime > :timelimit [
      make "EndReason "byTime
      GameEnded
][
make "timeRemain :timelimit - integer :curTime
localmake "lblRemaining word "| Remaining...| :timeRemain
localmake "lblRemaining word :lblRemaining "| minute(s) -- |
localmake "lblRemaining word :lblRemaining :TurnsRemain
localmake "lblRemaining word :lblRemaining "| turn(s)|
Ask "Remaining [label.settext :lblRemaining]
]
end
```

Ending game procedure

- The chronometer stops
- The ending reason is checked – A proper message is displaying
- The game properties are displayed by the procedure GetAndShowValues

```
to GameEnded ;shows proper message when game is ended
ask "chrono [stopchronometer]
ask "|Remaining| [hide]
if :EndReason="byTime [make "messageStr "| The Time Is Up!|]
if :EndReason="byConditions [make "messageStr "|The Game is ended!|]
if :EndReason="byMaxTurns [make "messageStr "| You have reached the maximum number of turns |]
make "messageStr word :messageStr "|
|
ask "TXT_end [area.append :messageSTR]
ask "TXT_end [restore]
ask "TXT_check [hide]
ifelse and :EndReason="byConditions :curPoint=:Maxturns []][
GetAndShowValues "TXT_end]
Ask "Accept [hide]
ask "new [button.settext "|Start Game|]
ask "new [button.setfgcolor [0 150 0]]
if :EndReason="byTime [
ask "SpinEndGame [sbutton.setvalue 1]
]
if :EndReason="byConditions [
ask "SpinEndGame [sbutton.setvalue 2]
]
if :EndReason="byMaxTurns [
ask "SpinEndGame [sbutton.setvalue 3]
]
end
```

Actions recording (recordings logging)

• This procedure is called at the start of the game, by selecting a point on the map and at the end of the game
• Records the time and property values

```
to LogRecord :newgame
ifelse :newgame = true [
  localmake "curDescr "|NEW GAME|
  ][
  localmake "curdbaserecord ask "map_data [ db.activerecord]
  localmake "curDescr ask "map_data [db.cell :curdbaserecord "Περιγραφή ]
]

make "curTime (chronometertime)
localmake "timeword item 1 :curtime
localmake "timeword word :timeword ":
localmake "timeword word :timeword item 2 :curtime
localmake "timeword word :timeword ":
localmake "timeword word :timeword item 3 :curtime

localmake "linetext word :curPoint + 1 "|# |
localmake "linetext word :lineText :timeword
localmake "linetext word :lineText "| [|
localmake "linetext word :lineText :curDescr
localmake "linetext word :lineText "|]|
localmake "fieldlist ask "map_data [db.fieldnames]
ask "game_data [
localmake "fieldlist (db.fieldnames "running_sum)
]
repeat length :fieldlist [
      localmake "curfield item repcount :fieldlist
           localmake "linetext word :lineText "| |
           localmake "lineText word :lineText :curField
           localmake "linetext word :lineText "=
           ifelse :newgame=true [
             localmake "curFieldVal ask "game_data [(db.cell "game_table 1 :curField )]
           ][
             localmake "curFieldVal ask "game_data [(db.dsum "game_table :curField )]
           ]
           localmake "lineText word :lineText :curFieldVal
]
ask "logtext [area.append :linetext]
ask "logtext [area.append "|
|]
end
```

Supporting procedures

• CreateInitials: Automatically keeps records on the table «Game_properties» who keeps the initial property values of points. This procedure sets initial value = 0 in all records and called by the "Set initial values." button
• dropTable: Removes all the database fields in a table.
• addTableFields: Adds a table in a list of fields.
• createGameDataTable: Creates the fields in the game data table («game_table») and also the fields in the table that keeps the score («running_sum»). This procedure is called by the "Create" button in the transition from the“design phase” to the “game phase”.
• updateRunningSum: Updates the “running_sum” table with the fields totals of «game_table» so it creates the score. It is called every time a new point is selected by the user in the game.

• ExcludeFieldsFromList: Receives a list of fields and returns the same list without fields starting by default characters. It is used to exclude from calculations DB fields of the map which are considered to be supporting and are containing text data format.

```
to CreateInitials ; create records in game_properties
localmake "mapfieldlist butfirst ask "map_data [db.fieldnames]
localmake "mapfieldlist butfirst :mapfieldlist
localmake "mapfieldlist ExcludeFieldsFromList(:mapfieldlist)
print :mapfieldlist
localmake "mapfieldcount length :mapfieldlist
print :mapfieldcount
repeat length :mapfieldlist [
  localmake "curField item repcount :mapfieldlist
  localmake "query word "Field_name= :curField
  localmake "FieldFound ask "Game_properties [db.select :query ]
  if emptyp :FieldFound [
    ask "Game_properties [db.addrecord]
    localmake "lastrecord ask "Game_properties [db.recordcount]
    ask "Game_properties [db.setcell :lastrecord "Field_name :curField]
    ask "Game_properties [db.setcell :lastrecord "Initial_value "0 ]
  ]
]
end


to dropTable :dbname :tablename
ask :dbname [
localmake "fieldlist (db.fieldnames :tablename)
repeat length :fieldlist [
(db.removefield :tablename item repcount :fieldlist)
]
]
end

to addTableFields :dbname :tablename :fieldlist
ask :dbname [
repeat length :fieldlist [
(db.addfield :tablename item repcount :fieldlist "float )
]
]
end


to createGameDataTable
dropTable "Game_data "game_table
dropTable "Game_data "running_sum
localmake "fieldlist ask "map_data [db.fieldnames]
localmake "fieldlist butfirst :fieldlist
localmake "fieldlist butfirst :fieldlist
localmake "fieldlist ExcludeFieldsFromList(:fieldlist)
ask "Game_data [(db.addfield "game_table " Description "string )]
addTableFields "Game_data "game_table :fieldlist
addTableFields "Game_data "running_sum :fieldlist
end



to updateRunningSum
ask "Game_data [
localmake "recordexist (db.recordcount "running_sum)
```

```
print :recordexist
if :recordexist = 0 [(db.addrecord "running_sum)]
localmake "fieldlist (db.fieldnames "running_sum)
repeat length :fieldlist [
        localmake "curfield item repcount :fieldlist
        localmake "cursum (db.dsum "game_table :curfield)
        print :curfield
        (db.setcell "running_sum 1 :curfield :cursum)
        ]
]
end


to ExcludeFieldsFromList :fieldnamelist ;finds and excludes fields with specific name pattern
localmake "fieldnamelist2 []
repeat length :fieldnamelist [
  localmake "curfieldname item repcount :fieldnamelist
  localmake "char1 item 1 :curfieldname
  localmake "char2 item 2 :curfieldname
  if  not (and :char1="# :char2="_ ) [
     localmake "fieldnamelist2 lput :curfieldname :fieldnamelist2
   ]
]
output :fieldnamelist2
end
```

Notes

• *The «SpinEndGame» component is updated by the ending reason of the game and exists for future use.*

• *For the transition from the "design form" to the "game form" is necessary to click the"Create" button to clean the previous game data. Nonetheless, the direct access to the "game" is possible through the corresponding "view" menu [Microworld]> [Views: view_game_with_logging]*

• *After every change to the LOGO code or when inserting new procedures the definition of them is required: Therefore, marking the whole procedure and pressing either the [INSERT] button or the "Run" button*  *on the toolbar of the LOGO component is needed.*

• *The “print” command is used on the code, at various points, for testing the function, and displays its results in the middle part of the LOGO component window.*