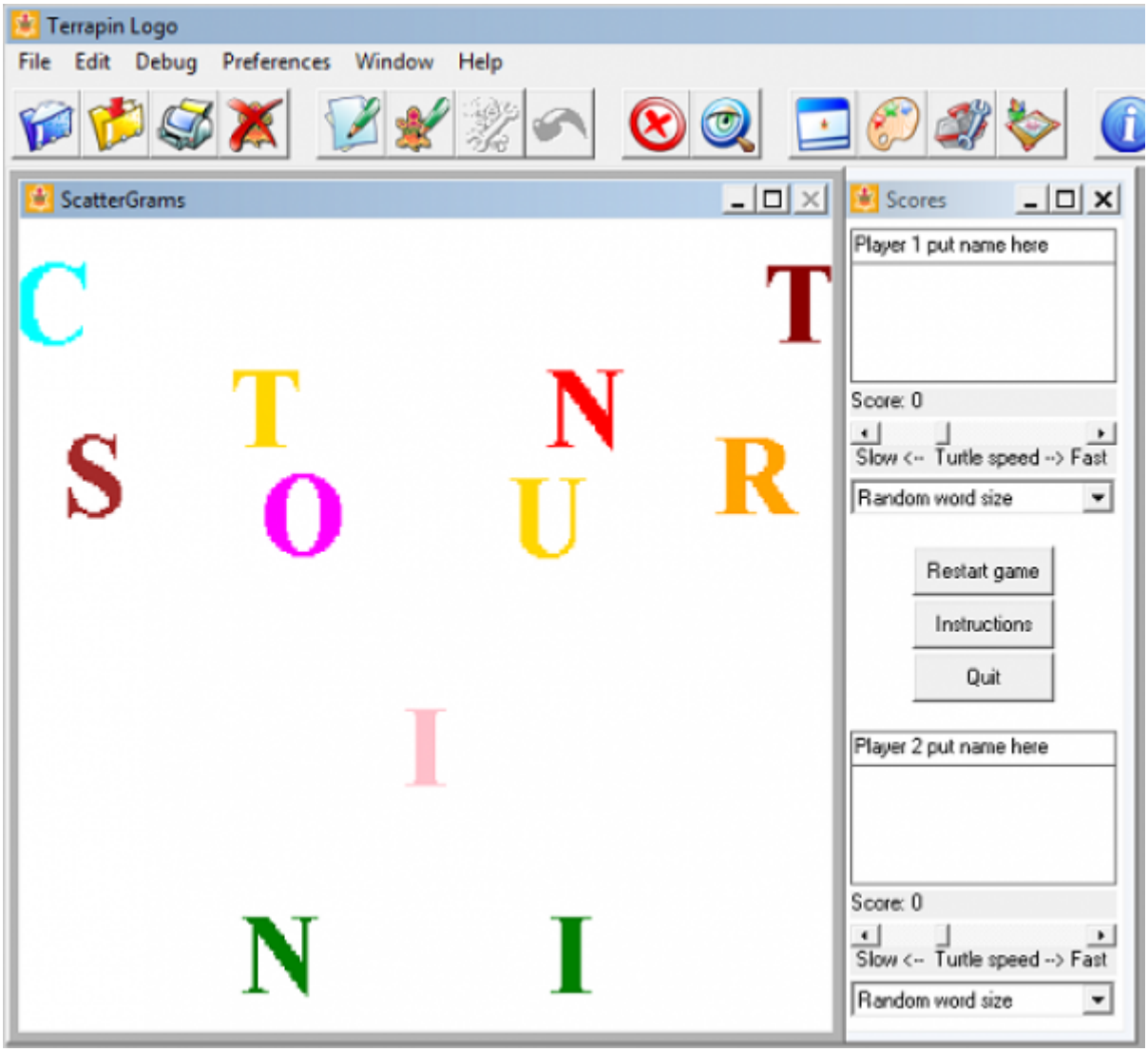
ScatterGrams

ScatterGrams is a two-player game of scrambled words. It is fully described in the Terrapin Logo Tutorial Manual.




```
; ScatterGrams
;
; A game of jumbled words in which two players
; compete for the highest score. Players can
; control the word size and turtle speed.


; The setup routine is invoked automatically after
; the ScatterGrams.lgo file is loaded into the
; workspace. The WAIT statement causes the
; Listener is 'sleep' in that it will not respond
; to any keystrokes made by the players during
; the game.
;
; It would seem like nothing could run with the
; Listener asleep, but since the game is actually
; started with a click of a button, this is not
; a problem.
```

```
to setup

; Differences between Logo 3 and Logo 4;

    make "this.version 3 ; default since that's where it started

    if member? "|Version 4| version make "this.version 4 ; adjustments are needed

    setup.screen
    setup.globals
    demo.screen
    pprop "scores "visible "true
    wait 999999999 ; force Listener to sleep
end


; The setup.screen routine adjusts the Listener
; and Grahpics windows, creates an additional
; graphic window, and them places the player controls
; in the extra graphic window.


to setup.screen
    command findmenuid "window "|standard window layout|
    wrap


    ; make the Listener invisible


    pprop "Listener "visible "false


    ; setup Graphics screen
    pprop "Graphics "title "|ScatterGrams|
    pprop "Graphics "drawsize [457 457]


    ; get rid of the color picker and the toolbox
    pprop "colorpicker "visible "false
    pprop "toolbox "visible "false


    ; setup the extra graphics screen for player controls
    ; initally invisible until all controls are placed
    declare "graphics "scores
    pprop "scores "visible "false
    pprop "scores "drawsize [157 457]
    pprop "scores "position [465 0]
    pprop "scores "title "|Scores|


    ; setup player 1 controls
    declare "editbox "name.1
    pprop "name.1 "size [150 20]
    pprop "name.1 "position [0 213]
    pprop "name.1 "text "|Player 1 put name here|


    declare "listbox "word.list.1
    pprop "word.list.1 "size [150 68]
    pprop "word.list.1 "position [0 170]
```

```
repeat 3 [ask "word.list.1 [remove 0]]


declare "statictext "score.1
pprop "score.1 "size [150 16]
pprop "score.1 "position [0 125]
pprop "score.1 "text "|Score: 0|


declare "scrollbar "speed.1
pprop "speed.1 "size [150 13]
pprop "speed.1 "position [0 108]
if :this.version = 3 [pprop "speed.1 "minimum 20][pprop "speed.1 "minimum 100]
if :this.version = 3 [pprop "speed.1 "maximum 200][pprop "speed.1 "maximum 800]
if :this.version = 3 [pprop "speed.1 "value 100][pprop "speed.1 "value 300]


declare "statictext "speednote.1
pprop "speednote.1 "size [150 16]
pprop "speednote.1 "position [0 93]
pprop "speednote.1 "text "| Slow <--  Turtle speed    --> Fast|


declare "popup "word.size.1
pprop "word.size.1 "size [150 21]
pprop "word.size.1 "position [0 72]
repeat 3 [ask "word.size.1 [remove 0]]
ignore ask "word.size.1 [append "|Random word size|]
ignore ask "word.size.1 [append "|3-letter words|]
ignore ask "word.size.1 [append "|4-letter words|]
ignore ask "word.size.1 [append "|5-letter words|]
ignore ask "word.size.1 [append "|6-letter words|]
ignore ask "word.size.1 [append "|7-letter words|]
ignore ask "word.size.1 [append "|8-letter words|]
ignore ask "word.size.1 [append "|9-letter words|]
ignore ask "word.size.1 [append "|10-letter words|]
ignore ask "word.size.1 [append "|11-letter words|]
ignore ask "word.size.1 [append "|12-letter words|]
pprop "word.size.1 "index 0


; setup player 2 controls
declare "editbox "name.2
pprop "name.2 "size [150 20]
pprop "name.2 "position [0 -70]
pprop "name.2 "text "|Player 2 put name here|


declare "listbox "word.list.2
pprop "word.list.2 "size [150 68]
pprop "word.list.2 "position [0 -113]
repeat 3 [ask "word.list.2 [remove 0]]


declare "statictext "score.2
pprop "score.2 "size [150 16]
pprop "score.2 "position [0 -158]
pprop "score.2 "text "|Score: 0|


declare "scrollbar "speed.2
pprop "speed.2 "size [150 13]
```

```
    pprop "speed.2 "position [0 -175]
    if :this.version = 3 [pprop "speed.2 "minimum 20][pprop "speed.2 "minimum 100]
    if :this.version = 3 [pprop "speed.2 "maximum 200][pprop "speed.2 "maximum 800]
    if :this.version = 3 [pprop "speed.2 "value 100][pprop "speed.2 "value 300]

    declare "statictext "speednote.2
    pprop "speednote.2 "size [150 16]
    pprop "speednote.2 "position [0 -190]
    pprop "speednote.2 "text "| Slow <--  Turtle speed    --> Fast|


    declare "popup "word.size.2
    pprop "word.size.2 "size [150 21]
    pprop "word.size.2 "position [0 -211]
    repeat 3 [ask "word.size.2 [remove 0]]
    ignore ask "word.size.2 [append "|Random word size|]
    ignore ask "word.size.2 [append "|3-letter words|]
    ignore ask "word.size.2 [append "|4-letter words|]
    ignore ask "word.size.2 [append "|5-letter words|]
    ignore ask "word.size.2 [append "|6-letter words|]
    ignore ask "word.size.2 [append "|7-letter words|]
    ignore ask "word.size.2 [append "|8-letter words|]
    ignore ask "word.size.2 [append "|9-letter words|]
    ignore ask "word.size.2 [append "|10-letter words|]
    ignore ask "word.size.2 [append "|11-letter words|]
    ignore ask "word.size.2 [append "|12-letter words|]
    pprop "word.size.2 "index 0


    ; place buttons initially disabled
    declare "button "play.button
    pprop "play.button "position [0 30]
    pprop "play.button "text "|Play|
    pprop "play.button "run [make "process.id launch [play.game]]
    pprop "play.button "enabled "false


    declare "button "instruction.button
    pprop "instruction.button "position [0 0]
    pprop "instruction.button "text "|Instructions|
    pprop "instruction.button "run [show.instructions]
    pprop "instruction.button "enabled "false


    declare "button "quit.button
    pprop "quit.button "position [0 -30]
    pprop "quit.button "text "|Quit|
    pprop "quit.button "run [quit.routine]
    pprop "quit.button "enabled "false


    ; now make the controls visible
    pprop "scores "visible "true
end


; The demo.screen routine in invoked by setup
; and sets up SCATTERGRAMS as the secret word
; which is then automatically solved.


to demo.screen
```

```
    make "secret "scattergrams
    make "scattered jumble :secret
    make "solved.x :solved.x.anchor - 32 * (count :secret) / 2
    assign.turtles :scattered
    wait 3000
    auto.solve :secret
    pprop "play.button "enabled "true
    pprop "instruction.button "enabled "true
    pprop "quit.button "enabled "true
end


; The assign.turtles routine creates the turtles needed
; for the current secret word which has been jumbled and
; passed as an input. Turtle.1 gets the first letter,
; Turtle.2 gets the second letter, and so on. Turtle.0
; is not used because I wanted the letter position and
; the turtle number to be the same. The turtles are made
; 50% larger than normal with SETTS (set turtle size).
; I make sure the turtles are showing (ST), their pens
; are up (drawing lines here would be messy), and each
; is assigned a random color (PICK :MY.COLORS). I set the
; turtles' SHAPELOCK attribute so the letters would always
; be upright and easier to read. Turtle headings are set
; using a formula similar to drawing a polygon. I decided
; that this symmetry was the best way to go even though the
; game is called ScatterGrams; a random heading assignment
; could have given multiple turtles the same heading which
; would have resulted in overlapped letters. Turtles are
; set in motion (SETVELOCITY) based on the setting of the
; current player's speed control.
;
; NOTE: Some of these commands had to been done with EACH
;       so that each individual turtle exectues the command
;       for itself. For example, if PICK :MY.COLORS was not
;       done inside EACH, all the turtles would end up the
;       same color. Other commands are applied to all active
;       turtles at the same time and in the same way.


to assign.turtles :word
    setactivewindow "graphics
    setturtles 1 + count :word
    ask 0 [ht]
    tellall 1 count :word
    each [setshape (word "~ (item who :word) ".bmp)]
    setts 1.5
    each [showturtle penup setpc pick :my.colors]
    each [pprop who "shapelock "true]
    each [setheading who * (360 / count :word)]
    setvelocity current.speed
end


; The auto.solve routine steps through the SCATTERGRAMS word
; and calls the disable routine for each letter.


to auto.solve :word
    if empty? :word [stop]
    disable first :word
    auto.solve butfirst :word
```

```
end


; The disable routine is called when a correct letter
; has been typed. The position number of the letter in
; the :scattered variable is the number of the turtle
; to work with. The turtle size is reset to normal, the
; speed is set to zero, the turtle is positioned at the
; bottom of the screen, a random musical note is played,
; and the who list is updated. Once a letter has been
; guessed and the corresponding turtle's speed has been set
; to zero, I don't want that turtle included in the list of
; currently active turtles. Also, the :scattered
; variable is updated by replacing the letter with a
; hyphen so that this letter will not be used again and
; also to keep the other letters in their original
; positions (remember that letter positions correspond
; to turtle numbers).


to disable :letter
    (local "turtle "current.who)
    make "current.who .who
    make "turtle find :letter :scattered
    make "scattered mark :letter :scattered
    if :this.version = 3 [pprop :turtle "velocity 0][pprop :turtle "crawl 1]
    pprop :turtle "size [31 31]
    pprop :turtle "position solved.position
    play list pick [o3 o4 o5] word 16 pick [a b c d e f g]
    make "current.who butmember :turtle :current.who
    tell :current.who
end


; The find routine outputs the position number of
; the first occurrence of :letter in its input word.
; It does not matter if the word has duplicate letters.


to find :letter :word
    if :letter = first :word [output 1]
    output 1 + find :letter butfirst :word
end


; The mark routine substitutes a hyphen in place of
; the first occurrence of :letter in its input word.
; It does not matter if the word has duplicate letters.


to mark :letter :word
    if :letter = first :word [output word "- butfirst :word]
    output word first :word mark :letter butfirst :word
end


; The solved.position routine outputs a list containing the
; x and y coordinates of where a turtle will be placed after
; its letter has been guessed. The :solved.x variable is
; updated for the next position.
```

```
to solved.position
    local "temp
    make "temp list :solved.x :solved.y
    make "solved.x :solved.x + :solved.x.inc
    output :temp
end


; The setup.globals procedure assigns values to variables that may be used
; by many procedures. I know there have been discussions about whether or
; not globals should be used at all, but I think this is a reasonable thing
; to do for this game. I felt it was simpler to have these global variables
; accessible to any procedure without worrying about remembering to pass them
; as parameters.


to setup.globals
    make "word.list.3 [cat hat bat sat dog log hog the him her his our tot rot got too
\
    sit mop man see son sun set hop one pun pin tin lot fun fan sap zip hip ray dig big
\
    fig rig pig red bed run nun win sip tie two cry sin fin can jam ran bit fit bee cow
\
    set pet wet bug rug hug put nut bag gag rag tag lip car jar far pen hen ask toe arm
\
    egg fly dry why try low sew mix six fix]


    make "word.list.4 [colt that what logo pogo good look mint book them ruin fail safe
\
    tree said bolt yell blue pink show from prom fast past buzz nest door best vein
part \
    rest cook look most bake rake lake fish dish wish hand sand band land crab bear
deaf \
    drip fall tall ball call wall base case vase dime tell bell vest test nail rail
sail \
    cart beam seam pail tail task doll pill hill will bill fill pair hair nose knee
gate \
    foot tear pear show fear skin skip clip bird word moon lamp fame ring sing]


    make "word.list.5 [print diary space place pizza salad green crime fight might
light \
    sight today mouse house cream dream flask earth still photo chair thumb elbow ankle
\
    wrist waist mouth grape melon apple peach lemon brush heard jewel equal angel fruit
\
    plant paint sheet shirt pants socks radio world great guest fling thing heart blood
\
    uncle horse nurse floor clock throw three eight cover]


    make "word.list.6 [marine people rewind legend career hockey museum artist sister \
    stereo finger orange yellow cherry banana breeze mirror branch cheese planet flower
\
    remote pillow bureau sermon church scream drawer hangar people mother father nephew
\
    doctor candle basket tissue saucer temper health]


    make "word.list.7 [receive vacuums beatles picture pitcher printer spinach survive
\
```

```
    bundles players between buttons physics]


    make "word.list.8 [receiver headache comedian keyboard baseball football function \
    macaroni terrapin reaching addition whenever]


    make "word.list.9 [comedians macintosh telephone procedure spaghetti astronaut \
    cartesian agreement functions geography chemistry implement workspace saxophone]


    make "word.list.10 [astronauts procedures microscope background winchester \
    characters television earthquake beekeepers]


    make "word.list.11 [interactive approximate evaporation radioactive vaccination \
    information programming mathematics instruction illustrator entertainer
hummingbird]


    make "word.list.12 [hummingbirds neighborhood entertainers condensation
subprocedure \
    approximates instructions experimental encyclopedia]


    make "min.word.size 3
    make "max.word.size 12
    make "already.used []
    make "solved.x.anchor 0
    make "solved.y -200
    make "solved.x.inc 32
    make "scattered "
    make "current.player 1

; To make this work with Logo 3 and Logo 4, I switched the colors to RGB triplets to
; avoid color name errors.
; It was: make "my.colors butmember "white bm "yellow bm "silver bm "gray colors

    make "my.colors [[0 0 0] [139 0 0] [255 0 0] [255 165 0] [0 255 0] [0 255 255] [0
128 0] [0 0 255] [0 0 128] [255 0 255] [128 0 128] [165 42 42] [255 192 203] [255 215
0]]

    make "total.score.1 0
    make "total.score.2 0
    make "words.per.game 5
end


; The setup.replay routine is invoked by either the 'Restart game' button
; or the 'Play again' button (actually they are the same button but the
; TEXT attribute has been changed). Certain global variables need to be
; reset and then the play.game procedure is launched to start a new game.


to setup.replay
    if :this.version = 3 [(halt :process.id)][halt :process.id]
    repeat 5 [ask "word.list.1 [remove 0]]
    repeat 5 [ask "word.list.2 [remove 0]]
    pprop "score.1 "text 0
    pprop "score.2 "text 0
    make "total.score.1 0
    make "total.score.2 0
```

```
    make "already.used []
    tell every "turtle
    hideturtle
    wait 1000
    make "process.id launch [play.game]
end


; The unique.random procedure selects a new random word for each round.
; Words that have already been chosen are kept in the :already.used
; list so they will not be chosen again for the current game.


to unique.random :word.list
    local "selection
    make "selection item (random count :word.list) :word.list
    if member? :selection :already.used [output unique.random :word.list]
    make "already.used fput :selection :already.used
    output :selection
end


; The jumble, jumble1 and remove.item procedures work together to
; mix up the letters of a word.


to jumble :word
    if (count :word) < 2 [output :word]
    output jumble1 :word random count :word
end


to jumble1 :word :n
    if (count :word) < 2 [output :word]
    output word (item :n :word) (jumble1 (remove.item :n :word) (random ((count :word)
- 1)))
end


to remove.item :n :word
    if :n < 2 [output butfirst :word]
    output word (first :word) (remove.item (:n - 1) (butfirst :word))
end


; The current.word.list procedure was made up as a shortcut for
; referencing the list of words to pick a word from. Notice that
; it uses the current.word.size procedure which outputs a number
; based on the current player's word size control setting.


to current.word.list
    output thing word "word.list. current.word.size
end


; The current.word.size procedure uses the index of the currently
; selected item in the current player's word size control to
; determine the word size to use for this round. The first item
; in the word size control is index 0 which contains the phrase
; 'Random word size.' Index 1 contains '3-letter words, Index 2
; contains '4-letter words' and so on.
```

```
to current.word.size
    local "index
    make "index gprop (word "word.size. :current.player) "index
    if :index = 0 \
        [output random.between :min.word.size :max.word.size] \
        [output 2 + :index]
end


; The current.speed procedure outputs the value of the
; current player's speed control.


to current.speed
    output round gprop (word "speed. :current.player) "value
end


; The random.between procedure outputs a random number
; that is between the starting value :a and the ending
; value :b.


to random.between :a :b
    output (:a - 1) + random (:b - :a + 1)
end


to show.instructions
    if empty? gprop "Instructions "visible [create.instructions]
    make "standard.output gprop "Instructions "channel
    setactivewindow "Instructions
    print "~C
    print "|Welcome to ScatterGrams!|
    print "
    print "|It's a fun game of jumbled words and letters.|
    print "
    print "|The object is to type the word more quickly and|
    print "|accurately than your opponent to score more|
    print "|points.|
    print "
    print "|You can control the word size and the turtle speed|
    print "|but keep in mind that longer words and faster turtles|
    print "|usually mean a higher score.|
    print "
    print "|So, fill in your names, make your selections, and|
    print "|press PLAY to begin the fun.|
end


to create.instructions
    declare "output "Instructions
    pprop "Instructions "position [0 0]
    pprop "Instructions "drawsize [457 457]
end


; The play.game routine is the real starting point of a game. It
; uses the subprocedures play.rounds, play.round, play.word and
; play.letter. Notice the changes that are made to the play button.
```

```
; When ScatterGrams is first loaded into the workspace, this button
; is labeled 'Play'. Once a game has started, the button's text is
; changed to 'Restart game' and the RUN attribute is changed to
; invoke the setup.replay procedure.


to play.game
    setactivewindow "Graphics
    pprop "play.button "run [setup.replay]
    pprop "play.button "text "|Restart game|
    play.rounds :words.per.game
    wait 1000
    game.done
end


; The play.rounds procedure is essentially a loop that
; counts down the number of rounds (defined as 5 in
; setup.globals). So, when it gets down to 1, it is
; really at the last round of the game. If player 2 is
; already ahead in the score by the last round, then
; there is no need to play this round. (Just like in
; baseball when the home team is ahead in the last
; inning.)


to play.rounds :rounds
    if :rounds = 0 [stop]
    make "current.player 1
    play.round
    if and (:rounds = 1) (:total.score.2 > :total.score.1) [stop]
    make "current.player 2
    play.round
    play.rounds :rounds - 1
end


; The play.round procedure selects the secret word for
; this round, resets the location for the solved letters,
; assigns the turtles, resets the typo counter, gets the
; starting time, clears any buffered keystrokes, and
; then invokes play.word.
;
; NOTE: The RUN attribute of the current player's speed
;       control is changed here to allow just this
;       player to affect the turtle speed.


to play.round
    get.ready
    make "secret unique.random current.word.list
    make "scattered jumble :secret
    make "solved.x :solved.x.anchor - 32 * (count :secret) / 2
    assign.turtles :scattered
    pprop word "speed. :current.player "run [setvelocity current.speed]
    make "typos 0
    make "time.0 time
    play.word :secret
end


; The play.word procedure is just a loop that invokes
```

```
; play.letter for each letter of its input word. After
; all the letters have been played, it invokes the
; word.finish procedure and stops.


to play.word :word
    if empty? :word [word.finish (discard 0) stop]
    play.letter first :word
    play.word butfirst :word
end


; The play.letter procedure is where most of the
; games time is spent. This procedure reads the
; keyboard input and compares it to the letter
; given as input. When a match is found, the
; corresponding turtle is disabled and this
; procedure stops. Any other input is a typo.


to play.letter :letter
    while [not equal? :letter (uppercase char getbyte)] [
        make "typos :typos + 1
        play [m10 n56 r]
    ]
    disable :letter
end

; The word.finish procedure gets the end time, updates the
; current player's word list with the word that was just
; played along with its score, updates the current player's
; total score, and clears the RUN attribute of the current
; player's speed control.


to word.finish
    local "put.word
    make "time.1 time
    make "put.word parse (word "append char 32 char 34 char 124 lowercase :secret char
32 char 40 word.score char 41 char 124)
    ignore ask word "word.list. :current.player :put.word
    update (word "total.score. :current.player) word.score
    pprop (word "score. :current.player) "text (word "|Score: | (thing word
"total.score. :current.player))
    pprop word "speed. :current.player "run []
end


; The word.score procedure compute the score of the word just played
; based on the length of the word, the number of seconds it took for
; the player to solve the word, the number of typos that were made,
; and the speed at which the turtles were moving.
;
; NOTE: If you want adjust the scoring, this is where to do it.
to word.score
    (local "start.time "end.time "time.penalty "points.per.letter "temp.score)
    make "start.time (3600 * first :time.0) + (60 * first bf :time.0) + last :time.0
    make "end.time (3600 * first :time.1) + (60 * first bf :time.1) + last :time.1
    make "time.penalty :end.time - :start.time - count :secret
    make "points.per.letter ((current.speed / 10) * (:max.word.size / count :secret))
    make "temp.score round (:points.per.letter * ((count :secret) - (:typos / 2)) -
:time.penalty)
```

```
    if :temp.score < 0 [output 0]
    output :temp.score
end


; The game.done procedure determines the winner (or a tie) and
; uses the winner's name (or 'TIE GAME') as the secret word
; which it then automatically solves. Once the game is
; finished, the 'Restart game' button is changed to 'Play again'.


to game.done
    (discard 0)
    make "secret get.winner
    make "scattered jumble :secret
    make "solved.x :solved.x.anchor - 32 * (count :secret) / 2
    assign.turtles :scattered
    if :this.version = 3 [setvelocity 100][setvelocity 200]
    wait 3000
    auto.solve :secret
    tellall 1 count :secret
    setheading 0
    if :this.version = 3 [setvelocity 60][setvelocity 100]
    pprop "play.button "text "|Play again|
end


; The get.winner procedure output either the name of
; the winning player or 'TIE GAME' if it's a tie score.


to get.winner
    if :total.score.1 = :total.score.2 [output "|tie game|]
    if :total.score.1 > :total.score.2 [output get.name 1]
    output get.name 2
end


; The get.name procedure uses format.name and displayable? to
; make up the name of the winning player. There is a limit to
; how many and which characters can be displayed. I only have
; shapes for letters, digits and a blank.


to get.name :player
    local "winner
    make "winner uppercase gprop word "name. :player "text
    if empty? :winner [output word "player :player]
    make "winner format.name :winner :max.word.size
    if empty? :winner [otuput word "player :player]
    output :winner
end


to format.name :name :max
    if empty? :name [output " ]
    if :max = 0 [output " ]
    if displayable? first :name [output word (first :name) (format.name butfirst :name
:max - 1)]
    output format.name butfirst :name :max
end
```

```
to displayable? :letter
    if :letter = char 32 [output "true]
    output member? :letter "abcdefghijklmnopqrstuvwxyz0123456789
end


; The update routine uses indirect references to add one
; thing to another.


to update :item :value
    make :item (thing :item) + :value
end


; The get.ready routine is just a pause while one player turns over the
; control of the keyboard to another player before the round starts.
; I did not want the timer to start until a player was ready to go.


to get.ready
    (discard 0)
    ignore alert (word "|Player | :current.player "| -- Just press Enter when ready|)
end


to quit.routine
    command findmenuid "debug "|restart logo|
end

TO ABOUT
    (LOCAL "LF "PP "SAMPLE.TEXT "P1 "P2 "P3 "P4 "P5 "P6 "P7 "P8 "P9 "P10)
    MAKE "LF CHAR 10
    MAKE "PP WORD :LF :LF

    MAKE "P1 ScatterGrams is a fun game of scrambled words written by Stan Munson.
    MAKE "P2 It is fully explained in the Terrapin Logo Tutorial Manual.

    MAKE "SAMPLE.TEXT (WORD :P1 :PP :P2)
    IGNORE ALERT :SAMPLE.TEXT
END

; Startup routine


setup
```

Procedure:
: setup.screen

Description:
: Two player game of scrambled words.

Level:
: Intermediate

Compatible:
: Logo 3, Logo 4

Tags:
: Game, Controls