4.2 Turtle Graphics

4.2a A Summary of Turtle Graphics Primitives

One of the graphics packages built into HyperGami allows you to decorate nets by using turtle graphics primitives much like those built into Logo implementations. The following Scheme procedures and special forms are available (see most Logo texts for an explanation of, and examples of, turtle graphics):

• `fd` *`steps`* procedure of one numerical argument
Causes the turtle to move forward by the given number of steps in the direction that the turtle is pointing. If the turtle's pen is down, a line will be drawn on the screen as the turtle changes position.

Example:
```
(fd 5)
```
Causes the turtle to move 5 units in its current direction.

• `bk` *`steps`* procedure of one numerical argument
Causes the turtle to move backward by the given number of steps.

• `rt` *`angle`* procedure of one numerical argument
Causes the turtle to turn right (clockwise as viewed from the top) by the given number of degrees.

Example:
```
(rt 90)
```
Causes the turtle to do a "right face", shifting its heading 90 degrees clockwise.

• `lt` *`angle`* procedure of one numerical argument
Causes the turtle to turn left (counterclockwise as viewed from the top).

• `pu` procedure of no arguments
Causes the turtle to "pick up its pen", after which it will not draw lines until the pen is subsequently "put back down."

Example:

(pu)

• pd procedure of no arguments
Causes the turtle to "put down its pen", after which the turtle will draw lines when moved until the pen is subsequently "picked up."

• setpos *x y* procedure of two numerical arguments
Causes the turtle to shift its position (without changing heading) to the point (x, y) on the screen. If the turtle's pen is down, a line is drawn between the turtle's starting and final position.

Example:
(setpos 0 0)
Moves the turtle to the origin.

• getpos procedure of no arguments
When called on no arguments, returns a list of the turtle's current x- and y-coordinates.

• seth *heading* procedure of one numerical argument
Causes the turtle to set its current heading to the desired value. Note that a heading of 0, by Logo tradition, corresponds to due north; 90 to due east; 180 to south; and 270 to west. Values outside of the range 0-360 are allowed, and cause the turtle to change its heading to the given value mod 360. (For instance, heading values of 40, 400, or -320 are all equivalent.)

• geth procedure of no arguments
When called on no arguments, returns the turtle's heading (in the coordinate system described under the seth procedure above).

• center-the-turtle procedure of no arguments
When called on no arguments, moves the turtle to the origin (0, 0), without drawing any lines; and sets the turtle's heading to 0 (due north). Essentially a "turtle reset."

• show-turtle procedure of no arguments
• hide-turtle procedure of no arguments

These procedures, when called on no arguments, render the turtle visible or invisible, respectively. In general, if you are writing programs that cause the turtle to execute many movements—and if you are confident that the program does what you wish—then it is wise to render the turtle invisible, as your programs will run much faster. On the other hand, if you wish to see the turtle moving about the screen, then you should render the turtle visible.

- `repeat` special form

The repeat special form in HyperGami is written to be similar in meaning (though not quite similar in syntax) to Logo's REPEAT form. In HyperGami, repeat is followed by at least two expressions: the first expression should evaluate to a positive integer n, and the remaining expressions denote a sequence of operations to perform n times. Thus, to take a couple of examples:

```
(repeat 4 (fd 20) (rt 90))
```

This causes the turtle to move forward (by 20 steps) and turn right 90 degrees four times in succession. If the turtle's pen is down, this will draw a square.

```
(repeat 6 (fd 10) (rt 60))
```

This causes the turtle to move forward (10 steps) and turn right 60 degrees, 6 times in succession. If the turtle's pen is down, this draws a hexagon.

```
(repeat 6 (repeat 6 (fd 10) (rt 60)) (rt 60))
```

This causes the turtle to generate six hexagons (with 60 degrees between them). That is the turtle repeats the following sequence six times: "draw a hexagon; turn right by 60 degrees."

4.2b Turtle Menu Commands; Turtle Icon in the Paint window.

Pen Up

Pick up the turtle's pen. Equivalent to evaluating `(pu)` at the interpreter.

Pen Down

Put down the turtle's pen. Equivalent to evaluating `(pd)`.

Home
Move the turtle to the origin, drawing a line if the pen is down. Sets the turtle's heading to 0 (north).

Center
Move the turtle to the origin, but avoiding drawing a line regardless of the turtle's current pen state. Sets the turtle's heading to 0 (north). Equivalent to evaluating `(center-the-turtle)` at the interpreter.

Show or Hide Turtle
If the turtle is visible (invisible), renders it invisible (visible).

Turtle Icon in the Paint Window
If the turtle is visible in the TwoD window, you can use the mouse to drag the turtle about. Here's how you do it: select the turtle icon (the little triangle) in the Paint window; then select the turtle itself, in the TwoD window, holding the mouse button down directly over the turtle's center. Now, as long as the mouse button is held down, you can drag the turtle all about the window. When the turtle is positioned where you want it, simply release the mouse button.

Likewise, if you click and hold the mouse button down near the turtle's front tip, you can use the mouse to turn the turtle to the desired heading (the turtle will turn to face the current position of the mouse).

4.3 Defining new colors

4.3a. Color-related primitives in HyperGami
In HyperGami, colors are represented as lists of three nonnegative integers in the range (0, 65535). These three integers represent the red, green, and blue components of the color, respectively. The following Scheme primitives are built into HyperGami for dealing with colors:

• make-color-object *r g b* procedure of three positive integer arguments

Calling make-color-object on three nonnegative integers in the range (0, 65535) produces a color with the appropriate amounts of red, green, and blue, respectively.

Examples:

```
(define brown (make-color-object 30000 20000 10000))
```
This expression creates a color object (which would appear brown if shown on the screen), and gives the name brown to it.

```
(define purple (make-color-object 60000 0 60000))
```
This expression creates a color object named purple with large amounts of red and blue (and no green component).

• `set-color!` *`colorobj`* procedure of one color-object argument

Calling set-color! on a color object as argument sets the pen color (and also the turtle's drawing color) to the given color object.

Example:

```
(set-color! purple)
```
Assuming that the color object purple had been defined as shown earlier, this expression would cause the current pen color (and turtle color) to change to purple. Thus, drawing with the pen (or evaluating turtle commands) immediately after evaluating this expression would cause the pen to draw in purple. Likewise, filling a region (without selecting a new color) after evaluating this expression would cause the region to fill in purple.

• `get-color-red` *`colorobj`*
• `get-color-green` *`colorobj`*
• `get-color-blue` *`colorobj`* procedures of one color-object argument

These procedures return the red (green, blue) component of a color object.

Example:

```
>> (get-color-red purple)
60000
```

Evaluating (get-color-red purple) returns the red component of the color object purple.

• `set-custom-color!` *`colornumber colorobj`*
procedure of one integer (1 or 2) and one color object

This procedure should be called on two arguments: the first argument should be either the integer 1 or 2, and the second argument should be a color object. The result of evaluating this expression will be to set one of the two empty circles in the Paint window to the desired color object (at which point it may be used just like any other color).

Example:

```
(set-custom-color! 1 brown)
```
Evaluating this expression would fill the first of the two blank circles in the Paint window to a brown color. This may now be selected just like any other available color in HyperGami.

• `interpolate-between-colors` *`colorobj1 colorobj2 fraction`*
procedure of two color objects and a number in the
range [0, 1]
This procedure, when called on two color objects and a number in the range [0, 1], will return a new color object calculated by interpolating (in RGB-values) between the first color and the second by the appropriate fraction. Thus,

```
(interpolate-between-colors brown purple 0.5)
```

returns the color object whose R, G, and B values are 45000, 10000, and 35000, respectively. Similarly,

```
(interpolate-between-colors brown purple 1)
```

returns a color object whose R, G, and B values are the same as those of the color object named purple.

4.3b Some Named Colors in HyperGami

The following color names (among others) are predefined in HyperGami:

```
red (60000 0 0))
green (0 60000 0)
blue (0 0 60000)
rgbblack (0 0 0)
rgboffwhite (60000 60000 60000)
yellow (59000 49000 32000)
orange (59500 24500 16000)
lightred (60000 20000 20000)
lightgreen 31000 64000 31000)
lightblue (20000 20000 60000)
darkgrey  (18000 18000 18000)
lightgrey (45000 45000 45000)
reallywhite (65000 65000 65000)
pinkish (60600 32600 25800)
lime (0 64000 34000)
cyan (0 50000 50000)
beige (50150 41650 27200)
```