sources/c64/Main/ -- general Habitat client code

Notable files:

beta.mud -- definition file that is used by the 'Muddle' utility to generate the disk database of class, sound, and image information that describe the various kinds of objects that can exist in the Habitat universe. Pay particular attention to class_avatar (starting on line 51), where it specifies 7 different possible images (body types) for the avatar class.

Note that an "image" in the usage here is not a simple raster image (like a JPEG or PNG or GIF file) but a complex data structure of cels containing animation sequences for different parts of the object (e.g., torso, legs, arms) from different points of view (e.g., front, back, side).

The source code for the Muddle utility itself is in sources/tools/muddle and some documentation is in the file chip/habitat/docs/muddle.t. I can produce a more involved explanation of what this does and how it fits into the overall picture if you require it, but I suspect that may be more detail than your needs call for right now so I will hold off getting into that unless and until you direct me otherwise.

dataequates.m -- defines offsets into object instance state for various properties of various objects. Notable values to pay attention to include

OBJECT_style_pointer (line 34)
-- byte that indicates which of the various image styles available for a given object instance's class is to be used in this particular instance. If the object is an avatar, this corresponds to which body type is to be displayed (i.e., which of the 7 avatar images specified in the beta.mud file to use).

AVATAR_customize (line 55)
-- two bytes of information that encode how the avatar's appearance is customized

AVATAR_contents (line 50)
-- this is the array of objects "contained" by the avatar, most notable is AVATAR_HEAD (defined on line 5 with the value 6), which is the index of the head object that the avatar is wearing. By changing this you change which object is displayed as the avatar's head.

custom.m -- this whole file implements the standalone avatar customization interface, which would normally be invoked automatically when you login for the first time, but could also be triggered by various keyboard actions.

animate.m -- this file implements the (fairly complex) rendering of an avatar, interpreting the body state and head selection. The main routine is display_avatar, starting on line 30. The interpretation of the avatar's custom image and head is extracted from the avatar instance state for usage on lines 40-46.

database.m -- this file manages the resources from the resource database (this includes avatar images, per claim #3). The principal routine to pay attention to is Create_object (line 14), which for any newly created object locates resources that are already loaded (since many objects can share resources) and loads resources not already loaded. The actual accessing of the database happens in load_class (line 238).

sources/c64/Behaviors/ -- implementation of the specific behavior of the
    various different object classes

Source file names in this directory mostly take the form <classname>_<verb>.m
Verbs that are in lower case correspond to user actions in the user interface
(generally the result of key presses or menu selections), whereas verbs that
are in UPPER CASE correspond to messages from the server notifying the object
of some event that was mediated by the server.

Notable files:

avatar_put.m -- The "put" verb in the user interface commands the user's
avatar to put whatever it is holding in its hands in or on whatever the
cursor is pointing at. A verb handler is invoked corresponding to what
the pointed at object is, so avatar_put gets invoked when the user issues
a "put" operation while pointing at an avatar. If the avatar is themselves
and they are holding a head and they don't already have a head on, this
causes the head to become their head, which is the primary avatar
customizatoin operation.

avatar_WEAR.m -- This is asynchronous counterpart of the above, invoked when
the server notifies the client that somebody has put on a new head.

head_get.m -- The "get" verb in the user interface commands the user's avatar
to get whatever object the cursor is pointing at (assuming they are empty
handed). The verb handler is invoked corresponding to what the pointed at
object is, so head_get gets invoked when the user issues the "get"
operation while pointing at a head. If that head is their own head, they
remove it, allowing the now headless avatar to be able to don a different
head.

avatar_REMOVE.m -- This is asynchronous counterpart of the above, invoked
when the server notifies the client that somebody has removed their head.

sex_changer_do.m -- The sex change machine switches an avatar's gender, which
involves altering the image that is used to display the avatar body. The
change is invoked by issuing the "do" operation while pointing at the sex
change machine, which then invokes the code in this file to perform the
operation.

sex_changer_SEXCHANGE.m -- This is the asynchronous counterpart of the above,
invoked when the server notifies the client that somebody has used the sex
change machine.

spray_can_do.m -- The "do" verb in the user interface commands the user's
avatar to do something with the object they are holding or pointing at.
What this means depends on the object in question. The "spray can" object
is used to alter the avatar's appearance by "spraying" one of their body
parts with a different color.

spray_can_SPRAY.m -- This is the asynchronous counterpart of the above,
invoked when the server notifies the client that somebody has sprayed
themselves.

toggle_ghost_mode.m -- This gets invoked when the user presses the key that
switches their avatar between ghost and corporeal modes. Note that there
is no asynchronous counterpart to this: when somebody ghosts, everyone is
notified by being informed that their avatar has disappeared, as if they
had logged off or walked out of the region. When somebody de-ghosts, it is
as if they had logged on or walked into the region.

I recall also that there is also a general mechanism for commanding one’s avatar to use one of the other avatar bodies (penguin, dragon, etc), but in the time available I was not able to find the code that does this. If you want that I’ll need some more time.