THANK you very much DAD!! from Jun Takada

Habitat Technology Transfer Seminar

Lucasfilm --> Fujitsu

Great! [illegible]

FaceSaver 2/88

Nobuo Takahashi
408-432-1300
Fujitsu America Inc.
3055 Orchard Drive
San Jose, CA 95134-2017

HABITAT IS a WONDERFULL GAME. Truly!
K. Fukuda

Thank you, dad.
Toru Miyoshi.
三好 徹.

See you again
Mata ai mashou!
Keiichi Matsunaga
☺

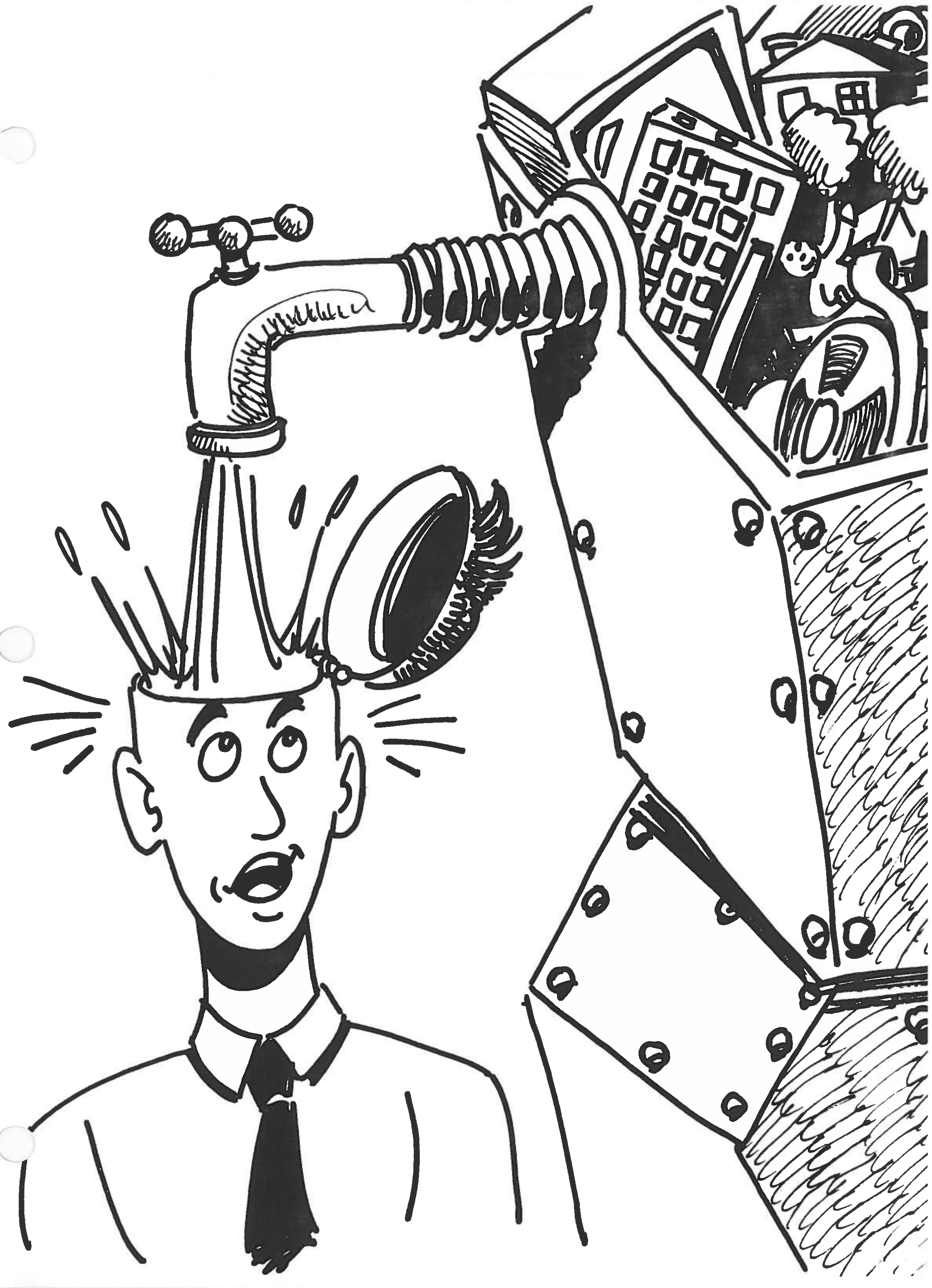
Skywalker Ranch
Marin County, California

Monday, August 21, 1988
through
Friday, August 25, 1988

Thank you!
Hisashi Naka
中恒

ご恩は決して忘れません
中森 博晃

0.cover.1



Lucasfilm ➡ Fujitsu Habitat Technology Transfer Seminar

- Preliminaries
- Overview of Habitat
- Supporting technology
- Habitat technology
- Operations

Lucasfilm ➡ Fujitsu
Habitat Technology Transfer Seminar

- Preliminaries
- *Overview of Habitat*
- *Supporting technology*
- *Habitat technology*
- *Operations*

Preliminaries

- Introductions: everybody meets everybody
- Seminar agenda
- Some notes on our style of presentation

Seminar Agenda

- Preliminaries
- Overview of Habitat
- Supporting technology
  - Commodore 64 system
  - Stratus host system
  - Unix development system
- Habitat technology
  - How the system was specified
  - Description of communications protocol
  - C64 system
  - Host system
  - Development system
  - The Habitat world database
- Operations
  - Support personnel
  - Day-to-day operations
  - Creating new experiences
  - Rules, policies, and laws

Some notes on our style of presentation

- Questions welcome at all times
- Plenty slack time for unanticipated things
- Some general issues to keep in mind:
  - Technical differences: Habitat on Fujitsu 386 vs. Commodore 64
  - Technical differences: Fujitsu host vs. Quantum's Stratus host
  - Cultural differences: Japan Habitat vs. U.S. Habitat

Lucasfilm ➡ Fujitsu
Habitat Technology Transfer Seminar

- *Preliminaries*
- Overview of Habitat
- *Supporting technology*
- *Habitat technology*
- *Operations*

Overview of Habitat

- Multi-player shared universe
- Distributed telecommunications game
- Built around a few fundamental concepts
- Simple and direct user interface

Multi-player shared universe

- An imaginary world...
- ...inhabited by real people...
- ...who enter it via their home computers

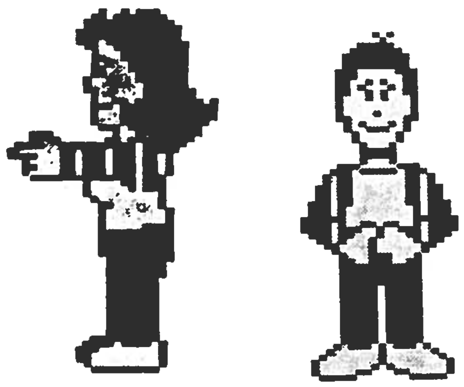
Distributed telecommunications game

- Home PC handles
  - Graphics
  - User interface
  - Data-intensive operations
- Network host handles
  - Large, expanding world database
  - Coordination of multiple players
  - Player-to-player communications

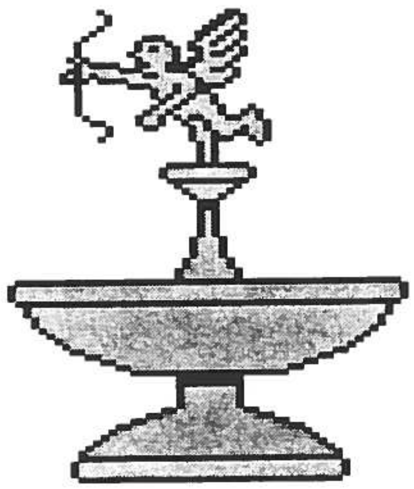
Built around a few fundamental concepts

- Avatars
  - Ghosts
- Objects
- Regions
  - "Turf"
  - Realms
- The Oracle
- Hall of Records

Avatars



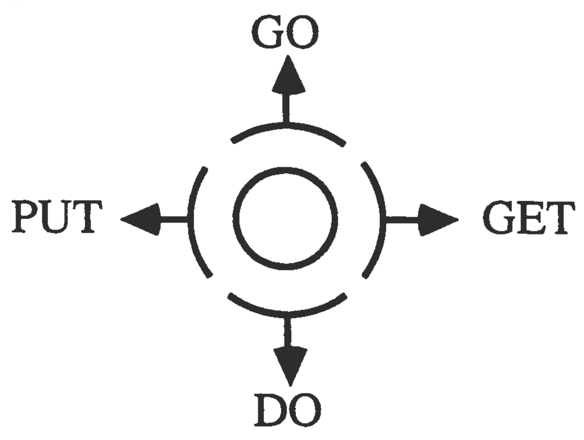
The Oracle



Simple and direct user interface

- Joystick command verbs
- Talking
  ESP
- Gesture keys
- Function keys

Joystick command verbs



Gesture keys

- CTRL-1 — wave hand
- CTRL-2 — point
- CTRL-3 — palms up
- CTRL-4 — jump
- CTRL-5 — face forward
- CTRL-6 — face backward
- CTRL-7 — bend over
- CTRL-8 — stand up
- CTRL-9 — punch
- CTRL-0 — frown

Function keys

- F1 — ghost/de-ghost
- F2 — toggle region transition music
- F3 — list some online Avatars
- F6 — change flesh color
- F7 — get help information
- SHIFT-RUN/STOP — exit Habitat

Overview of Habitat, cont'd.

- Common, important object classes
- Other notable object classes
- The Habitat world
- Live demonstration
- Experiences operating Habitat

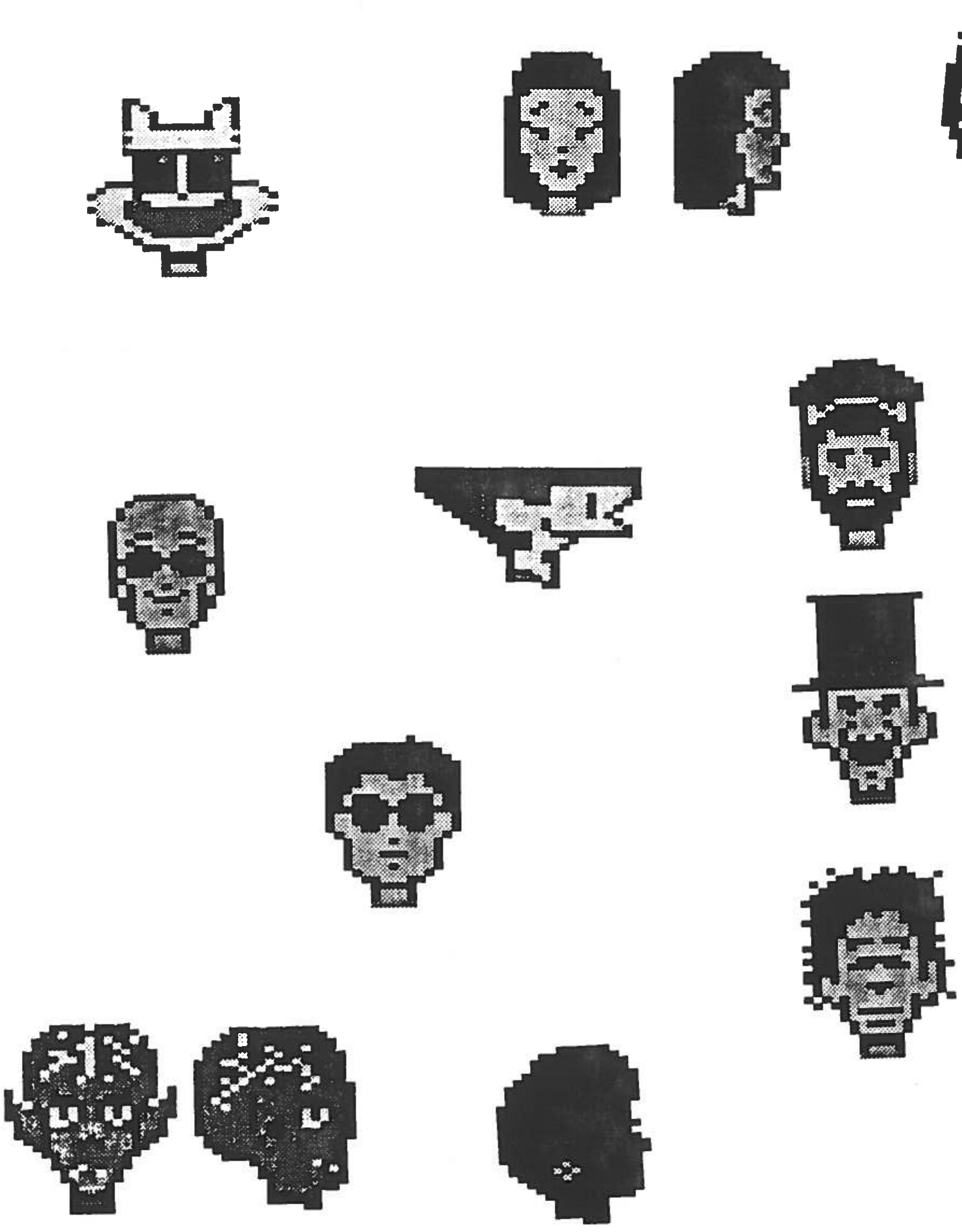
Overview of Habitat, cont'd.

- Common, important object classes
- *Other notable object classes*
- *The Habitat world*
- *Live demonstration*
- *Experiences operating Habitat*

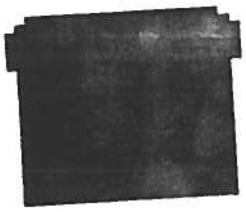
Common, important object classes

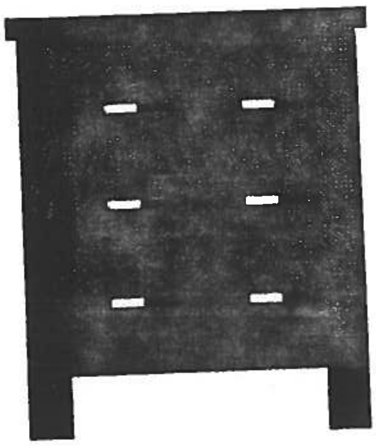
- Avatar
- Heads
- Containers
- Paper
    - Mail
    - Books
- Tokens
- Magic items

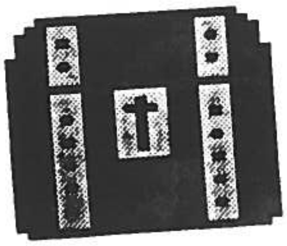
Heads

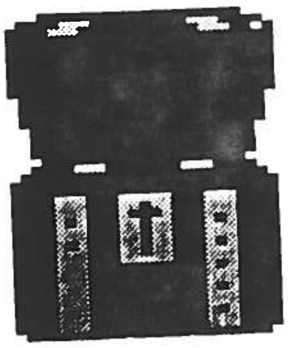


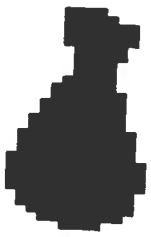
Containers



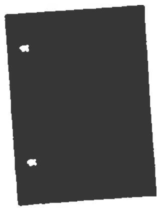






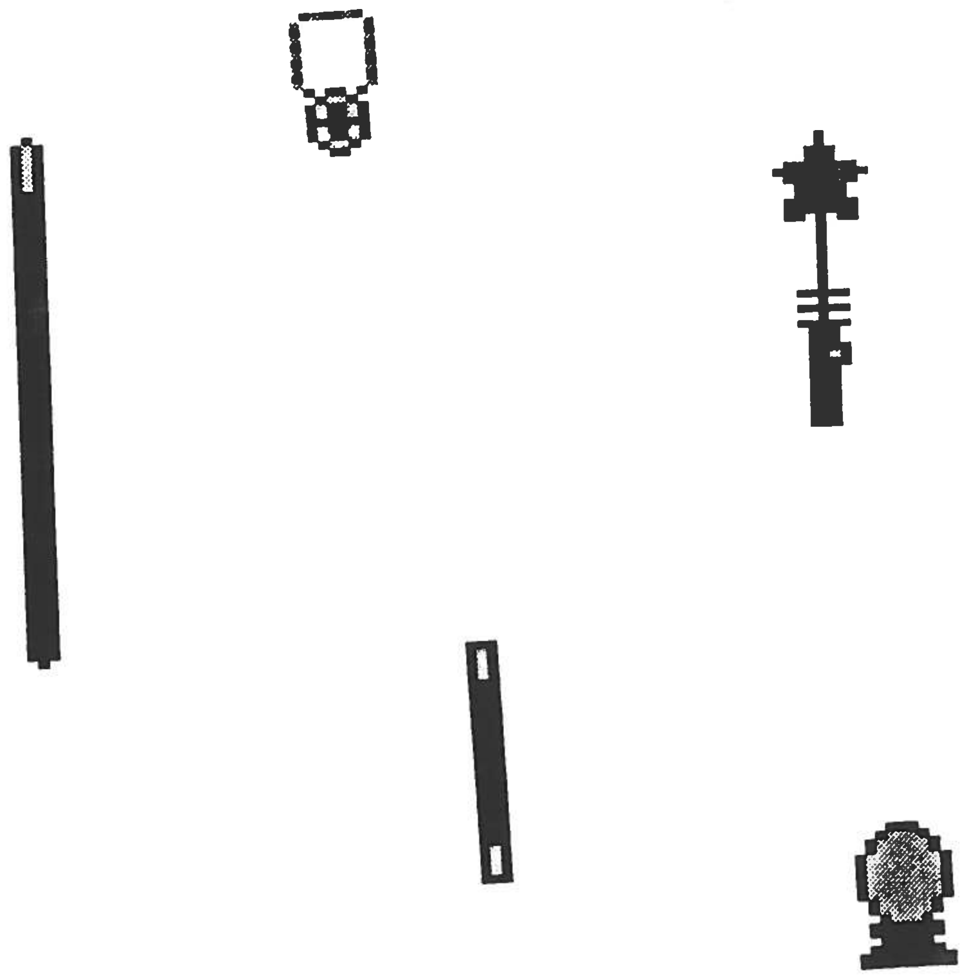


Paper and Books



Tokens

Magic items



Overview of Habitat, cont'd.

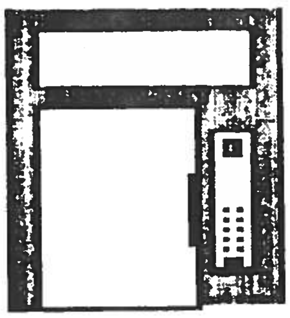
- *Common, important object classes*
- Other notable object classes
- *The Habitat world*
- *Live demonstration*
- *Experiences operating Habitat*

Other notable object classes

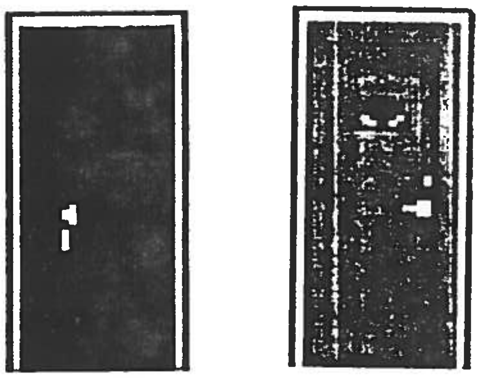
- Teleport
- Vendroid
- Doors
- Escape device
- Fountain
- Scenery
- ATM
- Body Sprayer
- Etc.

Teleport

Vendroid



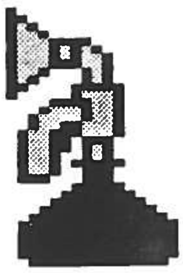
Doors



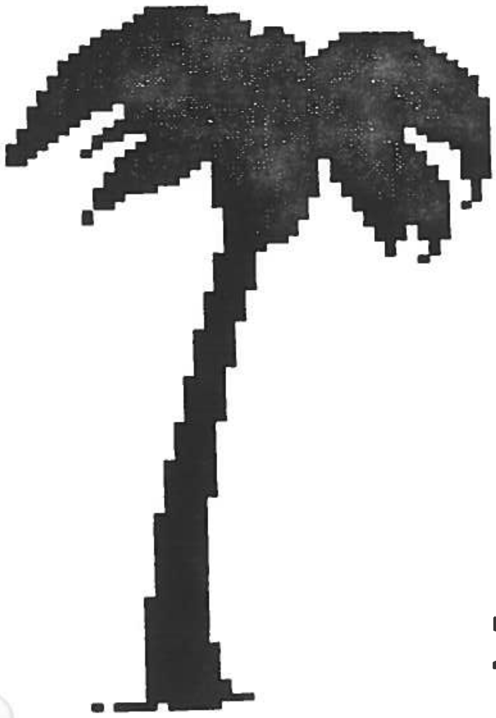
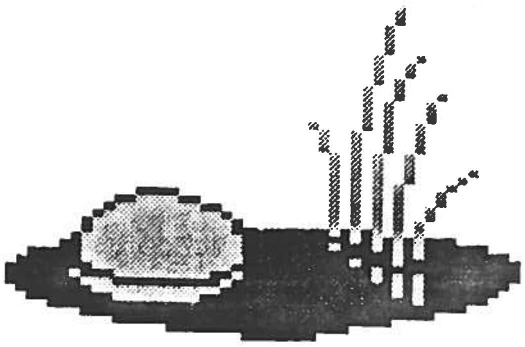
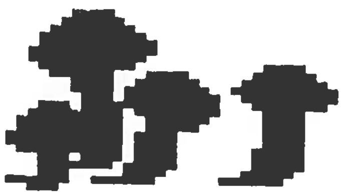
Escape Device

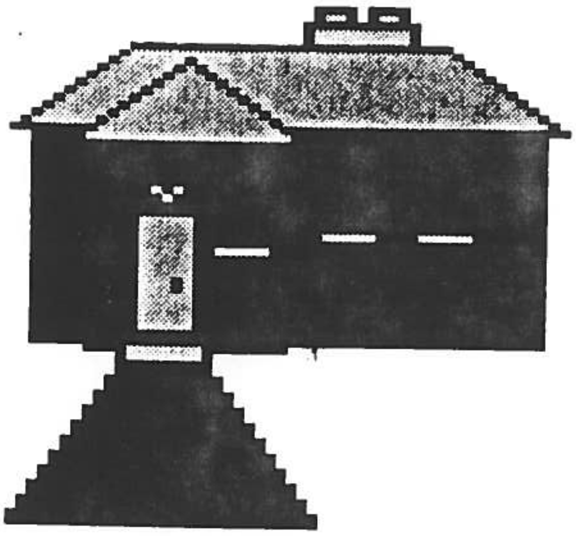
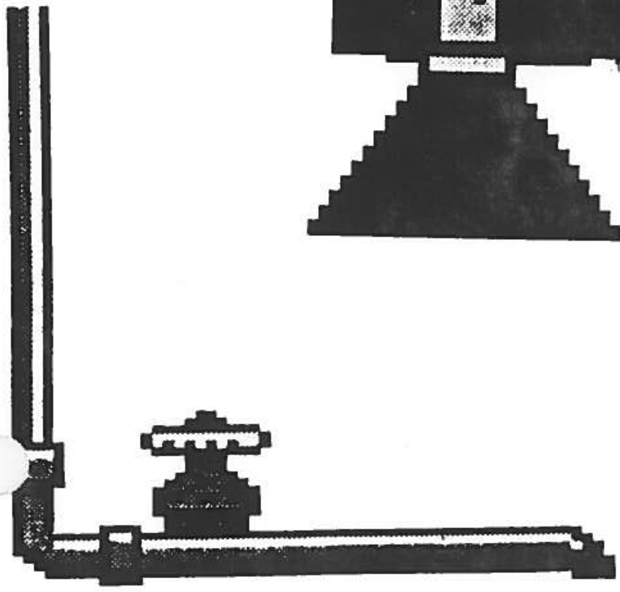
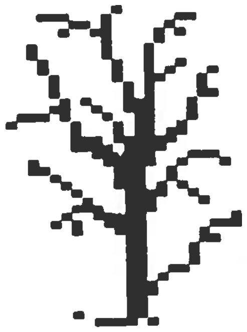
ATM

Body Sprayer



Scenery

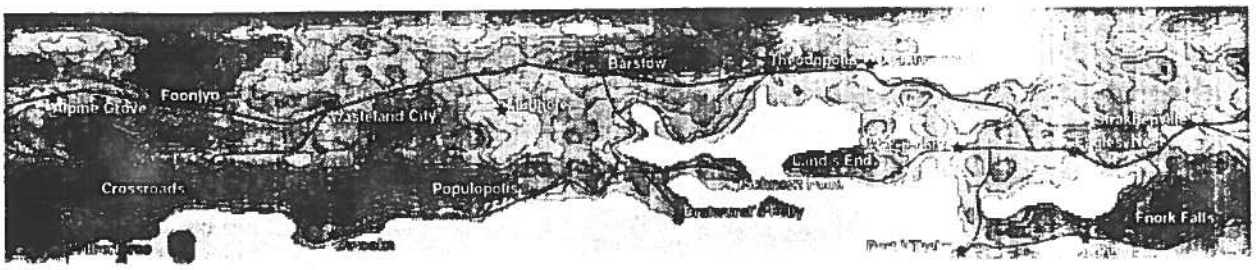
Overview of Habitat, cont'd.

- *Common, important object classes*
- *Other notable object classes*
- The Habitat world
- *Live demonstration*
- *Experiences operating Habitat*

The Habitat world

- The Habitat fantasy
  - Habitat mythos
  - The Oracle
- The world map
  - The geographic master plan
  - Cities, built and planned
    - Populopolis
    - Quantumgrad
    - Other cities
  - Other realms

World Map


Barstow
Theodopolis
Foonlyo
Alpine Grove
Wasteland City
Crossroads
Populopolis
Land's End
Fnork Falls

Overview of Habitat, cont'd.

- *Common, important object classes*
- *Other notable object classes*
- *The Habitat world*
- Live demonstration
- *Experiences operating Habitat*

Live demonstration

Overview of Habitat, cont'd.

- *Common, important object classes*
- *Other notable object classes*
- *The Habitat world*
- *Live demonstration*
- Experiences operating Habitat

Experiences operating Habitat

- Things to do
- Operational experience

Things to do

- Games
- Sports
- Quests
- Exploration
- Parties
- Hanging out
- Drama
- Business
- Politics
- Religion
- Sex
- "Events"

Operational experience

- Alpha test
- Pilot test
- Player reactions
- Funny stories

Lucasfilm ➡ Fujitsu
Habitat Technology Transfer Seminar

- *Preliminaries*
- *Overview of Habitat*
- Supporting technology
- *Habitat technology*
- *Operations*

Supporting technology

- Commodore 64 system
- Stratus host system
- Unix development system

Commodore 64 system

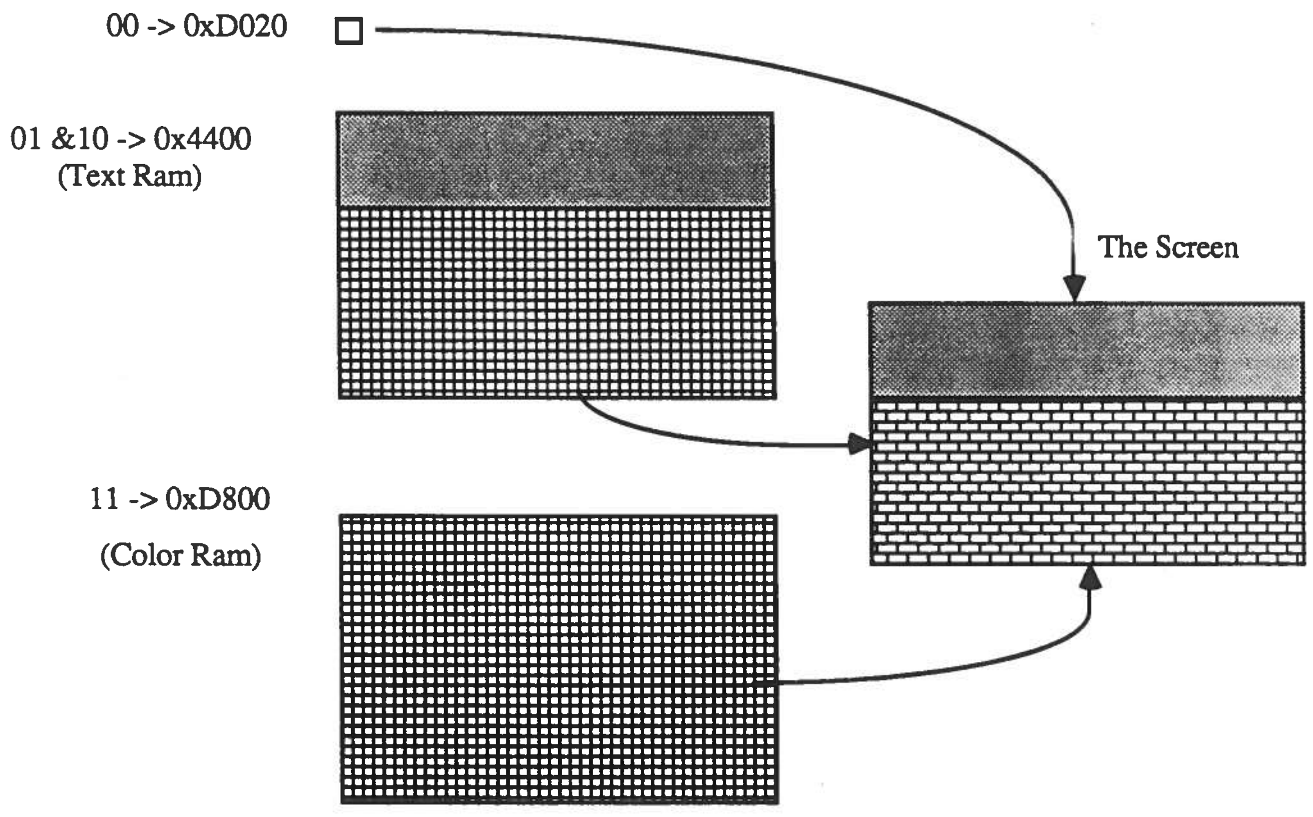
- 6502 microprocessor
- C64 graphics
- C64 disk
- C64 telecommunications
- C64 sound

C64 graphics

- Numerous graphics modes
- We use 160x128x2-bit mode graphics
- Mode switching with scan-line interrupts
- Weird color map

Habitat/C64 Color Map

Color bit pairs 01,10,11 are defined on a CHARACTER cel (4x8 color pixels)



Bit pair 00 -> Background Color Register (0xd020), always BLUE
Bit pair 01 -> Low 8 bits from Text Ram, default 'skin color'
Bit pair 10 -> High 8 bits from Text Ram, default BLACK
Bit pair 11 -> Color Ram, colored when background is rendered

C64 disk

- Slow, slow, slow
- Smart drive, stupid drive software
- Reprogrammable drive
- 160K per disk side
- “Turn the disk over”

C64 telecommunications

- No hardware UART
- Interrupt on every bit
- Port interferes with everything

C64 sound

- ADSR square-wave generator
- 3-channels

Stratus host system

- System overview
- PL/1 programming environment
- Database filesystem
- Interprocess messages

System overview

- High-reliability “non-stop” system
- Transparent multiprocessors
- Based on multiple redundant 68000 family CPUs
- Proprietary operating system VOS

PL/1 programming environment

- PL/1 is the Stratus' "native" language
- Pretty good optimizing compiler
- Rest of QuantumLink written in PL/1

Database filesystem

- Fast indexed record access
- All database I/O provided by system calls
- "A blunt instrument"
  - Never use 1 system call when you can use 3...
  - ...and never give it 3 parameters when you can give it 7

Interprocess messages

- Supported by underlying operating system
- High-speed StrataLINK bus between processor modules
- Up to 256 processor modules in a system

Supporting technology

- *Commodore 64 system*
- *Stratus host system*
- Unix development system

Unix development system

- Unix software tools
  - C
  - yacc
  - lex
  - make
  - etc.
- Macross assembler
- Fastlink
  - Face

yacc, lex, make docs
macross/slinky doc
fastlink doc
face doc

Sample Macross code

```
           do {
               lda     contents_counter
               clc
               adc     #OBJECT_contents
               tay
               lda     y[@object_address]      ; object contained?
               if (!equal) {
                       sta     contained_object

                       ldy     #OBJECT_class_pointer   ; Variable trap object
                       lda     y[@object_address]
                       cmp     #class_glue
                       if (equal) {
                           lda contents_counter
                           asl a
                           clc
                           adc #OBJECT_contents+9      ; location of x/y data
                           tay
                           movew       object_address,prop_address

                       } else {
                           lda contents_counter
                           asl a
                           clc
                           ldy #3                  ; contents_xy table
                           adc y[@prop_address]
                           and #0x7f               ; mask off high bit
                           tay                     ; draw before/after
                       }

                       lda     cel_x_origin
                       ldx     temp_cel_dx     ; facing?
                       if (equal) {
                               clc
                               adc     y[@prop_address]
;;                             adc     last_cel_x_rel
                       } else {
                               sec
                               sbc     y[@prop_address]
;;                             sbc     last_cel_x_rel
                       }
                       sta     cel_x

                       iny
                       lda     y[@prop_address]
                       clc
                       adc     cel_y_origin
;;                     adc     last_cel_y_rel
                       sta     cel_y

                       jsr     draw_contained_object
                       moveb   #1,stand_alone
               }
               dec     contents_counter
           } while (plus)
```

Lucasfilm ➡ Fujitsu
Habitat Technology Transfer Seminar

- *Preliminaries*
- *Overview of Habitat*
- *Supporting technology*
- Habitat technology
- *Operations*

Habitat technology

- How the system was specified
- Description of communications protocol
- C64 system
- Host system
- Development system
- The Habitat world database

Habitat technology

- How the system was specified
- *Description of communications protocol*
- *C64 system*
- *Host system*
- *Development system*
- *The Habitat world database*

How the system was specified

- Implementation constraints
- Resulting limits
- Solutions

Implementation constraints

- The Prime Directive
  "You can never guarantee that the user hasn't hacked the system...
  ...so NEVER trust the PC to tell the truth."
- Limited bandwidth communications channel!
  300 baud/1200 baud
  Packet network running in surreal time
  Existing Q-Link communications protocol
- Commodore 64
- Need low host overhead
  low host overhead = low operations cost

Resulting limits

- Limited number of Avatars per region (6)
- Limited number of objects per region (128)
  - Theoretical maximum of 256 (protocol limit)
  - Limit of 32 unique heads per region ("head hack")
- Rationale
  - Limited by frame rate
  - Limited by communications bandwidth
  - Limited by amount of available memory
- Note that these limits will be higher with 386 system...
  - ...but they will still exist
- More on ghosts

Solutions

- Object/message model
- C64 objects vs. Host objects
- Dynamic heap and object oriented virtual memory
- Very compact message protocol

Habitat technology

- *How the system was specified*
- Description of communications protocol
- *C64 system*
- *Host system*
- *Development system*
- *The Habitat world database*

Description of communications protocol

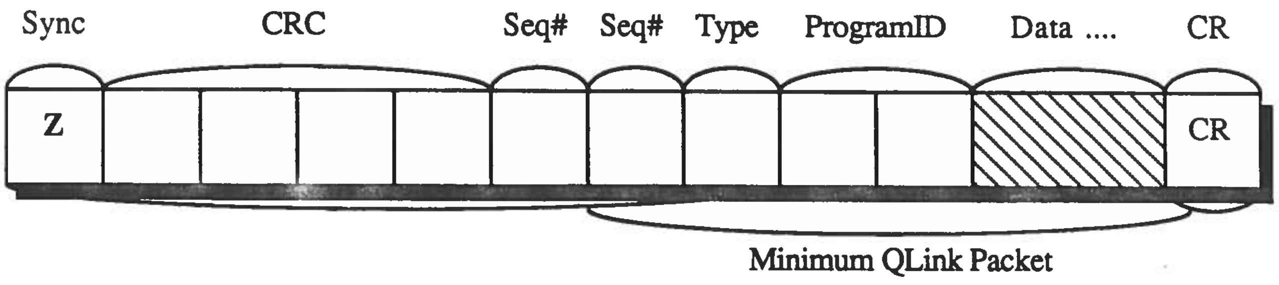
- Q-Link silent protocol
- Problems with Q-Link protocol

Q-Link silent protocol

- Message types
  - SS
  - SSR
  - INIT
  - ACK
  - NAK
  - HEARTBEAT
- NAK-based
- ACK every 8 messages
- Heartbeats

The Q-Link Packet

The Quantumlink Packet looks like this:



The 'SYNC' byte is ASCII 'Z'
The CRC is a CRC-16 that is sent as 4 bytes (see C64 protocol.m for details).
Next comes the transmit and receive sequence numbers; these are ring counters: 0x10-0x7f
The 'Type' is a number 0x20-0x26 representing the message type:

- 0x20 - Application Data
- 0x21 - SS (not used)
- 0x22 - SSR (not used)
- 0x23 - Init (not used)
- 0x24 - Ack
- 0x25 - Nak
- 0x26 - Heartbeat

The Progam ID bytes are used only with message type 0x20. They indicate the application to/from which the message is directed/comes. Habitat recognizes these Program ID codes:

- SS - Suspend Communications until SG
- SG - Start Game (Resume Communications)
- XS, AB - System-wide/Habitat-wide messages
- U? - A Habitat Message

Packets are limited to 127 bytes in length.

Problems with Q-Link protocol

- The Q-Link packet
- Wasted bandwidth and CPU
- Excessive retransmission

Habitat technology

- *How the system was specified*
- *Description of communications protocol*
- C64 system
- *Host system*
- *Development system*
- *The Habitat world database*

C64 system

- “Skeleton” system
- Object behaviors

“Skeleton” system

- Animation engine
- Communications driver
- Heap manager
- Sound effects driver
- User interface
- Interface to object behaviors

Animation engine

- Double buffered
- The foreground bit
- Background rendered once

    Foreground: back to front

    Background: front to back

Fig. 2a - typical object

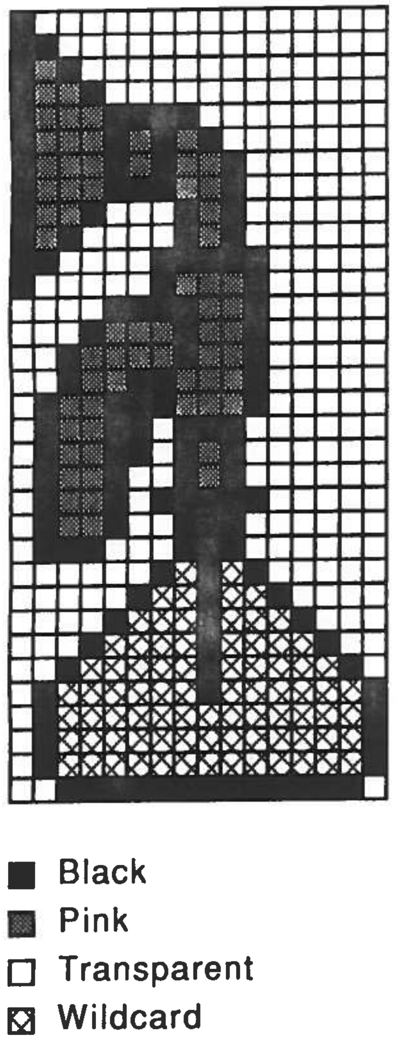


Fig. 2b - random background

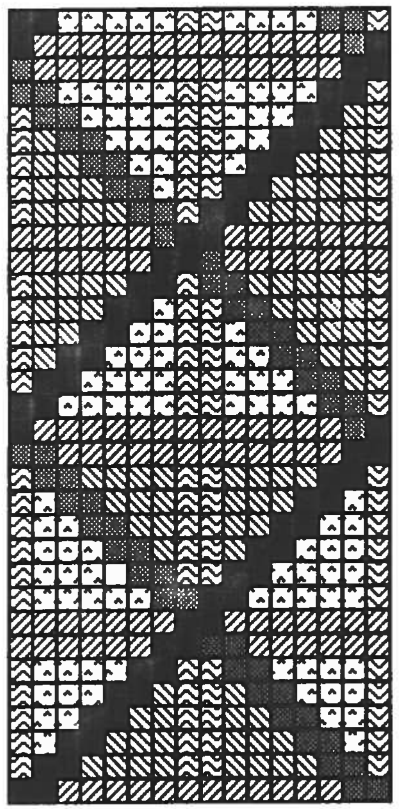


Fig. 2c - composite image

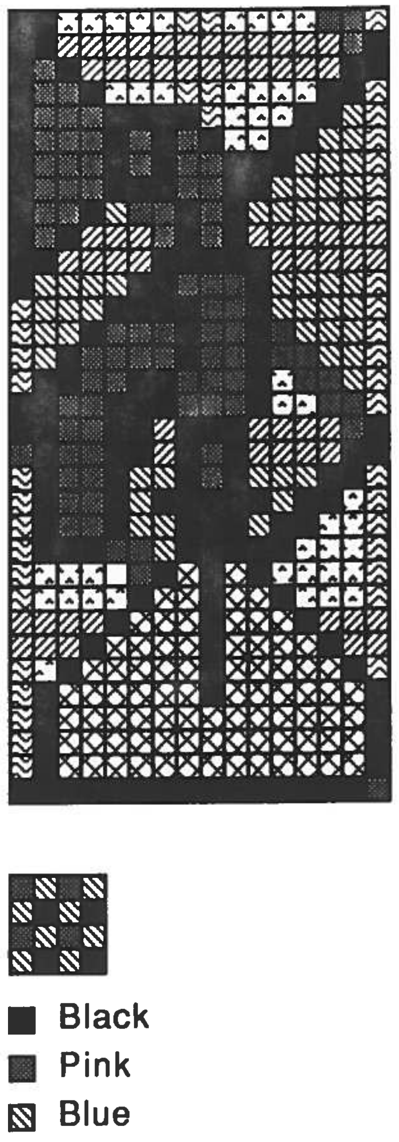


Fig. 2d - with pattern

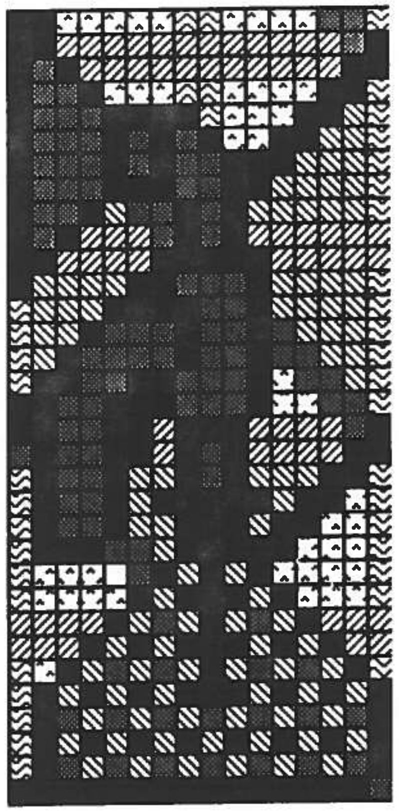
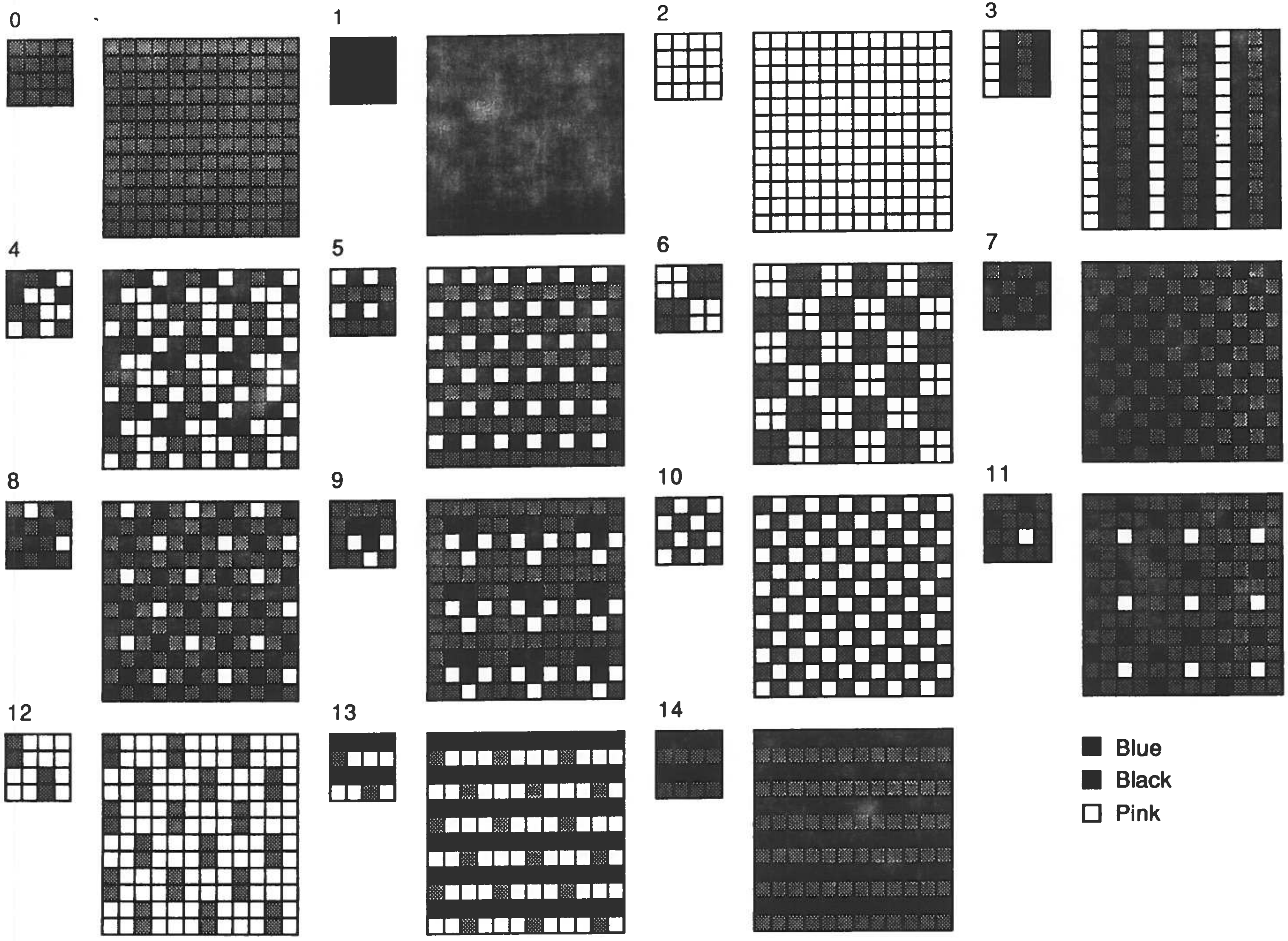


Fig. 2 - How an object is drawn

Fig. 3 - Habitat standard patterns

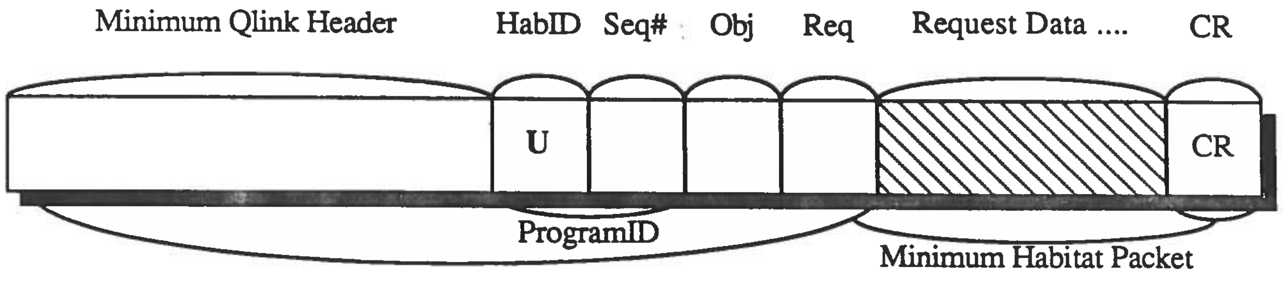


Communications driver

- Packets processed during VBLANK
- Valid Habitat messages escaped and buffered
- Messages processed during main loop

The Habitat Message Packet

The Habitat Message Packet looks like this:



The ASCII character 'U' is the Habitat Program ID byte. In order to reduce bandwidth we made the second Program ID byte the Habitat Message Sequence Number. This number is separate from the Q-Link protocol sequence number. This Habitat Message Sequence Number is generated by the C64 side only! The exact same number is used in any reply from the host. If the Host has something to tell the C64, it sends the ASYNC Seq#. This is how the Seq# is built:

e1s0xxxx

where:

e : This bit is the 'end of message flag
s : This bit indicates 'start of message
x : 0-15 ring counter sequence number

This makes the valid Seq#s:
@ABCDEFGHIJKLMNO Z (and same with high bits set)
`abcdefghijklmno z (and same with high bits set)
or: Lower Case, High bit set = single packet request (start & end)
Upper Case, High bit set = last packet of a multi-packet message
Lower Case, High bit clear = first packet of a multi-packet message
Upper Case, High bit clear = Nth packet of a multi-packet message

Z, z, ~Z, and ~z indicate an incomming ASYNC message from the host.

Remaining message byte values all have a potential range of 0-255. Since the packet network uses Carriage Return (0x0d & 0x8d) as the packet delimiter we must 'escape' Carriage Return bytes. We chose '[' as the escape character, so we must escape it also. An escape sequence is '[' and the value XORed with 0x55.

The OBJect number comes next. This is the host-assigned number for the object acting. Range 0-255

Next comes the REQuest number: This number is relative to the class of the OBJect being referenced. Different classes of objects do different things with the same request number.

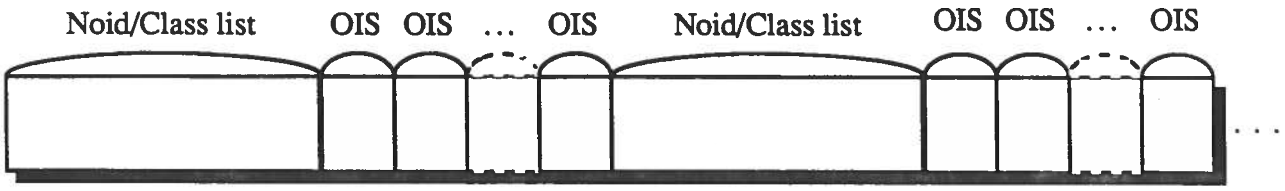
The bytes that follow are either the data going to the host or the reply data coming back. Like the request number, interpretation is class dependent.

Heap manager

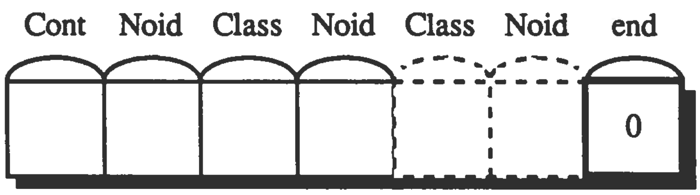
- Pointers by object type (COISA)
- The block header
- The delete blocks
- Alloc
    - Best fit
    - Garbage collection
- Dealloc
    - LRU scheme
- "The Head Hack"
    - Rationale
    - This can go away

The Contents Vector

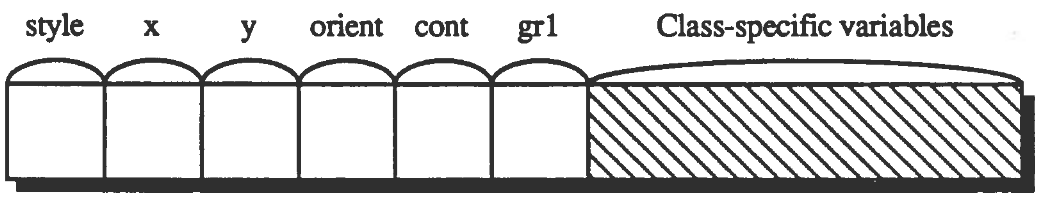
The Contents Vector



Noid/Class List



The Object Instance String (OIS)



Sound effects driver

- Handles object sound requests and collisions
- No priorities
- No continuous sounds
- Updates during VBLANK
- Sounds stored as tokens
- “Music” is really just long SFX
- Example

```
; magic_2_pw
        ; Pulsewidth Ramp starts here
        byte    0x4             ;Ramp
        byte    0x0,0x0,0x10,0x1,0x80,0x2       ; Start_Freq,Duration,Increment

        byte    0x2             ;Stop
        byte    0x0     ; next_sfx_number
```

magic_2.sob

```
; magic_2
;    Uses PW
; voice header
;       byte    0x0,0x0,0x0,0x0 ;Ring/Sync flag,User addr,Filter Flag
        byte    0x2d            ;Gate Ramp ADSR Wave
        byte    0x0,0x0,0x5a,0x0,0x70,0x81      ; Start_Freq,Duration,Increment
        byte    0xda,0x0        ; ADSR
        byte    0x40    ; Waveform

        byte    0x2d            ;Gate Ramp ADSR Wave
        byte    0x0,0x2,0x20,0x0,0x50,0x80      ; Start_Freq,Duration,Increment
        byte    0x0,0xf9        ; ADSR
        byte    0x80    ; Waveform

        byte    0x4             ;Ramp
        byte    0x0,0x2,0x40,0x0,0x20,0x40      ; Start_Freq,Duration,Increment

        byte    0x20            ;Wave
        byte    0x0     ; Waveform

        byte    0x2             ;Stop
        byte    0x0     ; next_sfx_number
```

User interface

- Cursor
  - Scanned every VBLANK
  - Can be “locked out” while a command is in progress
- Keyboard
  - Scanned every VBLANK
  - Can also be locked out
  - Output route depends on mode (paper or talk)

Interface to object behaviors

- Command selection
  - Icon
  - Press return on text line
  - Paper "buttons"
- The "one command" queue
  - Recursive commands (facing, walking)
  - Processed in main loop
  - Locks interface (flashing cursor)
- Command processing
  - Waiting
    - For animation
    - For a reply
  - The "Host always replies" rule
    - Rationale
  - The Throttle
    - Rationale
  - Asynchronous interrupts
    - What if the object goes away?

Object behaviors

- General form
- Macro library
- Standard variables in environment
- Specific details of specific objects

pawn_machine_MUNCH.m

```
;
;       pawn_machine_MUNCH.m
;
;       Asynch code for the pawn machine operation.
;       This is the asynch behavior for pawn machines.
;
;       This file should be assembled as position independent code.
;
;       Chip Morningstar/F Randall Farmer
;       Lucasfilm Ltd.
;       Sept, 1986
;
        include "action_head.i"
        include "class_equates.m"

        actionStart

; sent by host
define  MUNCHER_NOID    =       0               ; avatar 'do'ing the machine

        moveb           actor_noid, subject_noid
        complexSound    PAWN_MUNCH, actor_noid
        getResponse     MUNCHER_NOID
        sta             actor_noid
        jsr             v_set_up_actor_pointers
        chore           AV_ACT_operate
        newImage        subject_noid, 1
        asyncAnimationWait
        chore           AV_ACT_hand_back
        asyncAnimationWait
        newImage        subject_noid, 0
        rts

        actionEnd
```

```
;
;       pawn_machine_do.m
;
;       Action code for the pawn machine 'munch and spit out token'
;       This is the 'do' behavior for pawn machines.
;
;       This file should be assembled as position independent code.
;
;       Chip Morningstar/F Randall Farmer
;       Lucasfilm Ltd.
;       Sept, 1986
;
        include "action_head.i"
        include "class_equates.m"

        actionStart

; returned by host
define  MUNCH_SUCCESS   =       0

        jsr     v_punt_if_not_adjacent
        chore   AV_ACT_operate
        complexSound    PAWN_MUNCH
        newImage pointed_noid, 1
        sendMsg pointed_noid, MSG_MUNCH, 0
        newImage pointed_noid, 0
        chore   AV_ACT_hand_back
        getResponse     MUNCH_SUCCESS
        cmp     #SUCCESS_VALUE
        if (equal) {
            lda pointed_noid
            chainTo v_purge_contents            ; tokens arrive async.
        }
        chainTo v_beep_or_boing

        actionEnd
```

```
;
;       spray_can_SPRAY.m
;
;       Action code for the asynchronous spray can behavior.
;
;       This file should be assembled as position independent code.
;
;       Chip Morningstar
;       Lucasfilm Ltd.
;       18-August-1986
;
;

        include "action_head.i"

; These are the parameters for the SPRAY message
define SPRAY_SPRAYEE = 0
define SPRAY_CUSTOMIZE_0 = 1
define SPRAY_CUSTOMIZE_1 = 2

        actionStart

        sound   SPRAY, actor_noid
        getResponse SPRAY_SPRAYEE
        jsr v_get_subject_object
        getResponse SPRAY_CUSTOMIZE_0
        putProp subject, AVATAR_customize
        getResponse SPRAY_CUSTOMIZE_1
        putProp subject, AVATAR_customize+1
        rts

        actionEnd
```

```
;
;       spray_can_do.m
;
;       Action code for the spray can 'do' behavior.
;
;       This file should be assembled as position independent code.
;
;       Chip Morningstar
;       Lucasfilm Ltd.
;       18-August-1986
;
;
        include "action_head.i"

;This is the parameter to the SPRAY request
define SPRAY_LIMB = 0
; And these are the parameters it returns
define SPRAY_SUCCESS = 0
define SPRAY_CUSTOMIZE_0 = 1
define SPRAY_CUSTOMIZE_1 = 2

        actionStart

        lda     in_hand_noid            ; Are we holding the spray can?
        cmp     pointed_noid
        if (equal) {                    ; Yes, change ourselves
                sound   SPRAY
                lda pointed_at_limb     ; What body part are we indicating?
                putArg SPRAY_LIMB
                sendMsg pointed_noid, MSG_SPRAY, 1
                getResponse SPRAY_SUCCESS
                if (!zero) {
                        getResponse SPRAY_CUSTOMIZE_0
                        putProp actor, AVATAR_customize
                        getResponse SPRAY_CUSTOMIZE_1
                        putProp actor, AVATAR_customize+1
                        rts
                }
        }
        chainTo v_beep

        actionEnd
```

Habitat technology

- *How the system was specified*
- *Description of communications protocol*
- *C64 system*
- Host system
- *Development system*
- *The Habitat world database*

Host system

- Relationship between processes
- habitat
- regionproc
- habitat_db
- hatchery
- object_ids
- records

· host listings
ephemera doc

regionproc

- Roomer model
- Various routines and files
- Object behaviors
    - Layout of a class file
    - Layout of a struct file
    - Standard, generic behaviors
    - Peculiar, idiosyncratic behaviors
    - Special classes
        - Region
        - Avatar
    - Survey of various classes' implementation

```
/*
 *   struct_pawn_machine.incl.pl1
 *
 *   Struct stub for pawn_machine instance descriptor.
 *
 *   Chip Morningstar
 *   Lucasfilm Ltd.
 *   6-October-1986
 *
 */
,    2    common_head          like instance_head,
     2    contents             pointer,
     2    class_specific       ,
           3   open_flags      binary(15),
           3   key_hi          binary(15),
           3   key_lo          binary(15);
```

```
/*
 *   class_pawn_machine.pl1
 *
 *   Behavior module for object class pawn_machine.
 *
 *   Chip Morningstar
 *   Lucasfilm Ltd.
 *   6-October-1986
 */

%replace PAWN_MACHINE_CAPACITY by 1;

%include 'microcosm.incl.pl1';
%include 'defs_helper.incl.pl1';
%include 'defs_action.incl.pl1';

initialize_class_pawn_machine: procedure;

     %replace PAWN_MACHINE_REQUESTS by 6;

     declare a(0:PAWN_MACHINE_REQUESTS) entry based;
     declare class_pawn_machine_actions pointer;
     declare 1 pawn_machine based %include struct_pawn_machine;

     %replace I by CLASS_PAWN_MACHINE;

     Class_Table(I).capacity = PAWN_MACHINE_CAPACITY;
     Class_Table(I).max_requests = PAWN_MACHINE_REQUESTS;
     Class_Table(I).alloc_size = size(pawn_machine);
     Class_Table(I).pc_state_bytes = 3;
     Class_Table(I).known = true;
     Class_Table(I).opaque_container = true;
     Class_Table(I).filler = false;

     allocate a set(class_pawn_machine_actions);
     Class_Table(I).actions = class_pawn_machine_actions;

     Class_Table(I).actions->a(HELP)   = generic_HELP;      /* 0 */
     Class_Table(I).actions->a(1)      = illegal;           /* 1 */
     Class_Table(I).actions->a(2)      = illegal;           /* 2 */
     Class_Table(I).actions->a(3)      = illegal;           /* 3 */
     Class_Table(I).actions->a(4)      = illegal;           /* 4 */
     Class_Table(I).actions->a(5)      = illegal;           /* 5 */
     Class_Table(I).actions->a(MUNCH)  = pawn_machine_MUNCH;/* 6 */

end initialize_class_pawn_machine;

pawn_machine_MUNCH: procedure;
     declare 1 self based(selfptr) %include struct_pawn_machine;

     if (adjacent(selfptr) & self.contents->c(0) ^= NULL) then do;
          if (pay_to(avatarptr, item_value(ObjList(self.contents->c(0))))) then
+ do;
               call n_msg_1(selfptr, MUNCH$, avatar.noid);
               call n_msg_1(null(), GOAWAY_$, self.contents->c(0));
               call destroy_contents(selfptr);
               call r_msg_1(TRUE);
               return;
          end;
          call r_msg_1(BOING_FAILURE);
          return;
     end;
     call r_msg_1(FALSE);
end pawn_machine_MUNCH;
```

```
%page;

declare RoomNumber                        binary(15) external initial(0);
declare RoomPtr                           pointer external;
declare RoomPtrs(regions_per_process)     pointer external;
declare CapMonPtr                         pointer external;
declare CapMonPtrs(regions_per_process)   pointer external;

declare 1 RoomDBank                       based (RoomPtr),
          2  Region                       binary(31),
          2  Region_name                  character(20),
          2  RoomQId                      binary(31),
          2  RoomBQId                     binary(31),
          2  last_noid                    binary(15),
          2  total_ghosts                 binary(15),
          2  Pending                      pointer,
          2  flags,
             3 private                    bit(1),
             3 owner_here                 bit(1),
             3 initialized                bit(1),
             3 filler_flags               bit(13),
          2  current_region,
             3 lighting                   binary(15),
             3 depth                      binary(15),
             3 neighbor(4)                binary(31),
             3 exit_type(4)               binary(15),
             3 restriction(4)             bit(1),
             3 nitty_bits(28)             bit(1),
             3 max_avatars                binary(15),
             3 owner                      binary(31),
             3 entry_proc                 binary(15),
             3 exit_proc                  binary(15),
             3 class_group                binary(15),
             3 orientation                binary(15),
             3 object_count               binary(15),
             3 space_usage                binary(15),
             3 town_dir                   character(1),
             3 port_dir                   character(1),
          2  oracle,
             3 object                     binary(15),
             3 person                     binary(15),
             3 control                    pointer,
          2  UserList(UsersPerRegion)     pointer,
          2  ObjList(0:ObjectsPerRegion) pointer,
          2  GhostList                    pointer,
          2  Block_addr                   pointer;

declare 1 RoomCMon                        based (CapMonPtr),
          2  class_ref_count(0:MAX_CLASS_NUMBER)      binary(15),
          2  resource_ref_count(NUMBER_OF_RESOURCES)  binary(15);

declare 1 Memory_Block                    based,
          2  free                         bit(64),
          2  entry(64)                    char(40);

declare  DayNight   bin(15) external initial(0);

declare 1         player    based                     /* entry for avatar    */
%include 'struct_user.incl.pl1';

declare 1         object    based                     /* entry for object    */
%include 'instance_head.incl.pl1';,
        2         param1    pointer,                  /* to contents list    */
        2         param2    char(1);                  /* depends on class    */
```

```
declare  current_noid                    binary (15) external;
declare  current_request                 binary (15) external;
declare  current_header                  char(1) external;
declare  current_qid                     binary (31) external;
declare  selfptr                         pointer external initial(null());
declare  avatarptr                       pointer external initial(null());
declare  userptr                         pointer external initial(null());

declare  request_string char(646) var external;

declare fan_cnt                          binary(15) external;
declare fan_list(UsersPerRegion+200)     pointer external;  /* + 200 ghosts */

declare 1 now_in    external,
          2 last    bin(31)         initial(0),
          2 count   bin(15)         initial(0),
          2 line    (4) char(40) var;

declare 1 enter_info  based,                /* remember info for later retry */
          2 room         binary(31),
          2 user         binary(31),
          2 que          binary(31),
          2 attempts     binary(15),
          2 params       char(254) var;

declare 1 fountain  based,
          2 type         binary(15),
          2 which_room   binary(15),
          2 start_time   binary(31),
          2 end_time     binary(31),
          2 interval     binary(31),
          2 msg_text     char(100) var;

declare bit_mask(UsersPerRegion)   bit(32) external;   /* user index bit mask */

%replace Separation_Char by 144;
```

```
/*
 *   class_region.pl1
 *
 *   Region object behavior module for MicroCosm(TM).
 *
 *   Chip Morningstar
 *   Lucasfilm Ltd.
 *   8-April-1986
 *
 * Revised to add 'prompt-reply' engine
 * 28-Jan-1987 FRF
 */

%include 'microcosm.incl.pl1';
%include 'defs_action.incl.pl1';
%include 'defs_helper.incl.pl1';

%replace NUMBER_OF_REPLYS by 3; /*This should GROW*/
declare reply_actions(NUMBER_OF_REPLYS) entry variable external;
declare reply_strings(NUMBER_OF_REPLYS) character(114) varying static init(
          'Edit:',
               /* 1 = God Tool */
          'Yes?',
               /* 2 = Magic Opener */
          'Enter your 3 digit number:'
               /* 3 = Lottery */
);


declare set_offline entry (binary(15));
declare update_object_disk entry;
declare god_tool_revisited entry;
declare magic_opener_revisited entry;
declare lottery_revisited entry;

initialize_class_region: procedure;

      %replace REGION_REQUESTS by 7;

      declare a(0:REGION_REQUESTS) entry based;
      declare class_region_actions pointer;

      %replace I by CLASS_REGION;

      Class_Table(I).capacity = 0;  /* Contents is in ObjList, not instance */
      Class_Table(I).max_requests = REGION_REQUESTS;
      Class_Table(I).alloc_size = size(self);
      Class_Table(I).pc_state_bytes = 0; /* Not really true */
      Class_Table(I).known = true;
      Class_Table(I).opaque_container = false;
      Class_Table(I).filler = false;

      allocate a set(class_region_actions);
      Class_Table(I).actions = class_region_actions;

      Class_Table(I).actions->a(HELP)        = generic_HELP;          /* 0 */
      Class_Table(I).actions->a(DESCRIBE)    = region_DESCRIBE;       /* 1 */
      Class_Table(I).actions->a(LEAVE)       = region_LEAVE;          /* 2 */
      Class_Table(I).actions->a(IMALIVE)     = region_IM_ALIVE;       /* 3 */
      Class_Table(I).actions->a(CUSTOMIZE)   = region_CUSTOMIZE;      /* 4 */
      Class_Table(I).actions->a(FINGER_IN_QUE) = region_FINGER_IN_QUE; /* 5 */
      Class_Table(I).actions->a(I_AM_HERE)   = region_I_AM_HERE;    /* 6 */
      Class_Table(I).actions->a(PROMPT_REPLY) = region_PROMPT_REPLY;    /* 7 */
```

```
     reply_actions(1) = god_tool_revisited;
     reply_actions(2) = magic_opener_revisited;
     reply_actions(3) = lottery_revisited;

end initialize_class_region;

region_DESCRIBE: procedure;
     %replace CURRENT_OBJECT_VERSION by '0';

     if (substr(request_string,3,1) = CURRENT_OBJECT_VERSION)
          then call send_region_contents;
          else call update_object_disk;
end region_DESCRIBE;

region_LEAVE: procedure;
     call set_offline (avatar.avatarslot);
end region_LEAVE;

     /*  The IM_ALIVE and CUSTOMIZE request processing deserve special  */
     /*  comment.  First, the fact that they get to this point means    */
     /*  that a successful response should be sent.  This was           */
     /*  determined by the master process (microcosm), but the          */
     /*  actual response must be sent by this process (regionproc)      */
     /*  in order to insure that the terminal handler's default output  */
     /*  queue has been switched to this process.  JDH                  */

region_IM_ALIVE: procedure;
     call r_msg_2_s (TRUE, 48   /* 48 = ascii "0" */,
                'Insert Habitat data disk; press any key.');
end region_IM_ALIVE;

region_CUSTOMIZE: procedure;
     call r_msg_1 (TRUE);
end region_CUSTOMIZE;

region_FINGER_IN_QUE: procedure;
     call p_msg_1(null(),avatarptr,CAUGHT_UP_$,TRUE);
end region_FINGER_IN_QUE;

region_I_AM_HERE: procedure;
     call clear_bit(avatar.gr_state,7);
     call b_msg_1(null(),APPEARING_$,avatar.noid);
end region_I_AM_HERE;

region_PROMPT_REPLY: procedure;
declare i binary(15);
declare L binary(15);
declare strlen binary(15);
declare reply_number binary(15);

     do i=0 to NUMBER_OF_REPLYS;
          L = length(reply_strings(i));
          if (length(request_string) >= L) then do;
               if (substr(request_string,1,L)=reply_strings(i)) then do;
                    strlen=L;
                    reply_number=i;
               end;
          end;
     end;
     if (reply_number>0) then do;
          request_string = substr(request_string,strlen+1);
          call reply_actions(reply_number);
     end;
end region_PROMPT_REPLY;
```

```
/*
 *   struct_avatar.incl.pl1
 *
 *   Struct stub for avatar instance descriptor.
 *
 *   Chip Morningstar
 *   Lucasfilm Ltd.
 *   8-April-1986
 *   revised 13-March-1987 JH new database structure
 *
 */
,    2    common_head                        like instance_head,
     2    contents                           pointer,
     2    class_specific                     ,
           3   activity                      binary(15),
           3   action                        binary(15),
           3   health                        binary(15),
           3   restrainer                    binary(15),
           3   customize(3)                  binary(15),
/* Host specific: */
           3   bank_account_balance          binary(31),
           3   turf                          binary(31),
           3   stun_count                    binary(15),
           3   nitty_bits(32)                bit(1),
           3   true_orientation              binary(15),
           3   true_head_style               binary(15),
           3   true_custom(3)                binary(15),
           3   curse_type                    binary(15),
           3   curse_counter                 binary(15),
           3   last_login                    binary(31);
```

```
/*
 *   class_avatar.pl1
 *
 *   Avatar object behavior module for Habitat.
 *
 *   Chip Morningstar
 *   Lucasfilm Ltd.
 *   8-April-1986
 */

%include 'microcosm.incl.pl1';
%include 'defs_action.incl.pl1';
%include 'defs_helper.incl.pl1';

declare process_messager_name entry;
declare process_messager_msg entry;
declare request_player_list entry;
declare handle_internal_trace entry (char(*) var);

%replace SIT_GROUND by 132;
%replace SIT_CHAIR by 133;
%replace SIT_FRONT by 157;
%replace STAND_FRONT by 146;
%replace STAND_LEFT by 251;
%replace STAND_RIGHT by 252;
%replace STAND by 129;
%replace FACE_LEFT by 254;
%replace FACE_RIGHT by 255;

%replace COLOR_POSTURE by 253;

initialize_class_avatar: procedure;

     %replace AVATAR_REQUESTS by 14;

     declare a(0:AVATAR_REQUESTS) entry based;
     declare class_avatar_actions pointer;
     declare 1 avatar based %include struct_avatar;

     %replace I by CLASS_AVATAR;

     Class_Table(I).capacity = AVATAR_CAPACITY;
     Class_Table(I).max_requests = AVATAR_REQUESTS;
     Class_Table(I).alloc_size = size(avatar);
     Class_Table(I).pc_state_bytes = 6;
     Class_Table(I).known = true;
     Class_Table(I).opaque_container = true;
     Class_Table(I).filler = false;

     allocate a set(class_avatar_actions);
     Class_Table(I).actions = class_avatar_actions;

     Class_Table(I).actions->a(HELP) = avatar_IDENTIFY;      /* 0 */
     Class_Table(I).actions->a(1) = illegal;                 /* 1 */
     Class_Table(I).actions->a(2) = illegal;                 /* 2 */
     Class_Table(I).actions->a(3) = illegal;                 /* 3 */
     Class_Table(I).actions->a(GRAB) = avatar_GRAB;          /* 4 */
     Class_Table(I).actions->a(HAND) = avatar_HAND;          /* 5 */
     Class_Table(I).actions->a(POSTURE) = avatar_POSTURE;    /* 6 */
     Class_Table(I).actions->a(SPEAK) = avatar_SPEAK;        /* 7 */
     Class_Table(I).actions->a(WALK) = avatar_WALK;          /* 8 */
     Class_Table(I).actions->a(NEWREGION) = avatar_NEWREGION;/* 9 */
     Class_Table(I).actions->a(DISCORPORATE) = avatar_DISCORPORATE;/*10 */
     Class_Table(I).actions->a(ESP) = avatar_ESP;            /* 11 */
     Class_Table(I).actions->a(SIT) = avatar_SITORSTAND;    /* 12 */
     Class_Table(I).actions->a(TOUCH) = avatar_TOUCH;       /* 13 */
     Class_Table(I).actions->a(FNKEY) = avatar_FNKEY;  /* 14 */

end initialize_class_avatar;

avatar_DISCORPORATE: procedure;
     if (holding_class(CLASS_MAGIC_LAMP)) then do;
          if (ObjList(avatar.contents->c(HANDS))->o.gr_state =
                    MAGIC_LAMP_GENIE) then
               call object_say(self.noid, 'You can''t turn into a ghost while you are hold
     end; else if (holding_restricted_object(avatarptr)) then do;
          call object_say(self.noid, 'You can''t turn into a ghost while you are holding t
     end; else do;
          call lights_off(avatarptr);
          call switch_to_ghost;
          return;
     end;
     call r_msg_1(FALSE);
end avatar_DISCORPORATE;

avatar_GRAB: procedure;
     declare item_noid binary(15);
     declare 1 self based(selfptr) %include struct_avatar;
     declare 1 item based(itemptr) %include struct_gen_object;
     declare itemptr pointer;

     item_noid = NULL;
     if (empty_handed(avatarptr) & ^empty_handed(selfptr)) then do;
          item_noid = self.contents->c(HANDS);
          itemptr = ObjList(item_noid);
          if (^grabable(itemptr)) then do;
               call r_msg_1(NULL);
               if (current_region.nitty_bits(STEAL_FREE)) then
                    call object_say(self.noid,'This is a theft-free zone.');
               return;
          end;
          if (^UserList(self.avatarslot)->u.online) then do;
               call r_msg_1(NULL);
               return;
          end;
          if (^ change_containers(item_noid, avatar.noid, HANDS, true)) then do;
               call r_msg_1(NULL);
               return;
          end;
          call n_msg_1(avatarptr, GRABFROM$, self.noid);
          call inc_record(avatarptr, HS$grabs);
     end;
     call r_msg_1(item_noid);
end avatar_GRAB;

avatar_HAND: procedure;
     declare success binary(15);
     declare item_noid binary(15);
     declare 1 self based(selfptr) %include struct_avatar;

     if (self.class = CLASS_MAGIC_LAMP & self.gr_state =
               MAGIC_LAMP_GENIE) then do;
          call object_say(item_noid, 'You can''t give away the Genie!');
          success = FALSE;
     end; else if (^empty_handed(avatarptr) & empty_handed(selfptr) &
               self.container = THE_REGION) then do;
          success = TRUE;
          item_noid = avatar.contents->c(HANDS);
          if (^change_containers(item_noid, self.noid, HANDS, true)) then
               success = FALSE;
          else do;
               self.activity = STAND;
               self.gen_flags(MODIFIED) = true;
               call n_msg_1(selfptr, GRABFROM$, avatar.noid);
          end;
     end; else do;
          success = FALSE;
     end;
     call r_msg_1(success);
end avatar_HAND;

avatar_POSTURE: procedure;
     declare new_posture binary(15);

     new_posture = rank(request(FIRST));
     if (selfptr = avatarptr) then do;
          if (0 <= new_posture & new_posture < 256) then do;
               if (new_posture = SIT_GROUND | new_posture = SIT_CHAIR |
                         new_posture = SIT_FRONT | new_posture = STAND |
                         new_posture = STAND_FRONT | new_posture = STAND_LEFT|
                         new_posture = STAND_RIGHT | new_posture = FACE_LEFT |
                         new_posture = FACE_RIGHT) then
                    avatar.activity = new_posture;
               if (new_posture ^= COLOR_POSTURE) then
                    call n_msg_1(avatarptr, POSTURE$, new_posture);
               if (new_posture < STAND_LEFT | new_posture = COLOR_POSTURE) then
                    call r_msg_1(TRUE);
          end;
     end;
end avatar_POSTURE;

avatar_SPEAK: procedure;
     declare audience binary(15);
     declare audienceptr pointer;
     declare text character(TEXT_LENGTH) varying;

     audience = rank(request(FIRST));
     text = substr(request_string, 2);
     if (length(text) < 4) then text = ' ' || text || ' ';
     if (substr(text,1,3) = '=!T') then do;
          call handle_internal_trace (substr(text,4));
          call r_msg_1(FALSE);
          return;
     end;
     if (index(text, 'TO:') = 1 | index(text, 'To:') = 1 |
               index(text, 'to:') = 1) then do;
          request_string = ltrim(substr(text, 4));
          call process_messager_name;
     end; else do;
          if (avatar.curse_type = CURSE_SMILEY) then
               text = 'Have a nice day!';
          else if (avatar.curse_type = CURSE_FLY) then
               text = buzzify(text);
          call b_msg_s(avatarptr, SPEAK$, text);
/*        call n_msg_s(avatarptr, SPEAK$, text);
          call r_msg_1_s(avatarptr, FALSE, text);  */
          call inc_record(avatarptr, HS$talkcount);
          call r_msg_1(FALSE);
     end;
end avatar_SPEAK;

avatar_ESP: procedure;
```

```
          if (length(request_string) <= 5) then
               call r_msg_1(FALSE);
          else do;
               request_string = substr(request_string, 6); /* remove 'ESP:' */
               if (length(request_string) < 4) then request_string = ' ' || request_string || '
               call inc_record(avatarptr, HS$esp_send_count);
               call process_messager_msg;
               call r_msg_1(TRUE);
          end;
     end avatar_ESP;

     avatar_WALK: procedure;
          declare x binary(15);
          declare y binary(15);
          declare walk_how binary(15);
          declare destination_x binary(15);
          declare destination_y binary(15);
          declare flip_path bit(1) aligned;

          x = rank(request(FIRST));
          y = rank(request(SECOND));
          walk_how = rank(request(THIRD));
          if (selfptr ^= avatarptr) then do;
               destination_x = avatar.x;
               destination_y = avatar.y;
          end; else if (avatar.stun_count > 0) then do;
               avatar.stun_count = avatar.stun_count - 1;
               call r_msg_3(avatar.x, avatar.y, walk_how);
               if (avatar.stun_count >= 2) then
                    call p_msg_s(selfptr, selfptr, SPEAK$, 'I can''t move.  I am stunned.');
               else if (avatar.stun_count = 1) then
                    call p_msg_s(selfptr, selfptr, SPEAK$, 'I am still stunned.');
               else
                    call p_msg_s(selfptr, selfptr, SPEAK$, 'The stun effect is wearing off now.
               return;
          end; else do;
               call check_path(THE_REGION, x, y, destination_x, destination_y, flip_path);
               if (flip_path) then
                    call set_bit(walk_how, 8);
               else
                    call clear_bit(walk_how, 8);
               if (destination_x ^= self.x | destination_y ^= self.y) then do;
                    self.x = destination_x;
                    self.y = destination_y;
                    call n_msg_3(selfptr, WALK$, destination_x, destination_y,
                         walk_how);
               end;
          end;
          call r_msg_3(destination_x, destination_y, walk_how);
     end avatar_WALK;

     avatar_NEWREGION: procedure;
          declare direction binary(15);
          declare new_region binary(31);
          declare new_entry_mode binary(15);
          declare direction_index binary(15);
          declare passage_id binary(15);
          declare passageptr pointer;
          declare 1 passage based(passageptr) %include struct_door;

          direction = rank(request(FIRST));
          passage_id = rank(request(SECOND));
          new_region = 0;
          new_entry_mode = 0;
```

```
if (direction = AUTO_TELEPORT_DIR) then do;
     if (UserList(self.avatarslot)->u.auto_destination ^= 0) then do;
          new_region = UserList(self.avatarslot)->u.auto_destination;
          UserList(self.avatarslot)->u.auto_destination = 0;
          new_entry_mode = UserList(self.avatarslot)->u.auto_mode;
     end; else do;
          /* If we get here the user has hacked his C64.  By executing a
             return statement here, we don't respond to his message, thus
             hanging him. */
          call object_say(self.noid, 'Hi hacker!');
          return;
     end;
end; else do;
     passageptr = ObjList(passage_id);
     if (passageptr ^= null()) then do;
          if (passage.class = CLASS_DOOR |
                    passage.class = CLASS_BUILDING) then do;
               if (passage.class = CLASS_DOOR) then do;
                    if (^test_bit(passage.open_flags, OPEN_BIT) |
                              passage.gen_flags(
                              DOOR_AVATAR_RESTRICTED_BIT)) then do;
                         call r_msg_1(FALSE);
                         return;
                    end;
               end;
               new_region = passage.connection;
          end;
     end;
end;

direction_index = mod(direction + current_region.orientation + 2, 4) + 1;

if (selfptr = avatarptr & 0 <= direction & direction <= 4) then do;
     if (holding_class(CLASS_MAGIC_LAMP)) then
          if (ObjList(avatar.contents->c(HANDS))->o.gr_state =
                    MAGIC_LAMP_GENIE) then do;
               if (direction = AUTO_TELEPORT_DIR) then
                    call drop_object_in_hand(selfptr);
               else do;
                    call object_say(self.noid,
                    'You can''t leave while you are holding the Genie.');
                    call r_msg_1(FALSE);
                    return;
               end;
          end;
     if (holding_restricted_object(avatarptr)) then do;
          if (direction = AUTO_TELEPORT_DIR) then
               call drop_object_in_hand(selfptr);
          else if (current_region.restriction(direction_index)) then do;
               call object_say(self.noid,
                    'You can''t leave while you are holding that.');
               call r_msg_1(FALSE);
               return;
          end;
     end;
     if (new_region = 0) then
          new_region = current_region.neighbor(direction_index);
     if (new_region ^= 0 & new_region ^= -1) then do;
          call n_msg_1(null(), WAITFOR_$, avatar.noid);
          call goto_new_region(avatarptr, new_region, direction,
               new_entry_mode);
          return;
     end;
end;
```

```
          call object_say(self.noid, 'There is nowhere to go in that direction.');
          call r_msg_1(FALSE);
     end avatar_NEWREGION;

     holding_restricted_object: procedure(whoptr) returns(bit(1));
          declare whoptr pointer;
          declare 1 who based(whoptr) %include struct_avatar;
          declare obj_noid binary(15);
          declare objptr pointer;
          declare 1 obj based(objptr) %include struct_gen_object;

          obj_noid = who.contents->c(HANDS);
          if (obj_noid ^= NULL) then do;
               objptr = ObjList(obj_noid);
               if (objptr ^= null()) then
                    return(obj.gen_flags(RESTRICTED));
          end;
          return(false);
     end holding_restricted_object;

     avatar_IDENTIFY: procedure;
          declare selfname character(20) varying;
          declare avatarname character(20) varying;
          declare result character(256) varying;

          selfname = ltrim(rtrim(UserList(self.avatarslot)->u.U_Name));
          avatarname = ltrim(rtrim(UserList(avatar.avatarslot)->u.U_Name));
          if (avatarptr = selfptr) then do;
               result = 'Your name is ' || avatarname || '.  You are ';
               if (avatar.stun_count > 0) then
                    result = result || 'stunned, and you are ';
               if (avatar.health > 200) then
                    result = result || 'in the peak of health.';
               else if (avatar.health > 150) then
                    result = result || 'in good health.';
               else if (avatar.health > 100) then
                    result = result || 'in fair health.';
               else if (avatar.health > 50) then
                    result = result || 'in poor health.';
               else
                    result = result || 'near death.';
          end; else do;
               call p_msg_s(avatarptr, selfptr, SPEAK$, 'I am ' || avatarname);
               if (UserList(self.avatarslot)->u.online) then
                    result = 'This is ' || selfname;
               else
                    result = 'Turned to stone: ' || selfname;
          end;
          call r_msg_s(result);
     end avatar_IDENTIFY;

     %replace STAND_UP by 0;
     %replace SIT_DOWN by 1;

     avatar_SITORSTAND: procedure;
          declare up_or_down binary(15);
          declare seat_id binary(15);
          declare 1 seat based(seatptr) %include struct_gen_container;
          declare seatptr pointer;
          declare i binary(15);

          up_or_down = rank(request(FIRST));
          seat_id = rank(request(SECOND));
          seatptr = ObjList(seat_id);
     if (seatptr ^= null()) then do;
          if (seat.class = CLASS_CHAIR | seat.class = CLASS_COUCH |
                    seat.class = CLASS_BED) then do;
               if (up_or_down = STAND_UP) then do;
                    if (avatar.container = seat_id) then do;
                         if (^ change_containers(avatar.noid, THE_REGION,
                                                          0, false)) then do;
                              call r_msg_2 (FALSE,0);
                              return;
                         end;
                         avatar.activity = STAND;
                         avatar.gen_flags(MODIFIED) = true;
                         call checkpoint_object (0, avatar.noid);
                         call r_msg_2(TRUE, 0);
                         call n_msg_3(avatarptr, SIT$, STAND_UP, seat_id, 0);
                         return;
                    end;
               end; else do;
                    i = 0;
                    do while (i < Class_Table(seat.class).capacity);
                         if (seat.contents->c(i) = NULL) then do;
                              if (^ change_containers(avatar.noid, seat_id, i, true)) then
                                   call r_msg_2 (FALSE,0);
                                   return;
                              end;
                              call r_msg_2(TRUE, i);
                              call n_msg_3(avatarptr,SIT$,SIT_DOWN,seat_id,i);
                              return;
                         end;
                         i = i + 1;
                    end;
               end;
          end;
     end;
     call r_msg_2(FALSE, 0);
end avatar_SITORSTAND;

avatar_TOUCH: procedure;
     declare touchee_id binary(15);
     declare toucheeptr pointer;
     declare 1 touchee based(toucheeptr) %include struct_avatar;
     declare new_posture binary(15);
     declare 1 self based(selfptr) %include struct_avatar;

     touchee_id = rank(request(FIRST));
     new_posture = rank(request(SECOND));
     toucheeptr = ObjList(touchee_id);
     call n_msg_1(avatarptr, POSTURE$, new_posture);
     call p_msg_s(selfptr, toucheeptr, SPEAK$, 'Gotcha!');
     call p_msg_s(selfptr, avatarptr, SPEAK$, 'Gotcha!');
     call r_msg_1(TRUE);
     if (avatar.curse_type ^= 0) then
          call curse_touch(avatarptr, toucheeptr);
     else if (touchee.curse_type ^= 0) then
          call curse_touch(toucheeptr, avatarptr);
end avatar_TOUCH;

buzzify: procedure(text) returns(character(TEXT_LENGTH) varying);
     declare text character(*) varying;
     declare result character(TEXT_LENGTH) varying;
     declare resultstr(-1:TEXT_LENGTH) character(1) defined(result);
     declare i binary(15);

     result = translate(text,
        'zzzzzzzzzzzzzzzzzzzzzzzzzZZZZZZZZZZZZZZZZZZZZZZZZZZzzzzzzzzzz',
        'abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789');
     do i=length(result) to 1 by -1;
          if (((resultstr(i-1) ^= 'z' & resultstr(i-1) ^= 'Z') | i = 1) &
               (resultstr(i) = 'z' | resultstr(i) = 'Z')) then do;
               if (resultstr(i) = 'z') then
                    resultstr(i) = 'b';
               else
                    resultstr(i) = 'B';
          end;
     end;
     return(result);
end buzzify;

avatar_FNKEY: procedure;
declare fn_key binary(15);
declare ref_obj binary(15);

%replace F3 by 11;
%replace F4 by 12;
%replace F5 by 13;
%replace F8 by 16;

     fn_key  = rank(request(FIRST));
     ref_obj = rank(request(SECOND));

   if (fn_key = F3)
        then call request_player_list;
          else if (fn_key = F4) then do;
               call p_msg_s(null(),avatarptr,20,'What is Sekrit Word?');
               call r_msg_1(TRUE);
               end;
               else call r_msg_1(BOING_FAILURE);

end avatar_FNKEY;
```

Habitat technology

- *How the system was specified*
- *Description of communications protocol*
- *C64 system*
- *Host system*
- Development system
- *The Habitat world database*

Development system

- Code development
- Image creation
- A word on sound creation
- Misc stuff

Code development

- Macross and Slinky
  a note on syntax differences between Macross and generic 6502
- Make, et al
- Muddle and Puddle

Image creation

- Fastlink
- Face
- Utilities for image conversion

Misc stuff

- Weird Fastlink utilities
  - db, etc.
  - Folded into applications
- Heap dump analysis tools
  - describe
  - heapdiff
  - mem_map
  - mem_use
  - object_map

Habitat technology

- *How the system was specified*
- *Description of communications protocol*
- *C64 system*
- *Host system*
- *Development system*
- The Habitat world database

The Habitat world database

- The world map
- Cities, built and planned (internal view)
- Internal view of the world database

The world map

- How the map was generated
- Other realms

Cities, built and planned (internal view)

- Populopolis
- Quantumgrad
- Other cities

Internal view of the world database

- Structure of the primary databases
- The Habitat disk vs. memory model: regionproc vs. habitat_db
- Ghu
- Region creation tools

Structure of the primary databases

- Region
- Object
- Avatar
- Text
- Miscellaneous databases
  - Teleport
  - Oracle requests
  - System log
  - Mail

Ghu

- Command oriented database editor
  - A tool for techies!
- Can access all databases
- Macro and variable definition capability
- Flexible access control capability
  - Command set limitation allows special setup for non-techies
- Can run interactively or batch

ghu ghuide
ghu listing
macro library listing

Sample Ghu macros

```
macro  tome_awards
    value     "awarding Tome Hider"
     award  100,"skyline"
endmacro

macro  where_is_tome
      tome_winnings_report
    if (tome.container != tome_location)
          set tome_location = tome.container
        value '!!!!!!! The Tome has been Moved !!!!!!!  '.
         value tome_location
    endif
endmacro

macro  tome_winnings_report
     value last_winner.
    value ' has won $'.
     set winnings = (today - last_hidden)/one_day*100
     value winnings
endmacro

macro  tomewin
      tome_winnings_report
     set last_winner=$1
     set last_hidden=today
     read tome into tometext.doc
endmacro

macro  tomeupdate
     write tome file tometext.doc
     exec tome.macro
     v 'Type this line:'
      v 'sendmail rant_editor file tometorant.doc'
endmacro
```

Region creation tools

- Griddle and Fred
- Demonstration of Fred
- Plex

griddle/fred docs/listings
plex doc & listing
reno/red listing

Sample Griddle file

```
use region {
  owner: a 0
  light_level: 0
  depth: 24
  east_neighbor: r -1
  west_neighbor: r -1
  north_neighbor: r -1
  south_neighbor: r -1
  class_group: 0
  orient: 0
  entry_proc: 0
  exit_proc: 0
  east_exit: 0
  west_exit: 0
  north_exit: 0
  south_exit: 0
  east_restriction: '0'b
  west_restriction: '0'b
  north_restriction: '0'b
  south_restriction: '0'b
  weapons_free: '1'b
  nitty_bits: '10000000000000000000000000000'b
  name: "'1 #'2"
  avatars: 6
  town_dir:'\x7c'
  port_dir:'\x7c'
}
use ground {
  container: r -1001
  x: 0
  y: 4
  style: 1
  gr_state: 0
  orient: 228
  gr_width: 0
  restricted: '0'b
  nitty_bits: '00000000000000000000000000000000'b
}
use wall {
  container: r -1001
  x: 0
  y: 0
  style: 4
  gr_state: 0
  orient: 220
  gr_width: 0
  restricted: '0'b
  nitty_bits: '00000000000000000000000000000000'b
  length: 0
  height: 0
  pattern: 0
}
use window {
  container: r -1001
  x: 132
  y: 59
```

Sample Plex file

```
/*
        The Vanilla Habitat Apt Building

extn defined:   bnames[] (building names)
                afront() (apartment front regions)
                a()      (apartments interiors)
                h()      (3 door hallway segments)
                e()      (elevators)
                elby     (the generic near elevator lobby)
                bboard   (a residents bulliten board)
*/

hlseg:                  /* in $1= base room #, $2 = color $3=name */
          a?!($3,$1) a?!($3,$1+1) a?!($3,$1+2)
          |           |            |
        -###h*($1+0,$1+1,$1+2,c[$2],$3,$4)-

hl:                             /* in $1= base room #, $2 = color $3=name */
        -hlseg($1,$2,$3,$4)-hlseg($1+3,$2,$3,$4)-hlseg($1+6,$2,$3,$4)-hlseg($1+9,$2,$3,$4)

elby135:                        /* in $1 floor number (color) $2-name */
            |
        -elby($1*100,$1*100+11,$1*100+12,$1*100+23,$1*100+24,$1*100+35,c[$1-1],$2)-
            |

elby3671:
            |
        -elby($1*100+36,$1*100+47,$1*100+48,$1*100+59,$1*100+60,$1*100+71,c[$1-1],$2)-
            |

f:                      /* $1=name */
                     hl<(.*100+124,.,$1,la) -stairs(.+1,c[.],$1,la)- hl>(.*100+136,.,$1,ra
                                |                |                      |
        hl(.*100+112,.,$1,ra)-elby135<(.+1,$1)-e?(.+1,c[.],$1,ua)-elby3671>(.+1,$1)-hl(.*1
                                |                                       |
                     hl<(.*100+100,.,$1,ra)                   hl>(.*100+160,.,$1,la)


apartments:                     /* $1=name */
        -f($1)-f($1)-f($1)-f($1)-f($1)-f($1)-f($1)-f($1)-f($1)-f($1)-f($1)-

lobby:                  /* $1=name */
        -stairs("Lobby",c[.],$1,da)-
           |
        #e?("Lobby",c[.],$1,da)-bboard(bnames[$1])
           |

aptbuilding:
        basement-lobby(bnames[$1])-apartments(bnames[$1])-roof
                      |
                 afront?(bnames[$1])
```

Lucasfilm ➡ Fujitsu Habitat Technology Transfer Seminar

- *Preliminaries*
- *Overview of Habitat*
- *Supporting technology*
- *Habitat technology*
- Operations

Operations

- Support personnel
- Day-to-day operations
- Creating new experiences
- Rules, policies, and laws

Support personnel

- Rangers (guides and docents)
- Limited access operators
  - Oracles
  - Publishers
  - Property managers
  - etc.
- THE Oracle
- Designers

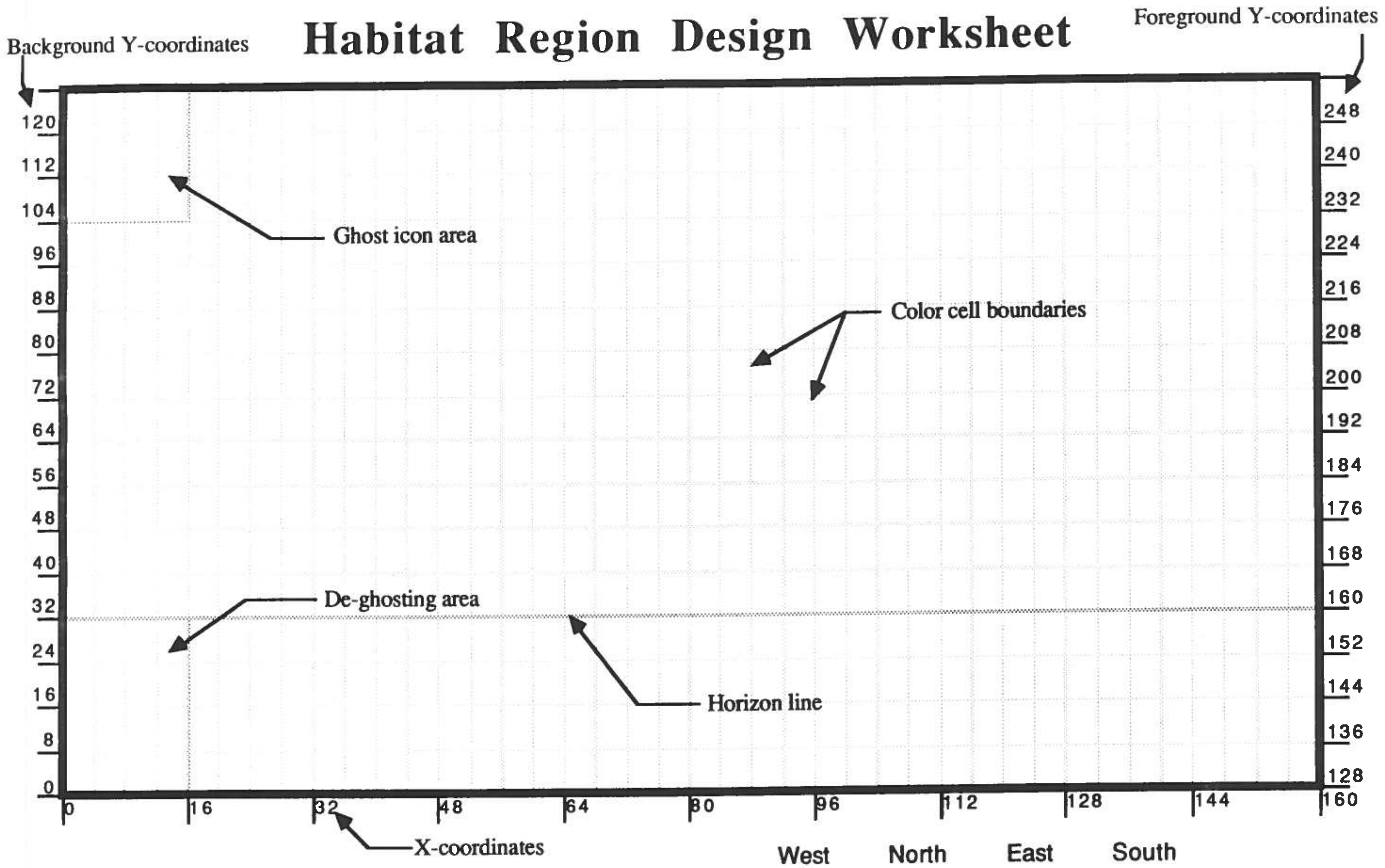
Day-to-day operations

- Automatic operations with Ghu
  - Prizes
  - Ongoing games
  - Rents and property management
- Batch processes to keep things tidy
- Running the Oracle
  - Publications and information distribution

Creating new experiences

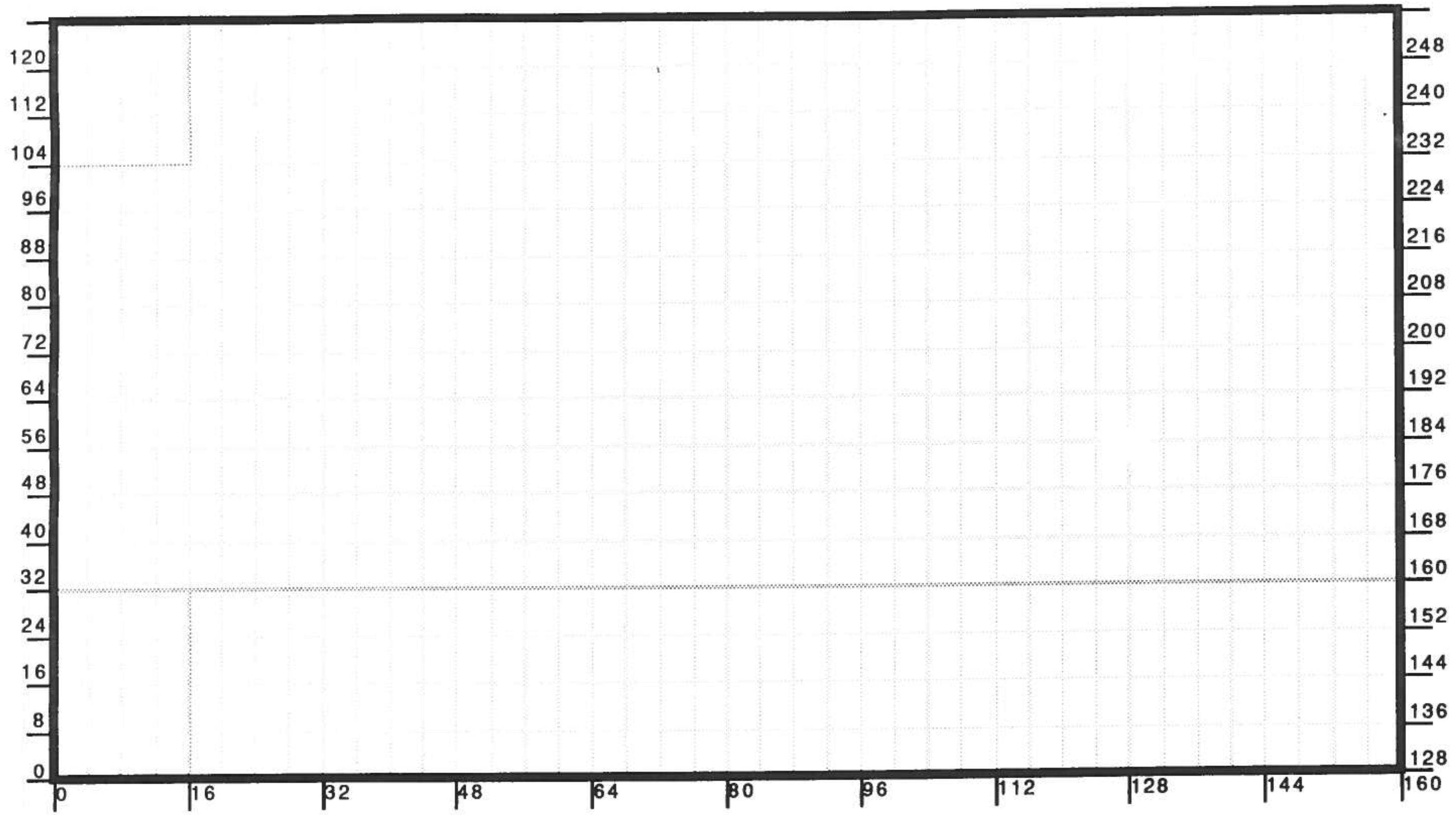
- New activities
- New places
- Some ideas never implemented
- Brainstorm session on experiences

Habitat Region Design Worksheet



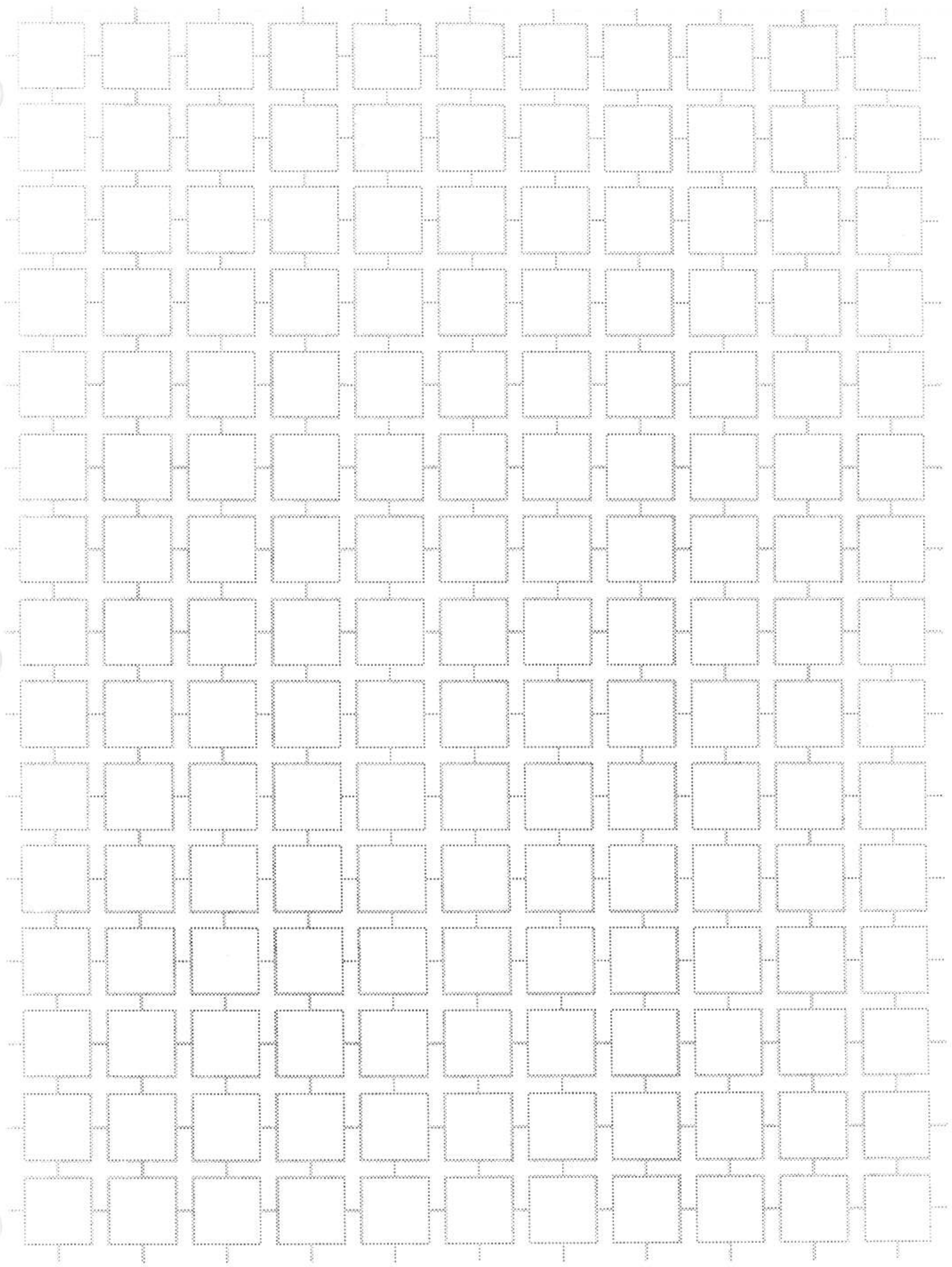
					West	North	East	South
Light level	_____	Orientation	_____	Neighbor	_____	_____	_____	_____
Weapons free?	_____	Avatar limit	_____	Restricted?	_____	_____	_____	_____
Theft free?	_____	Owner	_____	(Exit action)	_____	_____	_____	_____
Name	_____________			(Entry action)	_____	_____	_____	_____

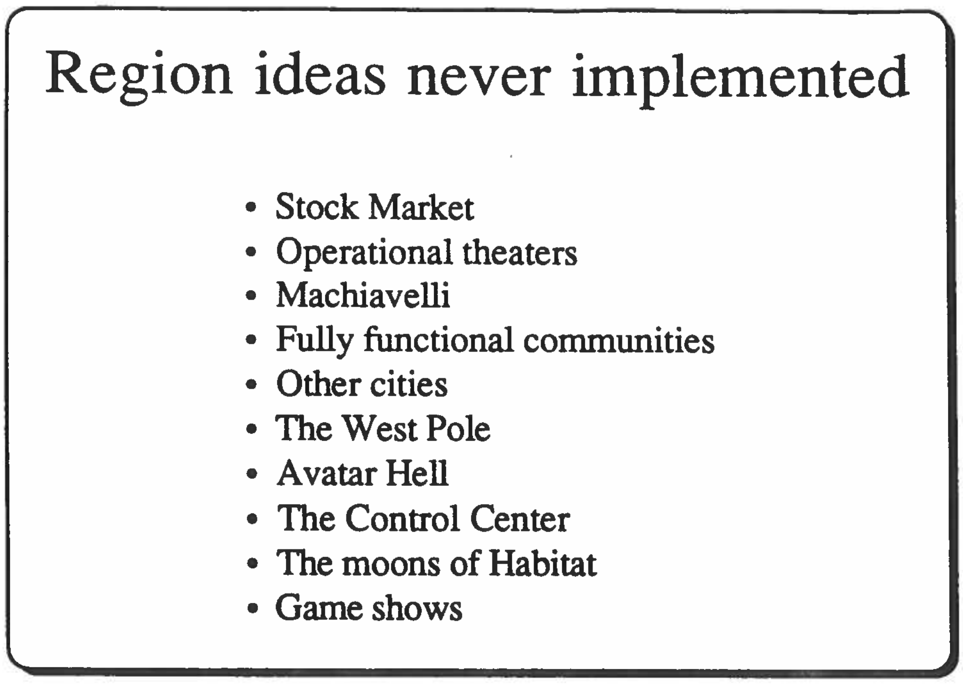
Habitat Region Design Worksheet



					West	North	East	South
Light level	_____	Orientation	_____	Neighbor	_____	_____	_____	_____
Weapons free?	_____	Avatar limit	_____	Restricted?	_____	_____	_____	_____
Theft free?	_____	Owner	_____	(Exit action)	_____	_____	_____	_____
Name	_____			(Entry action)	_____	_____	_____	_____

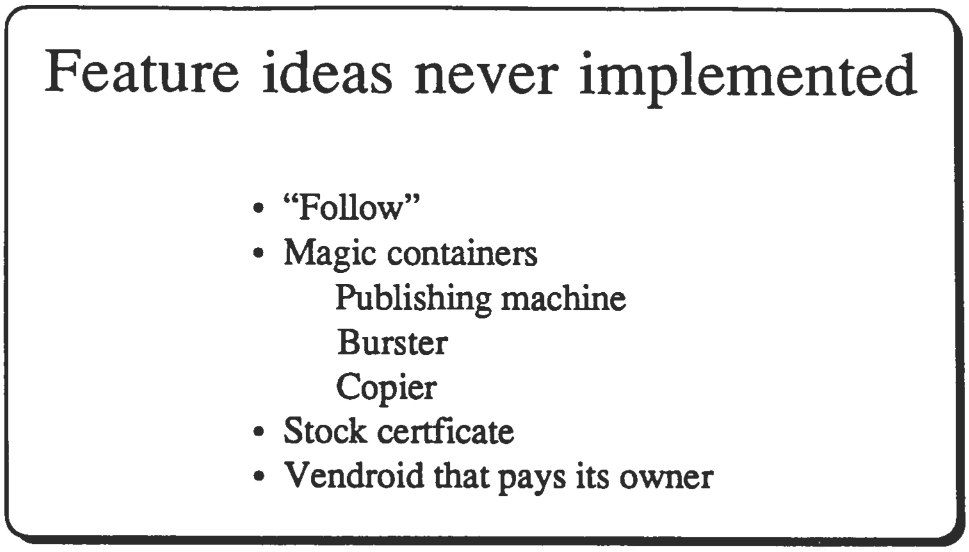
Habitat Region Map Worksheet





Region ideas never implemented

- Stock Market
- Operational theaters
- Machiavelli
- Fully functional communities
- Other cities
- The West Pole
- Avatar Hell
- The Control Center
- The moons of Habitat
- Game shows



Feature ideas never implemented

- “Follow”
- Magic containers
  - Publishing machine
  - Burster
  - Copier
- Stock certficate
- Vendroid that pays its owner

Brainstorm session on experiences

- What will make Japanese Habitat experiences different from American?

Rules, policies, and laws

- Maintaining the consistency of the user's worldview
- Keeping it fun
    - Fun for the users
    - Fun for the operators
        - If the operators aren't having fun it won't work
- Relationship to policies of company operating the host system
- Relationship to laws and regulations of the "real" world

Habitat Ephemera

Where Stuff Is

The following is an overview of the various Habitat directories. The detailed information about each chunk of stuff will be given later.

The Habitat source that we have developed is kept on the Quantum Stratus system under the user account `guest.lucas`. The important elements of the directory hierarchy are:

```
#d010>lucas
     microcosm
          Actions
          Classes
          Ghu
          Grabthese
          Linkable
          Misc
          Structs
     toolbox
```

`#d010>lucas` is the account's home directory. Not much of interest is here except for our abbreviations file, which is highly non-standard by Quantum's reckoning (we have tried to emulate the Unix environment as much as possible, rather than to use the Quantum Stratus abbreviation set; this causes no end of trouble when Quantum folks try to do things while logged in as us; if we had it to do over again...)

`#d010>microcosm` is the master source directory for Habitat. The name is historical, since Habitat used to be called "MicroCosm". In this directory are

1) various include files that are used by our code
2) directories for the rest of the source
3) command files and other miscellaneous working junk files

`#d010>microcosm>Actions` contains the source for the generic action routines. These are object behaviors which are shared among multiple object classes.

`#d010>microcosm>Classes` contains the source for the various object classes.

`#d010>microcosm>Ghu` is the working directory for Ghu development and maintenance.

`#d010>microcosm>Grabthese` is where we place updated source files for Janet to grab them and incorporate them into the production system.

`#d010>microcosm>Linkable` is where all the compiled object files for all the Habitat code lives (all the code that we maintain, that is).

`#d010>microcosm>Misc` contains miscellaneous other Habitat source files that defy categorization.

`#d010>microcosm>Structs` contains `.incl.pl1` files for the various different object class structs.

`#d010>toolbox` has various utilities that we use. The most notable thing to be found here is the release version of Ghu, but it also has a number of other minor utilities which we have found useful.

Overall Structure of the Host System

The Habitat system is built around an "object oriented" design philosophy. The entire Habitat world is a collection of "objects". Abstractly, an object is a set of code and data that together act like some concrete entity. For example, regions, avatars, vending machines, magic wands, heads and teleport booths are all different kinds of objects. Ideally, we would like some sort of programming environment/operating system that incorporated this object model into its fundamental basis for operations. However, the world is not ideal, so we have approximated an object-oriented system with a conventional system that has a sort of object-oriented form.

The host system code can be divided into three very broad categories.

The first could be called the "skeleton". This is the framework on top of which the Habitat system runs. The skeleton consists of the main code of the `regionproc`; the various external Habitat processes such as the database process, the hatchery, and so on; and the non-Habitat host code that forms the rest of the Q-Link system. All of this code was provided by Quantum and we will not describe it here except in the most summary outlines (though we will make reference to it where appropriate).

The second code category we'll call the "objects". This is the collection of routines which implement the definition and behaviors of the various different objects that make up the Habitat world. These follow a rather rigid format and can be described as a group. Along with the objects are a few sets of special code that handle major sub-systems. Notable among these are the help system, the magic system, and the curse system. In the interest of narrative clarity, these will be described in sections of their own.

The third category of code could be called the "helpers". These are various routines we have written to make the world work more smoothly for us. They are not, as a rule, essential to the operation of the system, but understanding them is essential to understanding the system. Fortunately, their job is to make things simpler rather than more complex.

Summary of Operation

The following section is a summary of the operation of the main body of the Habitat system. This is all described elsewhere in the early design documents, but we'll repeat it here in broad outline in order to provide some context for what follows.

The Habitat world consists of a set of regions, each of which is a place you can visit with your Avatar. Each region contains some number of objects. Each object belongs to a particular "class". The class determines, in effect, what sort of object it is: how it is to behave and how the data that describes its state are to be interpreted. In principle there are 256 possible object classes, numbered from 0 to 255. In practice only a little more than half of these are actually used. Each object is represented by a data record that is kept in the host's memory when the object is in an "active" region and in a disk database when the object is not in an active region. A region is said to be active when there are one or more online users (avatars or ghosts) in it.

The process of activating a region when the first user enters consists of reading the object records into memory from the appropriate databases ("databases" is plural because regions and avatars, being special in various ways, are kept in separate databases from the other objects). Deactivating a region consists of writing the records back out again when the last user leaves (for efficiency, we only bother to actually write those records which have changed in the interim). Each object in an active region is assigned a temporary identifier called a "noid" (short for Numeric Object IDentifier) which is a number in the range from 0 to 255. The region itself is represented by an object which always has noid 0. The `regionproc` keeps an internal table that maps from noids to the object records themselves.

Each transaction between the C64 and the host is couched in terms of the object model. Each message from the C64 is directed to a particular object in the region that the C64's user finds himself in. The first byte of the message, in fact (after stripping off the various telecommunications protocol bytes), is the noid of the object to which the message is addressed. The byte after that is a number that indicates what the C64 is requesting of the object. The remaining bytes, if any, are request-specific parameter information.

When a message arrives, the `regionproc` extracts the noid from the message and uses this to locate the record corresponding to the object itself. This record contains, among other things, the class number of the object. The class number is used as an index into another table kept by the `regionproc` that contains the "class definitions". Part of a class's definition is an array of pointers to procedure entry points that correspond to the class's various "behaviors". The request number (the second byte of the message) is used as an index into this array, and the indicated procedure is called. This procedure then carries out whatever action is appropriate for the given request, including transmitting a response message to the C64 if that is appropriate. Before calling the behavior procedure, the `regionproc` also sets up a number of standard pointers and global variables that the behavior can look at to find out about the environment in which it is executing.

All of the above machinations, with the exception of the behavior routines themselves, are performed by the "skeleton" code mentioned earlier. This is the code that was developed as Quantum's portion of the project.

The Objects and Class Definition

Each object is defined by two PL/1 source files. The first defines the procedures to initialize the object and implement the object's various behaviors, if any. The second is an include file that defines a PL/1 structure that describes the object's state information (i.e., the contents of the object's database record *after* having been read into memory).

The first file is called the "class" file and lives in the `Classes` directory. The second is the "struct" file and lives in the `Structs` directory. For a given class, say "`foo`", we would have the files `Classes>class_foo.pl1` and `Structs>struct_foo.incl.pl1`.

The class file defines a procedure `initialize_class_foo` that is called by the `regionproc` at system startup time. This procedure sets up the class table entry for this class so that it will work right when the time comes. This file also contains the definitions of any class-specific behavior procedures that may be required for the object. For example, here is a very simple class file which defines the `ball` class:

```
/*
 *   class_ball.pl1
 *
 *   Ball object behavior module for Habitat.
 *
 *   Chip Morningstar
 *   Lucasfilm Ltd.
 *   9-April-1986
 */

%include 'microcosm.incl.pl1';
%include 'defs_action.incl.pl1';

initialize_class_ball: procedure;

     %replace BALL_REQUESTS by 3;

     declare a(0:BALL_REQUESTS) entry based;
     declare class_ball_actions pointer;
     declare 1 ball based %include struct_ball;

     %replace I by CLASS_BALL;

     Class_Table(I).capacity = 0;
     Class_Table(I).max_requests = BALL_REQUESTS;
     Class_Table(I).alloc_size = size(ball);
     Class_Table(I).pc_state_bytes = 0;
     Class_Table(I).known = true;
     Class_Table(I).opaque_container = false;
     Class_Table(I).filler = false;

     allocate a set(class_ball_actions);
     Class_Table(I).actions = class_ball_actions;

     Class_Table(I).actions->a(HELP)  = generic_HELP;  /* 0 */
     Class_Table(I).actions->a(GET)   = generic_GET;   /* 1 */
     Class_Table(I).actions->a(PUT)   = generic_PUT;   /* 2 */
     Class_Table(I).actions->a(THROW) = generic_THROW; /* 3 */

end initialize_class_ball;
```

The following notes apply:

`microcosm.incl.pl1` is the general purpose Habitat include file. It declares all the basic global variables, types and constants that are used throughout the system. It should be included in just about everything.

`defs_action.incl.pl1` declares the entry points for the generic behavior routines found in the various files in the `Actions` directory. Note that many objects (this one among them) share common behaviors. For example, the code to put down or pick up an object is, with rare exceptions, the same regardless of the object's class. Thus, we have generic routines to handle the `GET` and `PUT` requests, instead of implementing these procedures anew in each object class.

`initialize_class_ball` is the required initialization routine for this class. It is called by the `regionproc` at system startup time. All classes MUST have a routine of this sort.

`BALL_REQUESTS` is the maximum request number that objects of class ball will be expected to receive. It is defined here as a convenient constant that you will see reference to in several places in the routine.

`struct_ball` is a string constant that is defined by the include file `defs_struct.incl.pl1` which is in turn included automatically by the include file `microcosm.incl.pl1`. This string constant expands to the file name for the "struct" file, mentioned above. The `defs_struct.incl.pl1` include file defines one of these string constants for each class. The idiom

```
declare 1 foo based %include struct_foo;
```

is a common one that will be seen again and again throughout the Habitat code.

`Class_Table` is a global table that is maintained by the `regionproc`. The primary purpose of this init procedure is to fill in the `Class_Table` entry for this class. This consists of assigning various properties and allocating and setting up the array of pointers to the behaviors.

`capacity` is the maximum number of objects that objects of this class may contain. For objects which are not containers (e.g., this one), this should be set to 0.

`max_requests` is the maximum request number that objects of this class will accept. If the `regionproc` receives a request to an object of this class that is greater than this number, it will drop the request on the floor and put a diagnostic message in the run-time log file (i.e., this should never happen).

`alloc_size` is the amount of memory to allocate for objects of this class when they are read from the object database at region activation time.

`pc_state_bytes` is the number of bytes of data from the in-memory record to send to the C64 when someone sees an object of this class (the number of bytes in addition to the 6 which are sent for every object regardless of class). note that objects may have state information on the host which is not revealed to the C64.

`known` is simply a flag that says, "Yes, this class exists". This is used in the course of various diagnostics.

`opaque_container` is a flag that is set to `true` if and only if the object is an opaque container, i.e., a container whose contents are not visible without explicitly looking inside it. This is `false` if the object is either not a container at all (the case here) or if the object is "transparent" (e.g., a table).

`actions` is the array of pointers to the behavior procedures for this class. Note that it must be allocated dynamically since its size may vary depending on how many behaviors the class has.

The particular elements of the `actions` array correspond to the various requests that the object will respond to. The request numbers are defined in the include file `defs_message.incl.pl1` which is included automatically by `microcosm.incl.pl1`. These requests are always interpreted relative to the object class (e.g., request 5 for class A does not mean the same thing as request 5 for class B). However, for the sake of consistency and diagnostics, we DO enforce the following conventions:

request 0 is always `HELP`
request 1 is always `GET`
request 2 is always `PUT`
request 3 is always `THROW`

if the class in question does not respond to one of these requests, it should set the corresponding array entry to `illegal`. In the case of the ball object, ALL of these requests are handled by the generic behaviors, and so there are no ball-specific behaviors defined here.

The struct file for this example looks like this:

```
/*
 *   struct_ball.incl.pl1
 *
 *   Struct stub for ball instance descriptor.
 *
 *   Chip Morningstar
 *   Lucasfilm Ltd.
 *   9-April-1986
 *
 */
,    2    common_head          like instance_head
; /* terminates struct header from include file */
```

This is a trivial struct file, since the class ball has no state information that is peculiar to the class. It merely has the common information that all objects have which is defined by the struct `instance_head` in the `microcosm.incl.pl1` file. For the record, it is:

```
/*
 *   instance_head.def.incl.pl1
 *
 *   The common header shared by ALL object instance descriptors.
 *
 *   Chip Morningstar
 *   Lucasfilm Ltd.
 *   9-April-1986
 *
 */
declare 1 instance_head       based,
          2    avatarslot     binary(15),
          2    obj_id         binary(31),
          2    noid           binary(15),
          2    class          binary(15),
          2    style          binary(15),
          2    x              binary(15),
          2    y              binary(15),
          2    position       binary(15),
          2    orientation    binary(15),
          2    gr_state       binary(15),
          2    container      binary(15),
          2    gr_width       binary(15),
          2    gen_flags(32)  bit(1);
```

The meanings of the various fields are described elsewhere.

An example of a slightly less trivial class definition is the pawn machine. Note the similarities of form with the definition of class ball:

```
/*
 *   class_pawn_machine.pl1
 *
 *   Behavior module for object class pawn_machine.
 *
 *   Chip Morningstar
 *   Lucasfilm Ltd.
 *   6-October-1986
 */

%replace PAWN_MACHINE_CAPACITY by 1;

%include 'microcosm.incl.pl1';
%include 'defs_helper.incl.pl1';
%include 'defs_action.incl.pl1';

initialize_class_pawn_machine: procedure;

     %replace PAWN_MACHINE_REQUESTS by 6;

     declare a(0:PAWN_MACHINE_REQUESTS) entry based;
     declare class_pawn_machine_actions pointer;
     declare 1 pawn_machine based %include struct_pawn_machine;

     %replace I by CLASS_PAWN_MACHINE;

     Class_Table(I).capacity = PAWN_MACHINE_CAPACITY;
     Class_Table(I).max_requests = PAWN_MACHINE_REQUESTS;
     Class_Table(I).alloc_size = size(pawn_machine);
     Class_Table(I).pc_state_bytes = 3;
     Class_Table(I).known = true;
     Class_Table(I).opaque_container = true;
     Class_Table(I).filler = false;
```

```
        allocate a set(class_pawn_machine_actions);
        Class_Table(I).actions = class_pawn_machine_actions;

        Class_Table(I).actions->a(HELP)   = generic_HELP;       /* 0 */
        Class_Table(I).actions->a(1)      = illegal;            /* 1 */
        Class_Table(I).actions->a(2)      = illegal;            /* 2 */
        Class_Table(I).actions->a(3)      = illegal;            /* 3 */
        Class_Table(I).actions->a(4)      = illegal;            /* 4 */
        Class_Table(I).actions->a(5)      = illegal;            /* 5 */
        Class_Table(I).actions->a(MUNCH)  = pawn_machine_MUNCH;/* 6 */
    end initialize_class_pawn_machine;

    pawn_machine_MUNCH: procedure;
        declare 1 self based(selfptr) %include struct_pawn_machine;

        if (adjacent(selfptr) & self.contents->c(0) ^= NULL) then do;
            if (pay_to(avatarptr, item_value(ObjList(self.contents->c(
                call n_msg_1(selfptr, MUNCH$, avatar.noid);
                call n_msg_1(null(), GOAWAY_$, self.contents->c(0));
                call destroy_contents(selfptr);
                call r_msg_1(TRUE);
                return;
            end;
            call r_msg_1(BOING_FAILURE);
            return;
        end;
        call r_msg_1(FALSE);
    end pawn_machine_MUNCH;
```

The following points are worthy of mention:

`defs_helper.incl.pl1` is an include file that declares a variety of "helper" routines that a behavior can call to perform various services. These will be discussed in greater detail below.

`capacity` is set to `PAWN_MACHINE_CAPACITY`, a constant that has no counterpart in class ball. This is because the pawn machine is container (capable of holding 1 object) and the ball is not.

Note that the actions array has a number of `illegal` entries, since the pawn machine is not a mobile object (i.e., it cannot be picked up and carried).

The pawn machine has one behavior of its own, which is defined here. Notice the naming convention used for behavior routines: `classname_REQUEST`. Generic behaviors (those corresponding to more than one class) have names of the form `generic_REQUEST`.

The behavior itself contains many items worth discussing, but we will cover them in the following section when we explain the execution environment that behavior procedures live in.

The pawn machine's struct file looks like this:

```
/*
 *   struct_pawn_machine.incl.pl1
 *
 *   Struct stub for pawn_machine instance descriptor.
 *
 *   Chip Morningstar
 *   Lucasfilm Ltd.
 *   6-October-1986
 *
 */
,    2    common_head          like instance_head,
     2    contents             pointer,
     2    class_specific       ,
           3   open_flags      binary(15),
           3   key_hi          binary(15),
           3   key_lo          binary(15);
```

The `contents` field appears only in those objects which are containers. It is filled in automagically by the `regionproc` when the container is opened. This class has class specific fields which are defined here. The fields shown here are required for any container, though in the case of the pawn machine they are a formality, since it can never be opened or closed, locked or unlocked, by a player. (These container-specific fields will be discussed later in more detail).

The Behavior Execution Environment

In addition to a variety of "helper" routines, which will be discussed in the next section, there are a number of important elements in a behavior procedure's execution environment that require explanation. Most of these are global variables that are declared by the include file `microcosm.incl.pl1` and set by the `regionproc` before the behavior is called.

```
%replace THE_REGION by 0;
```

This is (always) the noid of the region object in this region.

A series of string constants of the form `struct_thingname` are defined to enable easy declaration of common data types. This was discussed in greater detail above.

```
declare 1 o based %include struct_gen_object;
```

`struct_gen_object` is a generic object header that can be used to refer to the common state information of all objects. The based type `o` lets us access such information with minimal fuss.

```
declare 1 u based %include struct_user;
```

similarly, there is a "user struct" that contains user information that is pointed to by a field of avatar objects. This based struct lets us access its fields and thus do rare but sometimes necessary things that require access to the user's queue. This struct looks like:

```
/*
```

```
 *   struct_user.incl.pl1
 *
 *   Struct stub for UserList structure.
 *
 *   Chip Morningstar
 *   Lucasfilm Ltd.
 *   9-April-1986
 *
 */
,    2    U_Name               character(10) varying,
     2    U_Id                 binary(31),
     2    U_Q_Id               binary(31),
     2    U_Q                  pointer,
     2    U_version            binary(15),
     2    object_slot          binary(15),
     2    esp                  ,
          3 to_uid             binary(31),
          3 to_qid             binary(31),
          3 que                pointer,
          3 lines              binary(15),
     2    last_mail_ts         binary(31),
     2    auto_destination     binary(31),
     2    auto_mode            binary(31),
     2    flags                ,
          3    U_mail               bit(1),
          3    cr_pending           bit(1),
          3    online               bit(1),
          3    incoming             bit(1),
        . 3    new_session          bit(1),
          3    ck_last_login        bit(1),
          3    filler               bit(10);
```

The useful fields are typically `U_Name`, `U_Id`, and `online`. The user structs may be accessed via

```
declare UserList(UsersPerRegion) pointer;
```

which points to the various users. You can find out the index into the `UserList` for a particular avatar via the avatar object's `avatarslot` field.

You can map a noid to a pointer to an object via the global `ObjList`:

```
declare ObjList(0:255) pointer;
```

entries in this list are index by noid and will be `null()` if there is no object corresponding to a given noid..

```
declare c(0:255) binary(15) based;
```

is declared so we can access the contents of a container object. Container objects always have a `contents` field which is simply a pointer to an array of this form. Thus, if `foo` is a container object, we can refer to, say, the third item in `foo` as:

```
foo.contents->c(2)
```

note that this value is a noid, not an object pointer. If there is no object in the particular container slot, this value will be `NULL` (i.e., 0). To get a pointer to the object itself you would have to say

```
ObjList(foo.contents->c(2))
```

being careful, of course, to make sure that the object exists (i.e., that the pointer from the `ObjList` is not `null()`) before you try to do anything with it.

```
declare avatarptr pointer external;
declare 1 avatar based(avatarptr) %include struct_avatar;
declare selfptr pointer external;
declare 1 self based(selfptr) %include struct_gen_object;
```

Before any behavior is executed, `avatarptr` (and thus `avatar`) is set to point to the object record for the object corresponding to the avatar whose C64 issued the request that we are processing. (Warning: this will be invalid if the user is a ghost.) Similarly, `selfptr` (and thus `self`) is set to point to the object to which this request was sent. By this means any object can refer to itself as `self` in its behavior code. Often, this declaration of `self` is overridden by behaviors in order to make `selfptr` point to a different type of struct than `struct_gen_object` (e.g., a flashlight behavior would want `selfptr` to be declared as pointing to a `struct_flashlight`).

```
declare request_string character(646) varying external;
declare request(258) character(1) defined(request_string);
%replace FIRST by 3;
%replace SECOND by 4;
%replace THIRD by 5;
%replace FOURTH by 6;
%replace FIFTH by 7
```

Before executing the behavior, `request_string` is assigned the request message itself (after the telecommunications protocol information is stripped off). `request` lets us individually index the bytes of the request. `FIRST`, `SECOND`, etc. are defined so that we can neatly refer to the parameters of the request (remember that the first byte is the noid to which the request is addressed and the second byte is the request number). Thus, the second parameter byte of a request would be

```
request(SECOND)
```

often (usually, in fact) you will want the byte itself as a number, not as a character, so you will frequently see the idiom

```
rank(request(SECOND))
```

which simply gets the byte as an integer.

The region itself is not represented on the host as an object, unfortunately. Information about the region is found in various globals. Notable are:

```
declare   Region binary(31);
```

The current region number.

```
        declare Region_name character(20);
```

The current region name.

```
        declare total_ghosts binary(15);
```

The number of ghosts in the region, and the notable sub-struct:

```
        declare 2  current_region,
             3 lighting                  binary(15),
             3 depth                     binary(15),
             3 neighbor(4)               binary(31),
             3 exit_type(4)              binary(15),
             3 restriction(4)            bit(1),
             3 nitty_bits(28)            bit(1),
             3 max_avatars               binary(15),
             3 owner                     binary(31),
             3 entry_proc                binary(15),
             3 exit_proc                 binary(15),
             3 class_group               binary(15),
             3 orientation               binary(15),
             3 object_count              binary(15),
             3 space_usage               binary(15),
             3 town_dir                  character(1),
             3 port_dir                  character(1);
```

full of all kinds of useful information. The global

```
        declare DayNight binary(15) external init(0);
```

contains the global illumination level, which controls whether it is day or night. For the time being it is always day, but this may change in the future.

Avatar, region and object records each have arrays of general purpose bit-flags called `nitty_bits`. (Actually, avatars and objects have two sets of flags, `nitty_bits` associated with the object or avatar record and `general_flags` associated with the `instance_head` struct.) These bits are available for general use as needed. However, some are already allocated and you should avoid stepping on them:

```
        /* instance_head general flag constants */
        %replace RESTRICTED by 1;
        %replace MODIFIED   by 2;
```

The `RESTRICTED` bit means that the object can't be taken out of a restricted region exit. The `MODIFIED` bit means that the object has been changed and should be written to the database (it is interrogated when the region is deactivated).

```
        /* region nitty_bits constants */
        %replace WEAPONS_FREE by 1;
        %replace STEAL_FREE by 2;
```

These make a region weapons free or theft free.

```
        /* avatar nitty_bit constants */
        %replace CURSE_IMMUNE by 32
        %replace VOTED_FLAG by 3;
        %replace GOD_FLAG by 4;
        %replace MISC_FLAG1 by 5;
        %replace MISC_FLAG2 by 6;
        %replace MISC_FLAG3 by 7;
```

`CURSE_IMMUNE` is used to keep one from being infected more than once. `VOTED_FLAG` is used to prevent people from voting twice in an election. `GOD_FLAG` is used for superuser avatars. `MISC_FLAG`s are temporaries.

```
        /* object nitty-bits constants */
        %replace DOOR_AVATAR_RESTRICTED_BIT by 32;
        %replace DOOR_GHOST_RESTRICTED_BIT by 31;
```

Setting these on door objects allows you to prevent avatars or ghosts from going through the door.

Helper Routines

All of the "helper" routines are contained in the `Misc` directory. Most of them are in the large file `helpers.pl1`. We will summarize them here.

```
        accessable: procedure(objptr) returns(bit(1));
```

given a pointer to an object, returns `true` iff the object is accessable to `avatar`, i.e., if it is adjacent to the avatar or in an open container which is adjacent or in a container which is in a container which is adjacent, etc.

```
        announce_object: procedure(objptr);
```

given a pointer to an object, broadcasts a `HERE_IS` message to everyone in the current region describing the object. Takes care of building the description vector and everything. For use when you create a new object.

```
        at_water: procedure returns(bit(1));
```

Obsolete.

```
        drop_object_in_hand: procedure(whoptr);
```

Takes the object in the hand of the avatar pointed to by `whoptr` and drops it on the ground at that avatar's `(x,y)` position in the region. If the avatar is empty handed, this is a no-op. It takes care of sending out messages so everyone in the region knows that this has happened.

```
        auto_teleport: procedure(whoptr, where, entry_mode);
```

Teleports the avatar pointed to by `whoptr` to region number `where` using the given entry mode. Works asynchronously, i.e., this is what you do to move an avatar who isn't expecting to be moved. Takes care of informing everyone involved, including the victim and anyone in the region from which he departs. Allowed entry modes are:

```
%replace WALK_ENTRY by 0;
%replace TELEPORT_ENTRY by 1;
%replace DEATH_ENTRY by 2;
```

in general, walking should never be used with `auto_teleport`. `DEATH_ENTRY` only applies when the avatar is being killed, which you should never be doing yourself (use `kill_avatar` instead). In other words, always use `TELEPORT_ENTRY`.

```
available: procedure(container_noid, x, y) returns(bit(1));
```

Returns `true` iff the container slot `(x,y)` in the container with the given noid is empty, i.e., it is available to have something put in it. Ordinarily, the `y` value is the only value that matters in terms of indicating container slots. The `x` parameter only matters with regions (and for regions `available` always returns `true`), thus you should usually call `available` with an `x` value of 0 and a `y` value of whatever container slot you are interested in.

```
cancel_event: procedure(event);
```

Obsolete.

```
change_containers: procedure(obj_noid, new_container_noid,
     new_position, checkpoint) returns(bit(1));
```

Tries to move the object indicated by `obj_noid` from its present location to slot `new_position` in the container indicated by `new_container_noid`. If `checkpoint` is `true` it will checkpoint the object to the database after moving it. It returns `true` iff it was able to move the object (moving an object out of an opaque container increases the C64 memory usage and so will fail if it would overflow memory).

```
change_region_fail: procedure(who_noid);
```

This is a procedure that the `regionproc` calls whenever a region change attempt fails. Its job is to undo various things that are done in preparation for a region change in the expectation that it will succeed. Right now all that really needs to be worried about are the lights, but this routine is a nice hook in case something comes up in the future.

```
dequeue_player: procedure(whatptr);
```

Obsolete.

```
destroy_contents: procedure(containerptr);
```

Destroys (i.e., removes from the world) all the objects contained in the container object pointed to by `containerptr`.

```
empty_handed: procedure(whoptr) returns(bit(1));
```

Returns `true` iff the avatar pointed to by `whoptr` has nothing in its hand.

```
enqueue_player: procedure(whatptr);
```

Obsolete.

```
getable: procedure(objptr) returns(bit(1));
```

Returns `true` iff the the object pointed to by `objptr` is "getable", i.e., if it would be possible for `avatar` to pick it up (i.e., it is accessable and it is of a type that it is possible to pick up).

```
ghost_say: procedure(obj_noid, text);
```

Like `object_say`, (see below) but for use when the user is a ghost and so `avatar` is invalid.

```
goto_new_region: procedure(whoptr, where, direction, transit
```

Makes the avatar pointed to by `whoptr` go to region number `where` with the transition type (as explained above under `auto_teleport`) of `transition_type`. If the avatar is walking, `direction` indicates which direction he is going. This procedure is called for any region transition. For the asynchronous case it is called by `auto_teleport`. For the synchronous case it is called by `avatar_CHANGE_REGION`.

```
grabable: procedure(objptr) returns(bit(1));
```

Returns `true` iff the object pointed to by `objptr` may be grabbed from another avatar's hand by `avatar` (i.e., if it is in the hand of an adjacent avatar and is of a type that is allowed to be grabbed).

```
holding: procedure(objptr) returns(bit(1));
```

Returns `true` iff `avatar` is holding the object pointed to by `objptr`.

```
holding_class: procedure(class_number) returns(bit(1));
```

Returns `true` iff `avatar` is holding an object of class `class_number`.

```
inc_record: procedure(whoptr, record);
```

Increments (by 1) the Hall of Records entry for record `record` and avatar `whoptr`.

```
item_value: procedure(itemptr) returns(binary(15));
```

Returns the intrinsic value (e.g., what a pawn machine will pay for it) of the object pointed to by `itemptr`.

```
kill_avatar: procedure(victimptr);
```

Kills the avatar pointed to by `victimptr`. Takes care of notifying all interested parties, including the victim. Works asynchronously.

```
lights_off: procedure(whoptr);
```

Turns down the lights in the current region on the assumption that the avatar pointed to by `whoptr` is leaving (lights go down if he is carrying a lit flashlight out of the region).

```
lights_on: procedure(whoptr);
```

Turns the lights back up.

```
lookfor_string: procedure(sourcestring,substring) returns(bi
```

Like `index`, but performs a case-independent match flashlight out of the region).

```
lights_on: procedure(whoptr);
```

Turns the lights back up.

```
lookfor_string: procedure(sourcestring,substring) returns(bi
```

Like `index`, but performs a case-independent match.

```
lowercase: procedure(mixedstring) returns(character(256) var
```

Returns a copy of the string `mixedstring` with all upper case characters converted to lower case.

```
max_record: procedure(whoptr, record, value);
```

Sets the Hall of Records record for record `record` and avatar `whoptr` to the maximum of its current value and `value`.

```
object_broadcast: procedure(obj_noid, text);
```

Broadcasts an `OBJECT_SPEAK` message to everyone in the region to make the object `obj_noid` say in a word balloon the string `text`.

```
object_say: procedure(obj_noid, text);
```

Sends an `OBJECT_SPEAK` message to the current avatar (only) to make the object `obj_noid` say in a word balloon the string `text`.

```
pay_to: procedure(whoptr, amount) returns(bit(1));
```

Try to pay `amount` tokens from `avatar` to the avatar pointed to by `whoptr`. Returns `true` iff it was able to do this (i.e., if `avatar` had sufficient tokens in hand to pay the amount).

```
random: procedure(top) returns(binary(15));
```

Returns a random number in the range from 1 to top.

```
random_time_in_the_future: procedure returns(binary(31));
```

Obsolete.

```
region_entry_daemon: procedure(direction, transition_type,
     old_orientation, from_region);
```

Called by the regionproc on entry to a region. This is the place to put any region entry-dependent actions. direction is the direction the avatar is walking, if he is walking. transition_type is the transition type. old_orientation is the orientation of the region departed from. from_region is the region number of the region departed from.

```
schedule_event: procedure(objptr, event_procedure, delay)
     returns(pointer);
```

Obsolete.

```
set_record: procedure(whoptr, record, value);
```

Sets the Hall of Records record record for avatar whoptr to value value.

```
spend: procedure(amount) returns(binary(15));
```

Tries to spend amount tokens (out of hand) on behalf of avatar. Returns 0 if unsuccessful, 1 if successful.

```
spend_check: procedure(amount) returns(bit(1));
```

Returns true iff avatar could successfully spend amount tokens out of hand.

```
tget: procedure(tokenptr) returns(binary(31));
```

Returns the denomination of the token object pointed to by tokenptr.

```
tset: procedure(tokenptr, amount);
```

Sets the denomination of the token object pointed to by tokenptr to amount.

```
unescape_string: procedure(string);
```

Expands all the Ghu character string escape sequences (\etc) in the string string (used in God-tool magic).

```
vectorize: procedure(objptr) returns(character(256) varying)
```

Generates a contents vector for the object pointed to by objptr.

```
wearing: procedure(objptr) returns(bit(1));
```

Returns `true` iff `avatar` is wearing the head object pointed to by `objptr`.

More Helper Routines: Sending Messages

To facilitate telecommunications, a variety of messaging routines are defined in `Misc>messages.pl1`. These send messages from the host to the C64. There are many of these routines, but they fall into four functional groups. Within each functional group, the routines are distinguished only by their parameters (in fact, if PL/1 had a variable argument-count procedure call mechanism, there would only be four routines at all).

```
n_msg_XXX: procedure(to_objectptr, msg_number, args...);
```

Sends message number `msg_number` with arguments `args`... to the object pointed to by `to_objectptr` on all C64's in the region EXCEPT the one belonging to `avatar`. I.e., `n_msg` == "neighbor message". If `to_objectptr` is `null()`, the message is sent to the region object.

```
b_msg_XXX: procedure(to_objectptr, msg_number, args...);
```

Similarly sends a message to all C64's in the region with no exceptions. I.e., `b_msg` == "broadcast message".

```
p_msg_XXX: procedure(to_objectptr, to_whomptr, msg_number, a
```

This one sends the message only to the machine whose user is the avatar pointed to by `to_whomptr`. If `to_whomptr` is `null()`, it assumes that you mean the current avatar but that the current avatar is a ghost. I.e., `p_msg` == "point-to-point message".

```
r_msg_XXX: procedure(args...);
```

Sends a reply message to the current request to the currently requesting avatar.

In all of the above, `XXX` determines the format of `args....` `XXX` is either a digit, in which case `args...` consists of that many integer (`bin(31)`) arguments, or a digit followed by '`_s`', in which case `args...` consists of that many integer arguments followed by a single character string argument. For example,

```
n_msg_0: procedure(to_objectptr, msg_number);
n_msg_1: procedure(to_objectptr, msg_number, arg1);
n_msg_2: procedure(to_objectptr, msg_number, arg1, arg2);
n_msg_2_s: procedure(to_objectptr, msg_number, arg1, arg2, a
n_msg_s: procedure(to_objectptr, msg_number, argstr);
```

etc.

More Helper Routines: Width and Collision Detection

A small number of routines relating to collision detection have been isolated in the file `Misc>width.pl1`. These are separate because they have a large run-time table that they must refer to. This table describes the graphic characteristics, in terms of size and placement, of all the various objects. It is generated automatically by our C64 disk database generation tools which create the include file `width.incl.pl1` that is included here.

```
        adjacent: procedure(objptr) returns(bit(1));
```

Returns `true` iff `avatar` is adjacent to the object pointed to by `objptr`.

```
        check_path: procedure(target_noid, x, y, new_x, new_y, flip_
```

Performs a collision detection check on a trajectory from location `(x,y)` to the object `object_noid`. The trajectory is a city-block path, i.e., all horizontal movement followed by all vertical movement. `new_x` and `new_y` are where we wound up. They will be the same as `x` and `y` if there was no collision, or the point of collision if we hit something. It will first try vertical-then-horizontal movement. If it hits something it will then try horizontal-then-vertical movement. If the second try succeeds, it will set `flip_path` to `true` to indicate that this happened.

```
        elsewhere: procedure(objptr) returns(bit(1));
```

Returns `true` iff the object pointed to by `objptr` is neither adjacent nor accessable.

```
        here: procedure(objptr) returns(bit(1));
```

Returns `true` iff the object pointed to by `objptr` is adjacent, accessable, or in hand.

More Helper Routines: Bit Manipulation

Since PL/1's facilities for performing bit manipulation on integers are dreadful, we have written some routines to do this for us. All of the following operate on `bin(15)` integers, NOT bit strings. These are defined in `Misc>bits.pl1`.

```
        clear_bit: procedure(num, the_bit);
```

Clears (sets to 0) bit number `the_bit` in `num` (bits are numbered with the least significant bit as 0 and the most significant bit as 15).

```
        set_bit: procedure(num, the_bit);
```

Sets (sets to 1) bit number `the_bit` in `num`.

```
        test_bit: procedure(num, the_bit) returns(bit(1));
```

Returns `true` iff bit number `the_bit` of `num` is 1.

```
        and_bit: procedure(num1, num2) returns(binary(15));
```

Does a bitwise AND of `num1` and `num2` and returns it.

```
        or_bit: procedure(num1, num2) returns(binary(15));
```

Does a bitwise OR of `num1` and `num2` and returns it.

More Helper Routines: Capacity Monitor

A series of procedures maintain a model in the host of the C64's memory capacity utilization. They use the include file `capacity.incl.pl1` which is generated by our C64 disk database creation utilities. These routines are defined in `Misc>capacity_monitor.pl1`.

```
note_object_creation: procedure(class_number, style);
```

Notes that an object of class `class_number` and style `style` as been added to the region.

```
note_object_deletion: procedure(class_number, style);
```

Similarly notes the removal of such an object.

```
reconstruct_memory_usage: procedure;
```

Rebuilds the memory usage model from scratch.

We have not describes the remaining routines in `capacity_monitor.pl1` because they are local to that file only.

Generic Actions

Many different classes of objects have, at least in part, the same behavior. For the common cases we have created separate generic behavior procedures. These live in the `Actions` directory and are grouped into files functionally. Because of a problem with the Stratus debugger, we needed to somehow reduce the number of object files that were linked into the program, so the `Actions` files are handled in a way that is a little peculiar: There is a file named `actions.pl1` which simply `%includes` the various sources. Since the brain-damaged Stratus PL/1 compiler insists that include files must have names ending in `.incl.pl1`, there are a bunch of .incl.pl1 files in Actions directory which are simply links to the corresponding `.pl1` files. Here is what is here:

```
actions_clothing.pl1:
        generic_WEAR
        generic_REMOVE
```
Behaviors for putting on and taking off "clothing". The reference to "clothing" is historical. All that is left to put on or take off are heads, which use these routines.

```
actions_container.pl1:
        generic_CLOSECONTAINER
        generic_OPENCONTAINER
```
Behaviors for opening and closing containers. These worry about locks and keys too.

```
actions_door.pl1:
        generic_CLOSE
        generic_OPEN
```
Similarly, behaviors for opening and closing doors.

```
actions_gpt.pl1:
        generic_GET
        generic_PUT
        generic_THROW
```
The most common behaviors of all, for picking up, putting down, and throwing objects.

```
actions_help.pl1:
        generic_HELP
```
Behavior for getting help. Objects which only require a fixed-string help message use this. The bulk of this procedure is a giant text array with one entry for each possible class. I'm sure this wastes a lot of memory, but PL/1 doesn't allow us to declare static text arrays properly. More on help below.

```
actions_mail.pl1:
        generic_READMAIL
        generic_SENDMAIL
```
Behavior for mail. Obsolete, I think.

```
actions_oracle.pl1:
        generic_ASK
```
Behavior for talking to oracular things.

```
actions_switch.pl1:
        generic_OFF
        generic_ON
```
Behavior for turning switchable objects on or off.

```
actions_weapon.pl1:
        generic_ATTACK
```
Behavior for using weapons.

Major Subsystems

There are three major subsystems underneath the behavior code which are complex enough to deserve special discussion on their own. These are the mechanisms for magic, sensors and drugs; the help system; and the "curse" system.

Magic, Sensors and Drugs

Magic objects, sensors, and drugs all employ essentially the same mechanism. These classes are distinguished by the fact that a given type of object can have one of a number of possible (very different) behaviors. In each case, the object record contains a type number that indicates just exactly what sort of magic item, sensor or drug the object is. This number is used as an index into an array of procedure entry points which fans out to a number of procedures which then implement the specified function. Beyond this, there are some slight differences between the three types of variable-behavior object:

Sensor routines live in `Classes>class_sensor.pl1`. The dispatch array is initialized by the procedure `initialize_sensors` which must be called by the `regionproc` at system start-up time. The sensor routines are called by the class sensor behavior routine `sensor_SCAN`. Such routines should look about the region for some characteristic of interest and then return a 1 or 0 depending on whether or not they find it. This success/failure value is then transmitted back to the C64 by `sensor_SCAN`.

Drug routines live in `Classes>class_drugs.pl1`. The dispatch array is initialized by the procedure `initialize_drugs` which must be called by the `regionproc` at system start-up time. The drug routines are called by the class drugs behavior routine `drugs_TAKE`. By convention, such routines should take some action effecting the player's avatar `avatar`. In general, they should not effect anyone else's avatar or the region environment. Drug routines do not have to worry about sending a response message to the player, as this will be taken care of by `drugs_TAKE`, though they DO have to worry about sending any asynchronous notification messages regarding any specific actions they perform. `drugs_TAKE` also worries about whether there are any pills left in the pill bottle when the user tries to take one, and about decrementing the pill count after one is taken.

Magic routines live in `Misc>magic.pl1`. The dispatch array is initialized by the procedure `initialize_magic` which must be called by the `regionproc` at system start-up time. The magic routines are called by the generic behavior routine `generic_MAGIC` which, violating the above described conventions about the `Actions` directory, is also located in `Misc>magic.pl1`. `generic_MAGIC` sends out an unconditional success response and dispatches to the appropriate magic routine. Thus, magic routines are running free of the C64 which has already received a response to its request. This is significant if the user issues some other request in the meantime (such as leaving the region).

There are actually two types of magic objects, "switches" and portable magic items. In the case of the latter, a parameter is supplied by the C64 when requesting magic action that is the noid of the object or avatar at which the player is pointing with his cursor when he issued the request. This allows the action of magic objects to be directed at or against something specific. The helper routine

```
        avatar_target_check: procedure(targetptr) returns(bit(1));
```

is defined in `Misc>magic.pl1` to check if the thing pointed at by the user is another avatar, since frequently one wishes to have magic which operates on avatars only. The range of possible actions that a magic routine may take is almost unlimited; the interested reader is advised to look at the source file `Misc>magic.pl1` itself for examples of the sorts of things we can and do do with magic.

Curses

A less important but still significant sub-system is the curse mechanism. This is what we use to implement "cooties" as well as other sorts of plagues. A curse temporarily modifies the attributes of an avatar. Each avatar record has two fields for dealing with curses, `curse_type` and `curse_count`. Normally, `curse_type` is 0, meaning no curse. However, if `curse_type` is not 0, the avatar has some temporary attribute that he is (probably) trying to get rid of. An avatar becomes cursed by a mechanism that varies with the type of curse. Typically it is started by some sort of magic. The routine that starts the curse must set the curse fields appropriately. To help, the file `Misc>curses.pl1` defines the procedure

```
        activate_head_curse: procedure(victimptr, curse_type) return
```

This procedure attempts to inflict the curse `curse_type` on the avatar `victimptr`. It returns `true` iff it succeeds. This procedure specifically deals with curses whose manifestation is a weird head of some sort. It takes care of notifying the player that he has a new head.

Curses are typically contagious. The transmission of curses is handled by the routine

```
        curse_touch: procedure(curserptr, curseeptr);
```

which is called by `avatar_TOUCH` when one player tags another while cursed. This routine transmits the curse (if it is contagious) from the avatar `curserptr` to the avatar `curseeptr` and decrements (if appropriate) `curserptr`'s `curse_count` field. When this counter runs down to zero the curse is removed and the avatar's old head is restored. By the way, when an avatar loses a curse a bit is set in the avatar record that makes him immune to getting it again until we reset the game. Of course, the curse is only transmitted by a tag if the victim is not himself immune by this means.

By controlling the initial setting of `curse_count` when an avatar is given a curse, you can manipulate the nature of the spread. Setting it to one gives a curse that passes from player to player, such as cooties. Setting it to a small number (such as two) causes a plague that spreads exponentially. Setting it to a large number causes it to infect the whole population eventually.

Both the above mentioned routines in `Misc>curses.pl1` have case statements in them that vector on the curse type, so that curse specific actions may be taken.

Help

The help system is invoked when the player presses the F7 key. The basic action to be taken by any help behavior is simply to send out a response message with a character string of up to 114 characters in length. However, owing to the variety of things we might want to say about an object in its help message, there are a number of complications.

If the help message associated with a particular class of object can be expressed in a string of up to 114 characters whose text never changes, then you should set the class's HELP behavior to `generic_HELP`. `generic_HELP` contains an array of strings, which it indexes by class number. When called, it looks up the object's class and transmits the appropriate string. There are a few special purpose entries in this array, however. If a class does not exist its help array entry should be `'-'`. If a class exists but does not use the `generic_HELP` routine, its help array entry should be `'i'`. In both cases it triggers an error diagnostic, since these help messages should never be encountered. If you haven't gotten around to figuring out what a class's help message should be, set its entry to `'u'`. This will give an appropriate apology for there not being any help. Finally, if an object is a non-functional scenic object, set its help entry to `'s'`. This will give help that describes how to use HELP.

If an class's help information cannot be expressed in 114 characters or if it must vary depending on other state information, then the class needs its own help behavior. Long help messages can be accomplished simply by breaking the help text into multiple messages. The first is a response message that answers the HELP request itself, while the remaining messages should be sent with calls to `object_say`. Variable content messages can simply be built up as needed and sent via the same means.

Certain types of objects have HELP information requirements which are more complex still. In particular, classes which exhibit large stylistic or functional variations require special treatment. Such classes include drugs, sensors, knick-knacks, and all magic items. These classes have arrays complex still. In particular, classes which exhibit large stylistic or functional variations require special treatment. Such classes include drugs, sensors, knick-knacks, and all magic items. These classes have arrays of messages of their own which are indexed by style, magic type, or whatever other parameter is appropriate for the class in question. When adding a new type of magic or a new sensor, then, you must also add an entry to the appropriate help text array. By the way, it is our convention that the help for sensors and drugs should be descriptive while the help for a magic item should be phrased as a riddle or cryptic remark that only hints at what the magic item does.

The final complexity is introduced by vending machines. Vending machines issue help message which describe not only how to use the vending machine but what it is that is for sale. As with `generic_HELP`, `vendo_HELP` (located in `Classes>class_vendo_front.pl1`) maintains an array of help messages. This array is index by the class of the object on display in the vending machine. It works pretty much just like `generic_HELP` except that the messages are limited to 80 characters. Also, like `generic_HELP`, there are some special entries in the array which cause special action to be taken. Not only do we have to deal with non-existent classes and such, but the problem of the variability of knick-knacks, magic items, and so forth creeps up on us once again. To handle these cases, a number of procedures with names of the form `classname_vendo_info` are defined which return appropriate character strings based on a pointer to the object on display. `vendo_HELP` calls these based on the entries in the vendo help message array. Here are the special entries in this array:

`'-'` means that the object class does not exist. Hitting such an entry is a run-time system error.
`'i'` means that the object is a class that may not be placed in a vending machine. Hitting such an entry is a run-time system error.
`'b'` uses whatever is returned by `book_vendo_info`
`'d'` uses whatever is returned by `drugs_vendo_info`
`'m'` uses whatever is returned by `magic_vendo_info`
`'k'` uses whatever is returned by `key_vendo_info`
`'s'` uses whatever is returned by `sensor_vendo_info`

In the case of drugs, magic and sensors, the information used by the `xxx_vendo_info` routine and the information used by the `xxx_HELP` routine comes from the same array which does double-duty.

What To Do When Sister Picks Up the Phone

and solutions to other multi-user network game design problems

by

F. Randall Farmer

There are several tough questions that arise when designing a multi-user telecommunications game like Lucasfilm's Habitat. Most seem to have obvious solutions. But after (or during) implementation & testing you often discover that your 'ideal' answers to the problems 1) are simply wrong 2) severely unbalance gameplay or 3) cause even WORSE problems than the original! In this document I will attempt to recall some of the "solution-histories" of these various "tough questions" as applied to Lucasfilm's Habitat.

The Metaphor, a misunderstood, over-used tool for game design:

The Habitat design was built on a "real-world" metaphor. There would be houses, money, human-like figures, guns, malls, shops, streets, clothing, mailboxes, plazas, cars, parking meters, telephones, busses, doorbells, trees, grass, bushes, beaches, courtrooms, police, criminals. On top of this were to be added fiction and fantasy elements like teleport-booths, techno-gadgets, magic wands, shovels, escape devices, crystal balls, magic lamps, 'gods' and others to be added as the system grew. This metaphor was chosen for it's ease of reference for the customer. Everybody would have some idea how Habitat should work because it worked just like the real world.

It was a big mistake. We promised too much. Even God took 6 days. This metaphor presented us with twofold confusion: 1) How can you describe to others "what it is", and 2) It turns out that the real-world has reasons for it's various institutions that DON'T APPLY in a computer network. Here are some examples of real-world/game-world clashes:

Transportation: Cars and Busses get you from place to place faster than walking. We had teleporters, so who needs cars? Well, you might WANT a car to get to places that has no teleports. The problem turns out to be that walking and driving would take place at *about the same speed*! And not only that, a graphic image of a car would take us GOBS of space in the home computer.

Solution: Elimination of cars/busses & further refinement of the teleport system to include the 'home' feature to return an avatar to his home turf.

Communications: We were steadfast in our desire to use telephones for on-line-messages and mailboxes for mail. We were really wrong here. (Quantum was right on this design issue!) We lost sight of the real goal by looking through the grey-goo of our metaphor. The goal is to *increase interaction at all costs!* Interaction is the single largest key to Habitat's potential success.

Solutions: OLMs became ESP (a metaphor to fit the solution) and mail is delivered directly to a users 'pocket'. The text interface was modified so that any sheet of paper could be sent as mail.

What to do when sister picks up the phone:

Disconnect. End of game. A fact of life in telecommunications. There are 5 possible causes of a disconnect:

Intentional, Normal Log out (chosing quit)
Intentional, Abnormal Log Out (yanking the phone/ power down)
a) when alone or for no 'gain'
b) for gain, or to avoid loss (of life, money, stature, account, whatever)
Unintentional, Abnormal Log Out (power failure, sister pulls phone, etc.)
Unintentional,Host system shutdown (never supposed to happen, right?)

Only the first and last causes are detectable. There is *no way* to tell if a user got disconnected by accident or on purpose!

"Stick-um up! Give me all yer money or you get free lead ear piercing!" >click!< (the sound of the victim turning his computer off). What happens on the criminals screen? Is it Fair? What should happen to characters *whenever* its human is not on-line?

Our first answer was when there is any logout the character 'turns to stone', becoming a statue standing in his last known location. This would not occur in your 'turf' or 'hotels' where you would go to 'sleep'. The idea was that people could still interact with you (one sided) even when you were off-line. During early alpha testing, it became clear that this was yet another mistake. First, unintentional disconnects were unfair, because often people couldn't get back on right away, and lost whatever they had in hand. Second, the world was soon littered with stoned avatars and we re-encountered an old specter:

Overcrowding, Traffic Jams, Blockades and Public Assemblies

We implemented the system on a toy computer, with 64k and a 1mhz clock. We could only draw "so many" objects per frame and "so many" turned out to be 2-3 foreground objects, 6 avatars and their stuff. We knew this very early, so we designed around it. Early on we thought that traffic jams would be a *feature* instead of a headache. Groups of people could get together and bar entrance to a place. Yeah! That would be neat! Forced interaction. We even planned a 'shout' feature that would let you yell into the next region so you could talk to blocked regions.

But those damn blasted stone statues kept collecting in the favorite meeting places, and it was hard as hell to design a city that was both small and difficult to 'clog up'. We had also missed the boat on group activities of over 6 persons. We couldn't have any!

Solution: 'Ghosts'. We invented a state where a user could watch the interaction in a region and leave, but not effect the region in any way. In this manner, there would never be a region you couldn't get thru because you could always turn into a ghost & there was ALWAYS room for a ghost. To resolve some of the 'fairness' issue we allowed people to become ghosts at ANYTIME, and allow them to 'de-ghost' whenever there is room. If you ever disconnected NO MATTER WHAT THE CAUSE you were turned into a ghost, and were safe. Also, since there could be any number of ghosts in the room (no increased animation time) we could host public assemblies in Habitat.

'Problems' with this solution: You might think 'Well, if you are safe as a ghost, and you can become one at any time... where is the danger of harm?' It turns out that this is no problem at all. Dozens of people during the Alpha and Beta tests died during shootouts and robberies. The cost of losing your life is small (a free trip home and loss of pocket contents) and the value of personal interaction is high (you can do NOTHING as a ghost). There are a few things that make it still work: 1) It takes several hits with a weapon to kill 2) if you turn into a ghost, anything you have on the ground is available for other to pick up 3) you might be able to (with others help?) kill him first!

It works! We did it! And people will pay to play it!

We got Habitat to run on a Commodore 64! This is undoubtedly the most complex program ever written for this computer. The designer made it possible. Chip Morningstar designed a program built on 'big-system' principles like distributed-processing, virtual-memory, object-based programming, and multi-processing. Say those words in the same sentence with Commodore 64. We did. It works but it was close. At least 3 times the program had to be significantly altered because the C64 did not perform to spec.. Habitat 'just barely' runs on a C64, and would NEVER run on an Apple II.

Habitat Anecdotes and other boastings

by

F. Randall Farmer

Actually, the word "anecdote" comes from a Greek word that means "unpublished". I guess these anecdotes are now "stories".

Preface:

These stories are hopelessly intermingled. I wish this were a hypertext document so I could link them all properly. It's not, so, if you find an interesting topic, feel free to skip around as the "see:" notes direct.

The People:

The entire point of Habitat is The People. It is an interactive environment where people define the parameters of their experience. Chip likes to call it "A Social Crucible": throw some people in a room with some fun toys, and see what happens. If a situation arises that requires modification, first let them try to sort it out —avoid changing the rules— and if they can't, take their input on how to change things. From this, it is clear that to understand Habitat, we must **first** understand its users.

There are basically 5 types of people in the Habitat universe:

1) The Passive
2) The Active
3) The Motivators
4) The Caretakers
5) The Geek Gods (system operators)

The Passive:

Easily 50% of the *number* of users fall into this category, but they probably use only 20% of the connect time (rough estimates). They tend to "cross over" to Habitat only to read their mail, collect their 100t bonus, and read the weekly newspaper. They tend to show up for events ad-hoc and when the mood strikes. This is the most important area for development. Special events and activities need to target this "on for just a few minutes" group. This group must be lead by the hand to participate. They tend to want to "be entertained" with no effort, like watching TV. The trick here is to encourage active participation.

The Active:
This group is the next largest, and made up the bulk of the paying user-hours. The active user participates in 2-5 hours of activities a week. They tend to log into Habitat right after connecting. They send out ESP messages to others on-line to find out what is going on. They ALWAYS have a copy of the latest paper (and gripe if it comes out late). This group's biggest problem is overspending. They really like Habitat, and lose track of the time spent "out there". The watch word here is "be thrifty". (See **Events:***Quests* for more on this)

The Motivators:
The real heros of Habitat. The Motivators understand that Habitat is what they make of it. They set out to change it. They throw parties, start institutions, open businesses, run for office, start moral debates, become outlaws, and win contests. Motivators are worth their weight in gold. One motivator for every 50 Passive/Active users is wonderful. Nurture these people.

The Caretakers:
Usually already employees. The Caretakers are "mature" Motivators. They tend to help the new users, control personal conflicts, record bugs, suggest improvements, run their own contests, officiate at functions, and in general keep things running smoothly. There are far fewer of these than Motivators. Out of a Pilot group of about 400, we had 3. What you want to do with a Caretaker is groom him for Geek Godhood.

The Geek Gods: (Not a spelling error)
I was the first Oracle/Operator. (I talk about that experience in **Gods**). The operator's job is most important. It really is like being a Greek God from the ancient writings. The Oracle grants wishes and introduces new items/rules into the world. With one bold stroke of the keyboard, the operator can create/eliminate bank accounts, entire city blocks, or the family business. This is a difficult task as one must consider the repercussions of any "external" effects to the world. Think about this: Would you be mad at "God" if one day suddenly electricity didn't work anymore? Habitat IS a world. As such, someone should run it that has experience in that area. I suggest at least 10 years experience in Fantasy Role Playing and 2 years on telecommunications networks (specifically ones with CHAT programs). A Geek God must understand both consistency in fictional worlds, and the people who inhabit it.

To optimize the Habitat "funativity" experience, the goal is to move the user from his/her present category to the next one up:

Passive->Active->Motivator->Caretaker->Geek God.

Move everyone one role to the right, and you will have a successful, self maintaining system. (Read: you will make bags of money.)

Real Money:

The Habitat Beta Test was actually a paying pilot-test. The testers would be paying $0.08 per minute to play and in this way we could see if Habitat was financially feasible. There were exceptions; about 25% of the testers would be QLink staff, who either had free accounts or were given a certain number of free hours. This distinction caused some difficulty in deciding if any Habitat activity was a success (see **Events**). We wanted to see if Habitat was *fun enough* for *paying* customers.

Read these (don't forget to read *between* the lines):

A certain user posted this message (edited for brevity):

> As of today I am quitting Habitat. It costs too much.
> I have been a Q-Link subscriber for 2 years.
> The first year I used only 2 plus hours. ($10)
> The next year I used only 5. ($25)
> But in the last month, while I was playing Habitat I spent $270!!!
> I can't afford that. You need to make it cheaper.

$270 = 57 hours or over 100 times his previous peak usage!
We must have made it "too much fun!"

another user said:
"I didn't realize that I was going to want to play 50 hours/month!"

Habitat (for some) was addictive. Because of this, there was a call for "Bulk Discounts" and various schemes were proposed by the users. None of them were implementable, and all of them would have resulted in significant losses. I fully expect the call to go out again when it is released.

Yet another spent *over* $1000 in *one month* in Habitat. At around $300 and $600 dollars, he was mailed a message suggesting he "check out his usage in the billing section". If we could get 20 more of this type of "rich" user, we would be profitable!

Habitat Money:

The Habitat official currency is the Token. (Chip's story of this choice of terminology goes here).

This is the economic model: You are "hatched" with 2000t, and every day you log in, you receive 100t. Money can be won in contests/quests. You can buy and sell objects using automated machinery. The Vendroid sells stuff. The Pawn Machines buy it back. Each Vendroid makes the purchased item out of thin air. That's right, *no production costs.* This leads to an interesting problem of runaway inflation. We never got enough people in the system to understand this effect, but got a taste of in when "The Great Scam" happened:

The Scam:

During the Alpha test, "The Big Doll-Crystal Ball Scam" took place. In order to make the automated economy interesting, we made Vendroids so that the could have any price for any item. This was so we could have local, specialized economies (i.e. a widget could cost a little less if you bought it at Jack's Place instead of The Emporium). In two vending machines across town from each other were two items that were for sale *for less* than the pawn machine would buy them back for: Dolls (for sale at 75t, hock at 100t) and Crystal Balls (for sale at 18000t, hock at 30000t). One weekend several persons participated in the Scam, they took their money, purchased many boxes, walked to the Doll Vendo and bought as many as they could afford, walked back to town and pawned them. They repeated this process until they had enough to purchase Crystal Balls. This took many hours. The final result was at least 3 people with 100,000t - 500,000t. In one night the economy had been diluted as the T1 (the Token Supply) has jumped 5 times! (for more on this Scam, see **Geek Gods:***They Cheated!*).

The new *rich* class now began to distribute their wealth by having treasure hunts. There were other quests and hunts that gave many users fat bank accounts. Soon a *true* economy began to emerge: Heads. Since you can change heads in Habitat, and unique heads were often prizes or gifts from the oracle or *very* expensive, their value skyrocketed. This would definitely be true when thousands of users came along, as there are only 200 or so styles of heads, and each user is initially given a choice from about 30 of those. Heads are the only *obvious* form of customization an Avatar has.

The Issues:

As I have said before, Habitat is a society, and as such, has spawned many debates about how the Habitat world should be. Very few "rules" were imposed on the world from the start.

A theme at the core of many of the arguments is philosophical. Is an Avatar an extension of a human being, a Pac-man like critter — destined to die 1000 deaths — or something else. Our answer is all of the above and none. Again the people decide what is right. In reading about the issues, keep in mind that our sample was very small, and skewed towards Actives and Motivators.

At first, during early testing, we found out that people were taking stuff out of others hands and shooting people in their own homes. We changed the system to allow thievery and gunplay only in non-city regions. (That one was *easy*! It gets more complicated from here)

Dial H for Murder:
The hottest issue was, by far, *murder*. In Habitat, if an Avatar is "killed" he is teleported back home, with his pocket emptied, what he was holding dropped, his hit-points restored, and his head put in his hand. However, only what he had with him and his position in the universe has changed. One of the Motivators took to randomly shooting people roaming outside of town. A debate arose: Is Habi-Murder a crime? Should all weapons be banned? Is it all "just a game"? There was such a debate on the issue, that a vote was taken. We were surprised by the results. 50% said "A crime" and 50% said "no — it is part of the fun". Our outlaw had in fact demonstrated that human-human interactive combat was fun for over half the audience. And since anyone who didn"t want to fight could just "ghost" and run away, there was no reason to consider the banning of weapons.

The Order of the Holy Walnut:
One of the outstanding proponents of the anti-violence-in-Habitat view was also the first Habitat Minister. A Greek Orthodox Minister opened the first church in Habitat. His canons forbid his disciples to carry weapons,

steal, or participate in violence of any kind. It was unfortunate that I had to eventually put a lock on the Church's front door because every time he decorated (with flowers), someone would steal and pawn theme while he was not logged in!

Wedded Bliss:
Three Habitat weddings took place in that church. These were not human-human weddings, but Avatar-Avatar. Their turfs were joined so that they could cohabit. There were some technical problems with this that should be resolved in any new versions. Only one account could enter a turf if the owner were not home. We hadn't properly handled cohabitation.

The first Habitat divorce occurred 2 weeks after the 3rd wedding. I guess Habitat is a bit too close to the real world for my taste! The 1st habitat lawyers handled the divorce, including public postings all about town.

Entertaining the neighbors:
The Party was one of my favorite activities. I liked to throw them at new Avatars' houses. I would ESP a known "Passive" Avatar, and ask him where he lived. If he told me, I would send ESP to "Actives" and "Motivators" that were on-line teleport to the address. Great fun.

A close cousin to parties was the Sleep-Over. The users invented this on their own. Often private discussions would take place in a turf. It was considered a minor social honor to be invited to sleep-over. This meant to log-out while still in another's turf. This was an honor because you would be able to log in later even if the host was not on. This would leave the host's belongings open to plunder.

More on Stealing:
Speaking of plunder... Stealing is still possible, even within city limits, as once an item is placed on the ground, it has no owner. Like murder, opinions on this issue are deeply divided and we think the best way to resolve it is to let (help) the players devise a limiting mechanism.

Secret Identities:
In the original proposal, all Avatars would be able to have unique names (separate from their log-in names) and they could say they were anybody they wanted. Like a big costume party, no one would know who was who. I lost the battle for unique names, and Quantumlink wanted an "identify" function. It seemed the anonymity I wanted was lost. But I suggested a

counter-proposal. A tit-for-tat rule. If you "peek" at someone else's secret identity, you will be unmasked to that Avatar, and no one else would know the results. Some very interesting dynamics developed. Some people were offended if they were ID'ed right away. And others never bothered, if you said "HI! I'm WINGO". I remember one time that I convinced someone that I was another person by sending ESP as "myself" to the person *in the same regio.*

Business:

The economy was a minor issue. Most everybody had plenty of tokens (except the Passives). In an attempt to open the retail business to Avatars, a Drug store was opened, with a locked room in back that only the owner could enter that contained the only vendo that sold Habitat healing potions and poisons. The shopkeeper would pay the fixed price, and could charge whatever he wanted for resale. It was a success except for the fact that the owner logged in at strange hours.

To Govern or Not to Govern:

Our design directive was not to interfere in Habitat politics or set up a government or law establishment. Many people thought that crimes of killing and theft ought to be punished. We decided to hold sheriff elections. The favorite candidate was a friendly guy, but many didn't know that this very same Avatar was the brains behind "The Scam". There was a public debate in the Populopolis Meeting Hall with the 3 AvaCandidates making statements and fielding questions. I was among the ghosted attendees. I would pre-type some comment like "Vote for Foon!", deghost quickly, press return to send my message, and become a ghost again. No one would have any time to tell who I was before I was gone. This was fun. During the Question and answer period I pretyped this question: "Please explain to us why we should vote for a sheriff who obtained his campaign fees rather -ah- UNUSUALLY?". This started a real-life-like mud slinging fight. As it turns out, he won by a landslide anyway. Populopolis had a sheriff.

For weeks he was nothing but a figurehead. We were stumped about what powers to give him. Should we give him the right to shoot anyone anywhere? Give him a more powerful gun? A wand to >zap< people to jail? What about courts? Lawyers? Laws? Late in the test the answer struck me: *ask the users!* A "Committee for a Safer Habitat" sent out a mailer to everyone asking this question: "What should the sheriff be able to do?". Then another election was held "What is a crime?" and "What should the sheriff be empowered to do?". The results were unable to be acted on before the test ended. An interesting side effect of this was that it became apparent there there are two basic camps: anarchy and government. It will be great to see what happens with thousands of users facing this decision. Habitat need *not* be set up with a "default" government (like reality).

Magic Inflation:

Besides economic inflation, we also had Magic inflation. In the Dungeon of Death (see EVENTS), the designer had a vending machine that sold magic wand that teleport to the oracle anyone you point them at for *only* 1000t. At this time magic wands worked forever. Soon everyone had one of these wands and people were zapping each other all over the place. Crime got really out of hand when criminals would travel as ghosts, wait for people to put their belongings down for a second, deghost - zap - and steal. I had always planned on implementing a limited "charges" feature but was to busy tracking down bugs. Soon it was clear it was time to act. "God" changed the rules, and limited magic. The issue became foremost in the discussion arena: Some people were using these rods for the "good" cause of rescuing people when they got lost. Many were outraged that the rules changed. Ask yourself this question: What would you have done? This is a tricky question, fundamental for the chief operator to understand.

Motivators & Caretakers at work:

The Rant:

By far the Caretaker who had the greatest on his fellow users was the editor of the Habitat newspaper "The Weekly Rant". This user tirelessly spent 20-40 hours a week (free account) composing a 20, 30, 40 or even 50 page tabloid with containing the latest news, events, rumors, and even fictional articles. This was no small feat, he had only the barbaric Habitat paper editor, and no other tools. After he had composed the pages of an issue, he would arrange them in several chests of drawers in The Rant office and send me mail. I would publish it by running some Ghu macros that would bind them into a newspaper and distribute it to the news vendos, check the copy by hand for errors, and deliver a copy to the office (in Habitat). This worked great, but took massive amounts of his personal time. I began to automate the process further just as Habitat operation changed hands. The new publisher didn't publish on time, delayed getting the tools ready to speed up creation, made editorial changes (he wanted it to be shorter, less fiction), and didn't hand-deliver a copy of the final product. The editor quit. Just like real life: Someone new runs the show and the sensitive leave. Again, these people are *rare* and should be handled carefully. The Rant will never be the same.

Duels:
One of the wands we implemented caused the victim to perform the "jump" gesture, accompanied by a "Hah!" word balloon. It was fun for a while, mostly because you could really effect another Avatar, but it got old fast. Soon a game was developed *completely by the users* involving these wands: The Duel. The rules were simple: two combatants, two wands, one judge. When the judge says "go" the first to "hit" the other with the wand 3 times wins. Not as easy as it sounds, since the duelists are allowed to run around.

Tours:
Another Caretaker was the number #1 all-time most-traveled Avatar. He also was the longest lived. When new people started logging in, he took them on guided tours of this strange new world. He made them feel like they had a "friend" in town.

Combat:

"Conflict is the essence of drama". We used this quote in the initial Habitat design document. Habitat (it was then named "Microcosm") was to have personal combat in the forms of weapons. Most computer games had combat, and *we* were offering a chance for users to effect each other!

Here I will explain how it actually ended up working. There were ranged weapons and hand-to-hand. An Avatar is born with 255 hit points (the actual number is masked from the user, and a "general state of health" message gives the user some idea how bad off he is.) While holding the weapon, you select a target and DO (attack). There is a telecommunications delay that may effect the hit-or-miss result. Each successful attack does some small amount of damage (i.e. 20 pts.). You are always informed when you are shot, as your Avatar is knocked onto his rump.

As you can see, it would take quite a few hits to "kill" a healthy Avatar. Not only that, but you can avoid being damaged if the attacker can't "touch" you in 2 ways: 1) by turning into a ghost or 2) running around (not standing still). You use #2 when you are in a gunfight where you are shooting back. This seems to be a working dynamic. If you *really, really* are low on hit points, you travel the "wild" regions as a ghost. There are also devices that will restore your hit points. The real problem is communicating this to new users, who are often standing around in a region when a bandit comes along with a gun. The neophyte hears a "bang" and sees his Avatar knocked on his can. Instead of acting, he types a message like "What was that? Why am I sitting down?". Meanwhile, El Bandito cranks out another 12 bullets.... Dead newbie probably had all of his money and stuff in his pocket too! This problem should be corrected in the Avatar Handbook, explaining that guns are dangerous (something we *thought* people would assume on their own).

For more on special types of combat see **Issues:***Magic Inflation*, **Motivators in Action:***Duels*, and **Events:***Dungeon of Death.*

The Scheduled events:

The D'nalsi Island Adventure:

The first treasure-hunt ever planned for Habitat was mine, the D'nalsi Island Adventure. I took me hours to design, weeks to build (including a 100-region island), and days to coordinate the actors involved. I had taken several guesses as to what how long it would take the players to perform each "segment" of the quest. The mission: recover the lost "Amulet of Salesh". First: A trial, introducing the characters and the first clues. Second: Salesh hires the adventurers. Third: The players needed to figure out the "secret" teleport address. Fourth, they must find the door to the hidden cave, solve the riddle. Last: find the hidden crawlway and the buried chest containing the amulet. The prize was 25,000t.

The first part was in the form of a "dinner theater"-like play, set in the county courthouse. It was heavily attended. Since it was set up as an introduction, there was no appropriate "time" for the players to discover anything.

On the day that Salesh "hired" adventurers to find his amulet, he gave out copies of a map of the island. Hidden on this map was a word that was the teleport address to the island. After about 15 minutes of hiring, when about the 10th Avatar was hired, Salesh (me) received an ESP from one of the Motivators: He had discovered the teleport address. Darn! It seemed that the others had no idea where to start, so I sent ESP to all the players announcing that the teleport address had been discovered to be a word on the map.

Within 8 hours the treasure had been recovered by that person who had 1st discovered the island. This was so soon that almost half the adventurers (the novices) had not yet even discovered the teleport address! It was clear that there is a very wide range of "adventuring" skills in the Habitat audience, and various events need to be better targeted, and should include handicapping mechanisms so that those behind don't get more and more behind.

The Dungeon of Death:

This "combat oriented" dungeon was the brainchild of a Caretaker that had recently become a Q-Link in-house employee. It shows that experienced "insider" could design an successful event using his understanding gained thru being a player first. (Note: I had nothing to do with this design, so it was my first event as a participant)

For weeks ads appeared in The Rant announcing that that Duo of Dread, DEATH and THE SHADOW were challenging the adventurers to come to their lair. Soon, on the outskirts of town, a dungeon was discovered. Outside a sign read "Danger, enter at your own risk.". Two operators were logged in as DEATH and SHADOW, armed with guns that could kill in 1 shot (instead of the usual 12). The dungeon had totally dark (light did not help), dead end (trapped), and duplicate regions. It was clear that any explorer had better be prepared to "die" several times before mastering the dungeon. The rewards were pretty good: 1,000t minimum and access to a vending machine that sold "teleport" wands (see **Issues:***Magic Inflation*). I even got a chance to play DEATH for one night. It was a slaughter. Avatars were dropping like flies... but most of them had prepared by emptying their pockets. When I got to play DEATH, I found him in one of the "dead ends" with four other trapped Avatars. I deghosted and started shooting, but was shot twice myself and died. Shoot! The last operator had not healed damage from his last encounters! The worst part of this is that "when you die, what is in your hands is dropped". Yep. Some normal Avatar now had the "elephant" gun that could kill in one shot. The most valuable weapon in Habitat. What should I do? I later found out that this was not the first time this happened, it happened to a Q-Link operator and they "forced" the Avatar to give it back. I did something else: As DEATH (never identifying my true self) I threatened to kill the new owner. She replied that she would never leave town, thus being safe. OK, I think, she's smart. After about an hour we settle on a deal, 10,000t to buy the gun back. We meet at The Oracle in town, where it is safe and make the exchange. It was great. The entire "operations accident" was handled within the game universe with no "external" interference.

R&R weekend adventures:

These were short (1-2 hours) quests where a user pressed one of ten magic buttons to receive a clue to find one of ten hidden keys to be used in one of ten hidden safes. This were the all-around best quests to run (there were 3 of them) because there were always 7-10 winners. The only problem here was the Time Zone problem: The event had to be scheduled so that as many people as possible could participate from the moment it started. Q-Link access started at 6pm local time. This meant that for the Californians to have a chance, the adventure would have to start at 9pm East coast time at the earliest.

The Money Tree:

The Quest for the Money Tree is the first quest an Avatar learns about from reading his free Welcome Wagon version of the Rant placed in his Turf. There is a tree in a forest that will dispense 100t for every Avatar once. Everyone can feel like they have "found" the magic tree.

The Tome of Wealth and Fame:

This was also one of the originally conceived of quests. A certain set of tablets contained the Tome of Wealth and Fame. If you found it, you were to hide it somewhere else. You would receive a reward based on how long it took another to find it. The problem with this was that the world was so large that it often took weeks for someone to find the tome. It wasn't an active process because, if you tried, it would take days of on-line time to find.

The Long and Short of Quests:

A trend became clear about quests in Habitat. The winners of the "long range" quests like *The D'nalsi Island Adventure* were almost *always* people with free accounts. The freebies would stay on for hours on end to gain wealth, things and status (See **Habitat Money:***"The Scam"*). The paying customers could only come on 1-2 hours/week. The idea that people would be able to "work on" a quest for weeks is bogus. The long range quest must be something that either "everyone" can win or does not provide some significant advantage in the world. (See *The Money Tree*)

Grand Openings:

A real surprise was the popularity of the "Grand Opening". This the ribbon-cutting event when new regions were added to the world. Tokens and prizes were often hidden in the new regions, but it seems that the audience (especially the Passives) had an insatiable hunger to see new places and things. The Grand Opening of the Popustop Arms apartment building was the most heavily attended event of the Pilot test.

Disease:

One of the more successful "games" we invented for Habitat was the disease. There are three strains currently defined: 1) Cooties, 2) Happy Face, 3) Mutant (aka The Fly). We only were able to test Cooties with live players, but it was a hit. It works like this: Several initial Avatars are infected with a "Cootie" head. This head replaces the current one, and cannot be removed except by touching another non-infected Avatar. Once infected, you can not be infected again that day. In effect, this game is "tag" and "keep away" at the same time. Often people would allow themselves be infected just so he could infect "that special person that they *know* would just *hate* it!" Every time the disease was spread, there was an announcement at least a week before, and for at least a week afterward it was the subject of major discussions. One day that the plague was spread, a female Avatar that was getting married got infected 1 hour before her wedding! Needless to say, she was *very* excited, and in a panic until a friend offered to take it off her hands.

Some interesting variations to try on this are: Touch 2 people to cure; this would cause quite a preponderance of infected people late in the day. The "Happy Face" plague: This simple head has the side effect of changing any talk message (word balloons) to come out as "HAVE A NICE DAY!"... can you imagine infecting some unsuspecting soul, and him saying back to you HAVE A NICE DAY! ??? ESP and mail still work normally, so the user is not without communications channels. The Mutant Plague: The head looks like the head of a giant housefly and it has the effect of changing talk text to "Bzzz zzzz zzzz". We think these all will be great fun.

Deception & Trickery

These were fun things to do to your fellow Avatar.

My invention - Type this: "You have *mail* in your pocket." and watch the fun as people say "That's strange! I don't have mail!"

Chip thought this up - Send this ESP message "ESP from: yournamehere", then quickly send a "Hello" also. Your "Hello" ESP will be announced 3 times!

We developed a form of communication "harassment". You can do this on almost any network. Just coordinate a few people all sending very short ESP messages the the victim. His screen will scroll faster than he can read. This was used against the social outcast mentioned in *Dial H for Murder.*

They could only do this by recruiting more and more new members (while still keeping the organization a secret!). Secret "handshakes" could be set up. Meetings. Recruiting drives. Of course, soon there would be gang warfare. Who knows where it might go?

A Final Word:

As I close this document I find I keep remembering dozens of other stories to tell. And all of these come from my experiences with only 200 or so people! Imagine what it will be like with tens of thousands of creative minds at work! Though as of this writing Habitat is still not a released product, I still am proud of the world we created. I really expect to be meeting you soon "On The Other Side" in a world not unlike Habitat.

F. Randall Farmer
a.k.a. SPBLives