PRINCE OF PERSIA

Technical Information

October 12, 1989

BACKGROUND GRAPHICS

Each level consists of up to 24 screens. Each screen consists of 30 blocks (10 blocks wide x 3 blocks high), also referred to as "pieces." Examples of pieces are: floorpiece; pressure plate; spikes; solid block.
The complete layout of each level is specified by a "level blueprint" ($900 bytes long).

COORDINATE SYSTEM

The Apple screen is 40 bytes wide x 192 lines high, with 7 bits per byte. Most of the background graphics are done on the byte boundaries (e.g., an X-coordinate of 39 puts you at the far right-hand edge of the screen). Since each screen breaks down into 10 blocks across x 3 blocks high, that works out nicely with each block being 4 bytes wide.

The moving characters (player, guard, etc.) use a slightly different coordinate system. The X-coordinate is represented by one byte. The screen is 140 pixels wide, with 58 as the screen's left edge--giving us offscreen margins of 58 pixels on either side of the screen. (This makes it possible to compare the X-coordinates of two characters even if one is offscreen.)

Since all the character movements throughout the program are specified according to the 140-wide coordinate system, we would like to keep the character's X-coordinate on a 140-wide screen (or a width that is a multiple of 140) for all the different conversions. The character's position could then be mapped to the actual screen width for that machine--320, 512, 640, or whatever--at the time the character is drawn.

IMAGE TABLES

There are two background image tables (bgtab1-2) and 7 character image tables (chtab1-7).

Ch. table #	Contents
1,2,3,5	Player (kid); misc. shapes
4	Enemy (guard, fat guard, skeleton, vizier, shadow)
6,7	Shapes for princess scenes (princess, vizier, hourglass)

Note: chtab6 is too big to fit into memory all at once, so it is split into two files:

chtab6.a	used for opening title sequence & levels 1-2
chtab6.b	used for levels 3 and up, including victory sequence

There are two sets of background images, one for the dungeon and one for the palace.

See accompanying lists that describe the contents of chtab6-7 and bgtab1-2.

MEMORY USE

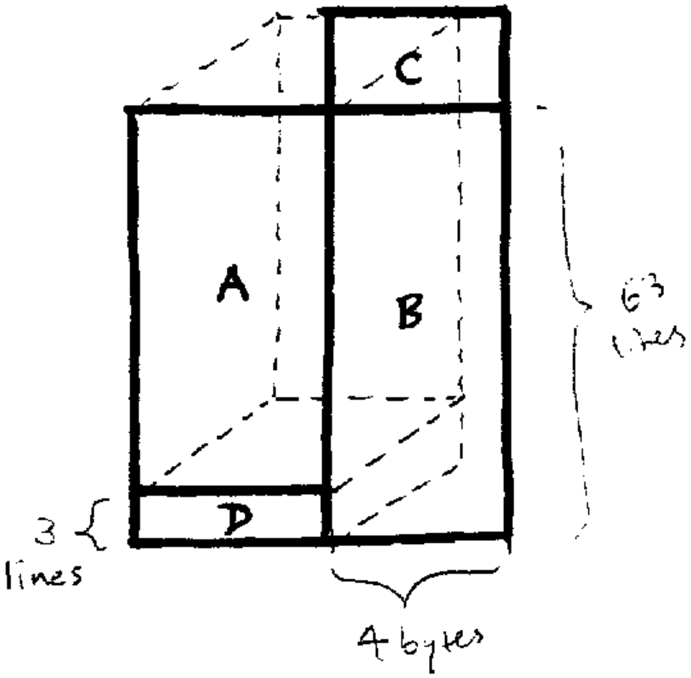
The Apple version uses memory, roughly, as follows:

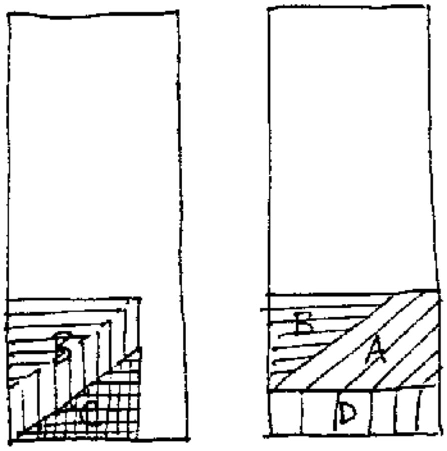
	RAM		DISK
	(during game play)		
Image tables			
Player	30K		30K
Enemy	6K	x5 =	30K
Princess & Vizier			9K
Background	13.5K	x2 =	27K
TOTAL IMAGE TABLES	49.5K		96K
Code	48K		48K
Buffer space	8K		
Music	4K		6K
Level blueprint	2.25K	x16 -	54K
Screen memory	16K		
TOTAL RAM	127.75K		
Packed dblhires screens			42K
Packed princess's room			6.5K
TOTAL DISK SPACE			252.5K

LEVEL SUMMARY

#	Name	B.g.	Opponent
0	Demo	Dun	Gd
1	Cell	Dun	Gd
2	Guards	Dun	Gd
3	Newskel	Dun	Skel
4	Newmirror	Pal	Gd
5	Thief	Pal	Gd
6	Plunge	Pal	Fat
7	Wtless	Dun	Gd
8	"329"	Dun	Gd
9	Twisty	Dun	Gd
10	Quad	Pal	Gd
11	Wild	Pal	Gd
12	Tower	Dun	Shad
13	Final	Dun	Viz
14	Victory	Pal	Princess

Level 0 is the demo level. Although levels 13 and 14 are technically separate levels, the "official" highest level number is 12.

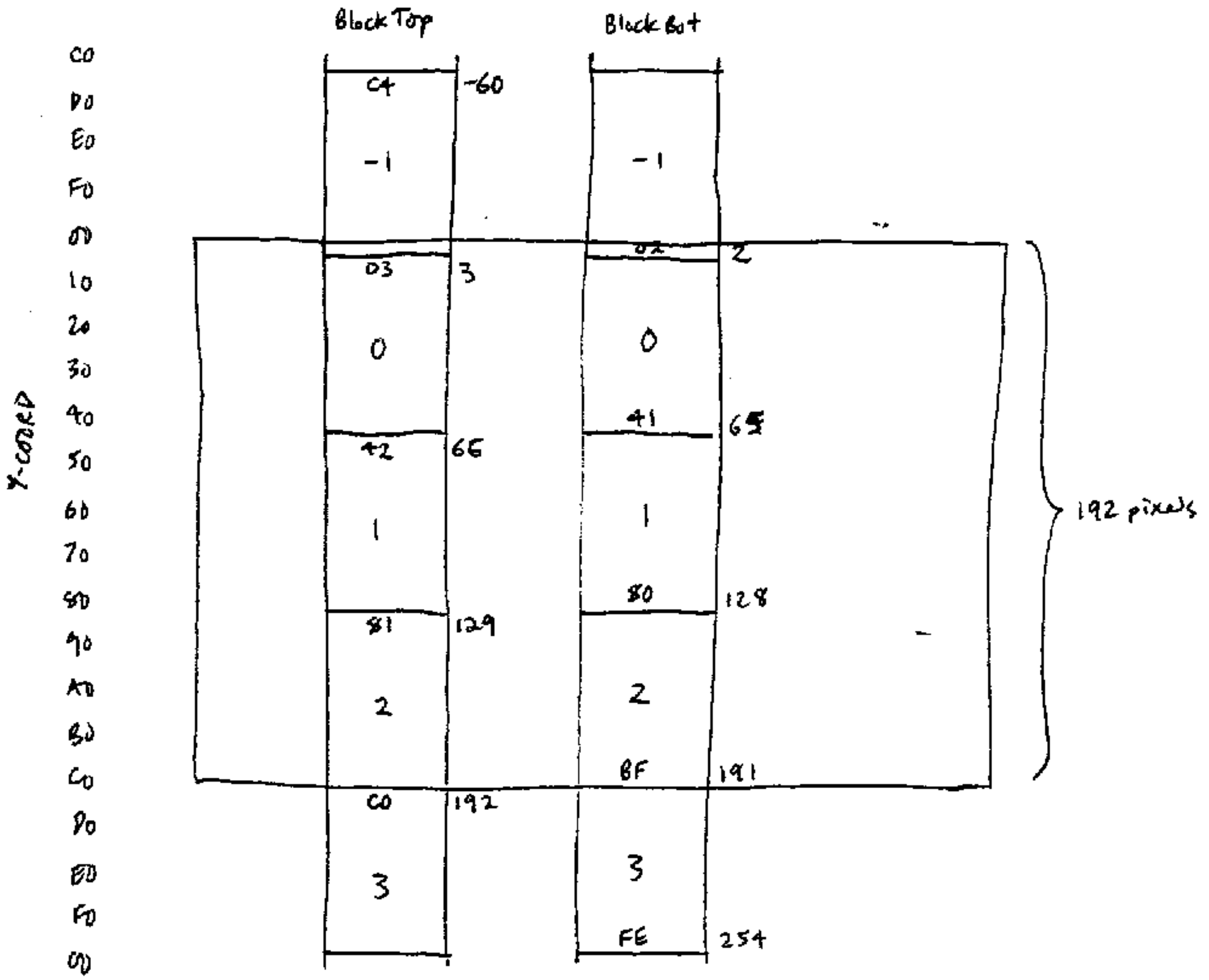


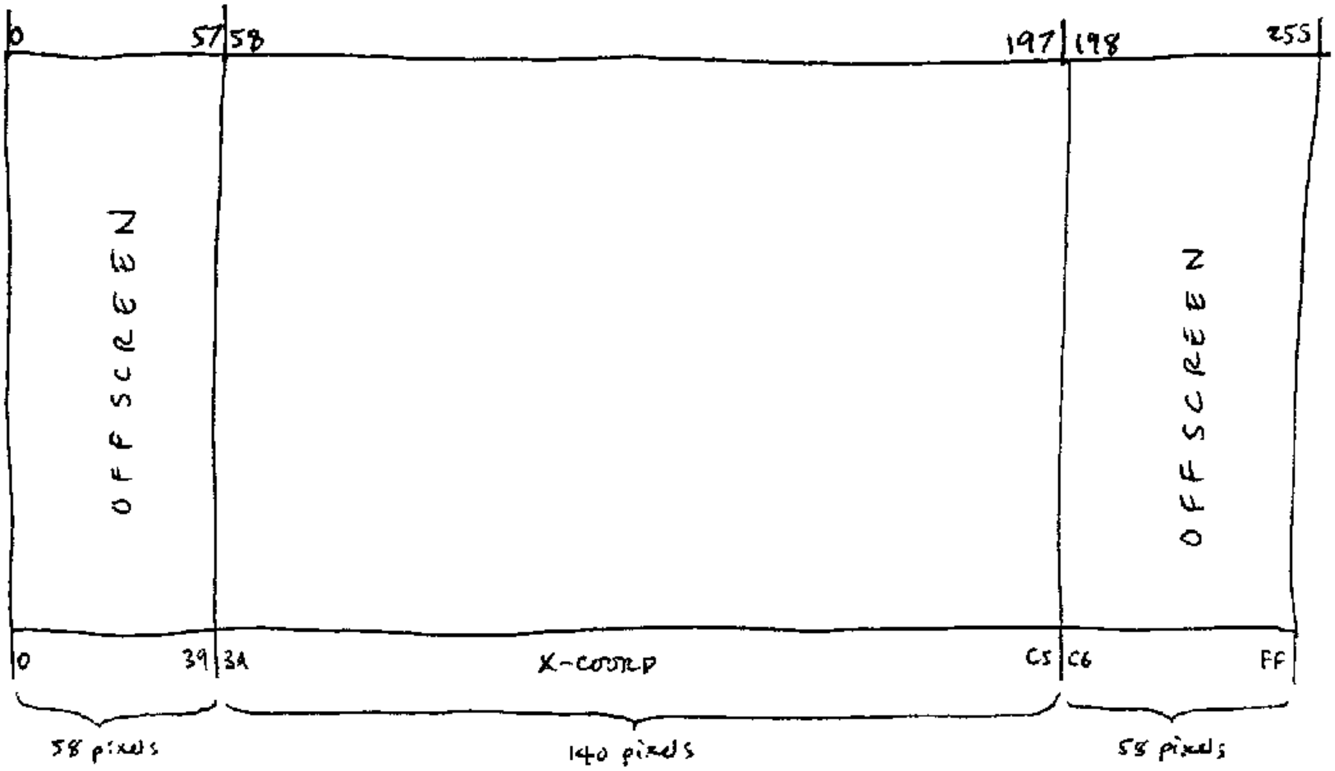


To draw a block:

1. Draw C-section of block below & to left
2. Draw B-section of block to left
3. Draw D-section
4. Draw A-section
5. Draw frontpiece

X & Y COORDS


Block Top
Block Bot
C0
D0
E0
F0
00
10
20
30
40
50
60
70
80
90
A0
B0
C0
D0
E0
F0
00
Y-COORD
C4
-60
-1
-1
03
3
02
2
0
0
41
65
42
66
1
1
80
128
81
129
2
2
BF
191
C0
192
3
3
FE
254
192 pixels


0
57
58
197
198
255
OFFSCREEN
OFFSCREEN
0
39
3A
X-COORD
C5
C6
FF
58 pixels
140 pixels
58 pixels

```
* comments
*-------------------------------
*
* M  O  V  E  R
*
*-------------------------------
*
* Routines to keep track of moving objects are contained
* in the file MOVER.
*
* There are 2 types of moving objects: Transitional objects
* (TROBs) and mobile objects (MOBs).
*
* TROBs (e.g. gate, spikes, pressplate, torch) can have moving
* parts & change appearance, but remain in a fixed location.
*
* MOBs (falling floor) can move all over the place, including
* between screens.
*
*-------------------------------
*
* TROBs are kept track of in a data structure called
* the "trans list" as follows:
*
* numtrans = # of active TROBs
*
* For x = 1 to numtrans:
*
*  trloc,x = block location (0-29)
*  trscrn,x = screen # (1-24)
*  trdirec,x = direction of motion (means something different
*   for different kinds of objects)
*
* When an object stops moving, we set its trdirec = -1, then
* remove the object from trans list on the next cycle.
*
*-------------------------------
*
* MOBs are kept track of in a similar data structure called
* the "MOB list":
*
* nummob = # of active MOBs
*
* For x = 1 to nummob:
*
*  mobx,x = byte (0-39)
*  moby,x = y-coord
*  mobscrn,x = screen #
*  mobvel,x = velocity
*  mobtype,x = type
*    0: falling floor
*    (No other MOB types defined at present)
*  moblevel,x = level (0-2)
*
*-------------------------------
*
*  The basic routine in MOVER is ANIMTRANS.  This
*  routine, which is called once per cycle, advances
*  all active TROBs to their next phase (this includes
*  deleting TROBs that become inactive) and marks the
*  appropriate redraw buffers for each TROB.
*
*  Other routines such as TRIGSPIKES, PUSHPP, BREAKLOOSE,
*  etc. are called to add new TROBs to the trans list (when,
*  for example, a character jumps over spikes or steps on
*  a pressure plate or loose floor).
*
*  The routine ANIMMOBS performs the same function for
*  the MOB list that ANIMTRANS does for the trans list.
*  It advances all active MOBs to the position they will
*  occupy in the next frame.
*
*  Example: When a character steps on a loose floor, the
*  control routine senses this & puts in a call to
*  BREAKLOOSE (which adds the loose floor to the trans
*  list).  For the next 10 frames or so, the loose floor
*  wiggles (under the control of ANIMTRANS) until ANIMTRANS
*  decides it's time for it to fall.  At that point, the
*  loose floor is deleted from the trans list, the block
*  is replaced by "empty space", and a new MOB is
*  created to take its place.  Under the control of
*  ANIMMOBS, the MOB then falls until it hits the ground,
*  at which point ANIMMOBS deletes the falling floor from
*  the MOB list and changes the objid of the block it landed
*  on to "rubble."
*
*-------------------------------
*
*  F R A M E A D V
*
*-------------------------------
*
*  IMAGE LISTS
*
*  FRAMEADV never calls hires routines directly.  Instead,
*  parameters of images to be drawn are stored in "image
lists."
*
*  There are 6 separate image lists:
*
*  bg:   Images in background plane (drawn first)
*        X, Y, IMG, OP
*
*  wipe: Solid-color wipes (drawn with b.g. plane)
*        X, Y, H, W, COL
*
*  fg:   Images in foreground plane (drawn last)
*        X, Y, IMG, OP
*
*  mid:  Images between b.g. and f.g. planes
*        X, OFF, Y, IMG, OP, TYP, CU, CD, CL, CR
*
*  msg:  Images in message plane (drawn last of all)
*        X, OFF, Y, IMG, OP
*
*  gen:  General instructions (e.g. clear screen)
```

```
*
*
*  Explanation of parameters:
*
*  X     = X-coord (in bytes)
*  OFF   = X-offset (0-6)
*  Y     = Y-coord
*  IMG   = image # in table (1-n)
*  OP    = opacity
*  TYP   = image type (for mid only)
*  CU    = top cutoff
*  CD    = bottom cutoff
*  CL    = left cutoff
*  CR    = right cutoff
*  H     = height
*  W     = width (in bytes)
*  COL   = color (for wipe only)
*
*  NOTE--bg, fg, and wipe calls assume offset=0
*
*
*  There is also an "object list" with params similar to
*  mid list:
*
*        X, OFF, Y, IMG, FACE, TYP, CU, CD, CL, CR
*
*  Note that obj list has 2 additional params:
*
*  objFACE = left/right
*  objTYP  = object type (Not to be confused with midTYP)
*
*  The object list has one entry for each object to be
*  drawn (e.g., "kid," "falling floor").  FRAMEADV uses
*  the object list to build the actual mid list
*  of images to be drawn.  E.g., the single object "falling
*  floor" might translate into 3 separate images:
*  A-section, B-section, and D-section.
*
*-------------------------------
*
*  REDRAW BUFFERS
*
*  Each redraw buffer contains 30 counters, one for each
*  block on screen: 0 = skip, non-0 = redraw and decrement.
*
*  REDBUF:
*  The most general-purpose buffer.  Marking REDBUF for a
*  block will cause all sections of the block to be redrawn.
*
*  WIPEBUF:
*  Wipe square (usually to black).  WHITEBUF contains
*  wipe height, in lines.
*
*  Marking both REDBUF and WIPEBUF for a block will cause
*  the entire block to be erased & redrawn.  This is the
*  safest way to redraw a block.
*
*  MOVEBUF:
```

```
*  Refers only to movable portion of object (e.g. lowering
*  gate).  Superseded by REDBUF.
*
*  FREDBUF:
*  Refers only to foreground plane.  Marked when character
*  goes behind a post or other object with a frontpiece.
*  Superseded by REDBUF.
*
*  FLOORBUF:
*  Refers to floorpieces.  Marked to the right of a
*  falling or hanging character.  FLOORBUF causes floorpiece
*  to be drawn in the mid plane (where it will cover up
*  character if appropriate).
*
*  HALFBUF:
*  Like FLOORBUF, but redraws a triangular section of
*  the floorpiece instead of the whole thing.  Used when
*  a character climbs up on the left side of a floorpiece
*  and we want to mask out his lower body while letting his
*  upper body show.  (Superseded by FLOORBUF.)
*
*  OBJBUF:
*  Marked whenever objects need to be drawn in a given block.
*  (Objects are always the last mid elements drawn in
*  a block.  Objects are assigned to blocks based on
*  their lower left x-y coords.  Characters are considered
*  objects.  There can be multiple objects in a given block.)
*
*  TOPBUF:
*  10-byte buffer for row of D-sections across top of screen
*  (from screen above).
*
*  Note that TOPBUF is a 10-byte buffer while the others are
*  all 30 bytes.
*
*-------------------------------
*
*  The specific routines called by SURE (in FRAMEADV) for
*  each of these buffers are:
*
*  REDBUF: redblock (drawc, drawb, drawmb, drawd, drawmd,
*     drawa, drawma, drawfrnt)
*
*  WIPEBUF: wipesq
*
*  MOVEBUF: drawc, drawmc, drawmb, drawma
*
*  FREDBUF: drawfrnt
*
*  FLOORBUF: drawfloor
*
*  HALFBUF: drawhalf
*
*  OBJBUF: drawobjs
*
*  TOPBUF: drawc, drawb, redrawd, drawmd, drawfrnt
*
*-------------------------------
```

```
*
*  B  L  U  E  P  R  I  N  T
*
*-------------------------------
*
*  LEVEL BLUEPRINT ($900 bytes)
*
*                  Start      Length
*                  -----      ------
*  BlueType        B700       720
*  BlueSpec        B9D0       720
*  LinkLoc         BCA0       256
*  LinkMap         BDA0       256
*  Map             BEA0       96
*  Info            BF00       256
*
*  TOTAL: 2304 bytes
*
*-------------------------------
*
* BLUETYPE
*
* Bytes 0-29 describe screen #1
* Bytes 30-59   "     screen #2
* etc.
* 24 screens total.
*
* Each BLUETYPE byte corresponds to one block.
* (30 blocks per screen.)  Blocks are mapped
* into BLUETYPE left-right, top-bottom.
*
* AND with #$1F to get the "objid," or object
* identification number (0-31), of each block.
*
*-------------------------------
*
* BLUESPEC
*
* (Screen blocks mapped the same way as in BLUETYPE.)
*
* Taken together, each pair of corresponding bytes in
* BLUETYPE and BLUESPEC contains all the information
* about an object.  The BLUETYPE byte always contains
* the object id.  The BLUESPEC byte functions differently
* depending on what the object is.
*
* For movable objects (gates, spikes, torches, etc.)
* BLUESPEC specifies the object's "state" (e.g. is it
* open, closed, somewhere in between?)
*
* For static objects (floor, space, solid block, etc.)
* BLUESPEC specifies the object's "pattern" (e.g. which
* design appears on the wall behind it?)
*
* For pressure plates, the BLUESPEC byte tells which
* gates the pressure plate controls.  Specifically, the
* BLUESPEC byte is a pointer (0-255) to the first entry
* in the link list for this pressure plate.
```

```
*
*------------------------------
*
* Link list (LINKLOC/LINKMAP)
*
* Contains a list of the gates controlled by each pressure
* plate.  Each pair of bytes specifies one plate-to-gate
* linkage.  There can be up to 256 such linkages in a level.
*
* LINKLOC:
*  Bits 0-4: gate screen posn (0-29)
*  Bits 5-6: low 2 bits of gate screen # (1-24)
*  Bit 7: 1 = this is last entry, 0 = more gates to come
*
* LINKMAP:
*  Bits 0-4: pressplate timer (0-31)
*  Bits 5-7: high 3 bits of gate screen #
*
* If a pressplate controls nothing, LINKLOC = FF; LINKMAP
* still functions as pressplate timer.
*
*------------------------------
*
* MAP
*
* Specifies how the 24 screens of the level are connected.
*
* Each screen gets 4 bytes corresponding to the screen #s
* of the 4 adjacent screens.
*
* Bytes 0-3 = screen #1
* Bytes 4-7 = screen #2
* etc.
*
* For each screen:
* Byte #1 = screen to left
* Byte #2 = screen to right
* Byte #3 = screen above
* Byte #4 = screen below
*
*------------------------------
*
*  INFO
*
*  Bytes 0-63: reserved for editor
*
*  Bytes 64-255: Information about starting positions
*  of player & other characters on this level.
*  (See [redacted]EQ for details.)
*
*------------------------------
*
*  S E Q T A B L E
*
*------------------------------
*
*  Frame def list:
*
```

```
*  1200 bytes allocated -- 240 frames, 5 bytes each
*  (241-255 reserved as commands)
*
*  Frame definition consists of:
*
*  (1) Fimage
*
*     Bit 7 = chtable # (0-7), bit 2
*
*     Bits 0-6 = image # (0-127)
*
*     SUMMARY:
*       $00 + x: chtable 1,2,3,4
*       $80 + x: chtable 5,6,7,8
*
*  (2) Fsword
*
*     Bits 6-7 = chtable # (0-7), bits 0-1
*
*     Bits 0-5 = pointer to SWORDTAB frame (0-63)
*
*     SUMMARY:
*       $00 + x: chtable 1,5
*       $40 + x: chtable 2,6
*       $80 + x: chtable 3,7
*       $c0 + x: chtable 4,8
*
*  (3) Fdx
*
*     X-shift in pixels (+ = fwd, - = bkwd)
*     (NOTE -- horizontal resolution is 140 pixels)
*
*  (4) Fdy
*
*     Y-shift in pixels (+ = down, - = up)
*     (Frame #15 is defined as unshifted)
*
*  (5) Fcheck
*
*     Bit 7 = odd/even pixel
*
*     Bit 6 = 1 if floor check is required (i.e., if weight
*       is on floor)
*
*     Bit 5 = 1 to "thin" this frame for collision detection
*
*     Bits 0-4 = number of pixels (0-31) from left edge of
*       image block to base x-coord
*       (usually center of foot bearing character's weight)
*
*     SUMMARY:
*       $c0 + x: check, odd
*       $40 + x: check, even
*       $80 + x: no check, odd
*       $00 + x: no check, even
*
*     + $20 to set bit 5
*
```

```
*-------------------------------
*
*  SEQPOINT is 2-byte pointer to character's current
*  position in sequence table.
*
*  Seq pointer is incremented by 1 with each frame-advance.
*
*  CTRL can jump seq pointer around at will (e.g., in response
*  to joystick command)
*
*  POSITIVE seq table values represent frame numbers.
*  NEGATIVE values are instruction codes.
*
*  Sequence table instructions:
*
*   goto NN     jump seq pointer to NN (low byte first)
*   aboutface   change KIDFACE direction
*   up          up one floor
*   down        down one floor
*   chx N       KIDX := KIDX + N (BEFORE we draw next frame)
*   chy N       KIDY := KIDY + N (ditto)
*   act N       change action code to N
*   setfall X,Y  set initial x,y velocity for freefall
*
*  Action codes:
*
*  -1 = dead
*   0 = standing still
*   1 = running, jumping, other actions
*        that require a floor beneath your feet
*   2 = hanging, climbing, and all other actions that
*        require holding onto a ledge
*   3 = in midair (briefly)
*   4 = in freefall
*   5 = being bumped
*   6 = hanging straight
*   7 = turning
*
*  Screen resolution is 140 x 192.
*
*-------------------------------
*
*  NOTE: Frame table offsets are TEMPORARY; sequence table
*  offsets are PERMANENT.
*
*  CTRL draws each frame at [KIDX + Fdx, KIDY + Fdy],
*  but leaves KIDX & KIDY unchanged for the next frame.
*  "Chx" and "Chy" instructions in sequence table, however,
*  change KIDX & KIDY permanently.
*
*  For JUMPHANG, CLIMBUP, etc., the idea is for KIDX, KIDY &
*  KIDLEVEL to keep the kid where he started -- at the end
*  of the block behind & below the one he's hanging from --
*  and use only the frame list x & y offsets, until he's back
*  on the ground.  This way, we can branch into either HANGDROP
*  or CLIMBUP from any point in HANG.
*
*  The first 4 frames of STARTRUN also use only the frame list
*  offsets.  This lets us switch easily to, say, FULLSTEP
*  or STANDJUMP.
*
*-------------------------------
*
*  M  I  S  C
*
*-------------------------------
*
* Potion IDs
*
* 0  Empty
* 1  Regular healing
* 2  Boost strength
* 3  Weightless
* 4  Upside down
* 5  Poison
*
*-------------------------------
*
* Character IDs
*
* 0  Kid
* 1  Shadow
* 2  Guard
* [redacted] Vizier (in game)
* 4  Skeleton
* 5  Princess (in princess cuts)
* 6  Vizier (in princess cuts)
*24  Mouse
*-------------------------------

--End assembly, 0 bytes, Errors: 0
```

```
*-------------------------------
*
* Character animation comments
*
*-------------------------------
*
* For each character, we maintain a 16-byte block of data
* (referred to as "character data" or "character vars")
* that describes the character's current position & what
* he is doing.  The 16 bytes are allocated as follows:
*
* CharPosn
*
*   Frame # of the character's current position.  E.g.,
*   CharPosn = 15 refers to frame #15 of the frame list, or
*   "standing still."
*
* CharX
* CharY
*
*   Character X & Y coords, based on a 140 x 192 screen.
*   (Upper left corner is X = 58, Y = 0)
*
* CharFace
*
*   Direction character is facing: 0 = right, -1 = left
*
* CharBlockX
* CharBlockY
*
*   Coords of character's current block (X = 0-9, Y = 0-2).
*   (0,0) is upper left block.
*
* CharAction
*
*   Code containing information about character's current
*   action.  E.g., CharAction = 4 means "falling."  This
*   variable is used in a variety of different ways in
*   different situations
*
* CharXVel
* CharYVel
*
*   X & Y components of character's velocity (during
*   freefall).  Every frame, CharXVel is added to CharX and
*   CharYVel is added to CharY.
*
* CharSeq (2 bytes)
*
*   Pointer to current address in the sequence table.
*
* CharScrn
*
*   Screen # of character's current screen.  (0 for null
*   screen.)
*
* CharRepeat
*
*   When character stands at the edge of a chasm & takes a
*   cautious step forward, the first time he tries it he only
*   "tests" with his foot.  This causes CharRepeat (usually
*   a non-0 value) to be set to 0.  The next time he tries a
*   cautious step, he will step right off the edge.
*
* CharID
*
*   Identifies character:
*   0 = kid
*   1 = shadow man
*   2 = guards, vizier
*   4 = skeleton
*   5 = princess (in princess scenes)
*   6 = vizier (in princess scenes)
*   24 = mouse
*
* CharSword
*
*   2: sword drawn
*   0: sword sheathed
*
* CharLife
*
*   -1: alive
*   0-127: dead
*
*------------------------------
*
*  Two permanent sets of CharData are maintained: KidData
*  (for the kid) and ShadData (for his opponent).  Note
*  that the opponent data is always referred to by the
*  prefix "Shad" although the character may be the shadow
*  man, skeleton, Vizier, etc.
*
*  CharData itself is used as temporary storage for whichever
*  character we want to deal with.  Typically, we will call
*  LoadKid to "load" the kid as our current character (i.e.,
*  load the 16 bytes of KidData into the CharData space),
*  then call the control routines that change CharData, then
*  when we're done call SaveKid to "save" the modified
*  CharData back into KidVars.  This way we can use the
*  same control routines for both the kid & his opponent.
*
*  There is a second data set used for temporary storage
*  OppData.  OppData always contains the "other" character
*  than CharData -- e.g., when we call LoadKid, we load
*  KidData into CharData and ShadData into OppData.
*
*------------------------------
```

INFO

The INFO block of the level blueprint specifies the starting positions & characteristics of the player & all guards in the level:

There are three player variables:

KidStartScrn	player starting screen (1-24)
KidStartBlock	starting block # on that screen (0-29)
KidStartFace	facing direction (-1 = left, 0 = right)

There are six guard variables. A complete set of six variables is maintained for each of the level's 24 screens.

GdStartBlock	starting block # on this screen (value >29 means no guard on this screen)
GdStartFace	facing direction (-1 = left, 0 = right)
GdStartX	starting X-coord (140-width coord)
GdStartSeqL˜H	starting posn in sequence table (2-byte)
GdStartProg	guard program # (0-11)

AUTO

ADDGUARD

On a cut to a new screen, we call ADDGUARD. ADDGUARD first checks for the hard-wired shadowman appearances on certain levels. (The shadowman is different from a normal guard and will be discussed later. The Vizier and the skeleton are treated as guards.) If none of these apply, we fall through to AddNormalGd.

AddNormalGd checks the variables described in the section above (in the INFO block of the blueprint for this screen) to see if there is a guard on the new screen.

Note: GdStartBlock is used to derive CharY. CharBlockX is derived from GdStartX. The precise value of GdStartBlock therefore doesn't matter; there are effectively only three values.

The guard's starting position is derived from GdStartSeqL˜H, which retain the 2-byte value of CharSeq. If GdStartSeqH = 0, that tells us to use the default CharSeq value (usually "alertstand"). Once we have set CharSeq, we call ANIMCHAR to get the initial value of CharPosn. The next step is to check CharPosn to see if the guard is dead or alive.

CUTCHECK

When the character leaves a screen, we need to save the new position of the guard (if any) on that screen--so that if he comes back to that screen later, the guard will be in the correct position. This is accomplished by CUTCHECK.

When we leave a screen, we have two choices: UPDATEGUARD or TRANSFERGUARD.

UPDATEGUARD updates all the guard data for the old screen (GdStartBlock, etc.) so that the guard will be in the correct position when we return. It also removes the guard as an active character (i.e., sets ShadFace = 86 and OppStrength = 0).

TRANSFERGUARD is used when we want the guard to follow the player to the next screen. In this case, all we need to do is remove the guard from his old screen (set GdStartBlock > 29 for that screen) and change CharScrn, CharX, & CharBlockX to put the guard on the new screen. Theoretically, a guard can follow the player across any number of screens (by repeated use of TRANSFERGUARD). When the player finally leaves the guard behind, UPDATEGUARD saves the guard's new position, so that the guard can end up many screens away from where he started.

CharFace = 86 is the code for "nonexistent character." You can also "get rid of" a character by setting his CharScrn ≠ VisScrn. Note that OppStrength is handled independently from the CharVars, so if you want to remove a guard's strength meter as well as the guard himself, you need to set OppStrength = 0 as well.

Shadow man

The shadowman is not treated as a guard. Throughout the program, various specialized bits of code check for specific circumstances which will cause the shadowman to appear, disappear, change position, etc.

The routine CSPS ("change shadowman position") puts the shadowman in a new position. His new position is specified by a data set of 8 bytes: CharPosn, X, Y, Face, BlockX, BlockY, Action, and the sequence table entry point.

ENEMY FIGHTING LOGIC

There are 12 different enemy fighting programs. Each is defined by a set of 8 values:

strikeprob
restrikeprob
blockprob
impblockprob
advprob
refractimer

specialcolor
extrastrength

The first 5 values specify fighting characteristics. E.g., STRIKEPROB = probability x255 of striking from a ready position. (0 = never, 255 = always).

REFRACTIMER is the length (# of frames) of the guard's "refractory period" after being hit.

SPECIALCOLOR specifies whether the guard's uniform is the basic color for that level (0) or the special color (1).

EXTRASTRENGTH specifies how many extra strength points (0-1) that guard gets above the basic strength for that level.

"Basic strength" and "basic color" are indexed by level number.

ENEMY SHAPE SETS

The enemy shape sets are:
guard
fat guard
Vizier
skeleton
shadowman

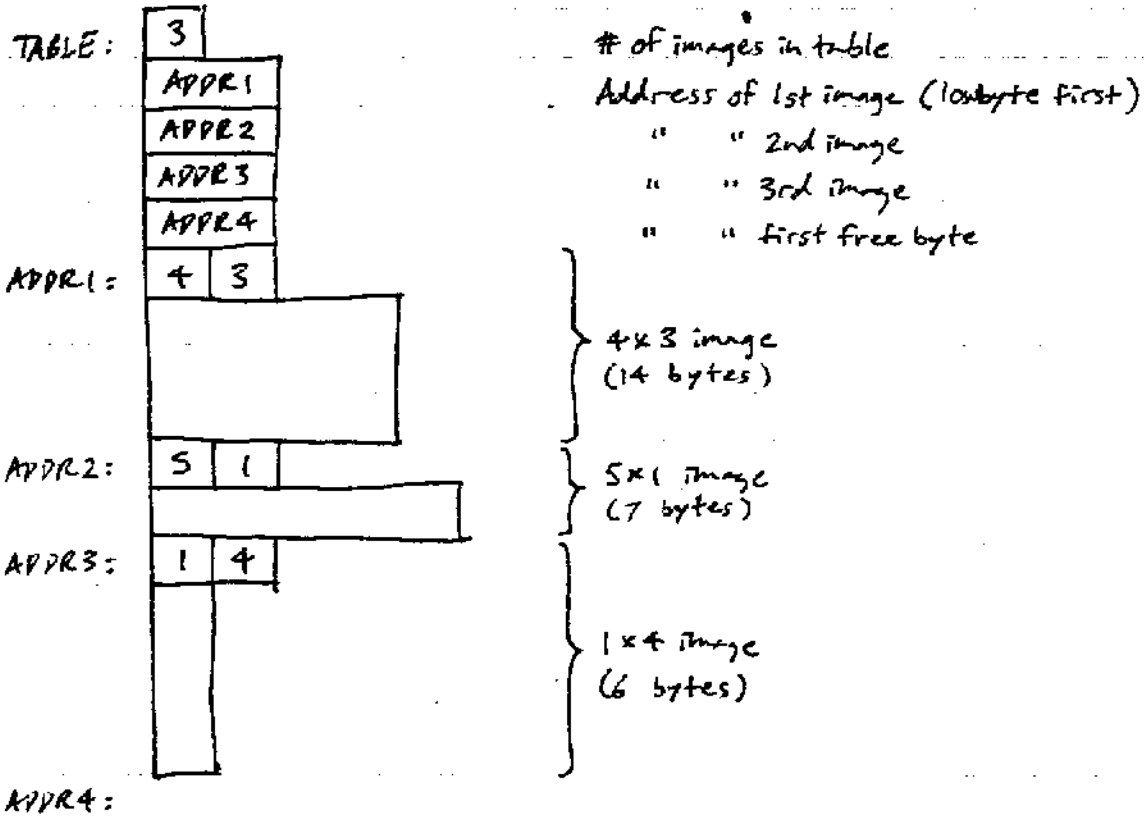
Since the enemy shape set (chtable4) is loaded in at the beginning of the level, all the enemies on a given level must use the same shape set. (The guards' uniform colors, however, can vary from screen to screen within the level.) This is why Levels 3 (skeleton) and Level 6 (fat guard) have no other enemies.

Level 12

Level 12 is actually three separate levels, all of which claim to be Level 12. The real Level 12 starts at the bottom of the tower. After you've defeated the shadowman and run off to the left, Level 13 is loaded in, with the Vizier shape set replacing the shadowman shape set. The first screen of Level 13 is identical to the last screen of Level 12, for continuity, but there is no way for you to get back. (If you die on Level 13, you will be restarted at the beginning of Level 13--but the message will still say Level 12.) After you kill the Vizier and climb the stairs at the end of Level 13, you go to the Princess's level, which is technically Level 14. (Levels 12 and 13 both use the dungeon background set, while Level 14 uses the palace set.)

If your time runs out while you're on Level 13, the game doesn't end (on the theory that while the Vizier is engaged in combat with you, he can't do anything to the Princess) until you die.

SAMPLE IMAGE TABLE



Note: Image data bytes are read off left-right, bottom-top.

BGTABLE 1

01	floor A
02	floor B
03	floor A mask
04	floor B mask
05	spikes A (1)
06	spikes B (2)
07	posts A
08	posts B
09	posts C
0A	gate A
0B	gate B
0C	gate C
0D	gate C mask
0E	posts half A (palace)
0F	mirror frontpiece
10	pillar bottom A
11	floor half mask
12	floor half A
13	rubble frontpiece
14	mirror A
15	floor D
16	panel w/floor D
17	panel w/o floor D
18	down pressplate D
19	pressplate D
1A	pillar bottom B
1B	loose floor B
1C	pillar top B
1D	pillar top C
1E	loose floor A (1)
1F	loose floor A (2)
20	rubble A
21	rubble B
22	spikes A (2)
23	spikes B (2)
24	spikes A (3)
25	spikes B (3)
26	spikes A (4)
27	spikes B (4)
28	spikes A (5)
29	spikes B (5)
2A	spikes A (6)
2B	spikes B (6)
2C	loose floor D (1)
2D	loose floor D (2)
2E	rubble D
2F	gate C (1)
30	gate C (2)
31	gate C (3)
32	gate C (4)
33	gate C (5)
34	gate C (6)
35	gate C (7)
36	gate C (8)
37	gate B (1)
38	gate B (2)
39	gate B (3)
3A	gate B (4)
3B	gate B (5)
3C	gate B (6)
3D	gate B (7)
3E	gate B (8)
3F	[redacted]
40	[redacted]
41	[redacted]
42	[redacted]
43	gate bottom (STA)
44	gate bottom (ORA)
45	posts frontpiece
46	gate frontpiece
47	stripe
48	pillar bottom frontpiece
49	pillar top frontpiece
4A	mirror B
4B	up pressplate A
4C	up pressplate D
4D	exit1 B
4E	exit2 B
4F	exit1 C
50	exit2 C
51	torch B
52	flame (1)
53	flame (2)
54	flame (3)
55	flame (4)
56	flame (5)
57	bottom jaw (1)
58	top jaw (2)
59	bottom jaw (2)
5A	top jaw (3)
5B	bottom jaw (3)
5C	top jaw (4)
5D	bottom jaw (4)
5E	top jaw (5)
5F	bottom jaw (5)
60	
61	flame (6)
62	flame (7)
63	flame (8)
64	flame (9)
65	jaw front (1)
66	jaw front (2)
67	jaw front (3)
68	jaw front (4)
69	jaw front (5)

6A	
6B	stairs
6C	door
6D	door mask
6E	door top
6F	block B (type 2)
70	"0"
71	"1"
72	"2"
73	"3"
74	"4"
75	"5"
76	"6"
77	"7"
78	"8"
79	"9"
7A	"LEVEL"
7B	empty message box
7C	"CONTINUE" msg box
7D	"MINUTES LEFT" msg box
7E	"TURN DISK OVER" msg box
7F	"SECONDS"

BGTABLE 2

01	panel B (type 2)
02	panel C (type 2)
03	block frontpiece (type 1)
04	block B (type 1)
05	block C (type 1)
06	block D (type 1)
07	flask
08	bullet
09	1+ bullet line
0A	2+ bullet line
0B	3+ bullet line
0C	blank
0D	
0E	smeared bottom jaw (1)
0F	smeared bottom jaw (2)
10	smeared bottom jaw (3)
11	balcony1 B
12	balcony2 B
13	balcony1 C
14	balcony2 C
15	special flask
16	
17	skeleton A
18	skeleton B
19	sword A
1A	panel B (type 1)
1B	panel C (type 1)
1C	
1D	
1E	panel B (type 0)
1F	panel C (type 0)
20	
21	panel w/arch A
22	floor B (type 1)
23	space B (type 1)
24	floor B (type 2)
25	space B (type 2)
26	space B (type 3: window)
27	arch support A
28	arch support frontpiece
29	arch 1 A
2A	arch 2 A
2B	arch 1 D
2C	arch 3 A
2D	arch 4 A
2E	arch top
2F	bubbles (1)
30	bubbles (2)
31	bubbles (3)
32	bubbles (4)
33	sword gleam

* = side B only
† = side A only

CHTABLE 6

	01	flame 1	
	02	flame 2	
	03	flame 3	
	04	flame 4	
	05	flame 5	
	06	flame 6	
	07	flame 7	
	08	flame 8	
	09	flame 9	
	0A	† pslump - 1	
	0B	† pslump - 2	
	0C	post	
	0D	glass 1	
	0E	glass 2	
	0F	glass 3	
	10	glass 4	
	11	glass 5	
	12	glass 6	
	13	glass 7	
	14	glass 8 (empty)	
	15	glass 0 (full)	
	16	sand 1	
	17	sand 2	
	18	sand 3	
	19	pturn-15	11
	1A	pturn-4	2
	1B	pturn-5	3
	1C	pturn-6	4
	1D	pturn-7	5
	1E	pturn-8	6
	1F	pturn-9	7
	20	pturn-10	8
	21	pturn-11	9
C6.A	22	† pslump mask	
	23	pback-3	12
	24	pback-5	13
	25	pback-7	14
	26	pback-9	15
	27	pback-11	16
	28	pback-13	17
	29	*plie	19
	2A	star 1 EOR	
	2B	star 2 EOR	
	2C	*plie mask	

	2D	*embrace-1	20
	2E	*embrace-2	21
	2F	*embrace-3	22
	30	*embrace-4	23
	31	*embrace-5	24
	32	*embrace-6	25
	33	*embrace-7	26
C6.B	34	*embrace-8	27
	35	*embrace-9	28
	36	*embrace-10	29
	37	*embrace-11	30
	38	*embrace-12	31
	39	*embrace-13	32
	3A	*embrace-14	33
	3B	*prise-1	34
	3C	*prise-2	35
	3D	*prise-3	36
	3E	*prise-4	37
	3F	*prise-5	38
	40	*prise-6	39
	41	*prise-7	40
C6.C	42	*prise-8	41
	43	*prise-9	42
	44	*prise-10	43
	45	*prise-11	44
	46	*prise-12	45
	47	*prise-13	46
	48	*prise-14	47
	49		
	4A	†vwalk-8	48
	4B	†vwalk-9	49
	4C	†vwalk-10	50
	4D	†vwalk-11	51
	4E	†vwalk-12	52
	4F	†vwalk-13	53
	50	†vwalk-14 (side A)	54
	51	†vstop-1	55
C6.D	52	†vstop-2	56
	53	†vturn-5	57
	54	†vturn-6	58
	55	†vturn-7	59
	56	†vturn-8	60
	57	†vturn-9	61
	58	†vturn-10	62
	59	†vturn-11	63
	5A	†vturn-12	64
	5B	†vturn-13	65
	5C	†vturn-14	66
	5D	†vcast-2	67

	5E	†vcast-3	68
	5F	†vcast-4	69
	60	†vcast-5	70
	61	†vcast-6	71
	62	†vcast-7	72
	63	†vcast-8	73
	64	†vcast-9	74
	65	†vcast-10	75
C6.E	66	†vcast-11(wd)	76
	67	†vcast-13	77

CHTABLE 7

01	†vcast-14	78
02	†vcast-15	79
03	†vcast-16	80
04	†vcast-17	81
05	†vcast-18	82
06	†vcast-10a	83
07	†vcast-10b	84
08	†vcast-1	85

PRINCE OF PERSIA--Cheat Key Version
August 27, 1989

SKIP	Skip to next level (up to Level 4)--Reduces time remaining to 15 minutes
???	Enable cheat keys

With cheat keys enabled:

[Return]	Disable cheat keys
SKIP	Skip to next level (up to Level 12)
[redacted]	
TINA	Go to end of level 12
R	Restore full strength
BOOST	Boost max strength (origstrength) by 1
Z	Reduce guard to 1 unit of strength
ZAP	Zap guard (he drops dead)

Prince of Persia sound effects

Note: Not all of these are in the Apple version.

Footsteps
Soft landing
Medium landing ("Oof!")
Hard landing (Splat!)
Sword clash
Stab opponent
Stab skeleton
Stabbed by opponent
Bones leap to life
Impaled by spikes
Slicer blades clash
Character gets sliced in half
Gate rising
Gate stops at top
Gate coming down slow
Gate reaches bottom (Clang!)
Gate crashes down
Entrance door closes
Exit door opening
Bump into wall (soft)
Bump into wall (hard)
Bump into mirror
Falling floor lands on your head
Loose floor shakes
Falling floor lands
Drink potion--1 unit of strength restored
Drink special potion--strength boosted to higher level
Drink poison (lose 1 unit of strength)
Unsheathe sword
Jump through mirror
Grab on to ledge
Drink potion (glug glug)