```
-- edited by Sandman on Jun 22, 1979 10:20 AM

PACK MazeWar, TheMaze, Tables, Rats, MazeDisplay;

Maze: CONFIGURATION
  IMPORTS FrameDefs, ImageDefs, MiscDefs  ProcessDefs,
    SegmentDefs, StreamDefs, StringDefs, SystemDefs
  CONTROL MazeWar =
  BEGIN

  TinyPup;

  TheMaze;
  Tables;
  Rats;
  MazeDisplay;
  MazeWar;
  NMFont;

  MazeInit;  -- Item 3 in NMOps is unbound
  NMDisplay;
  NMKeyboard;
  StreamIO;

  END.
```

```
-- MazeDefs.mesa; edited by Guyton; June 25, 1979  11:02 AM

DIRECTORY
  PupDefs: FROM "pupdefs" USING [PupAddress],
  PupTypes: FROM "puptypes" USING [PupSocketID, PupType];

MazeDefs: DEFINITIONS =
BEGIN

MazeType: TYPE =
  POINTER TO ARRAY [0..16) OF PACKED ARRAY [0..32) OF BOOLEAN;
Direction: TYPE = {NORTH, SOUTH,  EAST, WEST};
XYpair: TYPE = RECORD [p1,p2: XYpoint];
XYpoint: TYPE = RECORD [x,y: INTEGER];
XY:   TYPE = RECORD[xcor: INTEGER, ycor: INTEGER];
szXYpair:       CARDINAL = SIZE[XYpair];
szTwoXYpair:    CARDINAL = 2*SIZE[XYpair];

View: TYPE = {left, right, rear, front};

ratLocation: PupTypes.PupType = pt170;
ratKill: PupTypes.PupType = pt171;
ratDead: PupTypes.PupType = pt172;
ratStatus: PupTypes.PupType = pt173;
ratNew: PupTypes.PupType = pt174;
ratGoing: PupTypes.PupType = pt175;
ratQuery: PupTypes.PupType = pt176;
ratAlive: PupTypes.PupType = pt177;
ratSurvey: PupTypes.PupType = pt100;
Password: CARDINAL = 123466B;
mazeSocket: PupTypes.PupSocketID = [046501B, 055105B];  -- Magic socket

Loc: TYPE = CARDINAL [0..32];
Score: TYPE = INTEGER;
RatName: TYPE = PACKED ARRAY [0..20) OF CHARACTER;
MaxRats: CARDINAL = 8;
RatId: TYPE = CARDINAL [0..MaxRats);                    -- Index into rats array

RatLocation: TYPE = POINTER TO AqRatLocation;
AqRatLocation: TYPE = RECORD [
  ratId: RatId,
  xLoc: Loc,
  yLoc: Loc,
  dir: Direction,
  score: Score];

RatKill: TYPE = POINTER TO AqRatKill;
AqRatKill: TYPE = RECORD [
  ratId: RatId,
  xLoc: Loc,
  yLoc: Loc,
  dir: Direction];

RatDead: TYPE = POINTER TO AqRatDead;
AqRatDead: TYPE = RECORD [
  ratId: RatId,
  killedBy: RatId];

RatNew: TYPE = POINTER TO AqRatNew;
AqRatNew: TYPE = RECORD [
  pass: CARDINAL,
  xLoc: Loc,
  yLoc: Loc,
  dir: Direction,
  addr: PupDefs.PupAddress,
  name: RatName];

RatGone: TYPE = POINTER TO AqRatGone;
AqRatGone: TYPE = RECORD [
  ratId: RatId];

RatQuery: TYPE = POINTER TO AqRatQuery;
AqRatQuery: TYPE = RECORD [
  ratId: RatId];

RatAlive: TYPE = POINTER TO AqRatAlive;
```

```
AqRatAlive: TYPE = RECORD [
  ratId: RatId];

RatStatus: TYPE = POINTER TO RatCb;

RatObject: TYPE = RECORD [
  playing: BOOLEAN,
  xLoc, yLoc: Loc,
  dir: Direction,
  score: Score,
  addr: PupDefs.PupAddress,
  name: RatName];

RatInfo: TYPE = POINTER TO RatObject;

RatCb: TYPE = RECORD [
  dukeRat: RatId,
  rats: ARRAY RatId OF RatObject];

TokenId:        TYPE = View;

R2d2: TYPE = ARRAY RatId OF RatAppearance;

RatLook: TYPE = POINTER TO RatAppearance;

RatAppearance: TYPE = RECORD [
  visible: BOOLEAN,
  x, y: CARDINAL,       -- bitmap location of the token bitmap
  distance: CARDINAL,   -- distance away from the viewer
  tokenId: TokenId];

RelativeTokens: TYPE = ARRAY Direction OF ARRAY Direction OF TokenId;

RatPupHealth: TYPE = MACHINE DEPENDENT RECORD [
 pupSend: BOOLEAN,
 pupRcvd: BOOLEAN,
 count: CARDINAL [0..37777B] ];

RatHealth: TYPE = ARRAY RatId OF RatPupHealth;

TheMaze: PROGRAM;
Tables: PROGRAM;
Rats: PROGRAM;
MazeInit: PROGRAM;

DispatchKeys: PROCEDURE;
NewPosition: PROCEDURE;
NewScoreCard: PROCEDURE;
ReadKeys: PROCEDURE;
ReadRats: PROCEDURE;
TokenVisible: PROCEDURE [hisRatId: RatId];
ShowView: PROCEDURE [x, y: CARDINAL, dir: Direction];
RatDoctor: PROCEDURE;

END.
```

```
-- MazeDisplay.mesa Edited by Gobbel on June 22, 1979  3:51 PM
--Hacked version by: BAM; Last edit:  October 28, 1979  4:31 PM
DIRECTORY
  BitBltDefs: FROM "bitbltdefs" USING [BBptr, BBTable, BITBLT],
  FontDefs: FROM "fontdefs" USING [BitmapState],
  InlineDefs: FROM "inlinedefs" USING [BITAND, DIVMOD, BITOR],
  MazeDefs: FROM "mazedefs" USING [Direction, R2d2, RatId, View, XYpair, MaxRats],
  MazeDisplayDefs: FROM "mazedisplaydefs" USING
    [DCBHandle, DCBRec, DoubleWord, pRat],
  Mopcodes: FROM "mopcodes" USING [zINC];

MazeDisplay: MONITOR
  IMPORTS BitBltDefs, InlineDefs
  EXPORTS MazeDisplayDefs =
PUBLIC BEGIN OPEN MazeDisplayDefs;


RatId: TYPE = MazeDefs.RatId;

pages: CARDINAL = 6;
sbmr: CARDINAL = pages*4;
sbca: POINTER;
topMargin, current, hidden, bigMazeMap, scoreDCB: DCBHandle;  --mazeMap removed
font: PUBLIC Fptr;
ratBB, lineBB, diagBB: PUBLIC BitBltDefs.BBptr;
leftBBpattern, rightBBpattern: ARRAY [0..32) OF DoubleWord ← ALL[[0, 0]];

myRatId: RatId ← 0;
Invincible: BOOLEAN ← FALSE;
BigMaze: BOOLEAN ← FALSE;
halt: BOOLEAN ← FALSE;

RatState: TYPE = RECORD
        [
        playing: BOOLEAN,
        x, y: CARDINAL,
        dir: MazeDefs.Direction
        ];

ClearArray: ARRAY[0..MazeDefs.MaxRats) OF RatState ←.ALL[[FALSE,1,1,NORTH]];

SetMyRatId: PROCEDURE [ratId: RatId] =
  BEGIN
  myRatId ← ratId;
  ClearArray[myRatId].playing ← TRUE;
  END;


DrawVerticalLine: PROCEDURE [x, y, height: CARDINAL] = INLINE
  BEGIN
IF halt THEN RETURN;
  lineBB.dw ← 1;
  lineBB.dh ← height;
  lineBB.dlx ← x;
  lineBB.dty ← y;
  BitBltDefs.BITBLT[lineBB];
  END;

DrawHorizontalLine: PROCEDURE [x, y, width: CARDINAL] = INLINE
  BEGIN
IF halt THEN RETURN;
  lineBB.dlx ← x;
  lineBB.dty ← y;
  lineBB.dh ← 1;
  lineBB.dw ← width;
  BitBltDefs.BITBLT[lineBB];
  END;

-- Only for slopes of 1 or -1
DrawDiagonalLine: PROCEDURE [x1,y1,x2,y2: INTEGER] = INLINE
  BEGIN
  width: CARDINAL = x2-x1;
  i, rem, count: CARDINAL;
  rightSlope: BOOLEAN = y1>y2;
IF halt THEN RETURN;
  [count, rem] ← InlineDefs.DIVMOD[width, 32];
```

```
  diagBB.dbca ← hidden.bitmap;
  diagBB.sbca ← IF rightSlope THEN @rightBBpattern ELSE @leftBBpattern;
  IF count>0 THEN BEGIN
    diagBB.dw ← diagBB.dh ← 32;
    IF rightSlope THEN
      BEGIN
      FOR i IN [0..count) DO
        diagBB.dlx ← x1;
        diagBB.dty ← y1-32;
        BitBltDefs.BITBLT[diagBB];
        x1 ← x1+32;
        y1 ← y1-32;
        ENDLOOP;
      END
    ELSE
      BEGIN
      FOR i IN [0..count) DO
        diagBB.dlx ← x1;
        diagBB.dty ← y1;
        BitBltDefs.BITBLT[diagBB];
        x1 ← x1+32;
        y1 ← y1+32;
        ENDLOOP;
      END;
    END;
  IF rem>0 THEN
    BEGIN
    diagBB.dw ← diagBB.dh ← rem;
    diagBB.dlx ← x1;
    diagBB.dty ← IF rightSlope THEN y1-rem ELSE y1;
    IF rightSlope THEN diagBB.sty ← 32-rem;
    BitBltDefs.BITBLT[diagBB];
    diagBB.sty ← 0;
    END;
  RETURN;
  END;

MakeBigMaze: PUBLIC PROCEDURE =
BEGIN
i: CARDINAL;
IF  halt OR BigMaze THEN RETURN;
halt ← TRUE;  --multi-process coordination
--FOR i IN [0..MazeDefs.MaxRats) DO
--      IF ClearArray[i].playing THEN
--              ClearSquare[ClearArray[i].x, ClearArray[i].y];
--      ENDLOOP;
BigMaze ← TRUE;
--now switch bit maps
topMargin.next ← bigMazeMap;
bigMazeMap.next ← scoreDCB;
halt ← FALSE;
--DisplayAll[];
END;
MakeSmallMaze: PUBLIC PROCEDURE =
BEGIN
i: CARDINAL;
IF  halt OR ~BigMaze THEN RETURN;
halt ← TRUE;  --multi-process coordination
FOR i IN [0..MazeDefs.MaxRats) DO
        IF ClearArray[i].playing THEN
                BigClearSquare[ClearArray[i].x, ClearArray[i].y];
        ENDLOOP;
BigMaze ← FALSE;
--now switch bit maps
topMargin.next ← current;
--current.next ← mazeMap;
--mazeMap.next ← scoreDCB;
current.next ← scoreDCB;
halt ← FALSE;
BigDisplayAll[];
END;
--DisplayAll: PROCEDURE =
--BEGIN
--i: CARDINAL;
--IF halt THEN RETURN;
--FOR i IN [0..MazeDefs.MaxRats) DO OPEN ClearArray[i];
```

```
--      IF playing THEN
--              IF i=myRatId THEN ShowMe[x, y, Invincible, dir]
--              ELSE ShowOther[i, x, y, dir];
--      ENDLOOP;
--END;
BigDisplayAll: PROCEDURE =
BEGIN
i: CARDINAL;
IF halt THEN RETURN;
FOR i IN [0..MazeDefs.MaxRats) DO OPEN ClearArray[i];
        IF playing THEN
                IF i=myRatId THEN BigShowMe[x, y, Invincible, dir]
                ELSE BigShowOther[i, x, y, dir];
        ENDLOOP;
END;

--ClearPosition: PROCEDURE [ratId: RatId, xClear, yClear: CARDINAL] =
--  BEGIN
--  tempRatId: RatId;
--IF halt THEN RETURN;
--  ClearSquare[xClear, yClear];
--  now  check if anyone needs to be redrawn
--    FOR tempRatId IN RatId DO OPEN ClearArray[tempRatId];
--      IF tempRatId=ratId OR ~playing THEN LOOP
--      IF x = xClear AND y = yClear THEN
--              BEGIN
--              IF tempRatId=myRatId THEN ShowMe[x, y, Invincible, dir]
--              ELSE ShowOther[tempRatId, x, y, dir];
--              EXIT;
--              END;
--      ENDLOOP;
--  RETURN;
--  END;
--
--ClearSquare: PROCEDURE [xClear, yClear: CARDINAL] =
--BEGIN
--  mazeIndex: CARDINAL ← xClear*32*8 + yClear;
--  lastIndex: CARDINAL = mazeIndex+8*32;
--  maze: POINTER TO PACKED ARRAY OF [0..377B] ← mazeMap.bitmap;
--  DO
--    maze[mazeIndex] ← 0;
--    mazeIndex ← mazeIndex + 32;
--    IF mazeIndex = lastIndex THEN EXIT;
--    ENDLOOP;
--END;
--
BigClearSquare: PROCEDURE [xClear, yClear: CARDINAL] =
BEGIN
  mazeIndex: CARDINAL ← xClear*32*16 + yClear;
  maze: POINTER TO PACKED ARRAY OF WORD ← bigMazeMap.bitmap;
  THROUGH [0..16) DO
    maze[mazeIndex] ← 0;
    mazeIndex ← mazeIndex + 32;
    ENDLOOP;
END;

BigClearPosition: PROCEDURE [ratId: RatId, xClear, yClear: CARDINAL] =
  BEGIN
  tempRatId: RatId;
IF halt THEN RETURN;
  BigClearSquare[xClear, yClear];
  --now  check if anyone needs to be redrawn
    FOR tempRatId IN RatId DO OPEN ClearArray[tempRatId];
        IF tempRatId=ratId OR ~playing THEN LOOP;
        IF x = xClear AND y = yClear THEN
                BEGIN
                IF tempRatId=myRatId THEN BigShowMe[x, y, Invincible, dir]
                ELSE BigShowOther[tempRatId, x, y, dir];
                EXIT;
                END;
        ENDLOOP;
RETURN;
  END;


InvertTop: PROCEDURE =
```

```
        BEGIN
        SELECT topMargin.background FROM
                white => topMargin.background ← black;
                black => topMargin.background ← white;
                ENDCASE;
        RETURN;
        END;

ShowPosition: PROCEDURE [xloc, yloc: CARDINAL, invincible: BOOLEAN,
                         tdir: MazeDefs.Direction] =
        BEGIN
        IF halt THEN RETURN;
        IF BigMaze THEN
                BEGIN
                BigClearPosition[myRatId, ClearArray[myRatId].x, ClearArray[myRatId].y];
                BigShowMe[xloc, yloc, invincible, tdir];
                END
--      ELSE    BEGIN
--              ClearPosition[myRatId, ClearArray[myRatId].x, ClearArray[myRatId].y];
--              ShowMe[xloc, yloc, invincible, tdir];
--              END;
        END;

--ShowMe: PROCEDURE [xloc, yloc: CARDINAL, invincible: BOOLEAN, tdir: MazeDefs.Direction] =
--  BEGIN
--  i, mazeIndex, arrowIndex: CARDINAL;
--  maze: POINTER TO PACKED ARRAY OF [0..377B] ← mazeMap.bitmap;
--  normalarrows: PACKED ARRAY [0..32) OF [0..377B] = [
--    0B, 10B, 14B, 376B, 377B, 376B, 14B, 10B,
--    20B, 60B, 177B, 377B, 177B, 60B, 20B, 0B,
--    34B, 34B, 34B, 34B, 177B, 76B, 34B, 10B,
--    10B, 34B, 76B, 177B, 34B, 34B, 34B, 34B];
--  invinciblearrows: PACKED ARRAY [0..32) OF [0..377B] = [
--    0B, 30B, 214B, 256B, 377B, 256B, 214B, 30B,
--    30B, 61B, 165B, 377B, 165B, 61B, 30B, 0B,
--    76B, 10B, 34B, 111B, 177B, 76B, 34B, 10B,
--    10B, 34B, 76B, 177B, 111B, 34B, 10B, 76B];
--  arrowIndex ← LOOPHOLE[tdir, CARDINAL]*8;
--  mazeIndex ← xloc*32*8 + yloc;
--  IF invincible THEN
--        FOR i IN [0..8) DO
--          maze[mazeIndex] ← invinciblearrows[arrowIndex+i];
--          mazeIndex ← mazeIndex + 32;
--          ENDLOOP
--  ELSE FOR i IN [0..8) DO
--    maze[mazeIndex] ← normalarrows[arrowIndex+i];
--    mazeIndex ← mazeIndex + 32;
--    ENDLOOP;
--  ClearArray[myRatId] ← [playing: TRUE, x: xloc, y: yloc, dir: tdir]; Save for clearing next ti
--  Invincible ← invincible;
--  RETURN;
--  END;
--
BigShowMe: PROCEDURE [xloc, yloc: CARDINAL, invincible: BOOLEAN,
                         tdir: MazeDefs.Direction] =
  BEGIN
  i, mazeIndex: CARDINAL;
  maze: POINTER TO PACKED ARRAY OF WORD ← bigMazeMap.bitmap;
  normalArrows: ARRAY MazeDefs.Direction OF ARRAY [0..16) OF WORD = [
        [0B,    200B,   300B,   340B,   --right
         360B,  370B,   177774B,        177776B,
         177777B,       177776B,        177774B,
         360B,  340B,   300B,   200B],
        [400B,  1400B,  3400B,  7400B,  --left
         17400B,        37777B, 77777B, 177777B,
         77777B,        37777B, 17400B, 7400B,
         3400B, 1400B,  400B,   0B],
        [1740B, 1740B,  1740B,  1740B,  --down
         1740B, 1740B,  1740B,  1740B,
         77777B,        37776B, 17774B, 7770B,
         3760B, 1740B,  700B,   200B],
        [200B,  700B,   1740B,  3760B,  --up
         7770B, 17774B, 37776B, 77777B,
         1740B, 1740B,  1740B,  1740B,
         1740B, 1740B,  1740B,  1740B]];
  invincibleArrows: ARRAY MazeDefs.Direction OF ARRAY [0..16) OF WORD = [
```

```
        [0B,    600B,   300B,   340B,   --right
         160B,  210B,   177464B,        177102B,
         177043B,       177022B,        177544B,        210B,
         160B,  340B,   300B,   600B],
        [700B,  1400B,  3400B,  7000B,  --left
         10400B,        233778,  44177B, 142177B,
         41177B,        263778,  10400B, 7000B,
         3400B, 1400B,  700B,   0B],
        [1740B, 1740B,  1740B,  1740B,  --down
         1740B, 1740B,  1740B,  41041B,
         72327B,        34416B,  14214B, 4110B,
         2620B, 1040B,  700B,   200B],
        [200B,  700B,   1040B,  2320B,  --up
         4410B, 14214B, 34116B, 72627B,
         41041B,        1740B,  1740B,  1740B,
         1740B, 1740B,  1740B,  1740B]];
  mazeIndex ← xloc*32*16 + yloc;
  IF invincible THEN
        FOR i IN [0..16) DO
        maze[mazeIndex] ← invincibleArrows[tdir][i];
        mazeIndex ← mazeIndex + 32;
        ENDLOOP
  ELSE
        FOR i IN [0..16) DO
        maze[mazeIndex] ← normalArrows[tdir][i];
        mazeIndex ← mazeIndex + 32;
        ENDLOOP;
  ClearArray[myRatId] ← [playing: TRUE, x: xloc, y: yloc, dir: tdir];-- Save for clearing next
  Invincible ← invincible;
  RETURN;
  END;


DisplayOthersPosition: PROCEDURE [which, xloc, yloc: CARDINAL,
                                  tdir: MazeDefs.Direction] =
        BEGIN
IF halt THEN RETURN;
        IF BigMaze THEN
                BEGIN
                BigClearPosition[which, ClearArray[which].x, ClearArray[which].y];
                IF ClearArray[which].playing THEN BigShowOther[which, xloc, yloc, tdir];
                END
--      ELSE    BEGIN
--              ClearPosition[which, ClearArray[which].x, ClearArray[which].y];
--              IF ClearArray[which].playing THEN ShowOther[which, xloc, yloc, tdir];
--              END;
        END;

--ShowOther: PROCEDURE [which, xloc, yloc: CARDINAL,
--                                  tdir: MazeDefs.Direction] =
--  BEGIN
--  i, mazeIndex, arrowIndex, oldX, oldY: CARDINAL;
--  maze: POINTER TO PACKED ARRAY OF [0..377B] ← mazeMap.bitmap;
--  arrows: PACKED ARRAY [0..32) OF [0..377B] = [
--    0B, 10B, 14B, 376B, 203B, 376B, 14B, 10B,
--    20B, 60B, 177B, 301B, 177B, 60B, 20B, 0B,
--    34B, 24B, 24B, 24B, 167B, 66B, 34B, 10B,
--    10B, 34B, 66B, 167B, 24B, 24B, 24B, 34B];
--      arrowIndex ← LOOPHOLE[tdir, CARDINAL]*8;
--      mazeIndex ← xloc*32*8 + yloc;
--      FOR i IN [0..8) DO
--              maze[mazeIndex] ← arrows[arrowIndex+i];
--              mazeIndex ← mazeIndex + 32;
--              ENDLOOP;
--      ClearArray[which] ← [playing: TRUE, x: xloc, y: yloc,
--  RETURN;
--  END;
--
BigShowOther: PROCEDURE [which, xloc, yloc: CARDINAL,
                                  tdir: MazeDefs.Direction] =
  BEGIN
  i, mazeIndex: CARDINAL;
  maze: POINTER TO PACKED ARRAY OF WORD ← bigMazeMap.bitmap;
  arrows: ARRAY MazeDefs.Direction OF ARRAY [0..16) OF WORD = [
        [0B,    200B,   300B,   340B,   --right
         177660B,       100030B,        100014B,        100006B,
```

```
        100003B,        100006B,        100014B,        100030B,
        177660B,        340B,   300B,   200B],
       [0B,     400B,   1400B,  3400B,  --left
        8777B,  14001B, 30001B, 60001B,
        140001B,        60001B, 30001B, 14001B,
        8777B,  3400B,  1400B,  400B],
       [7770B,  4010B,  4010B,  4010B,  --down
        4010B,  4010B,  4010B,  4010B,
        74017B,         34016B, 14014B, 6030B,
        3060B,  1540B,  700B,   200B],
       [200B,   700B,   1540B,  3060B,  --up
        6030B,  14014B, 30006B, 74014B,
        4010B,  4010B,  4010B,  4010B,
        4010B,  4010B,  4010B,  7770B]];
  numbers: ARRAY [0..7] OF ARRAY [0..16) OF WORD = [
       [0,      0,      0,      0,      --0
        0,      0,      300B,   440B,
        440B,   440B,   300B,   0,
        0,      0,      0,      0],
       [0,      0,      0,      0,      --1
        0,      0,      600B,   200B,
        200B,   200B,   700B,   0,
        0,      0,      0,      0],
       [0,      0,      0,      0,      --2
        0,      0,      600B,   1100B,
        200B,   400B,   1700B,  0,
        0,      0,      0,      0],
       [0,      0,      0,      0,      --3
        0,      0,      1600B,  100B,
        600B,   100B,   1600B,  0,
        0,      0,      0,      0],
       [0,      0,      0,      0,      --4
        0,      0,      1100B,  1100B,
        1740B,  100B,   100B,   0,
        0,      0,      0,      0],
       [0,      0,      0,      0,      --5
        0,      0,      1700B,  1000B,
        1700B,  100B,   1700B,  0,
        0,      0,      0,      0],
       [0,      0,      0,      0,      --6
        0,      0,      1000B,  1000B,
        1700B,  1100B,  1700B,  0,
        0,      0,      0,      0],
       [0,      0,      0,      0,      --7
        0,      0,      1700B,  100B,
        200B,   400B,   1000B,  0,
        0,      0,      0,      0]];
      mazeIndex ← xloc*32*16 + yloc;
      FOR i IN [0..16) DO
            maze[mazeIndex] ← InlineDefs.BITOR[arrows[tdir][i], numbers[which][i]];
            mazeIndex ← mazeIndex + 32;
            ENDLOOP;
      ClearArray[which] ← [playing: TRUE, x: xloc, y: yloc, dir: tdir];
  RETURN;
  END;

ExitPlayer: PROCEDURE [ratId: RatId] =
      BEGIN
      IF BigMaze THEN BigClearPosition[ratId, ClearArray[ratId].x, ClearArray[ratId].y];
      ELSE ClearPosition[ratId, ClearArray[ratId].x, ClearArray[ratId].y];
      ClearArray[ratId].playing ← FALSE;
      END;
AddNewPlayer: PROCEDURE [ratId: RatId, xloc, yloc: CARDINAL, tdir: MazeDefs.Direction] =
      BEGIN
      ClearArray[ratId].playing ← TRUE;
      DisplayOthersPosition[ratId, xloc, yloc, tdir];
      END;

ShowAllPositions: PROCEDURE [rats: pRat] =
      BEGIN
      ratId: RatId;
      FOR ratId IN RatId DO
            IF ratId = myRatId THEN LOOP;
            IF ClearArray[ratId].playing  AND ~rats[ratId].playing THEN
                        ExitPlayer[ratId]
            ELSE IF ~ClearArray[ratId].playing AND rats[ratId].playing THEN
```

```
                                AddNewPlayer[ratId, rats[ratId].xLoc,
                                    rats[ratId].yLoc, rats[ratId].dir]
                ELSE IF ClearArray[ratId].playing THEN DispTayOthersPosition[ratId,
                                    rats[ratId].xLoc, rats[ratId].yLoc, rats[ratId].dir]
                ENDLOOP;
        END;

--*****************************************************************************--

EVEN: PROCEDURE[v: UNSPECIFIED] RETURNS [UNSPECIFIED] =
  BEGIN RETURN[v+InlineDefs.BITAND[v, 1]] END;

PlotLine: PROCEDURE [p: POINTER TO MazeDefs.XYpair, two: BOOLEAN]
  RETURNS [POINTER TO MazeDefs.XYpair] =
  BEGIN
  fromx, fromy, tox, toy, ix, iy: INTEGER;
  fromx ← 0;  -- force an fgt entry
  lineBB.dbca ← hidden.bitmap;
  DO
    [x: fromx, y: fromy] ← p.p1;
    [x: tox, y: toy] ← p.p2;
    SELECT TRUE FROM
      fromx = tox =>  -- vertical?
        BEGIN
        y: CARDINAL ← MIN[fromy, toy];
        h: CARDINAL ← ABS[fromy-toy];
        IF h # 0 THEN DrawVerticalLine[fromx, y, h];
        END;
      fromy = toy =>  -- horizontal?
        BEGIN
        x: CARDINAL ← MIN[fromx, tox];
        w: CARDINAL ← ABS[fromx-tox];
        IF w # 0 THEN DrawHorizontalLine[x, fromy, w];
        END;
      ENDCASE =>        BEGIN
        IF fromx > tox THEN  -- diag routine is dumb, be careful for it
          BEGIN
          iy←fromy; fromy←toy; toy←iy;
          ix←fromx; fromx←tox; tox←ix;
          END;
        DrawDiagonalLine[fromx,fromy,tox,toy];
        END;
    p ← p + SIZE[MazeDefs.XYpair];
    IF NOT two THEN RETURN[p];
    two ← FALSE;
    ENDLOOP;
  END;

XORToken: ENTRY PROCEDURE [
  hisRatId: MazeDefs.RatId, r2d2: POINTER TO MazeDefs.R2d2] =
  BEGIN OPEN MazeDefs;
  sw, sh: CARDINAL;
  [slx: ratBB.slx, sty: ratBB.sty, sw: sw, sh: sh] ← Rat[
    ratId: hisRatId, distance: r2d2[hisRatId].distance,
    view: r2d2[hisRatId].tokenId];
IF halt THEN RETURN;
  ratBB.dbca ← current.bitmap;
  ratBB.dlx ← r2d2[hisRatId].x-sw/2;
  ratBB.dty ← r2d2[hisRatId].y-sh/2;
  ratBB.dw ← sw;
  ratBB.dh ← sh;
  BitBltDefs.BITBLT[ratBB];
  RETURN;
  END;

Switch: PROCEDURE RETURNS [DCBHandle]=
  BEGIN
  temp: DCBHandle ← current;
IF halt THEN ERROR;  --!!!!!!???????!!!*********!!!!
  current ← hidden;
  topMargin.next ← current;
  hidden ← temp;
  RETURN [hidden]
  END;

SetDCBs: PROCEDURE [p: POINTER TO DCBRec, rats: POINTER] =
```

```
  BEGIN OPEN p;
  topMargin ← top;
  current ← curr;
  hidden ← hid;
--  mazeMap ← maze;
  bigMazeMap ← bigMaze;
  scoreDCB ← score;
  sbca ← rats;
  END;

View: TYPE = MazeDefs.View;

Succ: PROCEDURE [view: View] RETURNS [View] =
  MACHINE CODE BEGIN Mopcodes.zINC END;

Rat: PUBLIC PROCEDURE [ratId, distance: CARDINAL, view: View]
  RETURNS [slx, sty, sw, sh: CARDINAL] =
  BEGIN
  viewT: View ← FIRST[View];
  SELECT distance FROM
    1 => BEGIN
      slx ← 0;
      FOR viewT IN View WHILE viewT#view DO slx ← slx+64 ENDLOOP;
      RETURN [slx, 0, 64, 64];
      END;
    2 =>
      FOR slx ← 4*64, slx+32 WHILE slx # 64*5 DO
        FOR sty ← 0, sty+32 WHILE sty # 64 DO
          IF viewT=view THEN RETURN [slx, sty, 32, 32];
          viewT ← Succ[viewT]
          ENDLOOP;
        ENDLOOP;
    3 =>
      FOR slx ← 5*64, slx+24 WHILE slx # 64*5+48 DO
        FOR sty ← 0, sty+24 WHILE sty # 48 DO
          IF viewT=view THEN RETURN [slx, sty, 24, 24];
          viewT ← Succ[viewT]
          ENDLOOP;
        ENDLOOP;
    4,5 =>
      FOR sty ← 0, sty+16 WHILE sty # 64 DO
        IF viewT=view THEN RETURN [64*5+48, sty, 16, 16];
        viewT ← Succ[viewT]
        ENDLOOP;
    6,7,8 =>
      FOR slx ← 64*5, slx+9 WHILE slx # 64*5+4*9 DO
        IF viewT=view THEN RETURN [slx, 48, 9, 9];
        viewT ← Succ[viewT]
        ENDLOOP;
    IN [9..12] =>
      FOR slx ← 64*5, slx+6 WHILE slx # 64*5+4*6 DO
        IF viewT=view THEN RETURN [slx, 48+9, 6, 6];
        viewT ← Succ[viewT]
        ENDLOOP;
    IN [13..18] =>
      FOR slx ← 64*5+4*6, slx+4 WHILE slx # 64*5+4*6+4*4 DO
        IF viewT=view THEN RETURN [slx, 48+9+3, 4, 4];
        viewT ← Succ[viewT]
        ENDLOOP;
    ENDCASE =>
      FOR slx ← 64*5+4*6, slx+3 WHILE slx # 64*5+4*6+4*3 DO
        IF viewT=view THEN RETURN [slx, 48+9, 3, 3];
        viewT ← Succ[viewT]
        ENDLOOP;
  ERROR -- can't happen
  END;  -- of Rat

  FHptr: TYPE = POINTER TO FontHeader;
  Fptr: TYPE = POINTER TO Font;
  FCDptr: TYPE = POINTER TO FCD;
  FAptr: TYPE = POINTER TO FontArray;
  FontArray: TYPE = ARRAY [0..255] OF FCDptr;

  Font: TYPE = MACHINE DEPENDENT RECORD [
    header: FontHeader,
    FCDptrs: FontArray,          -- array of self-relative pointers to
```

```
    -- FCD's.  Indexed by char value.
    -- font pointer points hear!
  extFCDptrs: FontArray       --  array of self-relative pointers to
    -- FCD's for extentions.  As large an
    -- array as needed.
  ];

FontHeader: TYPE = MACHINE DEPENDENT RECORD
  [
  maxHeight: CARDINAL,        -- height of tallest char in font (scan lines)
  variableWidth: BOOLEAN,     -- IF TRUE, proportionally spaced font
  blank:       [0..177B],     -- not used
  maxWidth: [0..377B] -- width of widest char in font (raster units).
  ];

FCD: TYPE = MACHINE DEPENDENT RECORD [
  widthORext: [0..77777B],    -- width or extention index
  hasNoExtension: BOOLEAN,    -- TRUE=> no ext.;prevfield=width
  height: [0..377B],          -- # scan lines to skip for char
  displacement: [0..377B]     -- displacement back to char bitmap
  ];

PaintChar: PROCEDURE
  [char: CHARACTER, bmState: POINTER TO FontDefs.BitmapState] =
  BEGIN OPEN BitBltDefs, bmState;
  bba: ARRAY [0..SIZE[BBTable]] OF UNSPECIFIED;
  bbt: BBptr = LOOPHOLE[BASE[bba] + LOOPHOLE[BASE[bba],CARDINAL] MOD 2];
  cw: FCDptr;
  fontdesc: FAptr = @font.FCDptrs;
  code: CARDINAL ← LOOPHOLE[char];
  bbt↑ ← [
    pad: 0,
    sourcealt: FALSE,
    destalt: FALSE,       sourcetype: block,
    function: paint,
    unused:,
    dbca: origin,
    dbmr: wordsPerLine,
    dlx: x,
    dty:,
    dw: 16,
    dh:,
    sbca:,
    sbmr: 1,
    slx: 0,
    sty: 0,
    gray0:, gray1:, gray2:, gray3:];
  DO
    cw ← LOOPHOLE[fontdesc[code]+LOOPHOLE[fontdesc,CARDINAL]+code];
    bbt.dty ← y + cw.height;
    bbt.dh ← cw.displacement;
    bbt.sbca ← cw - (bbt.dh ← cw.displacement);
    IF cw.hasNoExtension THEN
      BEGIN
      x ← x + (bbt.dw ← cw.widthORext);
      BITBLT[bbt];
      EXIT
      END
    ELSE
      BEGIN
      BITBLT[bbt];
      bbt.dlx ← x ← x + 16;
      END;
    code ← cw.widthORext;
    ENDLOOP;
  RETURN
  END;

WriteScoreString: PUBLIC PROCEDURE [rat: MazeDefs.RatId, s: STRING] =
  BEGIN
  i: CARDINAL;
  bmState: FontDefs.BitmapState ← [
    x: 0, y: rat*12, wordsPerLine: scoreDCB.width, origin: scoreDCB.bitmap];
  FOR i IN [0..s.length) DO PaintChar[s[i], @bmState]; ENDLOOP;
  RETURN
  END;
```

```
  ClearScoreLine: PUBLIC PROCEDURE [rat: MazeDefs.RatId] =
    BEGIN
    ChangeScoreLine[gray: 0B, dty: rat*12];
    RETURN
    END;

  InvertScoreLine: PUBLIC PROCEDURE [rat: MazeDefs.RatId] =
    BEGIN
    ChangeScoreLine[gray: 177777B, dty: rat*12];
    RETURN
    END;

  ChangeScoreLine: PROCEDURE [gray, dty: CARDINAL] =
    BEGIN OPEN BitBltDefs;
    bbTable: ARRAY [0..SIZE[BBTable]] OF UNSPECIFIED;
    bbptr: BBptr ← EVEN[@bbTable];
    bbptr↑ ← [
      pad: 0, sourcealt: FALSE, destalt: FALSE, sourcetype: gray,
      function: IF gray = 0 THEN replace ELSE invert, unused: 0,
      dbca: scoreDCB.bitmap, dbmr: 12,
      dlx: 0, dty: dty, dw: 12*16, dh: 12, sbca: NIL, sbmr: 0, slx: 0, sty: 0,
      gray0: gray, gray1: gray, gray2: gray, gray3: gray];
    BITBLT[bbptr];
    RETURN
    END;

END...
```

```
-- MazeDisplayDefs.mesa  Edited by Gobbel on June 22, 1979  3:50 PM

DIRECTORY
  AltoDisplay: FROM "altodisplay" USING [DCBHandle],
  BitBltDefs: FROM "bitbltdefs" USING [BBptr],
  InlineDefs: FROM "inlinedefs" USING [COPY],
  MazeDefs: FROM "mazedefs" USING [Direction, R2d2, RatId, XYpair, RatObject];

MazeDisplayDefs: DEFINITIONS IMPORTS InlineDefs =
  BEGIN OPEN MazeDefs;

  WindowWordsPerLine: CARDINAL = 26;
  WindowLines: CARDINAL = 400;

  pRat: TYPE = POINTER TO ARRAY RatId OF RatObject;
  DCBHandle: TYPE = AltoDisplay.DCBHandle;

  DCBRec: TYPE = RECORD [top, curr, hid, maze, bigMaze, score: DCBHandle];

  DoubleWord: TYPE = RECORD[w1, w2: WORD];

  MakeBigMaze: PROCEDURE;
  MakeSmallMaze: PROCEDURE;

  ClearPosition: PROCEDURE [ratId: MazeDefs.RatId, xClear, yClear: CARDINAL];
  ShowPosition: PROCEDURE [xloc, yloc: CARDINAL, invincible: BOOLEAN,
                                        tdir: Direction];
  SetMyRatId: PROCEDURE [ratId: RatId];
  ShowAllPositions: PROCEDURE [rats: pRat];
  DisplayOthersPosition: PROCEDURE [which, xloc, yloc: CARDINAL, tdir: Direction];
  ExitPlayer: PROCEDURE [ratId: RatId];
  AddNewPlayer: PROCEDURE [ratId: RatId, xloc, yloc: CARDINAL, tdir: Direction];
  SetDCBs: PROCEDURE [p: POINTER TO DCBRec, rats: POINTER];

  PlotLine: PROCEDURE [p: POINTER TO XYpair, two: BOOLEAN]
    RETURNS [POINTER TO XYpair];
  Switch: PROCEDURE RETURNS [DCBHandle];
  XORToken: PROCEDURE  [hisRatId: RatId, r2d2: POINTER TO R2d2];
  WriteScoreString: PROCEDURE [rat: RatId, s: STRING];
  ClearScoreLine: PROCEDURE [rat: RatId];
  InvertScoreLine: PROCEDURE [rat: RatId];
  InvertTop: PROCEDURE;
  InvertHidden: PROCEDURE = INLINE
    BEGIN
    hidden.background ← IF hidden.background=white THEN black ELSE white
    END;

  ratBB, lineBB, diagBB: BitBltDefs.BBptr;
  font: POINTER;
  hidden: DCBHandle;
  leftBBpattern, rightBBpattern: ARRAY [0..32) OF DoubleWord;

  MazeDisplay: PROGRAM;

  InitWindow: PROCEDURE [dcb: DCBHandle] = INLINE
    BEGIN OPEN InlineDefs;
    p: POINTER ← dcb.bitmap;
    p↑ ← 177777B;
    COPY[from: p, to: p+1, nwords: WindowWordsPerLine-2];
    (p+WindowWordsPerLine-1)↑ ← 0;
    COPY[from: p, to: p+WindowWordsPerLine*(WindowLines-1),
      nwords: WindowWordsPerLine];
    (p ← p + WindowWordsPerLine)↑ ← 100000B;
    (p+1)↑ ← 0B;
    COPY[from: p+1, to: p+2, nwords: WindowWordsPerLine-2];
    (p + WindowWordsPerLine-2)↑ ← 1B;
    COPY[from: p, to: p+WindowWordsPerLine,
      nwords: WindowWordsPerLine*(WindowLines-3)];
    END;

  END...
```

```
-- MazeInit.mesa edit by Bruce on June 25, 1979  7:35 PM

DIRECTORY
  AltoDisplay: FROM "altodisplay" USING [DCB,DCBHandle, DCBchainHead, DCBnil],
  BitBltDefs: FROM "bitbltdefs" USING [BBptr, BBTable],
  BitOps: FROM "BitOps" USING [Set, BD, Descriptor],
  DisplayDefs: FROM "displaydefs" USING [DisplayOff, StopCursor],
  FrameDefs: FROM "framedefs" USING [
    GlobalFrame, MakeCodeResident, SwapInCode, UnNew],
  ImageDefs: FROM "imagedefs" USING [MakeImage],
  InlineDefs: FROM "inlinedefs" USING [BITAND, BITOR, BITSHIFT],
  IODefs: FROM "iodefs" USING [
    CR, ReadID, Rubout, SP, WriteChar, WriteString, WriteLine],
  KeyDefs: FROM "keydefs" USING [Keys],
  MazeNewDefs: FROM "mazeNewdefs" USING [Password],
  MazeDefs: FROM "mazedefs" USING [AqRatNew, DispatchKeys, mazeSocket,
    MaxRats, NewPosition, NewScoreCard, RatId, RatName, RatNew, ratNew, Rats,
    RatStatus, ratStatus, ReadKeys, ReadRats, Tables, TheMaze, TokenVisible,
    ShowView, RatDoctor, ratSurvey],
  MazeDisplayDefs: FROM "mazedisplaydefs" USING [
    font, lineBB, MazeDisplay, ratBB, SetDCBs, ShowPosition,
    DCBRec, WindowWordsPerLine, leftBBpattern, rightBBpattern, diagBB, SetMyRatId],
  MazeWar: FROM "mazewar" USING [bvSendAllStatus, bvSendDead, bvSendGoing,
    bvSendKill, bvSendLocAll, bvSendOneStatus, bvSendQuery, bvSendAlive,
    dcb, keyStroke, Maze, myAddr, myRatId, ps, PupOutput, ratcb, score,
    table, tdir, wakeup, xloc, yloc, Wait, duke, randomVector, VectorSize],
  MiscDefs: FROM "miscdefs" USING [Zero],
  MMOps: FROM "mmops" USING [MMFont],
  ProcessDefs: FROM "processdefs" USING [CV, Detach, DIW,
    InterruptLevel, SetPriority],
  PupDefs: FROM "pupdefs" USING [AdjustBufferParms, GetFreePupBuffer,
    PupAddress, PupBuffer, GetPupAddress, PupNameTrouble,
    PupPackageMake, PupSocket, PupSocketDestroy, PupSocketMake,
    ReturnFreePupBuffer, SecondsToTocks, SetPupContentsWords,
    veryLongWait, PupAddressLookup, AppendPupAddress, PupRouterBroadcastThis,
    PupRouterSendThis, PupNameLookup],
  PupTypes: FROM "puptypes" USING [fillInNetID, allHosts, fillInHostID,
    fillInSocketID],
  StreamDefs: FROM "streamdefs" USING [
    GetDefaultDisplayStream, GetDefaultKey, StreamHandle],
  StringDefs: FROM "stringdefs" USING [StringBoundsFault],
  SystemDefs: FROM "systemdefs" USING [AllocateHeapNode, AllocateSegment];

MazeInit: PROGRAM
  IMPORTS FrameDefs, InlineDefs, MazeDefs, MazeDisplayDefs,
    MiscDefs, PupDefs, ProcessDefs, DisplayDefs,
    StreamDefs, StringDefs, SystemDefs, M: MazeWar, MMOps, IODefs, BitOps, ImageDefs
  EXPORTS MazeDefs
  SHARES MazeWar =
BEGIN OPEN MazeDefs, AltoDisplay;

m: POINTER TO FRAME[MazeWar] ← M;
rats:   POINTER;
ratBBTable, lineBBTable, diagBBTable: ARRAY [0..SIZE[BitBltDefs.BBTable]] OF UNSPECIFIED;
ratName: STRING ← [20];
dukeName: STRING ← [20];

GetMaze: PROCEDURE =
  BEGIN
  FrameDefs.SwapInCode[LOOPHOLE[TheMaze]];
  m.Maze ← FrameDefs.GlobalFrame[TheMaze].code.shortbase;
  END;

GetTable: PROCEDURE =
  BEGIN
  FrameDefs.SwapInCode[LOOPHOLE[Tables]];
  m.table ← FrameDefs.GlobalFrame[Tables].code.shortbase;
  END;

GetRats: PROCEDURE =
  BEGIN
  FrameDefs.SwapInCode[LOOPHOLE[Rats]];
  rats ← FrameDefs.GlobalFrame[Rats].code.shortbase;
  END;

CreateDCB: PROCEDURE [wordsPerLine, height, indenting: CARDINAL]
```

```
    RETURNS[dcb: DCBHandle] =
  BEGIN OPEN AltoDisplay;
  bitmap: POINTER ← IF wordsPerLine = 0 THEN NIL
    ELSE SystemDefs.AllocateSegment[wordsPerLine*height];
  dcb ← EVEN[SystemDefs.AllocateHeapNode[SIZE[DCB]+1]];
  dcb↑ ← [next: DCBnil, resolution: high, background: white,
    indenting: indenting, width: wordsPerLine, height: height/2,
    bitmap: bitmap];
  IF bitmap # NIL THEN MiscDefs.Zero[bitmap, wordsPerLine*height];
  END;

InitDisplay: PROCEDURE =
  BEGIN OPEN MazeDisplayDefs;
  dcbs: DCBRec;
  bbptr: BitBltDefs.BBptr ← EVEN[@ratBBTable];
  temp: DCBHandle;
  p: POINTER;
  i: INTEGER;
  desc: BitOps.Descriptor;
  curX: POINTER TO INTEGER = LOOPHOLE[426B];
  curX↑ ← -1;
  START MazeDisplay;
  FrameDefs.SwapInCode[LOOPHOLE[MMOps.MMFont]];
  font ← FrameDefs.GlobalFrame[MMOps.MMFont].code.shortbase;
  dcbs.top ← CreateDCB[wordsPerLine: 0, height: 50, indenting: 0];
  temp ← CreateDCB[wordsPerLine: 0, height: 20, indenting: 0];
  dcbs.curr ← CreateDCB[wordsPerLine: 26, height: 400, indenting: 6];
  dcbs.hid ← CreateDCB[wordsPerLine: 26, height: 400, indenting: 6];
--  dcbs.maze ← CreateDCB[wordsPerLine: 16, height: 128, indenting: 10];
  dcbs.maze ← NIL;
  dcbs.bigMaze ← CreateDCB[wordsPerLine: 32, height: 256, indenting: 2];
  dcbs.score ← CreateDCB[wordsPerLine: 12, height: 12*MaxRats, indenting: 11];
  SetDCBs[@dcbs, rats];
  dcbs.top.next ← dcbs.curr;
  dcbs.curr.next ← temp; dcbs.hid.next ← temp;
--  temp.next ← dcbs.maze;
--  dcbs.maze.next ← dcbs.score;
  temp.next ← dcbs.score;
--  InitMaze[dcbs.maze];
  BigInitMaze[dcbs.bigMaze];
  --this doesn't seem to work????
  MiscDefs.Zero[dcbs.score.bitmap, dcbs.score.width*dcbs.score.height*2];
  m.dcb ← dcbs.hid;
  AltoDisplay.DCBchainHead↑ ← dcbs.top;
  bbptr↑ ← [
    pad: 0, sourcealt: FALSE, destalt: FALSE, sourcetype: block,
    function: invert, unused: 0, dbca: NIL, dbmr: WindowWordsPerLine,
    dlx: 0, dty: 0, dw: 0, dh: 0, sbca: NIL, sbmr: 6*4, slx: 0, sty: 0,
    gray0:, gray1:, gray2:, gray3:];
  bbptr.sbca ← rats;
  MazeDisplayDefs.ratBB ← bbptr;
  bbptr ← EVEN[@lineBBTable];
  bbptr↑ ← [
    pad: 0, sourcealt: FALSE, destalt: FALSE, sourcetype: gray,
    function: paint, unused: 0, dbca: NIL, dbmr: WindowWordsPerLine,
    dlx: 0, dty: 0, dw: 0, dh: 0, sbca: NIL, sbmr: 0, slx: 0, sty: 0,
    gray0:177777B, gray1:177777B, gray2:177777B, gray3:177777B];
  MazeDisplayDefs.lineBB ← bbptr;
  bbptr ← EVEN[@diagBBTable];
  bbptr↑ ← [
    pad: 0, sourcealt: FALSE, destalt: FALSE, sourcetype: block,
    function: paint, unused: 0, dbca: NIL, dbmr: WindowWordsPerLine,
    dlx: 0, dty: 0, dw: 0, dh: 0, sbca: NIL, sbmr: 2, slx: 0, sty: 0,
    gray0: , gray1: , gray2: , gray3: ];
  MazeDisplayDefs.diagBB ← bbptr;
  p ← @MazeDisplayDefs.leftBBpattern[0];
  desc ← BitOps.BD[0];
  FOR i IN [0..32) DO
    BitOps.Set[1, p, desc];
    p ← p+2;
    desc ← LOOPHOLE[LOOPHOLE[desc, CARDINAL]+16];
    ENDLOOP;
  p ← @MazeDisplayDefs.rightBBpattern[0];
  desc ← BitOps.BD[31];
  FOR i IN [0..32) DO
    BitOps.Set[1, p, desc];
```

```
    p ← p+2;
    desc ← LOOPHOLE[LOOPHOLE[desc, CARDINAL]-16];
    ENDLOOP;
  END;

--InitMaze: PROCEDURE [mazeDCB: DCBHandle] =
--  BEGIN
--  i, line, j, index: CARDINAL;
--  maze: POINTER TO PACKED ARRAY OF [0..377B] ← mazeDCB.bitmap;
--  MiscDefs.Zero[maze, mazeDCB.width*mazeDCB.height*2];
--  FOR i IN [0..16) DO
--    line ← i*32*8;
--    FOR j IN [0..32) DO
--      IF m.Maze[i][j] THEN
--      BEGIN
--      index ← line + j;
--      THROUGH [0..8) DO maze[index] ← 377B; index ← index + 32; ENDLOOP;
--      END;
--      ENDLOOP;
--    ENDLOOP;
--  END;
--
BigInitMaze: PROCEDURE [mazeDCB: DCBHandle] =
  BEGIN
  i, line, j, index: CARDINAL;
  maze: POINTER TO PACKED ARRAY OF WORD ← mazeDCB.bitmap;
  MiscDefs.Zero[maze, mazeDCB.width*mazeDCB.height];
  FOR i IN [0..16) DO
    line ← i*32*16;
    FOR j IN [0..32) DO
      IF m.Maze[i][j] THEN
        BEGIN
        index ← line + j;
        THROUGH [0..16) DO maze[index] ← 177777B; index ← index + 32; ENDLOOP;
        END;
      ENDLOOP;
    ENDLOOP;
  END;

EVEN: PROCEDURE [v: UNSPECIFIED] RETURNS [UNSPECIFIED] =
  BEGIN RETURN[v+InlineDefs.BITAND[v,1]] END;

InitKeyboard: PROCEDURE RETURNS[ph: PROCESS] =
  BEGIN OPEN ProcessDefs;
  KeyboardLevel: InterruptLevel = 7;
  KeyboardBit: WORD = InlineDefs.BITSHIFT[1,KeyboardLevel];

  SetPriority[6];  -- Increase my priority so that the fork
  ph ← FORK ReadKeys;  -- goes off at the new priority
  SetPriority[1];  -- reset my original priority
  CV[KeyboardLevel] ← @m.wakeup;  -- set the cv for the naked notify
  -- enable interrupts every display vertical trace
  DIWT ← InlineDefs.BITOR[DIWT,KeyboardBit];
  RETURN[ph];
  END;

InitRandom: PROCEDURE =
  BEGIN
  p: POINTER TO CARDINAL = LOOPHOLE[430B];
  i: CARDINAL;
  t: CARDINAL = pt;
  FOR i IN [0..m.VectorSize) DO
    m.randomVector[i] ← t + m.randomVector[i] ENDLOOP;
  END;

GetNames: PROCEDURE =
  BEGIN OPEN IODefs;
  keys: StreamDefs.StreamHandle ← StreamDefs.GetDefaultKey[];
  WriteLine["MazeWar modified version by Brad Myers"L];
  WriteLine["Holding 'E' down shows big maze"L];
  WriteString["Your Name: "L];
  ReadID[ratName ! StringDefs.StringBoundsFault => CONTINUE];
  WriteChar[CR];
  WriteString["Duke Host (CR for any game): "L];
  ReadID[dukeName ! StringDefs.StringBoundsFault => CONTINUE];
  WriteChar[CR];
```

```
  keys.dostroy[keys];
  m.Wait[10]; -- make sure keyboard process dies
  DisplayDefs.StopCursor[];
  DisplayDefs.DisplayOff[white];
  FrameDefs.UnNew[FrameDefs.GlobalFrame[WriteString]];
  FrameDefs.UnNew[FrameDefs.GlobalFrame[StreamDefs.GetDefaultKey]];
  FrameDefs.UnNew[FrameDefs.GlobalFrame[StreamDefs.GetDefaultDisplayStream]];
  RETURN
  END;

CheckRatName: PROCEDURE =
  BEGIN OPEN PupDefs;
  i: CARDINAL;
  addr: PupAddress;
  PupNameLookup[@addr,"ME"L];
  addr.socket ← mazeSocket;
  m.myAddr ← addr;
  IF ratName.length # 0 THEN RETURN;
  PupAddressLookup[addr, ratName ! PupNameTrouble =>
    BEGIN AppendPupAddress[ratName, @addr]; CONTINUE END];
  FOR i DECREASING IN [0..ratName.length] DO
    IF ratName[i] # '+ AND ratName[i] # '# THEN LOOP;
    ratName.length ← i;
    EXIT;
    ENDLOOP;
  RETURN;
  END;

RatNameFromString: PROCEDURE [name: STRING, pt: POINTER TO RatName] =
  BEGIN
  i: CARDINAL;
  FOR i IN [0..name.length) DO pt[i] ← name[i]; ENDLOOP;
  FOR i IN [name.length..20) DO pt[i] ← IODefs.SP; ENDLOOP;
  END;

PupInit: PROCEDURE =
  BEGIN OPEN PupDefs;
  ratId: RatId;
  AdjustBufferParms[10, 266];       -- (20) buffers, 266 bytes each now 10
  PupPackageMake[];
  CheckRatName[];
  IF NOT JoinGame[] THEN StartGame[];
  FOR ratId IN RatId DO   -- check for visible rats for the scorecard
    TokenVisible[ratId];
    ENDLOOP;
  NewScoreCard[];             -- Init the score window
  m.ps ← PupSocketMake[mazeSocket, m.myAddr, veryLongWait];
  ProcessDefs.Detach[FORK ReadRats[]];  -- Fork the read pup process
  m.PupOutput ← FALSE;
  m.bvSendLocAll ← FALSE;
  m.bvSendOneStatus ← FALSE;
  m.bvSendAllStatus ← FALSE;
  m.bvSendKill ← FALSE;
  m.bvSendDead ← FALSE;
  m.bvSendGoing ← FALSE;
  m.bvSendQuery ← FALSE;
  m.bvSendAlive ← FALSE;
  m.keyStroke ← TRUE;  -- Can now display screen for first time
  END;

SetUpSurvey: PROCEDURE [b: PupDefs.PupBuffer] =
  BEGIN
  ratsurvey: RatNew;
  b.pupType ← ratSurvey;
  b.source ← m.myAddr;
  PupDefs.SetPupContentsWords[b, SIZE[AqRatNew]];
  ratsurvey ← LOOPHOLE[@b.pupBody];
  ratsurvey↑ ← [MazeNewDefs.Password,,,,,];
  END;

NullAddress: PupDefs.PupAddress = [net: PupTypes.fillInNetID,
  host: PupTypes.fillInHostID, socket: PupTypes.fillInSocketID];

FindDuke: PROCEDURE RETURNS [addr: PupDefs.PupAddress] =
  BEGIN OPEN PupDefs;
  rmtAddr: PupAddress ←
```

```
    [net: PupTypes.fillInNetID, host: PupTypes.allHosts, socket: mazeSocket];
  ps: PupSocket ← PupSocketMake[mazeSocket, rmtAddr, SecondsToTocks[5]];
  b: PupBuffer ← GetFreePupBuffer[];
  maxAnswers: CARDINAL = 10;
  answers: ARRAY [0..maxAnswers) OF PupBuffer;
  i,j,cnt: CARDINAL ← 0;
  SetUpSurvey[b];
  b.dest ←
    [net: PupTypes.fillInNetID, host: PupTypes.allHosts, socket: mazeSocket];
  PupRouterBroadcastThis[b];
  UNTIL (b ← ps.get[]) = NIL DO
    IF b.pupType # ratStatus OR cnt = maxAnswers THEN
      BEGIN ReturnFreePupBuffer[b]; LOOP; END;
    IF ~Duke[b] THEN
      BEGIN
      rs: RatStatus ← LOOPHOLE[@b.pupBody];
      SetUpSurvey[b];
      b.dest ← rs.rats[rs.dukeRat].addr;
      [] ← PupRouterSendThis[b];
      LOOP;
      END;
    answers[cnt] ← b;
    IF FreeSlot[b] THEN cnt ← cnt + 1 ELSE ReturnFreePupBuffer[b];
    ENDLOOP;
  PupSocketDestroy[ps];
  FOR i IN [0..cnt) DO
    IF ~Duke[answers[i]] THEN BEGIN ReturnFreePupBuffer[answers[i]]; LOOP END;
    addr ← answers[i].source;
    FOR j IN [i..cnt) DO ReturnFreePupBuffer[answers[j]] ENDLOOP;
    RETURN
    ENDLOOP;
  RETURN[NullAddress]
  END;

Duke: PROCEDURE [b: PupDefs.PupBuffer] RETURNS [BOOLEAN] =
  BEGIN
  test: RatStatus ← LOOPHOLE[@b.pupBody];
  RETURN[test.rats[test.dukeRat].addr = b.source]
  END;

FreeSlot: PROCEDURE [b: PupDefs.PupBuffer] RETURNS [BOOLEAN] =
  BEGIN
  test: RatStatus ← LOOPHOLE[@b.pupBody];
  id: RatId;
  FOR id IN RatId DO
    IF ~test.rats[id].playing THEN RETURN[TRUE];
    ENDLOOP;
  RETURN[FALSE];
  END;

Join: PROCEDURE [addr: PupDefs.PupAddress] RETURNS [okToPlay: BOOLEAN] =
  BEGIN OPEN PupDefs;
  ps: PupSocket ← PupSocketMake[mazeSocket, addr, SecondsToTocks[5]];
  b: PupBuffer ← GetFreePupBuffer[];
  ratnew: RatNew;
  id: RatId;
  status: RatStatus;
  okToPlay ← FALSE;
  b.pupType ← ratNew;
  b.dest ← addr;
  b.source ← m.myAddr;
  SetPupContentsWords[b, SIZE[AqRatNew]];
  ratnew ← LOOPHOLE[@b.pupBody];
  ratnew↑ ← [MazeNewDefs.Password, m.xloc, m.yloc, m.tdir, m.myAddr, ];
  RatNameFromString[ratName, @ratnew.name];
  [] ← PupRouterSendThis[b];
  b ← ps.get[];
  IF b = NIL OR b.pupType # ratStatus THEN
    BEGIN
    IF b # NIL THEN ReturnFreePupBuffer[b];
    PupSocketDestroy[ps];
    RETURN
    END;
  status ← LOOPHOLE[@b.pupBody];
```

```
      BEGIN m.myRatId ← id; MazeDisplayDefs.SetMyRatId[id]; EXIT END;
    REPEAT FINISHED => RETURN
    ENDLOOP;
  m.ratcb ← statust;
  okToPlay ← TRUE;
  ReturnFreePupBuffer[b];
  PupSocketDestroy[ps];
  END;

JoinGame: PROCEDURE RETURNS [okToPlay: BOOLEAN] =
  BEGIN OPEN PupDefs;
  rmtAddr: PupAddress;
  duke: BOOLEAN ← dukeName.length # 0;
  IF duke THEN GetPupAddress[@rmtAddr, dukeName !
    PupNameTrouble => BEGIN duke ← FALSE; CONTINUE END];
  rmtAddr.socket ← mazeSocket;
  IF ~duke THEN rmtAddr ← FindDuke[];
  IF rmtAddr = NullAddress THEN RETURN [FALSE];
  RETURN[Join[rmtAddr]]
  END;

StartGame: PROCEDURE =
  BEGIN
  i: CARDINAL;
  m.myRatId ← 0;          -- Give me first entry in the table
  MazeDisplayDefs.SetMyRatId [0];
  m.ratcb.dukeRat ← m.myRatId;      -- Set myself up as the duke
  m.ratcb.rats[m.myRatId] ←
    [TRUE, m.xloc, m.yloc, m.tdir, m.score, m.myAddr, ];
  RatNameFromString[ratName, @m.ratcb.rats[m.myRatId].name];
  FOR i IN [1..MaxRats) DO
    m.ratcb.rats[i].playing ← FALSE;
    ENDLOOP;
  m.duke ← TRUE;
  END;

RV: ARRAY [0..m.VectorSize) OF WORD = [
  031575B, 055465B, 147160B, 176745B, 173126B, 117426B, 033612B, 130620B,
  054013B, 167672B, 070252B, 033100B, 015700B, 113523B, 170465B, 024344B,
  175535B, 137325B, 126211B, 010207B, 173547B, 016071B, 056622B, 014433B,
  113225B, 047563B, 103025B, 110174B, 000125B, 173304B, 076700B, 104042B,
  135030B, 126234B, 175154B, 140123B, 167542B, 000405B, 035464B, 166537B,
  050260B, 167655B, 123615B, 175164B, 172206B, 140365B, 074606B, 075656B,
  176163B, 030027B, 022102B, 040051B, 154630B, 017144B, 073372B];

FrameDefs.MakeCodeResident[LOOPHOLE[TheMaze]];

GetMaze[];
GetTable[];  -- Read vector table from file (30 views, 12 lines per view)
GetRats[];  -- Read rat bitmaps, init ratpack

IF KeyDefs.Keys.Spare2 = down THEN ImageDefs.MakeImage["MazeWar.image"L];

m.randomVector ← RV;
InitRandom[];
GetNames[ ! IODefs.Rubout => RETRY];
InitDisplay[];
NewPosition[];  -- Get random positon in the maze
MazeDisplayDefs.ShowPosition[m.xloc, m.yloc, FALSE, m.tdir];
MazeDefs.ShowView[m.xloc, m.yloc, m.tdir];
PupInit[];
ProcessDefs.Detach[InitKeyboard[]];
ProcessDefs.Detach[FORK MazeDefs.DispatchKeys[]];
ProcessDefs.Detach[FORK MazeDefs.RatDoctor[]];

--PupInit[];

END...
```

```
--MAZE WATCHER by: Brad A. Myers  Last change: July 26, 1979  9:04 PM

DIRECTORY
  MazeDefs: FROM "mazedefs",
  PupTypes: FROM "puptypes" USING [PupType];

MazeNewDefs: DEFINITIONS =
BEGIN OPEN MazeDefs;

Password: CARDINAL = 176543B;

StayAlive: PROCEDURE;
RatWatching: PROCEDURE [ hisRatId: RatId];

ratTalk: PupTypes.PupType;
END.
```

```
--HACKED VERSION by Brad A. Myers!! October 23, 1979 10:35 PM
-- MazeWar.mesa edit by Bruce on June 26, 1979  7:09 PM

DIRECTORY
  AltoDisplay: FROM "altodisplay" USING [DCBchainHead, DCBnil],
  FrameDefs: FROM "framedefs" USING [UnNew],
  ImageDefs: FROM "imagedefs" USING [StopMosa],
  InlineDefs: FROM "inlinedefs",
  KeyDefs: FROM "keydefs" USING [KeyBits, Keys],
  MazeDefs: FROM "mazedefs",
  MazeNewDefs: FROM "mazeNewdefs" USING [Password],
  MazeDisplayDefs: FROM "mazedisplaydefs" USING [
    ClearScoreLine, DCBHandle, InitWindow, InvertScoreLine,
    PlotLine, ShowPosition, InvertTop, DisplayOthersPosition, ExitPlayer, AddNewPlayer,
    ShowAllPositions, MakeBigMaze, MakeSmallMaze,
    Switch, WriteScoreString,
    XORToken, InvertHidden],
  ProcessDefs: FROM "processdefs" USING [
    Detach, InitializeCondition, MsecToTicks, SetPriority, Yield],
  PupDefs: FROM "pupdefs" USING [GetFreePupBuffer, PupAddress, PupBuffer,
    PupRouterSendThis, PupSocket, ReturnFreePupBuffer, SetPupContentsWords],
  StringDefs: FROM "stringdefs" USING [AppendDecimal];

MazeWar: MONITOR
  IMPORTS ImageDefs, InlineDefs, MazeDefs, MazeDisplayDefs,
    PupDefs, ProcessDefs, StringDefs, FrameDefs
  EXPORTS MazeDefs =
PUBLIC BEGIN OPEN MazeDefs, MazeDisplayDefs, PupDefs;

table: POINTER TO ARRAY [0..360) OF XYpair;  -- 30 frames, 12 XYpairs each

xloc: CARDINAL;
yloc: CARDINAL; -- Where am I?
tdir: Direction; -- Which way am I looking?
dcb: DCBHandle;
Maze: MazeType;

-- Delta Arrays contain the deltas to map the current x,y,direction into the neighboring cubes
**tes
L1Delta: ARRAY Direction OF XY ← [ [-1,0], [1,0], [0,1], [0,-1] ];
L2Delta: ARRAY Direction OF XY ← [ [-1,1], [1,-1], [1,1], [-1,-1] ];
C2Delta: ARRAY Direction OF XY ← [ [0,1], [0,-1], [1,0], [-1,0] ];
R1Delta: ARRAY Direction OF XY ← [ [1,0], [-1,0], [0,-1], [0,1] ];
R2Delta: ARRAY Direction OF XY ← [ [1,1], [-1,-1], [1,-1], [-1,1] ];
Edge1, Edge2, Edge3, Edge4, Edge5, Edge6, Edge7: BOOLEAN;
edge3Lines, edge7Lines: ARRAY [0..1] OF XYpair;
prevEdge3, prevEdge7: BOOLEAN;

wakeup: CONDITION;                -- Posted by the DIW u-code process
keyStroke: BOOLEAN ← FALSE;       -- new keys to process

peeking: BOOLEAN ← FALSE;         -- peeking left or right in process
xPeek, yPeek: CARDINAL;           -- x,y,dir for peek display
dirPeek: Direction;

ctBreaths: CARDINAL ← 0;          -- Number of shots current fired at me (# breaths being held)

-- Global Data For Pup Stuff

myAddr: PupAddress;     -- My pup network/host address (hmm)
myRatId: RatId;       -- My index into the ratcb
killerRatId: RatId;       -- My killer's index into the ratcb
ratcb: RatCb;       -- my copy of the rest of the world!
ps: PupSocket;     -- the main socket
PupOutput: BOOLEAN;     -- Set to wake up pup writer
bvSendLocAll: BOOLEAN;       -- Set if all are to get my new location
bvSendOneStatus: BOOLEAN;     -- Set if to send my ratcb to new member
bvSendAllStatus: BOOLEAN;     -- Set if to send my ratcb to all players
bvSendDead: BOOLEAN;       -- Send I'm dead msg to my killer
bvSendKill: BOOLEAN;       -- Send off a fire msg
bvSendGoing: BOOLEAN;       -- Send off a I'm going msg
bvSendQuery: BOOLEAN;       -- Send off one or more query msgs
bvSendAlive: BOOLEAN;       -- Send off response to query msg
newAddr: PupAddress;     -- Address of the new dude for SendOneStatus
score: Score ← 0;       -- My total score so far
duke: BOOLEAN ← FALSE;
```

```
ratHealth: RatHealth;  -- For the rat doctor

invincible: BOOLEAN ← FALSE;            --Can I be killed?
ratId: RatId;           --for initialization at end (BUG FIX)
-- Global Data for Tokens

r2d2: R2d2;                     -- The array

-- Array [my direction] by [his direction]
-- 0=left arrow, 1=right arrow, 2=up arrow, 3=down arrow
relativeTokens: RelativeTokens ← [
        [ rear, front, right, left],            -- Me north, him [n, s, e, w]
        [ front, rear, left, right],            -- Me south, him [n, s, e, w]
        [ left, right, rear, front],    -- Me east, him [n, s, e, w]
        [ right, left, front, rear] ];  -- Me west, him [n, s, e, w]

-- Global Data For Random number generator

VectorSize: CARDINAL = 55;
i1: CARDINAL ← 0;
i2: CARDINAL ← 24;

randomVector: ARRAY [0..VectorSize) OF WORD;

Random: PROCEDURE [limit: CARDINAL] RETURNS [ret: WORD] =
  BEGIN
  ret←randomVector[i1] ← randomVector[i1] + randomVector[i2];
  IF (i1 ← i1+1) >= VectorSize THEN i1 ← 0;
  IF (i2 ← i2+1) >= VectorSize THEN i2 ← 0;
  RETURN[ret MOD limit];
  END;

-- Calculate which lines to display and display them!
Hidden: PROCEDURE [x,y: CARDINAL, dir: Direction, p: POINTER TO XYpair]
  RETURNS [POINTER TO XYpair] = INLINE
  BEGIN

  -- Local variables and tables
  L1x, L1y, L2x, L2y: CARDINAL;
  R1x, R1y, R2x, R2y: CARDINAL;
  C2x, C2y:           CARDINAL;

  -- 1st calculate the coordinates of the neighboring cubes

  L1x ← x + L1Delta[dir].xcor;      -- Find left cube
  L1y ← y + L1Delta[dir].ycor;
  L2x ← x + L2Delta[dir].xcor;      -- Find left forward cube
  L2y ← y + L2Delta[dir].ycor;
  R1x ← x + R1Delta[dir].xcor;      -- Find right cube
  R1y ← y + R1Delta[dir].ycor;
  R2x ← x + R2Delta[dir].xcor;      -- Find right forward cube
  R2y ← y + R2Delta[dir].ycor;
  C2x ← x + C2Delta[dir].xcor;      -- Find forward cube
  C2y ← y + C2Delta[dir].ycor;

  -- Next calculate which of the 7 possible cube edges are visible

  Edge2 ← Maze[C2x][C2y];                       -- C2
  Edge3 ← Maze[L1x][L1y];                       -- L1
  Edge4 ← NOT Edge3;                            -- \ L1

  Edge7 ← Maze[R1x][R1y];                       -- R1
  Edge6 ← NOT Edge7;                            -- \ R1

  Edge1 ← Edge3 AND (Edge2 OR NOT(Maze[L2x][L2y]))
    OR (NOT(Edge2) AND Edge4);
  Edge5 ← Edge7 AND (Edge2 OR NOT(Maze[R2x][R2y]))
    OR (NOT(Edge2) AND Edge6);

  -- Should be matching the following:
  -- X1 = L1 (C2 + \L2) + \C2 \L1
  -- X2 = C2
  -- X3 = L1
  -- X4 = \L1
```

```
  -- X6 = \R1
  -- X7 = R1

  IF Edge1 THEN p ← PlotLine[p, FALSE] ELSE p ← p + szXYpair;
  IF Edge2 THEN p ← PlotLine[p, TRUE] ELSE p ← p + szTwoXYpair;
  IF Edge3 THEN
    BEGIN
    IF prevEdge3 THEN
      BEGIN
      edge3Lines[0].p2 ← p.p2;
      p ← p+szXYpair;
      edge3Lines[1].p2 ← p.p2;
      END
    ELSE
      BEGIN
      edge3Lines[0] ← p↑;
      p ← p+szXYpair;
      edge3Lines[1] ← p↑;
      prevEdge3 ← TRUE;
      END;
    p ← p+szXYpair;
    END
  ELSE
    BEGIN
    IF prevEdge3 THEN
      BEGIN
      [] ← PlotLine[@edge3Lines[0], TRUE];
      prevEdge3 ← FALSE;
      END;
    p ← p + szTwoXYpair;
    END;
  IF Edge4 THEN p ← PlotLine[p, TRUE] ELSE p ← p + szTwoXYpair;
  IF Edge5 THEN p ← PlotLine[p, FALSE] ELSE p ← p + szXYpair;
  IF Edge6 THEN p ← PlotLine[p, TRUE] ELSE p ← p + szTwoXYpair;
  IF Edge7 THEN
    BEGIN
    IF prevEdge7 THEN
      BEGIN
      edge7Lines[0].p1 ← p.p1;
      p ← p+szXYpair;
      edge7Lines[1].p1 ← p.p1;
      END
    ELSE
      BEGIN
      edge7Lines[0] ← p↑;
      p ← p+szXYpair;
      edge7Lines[1] ← p↑;
      prevEdge7 ← TRUE;
      END;
    p ← p+szXYpair;
    END
  ELSE
    BEGIN
    IF prevEdge7 THEN
      BEGIN
      [] ← PlotLine[@edge7Lines[0], TRUE];
      prevEdge7 ← FALSE;
      END;
    p ← p + szTwoXYpair;
    END;
  RETURN[p];
  END;  -- End of Hidden Procedure

ShowView: PROCEDURE [x,y: CARDINAL, dir: Direction] =
  BEGIN
  tx: CARDINAL ← x;  -- Temp position as the view
  ty: CARDINAL ← y;  --       procedes down the hall
  tp: POINTER ← table;  -- pointer into the table of vectors
  ratId: RatId;
  ratLook: RatLook;
  bvOldVisible: BOOLEAN;
  MazeDisplayDefs.InitWindow[dcb];       -- Clear work bitmap
  prevEdge3 ← prevEdge7 ← FALSE; -- Init flags for plotter smarts
  UNTIL Maze[tx][ty] DO -- Keep going till bump into wall
    IF keyStroke THEN EXIT;  -- Exit if type-ahead
    tp ← Hidden[tx,ty,dir,tp];
```

```
    SELECT dir FROM
      NORTH => ty ← ty + 1;
      SOUTH => ty ← ty - 1;
      EAST  => tx ← tx + 1;
      WEST  => tx ← tx - 1;
      ENDCASE;
    REPEAT
      FINISHED =>
        BEGIN
        IF prevEdge3 THEN [] ← PlotLine[@edge3Lines[0], TRUE];
        IF prevEdge7 THEN [] ← PlotLine[@edge7Lines[0], TRUE];
        END;
    ENDLOOP;

  dcb ← MazeDisplayDefs.Switch[]; -- display entire screen at once

  -- This is sort of poor, it should really be able to put the tokens in the invisible map
  -- so that the display all comes on at once, but if I do that now, I have problems
  -- with keeping track of what tokens are in the visible/invisible display
  FOR ratId IN RatId DO          -- Scan for visible tokens
    IF ratId = myRatId THEN LOOP;
    ratLook ← @r2d2[ratId];
    bvOldVisible ← ratLook.visible;
    TokenVisible[ratId];         -- Set the token info in the r2d2 array
    IF ratLook.visible THEN XORToken[ratId, @r2d2];
    IF ratLook.visible # bvOldVisible THEN UpdateScoreCard[ratId];
    ENDLOOP;
  END; -- End of View Procedure


SendLocToAll: PROCEDURE =
  BEGIN
  b: PupBuffer;                  -- My pup buffer
  ratloc: RatLocation;                   -- Pointer into the ratlocation msg
  i: CARDINAL;
  ratInfo: RatInfo ← @ratcb.rats[myRatId];
  ratInfo.xLoc ← xloc;                   -- Update my table too
  ratInfo.yLoc ← yloc;
  ratInfo.dir ← tdir;
  ratInfo.score ← score;
  bvSendLocAll ← FALSE;.
  -- Following code sends from my ratcb in case the data changes while
  -- I'm waiting for one of the pups to be sent.
  FOR i IN [0..MaxRats) DO
    IF i # myRatId AND ratcb.rats[i].playing THEN
      BEGIN
      b ← GetFreePupBuffer[];                    -- Get a new buffer
      b.pupType ← ratLocation;
      b.dest ← ratcb.rats[i].addr;
      b.source ← myAddr;
      SetPupContentsWords[b, SIZE[AqRatLocation]];
      ratloc ← LOOPHOLE[@b.pupBody];     -- Get msg body set
      ratloc.ratId ← myRatId;
      ratloc.xLoc ← ratInfo.xLoc;
      ratloc.yLoc ← ratInfo.yLoc;
      ratloc.dir ← ratInfo.dir ← tdir;
      ratloc.score ← ratInfo.score ← score;
      [] ← PupRouterSendThis[b];
      END;
    ENDLOOP;
  RETURN;
  END;

SendAllStatus: PROCEDURE =
  BEGIN
  i: CARDINAL;
  bvSendAllStatus ← FALSE;
  FOR i IN [0..MaxRats) DO
    IF i # myRatId AND ratcb.rats[i].playing THEN
      SendStatus[ratcb.rats[i].addr];
    ENDLOOP;
  RETURN;
  END;

SendStatus: PROCEDURE [addr: PupAddress] =
```

```
  b: PupBuffer;                          -- My pup buffer
  status: RatStatus;                     -- Pointer into the status msg
  bvSendOneStatus ← FALSE;
  b ← GetFreePupBuffer[];                -- Get a new buffer
  b.pupType ← ratStatus;
  b.dest ← addr;
  b.source ← myAddr;
  SetPupContentsWords[b, SIZE[RatCb]];
  status ← LOOPHOLE[@b.pupBody];         -- Get msg body set
  status↑ ← ratcb;                       -- set the data in the msg
  [] ← PupRouterSendThis[b];
  RETURN;
  END;

SendKill: PROCEDURE =
  BEGIN
  b: PupBuffer;                  -- My pup buffer
  ratkill: RatKill;              -- Pointer into the kill msg
  ixRatId: RatId;                -- Index into the r2d2 array looking for opponents
  bvSendKill ← FALSE;
  FOR ixRatId IN RatId DO
    IF ixRatId = myRatId THEN LOOP;      -- See about getting rid of this
    IF r2d2[ixRatId].visible THEN
      BEGIN
      b ← GetFreePupBuffer[];            -- Get a new buffer
      b.pupType ← ratKill;
      b.dest ← ratcb.rats[ixRatId].addr;
      b.source ← myAddr;
      SetPupContentsWords[b, SIZE[AqRatKill]];
      ratkill ← LOOPHOLE[@b.pupBody];    -- Get msg body set
      ratkill.ratId ← myRatId;
      ratkill.xLoc ← xloc;
      ratkill.yLoc ← yloc;
      ratkill.dir ← tdir;
      [] ← PupRouterSendThis[b];
      END;
    ENDLOOP;
  RETURN;
  END;

SendDead: PROCEDURE =
  BEGIN
  b: PupBuffer;                  -- My pup buffer
  ratdead: RatDead;                      -- Pointer into the dead msg
  bvSendDead ← FALSE;
  b ← GetFreePupBuffer[];                        -- Get a new buffer
  b.pupType ← ratDead;
  b.dest ← ratcb.rats[killerRatId].addr;
  b.source ← myAddr;
  SetPupContentsWords[b, SIZE[AqRatDead]];
  ratdead ← LOOPHOLE[@b.pupBody];                -- Get msg body set
  ratdead.ratId ← myRatId;
  ratdead.killedBy ← killerRatId;
  [] ← PupRouterSendThis[b];
  RETURN;
  END;

SendGoing: PROCEDURE =
  BEGIN
  b: PupBuffer;                  -- My pup buffer
  ratgone: RatGone;                      -- Pointer into the gone msg
  bvSendGoing ← FALSE;
  b ← GetFreePupBuffer[];                        -- Get a new buffer
  b.pupType ← ratGoing;
  b.dest ← ratcb.rats[ratcb.dukeRat].addr;
  b.source ← myAddr;
  SetPupContentsWords[b, SIZE[AqRatGone]];
  ratgone ← LOOPHOLE[@b.pupBody];                -- Get msg body set
  ratgone.ratId ← myRatId;
  [] ← PupRouterSendThis[b];
  RETURN;
  END;

SendQuery: PROCEDURE =
```

```
  ratquery: RatQuery;                    -- Pointer into the query msg
  ratid: RatId;
  bvSendQuery ← FALSE;
  FOR ratid IN RatId DO
    IF ratHealth[ratid].pupSend THEN
      BEGIN
      ratHealth[ratid].pupSend ← FALSE;
      b ← GetFreePupBuffer[];                    -- Get a new buffer
      b.pupType ← ratQuery;
      b.dest ← ratcb.rats[ratid].addr;
      b.source ← myAddr;
      SetPupContentsWords[b, SIZE[AqRatQuery]];
      ratquery ← LOOPHOLE[@b.pupBody];           -- Get msg body set
      ratquery.ratId ← myRatId;
      [] ← PupRouterSendThis[b];
      END;
    ENDLOOP;
  RETURN;
  END;

SendAlive: PROCEDURE =
  BEGIN
  b: PupBuffer;                  -- My pup buffer
  ratid: RatId;
  ratalive: RatAlive;                    -- Pointer into the alive msg
  FOR ratid IN RatId DO
    IF ratid=myRatId OR ~ ratcb.rats[ratid].playing THEN LOOP;
    b ← GetFreePupBuffer[];                      -- Get a new buffer
    b.pupType ← ratAlive;
    b.dest ← ratcb.rats[ratid].addr;
    b.source ← myAddr;
    SetPupContentsWords[b, SIZE[AqRatAlive]];
    ratalive ← LOOPHOLE[@b.pupBody];             -- Get msg body set
    ratalive.ratId ← myRatId;
    [] ← PupRouterSendThis[b];
    ENDLOOP;
  bvSendAlive ← FALSE;
  RETURN;
  END;

ReadRats: PROCEDURE =
  BEGIN
  b: PupBuffer;                  -- My pup buffer
  ratLook: RatLook;
  ratInfo: RatInfo;
  DO
    b ← ps.get[];                                -- Wait for a pup buffer
    IF b = NIL THEN LOOP;        -- Ignore nils (shouldn't get any really)
    SELECT b.pupType FROM        -- See what we have here
      ratLocation =>
        BEGIN
        ratloc: RatLocation ← LOOPHOLE[@b.pupBody];
        bvOldVisible: BOOLEAN;
        ratLook ← @r2d2[ratloc.ratId];
        IF (bvOldVisible ← ratlook.visible) THEN
          XORToken[ratloc.ratId, @r2d2];
        ratInfo ← @ratcb.rats[ratloc.ratId];
        ratInfo.xLoc ← ratloc.xLoc;
        ratInfo.yLoc ← ratloc.yLoc;
        ratInfo.dir ← ratloc.dir;
        DisplayOthersPosition[ratloc.ratId, ratloc.xLoc, ratloc.yLoc,
                              ratloc.dir];
        TokenVisible[ratloc.ratId];      -- Set the token info in the r2d2 array
        IF ratLook.visible THEN XORToken[ratloc.ratId, @r2d2];
        IF bvOldVisible # ratLook.visible OR
          ratInfo.score # ratloc.score THEN      -- If his score has changed
          BEGIN
          ratInfo.score ← ratloc.score;
          UpdateScoreCard[ratloc.ratId];
          END;
        ratHealth[ratloc.ratId].pupRcvd ← TRUE;
        END;
      ratStatus =>
        BEGIN
        status: RatStatus ← LOOPHOLE[@b.pupBody];
        ratid: RatId;
```

```
      IF status.rats[myRatId].addr # myAddr THEN GOTO notMyPackot;
      --Have a new table, turn off any visible opponent
      FOR ratid IN RatId DO
        IF r2d2[ratid].visible THEN XORToken[ratid, @r2d2];
        ENDLOOP;
      ratcb ← status↑;
      IF ratcb.dukeRat = myRatId THEN duke←TRUE;
      FOR ratid IN RatId DO
        TokenVisible[ratid];
        IF r2d2[ratid].visible THEN
        XORToken[ratid, @r2d2];
        ENDLOOP;
      NewScoreCard[];
      ratInfo ← @ratcb.rats[myRatId];
      IF ratInfo.xLoc # xloc OR ratInfo.yLoc # yloc
        OR ratInfo.dir # tdir THEN
        BEGIN
        bvSendLocAll ← TRUE;
        PupOutput ← TRUE;
        END;
      EXITS
        notMyPacket => NULL;
      END;
    ratNew =>
      BEGIN
      ratnew: RatNew ← LOOPHOLE[@b.pupBody];
      IF ratnew.pass = MazeNewDefs.Password THEN
        BEGIN
        IF duke THEN
          BEGIN
          AllocateNewRat[ratnew];
          bvSendAllStatus ← TRUE;
          END
        ELSE
          BEGIN
          newAddr ← ratnew.addr;
          bvSendOneStatus ← TRUE;
          END;
        PupOutput ← TRUE;
        END;
      END;
    ratSurvey =>
      BEGIN
      ratnew: RatNew ← LOOPHOLE[@b.pupBody];
      IF ratnew.pass = MazeNewDefs.Password THEN
        BEGIN
        newAddr ← b.source;
        bvSendOneStatus ← TRUE;
        PupOutput ← TRUE;
        END;
      END;
    ratKill =>  IF ~Invincible THEN
      IF ctBreaths < 5 THEN           -- Only if within process limit
      BEGIN
      p: PROCESS;
      ratkill: RatKill ← LOOPHOLE[@b.pupBody];
      ctBreaths ← ctBreaths + 1;
      p ← FORK HoldBreath[tx: ratkill.xLoc, ty: ratkill.yLoc,
        td: ratkill.dir, ratId: ratkill.ratId];
      ProcessDefs.Detach[p];
      END;
    ratDead =>
      BEGIN
      ratdead: RatDead ← LOOPHOLE[@b.pupBody];
      IF ratdead.killedBy = myRatId THEN
        BEGIN
        InvertTop[];  --got him
        score ← score + 10;           -- 10 points for a kill
        ratcb.rats[myRatId].score ← score;
        UpdateScoreCard[myRatId];     -- Display new score
        bvSendLocAll ← TRUE;          -- Send loc change to all but
        PupOutput ← TRUE;             -- don't care how long it takes
        END;
      END;
```

```
        ratgone: RatGone ← LOOPHOLE[@b.pupBody];
        RatLeft[ratgone.ratId];
        END;
      ratQuery =>
        BEGIN
        bvSendAlive ← TRUE;
        PupOutput ← TRUE;
        END;
      ratAlive =>
        BEGIN
        ratalive: RatAlive ← LOOPHOLE[@b.pupBody];
        ratHealth[ratalive.ratId].pupRcvd ← TRUE;
        END;
      ENDCASE;
    ReturnFreePupBuffer[b];
    ENDLOOP;
  END;

RatLeft: ENTRY PROCEDURE [ratId: RatId] =
  BEGIN
  IF r2d2[ratId].visible THEN XORToken[ratId, @r2d2];
  r2d2[ratId].visible ← FALSE;
  ratcb.rats[ratId].playing ← FALSE;
  ExitPlayer[ratId];
  UpdateScoreCard[ratId];  -- Update my display
  bvSendAllStatus ← TRUE;    -- Send new player table to all
  PupOutput ← TRUE;
  END;

AllocateNewRat: PROCEDURE [ratnew: RatNew] =
  BEGIN
  ratId: RatId;
  ratInfo: RatInfo;
  FOR ratId IN RatId DO
    ratInfo ← @ratcb.rats[ratId];
    IF NOT ratInfo.playing THEN
      BEGIN
      ratInfo.playing ← TRUE;
      ratInfo.xLoc ← ratnew.xLoc;
      ratInfo.yLoc ← ratnew.yLoc;
      ratInfo.dir ← ratnew.dir;
      ratInfo.score ← 0;
      ratInfo.addr ← ratnew.addr;
      ratInfo.name ← ratnew.name;
      TokenVisible[ratId];              -- See if new guy is visible
      UpdateScoreCard[ratId];           -- Update my score card
      IF r2d2[ratId].visible THEN XORToken[ratId, @r2d2];       -- And my view
      AddNewPlayer [ratId, ratnew.xLoc, ratnew.yLoc, ratnew.dir];
      EXIT;
      END;
    ENDLOOP;
  RETURN;
  END;

TokenVisible: PROCEDURE  [hisRatId: RatId] =
  BEGIN
  -- Sets R2D2[hisRatId] variables
  -- Uses RatCb[hisRatId] as input for his position and direction
  ratLook: RatLook ← @r2d2[hisRatId];
  ratInfo: RatInfo ← @ratcb.rats[hisRatId];
  tx, ty: Loc;
  td: Direction;        -- My position and direction
  ix: CARDINAL;         -- Major index into the vectors table
  ix12: CARDINAL;       -- Minor index into the vectors table (12 vectors per frame)
  ratLook.visible ← FALSE;      -- Init in case we never see the turkey
  IF NOT ratInfo.playing THEN RETURN;
  IF peeking THEN
    BEGIN tx ← xPeek; ty ← yPeek; td ← dirPeek; END
  ELSE
    BEGIN tx ← xloc; ty ← yloc; td ← tdir; END;
  ix ← 0;                               -- Start at beginning of the table
  UNTIL Maze[tx][ty] DO
    SELECT td FROM
      NORTH => ty ← ty + 1;
      SOUTH => ty ← ty - 1;
```

```
      EAST => tx ← tx + 1;
      ENDCASE => ERROR;
    ix ← ix + 1;                          -- Have gone to next "frame" in the table
    IF tx = ratInfo.xLoc AND ty = ratInfo.yLoc THEN
      BEGIN
      ratLook.visible ← TRUE;
      -- See brd for an explaination of the following lines
      -- vector numbers start at zero (doc has 1)
      ix12 ← ix*12;
      ratLook.x ← (table[ix12+3].p2.x + table[ix12+10].p1.x) / 2;
      ratLook.y ← (table[ix12+3].p1.y + table[ix12+3].p2.y) / 2;
      ratLook.tokenId ← relativeTokens [td][ratInfo.dir];
      ratLook.distance ← ix;
      EXIT;
      END;
    ENDLOOP;
  RETURN;
  END;


HoldBreath: PROCEDURE [tx,ty: Loc, td: Direction, ratId: RatId] =
  BEGIN
  Wait[1000];                                     -- Wait a second
  UNTIL Maze[tx][ty] DO
    SELECT td FROM
      NORTH => ty ← ty + 1;
      SOUTH => ty ← ty - 1;
      WEST => tx ← tx - 1;
      EAST => tx ← tx + 1;
      ENDCASE => ERROR;
    IF xloc = tx AND yloc = ty THEN       -- Oh oh, I've been hit
      BEGIN
      killerRatId ← ratId;                -- Set global variable for other process
      bvSendDead ← TRUE;
      PupOutput ← TRUE;
      NewPosition[];                      -- Set new position (to keep from getting multiple
      FlashScreen[];
      score ← score - 5;                  -- minus five points for getting killed
      ratcb.rats[myRatId].score ← score;        -- Update my score now
      UpdateScoreCard[myRatId];           -- Update viewer window
      keyStroke ← TRUE;                   -- Tell the viewer to update display (and send out
      END;
    ENDLOOP;
  ctBreaths ← ctBreaths - 1;
  RETURN[];                               -- Stop this process
  END;

FlashScreen: PROCEDURE =
  BEGIN
  THROUGH [1..4] DO
    InvertScreen[];
    Wait[250];                                    -- Wait a 1/4 second
    ENDLOOP;
  RETURN[];
  END;

InvertScreen: ENTRY PROCEDURE =
  BEGIN OPEN AltoDisplay;
  dcb: DCBHandle;
  FOR dcb ← DCBchainHead.next, dcb.next UNTIL dcb = DCBnil DO
  SELECT dcb.background FROM
    white => dcb.background ← black;
    black => dcb.background ← white;
    ENDCASE;
  ENDLOOP;
  MazeDisplayDefs.InvertHidden[];
  RETURN[];
  END;

NewPosition: PROCEDURE =
  BEGIN
  rndCnt: CARDINAL ← 0;
  xloc ← yloc ← 0;                  -- Start off on an occupied square for loop test sake
  UNTIL NOT Maze[xloc][yloc] DO -- Spin till hit an empty square(can't coexist with a cub
    xloc ← Random[16];
    yloc ← Random[32];
```

```
    IF (rndCnt ← rndCnt + 1) = 100 THEN BEGIN rndCnt ← 0; InitRandom[] END;
    ENDLOOP;
  -- The following is designed to prevent a blank wall at first glimpse
  SELECT FALSE FROM
    Maze[xloc][yloc+1] => tdir ← NORTH;
    Maze[xloc][yloc-1] => tdir ← SOUTH;
    Maze[xloc+1][yloc] => tdir ← EAST;
    Maze[xloc-1][yloc] => tdir ← WEST;
    ENDCASE;
  RETURN
  END;

InitRandom: PROCEDURE =
  BEGIN
  i: CARDINAL;
  p: POINTER TO CARDINAL = LOOPHOLE[430B];
  t: CARDINAL = p↑;
  FOR i IN [0..VectorSize) DO randomVector[i] ← t + randomVector[i] ENDLOOP;
  END;

Wait: ENTRY PROCEDURE [msecs: CARDINAL] =
  BEGIN
  cv: CONDITION;
  ProcessDefs.InitializeCondition[@cv, ProcessDefs.MsecToTicks[msecs]];
  WAIT cv;
  RETURN[];
  END;

Quit: PROCEDURE =
  BEGIN
  ratId: RatId;
  IF ~duke THEN
    BEGIN
    bvSendGoing ← TRUE;
    PupOutput ← TRUE;
    END
  ELSE                              -- Oh oh, I'm the dukerat
    BEGIN
    ratcb.rats[myRatId].playing ← FALSE;                -- Turn me off!
    FOR ratId IN RatId DO
      IF ratcb.rats[ratId].playing THEN          -- If have found a new duke
        BEGIN
        ratcb.dukeRat ← ratId;
        bvSendAllStatus ← TRUE;
        PupOutput ← TRUE;
        EXIT;
        END;
      ENDLOOP;
    END;
  ProcessDefs.SetPriority[1];  --Set old keyboard priority back to normal
  WHILE PupOutput DO ProcessDefs.Yield[] ENDLOOP;
  ImageDefs.StopMesa[];
  END;

NewScoreCard: PROCEDURE =
  BEGIN
  ratid: RatId;
  FOR ratid IN RatId DO UpdateScoreCard[ratid]; ENDLOOP;
  ShowAllPositions [ @ratcb.rats];
  RETURN;
  END;

UpdateScoreCard: PROCEDURE [ratId: RatId] =
  BEGIN
  sv: STRING ← [40];                        -- Temp String
  ix: CARDINAL;
  MazeDisplayDefs.ClearScoreLine[ratId];
  IF ratcb.rats[ratId].playing THEN
    BEGIN
    FOR ix IN [0..20) DO
      sv[ix] ← ratcb.rats[ratId].name[ix];
      ENDLOOP;
    sv.length ← 20;
    StringDefs.AppendDecimal[sv, ratcb.rats[ratId].score];
    MazeDisplayDefs.WriteScoreString[ratId, sv];
    END;
```

```
  If r2d2[ratId].visible THEN MazeDisplayDefs.InvertScoreLine[ratId];
  RETURN;
  END;

oldBits: KeyDefs.KeyBits ← KeyDefs.Keyst;
old: POINTER TO KeyDefs.KeyBits ← @oldBits;
nowBits: KeyDefs.KeyBits ← KeyDefs.Keyst;
now: POINTER TO KeyDefs.KeyBits ← @nowBits;

bigMaze: BOOLEAN ← FALSE;

ReadKeys: ENTRY PROCEDURE =
  BEGIN OPEN KeyDefs;
  DO
    WAIT wakeup;                                    -- wait for the naked notify
    oldBits ← nowBits;  -- Save "old" bits for diff check
    nowBits ← Keyst;  -- Get current status
    now.blank ← 0;  -- blank out trash
    IF nowt = oldt THEN LOOP;  -- if the same, forget it
   IF ~ bigMaze THEN
    IF NOT peeking THEN
      BEGIN
      IF (now.Keyset1 = down AND old.Keyset1 = up) OR
        (now.A = down AND old.A = up) THEN AboutFace[];
      IF (now.Keyset2 = down AND old.Keyset2 = up) OR
        (now.S = down AND old.S = up) THEN LeftTurn[];
      IF (now.Keyset3 = down AND old.Keyset3 = up) OR
        (now.D = down AND old.D = up) THEN Forward[];
      IF (now.Keyset4 = down AND old.Keyset4 = up) OR
        (now.F = down AND old.F = up) THEN RightTurn[];
      IF (now.Keyset5 = down AND old.Keyset5 = up) OR
        (now.Space = down AND old.Space = up) THEN Backward[];
      IF now.Red = down AND old.Red = up THEN PeekLeft[];
      IF now.Blue = down AND old.Blue = up THEN PeekRight[];
      IF now.Yellow = down AND old.Yellow = up THEN Shoot[];
      IF now.I = down AND old.I = up THEN MakeInvincible[TRUE];
      IF now.K = down AND old.K = up THEN MakeInvincible[FALSE]; --killable
      IF now.E = down THEN BigMaze[];
      END
    ELSE                            -- if peeking
      IF (now.Red = up) OR (now.Blue = up) THEN PeekStop[]
  ELSE  --if BigMaze
      IF now.E = up THEN SmallMaze[];
    IF now.DEL = down THEN Quit[];      -- TIme to leave the game
    ENDLOOP;
  END;  -- of ReadKeys

DispatchKeys: PROCEDURE =
  BEGIN
  DO
    WHILE NOT keyStroke DO
        ProcessDefs.Yield[];
        ENDLOOP;
    keyStroke ← FALSE;
    IF NOT peeking THEN
      BEGIN
      ShowPosition[xloc,yloc, Invincible, tdir];  -- show new location
      ShowView[xloc,yloc,tdir];          -- and display the view
      END
    ELSE                            -- if peeking
      ShowView[xPeek,yPeek,dirPeek];    -- then display the peek view
    bvSendLocAll ← TRUE;                -- Send loc change to all and
    PupOutput ← TRUE;             -- wait till they are sent!
    WHILE PupOutput DO ProcessDefs.Yield[] ENDLOOP;
    ENDLOOP;
  END;  -- OF DispatchKeys

BigMaze: PROCEDURE =
        BEGIN
        bigMaze ← TRUE;
        MakeBigMaze[];
        END;

SmallMaze: PROCEDURE =
        BEGIN
        bigMaze ← FALSE;
```

```
        MakeSmallMaze[];
        END;
MakeInvincible: PROCEDURE [neverDie: BOOLEAN] =
  BEGIN
  Invincible ← neverDie;
  ShowPosition[xloc,yloc, Invincible, tdir];  -- show invincibility
  END;

AboutFace: PROCEDURE = INLINE                    -- About face
  BEGIN
  aboutFace: ARRAY Direction OF Direction = [SOUTH, NORTH, WEST, EAST];
  tdir ← aboutFace[tdir];
  keyStroke ← TRUE;
  END;

LeftTurn: PROCEDURE = INLINE                     -- Left turn
  BEGIN
  leftTurn: ARRAY Direction OF Direction = [WEST, EAST, NORTH, SOUTH];
  tdir ← leftTurn[tdir];
  keyStroke ← TRUE;
  END;

RightTurn: PROCEDURE = INLINE                    -- Right turn
  BEGIN
  rightTurn: ARRAY Direction OF Direction = [EAST, WEST, SOUTH, NORTH];
  tdir ← rightTurn[tdir];
  keyStroke ← TRUE;
  END;

Forward: PROCEDURE = INLINE                      -- Forward one
  BEGIN -- Move forward
  tx: CARDINAL ← xloc;
  ty: CARDINAL ← yloc;
  SELECT tdir FROM
    NORTH => IF ~Maze[xloc][yloc+1] THEN ty ← yloc + 1;
    SOUTH => IF ~Maze[xloc][yloc-1] THEN ty ← yloc - 1;
    EAST => IF ~Maze[xloc+1][yloc] THEN tx ← xloc + 1;
    WEST => IF ~Maze[xloc-1][yloc] THEN tx ← xloc - 1;
    ENDCASE => ERROR;
  IF tx#xloc OR ty#yloc THEN
  BEGIN xloc ← tx; yloc ← ty; keyStroke ← TRUE; END;
  END;

Backward: PROCEDURE = INLINE                     -- Backward one
  BEGIN -- Move backward
  tx: CARDINAL ← xloc;
  ty: CARDINAL ← yloc;
  SELECT tdir FROM
    NORTH => IF ~Maze[xloc][yloc-1] THEN ty ← yloc - 1;
    SOUTH => IF ~Maze[xloc][yloc+1] THEN ty ← yloc + 1;
    EAST => IF ~Maze[xloc-1][yloc] THEN tx ← xloc - 1;
    WEST => IF ~Maze[xloc+1][yloc] THEN tx ← xloc + 1;
    ENDCASE => ERROR;
  IF tx#xloc OR ty#yloc THEN
  BEGIN xloc ← tx; yloc ← ty; keyStroke ← TRUE; END;
  END;

PeekLeft: PROCEDURE = INLINE
  BEGIN
  xPeek ← xloc;
  yPeek ← yloc;
  dirPeek ← tdir;
  SELECT tdir FROM
    NORTH =>
      IF ~Maze[xloc][yloc+1] THEN BEGIN yPeek ← yloc+1; dirPeek ← WEST END;
    SOUTH =>
      IF ~Maze[xloc][yloc-1] THEN BEGIN yPeek ← yloc-1; dirPeek ← EAST END;
    EAST =>
      IF ~Maze[xloc+1][yloc] THEN BEGIN xPeek ← xloc+1; dirPeek ← NORTH END;
    WEST =>
      IF ~Maze[xloc-1][yloc] THEN BEGIN xPeek ← xloc-1; dirPeek ← SOUTH END;
    ENDCASE => ERROR;
  -- If any change; display the new view without moving!
  IF xPeek#xloc OR yPeek#yloc THEN
  BEGIN peeking ← TRUE; keyStroke ← TRUE END;
```

```
  END;

PeekRight: PROCEDURE = INLINE
  BEGIN
  xPeek ← xloc;
  yPeek ← yloc;
  dirPeek ← tdir;
  SELECT tdir FROM
    NORTH =>
      IF ~Maze[xloc][yloc+1] THEN BEGIN yPeek ← yloc+1; dirPeek ← EAST END;
    SOUTH =>
      IF ~Maze[xloc][yloc-1] THEN BEGIN yPeek ← yloc-1; dirPeek ← WEST END;
    EAST =>
      IF ~Maze[xloc+1][yloc] THEN BEGIN xPeek ← xloc+1; dirPeek ← SOUTH END;
    WEST =>
      IF ~Maze[xloc-1][yloc] THEN BEGIN xPeek ← xloc-1; dirPeek ← NORTH END;
    ENDCASE => ERROR;
  -- If any change; display the new view without moving!
  IF xPeek#xloc OR yPeek#yloc THEN
  BEGIN peeking ← TRUE; keyStroke ← TRUE END;
  RETURN
  END;

PeekStop: PROCEDURE = INLINE
  BEGIN
  peeking ← FALSE;
  keyStroke ← TRUE;
  RETURN
  END;

Shoot: PROCEDURE = INLINE
  BEGIN
  bvSendKill ← TRUE;              -- Send out a kill msg
  PupOutput ← TRUE;               -- Wake up the pup output process
  RETURN
  END;

RatDoctor: PUBLIC PROCEDURE =
  BEGIN
  ratId: RatId;
  FOR ratId IN RatId DO
    ratHealth[ratId].count ← 0;
    ratHealth[ratId].pupSend ← FALSE;
    ENDLOOP;
  DO
    FOR ratId IN RatId DO
      ratHealth[ratId].pupRcvd ← FALSE;
      ENDLOOP;
    Wait[10*1000]; -- wait 10 seconds
    FOR ratId IN RatId DO
      IF ~ratcb.rats[ratId].playing OR ratId=myRatId THEN LOOP;
      IF ratHealth[ratId].pupRcvd THEN
        BEGIN
        ratHealth[ratId].count ← 0;
        LOOP
        END;
      IF (ratHealth[ratId].count ← ratHealth[ratId].count + 1) < 5 THEN
        BEGIN
        ratHealth[ratId].pupSend ← TRUE;
        bvSendQuery ← TRUE;
        PupOutput ← TRUE;
        LOOP;
        END;
      IF duke OR (ratcb.dukeRat = ratId AND NextRatId[ratId]=myRatId) THEN
        BEGIN
        duke ← TRUE;
        ratcb.dukeRat ← myRatId;
        RatLeft[ratId];
        END;
      ENDLOOP;
    ENDLOOP;
  END;

NextRatId: PROCEDURE [ratId: RatId] RETURNS [ixRatId: RatId] =
  BEGIN
  FOR ixRatId IN RatId DO
```

```
    IF ratcb.rats[ixRatId].playing AND ixRatId#ratid THEN RETURN[ixRatId];
    ENDLOOP;
  RETURN[ratid];
  END;

Play: PROCEDURE =
  BEGIN
  DO
    WHILE NOT PupOutput DO ProcessDefs.Yield[] ENDLOOP;
    PupOutput ← FALSE;
    IF bvSendLocAll THEN SendLocToAll[];
    IF bvSendOneStatus THEN SendStatus[newAddr];
    IF bvSendAllStatus THEN SendAllStatus[];
    IF bvSendKill THEN SendKill[];
    IF bvSendDead THEN SendDead[];
    IF bvSendGoing THEN SendGoing[];
    IF bvSendQuery THEN SendQuery[];
    IF bvSendAlive THEN SendAlive[];
    ENDLOOP;
  END;


-- Main Line Code

--*************************************************************************--
FOR ratId IN [0..MaxRats) DO
        ratcb.rats[ratId].playing ← FALSE;
        ENDLOOP;
--*************************************************************************--
START MazeInit;
FrameDefs.UnNew[LOOPHOLE[MazeInit]];
Play[];

END...
```

```
-- edited by Sandman on Jun 22, 1979 10:20 AM

PACK MazeWar, TheMaze, Tables, Rats, MazeDisplay;

Maze: CONFIGURATION
  IMPORTS FrameDefs, ImageDefs, MiscDefs  ProcessDefs,
    SegmentDefs, StreamDefs, StringDefs, SystemDefs
  CONTROL MazeWar =
  BEGIN

  TinyPup;

  TheMaze;
  Tables;
  Rats;
  MazeDisplay;
  MazeWar;
  MMFont;

  MazeInit;  -- Item 3 in MMOps is unbound
  MMDisplay;
  MMKeyboard;
  StreamIO;

  END.
```

```
-- MazeDefs.mesa; edited by Guyton; June 25, 1979 11:02 AM

DIRECTORY
  PupDefs: FROM "pupdefs" USING [PupAddress],
  PupTypes: FROM "puptypes" USING [PupSocketID, PupType];

MazeDefs: DEFINITIONS =
BEGIN

MazeType: TYPE =
  POINTER TO ARRAY [0..16) OF PACKED ARRAY [0..32) OF BOOLEAN;
Direction: TYPE = {NORTH, SOUTH,  EAST, WEST};
XYpair: TYPE = RECORD [p1,p2: XYpoint];
XYpoint: TYPE = RECORD [x,y: INTEGER];
XY:   TYPE = RECORD[xcor: INTEGER, ycor: INTEGER];
szXYpair:       CARDINAL = SIZE[XYpair];
szTwoXYpair:    CARDINAL = 2*SIZE[XYpair];

View: TYPE = {left, right, rear, front};

ratLocation: PupTypes.PupType = pt170;
ratKill: PupTypes.PupType = pt171;
ratDead: PupTypes.PupType = pt172;
ratStatus: PupTypes.PupType = pt173;
ratNew: PupTypes.PupType = pt174;
ratGoing: PupTypes.PupType = pt175;
ratQuery: PupTypes.PupType = pt176;
ratAlive: PupTypes.PupType = pt177;
ratSurvey: PupTypes.PupType = pt100;
Password: CARDINAL = 123456B;
mazeSocket: PupTypes.PupSocketID = [046501B, 055105B];  -- Magic socket

Loc: TYPE = CARDINAL [0..32];
Score: TYPE = INTEGER;
RatName: TYPE = PACKED ARRAY [0..20) OF CHARACTER;
MaxRats: CARDINAL = 8;
RatId: TYPE = CARDINAL [0..MaxRats);                    -- Index into rats array

RatLocation: TYPE = POINTER TO AqRatLocation;
AqRatLocation: TYPE = RECORD [
  ratId: RatId,
  xLoc: Loc,
  yLoc: Loc,
  dir: Direction,
  score: Score];

RatKill: TYPE = POINTER TO AqRatKill;
AqRatKill: TYPE = RECORD [
  ratId: RatId,
  xLoc: Loc,
  yLoc: Loc,
  dir: Direction];

RatDead: TYPE = POINTER TO AqRatDead;
AqRatDead: TYPE = RECORD [
  ratId: RatId,
  killedBy: RatId];

RatNew: TYPE = POINTER TO AqRatNew;
AqRatNew: TYPE = RECORD [
  pass: CARDINAL,
  xLoc: Loc,
  yLoc: Loc,
  dir: Direction,
  addr: PupDefs.PupAddress,
  name: RatName];

RatGone: TYPE = POINTER TO AqRatGone;
AqRatGone: TYPE = RECORD [
  ratId: RatId];

RatQuery: TYPE = POINTER TO AqRatQuery;
AqRatQuery: TYPE = RECORD [
  ratId: RatId];

RatAlive: TYPE = POINTER TO AqRatAlive;
```

```
AqRatAlive: TYPE = RECORD [
  ratId: RatId];

RatStatus: TYPE = POINTER TO RatCb;

RatObject: TYPE = RECORD [
  playing: BOOLEAN,
  xLoc, yLoc: Loc,
  dir: Direction,
  score: Score,
  addr: PupDefs.PupAddress,
  name: RatName];

RatInfo: TYPE = POINTER TO RatObject;

RatCb: TYPE = RECORD [
  dukeRat: RatId,
  rats: ARRAY RatId OF RatObject];

TokenId:        TYPE = View;

R2d2: TYPE = ARRAY RatId OF RatAppearance;

RatLook: TYPE = POINTER TO RatAppearance;

RatAppearance: TYPE = RECORD [
  visible: BOOLEAN,
  x, y: CARDINAL,       -- bitmap location of the token bitmap
  distance: CARDINAL,   -- distance away from the viewer
  tokenId: TokenId];

RelativeTokens: TYPE = ARRAY Direction OF ARRAY Direction OF TokenId;

RatPupHealth: TYPE = MACHINE DEPENDENT RECORD [
 pupSend: BOOLEAN,
 pupRcvd: BOOLEAN,
 count: CARDINAL [0..37777B] ];

RatHealth: TYPE = ARRAY RatId OF RatPupHealth;

TheMaze: PROGRAM;
Tables: PROGRAM;
Rats: PROGRAM;
MazeInit: PROGRAM;

DispatchKeys: PROCEDURE;
NewPosition: PROCEDURE;
NewScoreCard: PROCEDURE;
ReadKeys: PROCEDURE;
ReadRats: PROCEDURE;
TokenVisible: PROCEDURE [hisRatId: RatId];
ShowView: PROCEDURE [x, y: CARDINAL, dir: Direction];
RatDoctor: PROCEDURE;

END.
```

```
-- MazeDisplay.mesa Edited by Gobbel on June 22, 1979  3:51 PM
--Hacked version by: BAM; Last edit:  October 28, 1979  4:31 PM
DIRECTORY
  BitBltDefs: FROM "bitbltdefs" USING [BBptr, BBTable, BITBLT],
  FontDefs: FROM "fontdefs" USING [BitmapState],
  InlineDefs: FROM "inlinedefs" USING [BITAND, DIVMOD, BITOR],
  MazeDefs: FROM "mazedefs" USING [Direction, R2d2, RatId, View, XYpair, MaxRats],
  MazeDisplayDefs: FROM "mazedisplaydefs" USING
    [DCBHandle, DCBRec, DoubleWord, pRat],
  Mopcodes: FROM "mopcodes" USING [zINC];

MazeDisplay: MONITOR
  IMPORTS BitBltDefs, InlineDefs
  EXPORTS MazeDisplayDefs =
PUBLIC BEGIN OPEN MazeDisplayDefs;


RatId: TYPE = MazeDefs.RatId;

pages: CARDINAL = 6;
sbmr: CARDINAL = pages*4;
sbca: POINTER;
topMargin, current, hidden, bigMazeMap, scoreDCB: DCBHandle;  --mazeMap removed
font: PUBLIC Fptr;
ratBB, lineBB, diagBB: PUBLIC BitBltDefs.BBptr;
leftBBpattern, rightBBpattern: ARRAY [0..32) OF DoubleWord ← ALL[[0, 0]];

myRatId: RatId ← 0;
Invincible: BOOLEAN ← FALSE;
BigMaze: BOOLEAN ← FALSE;
halt: BOOLEAN ← FALSE;

RatState: TYPE = RECORD
        [
        playing: BOOLEAN,
        x, y: CARDINAL,
        dir: MazeDefs.Direction
        ];

ClearArray: ARRAY[0..MazeDefs.MaxRats) OF RatState ← ALL[[FALSE,1,1,NORTH]];

SetMyRatId: PROCEDURE [ratId: RatId] =
  BEGIN
  myRatId ← ratId;
  ClearArray[myRatId].playing ← TRUE;
  END;


DrawVerticalLine: PROCEDURE [x, y, height: CARDINAL] = INLINE
  BEGIN
IF halt THEN RETURN;
  lineBB.dw ← 1;
  lineBB.dh ← height;
  lineBB.dlx ← x;
  lineBB.dty ← y;
  BitBltDefs.BITBLT[lineBB];
  END;

DrawHorizontalLine: PROCEDURE [x, y, width: CARDINAL] = INLINE
  BEGIN
IF halt THEN RETURN;
  lineBB.dlx ← x;
  lineBB.dty ← y;
  lineBB.dh ← 1;
  lineBB.dw ← width;
  BitBltDefs.BITBLT[lineBB];
  END;

-- Only for slopes of 1 or -1
DrawDiagonalLine: PROCEDURE [x1,y1,x2,y2: INTEGER] = INLINE
  BEGIN
  width: CARDINAL = x2-x1;
  i, rem, count: CARDINAL;
  rightSlope: BOOLEAN = y1>y2;
IF halt THEN RETURN;
  [count, rem] ← InlineDefs.DIVMOD[width, 32];
```

```
    ;
  diagBB.dbca ← hidden.bitmap;
  diagBB.sbca ← IF rightSlope THEN @rightBBpattern ELSE @leftBBpattern;
  IF count>0 THEN BEGIN
    diagBB.dw ← diagBB.dh ← 32;
    IF rightSlope THEN
      BEGIN
      FOR i IN [0..count) DO
        diagBB.dlx ← x1;
        diagBB.dty ← y1-32;
        BitBltDefs.BITBLT[diagBB];
        x1 ← x1+32;
        y1 ← y1-32;
        ENDLOOP;
      END
    ELSE
      BEGIN
      FOR i IN [0..count) DO
        diagBB.dlx ← x1;
        diagBB.dty ← y1;
        BitBltDefs.BITBLT[diagBB];
        x1 ← x1+32;
        y1 ← y1+32;
        ENDLOOP;
      END;
    END;
  IF rem>0 THEN
    BEGIN
    diagBB.dw ← diagBB.dh ← rem;
    diagBB.dlx ← x1;
    diagBB.dty ← IF rightSlope THEN y1-rem ELSE y1;
    IF rightSlope THEN diagBB.sty ← 32-rem;
    BitBltDefs.BITBLT[diagBB];
    diagBB.sty ← 0;
    END;
  RETURN;
  END;

MakeBigMaze: PUBLIC PROCEDURE =
BEGIN
i: CARDINAL;
IF  halt OR BigMaze THEN RETURN;
halt ← TRUE;  --multi-process coordination
--FOR i IN [0..MazeDefs.MaxRats) DO
--      IF ClearArray[i].playing THEN
--              ClearSquare[ClearArray[i].x, ClearArray[i].y];
--      ENDLOOP;
BigMaze ← TRUE;
--now switch bit maps
topMargin.next ← bigMazeMap;
bigMazeMap.next ← scoreDCB;
halt ← FALSE;
--DisplayAll[];
END;
MakeSmallMaze: PUBLIC PROCEDURE =
BEGIN
i: CARDINAL;
IF  halt OR ~BigMaze THEN RETURN;
halt ← TRUE;  --multi-process coordination
FOR i IN [0..MazeDefs.MaxRats) DO
        IF ClearArray[i].playing THEN
                BigClearSquare[ClearArray[i].x, ClearArray[i].y];
        ENDLOOP;
BigMaze ← FALSE;
--now switch bit maps
topMargin.next ← current;
--current.next ← mazeMap;
--mazeMap.next ← scoreDCB;
current.next ← scoreDCB;
halt ← FALSE;
BigDisplayAll[];
END;
--DisplayAll: PROCEDURE =
--BEGIN
--i: CARDINAL;
--IF halt THEN RETURN;
--FOR i IN [0..MazeDefs.MaxRats) DO OPEN ClearArray[i];
```

```
--      IF playing THEN
--            IF i=myRatId THEN ShowMe[x, y, Invincible, dir]
--            ELSE ShowOther[i, x, y, dir];
--      ENDLOOP;
--END;
BigDisplayAll: PROCEDURE =
BEGIN
i: CARDINAL;
IF halt THEN RETURN;
FOR i IN [0..MazeDefs.MaxRats) DO OPEN ClearArray[i];
        IF playing THEN
                IF i=myRatId THEN BigShowMe[x, y, Invincible, dir]
                ELSE BigShowOther[i, x, y, dir];
        ENDLOOP;
END;

--ClearPosition: PROCEDURE [ratId: RatId, xClear, yClear: CARDINAL] =
--  BEGIN
--  tempRatId: RatId;
--IF halt THEN RETURN;
--  ClearSquare[xClear, yClear];
--  now  check if anyone needs to be redrawn
--    FOR tempRatId IN RatId DO OPEN ClearArray[tempRatId];
--      IF tempRatId=ratId OR ~playing THEN LOOP;
--      IF x = xClear AND y = yClear THEN
--              BEGIN
--              IF tempRatId=myRatId THEN ShowMe[x, y, Invincible, dir]
--              ELSE ShowOther[tempRatId, x, y, dir];
--              EXIT;
--              END;
--      ENDLOOP;
--  RETURN;
--  END;
--
--ClearSquare: PROCEDURE [xClear, yClear: CARDINAL] =
--BEGIN
--  mazeIndex: CARDINAL ← xClear*32*8 + yClear;
--  lastIndex: CARDINAL = mazeIndex+8*32;
--  maze: POINTER TO PACKED ARRAY OF [0..377B] ← mazeMap.bitmap;
--  DO
--    maze[mazeIndex] ← 0;
--    mazeIndex ← mazeIndex + 32;
--    IF mazeIndex = lastIndex THEN EXIT;
--    ENDLOOP;
--END;
--
BigClearSquare: PROCEDURE [xClear, yClear: CARDINAL] =
BEGIN
  mazeIndex: CARDINAL ← xClear*32*16 + yClear;
  maze: POINTER TO PACKED ARRAY OF WORD ← bigMazeMap.bitmap;
  THROUGH [0..16) DO
    maze[mazeIndex] ← 0;
    mazeIndex ← mazeIndex + 32;
    ENDLOOP;
END;

BigClearPosition: PROCEDURE [ratId: RatId, xClear, yClear: CARDINAL] =
  BEGIN
  tempRatId: RatId;
IF halt THEN RETURN;
  BigClearSquare[xClear, yClear];
  --now  check if anyone needs to be redrawn
    FOR tempRatId IN RatId DO OPEN ClearArray[tempRatId];
        IF tempRatId=ratId OR ~playing THEN LOOP;
        IF x = xClear AND y = yClear THEN
                BEGIN
                IF tempRatId=myRatId THEN BigShowMe[x, y, Invincible, dir]
                ELSE BigShowOther[tempRatId, x, y, dir];
                EXIT;
                END;
        ENDLOOP;
RETURN;
  END;


InvertTop: PROCEDURE =
```

```
        BEGIN
        SELECT topMargin.background FROM
                white => topMargin.background ← black;
                black => topMargin.background ← white;
                ENDCASE;
        RETURN;
        END;

ShowPosition: PROCEDURE [xloc, yloc: CARDINAL, invincible: BOOLEAN,
                         tdir: MazeDefs.Direction] =
        BEGIN
        IF halt THEN RETURN;
        IF BigMaze THEN
                BEGIN
                BigClearPosition[myRatId, ClearArray[myRatId].x, ClearArray[myRatId].y];
                BigShowMe[xloc, yloc, invincible, tdir];
                END
--      ELSE    BEGIN
--              ClearPosition[myRatId, ClearArray[myRatId].x, ClearArray[myRatId].y];
--              ShowMe[xloc, yloc, invincible, tdir];
--              END;
        END;

--ShowMe: PROCEDURE [xloc, yloc: CARDINAL, invincible: BOOLEAN, tdir: MazeDefs.Direction] =
--  BEGIN
--  i, mazeIndex, arrowIndex: CARDINAL;
--  maze: POINTER TO PACKED ARRAY OF [0..377B] ← mazeMap.bitmap;
--  normalarrows: PACKED ARRAY [0..32) OF [0..377B] = [
--    0B, 10B, 14B, 376B, 377B, 376B, 14B, 10B,
--    20B, 60B, 177B, 377B, 177B, 60B, 20B, 0B,
--    34B, 34B, 34B, 34B, 177B, 76B, 34B, 10B,
--    10B, 34B, 76B, 177B, 34B, 34B, 34B, 34B];
--  invinciblearrows: PACKED ARRAY [0..32) OF [0..377B] = [
--    0B, 30B, 214B, 256B, 377B, 256B, 214B, 30B,
--    30B, 61B, 165B, 377B, 165B, 61B, 30B, 0B,
--    76B, 10B, 34B, 111B, 177B, 76B, 34B, 10B,
--    10B, 34B, 76B, 177B, 111B, 34B, 10B, 76B];
--  arrowIndex ← LOOPHOLE[tdir, CARDINAL]*8;
--  mazeIndex ← xloc*32*8 + yloc;
--  IF invincible THEN
--        FOR i IN [0..8) DO
--          maze[mazeIndex] ← invinciblearrows[arrowIndex+i];
--          mazeIndex ← mazeIndex + 32;
--          ENDLOOP
--  ELSE FOR i IN [0..8) DO
--    maze[mazeIndex] ← normalarrows[arrowIndex+i];
--    mazeIndex ← mazeIndex + 32;
--    ENDLOOP;
--  ClearArray[myRatId] ← [playing: TRUE, x: xloc, y: yloc, dir: tdir]; Save for clearing next time
--  Invincible ← invincible;
--  RETURN;
--  END;
--
BigShowMe: PROCEDURE [xloc, yloc: CARDINAL, invincible: BOOLEAN,
                         tdir: MazeDefs.Direction] =
  BEGIN
  i, mazeIndex: CARDINAL;
  maze: POINTER TO PACKED ARRAY OF WORD ← bigMazeMap.bitmap;
  normalArrows: ARRAY MazeDefs.Direction OF ARRAY [0..16) OF WORD = [
        [0B,    200B,   300B,   340B,   --right
         360B,  370B,   177774B,        177776B,
         177777B,       177776B,        177774B,        370B,
         360B,  340B,   300B,   200B],
        [4006,  1400B,  3400B,  7400B,  --left
         17400B,        37777B, 77777B, 177777B,
         77777B,        37777B, 17400B, 7400B,
         3400B, 1400B,  400B,   0B],
        [1740B, 1740B,  1740B,  1740B,  --down
         1740B, 1740B,  1740B,  1740B,
         77777B,        37776B, 17774B, 7770B,
         3760B, 1740B,  700B,   200B],
        [200B,  700B,   1740B,  3760B,  --up
         7770B, 17774B, 37776B, 77777B,
         1740B, 1740B,  1740B,  1740B,
         1740B, 1740B,  1740B,  1740B]];
  invincibleArrows: ARRAY MazeDefs.Direction OF ARRAY [0..16) OF WORD = [
```

```
        [0B,    600B,   300B,   340B,   --right
         160B,  210B,   177464B,        177102B,
         177043B,       177022B,        177544B,        210B,
         160B,  340B,   300B,   600B],
        [700B,  1400B,  3400B,  7000B,  --left
         10400B,        233778,  44177B, 142177B,
         41177B,        263778,  10400B, 7000B,
         3400B, 1400B,  700B,   0B],
        [1740B, 1740B,  1740B,  1740B,  --down
         1740B, 1740B,  1740B,  41041B,
         72327B,        34416B, 14214B, 4110B,
         2620B, 1040B,  700B,   200B],
        [200B,  700B,   1040B,  2320B,  --up
         4410B, 14214B, 34116B, 72627B,
         41041B,        1740B,  1740B,  1740B,
         1740B, 1740B,  1740B,  1740B]];
  mazeIndex ← xloc*32*16 + yloc;
  IF invincible THEN
        FOR i IN [0..16) DO
        maze[mazeIndex] ← invincibleArrows[tdir][i];
        mazeIndex ← mazeIndex + 32;
        ENDLOOP
  ELSE
        FOR i IN [0..16) DO
        maze[mazeIndex] ← normalArrows[tdir][i];
        mazeIndex ← mazeIndex + 32;
        ENDLOOP;
  ClearArray[myRatId] ← [playing: TRUE, x: xloc, y: yloc, dir: tdir];-- Save for clearing next time
  Invincible ← invincible;
  RETURN;
  END;


DisplayOthersPosition: PROCEDURE [which, xloc, yloc: CARDINAL,
                                  tdir: MazeDefs.Direction] =
        BEGIN
IF halt THEN RETURN;
        IF BigMaze THEN
                BEGIN
                BigClearPosition[which, ClearArray[which].x, ClearArray[which].y];
                IF ClearArray[which].playing THEN BigShowOther[which, xloc, yloc, tdir];
                END
--      ELSE    BEGIN
--              ClearPosition[which, ClearArray[which].x, ClearArray[which].y];
--              IF ClearArray[which].playing THEN ShowOther[which, xloc, yloc, tdir];
--              END;
        END;

--ShowOther: PROCEDURE [which, xloc, yloc: CARDINAL,
--                                  tdir: MazeDefs.Direction] =
--  BEGIN
--  i, mazeIndex, arrowIndex, oldX, oldY: CARDINAL;
--  maze: POINTER TO PACKED ARRAY OF [0..377B] ← mazeMap.bitmap;
--  arrows: PACKED ARRAY [0..32) OF [0..377B] = [
--    0B, 10B, 14B, 376B, 203B, 376B, 14B, 10B,
--    20B, 60B, 177B, 301B, 177B, 60B, 20B, 0B,
--    34B, 24B, 24B, 24B, 167B, 66B, 34B, 10B,
--    10B, 34B, 66B, 167B, 24B, 24B, 24B, 34B];
--      arrowIndex ← LOOPHOLE[tdir, CARDINAL]*8;
--      mazeIndex ← xloc*32*8 + yloc;
--      FOR i IN [0..8) DO
--              maze[mazeIndex] ← arrows[arrowIndex+i];
--              mazeIndex ← mazeIndex + 32;
--              ENDLOOP;
--      ClearArray[which] ← [playing: TRUE, x: xloc, y: yloc, dir: tdir];
--  RETURN;
--  END;
--
BigShowOther: PROCEDURE [which, xloc, yloc: CARDINAL,
                                  tdir: MazeDefs.Direction] =
  BEGIN
  i, mazeIndex: CARDINAL;
  maze: POINTER TO PACKED ARRAY OF WORD ← bigMazeMap.bitmap;
  arrows: ARRAY MazeDefs.Direction OF ARRAY [0..16) OF WORD = [
        [0B,    200B,   300B,   340B,   --right
         177660B,       1000308,        1000146,        100006B,
```

```
        100003B,        100006B,        100014B,        100030B,
        177660B,        340B,   300B,   200B],
       [0B,     400B,   1400B,  3400B,  --left
        6777B,  14001B, 30001B, 60001B,
        140001B,        60001B, 30001B, 14001B,
        6777B,  3400B,  1400B,  400B],
       [7770B,  4010B,  4010B,  4010B,  --down
        4010B,  4010B,  4010B,  4010B,
        74017B,         34016B, 14014B, 6030B,
        3060B,  1540B,  700B,   200B],
       [200B,   700B,   1540B,  3060B,  --up
        6030B,  14014B, 30006B, 74014B,
        4010B,  4010B,  4010B,  4010B,
        4010B,  4010B,  4010B,  7770B]];
  numbers: ARRAY [0..7] OF ARRAY [0..16) OF WORD = [
       [0,      0,      0,      0,      --0
        0,      0,      300B,   440B,
        440B,   440B,   300B,   0,
        0,      0,      0,      0],
       [0,      0,      0,      0,      --1
        0,      0,      600B,   200B,
        200B,   200B,   700B,   0,
        0,      0,      0,      0],
       [0,      0,      0,      0,      --2
        0,      0,      600B,   1100B,
        200B,   400B,   1700B,  0,
        0,      0,      0,      0],
       [0,      0,      0,      0,      --3
        0,      0,      1600B,  100B,
        600B,   100B,   1600B,  0,
        0,      0,      0,      0],
       [0,      0,      0,      0,      --4
        0,      0,      1100B,  1100B,
        1740B,  100B,   100B,   0,
        0,      0,      0,      0],
       [0,      0,      0,      0,      --5
        0,      0,      1700B,  1000B,
        1700B,  100B,   1700B,  0,
        0,      0,      0,      0],
       [0,      0,      0,      0,      --6
        0,      0,      1000B,  1000B,
        1700B,  1100B,  1700B,  0,
        0,      0,      0,      0],
       [0,      0,      0,      0,      --7
        0,      0,      1700B,  100B,
        200B,   400B,   1000B,  0,
        0,      0,      0,      0]];
       mazeIndex ← xloc*32*16 + yloc;
       FOR i IN [0..16) DO
              maze[mazeIndex] ← InlineDefs.BITOR[arrows[tdir][i], numbers[which][i]];
              mazeIndex ← mazeIndex + 32;
              ENDLOOP;
       ClearArray[which] ← [playing: TRUE, x: xloc, y: yloc, dir: tdir];
  RETURN;
  END;

ExitPlayer: PROCEDURE [ratId: RatId] =
       BEGIN
       IF BigMaze THEN BigClearPosition[ratId, ClearArray[ratId].x, ClearArray[ratId].y];
--     ELSE ClearPosition[ratId, ClearArray[ratId].x, ClearArray[ratId].y];
       ClearArray[ratId].playing ← FALSE;
       END;
AddNewPlayer: PROCEDURE [ratId: RatId, xloc, yloc: CARDINAL, tdir: MazeDefs.Direction] =
       BEGIN
       ClearArray[ratId].playing ← TRUE;
       DisplayOthersPosition[ratId, xloc, yloc, tdir];
       END;

ShowAllPositions: PROCEDURE [rats: pRat] =
       BEGIN
       ratId: RatId;
       FOR ratId IN RatId DO
              IF ratId = myRatId THEN LOOP;
              IF ClearArray[ratId].playing  AND ~rats[ratId].playing THEN
                            ExitPlayer[ratId]
              ELSE IF ~ClearArray[ratId].playing AND rats[ratId].playing THEN
```

```
                                AddNewPlayer[ratId, rats[ratId].xLoc,
                                    rats[ratId].yLoc, rats[ratId].dir]
                ELSE IF ClearArray[ratId].playing THEN DisplayOthersPosition[ratId,
                                    rats[ratId].xLoc, rats[ratId].yLoc, rats[ratId].dir];
                ENDLOOP;
        END;

--*************************************************************************--

EVEN: PROCEDURE[v: UNSPECIFIED] RETURNS [UNSPECIFIED] =
  BEGIN RETURN[v+InlineDefs.BITAND[v, 1]] END;

PlotLine: PROCEDURE [p: POINTER TO MazeDefs.XYpair, two: BOOLEAN]
  RETURNS [POINTER TO MazeDefs.XYpair] =
  BEGIN
  fromx, fromy, tox, toy, ix, iy: INTEGER;
  fromx ← 0;  -- force an fgt entry
  lineBB.dbca ← hidden.bitmap;
  DO
    [x: fromx, y: fromy] ← p.p1;
    [x: tox, y: toy] ← p.p2;
    SELECT TRUE FROM
      fromx = tox =>  -- vertical?
        BEGIN
        y: CARDINAL ← MIN[fromy, toy];
        h: CARDINAL ← ABS[fromy-toy];
        IF h # 0 THEN DrawVerticalLine[fromx, y, h];
        END;
      fromy = toy =>  -- horizontal?
        BEGIN
        x: CARDINAL ← MIN[fromx, tox];
        w: CARDINAL ← ABS[fromx-tox];
        IF w # 0 THEN DrawHorizontalLine[x, fromy, w];
        END;
      ENDCASE =>        BEGIN
        IF fromx > tox THEN  -- diag routine is dumb, be careful for it
          BEGIN
          iy←fromy; fromy←toy; toy←iy;
          ix←fromx; fromx←tox; tox←ix;
          END;
        DrawDiagonalLine[fromx,fromy,tox,toy];
        END;
    p ← p + SIZE[MazeDefs.XYpair];
    IF NOT two THEN RETURN[p];
    two ← FALSE;
    ENDLOOP;
  END;

XORToken: ENTRY PROCEDURE [
  hisRatId: MazeDefs.RatId, r2d2: POINTER TO MazeDefs.R2d2] =
  BEGIN OPEN MazeDefs;
  sw, sh: CARDINAL;
  [slx: ratBB.slx, sty: ratBB.sty, sw: sw, sh: sh] ← Rat[
    ratId: hisRatId, distance: r2d2[hisRatId].distance,
    view: r2d2[hisRatId].tokenId];
IF halt THEN RETURN;
  ratBB.dbca ← current.bitmap;
  ratBB.dlx ← r2d2[hisRatId].x-sw/2;
  ratBB.dty ← r2d2[hisRatId].y-sh/2;
  ratBB.dw ← sw;
  ratBB.dh ← sh;
  BitBltDefs.BITBLT[ratBB];
  RETURN;
  END;

Switch: PROCEDURE RETURNS [DCBHandle]=
  BEGIN
  temp: DCBHandle ← current;
IF halt THEN ERROR;  --!!!!!!!???????!!!********!!!!
  current ← hidden;
  topMargin.next ← current;
  hidden ← temp;
  RETURN [hidden]
  END;

SetDCBs: PROCEDURE [p: POINTER TO DCBRec, rats: POINTER] =
```

```
  BEGIN OPEN p;
  topMargin ← top;
  current ← curr;
  hidden ← hid;
--  mazeMap ← maze;
  bigMazeMap ← bigMaze;
  scoreDCB ← score;
  sbca ← rats;
  END;

View: TYPE = MazeDefs.View;

Succ: PROCEDURE [view: View] RETURNS [View] =
  MACHINE CODE BEGIN Mopcodes.zINC END;

Rat: PUBLIC PROCEDURE [ratId, distance: CARDINAL, view: View]
  RETURNS [slx, sty, sw, sh: CARDINAL] =
  BEGIN
  viewT: View ← FIRST[View];
  SELECT distance FROM
    1 => BEGIN
      slx ← 0;
      FOR viewT IN View WHILE viewT#view DO slx ← slx+64 ENDLOOP;
      RETURN [slx, 0, 64, 64];
      END;
    2 =>
      FOR slx ← 4*64, slx+32 WHILE slx # 64*5 DO
        FOR sty ← 0, sty+32 WHILE sty # 64 DO
          IF viewT=view THEN RETURN [slx, sty, 32, 32];
          viewT ← Succ[viewT]
          ENDLOOP;
        ENDLOOP;
    3 =>
      FOR slx ← 5*64, slx+24 WHILE slx # 64*5+48 DO
        FOR sty ← 0, sty+24 WHILE sty # 48 DO
          IF viewT=view THEN RETURN [slx, sty, 24, 24];
          viewT ← Succ[viewT]
          ENDLOOP;
        ENDLOOP;
    4,5 =>
      FOR sty ← 0, sty+16 WHILE sty # 64 DO
        IF viewT=view THEN RETURN [64*5+48, sty, 16, 16];
        viewT ← Succ[viewT]
        ENDLOOP;
    6,7,8 =>
      FOR slx ← 64*5, slx+9 WHILE slx # 64*5+4*9 DO
        IF viewT=view THEN RETURN [slx, 48, 9, 9];
        viewT ← Succ[viewT]
        ENDLOOP;
    IN [9..12] =>
      FOR slx ← 64*5, slx+6 WHILE slx # 64*5+4*6 DO
        IF viewT=view THEN RETURN [slx, 48+9, 6, 6];
        viewT ← Succ[viewT]
        ENDLOOP;
    IN [13..18] =>
      FOR slx ← 64*5+4*6, slx+4 WHILE slx # 64*5+4*6+4*4 DO
        IF viewT=view THEN RETURN [slx, 48+9+3, 4, 4];
        viewT ← Succ[viewT]
        ENDLOOP;
    ENDCASE =>
      FOR slx ← 64*5+4*6, slx+3 WHILE slx # 64*5+4*6+4*3 DO
        IF viewT=view THEN RETURN [slx, 48+9, 3, 3];
        viewT ← Succ[viewT]
        ENDLOOP;
  ERROR -- can't happen
  END;  -- of Rat

  FHptr: TYPE = POINTER TO FontHeader;
  Fptr: TYPE = POINTER TO Font;
  FCDptr: TYPE = POINTER TO FCD;
  FAptr: TYPE = POINTER TO FontArray;
  FontArray: TYPE = ARRAY [0..255] OF FCDptr;

  Font: TYPE = MACHINE DEPENDENT RECORD [
    header: FontHeader,
    FCDptrs: FontArray,          -- array of self-relative pointers to
```

```
    -- FCD's.  Indexed by char value.
    -- font pointer points hear!
  extFCDptrs: FontArray       -- array of self-relative pointers to
    -- FCD's for extentions.  As large an
    -- array as needed.
  ];

FontHeader: TYPE = MACHINE DEPENDENT RECORD
  [
  maxHeight: CARDINAL,        -- height of tallest char in font (scan lines)
  variableWidth: BOOLEAN,     -- IF TRUE, proportionally spaced font
  blank:       [0..177B],     -- not used
  maxWidth: [0..377B] -- width of widest char in font (raster units).
  ];

FCD: TYPE = MACHINE DEPENDENT RECORD [
  widthORext: [0..77777B],    -- width or extention index
  hasNoExtension: BOOLEAN,    -- TRUE=> no ext.;prevfield=width
  height: [0..377B],          -- # scan lines to skip for char
  displacement: [0..377B]     -- displacement back to char bitmap
  ];

PaintChar: PROCEDURE
  [char: CHARACTER, bmState: POINTER TO FontDefs.BitmapState] =
  BEGIN OPEN BitBltDefs, bmState;
  bba: ARRAY [0..SIZE[BBTable]] OF UNSPECIFIED;
  bbt: BBptr = LOOPHOLE[BASE[bba] + LOOPHOLE[BASE[bba],CARDINAL] MOD 2];
  cw: FCDptr;
  fontdesc: FAptr = @font.FCDptrs;
  code: CARDINAL ← LOOPHOLE[char];
  bbt↑ ← [
    pad: 0,
    sourcealt: FALSE,
    destalt: FALSE,       sourcetype: block,
    function: paint,
    unused:,
    dbca: origin,
    dbmr: wordsPerLine,
    dlx: x,
    dty:,
    dw: 16,
    dh:,
    sbca:,
    sbmr: 1,
    slx: 0,
    sty: 0,
    gray0:, gray1:, gray2:, gray3:];
  DO
    cw ← LOOPHOLE[fontdesc[code]+LOOPHOLE[fontdesc,CARDINAL]+code];
    bbt.dty ← y + cw.height;
    bbt.dh ← cw.displacement;
    bbt.sbca ← cw - (bbt.dh ← cw.displacement);
    IF cw.hasNoExtension THEN
      BEGIN
      x ← x + (bbt.dw ← cw.widthORext);
      BITBLT[bbt];
      EXIT
      END
    ELSE
      BEGIN
      BITBLT[bbt];
      bbt.dlx ← x ← x + 16;
      END;
    code ← cw.widthORext;
    ENDLOOP;
  RETURN
  END;

WriteScoreString: PUBLIC PROCEDURE [rat: MazeDefs.RatId, s: STRING] =
  BEGIN
  i: CARDINAL;
  bmState: FontDefs.BitmapState ← [
    x: 0, y: rat*12, wordsPerLine: scoreDCB.width, origin: scoreDCB.bitmap];
  FOR i IN [0..s.length) DO PaintChar[s[i], @bmState]; ENDLOOP;
  RETURN
  END;
```

```
  ClearScoreLine: PUBLIC PROCEDURE [rat: MazeDefs.RatId] =
    BEGIN
    ChangeScoreLine[gray: 0B, dty: rat*12];
    RETURN
    END;

  InvertScoreLine: PUBLIC PROCEDURE [rat: MazeDefs.RatId] =
    BEGIN
    ChangeScoreLine[gray: 177777B, dty: rat*12];
    RETURN
    END;

  ChangeScoreLine: PROCEDURE [gray, dty: CARDINAL] =
    BEGIN OPEN BitBltDefs;
    bbTable: ARRAY [0..SIZE[BBTable]] OF UNSPECIFIED;
    bbptr: BBptr ← EVEN[@bbTable];
    bbptr↑ ← [
      pad: 0, sourcealt: FALSE, destalt: FALSE, sourcetype: gray,
      function: IF gray = 0 THEN replace ELSE invert, unused: 0,
      dbca: scoreDCB.bitmap, dbmr: 12,
      dlx: 0, dty: dty, dw: 12*16, dh: 12, sbca: NIL, sbmr: 0, slx: 0, sty: 0,
      gray0: gray, gray1: gray, gray2: gray, gray3: gray];
    BITBLT[bbptr];
    RETURN
    END;

END...
```

```
-- MazeDisplayDefs.mesa  Edited by Gobbel on June 22, 1979  3:50 PM

DIRECTORY
  AltoDisplay: FROM "altodisplay" USING [DCBHandle],
  BitBltDefs: FROM "bitbltdefs" USING [BBptr],
  InlineDefs: FROM "inlinedefs" USING [COPY],
  MazeDefs: FROM "mazedefs" USING [Direction, R2d2, RatId, XYpair, RatObject];

MazeDisplayDefs: DEFINITIONS IMPORTS InlineDefs =
  BEGIN OPEN MazeDefs;

  WindowWordsPerLine: CARDINAL = 26;
  WindowLines: CARDINAL = 400;

  pRat: TYPE = POINTER TO ARRAY RatId OF RatObject;
  DCBHandle: TYPE = AltoDisplay.DCBHandle;

  DCBRec: TYPE = RECORD [top, curr, hid, maze, bigMaze, score: DCBHandle];

  DoubleWord: TYPE = RECORD[w1, w2: WORD];

  MakeBigMaze: PROCEDURE;
  MakeSmallMaze: PROCEDURE;

  ClearPosition: PROCEDURE [ratId: MazeDefs.RatId, xClear, yClear: CARDINAL];
  ShowPosition: PROCEDURE [xloc, yloc: CARDINAL, invincible: BOOLEAN,
                           tdir: Direction];
  SetMyRatId: PROCEDURE [ratId: RatId];
  ShowAllPositions: PROCEDURE [rats: pRat];
  DisplayOthersPosition: PROCEDURE [which, xloc, yloc: CARDINAL, tdir: Direction];
  ExitPlayer: PROCEDURE [ratId: RatId];
  AddNewPlayer: PROCEDURE [ratId: RatId, xloc, yloc: CARDINAL, tdir: Direction];
  SetDCBs: PROCEDURE [p: POINTER TO DCBRec, rats: POINTER];

  PlotLine: PROCEDURE [p: POINTER TO XYpair, two: BOOLEAN]
    RETURNS [POINTER TO XYpair];
  Switch: PROCEDURE RETURNS [DCBHandle];
  XORToken: PROCEDURE [hisRatId: RatId, r2d2: POINTER TO R2d2];
  WriteScoreString: PROCEDURE [rat: RatId, s: STRING];
  ClearScoreLine: PROCEDURE [rat: RatId];
  InvertScoreLine: PROCEDURE [rat: RatId];
  InvertTop: PROCEDURE;
  InvertHidden: PROCEDURE = INLINE
    BEGIN
    hidden.background ← IF hidden.background=white THEN black ELSE white
    END;

  ratBB, lineBB, diagBB: BitBltDefs.BBptr;
  font: POINTER;
  hidden: DCBHandle;
  leftBBpattern, rightBBpattern: ARRAY [0..32) OF DoubleWord;

  MazeDisplay: PROGRAM;

  InitWindow: PROCEDURE [dcb: DCBHandle] = INLINE
    BEGIN OPEN InlineDefs;
    p: POINTER ← dcb.bitmap;
    p↑ ← 177777B;
    COPY[from: p, to: p+1, nwords: WindowWordsPerLine-2];
    (p+WindowWordsPerLine-1)↑ ← 0;
    COPY[from: p, to: p+WindowWordsPerLine*(WindowLines-1),
      nwords: WindowWordsPerLine];
    (p ← p + WindowWordsPerLine)↑ ← 100000B;
    (p+1)↑ ← 0B;
    COPY[from: p+1, to: p+2, nwords: WindowWordsPerLine-2];
    (p + WindowWordsPerLine-2)↑ ← 1B;
    COPY[from: p, to: p+WindowWordsPerLine,
      nwords: WindowWordsPerLine*(WindowLines-3)];
    END;

  END...
```

```
-- MazeInit.mesa edit by Bruce on June 25, 1979  7:35 PM

DIRECTORY
  AltoDisplay: FROM "altodisplay" USING [DCB,DCBHandle, DCBchainHead, DCBnil],
  BitBltDefs: FROM "bitbltdefs" USING [BBptr, BBTable],
  BitOps: FROM "BitOps" USING [Set, BD, Descriptor],
  DisplayDefs: FROM "displaydefs" USING [DisplayOff, StopCursor],
  FrameDefs: FROM "framedefs" USING [
    GlobalFrame, MakeCodeResident, SwapInCode, UnNew],
  ImageDefs: FROM "imagedefs" USING [MakeImage],
  InlineDefs: FROM "inlinedefs" USING [BITAND, BITOR, BITSHIFT],
  IODefs: FROM "iodefs" USING [
    CR, ReadID, Rubout, SP, WriteChar, WriteString, WriteLine],
  KeyDefs: FROM "keydefs" USING [Keys],
  MazeNewDefs: FROM "mazeNewdefs" USING [Password],
  MazeDefs: FROM "mazedefs" USING [AqRatNew, DispatchKeys, mazeSocket,
    MaxRats, NewPosition, NewScoreCard, RatId, RatName, RatNew, ratNew, Rats,
    RatStatus, ratStatus, ReadKeys, ReadRats, Tables, TheMaze, TokenVisible,
    ShowView, RatDoctor, ratSurvey],
  MazeDisplayDefs: FROM "mazedisplaydefs" USING [
    font, lineBB, MazeDisplay, ratBB, SetDCBs, ShowPosition,
    DCBRec, WindowWordsPerLine, leftBBpattern, rightBBpattern, diagBB, SetMyRatId],
  MazeWar: FROM "mazewar" USING [bvSendAllStatus, bvSendDead, bvSendGoing,
    bvSendKill, bvSendLocAll, bvSendOneStatus, bvSendQuery, bvSendAlive,
    dcb, keyStroke, Maze, myAddr, myRatId, ps, PupOutput, ratcb, score,
    table, tdir, wakeup, xloc, yloc, Wait, duke, randomVector, VectorSize],
  MiscDefs: FROM "miscdefs" USING [Zero],
  MMOps: FROM "mmops" USING [MMFont],
  ProcessDefs: FROM "processdefs" USING [CV, Detach, DIW,
    InterruptLevel, SetPriority],
  PupDefs: FROM "pupdefs" USING [AdjustBufferParms, GetFreePupBuffer,
    PupAddress, PupBuffer, GetPupAddress, PupNameTrouble,
    PupPackageMake, PupSocket, PupSocketDestroy, PupSocketMake,
    ReturnFreePupBuffer, SecondsToTocks, SetPupContentsWords,
    veryLongWait, PupAddressLookup, AppendPupAddress, PupRouterBroadcastThis,
    PupRouterSendThis, PupNameLookup],
  PupTypes: FROM "puptypes" USING [fillInNetID, allHosts, fillInHostID,
    fillInSocketID],
  StreamDefs: FROM "streamdefs" USING [
    GetDefaultDisplayStream, GetDefaultKey, StreamHandle],
  StringDefs: FROM "stringdefs" USING [StringBoundsFault],
  SystemDefs: FROM "systemdefs" USING [AllocateHeapNode, AllocateSegment];

MazeInit: PROGRAM
  IMPORTS FrameDefs, InlineDefs, MazeDefs, MazeDisplayDefs,
    MiscDefs, PupDefs, ProcessDefs, DisplayDefs,
    StreamDefs, StringDefs, SystemDefs, M: MazeWar, MMOps, IODefs, BitOps, ImageDefs
  EXPORTS MazeDefs
  SHARES MazeWar =
BEGIN OPEN MazeDefs, AltoDisplay;

m: POINTER TO FRAME[MazeWar] ← M;
rats:   POINTER;
ratBBTable, lineBBTable, diagBBTable: ARRAY [0..SIZE[BitBltDefs.BBTable]] OF UNSPECIFIED;
ratName: STRING ← [20];
dukeName: STRING ← [20];

GetMaze: PROCEDURE =
  BEGIN
  FrameDefs.SwapInCode[LOOPHOLE[TheMaze]];
  m.Maze ← FrameDefs.GlobalFrame[TheMaze].code.shortbase;
  END;

GetTable: PROCEDURE =
  BEGIN
  FrameDefs.SwapInCode[LOOPHOLE[Tables]];
  m.table ← FrameDefs.GlobalFrame[Tables].code.shortbase;
  END;

GetRats: PROCEDURE =
  BEGIN
  FrameDefs.SwapInCode[LOOPHOLE[Rats]];
  rats ← FrameDefs.GlobalFrame[Rats].code.shortbase;
  END;

CreateDCB: PROCEDURE [wordsPerLine, height, indenting: CARDINAL]
```

```
    RETURNS[dcb: DCBHandle] =
  BEGIN OPEN AltoDisplay;
  bitmap: POINTER ← IF wordsPerLine = 0 THEN NIL
    ELSE SystemDefs.AllocateSegment[wordsPerLine*height];
  dcb ← EVEN[SystemDefs.AllocateHeapNode[SIZE[DCB]+1]];
  dcb↑ ← [next: DCBnil, resolution: high, background: white,
    indenting: indenting, width: wordsPerLine, height: height/2,
    bitmap: bitmap];
  IF bitmap # NIL THEN MiscDefs.Zero[bitmap, wordsPerLine*height];
  END;

InitDisplay: PROCEDURE =
  BEGIN OPEN MazeDisplayDefs;
  dcbs: DCBRec;
  bbptr: BitBltDefs.BBptr ← EVEN[@ratBBTable];
  temp: DCBHandle;
  p: POINTER;
  i: INTEGER;
  desc: BitOps.Descriptor;
  curX: POINTER TO INTEGER = LOOPHOLE[426B];
  curX↑ ← -1;
  START MazeDisplay;
  FrameDefs.SwapInCode[LOOPHOLE[MMOps.MMFont]];
  font ← FrameDefs.GlobalFrame[MMOps.MMFont].code.shortbase;
  dcbs.top ← CreateDCB[wordsPerLine: 0, height: 50, indenting: 0];
  temp ← CreateDCB[wordsPerLine: 0, height: 20, indenting: 0];
  dcbs.curr ← CreateDCB[wordsPerLine: 26, height: 400, indenting: 6];
  dcbs.hid ← CreateDCB[wordsPerLine: 26, height: 400, indenting: 6];
--  dcbs.maze ← CreateDCB[wordsPerLine: 16, height: 128, indenting: 10];
  dcbs.maze ← NIL;
  dcbs.bigMaze ← CreateDCB[wordsPerLine: 32, height: 256, indenting: 2];
  dcbs.score ← CreateDCB[wordsPerLine: 12, height: 12*MaxRats, indenting: 11];
  SetDCBs[@dcbs, rats];
  dcbs.top.next ← dcbs.curr;
  dcbs.curr.next ← temp; dcbs.hid.next ← temp;
--  temp.next ← dcbs.maze;
--  dcbs.maze.next ← dcbs.score;
  temp.next ← dcbs.score;
--  InitMaze[dcbs.maze];
  BigInitMaze[dcbs.bigMaze];
  --this doesn't seem to work????
  MiscDefs.Zero[dcbs.score.bitmap, dcbs.score.width*dcbs.score.height*2];
  m.dcb ← dcbs.hid;
  AltoDisplay.DCBchainHead↑ ← dcbs.top;
  bbptr↑ ← [
    pad: 0, sourcealt: FALSE, destalt: FALSE, sourcetype: block,
    function: invert, unused: 0, dbca: NIL, dbmr: WindowWordsPerLine,
    dlx: 0, dty: 0, dw: 0, dh: 0, sbca: NIL, sbmr: 6*4, slx: 0, sty: 0,
    gray0:, gray1:, gray2:, gray3:];
  bbptr.sbca ← rats;
  MazeDisplayDefs.ratBB ← bbptr;
  bbptr ← EVEN[@lineBBTable];
  bbptr↑ ← [
    pad: 0, sourcealt: FALSE, destalt: FALSE, sourcetype: gray,
    function: paint, unused: 0, dbca: NIL, dbmr: WindowWordsPerLine,
    dlx: 0, dty: 0, dw: 0, dh: 0, sbca: NIL, sbmr: 0, slx: 0, sty: 0,
    gray0:177777B, gray1:177777B, gray2:177777B, gray3:177777B];
  MazeDisplayDefs.lineBB ← bbptr;
  bbptr ← EVEN[@diagBBTable];
  bbptr↑ ← [
    pad: 0, sourcealt: FALSE, destalt: FALSE, sourcetype: block,
    function: paint, unused: 0, dbca: NIL, dbmr: WindowWordsPerLine,
    dlx: 0, dty: 0, dw: 0, dh: 0, sbca: NIL, sbmr: 2, slx: 0, sty: 0,
    gray0: , gray1: , gray2: , gray3: ];
  MazeDisplayDefs.diagBB ← bbptr;
  p ← @MazeDisplayDefs.leftBBpattern[0];
  desc ← BitOps.BD[0];
  FOR i IN [0..32) DO
    BitOps.Set[1, p, desc];
    p ← p+2;
    desc ← LOOPHOLE[LOOPHOLE[desc, CARDINAL]+16];
    ENDLOOP;
  p ← @MazeDisplayDefs.rightBBpattern[0];
  desc ← BitOps.BD[31];
  FOR i IN [0..32) DO
    BitOps.Set[1, p, desc];
```

```
    p ← p+2;
    desc ← LOOPHOLE[LOOPHOLE[desc, CARDINAL]-16];
    ENDLOOP;
  END;

--InitMaze: PROCEDURE [mazeDCB: DCBHandle] =
--  BEGIN
--  i, line, j, index: CARDINAL;
--  maze: POINTER TO PACKED ARRAY OF [0..377B] ← mazeDCB.bitmap;
--  MiscDefs.Zero[maze, mazeDCB.width*mazeDCB.height*2];
--  FOR i IN [0..16) DO
--    line ← i*32*8;
--    FOR j IN [0..32) DO
--      IF m.Maze[i][j] THEN
--      BEGIN
--      index ← line + j;
--      THROUGH [0..8) DO maze[index] ← 377B; index ← index + 32; ENDLOOP;
--      END;
--      ENDLOOP;
--    ENDLOOP;
--  END;
--
BigInitMaze: PROCEDURE [mazeDCB: DCBHandle] =
  BEGIN
  i, line, j, index: CARDINAL;
  maze: POINTER TO PACKED ARRAY OF WORD ← mazeDCB.bitmap;
  MiscDefs.Zero[maze, mazeDCB.width*mazeDCB.height];
  FOR i IN [0..16) DO
    line ← i*32*16;
    FOR j IN [0..32) DO
      IF m.Maze[i][j] THEN
        BEGIN
        index ← line + j;
        THROUGH [0..16) DO maze[index] ← 177777B; index ← index + 32; ENDLOOP;
        END;
      ENDLOOP;
    ENDLOOP;
  END;

EVEN: PROCEDURE [v: UNSPECIFIED] RETURNS [UNSPECIFIED] =
  BEGIN RETURN[v+InlineDefs.BITAND[v,1]] END;

InitKeyboard: PROCEDURE RETURNS[ph: PROCESS] =
  BEGIN OPEN ProcessDefs;
  KeyboardLevel: InterruptLevel = 7;
  KeyboardBit: WORD = InlineDefs.BITSHIFT[1,KeyboardLevel];

  SetPriority[6];  -- Increase my priority so that the fork
  ph ← FORK ReadKeys;  -- goes off at the new priority
  SetPriority[1];  -- reset my original priority
  CV[KeyboardLevel] ← @m.wakeup;  -- set the cv for the naked notify
  -- enable interrupts every display vertical trace
  DIW↑ ← InlineDefs.BITOR[DIW↑,KeyboardBit];
  RETURN[ph];
  END;

InitRandom: PROCEDURE =
  BEGIN
  p: POINTER TO CARDINAL = LOOPHOLE[430B];
  i: CARDINAL;
  t: CARDINAL = p↑;
  FOR i IN [0..m.VectorSize) DO
    m.randomVector[i] ← t + m.randomVector[i] ENDLOOP;
  END;

GetNames: PROCEDURE =
  BEGIN OPEN IODefs;
  keys: StreamDefs.StreamHandle ← StreamDefs.GetDefaultKey[];
  WriteLine["MazeWar modified version by Brad Myers"L];
  WriteLine["Holding 'E' down shows big maze"L];
  WriteString["Your Name: "L];
  ReadID[ratName ! StringDefs.StringBoundsFault => CONTINUE];
  WriteChar[CR];
  WriteString["Duke Host (CR for any game): "L];
  ReadID[dukeName ! StringDefs.StringBoundsFault => CONTINUE];
  WriteChar[CR];
```

```
  keys.destroy[keys];
  m.Wait[10]; -- make sure keyboard process dies
  DisplayDefs.StopCursor[];
  DisplayDefs.DisplayOff[white];
  FrameDefs.UnNew[FrameDefs.GlobalFrame[WriteString]];
  FrameDefs.UnNew[FrameDefs.GlobalFrame[StreamDefs.GetDefaultKey]];
  FrameDefs.UnNew[FrameDefs.GlobalFrame[StreamDefs.GetDefaultDisplayStream]];
  RETURN
  END;

CheckRatName: PROCEDURE =
  BEGIN OPEN PupDefs;
  i: CARDINAL;
  addr: PupAddress;
  PupNameLookup[@addr,"ME"L];
  addr.socket ← mazeSocket;
  m.myAddr ← addr;
  IF ratName.length # 0 THEN RETURN;
  PupAddressLookup[addr, ratName ! PupNameTrouble =>
    BEGIN AppendPupAddress[ratName, @addr]; CONTINUE END];
  FOR i DECREASING IN [0..ratName.length] DO
    IF ratName[i] # '+ AND ratName[i] # '# THEN LOOP;
    ratName.length ← i;
    EXIT;
    ENDLOOP;
  RETURN;
  END;

RatNameFromString: PROCEDURE [name: STRING, pt: POINTER TO RatName] =
  BEGIN
  i: CARDINAL;
  FOR i IN [0..name.length) DO pt[i] ← name[i]; ENDLOOP;
  FOR i IN [name.length..20) DO pt[i] ← IODefs.SP; ENDLOOP;
  END;

PupInit: PROCEDURE =
  BEGIN OPEN PupDefs;
  ratId: RatId;
  AdjustBufferParms[10, 266];       -- (20) buffers, 266 bytes each now 10
  PupPackageMake[];
  CheckRatName[];
  IF NOT JoinGame[] THEN StartGame[];
  FOR ratId IN RatId DO    -- check for visible rats for the scorecard
    TokenVisible[ratId];
    ENDLOOP;
  NewScoreCard[];              -- Init the score window
  m.ps ← PupSocketMake[mazeSocket, m.myAddr, veryLongWait];
  ProcessDefs.Detach[FORK ReadRats[]];  -- Fork the read pup process
  m.PupOutput ← FALSE;
  m.bvSendLocAll ← FALSE;
  m.bvSendOneStatus ← FALSE;
  m.bvSendAllStatus ← FALSE;
  m.bvSendKill ← FALSE;
  m.bvSendDead ← FALSE;
  m.bvSendGoing ← FALSE;
  m.bvSendQuery ← FALSE;
  m.bvSendAlive ← FALSE;
  m.keyStroke ← TRUE;  -- Can now display screen for first time
  END;

SetUpSurvey: PROCEDURE [b: PupDefs.PupBuffer] =
  BEGIN
  ratsurvey: RatNew;
  b.pupType ← ratSurvey;
  b.source ← m.myAddr;
  PupDefs.SetPupContentsWords[b, SIZE[AqRatNew]];
  ratsurvey ← LOOPHOLE[@b.pupBody];
  ratsurvey↑ ← [MazeNewDefs.Password,,,,,];
  END;

NullAddress: PupDefs.PupAddress = [net: PupTypes.fillInNetID,
  host: PupTypes.fillInHostID, socket: PupTypes.fillInSocketID];

FindDuke: PROCEDURE RETURNS [addr: PupDefs.PupAddress] =
  BEGIN OPEN PupDefs;
  rmtAddr: PupAddress ←
```

```
    [net: PupTypes.fillInNetID, host: PupTypes.allHosts, socket: mazeSocket];
  ps: PupSocket ← PupSocketMake[mazeSocket, rmtAddr, SecondsToTocks[5]];
  b: PupBuffer ← GetFreePupBuffer[];
  maxAnswers: CARDINAL = 10;
  answers: ARRAY [0..maxAnswers) OF PupBuffer;
  i,j,cnt: CARDINAL ← 0;
  SetUpSurvey[b];
  b.dest ←
    [net: PupTypes.fillInNetID, host: PupTypes.allHosts, socket: mazeSocket];
  PupRouterBroadcastThis[b];
  UNTIL (b ← ps.get[]) = NIL DO
    IF b.pupType # ratStatus OR cnt = maxAnswers THEN
      BEGIN ReturnFreePupBuffer[b]; LOOP; END;
    IF ~Duke[b] THEN
      BEGIN
      rs: RatStatus ← LOOPHOLE[@b.pupBody];
      SetUpSurvey[b];
      b.dest ← rs.rats[rs.dukeRat].addr;
      [] ← PupRouterSendThis[b];
      LOOP;
      END;
    answers[cnt] ← b;
    IF FreeSlot[b] THEN cnt ← cnt + 1 ELSE ReturnFreePupBuffer[b];
    ENDLOOP;
  PupSocketDestroy[ps];
  FOR i IN [0..cnt) DO
    IF ~Duke[answers[i]] THEN BEGIN ReturnFreePupBuffer[answers[i]]; LOOP END;
    addr ← answers[i].source;
    FOR j IN [i..cnt) DO ReturnFreePupBuffer[answers[j]] ENDLOOP;
    RETURN
    ENDLOOP;
  RETURN[NullAddress]
  END;

Duke: PROCEDURE [b: PupDefs.PupBuffer] RETURNS [BOOLEAN] =
  BEGIN
  test: RatStatus ← LOOPHOLE[@b.pupBody];
  RETURN[test.rats[test.dukeRat].addr = b.source]
  END;

FreeSlot: PROCEDURE [b: PupDefs.PupBuffer] RETURNS [BOOLEAN] =
  BEGIN
  test: RatStatus ← LOOPHOLE[@b.pupBody];
  id: RatId;
  FOR id IN RatId DO
    IF ~test.rats[id].playing THEN RETURN[TRUE];
    ENDLOOP;
  RETURN[FALSE];
  END;

Join: PROCEDURE [addr: PupDefs.PupAddress] RETURNS [okToPlay: BOOLEAN] =
  BEGIN OPEN PupDefs;
  ps: PupSocket ← PupSocketMake[mazeSocket, addr, SecondsToTocks[5]];
  b: PupBuffer ← GetFreePupBuffer[];
  ratnew: RatNew;
  id: RatId;
  status: RatStatus;
  okToPlay ← FALSE;
  b.pupType ← ratNew;
  b.dest ← addr;
  b.source ← m.myAddr;
  SetPupContentsWords[b, SIZE[AqRatNew]];
  ratnew ← LOOPHOLE[@b.pupBody];
  ratnew↑ ← [MazeNewDefs.Password, m.xloc, m.yloc, m.tdir, m.myAddr, ];
  RatNameFromString[ratName, @ratnew.name];
  [] ← PupRouterSendThis[b];
  b ← ps.get[];
  IF b = NIL OR b.pupType # ratStatus THEN
    BEGIN
    IF b # NIL THEN ReturnFreePupBuffer[b];
    PupSocketDestroy[ps];
    RETURN
    END;
  status ← LOOPHOLE[@b.pupBody];
  FOR id IN RatId DO
    IF status.rats[id].playing AND status.rats[id].addr = m.myAddr THEN
```

```
      BEGIN m.myRatId ← id; MazeDisplayDefs.SetMyRatId[id]; EXIT END;
    REPEAT FINISHED => RETURN
    ENDLOOP;
  m.ratcb ← status↑;
  okToPlay ← TRUE;
  ReturnFreePupBuffer[b];
  PupSocketDestroy[ps];
  END;

JoinGame: PROCEDURE RETURNS [okToPlay: BOOLEAN] =
  BEGIN OPEN PupDefs;
  rmtAddr: PupAddress;
  duke: BOOLEAN ← dukeName.length # 0;
  IF duke THEN GetPupAddress[@rmtAddr, dukeName !
    PupNameTrouble => BEGIN duke ← FALSE; CONTINUE END];
  rmtAddr.socket ← mazeSocket;
  IF ~duke THEN rmtAddr ← FindDuke[];
  IF rmtAddr = NullAddress THEN RETURN [FALSE];
  RETURN[Join[rmtAddr]]
  END;

StartGame: PROCEDURE =
  BEGIN
  i: CARDINAL;
  m.myRatId ← 0;          -- Give me first entry in the table
  MazeDisplayDefs.SetMyRatId [0];
  m.ratcb.dukeRat ← m.myRatId;      -- Set myself up as the duke
  m.ratcb.rats[m.myRatId] ←
    [TRUE, m.xloc, m.yloc, m.tdir, m.score, m.myAddr, ];
  RatNameFromString[ratName, @m.ratcb.rats[m.myRatId].name];
  FOR i IN [1..MaxRats) DO
    m.ratcb.rats[i].playing ← FALSE;
    ENDLOOP;
  m.duke ← TRUE;
  END;

RV: ARRAY [0..m.VectorSize) OF WORD = [
  031575B, 055455B, 147160B, 176745B, 173126B, 117426B, 033612B, 130620B,
  054013B, 167672B, 070252B, 033100B, 015700B, 113523B, 170465B, 024344B,
  175535B, 137325B, 126211B, 010207B, 173547B, 016071B, 056622B, 014433B,
  113225B, 047553B, 103025B, 110174B, 000125B, 173304B, 076700B, 104042B,
  135030B, 126234B, 175154B, 140123B, 167542B, 000405B, 035464B, 166537B,
  050260B, 167655B, 123615B, 175164B, 172206B, 140365B, 074606B, 075656B,
  176163B, 030027B, 022102B, 040051B, 154630B, 017144B, 073372B];

FrameDefs.MakeCodeResident[LOOPHOLE[TheMaze]];

GetMaze[];
GetTable[];  -- Read vector table from file (30 views, 12 lines per view)
GetRats[];  -- Read rat bitmaps, init ratpack

IF KeyDefs.Keys.Spare2 = down THEN ImageDefs.MakeImage["MazeWar.image"L];

m.randomVector ← RV;
InitRandom[];
GetNames[ ! IODefs.Rubout => RETRY];
InitDisplay[];
NewPosition[];  -- Get random positon in the maze
MazeDisplayDefs.ShowPosition[m.xloc, m.yloc, FALSE, m.tdir];
MazeDefs.ShowView[m.xloc, m.yloc, m.tdir];
PupInit[];
ProcessDefs.Detach[InitKeyboard[]];
ProcessDefs.Detach[FORK MazeDefs.DispatchKeys[]];
ProcessDefs.Detach[FORK MazeDefs.RatDoctor[]];

--PupInit[];

END...
```

```
--MAZE WATCHER by: Brad A. Myers  Last change: July 26, 1979  9:04 PM

DIRECTORY
  MazeDefs: FROM "mazedefs",
  PupTypes: FROM "puptypes" USING [PupType];

MazeNewDefs: DEFINITIONS =
BEGIN OPEN MazeDefs;

Password: CARDINAL = 176543B;

StayAlive: PROCEDURE;
RatWatching: PROCEDURE [ hisRatId: RatId];

ratTalk: PupTypes.PupType;
END.
```

```
--HACKED VERSION by Brad A. Myers!! October 23, 1979  10:35 PM
-- MazeWar.mesa edit by Bruce on June 25, 1979  7:09 PM

DIRECTORY
  AltoDisplay: FROM "altodisplay" USING [DCBchainHead, DCBnil],
  FrameDefs: FROM "framedefs" USING [UnNew],
  ImageDefs: FROM "imagedefs" USING [StopMesa],
  InlineDefs: FROM "inlinedefs",
  KeyDefs: FROM "keydefs" USING [KeyBits, Keys],
  MazeDefs: FROM "mazedefs",
  MazeNewDefs: FROM "mazeNewdefs" USING [Password],
  MazeDisplayDefs: FROM "mazedisplaydefs" USING [
    ClearScoreLine, DCBHandle, InitWindow, InvertScoreLine,
    PlotLine, ShowPosition, InvertTop, DisplayOthersPosition, ExitPlayer, AddNewPlayer,
    ShowAllPositions, MakeBigMaze, MakeSmallMaze,
    Switch, WriteScoreString,
    XORToken, InvertHidden],
  ProcessDefs: FROM "processdefs" USING [
    Detach, InitializeCondition, MsecToTicks, SetPriority, Yield],
  PupDefs: FROM "pupdefs" USING [GetFreePupBuffer, PupAddress, PupBuffer,
    PupRouterSendThis, PupSocket, ReturnFreePupBuffer, SetPupContentsWords],
  StringDefs: FROM "stringdefs" USING [AppendDecimal];

MazeWar: MONITOR
  IMPORTS ImageDefs, InlineDefs, MazeDefs, MazeDisplayDefs,
    PupDefs, ProcessDefs, StringDefs, FrameDefs
  EXPORTS MazeDefs =
PUBLIC BEGIN OPEN MazeDefs, MazeDisplayDefs, PupDefs;

table: POINTER TO ARRAY [0..360) OF XYpair;  -- 30 frames, 12 XYpairs each

xloc: CARDINAL;
yloc: CARDINAL; -- Where am I?
tdir: Direction; -- Which way am I looking?
dcb: DCBHandle;
Maze: MazeType;

-- Delta Arrays contain the deltas to map the current x,y,direction into the neighboring cubes coordi
**tes
L1Delta: ARRAY Direction OF XY ← [ [-1,0], [1,0], [0,1], [0,-1] ];
L2Delta: ARRAY Direction OF XY ← [ [-1,1], [1,-1], [1,1], [-1,-1] ];
C2Delta: ARRAY Direction OF XY ← [ [0,1], [0,-1], [1,0], [-1,0] ];
R1Delta: ARRAY Direction OF XY ← [ [1,0], [-1,0], [0,-1], [0,1] ];
R2Delta: ARRAY Direction OF XY ← [ [1,1], [-1,-1], [1,-1], [-1,1] ];
Edge1, Edge2, Edge3, Edge4, Edge5, Edge6, Edge7: BOOLEAN;
edge3Lines, edge7Lines: ARRAY [0..1] OF XYpair;
prevEdge3, prevEdge7: BOOLEAN;

wakeup: CONDITION;              -- Posted by the DIW u-code process
keyStroke: BOOLEAN ← FALSE;     -- new keys to process

peeking: BOOLEAN ← FALSE;       -- peeking left or right in process
xPeek, yPeek: CARDINAL;         -- x,y,dir for peek display
dirPeek: Direction;

ctBreaths: CARDINAL ← 0;        -- Number of shots current fired at me (# breaths being held)

-- Global Data For Pup Stuff

myAddr: PupAddress;     -- My pup network/host address (hmm)
myRatId: RatId;       -- My index into the ratcb
killerRatId: RatId;       -- My killer's index into the ratcb
ratcb: RatCb;       -- my copy of the rest of the world!
ps: PupSocket;     -- the main socket
PupOutput: BOOLEAN;     -- Set to wake up pup writer
bvSendLocAll: BOOLEAN;       -- Set if all are to get my new location
bvSendOneStatus: BOOLEAN;     -- Set if to send my ratcb to new member
bvSendAllStatus: BOOLEAN;     -- Set if to send my ratcb to all players
bvSendDead: BOOLEAN;       -- Send I'm dead msg to my killer
bvSendKill: BOOLEAN;       -- Send off a fire msg
bvSendGoing: BOOLEAN;       -- Send off a I'm going msg
bvSendQuery: BOOLEAN;       -- Send off one or more query msgs
bvSendAlive: BOOLEAN;       -- Send off response to query msg
newAddr: PupAddress;     -- Address of the new dude for SendOneStatus
score: Score ← 0;       -- My total score so far
duke: BOOLEAN ← FALSE;
```

```
ratHealth: RatHealth;  -- For the rat doctor

Invincible: BOOLEAN ← FALSE;            --Can I be killed?
ratId: RatId;           --for initialization at end (BUG FIX)
-- Global Data For Tokens

r2d2: R2d2;                     -- The array

-- Array [my direction] by [his direction]
-- 0=left arrow, 1=right arrow, 2=up arrow, 3=down arrow
relativeTokens: RelativeTokens ← [
        [ rear, front, right, left],            -- Me north, him [n, s, e, w]
        [ front, rear, left, right],            -- Me south, him [n, s, e, w]
        [ left, right, rear, front],    -- Me east, him [n, s, e, w]
        [ right, left, front, rear] ];  -- Me west, him [n, s, e, w]

-- Global Data For Random number generator

VectorSize: CARDINAL = 55;
i1: CARDINAL ← 0;
i2: CARDINAL ← 24;

randomVector: ARRAY [0..VectorSize) OF WORD;

Random: PROCEDURE [limit: CARDINAL] RETURNS [ret: WORD] =
  BEGIN
  ret←randomVector[i1] ← randomVector[i1] + randomVector[i2];
  IF (i1 ← i1+1) >= VectorSize THEN i1 ← 0;
  IF (i2 ← i2+1) >= VectorSize THEN i2 ← 0;
  RETURN[ret MOD limit];
  END;

-- Calculate which lines to display and display them!
Hidden: PROCEDURE [x,y: CARDINAL, dir: Direction, p: POINTER TO XYpair]
  RETURNS [POINTER TO XYpair] = INLINE
  BEGIN

  -- Local variables and tables
  L1x, L1y, L2x, L2y: CARDINAL;
  R1x, R1y, R2x, R2y: CARDINAL;
  C2x, C2y:           CARDINAL;

  -- 1st calculate the coordinates of the neighboring cubes

  L1x ← x + L1Delta[dir].xcor;          -- Find left cube
  L1y ← y + L1Delta[dir].ycor;
  L2x ← x + L2Delta[dir].xcor;          -- Find left forward cube
  L2y ← y + L2Delta[dir].ycor;
  R1x ← x + R1Delta[dir].xcor;          -- Find right cube
  R1y ← y + R1Delta[dir].ycor;
  R2x ← x + R2Delta[dir].xcor;          -- Find right forward cube
  R2y ← y + R2Delta[dir].ycor;
  C2x ← x + C2Delta[dir].xcor;          -- Find forward cube
  C2y ← y + C2Delta[dir].ycor;

  -- Next calculate which of the 7 possible cube edges are visible

  Edge2 ← Maze[C2x][C2y];                       -- C2
  Edge3 ← Maze[L1x][L1y];                       -- L1
  Edge4 ← NOT Edge3;                            -- \ L1

  Edge7 ← Maze[R1x][R1y];                       -- R1
  Edge6 ← NOT Edge7;                            -- \ R1

  Edge1 ← Edge3 AND (Edge2 OR NOT(Maze[L2x][L2y]))
    OR (NOT(Edge2) AND Edge4);
  Edge5 ← Edge7 AND (Edge2 OR NOT(Maze[R2x][R2y]))
    OR (NOT(Edge2) AND Edge6);

  -- Should be matching the following:
  -- X1 = L1 (C2 + \L2) + \C2 \L1
  -- X2 = C2
  -- X3 = L1
  -- X4 = \L1
  -- X5 = R1 (C2 + \R2) + \C2 \R1
```

```
  -- X6 = \R1
  -- X7 = R1

  IF Edge1 THEN p ← PlotLine[p, FALSE] ELSE p ← p + szXYpair;
  IF Edge2 THEN p ← PlotLine[p, TRUE] ELSE p ← p + szTwoXYpair;
  IF Edge3 THEN
    BEGIN
    IF prevEdge3 THEN
      BEGIN
      edge3Lines[0].p2 ← p.p2;
      p ← p+szXYpair;
      edge3Lines[1].p2 ← p.p2;
      END
    ELSE
      BEGIN
      edge3Lines[0] ← p↑;
      p ← p+szXYpair;
      edge3Lines[1] ← p↑;
      prevEdge3 ← TRUE;
      END;
    p ← p+szXYpair;
    END
  ELSE
    BEGIN
    IF prevEdge3 THEN
      BEGIN
      [] ← PlotLine[@edge3Lines[0], TRUE];
      prevEdge3 ← FALSE;
      END;
    p ← p + szTwoXYpair;
    END;
  IF Edge4 THEN p ← PlotLine[p, TRUE] ELSE p ← p + szTwoXYpair;
  IF Edge5 THEN p ← PlotLine[p, FALSE] ELSE p ← p + szXYpair;
  IF Edge6 THEN p ← PlotLine[p, TRUE] ELSE p ← p + szTwoXYpair;
  IF Edge7 THEN
    BEGIN
    IF prevEdge7 THEN
      BEGIN
      edge7Lines[0].p1 ← p.p1;
      p ← p+szXYpair;
      edge7Lines[1].p1 ← p.p1;
      END
    ELSE
      BEGIN
      edge7Lines[0] ← p↑;
      p ← p+szXYpair;
      edge7Lines[1] ← p↑;
      prevEdge7 ← TRUE;
      END;
    p ← p+szXYpair;
    END
  ELSE
    BEGIN
    IF prevEdge7 THEN
      BEGIN
      [] ← PlotLine[@edge7Lines[0], TRUE];
      prevEdge7 ← FALSE;
      END;
    p ← p + szTwoXYpair;
    END;
  RETURN[p];
  END;  -- End of Hidden Procedure

ShowView: PROCEDURE [x,y: CARDINAL, dir: Direction] =
  BEGIN
  tx: CARDINAL ← x;  -- Temp position as the view
  ty: CARDINAL ← y;  --       procedes down the hall
  tp: POINTER ← table;  -- pointer into the table of vectors
  ratId: RatId;
  ratLook: RatLook;
  bvOldVisible: BOOLEAN;
  MazeDisplayDefs.InitWindow[dcb];       -- Clear work bitmap
  prevEdge3 ← prevEdge7 ← FALSE; -- Init flags for plotter smarts
  UNTIL Maze[tx][ty] DO -- Keep going till bump into wall
    IF keyStroke THEN EXIT;  -- Exit if type-ahead
    tp ← Hidden[tx,ty,dir,tp];
```

```
    SELECT dir FROM
      NORTH => ty ← ty + 1;
      SOUTH => ty ← ty - 1;
      EAST  => tx ← tx + 1;
      WEST  => tx ← tx - 1;
      ENDCASE;
    REPEAT
      FINISHED =>
        BEGIN
        IF prevEdge3 THEN [] ← PlotLine[@edge3Lines[0], TRUE];
        IF prevEdge7 THEN [] ← PlotLine[@edge7Lines[0], TRUE];
        END;
    ENDLOOP;

  dcb ← MazeDisplayDefs.Switch[]; -- display entire screen at once

  -- This is sort of poor, it should really be able to put the tokens in the invisible map
  -- so that the display all comes on at once, but if I do that now, I have problems
  -- with keeping track of what tokens are in the visible/invisible display
  FOR ratId IN RatId DO          -- Scan for visible tokens
    IF ratId = myRatId THEN LOOP;
    ratLook ← @r2d2[ratId];
    bvOldVisible ← ratLook.visible;
    TokenVisible[ratId];         -- Set the token info in the r2d2 array
    IF ratLook.visible THEN XORToken[ratId, @r2d2];
    IF ratLook.visible # bvOldVisible THEN UpdateScoreCard[ratId];
    ENDLOOP;
  END; -- End of View Procedure


SendLocToAll: PROCEDURE =
  BEGIN
  b: PupBuffer;                  -- My pup buffer
  ratloc: RatLocation;                  -- Pointer into the ratlocation msg
  i: CARDINAL;
  ratInfo: RatInfo ← @ratcb.rats[myRatId];
  ratInfo.xLoc ← xloc;                  -- Update my table too
  ratInfo.yLoc ← yloc;
  ratInfo.dir ← tdir;
  ratInfo.score ← score;
  bvSendLocAll ← FALSE;
  -- Following code sends from my ratcb in case the data changes while
  -- I'm waiting for one of the pups to be sent.
  FOR i IN [0..MaxRats) DO
    IF i # myRatId AND ratcb.rats[i].playing THEN
      BEGIN
      b ← GetFreePupBuffer[];                    -- Get a new buffer
      b.pupType ← ratLocation;
      b.dest ← ratcb.rats[i].addr;
      b.source ← myAddr;
      SetPupContentsWords[b, SIZE[AqRatLocation]];
      ratloc ← LOOPHOLE[@b.pupBody];     -- Get msg body set
      ratloc.ratId ← myRatId;
      ratloc.xLoc ← ratInfo.xLoc;
      ratloc.yLoc ← ratInfo.yLoc;
      ratloc.dir ← ratInfo.dir ← tdir;
      ratloc.score ← ratInfo.score ← score;
      [] ← PupRouterSendThis[b];
      END;
    ENDLOOP;
  RETURN;
  END;

SendAllStatus: PROCEDURE =
  BEGIN
  i: CARDINAL;
  bvSendAllStatus ← FALSE;
  FOR i IN [0..MaxRats) DO
    IF i # myRatId AND ratcb.rats[i].playing THEN
      SendStatus[ratcb.rats[i].addr];
    ENDLOOP;
  RETURN;
  END;

SendStatus: PROCEDURE [addr: PupAddress] =
  BEGIN
```

```
  b: PupBuffer;                         -- My pup buffer
  status: RatStatus;                    -- Pointer into the status msg
  bvSendOneStatus ← FALSE;
  b ← GetFreePupBuffer[];               -- Get a new buffer
  b.pupType ← ratStatus;
  b.dest ← addr;
  b.source ← myAddr;
  SetPupContentsWords[b, SIZE[RatCb]];
  status ← LOOPHOLE[@b.pupBody];        -- Get msg body set
  status↑ ← ratcb;                      -- set the data in the msg
  [] ← PupRouterSendThis[b];
  RETURN;
  END;

SendKill: PROCEDURE =
  BEGIN
  b: PupBuffer;                 -- My pup buffer
  ratkill: RatKill;             -- Pointer into the kill msg
  ixRatId: RatId;               -- Index into the r2d2 array looking for opponents
  bvSendKill ← FALSE;
  FOR ixRatId IN RatId DO
    IF ixRatId = myRatId THEN LOOP;     -- See about getting rid of this
    IF r2d2[ixRatId].visible THEN
      BEGIN
      b ← GetFreePupBuffer[];           -- Get a new buffer
      b.pupType ← ratKill;
      b.dest ← ratcb.rats[ixRatId].addr;
      b.source ← myAddr;
      SetPupContentsWords[b, SIZE[AqRatKill]];
      ratkill ← LOOPHOLE[@b.pupBody];   -- Get msg body set
      ratkill.ratId ← myRatId;
      ratkill.xLoc ← xloc;
      ratkill.yLoc ← yloc;
      ratkill.dir ← tdir;
      [] ← PupRouterSendThis[b];
      END;
    ENDLOOP;
  RETURN;
  END;

SendDead: PROCEDURE =
  BEGIN
  b: PupBuffer;                 -- My pup buffer
  ratdead: RatDead;                     -- Pointer into the dead msg
  bvSendDead ← FALSE;
  b ← GetFreePupBuffer[];                       -- Get a new buffer
  b.pupType ← ratDead;
  b.dest ← ratcb.rats[killerRatId].addr;
  b.source ← myAddr;
  SetPupContentsWords[b, SIZE[AqRatDead]];
  ratdead ← LOOPHOLE[@b.pupBody];               -- Get msg body set
  ratdead.ratId ← myRatId;
  ratdead.killedBy ← killerRatId;
  [] ← PupRouterSendThis[b];
  RETURN;
  END;

SendGoing: PROCEDURE =
  BEGIN
  b: PupBuffer;                 -- My pup buffer
  ratgone: RatGone;                     -- Pointer into the gone msg
  bvSendGoing ← FALSE;
  b ← GetFreePupBuffer[];                       -- Get a new buffer
  b.pupType ← ratGoing;
  b.dest ← ratcb.rats[ratcb.dukeRat].addr;
  b.source ← myAddr;
  SetPupContentsWords[b, SIZE[AqRatGone]];
  ratgone ← LOOPHOLE[@b.pupBody];               -- Get msg body set
  ratgone.ratId ← myRatId;
  [] ← PupRouterSendThis[b];
  RETURN;
  END;

SendQuery: PROCEDURE =
  BEGIN
  b: PupBuffer;                 -- My pup buffer
```

```
  ratquery: RatQuery;                   -- Pointer into the query msg
  ratid: RatId;
  bvSendQuery ← FALSE;
  FOR ratid IN RatId DO
    IF ratHealth[ratid].pupSend THEN
      BEGIN
      ratHealth[ratid].pupSend ← FALSE;
      b ← GetfreePupBuffer[];                   -- Get a new buffer
      b.pupType ← ratQuery;
      b.dest ← ratcb.rats[ratid].addr;
      b.source ← myAddr;
      SetPupContentsWords[b, SIZE[AqRatQuery]];
      ratquery ← LOOPHOLE[@b.pupBody];          -- Get msg body set
      ratquery.ratId ← myRatId;
      [] ← PupRouterSendThis[b];
      END;
    ENDLOOP;
  RETURN;
  END;

SendAlive: PROCEDURE =
  BEGIN
  b: PupBuffer;                 -- My pup buffer
  ratid: RatId;
  ratalive: RatAlive;                   -- Pointer into the alive msg
  FOR ratid IN RatId DO
    IF ratid=myRatId OR ~ ratcb.rats[ratid].playing THEN LOOP;
    b ← GetFreePupBuffer[];                 -- Get a new buffer
    b.pupType ← ratAlive;
    b.dest ← ratcb.rats[ratid].addr;
    b.source ← myAddr;
    SetPupContentsWords[b, SIZE[AqRatAlive]];
    ratalive ← LOOPHOLE[@b.pupBody];        -- Get msg body set
    ratalive.ratId ← myRatId;
    [] ← PupRouterSendThis[b];
    ENDLOOP;
  bvSendAlive ← FALSE;
  RETURN;
  END;

ReadRats: PROCEDURE =
  BEGIN
  b: PupBuffer;                 -- My pup buffer
  ratLook: RatLook;
  ratInfo: RatInfo;
  DO
    b ← ps.get[];                           -- Wait for a pup buffer
    IF b = NIL THEN LOOP;       -- Ignore nils (shouldn't get any really)
    SELECT b.pupType FROM       -- See what we have here
      ratLocation =>
        BEGIN
        ratloc: RatLocation ← LOOPHOLE[@b.pupBody];
        bvOldVisible: BOOLEAN;
        ratLook ← @r2d2[ratloc.ratId];
        IF (bvOldVisible ← ratlook.visible) THEN
          XORToken[ratloc.ratId, @r2d2];
        ratInfo ← @ratcb.rats[ratloc.ratId];
        ratInfo.xLoc ← ratloc.xLoc;
        ratInfo.yLoc ← ratloc.yLoc;
        ratInfo.dir ← ratloc.dir;
        DisplayOthersPosition[ratloc.ratId, ratloc.xLoc, ratloc.yLoc,
                              ratloc.dir];
        TokenVisible[ratloc.ratId];     -- Set the token info in the r2d2 array
        IF ratLook.visible THEN XORToken[ratloc.ratId, @r2d2];
        IF bvOldVisible # ratLook.visible OR
          ratInfo.score # ratloc.score THEN     -- If his score has changed
          BEGIN
          ratInfo.score ← ratloc.score;
          UpdateScoreCard[ratloc.ratId];
          END;
        ratHealth[ratloc.ratId].pupRcvd ← TRUE;
        END;
      ratStatus =>
        BEGIN
        status: RatStatus ← LOOPHOLE[@b.pupBody];
        ratid: RatId;
```

```
      IF status.rats[myRatId].addr # myAddr THEN GOTO notMyPacket;
      --Have a new table, turn off any visible opponents
      FOR ratid IN RatId DO
        IF r2d2[ratid].visible THEN XORToken[ratid, @r2d2];
        ENDLOOP;
      ratcb ← status;
      IF ratcb.dukeRat = myRatId THEN duke←TRUE;
      FOR ratid IN RatId DO
        TokenVisible[ratid];
        IF r2d2[ratid].visible THEN
        XORToken[ratid, @r2d2];
        ENDLOOP;
      NewScoreCard[];
      ratInfo ← @ratcb.rats[myRatId];
      IF ratInfo.xLoc # xloc OR ratInfo.yLoc # yloc
        OR ratInfo.dir # tdir THEN
        BEGIN
        bvSendLocAll ← TRUE;
        PupOutput ← TRUE;
        END;
      EXITS
        notMyPacket => NULL;
      END;
    ratNew =>
      BEGIN
      ratnew: RatNew ← LOOPHOLE[@b.pupBody];
      IF ratnew.pass = MazeNewDefs.Password THEN
        BEGIN
        IF duke THEN
          BEGIN
          AllocateNewRat[ratnew];
          bvSendAllStatus ← TRUE;
          END
        ELSE
          BEGIN
          newAddr ← ratnew.addr;
          bvSendOneStatus ← TRUE;
          END;
        PupOutput ← TRUE;
        END;
      END;
    ratSurvey =>
      BEGIN
      ratnew: RatNew ← LOOPHOLE[@b.pupBody];
      IF ratnew.pass = MazeNewDefs.Password THEN
        BEGIN
        newAddr ← b.source;
        bvSendOneStatus ← TRUE;
        PupOutput ← TRUE;
        END;
      END;
    ratKill =>  IF ~Invincible THEN
      IF ctBreaths < 5 THEN           -- Only if within process limit
      BEGIN
      p: PROCESS;
      ratkill: RatKill ← LOOPHOLE[@b.pupBody];
      ctBreaths ← ctBreaths + 1;
      p ← FORK HoldBreath[tx: ratkill.xLoc, ty: ratkill.yLoc,
        td: ratkill.dir, ratId: ratkill.ratId];
      ProcessDefs.Detach[p];
      END;
    ratDead =>
      BEGIN
      ratdead: RatDead ← LOOPHOLE[@b.pupBody];
      IF ratdead.killedBy = myRatId THEN
        BEGIN
        InvertTop[];  --got him
        score ← score + 10;           -- 10 points for a kill
        ratcb.rats[myRatId].score ← score;
        UpdateScoreCard[myRatId];     -- Display new score
        bvSendLocAll ← TRUE;          -- Send loc change to all but
        PupOutput ← TRUE;             -- don't care how long it takes
        END;
      END;
    ratGoing =>
      BEGIN
```

```
        ratgone: RatGone ← LOOPHOLE[@b.pupBody];
        RatLeft[ratgone.ratId];
        END;
      ratQuery =>
        BEGIN
        bvSendAlive ← TRUE;
        PupOutput ← TRUE;
        END;
      ratAlive =>
        BEGIN
        ratalive: RatAlive ← LOOPHOLE[@b.pupBody];
        ratHealth[ratalive.ratId].pupRcvd ← TRUE;
        END;
      ENDCASE;
    ReturnFreePupBuffer[b];
    ENDLOOP;
  END;

RatLeft: ENTRY PROCEDURE [ratId: RatId] =
  BEGIN
  IF r2d2[ratId].visible THEN XORToken[ratId, @r2d2];
  r2d2[ratId].visible ← FALSE;
  ratcb.rats[ratId].playing ← FALSE;
  ExitPlayer[ratId];
  UpdateScoreCard[ratId];  -- Update my display
  bvSendAllStatus ← TRUE;    -- Send new player table to all
  PupOutput ← TRUE;
  END;

AllocateNewRat: PROCEDURE [ratnew: RatNew] =
  BEGIN
  ratId: RatId;
  ratInfo: RatInfo;
  FOR ratId IN RatId DO
    ratInfo ← @ratcb.rats[ratId];
    IF NOT ratInfo.playing THEN
      BEGIN
      ratInfo.playing ← TRUE;
      ratInfo.xLoc ← ratnew.xLoc;
      ratInfo.yLoc ← ratnew.yLoc;
      ratInfo.dir ← ratnew.dir;
      ratInfo.score ← 0;
      ratInfo.addr ← ratnew.addr;
      ratInfo.name ← ratnew.name;
      TokenVisible[ratId];              -- See if new guy is visible
      UpdateScoreCard[ratId];           -- Update my score card
      IF r2d2[ratId].visible THEN XORToken[ratId, @r2d2];       -- And my view
      AddNewPlayer [ratId, ratnew.xLoc, ratnew.yLoc, ratnew.dir];
      EXIT;
      END;
    ENDLOOP;
  RETURN;
  END;

TokenVisible: PROCEDURE  [hisRatId: RatId] =
  BEGIN
  -- Sets R2D2[hisRatId] variables
  -- Uses RatCb[hisRatId] as input for his position and direction
  ratLook: RatLook ← @r2d2[hisRatId];
  ratInfo: RatInfo ← @ratcb.rats[hisRatId];
  tx, ty: Loc;
  td: Direction;          -- My position and direction
  ix: CARDINAL;           -- Major index into the vectors table
  ix12: CARDINAL;         -- Minor index into the vectors table (12 vectors per frame)
  ratLook.visible ← FALSE;       -- Init in case we never see the turkey
  IF NOT ratInfo.playing THEN RETURN;
  IF peeking THEN
    BEGIN tx ← xPeek; ty ← yPeek; td ← dirPeek; END
  ELSE
    BEGIN tx ← xloc; ty ← yloc; td ← tdir; END;
  ix ← 0;                                 -- Start at beginning of the table
  UNTIL Maze[tx][ty] DO
    SELECT td FROM
      NORTH => ty ← ty + 1;
      SOUTH => ty ← ty - 1;
      WEST => tx ← tx - 1;
```

```
      EAST => tx ← tx + 1;
      ENDCASE => ERROR;
    ix ← ix + 1;                          -- Have gone to next "frame" in the table
    IF tx = ratInfo.xLoc AND ty = ratInfo.yLoc THEN
      BEGIN
      ratLook.visible ← TRUE;
      -- See brd for an explaination of the following lines
      -- vector numbers start at zero (doc has 1)
      ix12 ← ix*12;
      ratLook.x ← (table[ix12+3].p2.x + table[ix12+10].p1.x) / 2;
      ratLook.y ← (table[ix12+3].p1.y + table[ix12+3].p2.y) / 2;
      ratLook.tokenId ← relativeTokens [td][ratInfo.dir];
      ratLook.distance ← ix;
      EXIT;
      END;
    ENDLOOP;
  RETURN;
  END;

HoldBreath: PROCEDURE [tx,ty: Loc, td: Direction, ratId: RatId] =
  BEGIN
  Wait[1000];                                     -- Wait a second
  UNTIL Maze[tx][ty] DO
    SELECT td FROM
      NORTH => ty ← ty + 1;
      SOUTH => ty ← ty - 1;
      WEST => tx ← tx - 1;
      EAST => tx ← tx + 1;
      ENDCASE => ERROR;
    IF xloc = tx AND yloc = ty THEN       -- Oh oh, I've been hit
      BEGIN
      killerRatId ← ratId;                -- Set global variable for other process
      bvSendDead ← TRUE;
      PupOutput ← TRUE;
      NewPosition[];                      -- Set new position (to keep from getting multiple hits)
      FlashScreen[];
      score ← score - 5;                  -- minus five points for getting killed
      ratcb.rats[myRatId].score ← score;        -- Update my score now
      UpdateScoreCard[myRatId];           -- Update viewer window
      keyStroke ← TRUE;                   -- Tell the viewer to update display (and send out locs)
      END;
    ENDLOOP;
  ctBreaths ← ctBreaths - 1;
  RETURN[];                               -- Stop this process
  END;

FlashScreen: PROCEDURE =
  BEGIN
  THROUGH [1..4] DO
    InvertScreen[];
    Wait[250];                                    -- Wait a 1/4 second
    ENDLOOP;
  RETURN[];
  END;

InvertScreen: ENTRY PROCEDURE =
  BEGIN OPEN AltoDisplay;
  dcb: DCBHandle;
  FOR dcb ← DCBchainHead.next, dcb.next UNTIL dcb = DCBnil DO
  SELECT dcb.background FROM
    white => dcb.background ← black;
    black => dcb.background ← white;
    ENDCASE;
  ENDLOOP;
  MazeDisplayDefs.InvertHidden[];
  RETURN[];
  END;

NewPosition: PROCEDURE =
  BEGIN
  rndCnt: CARDINAL ← 0;
  xloc ← yloc ← 0;                -- Start off on an occupied square for loop test sake
  UNTIL NOT Maze[xloc][yloc] DO -- Spin till hit an empty square (can't coexist with a cube)
    xloc ← Random[16];
    yloc ← Random[32];
```

```
    IF (rndCnt ← rndCnt + 1) = 100 THEN BEGIN rndCnt ← 0; InitRandom[] END;
    ENDLOOP;
  -- The following is designed to prevent a blank wall at first glimpse
  SELECT FALSE FROM
    Maze[xloc][yloc+1] => tdir ← NORTH;
    Maze[xloc][yloc-1] => tdir ← SOUTH;
    Maze[xloc+1][yloc] => tdir ← EAST;
    Maze[xloc-1][yloc] => tdir ← WEST;
    ENDCASE;
  RETURN
  END;

InitRandom: PROCEDURE =
  BEGIN
  i: CARDINAL;
  p: POINTER TO CARDINAL = LOOPHOLE[430B];
  t: CARDINAL = p↑;
  FOR i IN [0..VectorSize) DO randomVector[i] ← t + randomVector[i] ENDLOOP;
  END;

Wait: ENTRY PROCEDURE [msecs: CARDINAL] =
  BEGIN
  cv: CONDITION;
  ProcessDefs.InitializeCondition[@cv, ProcessDefs.MsecToTicks[msecs]];
  WAIT cv;
  RETURN[];
  END;

Quit: PROCEDURE =
  BEGIN
  ratId: RatId;
  IF ~duke THEN
    BEGIN
    bvSendGoing ← TRUE;
    PupOutput ← TRUE;
    END
  ELSE                              -- Oh oh, I'm the dukerat
    BEGIN
    ratcb.rats[myRatId].playing ← FALSE;          -- Turn me off!
    FOR ratId IN RatId DO
      IF ratcb.rats[ratId].playing THEN           -- If have found a new duke
        BEGIN
        ratcb.dukeRat ← ratId;
        bvSendAllStatus ← TRUE;
        PupOutput ← TRUE;
        EXIT;
        END;
      ENDLOOP;
    END;
  ProcessDefs.SetPriority[1];  --Set old keyboard priority back to normal
  WHILE PupOutput DO ProcessDefs.Yield[] ENDLOOP;
  ImageDefs.StopMesa[];
  END;

NewScoreCard: PROCEDURE =
  BEGIN
  ratid: RatId;
  FOR ratid IN RatId DO UpdateScoreCard[ratid]; ENDLOOP;
  ShowAllPositions [ @ratcb.rats];
  RETURN;
  END;

UpdateScoreCard: PROCEDURE [ratId: RatId] =
  BEGIN
  sv: STRING ← [40];                    -- Temp String
  ix: CARDINAL;
  MazeDisplayDefs.ClearScoreLine[ratId];
  IF ratcb.rats[ratId].playing THEN
    BEGIN
    FOR ix IN [0..20) DO
      sv[ix] ← ratcb.rats[ratId].name[ix];
      ENDLOOP;
    sv.length ← 20;
    StringDefs.AppendDecimal[sv, ratcb.rats[ratId].score];
    MazeDisplayDefs.WriteScoreString[ratId, sv];
    END;
```

```
    IF r2d2[ratId].visible THEN MazeDisplayDefs.InvertScoreLine[ratId];
    RETURN;
    END;

  oldBits: KeyDefs.KeyBits ← KeyDefs.Keys↑;
  old: POINTER TO KeyDefs.KeyBits ← @oldBits;
  nowBits: KeyDefs.KeyBits ← KeyDefs.Keys↑;
  now: POINTER TO KeyDefs.KeyBits ← @nowBits;

  bigMaze: BOOLEAN ← FALSE;

  ReadKeys: ENTRY PROCEDURE =
    BEGIN OPEN KeyDefs;
    DO
      WAIT wakeup;                                    -- wait for the naked notify
      oldBits ← nowBits;  -- Save "old" bits for diff check
      nowBits ← Keys↑;  -- Get current status
      now.blank ← 0;  -- blank out trash
      IF now↑ = old↑ THEN LOOP;  -- if the same, forget it
     IF ~ bigMaze THEN
      IF NOT peeking THEN
        BEGIN
        IF (now.Keyset1 = down AND old.Keyset1 = up) OR
          (now.A = down AND old.A = up) THEN AboutFace[];
        IF (now.Keyset2 = down AND old.Keyset2 = up) OR
          (now.S = down AND old.S = up) THEN LeftTurn[];
        IF (now.Keyset3 = down AND old.Keyset3 = up) OR
          (now.D = down AND old.D = up) THEN Forward[];
        IF (now.Keyset4 = down AND old.Keyset4 = up) OR
          (now.F = down AND old.F = up) THEN RightTurn[];
        IF (now.Keyset5 = down AND old.Keyset5 = up) OR
          (now.Space = down AND old.Space = up) THEN Backward[];
        IF now.Red = down AND old.Red = up THEN PeekLeft[];
        IF now.Blue = down AND old.Blue = up THEN PeekRight[];
        IF now.Yellow = down AND old.Yellow = up THEN Shoot[];
        IF now.I = down AND old.I = up THEN MakeInvincible[TRUE];
        IF now.K = down AND old.K = up THEN MakeInvincible[FALSE]; --killable
        IF now.E = down THEN BigMaze[];
        END
      ELSE                          -- if peeking
        IF (now.Red = up) OR (now.Blue = up) THEN PeekStop[]
    ELSE  --if BigMaze
        IF now.E = up THEN SmallMaze[];
      IF now.DEL = down THEN Quit[];        -- TIme to leave the game
      ENDLOOP;
    END;  -- of ReadKeys

  DispatchKeys: PROCEDURE =
    BEGIN
    DO
      WHILE NOT keyStroke DO
          ProcessDefs.Yield[];
          ENDLOOP;
      keyStroke ← FALSE;
      IF NOT peeking THEN
        BEGIN
        ShowPosition[xloc,yloc, Invincible, tdir];  -- show new location
        ShowView[xloc,yloc,tdir];           -- and display the view
        END
      ELSE                          -- if peeking
        ShowView[xPeek,yPeek,dirPeek];      -- then display the peek view
      bvSendLocAll ← TRUE;                  -- Send loc change to all and
      PupOutput ← TRUE;             -- wait till they are sent!
      WHILE PupOutput DO ProcessDefs.Yield[] ENDLOOP;
      ENDLOOP;
    END;  -- OF DispatchKeys

  BigMaze: PROCEDURE =
          BEGIN
          bigMaze ← TRUE;
          MakeBigMaze[];
          END;

  SmallMaze: PROCEDURE =
          BEGIN
          bigMaze ← FALSE;
```

```
        MakeSmallMaze[];
        END;
MakeInvincible: PROCEDURE [neverDie: BOOLEAN] =
  BEGIN
  Invincible ← neverDie;
  ShowPosition[xloc,yloc, Invincible, tdir];  -- show invincibility
  END;

AboutFace: PROCEDURE = INLINE                    -- About face
  BEGIN
  aboutFace: ARRAY Direction OF Direction = [SOUTH, NORTH, WEST, EAST];
  tdir ← aboutFace[tdir];
  keyStroke ← TRUE;
  END;

LeftTurn: PROCEDURE = INLINE                     -- Left turn
  BEGIN
  leftTurn: ARRAY Direction OF Direction = [WEST, EAST, NORTH, SOUTH];
  tdir ← leftTurn[tdir];
  keyStroke ← TRUE;
  END;

RightTurn: PROCEDURE = INLINE                    -- Right turn
  BEGIN
  rightTurn: ARRAY Direction OF Direction = [EAST, WEST, SOUTH, NORTH];
  tdir ← rightTurn[tdir];
  keyStroke ← TRUE;
  END;

Forward: PROCEDURE = INLINE                      -- Forward one
  BEGIN -- Move forward
  tx: CARDINAL ← xloc;
  ty: CARDINAL ← yloc;
  SELECT tdir FROM
    NORTH => IF ~Maze[xloc][yloc+1] THEN ty ← yloc + 1;
    SOUTH => IF ~Maze[xloc][yloc-1] THEN ty ← yloc - 1;
    EAST => IF ~Maze[xloc+1][yloc] THEN tx ← xloc + 1;
    WEST => IF ~Maze[xloc-1][yloc] THEN tx ← xloc - 1;
    ENDCASE => ERROR;
  IF tx#xloc OR ty#yloc THEN
  BEGIN xloc ← tx; yloc ← ty; keyStroke ← TRUE; END;
  END;

Backward: PROCEDURE = INLINE                     -- Backward one
  BEGIN -- Move backward
  tx: CARDINAL ← xloc;
  ty: CARDINAL ← yloc;
  SELECT tdir FROM
    NORTH => IF ~Maze[xloc][yloc-1] THEN ty ← yloc - 1;
    SOUTH => IF ~Maze[xloc][yloc+1] THEN ty ← yloc + 1;
    EAST => IF ~Maze[xloc-1][yloc] THEN tx ← xloc - 1;
    WEST => IF ~Maze[xloc+1][yloc] THEN tx ← xloc + 1;
    ENDCASE => ERROR;
  IF tx#xloc OR ty#yloc THEN
  BEGIN xloc ← tx; yloc ← ty; keyStroke ← TRUE; END;
  END;

PeekLeft: PROCEDURE = INLINE
  BEGIN
  xPeek ← xloc;
  yPeek ← yloc;
  dirPeek ← tdir;
  SELECT tdir FROM
    NORTH =>
      IF ~Maze[xloc][yloc+1] THEN BEGIN yPeek ← yloc+1; dirPeek ← WEST END;
    SOUTH =>
      IF ~Maze[xloc][yloc-1] THEN BEGIN yPeek ← yloc-1; dirPeek ← EAST END;
    EAST =>
      IF ~Maze[xloc+1][yloc] THEN BEGIN xPeek ← xloc+1; dirPeek ← NORTH END;
    WEST =>
      IF ~Maze[xloc-1][yloc] THEN BEGIN xPeek ← xloc-1; dirPeek ← SOUTH END;
    ENDCASE => ERROR;
  -- If any change; display the new view without moving!
  IF xPeek#xloc OR yPeek#yloc THEN
  BEGIN peeking ← TRUE; keyStroke ← TRUE END;
  RETURN
```

```
  END;

PeekRight: PROCEDURE = INLINE
  BEGIN
  xPeek ← xloc;
  yPeek ← yloc;
  dirPeek ← tdir;
  SELECT tdir FROM
    NORTH =>
      IF ~Maze[xloc][yloc+1] THEN BEGIN yPeek ← yloc+1; dirPeek ← EAST END;
    SOUTH =>
      IF ~Maze[xloc][yloc-1] THEN BEGIN yPeek ← yloc-1; dirPeek ← WEST END;
    EAST =>
      IF ~Maze[xloc+1][yloc] THEN BEGIN xPeek ← xloc+1; dirPeek ← SOUTH END;
    WEST =>
      IF ~Maze[xloc-1][yloc] THEN BEGIN xPeek ← xloc-1; dirPeek ← NORTH END;
    ENDCASE => ERROR;
  -- If any change; display the new view without moving!
  IF xPeek#xloc OR yPeek#yloc THEN
  BEGIN peeking ← TRUE; keyStroke ← TRUE END;
  RETURN
  END;

PeekStop: PROCEDURE = INLINE
  BEGIN
  peeking ← FALSE;
  keyStroke ← TRUE;
  RETURN
  END;

Shoot: PROCEDURE = INLINE
  BEGIN
  bvSendKill ← TRUE;              -- Send out a kill msg
  PupOutput ← TRUE;               -- Wake up the pup output process
  RETURN
  END;

RatDoctor: PUBLIC PROCEDURE =
  BEGIN
  ratid: RatId;
  FOR ratid IN RatId DO
    ratHealth[ratid].count ← 0;
    ratHealth[ratid].pupSend ← FALSE;
    ENDLOOP;
  DO
    FOR ratid IN RatId DO
      ratHealth[ratid].pupRcvd ← FALSE;
      ENDLOOP;
    Wait[10*1000]; -- wait 10 seconds
    FOR ratid IN RatId DO
      IF ~ratcb.rats[ratid].playing OR ratid=myRatId THEN LOOP;
      IF ratHealth[ratid].pupRcvd THEN
        BEGIN
        ratHealth[ratid].count ← 0;
        LOOP
        END;
      IF (ratHealth[ratid].count ← ratHealth[ratid].count + 1) < 5 THEN
        BEGIN
        ratHealth[ratid].pupSend ← TRUE;
        bvSendQuery ← TRUE;
        PupOutput ← TRUE;
        LOOP;
        END;
      IF duke OR (ratcb.dukeRat = ratid AND NextRatId[ratid]=myRatId) THEN
        BEGIN
        duke ← TRUE;
        ratcb.dukeRat ← myRatId;
        RatLeft[ratid];
        END;
      ENDLOOP;
    ENDLOOP;
  END;

NextRatId: PROCEDURE [ratid: RatId] RETURNS [ixRatId: RatId] =
  BEGIN
  FOR ixRatId IN RatId DO
```

```
    IF ratcb.rats[ixRatId].playing AND ixRatId#ratid THEN RETURN[ixRatId];
    ENDLOOP;
  RETURN[ratid];
  END;

Play: PROCEDURE =
  BEGIN
  DO
    WHILE NOT PupOutput DO ProcessDefs.Yield[] ENDLOOP;
    PupOutput ← FALSE;
    IF bvSendLocAll THEN SendLocToAll[];
    IF bvSendOneStatus THEN SendStatus[newAddr];
    IF bvSendAllStatus THEN SendAllStatus[];
    IF bvSendKill THEN SendKill[];
    IF bvSendDead THEN SendDead[];
    IF bvSendGoing THEN SendGoing[];
    IF bvSendQuery THEN SendQuery[];
    IF bvSendAlive THEN SendAlive[];
    ENDLOOP;
  END;


-- Main Line Code

--*****************************************************************************--
FOR ratId IN [0..MaxRats) DO
        ratcb.rats[ratId].playing ← FALSE;
        ENDLOOP;
--*****************************************************************************--
START MazeInit;
FrameDefs.UnNew[LOOPHOLE[MazeInit]];
Play[];

END...
```