G-PASCAL NEWS

Registered by Australia Post - Publication Number VBG 6589

Published by Gambit Games for Commodore 64 G-Pascal owners.

Mail: G-Pascal News, P.O. Box 124, Ivanhoe, Victoria 3079. (Australia)
Phone: (03) 497 1283

Gambit Games is a trading name of Gammon & Gobbett Computer Services Proprietary Limited, a company incorporated in the State of Victoria.

VOLUME 2, NUMBER 4 — October 1985 **Annual Subscription $12**

WHAT'S IN THIS ISSUE

This issue has two main programs - an arcade game, and an adventure melee adjudicating game. Also on this page we describe the final techniques for customising your G-Pascal compiler by adjusting symbol ble and stack sizes. This is the final issue of G-Pascal News — we are not publishing it in 1986. Thank you for your support.

INCREASING SYMBOL TABLE SIZE

Page 38 of the G-Pascal Manual states that the maximum size of the symbol table used by the G-Pascal compiler is 4K bytes (4096 bytes). As each symbol used (i.e. each programmer-defined PROCEDURE, VAR, FUNCTION, or CONST) takes up 10 bytes in the symbol table, plus the length of the name of the symbol, you may conceivably fill up the symbol table if you have a program with a lot of symbols (although this has never happened to us with the Commodore 64 version of G-Pascal).

If you want to increase the size of the symbol table then you must move it to somewhere else in memory (for example, to the 8K block of memory between addresses $2000 and $4000). To do this, run the following program:

```
BEGIN
  MEMC [$9A3D] := $40;  (* end of table *)
  MEMC [$800B] := 32    (* size of table *)
END.
```

The byte at $9A3D controls where the symbol table **ends** (high-order byte of address) — the byte at $800B controls the number of 256-byte 'pages' in the symbol table (the figure 32 represents 32 times 256 = 8192 bytes, or 8K).

The normal figures for the symbol table are: end of symbol table = $D000, length of symbol table = 16.

INCREASING THE SIZE OF THE STACK

Page 39 of the G-Pascal manual states that the stack size used by the G-Pascal interpreter is 3,790 bytes (1,263 integers or 3,790 CHAR variables). A larger stack may be useful if you want to use large arrays (e.g. an array of 10,000 CHARs for a word-processor written in G-Pascal). If you want to have a larger run-time stack then you can run the following program:

```
BEGIN
  MEMC [$A002] := $00; (* start stack - LSB *)
  MEMC [$A008] := $40; (* start stack - MSB *)
  MEMC [$B1C3] := $20  (* bottom stack - MSB *)
END.
```

MSB and LSB refer to Most Significant Byte, and Least Significant Byte, respectively.

This program makes the run-time stack start at address $4000, and finish at $2000 (an attempt to go below $2000 would cause a 'stack full' error message). This program would give you an 8K stack, but you can make the stack even larger by making the bottom of the stack as low as $800. It is permissable to overlap the stack and the symbol table (described above) as one only applies at compile time, and the other at run time. You cannot however overlap the stack or the symbol table and your P-codes.

Note that the program described above patches the interpreter for the **next** program executed — the current one (with the patches in it) already has the stack defined, and will not change in the middle of execution.

The corresponding addresses when using the Run-Time System are: start of stack — low-order byte: $8277; start of stack — high-order byte: $827D; bottom of stack: $95A5. Once again, these changes will only take effect for the **next** program run - you would need to use the 'chain' option to make the patches in one program, and then execute the 'main' program that needs the larger stack.

ARCADE GAME — 'BALLS UP'

We are very pleased to present in this issue a complex arcade-style game which we have named 'Balls Up' (the name derives from the action of the game, which involves coloured balls bouncing around the screen).

This game illustrates many of the advanced techniques of programming an arcade game in G-Pascal, such as:

a) Using the hardware clock to time segments of a game (SETCLOCK and CLOCK).

b) Using sprites as the name of the game (see lines 250 to 285 to define the name, and 377 to 398 to display it).

c) Using sprites to hold score numbers (see lines 242 to 249 where score is defined, and lines 689 to 695 where it is displayed).

d) Checking for sprite collising by using SPRITEFREEZE and FREEZESTATUS.

e) Allowing input from keyboard, paddles or joystick (see lines 309 to 329).

f) Allowing user-defined direction keys if input is from keyboard (see lines 331 to 359 for asking player which one to use, lines 641 to 661 to move base depending on input mode).

g) Using ANIMATESPRITE to make the moving aliens (balls) look more interesting (see lines 537 to 540).

h) Using ANIMATESPRITE to draw an explosion automatically (see lines 710 to 715 and lines 731 and 732).

i) Using SPRITEX and SPRITEY to check which sprites have collided (see lines 745 to 754).

j) Automatic 'attract' mode between games.

k) When using paddles, game automatically selects correct paddle depending on which fire button is pressed (see lines 452 to 470).

l) Using RANDOM function to make games unique.

m) Nested PROCEDURES, CASE statements, and other interesting Pascal programming techniques.

How to play

Balls Up is a fast-moving arcade game. The player controls a 'defending base' using the keyboard, joystick or paddles. When the game first starts running select the input mode by pressing the appropriate function key as directed by the game. If you select keyboard input you will then be asked to press your choice of 'left', 'right' and 'fire' keys. Choose any keys that you find comfortable with (for example 'A' for left, 'L' for right and 'H' for fire). If you choose paddles then there are two paddles connected to the one gameport. The paddle used for a particular game is selected by pressing the appropriate 'fire' button when prompted by the game. This is so that two players can play alternate games, holding one paddle each.

The object of the game is to stay alive as long as possible, accumulating as high a score as possible. You initially start with three defending bases (playing with one at a time) — a base is destroyed if it collides with a 'ball'. Each 'attack wave' starts with six balls placed randomly on the screen. They then start moving, bouncing off the walls of the screen similarly to billiard balls. You must dodge out of their way, and shoot them as quickly as possible. Unlike ordinary billiard balls, the balls in this game speed up slightly each time they bounce off a wall.

You gain bonus points for clearing an attack wave quickly (in less than 15, 30 and 45 seconds respectively for levels 1, 2 and 3).

You also gain bonus points for accurate shooting, no matter how long it takes to clear the attack wave. Provided your shooting accuracy is greater than 66 percent (in other words, if at least two-thirds of your shots hit a target) then you gain an accuracy bonus.

Balls on the first wave are first level balls, are coloured white and black, and only need one hit to kill them. They are worth 50 points each.

Balls on the second wave are level two balls, are coloured yellow and black, and need two hits to kill them (the first hit makes them change colour to a 'level one' ball for identification purposes). They are worth 100 points each.

Balls on the third wave are level three balls, are coloured cyan and black, and need three hits to kill them (the first two hits make them change colour to the next lower level). They are worth 200 points each.

Balls on subsequent waves are a random mixture of level one to three balls (identifiable by their colours). They need the appropriate number of hits to kill them that their colours indicate. They are worth 500 points each, regardless of colour.

When a ball is hit but not killed (i.e. a ball at a higher level than level one is hit) then it immediately reverses direction. You must

be cautious when sitting underneath a ball travelling away from you, as hitting it will make it come towards you. On the other hand, if it is in a collision course with you then hitting it will make it go away.

Your base can only fire one missile at once. When your base is ready to fire you will see the missile in place at the top of it as a white square. If there is no white square on the top of the base you cannot fire yet. You can however fire repeatedly by holding down the fire button (this will make a missile fire as soon as it is available).

If a base is destroyed then the wave recommences after a short pause with the remaining balls for that wave repositioned on the screen. It is possible to finish a wave by the kamikaze technique of colliding with the last ball (this is wasteful of bases of course). Nevertheless, if your base collides with the last ball then that particular wave is over, and the game continues if you have some remaining bases.

When your score reaches 10, 50 and 100 thousand points, then an extra base is awarded.

Programming notes

First, if you are planning to add more features to this game you will need to be aware of a potential conflict between DEFINESPRITE statements and the program's P-codes. If compiled as listed, the program should end at P-code 2070 (as displayed at the end of the compile). The first DEFINESPRITE position used in the game is 130 (at lines 178 to 181). This represents memory address 2080 in hex. (130 times 64 = 8320 in decimal, 2080 in hex). Therefore the program can only have 16 bytes of P-codes added to it before this DEFINESPRITE statement will overwrite some P-codes. To make more room than that, renumber some sprite positions from 130 upwards to higher, unused, numbers (such as 250 to 255). For the game to still work you will then need to locate references to that sprite position (in ANIMATESPRITE statements or SPRITE statements with the 'POINT' option). For example, sprite position 130 is referred to in lines 711 and 732. Then change the original number to the new sprite number that you allocate. (E.g. change '130' to '250', if '250' is the new number you used in line 178).

You can use the 'Find' command in the Editor to easily locate occurrences of such numbers. Check, however, the context in which the number appears — sometimes the right number might be there but in a different context. For example, the number 130 occurs on line 549 but in this case it refers to the x-coordinate of sprite number eight.

Every sprite position that you renumber to further up in memory will free up 64 bytes of P-codes for program expansion.

Changes you could make to the game

a) Change the game port from 2 to 1 (see line 31).

b) Change the number of bases you are allocated (see line 35).

c) Change the rate at which the base moves in response to the keyboard or joystick (see line 34).

d) Allocate more bases when higher scores are reached or change the threshhold points for the current scores (see lines 125 to 134).

e) Change the colours of the ships and screens (various places in the program).

f) Change the bonus calculation rules (lines 821 to 906).

g) Add sound effects.

Understanding how the game works

Thanks to Pascal's modularity it is easy to follow how the game works. Extensive use is made of constants to make the program more readable. Each procedure has a simple function to perform (indicated by its name) and can generally be studied in isolation. For example DISPLAY_SCORE displays the current game score at the bottom of the screen, and checks for a bonus base allocation.

INIT does the sprite shape initialisation (DEFINESPRITE statements) and asks for the input mode (keyboard, joystick or paddles).

NEW_GAME sets up various variables for a new game (level and wave number etc.) and runs the 'attract' mode. When 'f1' is pressed to start a game, it then asks for a paddle button to be pressed to identify which paddle is in use.

NEW_LEVEL is used at the start of each attack wave. It redraws the screen in the correct colour for that wave and positions the balls in their initial (random) position. There is a deliberate delay of a quarter of a second between drawing each ball on the screen so as to make the game start more

dramatically (see line 547).

GAME is the main game loop – it is called repeatedly from the 'mainline' (lines 941 to 943) until the player runs out of bases. It checks for the end of a wave, moves the base, moves the balls around the screen, checks the 'fire' button, checks for collisions and displays the score.

CHECK_STATUS is used to replace the 'explosion' sprite with the appropriate 'score' sprite at the right time.

SPRITE_ACTIVE is a function that tests the 'display enable' bit in the VIC chip to see whether that sprite is being displayed at the time.

MOVE_BALLS checks to see if a ball has hit the border - if so it makes it 'bounce' off the wall and come back in the other direction (by reversing its X and Y coordinates). It also make the balls speed up at each bounce.

CHECK_BUTTON tests the 'fire' button on the joystick or paddle, or the nominated keyboard 'fire' key. If it is pressed, and a missile is not already in flight, it fires one.

MAKE_MOVE moves the base depending on which input mode is in operation (keyboard, joystick or paddle).

CHECK_COLLSN handles collisions between balls and the missile or base. It first tests FREEZESTATUS and if non-zero proceeds to isolate which sprites were really involved in the collision. To draw an explosion an ANIMATESPRITE command is given, followed by a MOVESPRITE with a zero X and Y increment (so the explosion stays in one spot) – see lines 731 and 732, and lines 757 and 758 for the 'base' explosion logic. If a ball is hit but not killed then this procedure reverses its direction and changes its colour. The nested procedure KILL_SPRITE is used to draw the explosion of a ball, and add in the appropriate score.

CALC_BONUS does the bonus calculations at the end of each wave.

Keying in the game

Typing in this game into your Commodore 64 should not be very difficult. Once again we have used our special 'printing' program to type the game out direct from the disk file. This program 'slashes' zeroes to distinguish them from the letter 'O', and types reserved words in upper case to make the program easier to read. DO NOT BOTHER to type in the reserved words in upper case – they are converted to lower case anyway internally. As mentioned in previous issues the 'underscore' character is entered as a 'left-arrow' above the 'CTRL' key. Also be careful to distinguish between the letter 'I' and the number '1' (they look somewhat different in the listing).

Lines 511 and 512 have 20 spaces between the quote symbols.

As mentioned earlier in this article, if you have typed in the program correctly it should compile with the P-codes ending at address 2070, as displayed at the end of the compile. As a general rule the exact number of spaces entered on a line will not matter (except between quotes of course), but in certain cases it will. For example, in line 158 there must be a space between the '+' and the '1', otherwise you will get a compile error. It is preferable to type in the program exactly as shown. If you get a compile error check carefully the line in error (and the previous line – you may have omitted a semicolon). If you get an 'Undeclared Identifier' message you may have mis-spelled a variable o constant near the start of the program.

Line 1 of the program is important as it forces the P-codes to be placed at address $800, thus allowing a larger program than would fit without it.

The only really tedious part of the program to key in is the DEFINESPRITE statements – it is helpful to have a friend read the code back to you after you have entered it to check for transposed numbers etc. If your sprites look a bit strange when the game runs, check the DEFINESPRITE statements.

Credits

This game was largely written by Sue Gobbett, with the bonus calculation section, and multiple input mode (keyboard, joystic paddles) added by Nick Gammon.

Permission to distribute

```
(* %a $800 [P-codes at $800] *)

(*  Balls Up!  - Game for C64
    Written in G-Pascal
    by Sue Gobbett and
       Nick Gammon.

    Copyright 1985 Gambit Games.
*)

CONST

  (* general *)
  true = 1;
  false = 0;
  on = true;
  off = false;
  home = 147;
  inverse = 18;
  normal = 146;
  alive = 1;
  angry = 2;
  killer = 3;
  exploding = 4;
  dead = 5;
  hours = 4;
  minutes = 3;
  seconds = 2;
  tenths = 1;
  disable_case = 8;
  gameport = 2; (* which game port *)
  keypress = $c5;
  no_key = 64;
  response = 150; (* kbd speed *)
  initial_bases = 3;
  sprite_display_enable = $d015;

  spritey8 = 224;
  starty7 = 218;

  (* graphics *)
  display = 6;
  charcolour = 10;
  spritecolour0 = 16;
  spritecolour1 = 17;
  colour = 1;
  point = 2;
  multicolour = 3;
  expandx = 4;
  expandy = 5;
  behindbackground = 6;
  active = 7;
  border = 11;
  background = 12;

  (* colours *)
  black = 0;
  white = 1;
  red = 2;
  cyan = 3;
  purple = 4;
  green = 5;
  blue = 6;
  yellow = 7;
  orange = 8;
  brown = 9;
  light_red = 10;
  dark_grey = 11;
  medium_grey = 12;
  light_green = 13;
  light_blue = 14;
  light_grey = 15;

  (* function keys *)
  f1 = $85;
  f3 = $86;
  f5 = $87;
  f7 = $88;

  (* input modes *)
  joystick_input = f1;
  paddle_input   = f3;
  keyboard_input = f5;

  (* sound effects *)
  frequency = 1;
  delay = 3;
  pulse = 13;
  noise = 14;

VAR
  input_mode,
  paddleno,
  wave,
  extra_ship,
  shots,
  hits,
  bonus,
  time_taken,
  bases,
  level,
  end_game,
  game_score,
  hi_score,
  killed
     :INTEGER ;
  left_key, right_key,
     fire_key : CHAR ;
  score,points,
  clr,ptr1,ptr2,ptr3,ptr4,ptr5
  :ARRAY [5] OF INTEGER ;
  status,
  xinc,yinc
     :ARRAY [10] OF INTEGER ;

FUNCTION sprite_active(i);
```

```
117 BEGIN
118 sprite_active :=
119 MEMC [sprite_display_enable]
120    AND (1 SHL (i - 1))
121 END ;
122
123 PROCEDURE display_score;
124 BEGIN
125 IF ((game_score >= 10000)
126     AND (extra_ship = 0))
127 OR ((game_score >= 50000)
128     AND (extra_ship = 1))
129 OR ((game_score >= 100000)
130     AND (extra_ship = 2)) THEN
131 BEGIN
132   bases := bases + 1;
133   extra_ship := extra_ship + 1
134 END ;
135 IF game_score > hi_score THEN
136   hi_score := game_score;
137 WRITE (CHR (normal));
138 CURSOR (25, 1);
139 WRITE ("Score ",game_score,
140        "   ");
141 CURSOR (25, 15);
142 WRITE ("Bases: ",bases);
143 CURSOR (25, 24);
144 WRITE ("Hi-Score ",hi_score)
145 END ;
146
147 PROCEDURE init;
148 VAR i, j
149 :INTEGER ;
150
151 PROCEDURE set_ptrs(a,b,c,d,e);
152 BEGIN
153 ptr1[i] := a;
154 ptr2[i] := b;
155 ptr3[i] := c;
156 ptr4[i] := d;
157 ptr5[i] := e;
158 i := i + 1
159 END ;
160
161 BEGIN
162 VOICE (
163   3,frequency,10000,
164   3,noise,on);
165 DEFINESPRITE (174, (* defender *)
166   $0,$0,$c00,
167   $00ccc0,$03ccf0,$0ffffc,
168   $3ffffff,$3ff3ff,$0fc0fc);
169 DEFINESPRITE (150, (* defendr 2 *)
170   $0,$0,$400,
171   $00c4c0,$03ccf0,$0ffffc,
172   $3ffffff,$3ff3ff,$0fc0fc);
173 DEFINESPRITE (175, (* missile *)
174   $1000,$1000,$1000,
175   $1000,$1000,$1000,
176   $1000,$1000,$1000,
177   $1000,$1000,$1000);
178 DEFINESPRITE (130, (* explsn 1 *)
179   0,0,0,0,0,0,0,0,0,0,
180   $3300,$3300,$c00,$c00,
181   $3300,$3300);
182 DEFINESPRITE (131, (* explsn 2 *)
183   0,0,0,0,0,0,0,0,
184   $c0c0,$c0c0,$3300,$3300,
185   0,0,
186   $3300,$3300,$c0c0,$c0c0);
187 DEFINESPRITE (132, (* explsn 3 *)
188   0,0,0,0,0,0,
189   $30030,$30030,0,0,
190   $18060,$18060,$1b00,$1b00,
191   $18060,$18060,0,0,
192   $30030,$30030);
193 DEFINESPRITE (152, (* explsn 4 *)
194   0,0,0,
195   $0c000c,$0c000c,0,0,
196   $180006,$180006,0,0,
197   $60c0,$60c0,00,
198   $180006,$180006,0,0,
199   $0c000c,$0c000c);
200 DEFINESPRITE (151, (* explsn 5 *)
201   $c00003,$c00003,0,0,
202   $600000,$600006,$6,0,0,
203   $60030,$60030,0,0,0,0,
204   $6,$60006,$600000,0,0,
205   $c00003);
206 DEFINESPRITE (133,  (* balls *)
207   $002800,$03fa80,$08fea0,
208   $0abfa0,$3eafe8,$3fabf8,
209   $2feafc,$2bfabc,$2afeac,
210   $3abfa8,$3eafe8,$0fabf0,
211   $0feaf0,$03fa80,$003c00);
212 DEFINESPRITE (134,
213   $003800,$02fe80,$0abf80,
214   $0eafe0,$3fabf8,$2feafc,
215   $2bfabc,$2afeac,$3abfa8,
216   $3eafe8,$3fabf8,$0feaf0,
217   $0bfab0,$02fe80,$003c00);
218 DEFINESPRITE (135,
219   $003c00,$02bf80,$0eafe0,
220   $0fabf0,$2feafc,$2bfabc,
221   $2afeac,$3abfa8,$3eafe8,
222   $3fabf8,$2feafc,$0bfab0,
223   $0afea0,$02bf80,$002c00);
224 DEFINESPRITE (136,
225   $003c00,$02afc0,$0fabf0,
226   $0feaf0,$2bfabc,$2afeac,
227   $3abfa8,$3eafe8,$3fabf8,
228   $2feaf8,$2bfabc,$0afea0,
229   $0abfa0,$02afc0,$002800);
230 DEFINESPRITE (137,
231   $002c00,$03abc0,$0feaf0,
232   $0bfab0,$2afeac,$3abfa8,
```

```
233   $3eafe8,$3fabf8,$2feafc,
234   $2bfabc,$2afeac,$0abfa0,
235   $0eafe0,$03abc0,$002800);
236 DEFINESPRITE (138,
237   $002800,$03eac0,$0bfab0,
238   $0afea0,$3abfa8,$3eafe8,
239   $3eabf8,$2feafc,$2bfabc,
240   $2afeac,$3abfa8,$0eafa0,
241   $0fabf0,$03eac0,$003800);
242 DEFINESPRITE (139,   (* scores *)
243   $e700,$8500,$e500,$2500,$e700);
244 DEFINESPRITE (140,
245   $9ce0,$94a0,$94a0,$94a0,$9ce0);
246 DEFINESPRITE (141,
247   $e738,$2528,$e528,$8528,$e738);
248 DEFINESPRITE (142,
249   $e738,$8528,$e528,$2528,$e738);
250 DEFINESPRITE (168, (* 'b' *)
251   $ff0,$f38,$f1c,
252   $f1c,$f38,$ff0,
253   $f38,$f1c,$f1c,
254   $f3c,$f7c,$f7c,
255   $e78,$c70);
256 DEFINESPRITE (169, (* 'a' *)
257   $020,$070,$0f8,
258   $178,$23c,$43c,
259   $c3c,$ffc,$c3c,
260   $e3c,$fbc,$fbc,
261   $7bc,$3bc);
262 DEFINESPRITE (170, (* 'l' *)
263   $f00,$f00,$f00,
264   $f00,$f00,$f00,
265   $f00,$f00,$e38,
266   $c78,$878,$878,
267   $c38,$ff8);
268 DEFINESPRITE (171, (* 's' *)
269   $3fc,$71c,$e3c,
270   $e3c,$f1c,$fc0,
271   $7f8,$3fc,$07c,
272   $e3c,$f3c,$f3c,
273   $e78,$ff0);
274 DEFINESPRITE (172, (* 'u' *)
275   $f1e,$f1e,$f1e,
276   $f1e,$f1e,$f1e,
277   $f1e,$f1e,$f1e,
278   $f1e,$f1e,$f3e,
279   $7de,$39e);
280 DEFINESPRITE (173, (* 'p' *)
281   $c70,$e78,$f7c,
282   $f7c,$f1c,$f0c,
283   $f0c,$f98,$f70,
284   $f00,$f00,$f00,
285   $f00,$f00);
286 i := 1;
287 FOR j := 1 TO 5 DO
288    set_ptrs(133,134,135,136,137);
289 clr[1] := cyan;
290 clr[2] := yellow;
291 clr[3] := white;
292 clr[4] := red;
293 score[1] := 50;
294 score[2] :=100;
295 score[3] :=200;
296 score[4] := 500;
297 points[1] := 139;
298 points[2] := 140;
299 points[3] := 141;
300 points[4] := 142;
301 GRAPHICS (charcolour,yellow);
302 WRITE (CHR (disable_case));
303 hi_score := 0;
304 game_score := 0;
305 bases := 0;
306 GRAPHICS (background, dark_grey,
307           border, dark_grey);
308 WRITE (CHR (home));
309 CURSOR (7, 10);
310 WRITELN ("----- BALLS UP -----");
311 CURSOR (10, 10);
312 WRITELN (CHR (inverse), "f1",
313          CHR (normal),
314          "  Joystick - port ",
315          gameport);
316 CURSOR (12, 10);
317 WRITELN (CHR (inverse), "f3",
318          CHR (normal),
319          "  Paddles  - port ",
320          gameport);
321 CURSOR (14, 10);
322 WRITELN (CHR (inverse), "f5",
323          CHR (normal),
324          "  Keyboard");
325 REPEAT
326   input_mode := GETKEY
327 UNTIL (input_mode = joystick_input)
328    OR (input_mode = paddle_input)
329    OR (input_mode = keyboard_input);
330 fire_key := 255;  (* none yet *)
331 IF input_mode = keyboard_input THEN
332   BEGIN
333   REPEAT
334     UNTIL MEMC [keypress] = no_key;
335   CURSOR (16, 10);
336   WRITE ("Press LEFT key: ");
337   REPEAT
338     left_key := MEMC [keypress]
339   UNTIL left_key <> no_key;
340   REPEAT
341     UNTIL MEMC [keypress] = no_key;
342   CURSOR (18, 10);
343   WRITE ("Press RIGHT key: ");
344   REPEAT
345     right_key := MEMC [keypress]
346   UNTIL (right_key <> no_key)
347     AND (right_key <> left_key);
348   REPEAT
```

```
349     UNTIL MEMC [keypress] = no_key;
350   CURSOR (20, 10);
351   WRITE ("Press FIRE key: ");
352   REPEAT
353     fire_key := MEMC [keypress]
354   UNTIL (fire_key <> no_key)
355     AND (fire_key <> left_key)
356     AND (fire_key <> right_key);
357   REPEAT
358     UNTIL MEMC [keypress] = no_key
359   END
360 END ;
361
362 PROCEDURE new_game;
363 VAR i
364 :INTEGER ;
365 reply : CHAR ;
366 BEGIN
367 FOR i := 1 TO 8 DO
368   BEGIN
369   STOPSPRITE (i);
370   SPRITE (i,active,off)
371   END ;
372 end_game := false;
373 level := 0;
374 wave := 0;
375 killed := 6;
376 WRITE (CHR (home));
377 SPRITE (1, point, 168,
378         2, point, 169,
379         3, point, 170,
380         4, point, 170,
381         5, point, 171,
382         6, point, 172,
383         7, point, 173);
384 GRAPHICS (border, orange,
385           background, orange,
386           charcolour, brown);
387 SPRITEFREEZE (0);
388 FOR i := 1 TO 7 DO
389   BEGIN
390   SPRITE (
391           i, expandx, off,
392           i, expandy, off,
393           i, multicolour, off);
394   MOVESPRITE (i, 139, 230,
395    ((107 + (i + (i >= 6)) * 15)
396    - 150) * 256 / 400,
397    -128 * 300 / 400, 400)
398   END ;
399 display_score;
400 REPEAT
401   i := i + 1;
402   SPRITE (i MOD 7 + 1, colour,
403           i);
404   WAIT (260);
405   reply := GETKEY
406 UNTIL (NOT SPRITESTATUS (1))
407    OR (reply = f1);
408 FOR i := 1 TO 7 DO
409   SPRITE (i, colour, yellow);
410 IF reply <> f1 THEN
411    BEGIN
412    CURSOR (9, 13);
413    WRITE ("by  Sue Gobbett");
414    CURSOR (11, 13);
415    WRITE ("and Nick Gammon");
416    CURSOR (18, 1);
417    WRITE ("Extra base at 10,000  ");
418    WRITE ("50,000 and 100,000");
419    CURSOR (16, 10);
420    WRITE ("Press f1 to start game");
421    SPRITE (8, multicolour, on,
422            8, colour, white,
423            8, expandx, off,
424            8, behindbackground, on,
425            8, expandy, off);
426    POSITIONSPRITE (8, 120, 120);
427    xinc [8] := 512;
428    yinc [8] := 512;
429    ANIMATESPRITE (8, 3, 133, 134,
430                   135, 136, 137)
431    REPEAT
432      IF SPRITEX (8) < 28 THEN
433         xinc [8] := 512
434      ELSE
435      IF SPRITEX (8) > 317 THEN
436         xinc [8] := -512
437      ELSE
438      IF SPRITEY (8) < 58 THEN
439         yinc [8] := 512
440      ELSE
441      IF SPRITEY (8) > 220 THEN
442         yinc [8] := -512;
443      MOVESPRITE (8, SPRITEX (8),
444                     SPRITEY (8),
445                     xinc [8],
446                     yinc [8], 500)
447    UNTIL GETKEY = f1
448    END ;
449 game_score := 0;
450 bases := initial_bases;
451 extra_ship := 0;
452 IF input_mode = paddle_input THEN
453   BEGIN
454   WRITE (CHR (home));
455   FOR i := 1 TO 8 DO
456     SPRITE (i,active,off);
457   GRAPHICS (border, light_grey,
458             background, light_grey,
459             charcolour, blue);
460   CURSOR (15, 1);
461   WRITE ("Press fire button to ");
462   WRITELN ("select paddle ...");
463   paddleno := 0;
464   REPEAT
```

```
465     CASE JOYSTICK (gameport) OF
466       4: paddleno := 1;
467       8: paddleno := 2
468     END
469   UNTIL paddleno > 0
470   END
471 END ;
472
473 PROCEDURE new_level;
474 VAR i,j,k
475 :INTEGER ;
476 BEGIN
477 FOR i := 1 TO 8 DO
478   SPRITE (i,active,off);
479 SPRITEFREEZE (0);
480 GRAPHICS (display, off);
481 CASE level OF
482   1: BEGIN
483       GRAPHICS (
484         charcolour, light_green,
485         background, green,
486         border, green)
487      END ;
488   2: BEGIN
489       GRAPHICS (
490         charcolour, light_blue,
491         background, blue,
492         border, blue)
493      END ;
494   3: BEGIN
495       GRAPHICS (
496         charcolour, orange,
497         background, brown,
498         border, brown)
499      END ELSE
500      BEGIN
501       GRAPHICS (
502         charcolour, light_red,
503         background, red,
504         border, red)
505      END
506 END ; (* of case *)
507 FOR i := 1 TO 24 DO
508 BEGIN
509   CURSOR (i, 1);
510   WRITE (CHR (inverse),
511     "                    ",
512     "                    ")
513 END ;
514 GRAPHICS (
515   charcolour,white,
516   display,on);
517 display_score;
518 IF killed = 6 THEN
519    IF level < 4 THEN
520      FOR i := 1 TO 6 DO
521        status[i] := 4 - level
522    ELSE
523      FOR i := 1 TO 6 DO
524        status[i] :=
525          (RANDOM MOD 3) + 1;
526 status[7] := dead;
527 GRAPHICS (
528   spritecolour1,black,
529   spritecolour0,white);
530 FOR i := 1 TO 6 DO
531   IF status [i] <> dead THEN
532     BEGIN
533     STOPSPRITE (i);
534     POSITIONSPRITE (i,
535          (RANDOM MOD 220) + 40,
536          (RANDOM MOD 50) + 80);
537     ANIMATESPRITE (i,3,
538       ptr1[level],ptr2[level],
539       ptr3[level],ptr4[level],
540       ptr5[level]);
541     SPRITE (
542       i,multicolour,on,
543       i,colour,clr[status[i]],
544       i,expandx,off,
545       i,expandy,off,
546       i,active,on);
547     SOUND (delay, 25)
548     END ;
549 POSITIONSPRITE (8,130,spritey8);
550 SPRITE (
551   7,point,175,
552   7,colour,white,
553   7,expandx,on,
554   8,point,150,
555   8,colour,white,
556   8,multicolour,on,
557   8,expandx,on,
558   8, behindbackground, off,
559   8,expandy,on,
560   8,active,on);
561 FOR i := 1 TO 6 DO
562   IF status [i] <> dead THEN
563     BEGIN
564     IF killed >= 6 THEN
565       BEGIN
566       xinc[i] := (15 - i) * 20;
567       yinc[i] := (7 + i) * 20
568       END ;
569     IF (RANDOM MOD 100) > 49 THEN
570        xinc[i] := - xinc[i];
571     IF i MOD 2 = 0 THEN
572        yinc[i] := - yinc[i];
573     MOVESPRITE (i,
574       SPRITEX (i),SPRITEY (i),
575       xinc[i],yinc[i],500)
576     END ;
577 SPRITEFREEZE ($c0);
578 end_game := false;
579 IF killed >= 6 THEN
580    killed := 0
```

```
581 END ;
582
583 PROCEDURE move_balls;
584 VAR i,j
585 :INTEGER ;
586 BEGIN
587 FOR i := 1 TO 6 DO
588  IF status[i] <> dead THEN
589  IF status[i] <> exploding THEN
590  IF SPRITESTATUS (i) <> 0 THEN
591   BEGIN
592   IF SPRITEX (i) < 28 THEN
593     IF xinc[i] < 0 THEN
594       xinc[i] := 10 - xinc[i]
595     ELSE
596   ELSE
597     IF SPRITEX (i) > 317 THEN
598       IF xinc[i] > 0 THEN
599         xinc[i] := -10 - xinc[i];
600   IF SPRITEY (i) < 58 THEN
601     IF yinc[i] < 0 THEN
602       yinc[i] := 10 - yinc[i]
603     ELSE
604   ELSE
605     IF SPRITEY (i) > 220 THEN
606       IF yinc[i] > 0 THEN
607         yinc[i] := -10 - yinc[i];
608   IF SPRITESTATUS (i) <> 0 THEN
609     MOVESPRITE (i,
610       SPRITEX (i),SPRITEY (i),
611       xinc[i],yinc[i],500)
612   END
613 END ;
614
615 PROCEDURE check_button;
616 BEGIN
617 IF (status[7] = dead) OR
618    (SPRITEY (7) < 30) THEN
619   BEGIN
620   SPRITE (8,point,150);
621   IF (JOYSTICK (gameport) > 15) OR
622    ((input_mode = paddle_input) AND
623      (JOYSTICK (gameport) <> 0)) OR
624    (MEMC [keypress] = fire_key) THEN
625     BEGIN (* fired it! *)
626     SPRITE (8,point,174);
627     MOVESPRITE (7,SPRITEX (8),
628       (spritey8 - 12),
629       0,-1000,50);
630     status[7] := alive;
631     shots := shots + 1
632     END
633   END
634 END ;
635
636 PROCEDURE make_move;
637 VAR i,move
638 :INTEGER ;
639 BEGIN ;
640 move := 0;
641 CASE input_mode OF
642   paddle_input:
643     CASE paddleno OF
644       1: move :=
645   (289 - (PADDLE (gameport) AND $ff))
646   - SPRITEX (8);
647       2: move :=
648   (289 - (PADDLE (gameport) SHR 8))
649   - SPRITEX (8)
650     END ;
651   joystick_input:
652     CASE JOYSTICK (gameport) AND 12 OF
653     4: move := - response; (* left *)
654     8: move :=   response  (* right *)
655     END ;
656   keyboard_input:
657     CASE MEMC [keypress] OF
658     left_key:  move := - response;
659     right_key: move :=   response
660     END
661   END ;
662 IF ABS (move) < 3 THEN
663   move := 0
664 ELSE
665   BEGIN
666   IF move < 0 THEN
667     move := move - 30
668   ELSE
669     move := move + 30;
670   IF ABS (move) > 36 THEN
671     move := move * 6
672   END ;
673 IF (FREEZESTATUS AND $80) = 0 THEN
674    MOVESPRITE (8,
675      SPRITEX (8),spritey8,
676      move,0,100);
677 check_button
678 END ;
679
680 PROCEDURE check_status;
681 VAR i
682 :INTEGER ;
683 BEGIN
684 FOR i := 1 TO 7 DO
685  IF sprite_active(i) THEN
686   IF (status[i] = exploding) AND
687      (SPRITESTATUS (i) = 0) THEN
688     BEGIN
689     SPRITE (
690       i,colour,white,
691       i,multicolour,off,
692       i,expandx,on,
693       i,point,points[level]);
694     MOVESPRITE (i,SPRITEX (i),
695       SPRITEY (i),0,0,60);
696     status[i] := dead
```

```
    END
  ELSE
  IF (status[i] = dead) AND
     (SPRITESTATUS (i) = 0) THEN
    SPRITE (i,active,off)
END ;

PROCEDURE check_collsn;
VAR i,j,k,endloop,collision
:INTEGER ;

PROCEDURE kill_sprite(i);
BEGIN
ANIMATESPRITE (i,
  12,130,131,132,152,151);
SPRITE (i, multicolour, off,
        i, colour,black);
MOVESPRITE (i,
  SPRITEX (i),SPRITEY (i),0,0,50);
status[i] := exploding;
game_score :=
  game_score + score[level];
killed := killed + 1
END ;

BEGIN
IF FREEZESTATUS <> 0 THEN
  BEGIN
  IF (FREEZESTATUS AND $80)
        THEN
    BEGIN
    STOPSPRITE (8);
    SPRITE (8, multicolour, off,
            8, colour, black);
    ANIMATESPRITE (8,
      24,130,131,132,152,151);
    FOR i := 1 TO 8 DO
    IF status [i] <> exploding THEN
      STOPSPRITE (i);
    i := SPRITEX (8);
    j := SPRITEY (8);
    k := 1;
    endloop := false;
    REPEAT
      IF NOT endloop THEN
      IF (status[k] <> dead) AND
         (status[k] <> exploding)
         THEN
      IF (FREEZESTATUS AND
        (1 SHL (k - 1))) <> 0 THEN
      IF i >= SPRITEX (k) - 45 THEN
      IF i < SPRITEX (k) + 25 THEN
      IF j >= SPRITEY (k) - 15 THEN
      IF j < SPRITEY (k) + 15 THEN
        BEGIN
        kill_sprite (k);
        endloop := true
        END ;
      k := k + 1
    UNTIL (k > 6) OR endloop;
    MOVESPRITE (8,SPRITEX (8),
      spritey8 - 20,0,0,100);
    CURSOR (12, 15);
    IF bases = 1 THEN
       WRITE ("GAME OVER");
    REPEAT
      check_status
    UNTIL NOT SPRITESTATUS (8);
    SPRITE (8, active, off);
    end_game := true
    END
  ELSE
    BEGIN
    i := SPRITEX (7);
    j := SPRITEY (7);
    k := 1;
    endloop := false;
    REPEAT
      IF NOT endloop THEN
      IF (status[k] <> dead) AND
         (status[k] <> exploding)
         THEN
      IF (FREEZESTATUS AND
        (1 SHL (k - 1))) <> 0 THEN
      IF i >= SPRITEX (k) - 25 THEN
      IF i < SPRITEX (k) + 5  THEN
      IF j >= SPRITEY (k) - 15 THEN
      IF j < SPRITEY (k) + 25 THEN
        BEGIN
        SPRITE (7,active,off);
        status[7] := dead;
        hits := hits + 1;
        IF status[k] = killer THEN
          kill_sprite(k)
        ELSE
          BEGIN
          status[k] :=
            status[k] + 1;
          SPRITE (k,colour,
            clr[status[k]]);
          xinc[k] :=
            xinc[k] * 5 / 4;
          yinc[k] :=
            -yinc[k] * 5 / 4;
          MOVESPRITE (k,
            SPRITEX (k),SPRITEY (k),
            xinc[k],yinc[k],500)
          END ;
        endloop := true
        END ;
      STARTSPRITE (k);
      k := k + 1
    UNTIL k > 6;
    STARTSPRITE (7)
    END ;
  SPRITEFREEZE ($c0);
```

```
  IF NOT end_game THEN
    BEGIN
    make_move;
    move_balls
    END
  END
END ;

PROCEDURE calc_bonus;
VAR i, ratio : INTEGER ;
BEGIN
  IF level > 1 THEN
  BEGIN
    FOR i := 1 TO 8 DO
       SPRITE (i, active, off);
    WRITE (CHR (home));
    GRAPHICS (border, orange,
              background, orange,
              charcolour, brown);
    i := CLOCK (hours); (* freeeze *)
    time_taken := CLOCK (minutes)
                  * 60 +
                  CLOCK (seconds);
    bonus := (15 * (level - 1)
           - time_taken)
           * 100 * (level - 1);
    IF bonus < 0 THEN
       bonus := 0;
    game_score := game_score + bonus;
    CURSOR (4, 7);
    WRITELN ("Wave ",wave,".");
    CURSOR (6, 7);
    WRITELN
    ("Time to complete this wave");
    CURSOR (8, 10);
    WRITE ("was");
    IF CLOCK (minutes) > 0 THEN
    BEGIN
      WRITE (" ",
             CLOCK (minutes),
             " minute");
      IF CLOCK (minutes) > 1 THEN
        WRITE ("s")
    END ;
    IF CLOCK (seconds) > 0 THEN
       BEGIN
       WRITE (" ",
                CLOCK (seconds),
                " second");
       IF CLOCK (seconds) > 1 THEN
         WRITE ("s")
       END ;
    WRITELN (".");
    CURSOR (10, 7);
    IF bonus = 0 THEN
       WRITELN ("No time bonus.")
    ELSE
       BEGIN
       GRAPHICS (charcolour, red);
       WRITELN ("Time bonus is ",
                bonus,
             " points.");
       GRAPHICS (charcolour,brown)
       END ;
    i := CLOCK (tenths);
    ratio := hits * 100 / shots;
    CURSOR (12, 7);
    WRITELN ("Hits scored: ",hits);
    CURSOR (14, 7);
    WRITELN ("Shots fired: ",shots);
    CURSOR (16, 7);
    WRITELN ("Hit/Shot ratio: ",
             ratio,"%");
    IF ratio < 67 THEN
       ratio := 0;
    bonus := ratio * 10 * (level - 1);
    game_score := game_score + bonus;
    CURSOR (18, 7);
    IF bonus = 0 THEN
       WRITELN ("No accuracy bonus.")
    ELSE
       BEGIN
       GRAPHICS (charcolour, red);
       WRITELN ("Accuracy bonus is ",
             bonus," points.");
       GRAPHICS (charcolour,brown)
       END ;
    display_score;
    SOUND (delay, 800)
   END ;
   SETCLOCK (0,0,0,0);
   shots := 0;
   hits := 0
END ;

PROCEDURE game;
VAR i,j,k
:INTEGER ;
BEGIN
IF killed > 5 THEN
BEGIN
  IF level < 4 THEN
    level := level + 1;
  IF level <> 1 THEN
  FOR i := 1 TO 6 DO
    REPEAT UNTIL SPRITESTATUS (i) = 0;
  calc_bonus;
  wave := wave + 1
END ;
IF (killed > 5)
OR ((bases > 0) AND end_game) THEN
  new_level;
make_move;
move_balls;
check_button;
check_collsn;
```

```
929 check_button;
930 check_status;
931 IF end_game THEN
932   bases := bases - 1;
933 check_button;
934 display_score
935 END ;
936
937 BEGIN
938 init;
939 REPEAT
940   new_game;
941   REPEAT
942     game
943   UNTIL bases <= 0
944 UNTIL false
945 END .
```

ADVENTURE GAME — 'MELEE'

If you are fond of role-playing games, such as 'Dungeons and Dragons', or their various computer equivalents ('Adventure', 'Zork' and so on) then the program over the page may interest you.

It is not a full-scale game as such, but shows a method of adjudicating fights that may occur within a dungeon (known as 'melees').

Most role-playing games involve the use of multi-sided dice to provided probability rolls (probability that a blow connects, or causes damage etc.). This is simulated in this program by the function ROLL, which rolls the specified type of dice the specified number of times. For example, if a game alls for 2d8 (2 rolls of an 8-sided die) then you would use ROLL (2, 8).

The procedure GENERATECHAR uses the dice rolls to generate a character with random characteristics, the same as in the preparation for a real game. It makes various adjustments to characteristics depending on other, related, characteristics. For example, characters with high dexterity gain an increase in their 'attack adjustment'.

Lines 118 to 126 demonstrate how to pass many arguments to a procedure — it is not necessary to put each 'dummy' argument on a new line, as we have done here, but doing that allows the use of comments to show what each argument means.

The procedure DISPLAYCHAR just displays a character's characteristics. Some of them, such as 'gold' are not used further in this program, but could be incorporated in a more elaborate game that you could write yourself.

The procedure GENERATEMONSTER just creates a 'monster' of a random type. It supplies parameters such as level, hit die, armour class and so on to the GENERATECHAR procedure for the actual monster generation. If you want to generate different monsters then just change the appropriate parameters.

The constant SHOW_MONSTERS controls whether the newly generated monsters are displayed at the start of the game.

Lines 19 to 40 of the program all define characteristics for each character, and are all integer arrays. The constant MAXCHARS defines the maximum number of array elements (and therefore the maximum number of characters).

The procedure MELEE actually conducts one round of fighting. It uses the sub-procedure HIT to actually calculate the outcome and amount of damage caused by a single blow. In one round, however, both characters might hit each other, or only one character might hit the other, depending on whether one of them was surprised. If neither is surprised then the first blow is governed by their respective dexterity. (The order of hits is important of course — a character might not survive the first blow in order to retaliate).

The procedure ENCOUNTER adjudicates the whole encounter with the monster. It calculates the initial surprise factor, then allows the human player to decide whether to fight, parry or run away. Using the 'Info' selection allows inspection of both the human's and monster's characteristics as the fight progresses.

The main program just generates the characters (the human gets to choose his/her 'level') and then keeps generating encounters until the human dies (a bit one-sided, this.)

You could adapt this program to assist in a real role-playing game by making slight changes so that at the start of the program you specify the actual characteristics of the players and monsters in your game, and then conduct a melee whenever the real game calls for it. This could save rolling a lot of dice.

```
(* Melee adjudication game.

   Written by Nick Gammon.

   May be copied for non-profit
      making purposes.

*)

CONST true = 1;
      false = 0;
      maxchars = 20;
      cr = 13;
      home = 147;
      show_monsters = false;
      maxlevel = 9;

VAR
  hp,  (* hit points *)
  ac,  (* armour class *)
  hitdie,
  dexterity,
  tohit,
  damagetimes,
  damagedie,
  damageplus,
  level,
  maxhp,
  constitution,
  hp_adjustment,
  attack_adjustment,
  defence_adjustment,
  experience,
  strength,
  iq,
  wisdom,
  charisma,
  type,
  gold
    : ARRAY [maxchars] OF INTEGER ;

  i, j  : INTEGER ;
  reply : CHAR ;

PROCEDURE name (x);
BEGIN
  CASE type [x] OF
   1: WRITE ("Human");
  10: WRITE ("Berserker");
  11: WRITE ("Bandit");
  12: WRITE ("Black Pudding");
  13: WRITE ("Bugbear");
  14: WRITE ("Chimera");
  15: WRITE ("Cockatrice");
  16: WRITE ("Doppleganger");
  17: WRITE ("White Dragon");
  18: WRITE ("Black Dragon");
  19: WRITE ("Red Dragon")
  END  (* of case *)
END ;

FUNCTION roll (times, die);
VAR cumulative, i : INTEGER ;
BEGIN
  cumulative := 0;
  FOR i := 1 TO times DO
    cumulative := cumulative +
                  (RANDOM MOD die + 1);
  roll := cumulative
END ;

PROCEDURE displaychar (x);
BEGIN
WRITE (CHR (home),"------------ ");
name (x);
WRITE (" ");
REPEAT
  WRITE ("-")
UNTIL CURSORX >= 40;
WRITELN ;
WRITELN ;
WRITELN
("Strength:         ", strength [x]);
WRITELN
("IQ:               ", iq [x]);
WRITELN
("Wisdom:           ", wisdom [x]);
WRITELN
("Constitution:     ", constitution[x]);
WRITELN
("Dexterity:        ", dexterity [x]);
WRITELN
("Charisma:         ", charisma [x]);
WRITELN
("Max. hit points:  ", maxhp [x]);
WRITELN
("Hit points:       ", hp [x]);
WRITELN
("Level:            ", level [x]);
WRITELN
("Armour class:     ", ac [x]);
WRITELN
("Roll to hit AC 0: ", tohit [x]);
WRITE
("Damage:           ",
  damagetimes [x],"d",damagedie [x]);
IF damageplus [x] > 0 THEN
  WRITE (" + ", damageplus [x]);
WRITELN ;
WRITELN
("Experience:       ", experience [x]);
WRITELN
("Gold:             ", gold [x]);
WRITELN ;
WRITELN
END ;
```

```
117
118 PROCEDURE generatechar
119  (x,       (* character number *)
120   typ,     (* type of character *)
121   lvl,     (* level *)
122   hd,      (* hit die *)
123   armour,  (* armour class *)
124   dt,      (* damage times *)
125   dd,      (* damage die *)
126   dp);     (* damage plus *)
127
128 VAR i, j : INTEGER ;
129 BEGIN
130   REPEAT
131     type [x] := typ;
132     damagetimes [x] := dt;
133     damagedie [x] := dd;
134     damageplus [x] := dp;
135     ac [x] := armour;
136     tohit [x] := 12 - lvl;
137     IF tohit [x] < 0 THEN
138        tohit [x] := 0;
139     experience [x] := 0;
140     dexterity [x] := roll (3, 6);
141     attack_adjustment [x] := 0;
142     CASE dexterity [x] OF
143      18: attack_adjustment [x] := 3;
144      17: attack_adjustment [x] := 2;
145      16: attack_adjustment [x] := 1;
146       5: attack_adjustment [x] := -1;
147       4: attack_adjustment [x] := -2;
148       3: attack_adjustment [x] := -3
149     END ;  (* of case *)
150     defence_adjustment [x] := 0;
151     CASE dexterity [x] OF
152      18: defence_adjustment [x] := -4;
153      17: defence_adjustment [x] := -3;
154      16: defence_adjustment [x] := -2;
155      15: defence_adjustment [x] := -1;
156       6: defence_adjustment [x] := 1;
157       5: defence_adjustment [x] := 2;
158       4: defence_adjustment [x] := 3;
159       3: defence_adjustment [x] := 4
160     END ;  (* of case *)
161     constitution [x] := roll (3, 6);
162     hp_adjustment [x] := 0;
163     CASE constitution [x] OF
164      18: hp_adjustment [x] := 4;
165      17: hp_adjustment [x] := 3;
166      16: hp_adjustment [x] := 2;
167      15: hp_adjustment [x] := 1;
168       6: hp_adjustment [x] := -1;
169       5: hp_adjustment [x] := -1;
170       4: hp_adjustment [x] := -1;
171       3: hp_adjustment [x] := -2
172     END ;  (* of case *)
173     iq [x] := roll (3, 6);
174     strength [x] := roll (3, 6);
175     wisdom [x] := roll (3, 6);
176     charisma [x] := roll (3, 6);
177     gold [x] := roll (3, 6) * 10;
178     hitdie [x] := hd;
179     level [x] := lvl;
180     maxhp [x] := 0;
181     FOR i := 1 TO lvl DO
182       BEGIN
183       j := roll (1, hitdie [x]);
184       j := j + hp_adjustment [x];
185       IF j < 1 THEN
186         j := 1;
187       maxhp [x] := maxhp [x] + j
188       END ;
189     hp [x] := maxhp [x];
190     IF x = 1 THEN
191       BEGIN
192       displaychar (x);
193       WRITELN ;
194       WRITE ("Keep this character? ");
195       WRITE (" Y = Yes ... ");
196       READ (reply);
197       WRITELN (CHR (reply))
198     END ELSE
199       reply := "y"
200   UNTIL (reply = "y")
201      OR (reply = "Y")
202 END ;
203
204 PROCEDURE generatemonster (x);
205 VAR y, z : INTEGER ;
206 BEGIN
207   z := roll (1, 8);
208    (* for dragon's hit dice *)
209   y := roll (1, 10) + 9;
210    (* monster type *)
211   CASE y OF
212     10: generatechar
213         (x, y,  1, 8, 7, 1,  8, 0);
214     11: generatechar
215         (x, y,  1, 8, 6, 1,  6, 0);
216     12: generatechar
217         (x, y, 10, 8, 6, 3,  8, 0);
218     13: generatechar
219         (x, y,  3, 8, 5, 2,  4, 0);
220     14: generatechar
221         (x, y,  9, 8, 4, 3,  4, 0);
222     15: generatechar
223         (x, y,  5, 8, 6, 1,  6, 0);
224     16: generatechar
225         (x, y,  4, 8, 5, 1, 12, 0);
226     17: generatechar
227         (x, y,  5, z, 2, 4,  6, 0);
228     18: generatechar
229         (x, y,  6, z, 2, 4,  6, 0);
230     19: generatechar
231         (x, y, 11, z, 2, 4,  6, 0)
232   END  (* of case *)
```

```
233 END ;
234
235 PROCEDURE melee (x, y, action);
236 (* character x fights character y *)
237 (* action:  1 = x surprises y
238                2 = y surprises x
239                3 = x parries
240                4 = x runs away    *)
241
242 VAR i, j, bonus : INTEGER ;
243
244 PROCEDURE hit (x, y);
245 (* character x attacks character y *)
246
247 VAR i, j  : INTEGER ;
248
249 BEGIN
250   WRITELN ;
251   WRITELN ;
252   name (x);
253   WRITE (" attacks ");
254   name (y);
255   WRITE (" and ");
256   j := 9 -
257    ac [y] + defence_adjustment [y];
258   j := j + tohit [x];
259   i := roll (1, 20);
260   IF i >= j - bonus THEN
261     BEGIN
262     IF CURSORX > 27 THEN
263       BEGIN
264       WRITELN ;
265       WRITE ("   ")
266       END ;
267     WRITE ("hits ");
268     IF damagetimes [x] > 1 THEN
269       WRITELN (damagetimes [x],
270             " times.")
271     ELSE
272       WRITELN ("once.");
273     i := roll (damagetimes [x],
274               damagedie [x])
275          + damageplus [x];
276     name (y);
277     WRITELN (" takes ",i,
278             " points damage.");
279     hp [y] := hp [y] - i;
280     IF hp [y] < 0 THEN
281        hp [y] := 0;
282     name (y);
283     IF hp [y] = 0 THEN
284        WRITELN (" is killed!!!")
285     ELSE
286        WRITELN (" has ", hp [y],
287                " hp remaining!");
288     END
289   ELSE
290     WRITELN ("misses!")
291 END ;
292
293 (* *** Startof melee procedure *** *)
294
295 BEGIN
296 IF action = 0 THEN (* no surprise *)
297   BEGIN
298   bonus := 0;
299   i := dexterity [x];
300   j := dexterity [y];
301   IF ABS (i - j) < 3 THEN
302     BEGIN
303     i := roll (1, 6);
304     j := roll (1, 6)
305     END ;
306   IF i > j THEN (* first blow? *)
307     BEGIN
308     hit (x, y); (* x hits first *)
309     IF hp [y] > 0 THEN
310       hit (y, x)  (* y retaliates *)
311     END
312   ELSE
313     BEGIN
314     hit (y, x); (* y hits first *)
315     IF hp [x] > 0 THEN
316       hit (x, y)  (* x retaliates *)
317     END
318   END
319 ELSE
320   IF action = 1 THEN
321     BEGIN  (* x surprises y *)
322     bonus := 2;
323     hit (x, y)
324     END
325   ELSE
326   IF action = 2 THEN
327     BEGIN  (* y surprises x *)
328     bonus := 2;
329     hit (y, x)
330     END
331   ELSE
332   IF action = 3 THEN
333     BEGIN  (* x parries *)
334     bonus := -2;
335     hit (y, x)
336     END
337   ELSE
338   IF action = 4 THEN
339     BEGIN  (* x runs *)
340     bonus := 2;
341     hit (y, x)
342     END
343 END ;
344
345 PROCEDURE encounter (y);
346 VAR action, ok_reply : INTEGER ;
347
348 BEGIN
```

```
IF hp [y] > 0 THEN
  BEGIN
  WRITELN ;
  WRITELN ;
  WRITE ("** ",
         "An encounter with a ");
  name (y);
  WRITE (" ");
  REPEAT
    WRITE ("*")
  UNTIL CURSORX >= 40;
  WRITELN ;
  WRITELN ;
  action := 0;
  i := roll (1, 6); (* x surprise y? *)
  IF i <= 2 THEN
    action := 1;
  i := roll (1, 6); (* y surprise x? *)
  IF i <= 2 THEN
    IF action = 1 THEN
      action := 0
       (* cannot both be surprised *)
    ELSE
      action := 2;
  IF action = 1 THEN
    WRITELN
    ("You surprised the monster!")
  ELSE
  IF action = 2 THEN
    WRITELN
    ("The monster surprised you!");
  REPEAT
    IF action = 0 THEN
      BEGIN
      REPEAT
        WRITELN ;
        WRITELN ;
        WRITE ("<F>ight, ",
               "<I>nfo, ",
               "<P>arry, ",
               "<R>un ... ");
        READ (reply);
        IF reply < " " THEN
           reply := " ";
        WRITELN (CHR (reply));
        IF reply = "I" THEN
           reply := "i";
        IF reply = "i" THEN
          BEGIN
          displaychar (1);
          WRITELN ("Press a key ...");
          REPEAT UNTIL GETKEY ;
          displaychar (y)
          END ;
        CASE reply OF
        "f", "F", "p", "P",
        "r", "R" : ok_reply := true
        ELSE
          ok_reply := false
        END  (* of case *)
      UNTIL ok_reply;
      CASE reply OF
        "f", "F" : action := 0;
        "p", "P" : action := 3;
        "r", "R" : action := 4
      END (* of case *)
    END ;
    melee (1, y, action);
    IF action <> 4 THEN
      action := 0
  UNTIL (action = 4) (* ran away *)
     OR (hp [1] = 0) (* player dead *)
     OR (hp [y] = 0) (* monster dead *)
  END
END ;


(* START of MAIN PROGRAM *)

BEGIN
REPEAT
  (* prime random numbers *)
  VOICE (3, 1, 10000,
         3, 14, 1);
  REPEAT
    WRITE (CHR (home));
    WRITE ("What level for human? ");
    READ (i)
  UNTIL (i >= 1) AND (i <= maxlevel);
  generatechar (1,1,i,6,9,1,6,0);
  WRITE (CHR (home),
         "Generating monsters ...");
  FOR i := 2 TO 10 DO
    generatemonster (i);
  IF show_monsters THEN
    FOR i := 2 TO 10 DO
      BEGIN
      displaychar (i);
      WRITE ("Press a key ... ");
      REPEAT UNTIL GETKEY
      END ;
  WRITE (CHR (home));
  REPEAT
    encounter (roll (1, 9) + 1)
  UNTIL hp [1] <= 0;
  WRITELN ;
  WRITE ("<Q>uit or <N>ext game: ");
  READ (reply)
UNTIL (reply = "q") OR (reply = "Q");
WRITE (CHR (home))
END .
```