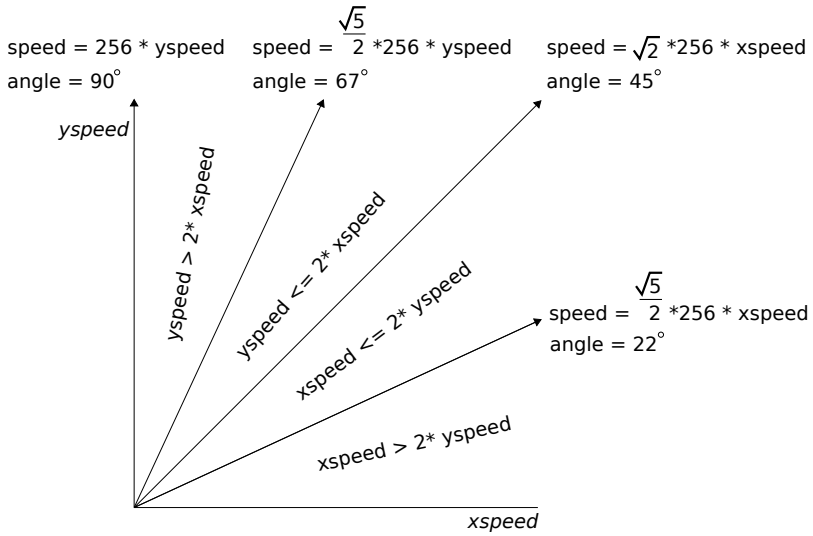
0.1 Tricks

This section describes random tricks used to speed up rendering.

0.1.1 Bouncing Flower

When Keen throws a flower it bounces of the walls. For flat walls and floors the bounce can be easily calculated by reversing either the x-speed (for vertical walls) or y-speed (for horizontal walls). It becomes more complicated for slopes. Making an accurate calculation of the bounce on a slope requires expensive `cos` and `sin` methods.

Instead, the game used a simple algorithm that approximates the angle to either 22°, 45° or 90°. Based on the ratio between the x- and y-speed it calculates the resulting speed and corresponding angle. Notice that for higher precision the speed is multiplied with 256.



For each of the eight type of slopes (Figure **??**) and incoming angle combination, the corresponding bounce is defined using a simple lookup table.

```
// bounceangle[walltype][angle]

unsigned  bounceangle[8][8] =
{
{0,0,0,0,0,0,0,0},
{7,6,5,4,3,2,1,0},
{5,4,3,2,1,0,15,14},
{5,4,3,2,1,0,15,14},
{3,2,1,0,15,14,13,12},
{9,8,7,6,5,4,3,2},
{9,8,7,6,5,4,3,2},
{11,10,9,8,7,6,5,4}
};
```

The value in the table refers to the corresponding bounce angle calculation. As example, waltype 3 with incoming angle of 22°, results in bounce calculation case 5.

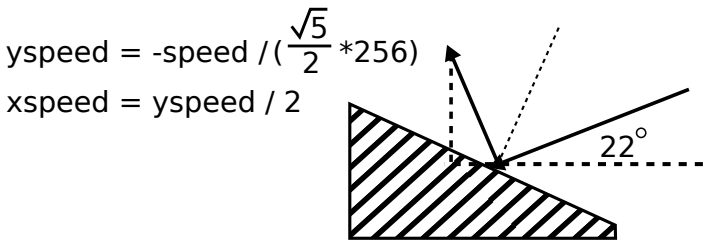


Figure 1: Walltype 3 with incoming angle of 22° (angle=0).

```
absx = abs(ob->xspeed);
absy = ob->yspeed;
if (absx>absy)
{
  if (absx>absy*2)         // 22 degrees
  {
    angle = 0;
    speed = absx*286;      // x*sqrt(5)/2 *256
  }
  else
  [...]              // Check for 45, 67 and 90 degrees
}

if (ob->xspeed > 0)
  angle = 7-angle;         // mirror angle

speed >>= 1;               // speed / 2 after bounce
newangle = bounceangle[ob->hitnorth][angle];
switch (newangle)
{
[...]

case 5:
  ob->yspeed = -(speed / 286);
  ob->xspeed = ob->yspeed / 2;
  break;

[...]
}
```

Notice that in several cases the bounce angle is not following the laws of physics.

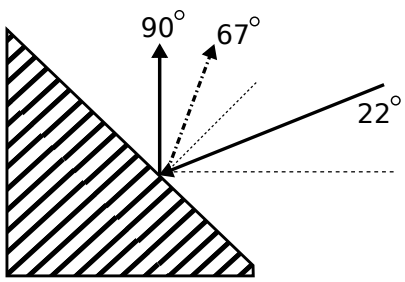


Figure 2: Incoming angle of 22° on a 45° slope results in 90° bounce.

0.2 Pseudo Random Generator

Random numbers are necessary for many things during runtime, such as calculating whether an enemy is able to hit the player based on its accuracy. This is achieved with a precalculated pseudo-random series of 256 elements.

```
rndindex  dw  ?

rndtable

db    0,   8, 109, 220, 222, 241, 149, 107,  75, 248, 254, 140,  16,  66
db   74,  21, 211,  47,  80, 242, 154,  27, 205, 128, 161,  89,  77,  36
db   95, 110,  85,  48, 212, 140, 211, 249,  22,  79, 200,  50,  28, 188
db   52, 140, 202, 120,  68, 145,  62,  70, 184, 190,  91, 197, 152, 224
db  149, 104,  25, 178, 252, 182, 202, 182, 141, 197,   4,  81, 181, 242
db  145,  42,  39, 227, 156, 198, 225, 193, 219,  93, 122, 175, 249,   0
db  175, 143,  70, 239,  46, 246, 163,  53, 163, 109, 168, 135,   2, 235
db   25,  92,  20, 145, 138,  77,  69, 166,  78, 176, 173, 212, 166, 113
db   94, 161,  41,  50, 239,  49, 111, 164,  70,  60,   2,  37, 171,  75
db  136, 156,  11,  56,  42, 146, 138, 229,  73, 146,  77,  61,  98, 196
db  135, 106,  63, 197, 195,  86,  96, 203, 113, 101, 170, 247, 181, 113
db   80, 250, 108,   7, 255, 237, 129, 226,  79, 107, 112, 166, 103, 241
db   24, 223, 239, 120, 198,  58,  60,  82, 128,   3, 184,  66, 143, 224
db  145, 224,  81, 206, 163,  45,  63,  90, 168, 114,  59,  33, 159,  95
db   28, 139, 123,  98, 125, 196,  15,  70, 194, 253,  54,  14, 109, 226
db   71,  17, 161,  93, 186,  87, 244, 138,  20,  52, 123, 251,  26,  36
db   17,  46,  52, 231, 232,  76,  31, 221,  84,  37, 216, 165, 212, 106
db  197, 242,  98,  43,  39, 175, 254, 145, 190,  84, 118, 222, 187, 136
db  120, 163, 236, 249
```

Each entry in the array has a dual function. It is an integer within the range [0-255][1] and it is also the index of the next entry to fetch for next call. This works overall as a 255 entry chained list. The pseudo-random series is initialized using the current time modulo 256 when the engine starts up.

[1] Or at least it was intended to!

```
;================================================
;
;
; void US_InitRndT (boolean randomize)
; Init table based RND generator
; if randomize is false, the counter is set to 0
;
;
;================================================


PROC  US_InitRndT randomize:word

  uses  si,di
  public  US_InitRndT

  mov ax,[randomize]
  or  ax,ax
  jne @@timeit    ;if randomize is true, really random

  mov dx,0      ;set to a definite value
  jmp @@setit

@@timeit:
  mov ah,2ch
  int 21h     ;GetSystemTime
  and dx,0ffh

@@setit:
  mov [rndindex],dx
  ret

ENDP
```

The random number generator saves the last index in `rndindex`. Upon request for a new number, it simply looks up the new value and updates `rndindex`.

```
;================================================
;
; int US_RndT (void)
; Return a random # between 0-255
; Exit : AX = value
;
;================================================
PROC  US_RndT
  public  US_RndT

  mov bx,[rndindex]
  inc bx
  and bx,0ffh
  mov [rndindex],bx
  mov al,[rndtable+BX]
  xor ah,ah
  ret

ENDP
```