Let's Make A Retro Game

Episode 13 – More Enemies

In this episode we are going to enhance the generation of the enemy objects. Currently we are only making the same type of asteroid appear, whereas we want a variety of enemies to appear on screen with different movement logic and behaviour.

We won't cover all the eventual enemy types in this episode, but we will put together the framework so we can add more in the next episode.

Enemy Management

So, before we get started we need to plan out our memory structures that will handle different types of enemies. The best way I find to work through this is to work out what different things you will need for each enemy type and then make a table of the common things you need to store.

There are two tables, one to store our base information about an enemy, which will be stored in ROM and the 2nd the information we need to store in Ram during a game session.

ROM Table

Name	Bytes	Description
Starting Shape	1	Starting Sprite Pattern
Ending Shape	1	Ending Sprite Pattern
DX	1	Any starting change in X (we will randomly choose the direction)
DY	1	Any starting change in Y
Score	1	The value added to the players score when destroyed
Width	1	The width of our enemy for collision detection
Spare	2	Round our bytes out to 8, makes are maths a little easier and you never know what you might want to add later.

In our code for our game we would have a table as follows:

```
; Enemy source data table
; Start Shape, End Shape, DX, DY, Score
DB 12, 16, 0, 1, 10, 16 ; Large Meteor
DB 20, 24, 0, 2, 20, 10 ; Small Meteor
DB 28, 40, 1, 2, 50, 10 ; Smart Bomb
DB 44,48, 0, 0, 0, 16   ; Explosion
```

The enemy type is used as the index into the table in ROM, and knowing that each table entry is 8 bytes long we can calculate the address quite simply as follows

```
; A contains the enemy type index
LD HL, ENEMY_TYPES
; multiply index by 8
SLA A
SLA A
SLA A
; add to our original location
LD C,A
LD B,0
ADD HL, BC
```

RAM Table

Name	Bytes	Description
Enemy Type	1	This indicates the type of our enemy object as follows: 0 – None 1 – Large Meteor 2 – Small Meteor 3 – Smart Bomb 4 – Explosion
DX	1	Our current change in X i.e. the amount to add/subtract from X to move our enemy
DY	1	Our current change in Y i.e. the amount to add/subtract from Y to move our enemy

Note: This could be made more efficient by sharing bytes for a number of indicators i.e. we only have a handful of enemy types so we don't really need 8 bits to store that, but there is always a trade-off.

In this case both speed and readability, the 2nd of which is important in a tutorial like this, so we will stick with using separate bytes for each of our values.

Updated Code

So let's apply this technique to our existing code in our templates.

Setup

First we need to add our data table, place this near the end of the file either before or after the TITLE_SCREEN_PAT section as follows:

```
; Enemy source data table
ENEMY_TYPES:
    ; Start Shape, End Shape, DX, DY, Score, Width, Spare1, Spare2
    DB 12, 16, 0, 1, 10, 16, 0, 0 ; Large Meteor
    DB 20, 24, 0, 2, 20, 12, 0, 0 ; Small Meteor
    DB 28, 40, 1, 2, 50, 12, 0, 0 ; Smart Bomb
    DB 44, 48, 0, 0, 0, 16, 0, 0   ; Explosion
```

Also add a useful function for calculating the address in ROM of a particular enemy type as follows:

```
; Calculate base enemy data location in ROM
; A = enemy type (assumed to be 1 or greater)
; Returns:
; IY = Enemy Type data in ROM
CALC_ENEMY:
    ; save registers we are going to change
    PUSH AF
    PUSH HL
    PUSH DE
    LD HL,ENEMY_TYPES
    DEC A
    ; divide by 8
    SLA A
    SLA A
    SLA A
    LD E,A
    LD D,0
    ADD HL,DE
    ; set our return data
    PUSH HL
    POP IY
    ; restore registers we changed
    POP DE
    POP HL
    POP AF
    RET
```

We need to allocate more Ram for our enemy data table as follows:

```
ENEMYDATA:    DS 60
```

And cover this in our INITRAM function as follows:

; Init Ram for a new game

```
INITRAM:
```

```
    LD A,3
    LD (LIVES),A
    LD HL,0
    LD (SCORE),HL
    LD (SCORE+1),HL
    LD A,1
    LD (LASTSCORE),A
    XOR A
    LD (LASTLIVES),A
    LD (LEVEL),A
    LD (ANIMATE),A
    ; initialise enemy data
    LD HL,ENEMYDATA
    LD (HL),A
    LD DE,ENEMYDATA+1
    LD BC,59
    LDIR
    RET
```

Spawn Enemies

We need to update our SPAWN_ENEMIES section of code to cater for the extra ram positions and storing our DX and DY values as follows:

```
; Spawn/create new enemies
SPAWN_ENEMIES:
    LD A,(LEVEL)
    INC A
    SLA A ; multiply by 4
    SLA A
    LD C,A
    CALL RND
    CP C
    RET NC
    ; see if there is an enemy object available
    LD HL,ENEMYDATA
    LD B,20
SE1:
    XOR A
    CP (HL)
    JR NZ,SE2
    ; enemy available
    PUSH HL
    ; calc our sprite memory position
    LD A,20
    SUB B
    SLA A
    SLA A
    LD C,A
    LD B,0
    LD HL,SPRTBL+12
```

```
    ADD HL,BC
    ; set Y to zero
    LD A,0
    LD (HL),A
    ; set X to a random value
    INC HL
    CALL RND
    LD (HL),A
    ; set pattern
    INC HL
    LD A,8
    LD (HL),A
    ; set colour
    INC HL
    LD A,0dh
    LD (HL),A
    POP HL
    ; hard wire to enemy type 1 for now
    LD A,1
    CALL CALC_ENEMY
    LD (HL),A
    INC HL
    LD A,(IY+2) ; DX
    LD (HL),A
    INC HL
    LD A,(IY+3) ; DY
    LD (HL),A
    RET
SE2:
    ; move to the next data position (now 3 Ram spaces per enemy)
    INC HL
    INC HL
    INC HL
    ; dec b and jump if non-zero
    DJNZ SE1
    RET
```

Let's Make A Retro Game

Episode 13 – More Enemies

Move Enemies

We need to update our MOVE_ENEMIES section of code as follows:

```
; Move any active enemies
MOVE_ENEMIES:
     LD HL,ENEMYDATA
     LD B,20
ME1:
     XOR A
     CP (HL)
     JP Z,ME2
     ; found active enemy
     ; calc our sprite memory position (20-B) * 4
     PUSH BC
     LD A,20
     SUB B
     SLA A
     SLA A
     LD C,A
     LD B,0
     LD IX,SPRTBL+12
     ADD IX,BC
     POP BC
     ; get our enemy data
     LD A,(HL)
     ; get the pointer to our enemy data
     CALL CALC_ENEMY

     LD A,(ANIMATE)
     CP 0
     JR NZ,ME5
     ; animate our sprite pattern (hardwired for the moment)
     LD A,(IX+2) ; get current pattern
     ADD A,4     ; add four
     CP (IY+1)   ; compare against our ending pattern
     JR NZ,ME4   ; if we have reached our ending pattern we need to
move back to the original pattern
     LD A,(IY+0) ; starting sprite shape
ME4:
     LD (IX+2),A ; store the value back
ME5:
     ; get current Y position
     LD E,(IX+0)
     ; change Y at the rate defined in the enemy ram data table
     INC HL
     INC HL
     LD A,(HL)
     DEC HL
     DEC HL
     ADD A,E
```

```
    CP 150
    JR C,ME3
    ; enemy has reached the bottom of the screen
    ; check whether the enemy has hit the player ship
    LD A,(SPRTBL+1)
    SUB A,(IY+5) ; width of enemy from enemy data table
    CP (IX+1)
    JR NC, ME7
    ; x + it's width is larger than the players X
    LD A,(SPRTBL+1)
    ADD A,(IY+5)
    CP (IX+1)
    JR C, ME7
    ; we should be hitting the player
    ; decrease the players life counter
    LD A,(LIVES)
    DEC A
    ; check we don't overflow
    JR NC,ME9
    XOR A
ME9:
    LD (LIVES),A
    ; at the moment we won't do any animation effect
    ; TODO: Animate players death

    ; continue on so that we finish our enemy loop and subroutine
    JR ME8
ME7:
    ; have not hit the player decrease score
    ; decrease score
    LD A,1
    CALL SCORESUB
    ; explosion?

ME8:
    ; clear enemy data
    XOR A
    LD (HL),A
    ; clear sprite
    LD A,209
    LD (IX+0),A
    JR ME2
ME3:
    LD (IX+0),A

    ; enemy object has been moved now do collision detection
    LD A,(SPRTBL+8) ; bullet y position
    CP 209 ; check that it is on screen
    JR Z,ME2
    PUSH HL ; save values so we can use the registers
```

```
    PUSH DE
    PUSH IY
    LD A,(IY+5) ; get our width
    LD L,A
    LD H,L
    LD IY,SPRTBL+8
    LD DE,0208h ; set our bullet size at 2x8
    CALL COLTST
    POP IY
    POP DE
    POP HL
    JR NC,ME2
    ; we have a hit, for the moment just make both objects disappear
    LD A,209
    LD (SPRTBL+8),A
    LD (IX+0),A
    XOR A
    LD (HL),A ; deactive the enemy
    ; increase our score for hitting the asteroid
    ; Note: later we will vary the score by type of enemy
    LD A,(IY+4) ; get our points from the enemy data table
    CALL SCOREADD

    ; later we will:
    ; - explosion sound
    ; - animate enemy

ME2:
    INC HL
    INC HL
    INC HL
    DEC B
    JP NZ,ME1
    ; adjust our animation timing
    LD A,(ANIMATE)
    DEC A
    JP P,ME6
    LD A,2
ME6:
    LD (ANIMATE),A

    RET
```