Assembly Programming for the Nintendo Entertainment System Part 2

Peter J Hopkins

Senior - CIT Major

Web Design Concentration

CIS 386

Contents:

- Backgrounds & NameTables
- Wall Collision & HotSpots
- Item Placement & Zipping
- Safe Zone Mechanics
- ScoreBoard Mechanics
- Stack and NMI Synchronozation
- Future Goals

Backgrounds & NameTables

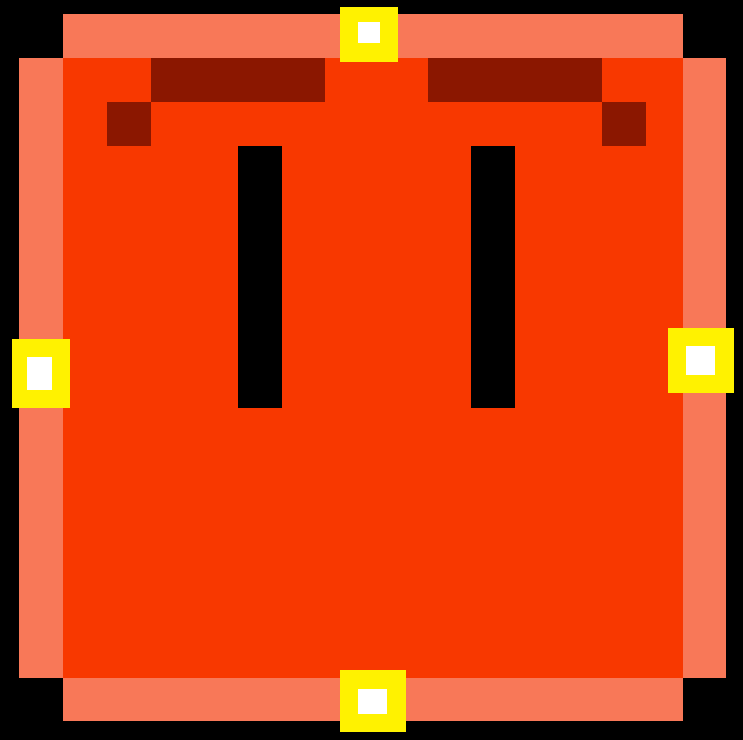
- Backgrounds are 32 x 30 tiles
- Attributes determine color palettes
- Backgrounds are stored in one of two nametables
- Only TWO backgrounds are stored in PPU RAM
- Backgrounds are changed one row or column at a time
- Address $2005 handles scrolling

Contents:

- Backgrounds & NameTables
- Wall Collision & HotSpots
- Item Placement & Zipping
- Safe Zone Mechanics
- ScoreBoard Mechanics
- Stack and NMI Synchronozation
- Future Goals

Wall Collision

- All moving objects have 4 hot spots
- Hot spots cannot overlap wall tiles
- Coordinates are compared to nametables:
  - Divide Horizontal by 8: 10100001 – 00010100
  - Divide Vertical by 8 : 01010010 – 00001010
  - Shift Vertical left 5 : 01000000
  - OR with Horizontal : 01010100 – Low Byte
  - OR Vertical upper 2 bits with Nametable Address: 11100001 – High Byte
  - Use High and Low Bytes to lookup background tile



```
GetBackgroundTile:      ;this loop is run 4 times, once for each tile above, below, left, and right of each s
    LDA topHorizontal, Y    ;get the horizontal coordinate for each edge center point
    LSR A                   ;divide it by 8 so it's only 5 bits long (to determine which background tile it's
    LSR A
    LSR A
    STA pointerLo       ;store it in the low byte pointer
    LDA topVertical, Y  ;divide the vertical coordinate by 8, shortening it to 5 bits
    LSR A
    LSR A
    LSR A
    STA temporary       ;save it temorarily
    ASL A               ;next, shift if left by 5 places. this will shorten it to 3 bits
    ASL A
    ASL A
    ASL A
    ASL A
    ORA pointerLo       ;combine it with the low pointer to get the low byte of the background tile address
    STA pointerLo
    LDA temporary       ;reload the vertical 8 pixel tile coordinate
    LSR A               ;shift it right 3 places, since the first 3 bits are already in the low pointer
    LSR A
    LSR A
    ORA #$E0            ;combine it with the high byte of the background nametable page
    STA pointerHi       ;and store the result in the high pointer. now the background tile type for the addre
    STY temporary       ;temporarily store Y so it can be reset
    LDY #$00            ;Y must be set to 0 for indirect indexing
    LDA [pointerLo], Y      ;get the tile data from the nametable
    LDY temporary       ;reload the loop counter into Y
    STA tileAbove, Y    ;and store the tile data in the appropriate variable for this loop iteration
    INY                 ;increment Y
    CPY #$04            ;check if Y has reached 4, one time for each side of the sprite
```

Contents:

- Backgrounds & NameTables
- Wall Collision & HotSpots
- Item Placement & Zipping
- Safe Zone Mechanics
- ScoreBoard Mechanics
- Stack and NMI Synchronozation
- Future Goals

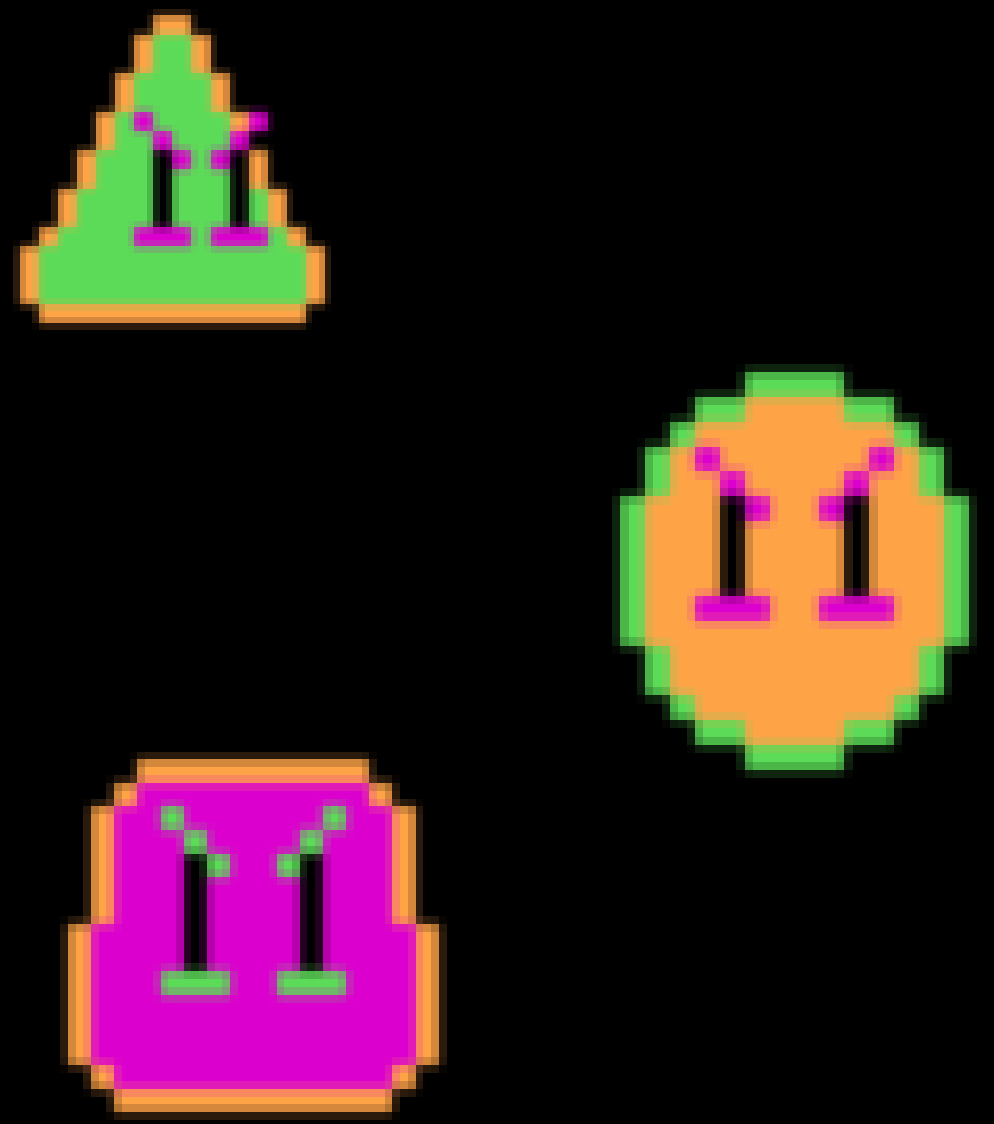
Item Drops

- Items are positioned randomly
- Items positioned in walls ‘pop’ out
- RNG can drop items off the bottom of the screen
  - Items are ‘zipped’ back into the play area

```
1112 DropItem:              ;The following section determines whether to drop an item, what Item to drop, and where to drop
1113     LDA currentItem
1114     CMP #$09           ;a current item of 9 means the item was just collected & a new item should be placed onscreen
1115     BEQ ChooseItem
1116     JMP DropItemDone
1117 ChooseItem:
1118     LDA frameTimer     ;the item that is chosen is based on the current frame
1119     AND #%00000111     ;pick from the 8 items based on the last 3 bits of the current frame
1120     STA currentItem
1121 GetItemCoordinates:    ;use the RNG to get item Coordinates.
1122     LDA random
1123     STA itemVer
1124     LDA cycleTimer
1125     STA itemHor
1126 SetItemController:     ;this chooses a random direction for the item to move in when dropped
1127 ItemControllerVertical:
1128     LDA player1Ver     ;the direction the item is moving in is based on player 1 and player 2 coordinates
1129     AND #%00000001
1130     BEQ ItemControllerDown
1131     ItemControllerUp:
1132         LDA #%00001000
1133         STA itemCont
1134         JMP ItemControllerHorizontal
1135     ItemControllerDown:
1136         LDA #%00000100
1137         STA itemCont
1138 ItemControllerHorizontal:
1139     LDA player1Hor
1140     AND #%00000001
1141     BEQ ItemControllerRight
1142     ItemControllerLeft:
1143         LDA #%00000010
```

Contents:

- Backgrounds & NameTables
- Wall Collision & HotSpots
- Item Placement & Zipping
- Safe Zone Mechanics
- ScoreBoard Mechanics
- Stack and NMI Synchronozation
- Future Goals



Safe Zone Mechanics

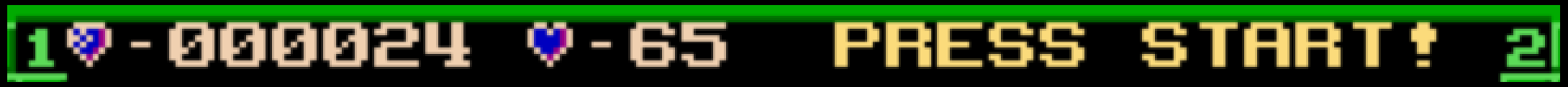
- Players begin in the Safe Zone
  - Enemies and items cannot enter!
- Use XOR to flip enemy controllers
  - Enemies get stuck the safe Zone
- Make the safe zone out of normal wall tiles
  - Enemies and items react with normal wall collision
  - Players move freely within safe zone, but can’t re-enter
  - Re-use of existing game physics (efficient programming)

Contents:

- Backgrounds & NameTables
- Wall Collision & HotSpots
- Item Placement & Zipping
- Safe Zone Mechanics
- ScoreBoard Mechanics
- Stack and NMI Synchronozation
- Future Goals

Scoreboard

- Convert Hexadecimal to Decimal
- Each digit is stored in a byte in the nametable
- If a digit equals $0A, reset to $00 and increment the next digit
- If a digit equals $FF, reset to $09 and decrement the next digit



```
1329      LDX #$1C              ;this section keeps the scoreboard running in base 10
1330  IncrementScoreBoard:
1331      DEX
1332      LDA scoreTile1, X
1333      CMP #$FF              ;check to see if the digit is less than 0
1334      BEQ RollBackScore     ;if so, ensure the score rolls back correctly
1335      CMP #$0A              ;check to see if the digit has reached 10
1336      BNE NextScoreDigit
1337      LDA #$00
1338      STA scoreTile1, X
1339      TXA
1340      TAY
1341      DEY
1342      LDA scoreTile1, Y
1343      CLC
1344      ADC #$01
1345      STA scoreTile1, Y
1346      JMP NextScoreDigit
1347  RollBackScore:
1348      LDA #$09
1349      STA scoreTile1, X
1350      TXA
1351      TAY
1352      DEY
1353      LDA scoreTile1, Y
1354      SEC
1355      SBC #$01
1356      STA scoreTile1, Y
1357  NextScoreDigit:
1358      CPX #$00
1359      BNE IncrementScoreBoard
1360  IncrementScoreBoardDone:
```

Contents:

- Backgrounds & NameTables
- Wall Collision & HotSpots
- Item Placement & Zipping
- Safe Zone Mechanics
- ScoreBoard Mechanics
- Stack and NMI Synchronozation
- Future Goals

The NES Stack

- Efficient memory structure
- Preserve game state during NMI
- At NMI, push all registers into the stack
- After NMI, pop register values back out of stack
- Easy Sync with main program loop

Contents:

- Backgrounds & NameTables
- Wall Collision & HotSpots
- Item Placement & Zipping
- Safe Zone Mechanics
- ScoreBoard Mechanics
- Stack and NMI Synchronozation
- Future Goals

Future Work

- Re-design maze layout
- Implement item behavior
- Sound effects
- 2 Player game
- Title Screen
- Full release online this summer