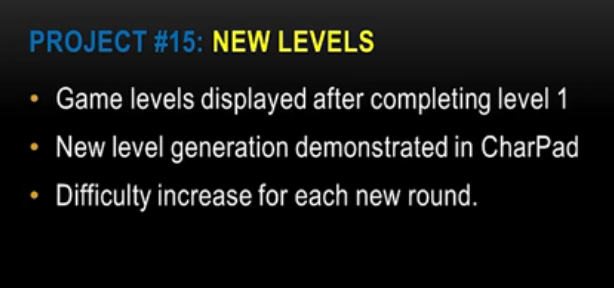
**Commodore 64 Game Project 14/15**

Project 14/15 YouTube Video Setup

For this video series we will be combining Project 14 and 15





**Project 14: Player Death**

- Player dies after losing so many lives  
- Player death display  
- Game over display

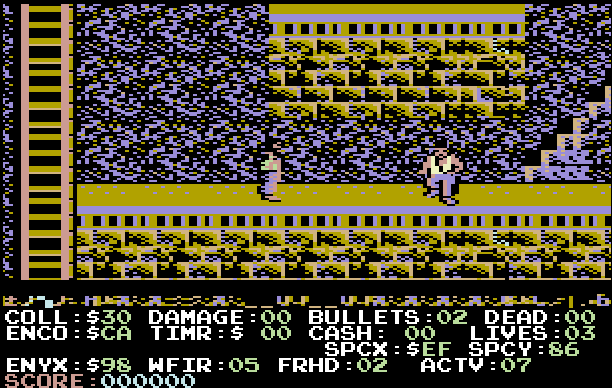
**Project 15**: **New Levels**  
- Player arrives at another level when walking through a specific doorway

**Extra information**

- How to download project(s) from github  
- How to unzip your project  
- How to run it within CBM Prg Studio (to play it)

**Extra Features**

* Show example of latest project work (enemy sprite chases Player up/down ladders)
* Anything else new
* Cash pickups (gold area)
* Vending machine pickups



**Intro**

For this video series which includes Project 14 and 15, we will be learning how to check when a player sprite has landed on the ground after falling for a great distance. This will result in a death and the sprite shape data will change. Then the Player position is reset to try again. After so many deaths, the game over screen is displayed in the status window.

**Bugs**: Plenty exist, but haven’t slowed down the progress since I can still play the game (without any crashes). In time I’ll get to work on perfecting the scrolling. I notice the problem is occurring only when the sprite falls at the far right or ledge edge of the screen. The original engine I’m working with is strange and has really tasked me greatly. I’m grateful for it, but it still is burdensome at time to work with.

**Player Death**

During the game the player will currently die whenever the sprite falls too high off a ledge. Later we will add bullets affecting a death, but let’s keep this simple for now since adding new code only weighs more heavily on sorting on logic. This can especially challenging for new people learning how to create their very first Commodore 64 assembly language games. Therefore when the player sprite death subroutine is executed a variable for PLAYER\_DAMAGE is tracked to see what type of damage was down (falling from ledge, player was shot, etc.).



When the player has run out of lives a variable called **PLAYER\_ISDEAD** is checked for the lives left and then the game over screen is triggered.

**Code for our player’s damage**

Within the file Attacker.asm, I’ve added several new subroutines that manage these tasks.

Inside of Player\_Routines.asm: Apply Gravity, the code below is reviewed after the player sprite has fallen off a cliff. The player sprite is moving down (MovePlayerDown). If the variable (that is increasing) for **PLAYER\_DAMAGE** is less than 68, then our player sprite has not fallen a whole lot, so he has recovered. However, if the variable exceeds 68, the variable **PLAYER\_IS\_DEAD** is set which means that player fell from too great a height and has died.

@falling

ldx #0

jsr MovePlayerDown

ldx #1

jsr MovePlayerDown

clc

lda PLAYER\_DAMAGE

adc #1

sta PLAYER\_DAMAGE

cmp #68

bcc @safeFall

lda #1

sta PLAYER\_ISDEAD

rts

As soon as our player sprite (that has fallen a great distance) reaches a floor again, the subroutine PlayerStateDead is executed in the code below. The damage counter variable **PLAYER\_DAMAGE** is also reset back to zero so he can try again and keep the game going for now.

**Code for death of our sprite**

The player sprite has died so we set a new variable called **PLAYER\_DIED** to indicate this. The variable is checked to see if it has gone done to zero (beq) and if so, the **PlayerStateDead** subroutine is executed. Otherwise, we continue the code for PlayerStateIdle which checks movement when the joystick is not being moved.

PlayerStateIdle

*;===============================================================================*

*; SET IDLE SPRITE*

*;===============================================================================*

lda PLAYER\_DIED

beq @contIdle

jsr PlayerStateDead

@contIdle

lda #1

sta SPRITE\_ANIM\_PLAY *; pause our animation*

lda PLAYER\_SUBSTATE *; First run PLAYER\_SUBSTATE=0*

bne @running *; set in ChangePlayerState*

*; This is executed every time since PLAYER\_SUBSTATE starts at zero.*

ldx #0 *; load sprite number (0) in X*

lda #<ANIM\_PLAYER\_IDLE *; load animation list in ZEROPAGE\_POINTER\_1*

sta ZEROPAGE\_POINTER\_1 *; byte %00000111*

lda #>ANIM\_PLAYER\_IDLE

sta ZEROPAGE\_POINTER\_1 + 1

jsr InitSpriteAnim *; setup the animation for Idle*

lda PLAYER\_SUBSTATE\_RUNNING *; PLAYER\_SUBSTATE\_RUNNING=1*

sta PLAYER\_SUBSTATE *; Now PLAYER\_STATE=1, so we can exit*

rts

The same subroutine (PlayerStateDead) that checks for a player’s death also will reset the variable **PLAYER\_DAMAGE** back to zero as I mentioned earlier in the code below.

A variable for **PLAYER\_LIVES** is checked and set in the PlayerStateDead subroutine to see if the player has died too many times. If so, then the subroutine PlayerGameOver is called. Otherwise the code will continue our game in progress.

@deathCD

*; Check if Player is dead yet*

lda PLAYER\_LIVES

cmp #1

bne @contGame

jsr PlayerGameOver

However, once the player has died the subroutine **ShowDeadSprite** is called which displayed the sprite dead player. Then the code calls @restartMap and then displays the screen the player first started on by resetting the map display.

*; Floor is found. Check if Player is dead*

lda PLAYER\_ISDEAD

beq @exitDeath *; Player is not dead*

lda #1

sta PLAYER\_DIED

@contGame

jsr JoystickReady

lda JOY\_X

bne @restartLife

beq @showDeathAnim

*; Restart a new map*

@restartLife

loadPointer CURRENT\_SCREEN,SCREEN1\_MEM

loadPointer CURRENT\_BUFFER,SCREEN2\_MEM

ldx #65 *; (129,26=default), 70,20*

ldy #20

jsr DrawMap *; Draw the level map (Screen1)*

*; And initialize it*

jsr DisplayConsolePanel

The same subroutine (PlayerStateDead) that checks for a player’s death also will reset the variable **PLAYER\_DAMAGE** back to zero as I mentioned earlier in the code below.

lda #COLOR\_LTRED

sta VIC\_SPRITE\_MULTICOLOR\_2

lda #0

sta PLAYER\_DIED

sta PLAYER\_ISDEAD

sta PLAYER\_DAMAGE

lda PLAYER\_LIVES

sec

sbc #1

sta PLAYER\_LIVES

rts

**Code for Game over display**

Unfortunately our sprite has died too many times, so the subroutine **PlayerGameOver** was called which shows the message for “Game over”. Then the game score is reset to zero and a variable called **GAMESCORE\_ACTIVE** is set which is used to determine when to tally the final totals for the end of a game.



PlayerGameOver

loadpointer ZEROPAGE\_POINTER\_1, GAMEOVER\_PANEL

lda #0 *; PARAM1 contains X screen coord (column)*

sta PARAM1

lda #19 *; PARAM2 contains Y screen coord (row)*

sta PARAM2

lda #COLOR\_WHITE *; PARAM3 contains the color to use*

sta PARAM3

jsr DisplayText *; Then we display the stats panel*

lda #0

sta 53250

sta 53251

sta 53252

sta 53253

sta 53254

jsr ShowDeadSprite

@waitJoyMove

jsr JoystickReady

*; Wait for fire button to restart Game*

lda #%00010000 *; Mask for bit 0*

bit JOY\_2 *; check zero = jumping (button pressed)*

bne @waitJoyMove *; continue other check*

*; lda JOY\_X*

*;; bne @resetGame*

*; beq @waitJoyMove*

@resetGame

lda #5

sta PLAYER\_LIVES

lda #0

sta gamescore

sta gamescore + 1

lda #1

sta GAMESCORE\_ACTIVE

rts