ZX Spectrum Manic Miner Movement, Collision Detection and Routine

Things to note

First of all, hats off to Matthew Smith, he is one smart dude.

All movement should governed by a replica of the orignal movement routine, if you try to replicate Willy's movement using modern game engine physics, you will probably not replicate Willy's movement exactly. Additionally, trying to code the movement by just examining the behaviour of Willy in the game is very difficult, as you'll circle about fixing one issue only to cause another. The best approach is to replicate the original movement routine, described below, as exactly as possible.

A Bit of Background

The original game screen (where the environment and Willy are drawn, not the lower part showing the score, room name, etc) is split into 32 (0-31) horizontal and 16 vertical (0-15) 'cells'. Each cell is 8x8 pixels. Therefore the game area is 256 (32 x 8) pixels by 128 (16 x 8) pixels. With cell 0,0 being the top left cell and cell 31,15 being the bottom right cell.

Each item, such as walls, floors, etc, must be placed wholly with in one of these cells. If you place floors, walls, etc, overlapping cells, the movement routine will not work. Collision with floors, walls, etc is determined by looking at what is in the cells surrounding Willy, this is described later.

Given the tiny size of modern pixels when compared to ZX Spectrum pixels, when coding you will probably want to multiple the original pixels positions by some factor (4 seems reasonable). I'll mention this in the pseudo code below as a reminder.

The original ZX Spectrum code represented Willy's x position as a cell position between 1-29 (Willy can't reach the extreme left or extreme right of the game area), rather than an x pixel position and Willy's y position as a value between 0-255. Willy's actual y pixel position, as used by the drawing routine is transformed to 0-127 to give the real y pixel position (all bits are shifted 1 bit right with bit 0 being dropped), so 151 becomes 75, 150 becomes 75, 149 becomes 74, 148 becomes 74, etc). When required in the original code the Willy's y position is transformed into a y cell position.

The ZX Spectrum didn't have any hardware sprites, so all of the sprite drawing had to be done in code. To make life a bit easier, and given the arrangement of the ZX spectrum video memory map, each animation frame of Willy is 32 bytes of data, enough to hold data for 16 x 16 pixels. There are 8 animation frames in total, 4 for when Willy is facing right and 4 for when Willy is facing left. The red square indicates Willy's cell position which is always the top left cell of the animation frame.

	Frame 0	Frame 1	Frame 2	Frame 3
Right				
Left				

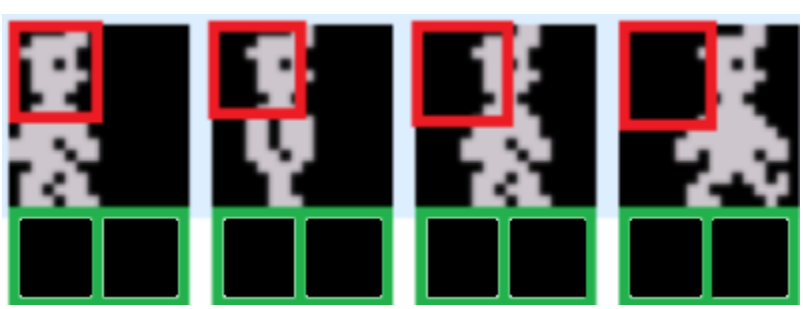
Moving left and right (when jumping and walking)

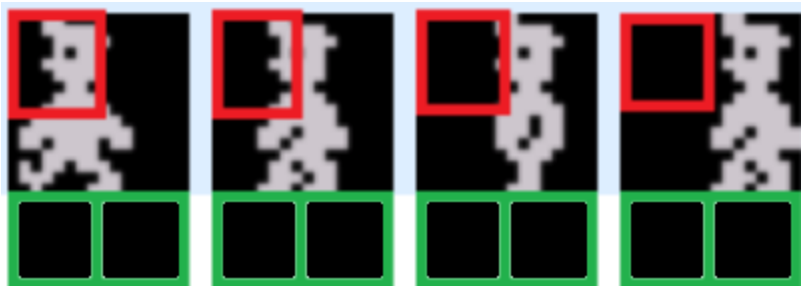
Rather than drawing the current animation frame for Willy a few pixels to the right for each step, when Willy is moving right, the 16x16 pixel frames are drawn in turn without changing Willy's x cell position. This gives the illusion of Willy moving right. A similar approach is taken when Willy is moving left. When the last frame of the animation is reached, collision is checked to see if Willy can continue to move. If not blocked by a wall, then Willy's x cell position is incremented by 1 in the case of walking right and decremented by 1 in the case of walking left and the frame cycle starts over. In this way, collision detection for wall tiles is only performed as Willy crosses a cell boundary, hence the importance of placing items, such as wall tiles, floor tiles, etc, wholly within in cell boundaries.

Landing on Platforms (Bottom Collision)

After 12 jumping movements (JAC is 13), Willy is a moving down and a full 2 cells above where he started from and therefore could land if a floor is directly below him. After 15 jumping movements (JAC is 16), Willy is moving down and 1 full cell above of where he started from and therefore could land if a floor is directly below him. When Willy is falling, the presence of a floor below Willy is checked each time Willy is contained completely within 2 y cells.

The cells checked are cells (x,y+2) and (x+1,y+2), regardless if Willy is moving left or right, or standing still. Willy's current cell (x,y) is show in red, with cells (x,y+2) and (x+1,y+2) shown in green.





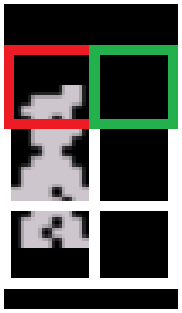
If any of the green tiles contain a floor, then Willy has landed, or, in the case of already standing, not falling. If any of the green tiles contain crumbling tiles, they should crumble. If any of the green tiles contain a conveyor tile, then the direction of the conveyor is used by the routine below to determine Willy's movement.

<u>Willy Hitting His Head (Top Collision)</u>

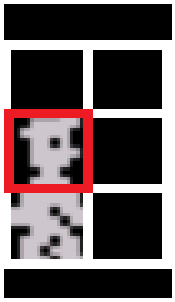
When Willy is jumping, the original code checks to see if the current jump means that Willy is hitting his head on wall. If so, Willy is immediately moved to the cell below the wall. The check, and if necessary, the change to Willy's y position is done in one movement cycle, not two.

The check to see if Willy is hitting his head is done by examining the tile contents of cells (x,y) and (x+1,y).

Please note, that Willy's current cell is the upper, left most cell that Willy occupies, even if only the very top of his head occupies the bottom of this cell. As shown below, Willy occupies six cells (remember Willy is always considered as occupying 2 x cells regardless of animation frame), and therefore Willy's current frame (x,y) is shown in red, cell (x+1,y) is shown in green. Both cell (x,y) and (x+1,y) are checked for the presence of a wall tile, if a wall tile is present then Willy has hit his head.



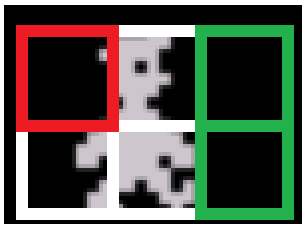
If a collision is detected, Willy is moved down to the cell below his current cell. Additionally, any left/right movement is stopped, meaning that Willy will fall straight down after hitting his head.



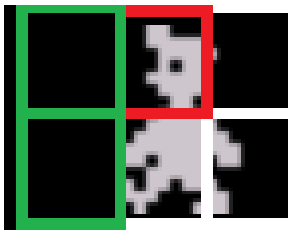
<u>Wall collisions when moving Left or Right (Side Collision)</u>
If Willy is moving left or right, whether jumping or walking, at the end of each animation cycle (0 for going left and 3 for going right), cells to the right (or left) are checked for the presence of wall tiles. If a wall tile exists in the checked cells, Willy is blocked and can't move. Here's the check logic

- If Willy's animation frame only covers two y cells, check the following.
    - If Willy is moving right, check cells (x+2,y) and (x+2,y+1).
    - If Willy is moving left, check cells (x-1,y) and (x-1,y+1).
- If Willy's animation frame covers three y cells, check the following.
    - If Willy is moving right, check cells (x+2,y), (x+2,y+1), (x+2,y+2)
    - If Willy is moving left, check cells (x-1,y), (x-1,y+1), (x-1,y+2)
- If any of the checked cells contain a wall tile, Willy is blocked and therefore his movement is stopped.

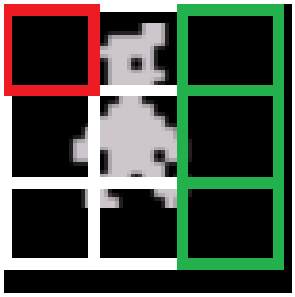
Examples
Willy is moving right, the animation frame = 3, and only exists in two y cells. The red cell is Willy's current cell location (x,y) and the green cells are the checked cells (x+2,y) and (x+2,y+1).



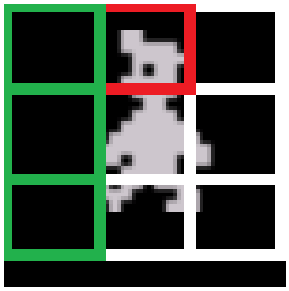
Willy is moving left, the animation frame is 0, and only exists in two y cells. The red cell is Willy's current cell location (x,y) and the green cells are the checked cells (x-1,y) and (x-1,y+1)



Willy is moving right, the animation frame = 3, and exists in three y cells. The red cell is Willy's current cell location (x,y) and the green cells are the checked cells (x+2,y), (x+2,y+1) and (x+2,y+2).



Willy is moving left, the animation frame is 0, and exists in three y cells. The red cell is Willy's current cell location (x,y) and the green cells are the checked cells (x-1,y), (x-1,y+1) and (x-1,y+2).



<u>Collision with Other Objects (Enemies, Obstacles and Items)</u>
The original code checked for collisions with other objects as Willy and the other objects are drawn. This is not part of the movement routine. It's a clever routine, that has pixel perfect collision. Therefore, do what you can with your game engine's collision routines to try to replicate the other collisions.

<u>Original Code 'Variables'</u>

- AnimFrame
    - The current animation frame to be drawn 0 - 3.
    - Should be set at 0 initially if Willy is facing right at start of level and 3 if Willy is facing left at the start of the level.

- AI (Airborne indicator)
    - Used by the movement routine to determine Willy's status and therefore determine Willy's update.
    - 0 = Willy is standing
    - 1 = Willy is jumping
    - 2 = Willy has walked off a floor and begins to fall.
    - 3 - 11 = Willy is falling (AI is increased for each falling step).
    - 6 = Willy has finished his jump cycle or is already falling.
    - 12 or greater = Willy is falling and fallen to far to land safely.

    - 255 = Willy has collided with a nasty (may not be required depending on your collision implementation)
    - Should be set as 0 initially.

- JAC (Jumping animation counter)
    - Can be any value between 0 and 18
    - Transformed into of the following values [-8,-8, -6, -6, -4, -4, -2, -2, 0, 0, 2, 2, 4, 4, 6, 6, 8, 8] the transformed value of 18 (10) is never actually used.
    - This transformed value is used to move Willy in the y direction during the jumping (and falling process) and gives a parabolic like jump.
    - As the ZX Spectrum version stores Willy's y position as 2 times the actual ZX Spectrum y pixel position, the actual **pixel** jump movement is [-4, -4, -3, -3, -2, -2, -1, 1, 0, 0,1, 1, 2, 2, 3, 3, 4, 4]
    - You may wish to implement the transformed values as an zero indexed array, with the JAC being the array reference. Or, as is the case with the original implementation, transform the JAC into the y change value by using a binary operation to reset the least significant bit of the JAC to zero and then subtracting 8. For instance, if the JAC is 11, then resetting the least significant bit to 0 gives 10 and subtracting 8 gives 2.
    - Should be set as 0 initially.

- MoveDir
    - Single variable used in a bitwise fashion to store both which way Willy is facing and also if Willy is moving
    - If bit 0 is 0 then Willy is facing right, if bit 0 is 1 then Willy is facing left.
    - If bit 1 is 0 then Willy is not moving, if bit 1 is 1 then Willy is moving.
    - Should be initially set as value 0 for those caverns where Willy starts facing right (Willy facing right and not moving), or set as 1 for those caverns where Willy starts facing left.
    - Bits are referenced as 0-7 (with 7 being the most significant bit) in the pseudo code below.

- MoveByte
    - A variable used a bitwise fashion to determine whether left and/or right is being pressed and the direction of the conveyor (if being stood on).
    - Bits are referenced as 0-7 (with 7 being the most significant bit) in the pseudo code below.
    - Set in pseudo code

- MoveArrayLookup
    - Set by the MoveByte and current MoveDir.
    - Used to lookup the MoveArray to get the next MoveDir.
    - Set in pseudo code

- MoveArray
    - An array used in conjunction with the MoveArrayLookup to determine the new moveDir.
    - Should be set as [0, 1, 0, 1, 1, 3, 1, 3, 2, 0, 2, 0, 0, 1, 2, 3] (one of these is chosen based on key press and current MoveDir to become the new MoveDir).

<u>Pseudo Movement Code</u>

The following pseudo code is based on the original ZX Spectrum assembly code and therefore doesn't naturally follow a functional or stateful approach. In my implementation, I created each step as separate function, calling functions where jumps are used. I also used Game Maker's exit instruction to immediately exit functions to replicate the jump instruction (for which a return doesn't return to the calling function)

I've used the phrase 'jump to' where code execution should move to a new section but **not return** to the step that jumped. The word 'return' means go back to 'calling' function, not, return to the step that jumped. (this 'calling' function will probably be the function that calls your movement routine). Please note, that none of the code below deals with collisions with enemies, items, or obstacles. This can be done outside of this code using your collision detection of choice after each iteration of the code below, or inserted at a suitable position in the pseudo code.

1. **Check if Jumping (35515)**
   If AI != 1, jump to 3.
   Jump to 2

2. **Jumping (35522)**
   Update Willy's y position with JAC *(remember to scale here if required!)*
   If top collision with wall tile jump to 7 (hit head)
   Increment JAC
   If JAC = 18, jump to 5 (finished jump cycle)
   If JAC = 13 or 16 , jump to 3. (one or two full cells above start of jump)
   Jump to 12 (move Willy left/right)

3. **Standing, Falling, or Jumping and JAC = 13 or 16 (35600)**
   If Willy occupies 6 cells, jump to 4. *[1]
   If bottom collision with crumbling floor tile, make tile crumble 1 frame (remove tile if it has crumbled away completely) and continue.
   if bottom collision with **any** type of floor, jump to 8 (Willy has landed).
   Jump to 4

   **1 this can be implemented by checking if Willy's y position is on a cell boundary, for instance in the case of a pixel scaling of 4, if y mod 32 !=0*

4. **Standing (on nothing), Falling or Jumping (35655)**
   If AI = 1, jump to 8 (Willy is jumping)
   Set movedir bit 1 to 0 (stop right/left movement as Willy is falling from standing, or falling).
   If AI = 0, jump to 6 (Willy is standing)
   Increment AI (Willy is falling)
   add 8 to y position (adds 4 to the actual y pixel position) *(remember to scale here if required!)*
   Return

5. **End of Jump (35734)**
   Set AI = 6 (falling down from jumping)
   Return

6. **Start to Fall (35740)**
   Set AI = 2
   Return

7. **Hit head (35746)**
   Set Willy's y position to just under wall that was hit[*2]
   Set AI to 2 (falling down)
   Set movedir bit 1 to 0 (Willy is now not moving left or right and falling straight down).
   Return

   **2 Remembering that y = 0 is top, the new y pixel position can be calculated using y = y + ((y div 32) * 32), if scaling by 4*

8. **Landed (35805)**
   If AI >= 12, jump to 18 (Kill Willy)
   Set AI = 0 (Willy has landed safely)
   Set MoveByte = 3 (bits are now 00000011)
   If not bottom collision with conveyor tile, jump 9
   If conveyor is going left, set MoveByte bit 1 as 0. (bits are now 00000001)
   If conveyor is going right, set MoveByte bit 0 as 0. (bits are now 00000010)
   jump to 9

9. **Start to Work out new MoveDir (35835)**
   If right key is being pressed, set MoveByte bit 0 to 0.
   If left key is being pressed, set MoveByte bit 1 to 0.
   Set MoveArrayLookup = 0
   If MoveByte bit 1 is 0, then set MoveArrayLookup += 4 (must be +4 not = 4)
   If MoveByte bit 0 is 0, then set MoveArrayLookup +=8 (must be +8 not = 8)
   MoveArrayLookup += MoveDir

Jump to 10

10. **Set new MoveDir (35892)**
    If (MoveArrayLookup = 6 or 4), then set animation to Willy going left, as Willy has turned
    If (MoveArrayLookup = 9 or 11), then set animation to Willy going right, as Willy has turned
    MoveDir = MoveDirArray[MoveArrayLookup]
    If jump key is pressed, jump to 11
    Jump to 12

11. **Start Jump (35963)**
    Set JAC = 0
    Set AI = 1
    Jump to 12

12. **Move Willy Left/Right (35971)**
    If bit 1 of MoveDir = 0, return (not moving left or right)
    If bit 0 of MoveDir = 0, jump 15 (Willy is moving right)
    (Willy is moving left)
    Is AnimFrame = 0, jump 13 (at cell boundary)
    Decrement AnimFrame
    Return

13. **Move Willy Left across cell boundary (35996)**
    If cell (x-1 ,y+1) contains a wall, return
    If Willy occupies only occupies 2 cells, jump to 14. *3
    If cell (x-1 cell, y+2) contains a wall, return (can't move left as wall blocks path)
    Jump to 14

    **3 this can be implemented by checking if Willy's y position is on a cell boundary, for instance in the case of a pixel scaling of 4, if y mod 32 = 0*

14. **Continue Move Willy Left across cell boundary (36025)**
    If cell (x-1, y) contains a wall, return (can't move left as wall blocks path)
    Update Willy's x cell position by -1 (one cell to the left) *(remember to scale here if required!).*
    Set AnimFrame = 3
    Return

15. **Move Right (36042)**
    If AnimFrame = 3, jump to 16 (at cell boundary)
    Increment AnimFrame
    Return

16. **Move Willy Right across cell boundary (36054)**
    If cell (x+2, y+1) contains a wall, return (can't move right as wall blocks path)
    If Willy occupies only 2 cells, jump to 17. [*4]
    If cell (x+2, y+2) contains a wall, return (can't move right as wall blocks path)
    Jump to 17

    **4 this can be implemented by checking if Willy's y position is on a cell boundary, for instance in the case of a pixel scaling of 4, if y mod 32 = 0*

17. **Continue Move Willy Right across cell boundary (36084)**
    If cell (x+2,y) contains a wall, return (can't move right as wall blocks path).
    AnimFrame = 0
    Return

18. **Kill Willy (36102)**
    Up to you. (forget him, he's gone!)