**Curriculum for the Tile Game Workshop:**

1. Create a Scratch account and promote exploration (perhaps point to a good curator) after our meeting.
2. Block programming - discuss similarities between Lego FFL robotics and Scratch.
3. Sprites and event based programming - every sprite can has it’s own “program” or script, and sprites can talk to each other through events that are “broadcast” by one sprite and received by another.
4. Make the (default) cat sprite move forward and spin around.
5. Add a new (dog) sprite and copy the code from the cat sprite (using the backpack) into it the dog sprite so that it independently moves in the same fashion as the cat sprite.
6. Add “On Click” to both sprites so that you can independently click on the sprites and watch them move around.
7. Add a broadcast to the cat sprite that is received by the dog sprite so that clicking on the cat sprite makes both sprites move (cat then dog, cat “talks” to the dog).
8. Swap the positions of the two sprites so that clicking on the cat sprite moves the cat sprite to the dog sprite’s position and visa versa.
9. Talk about “variables” describing them as boxes you can store things within. A list (i.e. array) is simply a box containing numbered drawers, you can store things within the numbered drawers and withdraw by selecting the desired drawer(s).

**NOW WE UNDERSTAND THE BASIC ELEMENTS FOR MAKING OUR TILE GAME!**

**THE TILE GAME**

1. What is the tile game (ideally show a physical one)? Here’s a YouTube example if you’re stuck conveying the visual without a physical example: <https://youtu.be/eOGrzaurDcs>
2. What are the key functional elements that we would need to convert this physical game into a “computer game”:

* Let’s start with definitions:
  + Our background represents a grid. In our case we will work with 3 rows and 3 columns of tiles.
  + Each tile has a “face” that DOESN’T change and each tile separately occupies a position within the grid that DOES change as that tile moves.
  + One of the grid spaces is a blank or an “empty space”, meaning we are really have a total of 8 tiles (3 rows X 3 columns = 9 minus one empty space).
  + What should we put on our tile faces? Similar to a jigsaw puzzle, we can start with a single picture that we’ll split into 9 (3 rows by 3 columns) even pieces. Eight of these split picture pieces will serve as the faces of our tiles; we will discard one of our picture pieces for the empty space.
* Physical “rules” of the game (discuss the physical example as you review these):
  + Only one tile at a time can be moved and then only moved into the “empty space”. The space that the “moved” tile occupied becomes the new empty space.
  + Expanding upon the above, it is only possible to move a tile that is immediately adjacent to the empty space. You can’t move a tile that is surrounded by other tiles.
  + Therefore, for any given grid position, we can identify the other positions a tile in that grid position COULD move, assuming that tile was adjacent to an empty space. Look at the image below with numbered grid positions:



A tile that is in position #1 could only be moved into adjacent positions 2 or 4, assuming that the empty space was at either of these two positions. But what about a tile located at position #5? A tile in position #5 could be moved into positions 2, 4, 6, or 8, again assuming the empty space was in any one of those adjacent positions.

* Translating our physical rules into “computer game”
  + Since only one tile at a time can be moved, we can simulate moving a tile by clicking on it. There really is no point in using drag if you can only drag in one direction anyway.
  + Each tile face can therefore be a sprite. The empty space can also be a sprite.
  + Clicking on a tile adjacent to the empty space moves that tile into the empty space. Clicking on a tile that is not next to the empty space (example: a tile in position 1 of the image above) does nothing.
  + Once a tile moves to the empty space, the grid position that tile formerly occupied becomes the new empty space.  **In other words, we are simply SWAPPING THE POSITIONS of the empty space sprite and the tile sprite we clicked on (**remember when we swapped the cat and the dog sprites?).
  + Given the above, we need:
    - If we ever want to put the tiles back into their original grid positions, we should keep track of the grid position of each tile face (each tile sprite) as it moves around within the grid.
    - Similarly, we should always know the grid position of the empty space (empty space sprite).
    - We should have a list of all of the valid adjacent positions for each of our grid positions. For example, position 1’s valid adjacent positions are positions 2 and 4 as we noted previously. If we click on the tile in position 1 and we know that the empty space is in position 2 or 4 we can move the tile in position 1 by swapping its position with the empty space. Alternatively, if we click on a tile in position 1 and the empty space IS NOT in positions 2 or 4, then we know to do nothing.
* In Scratch terms, what we need:
  + A sprite for each of our 8 tile faces and a sprite for the empty space. Scratch allows us to IMPORT pictures to use as sprites.
  + A Data LIST (i.e. Make a List) we will name “Adjacent”. The Adjacent list will contain 9 positions or “list elements”, each element corresponding to a position on our grid. Within position 1 of the adjacent list, we’ll enter the numbers 2 and 4, representative of the two valid adjacent positions for the position 1 of the grid. Within position 5 of the adjacent list, we’ll enter the numbers 2, 4, 6 and 8 as we discussed above. What should the other positions of the adjacent list contain?
  + Data variables (two) (i.e. Make a Variable) that will track the selected tile’s X and Y positions on the Scratch background and separately two variables that will track the empty space’s X and Y positions on the Scratch background. We will need these in order to perform the swap of a selected tile sprite and the empty space sprite.
  + Another Data LIST (i.e. Make a List) we will name “Position”. We will use this list to track the position of each of our tile faces. The first element of the Position list will always contain the grid position of the first tile sprite. Initially that position will be “1”, but as the first sprite’s tile moves we will update the first element of the Position list to contain the first sprite’s new position within the grid.
* Summary of the Scratch logic:
  + For each sprite, when clicked on, copy the sprite’s X and Y position to the two sprite variables created for this purpose.
  + From the Position list, get the current grid position of the sprite (i.e. a number 1 - 9) and copy that into a variable.
  + From the Position list, get the current grid position of the empty space (I used position 9 to hold the empty space’s position) and copy that into a variable.
  + Get the contents of the Adjacent list for the current position of that tile (i.e. the current grid position variable above)
  + Test each of the numbers (they are single digits 1-9) from the contents of the Adjacent list you obtained in the previous step.
  + If any of these numbers match the current position of the empty space, then move the sprite to the empty space’s Scratch background X/Y position and send the empty space sprite a SWAP broadcast message.
  + When the empty space sprite receives the SWAP broadcast message, it moves itself to the sprite’s previous Scratch X/Y position, completing the “swap”.
  + Update the Position list so that you it contains the new positions of the selected tile sprite and the empty space sprite.