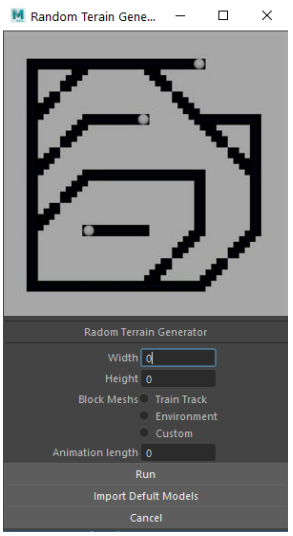
Maya Python Programming Document

For this project I created a program that randomly generates environments based on the blocks the user inputs. This will allow the user to do a small amount of work to establish the style of the scene but then my program creates a seamless level using the blocks as bases.

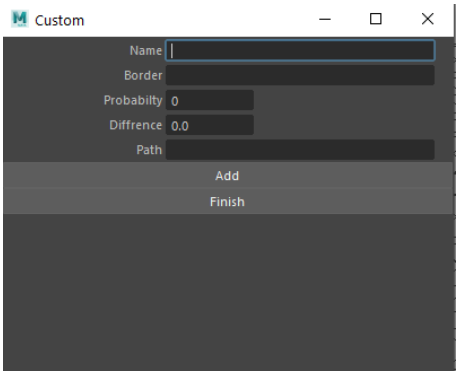


User manual

1. Open maya
2. Open the script .txt file or open the python file in maya
3. Copy the script into maya python accessible by the button bottom right
4. On line 394 and 366 correct the paths for your own computer
5. Run the code by pressing the two triangles top middle of your screen
6. Then this UI should appear

The UI

1. Enter the dimension for the terrain you want generated via width and height. These are limited between 1 and 20.
2. Pick a type of Block Mesh:
    1. Train Track creates an underground train layout. That will also have trains traveling around for as long as you specify in Animation length (Make sure that the Animation length is not less the 0)
    2. Environment creates a level design inspired environment with very very simple blocks.
    3. Custom allows you as the user to add your own blocks to the program. If selected a separate window will pop up for you to enter the details for each of the blocks to generate a terrain. Every time you click Add it will store the information inputted and you can enter new information.



Name = The name of the block in the outliner
Border = Enter a value for each edge and corner of your mesh with a space in between (must contain 8 values)
Probability = The Probability that this Block is going to be chosen (Must be between 1 and 100)

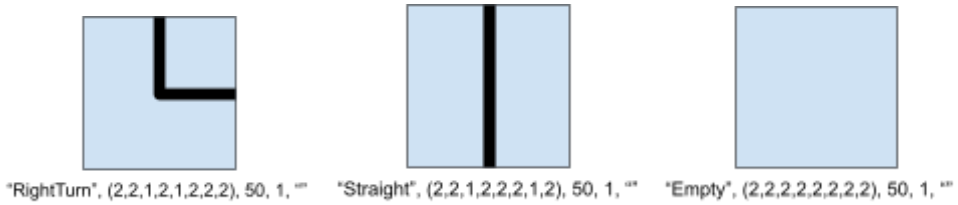
Difference = If lower than 1 then the probability of the Block being picked as you get closer to the center is less. If greater than the closer to the center of the Board the more likely it is to be picked (Must be between 0 and 2)
Paths = The name of the path on the Block that must start with a P and then the border the path starts at and the border the path ends at.

3. If you have selected Train Track or Environment from above then make sure to Import Default Models so that the mesh block used in those default mods are in the scene
4. Run and watch as the programme pieces the blocks together!
5. Occasionally the algorithm will not work because there aren't any blocks that can fit in the space needed. So just run it again and it should work fine.

Algorithm Explanation

Blocks:

Blocks are placed at different Tiles on the Board. The information for these blocks are contained within a class that holds the name, borders, rotation and picked. In this example 2 represents empty space and 1 represents a track on that border.

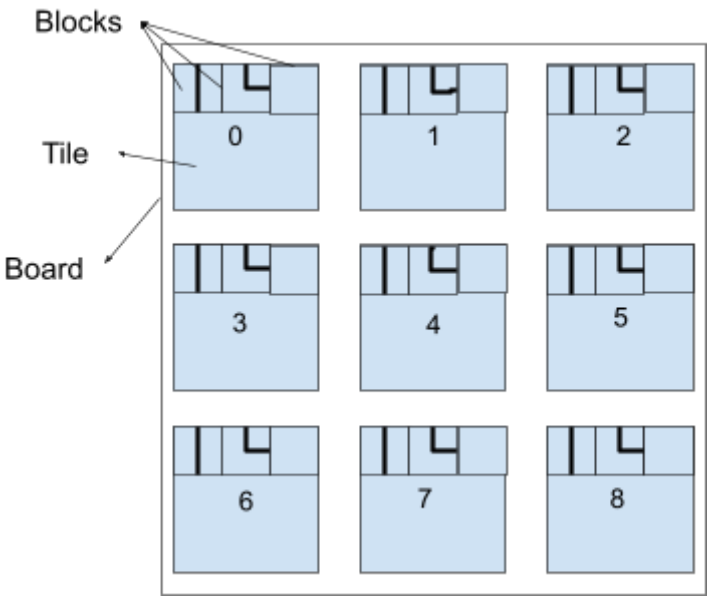


Tiles:

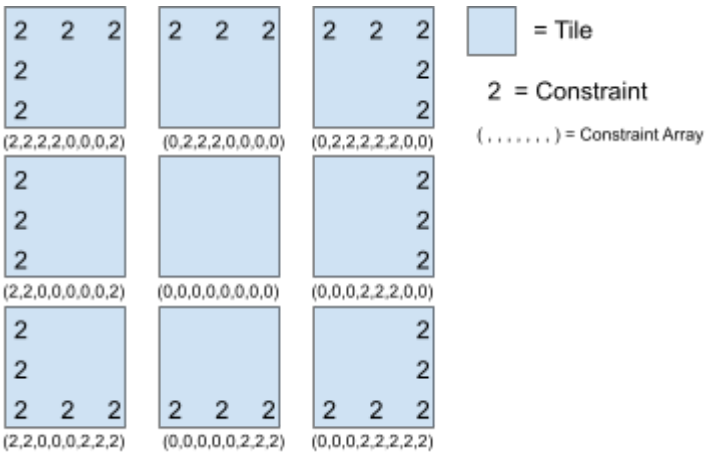
Tiles make up the array elements of the Board. They store which Blocks are possible at that location on the Board based on the Constraints for that tile.

Board:

The way the Board is structured in the program using the above classes is that there is an array that represents all of the tiles on the board. Each array element then contains a Tile class which allows information for that tile like the border constraints, all of the possible blocks etc to be stored. Then within the list of possible blocks stored on that tile each array element contains a Block class. This class contains the information for each Block which contains the blocks name, borders, rotation

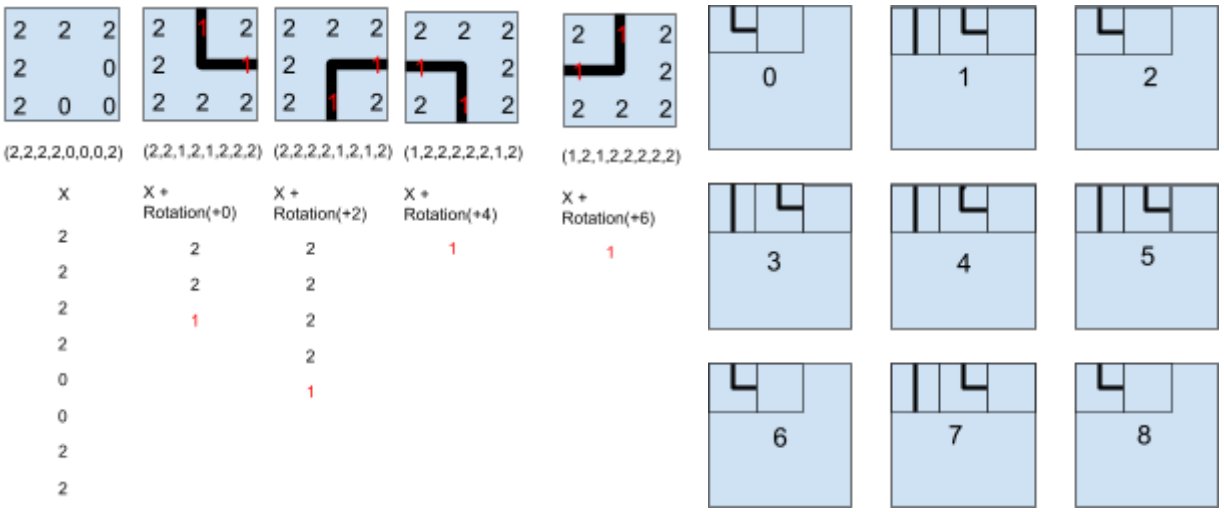


Constraints (Border_Constraints()):



Next step in the algorithm is to add constraints to the tiles on the border of the Board so that tracks don't run off the side or a solid floor is kept at the bottom.

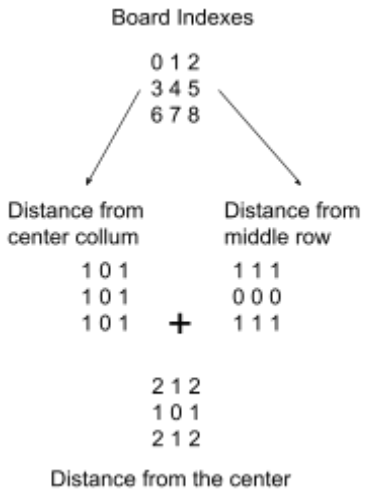
Checking tiles (check()):



After new constraints have been added some blocks may no longer be possible on that tile. Therefore a check is run comparing the borders of the block with the constraints on that tile. This comparison is repeated for each possible rotation of that tile. The rotation value is used as an offset for which index is compared first in the border array.

Weighted random Tile Pick (Random_Tile_Pick() and Random()):

Once the board has been checked and all tiles only have possible blocks the algorithm places a random block on a random tile with a random rotation. However the tile location is random at first and so is the rotation but the Block that is placed is picked randomly but each one with its own weight. This weight is defined by probability and Difference.



Difference takes the distance from the center of the array and multiplies it by the Difference variable for that Block

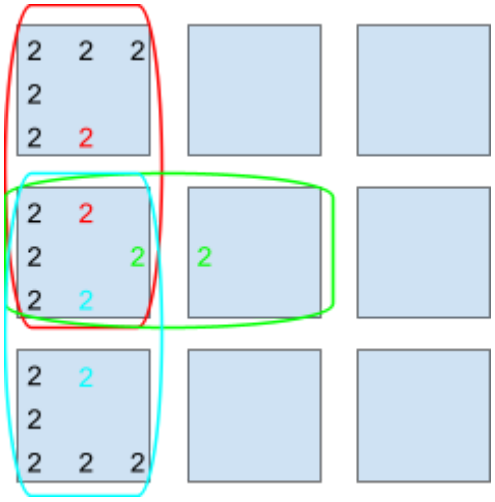
if Difference > 1 increases probability the closer to the center the tile is

if Difference < 1 decreases probability the closer to the center the tile is

The base probability of a Block being picked is stored in its class and used to be increased or decreased by multiplication by difference.

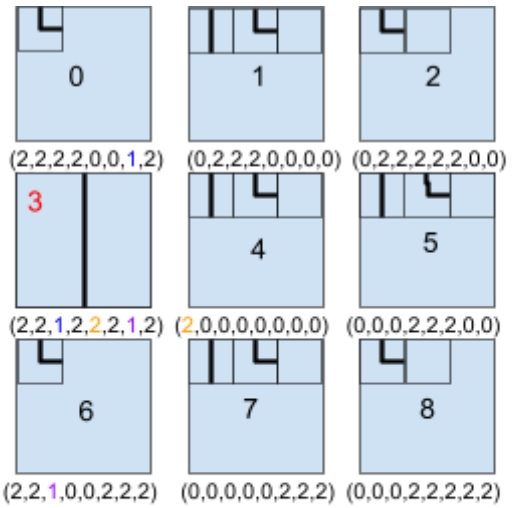
Finally once probability of the tile is established the tile is added the same number of times as the probability to an array.append(Tile * (Probability * (Difference*Distance))). This is repeated for all of the possible Blocks and then one is picked.

Checking borders (Tile_Checker()):



Once the Block is placed the program updates tiles that share a side. These tiles are assigned a new constraint based on the border value of the border that they share with the newly placed tile. Similarly to what happened after constraints where added to the edges the tiles that have had new constraints added are then checked to remove any possible blocks that may no longer be possible.

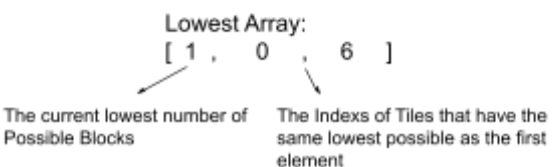
Output:



Lowest (Lowest_Tile()):

Lastly the Board is scanned to keep record of all the tiles that have the lowest number of possible blocks.

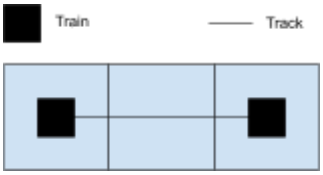
Then on next iterations through the array the program picks a random Tile from the array Lowest. This reduced the possibility of the program failing and finding an impossible



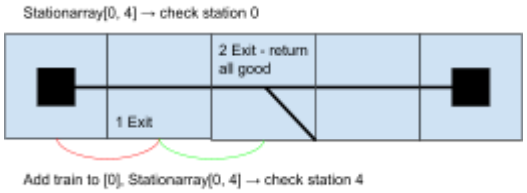
combination of borders. The program then loops back to **Weighted random Tile Pick.**

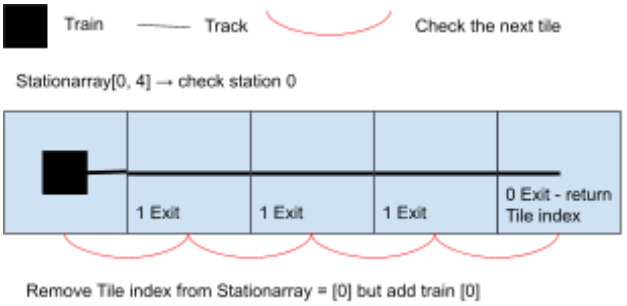
Next Step Train Animation (TrainAnimation() and Iteration()):

If the train track option was picked then once the board is complete the program starts with placing trains at station tiles around the board. However not all stations can have trains otherwise there will be guaranteed collision for example:



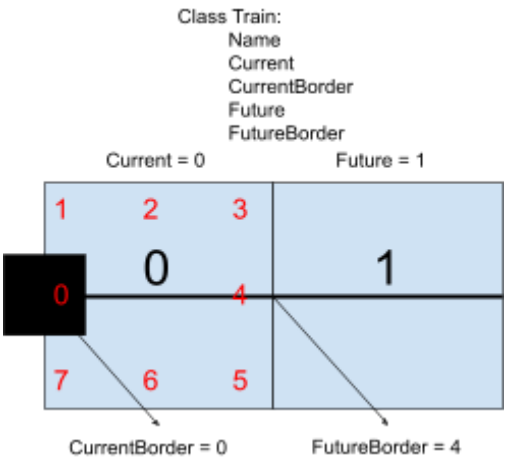
These trains are guaranteed to collide because there are no other routes for the train to follow. To solve this the program checks to see if there is a tile with more than 1 exit on the track from the station





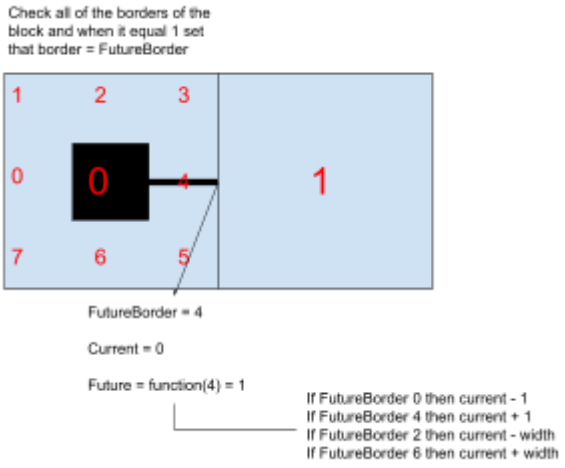
Train (FillTrainList()):

For each train added onto the board its information is stored in the train class which is assigned to an element with in the array TrainList eg:



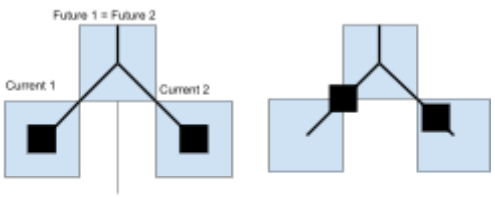
First Move (EnitialMoveOff()):

First Move is when the trains move away from the stations that they were assigned. First the train needs a border to exit out of which can then be used to calculate the future tile index.



Speed (Train_Speed()):

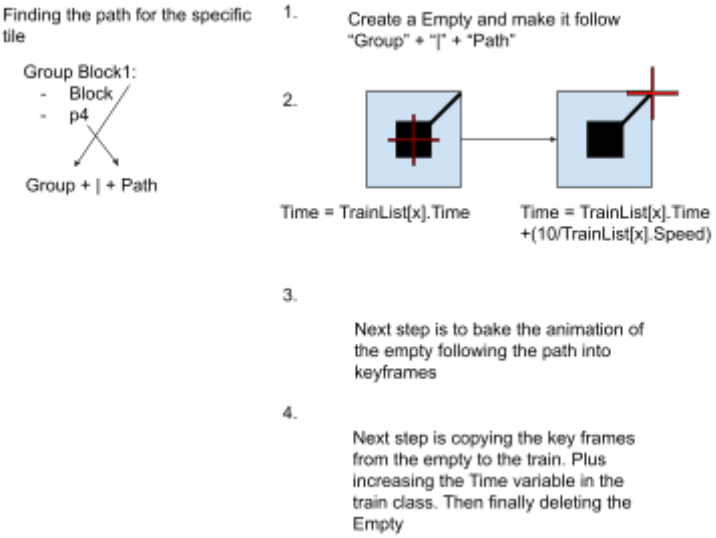
Once the future tile is known and assigned to the train class the speed needs to be determined before the train is able to be animated. Checking speed is important because it allows the trains to keep momentum whilst miss each other.



If 2 train have the same future tile then if they continue at normal speed they will arrive at same time. Therefore if a trains newly future tile is the same as another then half the speed so that they arrive at different time

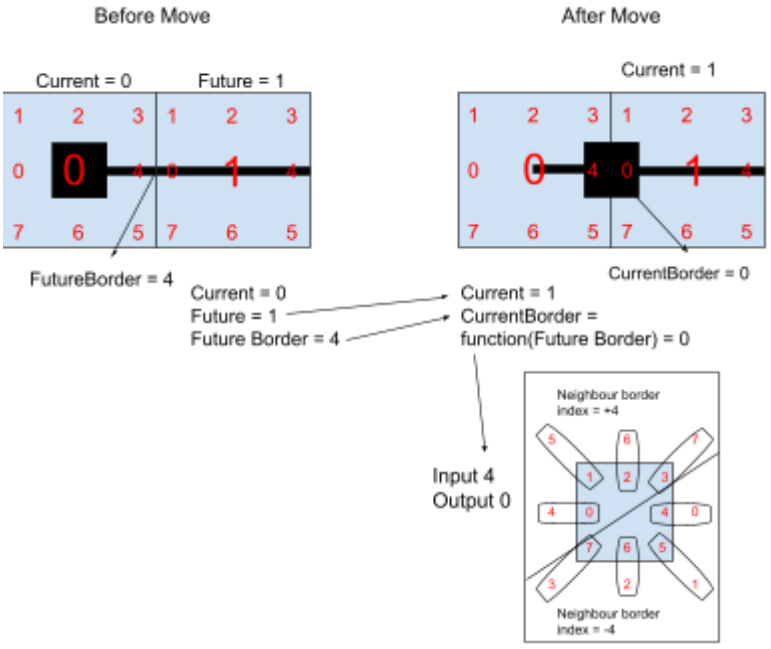
Animation (Train_Animation()):

Finally once the train has got a speed, FutureBorder and Future information then it can be animated along the path on the Tile following this method --->



Second Loop (UpdateTrain()):

This time the trains are no longer at their stations and the process is a little bit different. The main difference is the beginning because all of the trains have there locations updated eg:

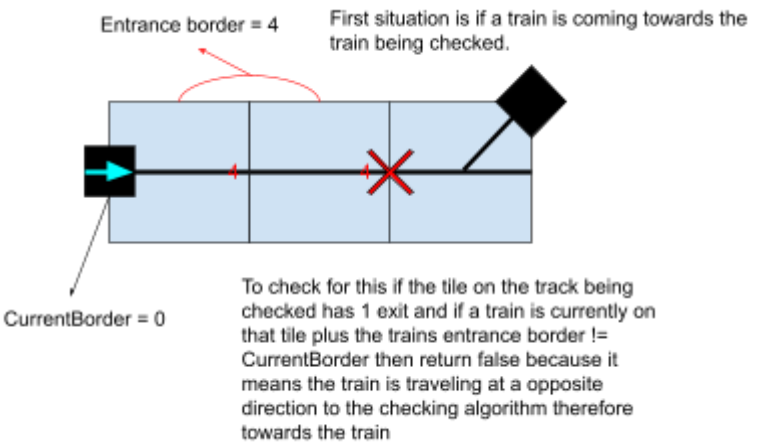


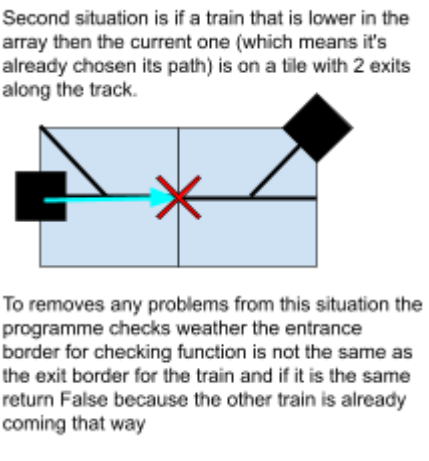
Exits:

Next the program checks how many exits the current tile of the train has. Depending on the number of exits different algorithms are executed. Whilst checking for exits it skips the border that is equal to CurrentBorder because that's where the train entered from.

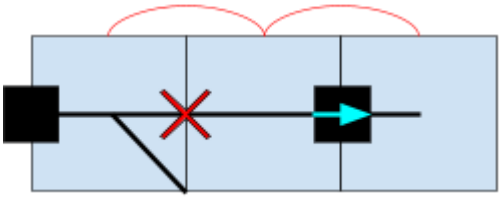
2 exits (PossibleBordersCheck () and Iteration()):

If the current tile has more than 1 exit that means the train has a choice of exit. However before the train can choose the program checks the 2 exit borders. When the program checks these borders it checks the track followed on after these exits and scans them for trains or a tile with multiple exits. Exactly the same to the station checking just this time it checks for trains as well.
When checking a tile on the track there are 4 different situations.





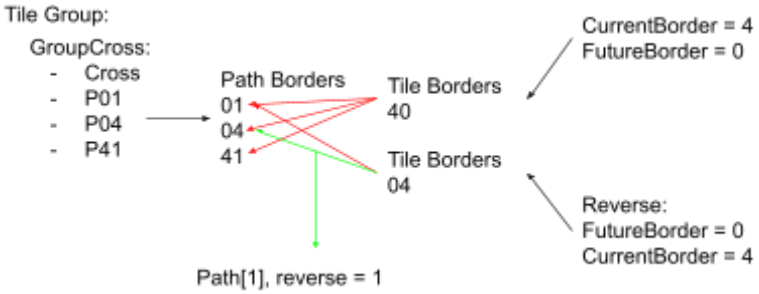
Third situation is that there is a train going away from the train at the cross junction however it ends with a station meaning it has to come back.



To combat this when checking the tiles on the track the algorithm keeps a record that a train was on one of the tiles. So that if the checking stops on a station it will returns False because it's not a viable border

Once the checks have finished a random border from the borders remaining is set as the trains FutureBorder.

However if there are no possible borders for the train to exit on then the animation process is repeated from the beginning however this time the train that got stuck should be removed.



Path Checking (MoveTrain()):

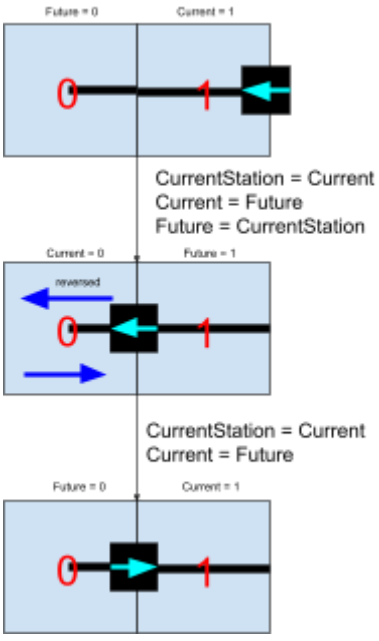
Before the train can be animated the path that the train is going to follow is needed. Eg a train with CurrentBorder 4 and FutureBorder 0

If there is 1 exit (Train_Speed() and MoveTrain()):

Then the only possible border is set to the train's future border and the future tile is calculated via the future border. Then the currentBorder and futureBorders are compared to the path to see if the path needs reversing.

After 1 Exit and 2 Exits Tiles the animation function is called as shown at the end of the first loop. Then the entire process is started looped again from **Second Loop:**

If there are 0 Exit (Train_Speed() and Train_Animation()):



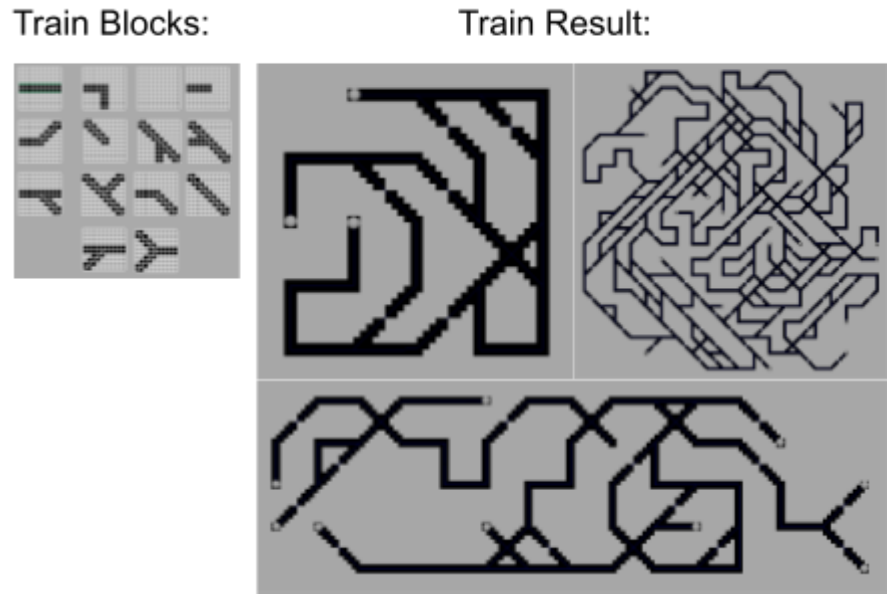
Future is set to the previous current because the train is just going back on itself. The speed is then set to 2 then passed into speed() to calculate if it needs to be slowed. This is because at stations the train follows the path in and then straight back. Therefore instead of it taking 10 frames for 1 path it needs to take 5 frames so that the train keeps up with the other trains. Direction of the train is shown by the blue arrows. To achieve this the programme calls the animation function twice, once in reverse.

.

Overall I am very pleased with my project it has the potential to do a lot more with a few adaptations ie. 3D terrain generation, infinite terrain generation, mazes and generation with other shaped tiles etc, and all though my program wasn't able to do any of those yet it stands as a solid foundation to build those ideas on. However, given some more time, I would like to implement an easier system to input custom mesh blocks. As well as stopping diagonal train tracks from crossing.

Results:

Train Block:



Environment: