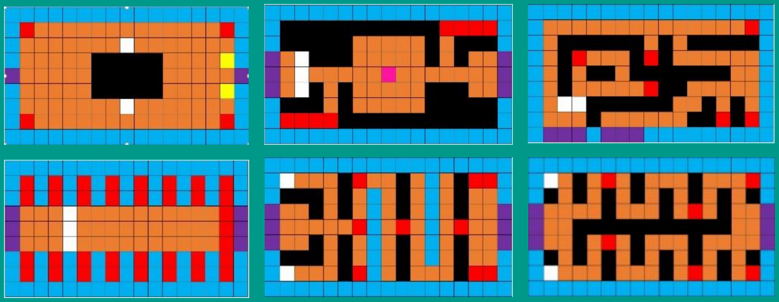
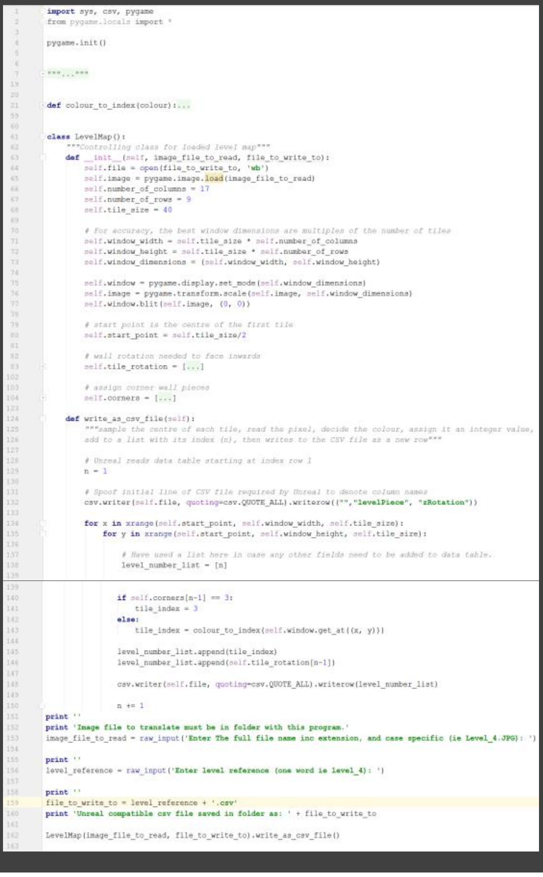
From Design to Unreal

When designing Nebula Knights' modular levels, many maps were exchanged between the team denoting different rooms colour coded to show specific tile properties. Measuring 9 x 17 tiles per room required placing over 150 tiles to see a single room in engine, and with over 50 room designs; building them, if only to visualise the design, became a daunting task. So I wrote a program to do it for us.

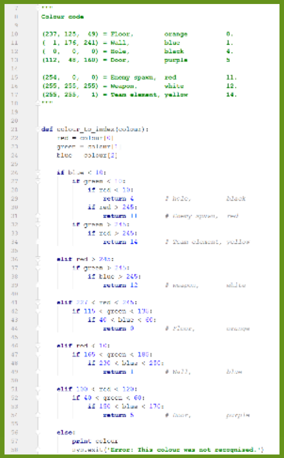


Python Map Reader



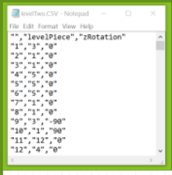
```
import sys, csv, pygame
from pygame.locals import *

pygame.init()


"""..."""


def colour_to_index(colour):...


class LevelMap():
    """Controlling class for loaded level map"""
    def __init__(self, image_file_to_read, file_to_write_to):
        self.file = open(file_to_write_to, 'wb')
        self.image = pygame.image.load(image_file_to_read)
        self.number_of_columns = 17
        self.number_of_rows = 9
        self.tile_size = 40

        # For accuracy, the best window dimensions are multiples of the number of tiles
        self.window_width = self.tile_size * self.number_of_columns
        self.window_height = self.tile_size * self.number_of_rows
        self.window_dimensions = (self.window_width, self.window_height)

        self.window = pygame.display.set_mode(self.window_dimensions)
        self.image = pygame.transform.scale(self.image, self.window_dimensions)
        self.window.blit(self.image, (0, 0))

        # start point is the centre of the first tile
        self.start_point = self.tile_size/2

        # wall rotation needed to face inwards
        self.tile_rotation = [...]

        # assign corner wall pieces
        self.corners = [...]

    def write_as_csv_file(self):
        """sample the centre of each tile, read the pixel, decide the colour, assign it an integer value,
        add to a list with its index (n), then writes to the CSV file as a new row"""

        # Unreal reads data table starting at index row 1
        n = 1

        # Spoof initial line of CSV file required by Unreal to denote column names
        csv.writer(self.file, quoting=csv.QUOTE_ALL).writerow(["","levelPiece", "zRotation"])

        for x in xrange(self.start_point, self.window_width, self.tile_size):
            for y in xrange(self.start_point, self.window_height, self.tile_size):

                # Have used a list here in case any other fields need to be added to data table.
                level_number_list = [n]

                if self.corners[n-1] == 3:
                    tile_index = 3
                else:
                    tile_index = colour_to_index(self.window.get_at((x, y)))

                level_number_list.append(tile_index)
                level_number_list.append(self.tile_rotation[n-1])

                csv.writer(self.file, quoting=csv.QUOTE_ALL).writerow(level_number_list)

                n += 1
print ''
print 'Image file to translate must be in folder with this program.'
image_file_to_read = raw_input('Enter The full file name inc extension, and case specific (ie Level_4.JPG): ')

print ''
level_reference = raw_input('Enter level reference (one word ie level_4): ')

print ''
file_to_write_to = level_reference + '.csv'
print 'Unreal compatible csv file saved in folder as: ' + file_to_write_to

LevelMap(image_file_to_read, file_to_write_to).write_as_csv_file()
```

The python program (top right) uses the PyGame library to create a singleton of a room map image, and the csv library to return a .csv file of quoted integer values each corresponding to a tile colour defined as a modular level piece within a key. Spoofing an initial line denoting the column headers results in a file that Unreal Engine can import as a data table to be interpreted and used in blueprints. Unreal Engine can only import data in the form of quoted .csv files shown in the format below, so the python program is essentially an adapter for these incompatible data types (maps) and interpreter (Unreal Engine), so high coupling can be excused.



```
"""
Colour code

(237, 125,  49) = Floor,        orange       0.
(  1, 176, 241) = Wall,         blue         1.
(  0,   0,   0) = Hole,         black        4.
(112,  48, 160) = Door,         purple       5

(254,   0,   0) = Enemy spawn,  red          11.
(255, 255, 255) = Weapon,       white        12.
(255, 255,   1) = Team element, yellow       14.
"""


def colour_to_index(colour):
    red = colour[0]
    green = colour[1]
    blue = colour[2]

    if blue < 10:
        if green < 10:
            if red < 10:
                return 4          # hole,          black
            if red > 245:
                return 11         # Enemy spawn,   red
        if green > 245:
            if red > 245:
                return 14         # Team element,  yellow

    elif red > 245:
        if green > 245:
            if blue > 245:
                return 12         # weapon,        white

    elif 227 < red < 245:
        if 115 < green < 135:
            if 40 < blue < 60:
                return 0          # Floor,         orange

    elif red < 10:
        if 165 < green < 185:
            if 230 < blue < 250:
                return 1          # Wall,          blue

    elif 100 < red < 120:
        if 40 < green < 60:
            if 150 < blue < 170:
                return 5          # Door,          purple

    else:
        print colour
        sys.exit('Error: This colour was not recognised.')
```

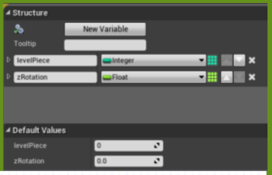
Colour tolerances checking script



levelTwo.CSV - Notepad

```
"","levelPiece","zRotation"
"1","3","0"
"2","1","0"
"3","1","0"
"4","5","0"
"5","5","0"
"6","5","0"
"7","1","0"
"8","1","0"
"9","3","-90"
"10","1","90"
"11","12","0"
"12","4","0"
```

Quoted .csv file



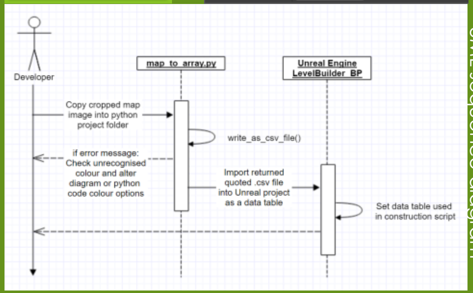
Structure

levelPiece	Integer
zRotation	Float

Default Values

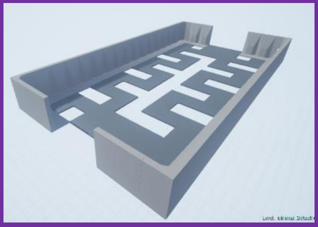
levelPiece	0
zRotation	0.0

Unreal struct for importing



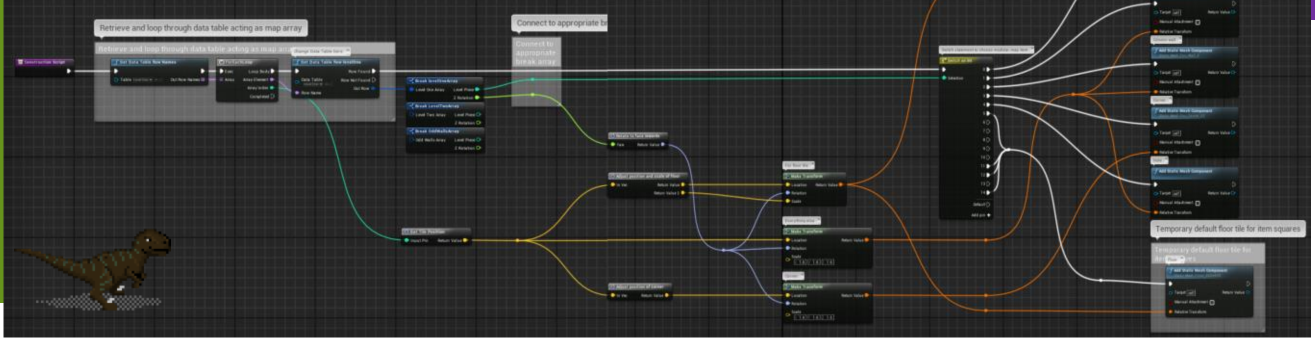
Developer
map_to_array.py
Unreal Engine LevelBuilder_BP
Copy cropped map image into python project folder
write_as_csv_file()
if error message: Check unrecognised colour and alter diagram or python code colour options
Import returned quoted .csv file into Unreal project as a data table
Set data table used in construction script

UML sequence diagram



The result is a serviceable mock up of the room (top). Whilst it lacks the personal touch of a hand-crafted designers level (below), it can very quickly help to visualise any of the designer's level maps in engine. Even were they crudely drawn on paper, this program could identify the colours of each tile, and parse the data into a quoted .csv file, which the blueprint could build a room from.

Finally, the blueprint below acts as a factory, looping through whichever data table's rows it is supplied; using the index to calculate the position of the tile, the modular level piece, and any transform needed. In this case, a rotation around the z-axis so that all walls will face inwards.



Retrieve and loop through data table acting as map array
Connect to appropriate break array
Temporary default floor tile for item squares

Richard Steele

Nebula Knights – Basilisk Studios -