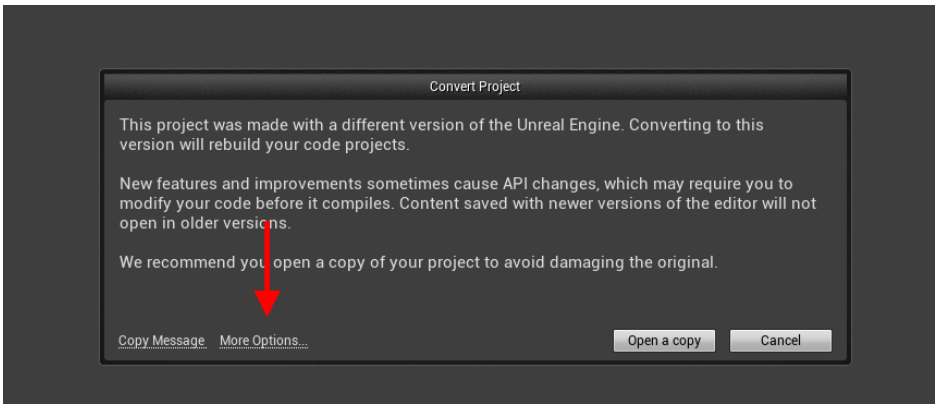
Social Force Model Notes

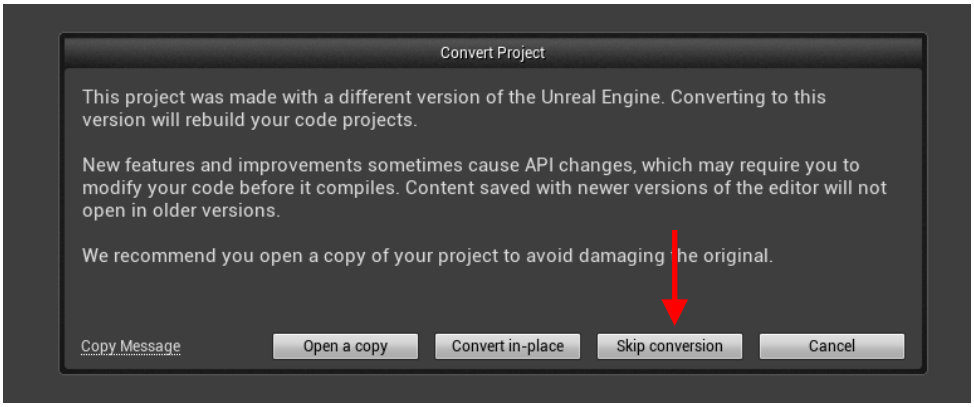
CBP 10-02-23.

Unreal on the Lab Machines

Download the Unreal 4 Assets zip. Click on the zip and drag the folder onto the Desktop which will unzip. Open Unreal 4, and hit **More** then **Browse** and find the file **Pedestrians.uproject** which you should open. A number of things may happen depending on which computer you are working on. Most likely you will be faced with this dialogue box, in which case select **More Options**.



Now a second dialogue will pop up, and choose **Skip conversion**.

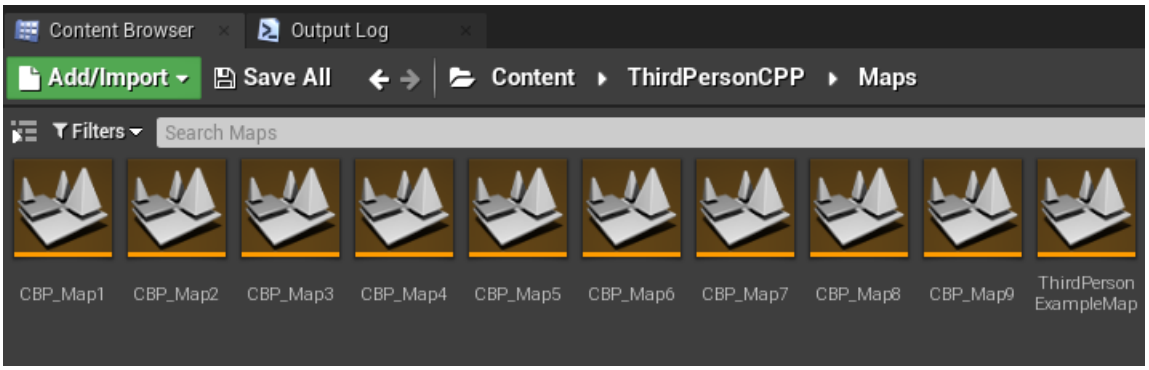


Hopefully the level will open

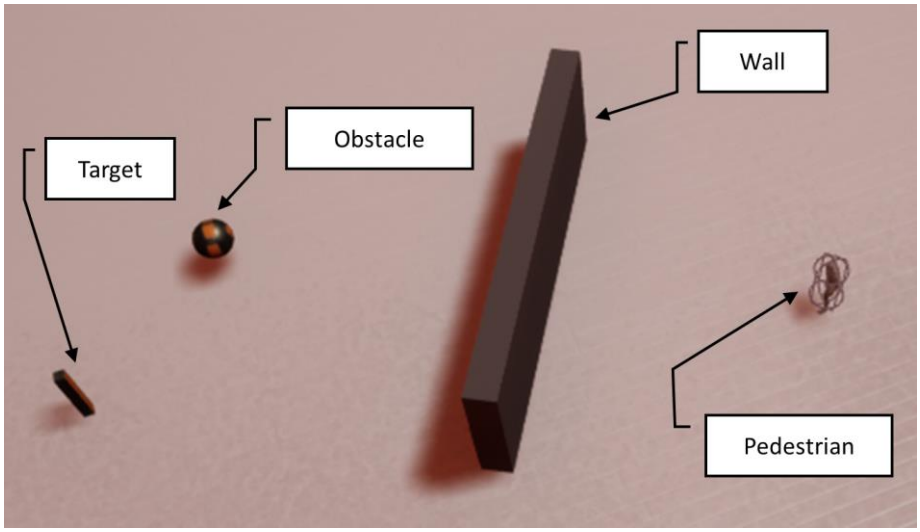
Working with the Unreal-4 Editor

These notes will help you to understand (i) How to create your own map (level) in the Unreal Editor and (ii) How to update the parameters for a load of pedestrians at the same time.

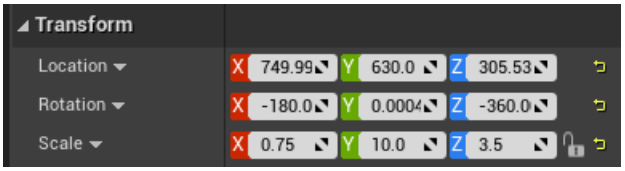
In the **Pedestrians** project you will find a number of test maps I have created. These were intended for me to test the code but could be a useful starting point for you. You can find these in the Content Browser in the folder shown here. The contents of each map will be described at the end of this document



A good place to start is to open the map **CBP_Map9** which contains the objects you will need for your own investigation. Here's the map with the objects labelled, there is one instance of each,



To create your own map

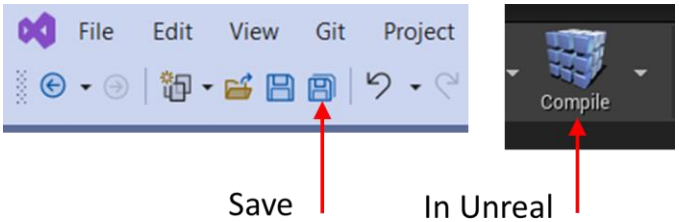
- In the Content Browser right-click on **CBP_Map9** and select Duplicate. Give the new map your preferred name.
- To create your own arrangement of pedestrians, walls, obstacles and targets
  - Select the object in the level.
  - Press CTRL-W to duplicate.
  - Move the object to where you want to.
  - To set the position and orientation use the tools 
  - To fine-tune the position and orientation, left-click on the object and find the Transform properties in the Details panel, bottom right. The Transform box is really useful to align things, e.g., giving the same x or y values.
  - 

Changing Parameters

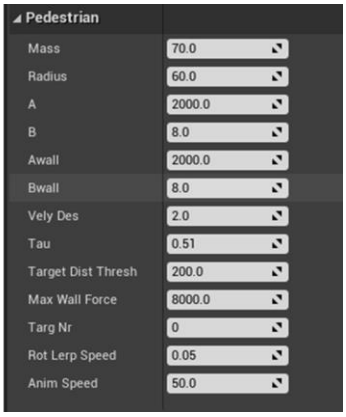
All parameters are changed using the Pedestrian class in Visual Studio. Open the file **Pedestrians.sln** and then open the file **Pedestrian.h**. You will see a list of parameters.

```
// Constants
UPROPERTY(EditAnywhere, BlueprintReadOnly)
    float mass = 70;
UPROPERTY(EditAnywhere, BlueprintReadOnly)
    float radius = 60;
UPROPERTY(EditAnywhere, BlueprintReadOnly)
    float A = 2000;
UPROPERTY(EditAnywhere, BlueprintReadOnly)
    float B = 8.0;
UPROPERTY(EditAnywhere, BlueprintReadOnly)
    float Awall = 2000;
UPROPERTY(EditAnywhere, BlueprintReadOnly)
    float Bwall = 8.0;
UPROPERTY(EditAnywhere, BlueprintReadOnly)
    float velyDes = 2.0;
UPROPERTY(EditAnywhere, BlueprintReadOnly)
    float tau = 0.51;
UPROPERTY(EditAnywhere, BlueprintReadOnly)
    float targetDistThresh = 200;
```

You can change these. But **do not build in Visual Studio**. Instead, **save** in Visual Studio and then **Compile** in the Editor like this.



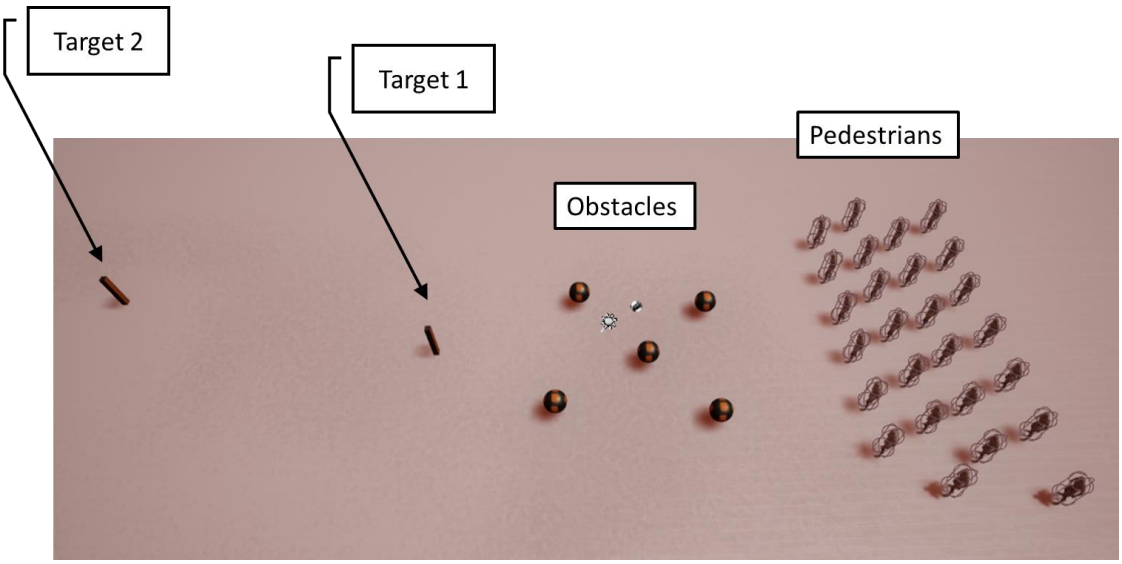
When you change parameters and compile as above, it's best to check the changes have reached the pedestrians. So in the map, select a pedestrian and check its properties like this



The Basic Idea

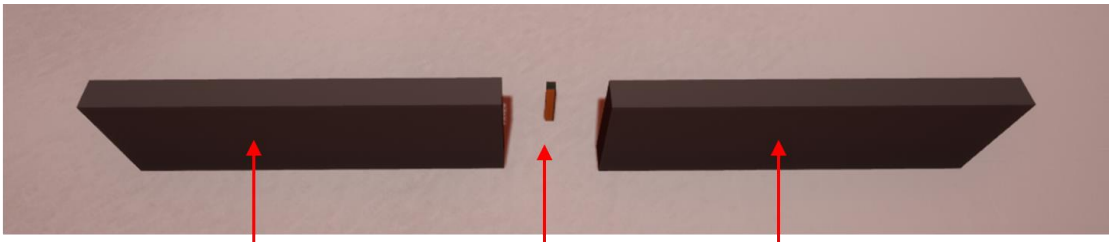
This is shown in the figure below. You create a load of pedestrians which must move. They move by being attracted to *targets.* So all pedestrians will be attracted to **Target 1** and when they have reached that target, then they are attracted to **Target 2**. In this case there are a load of obstacles in the way, and our SFModel enables the pedestrians to move around (and between) these. Of course you could have walls as well.

So in constructing your map, you need to place one or more targets, then one or more pedestrians, then some geometry made up of obstacles and walls in between. All maps will have this arrangement.



Passing through Openings

Getting the pedestrians to pass through openings such as doors can be problematical. The location of the Target in relation to the walls is important. Here's an arrangement which I've found works in most cases, for a large (=50) number of pedestrians.



Location {X = 750, Y = -630, Z = 300}
Rotation {X = -180, Y = 0, Z = 0}
Scale {X = 0.75, Y = 10, Z = 3.5}

Location {X = 750, Y = 630, Z = 300}
Rotation {X = -180, Y = 0, Z = 0}
Scale {X = 0.75, Y = 10, Z = 3.5}

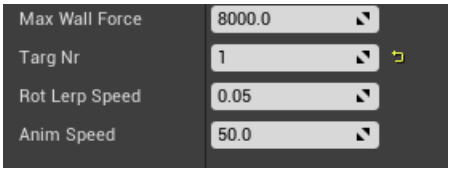
Location {X = 880, Y = 0, Z = 130}
Rotation {X = 0, Y = 0, Z = 0}
Scale {X = 0.25, Y = 0.25, Z = 3.5}

The crucial thing is that the X-location of the target is downwind of the walls. The walls have **X = 750 UUs** and the target has **X = 880 UUs**.

Colliding Crowds

The map **CBP_Map8** is an attempt to investigate collisions between crowds. The pedestrians near the left target move to the right target and vice-versa.

This is done by specifying the target number in the Pedestrian properties like this.

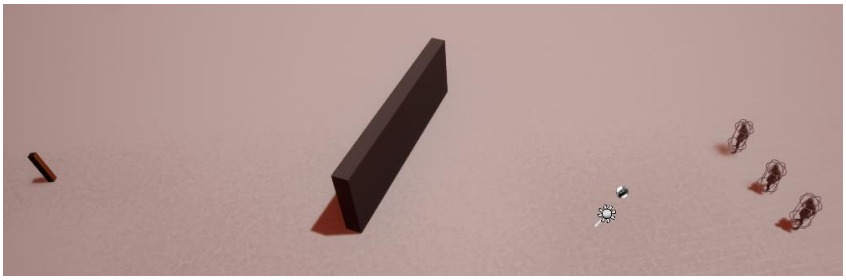


You'll have to do this by trial and error or copy and mangle the above map.

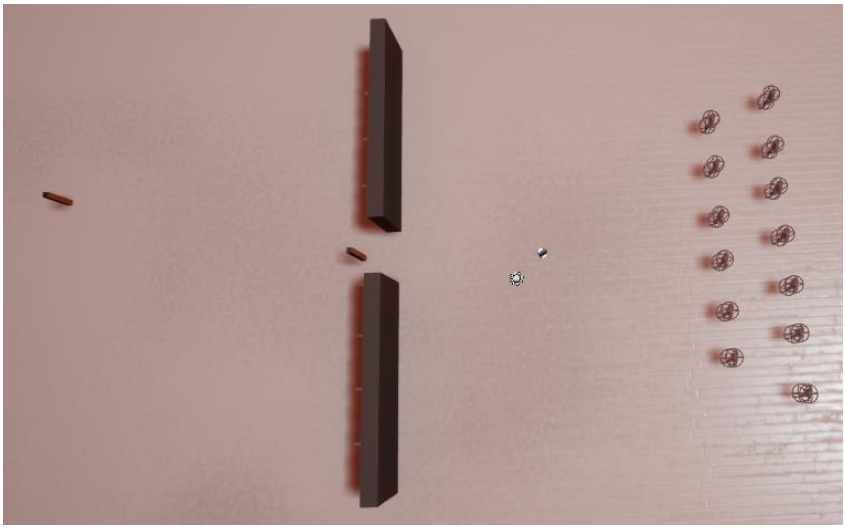
Plotting Trajectories with Octave

Some Test Maps

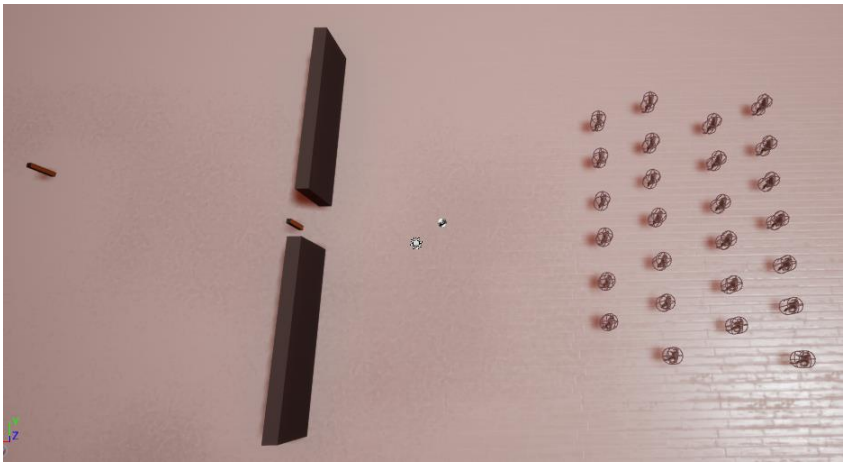
Here are some test maps I used. The majority were not thought out.



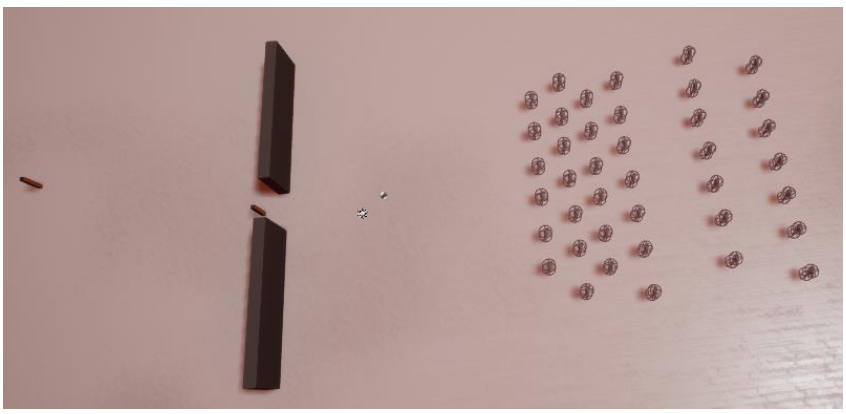
CBP_Map1 Three peds, one wall, one target



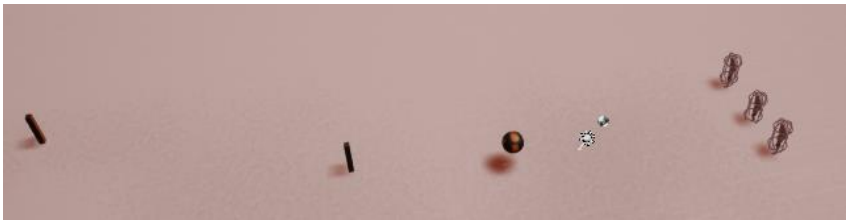
CBP_Map2 Peds going through a gap. First target carefully placed



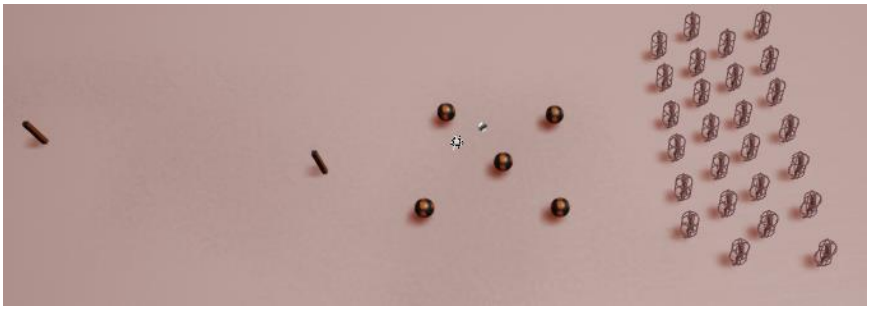
CBP_Map3 Same with more peds



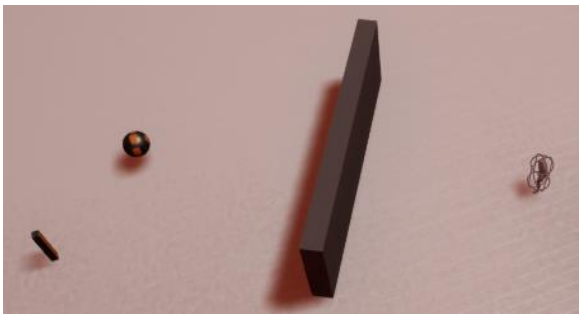
CBP_Map5 Same with more peds



CBP_Map6 Three peds, two targets and one obstacle



CBP_Map7 Crowd with arrangement of obstacles



CBP_Map9 One wall, one target and one ped.

CBP_Map8 Crowd Collision