

```

1  /*
2  -----
3      Procedural Map Generator v2
4      Written by Jeff Fisher
5      v.1 : July 3, 2013 - Junior group project)
6      v.2 : October 27, 2013 - Personal side project)
7      v.3 : August 31, 2014 - Personal extension of senior capstone)
8  -----
9      */
10 using UnityEngine;
11 using System;
12 using System.Collections;
13 using System.Collections.Generic;
14 using System.IO;
15
16 public class sMapBuilder : MonoBehaviour{
17     /*
18     -----
19         Custom data types
20     -----
21     */
22     // Enumerates the directions that can be used in this application to simplify angles in 45 degree increments
23     // --If the selection of map segment shapes is modified this may need to be changed.--
24     enum DIR { angle0, angle45, angle90, angle135, angle180, angle225, angle270, angle315 }; // These are casted to (int) in use
25
26     struct MAPCUBE{
27         public int cube;    // Identity of cube
28         public DIR yAngle;  // Angle of cube on Y axis expressed as DIR
29         public int height;  // height of the cube in the map (+1, 0, or -1)
30         // true is an open side in that direction false is a closed side
31         public bool dir0;
32         public bool dir2;
33         public bool dir4;
34         public bool dir6;
35
36         public void Set( int c, DIR a, int h, bool d0, bool d2, bool d4, bool d6 ){
37             cube = c;
38             yAngle = a;
39             height = h;
40             dir0 = d0;
41             dir2 = d2;
42             dir4 = d4;
43             dir6 = d6;
44         }
45     };
46
47     struct CUBELOC{
48         public int x;    // Column of cube in map
49         public int z;    // Row of cube in map
50         public void Set( int u, int w ){
51             x = u;
52             z = w;
53         }
54     };
55
56     class ASTARNODE{
57         private int x;    // Column of cube in map
58         private int z;    // Row of cube in map
59         private float dist; // Distance between start and finish
60         private bool open;
61         public void Set( int u, int w, float d, bool o ){
62             x = u;
63             z = w;

```

```

64         dist = d;
65         open = o;
66     }
67     public void Close(){
68         open = false;
69     }
70     public int GetX(){
71         return x;
72     }
73     public int GetZ(){
74         return z;
75     }
76     public float GetDist(){
77         return dist;
78     }
79     public bool GetOpen(){
80         return open;
81     }
82     public string StringX(){
83         return x.ToString();
84     }
85     public string StringZ(){
86         return z.ToString();
87     }
88     public string StringDist(){
89         return dist.ToString();
90     }
91     public string StringOpen(){
92         return open.ToString();
93     }
94 }
95
96 // Enumerates the specific sides and corners found in the set of map segments being used.
97 // --If the selection of map segments is modified, this may need to be changed.--
98 enum SIDE {        // Enumerated data type for different map cube side and corner characteristics
99     broken,        // Indicates a side or corner that is not applicable to matching purposes
100     solidCorner,    // Indicates that a corner must be matched to other corners that contain
101     structure
102     emptyCorner,    // Indicates that a corner must be matched to other corners that contain no
103     structure
104     hallSide,       // Indicates a side that is a hallway
105     solidSide,       // Indicates a side that is a solid wall
106     openSide,        // Indicates a side that is completely open
107     lWallSide,       // Indicates an otherwise open side with a wall on the left
108     rWallSide        // Indicates an otherwise open side with a wall on the right
109 };
110
111 // Struct containing the array of cube sides and code to test for sides matching
112 // --If the cube sides found in the selection of cubes is modified this will need to be changed--
113 struct CUBESIDES{
114     public SIDE[] sideArray;
115
116     public CUBESIDES( SIDE side0, SIDE side45, SIDE side90, SIDE side135,
117         SIDE side180, SIDE side225, SIDE side270, SIDE side315 ){
118         sideArray = new SIDE[8];
119         sideArray[0] = side0;
120         sideArray[1] = side45;
121         sideArray[2] = side90;
122         sideArray[3] = side135;
123         sideArray[4] = side180;
124         sideArray[5] = side225;
125         sideArray[6] = side270;
126         sideArray[7] = side315;
127     }
128     // This function matches cubes
129     // --It needs to be changed if the selection of cube side types is changed

```

```

128     public SIDE GetSide( DIR angle ){
129         return sideArray[ (int)angle ];
130     }
131
132     public bool CompareSides( SIDE trySideType, DIR chkCubeSide, DIR chkCubeAngle ){
133         int comboCubeAngle = (((int)chkCubeSide + 4) % 8) - (int)chkCubeAngle + 8 % 8;
134
135         if( sideArray[comboCubeAngle] == SIDE.broken ) return true;
136         if( sideArray[comboCubeAngle] == SIDE.lWallSide){
137             if( trySideType == SIDE.rWallSide) return true;
138             else return false;
139         }
140         if( sideArray[comboCubeAngle] == SIDE.rWallSide){
141             if( trySideType == SIDE.lWallSide ) return true;
142             else return false;
143         }
144         if( sideArray[comboCubeAngle] == trySideType ) return true;
145         else return false;
146     }
147 };
148
149 /*
150 -----
151     Make Prefabs Available
152 -----
153 */
154 public int mapLength = 16;
155 public int mapWidth = 16;
156 public bool spawnMonsters = false;
157 //public int numMonsters = 0;
158 public bool appendFileOutput = false;
159
160 public Transform p14E_EndHall;
161 public Transform p01H_StraightHall;
162 public Transform p02H_CornerHall;
163 public Transform p03H_CrossHall;
164 public Transform p04H_TeeHall;
165 public Transform p05H_HallRoomR;
166 public Transform p06H_HallRoomL;
167 public Transform p07H_HallRoomLR;
168 public Transform p08H_HallRoom;
169 public Transform p09H_RoomAngle;
170 public Transform p10H_SideRoom;
171 public Transform p11H_CornerRoom;
172 public Transform p12H_OffsetRoom;
173 public Transform p13H_OpenRoom;
174 public Transform p14H_EndHall;
175 public Transform p01L_StraightHall;
176 public Transform p02L_CornerHall;
177 public Transform p03L_CrossHall;
178 public Transform p04L_TeeHall;
179 public Transform p05L_HallRoomR;
180 public Transform p06L_HallRoomL;
181 public Transform p07L_HallRoomLR;
182 public Transform p08L_HallRoom;
183 public Transform p09L_RoomAngle;
184 public Transform p10L_SideRoom;
185 public Transform p11L_CornerRoom;
186 public Transform p12L_OffsetRoom;
187 public Transform p13L_OpenRoom;
188 public Transform p14L_EndHall;
189 public Transform p01W_StraightHall;
190 public Transform p02W_CornerHall;
191 public Transform p03W_CrossHall;
192 public Transform p04W_TeeHall;
193 public Transform p05W_HallRoomR;

```

```

194 public Transform p06W_HallRoomL;
195 public Transform p07W_HallRoomLR;
196 public Transform p08W_HallRoom;
197 public Transform p09W_RoomAngle;
198 public Transform p10W_SideRoom;
199 public Transform p11W_CornerRoom;
200 public Transform p12W_OffsetRoom;
201 public Transform p13W_OpenRoom;
202 public Transform p14W_EndHall;
203 public Transform p14X_EndHall;
204 public Transform pSquareGrate;
205 public Transform pSpiralStair;
206 public Transform p17_SolidCube;
207 public Transform pEnder;
208
209 public Transform pHangingCageSide;
210 public Transform pHangingCageQuad;
211 public Transform pJudasCradleCorner;
212 public Transform pJudasCradleSide;
213 public Transform pPilloryCorner;
214 public Transform pPillorySide;
215 public Transform pWristShacklesAngle;
216 public Transform pWristShacklesCorner;
217 public Transform pWristShacklesSide;
218 public Transform pWristShacklesQuad;
219 public Transform pSpreadShacklesCorner;
220 public Transform pSpreadShacklesSide;
221 public Transform pTortureRackCorner;
222 public Transform pTortureRackSide;
223 public Transform pSewerPipeSide;
224 public Transform pCoffinCageCorner;
225 public Transform pCoffinCageSide;
226
227 public Transform Spectre;
228 public Transform Lantern;
229 public Transform LanternCore;
230 public Transform SteadyLight;
231 public Transform pWaterPlane;
232 private int startX;
233 private int startZ;
234 private int endX;
235 private int endZ;
236
237 /*
238 -----
239 Initialize Data and Initiate Build
240 -----
241 */
242 void Start(){
243     MAPCUBE[,] MapArray = new MAPCUBE[(mapLength+2),(mapWidth+2)]; // Array containing all of the
244     cubes in the map
245     List<CUBELOC> OpenSquares = new List<CUBELOC>(); // Cubes that need to be filled by row and
246     column
247
248     // Initializes list of cubes used.
249     // --If selection of cubes is changed then this will need to be updated--
250     CUBESIDES[] CubeInfo = new CUBESIDES[16];
251     // Empty cube, may be used later, currently a space-filler
252     CubeInfo[0] = new CUBESIDES( SIDE.broken, SIDE.broken, SIDE.broken, SIDE.broken,
253     SIDE.broken, SIDE.broken, SIDE.broken, SIDE.broken );
254     // StraightHall cube (straight section of hallway)
255     CubeInfo[1] = new CUBESIDES( SIDE.hallSide, SIDE.solidCorner, SIDE.solidSide, SIDE.solidCorner
256     ,
257     SIDE.hallSide, SIDE.solidCorner, SIDE.solidSide, SIDE.solidCorner
258     );
259     // CornerHall cube (hall makes 90 degree turn)

```

```

256     CubeInfo[2] = new CUBESIDES( SIDE.hallSide, SIDE.solidCorner, SIDE.solidSide, SIDE.solidCorner ✓
257     ,
258         SIDE.solidSide, SIDE.solidCorner, SIDE.hallSide, SIDE.solidCorner ✓
259     );
260     // CrossHall cube (4-way intersection)
261     CubeInfo[3] = new CUBESIDES( SIDE.hallSide, SIDE.solidCorner, SIDE.hallSide, SIDE.solidCorner, ✓
262     ,
263         SIDE.hallSide, SIDE.solidCorner, SIDE.hallSide, SIDE.solidCorner ✓
264     );
265     // TeeHall cube (3-way intersetion)
266     CubeInfo[4] = new CUBESIDES( SIDE.solidSide, SIDE.solidCorner, SIDE.hallSide, SIDE.solidCorner ✓
267     ,
268         SIDE.hallSide, SIDE.solidCorner, SIDE.hallSide, SIDE.solidCorner ✓
269     );
270     // HallRoomR cube (hall meets corner of room, room opens to the right)
271     CubeInfo[5] = new CUBESIDES( SIDE.rWallSide, SIDE.emptyCorner, SIDE.lWallSide, SIDE. ✓
272     solidCorner,
273         SIDE.solidSide, SIDE.solidCorner, SIDE.hallSide, SIDE.solidCorner ✓
274     );
275     // HallRoomL cube (hall meeds corner of room, room opens to the left)
276     CubeInfo[6] = new CUBESIDES( SIDE.solidSide, SIDE.solidCorner, SIDE.rWallSide, SIDE. ✓
277     emptyCorner,
278         SIDE.lWallSide, SIDE.solidCorner, SIDE.hallSide, SIDE.solidCorner ✓
279     );
280     // HallRoomLR cube (2 halls enter corner of room from left and right)
281     CubeInfo[7] = new CUBESIDES( SIDE.rWallSide, SIDE.emptyCorner, SIDE.lWallSide, SIDE. ✓
282     solidCorner,
283         SIDE.hallSide, SIDE.solidCorner, SIDE.hallSide, SIDE.solidCorner ✓
284     );
285     // HallRoom (hall in center of only wall)
286     CubeInfo[8] = new CUBESIDES( SIDE.openSide, SIDE.emptyCorner, SIDE.lWallSide, SIDE.solidCorner ✓
287     ,
288         SIDE.hallSide, SIDE.solidCorner, SIDE.rWallSide, SIDE.emptyCorner ✓
289     );
290     // RoomAngle (inside corner of a room that turns)
291     CubeInfo[9] = new CUBESIDES( SIDE.openSide, SIDE.emptyCorner, SIDE.openSide, SIDE.emptyCorner, ✓
292     ,
293         SIDE.lWallSide, SIDE.solidCorner, SIDE.rWallSide, SIDE. ✓
294     emptyCorner);
295     // SideRoom (side of a room)
296     CubeInfo[10] = new CUBESIDES( SIDE.openSide, SIDE.emptyCorner, SIDE.lWallSide, SIDE. ✓
297     solidCorner,
298         SIDE.solidSide, SIDE.solidCorner, SIDE.rWallSide, SIDE. ✓
299     emptyCorner );
300     // CornerRoom (corner of a room)
301     CubeInfo[11] = new CUBESIDES( SIDE.rWallSide, SIDE.emptyCorner, SIDE.lWallSide, SIDE. ✓
302     solidCorner,
303         SIDE.solidSide, SIDE.solidCorner, SIDE.solidSide, SIDE. ✓
304     solidCorner );
305     // OffsetRoom (2 rooms meet at corners)
306     CubeInfo[12] = new CUBESIDES( SIDE.lWallSide, SIDE.solidCorner, SIDE.rWallSide, SIDE. ✓
307     emptyCorner,
308         SIDE.lWallSide, SIDE.solidCorner, SIDE.rWallSide, SIDE. ✓
309     emptyCorner );
310     // OpenRoom (open central area)
311     CubeInfo[13] = new CUBESIDES( SIDE.openSide, SIDE.emptyCorner, SIDE.openSide, SIDE.emptyCorner ✓
312     ,
313         SIDE.openSide, SIDE.emptyCorner, SIDE.openSide, SIDE.emptyCorner ✓
314     );
315     // EndHall (dead-end hallway)
316     CubeInfo[14] = new CUBESIDES( SIDE.hallSide, SIDE.solidCorner, SIDE.solidSide, SIDE. ✓
317     solidCorner,
318         SIDE.solidSide, SIDE.solidCorner, SIDE.solidSide, SIDE. ✓
319     solidCorner );
320     // SolidCube (what it says - used for borders)
321     CubeInfo[(17-2)] = new CUBESIDES( SIDE.solidSide, SIDE.solidCorner, SIDE.solidSide, SIDE. ✓

```

```

solidCorner,
296 SIDE.solidSide, SIDE.solidCorner, SIDE.solidSide, SIDE.
solidCorner );
297
298 //Initiates build of main map
299 int buildAttempts = 0;
300 do{
301     BuildMap( ref MapArray, ref OpenSquares, CubeInfo );
302     buildAttempts++;
303 }
304 while( !FindPath( ref MapArray ) && buildAttempts < 4 );
305
306 InstantiateMap (ref MapArray);
307
308 // Fills the map with stuff
309 SpawnStuff( ref MapArray );
310 }
311
312
313 /*
314 -----
315 Builds the core of the map inside the frame
316 -----
317 */
318 void BuildMap( ref MAPCUBE[,] MapArray, ref List<CUBELOC> OpenSquares, CUBESIDES[] CubeInfo ){
319     // Initializes map information variables
320     int lastCubeNumber = 14; // The highest numbered possible cube
321     int firstCubeNumber = 1; // The lowest numbered possible cube
322     startX = (mapWidth / 2) - 1; // default X position of start cube
323     startZ = 1; // Z position of start cube
324     endX = (mapWidth / 2) + (mapWidth % 2) + 2; // default X position of end cube
325     endZ = mapLength; // Z position of end cube
326
327     Vector3 tempPos = new Vector3(0.0f, 0.0f, 0.0f);
328     int x = 0; // The column of the current cube being modified within the map
329     int z = 0; // They row of the current cube being modified within the map
330     int yAngle = 0; // Temp variable containing the current rotation of a cube around the y axis
331     MAPCUBE tempMapCube = new MAPCUBE(); // Temp variable for a Map Cube
332     CUBELOC tempCubeLoc = new CUBELOC(); // Temp variable for the location of a cube
333     Vector3 cubePosVect = new Vector3(0,0,0); // Contains 3D locations of cubes to be placed
334     int cube = -1;
335     int dir = -1;
336
337     bool cubeWasSet = false; // used to determine if a cube was set successfully
338     bool[] wasTriedCubes = new bool[lastCubeNumber+1]; // Tracks cubes that have been determined
not to fit
339     MAPCUBE[] BorderCube = new MAPCUBE[8]; // Array containing the cubes around a cube to be
modified
340     int testCubePick = -1; // variable to hold the number of the cube being tried
341     int tryCubeAngle = -1; // tracks the angle that the cube is being tried at
342     int loopCount = 0; // Tracks the number of loops to exit if the count becomes excessive
343     int curWorkSquare = 0; // Tracks the index on the list of open sides that is being worked on
344
345     // Initializes values of Map Array
346     tempMapCube.Set( -1, 0, 0, false, false, false, false );
347     for( x = 0; x < (mapLength+2); x++){
348         for( z = 0; z < (mapWidth+2); z++){
349             MapArray[x,z] = tempMapCube;
350         }
351     }
352
353     // Sets row, column and angle of start cube
354     yAngle = 90;
355     x = startX;
356     z = startZ;
357

```

```

358 // Adds start point cube to map list
359 cube = 14;
360 dir = yAngle / 45;
361 tempMapCube.Set( cube, (DIR)dir, 0,
362     !(CubeInfo[cube].sideArray[((8-dir)%8)] == SIDE.solidSide),
363     !(CubeInfo[cube].sideArray[((10-dir)%8)] == SIDE.solidSide),
364     !(CubeInfo[cube].sideArray[((12-dir)%8)] == SIDE.solidSide),
365     !(CubeInfo[cube].sideArray[((14-dir)%8)] == SIDE.solidSide) );
366 MapArray[ x, z ] = tempMapCube;
367
368 // Adds a cross hall cube to the map at the open end of the start cube
369 z++;
370 cubePosVect.Set( ((x * 5.12f) - 40.96f), 1.8f, ((z * 5.12f)) );
371
372 // Adds the cross hall cube to the map list
373 cube = 3;
374 dir = yAngle / 45;
375 tempMapCube.Set( 3, (DIR)dir, 0,
376     !(CubeInfo[cube].sideArray[((8-dir)%8)] == SIDE.solidSide),
377     !(CubeInfo[cube].sideArray[((10-dir)%8)] == SIDE.solidSide),
378     !(CubeInfo[cube].sideArray[((12-dir)%8)] == SIDE.solidSide),
379     !(CubeInfo[cube].sideArray[((14-dir)%8)] == SIDE.solidSide) );
380 MapArray[ x, z ] = tempMapCube;
381
382 // Add squares surrounding the cross hall cube to the list of open squares
383 tempCubeLoc.Set( (x-1), z );
384 OpenSquares.Add( tempCubeLoc );
385 tempCubeLoc.Set( x, (z+1) );
386 OpenSquares.Add( tempCubeLoc );
387 tempCubeLoc.Set( (x+1), z );
388 OpenSquares.Add( tempCubeLoc );
389
390 // Sets row, column and angle of end cube
391 yAngle = 270;
392 x = endX;
393 z = endZ;
394
395 // Adds map end point to map list
396 cube = 14;
397 dir = yAngle / 45;
398 tempMapCube.Set( 14, (DIR)dir, 1,
399     !(CubeInfo[cube].sideArray[((8-dir)%8)] == SIDE.solidSide),
400     !(CubeInfo[cube].sideArray[((10-dir)%8)] == SIDE.solidSide),
401     !(CubeInfo[cube].sideArray[((12-dir)%8)] == SIDE.solidSide),
402     !(CubeInfo[cube].sideArray[((14-dir)%8)] == SIDE.solidSide) );
403 MapArray[ x, z ] = tempMapCube;
404
405 // Creates a cross hall cube at the opening of the end point cube
406 z--;
407 cubePosVect.Set( ((x * 5.12f) - 40.96f), 1.8f, ((z * 5.12f)) );
408
409 // Adds cross hall cube to the list of map cubes
410 cube = 3;
411 dir = yAngle / 45;
412 tempMapCube.Set( 3, (DIR)dir, 1,
413     !(CubeInfo[cube].sideArray[((8-dir)%8)] == SIDE.solidSide),
414     !(CubeInfo[cube].sideArray[((10-dir)%8)] == SIDE.solidSide),
415     !(CubeInfo[cube].sideArray[((12-dir)%8)] == SIDE.solidSide),
416     !(CubeInfo[cube].sideArray[((14-dir)%8)] == SIDE.solidSide) );
417 MapArray[ x, z ] = tempMapCube;
418
419 // Add squares surrounding the cross hall cube to the list of open squares
420 tempCubeLoc.Set( (x-1), z );
421 OpenSquares.Add( tempCubeLoc );
422 tempCubeLoc.Set( x, (z-1) );
423 OpenSquares.Add( tempCubeLoc );

```

```

424     tempCubeLoc.Set( (x+1), z );
425     OpenSquares.Add( tempCubeLoc );
426
427
428     // FOR loops build sides of "frame" composed of solid cubes to contain the map
429     // --If size of map is changed, this will need to be modified
430     tempMapCube.Set (15, (DIR)0, 0, false, false, false, false);
431
432     x = 0;
433     for( z = 0; z <= mapLength+1; z++ ){
434         cubePosVect.Set( ((x * 5.12f) - 40.96f), -1.28f, (z * 5.12f) );
435         Instantiate( p17_SolidCube, cubePosVect, Quaternion.Euler(-90,0,0) );
436         MapArray[x,z] = tempMapCube;
437     }
438
439     z = mapLength+1;
440     for( x = 0; x <= mapWidth+1; x++ ){
441         cubePosVect.Set( ((x * 5.12f) - 40.96f), -1.28f, (z * 5.12f) );
442         Instantiate( p17_SolidCube, cubePosVect, Quaternion.Euler(-90,0,0) );
443         MapArray[x,z] = tempMapCube;
444     }
445
446     x = mapWidth+1;
447     for( z = mapLength+1; z >= 0; z-- ){
448         cubePosVect.Set( ((x * 5.12f) - 40.96f), -1.28f, (z * 5.12f) );
449         Instantiate( p17_SolidCube, cubePosVect, Quaternion.Euler(-90,0,0));
450         MapArray[x,z] = tempMapCube;
451     }
452
453     z = 0;
454     for( x = mapWidth+1; x >= 0; x-- ){
455         cubePosVect.Set( ((x * 5.12f) - 40.96f), -1.28f, (z * 5.12f) );
456         Instantiate( p17_SolidCube, cubePosVect, Quaternion.Euler(-90,0,0));
457         MapArray[x,z] = tempMapCube;
458     }
459
460     // Loops as long as there are open squares to be filled
461     while( curWorkSquare < OpenSquares.Count ){
462         // Iterate through open edge square list placing blocks
463         // Verify that the open square has not been filled already
464         tempCubeLoc = OpenSquares[ curWorkSquare ];
465         x = tempCubeLoc.x;
466         z = tempCubeLoc.z;
467         tempMapCube = MapArray[x,z];
468         if( tempMapCube.cube == -1 ){ // -1 assigned to represent empty cube
469             // Create a list of all cubes bordering the open square
470             BorderCube[0] = MapArray[(x+1), z ]; // right side
471             BorderCube[1] = MapArray[(x+1),(z+1)]; // upper right corner
472             BorderCube[2] = MapArray[ x, (z+1)]; // top side
473             BorderCube[3] = MapArray[(x-1),(z+1)]; // upper left corner
474             BorderCube[4] = MapArray[(x-1), z ]; // left side
475             BorderCube[5] = MapArray[(x-1),(z-1)]; // lower left corner
476             BorderCube[6] = MapArray[ x, (z-1)]; // bottom side
477             BorderCube[7] = MapArray[(x+1),(z-1)]; // lower right corner
478
479             // Initialize the array of cubes that have been tried;
480             wasTriedCubes[0] = true; // Workaround so that search skips 0
481             for( int idx = firstCubeNumber; idx <= lastCubeNumber; idx++){
482                 wasTriedCubes[idx] = false;
483             }
484             while( !cubeWasSet ){ // Still searching for a cube?
485                 // Break out of endless loop
486                 loopCount++;
487                 if( loopCount > 5000 ) return;
488
489                 // Pick a random cube that hasn't been tried yet

```



```

490         bool alreadyTried = false;
491         do{
492             // Searches for an index which hasn't been tried (is false),
493             // If it finds one the equality is false
494             testCubePick = UnityEngine.Random.Range(1, 15);
495             alreadyTried = wasTriedCubes[testCubePick];
496             wasTriedCubes[testCubePick] = true;
497         }
498         while( alreadyTried );
499
500         // Check if that cube fits
501         tryCubeAngle = TryCube( testCubePick, x, z, ref MapArray, ref BorderCube,
502                                ref OpenSquares, ref curWorkSquare, CubeInfo );
503         if( tryCubeAngle > -1 ){ // -1 is no angle fits, other number is int cast of DIR
angle
504             cubWasSet = true;
505             UpdateMap( testCubePick, x, z, ref MapArray, ref BorderCube,
506                       ref OpenSquares, ref curWorkSquare, CubeInfo, (DIR)tryCubeAngle );
507         }
508     }
509     // reset control variable
510     cubWasSet = false;
511 }
512 else{
513     curWorkSquare++;
514 }
515 }
516 }
517 /*
518 -----
519 Checks and sets map cubes
520 -----
521 */
522 int TryCube( int testCube, int x, int z, ref MAPCUBE[,] MapArray, ref MAPCUBE[] BorderCube,
523              ref List<CUBELOC> OpenSquares, ref int curWorkSquare, CUBESIDES[] CubeInfo ){
524     int comboRotAngle = -1; // -1 is no match, a match is the int equivalent of the DIR angle
525     bool checkedCubeMatches = true;
526     int rndAngle = UnityEngine.Random.Range(0,3) * 2;
527     for( int forRot = 0; forRot < 8 ; forRot += 2 ){
528         comboRotAngle = (rndAngle + forRot) % 8;
529         checkedCubeMatches = true;
530         for( int forSide = 0; forSide < 8; forSide++ ){
531             // Gets the side of the cube being tested
532             DIR rotCubeSide = (DIR)((forSide - comboRotAngle + 8) % 8);
533             SIDE trySideType = CubeInfo[ testCube ].GetSide( rotCubeSide );
534             DIR chkCubeAngle = BorderCube[ forSide ].yAngle;
535             DIR chkCubeSide = (DIR)forSide;
536             int chkCubeNum = BorderCube[ forSide ].cube;
537             if( chkCubeNum > -1 ){
538                 // Compares side of cube being tested to bordering cube
539                 checkedCubeMatches = CubeInfo[ chkCubeNum ].CompareSides( trySideType, chkCubeSide
, chkCubeAngle );
540             if( !checkedCubeMatches ){
541                 break;
542             }
543         }
544     }
545     if( checkedCubeMatches ){
546         return comboRotAngle;
547     }
548     else{
549         comboRotAngle = -1;
550     }
551 }
552 return comboRotAngle = -1;
553 }

```

```

554
555  /*
556 -----
557 Updates the mapping data
558 -----
559 */
560
561 void UpdateMap( int testCube, int x, int z, ref MAPCUBE[,] MapArray, ref MAPCUBE[] BorderCube,
562 ref List<CUBELOC> OpenSquares, ref int curWorkSquare, CUBESIDES[] CubeInfo, DIR
563 comboRotAngle){
564     MAPCUBE tempMapCube = new MAPCUBE(); // Cube type that was selected
565     CUBELOC tempCubeLoc = new CUBELOC(); // Location the cube will be placed
566     Vector3 cubePosVect = new Vector3( ((x * 5.12f) - 40.96f), 1.8f, ((z * 5.12f)) ); // 3D cube
567     position
568     int newAngle = (int)comboRotAngle;// * 45;
569
570     //-----
571     // **WORKAROUND**
572     // Correcting rotation of map cube 6
573     if( testCube == 6 ) newAngle += 2; // This cube model needs a +90 degree correction
574     //End of workaround
575     //-----
576
577     Vector3 cubeAngle = new Vector3( -90, ((newAngle * -1) + 180), 0 );
578
579     tempMapCube.Set( testCube, (DIR)comboRotAngle, 0,
580         !(CubeInfo[testCube].sideArray[((8-(int)comboRotAngle)%8)] == SIDE.solidSide),
581         !(CubeInfo[testCube].sideArray[((10-(int)comboRotAngle)%8)] == SIDE.solidSide)
582     ,
583         !(CubeInfo[testCube].sideArray[((12-(int)comboRotAngle)%8)] == SIDE.solidSide)
584     ,
585         !(CubeInfo[testCube].sideArray[((14-(int)comboRotAngle)%8)] == SIDE.solidSide)
586 );
587 MapArray[x,z] = tempMapCube;
588
589 if( BorderCube[0].cube == -1 ){
590     SIDE side = CubeInfo[ testCube ].GetSide( (DIR)((8 - newAngle) % 8) );
591     if( ((side != SIDE.broken) && (side != SIDE.solidSide)) ){
592         tempCubeLoc.Set( (x+1), z );
593         OpenSquares.Add( tempCubeLoc );
594     }
595 }
596 if( BorderCube[2].cube == -1 ){
597     SIDE side = CubeInfo[ testCube ].GetSide( (DIR)((10 - newAngle) % 8) );
598     if( ((side != SIDE.broken) && (side != SIDE.solidSide)) ){
599         tempCubeLoc.Set( x, (z+1) );
600         OpenSquares.Add( tempCubeLoc );
601     }
602 }
603 if( BorderCube[4].cube == -1 ){
604     SIDE side = CubeInfo[ testCube ].GetSide( (DIR)((12 - newAngle) % 8) );
605     if( ((side != SIDE.broken) && (side != SIDE.solidSide)) ){
606         tempCubeLoc.Set( (x-1), z );
607         OpenSquares.Add( tempCubeLoc );
608     }
609 }
610 if( BorderCube[6].cube == -1 ){
611     SIDE side = CubeInfo[ testCube ].GetSide( (DIR)((14 - newAngle) % 8) );
612     if( ((side != SIDE.broken) && (side != SIDE.solidSide)) ){
613         tempCubeLoc.Set( x, (z-1) );
614         OpenSquares.Add( tempCubeLoc );
615     }
616 }
617 curWorkSquare++;
618 }
619

```

```

615  /*
616  -----
617  Verify Path to Goal (A*)
618  -----
619  */
620  bool FindPath( ref MAPCUBE[,] MapArray ){
621      CUBELOC tempLoc = new CUBELOC();
622      ASTARNODE tempCube = new ASTARNODE();
623      // Create a ASTARNODE list for the frontier
624      List<ASTARNODE> frontier = new List<ASTARNODE>();
625
626      float maxDistance = Mathf.Sqrt( Mathf.Pow (mapLength, 2) + Mathf.Pow (mapWidth, 2) );
627
628      // Set initial node equal to start cube of map
629      tempCube.Set( startX, startZ, PathDistance( startX, startZ ), true );
630      frontier.Add(tempCube);
631      MAPCUBE tempMapCube;
632      CUBELOC[] tempMapLinks = new CUBELOC[4];
633      CUBELOC[] rotMapLinks = new CUBELOC[4];
634      int tempDir = 0;
635      int listIdx = 0;
636      int loopCount = 0;
637
638      while( loopCount < (mapLength * mapWidth) ){
639          loopCount++;
640          float tempDistance = maxDistance;
641          int tempClosestNode = -1;
642          listIdx = 0;
643
644          // Iterate through the list of nodes on the frontier
645          while( listIdx < frontier.Count ){
646              if( frontier[listIdx].GetOpen() ){
647                  // Find the node on the frontier that is closest to the goal
648                  if( frontier[listIdx].GetDist() < tempDistance ){
649                      tempDistance = frontier[listIdx].GetDist();
650                      tempClosestNode = listIdx;
651                  }
652              }
653              listIdx++;
654          }
655          // If entire frontier is closed there is no path to the exit
656
657          if( tempClosestNode == -1 ){
658              return false;
659          }
660
661          // Remove closest node from open list
662          frontier[tempClosestNode].Close();
663
664          // Store X and Z values in temporary CUBELOC variable to simplify next steps
665          tempLoc.x = frontier[tempClosestNode].GetX();
666          tempLoc.z = frontier[tempClosestNode].GetZ();
667
668          tempMapCube = MapArray[tempLoc.x,tempLoc.z];
669          tempDir = (int)tempMapCube.yAngle / 2; // convirts DIR to int and gets range 0-3 instead
of 0-6
670
671          // Reads links data from map cube and adds linked cubes to an array
672          if( tempMapCube.dir0 ){
673              tempMapLinks[((4-tempDir)%4)].x = tempLoc.x+1;//tempMapLinks[((0+tempDir)%4)].x =
tempLoc.x-1;
674              tempMapLinks[((4-tempDir)%4)].z = tempLoc.z;//tempMapLinks[((0+tempDir)%4)].z =
tempLoc.z;
675
676              if( (tempMapLinks[((4-tempDir)%4)].x == endX) && (tempMapLinks[((4-tempDir)%4)].z ==
endZ) ){

```

```

677         return true;
678     }
679 }
680 else{
681     tempMapLinks[((4-tempDir)%4)].x = -1;//tempMapLinks[((0+tempDir)%4)].x = -1;
682     tempMapLinks[((4-tempDir)%4)].z = -1;//tempMapLinks[((0+tempDir)%4)].z = -1;
683 }
684
685     if( tempMapCube.dir2 ){
686         tempMapLinks[((5-tempDir)%4)].x = tempLoc.x;//tempMapLinks[((1+tempDir)%4)].x = ✓
tempLoc.x;
687         tempMapLinks[((5-tempDir)%4)].z = tempLoc.z+1;//tempMapLinks[((1+tempDir)%4)].z = ✓
tempLoc.z+1;
688
689         if( (tempMapLinks[((5-tempDir)%4)].x == endX) && (tempMapLinks[((5-tempDir)%4)].z == ✓
endZ) ){
690             return true;
691         }
692     }
693     else{
694         tempMapLinks[((5-tempDir)%4)].x = -1;//tempMapLinks[((1+tempDir)%4)].x = -1;
695         tempMapLinks[((5-tempDir)%4)].z = -1;//tempMapLinks[((1+tempDir)%4)].z = -1;
696     }
697
698     if( tempMapCube.dir4 ){
699         tempMapLinks[((6-tempDir)%4)].x = tempLoc.x-1;//tempMapLinks[((2+tempDir)%4)].x = ✓
tempLoc.x+1;
700         tempMapLinks[((6-tempDir)%4)].z = tempLoc.z;//tempMapLinks[((2+tempDir)%4)].z = ✓
tempLoc.z;
701
702         if( (tempMapLinks[((6-tempDir)%4)].x == endX) && (tempMapLinks[((6-tempDir)%4)].z == ✓
endZ) ){
703             return true;
704         }
705     }
706     else{
707         tempMapLinks[((6-tempDir)%4)].x = -1;//tempMapLinks[((2+tempDir)%4)].x = -1;
708         tempMapLinks[((6-tempDir)%4)].z = -1;//tempMapLinks[((2+tempDir)%4)].z = -1;
709     }
710
711     if( tempMapCube.dir6 ){
712         tempMapLinks[((7-tempDir)%4)].x = tempLoc.x;//tempMapLinks[((3+tempDir)%4)].x = ✓
tempLoc.x;
713         tempMapLinks[((7-tempDir)%4)].z = tempLoc.z-1;//tempMapLinks[((3+tempDir)%4)].z = ✓
tempLoc.z-1;
714
715         if( (tempMapLinks[((7-tempDir)%4)].x == endX) && (tempMapLinks[((7-tempDir)%4)].z == ✓
endZ) ){
716             return true;
717         }
718     }
719     else{
720         tempMapLinks[((7-tempDir)%4)].x = -1;//tempMapLinks[((3+tempDir)%4)].x = -1;
721         tempMapLinks[((7-tempDir)%4)].z = -1;//tempMapLinks[((3+tempDir)%4)].z = -1;
722     }
723
724     // Checks list
725     for( int mapIdx = 0; mapIdx < 4; mapIdx++ ){
726         if( tempMapLinks[mapIdx].x != -1 ){ // would be -1 if there was no link on that side
727             listIdx = 0;
728             bool duplicateNode = false;
729             while( listIdx < frontier.Count && !duplicateNode ){
730                 // Checks if there is a node on the list corresponding to the new location
731                 if( (frontier[listIdx].GetX() == tempMapLinks[mapIdx].x) &&
732                     (frontier[listIdx].GetZ() == tempMapLinks[mapIdx].z) ){
733                     duplicateNode = true;

```

```

734         }
735         listIdx++;
736     }
737     if( !duplicateNode ){
738         ASTARNODE nodeToAdd = new ASTARNODE();
739         nodeToAdd.Set( tempMapLinks[mapIdx].x, tempMapLinks[mapIdx].z,
740             PathDistance( tempMapLinks[mapIdx].x, tempMapLinks[mapIdx].z ),
741         true );
742         frontier.Add( nodeToAdd );
743     }
744 }
745 }
746 //////////////////////////////////////////////////// ✓
747 ////////////////////////////////////////////////////
748 using (System.IO.StreamWriter start = new System.IO.StreamWriter( @"C:\temp\OutputLog.txt",
749     true ))
750 { start.WriteLine( " ** Returning \"FALSE\" **" ); }
751 //////////////////////////////////////////////////// ✓
752 ////////////////////////////////////////////////////
753 return false;
754 }
755
756 /*
757 -----
758 Calculate Distance Between Start and End
759 -----
760 */
761 float PathDistance( int x, int z ){
762     float dStart = Mathf.Sqrt( Mathf.Pow( (startX - x), 2 ) + (Mathf.Pow( (startZ - z), 2 )) );
763     float dEnd = Mathf.Sqrt( Mathf.Pow( (endX - x), 2 ) + (Mathf.Pow( (endZ - z), 2 )) );
764     float dTotal = dStart + dEnd;
765     return dTotal;
766 }
767
768 /*
769 -----
770 Place Map Cubes
771 -----
772 */
773
774 void InstantiateMap( ref MAPCUBE[,] MapArray ){
775     Vector3 spawnPos = new Vector3( 0.0f, 0.0f, 0.0f );
776     Vector3 spawnAngle = new Vector3( 0.0f, 0.0f, 0.0f );
777     int cubeLevel = 0;
778     int props = 0;
779     float yMod = 0.0f;
780     int newCube = 0;
781     int newAngle = 0;
782
783     for( int x = 1; x < (mapWidth+1); x++ ){
784         for( int z = 1; z < (mapLength+1); z++ ){
785             if( MapArray[x,z].cube != -1 ){
786                 if( x == startX && z == startZ ){
787                     // Creates map cube containing player start point
788                     newAngle = 90;
789                     spawnPos.Set( ((x * 5.12f) - 40.96f), 1.8f, (z * 5.12f) );
790                     Instantiate( p14E_EndHall, spawnPos, Quaternion.Euler(-90,newAngle,0) );
791                     spawnPos.Set( ((x * 5.12f) - 40.96f), 1.82f, (z * 5.12f) );
792                     Instantiate( pSpiralStair, spawnPos, Quaternion.Euler (-90, (newAngle+180),
793 0));
794                     spawnPos.Set( ((x * 5.12f) - 40.96f), 1.8f, (z * 5.12f + 0.64f) );

```

```

795         Instantiate( Lantern, spawnPos, Quaternion.identity );
796         Instantiate( SteadyLight, spawnPos, Quaternion.identity );
797     }
798     else if( x == endX && z == endZ ){
799         // Creates map cube containing map end point
800         newAngle = 270;
801         spawnPos.Set( ((x * 5.12f) - 40.96f), 1.8f, ((z * 5.12f)) );
802         Instantiate( p14X_EndHall, spawnPos, Quaternion.Euler(-90,newAngle,0) );
803         Instantiate( pEnder, spawnPos, Quaternion.identity);
804         spawnPos.Set( ((x * 5.12f) - 40.96f), 7.5f, ((z * 5.12f)) );
805         Instantiate( pSpiralStair, spawnPos, Quaternion.Euler (-90, (newAngle+270),
0));
806
807         spawnPos.Set( ((x * 5.12f) - 40.96f), 5.1f, ((z * 5.12f)) );
808         Instantiate( pSpiralStair, spawnPos, Quaternion.Euler (-90, (newAngle+270),
0));
809
810         spawnPos.Set( ((x * 5.12f) - 40.96f), 1.8f, ((z * 5.12f - 1.28f)) );
811         Instantiate( Lantern, spawnPos, Quaternion.identity );
812         Instantiate( SteadyLight, spawnPos, Quaternion.identity );
813     }
814     else{
815         newCube = MapArray[x,z].cube;
816         newAngle = (int)MapArray[x,z].yAngle * 45;
817
818         // 3D cube position
819         Vector3 cubePosVect = new Vector3( ((x * 5.12f) - 40.96f), 1.8f, ((z * 5.12f))
);
820
821         Vector3 cubeAngle = new Vector3( -90, ((newAngle * -1) + 180), 0 );
822
823         if( (newCube == 1) ||
824             (newCube == 2) ||
825             (newCube == 3) ||
826             (newCube == 4) ||
827             (newCube == 14) ) yMod = 1.75f;
828         else yMod = 2.5f;
829
830         spawnPos = new Vector3( ((x * 5.12f) - 40.96f), yMod, ((z * 5.12f)) );
831         spawnAngle = new Vector3( 0, 180, 0 );
832
833         cubeLevel= UnityEngine.Random.Range(-1,2);
834
835         Instantiate( Lantern, spawnPos, Quaternion.Euler( spawnAngle ));
836         Instantiate( LanternCore, spawnPos, Quaternion.Euler( spawnAngle ));
837
838         if( x == startX && z == (startZ+1) ) cubeLevel = 0;
839         if( x == endX && z == (endZ-1) ) cubeLevel = 1;
840
841         switch( cubeLevel ){
842             case -1 :
843                 cubePosVect.Set( ((x * 5.12f) - 40.96f), 2.55f, ((z * 5.12f)) );
844                 Instantiate( pWaterPlane, cubePosVect, Quaternion.Euler( cubeAngle ));
845                 switch( newCube ){
846                     case 1 :
847                         cubePosVect.Set( ((x * 5.12f) - 40.96f), 1.8f, ((z * 5.12f)) );
848                         Instantiate( p01W_StraightHall, cubePosVect, Quaternion.Euler(
cubeAngle ));
849                         break;
850
851                     case 2 :
852                         cubePosVect.Set( ((x * 5.12f) - 40.96f), 1.8f, ((z * 5.12f)) );
853                         Instantiate( p02W_CornerHall, cubePosVect, Quaternion.Euler( cubeAngle
));
854                         break;
855
856                     case 3 :
857                         cubePosVect.Set( ((x * 5.12f) - 40.96f), 1.8f, ((z * 5.12f)) );

```

```

856         Instantiate( p03W_CrossHall, cubePosVect, Quaternion.Euler( cubeAngle
857     ));
858         break;
859     case 4 :
860         cubePosVect.Set( ((x * 5.12f) - 40.96f), 1.8f, ((z * 5.12f)) );
861         Instantiate( p04W_TeeHall, cubePosVect, Quaternion.Euler( cubeAngle ))
862     ;
863         break;
864     case 5 :
865         cubePosVect.Set( ((x * 5.12f) - 40.96f), 2.55f, ((z * 5.12f)) );
866         Instantiate( p05W_HallRoomR, cubePosVect, Quaternion.Euler( cubeAngle
867     ));
868         props = UnityEngine.Random.Range(1,4);
869         switch( props ){
870     case 1 :
871         cubePosVect.Set( cubePosVect.x, (cubePosVect.y-0.64f), cubePosVect
872     .z );
873         Instantiate( pSewerPipeSide, cubePosVect, Quaternion.Euler(
874     cubeAngle ));
875         break;
876     case 2 :
877         cubeAngle.Set( cubeAngle.x, cubeAngle.y, (cubeAngle.z-90.0f));
878         cubePosVect.Set( cubePosVect.x, (cubePosVect.y+0.32f), cubePosVect
879     .z );
880         Instantiate( pHangingCageSide, cubePosVect, Quaternion.Euler
881     (cubeAngle));
882         break;
883     default :
884         break;
885     }
886     break;
887     case 6 :
888         cubePosVect.Set( ((x * 5.12f) - 40.96f), 2.55f, ((z * 5.12f)) );
889         //-----
890         // **WORKAROUND**
891         Vector3 fixAngle = new Vector3( cubeAngle.x, (cubeAngle.y - 90),
892     cubeAngle.z);
893         // **End of workaround**
894         //-----
895         Instantiate( p06W_HallRoomL, cubePosVect, Quaternion.Euler( fixAngle
896     ));
897         props = UnityEngine.Random.Range(1,4);
898         switch( props ){
899     case 1 :
900         cubePosVect.Set( cubePosVect.x, (cubePosVect.y-0.64f), cubePosVect
901     .z );
902         cubeAngle.Set( cubeAngle.x, (cubeAngle.y+180), cubeAngle.z);
903         Instantiate( pSewerPipeSide, cubePosVect, Quaternion.Euler
904     (cubeAngle));
905         break;
906     case 2 :
907         cubeAngle.Set( cubeAngle.x, cubeAngle.y, (cubeAngle.z+90.0f));
908         cubePosVect.Set( cubePosVect.x, (cubePosVect.y+0.32f), cubePosVect
909     .z );
910         Instantiate( pHangingCageSide, cubePosVect, Quaternion.Euler
911     (cubeAngle));
912         break;
913     default :
914         break;
915     }
916     break;

```

```

909         }
910         break;
911
912     case 7 :
913         cubePosVect.Set( ((x * 5.12f) - 40.96f), 2.55f, ((z * 5.12f)) );
914         Instantiate( p07W_HallRoomLR, cubePosVect, Quaternion.Euler( cubeAngle
915     ));
916
917     props = UnityEngine.Random.Range(1,4);
918     switch( props ){
919     case 1 :
920     case 2 :
921         cubeAngle.Set( cubeAngle.x, cubeAngle.y, (cubeAngle.z+180.0f));
922         Instantiate( pCoffinCageCorner, cubePosVect, Quaternion.Euler
923     (cubeAngle));
924         break;
925
926     default :
927         break;
928     }
929     break;
930
931     case 8 :
932         cubePosVect.Set( ((x * 5.12f) - 40.96f), 2.55f, ((z * 5.12f)) );
933         Instantiate( p08W_HallRoom, cubePosVect, Quaternion.Euler( cubeAngle
934     ));
935
936     props = UnityEngine.Random.Range(1,4);
937     switch( props ){
938     case 1 :
939     case 2 :
940         cubePosVect.Set( cubePosVect.x, (cubePosVect.y+0.32f), cubePosVect
941     .z );
942         Instantiate( pHangingCageSide, cubePosVect, Quaternion.Euler(
943     cubeAngle ));
944         break;
945
946     default :
947         break;
948     }
949     break;
950
951     case 9 :
952         cubePosVect.Set( ((x * 5.12f) - 40.96f), 2.55f, ((z * 5.12f)) );
953         Instantiate( p09W_RoomAngle, cubePosVect, Quaternion.Euler( cubeAngle
954     ));
955
956     props = UnityEngine.Random.Range(1,4);
957     switch( props ){
958     case 1 :
959     case 2 :
960         cubeAngle.Set( cubeAngle.x, cubeAngle.y, (cubeAngle.z+180.0f));
961         Instantiate( pCoffinCageCorner, cubePosVect, Quaternion.Euler
962     (cubeAngle));
963         break;
964
965     default :
966         break;
967     }
968     break;
969
970     case 10 :
971         cubePosVect.Set( ((x * 5.12f) - 40.96f), 2.55f, ((z * 5.12f)) );
972         Instantiate( p10W_SideRoom, cubePosVect, Quaternion.Euler( cubeAngle
973     ));
974
975     props = UnityEngine.Random.Range(1,6);
976     switch( props ){
977     case 1 :
978         Instantiate( pCoffinCageSide, cubePosVect, Quaternion.Euler(

```



```

967         cubeAngle ));
968         break;
969     case 2 :
970         cubePosVect.Set( cubePosVect.x, (cubePosVect.y+0.32f), cubePosVect.z
971         .z );
972         Instantiate( pHangingCageSide, cubePosVect, Quaternion.Euler( cubeAngle ));
973         break;
974     case 3 :
975         cubePosVect.Set( cubePosVect.x, (cubePosVect.y-0.64f), cubePosVect.z
976         .z );
977         Instantiate( pSewerPipeSide, cubePosVect, Quaternion.Euler( cubeAngle ));
978         break;
979     case 4 :
980         cubePosVect.Set( cubePosVect.x, (cubePosVect.y-0.64f), cubePosVect.z
981         .z );
982         Instantiate( pSewerPipeSide, cubePosVect, Quaternion.Euler( cubeAngle ));
983         cubePosVect.Set( cubePosVect.x, (cubePosVect.y+0.96f), cubePosVect.z
984         .z );
985         Instantiate( pHangingCageSide, cubePosVect, Quaternion.Euler( cubeAngle ));
986         break;
987     }
988     break;
989 case 11 :
990     cubePosVect.Set( ((x * 5.12f) - 40.96f), 2.55f, ((z * 5.12f)) );
991     Instantiate( p11W_CornerRoom, cubePosVect, Quaternion.Euler( cubeAngle ));
992     props = UnityEngine.Random.Range(1,6);
993     switch( props ){
994     case 1 :
995         cubeAngle.Set( cubeAngle.x, cubeAngle.y, (cubeAngle.z+180.0f));
996         Instantiate( pCoffinCageCorner, cubePosVect, Quaternion.Euler( cubeAngle ));
997         break;
998     case 2 :
999     case 3 :
1000         cubePosVect.Set( cubePosVect.x, (cubePosVect.y-0.64f), cubePosVect.z
1001         .z );
1002         Instantiate( pSewerPipeSide, cubePosVect, Quaternion.Euler( cubeAngle ));
1003         break;
1004     case 4 :
1005         cubePosVect.Set( cubePosVect.x, (cubePosVect.y-0.64f), cubePosVect.z
1006         .z );
1007         cubeAngle.Set( cubeAngle.x, (cubeAngle.y-90.0f), cubeAngle.z);
1008         Instantiate( pSewerPipeSide, cubePosVect, Quaternion.Euler( cubeAngle ));
1009         break;
1010     }
1011     break;
1012 case 12 :
1013     cubePosVect.Set( ((x * 5.12f) - 40.96f), 2.55f, ((z * 5.12f)) );
1014     Instantiate( p12W_OffsetRoom, cubePosVect, Quaternion.Euler( cubeAngle ));
1015     break;
1016 }

```

```

1017         case 13 :
1018             cubePosVect.Set( ((x * 5.12f) - 40.96f), 2.55f, ((z * 5.12f)) );
1019             Instantiate( p13W_OpenRoom, cubePosVect, Quaternion.Euler( cubeAngle
1020     ));
1021             props = UnityEngine.Random.Range(1,6);
1022             switch( props ){
1023                 case 1 :
1024                     case 2 :
1025                         Instantiate( pCoffinCageSide, cubePosVect, Quaternion.Euler(
1026     cubeAngle ));
1027                         break;
1028                     case 3 :
1029                         //cubePosVect.Set( cubePosVect.x, (cubePosVect.y+0.32f),
1030     cubePosVect.z );
1031                         //Instantiate( pHangingCageQuad, cubePosVect, Quaternion.Euler(
1032     cubeAngle ));
1033                         //break;
1034                     case 4 :
1035                         cubePosVect.Set( cubePosVect.x, (cubePosVect.y+0.32f), cubePosVect
1036     .z );
1037                         Instantiate( pHangingCageSide, cubePosVect, Quaternion.Euler(
1038     cubeAngle ));
1039                         break;
1040                     default :
1041                         break;
1042                 }
1043             break;
1044         case 14 :
1045             cubePosVect.Set( ((x * 5.12f) - 40.96f), 1.8f, ((z * 5.12f)) );
1046             Instantiate( p14W_EndHall, cubePosVect, Quaternion.Euler( cubeAngle ))
1047     ;
1048             props = UnityEngine.Random.Range(1,4);
1049             switch( props ){
1050                 case 1 :
1051                     Instantiate( pSewerPipeSide, cubePosVect, Quaternion.Euler(
1052     cubeAngle ));
1053                     break;
1054                 case 2 :
1055                     cubePosVect.Set( cubePosVect.x, (cubePosVect.y+0.32f), cubePosVect
1056     .z );
1057                     Instantiate( pHangingCageSide, cubePosVect, Quaternion.Euler(
1058     cubeAngle ));
1059                     break;
1060                 }
1061             break;
1062         case 15 :
1063             cubePosVect.Set( ((x * 5.12f) - 40.96f), 2.4f, ((z * 5.12f)) );
1064             Instantiate( p17_SolidCube, cubePosVect, Quaternion.Euler( cubeAngle
1065     ));
1066             break;
1067         default : break;
1068     }
1069     break;
1070 case 0 :
1071     switch( newCube ){
1072         case 1 :
1073             cubePosVect.Set( ((x * 5.12f) - 40.96f), 1.8f, ((z * 5.12f)) );
1074             Instantiate( p01L_StraightHall, cubePosVect, Quaternion.Euler(

```

```

cubeAngle ));
1072             break;
1073
1074         case 2 :
1075             cubePosVect.Set( ((x * 5.12f) - 40.96f), 1.8f, ((z * 5.12f)) );
1076             Instantiate( p02L_CornerHall, cubePosVect, Quaternion.Euler( cubeAngle ));
1077         ));
1078             break;
1079
1080         case 3 :
1081             cubePosVect.Set( ((x * 5.12f) - 40.96f), 1.8f, ((z * 5.12f)) );
1082             Instantiate( p03L_CrossHall, cubePosVect, Quaternion.Euler( cubeAngle ));
1083         ));
1084             break;
1085
1086         case 4 :
1087             cubePosVect.Set( ((x * 5.12f) - 40.96f), 1.8f, ((z * 5.12f)) );
1088             Instantiate( p04L_TeeHall, cubePosVect, Quaternion.Euler( cubeAngle ));
1089         ));
1090             break;
1091
1092         case 5 :
1093             cubePosVect.Set( ((x * 5.12f) - 40.96f), 2.55f, ((z * 5.12f)) );
1094             Instantiate( p05L_HallRoomR, cubePosVect, Quaternion.Euler( cubeAngle ));
1095         ));
1096             props = UnityEngine.Random.Range(1,6);
1097             switch( props ){
1098                 case 1 :
1099                     cubeAngle.Set( (cubeAngle.x+90.0f), cubeAngle.y, cubeAngle.z );
1100                     Instantiate( pWristShacklesSide, cubePosVect, Quaternion.Euler( cubeAngle ));
1101                     break;
1102                 case 2 :
1103                     cubeAngle.Set( (cubeAngle.x+90.0f), cubeAngle.y, cubeAngle.z );
1104                     Instantiate( pSpreadShacklesSide, cubePosVect, Quaternion.Euler( cubeAngle ));
1105                     break;
1106                 case 3 :
1107                     cubeAngle.Set( cubeAngle.x, cubeAngle.y, (cubeAngle.z-90));
1108                     cubePosVect.Set( cubePosVect.x, (cubePosVect.y+0.32f), cubePosVect.z );
1109                     Instantiate( pHangingCageSide, cubePosVect, Quaternion.Euler( cubeAngle ));
1110                     break;
1111                 case 4 :
1112                     cubeAngle.Set( cubeAngle.x, cubeAngle.y, (cubeAngle.z-90.0f));
1113                     Instantiate( pPillorySide, cubePosVect, Quaternion.Euler( cubeAngle ));
1114                     break;
1115                 default :
1116                     break;
1117             }
1118             break;
1119
1120         case 6 :
1121             cubePosVect.Set( ((x * 5.12f) - 40.96f), 2.55f, ((z * 5.12f)) );
1122             //-----
1123             // **WORKAROUND**
1124             Vector3 fixAngle = new Vector3( cubeAngle.x, (cubeAngle.y - 90), cubeAngle.z);
1125             // **End of workaround**
1126             //-----

```

```

1127         Instantiate( p06L_HallRoomL, cubePosVect, Quaternion.Euler( fixAngle
1128     ));
1129     props = UnityEngine.Random.Range(1,6);
1130     switch( props ){
1131     case 1 :
1132         cubeAngle.Set( (cubeAngle.x+90.0f), cubeAngle.y, (cubeAngle.z+180.
1133     0f));
1134         Instantiate( pWristShacklesSide, cubePosVect, Quaternion.Euler
1135     (cubeAngle));
1136         break;
1137     case 2 :
1138         cubeAngle.Set( (cubeAngle.x+90.0f), cubeAngle.y, (cubeAngle.z+180.
1139     0f));
1140         Instantiate( pSpreadShacklesSide, cubePosVect, Quaternion.Euler
1141     (cubeAngle));
1142         break;
1143     case 3 :
1144         cubeAngle.Set( cubeAngle.x, cubeAngle.y, (cubeAngle.z+90.0f));
1145         cubePosVect.Set( cubePosVect.x, (cubePosVect.y+0.32f), cubePosVect
1146     .z );
1147         Instantiate( pHangingCageSide, cubePosVect, Quaternion.Euler
1148     (cubeAngle));
1149         break;
1150     case 4 :
1151         cubeAngle.Set( cubeAngle.x, cubeAngle.y, (cubeAngle.z+90.0f));
1152         Instantiate( pPillorySide, cubePosVect, Quaternion.Euler
1153     (cubeAngle));
1154         break;
1155     default :
1156         break;
1157     }
1158     break;
1159     case 7 :
1160         cubePosVect.Set( ((x * 5.12f) - 40.96f), 2.55f, ((z * 5.12f)) );
1161         Instantiate( p07L_HallRoomLR, cubePosVect, Quaternion.Euler( cubeAngle
1162     ));
1163         break;
1164     case 8 :
1165         cubePosVect.Set( ((x * 5.12f) - 40.96f), 2.55f, ((z * 5.12f)) );
1166         Instantiate( p08L_HallRoom, cubePosVect, Quaternion.Euler( cubeAngle
1167     ));
1168         props = UnityEngine.Random.Range(1,4);
1169         switch( props ){
1170         case 1 :
1171         case 2 :
1172             cubePosVect.Set( cubePosVect.x, (cubePosVect.y+0.32f), cubePosVect
1173     .z );
1174             Instantiate( pHangingCageSide, cubePosVect, Quaternion.Euler(
1175     cubeAngle ));
1176             break;
1177         default :
1178             break;
1179         }
1180         break;
1181     case 9 :
1182         cubePosVect.Set( ((x * 5.12f) - 40.96f), 2.55f, ((z * 5.12f)) );
1183         Instantiate( p09L_RoomAngle, cubePosVect, Quaternion.Euler( cubeAngle
1184     ));

```

```

1180         props = UnityEngine.Random.Range(1,6);
1181         switch( props ){
1182         case 1 :
1183         case 2 :
1184             cubeAngle.Set( (cubeAngle.x+90.0f), (cubeAngle.y+90.0f), cubeAngle
1185             .z);
1186             Instantiate( pJudasCradleCorner, cubePosVect, Quaternion.Euler
1187             (cubeAngle));
1188             break;
1189         case 3 :
1190             cubeAngle.Set( (cubeAngle.x+90.0f), cubeAngle.y, cubeAngle.z );
1191             Instantiate( pTortureRackCorner, cubePosVect, Quaternion.Euler(
1192             cubeAngle ));
1193             break;
1194         case 4 :
1195             cubeAngle.Set( cubeAngle.x, cubeAngle.y, (cubeAngle.z+90.0f));
1196             Instantiate( pPilloryCorner, cubePosVect, Quaternion.Euler
1197             (cubeAngle));
1198             break;
1199         default :
1200             break;
1201         }
1202         break;
1203     case 10 :
1204         cubePosVect.Set( ((x * 5.12f) - 40.96f), 2.55f, ((z * 5.12f)) );
1205         Instantiate( p10L_SideRoom, cubePosVect, Quaternion.Euler( cubeAngle
1206         ));
1207         props = UnityEngine.Random.Range(1,8);
1208         switch( props ){
1209         case 1 :
1210             cubeAngle.Set( (cubeAngle.x+90.0f), cubeAngle.y, cubeAngle.z );
1211             Instantiate( pWristShacklesSide, cubePosVect, Quaternion.Euler(
1212             cubeAngle ));
1213             break;
1214         case 2 :
1215             cubeAngle.Set( (cubeAngle.x+90.0f), cubeAngle.y, cubeAngle.z );
1216             Instantiate( pSpreadShacklesSide, cubePosVect, Quaternion.Euler(
1217             cubeAngle ));
1218             break;
1219         case 3 :
1220             cubePosVect.Set( cubePosVect.x, (cubePosVect.y+0.32f), cubePosVect
1221             .z );
1222             Instantiate( pHangingCageSide, cubePosVect, Quaternion.Euler(
1223             cubeAngle ));
1224             break;
1225         case 4 :
1226             cubeAngle.Set( (cubeAngle.x+90.0f), cubeAngle.y, cubeAngle.z );
1227             Instantiate( pTortureRackSide, cubePosVect, Quaternion.Euler(
1228             cubeAngle ));
1229             break;
1230         case 5 :
1231             cubeAngle.Set( (cubeAngle.x+90.0f), cubeAngle.y, cubeAngle.z );
1232             Instantiate( pJudasCradleSide, cubePosVect, Quaternion.Euler(
1233             cubeAngle ));
1234             break;
1235         case 6 :
1236             Instantiate( pPillorySide, cubePosVect, Quaternion.Euler(

```

```

cubeAngle ));
1235         break;
1236     }
1237     break;
1238
1239     case 11 :
1240         cubePosVect.Set( ((x * 5.12f) - 40.96f), 2.55f, ((z * 5.12f)) );
1241         Instantiate( p11L_CornerRoom, cubePosVect, Quaternion.Euler( cubeAngle ));
1242
1243         props = UnityEngine.Random.Range(1,8);
1244         switch( props ){
1245             case 1 :
1246                 cubeAngle.Set( (cubeAngle.x+90.0f), cubeAngle.y, cubeAngle.z );
1247                 Instantiate( pWristShacklesCorner, cubePosVect, Quaternion.Euler(
1248                     cubeAngle ));
1249                 break;
1250
1251             case 2 :
1252                 cubeAngle.Set( (cubeAngle.x+90.0f), cubeAngle.y, cubeAngle.z );
1253                 Instantiate( pSpreadShacklesCorner, cubePosVect, Quaternion.Euler(
1254                     cubeAngle ));
1255                 break;
1256
1257             case 3 :
1258             case 4 :
1259                 cubeAngle.Set( (cubeAngle.x+90.0f), cubeAngle.y, cubeAngle.z );
1260                 Instantiate( pTortureRackCorner, cubePosVect, Quaternion.Euler(
1261                     cubeAngle ));
1262                 break;
1263
1264             case 5 :
1265                 cubeAngle.Set( (cubeAngle.x+90.0f), (cubeAngle.y+90.0f), cubeAngle
1266                     .z );
1267                 Instantiate( pJudasCradleCorner, cubePosVect, Quaternion.Euler(
1268                     (cubeAngle)));
1269                 break;
1270
1271             case 6 :
1272                 cubeAngle.Set( cubeAngle.x, cubeAngle.y, (cubeAngle.z+90.0f));
1273                 Instantiate( pPilloryCorner, cubePosVect, Quaternion.Euler(
1274                     (cubeAngle)));
1275                 break;
1276             }
1277         }
1278         break;
1279
1280     case 12 :
1281         cubePosVect.Set( ((x * 5.12f) - 40.96f), 2.55f, ((z * 5.12f)) );
1282         Instantiate( p12L_OffsetRoom, cubePosVect, Quaternion.Euler( cubeAngle ));
1283
1284         break;
1285
1286     case 13 :
1287         cubePosVect.Set( ((x * 5.12f) - 40.96f), 2.55f, ((z * 5.12f)) );
1288         Instantiate( p13L_OpenRoom, cubePosVect, Quaternion.Euler( cubeAngle ));
1289
1290         props = UnityEngine.Random.Range(1,8);
1291         switch( props ){
1292             case 1 :
1293                 cubeAngle.Set( (cubeAngle.x+90.0f), cubeAngle.y, cubeAngle.z );
1294                 Instantiate( pWristShacklesQuad, cubePosVect, Quaternion.Euler(
1295                     cubeAngle ));
1296                 break;
1297
1298             case 2 :
1299             case 3 :
1300                 Instantiate( pPillorySide, cubePosVect, Quaternion.Euler(

```

```

cubeAngle ));
1290         break;
1291
1292     case 4 :
1293         cubeAngle.Set( (cubeAngle.x+90.0f), cubeAngle.y, cubeAngle.z );
1294         Instantiate( pTortureRackSide, cubePosVect, Quaternion.Euler(
cubeAngle ));
1295         break;
1296
1297     case 5 :
1298         cubeAngle.Set( (cubeAngle.x+90.0f), cubeAngle.y, cubeAngle.z );
1299         Instantiate( pJudasCradleSide, cubePosVect, Quaternion.Euler(
cubeAngle ));
1300         break;
1301
1302     case 6 :
1303         cubePosVect.Set( cubePosVect.x, (cubePosVect.y+0.32f), cubePosVect
.z );
1304         Instantiate( pHangingCageSide, cubePosVect, Quaternion.Euler(
cubeAngle ));
1305         break;
1306
1307     default :
1308         break;
1309     }
1310     break;
1311
1312     case 14 :
1313         cubePosVect.Set( ((x * 5.12f) - 40.96f), 1.8f, ((z * 5.12f)) );
1314         Instantiate( p14L_EndHall, cubePosVect, Quaternion.Euler( cubeAngle ))
;
1315         cubePosVect.Set( ((x * 5.12f) - 40.96f), -.64f, ((z * 5.12f)) );
1316         Instantiate( pSquareGrate, cubePosVect, Quaternion.Euler( cubeAngle ))
;
1317         break;
1318
1319     case 15 :
1320         cubePosVect.Set( ((x * 5.12f) - 40.96f), 2.4f, ((z * 5.12f)) );
1321         Instantiate( p17_SolidCube, cubePosVect, Quaternion.Euler( cubeAngle
));
1322         break;
1323
1324     default : break;
1325     }
1326     break;
1327
1328     case 1 :
1329         switch( newCube ){
1330         case 1 :
1331             cubePosVect.Set( ((x * 5.12f) - 40.96f), 1.8f, ((z * 5.12f)) );
1332             Instantiate( p01H_StraightHall, cubePosVect, Quaternion.Euler(
cubeAngle ));
1333             break;
1334
1335         case 2 :
1336             cubePosVect.Set( ((x * 5.12f) - 40.96f), 1.8f, ((z * 5.12f)) );
1337             Instantiate( p02H_CornerHall, cubePosVect, Quaternion.Euler( cubeAngle
));
1338             break;
1339
1340         case 3 :
1341             cubePosVect.Set( ((x * 5.12f) - 40.96f), 1.8f, ((z * 5.12f)) );
1342             Instantiate( p03H_CrossHall, cubePosVect, Quaternion.Euler( cubeAngle
));
1343             break;
1344

```

```

1345         case 4 :
1346             cubePosVect.Set( ((x * 5.12f) - 40.96f), 1.8f, ((z * 5.12f)) );
1347             Instantiate( p04H_TeeHall, cubePosVect, Quaternion.Euler( cubeAngle )) ✓
1348         ;
1349             break;
1350         case 5 :
1351             cubePosVect.Set( ((x * 5.12f) - 40.96f), 2.55f, ((z * 5.12f)) );
1352             Instantiate( p05L_HallRoomR, cubePosVect, Quaternion.Euler( cubeAngle ) ✓
1353         ));
1354             props = UnityEngine.Random.Range(1,6);
1355             switch( props ){
1356                 case 1 :
1357                     cubeAngle.Set( (cubeAngle.x+90.0f), cubeAngle.y, cubeAngle.z );
1358                     Instantiate( pWristShacklesSide, cubePosVect, Quaternion.Euler( ✓
1359             cubeAngle ));
1360                     break;
1361                 case 2 :
1362                     cubeAngle.Set( (cubeAngle.x+90.0f), cubeAngle.y, cubeAngle.z );
1363                     Instantiate( pSpreadShacklesSide, cubePosVect, Quaternion.Euler( ✓
1364             cubeAngle ));
1365                     break;
1366                 case 3 :
1367                     cubeAngle.Set( cubeAngle.x, cubeAngle.y, (cubeAngle.z-90.0f));
1368                     cubePosVect.Set( cubePosVect.x, (cubePosVect.y+0.32f), cubePosVect ✓
1369             .z );
1370                     Instantiate( pHangingCageSide, cubePosVect, Quaternion.Euler( ✓
1371             (cubeAngle));
1372                     break;
1373                 case 4 :
1374                     cubeAngle.Set( cubeAngle.x, cubeAngle.y, (cubeAngle.z-90.0f));
1375                     Instantiate( pPillorySide, cubePosVect, Quaternion.Euler( ✓
1376             (cubeAngle));
1377                     break;
1378             }
1379             break;
1380         case 6 :
1381             cubePosVect.Set( ((x * 5.12f) - 40.96f), 2.55f, ((z * 5.12f)) );
1382             //-----
1383             // **WORKAROUND**
1384             Vector3 fixAngle = new Vector3( cubeAngle.x, (cubeAngle.y - 90), ✓
1385             cubeAngle.z);
1386             // **End of workaround**
1387             //-----
1388             Instantiate( p06L_HallRoomL, cubePosVect, Quaternion.Euler( fixAngle ✓
1389         ));
1390             props = UnityEngine.Random.Range(1,6);
1391             switch( props ){
1392                 case 1 :
1393                     cubeAngle.Set( (cubeAngle.x+90.0f), cubeAngle.y, (cubeAngle.z+180. ✓
1394             0f));
1395                     Instantiate( pWristShacklesSide, cubePosVect, Quaternion.Euler( ✓
1396             (cubeAngle));
1397                     break;
1398                 case 2 :
1399                     cubeAngle.Set( (cubeAngle.x+90.0f), cubeAngle.y, (cubeAngle.z+180. ✓
1400             0f));
1401                     Instantiate( pSpreadShacklesSide, cubePosVect, Quaternion.Euler( ✓

```



```

    (cubeAngle));
1399         break;
1400
1401     case 3 :
1402         cubeAngle.Set( cubeAngle.x, cubeAngle.y, (cubeAngle.z+90.0f));
1403         cubePosVect.Set( cubePosVect.x, (cubePosVect.y+0.32f), cubePosVect
.z );
1404         Instantiate( pHangingCageSide, cubePosVect, Quaternion.Euler
(cubeAngle));
1405         break;
1406
1407     case 4 :
1408         cubeAngle.Set( cubeAngle.x, cubeAngle.y, (cubeAngle.z+90.0f));
1409         Instantiate( pPillorySide, cubePosVect, Quaternion.Euler
(cubeAngle));
1410         break;
1411
1412     default:
1413         break;
1414     }
1415     break;
1416
1417     case 7 :
1418         cubePosVect.Set( ((x * 5.12f) - 40.96f), 2.55f, ((z * 5.12f)) );
1419         Instantiate( p07L_HallRoomLR, cubePosVect, Quaternion.Euler( cubeAngle
));
1420         break;
1421
1422     case 8 :
1423         cubePosVect.Set( ((x * 5.12f) - 40.96f), 2.55f, ((z * 5.12f)) );
1424         Instantiate( p08L_HallRoom, cubePosVect, Quaternion.Euler( cubeAngle
));
1425         props = UnityEngine.Random.Range(1,4);
1426         switch( props ){
1427         case 1 :
1428         case 2 :
1429             cubePosVect.Set( cubePosVect.x, (cubePosVect.y+0.32f), cubePosVect
.z );
1430             Instantiate( pHangingCageSide, cubePosVect, Quaternion.Euler(
cubeAngle ));
1431             break;
1432
1433         default :
1434             break;
1435         }
1436         break;
1437
1438     case 9 :
1439         cubePosVect.Set( ((x * 5.12f) - 40.96f), 2.55f, ((z * 5.12f)) );
1440         Instantiate( p09L_RoomAngle, cubePosVect, Quaternion.Euler( cubeAngle
));
1441         props = UnityEngine.Random.Range(1,6);
1442         switch( props ){
1443         case 1 :
1444         case 2 :
1445             cubeAngle.Set( (cubeAngle.x+90.0f), (cubeAngle.y+90.0f), cubeAngle
.z);
1446             Instantiate( pJudasCradleCorner, cubePosVect, Quaternion.Euler
(cubeAngle));
1447             break;
1448
1449         case 3 :
1450             cubeAngle.Set( (cubeAngle.x+90.0f), cubeAngle.y, cubeAngle.z );
1451             Instantiate( pTortureRackCorner, cubePosVect, Quaternion.Euler(
cubeAngle ));
1452             break;

```

```

1453
1454         case 4 :
1455             cubeAngle.Set( cubeAngle.x, cubeAngle.y, (cubeAngle.z+90.0f));
1456             Instantiate( pPilloryCorner, cubePosVect, Quaternion.Euler(
1457                 (cubeAngle));
1458
1459                 break;
1460
1461             default :
1462                 break;
1463             }
1464             break;
1465
1466         case 10 :
1467             cubePosVect.Set( ((x * 5.12f) - 40.96f), 2.55f, ((z * 5.12f)) );
1468             Instantiate( p10L_SideRoom, cubePosVect, Quaternion.Euler( cubeAngle
1469                 ));
1470
1471             props = UnityEngine.Random.Range(1,8);
1472             switch( props ){
1473                 case 1 :
1474                     cubeAngle.Set( (cubeAngle.x+90.0f), cubeAngle.y, cubeAngle.z);
1475                     Instantiate( pWristShacklesSide, cubePosVect, Quaternion.Euler(
1476                         cubeAngle ));
1477
1478                     break;
1479
1480                 case 2 :
1481                     cubeAngle.Set( (cubeAngle.x+90.0f), cubeAngle.y, cubeAngle.z );
1482                     Instantiate( pSpreadShacklesSide, cubePosVect, Quaternion.Euler(
1483                         cubeAngle ));
1484
1485                     break;
1486
1487                 case 3 :
1488                     cubePosVect.Set( cubePosVect.x, (cubePosVect.y+0.32f), cubePosVect
1489                         .z );
1490                     Instantiate( pHangingCageSide, cubePosVect, Quaternion.Euler(
1491                         cubeAngle ));
1492
1493                     break;
1494
1495                 case 4 :
1496                     cubeAngle.Set( (cubeAngle.x+90.0f), cubeAngle.y, cubeAngle.z );
1497                     Instantiate( pTortureRackSide, cubePosVect, Quaternion.Euler(
1498                         cubeAngle ));
1499
1500                     break;
1501
1502                 case 5 :
1503                     cubeAngle.Set( (cubeAngle.x+90.0f), cubeAngle.y, cubeAngle.z );
1504                     Instantiate( pJudasCradleSide, cubePosVect, Quaternion.Euler(
1505                         cubeAngle ));
1506
1507                     break;
1508
1509                 case 6 :
1510                     Instantiate( pPillorySide, cubePosVect, Quaternion.Euler(
1511                         cubeAngle ));
1512
1513                     break;
1514             }
1515             break;
1516
1517         case 11 :
1518             cubePosVect.Set( ((x * 5.12f) - 40.96f), 2.55f, ((z * 5.12f)) );
1519             Instantiate( p11L_CornerRoom, cubePosVect, Quaternion.Euler( cubeAngle
1520                 ));
1521
1522             props = UnityEngine.Random.Range(1,8);
1523             switch( props ){
1524                 case 1 :
1525                     cubeAngle.Set( (cubeAngle.x+90.0f), cubeAngle.y, cubeAngle.z );
1526                     Instantiate( pWristShacklesCorner, cubePosVect, Quaternion.Euler(
1527                         cubeAngle ));

```

```

1508             break;
1509
1510         case 2 :
1511             cubeAngle.Set( (cubeAngle.x+90.0f), cubeAngle.y, cubeAngle.z );
1512             Instantiate( pSpreadShacklesCorner, cubePosVect, Quaternion.Euler(
cubeAngle ));
1513             break;
1514
1515         case 3 :
1516         case 4 :
1517             cubeAngle.Set( (cubeAngle.x+90.0f), cubeAngle.y, cubeAngle.z );
1518             Instantiate( pTortureRackCorner, cubePosVect, Quaternion.Euler(
cubeAngle ));
1519             break;
1520
1521         case 5 :
1522             cubeAngle.Set( (cubeAngle.x+90.0f), (cubeAngle.y+90.0f), cubeAngle
.z);
1523             Instantiate( pJudasCradleCorner, cubePosVect, Quaternion.Euler
(cubeAngle));
1524             break;
1525
1526         case 6 :
1527             cubeAngle.Set( cubeAngle.x, cubeAngle.y, (cubeAngle.z+90.0f));
1528             Instantiate( pPilloryCorner, cubePosVect, Quaternion.Euler
(cubeAngle));
1529             break;
1530         }
1531         break;
1532
1533     case 12 :
1534         cubePosVect.Set( ((x * 5.12f) - 40.96f), 2.55f, ((z * 5.12f)) );
1535         Instantiate( p12L_OffsetRoom, cubePosVect, Quaternion.Euler( cubeAngle
));
1536
1537
1538         break;
1539
1540     case 13 :
1541         cubePosVect.Set( ((x * 5.12f) - 40.96f), 2.55f, ((z * 5.12f)) );
1542         Instantiate( p13L_OpenRoom, cubePosVect, Quaternion.Euler( cubeAngle
));
1543
1544         props = UnityEngine.Random.Range(1,8);
1545         switch( props ){
1546         case 1 :
1547             cubeAngle.Set( (cubeAngle.x+90.0f), cubeAngle.y, cubeAngle.z );
1548             Instantiate( pWristShacklesQuad, cubePosVect, Quaternion.Euler(
cubeAngle ));
1549             break;
1550
1551         case 2 :
1552         case 3 :
1553             Instantiate( pPillorySide, cubePosVect, Quaternion.Euler(
cubeAngle ));
1554             break;
1555
1556         case 4 :
1557             cubeAngle.Set( (cubeAngle.x+90.0f), cubeAngle.y, cubeAngle.z );
1558             Instantiate( pTortureRackSide, cubePosVect, Quaternion.Euler(
cubeAngle ));
1559             break;
1560
1561         case 5 :
1562             cubeAngle.Set( (cubeAngle.x+90.0f), cubeAngle.y, cubeAngle.z );
1563             Instantiate( pJudasCradleSide, cubePosVect, Quaternion.Euler(
cubeAngle ));

```

```

1563             break;
1564
1565             case 6 :
1566                 cubePosVect.Set( cubePosVect.x, (cubePosVect.y+0.32f), cubePosVect
.z );
1567                 Instantiate( pHangingCageSide, cubePosVect, Quaternion.Euler(
cubeAngle ));
1568             break;
1569
1570             default :
1571                 break;
1572             }
1573             break;
1574
1575             case 14 :
1576                 cubePosVect.Set( ((x * 5.12f) - 40.96f), 1.8f, ((z * 5.12f)) );
1577                 Instantiate( p14H_EndHall, cubePosVect, Quaternion.Euler( cubeAngle ));
;
1578                 cubePosVect.Set( ((x * 5.12f) - 40.96f), 0.0f, ((z * 5.12f)) );
1579                 Instantiate( pSquareGrate, cubePosVect, Quaternion.Euler( cubeAngle ));
;
1580             break;
1581
1582             case 15 :
1583                 cubePosVect.Set( ((x * 5.12f) - 40.96f), 2.4f, ((z * 5.12f)) );
1584                 Instantiate( p17_SolidCube, cubePosVect, Quaternion.Euler( cubeAngle
));
1585             break;
1586
1587             default :
1588                 break;
1589             }
1590             break;
1591         }
1592     }
1593 }
1594 }
1595 }
1596 }
1597
1598 /*
1599 -----
1600 Spawn Enemies and Pickups
1601 -----
1602 */
1603 void SpawnStuff( ref MAPCUBE[,] MapArray ){
1604     Vector3 spawnPos = new Vector3();
1605     Vector3 spawnAngle = new Vector3();
1606
1607     for( int spawns = 0; spawns < 3; ++spawns ){
1608         int x = UnityEngine.Random.Range(1, mapWidth);
1609         int z = UnityEngine.Random.Range(1, mapLength);
1610
1611         if( MapArray[x,z].cube != -1 ){
1612             float yMod = 0.0f;
1613             if( (MapArray[x,z].cube == 1) ||
1614                 (MapArray[x,z].cube == 2) ||
1615                 (MapArray[x,z].cube == 3) ||
1616                 (MapArray[x,z].cube == 4) ||
1617                 (MapArray[x,z].cube == 14) ) yMod = 1.75f;
1618             else yMod = 2.5f;
1619             spawnPos = new Vector3( ((x * 5.12f) - 40.96f), yMod, ((z * 5.12f)) );
1620             spawnAngle = new Vector3( 0, 180, 0 );
1621             Instantiate( Lantern, spawnPos, Quaternion.Euler( spawnAngle ));
1622             Instantiate( LanternCore, spawnPos, Quaternion.Euler( spawnAngle ));
1623

```

```
1624         if( spawnMonsters ){
1625             if( !((x == ((mapWidth / 2) - 1) && z == 1) ||
1626                 (x == ((mapWidth / 2) - 1) && z == 2) ||
1627                 (x == ((mapWidth / 2) + (mapWidth % 2) + 1) && z == (mapLength - 2)) ||
1628                 (x == ((mapWidth / 2) + (mapWidth % 2) + 1) && z == (mapLength - 1))) ){
1629
1630                 spawnPos = new Vector3( ((x * 5.12f) - 40.96f), 0.5f, ((z * 5.12f)) );
1631                 spawnAngle = new Vector3( 0, 180, 0 );
1632                 Instantiate( Spectre, spawnPos, Quaternion.Euler( spawnAngle ));
1633             }
1634         }
1635     }
1636 }
1637 }
1638 }
```