```
/*
----------------------------------------------------------------------
    Procedural Map Generator v2
    Written by Jeff Fisher
    v.1 : July 3, 2013 - Junior group project)
    v.2 : October 27, 2013 - Personal side project)
    v.3 : August 31, 2014 - Personal extension of senior capstone)

----------------------------------------------------------------------
 */
using UnityEngine;
using System;
using System.Collections;
using System.Collections.Generic;
using System.IO;

public class sMapBuilder : MonoBehaviour{
    /*
----------------------------------------------------------------------
    Custom data types
----------------------------------------------------------------------
 */
    // Enumerates the directions that can be used in this application to simplify angles in 45 degree
    increments
    // --If the selection of map segment shapes is modified this may need to be changed.--
    enum DIR { angle0, angle45, angle90, angle135, angle180, angle225, angle270, angle315 }; // These
    are casted to (int) in use

    struct MAPCUBE{
        public int cube;    // Identity of cube
        public DIR yAngle;  // Angle of cube on Y axis expressed as DIR
        public int height;      // height of the cube in the map (+1, 0, or -1)
        // true is an open side in that direction false is a closed side
        public bool dir0;
        public bool dir2;
        public bool dir4;
        public bool dir6;

        public void Set( int c, DIR a, int h, bool d0, bool d2, bool d4, bool d6 ){
            cube = c;
            yAngle = a;
            height = h;
            dir0 = d0;
            dir2 = d2;
            dir4 = d4;
            dir6 = d6;
        }
    };

    struct CUBELOC{
        public int x;   // Column of cube in map
        public int z;   // Row of cube in map
        public void Set( int u, int w ){
            x = u;
            z = w;
        }
    };

    class ASTARNODE{
        private int x;  // Column of cube in map
        private int z;  // Row of cube in map
        private float dist; // Distance between start and finish
        private bool open;
        public void Set( int u, int w, float d, bool o ){
            x = u;
            z = w;
            dist = d;
            open = o;
        }
        public void Close(){
            open = false;
        }
        public int GetX(){
            return x;
        }
        public int GetZ(){
            return z;
        }
        public float GetDist(){
            return dist;
        }
        public bool GetOpen(){
            return open;
        }
        public string StringX(){
            return x.ToString();
        }
        public string StringZ(){
            return z.ToString();
        }
        public string StringDist(){
            return dist.ToString();
        }
        public string StringOpen(){
            return open.ToString();
        }
    }

    // Enumerates the specific sides and corners found in the set of map segments being used.
    // --If the selection of map segments is modified, this may need to be changed.--
    enum SIDE {     // Enumerated data type for different map cube side and corner characteristics
        broken,         // Indicates a side or corner that is not applicable to matching purposes
        solidCorner,    // Indicates that a corner must be matched to other corners that contain ↙
    structure
        emptyCorner,    // Indicates that a corner must be matched to other corners that contain no ↙
    structure
        hallSide,       // Indicates a side that is a hallway
        solidSide,      // Indicates a side that is a solid wall
        openSide,       // Indicates a side that is completely open
        lWallSide,      // Indicates an otherwise open side with a wall on the left
        rWallSide       // Indicates an otherwiste open side with a wall on the right
    };

    // Struct containing the array of cube sides and code to test for sides matching
    // --If the cube sides found in the selection of cubes is modified this will need to be changed--
    struct CUBESIDES{
        public SIDE[] sideArray;

        public CUBESIDES( SIDE side0, SIDE side45, SIDE side90, SIDE side135,
                          SIDE side180, SIDE side225, SIDE side270, SIDE side315 ){
            sideArray = new SIDE[8];
            sideArray[0] = side0;
            sideArray[1] = side45;
            sideArray[2] = side90;
            sideArray[3] = side135;
            sideArray[4] = side180;
            sideArray[5] = side225;
            sideArray[6] = side270;
            sideArray[7] = side315;
        }
        // This function matches cubes
        // --It needs to be changed if the selection of cube side types is changed
        public SIDE GetSide( DIR angle ){
            return sideArray[ (int)angle ];
        }

        public bool CompareSides( SIDE trySideType, DIR chkCubeSide, DIR chkCubeAngle ){
            int comboCubeAngle = ((((int)chkCubeSide + 4) % 8) - (int)chkCubeAngle + 8) % 8;

            if( sideArray[comboCubeAngle] == SIDE.broken ) return true;
            if( sideArray[comboCubeAngle] == SIDE.lWallSide){
                if( trySideType == SIDE.rWallSide) return true;
                else return false;
            }
            if( sideArray[comboCubeAngle] == SIDE.rWallSide){
                if( trySideType == SIDE.lWallSide ) return true;
                else return false;
            }
            if( sideArray[comboCubeAngle] == trySideType ) return true;
            else return false;
        }
    };

    /*
----------------------------------------------------------------------
    Make Prefabs Available
----------------------------------------------------------------------
 */
    public int mapLength = 16;
    public int mapWidth = 16;
    public bool spawnMonsters = false;
    //public int numMonsters = 0;
    public bool appendFileOutput = false;

    public Transform p14E_EndHall;
    public Transform p01H_StraightHall;
    public Transform p02H_CornerHall;
    public Transform p03H_CrossHall;
    public Transform p04H_TeeHall;
    public Transform p05H_HallRoomR;
    public Transform p06H_HallRoomL;
    public Transform p07H_HallRoomLR;
    public Transform p08H_HallRoom;
    public Transform p09H_RoomAngle;
    public Transform p10H_SideRoom;
    public Transform p11H_CornerRoom;
    public Transform p12H_OffsetRoom;
    public Transform p13H_OpenRoom;
    public Transform p14H_EndHall;
    public Transform p01L_StraightHall;
    public Transform p02L_CornerHall;
    public Transform p03L_CrossHall;
    public Transform p04L_TeeHall;
    public Transform p05L_HallRoomR;
    public Transform p06L_HallRoomL;
    public Transform p07L_HallRoomLR;
    public Transform p08L_HallRoom;
    public Transform p09L_RoomAngle;
    public Transform p10L_SideRoom;
    public Transform p11L_CornerRoom;
    public Transform p12L_OffsetRoom;
    public Transform p13L_OpenRoom;
    public Transform p14L_EndHall;
    public Transform p01W_StraightHall;
    public Transform p02W_CornerHall;
    public Transform p03W_CrossHall;
    public Transform p04W_TeeHall;
    public Transform p05W_HallRoomR;
    public Transform p06W_HallRoomL;
    public Transform p07W_HallRoomLR;
    public Transform p08W_HallRoom;
    public Transform p09W_RoomAngle;
    public Transform p10W_SideRoom;
    public Transform p11W_CornerRoom;
    public Transform p12W_OffsetRoom;
    public Transform p13W_OpenRoom;
    public Transform p14W_EndHall;
    public Transform p14X_EndHall;
    public Transform pSquareGrate;
    public Transform pSpiralStair;
    public Transform p17_SolidCube;
    public Transform pEnder;

    public Transform pHangingCageSide;
    public Transform pHangingCageQuad;
    public Transform pJudasCradleCorner;
    public Transform pJudasCradleSide;
    public Transform pPilloryCorner;
    public Transform pPillorySide;
    public Transform pWristShacklesAngle;
    public Transform pWristShacklesCorner;
    public Transform pWristShacklesSide;
    public Transform pWristShacklesQuad;
    public Transform pSpreadShacklesCorner;
    public Transform pSpreadShacklesSide;
    public Transform pTortureRackCorner;
    public Transform pTortureRackSide;
    public Transform pSewerPipeSide;
    public Transform pCoffinCageCorner;
    public Transform pCoffinCageSide;

    public Transform Spectre;
    public Transform Lantern;
    public Transform LanternCore;
    public Transform SteadyLight;
    public Transform pWaterPlane;
    private int startX;
    private int startZ;
    private int endX;
    private int endZ;

    /*
----------------------------------------------------------------------
    Initialize Data and Initiate Build
----------------------------------------------------------------------
 */
    void Start(){
        MAPCUBE[,] MapArray = new MAPCUBE[(mapLength+2),(mapWidth+2)]; // Array containing all of the ↙
    cubes in the map
        List<CUBELOC> OpenSquares = new List<CUBELOC>(); // Cubes that need to be filled by row and ↙
    collumn

        // Initializes list of cubes used.
        // --If selection of cubes is changed then this will need to be updated--
        CUBESIDES[] CubeInfo = new CUBESIDES[16];
        // Empty cube, may be used later, currently a space-filler
        CubeInfo[0] = new CUBESIDES( SIDE.broken, SIDE.broken, SIDE.broken, SIDE.broken,
                                     SIDE.broken, SIDE.broken, SIDE.broken, SIDE.broken );
        // StraightHall cube (straight section of hallway)
        CubeInfo[1] = new CUBESIDES( SIDE.hallSide, SIDE.solidCorner, SIDE.solidSide, SIDE.solidCorner↙
    ,
                                     SIDE.hallSide, SIDE.solidCorner, SIDE.solidSide, SIDE.solidCorner↙
     );
        // CornerHall cube (hall makes 90 degree turn)
        CubeInfo[2] = new CUBESIDES( SIDE.hallSide, SIDE.solidCorner, SIDE.solidSide, SIDE.solidCorner,
                                     SIDE.solidSide, SIDE.solidCorner, SIDE.hallSide, SIDE.solidCorner );
        // CrossHall cube (4-way intersection)
        CubeInfo[3] = new CUBESIDES( SIDE.hallSide, SIDE.solidCorner, SIDE.hallSide, SIDE.solidCorner,
                                     SIDE.hallSide, SIDE.solidCorner, SIDE.hallSide, SIDE.solidCorner );
        // TeeHall cube (3-way intersetion)
        CubeInfo[4] = new CUBESIDES( SIDE.solidSide, SIDE.solidCorner, SIDE.hallSide, SIDE.solidCorner,
                                     SIDE.hallSide, SIDE.solidCorner, SIDE.hallSide, SIDE.solidCorner );
        // HallRoomR cube (hall meets corner of room, room opens to the right)
        CubeInfo[5] = new CUBESIDES( SIDE.rWallSide, SIDE.emptyCorner, SIDE.lWallSide, SIDE.solidCorner,
                                     SIDE.solidSide, SIDE.solidCorner, SIDE.hallSide, SIDE.solidCorner );
        // HallRoomL cube (hall meeds corner of room, room opens to the left)
        CubeInfo[6] = new CUBESIDES( SIDE.solidSide, SIDE.solidCorner, SIDE.rWallSide, SIDE.emptyCorner,
                                     SIDE.lWallSide, SIDE.solidCorner, SIDE.hallSide, SIDE.solidCorner );
        // HallRoomLR cube (2 halls enter corner of room from left and right)
        CubeInfo[7] = new CUBESIDES( SIDE.rWallSide, SIDE.emptyCorner, SIDE.lWallSide, SIDE.solidCorner,
                                     SIDE.hallSide, SIDE.solidCorner, SIDE.hallSide, SIDE.solidCorner );
        // HallRoom (hall in center of only wall)
        CubeInfo[8] = new CUBESIDES( SIDE.openSide, SIDE.emptyCorner, SIDE.lWallSide, SIDE.solidCorner,
                                     SIDE.hallSide, SIDE.solidCorner, SIDE.rWallSide, SIDE.emptyCorner );
        // RoomAngle (inside corner of a room that turns)
        CubeInfo[9] = new CUBESIDES( SIDE.openSide, SIDE.emptyCorner, SIDE.openSide, SIDE.emptyCorner,
                                     SIDE.lWallSide, SIDE.solidCorner, SIDE.rWallSide, SIDE.emptyCorner);
        // SideRoom (side of a room)
        CubeInfo[10] = new CUBESIDES( SIDE.openSide, SIDE.emptyCorner, SIDE.lWallSide, SIDE.solidCorner,
                                      SIDE.solidSide, SIDE.solidCorner, SIDE.rWallSide, SIDE.emptyCorner );
        // CornerRoom (corner of a room)
        CubeInfo[11] = new CUBESIDES( SIDE.rWallSide, SIDE.emptyCorner, SIDE.lWallSide, SIDE.solidCorner,
                                      SIDE.solidSide, SIDE.solidCorner, SIDE.solidSide, SIDE.solidCorner );
        // OffsetRoom (2 rooms meet at corners)
        CubeInfo[12] = new CUBESIDES( SIDE.lWallSide, SIDE.solidCorner, SIDE.rWallSide, SIDE.emptyCorner,
                                      SIDE.lWallSide, SIDE.solidCorner, SIDE.rWallSide, SIDE.emptyCorner );
        // OpenRoom (open central area)
        CubeInfo[13] = new CUBESIDES( SIDE.openSide, SIDE.emptyCorner, SIDE.openSide, SIDE.emptyCorner,
                                      SIDE.openSide, SIDE.emptyCorner, SIDE.openSide, SIDE.emptyCorner );
        // EndHall (dead-end hallway)
        CubeInfo[14] = new CUBESIDES( SIDE.hallSide, SIDE.solidCorner, SIDE.solidSide, SIDE.solidCorner,
                                      SIDE.solidSide, SIDE.solidCorner, SIDE.solidSide, SIDE.solidCorner );
        // SolidCube (what it says - used for borders)
        CubeInfo[(17-2)] = new CUBESIDES( SIDE.solidSide, SIDE.solidCorner, SIDE.solidSide, SIDE.
    solidCorner,
                                          SIDE.solidSide, SIDE.solidCorner, SIDE.solidSide, SIDE.     ↙
    solidCorner );

        //Initiates build of main map
        int buildAttempts = 0;
        do{
            BuildMap( ref MapArray, ref OpenSquares, CubeInfo );
            buildAttempts++;
        }
        while( !FindPath( ref MapArray ) && buildAttempts < 4 );

        InstantiateMap (ref MapArray);

        // Fills the map with stuff
        SpawnStuff( ref MapArray );
    }


    /*
----------------------------------------------------------------------
    Builds the core of the map inside the frame
----------------------------------------------------------------------
 */
    void BuildMap( ref MAPCUBE[,] MapArray, ref List<CUBELOC> OpenSquares, CUBESIDES[] CubeInfo ){
        // Initializes map information variables
        int lastCubeNumber = 14; // The highest numbered possible cube
        int firstCubeNumber = 1; // The lowest numbered possible cube
        startX = (mapWidth / 2) - 1; // default X position of start cube
        startZ = 1; // Z position of start cube
        endX = (mapWidth / 2) + (mapWidth % 2) + 2; // default X position of end cube
        endZ = mapLength; // Z position of end cube

        Vector3 tempPos = new Vector3(0.0f, 0.0f, 0.0f);
        int x = 0; // The column of the current cube being modified within the map
        int z = 0; // They row of the current cube being modified within the map
        int yAngle = 0; // Temp variable containing the current rotation of a cube around the y axis
        MAPCUBE tempMapCube = new MAPCUBE(); // Temp variable for a Map Cube
        CUBELOC tempCubeLoc = new CUBELOC(); // Temp variable for the location of a cube
        Vector3 cubePosVect = new Vector3(0,0,0); // Contains 3D locations of cubes to be placed
        int cube = -1;
        int dir = -1;

        bool cubeWasSet = false; // used to determine if a cube was set successfully
        bool[] wasTriedCubes = new bool[lastCubeNumber+1]; // Tracks cubes that have been determined  ↙
    not to fit
        MAPCUBE[] BorderCube = new MAPCUBE[8]; // Array containing the cubes around a cube to be      ↙
    modified
        int testCubePick = -1; // variable to hold the number of the cube being tried
        int tryCubeAngle = -1; // tracks the angle that the cube is being tried at
        int loopCount = 0; // Tracks the number of loops to exit if the count becomes excessive
        int curWorkSquare = 0; // Tracks the index on the list of open sides that is being worked on

        // Initializes values of Map Array
        tempMapCube.Set( -1, 0, 0, false, false, false, false );
        for( x = 0; x < (mapLength+2); x++ ){
            for( z = 0; z < (mapWidth+2); z++ ){
                MapArray[x,z] = tempMapCube;
            }
        }

        // Sets row, column and angle of start cube
        yAngle = 90;
        x = startX;
        z = startZ;

        // Adds start point cube to map list
        cube = 14;
        dir = yAngle / 45;
        tempMapCube.Set( cube, (DIR)dir, 0,
                        !(CubeInfo[cube].sideArray[((8-dir)%8)] == SIDE.solidSide),
                        !(CubeInfo[cube].sideArray[((10-dir)%8)] == SIDE.solidSide),
                        !(CubeInfo[cube].sideArray[((12-dir)%8)] == SIDE.solidSide),
                        !(CubeInfo[cube].sideArray[((14-dir)%8)] == SIDE.solidSide) );
        MapArray[ x, z ] = tempMapCube;

        // Adds a cross hall cube to the map at the open end of the start cube
        z++;
        cubePosVect.Set( ((x * 5.12f) - 40.96f), 1.8f, ((z * 5.12f)) );

        // Adds the cross hall cube to the map list
        cube = 3;
        dir = yAngle / 45;
        tempMapCube.Set( 3, (DIR)dir, 0,
                        !(CubeInfo[cube].sideArray[((8-dir)%8)] == SIDE.solidSide),
                        !(CubeInfo[cube].sideArray[((10-dir)%8)] == SIDE.solidSide),
                        !(CubeInfo[cube].sideArray[((12-dir)%8)] == SIDE.solidSide),
                        !(CubeInfo[cube].sideArray[((14-dir)%8)] == SIDE.solidSide) );
        MapArray[ x, z ] = tempMapCube;

        // Add squares surrounding the cross hall cube to the list of open squares
        tempCubeLoc.Set( (x-1), z );
        OpenSquares.Add( tempCubeLoc );
        tempCubeLoc.Set( x, (z+1) );
        OpenSquares.Add( tempCubeLoc );
        tempCubeLoc.Set( (x+1), z );
        OpenSquares.Add( tempCubeLoc );

        // Sets row, column and angle of end cube
        yAngle = 270;
        x = endX;
        z = endZ;

        // Adds map end point to map list
        cube = 14;
        dir = yAngle / 45;
        tempMapCube.Set( 14, (DIR)dir, 1,
                        !(CubeInfo[cube].sideArray[((8-dir)%8)] == SIDE.solidSide),
                        !(CubeInfo[cube].sideArray[((10-dir)%8)] == SIDE.solidSide),
                        !(CubeInfo[cube].sideArray[((12-dir)%8)] == SIDE.solidSide),
                        !(CubeInfo[cube].sideArray[((14-dir)%8)] == SIDE.solidSide) );
        MapArray[ x, z ] = tempMapCube;

        // Creates a cross hall cube at the opening of the end point cube
        z--;
        cubePosVect.Set( ((x * 5.12f) - 40.96f), 1.8f, ((z * 5.12f)) );

        // Adds cross hall cube to the list of map cubes
        cube = 3;
        dir = yAngle / 45;
        tempMapCube.Set( 3, (DIR)dir, 1,
                        !(CubeInfo[cube].sideArray[((8-dir)%8)] == SIDE.solidSide),
                        !(CubeInfo[cube].sideArray[((10-dir)%8)] == SIDE.solidSide),
                        !(CubeInfo[cube].sideArray[((12-dir)%8)] == SIDE.solidSide),
                        !(CubeInfo[cube].sideArray[((14-dir)%8)] == SIDE.solidSide) );
        MapArray[ x, z ] = tempMapCube;

        // Add squares surrounding the cross hall cube to the list of open squares
        tempCubeLoc.Set( (x-1), z );
        OpenSquares.Add( tempCubeLoc );
        tempCubeLoc.Set( x, (z-1) );
        OpenSquares.Add( tempCubeLoc );
        tempCubeLoc.Set( (x+1), z );
        OpenSquares.Add( tempCubeLoc );


        // FOR loops build sides of "frame" composed of solid cubes to contain the map
        // --If size of map is changed, this will need to be modified
        tempMapCube.Set (15, (DIR)0, 0, false, false, false, false);

        x = 0;
        for( z = 0; z <= mapLength+1; z++ ){
            cubePosVect.Set( ((x * 5.12f) - 40.96f), -1.28f, (z * 5.12f) );
            Instantiate( p17_SolidCube, cubePosVect, Quaternion.Euler(-90,0,0 ));
            MapArray[x,z] = tempMapCube;
        }

        z = mapLength+1;
        for( x = 0; x <= mapWidth+1; x++ ){
            cubePosVect.Set( ((x * 5.12f) - 40.96f), -1.28f, (z * 5.12f) );
            Instantiate( p17_SolidCube, cubePosVect, Quaternion.Euler(-90,0,0 ));
            MapArray[x,z] = tempMapCube;
        }

        x = mapWidth+1;
        for( z = mapLength+1; z >= 0; z-- ){
            cubePosVect.Set( ((x * 5.12f) - 40.96f), -1.28f, (z * 5.12f) );
            Instantiate( p17_SolidCube, cubePosVect, Quaternion.Euler(-90,0,0));
            MapArray[x,z] = tempMapCube;
        }

        z = 0;
        for( x = mapWidth+1; x >= 0; x-- ){
            cubePosVect.Set( ((x * 5.12f) - 40.96f), -1.28f, (z * 5.12f) );
            Instantiate( p17_SolidCube, cubePosVect, Quaternion.Euler(-90,0,0));
            MapArray[x,z] = tempMapCube;
        }

        // Loops as long as there are open squares to be filled
        while( curWorkSquare < OpenSquares.Count ){
            // Iterate through open edge square list placing blocks
            // Verify that the open square has not been filled already
            tempCubeLoc = OpenSquares[ curWorkSquare ];
            x = tempCubeLoc.x;
            z = tempCubeLoc.z;
            tempMapCube = MapArray[x,z];
            if( tempMapCube.cube == -1 ){   // -1 assigned to represent empty cube
                // Create a list of all cubes bordering the open square
                BorderCube[0] = MapArray[(x+1), z    ];  // right side
                BorderCube[1] = MapArray[(x+1),(z+1)];  // upper right corner
                BorderCube[2] = MapArray[ x,    (z+1)];  // top side
                BorderCube[3] = MapArray[(x-1),(z+1)];  // upper left corner
                BorderCube[4] = MapArray[(x-1), z    ];  // left side
                BorderCube[5] = MapArray[(x-1),(z-1)];  // lower left corner
                BorderCube[6] = MapArray[ x,    (z-1)];  // bottom side
                BorderCube[7] = MapArray[(x+1),(z-1)];  // lower right corner

                // Initialize the array of cubes that have been tried;
                wasTriedCubes[0] = true; // Workaround so that search skips 0
                for( int idx = firstCubeNumber; idx <= lastCubeNumber; idx++){
                    wasTriedCubes[idx] = false;
                }
                while( !cubeWasSet ){ // Still searching for a cube?
                    // Break out of endless loop
                    loopCount++;
                    if( loopCount > 5000 ) return;

                    // Pick a random cube that hasn't been tried yet
                    bool alreadyTried = false;
                    do{
                        // Searches for an index which hasn't been tried (is false),
                        // If it finds one the equality is false
                        testCubePick = UnityEngine.Random.Range(1, 15);
                        alreadyTried = wasTriedCubes[testCubePick];
                        wasTriedCubes[testCubePick] = true;
                    }
                    while( alreadyTried );

                    // Check if that cube fits
                    tryCubeAngle = TryCube( testCubePick, x, z, ref MapArray, ref BorderCube,
                                            ref OpenSquares, ref curWorkSquare, CubeInfo );
                    if( tryCubeAngle > -1 ){ // -1 is no angle fits, other number is int cast of DIR ↙
    angle
                        cubeWasSet = true;
                        UpdateMap( testCubePick, x, z, ref MapArray, ref BorderCube,
                                   ref OpenSquares, ref curWorkSquare, CubeInfo, (DIR)tryCubeAngle );
                    }
                }
                // reset control variable
                cubeWasSet = false;
            }
            else{
                curWorkSquare++;
            }
        }
    }
    /*
-----------------------------------------------------------------------
   Checks and sets map cubes
-----------------------------------------------------------------------
 */
    int TryCube( int testCube, int x, int z, ref MAPCUBE[,] MapArray, ref MAPCUBE[] BorderCube,
                 ref List<CUBELOC> OpenSquares, ref int curWorkSquare, CUBESIDES[] CubeInfo ){
        int comboRotAngle = -1; // -1 is no match, a match is the int equivalent of the DIR angle
        bool checkedCubeMatches = true;
        int rndAngle = UnityEngine.Random.Range(0,3) * 2;
        for( int forRot = 0; forRot < 8 ; forRot += 2 ){
            comboRotAngle = (rndAngle + forRot) % 8;
            checkedCubeMatches = true;
            for( int forSide = 0; forSide < 8; forSide++ ){
                // Gets the side of the cube being tested
                DIR rotCubeSide = (DIR)((forSide - comboRotAngle + 8) % 8);
                SIDE trySideType = CubeInfo[ testCube ].GetSide( rotCubeSide );
                DIR chkCubeAngle = BorderCube[ forSide ].yAngle;
                DIR chkCubeSide = (DIR)forSide;
                int chkCubeNum = BorderCube[ forSide ].cube;
                if( chkCubeNum > -1 ){
                    // Compares side of cube being tested to bordering cube
                    checkedCubeMatches = CubeInfo[ chkCubeNum ].CompareSides( trySideType, chkCubeSide↙
    , chkCubeAngle );
                    if( !checkedCubeMatches ){
                        break;
                    }
                }
            }
            if( checkedCubeMatches ){
                return comboRotAngle;
            }
            else{
                comboRotAngle = -1;
            }
        }
        return comboRotAngle = -1;
    }

    /*
----------------------------------------------------------------------
   Updates the mapping data
----------------------------------------------------------------------
 */

    void UpdateMap( int testCube, int x, int z, ref MAPCUBE[,] MapArray, ref MAPCUBE[] BorderCube,
                    ref List<CUBELOC> OpenSquares, ref int curWorkSquare, CUBESIDES[] CubeInfo, DIR comboRotAngle){
        MAPCUBE tempMapCube = new MAPCUBE(); // Cube type that was selected
        CUBELOC tempCubeLoc = new CUBELOC(); // Location the cube will be placed
        Vector3 cubePosVect = new Vector3( ((x * 5.12f) - 40.96f), 1.8f, ((z * 5.12f)) ); // 3D cube position
        int newAngle = (int)comboRotAngle;// * 45;

        //-----------------------------------------------------------------
        // **WORKAROUND**
        // Correcting rotation of map cube 6
        if( testCube == 6 ) newAngle += 2; // This cube model needs a +90 degree correction
        //End of workaround
        //----------------------------------------------------------------

        Vector3 cubeAngle = new Vector3( -90, ((newAngle * -1) + 180), 0 );

        tempMapCube.Set( testCube, (DIR)comboRotAngle, 0,
                        !(CubeInfo[testCube].sideArray[((8-(int)comboRotAngle)%8)] == SIDE.solidSide),
                        !(CubeInfo[testCube].sideArray[((10-(int)comboRotAngle)%8)] == SIDE.solidSide),
                        !(CubeInfo[testCube].sideArray[((12-(int)comboRotAngle)%8)] == SIDE.solidSide),
                        !(CubeInfo[testCube].sideArray[((14-(int)comboRotAngle)%8)] == SIDE.solidSide) );
        MapArray[x,z] = tempMapCube;

        if( BorderCube[0].cube == -1 ){
            SIDE side = CubeInfo[ testCube ].GetSide( (DIR)((8 - newAngle) % 8) );
            if( ((side != SIDE.broken) && (side != SIDE.solidSide)) ){
                tempCubeLoc.Set( (x+1), z );
                OpenSquares.Add( tempCubeLoc );
            }
        }
        if( BorderCube[2].cube == -1 ){
            SIDE side = CubeInfo[ testCube ].GetSide( (DIR)((10 - newAngle) % 8) );
            if( ((side != SIDE.broken) && (side != SIDE.solidSide)) ){
                tempCubeLoc.Set( x, (z+1) );
                OpenSquares.Add( tempCubeLoc );
            }
        }
        if( BorderCube[4].cube == -1 ){
            SIDE side = CubeInfo[ testCube ].GetSide( (DIR)((12 - newAngle) % 8) );
            if( ((side != SIDE.broken) && (side != SIDE.solidSide)) ){
                tempCubeLoc.Set( (x-1), z );
                OpenSquares.Add( tempCubeLoc );
            }
        }
        if( BorderCube[6].cube == -1 ){
            SIDE side = CubeInfo[ testCube ].GetSide( (DIR)((14 - newAngle) % 8) );
            if( ((side != SIDE.broken) && (side != SIDE.solidSide)) ){
                tempCubeLoc.Set( x, (z-1) );
                OpenSquares.Add( tempCubeLoc );
            }
        }
        curWorkSquare++;
    }

    /*
----------------------------------------------------------------------
    Verify Path to Goal (A*)
----------------------------------------------------------------------
 */
    bool FindPath( ref MAPCUBE[,] MapArray ){
        CUBELOC tempLoc = new CUBELOC();
        ASTARNODE tempCube = new ASTARNODE();
        // Create a ASTARNODE list for the frontier
        List<ASTARNODE> frontier = new List<ASTARNODE>();

        float maxDistance = Mathf.Sqrt( Mathf.Pow (mapLength, 2) + Mathf.Pow (mapWidth, 2) );

        // Set initial node equal to start cube of map
        tempCube.Set( startX, startZ, PathDistance( startX, startZ ), true );
        frontier.Add(tempCube);
        MAPCUBE tempMapCube;
        CUBELOC[] tempMapLinks = new CUBELOC[4];
        CUBELOC[] rotMapLinks = new CUBELOC[4];
        int tempDir = 0;
        int listIdx = 0;
        int loopCount = 0;

        while( loopCount < (mapLength * mapWidth) ){
            loopCount++;
            float tempDistance = maxDistance;
            int tempClosestNode = -1;
            listIdx = 0;

            // Iterate through the list of nodes on the frontier
            while( listIdx < frontier.Count ){
                if( frontier[listIdx].GetOpen() ){
                    // Find the node on the frontier that is closest to the goal
                    if( frontier[listIdx].GetDist() < tempDistance ){
                        tempDistance = frontier[listIdx].GetDist();
                        tempClosestNode = listIdx;
                    }
                }
                listIdx++;
            }
            // If entire frontier is closed there is no path to the exit

            if( tempClosestNode == -1 ){
                return false;
            }

            // Remove closest node from open list
            frontier[tempClosestNode].Close();

            // Store X and Z values in temporary CUBELOC variable to simplify next steps
            tempLoc.x = frontier[tempClosestNode].GetX();
            tempLoc.z = frontier[tempClosestNode].GetZ();

            tempMapCube = MapArray[tempLoc.x,tempLoc.z];
            tempDir = (int)tempMapCube.yAngle / 2; // convirts DIR to int and gets range 0-3 instead of 0-6

            // Reads links data from map cube and adds linked cubes to an array
            if( tempMapCube.dir0 ){
                tempMapLinks[((4-tempDir)%4)].x = tempLoc.x+1;//tempMapLinks[((0+tempDir)%4)].x = tempLoc.x-1;
                tempMapLinks[((4-tempDir)%4)].z = tempLoc.z;//tempMapLinks[((0+tempDir)%4)].z = tempLoc.z;

                if( (tempMapLinks[((4-tempDir)%4)].x == endX) && (tempMapLinks[((4-tempDir)%4)].z == endZ) ){
                    return true;
                }
            }
            else{
                tempMapLinks[((4-tempDir)%4)].x = -1;//tempMapLinks[((0+tempDir)%4)].x = -1;
                tempMapLinks[((4-tempDir)%4)].z = -1;//tempMapLinks[((0+tempDir)%4)].z = -1;
            }

            if( tempMapCube.dir2 ){
                tempMapLinks[((5-tempDir)%4)].x = tempLoc.x;//tempMapLinks[((1+tempDir)%4)].x = tempLoc.x;
                tempMapLinks[((5-tempDir)%4)].z = tempLoc.z+1;//tempMapLinks[((1+tempDir)%4)].z = tempLoc.z+1;

                if( (tempMapLinks[((5-tempDir)%4)].x == endX) && (tempMapLinks[((5-tempDir)%4)].z == endZ) ){
                    return true;
                }
            }
            else{
                tempMapLinks[((5-tempDir)%4)].x = -1;//tempMapLinks[((1+tempDir)%4)].x = -1;
                tempMapLinks[((5-tempDir)%4)].z = -1;//tempMapLinks[((1+tempDir)%4)].z = -1;
            }

            if( tempMapCube.dir4 ){
                tempMapLinks[((6-tempDir)%4)].x = tempLoc.x-1;//tempMapLinks[((2+tempDir)%4)].x = tempLoc.x+1;
                tempMapLinks[((6-tempDir)%4)].z = tempLoc.z;//tempMapLinks[((2+tempDir)%4)].z = tempLoc.z;

                if( (tempMapLinks[((6-tempDir)%4)].x == endX) && (tempMapLinks[((6-tempDir)%4)].z == endZ) ){
                    return true;
                }
            }
            else{
                tempMapLinks[((6-tempDir)%4)].x = -1;//tempMapLinks[((2+tempDir)%4)].x = -1;
                tempMapLinks[((6-tempDir)%4)].z = -1;//tempMapLinks[((2+tempDir)%4)].z = -1;
            }

            if( tempMapCube.dir6 ){
                tempMapLinks[((7-tempDir)%4)].x = tempLoc.x;//tempMapLinks[((3+tempDir)%4)].x = tempLoc.x;
                tempMapLinks[((7-tempDir)%4)].z = tempLoc.z-1;//tempMapLinks[((3+tempDir)%4)].z = tempLoc.z-1;

                if( (tempMapLinks[((7-tempDir)%4)].x == endX) && (tempMapLinks[((7-tempDir)%4)].z == endZ) ){
                    return true;
                }
            }
            else{
                tempMapLinks[((7-tempDir)%4)].x = -1;//tempMapLinks[((3+tempDir)%4)].x = -1;
                tempMapLinks[((7-tempDir)%4)].z = -1;//tempMapLinks[((3+tempDir)%4)].z = -1;
            }

            // Checks list
            for( int mapIdx = 0; mapIdx < 4; mapIdx++ ){
                if( tempMapLinks[mapIdx].x != -1 ){ // would be -1 if there was no link on that side
                    listIdx = 0;
                    bool duplicateNode = false;
                    while( listIdx < frontier.Count && !duplicateNode ){
                        // Checks if there is a node on the list corresponding to the new location
                        if( (frontier[listIdx].GetX() == tempMapLinks[mapIdx].x) &&
                           (frontier[listIdx].GetZ() == tempMapLinks[mapIdx].z)  ){
                            duplicateNode = true;
                        }
                        listIdx++;
                    }
                    if( !duplicateNode ){
                        ASTARNODE nodeToAdd = new ASTARNODE();
                        nodeToAdd.Set( tempMapLinks[mapIdx].x, tempMapLinks[mapIdx].z,
                                       PathDistance( tempMapLinks[mapIdx].x, tempMapLinks[mapIdx].z ), ↙
    true );
                        frontier.Add( nodeToAdd );
                    }
                }
            }
        }
        ///////////////////////////////////////////////////////////////////////////////////////↙
////////
        using (System.IO.StreamWriter start = new System.IO.StreamWriter( @"C:\temp\OutputLog.txt",   ↙
    true ))
        { start.WriteLine(  " ** Returning \"FALSE\" **" ); }
        ///////////////////////////////////////////////////////////////////////////////////////↙
////////
        return false;
    }



    /*
----------------------------------------------------------------------
    Calculate Distance Between Start and End
----------------------------------------------------------------------
 */
    float PathDistance( int x, int z ){
        float dStart = Mathf.Sqrt( Mathf.Pow( (startX - x), 2 ) + (Mathf.Pow( (startZ - z), 2 )));
        float dEnd = Mathf.Sqrt( Mathf.Pow( (endX - x), 2 ) + (Mathf.Pow( (endZ - z), 2 )));
        float dTotal = dStart + dEnd;

        return dTotal;
    }


    /*
----------------------------------------------------------------------
    Place Map Cubes
----------------------------------------------------------------------
 */

    void InstantiateMap( ref MAPCUBE[,] MapArray ){
        Vector3 spawnPos = new Vector3( 0.0f, 0.0f, 0.0f );
        Vector3 spawnAngle = new Vector3( 0.0f, 0.0f, 0.0f );
        int cubeLevel = 0;
        int props = 0;;
        float yMod = 0.0f;
        int newCube = 0;
        int newAngle = 0;

        for( int x = 1; x < (mapWidth+1); x++ ){
            for( int z = 1 ; z < (mapLength+1); z++ ){
                if( MapArray[x,z].cube != -1 ){
                    if( x == startX && z == startZ ){
                        // Creates map cube containing player start point
                        newAngle = 90;
                        spawnPos.Set( ((x * 5.12f) - 40.96f), 1.8f, (z * 5.12f) );
                        Instantiate( p14E_EndHall, spawnPos, Quaternion.Euler(-90,newAngle,0) );
                        spawnPos.Set( ((x * 5.12f) - 40.96f), 1.82f, (z * 5.12f) );
                        Instantiate (pSpiralStair, spawnPos, Quaternion.Euler (-90, (newAngle+180),   ↙
    0));
                        spawnPos.Set( ((x * 5.12f) - 40.96f), 1.8f, (z * 5.12f + 0.64f) );
                        Instantiate( Lantern, spawnPos, Quaternion.identity );
                        Instantiate( SteadyLight, spawnPos, Quaternion.identity );
                    }
                    else if( x == endX && z == endZ ){
                        // Creates map cube containing map end point
                        newAngle = 270;
                        spawnPos.Set( ((x * 5.12f) - 40.96f), 1.8f, ((z * 5.12f)) );
                        Instantiate( p14X_EndHall, spawnPos, Quaternion.Euler(-90,newAngle,0) );
                        Instantiate (pEnder, spawnPos, Quaternion.identity);
                        spawnPos.Set( ((x * 5.12f) - 40.96f), 7.5f, ((z * 5.12f)) );
                        Instantiate (pSpiralStair, spawnPos, Quaternion.Euler (-90, (newAngle+270), 0));
                        spawnPos.Set( ((x * 5.12f) - 40.96f), 5.1f, ((z * 5.12f)) );
                        Instantiate (pSpiralStair, spawnPos, Quaternion.Euler (-90, (newAngle+270), 0));
                        spawnPos.Set( ((x * 5.12f) - 40.96f), 1.8f, ((z * 5.12f - 1.28f)) );
                        Instantiate( Lantern, spawnPos, Quaternion.identity );
                        Instantiate( SteadyLight, spawnPos, Quaternion.identity );
                    }
                    else{

                        newCube = MapArray[x,z].cube;
                        newAngle = (int)MapArray[x,z].yAngle * 45;

                        // 3D cube position
                        Vector3 cubePosVect = new Vector3( ((x * 5.12f) - 40.96f), 1.8f, ((z * 5.12f)) );
                        Vector3 cubeAngle = new Vector3( -90, ((newAngle * -1) + 180), 0 );

                        if( (newCube == 1) ||
                            (newCube == 2) ||
                            (newCube == 3) ||
                            (newCube == 4) ||
                            (newCube == 14)  ) yMod = 1.75f;
                        else yMod = 2.5f;

                        spawnPos = new Vector3( ((x * 5.12f) - 40.96f), yMod, ((z * 5.12f)) );
                        spawnAngle = new Vector3( 0, 180, 0 );

                        cubeLevel= UnityEngine.Random.Range(-1,2);

                        Instantiate( Lantern, spawnPos, Quaternion.Euler( spawnAngle ));
                        Instantiate( LanternCore, spawnPos, Quaternion.Euler( spawnAngle ));

                        if( x == startX && z == (startZ+1) ) cubeLevel = 0;
                        if( x == endX && z == (endZ-1) ) cubeLevel = 1;

                        switch( cubeLevel ){
                        case -1 :
                            cubePosVect.Set( ((x * 5.12f) - 40.96f), 2.55f, ((z * 5.12f)) );
                            Instantiate( pWaterPlane, cubePosVect, Quaternion.Euler( cubeAngle ));
                            switch( newCube ){
                            case 1 :
                                cubePosVect.Set( ((x * 5.12f) - 40.96f), 1.8f, ((z * 5.12f)) );
                                Instantiate( p01W_StraightHall, cubePosVect, Quaternion.Euler( cubeAngle ));
                                break;

                            case 2 :
                                cubePosVect.Set( ((x * 5.12f) - 40.96f), 1.8f, ((z * 5.12f)) );
                                Instantiate( p02W_CornerHall, cubePosVect, Quaternion.Euler( cubeAngle ));
                                break;

                            case 3 :
                                cubePosVect.Set( ((x * 5.12f) - 40.96f), 1.8f, ((z * 5.12f)) );
```

```
                                Instantiate( p03W_CrossHall, cubePosVect, Quaternion.Euler( cubeAngle ));
                                break;

                            case 4 :
                                cubePosVect.Set( ((x * 5.12f) - 40.96f), 1.8f, ((z * 5.12f)) );
                                Instantiate( p04W_TeeHall, cubePosVect, Quaternion.Euler( cubeAngle ));
                                break;

                            case 5 :
                                cubePosVect.Set( ((x * 5.12f) - 40.96f), 2.55f, ((z * 5.12f)) );
                                Instantiate( p05W_HallRoomR, cubePosVect, Quaternion.Euler( cubeAngle ));
                                props = UnityEngine.Random.Range(1,4);
                                switch( props ){
                                case 1 :
                                    cubePosVect.Set( cubePosVect.x, (cubePosVect.y-0.64f), cubePosVect.z );
                                    Instantiate( pSewerPipeSide, cubePosVect, Quaternion.Euler( cubeAngle ));
                                    break;

                                case 2 :
                                    cubeAngle.Set( cubeAngle.x, cubeAngle.y, (cubeAngle.z-90.0f));
                                    cubePosVect.Set( cubePosVect.x, (cubePosVect.y+0.32f), cubePosVect.z );
                                    Instantiate( pHangingCageSide, cubePosVect, Quaternion.Euler(cubeAngle));
                                    break;

                                default :
                                    break;
                                }
                                break;

                            case 6 :
                                cubePosVect.Set( ((x * 5.12f) - 40.96f), 2.55f, ((z * 5.12f)) );
                                //-------------------------------------
                                // **WORKAROUND**
                                Vector3 fixAngle = new Vector3( cubeAngle.x, (cubeAngle.y - 90), cubeAngle.z);
                                // **End of workaround**
                                //-------------------------------------
                                Instantiate( p06W_HallRoomL, cubePosVect, Quaternion.Euler( fixAngle ));
                                props = UnityEngine.Random.Range(1,4);
                                switch( props ){
                                case 1 :
                                    cubePosVect.Set( cubePosVect.x, (cubePosVect.y-0.64f), cubePosVect.z );
                                    cubeAngle.Set( cubeAngle.x, (cubeAngle.y+180), cubeAngle.z);
                                    Instantiate( pSewerPipeSide, cubePosVect, Quaternion.Euler(cubeAngle));
                                    break;

                                case 2 :
                                    cubeAngle.Set( cubeAngle.x, cubeAngle.y, (cubeAngle.z+90.0f));
                                    cubePosVect.Set( cubePosVect.x, (cubePosVect.y+0.32f), cubePosVect.z );
                                    Instantiate( pHangingCageSide, cubePosVect, Quaternion.Euler(cubeAngle));
                                    break;

                                default :
                                    break;
                                }
                                break;

                            case 7 :
                                cubePosVect.Set( ((x * 5.12f) - 40.96f), 2.55f, ((z * 5.12f)) );
                                Instantiate( p07W_HallRoomLR, cubePosVect, Quaternion.Euler( cubeAngle ));
                                props = UnityEngine.Random.Range(1,4);
                                switch( props ){
                                case 1 :
                                case 2 :
                                    cubeAngle.Set( cubeAngle.x, cubeAngle.y, (cubeAngle.z+180.0f));
                                    Instantiate( pCoffinCageCorner, cubePosVect, Quaternion.Euler(cubeAngle));
                                    break;

                                default :
                                    break;
                                }
                                break;

                            case 8 :
                                cubePosVect.Set( ((x * 5.12f) - 40.96f), 2.55f, ((z * 5.12f)) );
                                Instantiate( p08W_HallRoom, cubePosVect, Quaternion.Euler( cubeAngle ));
                                props = UnityEngine.Random.Range(1,4);
                                switch( props ){
                                case 1 :
                                case 2 :
                                    cubePosVect.Set( cubePosVect.x, (cubePosVect.y+0.32f), cubePosVect.z );
                                    Instantiate( pHangingCageSide, cubePosVect, Quaternion.Euler( cubeAngle ));
                                    break;

                                default :
                                    break;
                                }
                                break;

                            case 9 :
                                cubePosVect.Set( ((x * 5.12f) - 40.96f), 2.55f, ((z * 5.12f)) );
                                Instantiate( p09W_RoomAngle, cubePosVect, Quaternion.Euler( cubeAngle ));
                                props = UnityEngine.Random.Range(1,4);
                                switch( props ){
                                case 1 :
                                case 2 :
                                    cubeAngle.Set( cubeAngle.x, cubeAngle.y, (cubeAngle.z+180.0f));
                                    Instantiate( pCoffinCageCorner, cubePosVect, Quaternion.Euler(cubeAngle));
                                    break;

                                default :
                                    break;
                                }
                                break;

                            case 10 :
                                cubePosVect.Set( ((x * 5.12f) - 40.96f), 2.55f, ((z * 5.12f)) );
                                Instantiate( p10W_SideRoom, cubePosVect, Quaternion.Euler( cubeAngle ));
                                props = UnityEngine.Random.Range(1,6);
                                switch( props ){
                                case 1 :
                                    Instantiate( pCoffinCageSide, cubePosVect, Quaternion.Euler(
cubeAngle ));
                                    break;

                                case 2 :
                                    cubePosVect.Set( cubePosVect.x, (cubePosVect.y+0.32f), cubePosVect.z );
                                    Instantiate( pHangingCageSide, cubePosVect, Quaternion.Euler( cubeAngle ));
                                    break;

                                case 3 :
                                    cubePosVect.Set( cubePosVect.x, (cubePosVect.y-0.64f), cubePosVect.z );
                                    Instantiate( pSewerPipeSide, cubePosVect, Quaternion.Euler( cubeAngle ));
                                    break;

                                case 4 :
                                    cubePosVect.Set( cubePosVect.x, (cubePosVect.y-0.64f), cubePosVect.z );
                                    Instantiate( pSewerPipeSide, cubePosVect, Quaternion.Euler( cubeAngle ));
                                    cubePosVect.Set( cubePosVect.x, (cubePosVect.y+0.96f), cubePosVect.z );
                                    Instantiate( pHangingCageSide, cubePosVect, Quaternion.Euler( cubeAngle ));
                                    break;
                                }
                                break;

                            case 11 :
                                cubePosVect.Set( ((x * 5.12f) - 40.96f), 2.55f, ((z * 5.12f)) );
                                Instantiate( p11W_CornerRoom, cubePosVect, Quaternion.Euler( cubeAngle ));
                                props = UnityEngine.Random.Range(1,6);
                                switch( props ){
                                case 1 :
                                    cubeAngle.Set( cubeAngle.x, cubeAngle.y, (cubeAngle.z+180.0f));
                                    Instantiate( pCoffinCageCorner, cubePosVect, Quaternion.Euler(cubeAngle));
                                    break;

                                case 2 :
                                case 3 :
                                    cubePosVect.Set( cubePosVect.x, (cubePosVect.y-0.64f), cubePosVect.z );
                                    Instantiate( pSewerPipeSide, cubePosVect, Quaternion.Euler( cubeAngle ));
                                    break;

                                case 4 :
                                    cubePosVect.Set( cubePosVect.x, (cubePosVect.y-0.64f), cubePosVect.z );
                                    cubeAngle.Set( cubeAngle.x, (cubeAngle.y-90.0f), cubeAngle.z);
                                    Instantiate( pSewerPipeSide, cubePosVect, Quaternion.Euler(cubeAngle));
                                    break;
                                }
                                break;

                            case 12 :
                                cubePosVect.Set( ((x * 5.12f) - 40.96f), 2.55f, ((z * 5.12f)) );
                                Instantiate( p12W_OffsetRoom, cubePosVect, Quaternion.Euler( cubeAngle ));
                                break;

```

```
                        case 13 :
                            cubePosVect.Set( ((x * 5.12f) - 40.96f), 2.55f, ((z * 5.12f)) );
                            Instantiate( p13W_OpenRoom, cubePosVect, Quaternion.Euler( cubeAngle ));
                            props = UnityEngine.Random.Range(1,6);
                            switch( props ){
                            case 1 :
                            case 2 :
                                Instantiate( pCoffinCageSide, cubePosVect, Quaternion.Euler( cubeAngle ));
                                break;

                            case 3 :
                                //cubePosVect.Set( cubePosVect.x, (cubePosVect.y+0.32f), cubePosVect.z );
                                //Instantiate( pHangingCageQuad, cubePosVect, Quaternion.Euler( cubeAngle ));
                                //break;

                            case 4 :
                                cubePosVect.Set( cubePosVect.x, (cubePosVect.y+0.32f), cubePosVect.z );
                                Instantiate( pHangingCageSide, cubePosVect, Quaternion.Euler( cubeAngle ));
                                break;

                            default :
                                break;
                            }
                            break;

                        case 14 :
                            cubePosVect.Set( ((x * 5.12f) - 40.96f), 1.8f, ((z * 5.12f)) );
                            Instantiate( p14W_EndHall, cubePosVect, Quaternion.Euler( cubeAngle ));
                            props = UnityEngine.Random.Range(1,4);
                            switch( props ){
                            case 1 :
                                Instantiate( pSewerPipeSide, cubePosVect, Quaternion.Euler( cubeAngle ));
                                break;

                            case 2 :
                                cubePosVect.Set( cubePosVect.x, (cubePosVect.y+0.32f), cubePosVect.z );
                                Instantiate( pHangingCageSide, cubePosVect, Quaternion.Euler( cubeAngle ));
                                break;
                            }
                            break;

                        case 15 :
                            cubePosVect.Set( ((x * 5.12f) - 40.96f), 2.4f, ((z * 5.12f)) );
                            Instantiate( p17_SolidCube, cubePosVect, Quaternion.Euler( cubeAngle ));
                            break;

                        default : break;
                        }
                        break;

                    case 0 :
                        switch( newCube ){
                        case 1 :
                            cubePosVect.Set( ((x * 5.12f) - 40.96f), 1.8f, ((z * 5.12f)) );
                            Instantiate( p01L_StraightHall, cubePosVect, Quaternion.Euler(
cubeAngle ));
                            break;

                        case 2 :
                            cubePosVect.Set( ((x * 5.12f) - 40.96f), 1.8f, ((z * 5.12f)) );
                            Instantiate( p02L_CornerHall, cubePosVect, Quaternion.Euler( cubeAngle ));
                            break;

                        case 3 :
                            cubePosVect.Set( ((x * 5.12f) - 40.96f), 1.8f, ((z * 5.12f)) );
                            Instantiate( p03L_CrossHall, cubePosVect, Quaternion.Euler( cubeAngle ));
                            break;

                        case 4 :
                            cubePosVect.Set( ((x * 5.12f) - 40.96f), 1.8f, ((z * 5.12f)) );
                            Instantiate( p04L_TeeHall, cubePosVect, Quaternion.Euler( cubeAngle ));
                            break;

                        case 5 :
                            cubePosVect.Set( ((x * 5.12f) - 40.96f), 2.55f, ((z * 5.12f)) );
                            Instantiate( p05L_HallRoomR, cubePosVect, Quaternion.Euler( cubeAngle ));
                            props = UnityEngine.Random.Range(1,6);
                            switch( props ){
                            case 1 :
                                cubeAngle.Set( (cubeAngle.x+90.0f), cubeAngle.y, cubeAngle.z );
                                Instantiate( pWristShacklesSide, cubePosVect, Quaternion.Euler( cubeAngle ));
                                break;

                            case 2 :
                                cubeAngle.Set( (cubeAngle.x+90.0f), cubeAngle.y, cubeAngle.z );
                                Instantiate( pSpreadShacklesSide, cubePosVect, Quaternion.Euler( cubeAngle ));
                                break;

                            case 3 :
                                cubeAngle.Set( cubeAngle.x, cubeAngle.y, (cubeAngle.z-90));
                                cubePosVect.Set( cubePosVect.x, (cubePosVect.y+0.32f), cubePosVect.z );
                                Instantiate( pHangingCageSide, cubePosVect, Quaternion.Euler(cubeAngle));
                                break;

                            case 4 :
                                cubeAngle.Set( cubeAngle.x, cubeAngle.y, (cubeAngle.z-90.0f));
                                Instantiate( pPillorySide, cubePosVect, Quaternion.Euler(cubeAngle));
                                break;

                            default :
                                break;
                            }
                            break;

                        case 6 :
                            cubePosVect.Set( ((x * 5.12f) - 40.96f), 2.55f, ((z * 5.12f)) );
                            //-----------------------------------
                            // **WORKAROUND**
                            Vector3 fixAngle = new Vector3( cubeAngle.x, (cubeAngle.y - 90), cubeAngle.z);
                            // **End of workaround**
                            //-----------------------------------
                            Instantiate( p06L_HallRoomL, cubePosVect, Quaternion.Euler( fixAngle ));
                            props = UnityEngine.Random.Range(1,6);
                            switch( props ){
                            case 1 :
                                cubeAngle.Set( (cubeAngle.x+90.0f), cubeAngle.y, (cubeAngle.z+180.0f));
                                Instantiate( pWristShacklesSide, cubePosVect, Quaternion.Euler(cubeAngle));
                                break;

                            case 2 :
                                cubeAngle.Set( (cubeAngle.x+90.0f), cubeAngle.y, (cubeAngle.z+180.0f));
                                Instantiate( pSpreadShacklesSide, cubePosVect, Quaternion.Euler(cubeAngle));
                                break;

                            case 3 :
                                cubeAngle.Set( cubeAngle.x, cubeAngle.y, (cubeAngle.z+90.0f));
                                cubePosVect.Set( cubePosVect.x, (cubePosVect.y+0.32f), cubePosVect.z );
                                Instantiate( pHangingCageSide, cubePosVect, Quaternion.Euler(cubeAngle));
                                break;

                            case 4 :
                                cubeAngle.Set( cubeAngle.x, cubeAngle.y, (cubeAngle.z+90.0f));
                                Instantiate( pPillorySide, cubePosVect, Quaternion.Euler(cubeAngle));
                                break;

                            default :
                                break;
                            }
                            break;

                        case 7 :
                            cubePosVect.Set( ((x * 5.12f) - 40.96f), 2.55f, ((z * 5.12f)) );
                            Instantiate( p07L_HallRoomLR, cubePosVect, Quaternion.Euler( cubeAngle ));
                            break;

                        case 8 :
                            cubePosVect.Set( ((x * 5.12f) - 40.96f), 2.55f, ((z * 5.12f)) );
                            Instantiate( p08L_HallRoom, cubePosVect, Quaternion.Euler( cubeAngle ));
                            props = UnityEngine.Random.Range(1,4);
                            switch( props ){
                            case 1 :
                            case 2 :
                                cubePosVect.Set( cubePosVect.x, (cubePosVect.y+0.32f), cubePosVect.z );
                                Instantiate( pHangingCageSide, cubePosVect, Quaternion.Euler( cubeAngle ));
                                break;

                            default :
                                break;
                            }
                            break;

                        case 9 :
                            cubePosVect.Set( ((x * 5.12f) - 40.96f), 2.55f, ((z * 5.12f)) );
                            Instantiate( p09L_RoomAngle, cubePosVect, Quaternion.Euler( cubeAngle ));
```

```
                                props = UnityEngine.Random.Range(1,6);
                                switch( props ){
                                case 1 :
                                case 2 :
                                    cubeAngle.Set( (cubeAngle.x+90.0f), (cubeAngle.y+90.0f), cubeAngle.z);
                                    Instantiate( pJudasCradleCorner, cubePosVect, Quaternion.Euler(cubeAngle));
                                    break;

                                case 3 :
                                    cubeAngle.Set( (cubeAngle.x+90.0f), cubeAngle.y, cubeAngle.z );
                                    Instantiate( pTortureRackCorner, cubePosVect, Quaternion.Euler( cubeAngle ));
                                    break;

                                case 4 :
                                    cubeAngle.Set( cubeAngle.x, cubeAngle.y, (cubeAngle.z+90.0f));
                                    Instantiate( pPilloryCorner, cubePosVect, Quaternion.Euler(cubeAngle));
                                    break;

                                default :
                                    break;
                                }
                                break;

                            case 10 :
                                cubePosVect.Set( ((x * 5.12f) - 40.96f), 2.55f, ((z * 5.12f)) );
                                Instantiate( p10L_SideRoom, cubePosVect, Quaternion.Euler( cubeAngle ));
                                props = UnityEngine.Random.Range(1,8);
                                switch( props ){
                                case 1 :
                                    cubeAngle.Set( (cubeAngle.x+90.0f), cubeAngle.y, cubeAngle.z );
                                    Instantiate( pWristShacklesSide, cubePosVect, Quaternion.Euler( cubeAngle ));
                                    break;

                                case 2 :
                                    cubeAngle.Set( (cubeAngle.x+90.0f), cubeAngle.y, cubeAngle.z );
                                    Instantiate( pSpreadShacklesSide, cubePosVect, Quaternion.Euler( cubeAngle ));
                                    break;

                                case 3 :
                                    cubePosVect.Set( cubePosVect.x, (cubePosVect.y+0.32f), cubePosVect.z );
                                    Instantiate( pHangingCageSide, cubePosVect, Quaternion.Euler( cubeAngle ));
                                    break;

                                case 4 :
                                    cubeAngle.Set( (cubeAngle.x+90.0f), cubeAngle.y, cubeAngle.z );
                                    Instantiate( pTortureRackSide, cubePosVect, Quaternion.Euler( cubeAngle ));
                                    break;

                                case 5 :
                                    cubeAngle.Set( (cubeAngle.x+90.0f), cubeAngle.y, cubeAngle.z );
                                    Instantiate( pJudasCradleSide, cubePosVect, Quaternion.Euler( cubeAngle ));
                                    break;

                                case 6 :
                                    Instantiate( pPillorySide, cubePosVect, Quaternion.Euler(
    cubeAngle ));
                                    break;
                                }
                                break;

                            case 11 :
                                cubePosVect.Set( ((x * 5.12f) - 40.96f), 2.55f, ((z * 5.12f)) );
                                Instantiate( p11L_CornerRoom, cubePosVect, Quaternion.Euler( cubeAngle ));
                                props = UnityEngine.Random.Range(1,8);
                                switch( props ){
                                case 1 :
                                    cubeAngle.Set( (cubeAngle.x+90.0f), cubeAngle.y, cubeAngle.z );
                                    Instantiate( pWristShacklesCorner, cubePosVect, Quaternion.Euler( cubeAngle ));
                                    break;

                                case 2 :
                                    cubeAngle.Set( (cubeAngle.x+90.0f), cubeAngle.y, cubeAngle.z );
                                    Instantiate( pSpreadShacklesCorner, cubePosVect, Quaternion.Euler( cubeAngle ));
                                    break;

                                case 3 :
                                case 4 :
                                    cubeAngle.Set( (cubeAngle.x+90.0f), cubeAngle.y, cubeAngle.z );
                                    Instantiate( pTortureRackCorner, cubePosVect, Quaternion.Euler( cubeAngle ));
                                    break;

                                case 5 :
                                    cubeAngle.Set( (cubeAngle.x+90.0f), (cubeAngle.y+90.0f), cubeAngle.z );
                                    Instantiate( pJudasCradleCorner, cubePosVect, Quaternion.Euler(cubeAngle));
                                    break;

                                case 6 :
                                    cubeAngle.Set( cubeAngle.x, cubeAngle.y, (cubeAngle.z+90.0f));
                                    Instantiate( pPilloryCorner, cubePosVect, Quaternion.Euler(cubeAngle));
                                    break;
                                }
                                break;

                            case 12 :
                                cubePosVect.Set( ((x * 5.12f) - 40.96f), 2.55f, ((z * 5.12f)) );
                                Instantiate( p12L_OffsetRoom, cubePosVect, Quaternion.Euler( cubeAngle ));
                                break;

                            case 13 :
                                cubePosVect.Set( ((x * 5.12f) - 40.96f), 2.55f, ((z * 5.12f)) );
                                Instantiate( p13L_OpenRoom, cubePosVect, Quaternion.Euler( cubeAngle ));
                                props = UnityEngine.Random.Range(1,8);
                                switch( props ){
                                case 1 :
                                    cubeAngle.Set( (cubeAngle.x+90.0f), cubeAngle.y, cubeAngle.z );
                                    Instantiate( pWristShacklesQuad, cubePosVect, Quaternion.Euler( cubeAngle ));
                                    break;

                                case 2 :
                                case 3 :
                                    Instantiate( pPillorySide, cubePosVect, Quaternion.Euler(
    cubeAngle ));
                                    break;

                                case 4 :
                                    cubeAngle.Set( (cubeAngle.x+90.0f), cubeAngle.y, cubeAngle.z );
                                    Instantiate( pTortureRackSide, cubePosVect, Quaternion.Euler( cubeAngle ));
                                    break;

                                case 5 :
                                    cubeAngle.Set( (cubeAngle.x+90.0f), cubeAngle.y, cubeAngle.z );
                                    Instantiate( pJudasCradleSide, cubePosVect, Quaternion.Euler( cubeAngle ));
                                    break;

                                case 6 :
                                    cubePosVect.Set( cubePosVect.x, (cubePosVect.y+0.32f), cubePosVect.z );
                                    Instantiate( pHangingCageSide, cubePosVect, Quaternion.Euler( cubeAngle ));
                                    break;

                                default :
                                    break;
                                }
                                break;

                            case 14 :
                                cubePosVect.Set( ((x * 5.12f) - 40.96f), 1.8f, ((z * 5.12f)) );
                                Instantiate( p14L_EndHall, cubePosVect, Quaternion.Euler( cubeAngle ));
                                cubePosVect.Set( ((x * 5.12f) - 40.96f), -.64f, ((z * 5.12f)) );
                                Instantiate( pSquareGrate, cubePosVect, Quaternion.Euler( cubeAngle ));
                                break;

                            case 15 :
                                cubePosVect.Set( ((x * 5.12f) - 40.96f), 2.4f, ((z * 5.12f)) );
                                Instantiate( p17_SolidCube, cubePosVect, Quaternion.Euler( cubeAngle ));
                                break;

                            default : break;
                            }
                            break;

                        case 1 :
                            switch( newCube ){
                            case 1 :
                                cubePosVect.Set( ((x * 5.12f) - 40.96f), 1.8f, ((z * 5.12f)) );
                                Instantiate( p01H_StraightHall, cubePosVect, Quaternion.Euler( cubeAngle ));
                                break;

                            case 2 :
                                cubePosVect.Set( ((x * 5.12f) - 40.96f), 1.8f, ((z * 5.12f)) );
                                Instantiate( p02H_CornerHall, cubePosVect, Quaternion.Euler( cubeAngle ));
                                break;

                            case 3 :
                                cubePosVect.Set( ((x * 5.12f) - 40.96f), 1.8f, ((z * 5.12f)) );
                                Instantiate( p03H_CrossHall, cubePosVect, Quaternion.Euler( cubeAngle ));
                                break;

```

```
                            case 4 :
                                cubePosVect.Set( ((x * 5.12f) - 40.96f), 1.8f, ((z * 5.12f)) );
                                Instantiate( p04H_TeeHall, cubePosVect, Quaternion.Euler( cubeAngle ));
                                break;

                            case 5 :
                                cubePosVect.Set( ((x * 5.12f) - 40.96f), 2.55f, ((z * 5.12f)) );
                                Instantiate( p05L_HallRoomR, cubePosVect, Quaternion.Euler( cubeAngle ));
                                props = UnityEngine.Random.Range(1,6);
                                switch( props ){
                                case 1 :
                                    cubeAngle.Set( (cubeAngle.x+90.0f), cubeAngle.y, cubeAngle.z );
                                    Instantiate( pWristShacklesSide, cubePosVect, Quaternion.Euler( cubeAngle ));
                                    break;

                                case 2 :
                                    cubeAngle.Set( (cubeAngle.x+90.0f), cubeAngle.y, cubeAngle.z );
                                    Instantiate( pSpreadShacklesSide, cubePosVect, Quaternion.Euler( cubeAngle ));
                                    break;

                                case 3 :
                                    cubeAngle.Set( cubeAngle.x, cubeAngle.y, (cubeAngle.z-90.0f));
                                    cubePosVect.Set( cubePosVect.x, (cubePosVect.y+0.32f), cubePosVect.z );
                                    Instantiate( pHangingCageSide, cubePosVect, Quaternion.Euler(cubeAngle));
                                    break;

                                case 4 :
                                    cubeAngle.Set( cubeAngle.x, cubeAngle.y, (cubeAngle.z-90.0f));
                                    Instantiate( pPillorySide, cubePosVect, Quaternion.Euler(cubeAngle));
                                    break;

                                default :
                                    break;
                                }
                                break;

                            case 6 :
                                cubePosVect.Set( ((x * 5.12f) - 40.96f), 2.55f, ((z * 5.12f)) );
                                //-------------------------------------
                                // **WORKAROUND**
                                Vector3 fixAngle = new Vector3( cubeAngle.x, (cubeAngle.y - 90), cubeAngle.z);
                                // **End of workaround**
                                //-------------------------------------
                                Instantiate( p06L_HallRoomL, cubePosVect, Quaternion.Euler( fixAngle ));
                                props = UnityEngine.Random.Range(1,6);
                                switch( props ){
                                case 1 :
                                    cubeAngle.Set( (cubeAngle.x+90.0f), cubeAngle.y, (cubeAngle.z+180.0f));
                                    Instantiate( pWristShacklesSide, cubePosVect, Quaternion.Euler(cubeAngle));
                                    break;

                                case 2 :
                                    cubeAngle.Set( (cubeAngle.x+90.0f), cubeAngle.y, (cubeAngle.z+180.0f));
                                    Instantiate( pSpreadShacklesSide, cubePosVect, Quaternion.Euler
    (cubeAngle));
                                    break;

                                case 3 :
                                    cubeAngle.Set( cubeAngle.x, cubeAngle.y, (cubeAngle.z+90.0f));
                                    cubePosVect.Set( cubePosVect.x, (cubePosVect.y+0.32f), cubePosVect.z );
                                    Instantiate( pHangingCageSide, cubePosVect, Quaternion.Euler(cubeAngle));
                                    break;

                                case 4 :
                                    cubeAngle.Set( cubeAngle.x, cubeAngle.y, (cubeAngle.z+90.0f));
                                    Instantiate( pPillorySide, cubePosVect, Quaternion.Euler(cubeAngle));
                                    break;

                                default:
                                    break;
                                }
                                break;

                            case 7 :
                                cubePosVect.Set( ((x * 5.12f) - 40.96f), 2.55f, ((z * 5.12f)) );
                                Instantiate( p07L_HallRoomLR, cubePosVect, Quaternion.Euler( cubeAngle ));
                                break;

                            case 8 :
                                cubePosVect.Set( ((x * 5.12f) - 40.96f), 2.55f, ((z * 5.12f)) );
                                Instantiate( p08L_HallRoom, cubePosVect, Quaternion.Euler( cubeAngle ));
                                props = UnityEngine.Random.Range(1,4);
                                switch( props ){
                                case 1 :
                                case 2 :
                                    cubePosVect.Set( cubePosVect.x, (cubePosVect.y+0.32f), cubePosVect.z );
                                    Instantiate( pHangingCageSide, cubePosVect, Quaternion.Euler( cubeAngle ));
                                    break;

                                default :
                                    break;
                                }
                                break;

                            case 9 :
                                cubePosVect.Set( ((x * 5.12f) - 40.96f), 2.55f, ((z * 5.12f)) );
                                Instantiate( p09L_RoomAngle, cubePosVect, Quaternion.Euler( cubeAngle ));
                                props = UnityEngine.Random.Range(1,6);
                                switch( props ){
                                case 1 :
                                case 2 :
                                    cubeAngle.Set( (cubeAngle.x+90.0f), (cubeAngle.y+90.0f), cubeAngle.z);
                                    Instantiate( pJudasCradleCorner, cubePosVect, Quaternion.Euler(cubeAngle));
                                    break;

                                case 3 :
                                    cubeAngle.Set( (cubeAngle.x+90.0f), cubeAngle.y, cubeAngle.z );
                                    Instantiate( pTortureRackCorner, cubePosVect, Quaternion.Euler( cubeAngle ));
                                    break;
```

```

                            case 4 :
                                cubeAngle.Set( cubeAngle.x, cubeAngle.y, (cubeAngle.z+90.0f));
                                Instantiate( pPilloryCorner, cubePosVect, Quaternion.Euler(cubeAngle));
                                break;

                            default :
                                break;
                            }
                            break;

                        case 10 :
                            cubePosVect.Set( ((x * 5.12f) - 40.96f), 2.55f, ((z * 5.12f)) );
                            Instantiate( p10L_SideRoom, cubePosVect, Quaternion.Euler( cubeAngle ));
                            props = UnityEngine.Random.Range(1,8);
                            switch( props ){
                            case 1 :
                                cubeAngle.Set( (cubeAngle.x+90.0f), cubeAngle.y, cubeAngle.z);
                                Instantiate( pWristShacklesSide, cubePosVect, Quaternion.Euler( cubeAngle ));
                                break;

                            case 2 :
                                cubeAngle.Set( (cubeAngle.x+90.0f), cubeAngle.y, cubeAngle.z );
                                Instantiate( pSpreadShacklesSide, cubePosVect, Quaternion.Euler( cubeAngle ));
                                break;

                            case 3 :
                                cubePosVect.Set( cubePosVect.x, (cubePosVect.y+0.32f), cubePosVect.z );
                                Instantiate( pHangingCageSide, cubePosVect, Quaternion.Euler( cubeAngle ));
                                break;

                            case 4 :
                                cubeAngle.Set( (cubeAngle.x+90.0f), cubeAngle.y, cubeAngle.z );
                                Instantiate( pTortureRackSide, cubePosVect, Quaternion.Euler( cubeAngle ));
                                break;

                            case 5 :
                                cubeAngle.Set( (cubeAngle.x+90.0f), cubeAngle.y, cubeAngle.z );
                                Instantiate( pJudasCradleSide, cubePosVect, Quaternion.Euler( cubeAngle ));
                                break;

                            case 6 :
                                Instantiate( pPillorySide, cubePosVect, Quaternion.Euler( cubeAngle ));
                                break;
                            }
                            break;

                        case 11 :
                            cubePosVect.Set( ((x * 5.12f) - 40.96f), 2.55f, ((z * 5.12f)) );
                            Instantiate( p11L_CornerRoom, cubePosVect, Quaternion.Euler( cubeAngle ));
                            props = UnityEngine.Random.Range(1,8);
                            switch( props ){
                            case 1 :
                                cubeAngle.Set( (cubeAngle.x+90.0f), cubeAngle.y, cubeAngle.z );
                                Instantiate( pWristShacklesCorner, cubePosVect, Quaternion.Euler( cubeAngle ));
```

```
                                    break;

                                case 2 :
                                    cubeAngle.Set( (cubeAngle.x+90.0f), cubeAngle.y, cubeAngle.z );
                                    Instantiate( pSpreadShacklesCorner, cubePosVect, Quaternion.Euler( cubeAngle ));
                                    break;

                                case 3 :
                                case 4 :
                                    cubeAngle.Set( (cubeAngle.x+90.0f), cubeAngle.y, cubeAngle.z );
                                    Instantiate( pTortureRackCorner, cubePosVect, Quaternion.Euler( cubeAngle ));
                                    break;

                                case 5 :
                                    cubeAngle.Set( (cubeAngle.x+90.0f), (cubeAngle.y+90.0f), cubeAngle.z);
                                    Instantiate( pJudasCradleCorner, cubePosVect, Quaternion.Euler(cubeAngle));
                                    break;

                                case 6 :
                                    cubeAngle.Set( cubeAngle.x, cubeAngle.y, (cubeAngle.z+90.0f));
                                    Instantiate( pPilloryCorner, cubePosVect, Quaternion.Euler(cubeAngle));
                                    break;
                                }
                                break;

                            case 12 :
                                cubePosVect.Set( ((x * 5.12f) - 40.96f), 2.55f, ((z * 5.12f)) );
                                Instantiate( p12L_OffsetRoom, cubePosVect, Quaternion.Euler( cubeAngle ));


                                break;

                            case 13 :
                                cubePosVect.Set( ((x * 5.12f) - 40.96f), 2.55f, ((z * 5.12f)) );
                                Instantiate( p13L_OpenRoom, cubePosVect, Quaternion.Euler( cubeAngle ));
                                props = UnityEngine.Random.Range(1,8);
                                switch( props ){
                                case 1 :
                                    cubeAngle.Set( (cubeAngle.x+90.0f), cubeAngle.y, cubeAngle.z );
                                    Instantiate( pWristShacklesQuad, cubePosVect, Quaternion.Euler( cubeAngle ));
                                    break;

                                case 2 :
                                case 3 :
                                    Instantiate( pPillorySide, cubePosVect, Quaternion.Euler( cubeAngle ));
                                    break;

                                case 4 :
                                    cubeAngle.Set( (cubeAngle.x+90.0f), cubeAngle.y, cubeAngle.z );
                                    Instantiate( pTortureRackSide, cubePosVect, Quaternion.Euler( cubeAngle ));
                                    break;

                                case 5 :
                                    cubeAngle.Set( (cubeAngle.x+90.0f), cubeAngle.y, cubeAngle.z );
                                    Instantiate( pJudasCradleSide, cubePosVect, Quaternion.Euler( cubeAngle ));
                                    break;

                                case 6 :
                                    cubePosVect.Set( cubePosVect.x, (cubePosVect.y+0.32f), cubePosVect.z );
                                    Instantiate( pHangingCageSide, cubePosVect, Quaternion.Euler( cubeAngle ));
                                    break;

                                default :
                                    break;
                                }
                                break;

                            case 14 :
                                cubePosVect.Set( ((x * 5.12f) - 40.96f), 1.8f, ((z * 5.12f)) );
                                Instantiate( p14H_EndHall, cubePosVect, Quaternion.Euler( cubeAngle ));
                                cubePosVect.Set( ((x * 5.12f) - 40.96f), 0.0f, ((z * 5.12f)) );
                                Instantiate( pSquareGrate, cubePosVect, Quaternion.Euler( cubeAngle ));
                                break;

                            case 15 :
                                cubePosVect.Set( ((x * 5.12f) - 40.96f), 2.4f, ((z * 5.12f)) );
                                Instantiate( p17_SolidCube, cubePosVect, Quaternion.Euler( cubeAngle ));
                                break;

                            default :
                                break;
                            }
                            break;
                        }
                    }
                }
            }
        }
    }

    /*
---------------------------------------------------------------------
    Spawn Enemies and Pickups
--------------------------------------------------------------------- */
    void SpawnStuff( ref MAPCUBE[,] MapArray ){
        Vector3 spawnPos = new Vector3();
        Vector3 spawnAngle = new Vector3();

        for( int spawns = 0; spawns < 3; ++spawns ){
            int x = UnityEngine.Random.Range(1, mapWidth);
            int z = UnityEngine.Random.Range(1, mapLength);

            if( MapArray[x,z].cube != -1 ){
                float yMod = 0.0f;
                if( (MapArray[x,z].cube == 1) ||
                   (MapArray[x,z].cube == 2) ||
                   (MapArray[x,z].cube == 3) ||
                   (MapArray[x,z].cube == 4) ||
                   (MapArray[x,z].cube == 14)  ) yMod = 1.75f;
                else yMod = 2.5f;
                spawnPos = new Vector3( ((x * 5.12f) - 40.96f), yMod, ((z * 5.12f)) );
                spawnAngle = new Vector3( 0, 180, 0 );
                Instantiate( Lantern, spawnPos, Quaternion.Euler( spawnAngle ));
                Instantiate( LanternCore, spawnPos, Quaternion.Euler( spawnAngle ));

                if( spawnMonsters ){
                    if( !((x == ((mapWidth / 2) - 1) && z == 1) ||
                          (x == ((mapWidth / 2) - 1) && z == 2) ||
                          (x == ((mapWidth / 2) + (mapWidth % 2) + 1) && z == (mapLength - 2)) ||
                          (x == ((mapWidth / 2) + (mapWidth % 2) + 1) && z == (mapLength - 1))) ){

                        spawnPos = new Vector3( ((x * 5.12f) - 40.96f), 0.5f, ((z * 5.12f)) );
                        spawnAngle = new Vector3( 0, 180, 0 );
                        Instantiate( Spectre, spawnPos, Quaternion.Euler( spawnAngle ));
                    }
                }
            }
        }
    }
}
```