# General Setup

For the game to run smoothly, certain standards will need to be adhered to in levels.

* **Object positioning**: although it’s a 2D game, every object will still occupy a position in 3D space. This must be used carefully to ensure that everything renders correctly and collides properly. **To this end, all objects should start a Z position of 10.** From here, objects can be moved by moderate increments- preferably in steps of 1.
* **Layering**: Unity “layers” will be used to improve collision logic in the game. Layers in Unity are simply a way of specifying how objects collide with each other, for example, having a layer “A” and a layer “B”, you can choose whether or not objects in layer A collide with those in B. **In Partners in Crime, Player Characters and NPCs will occupy the “Characters” layer, and all static objects (e.g. level geometry, crates etc) will occupy the “Scenery” layer.** This will be used in the game code for certain key features such as Line-of-sight detection.

# Robot Player

The Robot Player prefab should be placed in every scene (even for testing) as it has a camera object that will automatically follow it.

It contains a RobotController component that can be used to control its movement:

* **Speed:** Used to set the main horizontal movement speed.
* **Jump Strength:** How much force the robot has when it jumps. Increase to allow it to jump higher
* **Gravity Factor:** This accelerates how gravity gets applied to the robot, but not his maximum fall speed. Increasing this should make the robot feel heavier

# Ai Player

Contains component Ai Controller. Set ‘Occupied Gadget’ to first device that ai player will possess (the robot player). Check ‘Debug Controls’ to enable using the mouse to jump the ai around instead of a controller.  
Any object that the ai can possess must have a component that implements the interface GadgetControllerInterface.

To interact with gadgets, the player can either use the ‘B’ button on the controller (or the space bar) or can send a direction command with the right analogue stick (or wsad) to gadgets that support it.

# Door

Ai player may open and close the door with the action button. Door will move upwards two tile blocks when opened. The door will close after a small amount of time if it is not currently possessed by the ai and the robot player is not underneath it.

# Security Camera

Ai player may temporarily disable the camera by pressing the action button. It also implement the FieldOfView component (see guards). The camera will turn back on after a small amount of time if not currently possessed by the AI player.

The securityCamera component has three variables, startRotation, endRotation and rotationSpeed. StartRotation and endRotation define the angles at which the camera will pan between and will override the rotation variable in FieldOfView. RotationSpeed defines how fast the camera pans between these two angles.

# Guards

Guards can be placed from the Robot Guard prefab. The important components for the Guard are the GuardController and FieldOfView.

The GuardController currently only requires a handle to a PatrolPath (covered later). If no PatrolPath is specified, the Guard will simply walk back and forth until he reaches a wall or ledge.

The FieldOfView component has the following properties:

* **Field Of View** (ok, maybe I should have named this property better): the angular width of the line of sight, e.g. an arc of 90 degrees
* **Range:** How far their line of sight extends
* **Material:** This is used purely for drawing the field of view, if unsure just use the Material in Art Assets -> Materials -> Colours
* **Starting Rotation:** Use to set the initial direction (in degrees), especially useful for cameras

# Patrol Paths

Guards must be assigned Patrol Paths to follow (otherwise they will just stand around). These are fairly simple (if not a little tedious currently) to set up.

To create a Patrol Path, first drop a Patrol Path prefab into the level. Its position isn’t especially relevant, but best to place it roughly where your path will be lying- and make sure it is correctly placed on the Z-axis (at 10 on the Z-axis, preferably). Then, to create waypoints on the path, select the Patrol Path object and click “Add Waypoint” to add as many waypoints as you think you need. Then, position each of the waypoints as you see fit- take note of the sequence of waypoints, such that the order is what you expect.

Note that each Patrol Path Waypoint has a component that allows you to change its position in the sequence of waypoints. However, changing one does not change others, so if you’re updating or removing waypoints, remember to make sure that a proper sequence is conserved, else you may get errors (but you should also get console messages that tell you what you’ve done wrong). Editing the positioning for each waypoint is likely to be tedious if done at length, but for now there is no time to implement a more user-friendly positioning system.

Once a Patrol Path has been set up, just remember to assign it to any Guards you want to follow it!

# Guard Spawner

This object will spawn a new guard at the location of the spawner a number of specified times when triggered by an alarm. It will spawn a copy of an object in the guard prefab parameter. This should be set to a prefab of a guard object, new prefabs should be created for guards with unique patrol paths etc. The guard will be spawned 'number' amount of times at a regular interval.