THE GAME ENGINE

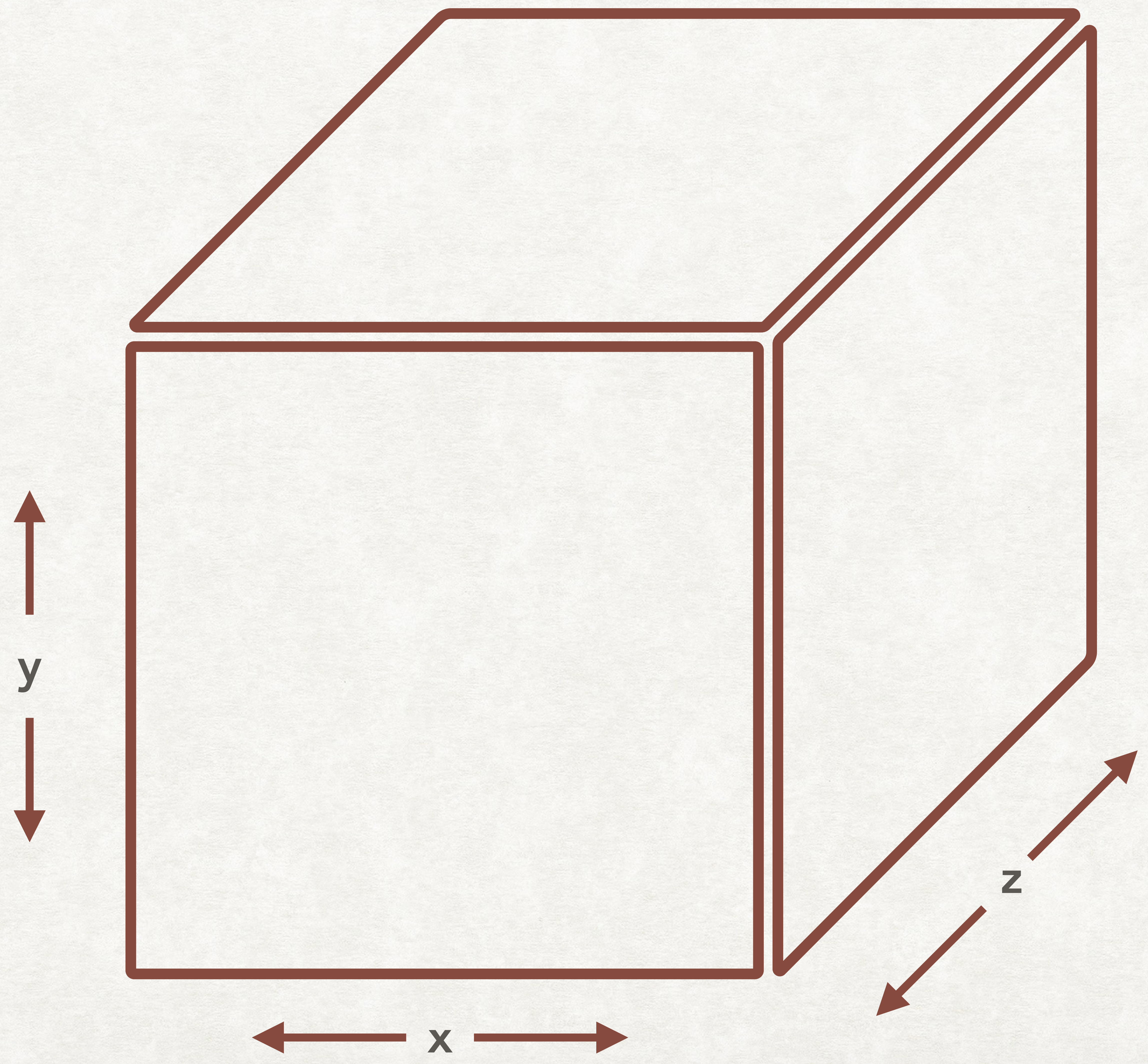
THE 3D GAME WORLD

The game world is represented by a three dimensional array of cubes.

world[x][y][z]

The world is 100 units in the x and z axis (forward/backward and left/right).

It is 50 units in the y axis (up/down).

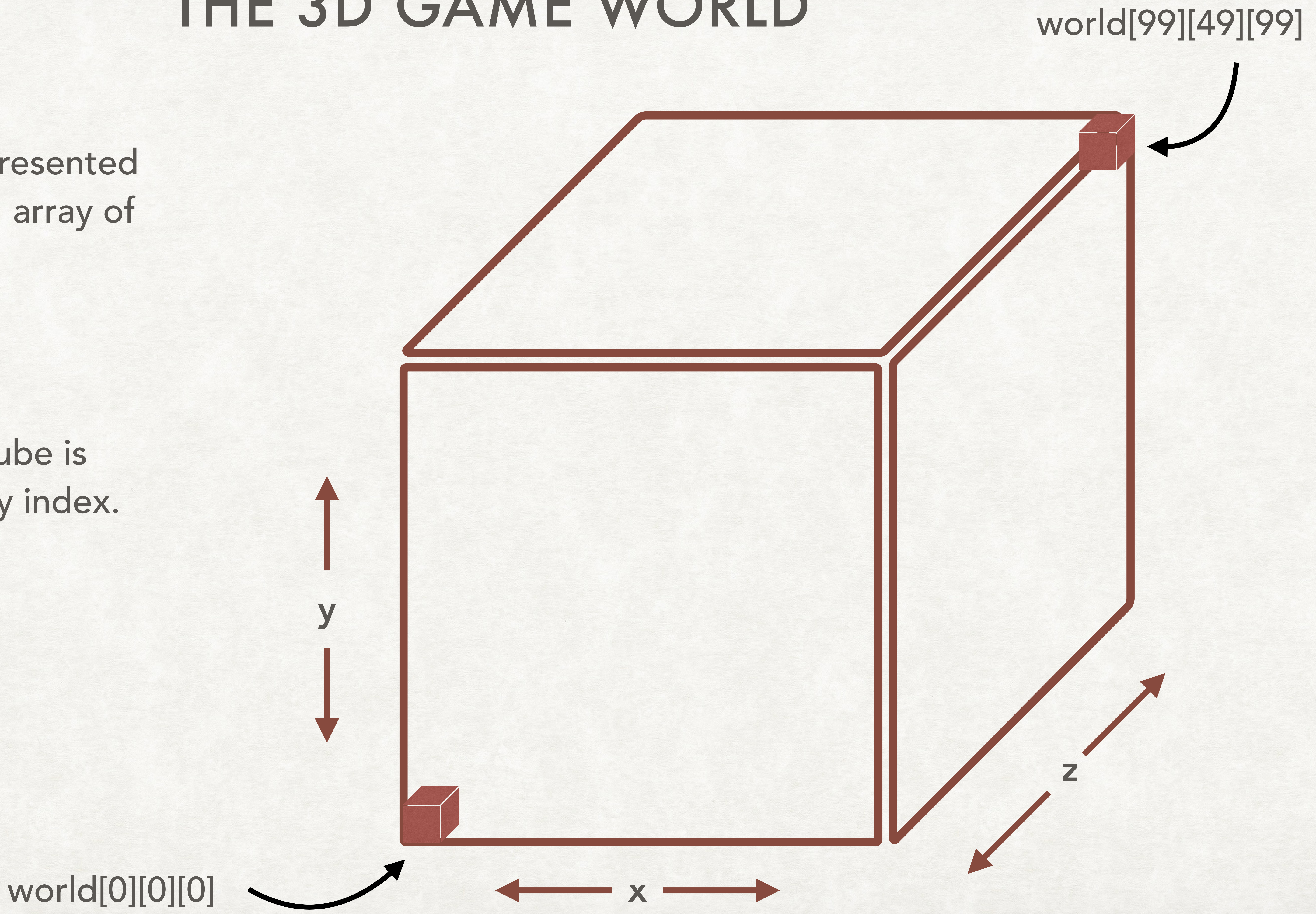


THE 3D GAME WORLD

The game world is represented by a three dimensional array of cubes.

world[x][y][z]

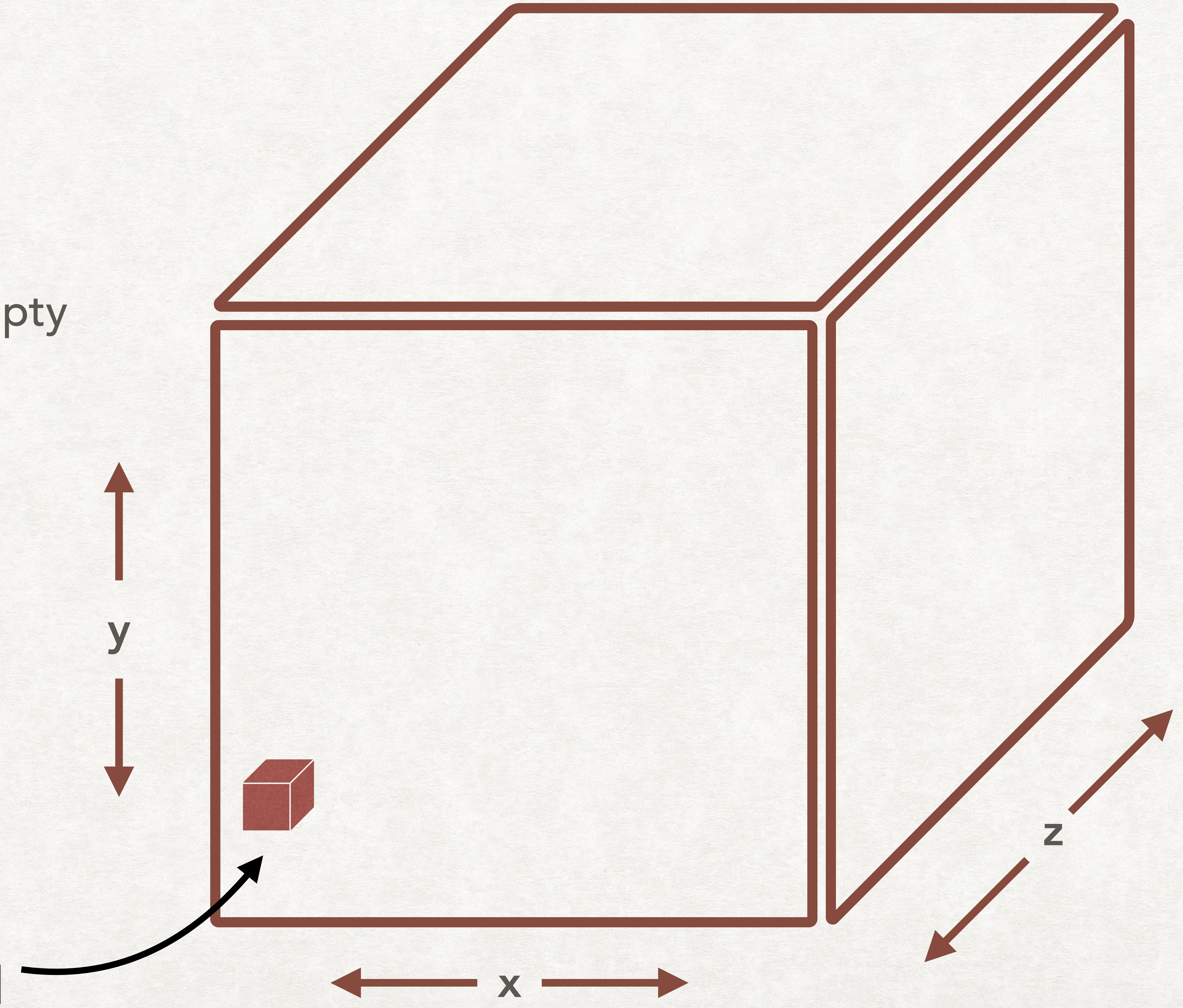
The position of each cube is determined by its array index.



THE 3D GAME WORLD

The value stored in the world array determines if a cube is drawn in that position and what colour it will be.

world[1][7][30] =0	the location is empty
world[1][7][30]=1	the cube is green
world[1][7]30] = 2	the cube is blue



COLLISIONS

The world created out of cubes makes it easy to detect collisions between objects.

A collision occurs when an object tries to enter a cube that is not empty (when it's world array value is not equal to zero).

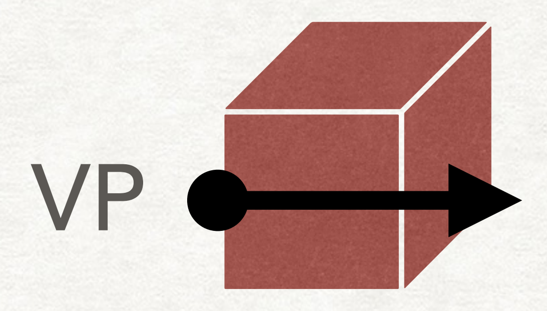
The player or viewpoint should not be allowed to enter a space which contains a cube (array value <> 0).

VP is the viewpoint. The direction which the player is looking. The viewpoint is the dot and the arrow indicates the viewing direction.

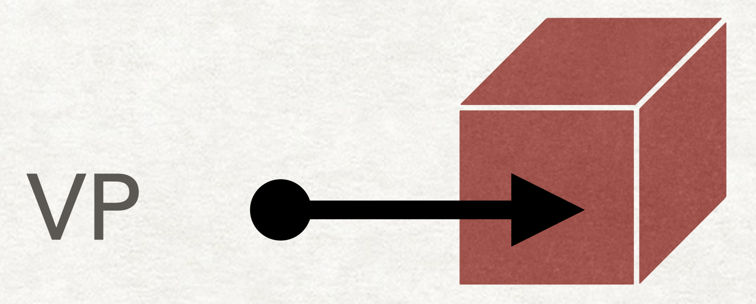
COLLISIONS

If the viewpoint tries to move into a space containing a filled square then it should be stopped from doing so.

The user controls the viewpoint using the mouse and keyboard. The system prevents the viewpoint from moving into occupied cubes.



VP collides with a cube



VP pushed back out of cube

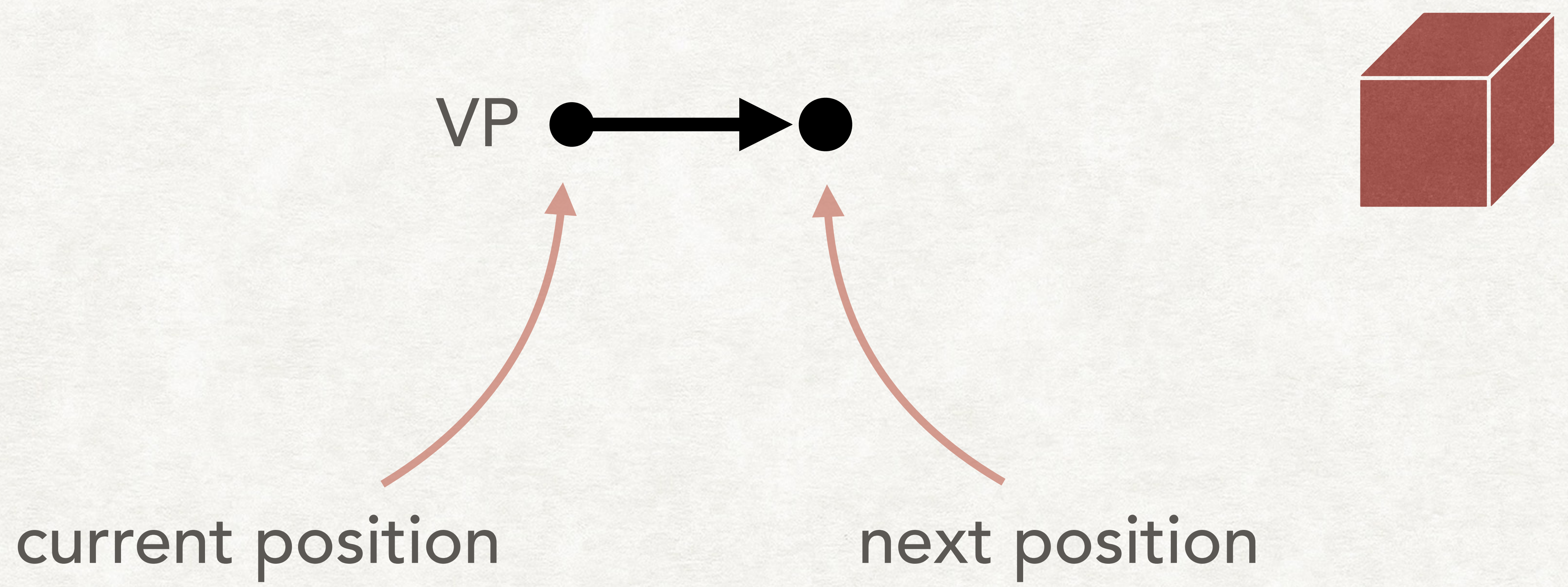
COLLISIONS

Functions in the game engine that let you control the motion of the viewpoint:

getViewPosition()	-**return** where the VP will move **next**
getOldViewPosition()	-**return** the **current** VP position
setViewPosition()	-**set** the position where the VP will move **next**

COLLISIONS

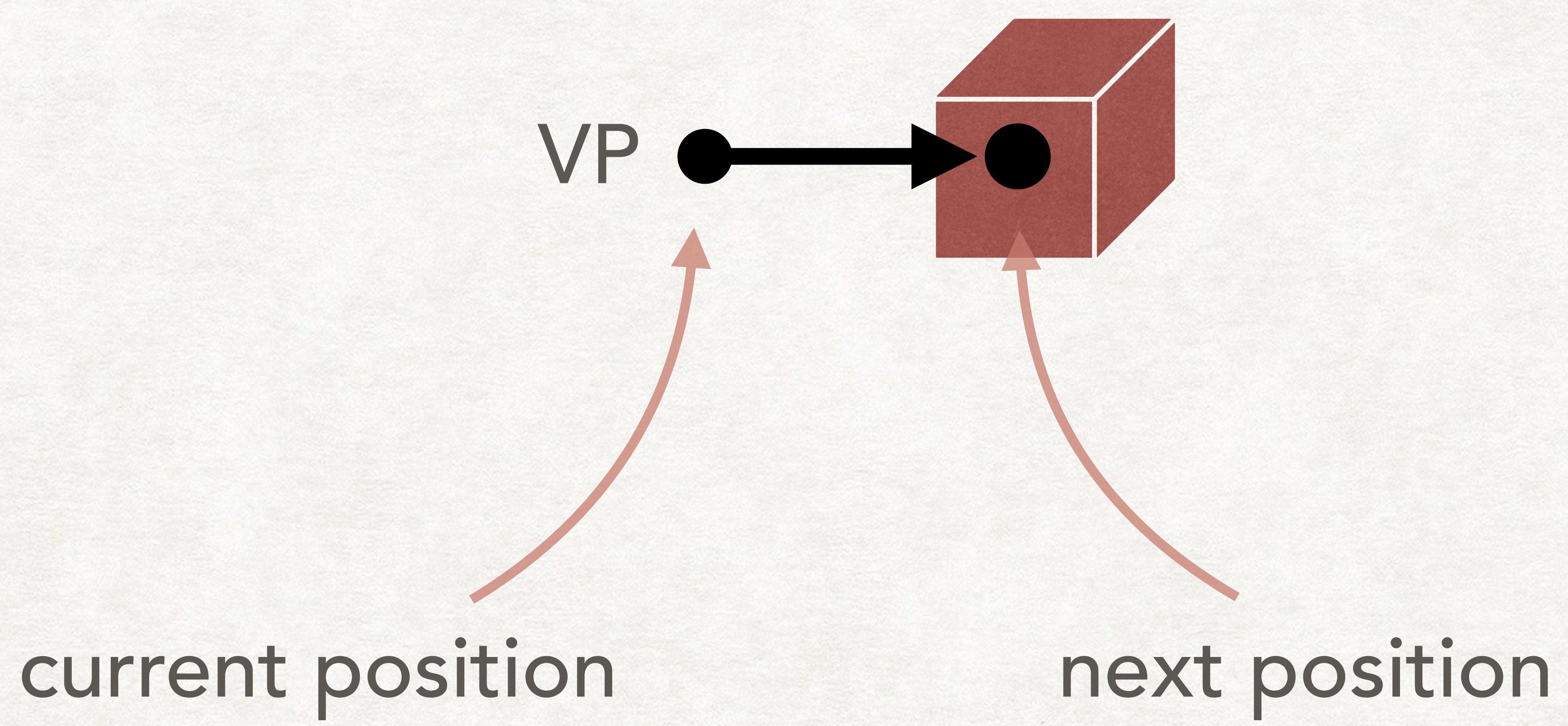
Example 1



No Collision
The next position obtained from getViewPosition() does not enter the object so you can do nothing.

COLLISIONS

Example 2

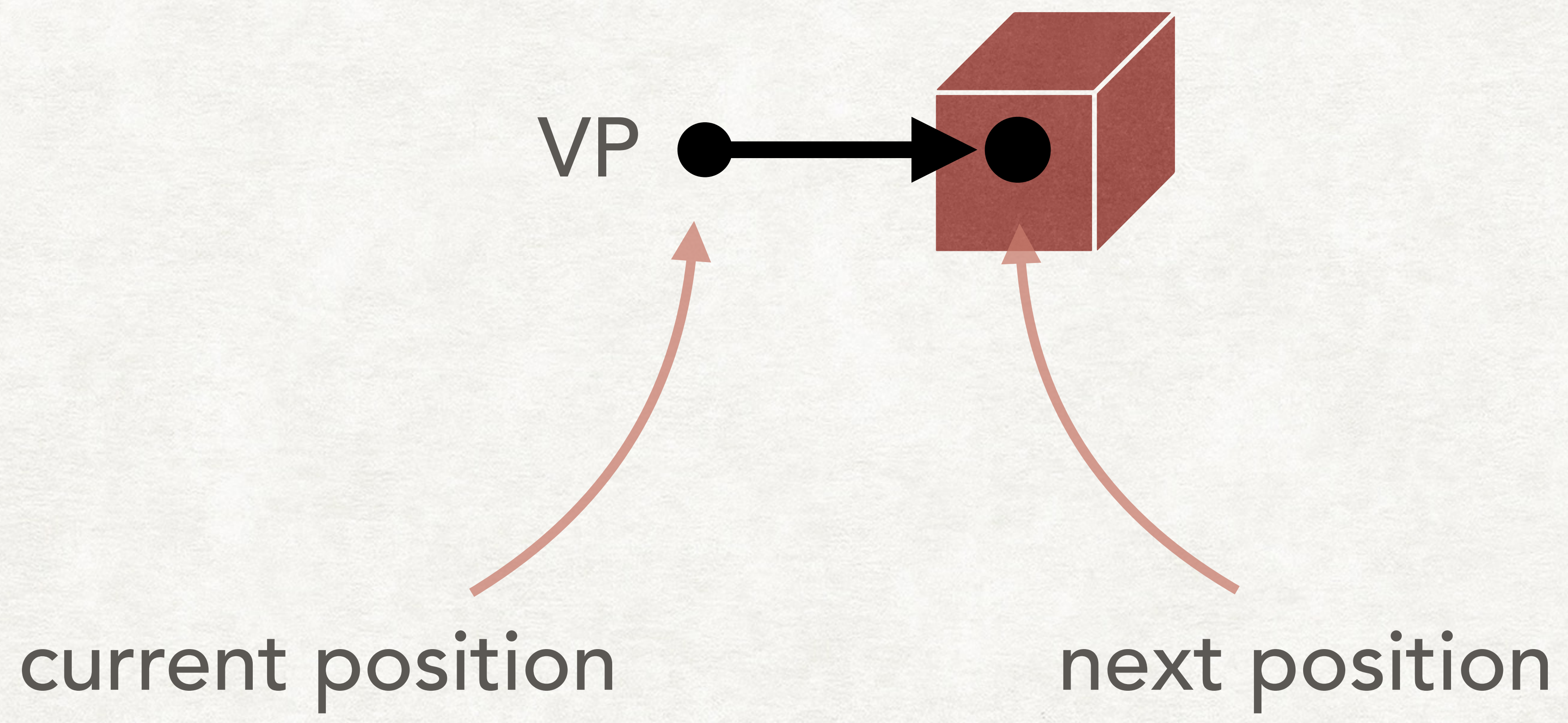


Collision

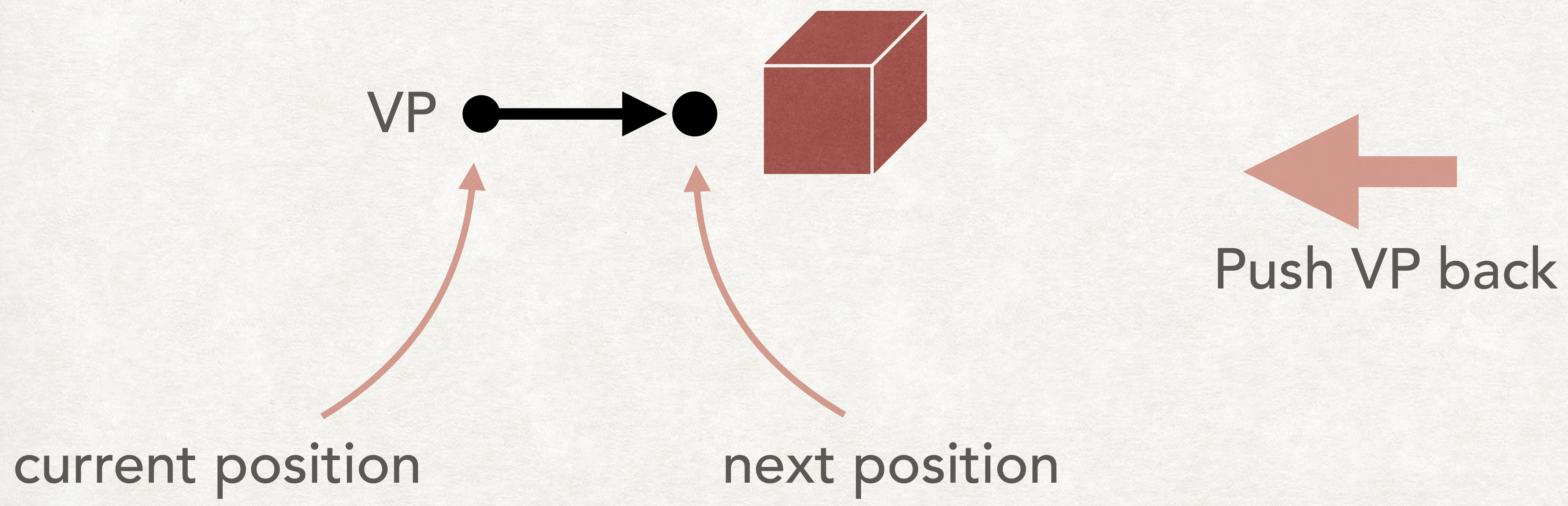
The next position obtained from getViewPosition() is inside an occupied space.

COLLISIONS

Example 2



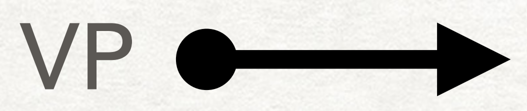
Collision
The next position obtained from getViewPosition() is inside an occupied space.



Use setViewPosition() to move the next position outside the cube. Set the view position equal to the old position which is obtained from getOldViewPosition().

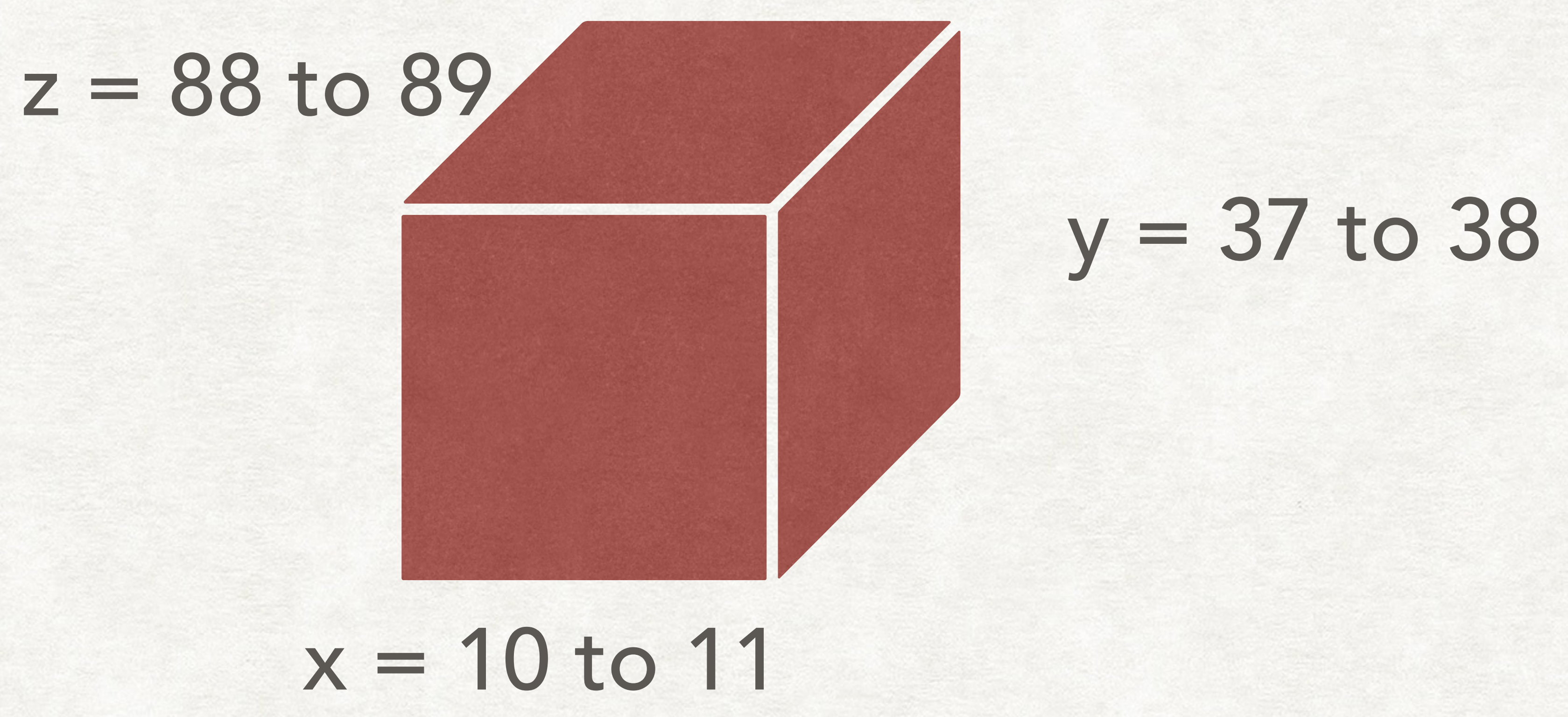
COLLISIONS

It is easy to determine if the viewpoint is moving inside a cube.
Compare the location of the viewpoint to a location in the world[][][] array.



(x,y,z)
= (10.2, 37.77, 88.21)

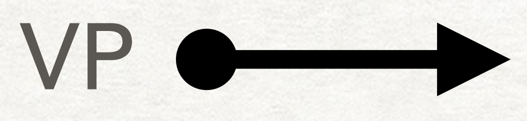
world[10][37][88]



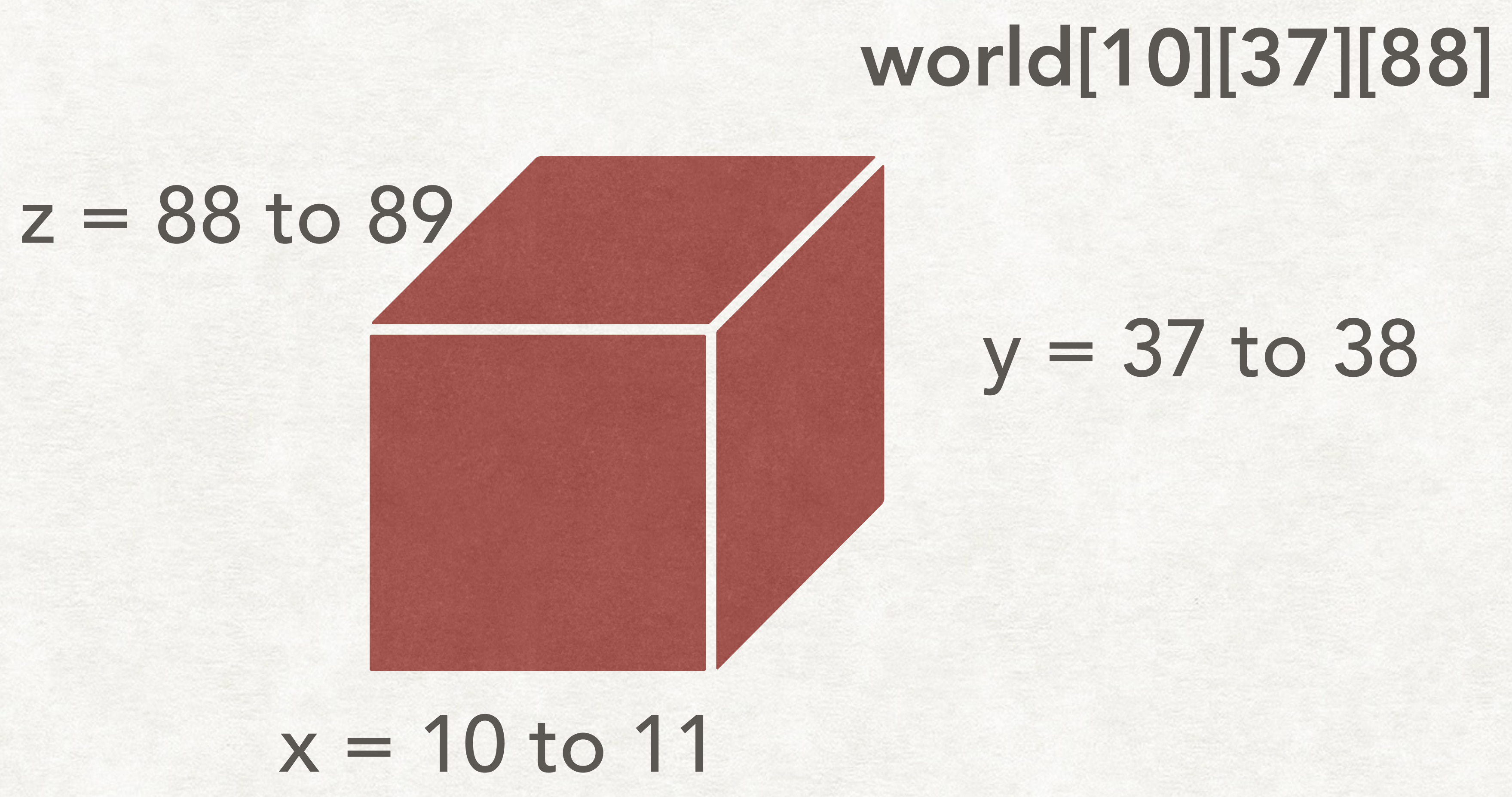
Cubes are one unit square. Use the array index as one corner of the cube and add one to each index to determine the end of that cube.

COLLISIONS

It is easy to determine if the viewpoint is moving inside a cube.
Compare the location of the viewpoint to a location in the world[][][] array.



(x,y,z)
= (10.2, 37.77, 88.21)

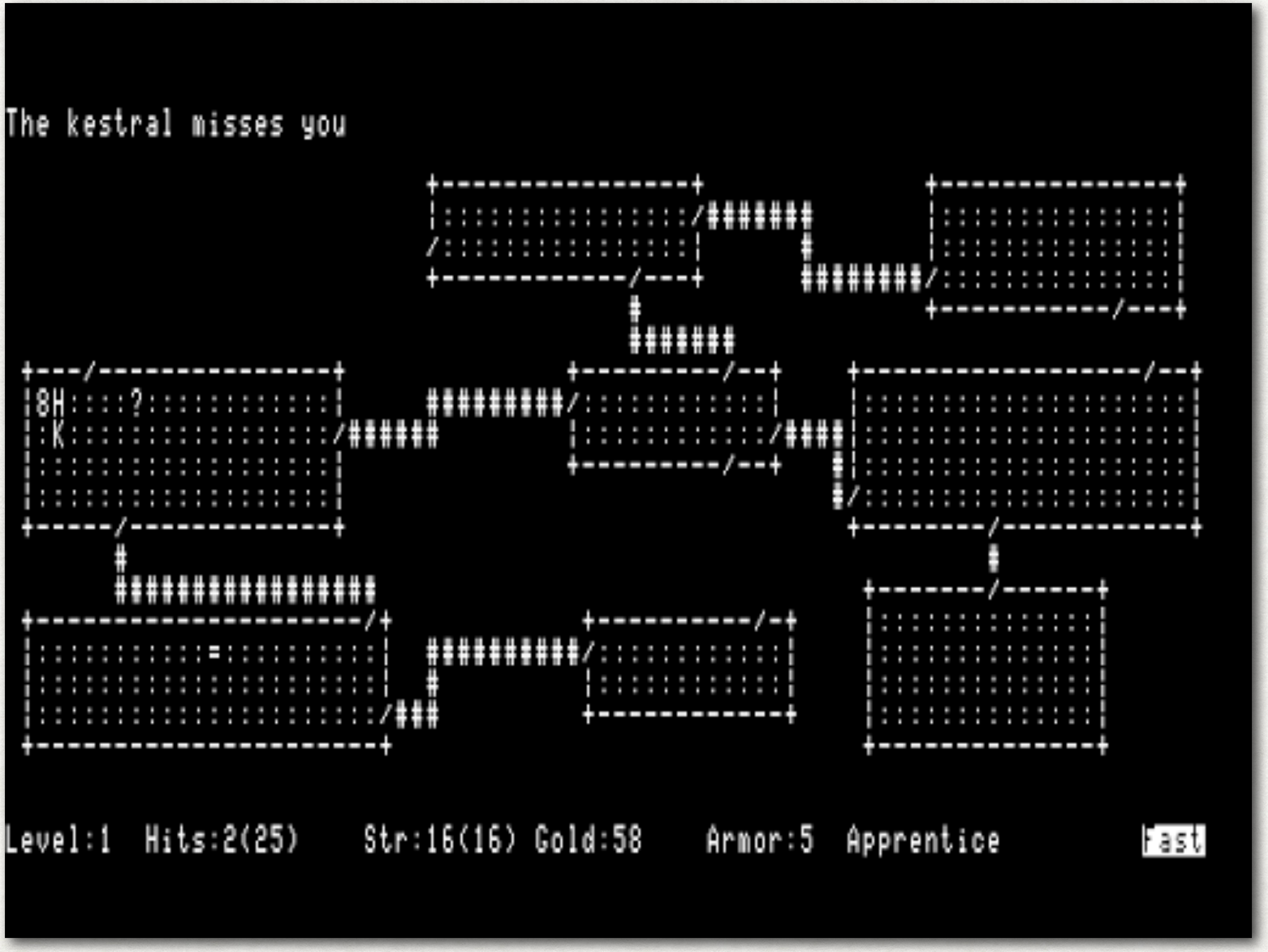


You can compare the viewpoint (x,y,z) position with the array indices to test if the VP is entering an occupied cube.
Is x=10.2 between 10 and 11?
Is y=37.77 between 37 and 38?
Is z=88.21 between 88 and 89?
All three are true to the viewpoint will collide with world[10][37][88] so you should stop that motion.

Cubes are one unit square. Use the array index as one corner of the cube and add one to each index to determine the end of that cube.

ASSIGNMENT 1

THE ORIGINAL ROGUE



```
The kestral misses you
```

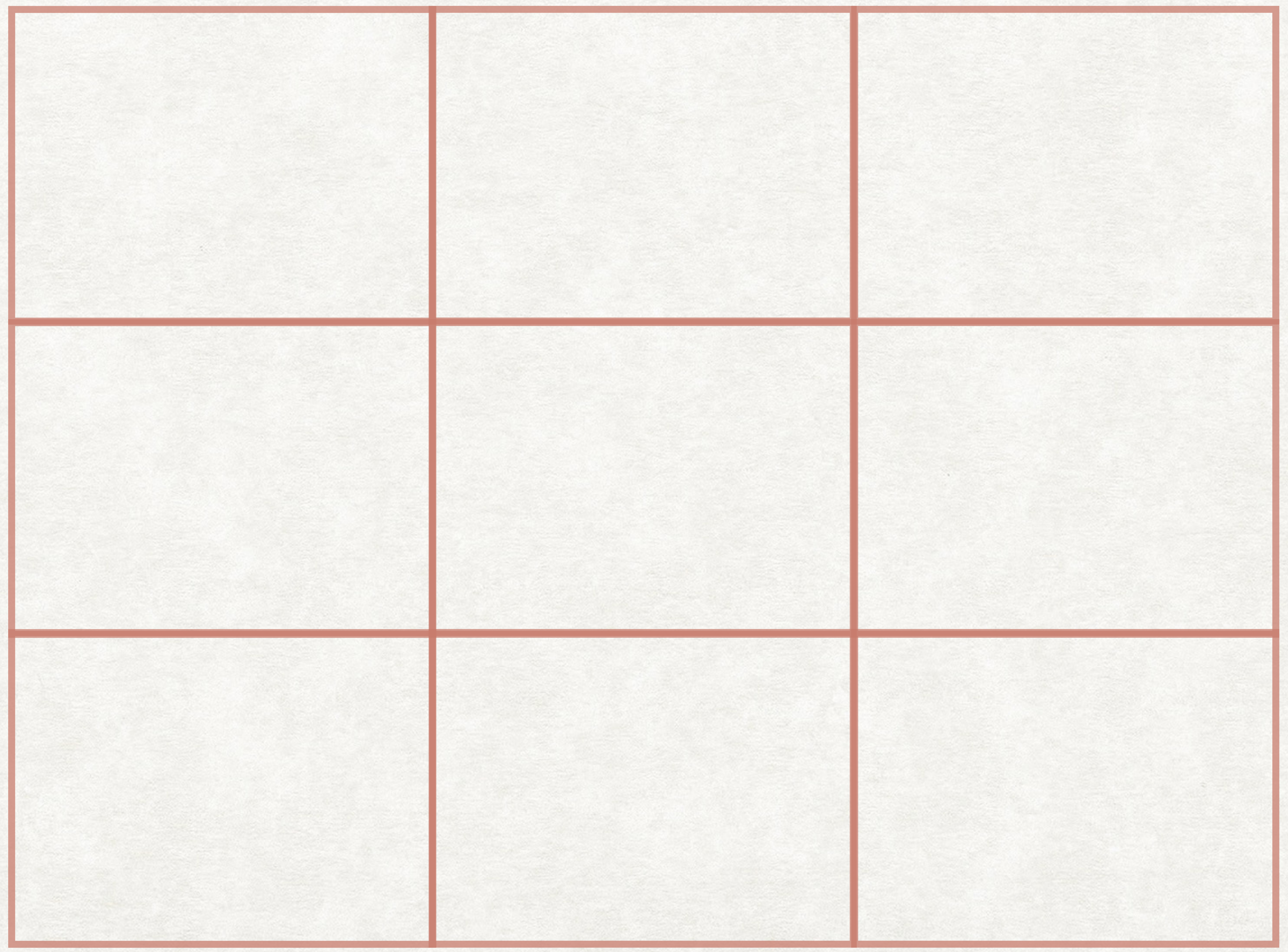
```
Level:1  Hits:2(25)    Str:16(16) Gold:58     Armor:5  Apprentice              Fast
```

Monochrome!
Rooms
Hallways
Multiple Levels (Stairs)
Monsters
Dynamically Created
Dungeon Crawl
Monster Path Finding
Eventually a Little Dog
Hidden Doors
Loot
Unforgiving

BUILDING A LEVEL

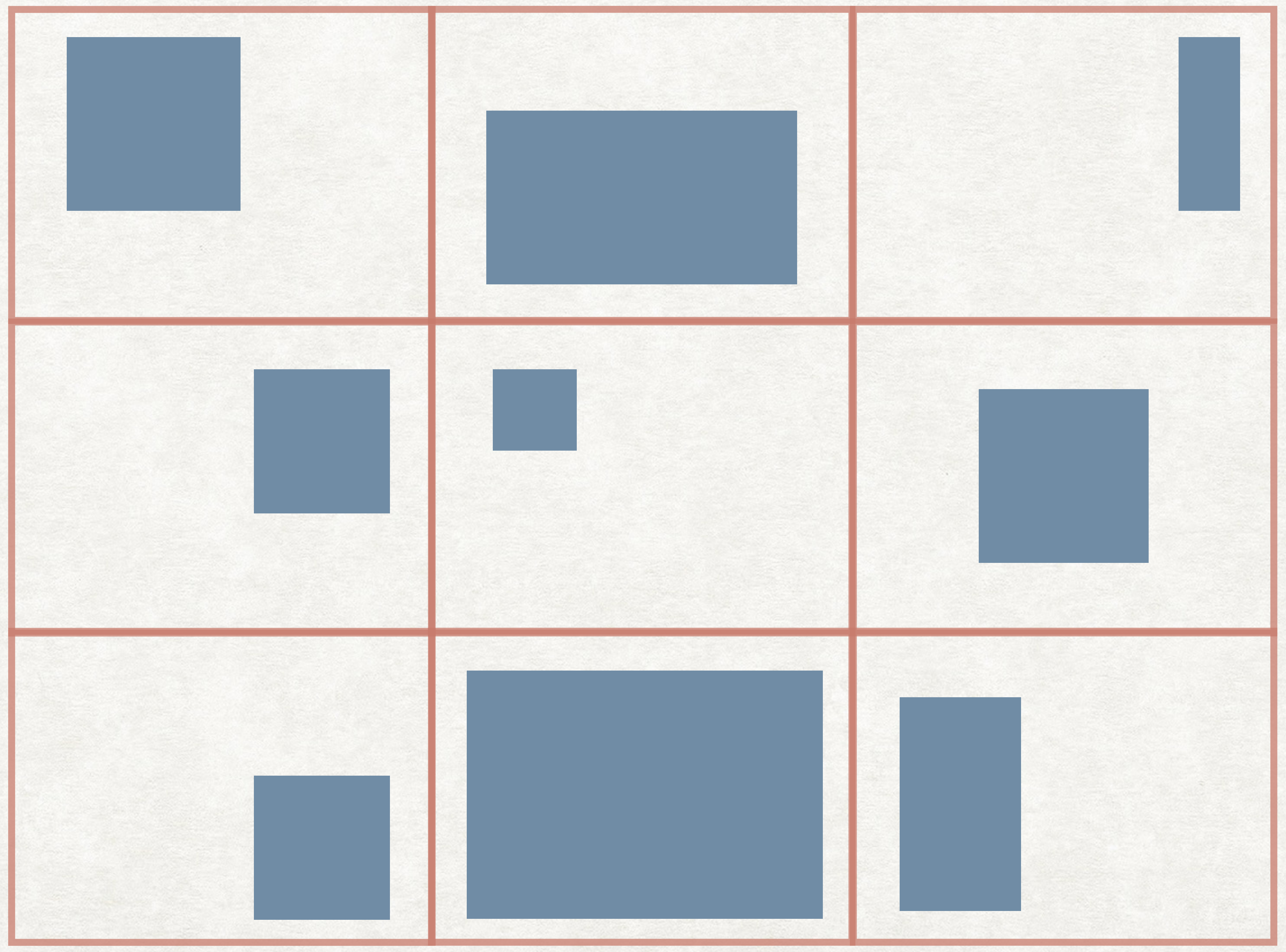
The maze is built by creating rooms which are randomly positioned and with random size and then joining them with corridors.

1. Divide the world space into a 3x3 grid.



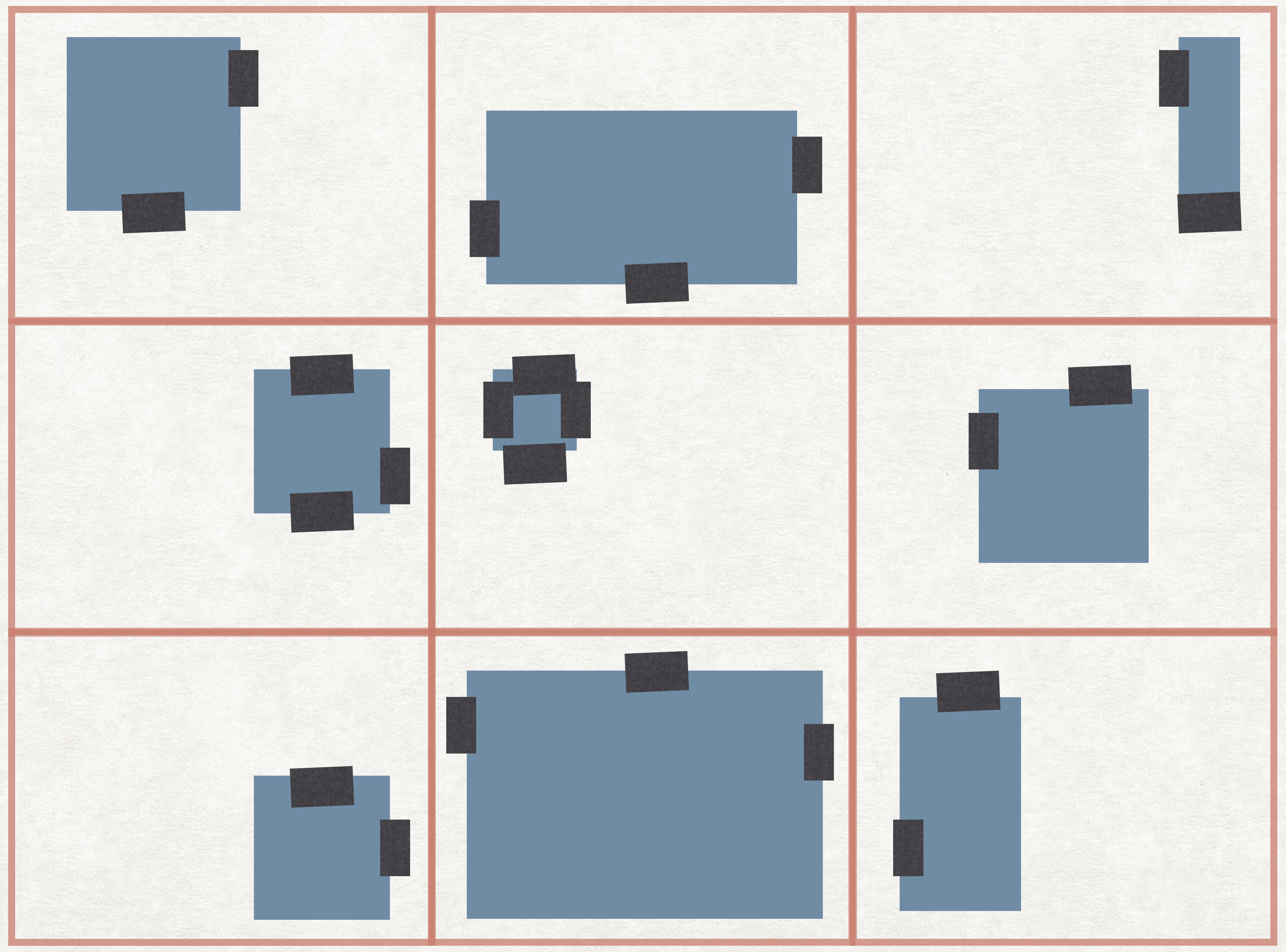
BUILDING A LEVEL

2. Randomly place a room in each grid location. Randomize the size of each room and it's position in the grid element. Don't make all of the rooms the same size or shape and don't always put them in the same position. Rooms should be rectangular. You can experiment with other shapes if you wish.



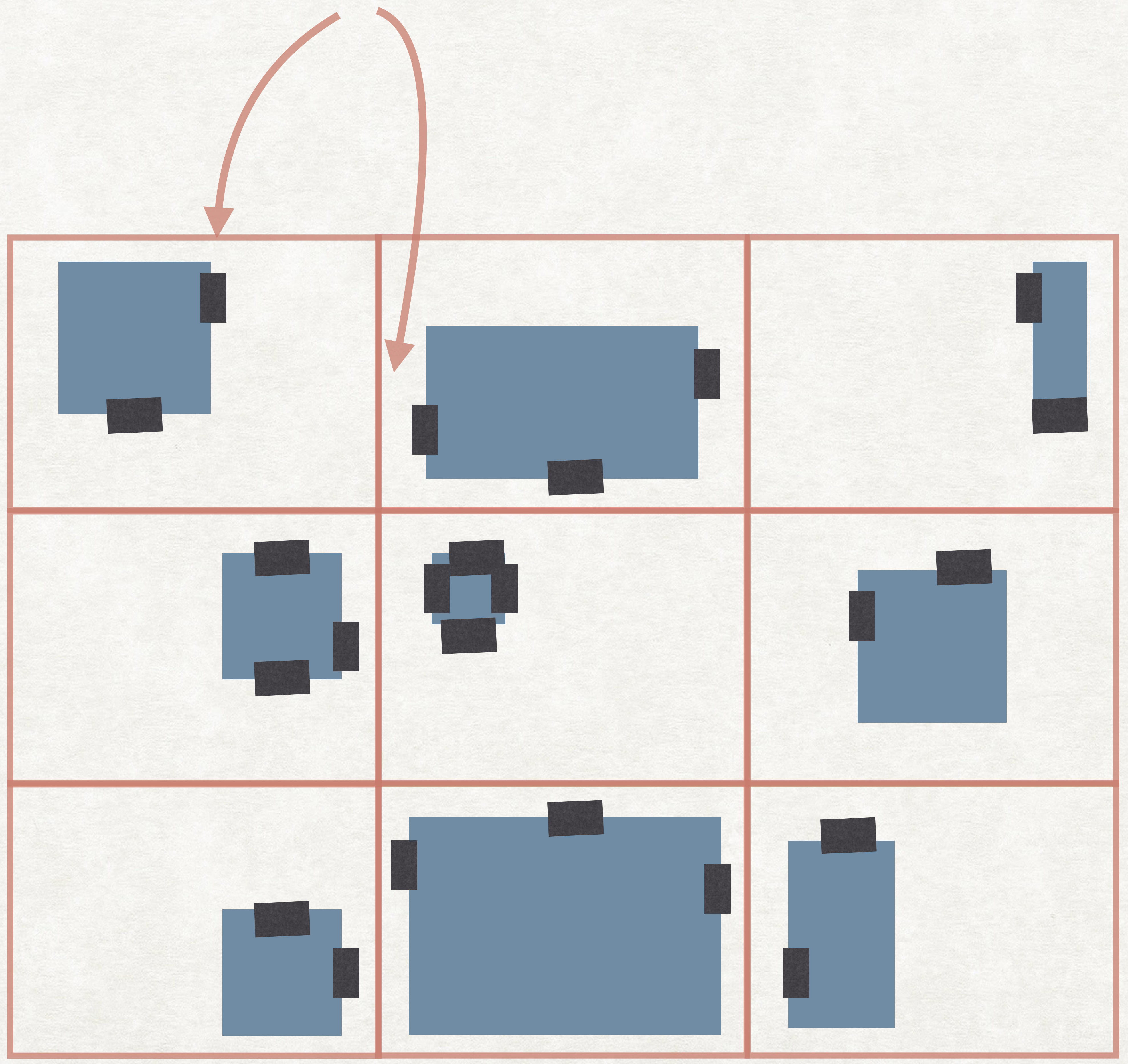
BUILDING A LEVEL

3. Identify random positions for doors on walls that are adjacent to other rooms. The doors aren't actually objects placed in the world. They are locations where hallways will attach to rooms.



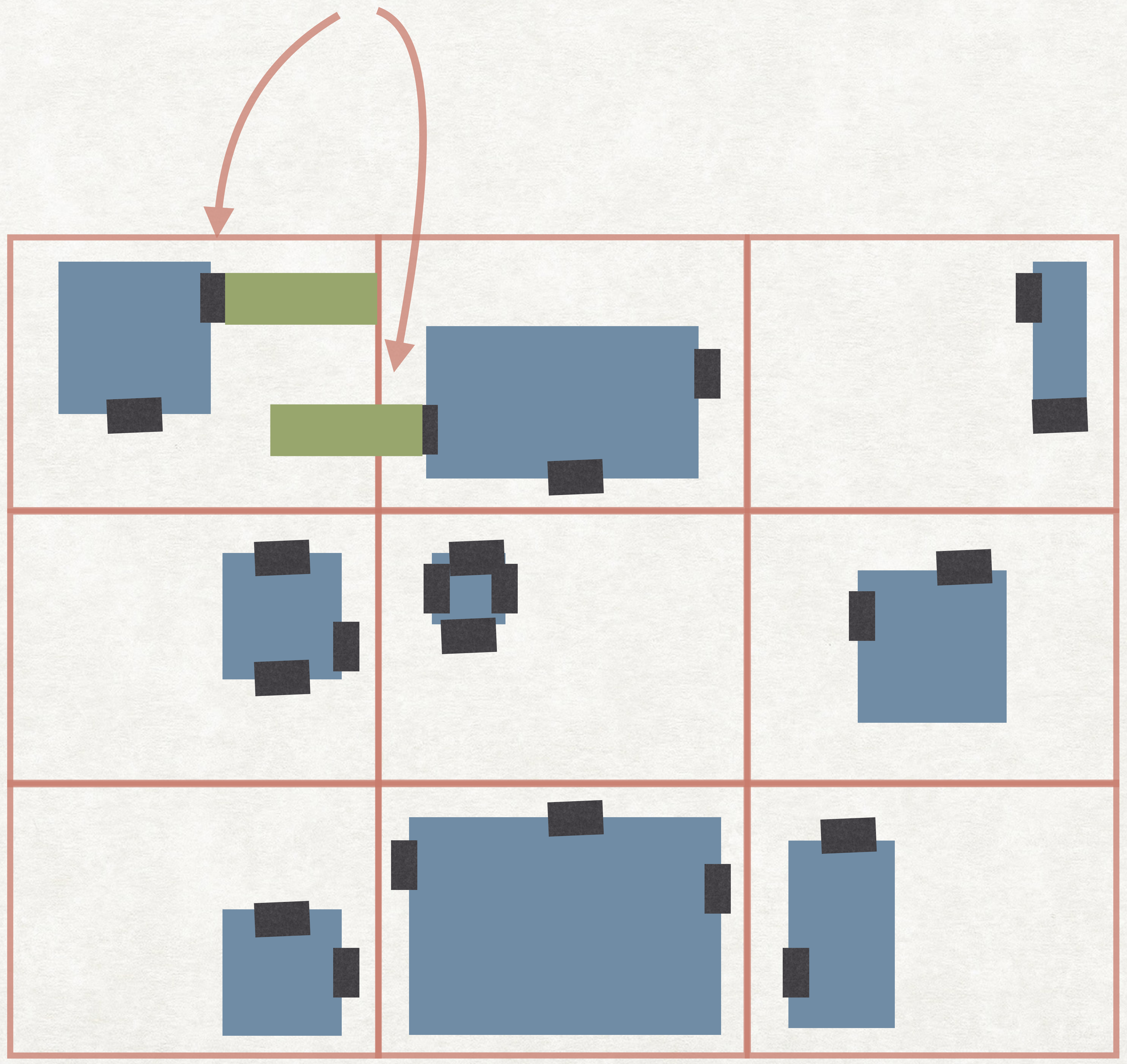
BUILDING A LEVEL

4. Attach adjacent doors to each other with hallways.
   Find the location for two doorways.



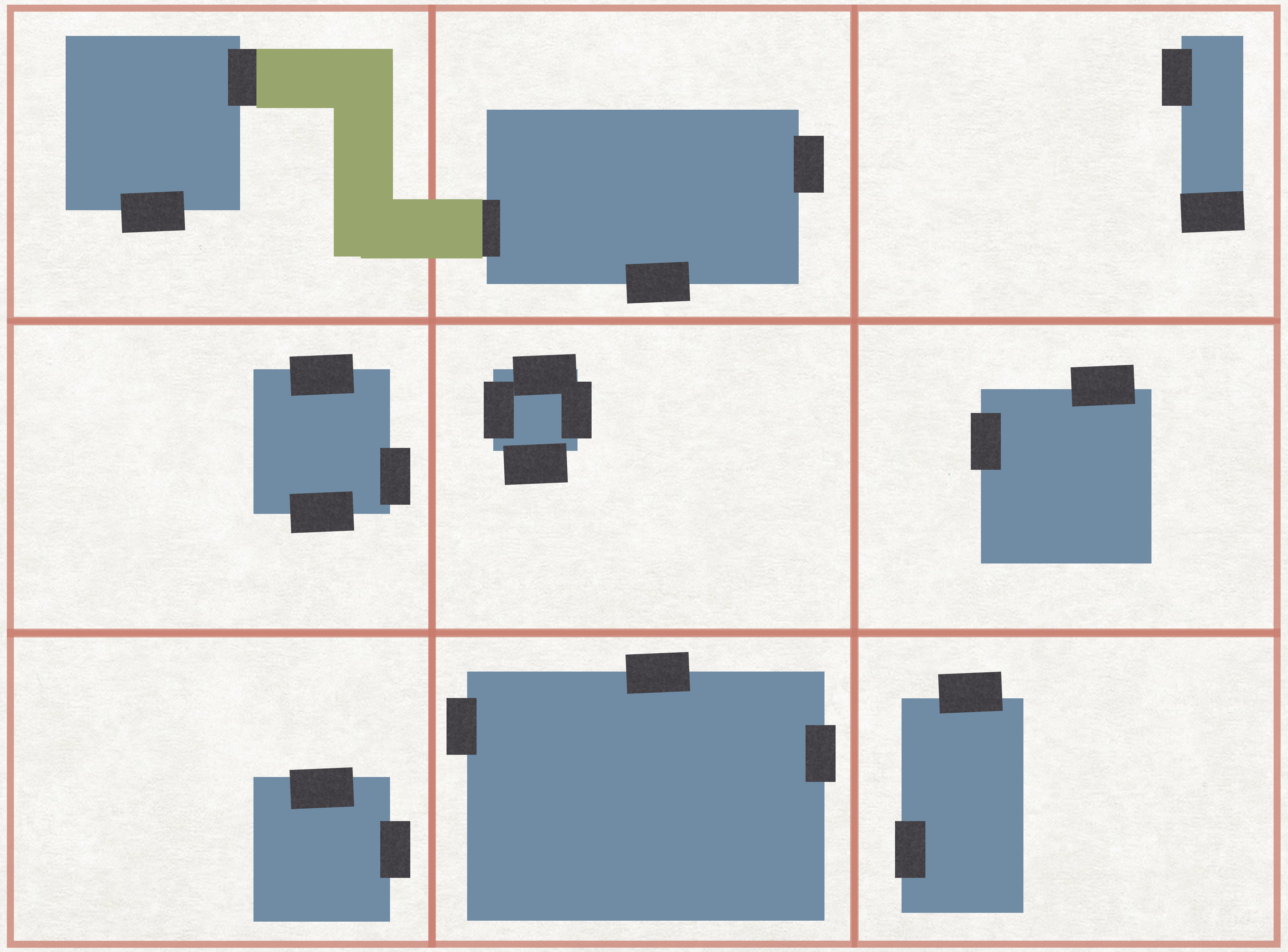
BUILDING A LEVEL

4. Attach adjacent doors to each other with hallways.
Extend hallways from the doorways.



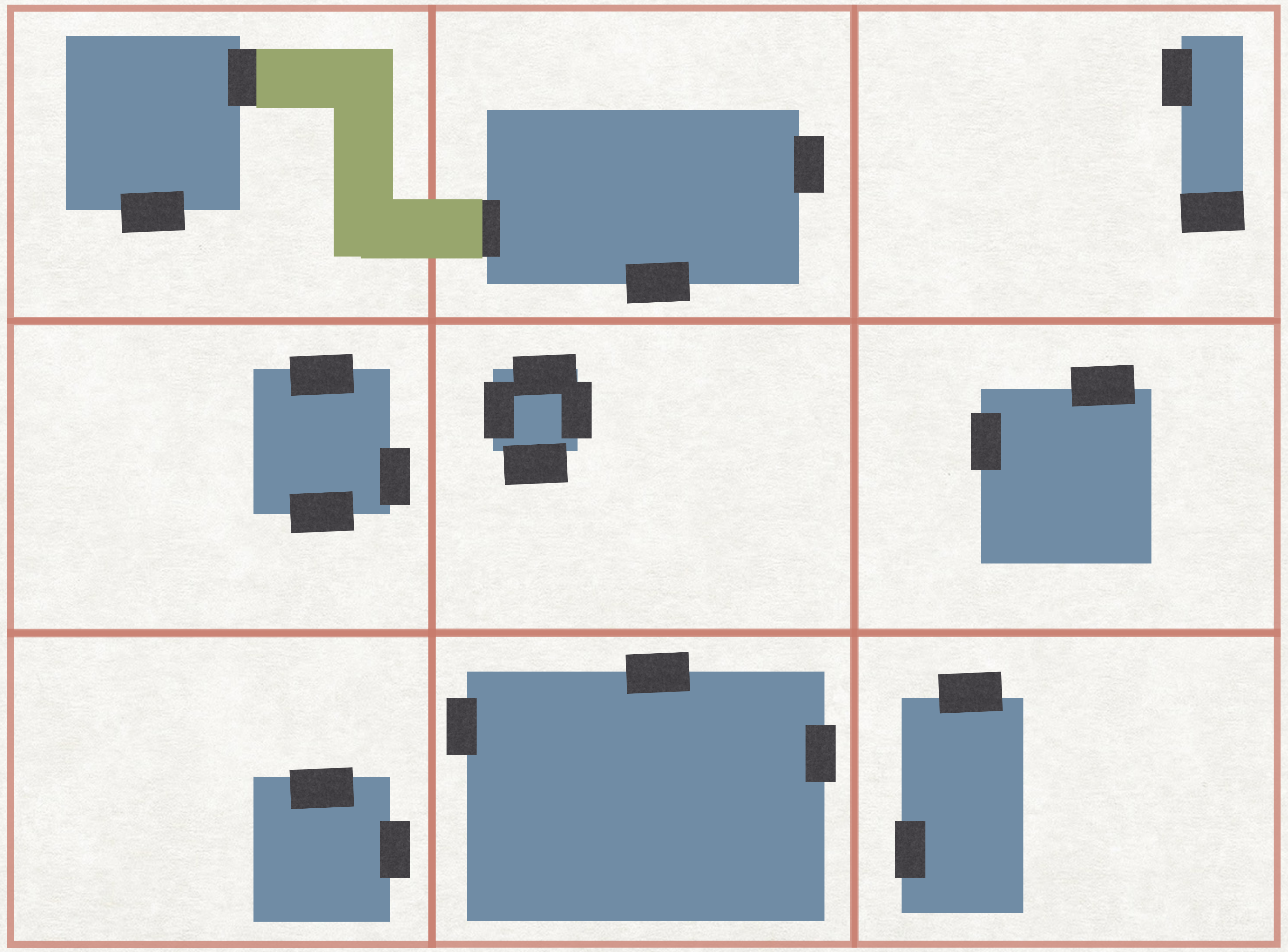
BUILDING A LEVEL

4. Attach adjacent doors to each other with hallways.
Find a point between the two hallways that can be joined to create a single path.
Repeat this for all pairs of doors.



BUILDING A LEVEL

This is all done in 3D so you will be creating rooms and hallways by placing blocks in the world[][][] array.



COLLISION DETECTION

Add collision detection so the viewpoint cannot pass through space that contains a cube (when the world[][][] array is not equal to zero).

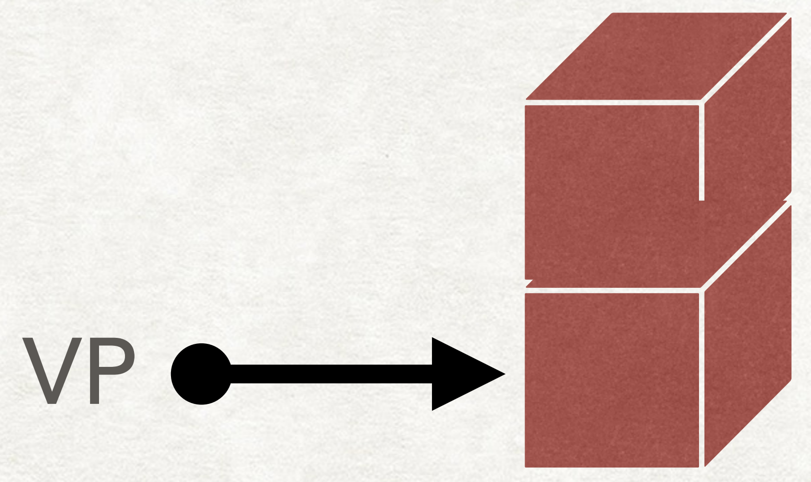
When the viewpoint tries to enter a space that is occupied it should be stopped before the user can see inside the cube.

COLLISION DETECTION

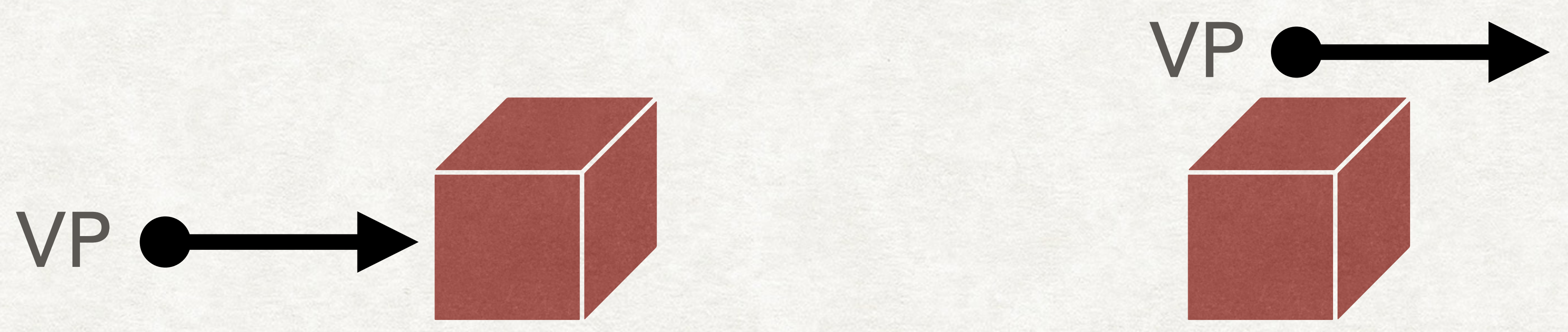
Add collision detection so the viewpoint cannot pass through space that contains a cube (when the world[][][] array is not equal to zero).

When the viewpoint tries to enter a space that is occupied it should be stopped before the user can see inside the cube.

There is one exception to this rule. If the viewpoint tries to move into a cube that is filled but there is an empty space above it then the viewpoint will move on top of the occupied space. You do this by setting the viewpoint's y value so it moves up and above the cube.



Solid Wall - the viewpoint stops.

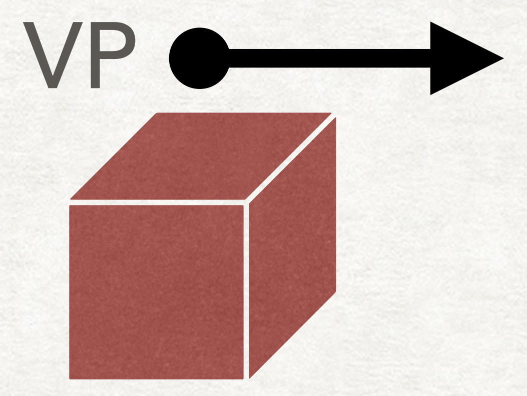


Only one cube - the viewpoint moves on top

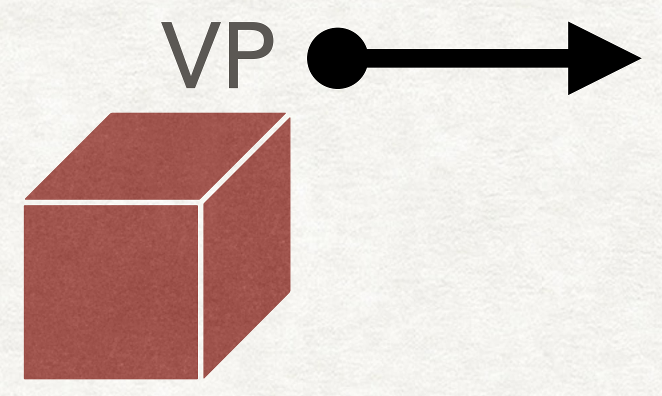
GRAVITY

Gravity should affect the viewpoint when there is empty space beneath it. If there are empty cubes beneath the viewpoint then the viewpoint should slowly drop to the ground.

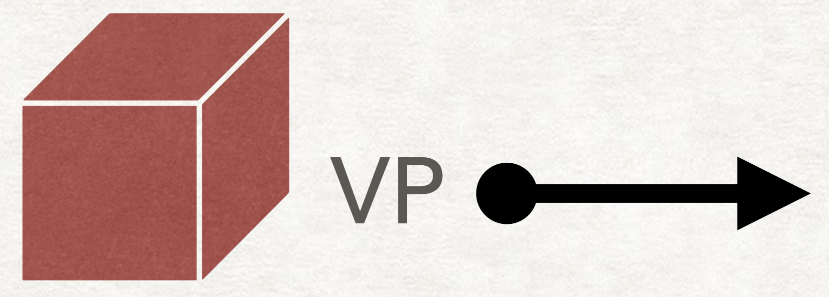
This should happen when the player moves the viewpoint off of a cube. Larger drops will be added later.



On top of the cube.



Step off the edge
of the cube.



Drop to the ground.

A FEW MORE THINGS

1. The valued returned by some of the functions may be **negative numbers**. You will need to consider this when comparing them to world array locations.

2. This is a simplified version of the maze algorithm. Different versions of the game are more interesting because they allows rooms to be missing, doors to be hidden, non-rectangular rooms, and different hallway patterns.

3. AI characters and objects in the rooms will be added to the game. If we add levels to the game then you will need a way to **store the state** of levels and be able to recreate a level that you return to later. This will become important if we add stairs to the game.

4. A lot of characteristics in a game are determined by the **parameters** you choose. Pick reasonable parameters to control things like speed, size, and colour. A game is unplayable if it is too fast or slow, if the objects are difficult to see, or if the colours are too dark or bright.