COMP313
Assignment Log

Paul Mathews *300190240*

1 | Log

First few weeks	Spent a while making mazes because mazes are fun & not a bad way to start learning a new language. Made a prefab that is a square maze generated at runtime. Needed to figure out how to load a prefab by name, this is easy if it is in a folder called "Resources". Managed to get the prefab loading new instances of itself. Added the rolling ball from the Unity tutorial to bounce around the maze and an invisible cube which has the script which loads a new maze and positions it so the corners line up. As the entrance of the mazes is fixed, this just needs to position it with a constant offset which is easy. It would be cooler (and probably necessary for the big project) to have it query the new maze for its door position and line them up based on that. Did a little bit of work on this, making a base Room class which extends MonoBehaviour and has a method to query for the entrance position. Individual rooms/mazes can have a script that extends this.
Upon receipt of model	Started doing player controller – fiddled with the built in Unity one but wanted to add in jumping (because it would be cool to be able to jump over the walls) and also get to grips with Unity a bit more so wrote it more-or-less from scratch. Implemented a jump by adding a big upwards force to the RigidBody as I wanted it to look reasonable. This caused some issues if you land partially on the walls as the character tends to bounce around confusingly and then get stuck upside down etc. A fairly hacky solution was to give the player unit a very large mass (and dial up the jump force correspondingly) which leads to a bit less rolling round (and less getting bumped around by the AI, see later).

Shortly afterwards	Added in AI character. Read the requirements backwards and started off by trying to make it chase the player. As we are operating in a randomly generated maze it seemed like a good idea to offload the pathfinding to Unity (not that A* would be that hard especially given we have the maze data structure, but if Unity will do it for us then why not). This works pretty well just add a NavMesh to the ground plane and NavMeshObstacles to all the walls. The issue happens when a new maze is loaded. Unity only allows one NavMesh per scene and refuses to modify it at runtime. If you need to load things additively then tough luck. The only way a NavMesh can be modified on the fly is with NavMeshObstacles. Fortunately the walls of the maze are NavMeshObstacles, so the solution is to make a gigantic invisible ground plane so that there is an enormous area of flat, square NavMesh. Then when new mazes are loaded on top of it the walls carve out holes in it and all is well. Unfortunately this is not a good solution. Firstly it limits the size of the game from infinite to however big the NavMesh is. Secondly it is not clear how much space this giant NavMesh takes in memory – hopefully not a lot because until the mazes are loaded it is only 4 vertices but you never know. Thirdly it only works because all the mazes are flat. If you wanted to load anything with stairs or ramps this would fail; it only works if the only modification to the NavMesh upon loading a new level/maze is carving out holes.
10/4/15	Had some problems with the AI character bumping into the player irritatingly. Ended up adding a second, large, spherical collider to the player to use as a trigger. This gives a good bit of space around the player to fire off the AI's OnTriggerEnter while it has enough time to stop. Might also help when interacting with objects.
13/4/15	Added some objects to interact with implemented as a prefab with a tag. The tag is the important part, any object with the tag "pickup" will be treated the same by the code. This seems like a flexible way to add new items or change them around. Wanted to be able to bump the items, thought it would be cool to give them a kick by applying a force at an angle. Needed to figure out if there were any objects in front of the play and if so get a hold of the nearest in code. Could use the above big collider, but would have had to balloon it out a bit further than I wanted to keep the AI nice. Physics.Raycast seemed like a good plan: cast a ray out straight ahead and if it hits something we are good go. It even provides a way to limit the distance which is perfect. In practice this required a lot of tweaking. Firstly it won't return an object you are colliding with, so had to add some code to OnTriggerEnter to keep track of pickups being collided with and in the case the raycast fails, check to see if we've hit something that is still close enough to interact with. Made a few silly errors just trying to get the logic right for this but got there in the end...

	Now `SpiderController.GetObjectInFront ()` is almost perfect except that I haven't gotten around to ensuring that the collided object is actually in front of the player's spider. This shouldn't be too hard, just have get a vector between the two and check the angle between that and a vector straight out of the player (which can be found with `transform.forward`). *(did this on 19/4 using* `Vector3.Angle`*, gave it 15 degrees either way)*
17/4/15	Let the user actually pick up the pickups (with left shift). No real issues here, keeps a keeps a list of the actual GameObjects, deactivated. If you press the pickup button again it throws the last object picked up. UI is fairly basic, just showing a count of how many things have been picked up so far. Using the actual GameObjects means any of their state is maintained, this can only be a good thing. Ideally would query them for an icon or something and display that.
18/4/15	Did the AI state machine properly. Has 4 states: patrolling, looking, chasing and found. Patrolling cycles through an array of positions. This is exposed as a public array of Transforms so any number of positions can be set up in the editor. In this state it will go to each position in turn, starting again from the beginning once it reaches the last one. Navigation and movement is handled by the NavMeshAgent, this part is pretty simple and easy. While patrolling the agent is constantly polling for nearby pickups. When one is within a specified distance, the agent stops and looks at it until it gets too far away. Stopping and starting the navigation was a little bit odd – `NavMeshAgent.Stop ()` didn't seem to actually do anything and the agent would only stop if the agent was disabled as well. This meant to start it up again it had to be re-enabled before calling any other methods or it would throw an error. When patrolling (but not while distracted by an object) the agent also checks to see how far away the player is. If the player gets too close, the agent's state changes to chasing. When chasing, the NavMeshAgent constantly has its destination set to the player's position. Getting it to stop appropriately was tricky as setting the NavMeshAgent's stopping distance property did not seem to change very much. The solution was, again, to explicitly disable navigation when the agent collides with the player. When the player collides with the agent (recall that the player has slightly oversized collider for precisely this purpose, so it actually slightly before the models touch) the agent starts to chase, regardless of whether it was patrolling or looking at an object. If the agent was chasing and collides, it moves to the "found" state, which simply disables navigation and looks at the player until they get far enough away to start chasing again.

	While chasing it is possible to distract the agent with a pickup object. It still looks for these and if they get close enough it will go into the ”looking” state. The only way out of this is back into patrolling. In this case the patrol should pick up where it left off, going to the waypoint it was going towards before it started chasing. It will not pick up the exact path, although if you had sufficiently finely grained waypoints it would appear to.
19/4/15	Tidied up, commented out debugging print statements etc. Also fixed the pickups so that objects being touched will be picked up only if they are within 25 degrees either side of wherever the player is facing to make it easier to pick up a ball you are pushing in front of you. Also changed the inventory to display a ’O’ for each ball picked up (rather than a count). This was done by using a UIText object and just changing the text. It is in a box so that if you pick up more than 6 it will start a new line. Was finding the camera awkward, being the child of the player GameObject meant that on the rare occasions when the player jumps and gets flipped around ending up upside down or sideways the camera would follow suit. Moved it outside and wrote a short script, based off the one from the Rolling Ball tutorial, which follows the player around. Making sure it rotated correctly was a little bit awkward, in the end the simplest way was to separate the offset on the x,z plane and the y axis and constantly set the camera to be offset by the x,z distance behind the player and then use `Transform.LookAt` to keep the player centered. Added a state and a method to make the AI character just move to a specific place. Still uses the NavMesh, so it’s really only a few lines. Seemed silly to implement a whole search and some kind of data structure to keep track of how to go where when Unity will just do it for you (as long as the NavMesh is alright, which may not be the case all the time if you want to load content additively).

2 | Summary

2.1 | Controls

Key	*Action*
w/up	move forward
s/down	move backward
a/left	turn left
d/right	turn right
f/left click	bump an object in front
left shift/right click	pickup an object in front, throw an object if none in front and at least one held
space	jump – be careful with this, if you land badly and get flipped around it can be very hard to right yourself.

2.2 | Brief Description

The player controls a spider-like character that starts in a corner of a square maze. There is a computer controlled spider that starts in a different corner and will initially be attempting to visit all four corners in turn. There are sparkly spherical objects in several locations on the initial maze. If the computer controlled character nears one it will stop and look at it until it

gets too far away at which point it will return to patrolling. The player can push them around, pick them up and throw them. If the player gets close to the AI while it is not distracted or bumps it at any time then the AI will start to chase the player. The only way to stop it is to distract it by putting one of the sparkly objects in its path.

If any object or spider makes it to the back-left corner of the maze, where there is a gap in the outer wall, a new adjoining square of maze will be generated. This new maze contains no extra pickups, but new mazes can be generated infinitely. The player can jump over the internal walls of the maze, but the AI will respect them. It is a good idea to be careful when jumping around as the spider has a tendency to flip if you land on a corner or a ball.