Exercises - Vectors and Input

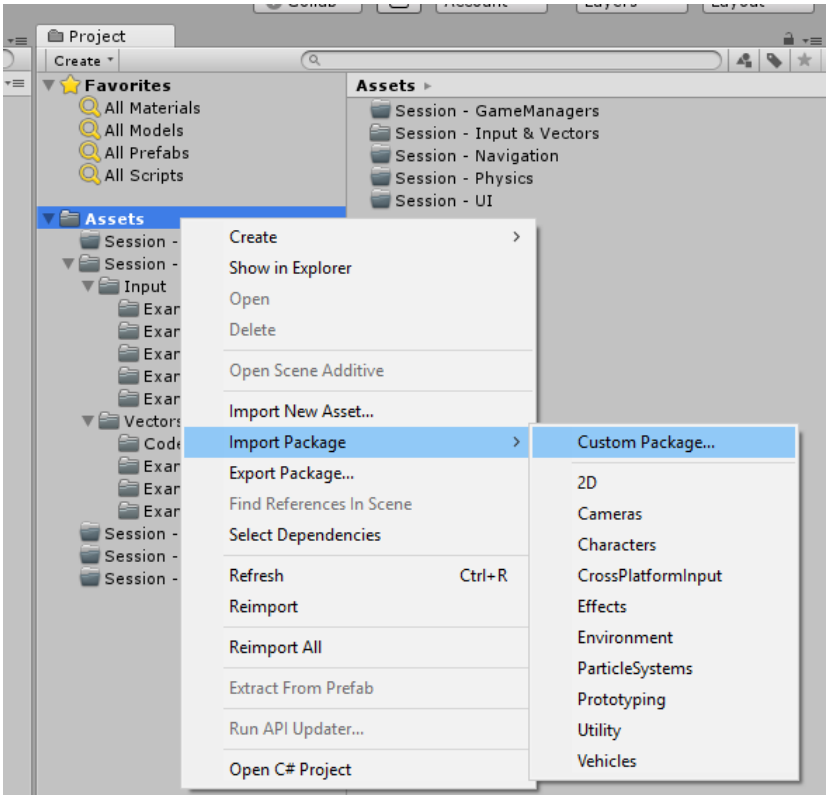
Exercise 1, The session examples

An Asset pack is a Unity specific package file, similar to .zip and the like. It will contain Unity stuff like models, scripts, materials, audio, etc. When you get something from the Unity Asset Store, it will be in an asset pack.

Download the Asset Pack from the course site named “Input and vector examples.unitypackage”. These are the examples from today’s session.

Look through them, run the scenes, check out the scripts. The idea is to make sure you understand the Input parts of the script.

You can import an Asset pack by drag and drop, or by right-clicking the asset folder in the project tab:



Exercise 2, Moving a GameObject

The idea here is to create a simple script to move a game object horizontally and vertically.

- Create a new scene
- Create a game object, e.g. a box or sphere or similar
- Create a script and attach it to the game object

- In the script you put the code to move it, you should just be able to move it in the x-, and z-direction.
- You should use the setup from the InputManager instead of using hardcoded values.
- Look at the slides for inspiration

Exercise 3, Modifying the InputManager

In this exercise you will create new input settings. By default, WASD are used as horizontal and vertical axes. You will create new settings, so that you can use IJKL too.

- Open the InputManager, Edit -> Project Settings -> Input Manager
- Right click and duplicate the “Horizontal” setting, change the positive and negative buttons. Rename it to something else, e.g. “MyHorizontal”.
- Do the same for the “Vertical”
- Duplicate the script from Exercise 2, but change the code so that you use your new axes.

Exercise 4, Reversing direction

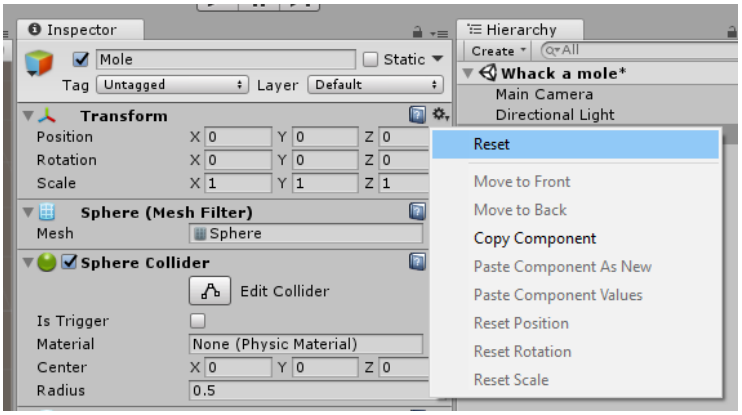
This exercise builds upon exercise 2 or 3. When a button is pressed, the movement directions should reverse.

- Duplicate the script from Exercise 2
- Create a new axis/button in the InputManager, or you can duplicate the “Jump”. Call your new setting “Reverse”. Bind a key to the positive button, or leave it as “space”.
- In your script, when “Reverse” is pressed down, it should reverse the directions of moving, i.e. pressing A previously moved the GameObject to the left, and now it should move to the right. Pressing Reverse again will make everything normal again.
- You can reverse a Vector by scaling it with -1.

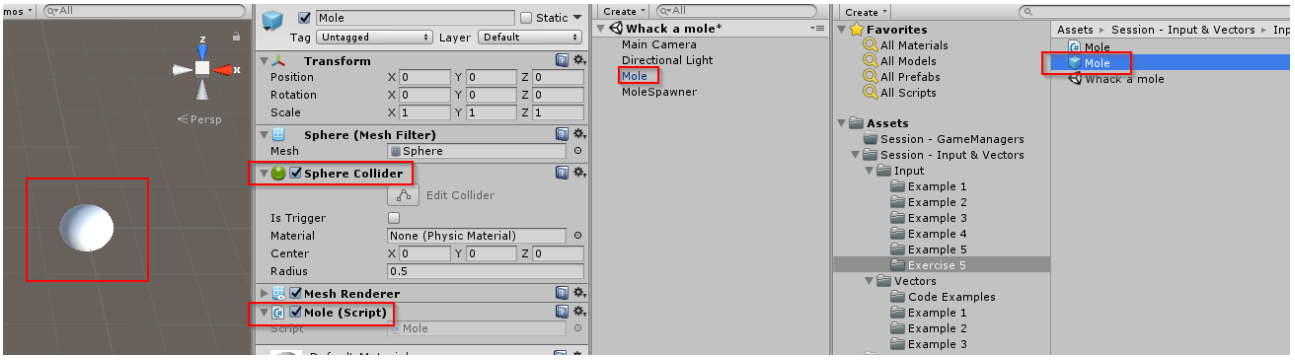
Exercise 5, Whack a mole

This is a reaction game. GameObjects will appear, and you need to click them before they disappear.

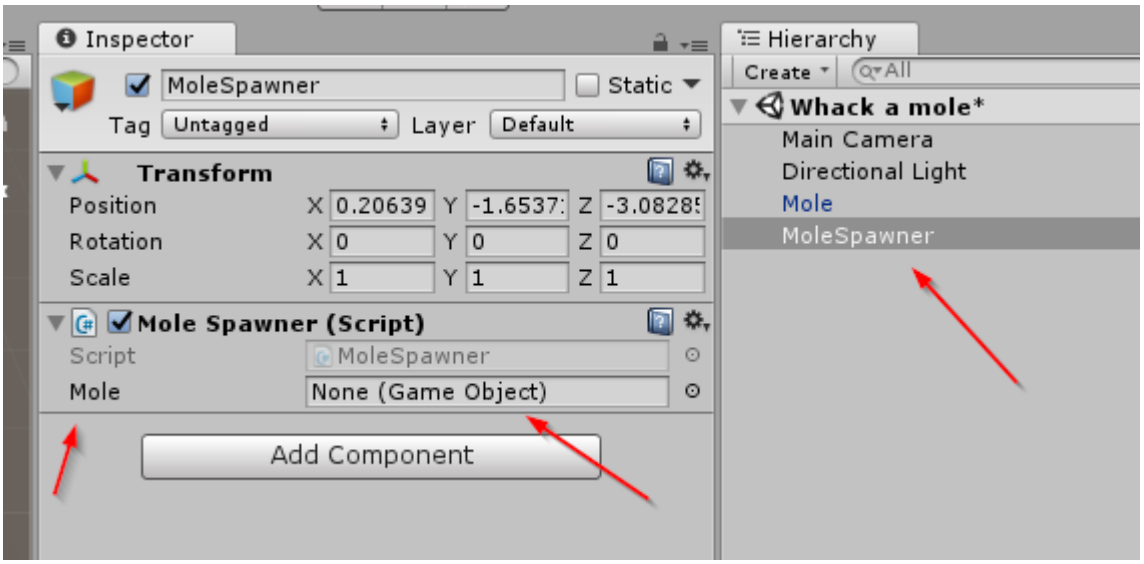
- Create a new scene
- Set the camera at position (0, 10, 0), and the rotation to (90, 0, 0). The camera should now be facing down.
- Create a new Sphere GameObject, rename it to “Mole”. Make sure it has a Sphere Collider component.
  Reset the sphere’s transform properties by right clicking the little cog on the Transform component:



- Create a script called Mole, attach it to the Mole GameObject in your scene. Drag and drop the Mole GameObject into a project folder. You have now created a Prefab. This is a stored GameObject, you can use this for later.



- Delete the Mole GameObject from the scene (NOT the prefab in the project folder hierarchy!)
- Create an empty GameObject, call it MoleSpawner
- Create Script, you can call it MoleSpawner too.
  - It should have a public variable of type GameObject, you can call the variable ‘mole’
- Attach the script MoleSpawner to the GameObject MoleSpawner. You will notice the script has a field named ‘Mole’, with the text “None (Game Object). See below:



- Drag and drop your Mole prefab from the project folder into the Mole field in the screenshot above
- Now, edit the MoleSpawner script:
  - Every other second it should instantiate a new Mole game object.
  - You can use “Time.time” to get the time in milliseconds.
  - You can instantiate new GameObjects with the following line of code:

```
GameObject moleGO = (GameObject)Instantiate (mole);
```

  - You can then access the ‘moleGO’ transform by moleGO.transform, and here you can set the position. It should be placed randomly somewhere x: -9, +9 and z: -5, +5
  - You can use the below method to create random numbers

```
float randomNumber = Random.Range (-9f, 9f);
```

  - Generate random x and y coordinates, and place the newly instantiated mole GameObject.
  - Destroy the mole after 2 seconds, you can do that with: Destroy(moleGO, 2f);
  - If you have done the above correctly, then when you run the scene, a new sphere should be spawned every two seconds, at a random position within the game view.
- Now open the Mole script
  - Create a static variable called score of type int, i.e. private static int score = 0;
  - Overwrite the OnMouseDown method:

```
public class Mole : MonoBehaviour {

    private static int score = 0;

    void OnMouseDown() {

    }

}
```

  - In the OnMouseDown method, destroy the gameobject: Destroy(gameObject). This will remove the GameObject in the next frame. Increment the score and print it out to the console with: Debug.Log(“Score: “ + score);
- When you run the game now, every 2 seconds a new sphere should be created. You can click the sphere, and your score will be printed out.

- You can now add other stuff to the game, e.g.:
  - Random time between instantiating objects.
  - For each object you can lower the time you have to click it before it disappears again, currently it was set to 2 seconds.
  - Can you figure out how to find the number of spheres you did not manage to click?
  - If you miss three spheres, you could print out “Game over”, and stop spawning more objects.

Exercise 6, Multiplayer reaction game

This is a two-player game. You should somehow indicate when a round starts, and then the player who clicks their button the fastest wins the round.

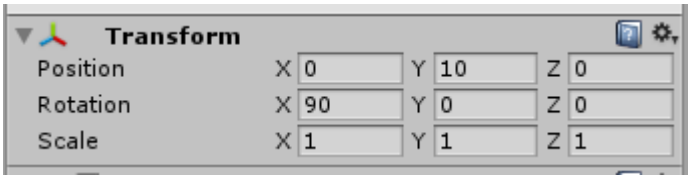
You could do like the previous exercise and spawn an object, when you should press. Then define two buttons, one for each player. Listen for input, and figure out who pressed first. Print out the result.

How do you make sure a player cannot press their button before the round has started?

Exercise 7, Type Attack

Type Attack is a type of game, where you have to type words to kill enemies, before they kill you.

Create a new scene, place the camera like so:



You need to create a 3D object of type “Text - TextMeshPro” and call it e.g. “Word”. Attach a script to it with the same name.

Create a prefab out of your Word GameObject. Remove the GameObject from the scene.

Create a GameObject to spawn words, e.g. WordSpawner. It should spawn the words on the right side of the screen, similar to Exercise 5.

The Word GameObject should move across the screen, to the left. Use the transform.Translate method in the Word script for this.

In the Word script you can access the text of the 3D text object like so:

```
TextMeshPro tm = GetComponent<TextMeshPro>();
tm.text = "hello";
```

In the WordSpawner keep track of the spawned Word objects. Whenever a letter on the keyboard is pressed, you should check against the existing words, to see if the letter matches the first letter of a word.

When you have started on a word, that word must be typed completely, before another word can be started upon.

You can listen for letter input with the method

```
string input = Input.inputString;
```

So

- You have your WordSpawner, it will occasionally spawn Word GameObjects. In this Word GameObject, you set the text to some random word.
- You now have a list of currently existing words. These words move across the screen, from right to left. If a word reaches the left side of the screen (check for its x coordinate), you should print out “Game Over”, and stop the game.
- Whenever a letter is pressed on the keyboard, search through the list of Words to see if the input letter matches the first letter of one of the existing words. You can e.g. use the methods string::SubString(..).
- If you find a word, this is now your ‘current’ word. This is the only word you check against. You must complete an entire word, before another word from the list can be started upon.
- Keep the current word in a private variable, so it’s easier to keep track of. Whenever the next letter of the word is pressed, remove that letter from the word. When there are no letters left, destroy the Word GameObject. Remember to remove the Word GameObject from the list of Words.
- E.g., you have the word “summer” on screen.
  - You first type ‘u’. But it’s not the first letter of a word on the screen, so nothing happens.
  - Then you type ‘s’, which matches the first letter in ‘summer’. ‘summer’ is now your current word, and you have removed ‘s’ from ‘summer’.
  - You then type ‘u’, and so only ‘mmer’ is left on screen. And so on
  - When the word is done, you destroy it, and can look for a new current word.
- You can either have a long list of words to pick randomly between, or you can generate random strings of letters. Random strings can be created like so:

```
string randomString = "";
for (int i = 0; i < numOfLetters; i++) {

    int num = Random.Range (0, 26);
    char letter = (char)('a' + num);
    randomString += letter;

}
```

In the below image, ‘winter’ is the current word, and I have already typed ‘wi’, so it’s removed.

Find ways to increase difficulty, e.g.:

- Increase the speed of the Word objects
- Increase the length of the words
- Increase how often words are spawned.