1 Scripting Overview

The whole note is devoted to introduce some special techniques towards engine.

1.1 Creating and Using Script

The GameObject is controlled by the *Component*, and script is used to modify and customize.

1.1.1 How to create script

1.1.2 Anatomy of Script File

Start(): It's called by Unity before gameplay begins.
Update(): Handling the frame update for the GameObject
The constructor is not needed as it's handled by the editor and may not take place at the start of gameplay.

1.1.3 Controlling a GameObject

Script instance must be attached to a GameObject to control.
Debug.Log("message") can output to the console in the engine.

1.2 Variables and the Inspector

Variables are declared public to be access in the Inspector. You can also make changes to the variables during the gameplay temporarily.

1.3 Controlling GameObject Using Component

1.3.1 Accessing Components

GetComponent function can be used to get reference to the component properties.

```
RigidBody rb = GetComponent<RigidBody>();
```

you can use it to access the parameter and manipulate the instances.

1.3.2 Accessing other object

You can either link objects with other variables with public gameObject, which is similar to the public variables. You can also declare a component and drag a other gameObject on it to attach component of other gameObject. This operation must be done in the editor rather than runtime.

Finding Child Objects and **Finding Objects by Name or Tag** can be used to locate gameObject in the runtime.

Finding Child Objects: when you want to keep track same kind of objects such as location of waypoints, you may make the child of a logical parent and can access them using foreach.

```
public gameObject wayPoints;
...
void Update() {
    int num = wayPoints.childCount;
    ...
    foreach(Transform t in wayPoints){
        wayPointArray[i ++] = t;
    }
    ...
    particularChild = wayPoints.Find("start");
}
```

Finding Objects by Name or Tag:

```
\\GameObject[] gameObject_3;
gameObject_1 = GameObject.Find("Car");
gameObject_2 = GameObject.FindWithTag("Player");
gameObject_3 = GameObject.FindGameObjectsWithTag("Enemy");
```

1.4 Event Functions

Event Functions are activated by Unity in response to events that occur during game play, such as Update() and Start().

1.4.1 Regular Update Events

Update() is the main function to make changes, taking place before each frames is rendered and each animations is calculated.

FixedUpdate() is called before each physics update. This function is dedicated to the change of physics.

LateUpdate() can be able to make additional changes after Update() and FixedUpdate(). It can be used to adjust camera and character orientation.

1.4.2 GUI events

Unity has a system for rendering GUI controls over the main action in the scene and responding to clicks on these control. These code should be included in *OnGUI()* function.

```
void OnGUI() {
    GUI.Label(labelRect, "Game Over"); //making a text or texture on screen
    /*
    Prototype: GUI.Label(Rect Position, string text);
               GUI.Label(Rect Position, Texture image);
    Rect: A 2D rectangle defined by left upper X and Y position, wight, height.
          It can also be constructed by xMin, xMax, yMin, yMax.
    */
}
```

You can also detect *mouse events* that occur over a GameObject as it appears in the scene, such as targeting weapons. *OnMouseDown()* is called when user has pressed the mouse button while over the GUIElement or Collider. A call to *OnMouseOver()* occurs on the first frame the mouse is over the object until the mouse moves away, at which point *OnMouseExit()* is called. It needs the collider to be marked as trigger. More specific mouse and keyboard operations can go to look up for Monobehaviour.

1.4.3 Physics events

A series of function will be called when collision is detected, such as *OnCollisionEnter()*. When collider is marked as trigger, it turns into *OnTriggerEnter()*. Object may contact with multiple objects, which requires for a parameter to identify.

1.5 Time and Framerate Management

The Update() allows monitoring inputs and other events from the script. A important thing is that neither framerate or length of time between Update() is called is constant. *Time.deltaTime* is the time in seconds it took to complete the last frame. It remains constant when called from FixedUpdate(). In order to subtract or add for a constant per frame, multiply it by Time.deltaTime can guarantee a constant offset per frame. For example:

```
void Update() {
    Transform.position = (0.0f, 0.0f, distancePerSeconds * Time.deltaTime);
}
```

1.5.1 Fixed Timestep

There is a fixed timestep between FixedUpdate() for physics engine. You can change it in *Time Manager* and read from *Time.fixedDeltaTime* in order to get better physical performance.

1.5.2 Maximum Allowed Timestep

If fixed timestep is too small that gameplay framerate is relatively low, many physical update will in one framerate update, causing even lower fresh rate and unsynchronized movement. With the set of *maximum allowed timestep* in *time manager*, when the rendering time of one frame exceeds it, the physical engine will stop calculating and staying at that conditions waiting for the update of frame. It's a trade off that causes the movement slower than the real world while having better framerate.