Unity/Vive Cheat Sheet

Terminology Notes

- **Static** game objects are objects that do not move in the game; it is best practice to mark non-moving objects (walls, floors, etc.) as static so that information about them can be precomputed. The static checkbox is in the top right of the inspector. https://docs.unity3d.com/Manual/StaticObjects.html
- **Kinematic**: game objects with rigid bodies are normally acted upon by physics. That is, they receive forces and torque (twisting) when they collide with other objects or other objects collide with them. However, if a rigid body is set to be *kinematic* then it essentially shuts off their interaction with normal physics (gravity is another matter that can be independently turned on/off). Objects can still collide with them, but they are unaffected and they do not affect the colliding objects. Instead they pass through as if it were a ghost. Use a kinematic rigid body if you want to "manually" move the object through the game environment through code rather than have it simply react to the environment's physics.
- `FixedUpdate()` vs. `Update()` vs. `LateUpdate()` – `FixedUpdate()` is (usually) called multiple times per frame at a predefined interval (thus there is no need to tie your code to a fixed time frame using `Time.deltaTime`) while `Update()` is called once per frame and `LateUpdate()` is called once per frame after `Update()`. As a general rule, time-dependent physics calculations should be done in `FixedUpdate()`, game mechanics should be done in `Update()` and `LateUpdate()` should contain any code that relies on the game mechanics being updated first. More: https://docs.unity3d.com/Manual/ExecutionOrder.html

Tips & Tricks

- Often you may want to export a 3D asset from Maya or similar program into Unity. The recommended way is to export it as a `.fbx` or `.obj` file. If you use `.fbx` you will also likely want to scale the objects by selecting the file in the project view, and in the inspector changing the scale factor to offset the file scale listed. For example, if the file scale is 0.01, set the scale factor to 100.

- Often in Unity one script needs to access code in another script. Scripts are generally attached to a `GameObject` (https://docs.unity3d.com/ScriptReference/GameObject.html). Suppose that the script's name is `GameData`. If we wanted access to this script's public methods, we can get the script itself using the following parameterized code.

```
GameObject go = ...;
GameData gd = go.getComponent<GameData>();
```

  Assuming that you've gotten a reference to the `GameObject` (the variable `go` above), then you can use the `getComponent` method to get the script. The `<GameData>` syntax is a parameterization (similar to Java or C++'s templates). The type in the parameterization needs to match the type of component you intend to get. For example, if you wanted a `RigidBody` instead, you would use

```
GameObject go = ...;
RigidBody rb = go.getComponent<RigidBody>();
```

More Resources

- A (more current) video tutorial series:

  https://www.youtube.com/watch?v=5C6zr4Q5AlA

  https://www.youtube.com/watch?v=MKOc8J877tI

- Video tutorial on using the Skeleton System:

  https://www.youtube.com/watch?v=a9EBILq2ep8

- Super in-depth tutorial on Unity/Vive:

  https://developer.valvesoftware.com/wiki/SteamVR/Environments/Environment_Tutorial:_Hammer_and_Basic_Lighting