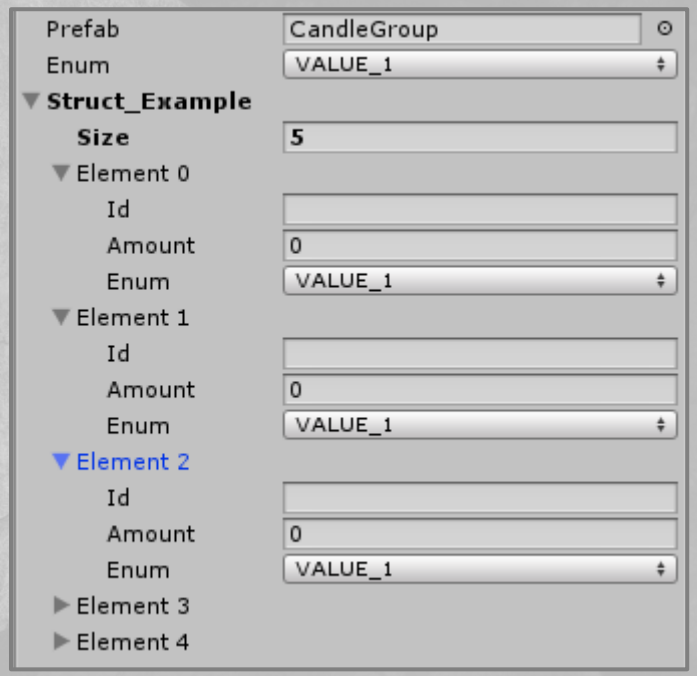
Serializing Variables

- Avoid public variables
- Serialize fields for access in editor
- Enums and stucts




```
[SerializeField] private GameObject m_Prefab;

--------

[SerializeField] private E_test m_enum;

private enum E_test
{
  VALUE_1,
  VALUE_2,
  VALUE_3
}

-------

[SerializeField] private S_STRUCT[] m_Struct;

[System.Serializable]
private struct S_STRUCT
{
  [SerializeField] private string m_Id;
  [SerializeField] private int m_amount;
  [SerializeField] private E_test m_enum;

  public S_STRUCT(string id, int am, E_test e)
  {
    m_Id = id;
    m_amount = am;
    m_enum = e;
  }
}
```

Events

- Avoid using broadcasts
- You MUST unsubscribe
- You MUST null check

SET UP EVENT

```
public delegate void SomeAction();
public static event SomeAction MoveGuy;
public event SomeAction OnSomeOtherAction;
```

USE EVENT

```
void Start()
{
   EventsExample.MoveGuy += OnMoveGuy;
}

void OnDestroy()
{
   EventsExample.MoveGuy -= OnMoveGuy;
}
```

```
if(MoveGuy != null)
{
   MoveGuy();
}
```

Prefabs within Prefabs

- Allows you to alter the core prefab or replace it with an entirely new one
- There is still a cost with loading at awake

```
public delegate void LoadComplete(GameObject loaded);
public static event LoadComplete OnLoadComplete;

[SerializeField] private GameObject m_Prefab;

void Awake ()
{
     LoadAsset ();
}

private void LoadAsset ()
{
     GameObject go = (GameObject)Instantiate(m_Prefab,
     transform.position, transform.rotation);

     go.transform.localScale = this.transform.localScale;
     go.transform.parent = this.transform.parent;

     if (OnLoadComplete != null)
          OnLoadComplete (go);

     RemovePlaceHolder ();
}

private void RemovePlaceHolder()
{
     Destroy (this.gameObject);
}
```

Prefabs within Prefabs

- New Unity UI

```
public delegate void LoadComplete(GameObject loaded);
public static event LoadComplete OnLoadComplete;

[SerializeField] private GameObject m_Prefab;

void Awake () {

    GameObject newI = Instantiate(m_Prefab) as GameObject;
    newI.transform.parent = transform.parent;

    RectTransform rectT = newI.GetComponent<RectTransform>();
    RectTransform oldR = GetComponent<RectTransform>();

    rectT.anchorMax = new Vector2 (oldR.anchorMax.x, oldR.anchorMax.y);
    rectT.anchorMin = new Vector2 (oldR.anchorMin.x, oldR.anchorMin.y);
    rectT.offsetMax = new Vector2 (oldR.offsetMax.x, oldR.offsetMax.y);
    rectT.offsetMin = new Vector2 (oldR.offsetMin.x, oldR.offsetMin.y);

    if (OnLoadComplete != null)
        OnLoadComplete (newI);

    //destroy the placeholder
    GameObject.Destroy (this.gameObject);
}
```

To Note....

- GetComponent..
  - Use VERY sparingly!
  - GetComponentInChildren & GetComponentsInChildren only search active gameobjects, to search inactive as well use GetComponentsInChildren ([Type],true)
- Namespaces and default values

```
Public void ExampleMethod(int required,
int optionalint = 10)
```

- Garbage Collection
- Dynamically loaded Resources
  - Any assets loaded using the Resources e.g. Resources.Load(…), MUST be released when you are done with them.
  - Loading texture assets cause a CPU spike when they are first rendered.
  - Unloading causes a Garbage Collection CPU Spike

Data Driven

- Longer life cycle & content sells
- Designers and artist can make new and tweak existing the content with very little (if any) code support
- Can reduce bugs, fixed once fixed everywhere
- Allows for more dynamic content

- Generics and <T>
- Unity and Data: Json – json.net
- Object Count
- Remember to release you data

```
public virtual void ParseData<T>() where
T:JSDataObject
{
  ...
  JSDataObject dObj =
  (JSDataObject)JsonConvert.DeserializeObject<T>
  ([loadedData], [dataconverter]);
  ...
}
```