VR best practices and UI design

Manuel Pezzera

manuel.pezzera@unimi.it

Lab 12

Designing VR applications

ONE MAIN GOAL

MAXIMIZE USER IMMERSION

What can go wrong?

- Sensory conflict
  - Unnatural stimuli in the virtual world may lead to bad VR experience and sometimes to VR sickness
- VR Sickness
  - Some people are more sensible to sensory conflicts and may feel sick while using your VR applications
- User Safety
  - It is important to be concerned about user safety when designing VR application as while operating in the virtual world they are more vulnerable in the real world.

Sensory conflict

- Happens when perception of self motion is based on incongruent inputs from visual, vestibular and non-vestibular systems.

VR Sickness

- Sensory conflict appear to be one of the main cause of VR sickness.
- We should be aware of this problem when integrating locomotion inside our VR application

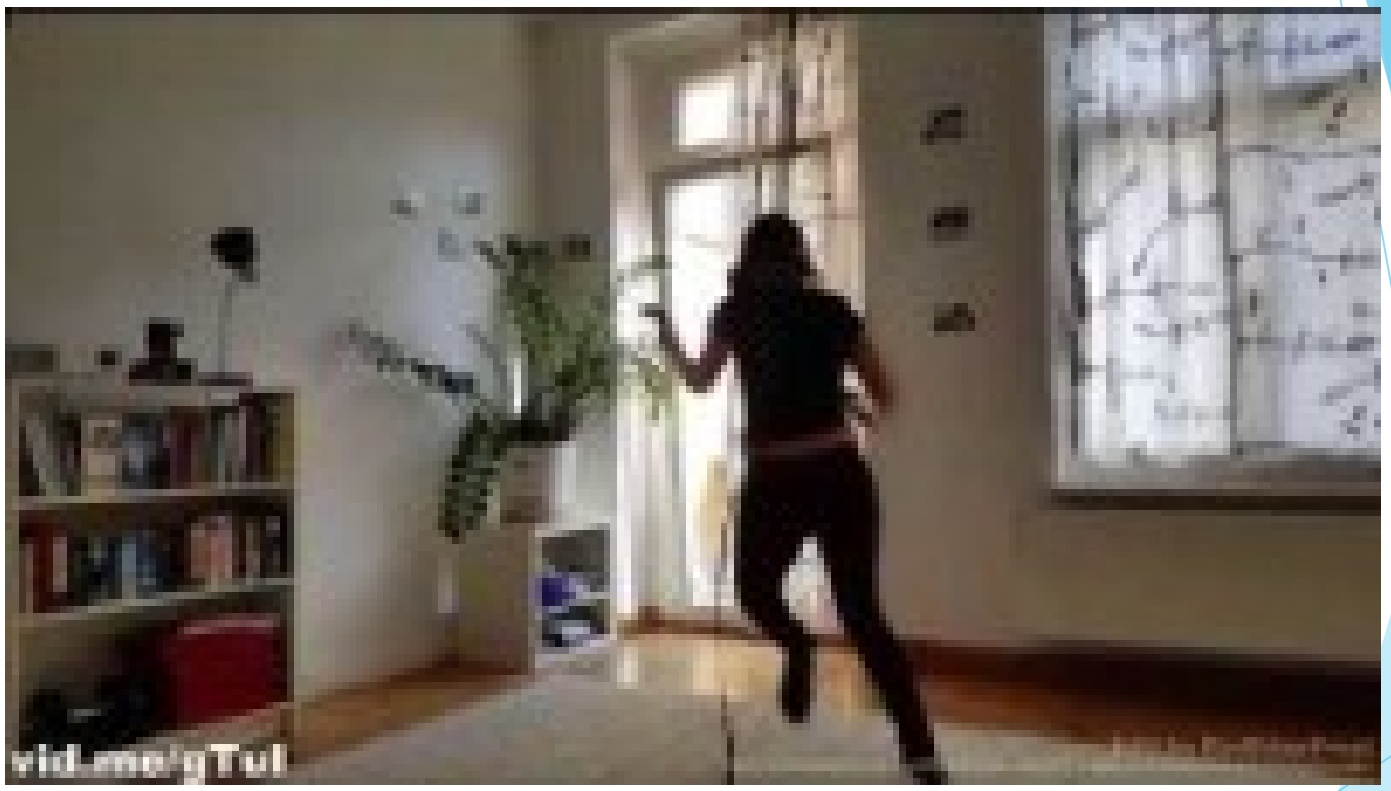
GOOD NEWS

- The more time users spend using VR, the less likely they are to experience discomfort, this is caused by our brain learning to reinterpret visual anomalies

How to minimize sickness issues

- **Always respond to user inputs**, even on menus, while game is in pause etc.
- **Do not instigate movements without user inputs** (e.g., rotate camera while user is not moving the head)
- Movement in Virtual World should be **consistent with head** and body movements.
- Think very well on how to implement locomotion in your application, the safest (even if harder) option could be to **avoid locomotion by design.**

Mind users' safety

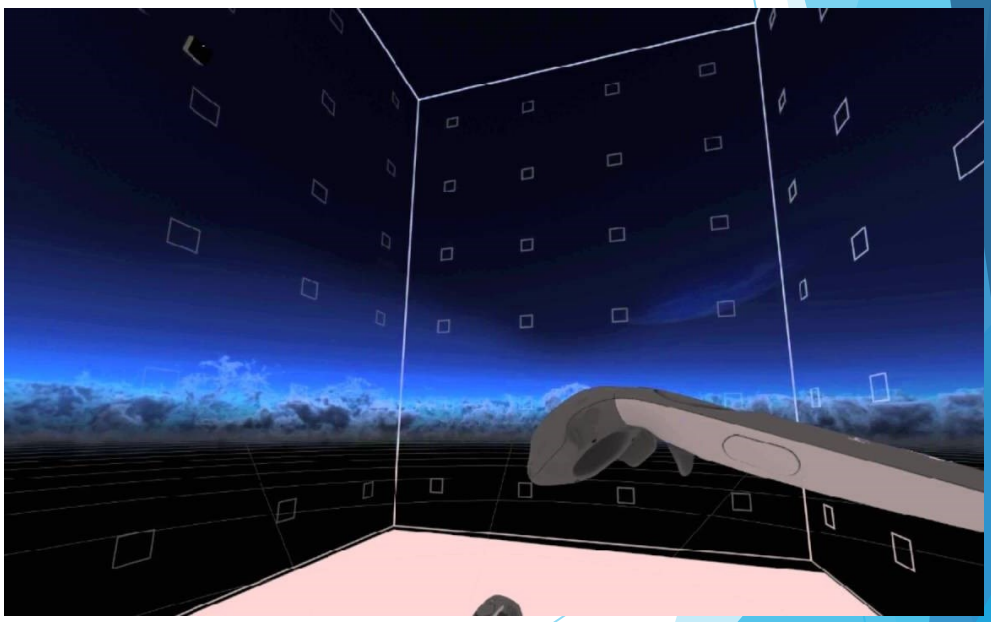


https://www.youtube.com/watch?v=0KcllPEe8y8

Mind users safety

- Allowing users to freely move in the real world while blinded by a VR headset, tends to be a bad idea in general.
- However, finding a good solution to this open problem, would greatly improve VR experience.

HTC Vive solutions to this problem is to place some gizmos in the VR world that remind the users where the boundaries of the safe movement area are in the real world.



Locomotion in VR

- Acceleration vs. Speed
  - While **acceleration** seems to be the **primary cause of discomfort** in VR experience, it appears there is **no straightforward relationship between speed and discomfort**.
- General guidelines:
  - Allowing the user to set the pace of movement may help in control discomfort
  - Forward movement is more natural than backward or side movement
  - Open spaces with respect to enclosed spaces are known to be more comfortable to users.

Locomotion Techniques

- **No locomotion by design:** To completely avoid locomotion discomfort, it is possible to design VR application that do not require locomotion at all (when possible).
- **Teleportation:** Allows users to move around the scene without having to deal with accelerations.
- **Artificial movement:** Moving on a railroad or on a predefined path at a fixed speed.
- **Fade to black:** Like teleportation, fade to black before switching location.
- **HMD acceleration:** Use HMD accelerometers and gyroscopes to allow users to control movement direction (e.g., airplane movement).

Interaction techniques (1)

- Gaze based interaction (Staring at objects causes something to happen in virtual world)
  - Requires a UI element to show what exactly we are pointing to (e.g., little dot in the middle of the screen)
- Dedicated controllers (e.g., Gear VR, Oculus or HTC Vive controllers)… usually 3 or 6 DOF
  - Good practice is to render the controller in the virtual world, with a ray coming out from it showing where the user is pointing at.
  - Changes in ray or target colors are encouraged to provide feedback on the presence of an interactable object

Interaction techniques (2)

- Hands (Leap motion controller):
  - **Rendering hands** in the scene may **greatly enhance VR experience**, however, **render a whole avatar only** if you are sure that it will be **perfectly aligned** with the user real position.
  - Mind Hardware Limitation, leap motion work best when hands are right in front of the HMD, so design your applications to encourage user to interact with the world in this way. Also consider the Leap field of view when designing your app (Do not require users to look at one side and interact with an object placed far away from where they are looking)
- Other way? Custom Hardware? It's a good project idea!

Catching user's attention

- This is particularly important in 360° videos, but meaningful also for real time applications. The main action in a video may happen in a place where the user is not looking at…

Catching user's attention

- An example of use of good design practices can be observed here…

https://www.youtube.com/watch?v=BEePFpC9qG8

Catching user's attention

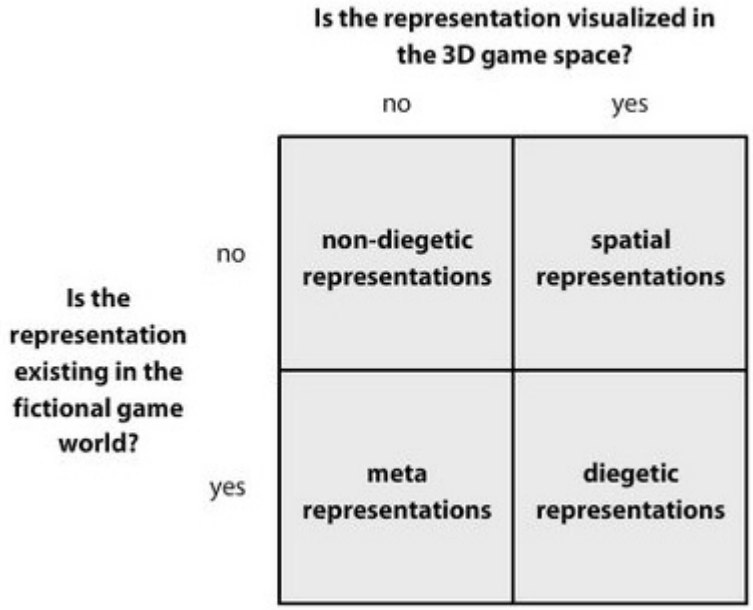
How To?

- **Lighting effects:** focus lights on the place you want the user to look at.
- **Visual effects:** also, particles, in-world UI elements (such as arrows), etc. may be used to focus on a particular element of the scene.
- **Spatial Audio:** information on where something interesting is happening in the scene may be provided by audio effects, spatialization will help the user find the source of the audio easily!

UIs

Diegetic UI: Interface that is included in the game world

Non-diegetic UI: Interface that is rendered outside the game world, only visible and audible to the players in the real world



Spatial UI: UI elements presented in the game's 3D space with or without being an entity of the actual game world

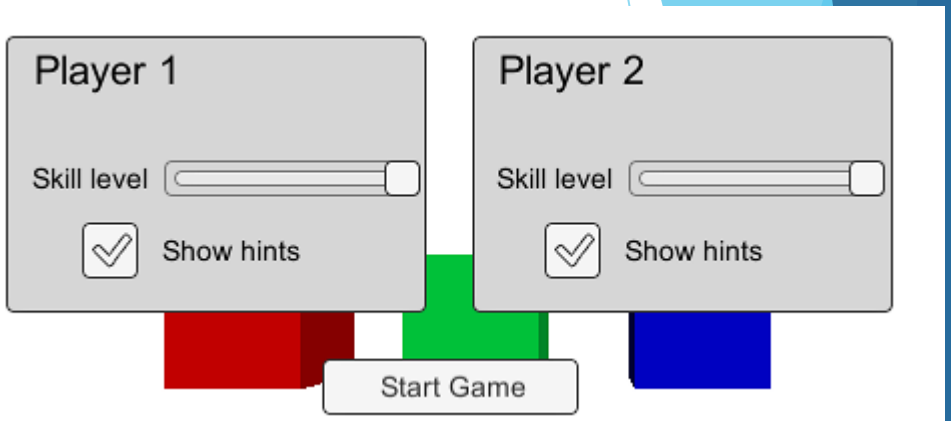
Meta UI: Representations can exist in the game world, but aren't necessarily visualized spatially for the player

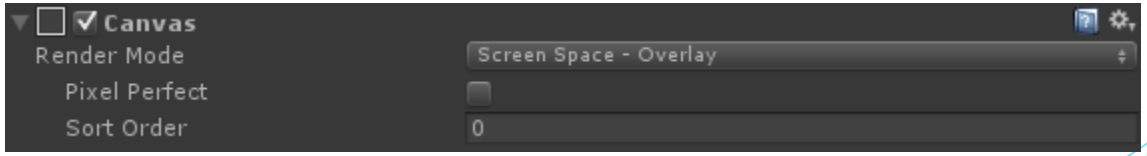
UIs in traditional games

- Called Non-Diegetic UI
- Used to show health, score…
- It does not exist in the world

UIs in traditional games - Unity

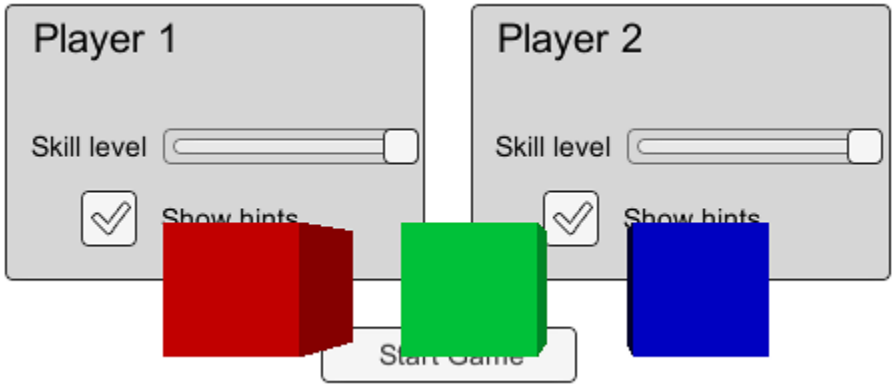
- Screen Space – Overlay
  - Canvas is scaled to fit the screen
  - UI drawn over any other graphics

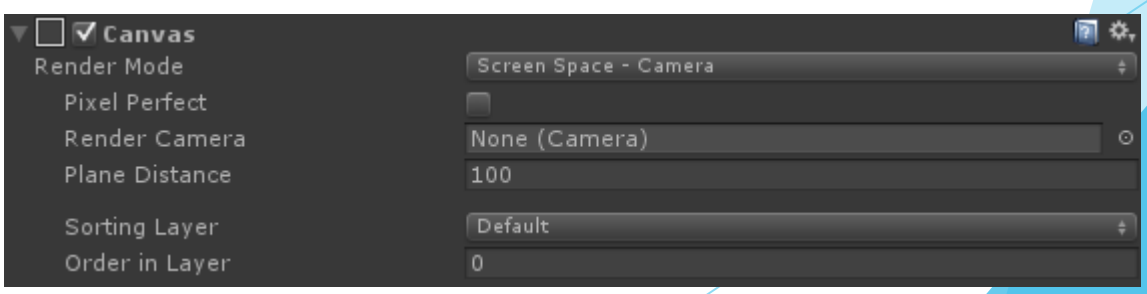




UIs in traditional games - Unity

- Screen Space – Camera Render Mode
  - Canvas is rendered as if it were drawn on a plane object some distance in front of a given camera
  - Objects that are closer than the plane will be rendered in front of the UI



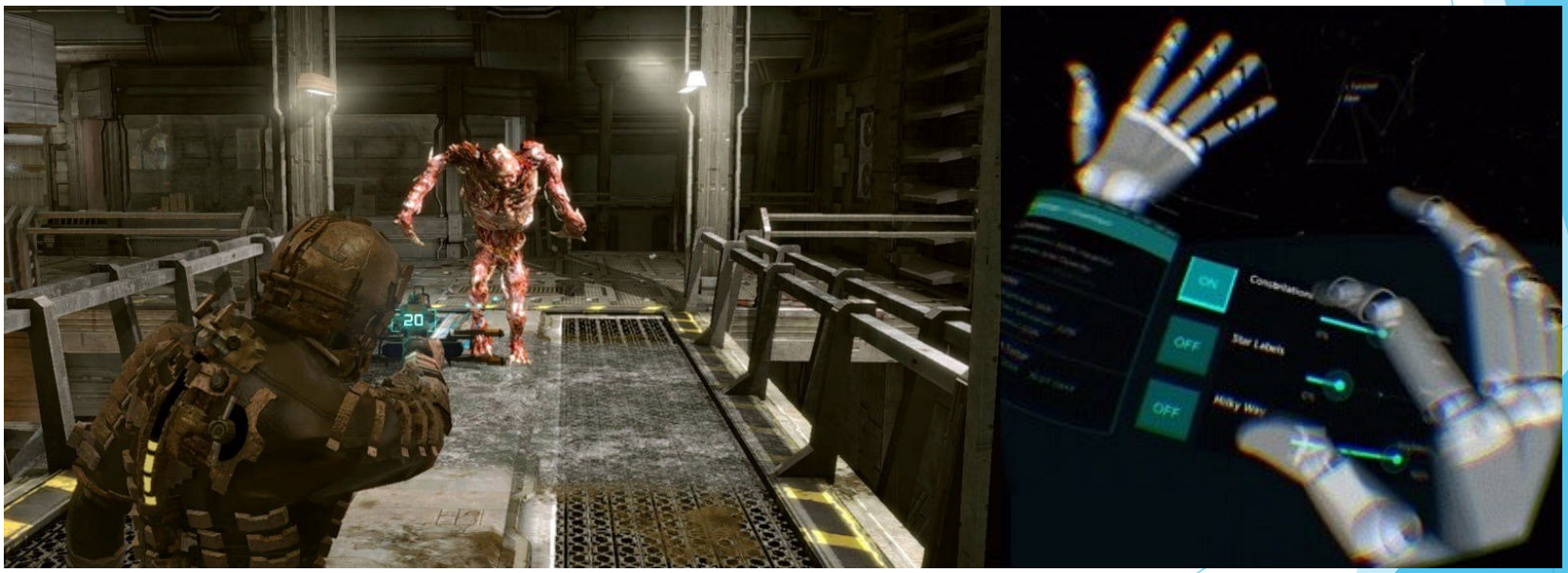


UIs in VR

- Traditional UI does now work in VR
  - Our eyes are unable to focus on something so close
  - Screen Space-Overlay is not supported in Unity VR
- Solution?
  - Diegetic UI or Spatial UI!

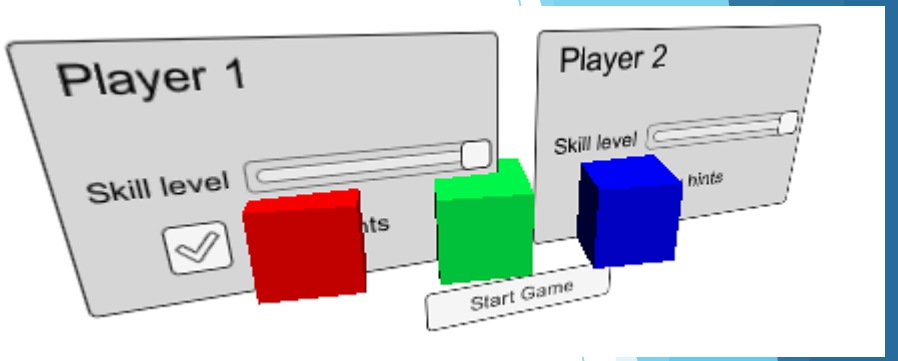
Diegetic UI

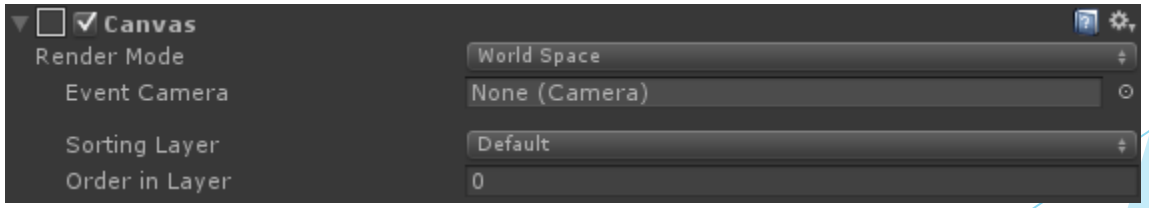
- Interface that is included in the game world -- i.e., it can be seen and heard by the game characters



Spatial UI

- UI inside the world
- Rendered if it were a plane object in the scene
- Plane oriented however you like





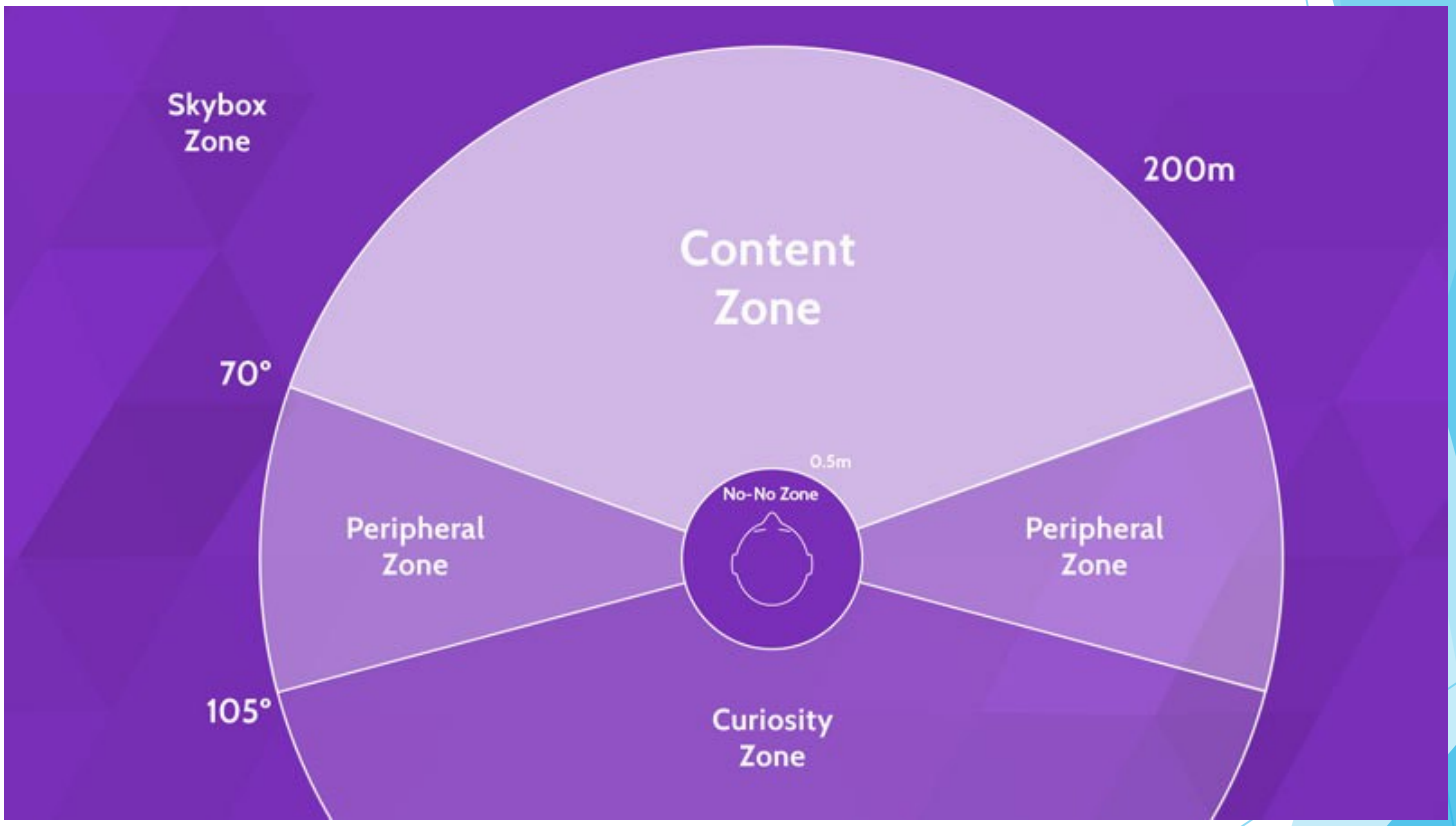
Spatial UI and Diegetic UI

- Where should it be placed?
  - Too close to the user can cause eye strain
  - Too far away can feel like focusing on the horizon

Spatial UI and Diegetic UI

- Where should it be placed?
  - It's best to position your UI at a comfortable reading distance and scale it accordingly.
  - Many developers will initially attach the UI to the camera, so that as the player moves around the UI will stay in a fixed position
    - This can lead to user discomfort or nausea!

Content Zones for VR

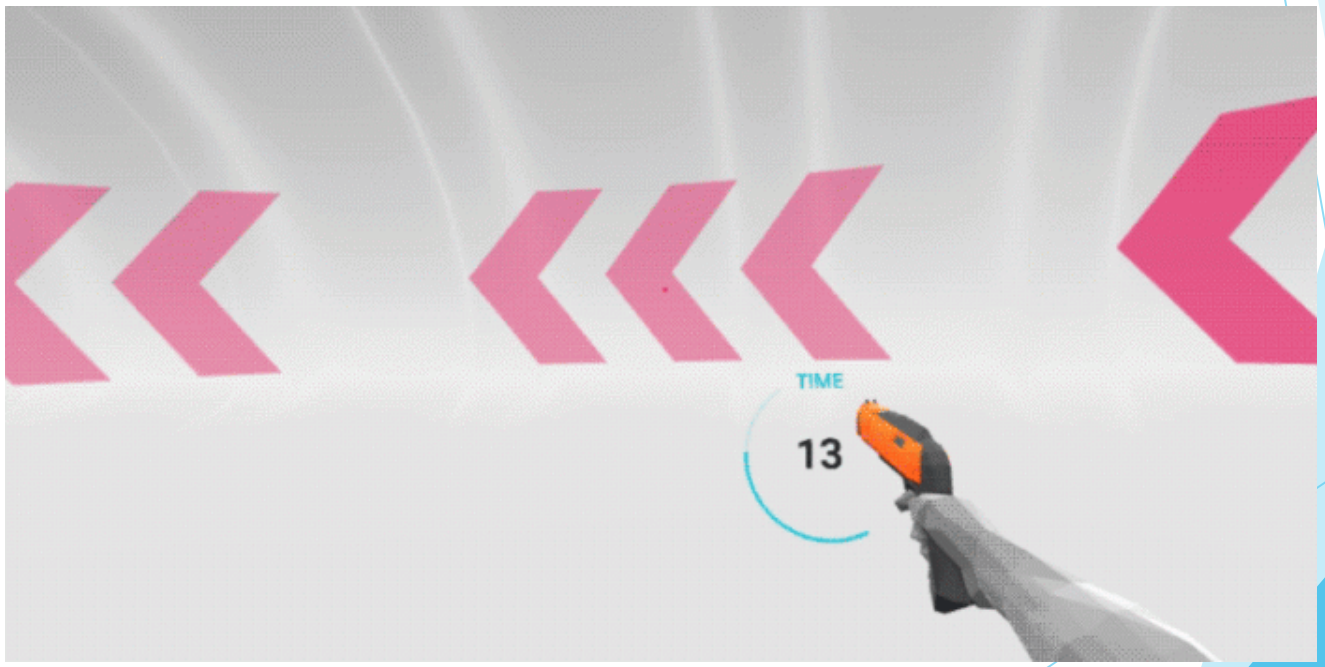


Content Zones for VR

- **Content Zone**: area of comfortable head rotation and view where things still give stereoscopic depth perception.
- **Peripheral Zone:** The area visible with maximum head rotation. Environment will still be seen more regularly, but no long-term content should be put in this zone
- **Curiosity Zone** : the user is literally turning their shoulders and trying with some effort to see what's behind them
- **No-No Zone:** As things get close to the face, the user becomes cross-eyed and experiences eye strain. Nothing should be displayed in this sphere around the head
- **Skybox Zone** - after 200m, the two displays are essentially showing the same image pixel for pixel and there is no depth perception.

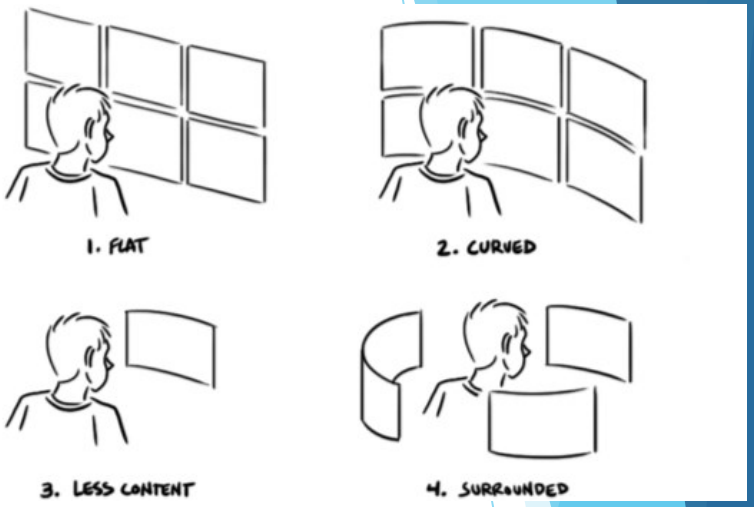
Catching user's attention

- Example of UI used to catch user's attention



Types of UI

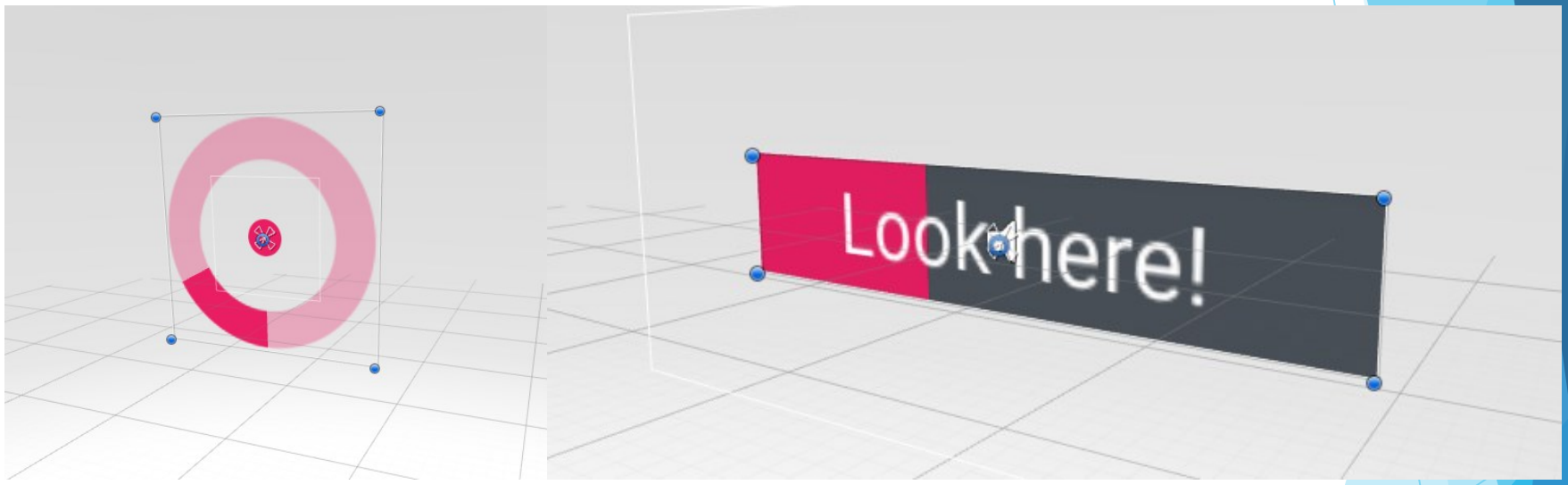
- **Flat:** difficult to read text or images in perspective. There is no sense of grounding in the space.
- **Curved**: The content is curved around the user, easier to read text or images.
- **Less content**: Better, even if that requires some way to move through it.
- **Surrounded**: Hierarchy can be implied by nearness to the cone of focus. Secondary content can be pushed out of immediate view but still remain accessible.



Feedback: Everything Should Be Reactive

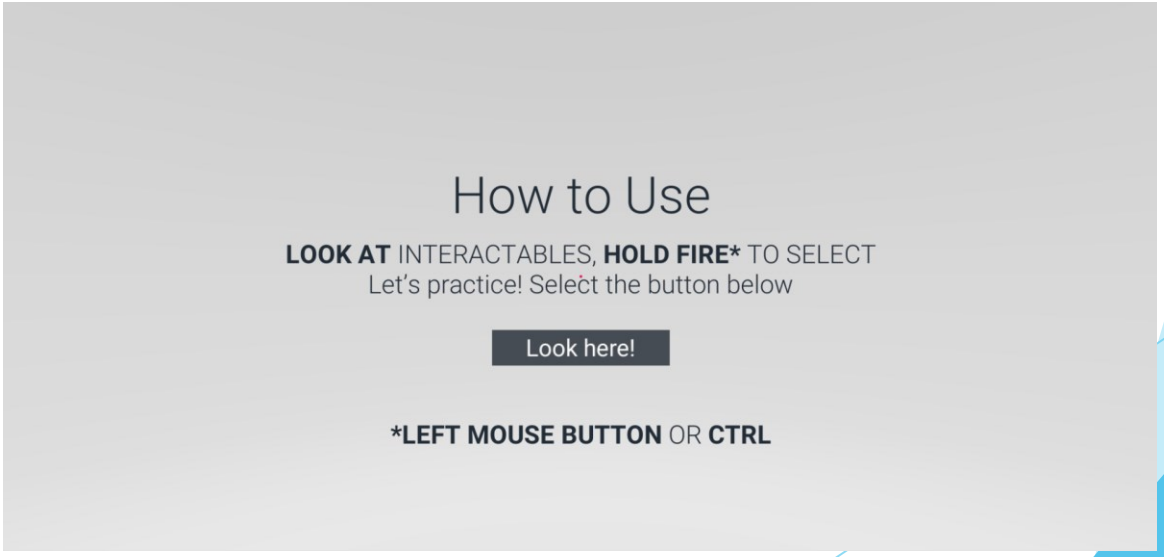
- Every interactive object should respond to any casual movement.
- Any casual touch should provoke movement
  - The kinetic response of the object coincides with a mental model, allowing people to move their muscles to interact with objects.

Interaction and feedback



Gesture Descriptions

- Text- or audio-based tutorial prompts can be essential for first-time users
- Be as specific as possible to get the best results, using motions and gestures that track reliably.



Interactive Element Targeting

- Appropriate scaling
  - Ensures the user can accurately hit the target without accidentally triggering targets next to it.
- Limit unintended interactions
  - Be sure to space out UI elements so that users don't accidentally trigger multiple elements.
- Limit hand interactivity
  - Make a single element of the hand able to interact with buttons and other UI elements.

Unity Demo

- Let’s see a small demo about UI and UX.

Unity Tips & Best practices

Scene management

- Make every scene runnable
  - To speed up debugging, it is important that each scene is runnable, without having to switch between other scenes (e.g., having to start the main menu to test the game scene).

Unity/Package Update

- Be careful when updating Unity or any of the packages you are using (e.g., URP/HDRP).
  - If you're close to a final release, it's probably best not to update.
  - In any case, always test thoroughly, and eventually roll back (via Git, not other weird methods).

Third-party assets

- Don't crack free assets (and no, you can't download them for free from unlikely sites).
- It might be better to import the packages into another project and copy only what you really care about (many packages have many models/textures that maybe we are not interested in).

Build process automation

- As a student, you have Unity Pro for free, which also includes online services, such as Cloud Build (service to build automatically).
- You can synchronize Cloud Build to build every time a commit is pushed to a given branch on a GitHub repository
  - You can use GitFlow and use the Master/Main branch for major updates and build automatically.

Use Assertions

- Assert.IsNotNull(element)
- Assert.IsTrue(A == B);
- These are just two examples of assertions. The magic of Asserts is that they work exclusively in the editor and development build, they are removed from the final build, so you can abuse them in development without worrying about final performance.

Use Coroutine (but be careful)

- Use Coroutines, they allow the code to be interrupted and can help make it much more readable.
- On the other hand, misusing them may make everything more complex, so be careful.

Game/App Localization

- If you plan to develop a multi-language game, the problem should be addressed immediately.
  - You can develop your own translation system (e.g., several files with game strings).
  - Or use the Localization package provided by Unity (still in preview, be careful).

Use C# Properties

- In some case, you may want a variable to be only readable from outside the class. In this case, you can use C# property.
- private float myVariable;
- public float MyVariable => myVariable;
  - In this way, outside the class you will be able to read the value of MyVariable, but you will not be able to change it.

Use C# Properties 2

- Again, properties can allow you to do advanced operation when you read/write a variable.
- private float myVariable;
- public float MyVariable {

```
get
{
        return myVariable; // Here you can do whatever you want
}
set
{
        myVariable = value; // Here you can do whatever you want
}}
```

Use Custom Editor

- If you have advanced script that can be configured from the Inspector (e.g., a script to generate a new level), use Custom Inspector to create more friendly interface.

Custom Class in inspector

- You may have your own class, you set it [SerializeField], but it is now showing up in the inspector.
- Wondering why?
- Use [System.Serializable] public class MyClass {}
- It will magically show up.

Inspector Variable

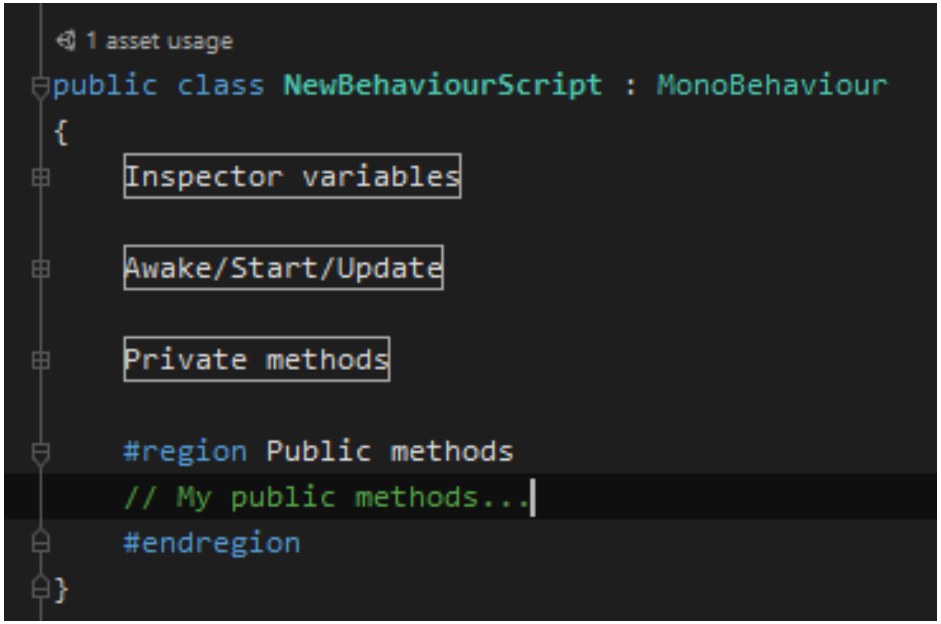
- Pay attention when you declare the value of a variable with the definition of the variable.
- public int MyVariable = 100;
- If you change its value from the Inspector, it will no longer be 100

Do not use Static Field

- Simple and clear rule. You should avoid using static variables as much as possible.

C# Region

- Use region to better organize the code.



LINQ

- Use LINQ to make the code more concise.

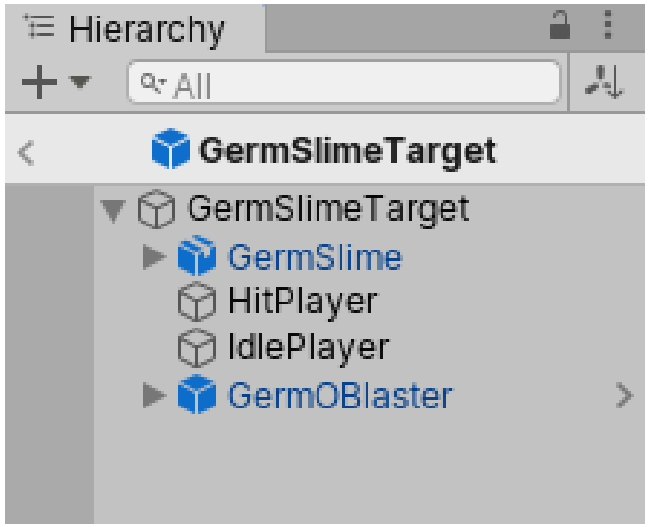
```
GameObject gameObjectToFind = null;
foreach(var go in yourListOfGameObjects)
{
        if(go.CompareTag("fooBar"))
        {
                gameObjectToFind = go;
                break;
        }
}
```

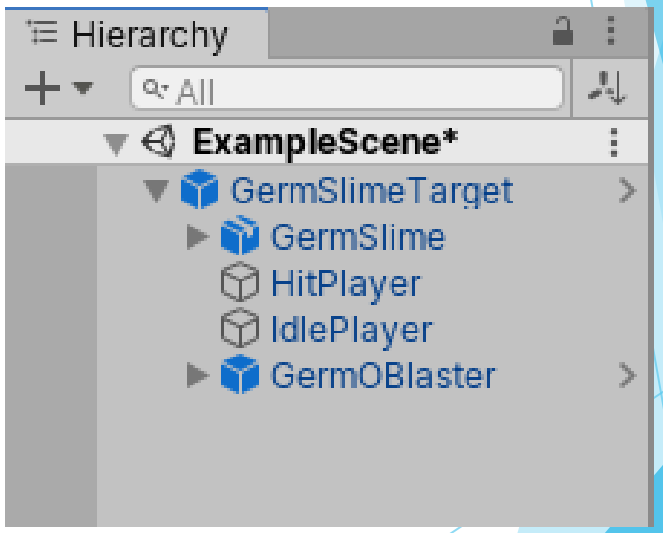
- But pay attention, using LINQ too much could make the code less readable sometimes. Performance is also at risk.

```
var gameObjectToFind = yourListOfGameObjects.FirstOrDefault(go => go.tag == "fooBar");
```

Use prefabs and nested prefabs

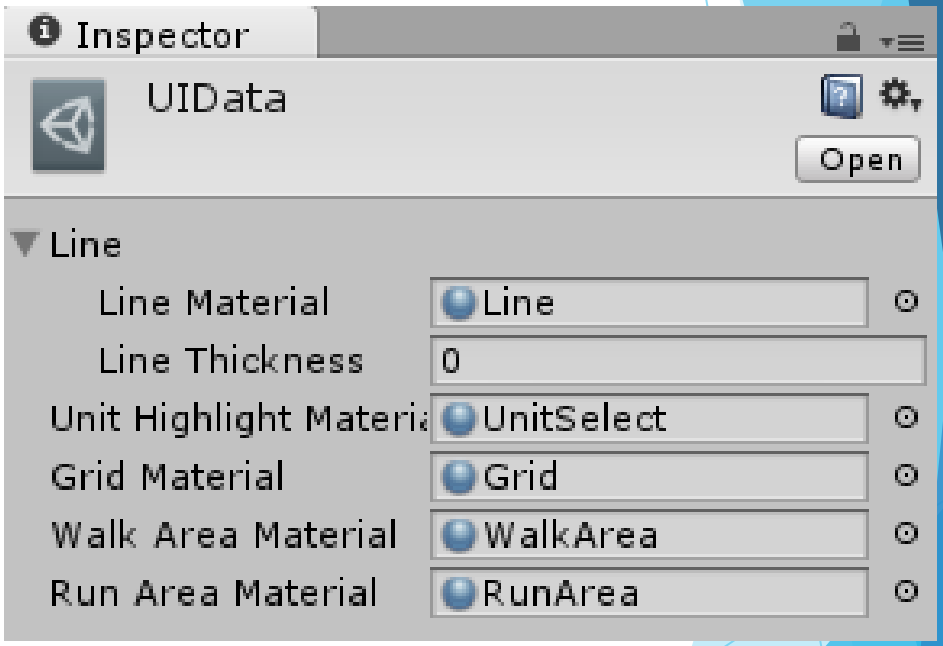
- From 2018.3, Unity allows nested prefab, they can be really useful to allow you the assets reuse.





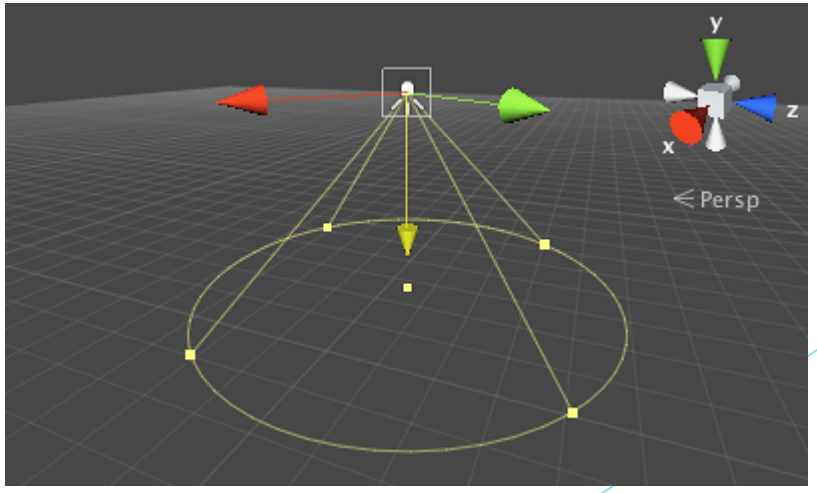
Scriptable objects

- You can use scriptable objects to store information, in this way you do not need to attach a script to a GameObject.



Debugging

- Unity has a few functions that can be helpful for debugging:
  - Debug.Log, Debug.LogWarning, Debug.LogError
  - Debug.Break (pause the execution)
  - Debug.DrawRay & Debug.DrawLine (useful for debugging raycasts, for example)
  - Use Gizmo for visual debugging



Debugging

- Debug.Log, Debug.LogWarning and Debug.LogError allow to specify a second parameter:

```
Debug.Log("Hello", this.gameObject);
```

- In this way, clicking on the message in the console, the GameObject given in input will be selected in the inspector.
  - Really useful if you do not know from which object the log is coming.

- You can also change the style of the text:

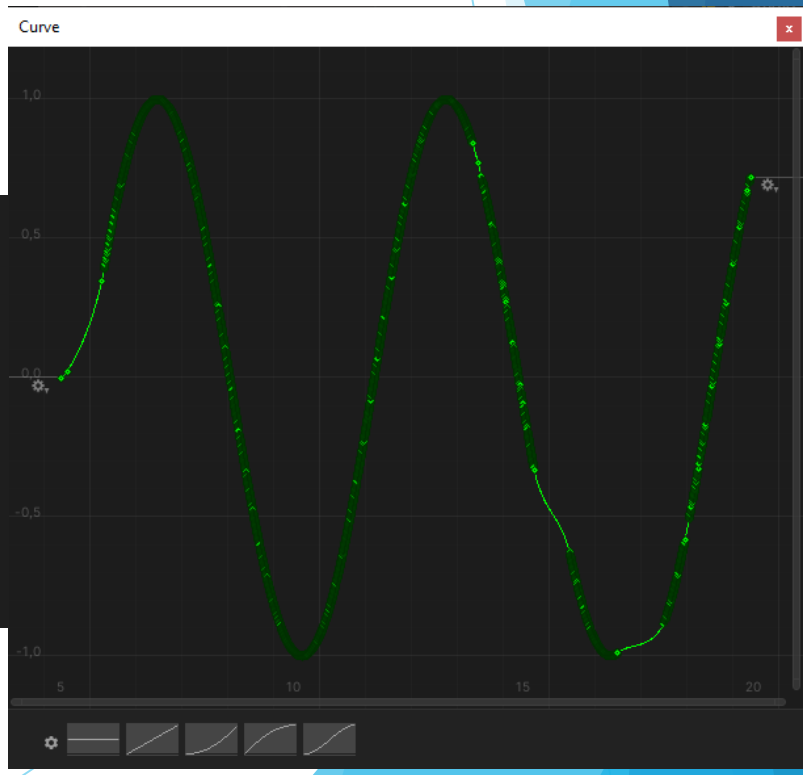
```
Debug.Log("<color=red>Red message/<color>");
```

- For complex things use breakpoints, not Debug.Log
  - I know, Logs are nice and beautiful, but not always effective.

Debugging

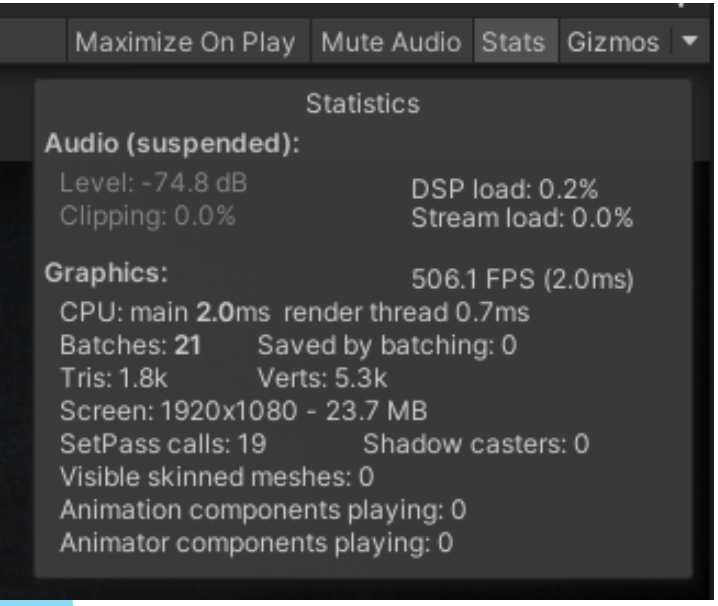
- A variable could change every frame, in this cases, it's easy to find a mistake printing its value every frame.
- A better way, it's to use the `AnimationCurve` class, to show the trend of the variable.

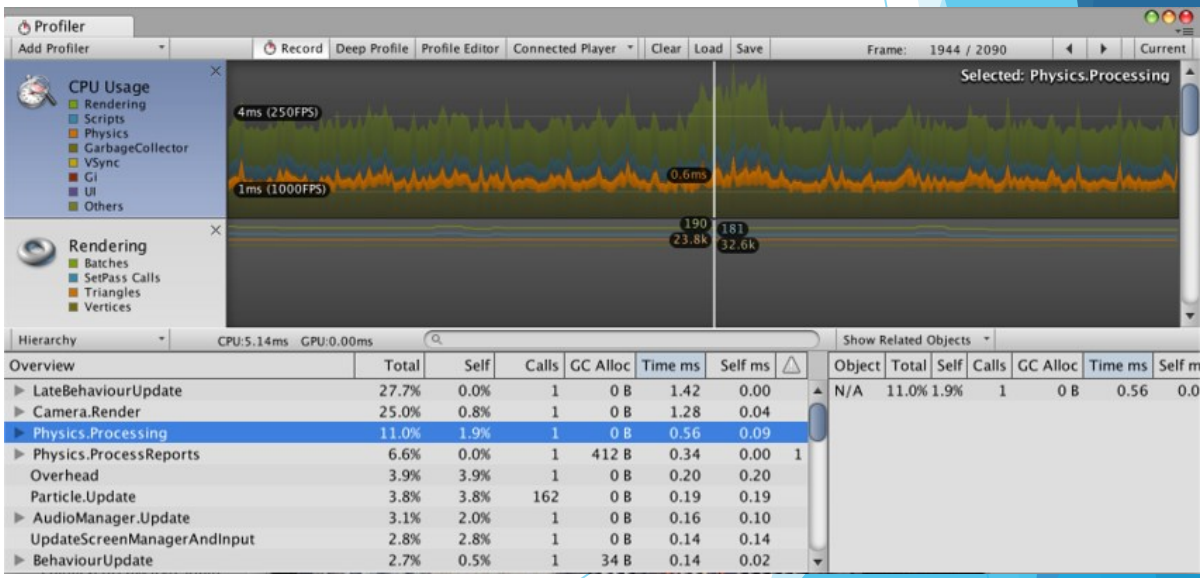
```
public AnimationCurve plot = new
AnimationCurve();

private void Update()
{
        plot.AddKey(Time.realtimeSinceStartup,
Mathf.Sin(Time.time));
}
```



Optimization: Stats & Profiler

- Use Stats button to analyze the performance of your game/app.
- If you want to discover exactly where the performance issue is, you can use the Profiler.





Optimization: Profiler

- If you want to measure exactly how much heavy is a method/piece of code:

```
private void Update()
{
    UnityEngine.Profiling.Profiler.BeginSample("CodeToTest");
    MyMethod();
    UnityEngine.Profiling.Profiler.EndSample();
}
```

Optimization: String Builder

- String concatenation can be very heavy, if performed several times each frame.
- Instructions like:

```
Debug.Log("Hel" + "lo" + " " + "wo" + "rld");
```

- should be avoided.

- Instead of them, you can use the StringBuilder:

```
StringBuilder b = new StringBuilder();
for (int i = 0; i < 5; i++)
    b.Append(i).Append(" ");
Debug.Log(b);
```

Optimization: Cash object

- Access multiple time to properties like transform or position is inefficient.
- Cache them into class variable to optimize.

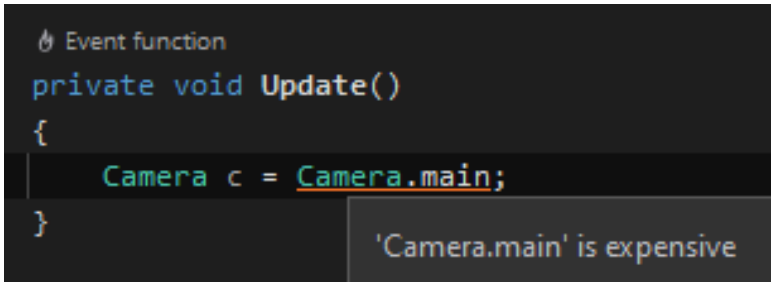
```
Event function
private void Update()
{
    this.transform.position += Vector3.forward * 5;
    this.transform.position += Vector3.left * 3;
}
```

Repeated property access of built in component is inefficient

```
Transform t;
Event function
private void Start()
{
    t = this.transform;
}

Event function
private void Update()
{
    Vector3 position = t.position;
    position += Vector3.forward * 5;
    position += Vector3.left * 3;
    t.position = position;
}
```

Optimization: Camera.main

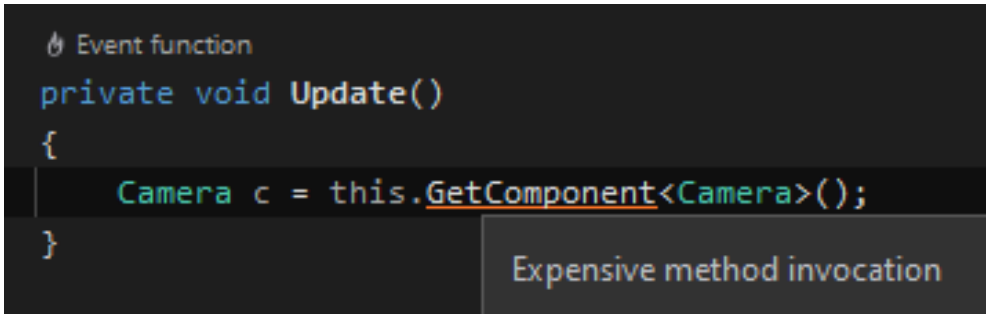
- Never used Camera.main to get the reference of the main camera.



- It has been improved in the last few years, however, right now it is roughly as expensive as GetComponent.

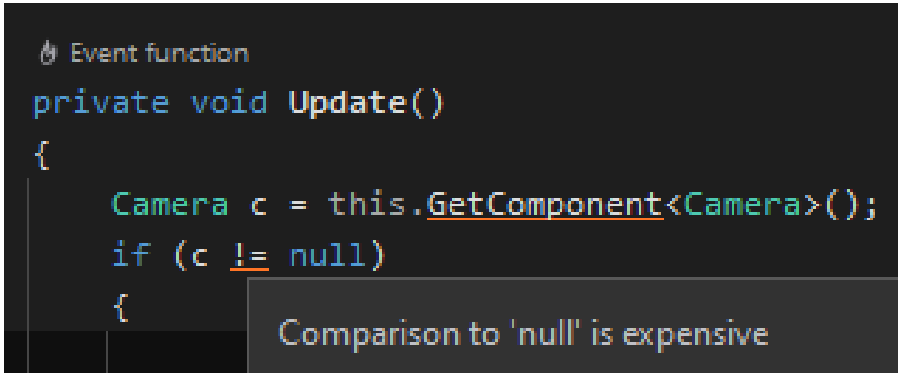
Optimization: GetComponent

- But, unfortunately, also GetComponent is quite inefficient, try to avoid it in Update.



Optimization: Null comparison

- Try also to avoid null comparison in Update.
- This could be harder to avoid, sometimes you maybe can't avoid it.
  - You can try to move something in the Awake/Start and reduce the workload of the Update method.



Debugging: Unity services

- Being a student, you have access to the Unity services, which include Unity Analytics.

```
Analytics.CustomEvent("gameOver", new Dictionary<string, object>
{
        { "potions", totalPotions },
        { "coins", totalCoins }
});
```

- You can create CustomEvent, that will be send to the Unity dashboard. In this way, you can remotely analyze if and how players play your game (e.g., how many of them win/lost a certain level, and so on).

https://docs.unity3d.com/ScriptReference/Analytics.AnalyticsEvent.html

Code execution without empty GameObject

- If you want to execute a piece of code without adding an empty GameObject, you can use the attribute **[RuntimeInitializeOnLoadMethod(RuntimeInitializeLoadType.AfterSceneLoad)]**
- You can use it also with a static class and a static method, without using Monobehaviour or adding it to the scene.

```
[RuntimeInitializeOnLoadMethod(RuntimeInitializeLoadType.AfterSceneLoad)]
public static void Do()
{
        Debug.Log("Do()");
}
```

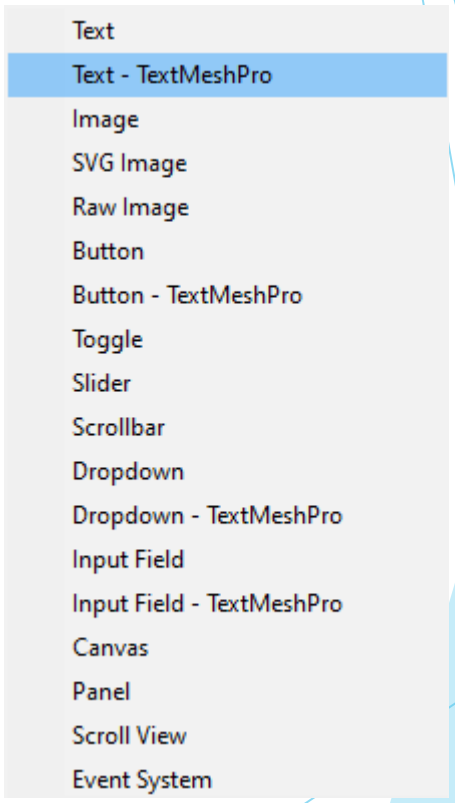
Calculating distance between object

- Use sqrMagnitude instead of Vector3.Distance to calculate the distance between two points, it's faster.

```
if (Vector3.Distance(a, b) < 100)
{

}

if ((a - b).sqrMagnitude < 100 * 100)
{

}
```

TextMeshPro vs Text

- Use TextMeshPro components (e.g., Text, Button), do not use old standard component like «Text».



Attribute

- [SerializeField]
- [HideInInspector]
- [FormerlySerializedAs]
- [Header(«Title»)]
- [Range(1,5)]
- [Tooltip(«bla bla bla»)]
- [RequireComponent(typeof(AudioSource))]
- [TextArea(minLines: 3, maxLines: 6)]

- These are just some examples of Attribute that can be used and that can help you.

[SerializeField]

- If you do not want to set Public all your variables in order to be able to modify them from the inspector, mark them with [SerializeField], to allow them to be visible.
- [SerializeField] private float myVisiblePrivateField

HelpUrl Attribute

- In complex project, remember to have an update documentation.
- You can also define a url that will be open when you click on the help button of a particular script.

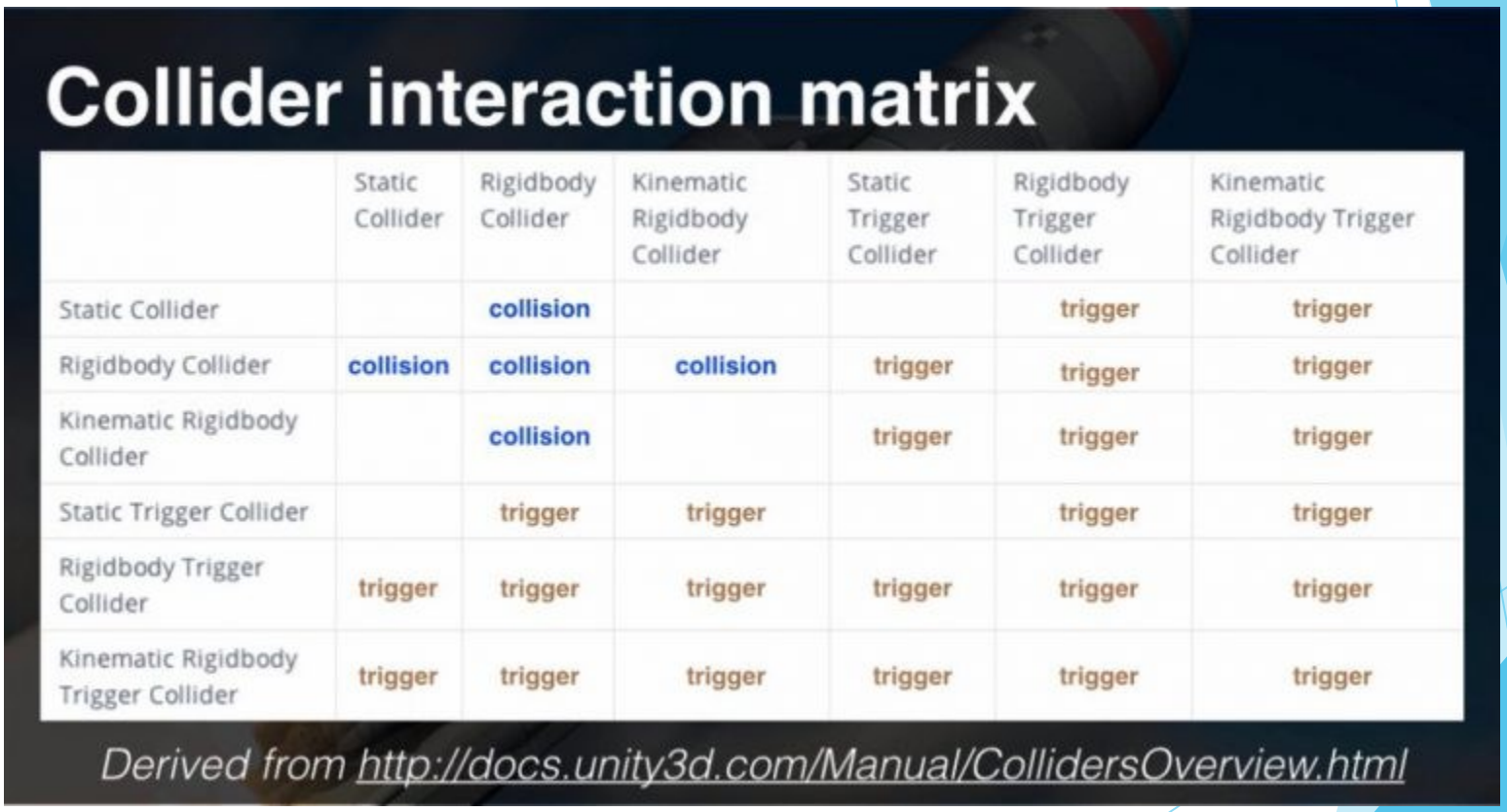
```
using UnityEngine; using UnityEditor;

[HelpURL("http://example.com/docs/MyComponent.html")]
public class MyComponent
{

}
```

MenuItem/Context Menu

- MenuItem and ContextMenu attributes allow you to define an item in the menu or context menu that call a method. Really useful in editor, if you want to test/try a certain method.

```
[MenuItem("MyMenu/Do Something")]
private static void DoSomething()
{
	Debug.Log("Doing Something...");
}

[ContextMenu("Do")]
Private void Do()
{
	Debug.Log("Doing Something...");
}
```

Collider interaction matrix

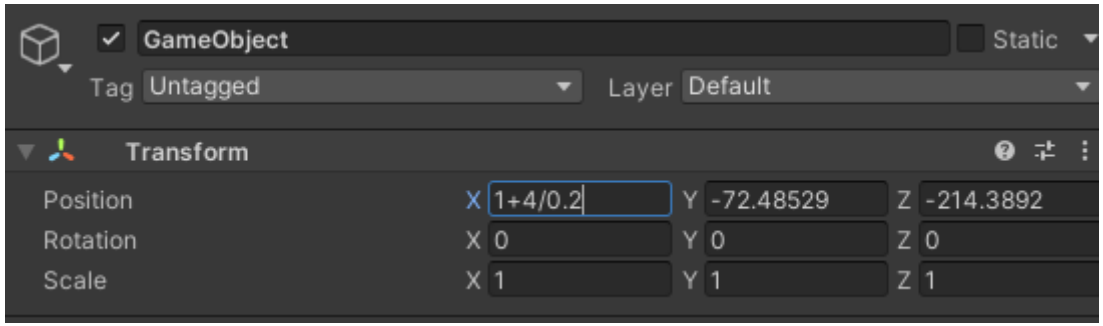


	Static Collider	Rigidbody Collider	Kinematic Rigidbody Collider	Static Trigger Collider	Rigidbody Trigger Collider	Kinematic Rigidbody Trigger Collider
Static Collider		collision			trigger	trigger
Rigidbody Collider	collision	collision	collision	trigger	trigger	trigger
Kinematic Rigidbody Collider		collision		trigger	trigger	trigger
Static Trigger Collider		trigger	trigger		trigger	trigger
Rigidbody Trigger Collider	trigger	trigger	trigger	trigger	trigger	trigger
Kinematic Rigidbody Trigger Collider	trigger	trigger	trigger	trigger	trigger	trigger

https://docs.unity3d.com/Manual/CollidersOverview.html

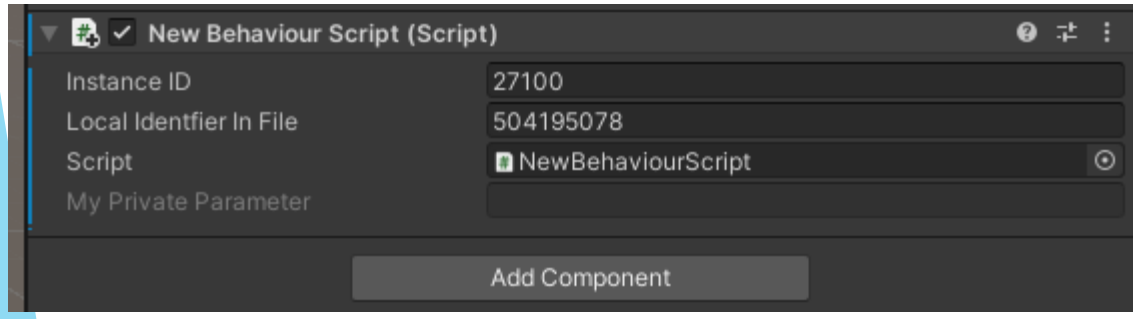
Math in Inspector

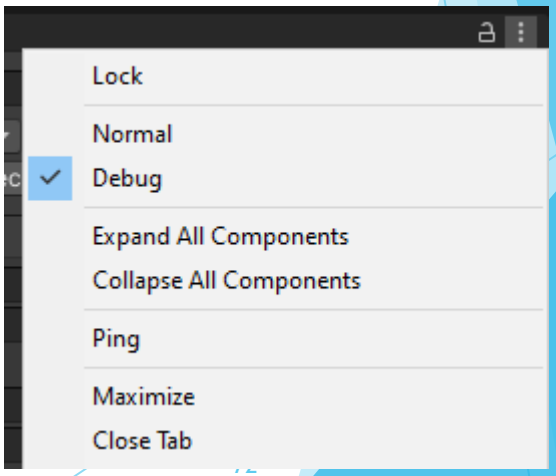
- You can type custom operation in inspector, pressing Enter will calculate the result for you.



Inspector: Lock & Debug

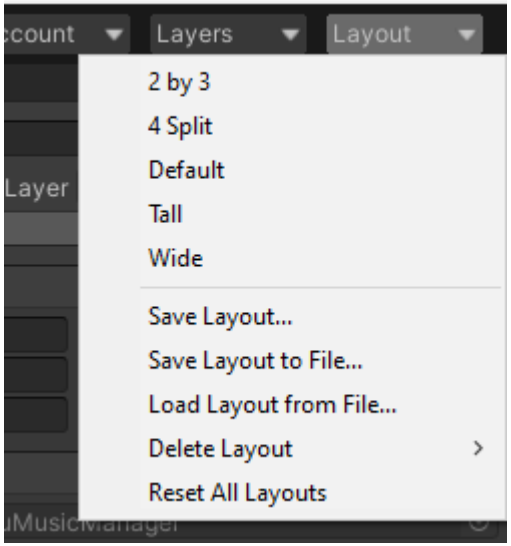
- You can lock the inspector. Opening a second inspector will allow you to compare the values of the current selected object, with the previously selected one (represented by the locked inspector).
- It is also possible to enable the Debug mode. This will allow you to see also the value of the private parameters.

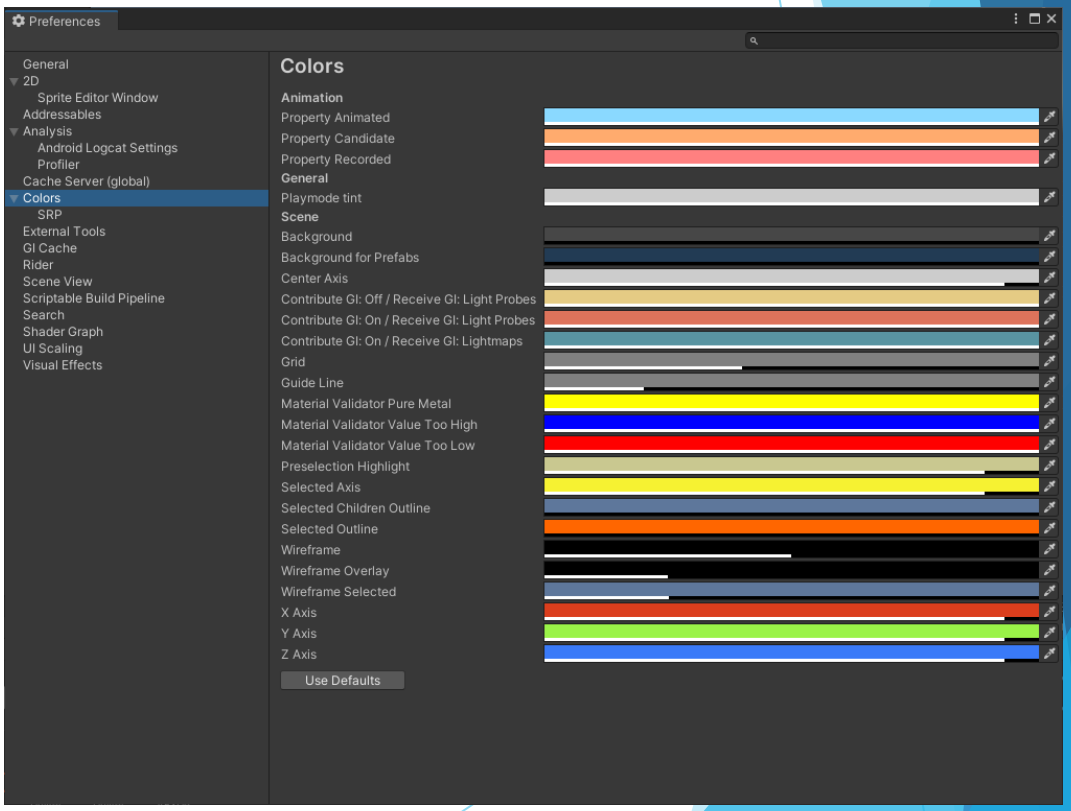




Editor Customization

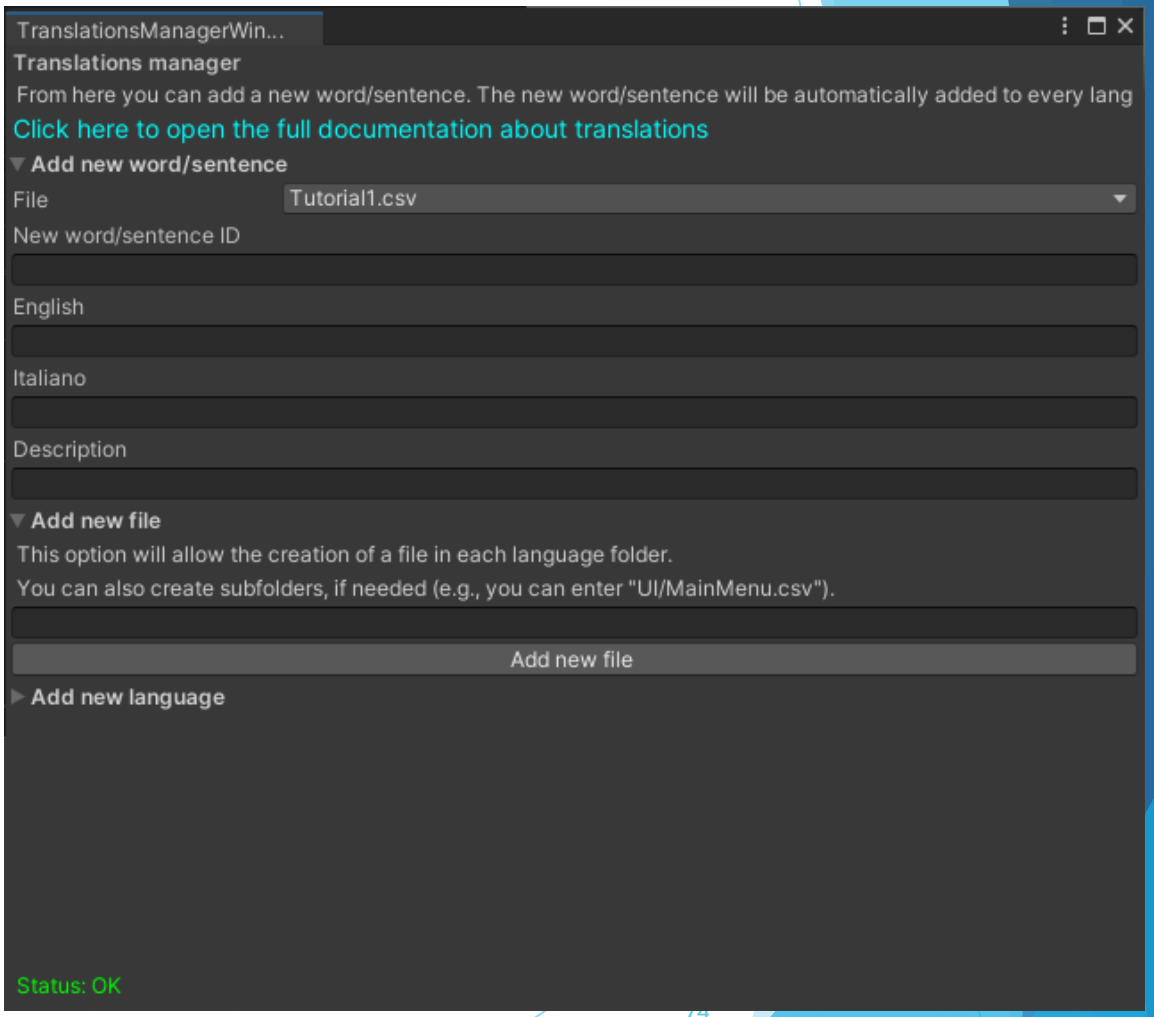
- Remember that you can customize the color and layout of the editor as you prefer.





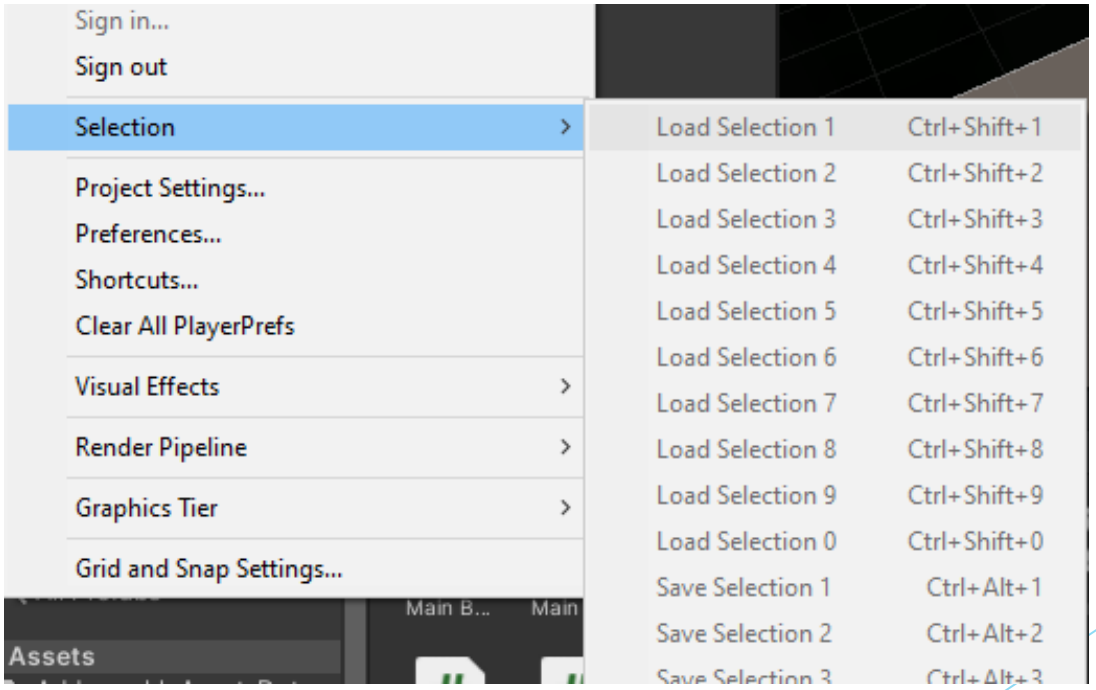
Editor Windows

- You can create your own windows and tools to manage whatever you want.
- In the same way, you can also create custom inspector.



Selection

- You can save and load selections of one or multiple objects.

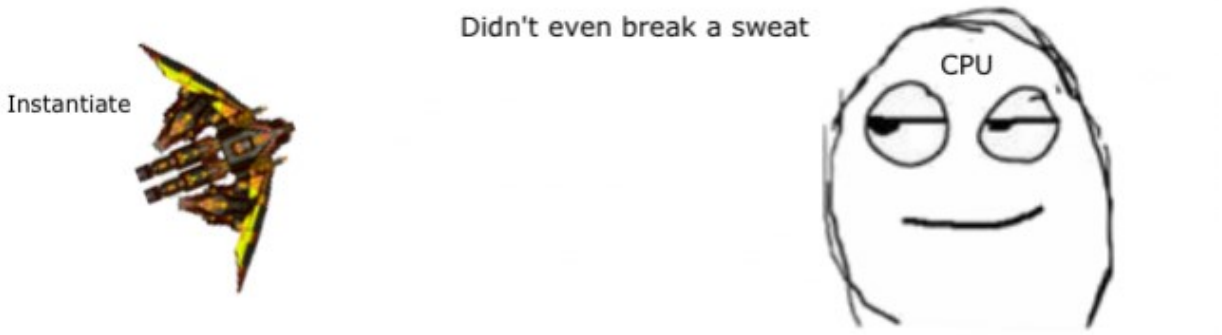


Design Pattern: Singleton

- Singleton class can be very useful.
- You can refer them from anywhere in the code, and they ensure you that there will be one and only one instance of that class.
- But don't abuse it. Static fields are always bad to use.

```
public class SomeClass : MonoBehaviour {
    private static SomeClass _instance;

    public static SomeClass Instance { get { return _instance; } }


    private void Awake()
    {
        if (_instance != null && _instance != this)
        {
            Destroy(this.gameObject);
        } else {
            _instance = this;
        }
    }
}
```

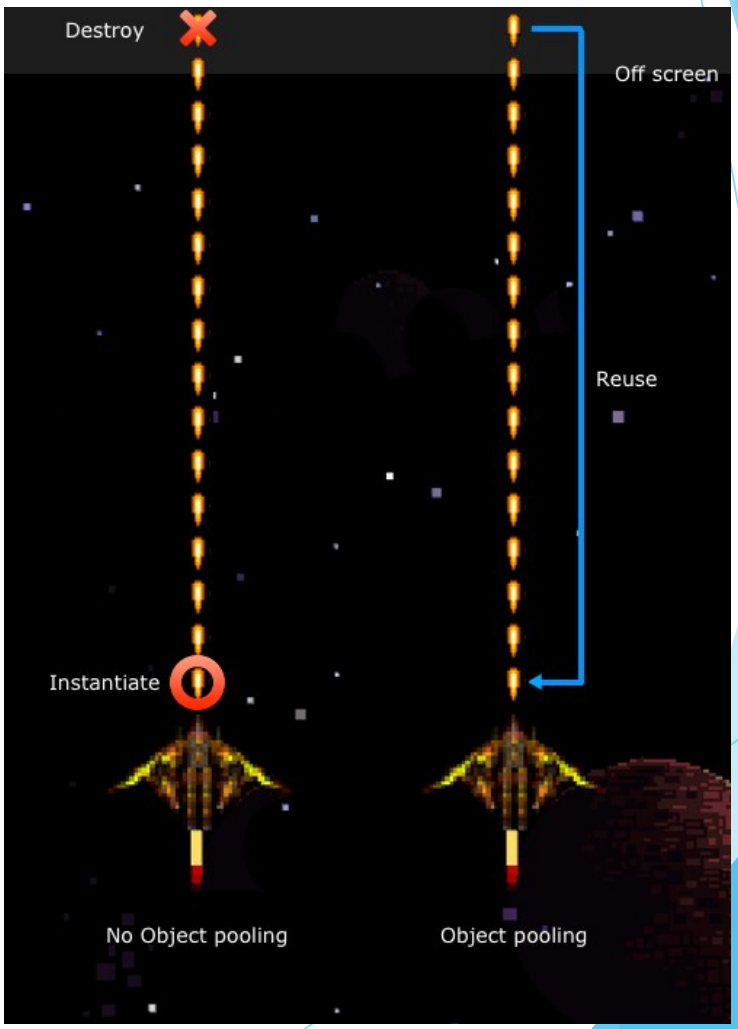
Design Pattern: Object Pooler



Source: https://www.raywenderlich.com/

Design Pattern: Object Pooler

Source:
https://www.raywenderlich.com/



Design Pattern: Object Pooler

- Instead of instantiating and destroying objects repeatedly (very computationally heavy), it is more convenient to disable and reuse objects.
- For bullets, for example, after a few seconds and/or when they are no longer visible, we can automatically disable and reuse them.
- Beware of initialization, disabling and re-enabling an object will not invoke Awake / Start again.
- There are several implementations of this pattern on Google.

Git/Github

- Use Git (or another similar software)
  - Always use Git (even when working alone). If you don't have enough practice, practice.
  - You can use it both from the command line and from the GUI (e.g., SourceTree, GitKraken - free for students).
  - You can (and maybe should) upload the repository to Github (Pro version free for students).

Git

- It is necessary to use Git and Github during the development of the project.
- In the next slides we see a very simple guide on how to use it.

- First thing: download Git
- Open Git Bash (right click wherever you want -> “Git Bash Here”)
  - `git config --global user.name «your username»`
  - `git config --global user.email «your.mail@domain.com»`

GitHub

- Github is required to upload your project on a server.
- You could use Git without Github, but if you want to work with someone else or have a backup online, you must use Github or similar services (e.g., BitBucket).
- For the project Github is required, so you need to register and account.

Git – Create new repository

- To create a new repository, just type:

```
git init
```

- This will create a new repository. Now, we need to commit something.
- A commit represents a snapshot of your work. This way, you can always keep track of your changes and possibly revert to a previous version if there is something wrong with the current version.

Git – Open an existing repository

- If you want to open an already existing repository, you have to use the following command:

```
git clone UrlOfTheRepositoryYouWantToClone
```

- This will start the download of the repository located at the url you have typed.

Git – Git Status

- Before commit something, you probably want to control what you have modified.
- You can do it with the command

```
git status
```

- Now you can add the file you want to commit:

```
git add your_file_name
```

```
git add .
```

- This will move your file to the stage area, they are ready to be committed.

Git – Git Commit

- Once you have selected all the files you want to commit, you can commit them using the command:

```
git commit –m "Description of the commit"
```

- Now you have successfully created a new commit.
- You are now ready to push it to a remote repository, in this way, your colleagues (or yourself, with another computer) can download the updated version of the project.
- The first time, if you have never pushed the repository to a remote server, you must setup the remote url with the command:

```
git remote add origin <REMOTE_URL>
```

- This must be done only once, the first time. You do not need it if you have cloned an already existing repository.

Git – Git push

- You can now push your commit with the command:
  - git push origin main
- Where main is the name of your branch.

- It could happen that you can't push your commit to the repository, this happens if you are not up to date with the remote repository commit.
- In this case, you must before pull the update, merge them with your local repository, and then push all the update.

Git – Git pull

- To download the commits from the remote repository, you must use the following command:

```
git pull
```

- Be careful, if you have not pushed commit, there is the possibility that you have one or more conflicts. A conflict happens when the same file has been modified both on the remote repository and on your local repository.
  - In this case, you must merge the two files (or, eventually, discharge your modification).

Git – Git checkout/Branch

- One terrible day, you have to do a huge refactoring of the project; this will render the project unusable for a few days, until you have completed the process.
- You do not want to break the project for all your colleagues. But, at the same time, you do not want to work for days without pushing your commit; if your computer dies, you can lose your work.
- In these cases, you can switch to another branch, you will be able to push your commits without break the work of your colleagues.

Git – Git checkout/Branch

- To create a new branch:

```
git branch nameOfMyBranch
```

- To switch to another branch:

```
git checkout nameOfMyBranch
```

- Now, you can commit and push your work on the branch you have checked out.

Git – Git checkout/Branch

- Once you have finished your fantastic refactoring process, you are now ready to merge your work with the main branch.
- First of all, you need to switch to the main branch:

```
git checkout main
```

- Eventually, do “git pull” to update your main branch, if not updated.
- Now, merge your work with the main branch:

```
git merge nameOfMyBranch
```

- Be prepared to merge problems, you have to fix them before being able to merge and push your work.

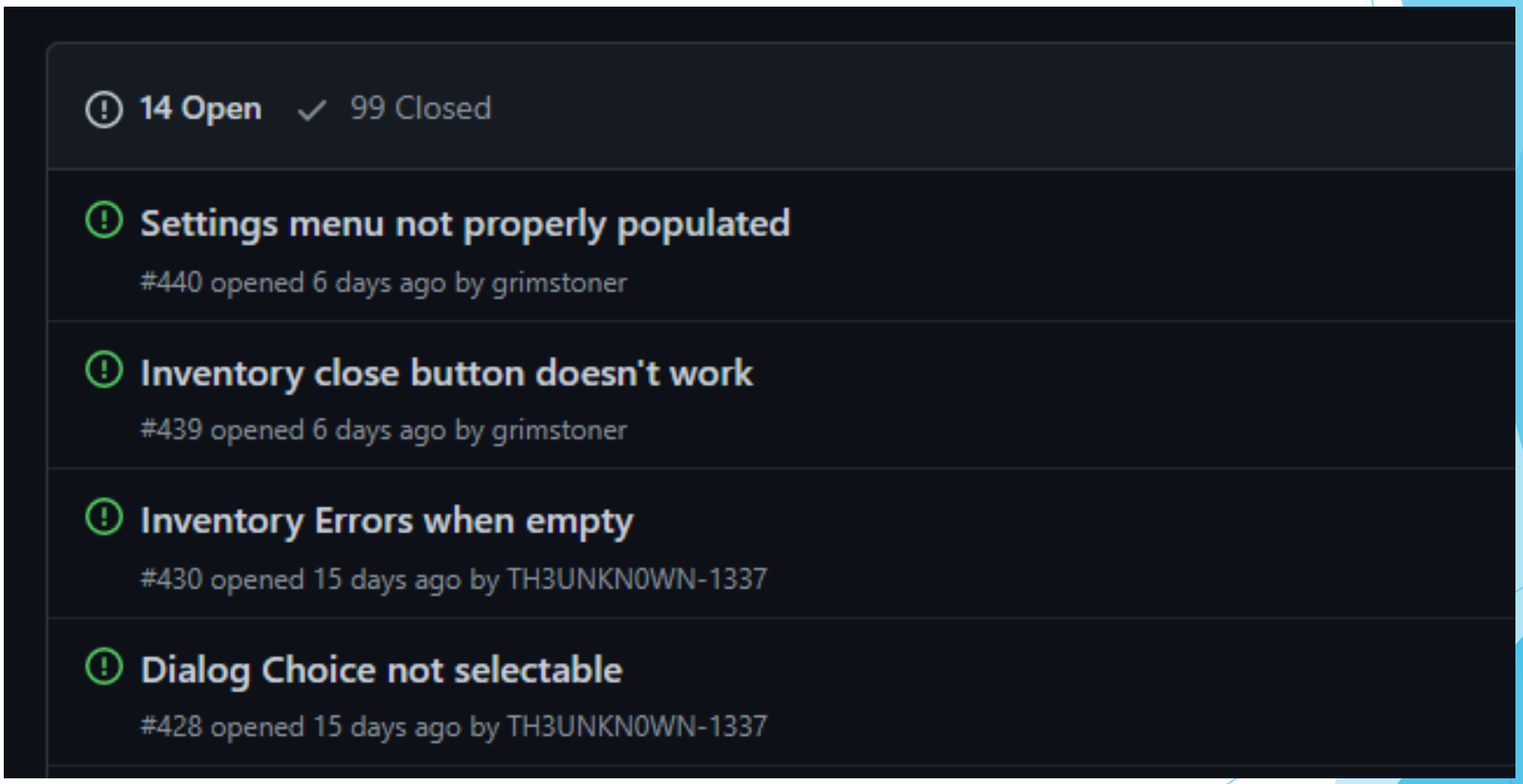
Git – Git checkout/Branch

- This was a very simple and short guide to Git; actually, there are many other features and commands that I have not mentioned. Like:
  - `git log, stash, revert, cherry-pick`
  - Git flow
  - Git Hooks
- If you are interest you can check out a more advanced tutorial/video to see all the features.

Github

- Github is useful not only for collaborate with your colleagues and as a remote backup, but it also has other very interesting features:
  - Issues
  - Wiki
  - Marketplace

Issues


14 Open
99 Closed
Settings menu not properly populated
#440 opened 6 days ago by grimstoner
Inventory close button doesn't work
#439 opened 6 days ago by grimstoner
Inventory Errors when empty
#430 opened 15 days ago by TH3UNKN0WN-1337
Dialog Choice not selectable
#428 opened 15 days ago by TH3UNKN0WN-1337

Wiki

Home

Ciro Continisio edited this page 24 days ago · 12 revisions

Welcome to the Wiki for Open Project #1: Chop Chop!

This is intended to be an instruction manual. (Almost) all of the game's systems and components are explained here, so you can download the project using Git, and start exploring it using this Wiki as documentation.

All pages are in continuous evolution as the game changes during its development, so check them out again before you modify a system, to make sure you are up to date with the changes.

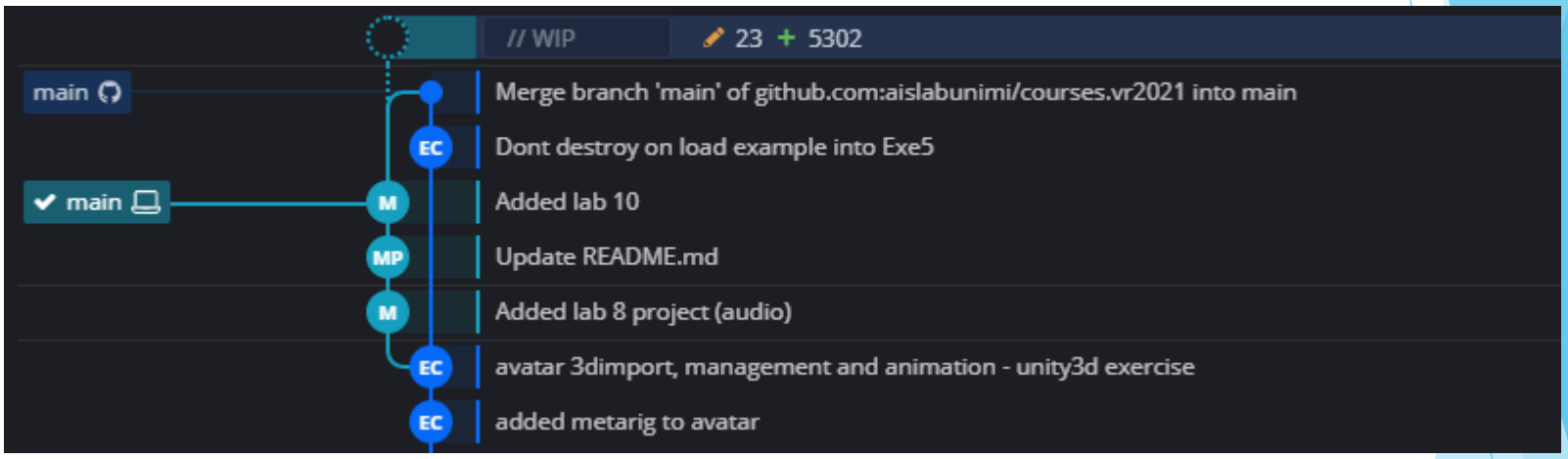
If you want to contribute to the game, don't forget to read the Contribution Guidelines first! They contain a lot of important things you should know before you make your first contribution. And especially for programmers, check out the Code Conventions or, for artists, the Art Contribution guidelines.

Git: GUI

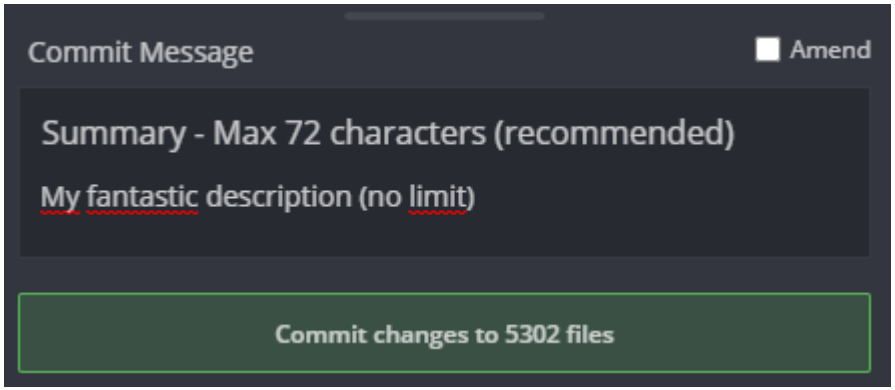
- It's possible to use git through a Graphical User Interface (GUI).
  - E.g., GitKraken, SourceTree, GitHub Desktop
- Even if you use one of those, it's important that you are still able to use the command line.

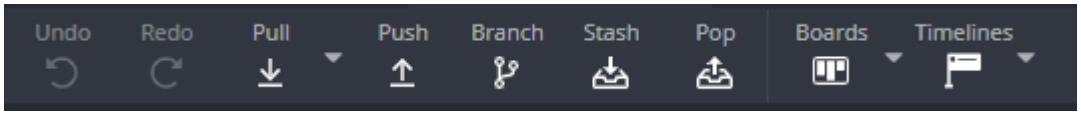
Git: GUI – Add files

Git: GUI – Commit lists



Git: GUI – Commit, Pull, Push, Branch





Unity Learn

- Questions, want to learn new things?
- Best recommeded sources for learning
  - Unity Learn
  - Unity Forum
  - Youtube (e.g., Brackeys)
- Best recommended source for problems
  - Stackoverflow