First project

Desktop

**Project registration (GitHub)**

Required material: Units 1 and 2

Duration of the project: 2 weeks

Submission deadline: 07/11/22 until 17:30 on GitHub

In this project we will create a user interface for a digital desktop environment containing icons and windows. The base must be fully redeemed, then select and redeem upgrades of at least 5 points.

Required:

- Opening windows 3 points
- Dragging icons | Window drag 2 points
- Recycling bin 1 point
- One window limit | Watch | compass

The document contains tutorial appendices for topics common to several upgrades. Any upgrade will refer to the relevant appendices.

**Highlights**

The tutorials detailed in the document are only a starting point. Read the tutorial in its entirety before starting the application. Be sure to meet all the requirements as stated in the characterization definitions.

Remember that the development qualities of the course must be met. Make sure before submitting the project that you meet all the requirements.

in this project (only) **Do not use material from advanced units**, we will use the tools studied in units 1 and 2 only. Later we will learn tools that facilitate the application of many of the features in this project (physics, coroutines and much more) but before that it is important that we practice working with the basic tools.

Use the means available to you to create a readable and clear user experience for users, try to consider the visual design aspects as well.

(A 'brief' is a starting point, raw material that can be leveraged into the creation of a project that is yours and brings you to expression!)

What does a full submission include?

- source code in (Runs without errors for me) GitHub
- perform! At least one every week Commit
- README file In which the selected upgrades are indicated
  - Upload a video of the product

Successfully!
opening windows (Required)

Characterization

- The screen must contain at least two icons.
- By clicking double (left mouse button) on an icon will open a new window.
- The window that opens should contain a button, by clicking (left mouse button) on the button the window will close.
- Objects must be created and deleted for the windows, it is not enough to hide/show the object.
- Each icon opens a different type of window (window content is different).
- The contents of the window not necessarily to be functional.

Instruction

Use the following recipes to realize the characterization:

**Click detection**

Two components are needed for click detection:

1. One of the Collider 2D components.

**Double click**

1. Define a maximum period of time between two clicks so that they count as a double click.
2. With each click, the time at which the click was made is saved (Time.time) - https://docs.unity3d.com/ScriptReference/Time-time.html
3. Check whether the difference between the current time and the saved time is lower than the defined time period.

**Creating an object**

1. Create a prefab that we want to duplicate.
2. Define a public field of type GameObject
3. In the inspector window, place the prepub in the field we created.

**Deleting an object**

1. Access to the object of the component:: GameObject
2. Access to a parent object:: transform.parent.gameObject
**Windows over Icons**

1. Create a new Sorting Layer for windows, position the layer so that it renders above the default layer.
2. Make sure that the icons are in the default layer, while the windows are in their designated layer.
Dragging icons (3 points)

Characterization

- While clicking on an icon, the positions can be changed according to the position of the cursor.
- The icon position must be updated at any time during the drag.
- When dragging, the dragged icon is rendered (displayed) on top of the other icons.
- After releasing the drag, the icon must be attached in animation to the position next to it in the grid (or another arrangement).

An example of an extreme case that would be desirable to address in order to improve the experience we create: preventing the possibility of placing an icon on top of another icon - but we will not deal with that in this project.

Instruction

We use the appendices Drag: continuous interaction and Where is the mouse cursor located?

Transparency

1. Create a field for a normalized value (0,1) representing the desired transparency at the end of the float animation.
2. At the beginning and end of the drag, change the desired value.
3. At the event Update gradually update the color of the icon:
   3.1. `Time.deltaTime` will help to determine the amount of change in this frame relative to a specified rate. - https://docs.unity3d.com/ScriptReference/Time-deltaTime.html
   3.3. `attache recipies` : access to render component, change color in code.
   3.4. White color keeps the original color - new Color(1, 1, 1, a)

Render order

Assuming that all the icons are in the same Sorting Order (and in the same layer), we can increase the property at the beginning of the drag `sortingOrder` of a component and reset it back at the end of the Sprite Renderer interaction - https://docs.unity3d.com/ScriptReference/Renderer-sortedOrder.html

Attaching to a grid (or other arrangement)

You can, for example, use `Mathf.Round` to round the cursor’s current position in the world. - https://docs.unity3d.com/ScriptReference/Mathf.Round.html
Location animation

1. Create a Vector3-type field (even in two dimensions, the location is represented in three dimensions) at the end of the drag we place the destination location where we want the icon to reach.


3. Make sure that there is no conflict between the determination of the position when dragging and the snapping animation to the grid, for example by Tracking whether there is currently an active drag.
Recycling basket (3 points)

A game where the player’s goal is to keep the recycling bin empty before it explodes.

Characterization

- Count down a random amount of time between $\alpha$ and $\beta$ seconds and then the recycle bin fills up (sprite replacement).
- The player must empty the recycle bin within $\gamma$ seconds, by clicking on it with the right mouse button.
- During the last second left for the player, the icon will gradually turn red (or any color additional highlight depending on your application).
- If the player has not had time to empty the recycle bin, type BOOM in the Console window and quit - The icon is displayed in red (or a color of your choice).
- If the player has had time to empty, the display is reset recycle Bin and starts a new random count.

Instruction

If and Use ‘drag icons’, the recycle bin can be excluded if it helps you avoid from collision so that each upgrade colors the sprites.

Use the appendix recipes To realize the characterization:

- Manual countdown
- Random duration in range
- Access to the rendering component
- Change sprite in code
- Color change in the code
- Is the cursor above me?

right button

1. We would like to use the event as OnMouseUpAsButton But we cannot do this because which only responds to the left button.
2. Pressing the right button of the past can be detected using Input.GetMouseButtonUp.
3. But... we have to make sure that the mouse is really over the icon when clicking, for that we will use the recipe ‘Is the cursor over me?’

Color interpolation

To drive in a combination (switching between) colors you can use Color.Lerp. -
https://docs.unity3d.com/ScriptReference/Color.Lerp.html
Dragging windows (2 points)

Characterization

- Each window has an area that when you click on it and move the mouse you can move the position of the window on the screen accordingly to the position of the cursor.
- When a window is opened it must appear above the other open windows.
- When you start dragging a window, it must appear above the other windows and stay in front until another window will open/drag over it.

Instruction

- Appendices are used Drag: continuous interaction and Where is the mouse cursor located?
- If necessary, the location of the parent can be accessed and moved by.transform.parent

Render order

1. Prefab
   1.1. Add to the prefab of the window a Sorting Group component that renders the entire window as one unit and avoid a situation where an element from a rear window appears above the front window.
   1.2. For the children of the window, you can use Order in Layer or the order of the objects in the hierarchy to determine the rendering order.

2. Script
   1.2. Add a static private field of type int to the drag script (int static privat).
   2.2. Add a public field of type SortingGroup and place the component we added in it to the prefab (it is also possible to place a component that is attached to the parent/child and not necessarily to the object itself).

   3.2. When the window appears (Start event) or when you start dragging it, change the value of sortingOrder.SortingGroup to the value of the static field and add 1 to the existing value in the field (because the static field is common to all windows and thus we guarantee that we will always have access to the largest value among the open windows) - https://docs.unity3d.com/ScriptReference/Rendering.SortingGroup-sortingOrder.html
One window limit (1 point)

Please note, some of the features in this upgrade only apply to those who have implemented 'windows dragging'.

Characterization:

- At most one window of each type can be opened at the same time.
- That is, it is possible to open several windows at the same time, but it is not possible for several windows of the same type to appear.
- If the window is already open, make sure it moves to the front of the screen. [Only for those who have implemented "windows drag"]

Instruction:

one window:
1. When you open a new window, save the icon that opened it as a reference to the window.
2. When we close the window, that is, delete the object, the reference will become null.
3. Before opening a new window, check whether the value is null, that is, whether the window is open or not.

The front of the screen [only for those who used "windows drag"]:  
1. You can save a reference to the window component instead of GameObject (see tutorial below).
2. In the script of this component, create a public method responsible for moving the window to the front of the screen.
3. When the window already exists, this public method is called.

Reference to the component [only for those who implemented "windows drag"]: 
If we want to call the method of a certain script, we have two different ways and you can choose the one that suits you For the way you implemented the windows:

1. If the script we wrote is attached to the main object of the window
   1.1. Change the type of public field that we created in the "Opening windows: Creating an object" section to the type the window's script (make sure the icons still point to the windows' prefabs as required).
   1.2. When you call Instantiate, you get a value of the script type instead of GameObject.

2. If the script we wrote is attached to one of the children of the window
   After creating a new window using Instantiate, a component of a certain type is requested which is attached to it using <>GetComponentInChildren - [https://docs.unity3d.com/ScriptReference/Component.GetComponentInChildren.html](https://docs.unity3d.com/ScriptReference/Component.GetComponentInChildren.html)  
   (Pay attention to the use of the feature Generics of C# to which we will pass the type of script we want to receive) - [https://learn.microsoft.com/en-us/dotnet/csharp/fundamentals/types/generics](https://learn.microsoft.com/en-us/dotnet/csharp/fundamentals/types/generics)
Watch (1 point)

Characterization

- One of the windows must display a clock with hands.
- The clock displays the real time (as defined in the system) at any given moment.
- The hand must rotate consistently, i.e. not jump between hours 5 to 6, but rotate between them throughout the hour (hours with an accuracy level of one minute, minutes with an accuracy level of a second).

Instruction

1. Define the Pivot Point of a pointer sprite so that it is placed at the edge (in the Sprite Editor window).
3. To rotate by angles use `Transform.eulerAngles` (and no rotation) - https://docs.unity3d.com/ScriptReference/Transform-eulerAngles.html
4. *tip:* Z rotation in unity is counter clockwise.
**compass (1 point)**

**Characterization**

- One of the windows must display a 'compass'.
- The compass needle always points in the direction of the mouse cursor.

**Instruction**

1. Use the appendix [Where is the mouse cursor located?](https://docs.unity3d.com/ScriptReference/UnityEngine.input.CommandBuffer.html) to check the position of the cursor in the world.
2. Calculate the direction from the center of the needle to the location of the marker (mousePos-needleCenter).
3. Use the method [Atan2](https://docs.unity3d.com/ScriptReference/Mathf.Atan2.html) and regularly [Rad2Deg](https://docs.unity3d.com/ScriptReference/Mathf.Rad2Deg.html) to calculate the angle in degrees.
4. Update the eulerAngles value of the needle accordingly.
Recipes (attaché)

**Manual countdown**

1. Define a private field of type `float` and place in it the number of seconds remaining.
2. In all `Time.deltaTime` until the remaining value is less than 0. - https://docs.unity3d.com/ScriptReference/Time-deltaTime.html

**Random duration in range**

1. Define use of Unity's random library by adding the following line to the top of the script file:
   ```cpp
   ; using Random = UnityEngine.Random;
   ```
2. Request a value in the desired range using `Range`.

**Access to the render component**

To change the display of sprites during the game, access the Sprite Renderer component by creating a public field inside it. Place the component in the SpriteRenderer window Inspector.

**Change sprite in code**

1. Create a public `Sprite` field and place inside it the asset we want to use later.
2. Change the value of the sprite property of the Renderer Sprite component to the desired asset. - https://docs.unity3d.com/ScriptReference/SpriteRenderer-sprite.html

**Color change in code**

1. Define a color using `new Color(a, b, g, r)` when the members are a float value in the range [0.0, 1.0] - https://docs.unity3d.com/ScriptReference/Color-ctor.html
2. Alternatively, you can choose one of the defined static colors. - https://docs.unity3d.com/ScriptReference/Color.html
4. ‘Reset’ coloring a sprite back to the original using white color. - https://docs.unity3d.com/ScriptReference/Color-white.html

**Is the cursor over me?**

1. Add a 2D Collider-type component.
2. Create a private field of type `bool`.
3. Use the following events to update the field: `OnMouseExit`, `OnMouseEnter`. -
   - https://docs.unity3d.com/ScriptReference/MonoBehaviour_OnMouseExit.html
   - https://docs.unity3d.com/ScriptReference/MonoBehaviour_OnMouseEnter.html
Drag: continuous interaction (attache)

Unlike pressing which is an immediate interaction, a dragging action lasts over time, that is, over several frames.

The interaction can be broken down into three stages:

1. Clicking - selecting the dragged object.
2. Mouse movement - changing the location of the object (many times).
3. Releasing the click - determining the final destination.

**Recipe**

1. Similar to clicking, one of the 2D Collider components is needed.

   1.2. Use the appendix to find the position of the mouse in the space world.
   2.2. Create a Vector3 field and store in it the difference between the mouse position and the object position (mousePos - objPos)

   1.3. Find the current mouse position.
   2.3. Place the object at the mouse position minus the difference calculated at the beginning of the migration (mousePos - originDelta)

   1.4. Optionally, it can be used to ‘clean’ the action or ‘fix’ its consequences.

**Why do we use the difference?**

We place objects in the world in relation to their starting point (pivot), the chance that we will ‘capture’ the object exactly at this point is very low. If we place the object exactly according to the time position, a jump will be created when capturing the object since the cursor is not at the starting point according to which the object is located. Therefore, we will want to ‘compensate’ for this to maintain an accurate relationship between the click position on the object relative to its starting point even after it is moved.
Where is the mouse cursor located?

In games, there is a separation between the screen space (screen space) and the world space (world space). The objects in the game are located in the world, but the mouse cursor is in the screen space (distance in pixels from the lower-left corner). In 3D games, there is a challenge in converting from the screen space (two dimensions) to the world space (three dimensions) since it is not necessarily possible to determine the depth at which the cursor is in relation to the world. There are several approaches and solutions for this, but in two-dimensional games (like in this project) we can make it easier by manually determining the Z position.

Main camera

Most of the solutions for checking the position of the cursor in the world involve direct or indirect use of the camera object. The Unity scene contains a camera object which is marked with a dedicated MainCamera tag. We can access the camera marked with this tag from any script by call to Camera.main (instead of creating references to the camera in the components we write). - https://docs.unity3d.com/2021.3/Documentation/ScriptReference/Camera-main.html

recipe

2. Convert the location from the screen space to the world Camera.main.ScreenToWorldPoint. - https://docs.unity3d.com/2021.3/Documentation/ScriptReference/Camera.ScreenToWorldPoint.html
3. Change a Z member of the received position to 0

Why do we change the resulting Z value? [enrichment]

When we pass a location to ScreenToWorldPoint the method uses the Z value to determine the distance of the plane parallel to the camera where the conversion from pixels to world units will take place. The Z term obtained from mousePosition is 0, which means the plane in which the conversion will be performed will be at the position of the camera itself. So if the camera is located in the world at the Z position equal to 10 - so is the converted position of the cursor will be at the Z position equal to 10.

Since the camera "sees" only what is in front of it (more precisely, only what is inside the frustum, i.e. inside a defined geometric shape), Everything that is in the position of the camera itself or behind it (to be exact, outside Frustum) will not be displayed. We are of course We want the object to be displayed, so we place it in such a way that the camera will 'see' it by 'pushing' the Z position which is calculated for the value that is sure to be in front of the camera lens