This is a "cheat sheet" for "Desktop Screenshots" package by YellowAfterlife.
The package can be acquired from itch.io or Unity Asset Store.

Important notes

Here are some things that you should know about desktops on Windows:

- Coordinate system is left-handed, +x is right and +y is down.
- Coordinates are strictly integers.
- Coordinates measure from primary display's top-left corner, which is at 0,0.
- As result, displays to the left/top of primary display can have negative coordinates.

Setting up

Add `DesktopScreenshot.cs` to your project.
That's all. You can now use the DesktopScreenshot class.

Optionally, also import `DesktopScreenshots.html` / `DesktopScreenshots.pdf` (offline version of this document) and/or `DesktopScreenshots` folder (demos).

Capturing

Statics:

Capture(RectInt rect) → Texture2D

Captures a region of a screen and returns a new BGRA32 texture for it.

This function is intended for "one-off" uses (e.g. capturing the desktop and using it as a background texture) - if you want to capture repeatedly, consider instantiating DesktopScreenshot.

```
var tex = DesktopScreenshot.Capture(new RectInt(200, 100, 400, 300));
GetComponent<RawImage>().texture = tex;
```

would capture a 400x300 region starting at x=200 and y=100 in primary display's top-left corner and assign it as a texture to caller's RawTexture.

Constructors:

DesktopScreenshot(int width, int height)

Creates a screenshoter for capturing regions of specified size.

This will create a BGRA32 texture to it.

DesktopScreenshot(Texture2D texture)

Creates a screenshoter for an capturing regions into the texture.

Note that the texture should have format set to `TextureFormat.BGRA32`.

Destructor:

Destroy()

Don't forget to call the destructor when you're done using a screenshoter to free up the associated native bitmap.

This will not do anything to your texture and can be called more than once without harm.

Fields:

texture : Texture2D

The screenshoter's target texture.

You can resize or swap it out as you see fit, but you should be aware that there is minimal overhead overhead related to re-creating a native bitmap prior to capturing with changed dimensions.

width : int

A shortcut for `texture.width`

height: int

A shortcut for `texture.height`

Methods:

Resize(int width, int height)

A shortcut for `texture.Resize`.

Clear()

Clears pixels in the texture to RGBA(0,0,0,0).

This can be slightly faster than doing `texture.SetPixels`.

Capture(int x, int y)

Captures a section of the screen into the current texture.

x,y point at the section's top-left corner, relative to top-left corner of the primary display.

Measuring

GetDisplayBounds() → RectInt

Returns a RectInt covering bounds of the primary display.

x/y of this rect are always 0.

The following would take a screenshot of the primary display and save it to `screenshot.png`:

```
var tex = DesktopScreenshot.Capture(DesktopScreenshot.GetDisplayBounds());
System.IO.File.WriteAllBytes("screenshot.png", tex.EncodeToPNG());
```

GetDesktopBounds() → RectInt

Returns a RectInt covering bounds of desktop (encompassing all displays).

x/y count from top-left and can be negative
(if a secondary display is to the left/top of the primary display).

The following would take a screenshot of desktop and save it to `screenshot.png`:

```
var tex = DesktopScreenshot.Capture(DesktopScreenshot.GetDesktopBounds());
System.IO.File.WriteAllBytes("screenshot.png", tex.EncodeToPNG());
```

GetDisplayInfos() → DisplayInfo[]

Returns an array of classes with per-display information.

Each `DisplayInfo` has the following fields:

screenArea : RectInt

Area occupied by display itself
(e.g. x=-1920,y=0,w=1920,h=1080)

workArea : RectInt

Area occupied by display, excluding taskbars
(e.g. x=-1920,y=0,w=1920,h=1040)

deviceName : string

Display name according to WinAPI.

This is usually a technical name like `\\.\DISPLAY1` and may not be useful for actual identification.

isPrimary : bool

Whether the display is the primary display.

The following would log information about all available displays:

```
foreach (var inf in DesktopScreenshot.GetDisplayInfos()) {
    Debug.Log(inf);
}
```

Technical

Implementation

The package is implemented entirely via P/Invoke, which means a few things:

- It does not have any additional dependencies.
- You can edit/extend the implementation (e.g. capture parts of windows instead of desktop) without any additional setup.

Performance

This package uses GDI's BitBlt, which is by far the most reliable way capture pixels of the screen, but is CPU-bound - for example, capturing a 1920x1080 region takes from 10ms on higher-end systems to 30ms on lower-end systems.

In other words, it is slightly less suitable for fullscreen video capture, though some capture software (e.g. OBS) does offer this method as a compatibility option.

Unity compatibility

The package is compatible with a multitude of Unity versions - it only needs `Texture2D` class to exist and to have a `LoadRawTextureData` method, which had been around since at least 4.x.

Platform support

Being something that uses WinAPI and GDI, this package is only compatible with Windows.

While it may be possible to have it work on Mac/Linux, that requires a complete rewrite per-platform.