Game Development BCN - Final Examination - January 18th 2017

YOUR FULL NAME: **Solution Provided by the teacher**

- You have 2 hours to complete the assignment.
- Only valid text will be the one inside each box, everything else will be ignored by the teacher

1. **(2 points)** Describe the visual artifact called "**tearing**". In which circumstances it will happen ? Name (and explain the use) the technologies in place that we can use to avoid this artifact.

Tearing is a visual artifact that happens when the refresh rate of the graphics code is not in sync with the screen refresh.

In those cases we can end up rendering the top of the screen in one frame and the bottom part with the following frame.

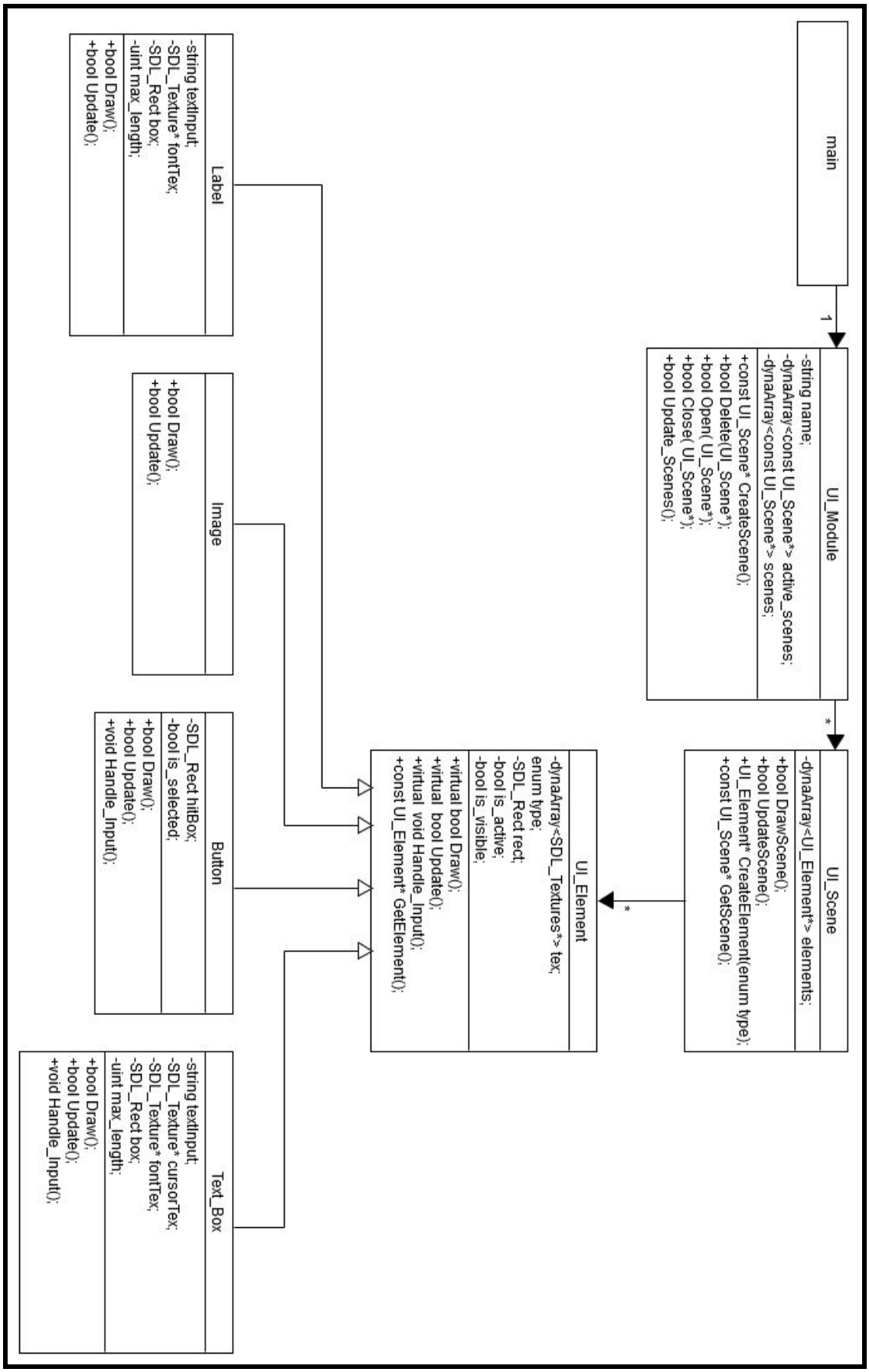
In order to avoid it, we should be render using the **double buffer** pattern, so we keep two copies of the frame buffer and keep swapping them.

Using **vertical synchronization** (or vsync) we can wait for the swap to happen only after the next screen refresh, effectively removing the problem completely.

2. **(2 points)** *Come up* with an XML structure that would define the game entities seen in this screenshot. The XML should define **both** their properties and their current situation. Avoid property repetition as much as possible: all information should be unique in the file.


```
<?xml version="1.0" encoding="utf-8"?>
<entities>
  <static>
    <bushes>
      <instance coords="50,25"/>
      <instance coords="51,25"/>
      <instance coords="52,25"/>
      ...
    </bushes>
  </static>
  <dynamic>
    <player coords=”47,27” facing=”east” total_hp=”4” hp=”1”
green_gems=”216” arrows=”10” bombs=”0”/>
    <chickens hp="1">
      <instance coords="80,50" facing=”west” flying=”true”/>
      <instance coords="80,50" facing=”west” flying=”true”/>
      <instance coords="82,50" facing=”east” flying=”false”
color=”yellow”/>
      ...
    </chickens>
  </dynamic>
</entities>
```

3. **(3 points)** Check the provided UML for a graphical user interface. Make a list what you would in improve in three categories: critical, important and optional. Do **not** create a new UML, focus on improving on this one.



Critical improvements:

- If we receive a const UIScene* when creating it, we cannot execute any other method (like delete) since those require a non-const pointer

Important improvements:

- Keeping two arrays of scenes is bad for performance, since every time we have to deactivate one scene we must remove it from the array and add it another one. Using arrays for this is very slow and potentially problematic on memory management.
- Every scene keeps an array of elements instead of a list. This assumes that the amount of elements is very static. If that would not be the case, we would need to consider using a linked list instead.
- UI Scene has a method to return itself, no apparent use. Same goes to UIElement::GetElement()
- Every UIElement keeps a list of textures. It should, instead, keep a list of rectangles to cut art from the atlas that should be defined, at least, on the scene level.
- UIElement keeps booleans for states like is_active. We should be implementing a full event system instead, to handle mouse/keyboard and any other events like focus from TAB.
- There is no TAB management in place.
- For the label we are not using bitmap fonts, but truetype fonts. For that case the font should be a pointer a font, not a texture.
- No point on having max_length in a label.
- In TextBox we can safely assume that the cursor texture will be shared across all textboxes.
- TextBox, in general, should rely on label to draw text instead of replicating all its data and functionality.

Optional improvements:

- The UI Scene concept probably too much abstraction and a case of over engineering.
- No point on having element::Update and element::Draw methods return a bool.

4. **(3 points)** Adapt the A* algorithm for it to accept two floors. Those two floors are connected by a elevator in a well known position of the map. Explain your reasoning behind your decisions.

To enable this functionality, we would need to generate clusters of graphs and connect them:

- Each node now would feature an identifier of the current cluster ID (level/floor).
- If we are trying to reach another position in the same cluster, we pathfind as normal (elevator would be non-walkable).
- Instead, if the destination position is in another cluster, we would need to pathfind to the exit and then normally from that cluster level to the destination.

The A* per se would not be changed, only the structure around it, with tiles with their cluster ID and known exit and entry points.

We could precalculate all paths to each elevator on each floor using Dijkstra.

Use this page for your own notes. You cannot use any other page. Do not remove this page from the set.