Game Development TRS - Final Examination - January 16th 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 all the specific benefits of using the PHYSFS library. Elaborate on the difference between having those benefits and lacking them.

```
PhysFS provides:

   -  Virtual file system: allow to abstract all the complexities of a real
      file system in a virtual one that simplifies finding and accessing to
      files. This is transparent to the programmer and allow access to virtual
      directories and compressed files transparently. It also simplifies
      modding and patching process.

   -  Not having a virtual file system would force the programmer to be
      subject to changes in the real file system more frequently and having to
      adapt his code every time a folder was renamed or moved. On top of that,
      the programmer would need to code his access specifically for compressed
      files.

   -  Sandbox protection for writing files: When writing with PHYSFS we add a
      layer of security since we sandbox all writing operations to a specific
      folder. Programmer does not need to know where this location is in the
      real file system.

   -  Without sandbox protection we could be subject to hacking is we allow
      modders / script writers to write files, creating a security issue when
      releasing the game.

   -  Multi Platform file operations: PHYSFS allow to access files the same
      way in a Linux/Windows or MacOS platform. On top of that provides
      non-standard functionality like listing directory contents.

   -  Without multiplatform access we would need to create our own library to
      abstract file access.

   -  (Bonus) PHYSFS allow buffering, which enabled us to load big files using
      only a portion of RAM. This is critical to streaming systems. This would
      have to be coded otherwise.
```

2. **(3 points)** Explain the concept behind the BFS search algorithm and write down its pseudocode (or use python/C).

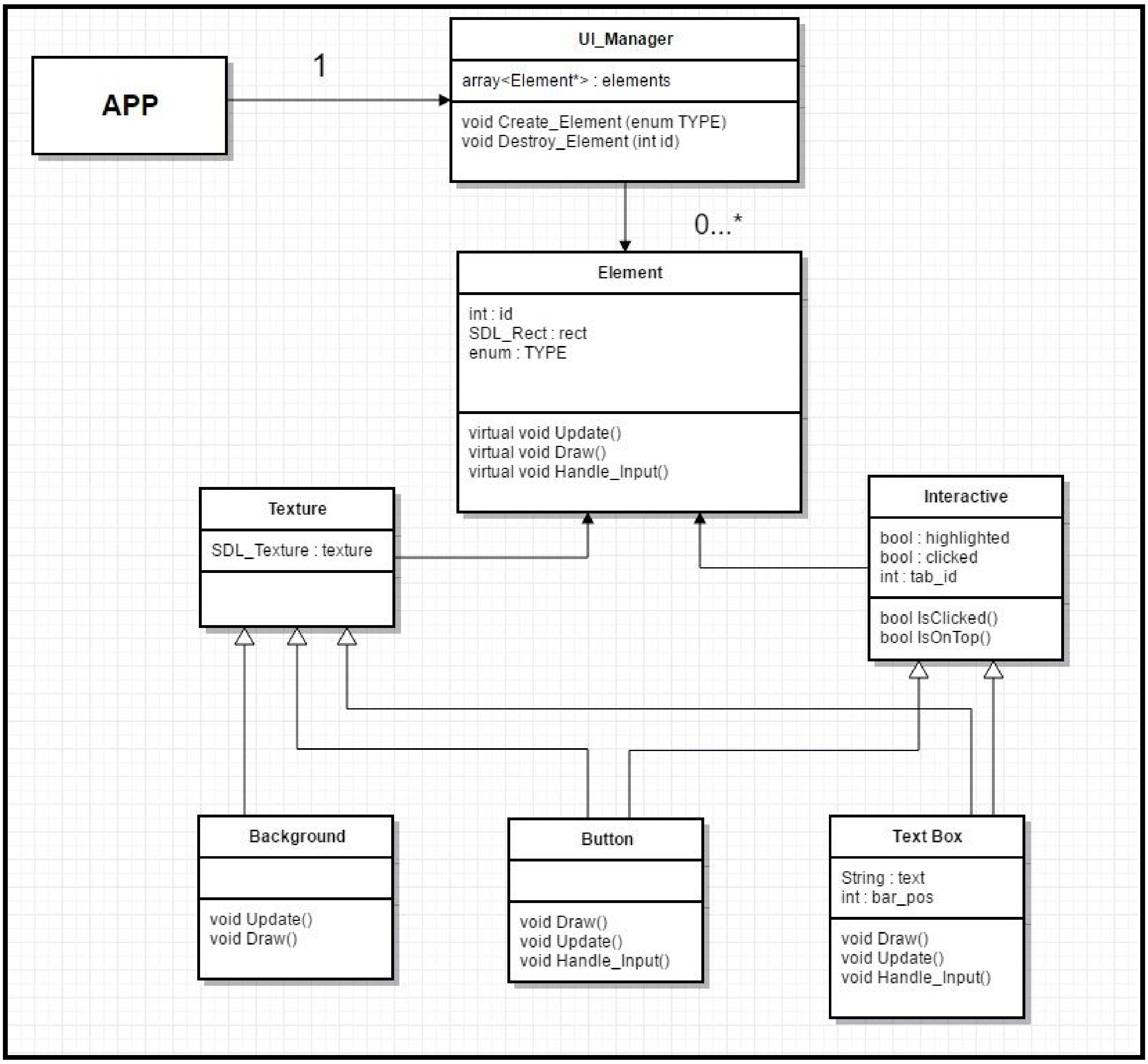
BFS stands for Breadth First Search and it is a method to visit all the nodes in a tree: it evaluates all children of a given node before proceeding to the next depth level.

-- using python:

```
frontier = Queue()
frontier.put(start )
visited = {}
visited[start] = True

while not frontier.empty():
   current = frontier.get()
   for next in graph.neighbors(current):
     if next not in visited:
          frontier.put(next)
          visited[next] = True
```

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.



List of **critical** things to improve

- Create_Element(int TYPE) does not return any handle for client code to refer back to the UI element. To be coherent with Destroy_Element, it should probably return and ID.
- No way to express parent->children relationship.
- The interactive class seems to ignore the keyboard.
- We normally aim to have one big atlas texture to have all UI elements. This UML keeps track of each individual texture element. It should, instead, just keep rectangles.
- There is no way to define callbacks to be executed in case of events on the UI like buttons clicked or textbox changed.

List of **important** things to improve

- Creating a full class “Texture” only to have a pointer to a texture is overengineering and making things slow because of inherency, we could totally remove it and have pointers to textures where needed.
- Background element has no apparent use for “Update()” method. Maybe a frame animation but there are not properties for storing that animation.
- Having the Interactive class storing things like “isClicked” is in general a bad idea since it calls for pulling this data. The GUI should prefer to use events instead.
- Button class lacks any properties. It should contain the rectangle/art for when the button is hovering/clicked/etc.

List of **optional** things to improve

- Using an array to store the elements could be a bad choice if we end up adding and deleting them often. In this case we would change to a list.
- The variable “rect” in element could be more self-explanatory, is it the input area ? the graphical area ?
- It uses multiple inherency for the button, a concept that it is a better to avoid.
- Arrow caps from Texture and Interactive should be white since it expresses inherency.
- It is not clear the use of the “Update()” virtual method, since we already have “Draw()” and “HandleInput()”
- It is not clear the use of “IsOnTop()” method for the interactive class.
- Mouse Hovering is “isHighlighted” ? Should prefer standard naming.

4. **(2 points)** Explain the key concepts behind a Quake-style console. Explain the main steps that need to be taken in order to implement it (assuming we have a functional GUI system in place).

A quake-style console is a tool were we can input commands into our game similar to a traditional command prompt of an operative system. It provides a tool for fast iteration on gameplay elements without closing the application. We can quickly check the current values of different gameplay elements, tweak them and save them for future uses.

The main elements are the commands and the cvars (console variables). The first are for executing specific functionality, like loading a new level, saving to a file or quitting the game. The cvars store values that are relevant to the game, like player HP, damage performed with different attacks and so on.

The console also outputs many debug information that can be scrolled up for fast tweaking/debugging of the game.

In order to implement it, the strategy would be (there are many valid ways, this is what was proposed in class):

- Creating a new module to handle all commands and cvars
- Using the GUI system we create a multiline label and an input box
- The GUI should appear/disappear on pressing a key (normally tilde, left of 1)
- We redirect all LOG entries to the multiline label
- We make sure there is a way to scroll up/down on the multiline label (normally with mouse scroll using the third mouse button)
- We create a small class to represent the concept of command and cvar
- We create method to add new commands and cvars with callbacks for each module to handle their specific implementation
- Optionally we can save the historic of commands and have autocomplete feature in the input box

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