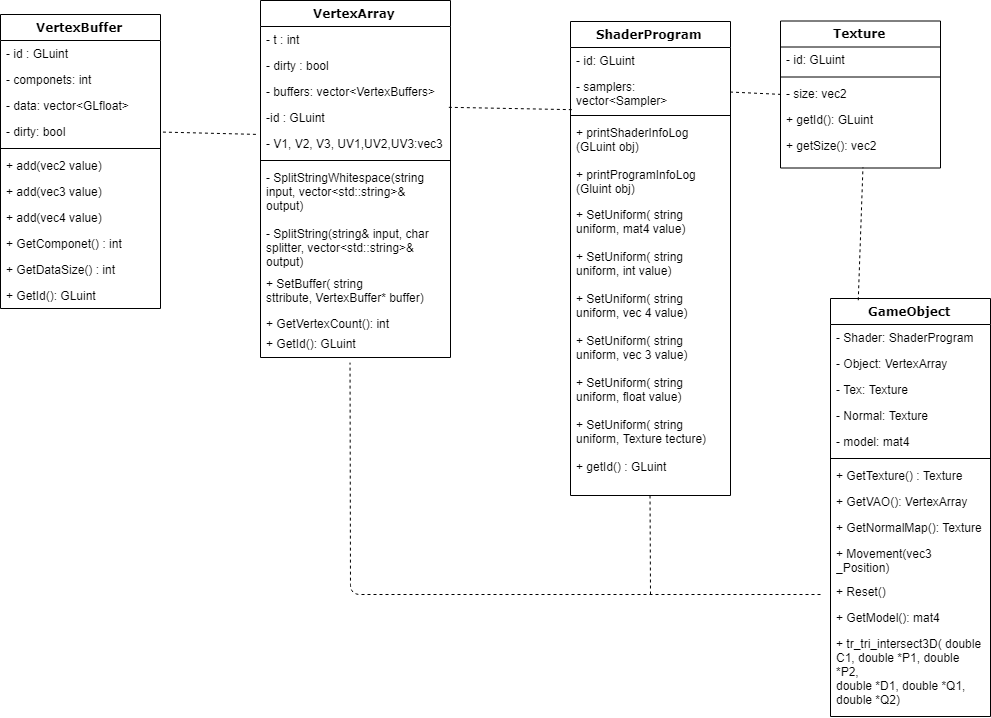
In this project I am attempting to make a simple 3D game that uses different types of shaders and use multiple textures and to load and render multiple objects. The game it-self will be simple where the player must move left to right dodging objects along the way.

One of the most complex parts to this project is normal mapping. Normal mapping is incredibly useful in modern video games, it allows to compute models that look more detailed without using more polygons. The first part to normal mapping is understanding how it does it. Models have normal and usually each normal point outwards straight, normal mapping will change these by small amounts that will then cause the model to have additional details. These details can vary in many ways from scars to winkles. The ways normal mapping works is simple until you try to apply it to a model, a model is usually in a different space as the normal map and this poses a problem, changing the normal while in world space would not wield the results needed for the normal would not change correctly. So, we needed to create a TBN matrix. TBN stands for Tangent Bitangent and normal, this is calculated from the model itself. To start we need the normal, this is easy when you load the model for that will be a part of the normal, the tangent and bitangent is not so easy since we have the normal we can calculate these once we form a triangle from different vertices using this triangle we can calculate the tangent and bitangent for the matrix. Once the matrix is calculated we can apply it to the model to move it into the tangent space that the normal map texture is in. This is not the only way to calculate this, we can move the texture into the models space so we can apply the normal changes to it but this turns out to be more computational heavy due to the fact that most calculation would be done in the shaders, particularly the fragment shader, so to make sure the program runs as smoothly as possible I will be calculating the TBN matrix on load up.



Keeping the setup simple is very useful, having all graphical calculation and call in the back allows for easier use of the program in the future and can make them safer to use. Using private and protected variable and only allowing use of get functions such as getVAO() means informtion can not be altered in a unintened way that could become an exploit in the future. The strength of this setup is that reusing it for future projects should be useful but a weakness to it is how I have developed the GameObject class, currently it is just a single class whilst this is not bad I should, to improve it, is create subclass’s or children classes so I can have more vartion to the classes and gives me greater control over them for different instances. The VertexBuffer class does not require anything from other classes and can process its information on it own without the other classes, whilst VertexArray is associated with VertexBuffer and uses it to create the its VertexArray’s. ShaderProgram requires VertexArray and Texture to create its shaders and so requires a link to them but it wont be a child of textures or VertexArray since it is not required to inherit from either just utilise information from them. Texture does not require any assoction to function but will give this information out to allow the program to run. GameObject does not require other classes but instead is storage for the access to the other classes allowing for easier use of the program and a smaller main function.

Overall the project is not yet finished and there are many places to improve apon, to start with, creating a better way to calculate the TBN matrix will be very useful to learn and potentional implement, the main problem I have found is that if I do not have the vertices as a multiple of 3 it seems to go out of scope and if it has multiple textures attached to it I will have to make sure it calculates it for each section of the model. Next is to fully conceal the loading of each texture, shaders etc. with the GameObject class so I can write minimal amount of code in main and makes it easier to use in the future and will also mean if someone else uses the program it can be easily used. Next would be to add more to the GameObject class by creating children for it, this will be great for making bigger more complex games. And lastly would be to create a component system to go along with the gameobject class allowing for certain objects to have certain objects to have particular traits instead of having to create a new child every time I want to add something to the game.  
The biggest advantage to this program is that I was able to implement a simple normal mapping system that works very well, also the GameObject class has a good setup for future improvement and should be very useful in the future. Overall the project did not reach the extent that it should of and whilst it has potentional, a lot of work is needed to make it better and it has given me experience in the use of opengl and the graphics cards system to allow for advanced graphic implmentation.

Opengl-tutorial.org. (2019). *Tutorial 13 : Normal Mapping*. [online] Available at: http://www.opengl-tutorial.org/intermediate-tutorials/tutorial-13-normal-mapping/ [Accessed 13 Jan. 2019]