*A few ideas for improvement of this project:*

General:

1. Figure out the aspect ratios of the built game, and work on the game interface (screen setup: where menus are, where modeled objects are, etc.). Maybe have drop-down menus for types of structures available for reconstruction (if the program needs to be a built game), or have a separate window in Tool bar or something (is the final product will be available in Unity Editor format) instead of buttons crowding the screen.
2. Help buttons to help navigate in the game.
3. Let the modeler choose the dimensions of the “ground”.
4. Give the modeler an option to choose which brick texture to use (maybe a dropdown menu listing/showing all the available textures in the library).
5. Give an option to choose bricks with default dimensions (the ones historically used in the Roman Empire, see the word document in Materials Gathered folder).
6. A clear majority of the scripts that I wrote focus around the global origin (0,0,0). These scripts were used to create simple structures, but as the complexity of the modeler’s desired design increases, we would need to give the modeler control over where some structures are built (for example, if they want to build an arch on the second floor at some z-offset aka away from the global origin of the building instead of on the “ground”). Also, scripts dealing with similar kind of structures do not allow the construction of multiple structures of the same kind (Corbelled script searches active game objects in the scene and deletes previous corbelled arch if a new one is to be built, and the same thing with circular arches). So, some manipulation needs to be done in the final version to allow multiple similar structures to be present in the model at the same time.

Populate a wall:

1. Come up with rulesets, e.g. x rows of A brick and y rows of B brick alternate. Also, do other types of walls listed in Roman Structures in Materials Gathered folder.
2. Let the modeler choose the max values on the parameter sliders (not really that important).

Arch:

1. Create a unit groin vault in Rhino and import it into Unity as an asset, and let modelers change its scale. Potential issue might be that the parameters of the groin vault (number of bricks in each row, and the type of vault, i.e. zippered or not) that the modeler wants may not match that of the unit groin vault.
2. An alternative to that might be to find an asset that does Boolean difference (scripts that perform the operation can potentially be written by a CS student or someone who is very good with meshes and knows how Unity renders its meshes. Look into Mesh Renderers in Unity.) For starters, check out the following link: <https://github.com/sabresaurus/SabreCSG> . It has an algorithm that does Boolean difference, might deconstruct the code and implement it, or use it to guide the programmer to come up with a code of his own. This is better than using an asset, because it gives you the control over the flow of operations and also because some assets need to be purchased for each machine that Unity is used on (as is the case with the current CSG 1.2 asset that I use in the project), which defeats the purpose of using Unity (free) instead of other programs (non-free) to make it accessible to the end-user free of any charge.
3. If Becca decides to make the final program accessible in Unity rather than a built game, errors on invalid inputs for parameters (like negative numbers for lengths and number of things, or angles that are <0 and >180 in circular arches) should be thrown. Exceptions (throw new Exception) and errors (Debug.Log) do not behave the same way when the game is built (nothing happens, maybe internally a script just halts) than they do in Editor (throw the error and pause the game). So if Becca decides to deliver the end product as a built game, a good idea might be to have a dialog box on the side where the errors and exceptions are displayed.

* Solmaz Jumakuliyeva, 8/18/17.