Project Proposal

Group: Alex Burnley, Jason Cummings, Riley Kirkham

1. What is your project?

   Our project will be building a 3D platforming game using C++, OpenGL, and Bullet. This game will have a 3D animated character model that will be controlled using the mouse and keyboard. We will create a game level that will be composed of static platforms and obstacles for the player to jump around on. There will also be a couple of moving platforms that will be activated by the player interacting with buttons. The level will have a "goal" platform that will end the game when the player reaches it.

   Our game will also have a main menu which will be brought up when the game is first opened. This menu will have buttons to enter the game or exit the program.

2. Technologies:

   Github - completely familiar

   All three of us have used Github extensively in the past for personal, school, and corporate projects. We'll be using some new mechanics that none of us have experimented with before, such as using submodules to include the libraries necessary for the project. By doing this, it should be much easier to make the project portable. It will also mean that it should be possible to simply clone the repo and build without having to install any additional libraries.

   C++ - completely familiar

   All three of us have coded extensively in C++ and done multiple large scale projects.

   SDL2 - completely familiar

   Alex and Jason have used the SDL2 framework for multiple projects before to great success. SDL2 will not be used that extensively in the project. We will use it to create the window and create basic handlers for keyboard and mouse inputs. Beyond that, SDL2 just creates the OpenGL context and all rendering functionality is passed on from there.

   OpenGL - completely familiar

   Alex and Jason both took Graphics and Advanced Graphics classes and have worked extensively with OpenGL. Some time will be taken to teach Riley some of the OpenGL basics, such as the rendering pipeline and the buffer objects which constitute the rendering system. We will be using OpenGL 3.3 for this project because we've found in past projects that it suits the needs of a game best both in terms of speed and flexibility.

   GLSL - completely familiar

   GLSL is the shader language for OpenGL. Shaders handle a lot of the special effects in graphics programming. Alex and Jason have used GLSL before in both class and projects. As shaders are very short programs designed to affect single

vertices or pixels, it will be simple to show Riley GLSL basics and allow him to help coding the shaders.

Blender/Maya/Cinema4D - completely familiar

Alex and Riley have experience doing 3D modeling and animation in both Blender, Maya, and Cinema4D while Jason has played around with Blender some. We will use these programs to create our models and scenes which will then be saved as FBX files and loaded into the game for OpenGL to display.

Autodesk FBX SDK - not familiar

The FBX SDK is software provided by Autodesk for reading FBX files. FBX files are a complex binary format and as such are very difficult to read. Using the FBX SDK will allow us to read these files and get them as a data structure that is much easier to read. The FBX SDK has a lot of documentation and as such should not be very difficult to learn and use.

stb image loading library - completely familiar

Alex and Jason have used this library in previous projects for loading textures into games and as such will have no problem incorporating it into this project as well.

Bullet - not familiar

Bullet is an open source and widely used physics engine that we plan to use to incorporate physics elements into our game. This engine provides the most uncertainty in our project as none of us have worked with Bullet before. Looking at the documentation and examples provided, it seems that much of the work with Bullet simply involves setting up interactable objects and providing physical constraints and impulses. In short, as long as we properly setup the world and create objects with the correct properties, Bullet should take care of the rest. The most difficult part may be figuring out how to incorporate Bullet into the project in the first place and compile it. Cmake may be required to do this.

Cmake - not familiar

None of us have used Cmake before, but after researching, it seems to be the best way to build our project. A lot of frontloaded work will be required to make sure that the libraries are all included properly and our project builds, but once we do that, we should have an easy-to-build cross-platform project.

3. Essential parts of project

-A map to play on/ reach the objective on

-Player Models

The essential parts of the project are to have a 3D world that a player can navigate to reach and end objective. This means that the basic rendering pipeline and integration with the physics engine are both critical to a successful project. Special effects such as bloom and advanced game mechanics using physics would create a better project, but they are not completely necessary for the project to be successful. A high priority for us will be an animated character model, but this is a visual effect, and does not affect the overall game mechanics. The game will also need floors that the player is able to run

and jump on, as well as a ending platform that, once the player reaches, will complete the game. If there is time, we could also implement holes in the floor that, when the player falls through them, forces them to respawn at an earlier position or checkpoint.

4. Outside resources:
   Aside from the external libraries and technologies listed above, we will use the following external resources:

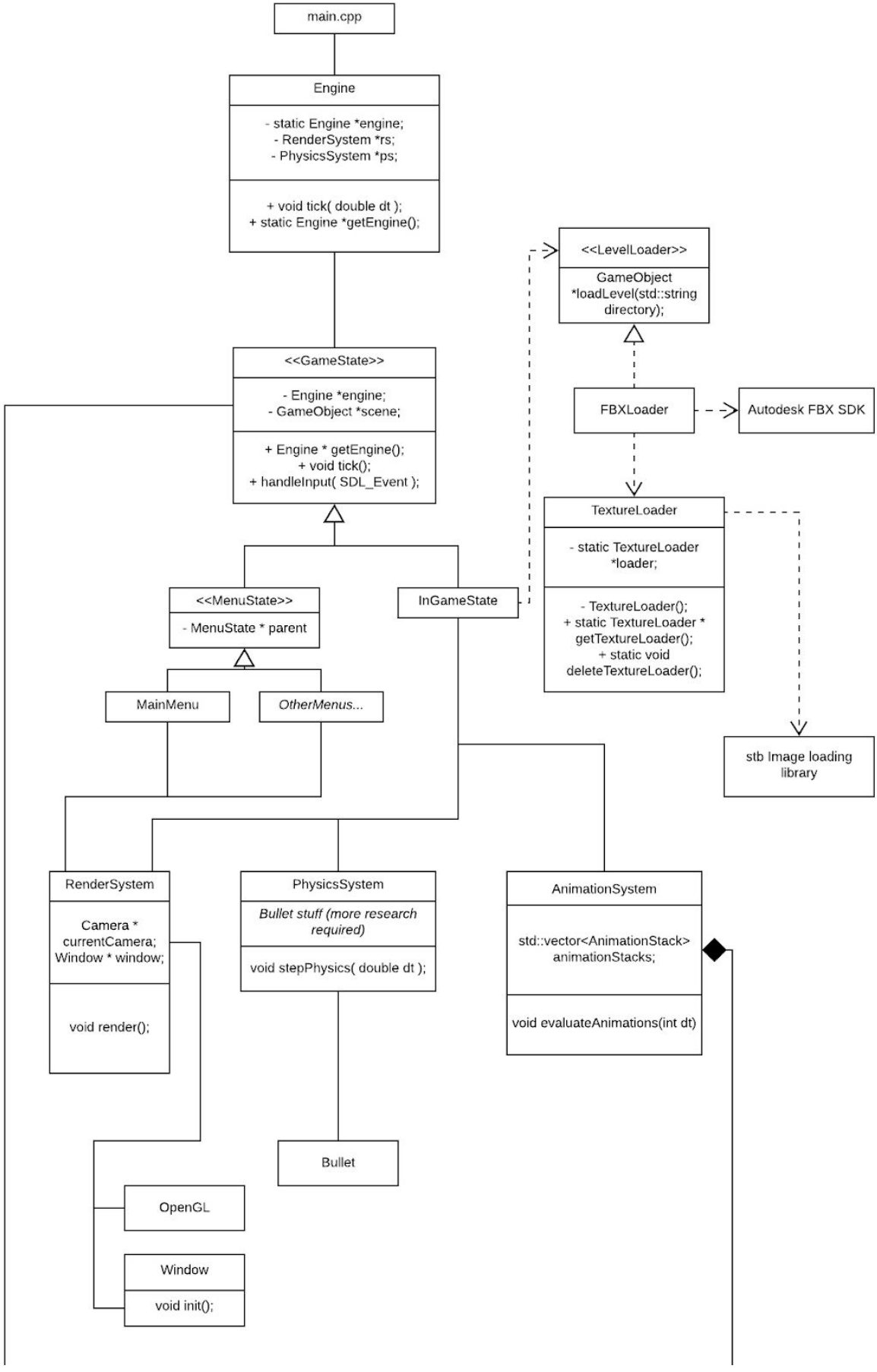
   Textures

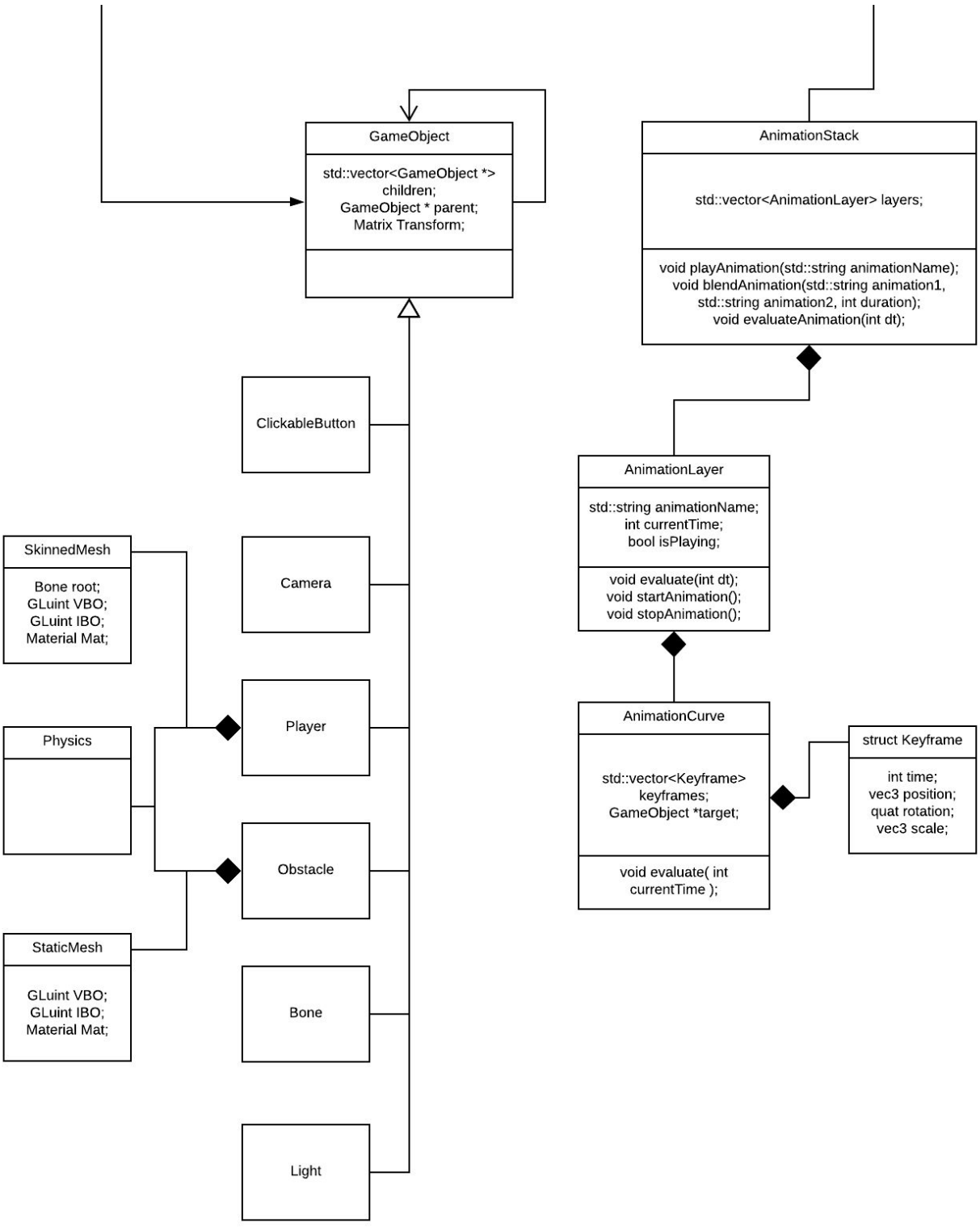
   Rather than create our own textures for in game objects and models, we will draw upon external texture libraries. There are many free texture libraries that offer high resolution and tileable textures, so we will use these rather than trying to spend hours coming up with textures that look good and are useable. One of these options is textures.pixel-furnace.com but, as stated before, there are many other choices if that one does not suit our needs.

   Reference Images for creating 3D models

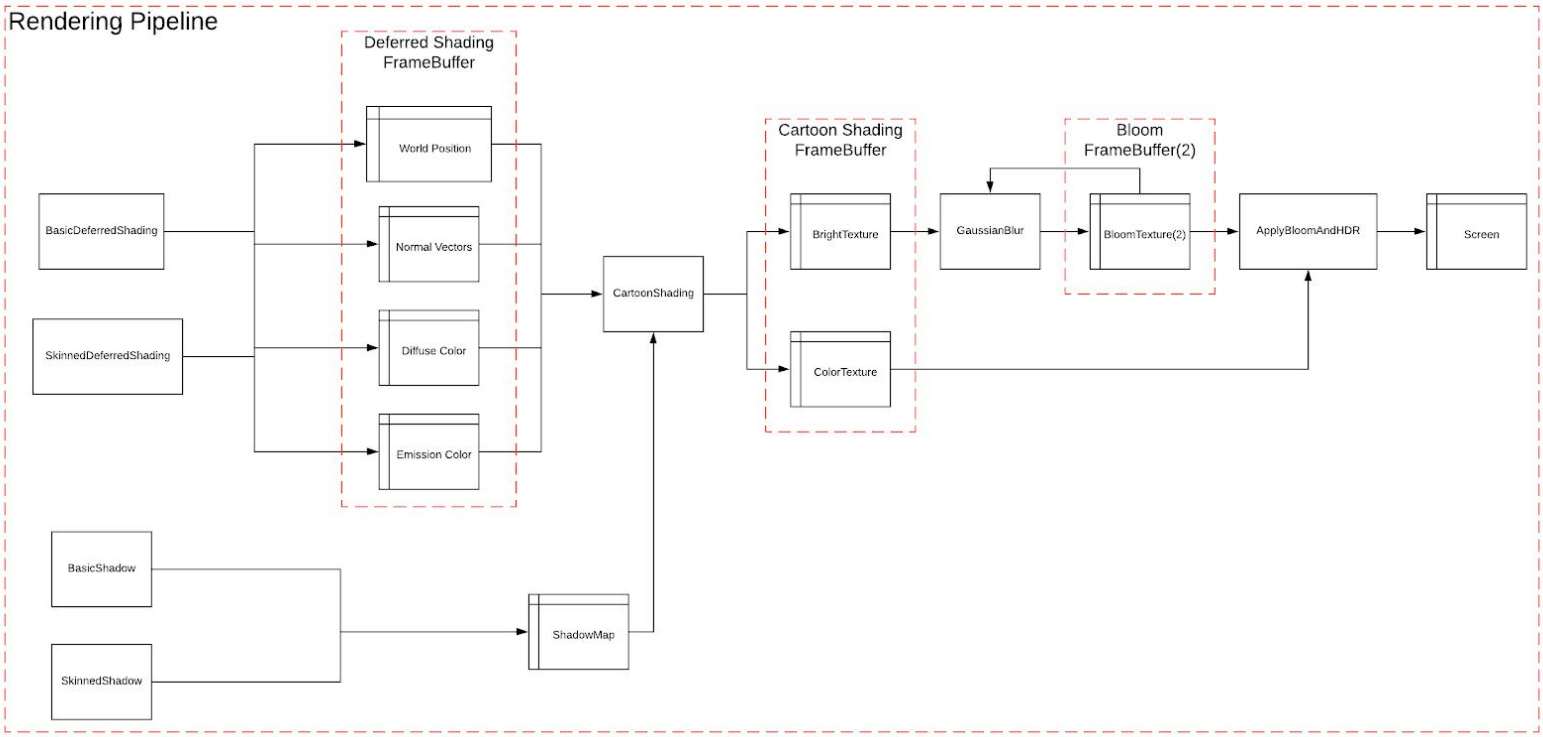
   Similar to textures, it will be a lot easier to draw on online reference images for our 3D models, rather than trying to create them ourselves. This will allow us to skip the hours of work that go into creating good and useful concept art from multiple angles and getting everything to look natural and aesthetic.

5. Architecture
   The specific architecture of the project is still a work in progress. We do have an overall idea for the design of the project however. The main function will create a singleton instance of an Engine class. This class will contain all subsidiary engines (rendering engine, physics engine,), and be responsible for updating all of them. The game will use a scene graph to contain all game objects. This will allow game objects to have a set of children that move with them. Inheritance will be used in the GameObjects as well as in GameState. The following is our current proposed architecture:


main.cpp
Engine
- static Engine *engine;
- RenderSystem *rs;
- PhysicsSystem *ps;
+ void tick(double dt);
+ static Engine *getEngine();
<<LevelLoader>>
GameObject *loadLevel(std::string directory);
<<GameState>>
- Engine *engine;
- GameObject *scene;
+ Engine * getEngine();
+ void tick();
+ handleInput(SDL_Event);
FBXLoader
Autodesk FBX SDK
TextureLoader
- static TextureLoader *loader;
- TextureLoader();
+ static TextureLoader * getTextureLoader();
+ static void deleteTextureLoader();
<<MenuState>>
- MenuState * parent
InGameState
MainMenu
OtherMenus...
stb Image loading library
RenderSystem
Camera * currentCamera;
Window * window;
void render();
PhysicsSystem
Bullet stuff (more research required)
void stepPhysics(double dt);
AnimationSystem
std::vector<AnimationStack> animationStacks;
void evaluateAnimations(int dt)
Bullet
OpenGL
Window
void init();


GameObject
std::vector<GameObject *> children;
GameObject * parent;
Matrix Transform;
AnimationStack
std::vector<AnimationLayer> layers;
void playAnimation(std::string animationName);
void blendAnimation(std::string animation1, std::string animation2, int duration);
void evaluateAnimation(int dt);
ClickableButton
AnimationLayer
std::string animationName;
int currentTime;
bool isPlaying;
void evaluate(int dt);
void startAnimation();
void stopAnimation();
SkinnedMesh
Bone root;
GLuint VBO;
GLuint IBO;
Material Mat;
Camera
Physics
Player
AnimationCurve
std::vector<Keyframe> keyframes;
GameObject *target;
void evaluate(int currentTime);
struct Keyframe
int time;
vec3 position;
quat rotation;
vec3 scale;
Obstacle
StaticMesh
GLuint VBO;
GLuint IBO;
Material Mat;
Bone
Light

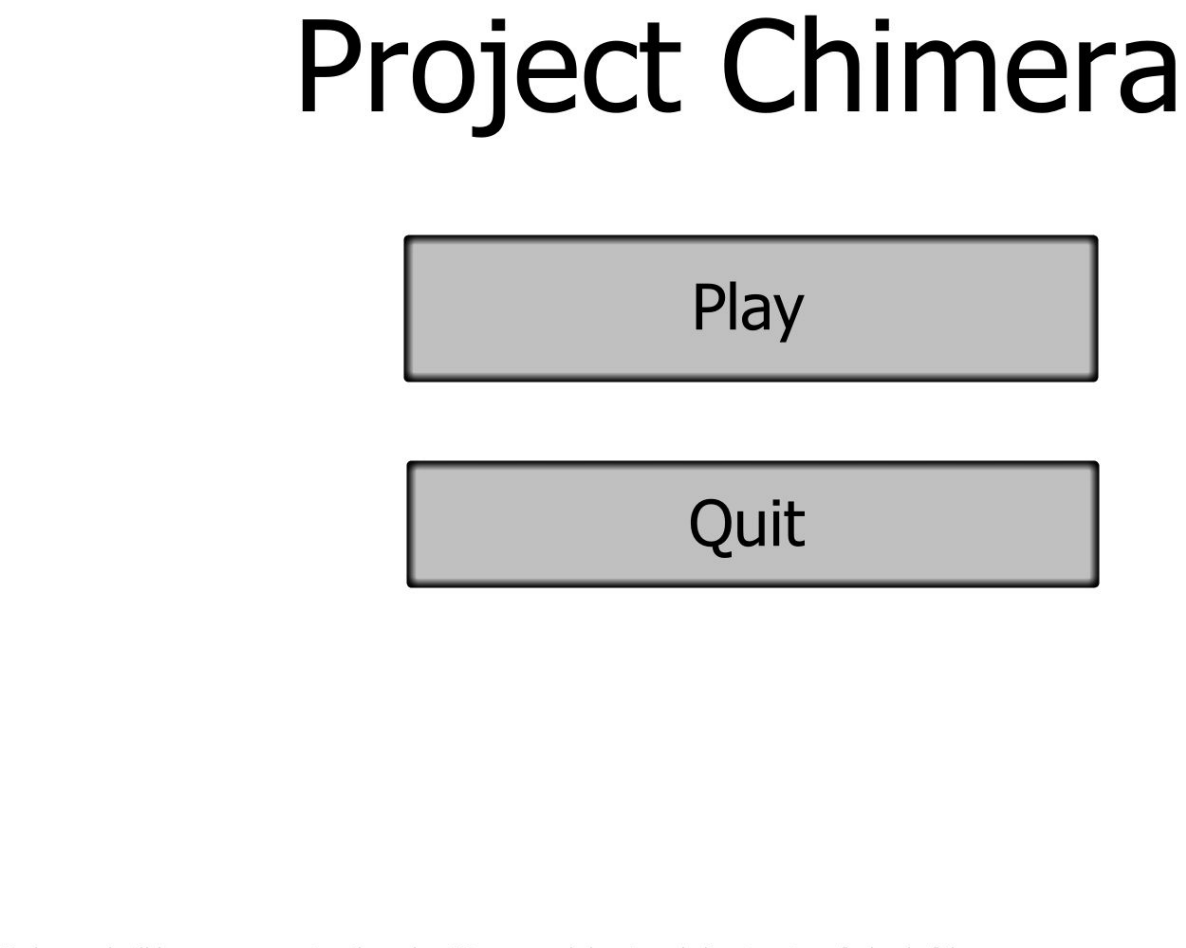
Architecture for the rendering pipeline is as follows: (This diagram does not follow UML, as the diagram shows the flow of data between shader programs and Frame Buffers)



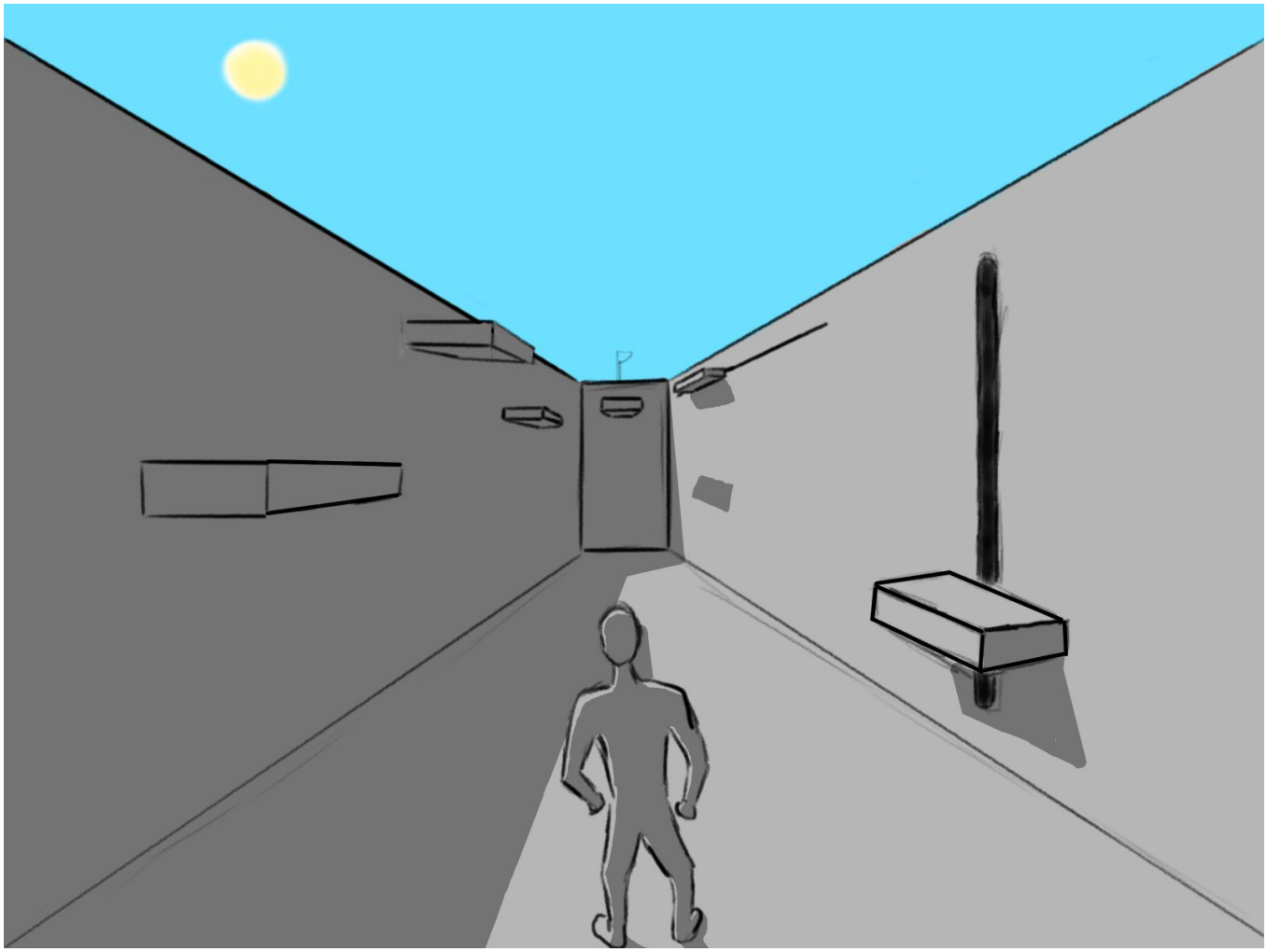
6. GUI

   Our “GUI” will be the game itself. When the game is started, a fullscreen viewport will be created with SDL2, which once loaded, will display the main menu. From the main menu, the player will be able to select to start the game. This will load in a world and models for everything in it, including the character, displayed using a 3rd person camera situated behind the player’s character model. This camera will move with the player and respond to player movements. While these sketches are very plain, they aim to show the cartoon shading and a basic look at what the GUI will look like. If we have time, we will be aiming to make the level and character models have a common theme.

Main Menu:


Project Chimera
Play
Quit
Background will be a camera moving through a 3D scene and showing a helicopter view of a level of the game

In Game:



7. Detailed plan
   The following is our outline for the tasks we want to accomplish in order. Because there is still some more research and playing around to do, we will almost certainly find that there are more tasks that we have not accounted for here or that we will need to change the order or timeline of some tasks, but we will attempt to follow this schedule.

   Rendering engine objectives:
   - Week 1:
     - Tasks:
       - Set up SDL and basic OpenGL - Jason
     - Knowledge:
       - This requires initializing an SDL window and configuring basic OpenGL, which we have done several times.
   - Week 2:
     - Tasks:

        - Set up Deferred Shading portion of rendering pipeline for static meshes - Jason
        - Set up FBX SDK for reading static meshes - Alex
    - Knowledge:
        - Deferred Shading requires knowledge of building an OpenGL custom rendering pipeline which we have done before and will only require implementation.
        - Setting up the FBX SDK will require knowledge about how the SDK's data structures store data about the meshes and how to extract that data, which require a bit more research.
- Week 3:
    - Tasks:
        - Set up shading part of rendering pipeline - Riley
        - Set up FBX SDK for reading lights and integrating them with rendering system - Alex
    - Knowledge:
        - The shading part of the rendering pipeline will require knowledge of GLSL and shading calculations.
        - Reading lights from the FBX SDK will likely be very similar to reading meshes, so the work for this week will mostly be writing the code to integrate that data into the rendering system.
- Week 4:
    - Tasks:
        - Set up bloom and HDR - Jason
        - Set up FBX SDK for reading skeletal data - Alex
    - Knowledge:
        - Bloom and HDR require knowledge of Gaussian Blur and HDR calculation, which we already know how to do, and will require only implementation.
        - Using the FBX SDK for skeletal data will require looking into Autodesk's documentation.
- Week 5:
    - Tasks:
        - Create shadows - Riley
        - Set up FBX SDK for reading animation data - Alex
    - Knowledge:
        - Shadows will require knowledge about shadow mapping, which we already have.
        - Reading animation data will require research in Autodesk's documentation
- Week 6:
    - Tasks:

            - Set up Skeletal animation system for compiling bone transformation matrix data and uploading to shader - Alex
        - Knowledge:
            - Setting up skeletal animation to compile bone transformation matrices requires knowledge of transformation matrices, which we have done in past projects.
- Week 7:
    - Tasks:
        - Set up blending animations - Alex
    - Knowledge:
        - Blending animations will not require new knowledge, rather it will require setting up code to transition between animations by weighting the transformation matrices of each bone.

Physics engine objectives:

- Week 1:
    - Tasks:
        - Set up Bullet dynamic world - Jason
        - Proof of concept for Bullet interaction using hard coded shapes - Jason
    - Knowledge:
        - We've been doing research to understand how to set up the basic Bullet physics world and we'll need to experiment with what we can do with it. Bullet provides a lot of examples with their libraries that we can use to build off of and understand the capabilities that are available to us.
- Week 2:
    - Tasks:
        - Set up adding GameObjects to Bullet's world - Jason/Riley
        - Set up reflecting transformation data from Bullet in GameObject's transformation matrix - Jason/Riley
    - Knowledge:
        - Most of the relevant knowledge here should be developed in our experimentation in week 1.
- Week 3:
    - Tasks:
        - Set up character control and integrating with Bullet - Riley
    - Knowledge:
        - We'll have the knowledge we need by this point
- Week 4:
    - Tasks:
        - Make player object interact with the environment - Riley
    - Knowledge:

            - Again, we should have the knowledge we need to do this by this point and it will just be a matter of making sure the player's physics object is registered with the PhysicsSystem.
    - Weeks 5-onwards:
        - Tasks:
            - Implement more physics based mechanics - All
        - Knowledge:
            - We're not sure what these mechanics could look like yet. We could possibly have boost pads or something like that, but we'll need to do more research and get a basic physics implementation working first.

Other:

- Week 1:
    - Tasks:
        - Set up basic project architecture - All
        - Include appropriate libraries through submodules - Jason
        - Setup Cmake to build the project - Jason
        - Ensure that a fresh repo clone builds as is - Jason
    - Knowledge:
        - A good bit of learning and experimenting will be required to create the basic architecture and make sure it can build without issue. However, once this is in place, it should require little maintenance for the rest of the project.
- Week 2:
    - Tasks:
        - Set up basic scenegraph implementation - Alex
        - Set up basic GameLoop - All
    - Knowledge:
        - We already have the basic idea of how a scenegraph should be set up, so it will simply require implementation.
- Week 3:
    - Tasks:
        - Set up mechanics for interacting with other GameObjects - Jason
    - Knowledge: This mainly requires working with Bullet to produce realistic physics. Since we have yet to work with Bullet in any previous projects, this task may take some research and time to accomplish.

- Week 4:
    - Tasks:
        - Create 3D models for a basic level - All
        - Create 3D model of a character - Alex
    - Knowledge:

        - All three of us have modeled before, so we're good here.
- Week 5:
    - Tasks:
        - Do rigging for the character model - Alex
    - Knowledge:
        - This will be mostly Alex's responsibility as he has experience with rigging.
- Weeks 6-7:
    - Tasks:
        - Create animations for walking, running, jumping, standing etc for character model - Riley
    - Knowledge:
        - As mentioned previously, the group has experience with animations, so it will just be a matter of the time necessary to put in to make the animations look good.
- Week 8:
    - Tasks:
        - Set up animation system to allow triggering animations and integrate with character control system - Alex/Jason
    - Knowledge:
        - This will require a more concrete implementation to understand how we'll want to trigger animations appropriately.

Due Dates:

- February 23 - Finalized Project Proposal - At this stage in development, there is very little that a user will be able to accomplish. If run, a window will appear featuring a black background with a white rectangle, but there will be no further interactions available to the player at this time.
- March 13th - At this point the user will be able to see the 3D world around them and control the player. The level at this point will not have a character model, and the level will only consist of a few boxes for testing purposes. The basic physics engine integration will be working at this point.
- April 3rd - At this point, basic skeletal animation should be implemented. We are not expecting animations to be triggered by any actions or events, but will likely run in a loop. Blending animations will also not be implemented. At this point the player will be able to interact with the world and physics will be applied to the player. If all goes well, the next steps for the physics engine after this point will be implementing more advanced game mechanics, but we have not defined those yet.
- April 17th - At this point the rendering system should be complete, and animations should be triggered by button presses and events. Some animations will be blended together if appropriate. At this point we should have all the 3D models for the player and level. The player will have an animated character model with a custom texture.

- April 29th - The last two weeks will be spent polishing bugs and the appearance of the game. We may add more details into our levels and models, and improve the transitions between character animations if necessary.

8. Staying engaged with the course
   While this is a rather large project, most of it will be coded using C++. This fits in well with this class as, at least so far, that is the primary language we are using. It is also important that we continue to attend lecture and complete all of the assigned lecture activities and programming exercises as they will provide information that could be useful in developing our game. I believe that this project will coincide well with this class and will give us opportunities to both deepen our understanding of C++ as well as learn new programs and libraries such as Bullet, Cmake, and Autodesk FBX SDK.