Refactoring the code was relatively painless, as I [Cody] had pushed Kyle and I to have good code architecture from the start. Before I knew the exact parameters of the assignment, I had moved all of the mini golf specific code into the engine so that GLUT just called the engine’s keyboard, mouse, reshape, etc. functions. Everything was nicely contained within the engine, including GLUT code. After realizing the exact specifications of the refactoring, I moved all of that data out, and removed everything that was specific to mini golf. The engine is now significantly slimmed down, and it is barely reliant on GLUT at all. One minor code change that I don’t have time to do before this deadline will remove its GLUT dependency entirely. The engine now contains information about where the camera and target are, what camera mode it is currently in, what the current level is, and a vector containing all of the levels. Each level object must inherit from LTObject. It is a nearly empty base class that just has update, draw, and GUI methods, as well as a bool indicating whether or not the level is complete so it can advance to the next one automatically. We write our own, mini golf specific level class that inherits from this LTObject, and pass the vector of them into the engine’s constructor, that way the engine has every level it will need from the moment of its creation and won’t run into issues. Slimming down the engine this much means that it doesn’t contain a ton of functionality, however it is generalized enough that it can be used by any game using 3D rendering, with any windowing system. For release, the engine will have LeprechaunTears, a class for the engine itself, LTObject, a class that levels will inherit from, and PhysicsObject, which has many fields and methods related to physics, but vague enough that it can be extended and customized to the creator’s content, or ignored entirely if they prefer to use their own physics, or another physics platform. Note that this architecture allows all level states to be maintained between level changes, which works great for some games. You must include your own reset level functionality otherwise.

Leprechaun Tears API

|  |  |
| --- | --- |
| Field/Method | Information |
| LeprechaunTears(vector<LTObject\*>, int w, int h) | Constructor. The vector of LTObject\*’s contains all of the levels. This can still be modified after construction. W and H are the width and height of the display window, respectively. |
| ~LeprechaunTears() | Deconstructor. Cleans up level information. |
| void update(bool& advanceLevel) | Calls the current levels’ update method. The bool being passed in will be false if the level is progressing normally, but true if the engine detects a level change and advances it for you. |
| void draw() | Call the current level’s draw and onGUI methods. |
| void nextLevel() | Manually advance the game to the next level. |
| void jumpToLevel( int level); | Enter an int where you want the game to jump to. Will print messages to stdout if your number is too low or too high. Although levels are 0-indexed in the vector containing levels, they are counted from 1 to n, not 0 to n-1. |
| OnGUI(string text, int x, int y) | Supplement’s the engine’s calls to OnGUI by drawing additional text to the screen. Text is the string you want to render, and x and y are the location on the screen. |
| int currentLevel | Current level, being used to index the vector containing the levels |
| int cameraMode | Contains information on the current camera mode. This must be defined outside of the engine, but can be kept track of here. |
| vector<LTObject\*> levels | Contains information on every level. It can be modified after the constructor was called from here. |
| float camx, camy, camz | Location of the camera in 3D space. |
| float targetx, targety, targetz | Location of the point that the camera is looking at in 3D space. |

Physics Object API

|  |  |
| --- | --- |
| Field/Method | Information |
| PhysicsObject() | Default constructor. |
| ~PhysicsObject() | Default destructor. |
| (virtual) void update() | Update this object’s logic. |
| (virtual) void render() | Render this object. |
| void addForce(glm::vec3 force) | Apply velocity according to the vec3. |
| (virtual) float getAngle() | Returns 0. Meant to be overridden. |
| (virtual) float getMagnitude() | Returns 0. Meant to be overridden. |
| (virtual) void changeAngle(float f) | Changes angle according to f. |
| (virtual) void changeMag(float f) | Changes magnitude according to f. |
| glm::vec3 getPosition() | Returns the position in 3D space of this physics object. |
| glm::vec3 getVelocity() | Returns the current velocity of this physics object. |
| (protected) glm::vec3 position | The position of this object in 3D space. |
| (protected) glm::vec3 velocity | The current velocity of this object. |
| (protected) glm::vec3 acceleration | The current acceleration of this object. |
| (protected) float mass | The mass of this object. |
| (protected) float friction | Constant value that will cause it to slow down on update ticks while moving. |
| (protected) void addDrag(glm::vec3 amount) | Adds drag to velocity by that amount. Handles x, y, and z components separately. |

LTObject API

|  |  |
| --- | --- |
| Field/Method | Information |
| (virtual) void update() | Update this object’s logic. |
| (virtual) void render() | Render this object. |
| (virtual) void onGUI() | Render information about this object to the GUI. |
| bool levelComplete() | Return whether or not this object has been completed. Engine will use this to advance levels automatically. |
| (protected) bool completed | Initially false. Turn this to true when you are ready for the engine to advance to the next level. |