Project Description:

Project Title: Third Person Portal

This project is my stab at recreating a game similar to the title created by Valve known as Portal, where the objective of the game is to solve puzzles or "rooms" that encompass a wide variety of interactable objects such as boxes and buttons by reaching the exit. The objective is to navigate and solve these puzzles using portals, which can be fired by the player (maximum 2) at walls, floors, or ceilings, and then traversed through going from portal to portal.

Competitive Analysis: Projects I've viewed (and played myself) that are being used as reference for my term project is the aforementioned Portal game. They will be similar in regards to implementing the idea of using portals that can be shot and placed throughout the room in order to solve the puzzle. Portal is played from a first person point of view, my term project will be played from a third person perspective, with a moveable camera observing from above the puzzle room as the player moves their character from inside, aiming their portal using their mouse as opposed to aiming with the camera. Portal also includes moveable platforms, harmful projectiles, and a boss, which my version will not.

Structural Plan: The game takes place in 3D space, so everything will be organized in terms of [x, y, z] coordinates, including boundaries, wall, the player, portals, objects, etc. and then converted into a 2D projected image to be called in the redrawAll() function. Objects, the player, portals, buttons, and walls/floor will be interpreted as object dataclasses. Maps will be made from lists of wall and object placement configuration. Boundary detection (walls, floors) will be done by detecting intersections with their planes in an isLegal() function. Portals will be placed by using mouse clicks, and the object will be updated according to its placement, and another function will detect which direction its facing by finding the orthogonal vector from the plane it lands on and then correct itself as to prevent the portal from going through the floor or other walls nearby. Physics in this world is done by using object methods I labeled drop() and bounce() to simulate gravity. Button detection will be done by calling a function isPressed(), players will traverse through portals by calling inPortal(entryPortal).

Algorithmic Plan: Ray casting from the camera toward the location of the mouse on the screen. This involved a series of some very technical linear algebra matrices conversions being:

1) creating a plane of some distance d in front of the player by taking the unit vector in the direction of the camera to the point being looked at, multiplying them together and subtracting them from the camera coordinates to get the point at which the camera is facing 90 degrees toward the plane.

2) Finding the unit vectors of the created plane pointing up and to the right from the perspective of our screen (2D), which took projecting an arbitrary point some distance [0, +y, 0] up from the plane center point to obtain the jHat unit vector, and then cross multiplying it by the plane to get an orthogonal vector from both jHat and the cameraToPlane vector, to get iHat (our x-direction on screen).

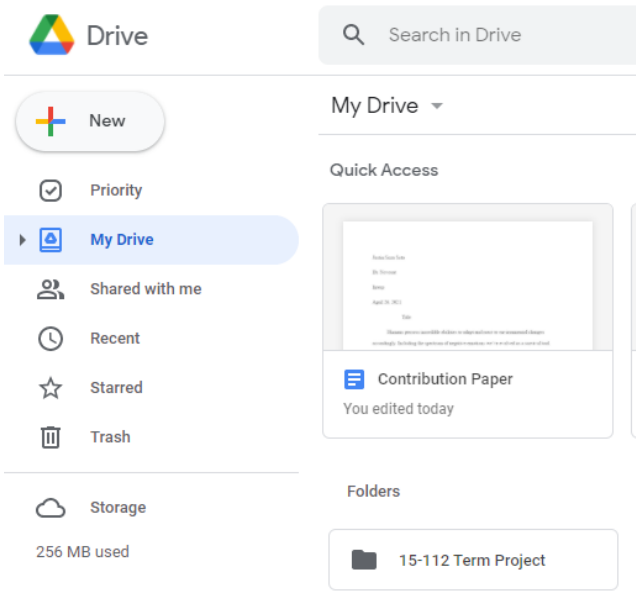
3) Scaling our mouse click/drag/movement down into a range of [-1:1] by creating a new function that takes in event.x and event.y, and then multiplying the new plane's jhat and ihat vectors by the scaled x and y coordinates, and then adding them to the plane's 'center point' to get the point we want to raycast toward.

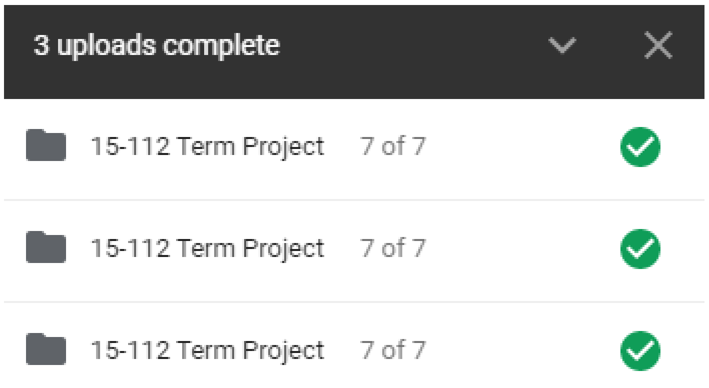
4) A line (ray) is created going from the camera to this new raycast point, and traveled continuously until it hits a boundary of some kind (floor/wall) by using a function that converts the line into a parameterized equation of x, y and z in terms of t, it then returns the [x, y, z] coordinates of where it lands.

Timeline Plan:

TP0 - TP1: Raycasting, 3D object physics, (Halfway done) Placing down portals
TP1 - TP2: Placing down portals, traversing through portals, player object class, moving player, player ability to jump, ability to pick up boxes, button being pressed detection (player weight or box weight)
TP2 - TP3: Maps, sprites, win condition, level clear cue, (enemies? - player detection, health, healing)

Version Control Plan: Folder on CMU account google drive




3 uploads complete

15-112 Term Project 7 of 7

15-112 Term Project 7 of 7

15-112 Term Project 7 of 7

Module List: Numpy