GeoDash 3D

A C++ 3D port of the popular game Geometry Dash

# Document Purpose

This document’s objective is to document the project overall, a sort of timeline of events of what I intend to accomplish and how I ended up executing.

## Project Goals

The goal for the project is to re-create the poplar game *Geometry Dash* using the Open GL API with C++ to put into practice the skills and knowledge I picked up during my time in university.

This project is a portfolio piece, so it doesn’t have a finished viable product in mind, just a showcase of skills and knowledge in software development and games development.



Figure 1: Geometry Dash

## Technical Goals

Technical goals are straightforward, use advanced low level programming techniques to establish a product that can easily be further developed by the public into their own creation.

Other outcomes I wish to achieve:

* Higher level of understanding and use of Graphics API’s
* A technical showcase of skills in developing low level programs
* Use of Object Orientation Programming paradigms, such as showing off different design patterns and optimisations
* Ability for developers to clone project and add their own features

Software tools being used:

* C++ 20
* GLAD/GLFW Libraries
* CLion IDE
* Github

A lot of inspiration on how systems will be established, such as the use of components, events, objects etc., will use Unreal Engine as an example.

# Initial Game Outline

GeoDash3D will draw heavy inspiration from the game Geometry Dash. The core objective is to guide your character through a series of platforms and obstacles to reach the end of each level. Unlike traditional platformers, the player does not control forward movement — the character moves automatically — and can only control jumps. The initial goal of development will be to replicate this game loop: a cube-shaped character progressing through levels by jumping at the right moments to avoid obstacles and reach the finish.

## Initial working prototype

What I will consider a fully working prototype will need to have the following:

* A Cube character that:
  + Player can input a button to make cube jump
  + Cube moves forward automatically
  + Cube has collision that stops it from falling through the floor
  + If a cube hits something front on it will reset the level (with other considerations based on the type of object it hits)
  + An end goal of reaching the end of the level with a win screen
* Objects & Obstacles
  + Basic wall/floor objects that player can be on top of but can’t collide head on
  + A “Spike” obstacle that destroys the player no matter how it collides with it
  + A Bounce Pad that launches the player into the air at a higher level then what a normal jump can do
* Level Editor
  + A in game level editor that allows players to create their own levels using the objects/obstacles that are built into the game
  + Levels will save into a binary file with the extension “.G3D”
  + The main menu will have a way to load in levels
    - This will open a file explorer window, asking the user to find and select their “.G3D” file to be loaded into a specific folder in the game’s files
    - The players will also be given the option to edit these files through the level editor
* Main Menu
  + Play button that opens a level select menu
  + Level editor button
  + Load level button
* Level Select Menu
  + Shows all loaded levels with preview widgets

All the above will be used as a groundwork for a prototype build of the project, it is not the final product. I will add more design and details as I further develop from this initial concept.

# Technical Design

## UML Diagrams

<https://drive.google.com/file/d/1Zo191hFVnJD_m2_beG95Pw8NEu7ZUg-m/view?usp=sharing>

## Research

<https://www.youtube.com/watch?app=desktop&v=jjaTTRFXRAk&t=3m11s>

This video discusses the OpenGL concept of a renderer. This was going to be an early issue I needed to solve before I got too deep, as I was working on the Static Shape Render I thought about the possible issue of sharing a camera system across all the instances of any Object that needed to be rendered in. The Cherno covers this concept in detail, giving me the idea of what I need to do before I go deeper in.

Changes from this:

* Added Renderer Class to UML
* Reworked code to use a global Renderer as a source for perspective rather than static “magic values”