Cube Jumper

This game was created by Joshua Petherick and Kyle Tuckey for our second semester coursework in Game Development.

Table of Contents

Introduction

Cube Jumper is a tribute to the original 3D platformers that were popular during the 90's. It features five levels where the goal is for the player to collect diamonds and reach the door at the end of each level. You, the player, must navigate through the challenging levels and gaze upon the surprise within the final level.

Our repository: https://github.com/Malithium/Semester_Two_Group_Project

Running the Game

Linux

This information is available on our README however I will repeat it here usefulness.

You will require:

- GNU Autotools
- OpenGL 3.0
- C++11 compiler (tested with GCC 4.8.3+)
- Boost
- GLEW
- SDL2
- GLM

On Fedora 20 or later you can install these using a single command (as root):

$ yum install boost- glew-devel SDL2_* glm-devel*

After cloning the source code or extracting a distributed archive, change to the directory where the source code is:

$ autoreconf -i

$ ./configure

$ make

The build process should create a binary that can be executed as follows:

$ ./src/shaderexample

This build process was obtained from: https://github.com/AidanDelaney/glex

Windows – Visual Studio 2013 setup

For visual studio you will need

- Opengl 3.0+
- GLEW

- SDL2
- GLM

Once you have downloaded the folders you will need to extract them somewhere and then copy the .dll files to the projects debug folder.

You will then need to access the project properties so that we can link our project to the include and library folders of each package.

In properties under configuration properties in the VC++ Directories menu, select the include directories field and click the edit button.

Look for the folders you extracted and select all of the include folders of each package.

Under the same menu look for library directories and again look for each library folder within each package.

Once you have included the library's and include folders in the project, go to the linker section within the properties and click on input.

Once you are in the input section click on additional dependencies and add SDL2.lib, SDL2main.lib, opengl32.lib, glu32.lib and glew32.lib.

Once you have done all of this the project should run.

How to Play

You move the player using W and S, with W going forward and S going backwards. You will use the mouse to look around and will move in the direction you are facing.

Game Construction Breakdown

Camera control

The camera works in a complicated manner. The camera class itself is responsible for handling the various calculations needed in order to move the Camera as well as look around, it also handles the maths behind gravity and jumping. The basic goal of the camera class is to take the spherical coordinates of our vertical and horizontal angles and convert them into Cartesian coordinates that is stored in a vector3 and added to our cameras position, this allows us to move our camera and look around. The user input to allow this however is handled in the events class where it uses the SDL event system to detect key presses and mouse movement and calls the corresponding action, so pressing W moves our camera forward and moving the mouse allows us to look around.

https://github.com/Malithium/Semester_Two_Group_Project/blob/master/Semester-2-game-build/src/camera.cpp

Block World

We have, technically, produced six cube worlds with each having a different layout and design. There are three cube sizes, five if you include the diamonds and doors, and each have their own resting spot. These levels are loaded in from .JSON files, found within the level folder, which contain XYZ co-ordinates for all the individual assets. Level.cpp loads these in, based on what the value of lvl is. The cubes are all coloured, but not textured, and there is no model for the character - the camera itself represents the character (Designed this way to make the player feel more involved).

https://github.com/Malithium/Semester_Two_Group_Project/blob/master/Semester-2-game-build/src/level.cpp

https://github.com/Malithium/Semester_Two_Group_Project/tree/master/Semester-2-game-build/levels

Voxel Structure

The game uses a vector with gameAssetManager to store different assets, such as cubes and diamonds. The level loop then uses commands to interact with these assets (Such as collision) and to draw them every frame.

https://github.com/Malithium/Semester_Two_Group_Project/blob/master/Semester-2-game-build/src/gameAssetManager.cpp

https://github.com/Malithium/Semester_Two_Group_Project/blob/master/Semester-2-game-build/src/CubeAsset.cpp

Use of Git

We have two branches on our repository, Dev and Master. We used Dev to make changes to our game, and then pushed it to Master if we were happy with the results. This meant that Master was ready and playable for people who wanted to clone it but gave us the freedom to play around and make changes to the program.

https://github.com/Malithium/Semester_Two_Group_Project/tree/dev

Automated building

We used a boost compiler produced by our Game Development tutor, referenced in the README, which allowed us to compile it and create a working executable. The program will need to be run from within the "Semester-2-game-build" folder, otherwise it cannot find the shaders or level folders (Which is a known limitation). If we created any new classes then we must edit the Makefile within src and add these to the current list - meaning that they will be involved in the compiling.

https://github.com/Malithium/Semester_Two_Group_Project/tree/master/Semester-2-game-build

Documentation

We continually commented work we did, making communication between us easier when we added new code, but it also meant that when editing the files we knew what segments of code was

designed to do. The week before the 30th we added extra comments to prepare an informative Doxygen file, and this can be found within the repository.

https://github.com/Malithium/Semester_Two_Group_Project/tree/master/Design

Multi-platform

The "Running the Game" section shows how we ran these programs on multiple operating systems, and we were in the process of testing if there was a way to run our program on iOS.

Project Milestone notes

This section features notes we made during the creation of this project, and is interesting to see how we hoped the program would look when finished and how it actually did.

11/02/2015 : It will feature two levels, to begin with, and hopefully time won't constrain us from expanding it beyond this small design. The player will run around collecting diamonds, which will try and avoid the player using A* path-finding, however we will be using OpenGL to load in blocks of cubes to create interactive levels.

11/03/2015 : The game will now include 5 tutorial levels, and once these are completed we hope to add an additional level with high difficulty. Core mechanics, so collecting diamonds and traversing through the level are still the main goals, but we hope the level designs will provide the challenge to our game. Code wise we have transitioned from into calling cube assets (Of different sizes) in preparation for bounding boxes (Collision detection). We are facing difficulties with the camera, but I trust my colleague to have it working completely by the end of the month.

27/03/2015 : Due to making progress a lot faster than anticipated I thought I would give a brief update regarding where we currently are in regards to a finished product. We have created a system that generates levels based on vector positions stored in text files, effectively giving us the ability to load and unload levels. We have also made some major changes to the Camera class and it's condition is now satisfactory, this means the majority of work on the camera will be bug fixes and performance tweaks.

Our next goal is to create bounding boxes around our objects so that we can start implementing collision detection into our program. This will enable us to implement gravity, jumping, collectables, doors and allow us to properly transition between level's. After this feature has been implemented we have effectively met all of our core goals and can work towards finishing the game. As my colleague has more experience with drawing our objects to the screen I shall leave the task of creating bounding boxes to him, in return I will work on

the math's involved with collision detection as I have more experience with the camera class where all the camera movement happens.

14/04/2015 : With hard work we have successfully created a fully functioning Platformer, with multiple levels and shy collectables. We endured many bugs and coding errors, and still have several improvements we'd like to implement, but we're still proud to announce that it is functional. This was created using several machines, all using different operating systems (Ubuntu, Windows 7 and Fedora) and we will post instructions on how we tested our game until we've looked into creating an executable. Until then, feel free to explore our code, study our commits and use it however you see fit.

https://github.com/Malithium/Semester_Two_Group_Project/blob/dev/README.md