Code Documentation

I decided early on to use Bullet Physics because from looking at demos it seemed to implement gravity and friction well, which is specifically what I needed. It also seemed fairly easy to use and understand. I started by looking at the demos and the user manual and trying to understand the process and how it worked. I took from the Bullet/NGL demo the idea of the “CollisionShape” and “PhysicsWorld” classes. Within the “CollisionShape” class I create a simple shape from a simple obj mesh, (for the ball and cube). The first issue I encountered was trying to create the collision shape for the maze mesh, this caused problems as it needed to have holes for the ball to fall through. This meant I needed to create a concave mesh, I used the “btBvhTriangleMeshShape” function. This took quite a few attempts to get it right but I did this by adding triangles to a triangle mesh then using the data from that to create the BvhTriangleMeshShape.

The “PhysicsWorld” class was more simple, creating rigid bodies for the objects and setting physics attributes. From the user manual I learnt about the difference between dynamic, kinematic and static bodies. I set the maze as a kinematic object as it should not be affected by physics, but the user should be able to move it, whereas the ball would be a dynamic object. I then had to create a new method that I could call to rotate the maze, for this I had to learn about quaternions as this is how bullet physics calculates rotation. Within this class I also defined the “contactTest” method which finds whether the ball and the cube have collided (to see whether the player has won or not). Originally I had not planned how I would know when the player had won. The first option was checking when the ball had reached a certain point in either world space or on the maze, however in world space this point would change as the maze moves. The easiest way therefore would be to place a cube at the end, make this a kinematic object and rotate it the same amount and same direction as the maze. Then return when the ball collides with this cube as the player has won. To know when the ball and the cube had collided was what I found to be one of the most difficult parts of the code, owing to the fact that I had to create a callback which was difficult to understand. I also found a problem with the physics once playing the game, the ball would bounce quite alot as the maze moved and also would get stuck to the walls. I managed to fix this fairly well by increasing the gravity and reducing the angle increment to reduce the bouncing. Additionally I reduced friction on the maze object to practically zero and the friction on the ball to below 1 (around 0.3 worked well) to be the best balance between keeping friction but little sticking.

The “NGLDraw” class contains the majority of methods to draw and set up the game, including binding shaders and textures, setting and initializing the physics world, additionally setting the different game states and creating the key and mouse methods and the reset for winning or losing the game. There is also a function to create extra balls, which I added in as an extra as it adds more entertainment but unfortunately I didn’t have time to properly set it so that these extra balls affect the win or lose state.

Within the main I set the keyboard controls, for example for the rotation to update the angle as the key pressed but to stop when the key is up. I also created the timer within this that counts the score (the quicker you complete, the better the score). The timer was another problem that took me awhile as I used the “SDL_Getticks” which counts immediately from when the program is started, so variables needed to be created to count when it is paused etc. Additionally I created the functions needed for parsing files. in my config file I decided to put gravity, friction and high score, so these are the options available to change that are read in and saved back out.

I chose to use SDL for the keyboard controls as it was most intuitive for game-style controls and handled and moved the maze well. I added text using he files written by Jon Macey to add more usability and menus to my game and then also texture shaders as well as the phong shaders so that my maze has a wooden texture but the ball and cube have material shaders. Unfortunately I didn't get lighting to work on my texture, it would have looked nice to have specular highlights on the maze as

well as the ball and the cube but this didn't work out. There were a couple more things I'd like to improved other than my shaders and textures, I would like to set up a procedural maze so that each time you complete one a new maze is created but I couldn't do that within the time limit unfortunately. Overall I enjoyed this assignment and I feel like I have improved a great amount especially at understanding how to read code and what you can and can't do. Using a physics engine I found challenging but once I understood it, it was really great to work with, especially changing different attributes to find different effects.

Data Structures and Algorithms

I've used object orientated programming in my code, using classes with headers and source files and public and protected members and functions.

Also techniques like if/else, switch/case statements, for loops, arrays, vectors, pointers etc.

Within my code I've used 'structs' e.g. in "PhysicsWorld" to create the callback for my contact test. Additionally within the .h file to create a record for an object containing it's name and rigid body information.

In the 'struct' for my contact test, I've used a virtual function to find whether the objects colliding are the cube and the ball.

I have also used inheritance and polymorphism where this struct is derived from the Dynamics world public class. Additionally the "PhysicsWorld" class is called within tthe "NGLDraw.h" file to be able to use it's protected and public functions and members.

In the main file I've created parsing functions to read in the config file which contains data for the gravity, friction and also saves the high score.

How To Operate

When my game is started you start in game state 0, which is the start screen and explains controls.
A- to play
Arrow keys - to rotate the maze
B - to add more balls
ESC - to quit
Mouse Scroll wheel to zoom
Use Left button to orbit around the maze
Use Right button to pan
W/S - wire frame/solid mode

Within the config file
Gravity - (Y value strength of gravity, set to -100 as default for best results)
Friction - (set 0.3 as default for best results)
High Score - will update as you win the game quicker