PHYSEXE

Technical Design Document

Aaron O Neill
C00241596@itcarlow.ie

Contents

Background

When creating games there are a lot of aspects that can be very tedious and also difficult to do. One of these aspects is creating a level without using an editor. It can be very tedious writing the code to set the position of everything, changing their properties and running it, closing it and making micro-changes and recompiling to see if you like the look of it, I wanted to combat this problem by making a program that would easily allow the user to experiment with level design and be able to run the level and see how the level would work in practice.

For this reason, I wanted to design a level editor that would allow the user to easily create a level, modify any shapes they wished, put any sprites they wanted into the level, change all the properties of each shape and export the level there. Although that alone would leave a lot for the developer to do, i.e they would have to bring that level into their project and unpack the level, this would be very troublesome for someone to learn, so to solve that problem I plan to make the project also come as aDLL so you can easily import the Level loader and a manager for all the shapes into any project you want.

Motivation | Requirements

I wish to create this program to make the creation of levels and maps much easierand less tedious for people to do. The software should be very simple to use, the flow of use should be as simple as:

- Launch the program
- Create the level | load a level they have madeSave
- the level
- Import the level into a new project.

That should be all the users would need to do to get the level to work. Once the userimports the level into his project he will need to code ‘what’ exactly happens when certain things collide. We can offer support for most of it, and we can have the ability to change what happens but a lot of the finer details will be left to the programmer implementing the level.

The user should be able to create a wide variety of shapes as well as seamlessly be able to modify the shapes in any way they might want to. This should all be handledthrough a very easy-to-use UI system. The user should also be able to ‘Run’ the level, this will allow them to test things and make sure that everything is working theway they want it to.

The user should be able to move | rotate | scale and connect all the shapes easily.

When moving, rotating, and scaling the user should be able to see a preview, i.e nothing should collide or fully more until they have finished editing the shape they are working on.

Technical Requirements

Overview

Throughout the document we will need to address a lot of features and areas surrounding the project, an overlook into each of the features and why we need themto consist of:

Box2D

Box2D is the library that I will be using for the physics side of my project, So I needto make sure that the visuals I'm showing off the level match up with the physics box2D are calculating, this was the first main hurdle for the project.
As box2D uses kilometers in its measurements so if I keep the scale the same forevery 1 pixel would be the equivalent to 1 kilometer in space, so everything wouldmove slowly, I would need to give things extremely high velocities to make them move, etc.

So I had to scale down the box2D world by a constant (in my case I used 32) soevery n number pixel on the screen was a kilometer in the world, where n is the constant.

World Manager

I made a world manager for the software as a few different classes needed to accessthe world and rather than passing it around and making everything need references Icreated a singleton class called ‘World Manager’. The responsibility for this class was to hold a single static instance of the b2world so that anything that needed could get a single instance to the world.

UI

The user will need to be able to interact with everything on the screen. For example,when the user loads into an empty level they will need to be able to create and modify any one of a select number of shapes. They will hopefully be met with some pre-baked shapes on the side of the screen that they will be able to select as a template and just stamp down on the screen.

A rough design for this screen is as follows:

**** INSERT DIAGRAM****

After researching a few different UI libraries (See Research Report), we will be usingTGUI to handle all the UI needs we have inside the project. TGUI offers all the necessary elements we will need for handling everything the user might want to do inside the project.

GUI Manager

The GUI manager uses TGUI so the user can create | modify and edit shapes usingall the UI

at their disposal, this includes having a panel that shows:

- Shapes that can be created.
- Menu to show the editing options. (move | rotate | scale | creating dist/wheeljoint)
- A menu to modify any selected shape. A
- menu to modify any selected Joints.

The creation of all the menus and the managing of all the menus will take quite a long time to create which is why it should be in a singleton, as there are going to be a lot of UI elements that will need to be managed and we don't want to have a duplicate number of them.

Shapes

As there is a wide variety of shapes that the user will need to be able to draw, we will need a very easy way to store all the shapes, like edges, circles, squares, hexagons, etc. So to accommodate drawing all different primitive shapes and also sprites I decided to use a vertex array for all shapes and sprites except for circles, I created a PolygonShape that will manage all shapes except circles, so this shape is drawn using a vertex array, the points on the array will match the number of sides the shape has, and then the points will be calculated by getting the center point, rotation and size to calculate the points of the shape at any given moment, this polygon shape can also be given a texture, it assumes that the texture will want all of it shown so it gives the vertex array the texture and gives the UV coords of all the corners of the texture. Alongside that I also created a CircleShape which is used only to handle the updating and rendering of a circle shape, for this, I just need to set the radius to the radius times the constant conversion rate so the circle size in pixels matches the size of the shape in accordance with the box2d world. Each shape is fitted with a conversion function to and from JSON, this allows the shapes to be saved and loaded using JSON. Each shape also gets a callback function that gets called when the shape either collides or triggers with another object, when 2 shapes collide with each other and neither of the colliders is a sensor the onCollision enter is called, if one of the shapes is a sensor the onTriggerEnter is called.

Shape Manager

The shape manager will be the most important class in the project. It is the class that will manage all the shapes, it is also the class that will be into all the projects that wish to use physexe to make their game.

The shape manager will be responsible is responsible for storing | updating and rendering any shapes that have either been loaded in from the level loader or created through the API for the shape manager. The manager gives the user the ability to create any primitive shape and sprites through the API on the manager class, the manager will

then create | set up andstore the needed shape depending on the manager. The manager also gives you the ability to instantiate a shape that has already been created, either stored in or outside of the class, all you need is a shape, and it will instantiate the shape, by doing so it will copy all the relevant information it needs from the shape and create a new shape based on the copied information. This canbe very handy for a spawner of any kind, or a gun that needs to shoot bullets. You can just make one bullet and then instantiate it whenever you need to. This featurewas added to make life easier for the user. The manager also allows you to clone shapes, the reasoning for this would be, if youwanted to have a bullet that you can instantiate whenever a gun fires in a game, rather than programmatically creating the first bullet you can simply make one in the editor and then clone it when the program starts, this can be a very useful feature forthe user.

The manager has an array of helper functions that are all overloaded to take a wide variety of overloaded arguments, so you can get find a shape by index, name, or tag, you can flag a shape to be destroyed by index, baseShape*, b2Body* or const baseShape& which is what you will get on the callback functions.

Shape Builder

The shape builder is the main component for creating | managing the shapes inside the software, this class will not be exported to the DLL as it is only used inside the software. The shape builder uses an IBuildState* as a base pointer and then uses that to polymorphically swap between all the states the builder can be in. The states it canbe in are:

Create

This state is in charge of showing an outline of the shape that is about to be createdand then confirming the shape's position when the user clicks the mouse. When the user selects a shape, the shape should be shown at a lower alpha so the player can see everything underneath. This state should have shortcuts for different shapes so the user doesn't have to goover to the UI all the time, so the user should be able to use the number keys to change shape.

Move

This state will give the user the ability to click onto any shape that is on the screen and freely drag it around, the shape will start locked to the mouse's position until the user releases the mouse button.

Each shape will also have two arrows laid over them that will allow the user to move the shape along one axis alone.

The arrow facing horizontal will move the shape along the x-axis and the verticalfacing arrow will move the shape along the y-axis.

Rotate

This state will allow the user to rotate the currently selected shape, once the player clicks the angle of the vector from the player's position to the mouse position will berecorded, and then once the player moves the mouse the new rotation will be calculated, and the shape will be rotated by the change in the angle.

This will allow for a smooth rotation and give the user a lot of control over the rotation of the shape.

Select

This state will simply poll the shape manager whenever the user clicks and check if the mouse position at the moment of clicking was hovering over any shapes on the screen.

If it was on top of a shape, then the shape manager will return the relevant shape and the select state can update the common static selected shape that is shared across all states.

Scale

This state will allow the shape to scale and get as big as the user wants.

The way the state works is you can click on a shape to select it or if you click it will scale the last scaled shape, it will take the distance from the shape to the mouse andthen set the scale to that distance.

Shape Editor

The shape editor is a panel inside the GUI that allows the user to modify all the elements of any given shape. When you create a square for example it's going to be at a specific size, density, rotation, body type, etc... Through this panel, the user can change everything about the shape. This is where the user is met with all the options that they can tinker around with to get a different experience, options that include name, tag, position, scale, body type,

trigger, visible, friction, density, threshold. By changing these values alone, the player will be able to create all different types ofmaps.

Joint Builder

The joint builder is another essential part of creating levels inside the software, the user will not have access to this class through the DLL, but it is an essential part of joining shapes together in the managing software, the main joints inside the softwarewill consist of distance joints and wheel joints.

Distance Joints

Distance joints ensure that 2 shapes don't move within the closest limit of each otherand ensure they don't move too far away from each other either. So, when creating a distance joint if the min distance == current distance == maxdistance, then the shapes will stay the same distance away from each other. This can be very good for games like cut the rope, where you one shape to staywithin a certain distance of another shape.

Wheel Joints

Wheel Joints only allow one body to move along one axis, so this joint works similarto the suspension on a car, when you apply upward pressure on the wheel (bottom joint) the wheel can compress the spring allowing it to move vertically but the wheelis not allowed to move along any other axis. This is very good for the simulation of the motion of wheels on a car.

Joint Editor

The joint editor is used whenever your level needs joints, the joints are initialized with data it assumes is correct based on the two shapes that you selected to be partof the joint, but the joint editor allows the user to tweak the variables to make sure you have the behavior exactly how you want it. Take distance joints for example, when creating distance joints assume you want tolock the shapes at that distance, but maybe you want to allow them to get closer butnot further away. To accomplish this, you only need to click on a white circle that will appear at the mid-point on the debug line for the joints, this will bring up the GUI specific to that joint

and allow you to change the properties for the joint.

Level Loader

The level loader is a very important part of the project, and it is a class that is included in the DLL. So, if you create a level using Physexe, you will get access to the internal level loader that it uses to save and load levels into the editor. The level loader is very easy to use, it comes with one simple function and that is load level, you need only call that single function and pass it the path to your level, and it will handle the creation and loading of any sprites needed for the level.

Contact Callback

Contact callbacks are what allow the user to add custom functionality to the collision. Using the contact callback, the user can add all the functionality they want. If you want to make it so when the “Enemy” shape collides with the “Player” Shape the “Player” loses health. All you must do is add to the OnCollisionEnter contact callback on either the player or the enemy and check if the shape they collided with is called the opposite and then decrease the player’s health. This is a vital part of the shapes for allowing the user to easily be able to add their functionality to the shapes.

Texture Manager

The texture manager is a singleton that will allow the player to load in textures and keep a reference to the texture while the level has been loaded. If the player tries to load a texture that the texture manager already has a reference to, the texture manager will respond as if it loaded it incorrectly but just give back the same texture again, this can be very handy when you only have a certain amount of memory, and you don’t want to waste it by having to load the same texture in multiple times.

Technical Design

As there are two main technical aspects to the project, that being the API that the user will get to use through the DLL and the backend design that will handle the GUI and everything about the creation | modification | saving and loading of levels.