The Lazy Man’s Cycle Starter

August 6, 2016

<table>
<thead>
<tr>
<th>GroupName</th>
<th>RollNumber</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>!JustCoders</td>
<td>140050031</td>
<td>C Vishwesh</td>
</tr>
<tr>
<td></td>
<td>140070003</td>
<td>Saurabh Garg</td>
</tr>
<tr>
<td></td>
<td>140070031</td>
<td>Aviral Kumar</td>
</tr>
</tbody>
</table>
HONOR CODE

"I pledge on my honour that I have not given or received any unauthorized assistance on this assignment or any previous task."
1 Introduction

1.1 Purpose
This project has been done as a part of the course CS 251 : Software Systems Lab offered to CS Majors at IIT Bombay in the second year. This SRS cum Report gives a semi-detailed description and the specific software requirements for running our Project.

1.2 Scope
- This is a project aimed at making a Rube Goldberg Machine in the Box2D environment
- The software basically simulates a Rube Goldberg Machine by the name *Lazy Man’s Cycle Starter*
- The applications of the software are:
  - It provides a simulation which is interesting to look at.
  - The software can be extended to add more complexities and a much trickier path.

1.3 Overview
The rest of the SRS contains a brief and a detailed description of the software including the requirements which need to be fulfilled to be able to run the software on the system and develop it further.
2 General Description

2.1 Product Perspective
Many other projects have been and are being made in box2d in the past and present respectively. The difference between all such projects comes when the idea behind the simulation is taken into account. The idea of this project i.e. the storyline of this project is what makes it different from the rest of the projects.

2.2 Product Functions
The product will basically give the user an opportunity to pull down a block which would trigger the chain reaction involved in the simulation, and the simulation will then go ahead as planned.

2.3 User Characteristics
The user should not pull the string to fast or too slow, or else the code will show some interesting collisions and some interesting random results.

2.4 General Constraints
Giving the user mouse-joint facilities can lead to some random results sometimes, so the user should pull the rope in a sensible manner.

2.5 Assumptions and Dependencies
The SRS document assumes that the system on which the software is being run has g++ compiler and Box2D is installed on the machine. Moreover OpenGL should also be installed on the machine. If not please refer to the Box2D and(or) OpenGL installation guide(s). Moreover, like any video game, any random input could lead to random results in the simulation.
3 The "Ram Katha" of the Project

3.1 What we planned to do

- Make a Rube Goldberg Machine which could make a lazy man stuck on a cycle with square shaped wheels, on a road made of Catenary curves move without making any effort

- The man pulls a string which initiates a mechanical chain reaction finally resulting in the above

- The elements include a stack of 2-dominos (one domino placed over the other), a spring-mass (2 bodies connected by a spring) system, a two-pulley system (Two pulleys sharing one common rope), Catenary curves, balloon, hourglass, Newton’s pendulum, a cut-rope system(a joint breaks when a collision occurs), funnel, bicycle, rotating turbines, etc

3.2 What we actually did

- We completed the main stuff we had proposed to do. We had maintained a local git repo individually on our computers as our parts were disjoint and then we finally integrated and maintained the new code.

- We had initially planned to make 2 pendulums and a few planks in the left part of the system but we replaced it with a spiral (Helix) and Newton’s pendulum. We did this so as to make the system more complex and to fill up the space that would have arisen in the earlier system.

- We adjusted the slopes of the two planks on which the spring mass system was to be kept so as to make the chain reaction possible without any problem.

- We increased the number of rotating turbine like structures to two to facilitate the ball to go into the spiral path.

3.3 Our work plan

- We divided the work into making objects and then integrated the work done by the three of us.

- Vishwesh made the spiral (Helix), catenary curves, bicycle, cut-rope system debugging, etc.
- Saurabh made the spring mass system, domino stack, balloon, funnel, hourglass, etc.

- Aviral made the Newton’s pendulum, 2-pulley system, hourglass, the cut-rope mechanism, mouse joint, etc.

- The integration of the project was done by Saurabh (Major), Aviral (Minor) and Vishwesh (Minor)

- All of us documented the individual parts of the code.

- Report and presentation were made by Aviral.

- Website was made by Vishwesh.

- Profiling was done by Saurabh.

3.4 Difficulties we faced

- Making a pulley system with two pulley joints sharing one common rope required the definition of a new joint (as per the Web search results) and we spent considerable amount of time thinking on how this had to be done. But we finally came up with a clever solution (of making an object common to both the pulley systems and setting its gravity to be zero) Resolved

- Making a spiral was also difficult as it involved a bit of math and a slightly longer code Resolved

- Making of the cut-rope system (i.e. ropes holding a piece of the Catenary curve break when it collides with a ball) turned out to be the most difficult part of the project as this kind of event handling didn’t work out well using any of the options tried. But finally we were able to implement it. Resolved

- Integration of code was also an issue where we had to think and give a lot of time.

- The output also depended on the time at which the user pulled the string down. Making the simulation certain to happen was a challenging task and we made it finally.
- This is not a challenge related to making the project. But one of our teammates lost his whole work due to rm -rf / which ran due to some mistake and we lost some of the changes in our code and this made us submit late by 1 day as we had to redo it once again.

3.5 Project Details

Figure 1: Screenshot of the working project

3.5.1 Components

Some of the major components in our simulation are:

1. **Helix** - This is a roller-coaster like path through which the ball goes. We made this using the equation of the spiral which we found on the web.

Figure 2: Helix
2. **Catenary curves** - These are special type of curves whose equation is given by \( y = \cosh(\frac{z}{a}) \). A square can smoothly move through a chain of these curves if the perimeter of the square and the length of one cycle of the catenary curve matches.

![Figure 3: Catenary curve Path](image)

3. **Bicycle** - We have simulated a bicycle by using a trapezium and two squares for wheels. The trapezium and squares have a revolute joint in between them.

![Figure 4: Newton’s Pendulum and Bicycle](image)

4. **Cut-Rope System** - A catenary curve (one cycle of it) is hanging from a rope above. Directly below it the road has a deep hole (a hole in the road exists). The simulation makes the part of the road (Catenary curve) fall and complete the road so that the bicycle is able to move on it without any interruption.

![Figure 5: Cut Rope System](image)
5. **Spring Mass System** - Two bodies are connected by a spring in between which can extend and compress (just like a spring, this is defined by a prismatic joint, although it doesn’t look like one).

![Figure 6: Spring Mass system](image)

6. **Two pulley system** - This is a system of two pulleys, one horizontal and the other vertical both of which share a common edge. This has been done by making a body attached to both the pulleys and setting its gravity to zero, so that it doesn’t fall down.

![Figure 7: Two-pulley system](image)

7. **Fans/Turbines** - These two systems of turbines facilitate the motion of the ball into the roller coaster like helix so that the whole simulation can occur without any difficulty.

![Figure 8: Fans](image)

8. **Hourglass** - Initially the upper part of the hourglass is filled with water and when the user pulls the plank out, it falls down into the empty container nearby starting the chain reaction.
3.5.2 Working

1. The man sitting on the cycle pulls the rope hanging in front of him downwards (this has been implemented by providing the user the chance to pull the rope down) which triggers a chain reaction.

2. The balloon gets freed and moves up, hits the rod connected by a hinge to the plank on which the dominos are kept. This rod hits the dominos which fall down one by one giving a nice visual effect.

3. Energy is transferred to the left end of this plank and then to the spring mass system, which hits a ball that starts moving up after that. There’s a plank above this ball too, which is ”active” state. The ball which goes into the spiral helix is prevented from moving by a small barrier. When that ball hits the plank, it gets deactivated (it eats up the ball and deactivates itself!!) which removes the barrier and frees the other ball.

4. This ball crosses over the fans/ turbines and enters the spiral helix path, finally leaving it and hitting the Newton’s pendulum, which then hits the cycle and brings it into motion.

5. Meanwhile, on the other side, the balloon hits the hanging catenary curve which is pointed, and this leads to bursting of the balloon. Due to the impulse created by this, the rope holding the Catenary curve breaks off. The curve falls down and completes the road.

3.6 Interesting facts

The idea behind the simulation, the storyline of the simulation itself is quite interesting and new in itself. Moreover, because we have provided the user a facility to initiate the chain reaction, final outputs might differ time to time. (For example, we tried, in fact played with it many times, and it worked perfectly almost two third number of times. Weird user inputs gave weird output results sometimes.)
4 Profiling Data

We have done some optimisations and dominos.cpp was significantly changed. In this optimised code of our project, debug_draw_t:: (i.e. it took the most time) whereas in the original code, b2World::DrawDebugData() was the highest time consumer. The images of the call graph are (We cut the image into multiple parts and put them):

Comparison

The top five time taking functions in base code given to us are:

1. b2World::DrawDebugData()
2. b2World::Solve(b2TimeStep const&)
3. b2World::DrawShape(b2Fixture*, b2Transform const&, b2Color const&)
4. debug_draw_t::DrawSolidCircle(b2Vec2 const&, float, b2Vec2 const&, b2Color const&)
5. b2CollidePolygons(b2Manifold*, b2PolygonShape const*, b2Transform const&, b2PolygonShape const*, b2Transform const&)

The top five time taking functions of our project are:

1. debug_draw_t::DrawCircle(b2Vec2 const&, float, b2Color const&)
2. b2Mul(b2Transform const&, b2Vec2 const&)
3. operator-(b2Vec2 const&, b2Vec2 const&)
4. b2Vec2::b2Vec2(float, float)
5. operator*(float, b2Vec2 const&)

DrawCircle(b2Vec2 const&, float, b2Color const) was highest in the queue of time consuming functions. DrawCircle was called many times because there are many balls (hourglass water droplets), points on the ChainShape (catenary curve) which were all circles and construction of each of these invoked this function.

As we have done significant changes, the most time taking functions have changed.
References


[2] FREYMAN ART. Square Wheeled Bicycle on a Catenary Curve https://www.youtube.com/watch?v=0BtZcmEkJF4


[9] Aamod CS 296 Project http://www.cse.iitb.ac.in/~aamod/cs296/project/html/classcs296_1_1dominos__t.html

[10] Iforce2D Box2D tutorials http://www.iforce2d.net/b2dtut/


[14] SHARE LATEX www.sharelatex.com