Report/images/iitb.jpeg

IIT Bombay

CS 251

Software Systems Lab

Eight Ball Pool

October '15

Group	Name	Roll No.
	Srajan	140050017
kuchbhi	Rishabh	140050019
	Anuj	140050024

1   Introduction

Our project deals with implementing the famous eight ball pool game using C++. We have used a physics simulation engine called Box2D for this purpose. It is a simplified version of the complete game including features like fouls and a complete two player game.

2   Motivation

Our hostel has a well maintained pool table with most of the required equipments to play. We often end up playing pool after having our lunch/dinner. We knew the game was all about physics, and as we were supposed to make a project based on a physics simulation engine, we thought why not implement pool?

3   Insight & Goals

3.1   Introduction to the game

Eight-ball is played with cue sticks and 16 balls: a cue ball, and 15 object balls consisting of seven striped balls, seven solid-colored balls and the black 8 ball. It is played on a table with six pockets.

After the balls are scattered with a break shot, the players are assigned either the group of solid balls or the stripes once a ball from a particular group is legally pocketed. The ultimate object of the game is to legally pocket the eight ball in a called pocket, which can only be done after all of the balls from a player's assigned group have been cleared from the table.

There are many other intricate rules of the game which will be discussed and implemented in our project.

3.2   Physics of the game

The whole game revolves around collisions and conservation of momentum. In detail, we look at friction, coefficient of restitution, spin and impulses.

In one dimensional motion we define the coefficient of restitution as the velocity of separation divided by the velocity of approach. We'll try to define this formally, and then look at some 2D aspects.
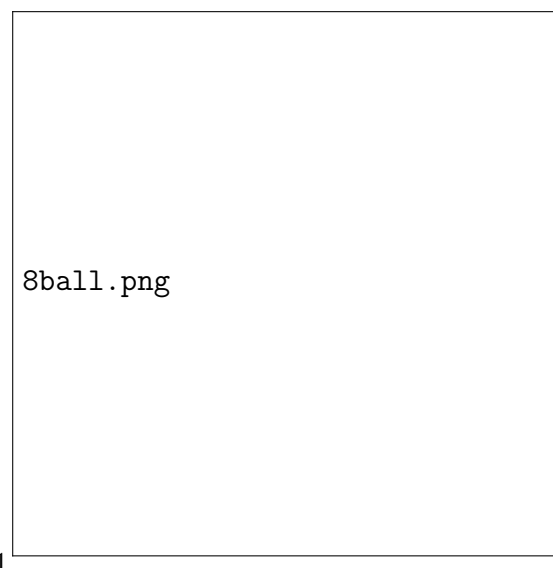
8ball.png

1

Figure 1: 1D Collision**?**

To solve collisions, we use the coefficient of restitution and the conservation of momentum.
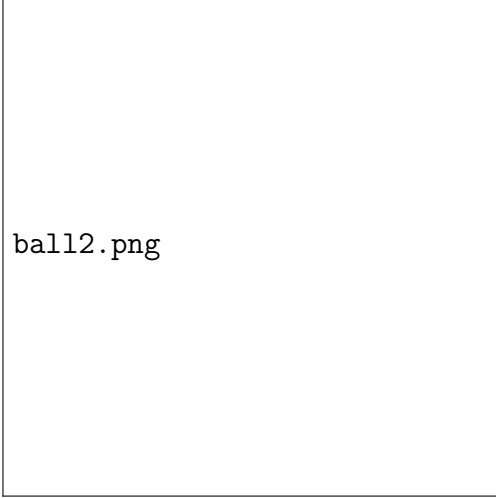
In **one dimensional** motion:

$$e = \frac{v_2' - v_1'}{v_1} \qquad (1)$$

$$mv_2' + mv_1' = mv_1 \qquad (2)$$

where,

e is the coefficient of restitution
$v_1 = $ is the initial velocity of cue ball
$v_1' = $ is the final velocity of cue ball
$v_2' = $ is the velocity attained by hit ball

Figure 2: 2D Collision**?**

In **two dimensional** motion:

$$e = \frac{v_{2B} - v_{2A}sin(\alpha + \theta)}{v_1}$$

$$mv_{2B}cos\theta = mv_{2A}sin\alpha$$

$$mv_{2B}sin\theta + mv_{2A}cos\alpha = mv_{1A}$$

Typically **?**,

$$e_{ball-ball} = 0.92 - 0.98$$

$$e_{ball-rail} = 0.65 - 0.85$$

Ofcourse, the balls are also always being acted upon by frictional forces offered by the carpet on the table. Which leads to a constant deceleration.

$$\frac{\mathrm{d}v}{\mathrm{d}t} = -\mu g \approx -2m/s^2$$

3.3 Our proposed goals

In our Box2D implementation of the game, we will cover all the basic aspects of the game including friction, multiple collisions and pocket detection. Some 3 dimensional aspects of the game like top spin and reverse spin are beyond the scope of this project. The following is a list of objectives that we aim to complete:

Core Goals

- Implement all physical interactions

- Implement all the standard rules

- Set up a two player game

Additional Goals

- Set up an AI to play with

- Make a tutorial/beginner mode
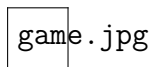
- Trickshot demonstrations

game.jpg

Figure 3: Stipulated Basic Design

3.3.1   Core Goals:

Our main aim of the project is get a basic game of pool up and running. Firstly, we have to set up our environment and all the variables correctly. The game will be played with a virtual cue stick which can be rotated around th cue ball with a mouse. It will also have a power setting, basically the speed with which it will hit the cue ball. We will also have to figure out how to detect that a ball has been potted or not, and subsequently remove it from the board if it has.

Secondly, we aim at developing a system which will detect all types of fouls which can be encountered while playing the game. After a foul has been committed, we also have to implement the penalty for the foul. This is either an auto-loss for the fouler, or a "ball in hand" for the other player. Thus, by completing a full fledged game, we intend to finish these as our main objectives.

3.3.2   Tutorial/Beginner mode:

In this setup, the player will get a "ball guiding system" which will show the trajectory of the cue ball and the ball being hit. This will help new players understand the mechanics of the game, and become more comfortable with the controls.

3.3.3   Trickshot demonstration:

Here, the computer will demonstrate "fun to watch" trickshots on the table. (like hitting 3 balls at once) The user will also get the option to try the trickshot himself.
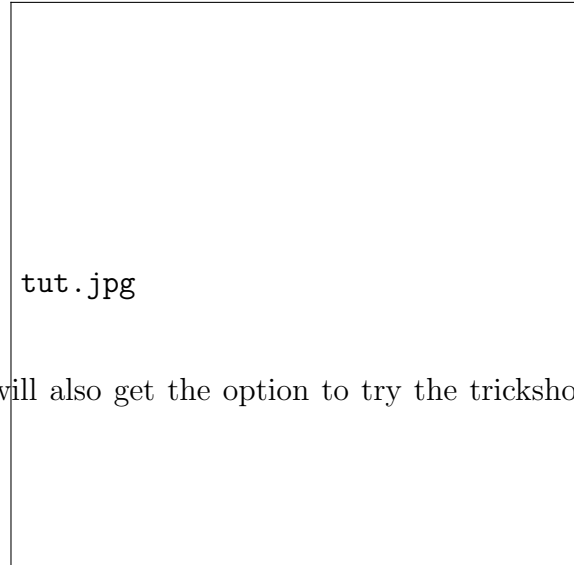
tut.jpg

Figure 4: Tutorial Mode

4   ?Our implementation

? To start out our project we began, by modifying the given base code to us. We simply, added circular balls and a rectangle to contain the balls initially. We then added circular pockets in the six places. This got us started. We then removed gravity, and set up environment variables like collisions and coefficients of restitution to properly simulate our game. This led us to make the very basic initial model. Most of this work was done by editing the file dominos.cpp .

? Next, we wanted to add the ability of the user to actually interact with the balls. So, we started working on the implementation of the pool stick and how the mouse will control this stick. We got the stick to rotate around the cue ball, and the user must pull the stick and let go to actually hit it. We were satisfied with how this mechanism was working. All this work was done by modifying the mouse_up , mouse_down and mouse_move functions.

? ? After this we started thinking about how to delete balls, when they got in contact with the pocket. We knew we couldn't do this as it happens in the real game, so we decided to erase the ball, if it came in contact with the pocket circles. We got this working using the b2ContactListener class. We then worked on differentiating the balls by colouring them up, and also looked at the aesthetic point of view of the game. After this the front end display was set up, showing the score and other details about the progress of the game.

Figure 5: Basic Design

We now needed to work about the backend of keeping the score and the foul system. The basic rules of the game were implemented, like the first touch rule, or the cue ball being pocketed foul. The opponent of the player who committed the foul gets two turns in succession. All this helped us shape the flow control of the whole game.

Unfortunately, we had run out of time, and also had to focus on our other project, the Web App. This resulted in us not being able to finish the other goals we had in mind. We have a basic GUI pool game for two players running, but we could not fulfill any of our additional goals.

5   Demo

You can find a video? demonstrating the project here.
Also, the project webpage? can be accessed at the following link

6 ?Profiling

The code profiling was done using gprof and the graphviz package. In our case, Profiling didn't help much to optimize the code as the most of time was taken inside the inbuilt Box2d functions.

The call graph when library is compiled using debug version and without using optimisation flags before making any changes in the code is There are a lot of func-

▫Report/images/call_graph.png

Figure 6: Debug Version

tions in the call graph. Most of the code written by us was optimized as the most time was taken by the inbuilt functions in box2D implementations.Almost all the functions in the call graph directly or indirectly calls the function b2Vec2::b2Vec2(float, float) The top 5 functions on the basis of total running time of the program used by this function are:-

1. operator*(float, b2Vec2 const&)

2. operator+(b2Vec2 const&, b2Vec2 const&)

3. debug_draw_t::DrawSolidCircle(b2Vec2 const&, float, b2Vec2 const&, b2Color const&)

4. b2Vec2::b2Vec2(float, float)

5. debug_draw_t::DrawString(int, int, char const*, ...)

Though the top two functions that are in green color in the call-graph are b2World::DrawShape(b2Fixture*, b2Transform const&, b2Color const&) and debug_draw_t::DrawSolidCircle(b2Vec2 const&, float, b2Vec2 const&, b2Color const&).

The call graph when library is compiled using Release version and using optimisation flags before making any changes in the code is

▫Report/images/call_graphRelease.png

Figure 7: Release Version

It is similar to what one obtains using the debug version, however since function is optimized the time spent decreases for each function and also the % spent in each function decreases. The top 5 functions on the basis of total running time of the program used by this function are:-

1. debug_draw_t::DrawSolidCircle(b2Vec2 const&, float, b2Vec2 const&, b2Color const&)

2. debug_draw_t::DrawString(int, int, char const*, ...)

3. b2World::Solve(b2TimeStep const&)

4. b2World::SolveTOI(b2TimeStep const&)

5. void b2DynamicTree::Query⟨b2BroadPhase⟩(b2BroadPhase*,
   b2AABB const&) const

Only the second function in the call-graph i.e cs251::callbacks_t::display_cb()
which contains a function cs251::base_sim_t::step(cs251::settings_t*) includes a func-
tion update_turn() implemented by us takes about 1% of total time.

The call graph of the Original Base code when library is compiled using release
version and using optimisation looks like

Report/images/call_graphOriginal.png

Figure 8: Debug Version

This is quite different in terms of the top functions as we changed the dominos.cpp
file completely and added a pool table, edges and multiple balls and the game
involves their interaction. The top 5 functions are:-

1. b2ContactSolver::SolveVelocityConstraints()

2. b2ContactManager::Collide()

3. debug_draw_t::DrawSolidCircle(b2Vec2 const&, float, b2Vec2 const&,
   b2Color const&)

4. b2WorldManifold::Initialize(b2Manifold const*, b2Transform const&,
   float,b2Transform const&, float)

5. b2StackAllocator::Allocate(int)

The top functions are different because our interface includes drawing a lot of
balls and updating them on the screen which involves debug_draw_t::DrawSolidCircle(
b2Vec2 const&, float, b2Vec2 const&, b2Color const&)and also keeping track of
the score and other variables for which the debug_draw_t::DrawString(int, int, char
const*, ...) is called multiple times.

7   Work distribution

We started discussion about what to implement as a Rube-Goldberg machine in mid-August. We decided not to implement the standard machine, rather make a game using the physics engine. We got confirmation from Prof. Sharat and started working on the project mid-September.

Though the work of each team member was closesly interlinked, we divided the work as follows:

- Srajan Garg (140050017):
    - Discussion of ideas and how they were to be implemented
    - Coding of the base strucutre of the game, the balls, table and pockets
    - Setting up environment variables like no gravity, restitution and friction
    - Coding up the contact listener to implement the pocketing of balls
    - Making of the report

- Rishabh Agarwal (140050019):
    - Discussion of how the code is going to be structured
    - Coding up aesthetic aspects of the game, like ball colours and the table
    - Set up the frontend interface and implemented the turn-based mechanism
    - Implemented the foul play and rules of the game by improving upon contactListener and Step function
    - Doxygen documentation of all the changes in the base code
    - Gprof Profiling and optimization of the code and adding its details to report
    - Making of the informative video on the project

- Anuj Mittal (140050024):
    - Discussion about the flow control of the program
    - Coded up the implpementation of cue stick
    - Integrated mouse support with stick, to 'rotate' and 'pull' it
    - Discussed implementation of fouls and improved on some functionality of game
    - Making of the webpage related to the project

8   Conclusion

All in all it was a great learning experience. We learned how to work in a team. One of the most important aspects of the project was to work, while using git. This helped us to understand version control and why it is essential for any project. More importantly we got a feel of how large organisations use repositories for storing codebases. We got to know how a distributed system works. This helped us to collaborate our work, and spend our time productively.

We also learnt how to use other poeples' projects and codebases. We learnt how a distributable version of a software is made. Box2D helped us understand te inner

implementations of how a large engine works. Learning about CMake and makefiles was also a good experience, and we realized how important they are for any large scale project.

9   Honour Code

I, Srajan Garg, pledge on our honour that I have not given or received any unauthorized assistance on this project or any previous task.

I, Rishabh Agarwal, pledge on our honour that I have not given or received any unauthorized assistance on this project or any previous task.

I, Anuj Mittal, pledge on our honour that I have not given or received any unauthorized assistance on this project or any previous task.