CS296 Group 09 - Final Project Report

Deepali Adlakha
`11D170020`
`dadlakha0111@gmail.com`

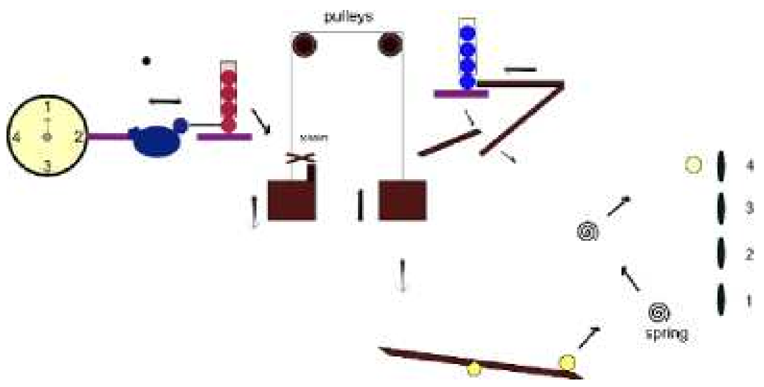
K. Kartik Pranav
`110050062`
`kartikpranav.k@gmail.com`

Vibhor Kanojia
`110050036`
`versatilevibhor@gmail.com`

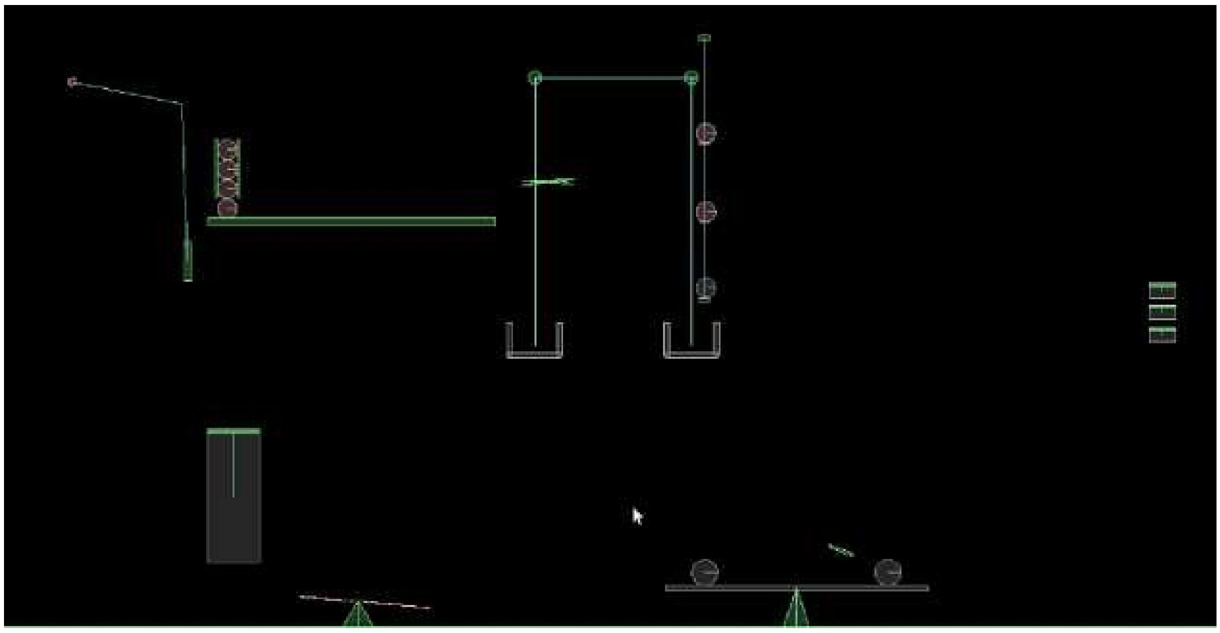
10th April, 2013

0 Introduction

The following is a report of the CS 296 project done by Group 09. We have designed a Rube Goldberg Machine,[3] and implemented our design using the Box2D API in C++.
Here's our initial design, made using Inkscape:



And here's our simulation in Box2D:

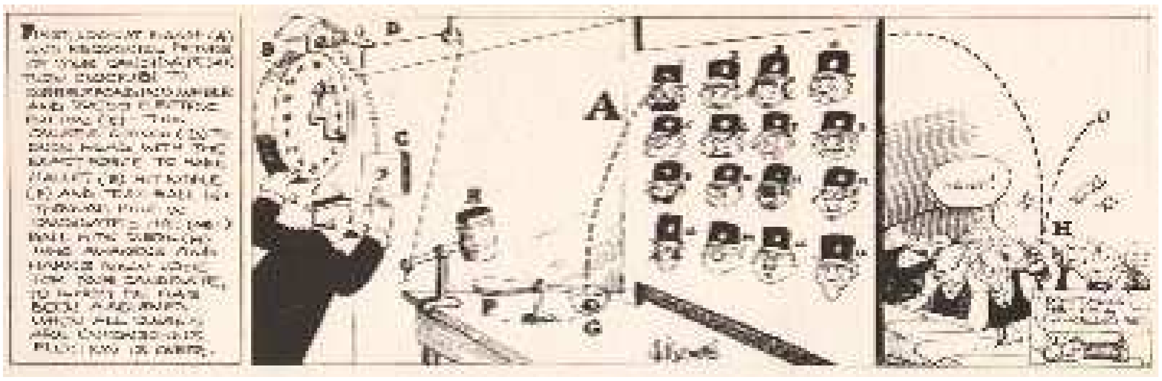


Note that we have changed the simulation a little, in place of the 'clock' we had initially, we now have a pendulum that changes its initial angle depending on the case.

1 The Idea: What's the big deal?

1.1 Inspiration

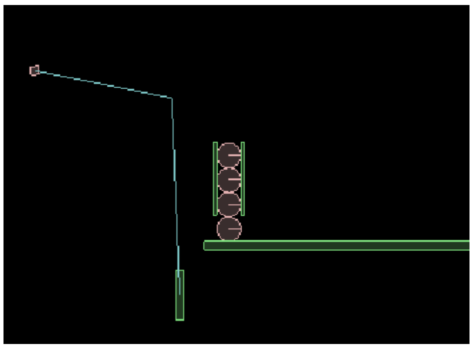
Representative of election day 2012 in the US, Rube Goldberg himself invented a machine that recorded votes from the general public:[1]



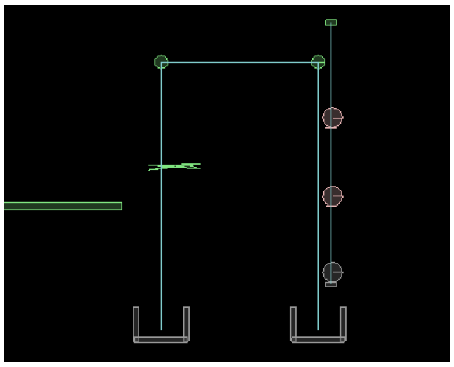
This was the essesntial idea behind our simulation. The aim - to get a ball to hit one of many different targets depending on initial conditions (in our case, the angle of the pendulum we started out with).

1.2 Our version

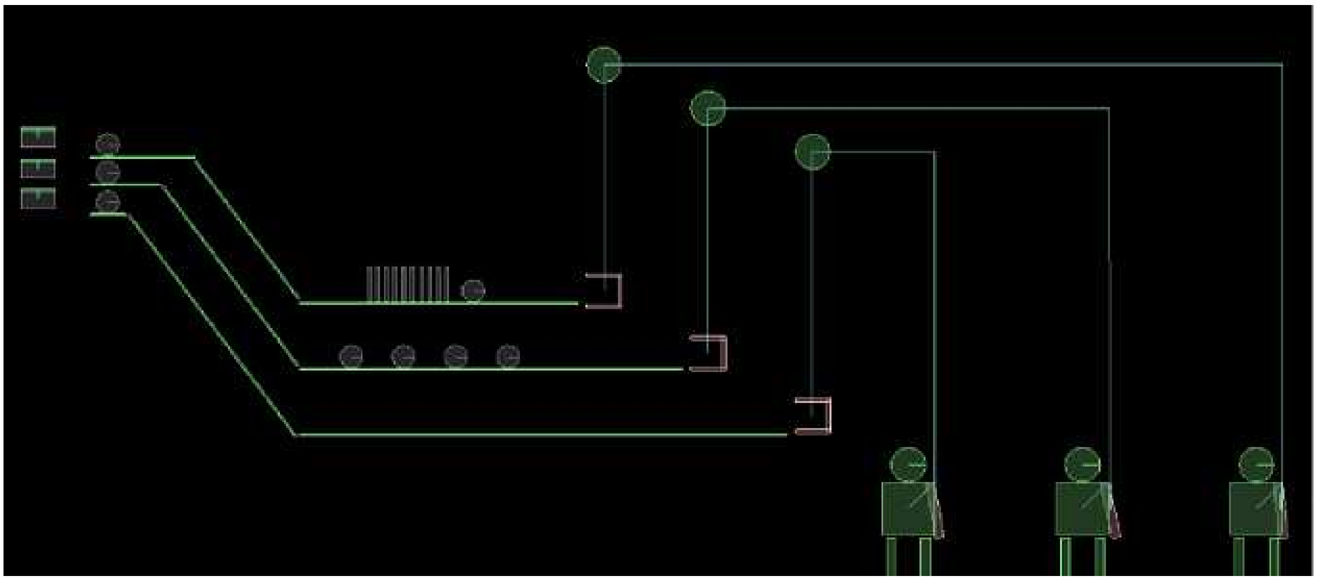
With this in mind, we used a pendulum to do something similar to what the clock was doing in the above machine. This is because implementing something that moved back and forth wouldn't be as appealing as a pendulum. It also makes it easier to measure time intervals between the falling of the balls into the lefthand- and righthand-side boxes.



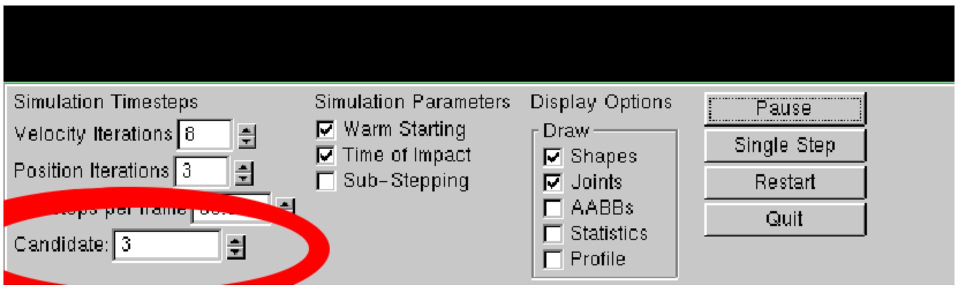
We have also implemented the equivalent of a pair of scissors, which cuts the string tying the boxes together, when the left box rises enough to hit it. For this, we had to create our own contact list of the bodies in Box2D, and our own scissors class, that takes as input the entity it must destroy (i.e. cut the string, in this case) when it comes into contact with it.



A major addition to our design, is that we have added an epilogue-style area, where there are three 'humans' (made of rods, blocks and spheres, at the end of the day) at the other end of the targets. Depending on which target is hit, a pulley system is activated, which eventually causes that person to raise his 'hand.' This makes up for a loss of elements that we weren't able to implement from our original design, and it alo conforms with the election-themed idea we were pursuing. And it looks cool! This part is shown below:



To select amongst these candidates, we have included an option at the bottom. You can select any one of the three candidates, and their respective hand will shoot up at the end of the simluation.



Simulation Timesteps
Velocity Iterations 8
Position Iterations 3
Candidate: 3

Simulation Parameters
☑ Warm Starting
☑ Time of Impact
☐ Sub-Stepping

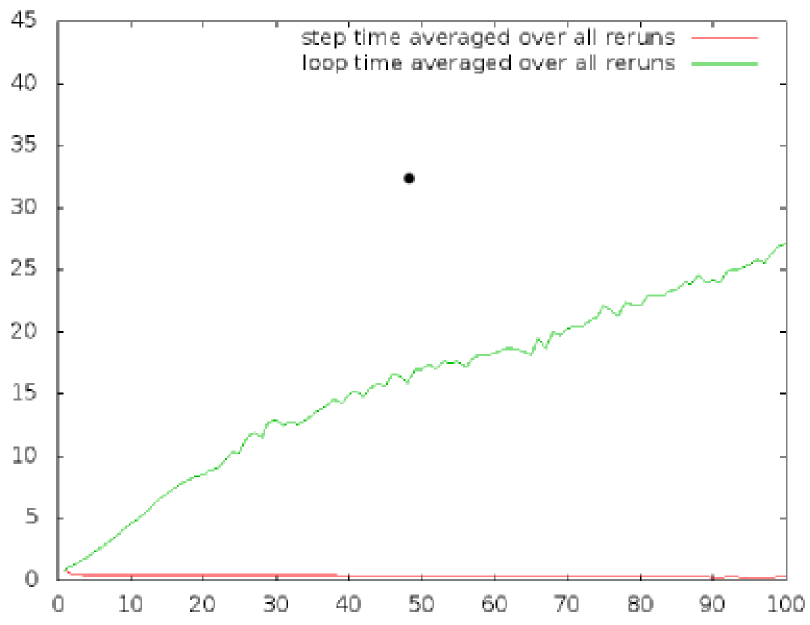
Display Options
Draw
☑ Shapes
☑ Joints
☐ AABBs
☐ Statistics
☐ Profile

2 Analysis of the Code

The Timing Part

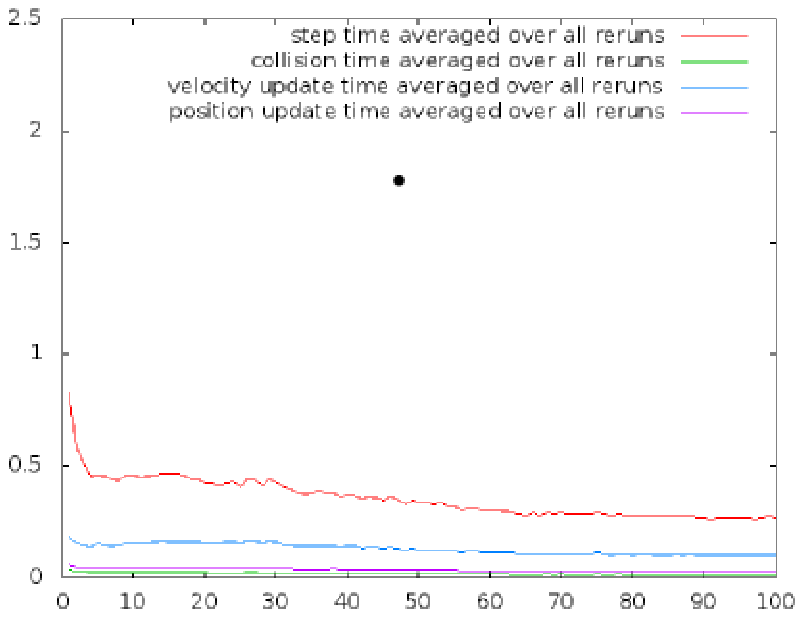
Similar to an earlier analysis of the base code using profiling and timing, we have plotted 6 graphs depicting the variation of various parameters, after running the executable 100 times for each iteration number (which also runs from 1 to 100). The results of this analysis are shown below:

1. Loop time vs. Step time



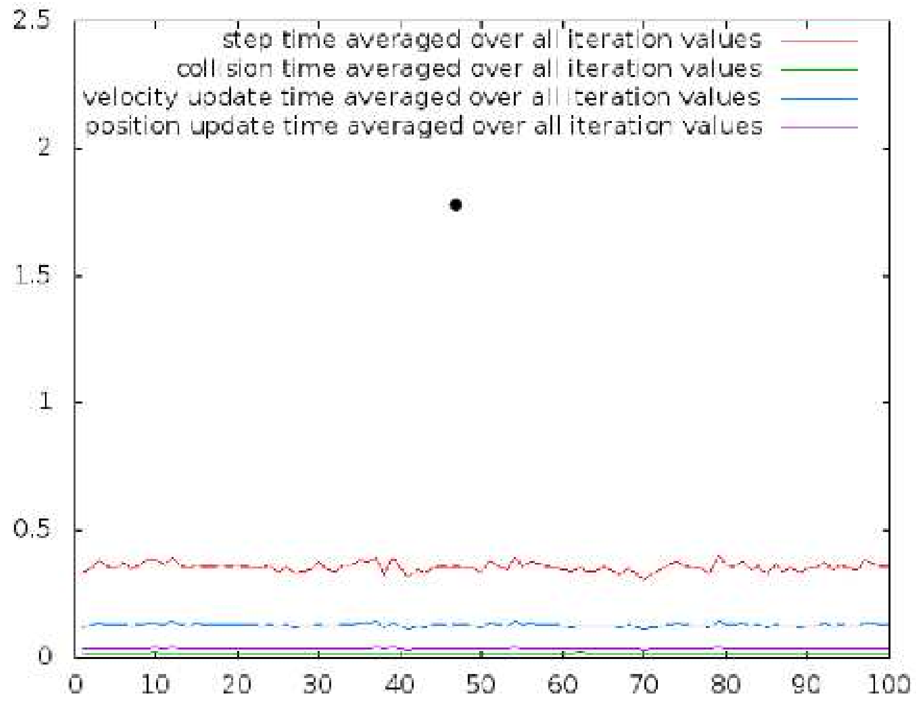
The loop time seems to increase almost linearly, or at a minutely slower rate as the step time increases. This is because the loop time is a cumulative sum of step times.

2. Step time and Update times



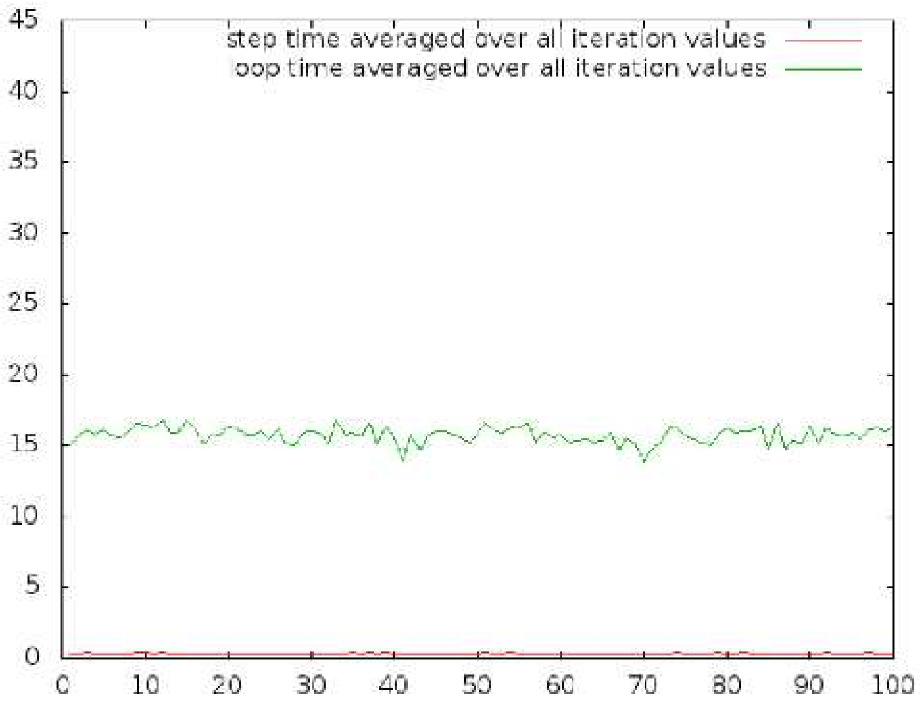
The step time must be greater than all the other times, and they all decrease as the iterations increase, due to scheduling and prioritizing of processes.

3. Step time and update times for Reruns



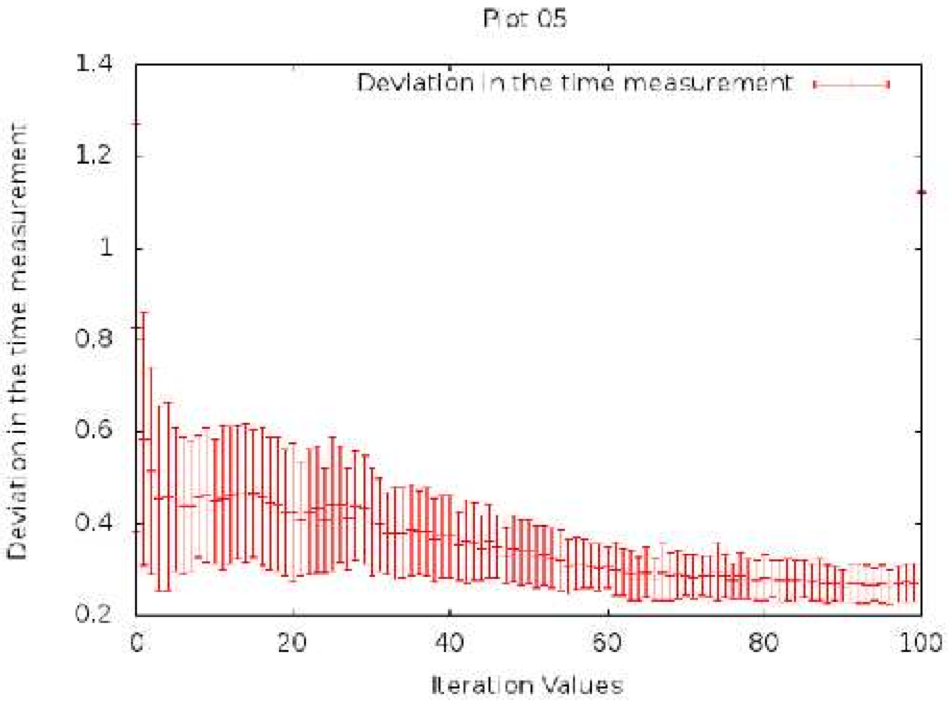
Here the number of iterations are constant, so the values obtained remain almost same throughout each rerun.

4. Loop time vs. Step time for Reruns



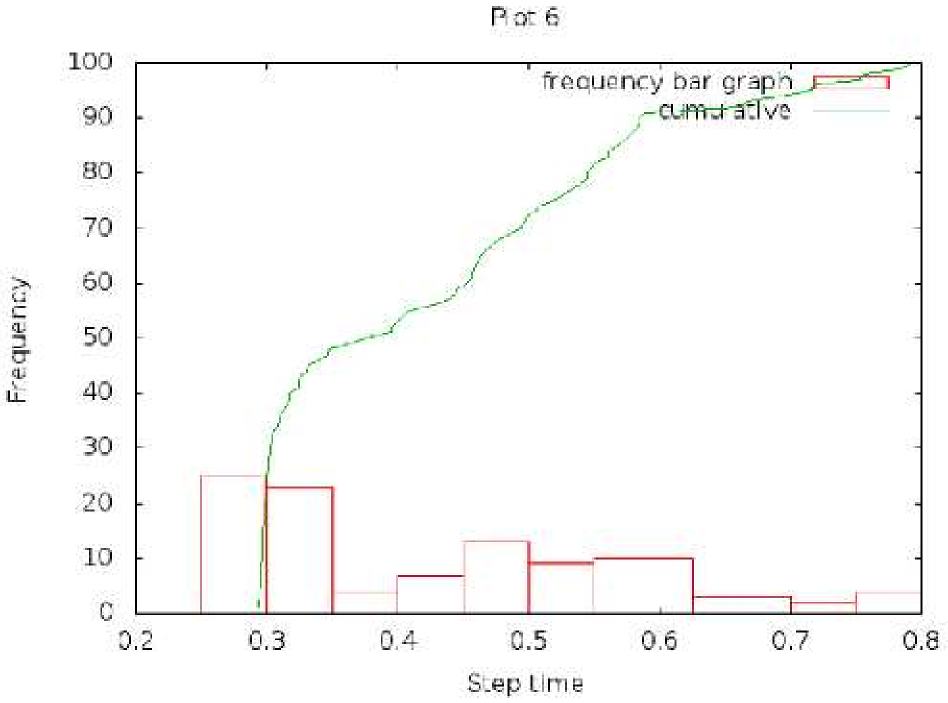
Again, the iteration number is kept constant, so not much change is seen across reruns. As usual, the loop time is greater than the step time.

5. Step time with standard deviation



The step time decreases over increasing iteration number, as inferred above. The standard deviation progressively decreases in the same way, because with higher iterations, there are more sample points in the space.

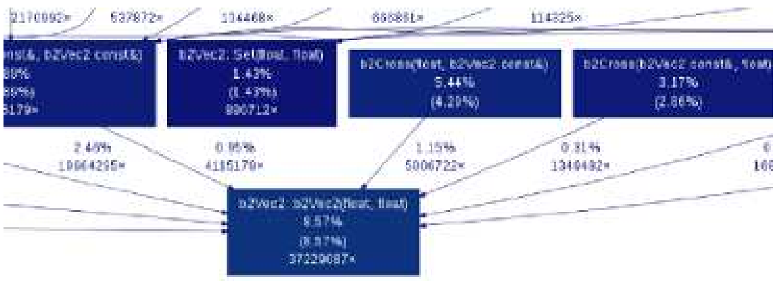
6. Cumulative Frequency Bar Graph



Here, we observe the maximum frequency at around 0.3, whereas it would have been at a higher value if the system were loaded. This shows that the process does indeed take more time in the case of a loaded system.

The Profiling and Call Graph

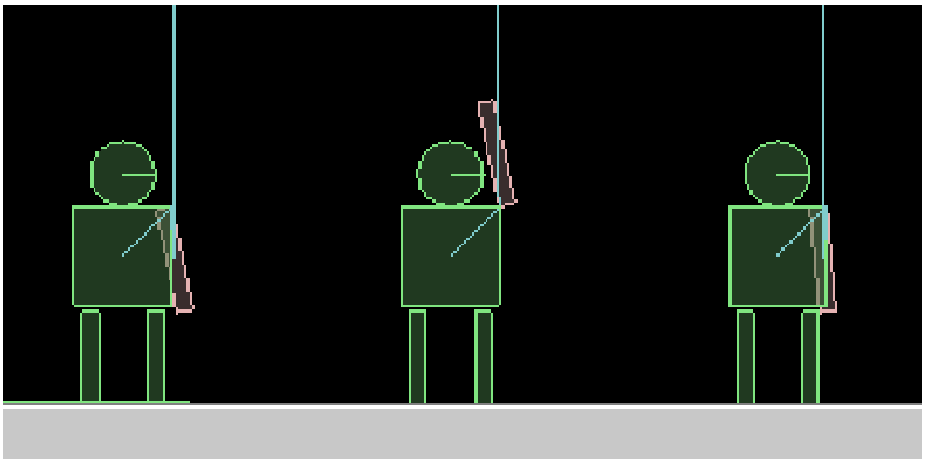
The call graph represents the time spent in every function, the calling of one function by another, their dependencies etc. We can find the functions that took a large amount of time, those that were called often, and such. This is made much easier when viewed visally, which can be done with the gprof2dot.py Python script .[2] Here is part of the call graph for the code without optimizations:



This shows that the function $b2Vec2$ takes a lot of function calls, and this can be improved upon, epsecially with function inlining.
When the call graph of the optimized (using $-O3$ flags) is drawn, it appears as a straight line of function names, indicating that there is no dependency of one function on another, i.e. complete function inlining has been done to optimize the code.

3 Conclusion

All in all, we have learnt a lot from this project. About the Box2D API (experience with such an API), analyzing and profiling code, and tidbits about C++ in general. We also learnt about source version control (using Git, in our case), and most importantly, how to create a distributable version of your source code for public usage. This is something that is in practice in the open-source world today.



References

[1] Screwball Comics. A Rube Goldberg Machine for Voting. `http://screwballcomics.blogspot.com/2012/11/a-rube-goldberg-machine-for-voting.html/`, 2012.

[2] Jose Fonsenca. Gprof2Dot. `https://code.google.com/p/jrfonseca/wiki/Gprof2Dot/`, 2007.

[3] Wikipedia. Rube Goldberg Machine. `en.wikipedia.org/Rube_Goldberg_Machine/`, 2013.