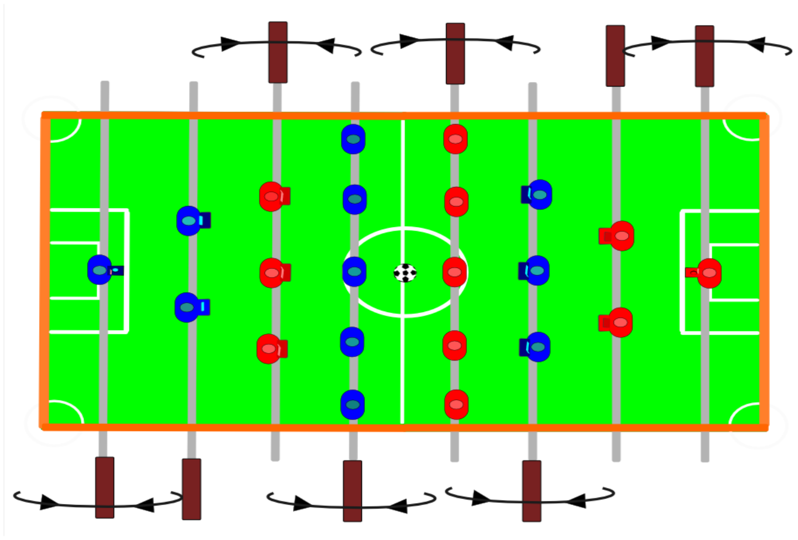
Foosball: CS296 Project, Group 27

Kiran Y - 120050049,
120050049@iitb.ac.in,
Mihir -12D020007
mihirk.1994@iitb.ac.in
Harinandan Teja-120050066
120050066@iitb.ac.in

April 10, 2014

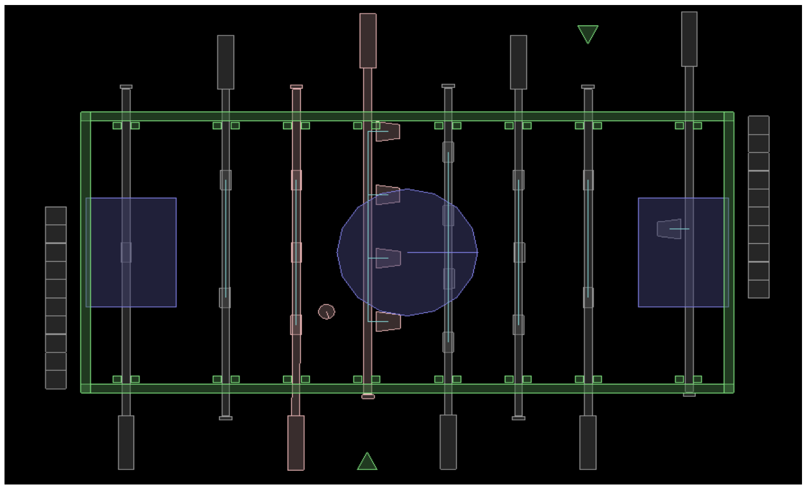
1 Introduction

This report explains in detail the project we created- a foosball table- in Box2d. We describe in brief the bodies,fixtures and constraints we used to create this simulation.
Here is the initial model of our simulation:



2 Design

2.1 Overall



1. We simulated the game of foosball using a bounding table, rods and rigid bodies representing players attached to each rod. Whenever the ball enters one of the goals situated at either end, a score variable is updated and this is shown by moving one of the relevant score blocks present on either side of the table. The first team to ten goals is the winner.

2. The simulation consists of a ball which moves around on the frictionless table colliding with everything it comes in contact with.

3. There are (transparent) rods at relevant locations on the table, constrained to move upto a certain limit in the upward or downward direction. The players attached to each rod can be moved towards either the left or right upto a certain limit. The user also has the option to turn the rod enough to lift all its players off the table.We used the `b2PrismaticJoint` to simulate motion of these players.

4. The simulation is heavily dependent on user input. We implemented this overloading the `keyboard()` function of the `base_sim_t` class of Box2D.

2.2 Challenges and changes from original plan

1. It was a challenge for us to implement the constraints required by the nature of the game, and we think that usage of prismatic joints and using the `filter` attributes of Box2D objects to cater to all aspects of motion of the players has made our design more efficient. We also feel that using static objects to constrain the motion of the rods (as explained later) was a simple yet effective feature.

2. We tried to use prismatic joints to restrict the motion of the rods as well. However, this made the motion considerably inefficient and we finally decided to not do so.

3. We changed players on the middle rod from five on the initial design to four. We also reduced the size of players. This was necessary to have enough space on the table for the ball to move around freely

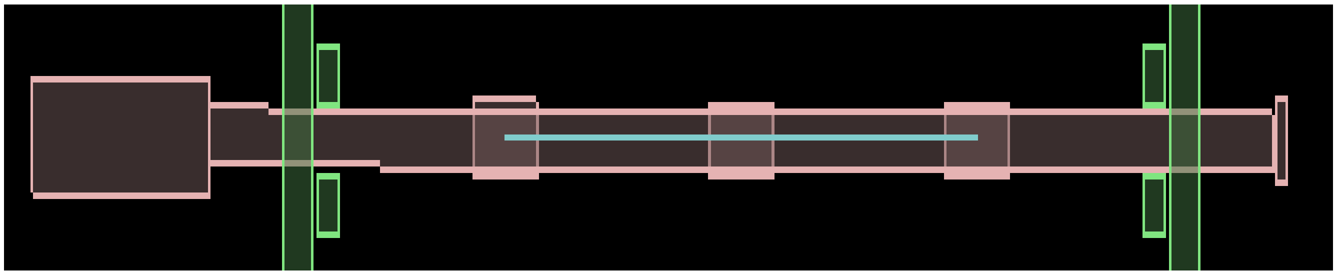
4. We also had to experiment with the density and restitution of the various bodies before deciding a final value to make the simulation seem as realistic as possible.

3 Parts of the simulation

3.1 Table

The entire game is played within the confines of the table, which serves to restrict the motion of the ball and rods. This was implemented using four box shapes.

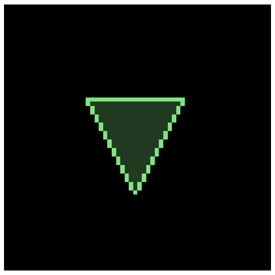
3.2 Rods



1. Rods are thin dynamic `Box` objects located at appropriate locations on the table. Two pairs of small static bodies named `constr` are located on either side of the rod at the point where it enters and leaves the table to constrain its motion in the y direction.

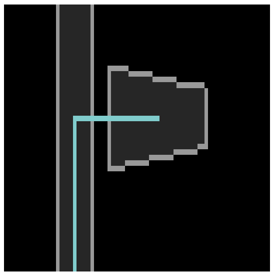
2. These are non-transparent however, and have `filter.maskBits` set to collide only with the `constr` bodies.

3. At any given point, only one rod per team is active. The user can change the active rod by giving a keyboard input.This is indicated by a pointer.



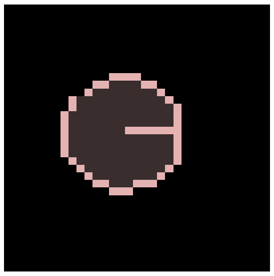
4. Each rod contains one anchor of a prismatic joint, the other end of which is connected to a Player body.

3.3 Player



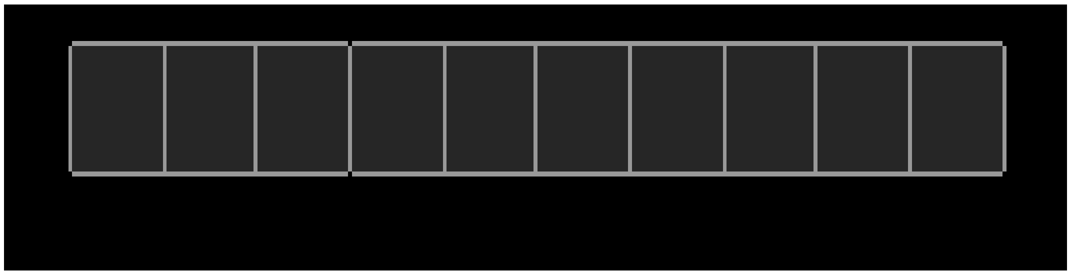
1. A player is a small,rigid and dynamic body which is attached to a rod via a prismatic joint. All players on the current active rod can be move right or left(upto a certain limit) by a keyboard input.Such a rotation has greater impact if it happens from a start position far away from the final position of the player.
2. The user can also lift the players of a rod off the table (meaning that the ball can pass underneath them) by pressing the right motion key while the players are at the maximum right position, and similarly for the left. This can be viewed as a change in the shape(fixture) of the players.
3. These are implemented by setting appropriate attributes on the prismatic joint and by setting the fixtures and filter bits of the players accordingly.

3.4 Ball



The ball is a `b2Circle` shaped `bullet` object with relatively low density. Throughout the simulation, the ball is being struck by/bouncing off players/table walls. Scores are updated when the ball enters either of the goals.

3.5 Score boxes

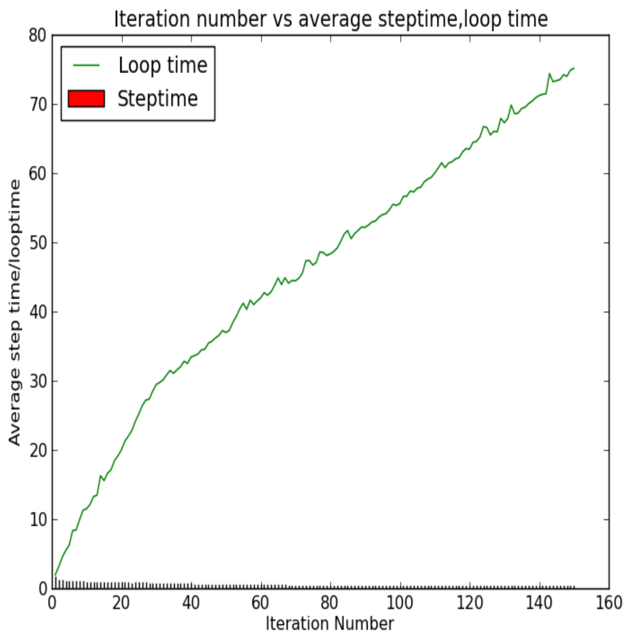


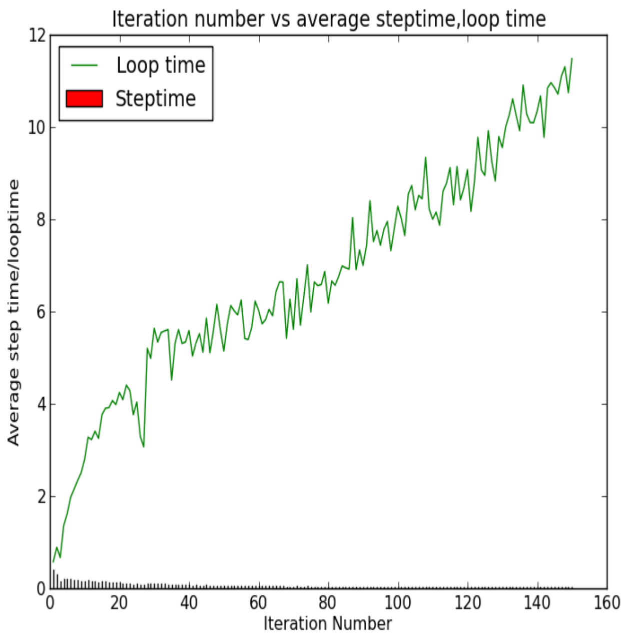
1. These are ten boxes located on either side of the table that represent the score.
2. A `dominos_t::check()` function is continuously called by the `base_sim_t::step()` function that checks whether a goal has been scored. If so, the appropriate box is pushed towards the appropriate side.
3. If the score of any team reaches ten or more, the game is restarted and the boxes are moved to their appropriate location.

4 Plots

4.1 Iteration number vs average steptime,loop time

Debug mode **Release** mode





1. In the above graphs we have average step time values and loop time values over all reruns vs iteration number.

2. Loop time increases with iteration number because as iteration number increases number of loops increases ,implies loop time increases.

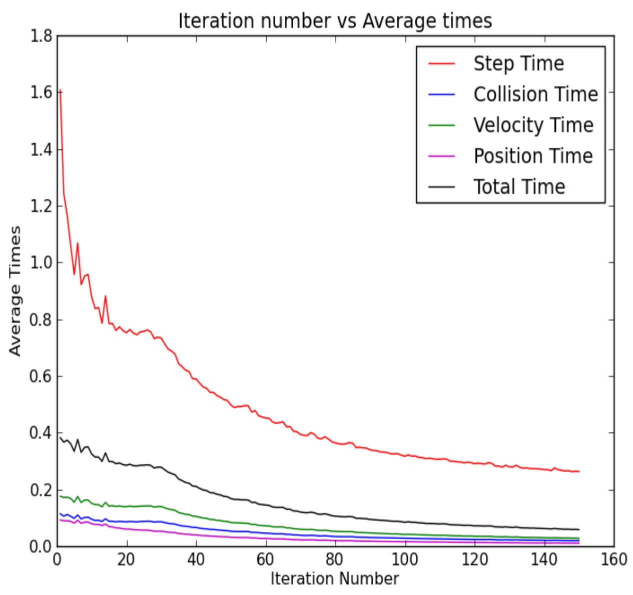
3. Average step time decreases with iteration number because simulation takes more time for every step at the initial stage than for each step at later stages.Hence the average step time is high at initial stages and and decreases as number of steps increases.

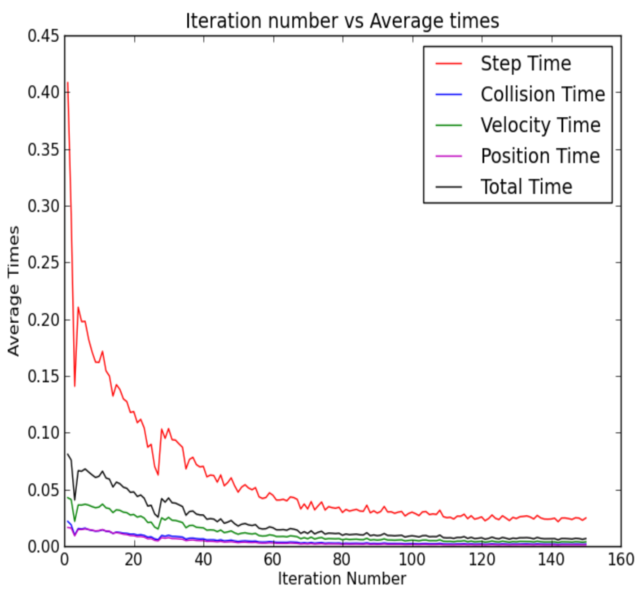
4. In **Debug** mode step time is greater than the step time in **Release** mode.

5. In **Debug** mode average time is greater than the average time in **Release** mode.

4.2 Iteration number vs All Average times

Debug mode **Release** mode





1. In the above graph we have average step time, average collision time, average velocity time, average position time , total time taken over all the reruns vs Iteration number.

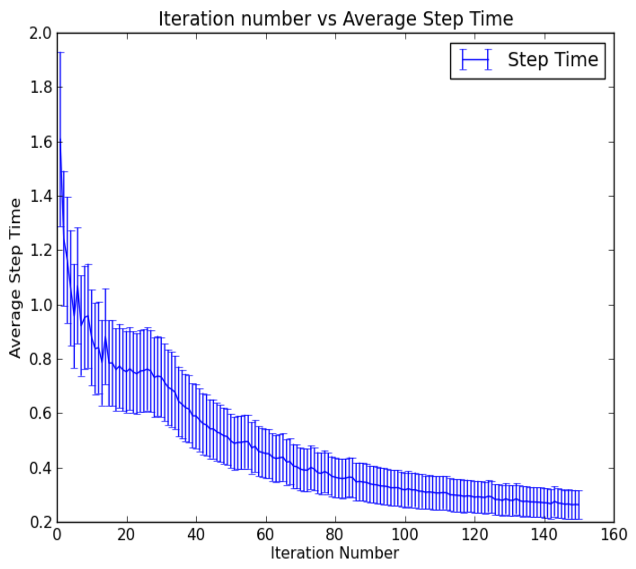
2. As discused above step time decreases with iteration number.

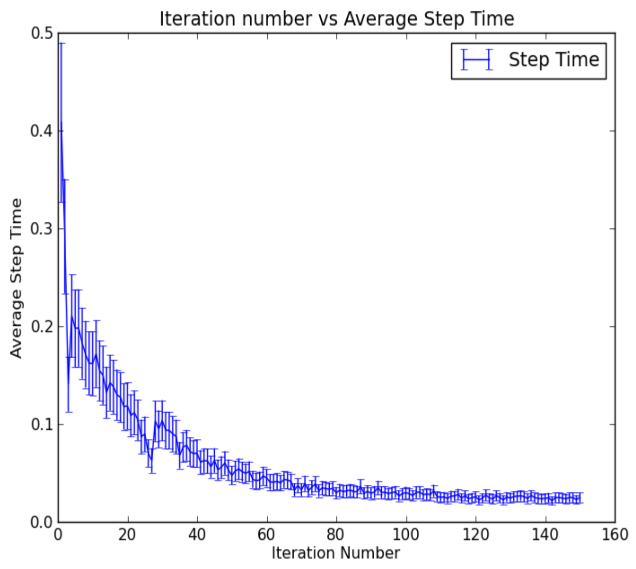
3. Vecocity time, Collision time, Position time and total time also decrease with iteration number but the rate at which thy decrease is lower than that of step time.

4. All the times (velocity, collision, position, total) in **Debug** mode are greater than the times in **Release** mode.

4.3 Iteration number vs Average Step Time with error bars

Debug mode **Release** mode

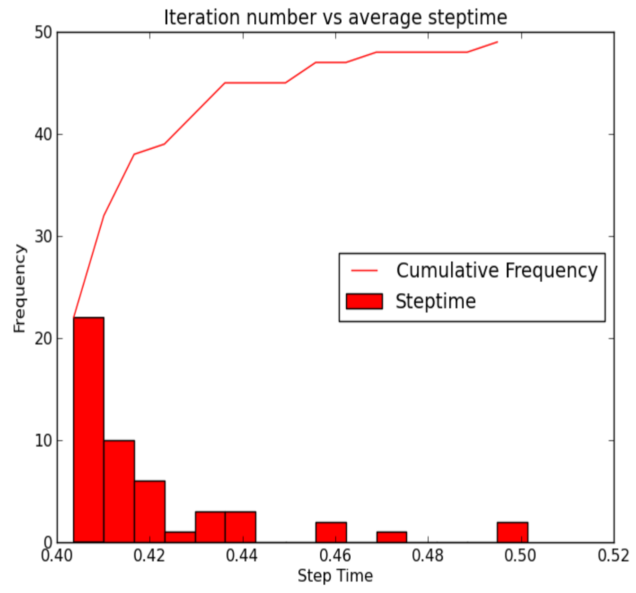


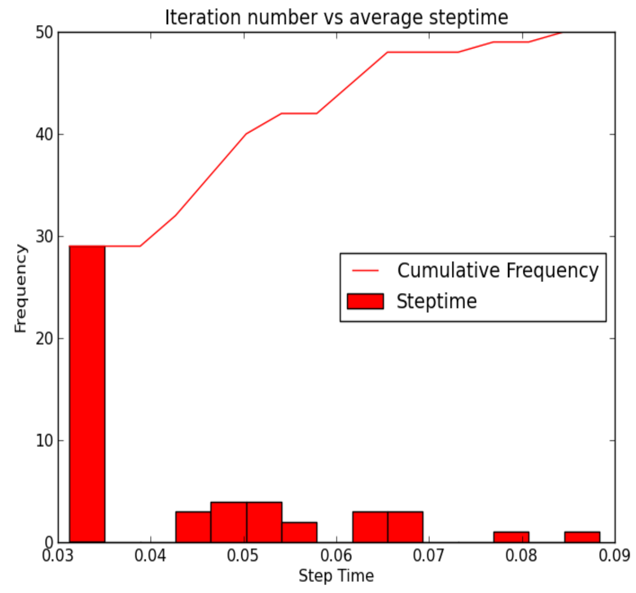


In the above graph we have average step times in **Debug** mode and **Release** mode over all reruns vs iteration number along with the error bars .

4.4 Iteration number vs average steptime

Debug mode **Release** mode





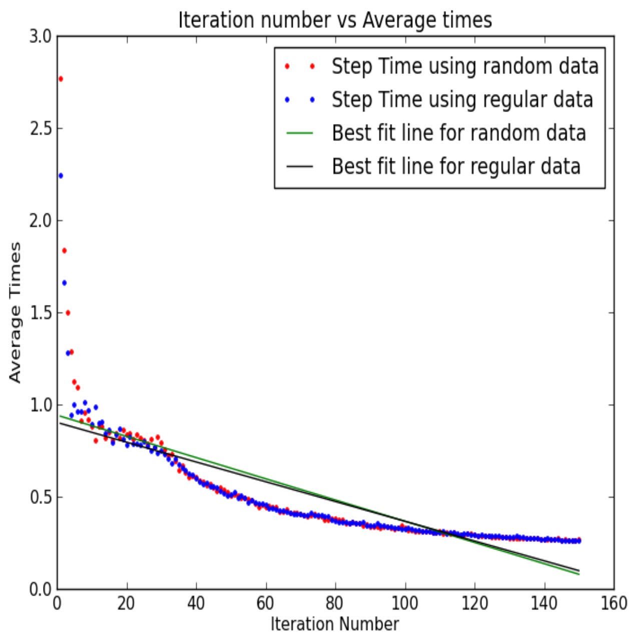
1. The above graph is histogram for step time (both **Debug** mode and **Release** mode) for the iteration number 66 along with the cumulative frequency.

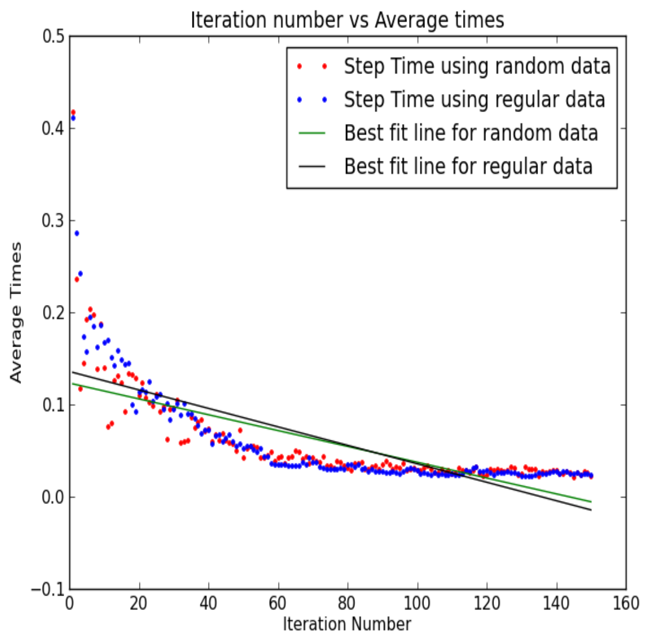
2. We can see that the most of the step times in the **Debug** mode are in the range 0.40 - 0.42, where as in the **Release** mode most of the step times are in the range 0.03-0.06.Average step time (iteration number 66) for **Debug** mode lie in

the range 0.40-0.42 and for **Release** in the range 0.03-0.06.

4.5 Iteration number vs Average times and best fit line

Debug mode **Release** mode





1. In the above graph we have Average Step time in random data generated vs iteration number along with the Average step time in regular data vs iteration number.

2. As discussed earlier step time decreases with iteration number for both **Debug** mode and **Release** mode.

3. The average step times calculated using regular data are more accurate than the average step times generated using because in random data we take just 15 value for calculating average step time where as in regullar data we take 50 values for calculating average step time.

5 Profiling

5.1 Gprof

1. In Gprof data about all the times taken by different classes, function calls etc. is stored while the .o (main.o callbacks.o,dominos.o ,cs296_base.0,render.o in this case) files are created and while these .o files are joined to form an executable (cs296_exe_27 in this case).

2. Hence gprof stores all the information during the compilation of makefile itself.

3. For using gprof we include -pg in all the commands that produce .o files and the commands that produce the final executable file.

5.2 Perf

1. In Perf data about all the times taken by diffenrent classes, function calls etc. is stored when the executable is executed (cs296_exe_27 in this case)

2. Perf gives more accurate time taken by the function calls etc.

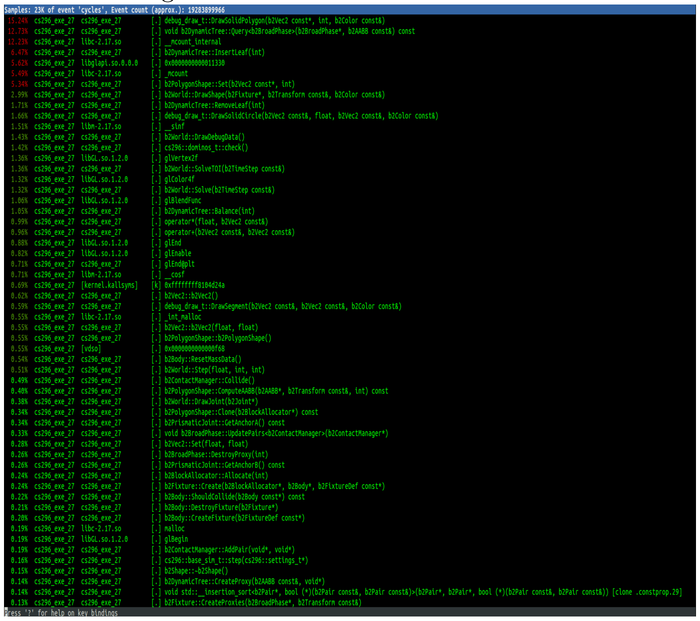
3. For using perf we just need to run two commands

   - perf record /path/to/the/exectuable – stores the data

   - perf report – gives the entire data in the terminal[1]

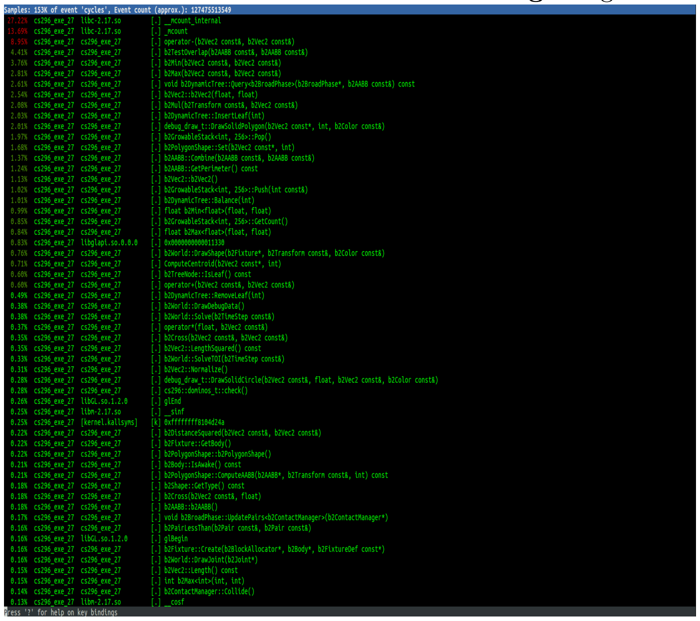
5.3 Profiling using perf

Release using PERF



```
Samples: 23K of event 'cycles', Event count (approx.): 19283899966
 15.24%  cs296_exe_27  cs296_exe_27        [.] debug_draw_t::DrawSolidPolygon(b2Vec2 const*, int, b2Color const&)
 12.73%  cs296_exe_27  cs296_exe_27        [.] void b2DynamicTree::Query<b2BroadPhase>(b2BroadPhase*, b2AABB const&) const
 12.23%  cs296_exe_27  libc-2.17.so        [.] __mcount_internal
  6.47%  cs296_exe_27  cs296_exe_27        [.] b2DynamicTree::InsertLeaf(int)
  5.62%  cs296_exe_27  libglapi.so.0.0.0   [.] 0x0000000000011330
  5.49%  cs296_exe_27  libc-2.17.so        [.] _mcount
  5.34%  cs296_exe_27  cs296_exe_27        [.] b2PolygonShape::Set(b2Vec2 const*, int)
  2.99%  cs296_exe_27  cs296_exe_27        [.] b2World::DrawShape(b2Fixture*, b2Transform const&, b2Color const&)
  1.71%  cs296_exe_27  cs296_exe_27        [.] b2DynamicTree::RemoveLeaf(int)
  1.66%  cs296_exe_27  cs296_exe_27        [.] debug_draw_t::DrawSolidCircle(b2Vec2 const&, float, b2Vec2 const&, b2Color const&)
  1.51%  cs296_exe_27  libm-2.17.so        [.] __sinf
  1.43%  cs296_exe_27  cs296_exe_27        [.] b2World::DrawDebugData()
  1.42%  cs296_exe_27  cs296_exe_27        [.] cs296::dominos_t::check()
  1.36%  cs296_exe_27  libGL.so.1.2.0      [.] glVertex2f
  1.36%  cs296_exe_27  cs296_exe_27        [.] b2World::SolveTOI(b2TimeStep const&)
  1.32%  cs296_exe_27  libGL.so.1.2.0      [.] glColor4f
  1.32%  cs296_exe_27  cs296_exe_27        [.] b2World::Solve(b2TimeStep const&)
  1.06%  cs296_exe_27  libGL.so.1.2.0      [.] glBlendFunc
  1.05%  cs296_exe_27  cs296_exe_27        [.] b2DynamicTree::Balance(int)
  0.99%  cs296_exe_27  cs296_exe_27        [.] operator*(float, b2Vec2 const&)
  0.96%  cs296_exe_27  cs296_exe_27        [.] operator+(b2Vec2 const&, b2Vec2 const&)
  0.88%  cs296_exe_27  libGL.so.1.2.0      [.] glEnd
  0.82%  cs296_exe_27  libGL.so.1.2.0      [.] glEnable
  0.71%  cs296_exe_27  cs296_exe_27        [.] glEnd@plt
  0.71%  cs296_exe_27  libm-2.17.so        [.] __cosf
  0.69%  cs296_exe_27  [kernel.kallsyms]   [k] 0xffffffff8104d24a
  0.62%  cs296_exe_27  cs296_exe_27        [.] b2Vec2::b2Vec2()
  0.59%  cs296_exe_27  cs296_exe_27        [.] debug_draw_t::DrawSegment(b2Vec2 const&, b2Vec2 const&, b2Color const&)
  0.55%  cs296_exe_27  libc-2.17.so        [.] _int_malloc
  0.55%  cs296_exe_27  cs296_exe_27        [.] b2Vec2::b2Vec2(float, float)
  0.55%  cs296_exe_27  cs296_exe_27        [.] b2PolygonShape::b2PolygonShape()
  0.55%  cs296_exe_27  [vdso]              [.] 0x0000000000000f68
  0.54%  cs296_exe_27  cs296_exe_27        [.] b2Body::ResetMassData()
  0.51%  cs296_exe_27  cs296_exe_27        [.] b2World::Step(float, int, int)
  0.49%  cs296_exe_27  cs296_exe_27        [.] b2ContactManager::Collide()
  0.40%  cs296_exe_27  cs296_exe_27        [.] b2PolygonShape::ComputeAABB(b2AABB*, b2Transform const&, int) const
  0.38%  cs296_exe_27  cs296_exe_27        [.] b2World::DrawJoint(b2Joint*)
  0.34%  cs296_exe_27  cs296_exe_27        [.] b2PolygonShape::Clone(b2BlockAllocator*) const
  0.34%  cs296_exe_27  cs296_exe_27        [.] b2PrismaticJoint::GetAnchorA() const
  0.33%  cs296_exe_27  cs296_exe_27        [.] void b2BroadPhase::UpdatePairs<b2ContactManager>(b2ContactManager*)
  0.28%  cs296_exe_27  cs296_exe_27        [.] b2Vec2::Set(float, float)
  0.26%  cs296_exe_27  cs296_exe_27        [.] b2BroadPhase::DestroyProxy(int)
  0.26%  cs296_exe_27  cs296_exe_27        [.] b2PrismaticJoint::GetAnchorB() const
  0.24%  cs296_exe_27  cs296_exe_27        [.] b2BlockAllocator::Allocate(int)
  0.24%  cs296_exe_27  cs296_exe_27        [.] b2Fixture::Create(b2BlockAllocator*, b2Body*, b2FixtureDef const*)
  0.22%  cs296_exe_27  cs296_exe_27        [.] b2Body::ShouldCollide(b2Body const*) const
  0.21%  cs296_exe_27  cs296_exe_27        [.] b2Body::DestroyFixture(b2Fixture*)
  0.20%  cs296_exe_27  cs296_exe_27        [.] b2Body::CreateFixture(b2FixtureDef const*)
  0.19%  cs296_exe_27  libc-2.17.so        [.] malloc
  0.19%  cs296_exe_27  libGL.so.1.2.0      [.] glBegin
  0.19%  cs296_exe_27  cs296_exe_27        [.] b2ContactManager::AddPair(void*, void*)
  0.16%  cs296_exe_27  cs296_exe_27        [.] cs296::base_sim_t::step(cs296::settings_t*)
  0.15%  cs296_exe_27  cs296_exe_27        [.] b2Shape::~b2Shape()
  0.14%  cs296_exe_27  cs296_exe_27        [.] b2DynamicTree::CreateProxy(b2AABB const&, void*)
  0.14%  cs296_exe_27  cs296_exe_27        [.] void std::__insertion_sort<b2Pair*, bool (*)(b2Pair const&, b2Pair const&)>(b2Pair*, b2Pair*, bool (*)(b2Pair const&, b2Pair const&)) [clone .constprop.29]
  0.13%  cs296_exe_27  cs296_exe_27        [.] b2Fixture::CreateProxies(b2BroadPhase*, b2Transform const&)
Press '?' for help on key bindings
```

Debug using PERF



```
Samples: 153K of event 'cycles', Event count (approx.): 127475513549
 27.22%  cs296_exe_27  libc-2.17.so        [.] __mcount_internal
 13.69%  cs296_exe_27  libc-2.17.so        [.] _mcount
  8.95%  cs296_exe_27  cs296_exe_27        [.] operator-(b2Vec2 const&, b2Vec2 const&)
  4.41%  cs296_exe_27  cs296_exe_27        [.] b2TestOverlap(b2AABB const&, b2AABB const&)
  3.76%  cs296_exe_27  cs296_exe_27        [.] b2Min(b2Vec2 const&, b2Vec2 const&)
  2.81%  cs296_exe_27  cs296_exe_27        [.] b2Max(b2Vec2 const&, b2Vec2 const&)
  2.61%  cs296_exe_27  cs296_exe_27        [.] void b2DynamicTree::Query<b2BroadPhase>(b2BroadPhase*, b2AABB const&) const
  2.54%  cs296_exe_27  cs296_exe_27        [.] b2Vec2::b2Vec2(float, float)
  2.08%  cs296_exe_27  cs296_exe_27        [.] b2Mul(b2Transform const&, b2Vec2 const&)
  2.03%  cs296_exe_27  cs296_exe_27        [.] b2DynamicTree::InsertLeaf(int)
  2.01%  cs296_exe_27  cs296_exe_27        [.] debug_draw_t::DrawSolidPolygon(b2Vec2 const*, int, b2Color const&)
  1.97%  cs296_exe_27  cs296_exe_27        [.] b2GrowableStack<int, 256>::Pop()
  1.68%  cs296_exe_27  cs296_exe_27        [.] b2PolygonShape::Set(b2Vec2 const*, int)
  1.37%  cs296_exe_27  cs296_exe_27        [.] b2AABB::Combine(b2AABB const&, b2AABB const&)
  1.24%  cs296_exe_27  cs296_exe_27        [.] b2AABB::GetPerimeter() const
  1.13%  cs296_exe_27  cs296_exe_27        [.] b2Vec2::b2Vec2()
  1.02%  cs296_exe_27  cs296_exe_27        [.] b2GrowableStack<int, 256>::Push(int const&)
  1.01%  cs296_exe_27  cs296_exe_27        [.] b2DynamicTree::Balance(int)
  0.99%  cs296_exe_27  cs296_exe_27        [.] float b2Min<float>(float, float)
  0.85%  cs296_exe_27  cs296_exe_27        [.] b2GrowableStack<int, 256>::GetCount()
  0.84%  cs296_exe_27  cs296_exe_27        [.] float b2Max<float>(float, float)
  0.83%  cs296_exe_27  libglapi.so.0.0.0   [.] 0x0000000000011330
  0.76%  cs296_exe_27  cs296_exe_27        [.] b2World::DrawShape(b2Fixture*, b2Transform const&, b2Color const&)
  0.71%  cs296_exe_27  cs296_exe_27        [.] ComputeCentroid(b2Vec2 const*, int)
  0.60%  cs296_exe_27  cs296_exe_27        [.] b2TreeNode::IsLeaf() const
  0.60%  cs296_exe_27  cs296_exe_27        [.] operator+(b2Vec2 const&, b2Vec2 const&)
  0.49%  cs296_exe_27  cs296_exe_27        [.] b2DynamicTree::RemoveLeaf(int)
  0.38%  cs296_exe_27  cs296_exe_27        [.] b2World::DrawDebugData()
  0.38%  cs296_exe_27  cs296_exe_27        [.] b2World::Solve(b2TimeStep const&)
  0.37%  cs296_exe_27  cs296_exe_27        [.] operator*(float, b2Vec2 const&)
  0.35%  cs296_exe_27  cs296_exe_27        [.] b2Cross(b2Vec2 const&, b2Vec2 const&)
  0.35%  cs296_exe_27  cs296_exe_27        [.] b2Vec2::LengthSquared() const
  0.33%  cs296_exe_27  cs296_exe_27        [.] b2World::SolveTOI(b2TimeStep const&)
  0.31%  cs296_exe_27  cs296_exe_27        [.] b2Vec2::Normalize()
  0.28%  cs296_exe_27  cs296_exe_27        [.] debug_draw_t::DrawSolidCircle(b2Vec2 const&, float, b2Vec2 const&, b2Color const&)
  0.28%  cs296_exe_27  cs296_exe_27        [.] cs296::dominos_t::check()
  0.26%  cs296_exe_27  libGL.so.1.2.0      [.] glEnd
  0.25%  cs296_exe_27  libm-2.17.so        [.] __sinf
  0.25%  cs296_exe_27  [kernel.kallsyms]   [k] 0xffffffff8104d24a
  0.22%  cs296_exe_27  cs296_exe_27        [.] b2DistanceSquared(b2Vec2 const&, b2Vec2 const&)
  0.22%  cs296_exe_27  cs296_exe_27        [.] b2Fixture::GetBody()
  0.22%  cs296_exe_27  cs296_exe_27        [.] b2PolygonShape::b2PolygonShape()
  0.21%  cs296_exe_27  cs296_exe_27        [.] b2Body::IsAwake() const
  0.21%  cs296_exe_27  cs296_exe_27        [.] b2PolygonShape::ComputeAABB(b2AABB*, b2Transform const&, int) const
  0.18%  cs296_exe_27  cs296_exe_27        [.] b2Shape::GetType() const
  0.18%  cs296_exe_27  cs296_exe_27        [.] b2Cross(b2Vec2 const&, float)
  0.18%  cs296_exe_27  cs296_exe_27        [.] b2AABB::b2AABB()
  0.17%  cs296_exe_27  cs296_exe_27        [.] void b2BroadPhase::UpdatePairs<b2ContactManager>(b2ContactManager*)
  0.16%  cs296_exe_27  cs296_exe_27        [.] b2PairLessThan(b2Pair const&, b2Pair const&)
  0.16%  cs296_exe_27  libGL.so.1.2.0      [.] glBegin
  0.16%  cs296_exe_27  cs296_exe_27        [.] b2Fixture::Create(b2BlockAllocator*, b2Body*, b2FixtureDef const*)
  0.16%  cs296_exe_27  cs296_exe_27        [.] b2World::DrawJoint(b2Joint*)
  0.15%  cs296_exe_27  cs296_exe_27        [.] b2Vec2::Length() const
  0.15%  cs296_exe_27  cs296_exe_27        [.] int b2Max<int>(int, int)
  0.14%  cs296_exe_27  cs296_exe_27        [.] b2ContactManager::Collide()
  0.13%  cs296_exe_27  libm-2.17.so        [.] __cosf
Press '?' for help on key bindings
```

1. For profiling we are running the code for 1000000 interations so that all those functions which are called very few times are also listed.

2. Running the program for very few iterations may not give a clear stats of those functions that take long time or that are called for large number of times during execution.

5.4 Optimized and Non-Optimized difference

1. From the **Release** profile it is clear that the function call DrawSolidPolygon is the most expensive function.

2. The **Debug** profile indicates that the operators -, operators +, b2Min, b2max etc are called for large number of times during the execution as they these functions are called in almost every function present in the code. Where as in the **Release** profile i.e. in the optimised profile these function calls dont appear. That implies in **Release** profile small function calls like +, -, max, min etc are minimised by using the flag -O3 of the g++ compiler.

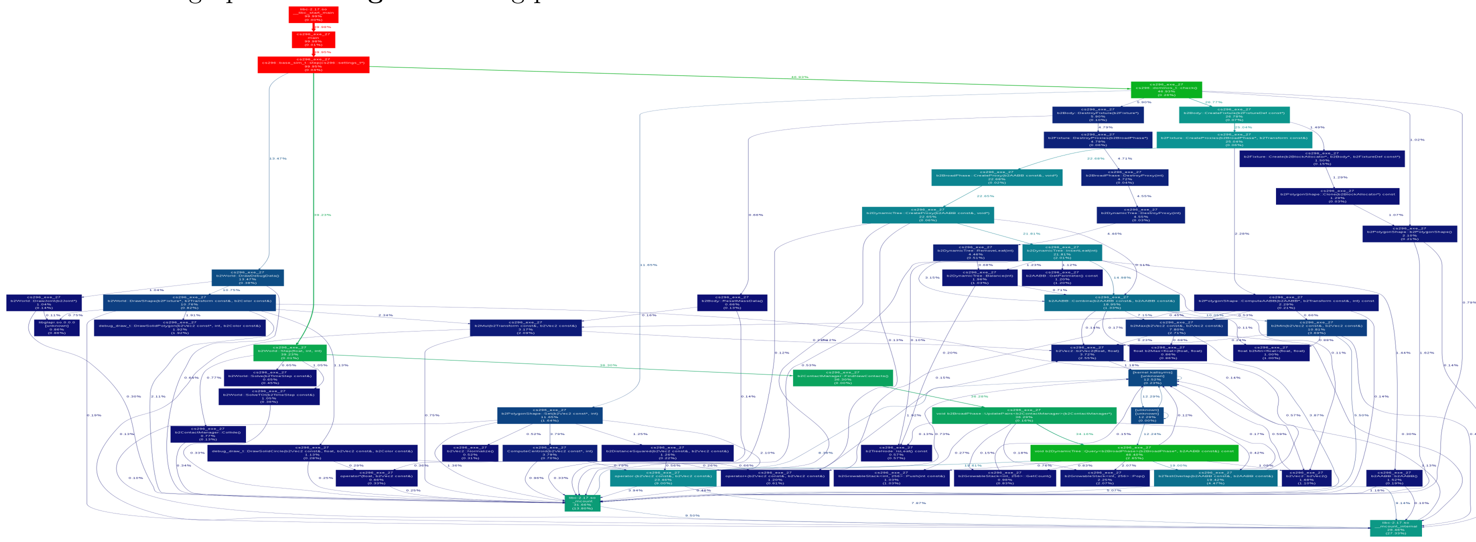
3. Essentially, turning on optimization flags makes the compiler attempt to improve the performance and/or code size at the expense of compilation time and possibly the ability to debug the program.

4. Different commands(-O, -O2, -O3, Ofast etc) turn on different optimization flags [7]

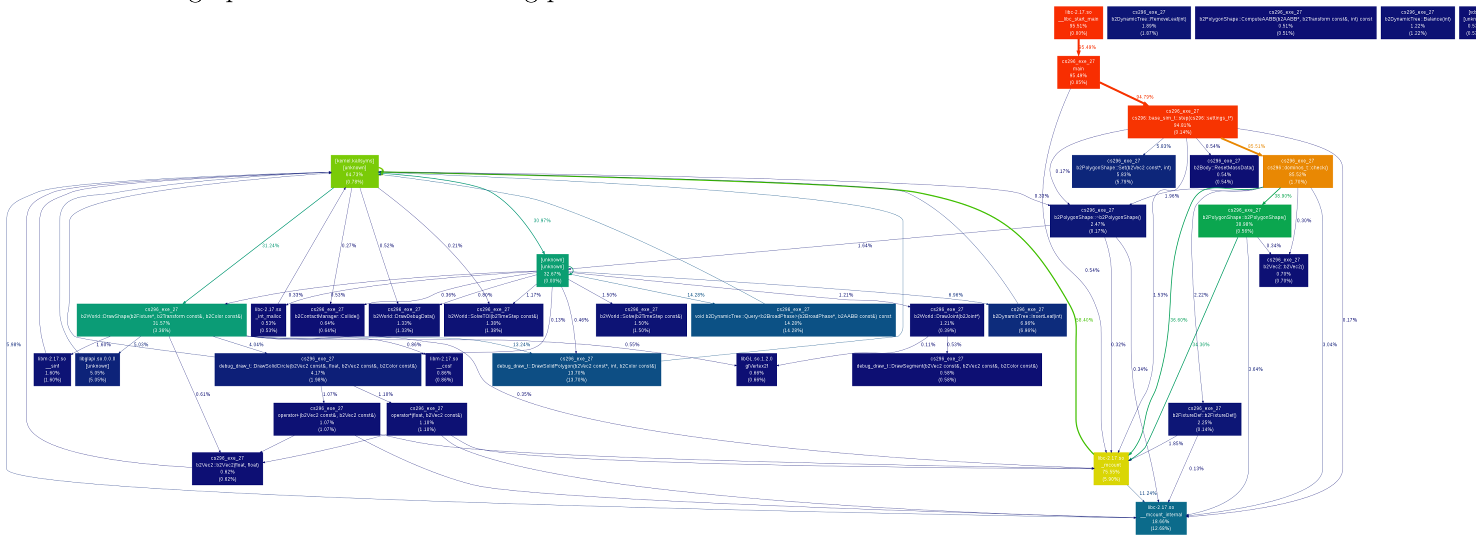
5. It is interesting to see that the number of function calls made by the function DrawSolidPolygon() in **Debug** mode and the **Release** mode are the same (order of 8)

5.5 Callgraph

1. Here is the call graph of **Debug** mode using perf



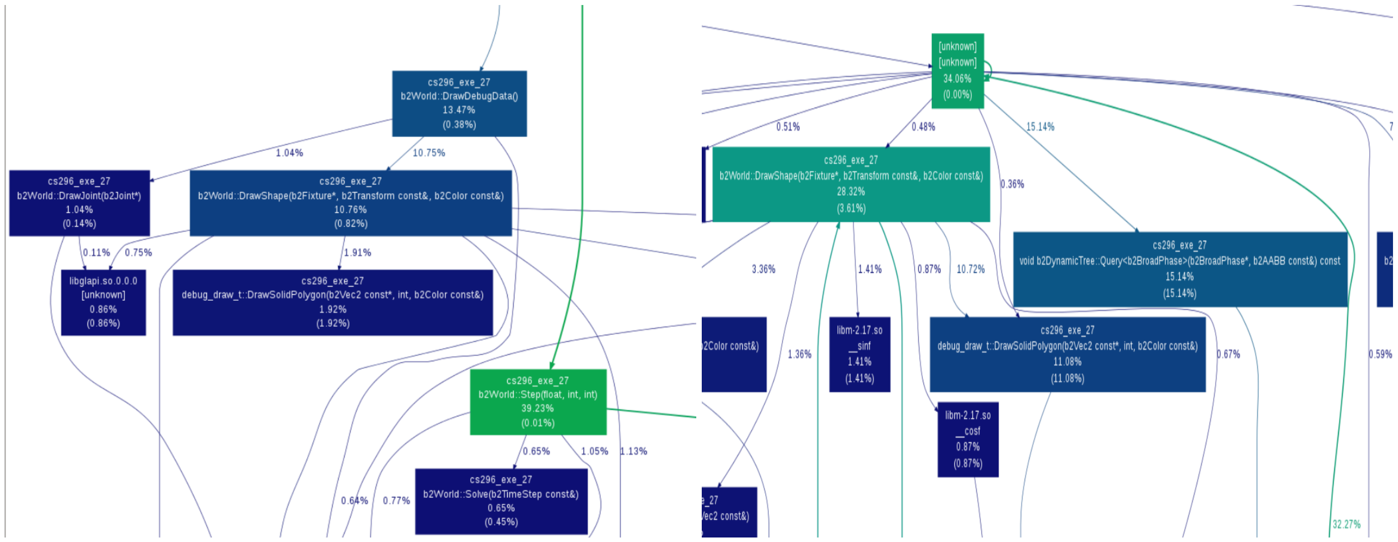
2. Here is the call graph of **Release** mode using perf



3. From these we can clearly see the difference in number of function called in total which is very much high in **Debug** mode than in **Release** mode.

4. Callgraph gives information about the name of the function % time it takes during the execution

5. In the call graph for **Debug** mode functions like b2Min, b2Mul take 3.69, 2.09 % of the total time where as in the callgraph for **Release** mode they dont even appear



6 Conclusion

We have hence created a self contained two player foosball game using keyboard inputs and Box2D simulation We have used the techniques learnt in earlier labs to make the code efficient and document it.

References

[1] Our batchmate Shivam Garg.

[2] https://www.google.com.

[3] https://www.iforce2d.net/b2dtut/.

[4] The Box 2D API. http://www.learn-cocos2d.com/api-ref/1.0/Box2D/html/.

[5] Jeff Atwood and Joel Spolsky. https://www.stackoverflow.com. 2009.

[6] Mayank Meghwanshi. http://www.cse.iitb.ac.in/~manku/files/g11_prof_report.pdf. 2013.

[7] Unknown. http://gcc.gnu.org/onlinedocs/gcc/Optimize-Options.html. 2013.