IMPLEMENTATION

Quick Tip: Use Quadtrees to Detect Likely Collisions in 2D Space

by Steven Lambert *3 Sep 2012* 73 Comments

Many games require the use of collision detection algorithms to determine when two objects have collided, but these algorithms are often expensive operations and can greatly slow down a game. In this article we'll learn about quadtrees, and how we can use them to speed up collision detection by skipping pairs of objects that are too far apart to collide.

Note: *Although this tutorial is written using Java, you should be able to use the same techniques and concepts in almost any game development environment.*

Introduction

Collision detection is an essential part of most video games. Both in 2D and 3D games, detecting when two objects have collided is important as poor collision detection can lead to some very interesting results:

Beyond Good & Evil *Ultimate* glitch - Part 2

0:00 / 1:53

However, collision detection is also a very expensive operation. Let's say there are 100 objects that need to be checked for collision. Comparing each pair of objects requires 10,000 operations - that's a lot of checks!

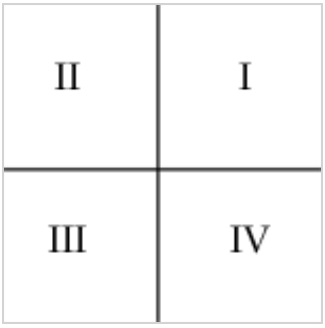
One way to speed things up is to reduce the number of checks that have to be made. Two objects that are at opposite ends of the screen can not possibly collide, so there is no need to check for a collision between them. This is where a quadtree comes into play.

What Is a Quadtree?

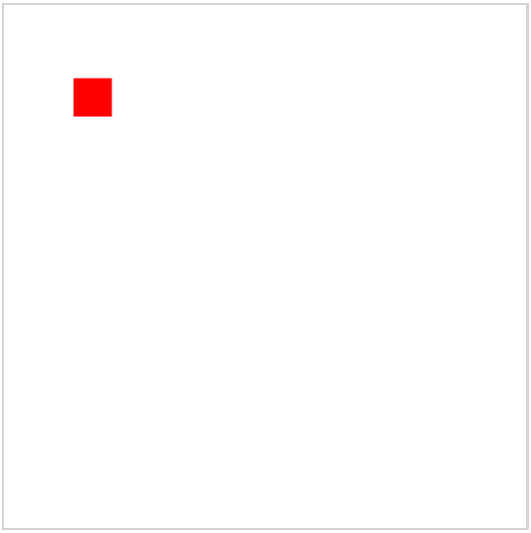
A quadtree is a data structure used to divide a 2D region into more manageable parts. It's an extended binary tree, but instead of two child nodes it has four.

In the images below, each image is a visual representation of the 2D space and the red squares represent objects. For the purposes of this article, subnodes will be labelled counter-clockwise as

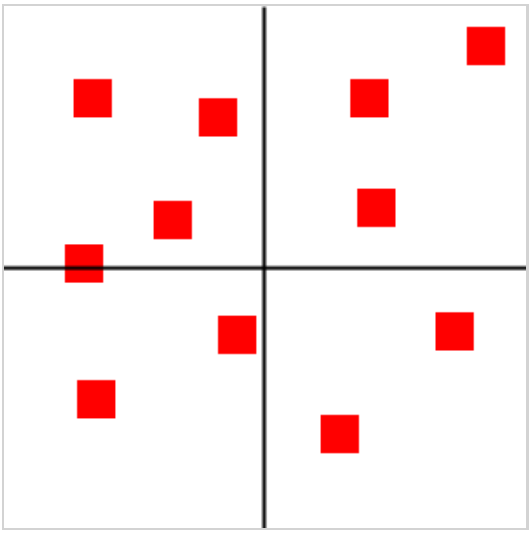
follows:



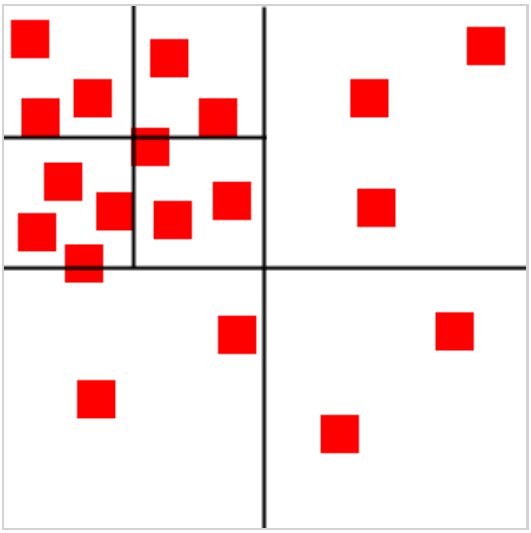
A quadtree starts as a single node. Objects added to the quadtree are added to the single node.



When more objects are added to the quadtree, it will eventually split into four subnodes. Each object will then be put into one of these subnodes according to where it lies in the 2D space. Any object that cannot fully fit inside a node's boundary will be placed in the parent node.



Each subnode can continue subdividing as more objects are added.



As you can see, each node only contains a few objects. We know then that, for instance, the objects in the top-left node cannot be colliding with the objects in the bottom-right node, so we don't need to run an expensive collision detection algorithm between such pairs.

Take a look at this JavaScript example to see a quadtree in action.

Implementing a Quadtree

Implementing a quadtree is fairly simple. The following code is written in Java, but the same techniques can be used for most other programming languages. I'll comment after each code snippet.

We'll start off by creating the main Quadtree class. Below is the code for `Quadtree.java`.

```
01 public class Quadtree {
02
03   private int MAX_OBJECTS = 10;
04   private int MAX_LEVELS = 5;
05
06   private int level;
07   private List objects;
08   private Rectangle bounds;
09   private Quadtree[] nodes;
10
11  /*
12   * Constructor
13   */
14   public Quadtree(int pLevel, Rectangle pBounds) {
15
```

```
    level = pLevel;
    objects = new ArrayList();
    bounds = pBounds;
    nodes = new Quadtree[4];
  }
}
```

The `Quadtree` class is straightforward. `MAX_OBJECTS` defines how many objects a node can hold before it splits and `MAX_LEVELS` defines the deepest level subnode. `Level` is the current node level (0 being the topmost node), `bounds` represents the 2D space that the node occupies, and `nodes` are the four subnodes.

In this example, the objects the quadtree will hold are `Rectangles`, but for your own quadtree it can be whatever you want.

Next, we'll implement the five methods of a quadtree: `clear`, `split`, `getIndex`, `insert`, and `retrieve`.

```
/*
 * Clears the quadtree
 */
public void clear() {
  objects.clear();

  for (int i = 0; i < nodes.length; i++) {
    if (nodes[i] != null) {
      nodes[i].clear();
      nodes[i] = null;
    }
  }
}
```

The `clear` method clears the quadtree by recursively clearing all objects from all nodes.

```
/*
 * Splits the node into 4 subnodes
 */
private void split() {
  int subWidth = (int)(bounds.getWidth() / 2);
  int subHeight = (int)(bounds.getHeight() / 2);
  int x = (int)bounds.getX();
  int y = (int)bounds.getY();


```

```
    nodes[0] = new Quadtree(level+1, new Rectangle(x + subWidth, y, subWidth, s
    nodes[1] = new Quadtree(level+1, new Rectangle(x, y, subWidth, subHeight));
    nodes[2] = new Quadtree(level+1, new Rectangle(x, y + subHeight, subWidth,
    nodes[3] = new Quadtree(level+1, new Rectangle(x + subWidth, y + subHeight,
  }
```

The `split` method splits the node into four subnodes by dividing the node into four equal parts and initializing the four subnodes with the new bounds.

```
/*
 * Determine which node the object belongs to. -1 means
 * object cannot completely fit within a child node and is part
 * of the parent node
 */
 private int getIndex(Rectangle pRect) {
   int index = -1;
   double verticalMidpoint = bounds.getX() + (bounds.getWidth() / 2);
   double horizontalMidpoint = bounds.getY() + (bounds.getHeight() / 2);

   // Object can completely fit within the top quadrants
   boolean topQuadrant = (pRect.getY() < horizontalMidpoint && pRect.getY() +
   // Object can completely fit within the bottom quadrants
   boolean bottomQuadrant = (pRect.getY() > horizontalMidpoint);

   // Object can completely fit within the left quadrants
   if (pRect.getX() < verticalMidpoint && pRect.getX() + pRect.getWidth() < ve
      if (topQuadrant) {
        index = 1;
      }
      else if (bottomQuadrant) {
        index = 2;
      }
    }
    // Object can completely fit within the right quadrants
    else if (pRect.getX() > verticalMidpoint) {
     if (topQuadrant) {
       index = 0;
     }
     else if (bottomQuadrant) {
       index = 3;
     }
   }

   return index;
 }
```

The `getIndex` method is a helper function of the quadtree. It determines where an object belongs

in the quadtree by determining which node the object can fit into.

```
/*
 * Insert the object into the quadtree. If the node
 * exceeds the capacity, it will split and add all
 * objects to their corresponding nodes.
 */
public void insert(Rectangle pRect) {
  if (nodes[0] != null) {
    int index = getIndex(pRect);

    if (index != -1) {
      nodes[index].insert(pRect);

      return;
    }
  }

  objects.add(pRect);

  if (objects.size() > MAX_OBJECTS && level < MAX_LEVELS) {
     if (nodes[0] == null) {
        split();
     }

    int i = 0;
    while (i < objects.size()) {
      int index = getIndex(objects.get(i));
      if (index != -1) {
        nodes[index].insert(objects.remove(i));
      }
      else {
        i++;
      }
    }
  }
}
```

The `insert` method is where everything comes together. The method first determines whether the node has any child nodes and tries to add the object there. If there are no child nodes or the object doesn't fit in a child node, it adds the object to the parent node.

Once the object is added, it determines whether the node needs to split by checking if the current number of objects exceeds the max allowed objects. Splitting will cause the node to insert any object that can fit in a child node to be added to the child node; otherwise the object will stay in the parent node.

```
/*
 * Return all objects that could collide with the given object
 */
public List retrieve(List returnObjects, Rectangle pRect) {
  int index = getIndex(pRect);
  if (index != -1 && nodes[0] != null) {
    nodes[index].retrieve(returnObjects, pRect);
  }

  returnObjects.addAll(objects);

  return returnObjects;
}
```

The final method of the quadtree is the `retrieve` method. It returns all objects in all nodes that the given object could potentially collide with. This method is what helps to reduce the number of pairs to check collision against.

Using This for 2D Collision Detection

Now that we have a fully functional quadtree, it's time to use it to help reduce the checks needed for collision detection.

In a typical game, you'll start by creating the quadtree and passing the bounds of the screen.

```
Quadtree quad = new Quadtree(0, new Rectangle(0,0,600,600));
```

At every frame, you'll insert all objects into the quadtree by first clearing the quadtree then using the `insert` method for every object.

```
quad.clear();
for (int i = 0; i < allObjects.size(); i++) {
  quad.insert(allObjects.get(i));
}
```

Once all objects have been inserted, you'll go through each object and retrieve a list of objects it could possibly collide with. You'll then check for collisions between each object in the list and the initial object, using a collision detection algorithm.

```
List returnObjects = new ArrayList();
for (int i = 0; i < allObjects.size(); i++) {
  returnObjects.clear();
  quad.retrieve(returnObjects, objects.get(i));

  for (int x = 0; x < returnObjects.size(); x++) {
    // Run collision detection algorithm between objects
  }
}
```

Note: Collision detection algorithms are beyond the scope of this tutorial. See this article for an example.

Conclusion

Collision detection can be an expensive operation and can slow down the performance of your game. Quadtrees are one way you can help speed up collision detection and keep your game running at top speeds.

Related Posts

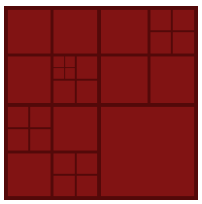
- Make Your Game Pop With Particle Effects and Quadtrees

Difficulty:
Intermediate

Length:
Quick

Tagged with:

Implementation | Platform Agnostic | Java



About Steven Lambert

Steven Lambert is a Computer Science graduate, a freelance web and applications developer, avid gamer,

+ Expand Bio

Related Posts

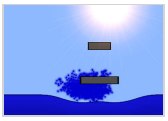
2 days ago • In this tutorial, I'll show you how to take advantage of the new 2D Tools included in Unity to cr...

7 Mar 2014 • In the previous tutorial, we laid the foundation of our Missile Command game by creating the

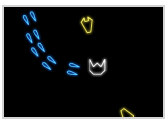
proj...

22 Feb 2014 • Learn about array lists, the building blocks for creating solutions to business problems via algo...

22 Feb 2014 • Learn about linked lists, the building blocks for creating solutions to business problems via alg...



18 Jan 2014 • In this tutorial, we're going to simulate a dynamic 2D body of water using simple physics. We wil...



11 Nov 2013 • In the first part of this series on building a Geometry Wars-inspired game in jMonkeyEngine, we i...

Steven Lambert ➚ Chris Lawhorn • a year ago

Ya, in JavaScript the method returns the removed object, which allows you to insert it into the subnode. Sorry about that, thought I removed all JavaScript style code from the quadtree in this tutorial, guess I missed one. If you're implementing it in C# or Java, you'll have to add the object first to the subnode first, then remove it from the array. If you're using an ArrayList for the objects, it would look something like this:

```
if (index != -1) {
   nodes[index].insert(objects.get(i));
   objects.removeAt(i);
}
else {
   i++;
}
```

Chris Lawhorn ➚ Steven Lambert • a year ago

Ooooh ok. Thank you for the tutorial and helpful reply.

Jacob Davis • a year ago

Great tutorial, but Im running into a bit of an issue with the retrieve method. I'm building this code in C#.

From what I can tell by debugging everything in my list of three objects get inserted and indexed correctly. But when calling the retrieve method, it never returns any matches. All three objects are in the exact same place and I have confirmed they are indexed the same way.

Bellow are two gists to my Quadtree.cs class and the main class calling it.

TestBed.cs
https://gist.github.com/ismyhc...

QuadTree.cs
https://gist.github.com/ismyhc...

Any help much appreciated!

1 • Reply • Share ›

Jacob Davis Jacob Davis • a year ago

Okay, so I actually figured out my problem. In my Retrieve method I was returning the empty list instead of the objects.

Ive updated the gists.

Everything seems to be working now except for on odd issue I am seeing. Im linking to a video that shows the behavour.

I change the color of the moving objects to red when they are returned from the Retrieve method as a possible collision. For whatever reason the object on the bottom quadrant apparent does one check everytime it crosses the into or out of the left and right bottom quadrants. Why is this?

One thing different about my setup is that 0,0 is the center of the screen. positive number to the right, and top.

Any ideas?

Video
https://dl.dropbox.com/u/33090...

Thanks!

• Reply • Share ›

Jacob Davis Jacob Davis • a year ago

Well, I figured this out too. Def had to change up how I calculated which index the object belonged too.

Now, hopefully my final question. How do you handle when an object is in multiple quads? Say if the you object is intersecting two quads, or even four?

• Reply • Share ›

Steven Lambert Jacob Davis • a year ago

Hey Jacob,

Nice work figuring out your own questions! Sorry I wasn't available to answer them.

To answer your question about the object entering "possible collision" as it passed from one major quadrant to another is that the object gets placed into the root's object array as it passes over multiple quadrants because it could belong to multiple quadrants at the same time. This causes the retrieve method to return it to ensure that it doesn't collide with any other objects that belong to those subnodes.

As for the question with multiple quads. I'm not sure how that would

work. I only know how 1 quadtree works and have not seen multiple quad trees in action at the same time. Sorry I couldn't be of more help.

• Reply • Share ›

liber145 ➚ Steven Lambert • 9 months ago

Hi, Steven.

I think there is a bug with retrieve() function. If (index == -1), returnObjects should include all children's rectangles. Because it's possible for pRect to collide with them.
Anyway, thank you for the tutorial, it really helps me a lot!

4 • Reply • Share ›

inceptive ➚ liber145 • 5 months ago

You are right! I came back to this article specifically to read if anyone has the same issue with this code.
Indeed you are 100% right. My drawing list was incomplete due to that - often lacking the player character itself! :D

2 • Reply • Share ›

Jacob Davis ➚ Steven Lambert • a year ago

So I actually got this to work. Or its working in my test! I plan on implementing this into my iOS retro space shooter.

Below is a video of the quadtree returning multiple indexes when the rect overlaps the vertical midpoint, horizontal midpoint, or both!

http://www.youtube.com/watch?v...

Thanks again for the tutorial. It helped me alot!

• Reply • Share ›

Steven Lambert ➚ Jacob Davis • a year ago

No problem, glad you were able to get this working!

I am curious though, what is your frame rate at with the additional overhead of adding the object to multiple subnodes of the quadtree? Do you constantly achieve 60FPS? If so, I'd like to know what you did haha.

• Reply • Share ›

Jacob Davis ➚ Steven Lambert • a year ago

Actually Im getting a pretty consistent 60FPS on my macbook pro. This is with 500 objects too!

Ive created a webplayer version of the test. It gets a consistent 59-

58FPS. The number of objects is pretty extreme so I have to say Im pretty happy with it so far. My next step is to implement this into my current game.

https://dl.dropbox.com/u/33090...

Let me know what you think!

3 ^ | v • Reply • Share ›

Steven Lambert ➚ Jacob Davis • a year ago

That is pretty sweet! Awesome to see that it gets really good FPS. So what was the secret to add an object to multiple subnodes?

^ | v • Reply • Share ›

Jacob Davis ➚ Steven Lambert • a year ago

Well first off I modified the getIndex method to return a List<int>. Inside of the the getIndex method I did calculations to find when an object was in both top and bottom quads. Then also calculated when the object was in both left and right quads. Once I determined that then I just simply added the indexes to the list. Then I had to modify the calls to the getIndex function to handle more than one index returning.

Probably easier just to show you the class ;)

There are probably room for improvements, but this is implementation I have so far.

https://gist.github.com/ismyhc...

Improvement suggestions are welcome ;)

1 ^ | v • Reply • Share ›

Massimo ➚ Jacob Davis • a year ago

Probably an int[] would be more performant than a List<int>. Since an object can be in 4 quadrants at time, an int[4] is all you need. Plus you can declare it as a field (and not a variable) and allocate it one time only.

^ | v • Reply • Share ›

Jacob Davis ➚ Steven Lambert • a year ago

No prob! I always feel a sense of accomplishment when I end up figuring out my own questions!

What I am trying to do now is assign an object more than one index at a time if its spanning more than one index. For instance say the object is spanning the verticalMidpoint. Which index should it belong to? I would think it should be in both. Reason being that if I have assigned to one of

the two possible then if another object was only in the index I didn't assign it to then its very possible I could miss a collision.

What Im working on now is returning a list indexes instead of just one index. I think what Im working on is going to work, but you can never be to sure! Ill post some video if it does ;)

• Reply • Share ›

Timo • 2 years ago

Hi, thanks for the article, interesting read!

I implemented your methods into a javascript class, you can check it out here:
Git: https://github.com/timohausman...
Demo: http://jsfiddle.net/2dchA/

I'm wondering about one thing: the retrieve-method in my implementation does not return the values that I expected, when the given rect overlaps the borders of a node.
You can see this in the JS-Fiddle Demo. As long as the green rect stays inside the bounds of a node, everything works fine. (You can move it with the mouse.)
But as soon as it overlaps any node borders, the method does not return objects inside the nodes nearby.

Is this behaviour "correct"? I don't think so ...

1 • Reply • Share ›

Timo ➔ Timo • 2 years ago

Overthinking the retrieve function, I came to a solution that adresses that problem.
What I changed:

When getIndex in the retrieve function returns "-1" (given pRect does not fit into a single subnode), you only add the objects that are directly in the parent node to returnObjects, but you don't add all the objects in subnodes.

I added a for loop for that case, retrieving all objects of the current parents subnode.

Please take a look at line 154 and following and tell me what you think:
https://github.com/timohausman...

Git and JS-Fiddle is updated, I think it now behaves like it should.

• Reply • Share ›

Steven Lambert ➔ Timo • 2 years ago

Hey Timo,

I love the JS-Fiddle you created, that is an awesome visual demonstration of a quadtree and I'm glad to see the program taken beyond just Java.

As for the concern with the retrieve method, this isn't really a problem at all depending on how you write the collision detection. It is true that the retrieve method will not return any subnode's objects when the object crosses a node's boundary. Thus using that object alone for testing collision detection will not work properly.

However collision detection involves at least two objects, so even though the object crossing a node's boundary will not return proper collision with other objects in subnodes, the other objects (which would be in subnodes) will return correct collision since the retrieve method returns all objects in parent nodes. Therefore when you find collision, the algorithm should set both objects collision. This will ensure that all collision detection cases are found and handled, and doing so makes the quadtree perform faster and more efficient.

Adding a loop to get all subnode's objects in the retrieve method would allow you to have proper collision detection for every object, so it just depends on how the collision detection works to say which way is better for your code.

Steven

• Reply • Share ›

Timo ➚ Steven Lambert • 2 years ago

Hi Steven,
thanks for your reply!

Guess you are right there! For the way I'm currently thinking of my collision detection, the subnode loop feels right.

Just for the record, this is the latest working fiddle:
http://jsfiddle.net/2dchA/2/

1 • Reply • Share ›

zepp • a month ago

are you clearing the quad and insert everything again every frame ?

• Reply • Share ›

Steven Lambert ➚ zepp • a month ago

That is correct.

• Reply • Share ›

Tron1 • 3 months ago

I'm trying to convert this to c++ and it's not making sense to me in a few areas.

They are this

private List objects;
private Rectangle bounds;
private Quadtree[] nodes;

What is List a vector<int> objects:?
What is Rectangle a rect like from windows using left,top,bottom,right?
What is Quadtree[]? Is it int *nodes;

I'm basically trying to convert this logic in to c++.

• Reply • Share ›

Steven Lambert ➚ Tron1 • 3 months ago

A list is just a collection of objects (kind of like an array). In c++ is it similar to a vector. Rectangle is just an object that keeps track of 4 properties: position x, position y, width, and height. This allows the Quadtree to know the bounds of the screen area it covers. Quadtree[] is an array of Quadtree objects. The Quadtree class is what we are defining, so each Quadtree object has an array of Quadtree objects inside of it. This allows us to keep track of subnodes and keep all of the objects organized by where they lie inside the bounds of a node or subnode.

• Reply • Share ›

Laggy44 • 4 months ago

What exactly are you doing at line #9, private Quadtree[] nodes;
I'm trying to convert this to AS3 but I'm not sure what to do there. I'm not familiar with Java.

• Reply • Share ›

Steven Lambert ➚ Laggy44 • 3 months ago

Quadtree[] is an array of Quadtree objects. The Quadtree class is what we are defining, so each Quadtree object has an array of Quadtree objects inside of it. This allows us to keep track of subnodes and keep all of the objects organized by where they lie inside the bounds of a node or subnode.

• Reply • Share ›

inceptive • 5 months ago

I have been testing and implementing this tree today based on your tutorials and here are two issues I found:
1. If an object's (input) rectangle "contains" a quadtree rectangle, then the children of that rectangle will not be returned. Therefore, I implemented intersection tests used for retrieve() operation. getIndex is perfect for insert(), but is not sufficient for retrieve().
2. if (objects.size() > MAX_OBJECTS && level < MAX_LEVELS) {
if (nodes[0] == null) { split(); }
int i = 0; while (i < objects.size()) { (...) }

The while loop should be inside the if(nodes[0] == null) - we do that only if we did split()!

• Reply • Share ›

Steven Lambert ➚ inceptive • 5 months ago

I see what you mean. That conditional was added after the article was written due to comments from others who found a bug with it. I can see that you are correct though. Thanks for finding that :)

In regards to the getIndex() method for retrieve(), it works if you are checking all objects. This is because if an object is located in the topmost part of the tree (i.e., it fits into all 4 sub-nodes) and retrieve() returned all objects in the tree, it would cause you to check the object against every other object, defeating the purpose of the quadtree.

Instead, if you loop through each object, the bottom most object will always get the object that is at the topmost part of the three and collision detection will still catch it. This saves unnecessary checks in the long run, but it will seem to be wrong if you check just a few objects and not all of them.

▲ | ▼ • Reply • Share ›

Colton • 7 months ago

I know this is old by now but I'll try anyway... I implemented this in c#, without any modifications, and the retrieve method is returning the object that I started with. For example, retrieve(retrievedList, objects[i]) will also include objects[i] in the retrieved list. I don't think this is meant to happen, but if it is, that object would always collide with itself when checking for collision... Which is happening in my implementation. Any ideas?

▲ | ▼ • Reply • Share ›

Adia ➚ Colton • 7 months ago

Maybe you can add a conditional in your collision function to avoid evaluate the object itself. Or just remove that element in the returned list. Ordinarily your evaluated object not have to be evaluated with himself.

▲ | ▼ • Reply • Share ›

Colton ➚ Adia • 7 months ago

Yeah, some sort of indexing to avoid comparing to itself. But, I am still wondering if it is because I have implemented the functions wrong somehow. I am asking if it normally happens , I'd rather not create a small fix to something that is broken on a larger scale.

▲ | ▼ • Reply • Share ›

Steven Lambert ➚ Colton • 6 months ago

The current implementation of the `retrieve()` method will return the object that was passed. This is because the method just looks at the current node the object belongs in and starts adding all objects from there down (which will include itself). You can do a one of two things to not have this occur:

not have this occur.

1) After the method is done, you can just remove the object from the returned list
2) Add a conditional check in the method that will not add the object that was passed (a bit harder since the code adds all the objects at the current node)

• Reply • Share ›

Mannician • a year ago

I definitely feel as though I have a better understanding because of your well thought out tutorial, but when I try to copy these concepts into my own java program, I run into one issue, and that is an IndexOutOfBoundsException, and it happens when I go to test my collisions. I think it has something to do with the way the inserting code works, but I cannot quite nail it down.

this is the chunk of code where the exception is being thrown from:

ArrayList<orb> possibleCollisions = new ArrayList<orb>();
for(int i=0; i<orbs.size(); i++)="" {="" possiblecollisions.clear();="" quad.retrieve(possiblecollisions,="" orbs.get(i));="" for(int="" j="0;" j<possiblecollisions.size();="" j++)="" {="" rectangle="" r="possibleCollisions.get(i).getBounds();" this="" is="" where="" the="" exception="" is="" thrown.="" }="" }="" although="" i="" cannot="" quite="" figure="" out="" why="" it's="" doing="" this,="" i="" have="" determined="" the="" number="" of="" orbs="" that="" i="" can="" get="" on="" screen="" is="" directly="" influenced="" by="" the="" max_objects="" variable.="" as="" soon="" as="" the="" number="" of="" orbs="" i="" get="" on="" the="" screen="" passes="" that="" thresh="" hold,="" the="" program="" throws="" the="" exception,="" ultimately="" leading="" me="" to="" believe="" that="" it="" has="" something="" to="" do="" with="" the="" way="" the="" quad="" tree="" splits,="" but="" i="" have="" even="" been="" reduced="" to="" copying="" your="" code="" letter="" for="" letter,="" still="" to="" no="" avail.="" i="" am="" completely="" stuck.="">

• Reply • Share ›

Michael James Williams Mod ➚ Mannician • a year ago

Formatted code:

```
ArrayList possibleCollisions = new ArrayList();
for(int i=0; i<orbs.size(); i++)
{
  possibleCollisions.clear();
  quad.retrieve(possibleCollisions, orbs.get(i));

  for(int j=0; j<possibleCollisions.size(); j++)
  {
    Rectangle r = possibleCollisions.get(i).getBounds(); //This is where the e
```

```
    }
}
```

Is it possibly just a typo? Try:

```
Rectangle r = possibleCollisions.get(j).getBounds(); //get(j) instead of get(i,
```

• Reply • Share ›

Steven Lambert ➚ Michael James Williams • a year ago

Michael,

Thanks for formatting the code, that helped a lot.

Mannician,

By the looks of the code, I think Michael is correct. It should be `.get(j)` instead of `.get(i)` as `possibleCollisions.size` is less than `orbs.size()`.

• Reply • Share ›

Stew Reive • a year ago

Steven, this is a really helpful tutorial! I have ported this quadtree to python/pygame for a small demo. I am now redoing it in C++ using OpenGL and GLUT APIs. Now, my goal is to squeeze out every ounce of performance. That said, I think there are two things about your implementation that could be changed to greatly improve the performance.

First and foremost, insert() should always store each object in the deepest possible node that it fits cleanly inside (level < MAX_LEVEL of course). This ensures that small objects will get buried deep into the quadtree and be tested for collision very infrequently. I cannot rigorously prove it, but I have a suspicion that this strategy minimizes the number of objects retrieve() will return for any given pRect. However, the trade-off is that this will require more memory than your current implementation as the tree splits more frequently.

Secondly, instead of rebuilding the tree every frame, you could have it update existing nodes starting from the leaves up to the root. I realize that you have already discussed this on here, but I just wanted to add to the discussion. By updating, you exploit temporal coherence (i.e. objects often stay in the same node in successive frames so update() will essentially do nothing for these). Of course, you also save the time used to free and rebuild the tree. However, this update() method is getting really, really tricky for me to implement, especially in C++ where memory management quickly turns into a nightmare. It's forcing me to modify insert() and getIndex() as well so I can have a case where an object doesn't fit at all into a node. I am very sure that it will ultimately increase my program's performance, but for now I am still just rebuilding the tree every frame :P Hopefully I'll finish this thing and post a link to my GitHub repository with the finished product!

Thanks again for the excellent tutorial! You rock!

Thanks again for the excellent tutorial! You rock!

• Reply • Share ›

Steven Lambert ➚ Stew Reive • a year ago

Hey Stew,

I'm glad you've found this article helpful. I'm always pleased to see this article being used as a guide to extend the quadtree to other languages.

I'm also always interested in feedback to my implementation. I am curious to know what you mean when you say that an object should be stored in the deepest possible node that it fits cleanly inside. I am not sure how that is exactly different from the currently implementation of the `insert()` method.

I would love to see your current work for how you are trying to solve not recreating the tree each frame, and would be willing to help out to get it to work if possible. This would be a wonderful thing to have available to improve the performance.

• Reply • Share ›

Stew Reive ➚ Steven Lambert • a year ago

Here's more or less how I did the insert method: http://pastebin.com/EyyWUNr7 An example of it in action would look something like this (not my video): http://www.youtube.com/watch?v...

see more

• Reply • Share ›

Yakobu • a year ago

I implemented a Quadtree, inspiring myself from your tutorial, but for my current small 2D game project.

I would really appreciate though if you could check it to see if it is accurate.

https://gist.github.com/Yakobu...

I have a file called QuadTree, being the QuadTree, should be accurate I think, but mind to see if it is really?

Also included in the file MainPanel the function I created for collision detection, following your explanation, but mind to explain line 12 to 14? I am REALLY not sure if that's what you meant, plus explain to me what does it do in more details even if it is accurate. As last, my detection algorithm just below,I have my doubt how it would work, if it does, if not, how to solve?

Thank you in advance.

• Reply • Share ›

Steven Lambert ➔ Yakobu • a year ago

Hey Yakobu,

Very nice implementation of the quadtree. I looked over what you asked me to, and here are my suggestions:

For the `hasCollision` function, lines 13 and 14 should be the outer for loop, and lines 17 to 21 should be an inner for loop. As it is currently written in your file, it doesn't really do anything. It should be written as follows:

```
//Creates another list of rectangles that share nodes with the original list
for(int j = 0; j < list.size(); j++)
{
   returnObj.clear();
   quad.retrieve(returnObj, list.get(j));

   //Loop that will check all colitions and return true if one occurs false oth
   for(int k = 0; k < returnObj.size(); k++)
   {
      if(list.get(k).intersects(returnObj.get(k)))
```

see more

1 • Reply • Share ›

Yakobu ➔ Steven Lambert • a year ago

I have updated my hasCollision method, https://gist.github.com/Yakobu... but still need help please. When I try my method, it brings back nullpointerexeption right at the first line even though I check if the node is empty or not. If I ignore and put in comment, at the insert line too I have the same exception (at QuadTree.insert(QuadTree.java:102).

Also if I understand, it checks if there is a collision at all if yes, return true. But how could I know which of the objects (rectangles) has collided to code a reaction accordingly?

Help please, thank you in advance.

• Reply • Share ›

Steven Lambert ➚ Yakobu • a year ago

Hey Yakobu, sorry it's taken me a bit to get back to you.

I looked into your code and found the problem. The problem occurs in your `clear()` function in the `QuadTree` class. The first time you try to clear your QuadTree, the `nodes` array is empty. Thus, when you try to access the array, you access an empty array element and get the nullpointerexception. To fix this, you need to ensure in the `clear()` function that the current level has child levels that can be accessed before you loop through the `nodes` array.

As for how to know which objects have collided and react to them, take a look at this tutorial where I implemented this exact code: http://blog.sklambert.com/html...

• Reply • Share ›

Guest ➚ Steven Lambert • a year ago

I corrected as suggested, didn't see the inner and outer loop parts.
But then my function leads me to another question, what does it actually returns? if there is a collisions, but in general? how can I know what object has a collision and make it do a reaction according to my wish for the game?

• Reply • Share ›

Chris Lawhorn • a year ago

Alright, I realized there was something else I didn't understand. If our collision algorithm is ran on the list of returned objects (which is refreshed every frame mind you), how would it effect the actual objects in their actual arrays/lists?

• Reply • Share ›

Steven Lambert ➚ Chris Lawhorn • a year ago

Since the quadtree only sets the `isColliding` variable of the objects, it doesn't affect the objects in their actual arrays/lists. The actual arrays/lists remain unchanged throughout the entire process. All the quadtree does is organize the objects of those lists into pieces that can be used to access them quickly.

• Reply • Share ›

Chris Lawhorn ➚ Steven Lambert • a year ago

Ok, but then why are collision algorithms being ran on the returnObjects?

Ok, but then why are collision algorithms being ran on the returnObjects? Wouldn't we want our collision code to be ran on the actual objects? Also, how would it set the isColliding variable of the original objects if the quad tree puts all of the objects into its own array list separate from the original?

• Reply • Share ›

Chris Lawhorn ➚ Chris Lawhorn • a year ago

I guess what (part of it at least) I'm trying to ask is where is the link? I'm not understanding how the objects in the quadtree's return list communicate with their counterparts in the original list so you know which original objects to run the collision algorithm on.

• Reply • Share ›

Steven Lambert ➚ Chris Lawhorn • a year ago

Sorry, I just realized that this tutorial did not go over anything such as an `isColliding` variable! My bad. I got my codes mixed up :P

```
List returnObjects = new ArrayList();
for (int i = 0; i < allObjects.size(); i++) {
   returnObjects.clear();
   quad.retrieve(returnObjects, allObjects.get(i));

   for (int x = 0; x < returnObjects.size(); x++) {
      // Run collision detection algorithm between objects
   }
}
```

When you use the `retrieve()` method of the quadtree, you pass it an object (which you got from the original list of objects). The method returns another list of objects, with this list being the objects the original object shares nodes with. By looping over the returned objects and running a collision detection algorithm between the original object and each returned object in turn, you are able to know when the original object (from the original list) collides with any object, and therefore do whatever you need to do with the original object.

• Reply • Share ›

Teaching skills to millions worldwide.

Design & Illustration Code Web Design Music & Audio Photography

3D & Motion Graphics Game Development Computer Skills Crafts & DIY

Business

About Us

FAQ

Blog

Write for Us

Advertise

Suggestions

Privacy Policy

Terms & Conditions