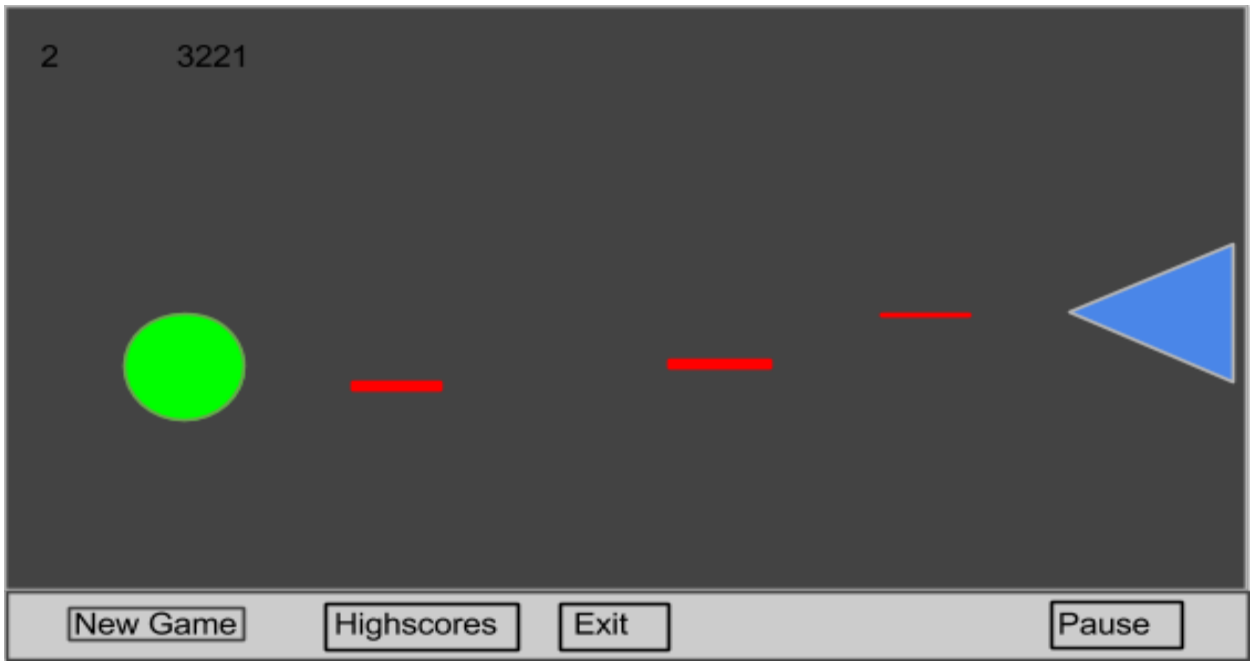
The Making Of 37Bec-Amvios



Real Men Wing It:

The basic layout of the game is above. The user is the triangle and can move up and down with the arrow keys. They will shoot by clicking the mouse and aim by pointing it. The aim of the user is to shoot and destroy the ball (probably won't be a ball in the end) before they overtake the player. If one of the ball hits the player then the ball is destroyed, the player is unable to move for a second and flashes to show this.

The user will control the non-game elements of the game from the menu along the bottom, they will click the button that will do stuff. The stuff that happens upon them clicking the button depends on the button. It will generally be pretty self explanatory from the label on the button. It may not though. Depend on what I feel like at the time of creating said button. I can say that they will be definitely be able to create a new game and pause. They may also be able to pause with a key (space or p, can't decide if it's necessary and which would be better if I did). This will result in game play stopping and a message saying "Paused' to display across the window. When the new game button is pressed a new game will be started, first some text will show, telling them how to play, or not. Watching people flounder would be rather funny. Then the game will start. The game will become progressively harder with more circles attacking the player the longer they last. When they die they will either be shown the high scores or be asked for their name then be taken to the highscores menu.

Who Needs Code When There's Pseudocode:

Pseudocode is false code that does not work and we do it for the fun, just to annoy people, computer scientists are like that. I chose to create my own language as you can see. It shows you the complexity and un-understandability of the code.

```
ljkfadsjlljk;ads jkl;l;;jlasdf ;ll;l; asdl;kl ;kadsf kl;l; l;adsf ;ll;l;lk;j
sadf
kl;jads jl;l;kl;kd asl;kl;kadsfl;kl;k;lkasd l;kl;;l  adsl;l;kl;jkds af;lk l;k
lk; fds
kjfod
fds
fsdjkjads ;lkl;l;kl;k dskl;l;kl; sdkl;l;ksad lk;l;klk;ads lkl;jkl;jksad f
k;sadf l;lk;l;ads lk;l;asd ;l;ll;ads l;;ll;ds l;lk;lk;sad jkl;ljk;dsa
fjkdsjkjkfdasljklk k dasfljkljkl dsfljk jlj
fadsjkl
jfdjkl;kldf
jadfs ll;kdsa kl;l;k;lkds al;k l;kl;k dfs

l;sadf lllk;dsl;kl;kl;k adfsl jljkl; adfsl;dfgsldf sjkl;l;jk adfs

fsadhjk fasdkl

jlkdfsal;lk;l;dslk;;lkl;akds l;l;k dsaf kl;l;k l;ksad l;klk;l;k lk; dslk; l;k
dfs
l;adsl;adsf l;l;l;'ads l ;kl;'s ;adl;';';'ds a;';;'dsa ;';'l ;'dfsa

dsall;lk;fdas
jfdjsaklf
fjkdls;a;dlkfjsljkdwkjl dsljk;lkj;  adskfljk;lkj;sad
ajfdksa;fdasf

ljk;fads ;jl lk;dfsljk; jlk;adfsljk;jkp adfs ;dkl ; ;
adfs
fsd a
 dfadfs jkl dfl
 ;adfadfg adfg lk ;
.
```

Done. That is the how I will code my game. The actual code will be in java compiled with the openjdk build on linux using vim as the text editor for the code.

Only A Psychopath Would Want To Shoot Things:

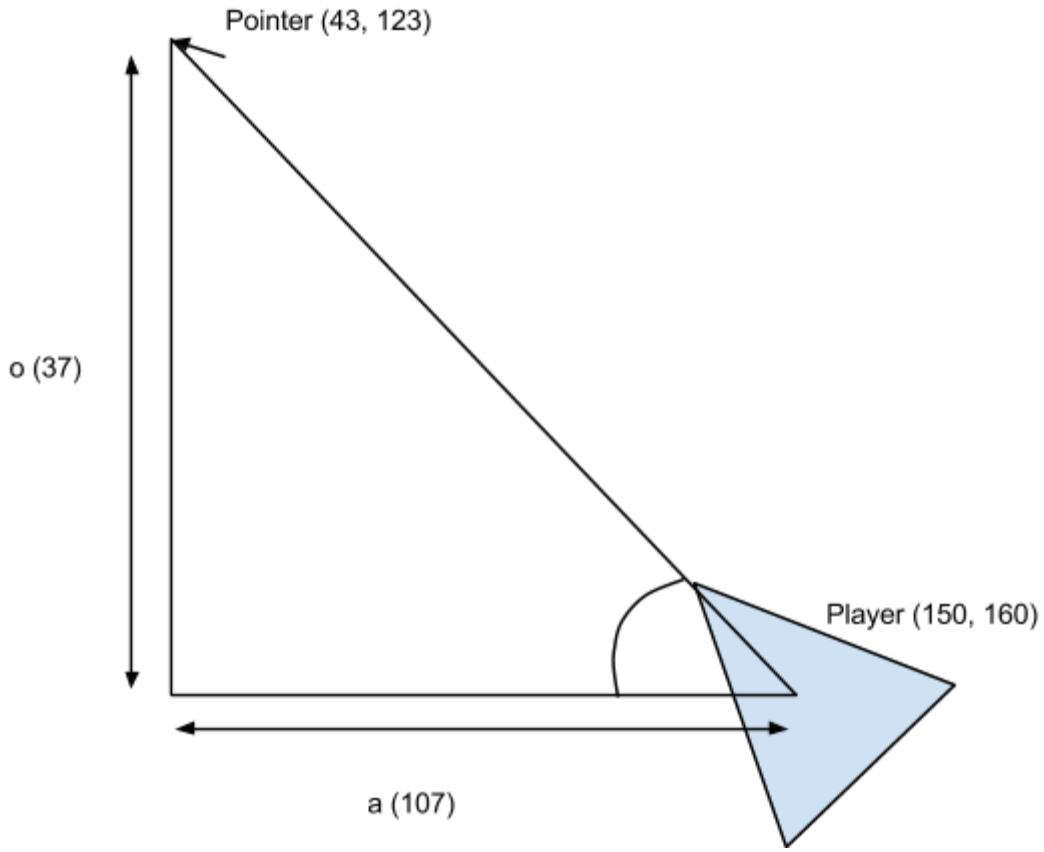
The player controls the triangle with the mouse and keys, the keys move it up and down. The mouse controls the direction they are facing and when they shoot. To work out the direction of the player I used some basic trigonometry to work out the angle from the mouse.

Using trigonometry we can see that

$$\theta = atan\frac{Y\,player - Y\,pointer}{Xplayer - Xpointer}$$

With this we can rotate the player to that angle, this does not quite work though as it starts shooting the opposite direction if aim to the right of the player, fuck you though, deal with it, too much effort to fix that fantastic equation to please you. I'll just move the player so you can't do that.

With this we can give the bullets an angle and speed then we can get the x and y displacement for each frame with this.



$$xv = -SPEED * cos\,\theta$$
$$yv = -SPEED * sin\,\theta$$

The speed is negative because we want the bullet to move to the left, decreasing its x value rather than the right. And negative for the vertical because it just is and that works and I have no idea it just works, stop asking questions, it isn’t good for you.

THIS IS ALL YOU ARE GETTING, REAL WORK AWAITS.

OKAY, I’LL DO SOME MORE.

Specially for you.

And All Wrapped Up In A Tasty Jar File:

For simplicity of distribution and user friendliness I am going to distribute this wonderful game as a jar file. A jar file is just a zip file that the java runtime treats specially because it starts with a ‘j’. It contains the classes and any resources that they may need in one simple file so that things don’t get lost. To create one we go through the horrible process of something like this.

```
jar cvef Game Shooter.jar *.class images fonts # and any other files needed
```

What this does is creates a jar file (hence the c in that vomit of char’s at the start), set Game as the class to be executed when the jar is run (the e in the mess), name it Shooter.jar (for lack of a better name) and include all the .class files (that’s a bash thing but I’ll assume you know a bit of that. If not, look it up, install linux, play with that for a while, then come back) and the images file that contains some pretty triangles at the time of writing.

This all assumes that the files have already been compiled with something like:

```
javac *.java
```

Else you could get some interesting results.

Then to run it on linux we’d use something like this:

```
java -jar Shooter.jar
```

And walla, you have you fantastic game ready and waiting.If you use one of those BDSM operating systems like Microsoft Windows or Mac OS X then you can probably just double click it and hope it works. Of course if you were to use a spoon feeder you could probably do all this with a couple hours of tutorials on youtube to find the right buttons then click them and have someone else do this for you.

Coming back this is incredibly annoying. So I made a bash script to do it for me. All it does is move into the src directory, run that horrible command,

move the jar file up a directory then exit. Simple stuff that saves a lot of time.

Fullscreen:

Because this is such a wonderful game everybody wants it to take up the entire screen and stop them from doing anything else.On linux this is fairly easy, we can just ask for the screen size and set our frame to said size. Mac however, is a more fickle bitch. Needing special treatment because of that annoying dock.

Getting the window to take over isn't much trouble. Add the following lines after you've initialized the frame and away you go:

```
GraphicsEnvironment ge = GraphicsEnvironment.getLocalGraphicsEnvironment();
GraphicsDevice[] gs = ge.getScreenDevices();

gs[0].setFullScreenWindow(frame);
```

This however seems to have some problems. YOU CAN NOT TYPE!!!.

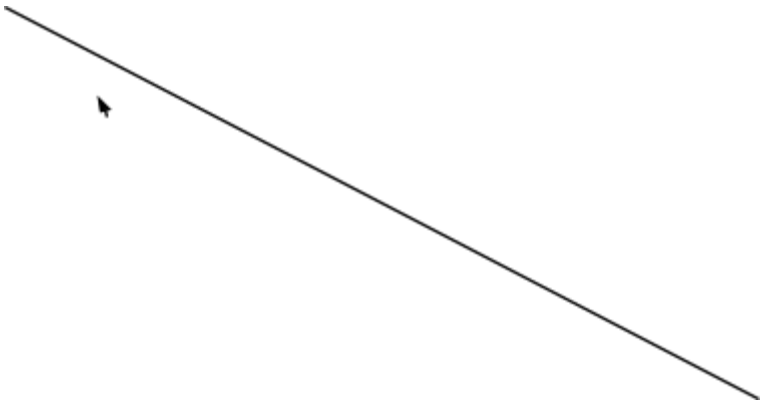
I'll come back to this later.

Ok.

You have to set the visibility to false, then set it to true afterwards. God damn macs.

When I Try To Kill Things I Want To Damn Well Kill Things:

I CAN NOT HIT ANYTHING. And this is because the bullets are inaccurate.And doing stuff like this. With the bullets going off course from the cursor.



It is doing this annoyance because because the values in bullet for x, y, xv and yv are stored as ints. To fix this I've changed them (and all location, velocity and width values in all the game parts) to doubles. This fixed it. Only problem is I have to convert them back to ints when drawing, not a problem.

It’s Not A Bug, It’s A Feature:

New feature added. Slow motion… This initialises when the player shoot the squares with the other new feature. When you hit the squares they fall apart into particles. So, slow motion was added so the user has more time to admire the pretty colors.

I don’t like waiting. This will be removed. Thread and over complication will be added.

Into The Fray:

Ok, separating the updating from the rendering into another thread has improved performance. The game runs a lot more smoothly but has a new problem. When the rendering lags like hell (and it does) it halts the drawing but not the updating, so the player has no idea what is going on. But that’s not the worst of it. There are some problems with things being semi-ready and throwing out null pointer exceptions. The entire window also seems to shake, I have no idea what this is caused by but I loath it. It seems to have something to do with keypresses.

Next step is to change as many of the ArrayLists into arrays, this should speed it up something drastic. Then I’ll try drawing to an image then drawing the image next. That’s called double buffering.

Changing the ArrayLists to arrays definitely helped. Now the only arraylists are the ones holding the parts and bullets. I am fine with this, it makes everything else easier and is not worth the optimisation.

Arrays are faster than ArrayLists if you hadn’t picked that up. It’s a simplicity thing mainly. Arrays are just a list: cell,cell,cell and a bit of information. ArrayLists however do lots of funny shit like sorting (I think) and change the size of the array they deal with constantly, have to find memory for that, have to know where everything is and so on. This is not worth it unless the benefit of being able to change it is very large.

So that fixed most of the lag. I think I’ll still try double buffering though, might fix some stuff.

It would seem I don’t know how to double buffer. It made it worse. I will come back to this.

Ok, java sucks, it seems to be running repaint asynchronously so I have to do some annoying stuff to work out how long it actually takes to render a frame, I am getting somewhere now, double buffering may be a good idea after all.

Well that was a better attempt, it's playable with double buffering but it flickers. And when I think about it double buffering isn't going to solve this, double buffering is to help with tearing and so on when the user can see the game rendering. That is not my problem. My problem is that the rendering takes too long and holds it up. I think I am better off trying to find the bottlenecks in the code because it's actually surprisingly good for what it's doing. It's only missing frames by a couple of milliseconds.

Using a VolatileImage for the frame of only the game (not the menu) helped a lot. There is noticeably less lag, the game is playable, and I can now have the menu draw less often, taking up less time.

It is working. There is very little lag. I will still need to look through and find the slowest parts.

Damn macs, making me draw everything. I can't have the menu only draw every so often, it has to draw constantly.

Come on, you can do this, just a little bit faster my little download.. . . .
. . . .
.....

I died.

I'll come back to optimization, I can't find any good (simple) profilers that I can get to work.

Changing all the doubles to floats seems to have helped a fair bit. Also, most of the lag seems to be caused by the creation of the particles, so I could gain quite a lot by changing it so that as much of that as possible is only done once, not for each particle.

Floats are better than doubles because they are smaller, requiring half the space to hold them, they are however less accurate. Still accurate enough for this.

Yep, moving as some of the particles stuff into the enemy part has helped a bit. I moved the calculations for the angle, maxspeed and all the color stuff. Now it just has to store them, not compute them over and over.

I’m resorting to a large number of System.out.println statements with System.currentTimeMillis to work out what is taking all the time in rendering. It’s all in the actual game which is good, nothing going towards the menu. And drawing the frame from the back buffer (image stored in memory) to the graphics on the panel takes 0 to 1 milliseconds. And it’s all in the drawing parts, not the background or strings. It’s in the parts, not bullets or the player, as I expected. This is going to be hard. It’s all pretty simple and can’t really have much taken out.

I’ve removed the double buffering. It doesn’t seem to be helping at all, I was right. I hate it when I’m right. Double buffering is for when you have video tearing, I don’t have that. I have plain old lag.

I’ve also changed it so that the panel has a boolean that changes between true and false when the rendering starts and finishes, this is so that I can use the JPanel’s repaint method (which works much better than grabbing the graphics object and passing that to paint) and then wait for it to finish painting before calling it again. This works quite well.

The shaking seems to be caused when I rotate the graphics object for drawing the player. Probably due to the angle being changed in the update method half way through rendering. That was it. I’ve fixed it by saving the x, y and angle as variables in the method at the start, then rotating and drawing with these rather than the global ones that are changed by the update method part way through.

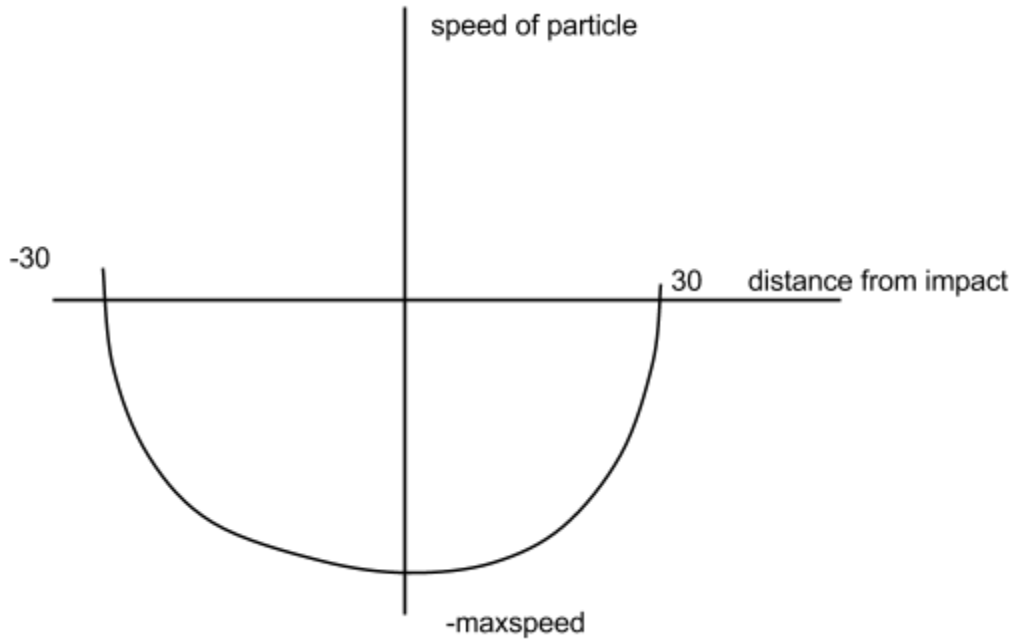
Quadratics, Graphs and Those Old Greeks:

Random is boring, a bit of a pattern is much better, so I’m adding a bit of coherence to the particle explosions. Basically, from the speed, direction and point of impact of the bullet on the part I am going to work out a good direction and speed.

For the speed, the closer the impact is to the particle the faster that particle wants to be moving. I’ve decided that a maximum speed of

$$maxspeed = bspeed / 8$$

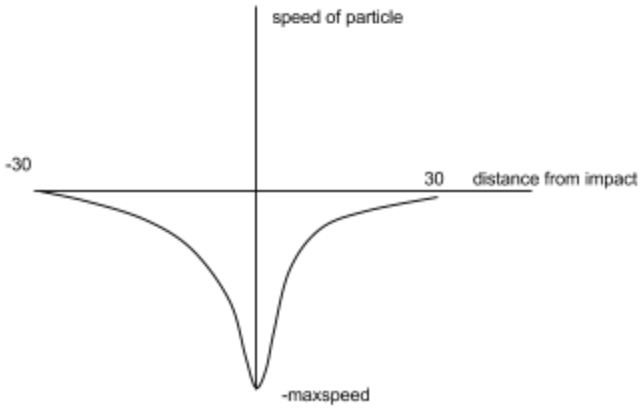
where bspeed is the speed of the bullet, is pretty good. The speed of the particle can be modeled with a graph like this, where the particle will be fastest when it is closest to the impact and will be still at 40 pixels away from the point of impact.



The equation for this is

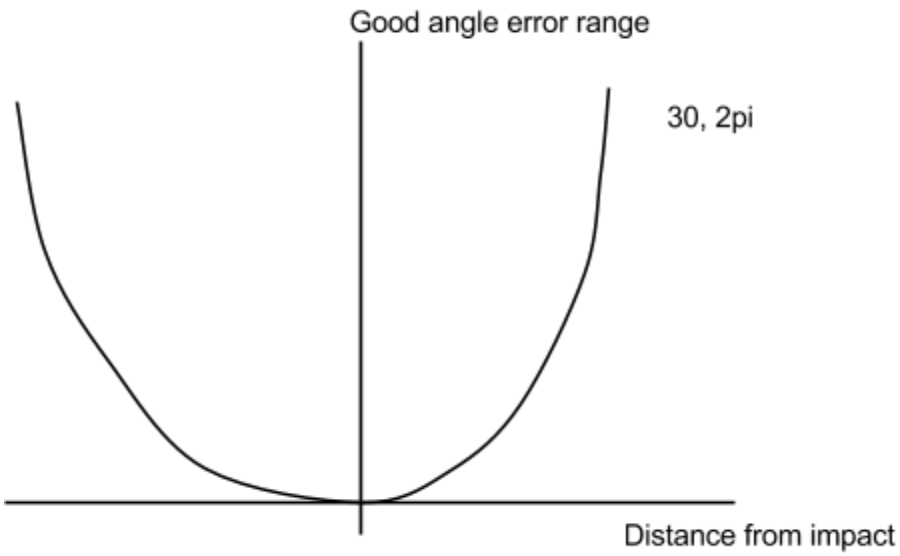
$$speed = -(maxspeed / 900) * (d - 30) * (d + 30)$$

This looks pretty good and gets some nice explosions. It isn't quite how I want it but it will do for now. I think the speed I want could be better modeled like this. But this can wait.



Ignore the bad drawings. I'm sure you get the idea.

The direction of the particles want's to be the same as that of the bullet, with an error. The error want to increase as the distance from the impact increases so that particles that are hit by the bullet move straight away from it and particles far away from the bullet go whatever direction they want to. So the angle error can be modeled with something like this:



A good enough equation for this is:

$$\theta error = (2\pi / 1600) * d^2$$

Where d is the distance from the particle to the point of impact. This can then be given as a range for a random number to sit inside of then given to the angle.

This results in acceptably awesome explosions.

Aiming is Hard:

Using the cursor to aim looks bad, unprofessional, to easy, not enough can go wrong with it. So the options are basically: remove the pointer and let them cry while they can't aim, remove the pointer and have a line like a laser for them to aim with (looks a bit childish and stupid), or replace the cursor with a crosshair type cursor. For the moment I've chosen the second one. There is no longer a cursor, they just get a line that shows where the bullets will go. This looks alright, not great, but better than it was.

To make this work a bit better I've changed it so that the mouse does not actually move anymore. When you move the mouse it takes how far it has moved vertically, gives that to the player for it to compute the angle and add that to the existing angle, then moves the mouse back to the middle of the screen. This has some challenges as the responsiveness isn't great but this can be changed by changing how far away from the player we think the mouse is.

Or I could just leave it, I think I'll just leave it for now, works better than the line because the user can change the sensitivity by moving the mouse closer to the player, makes it easier to hit the blocks close in when it matters.

Who Needs Bullets To Move Anyway?

With the addition of lasers that kill everything that hits them and don't actually move I decided I'd kill to problems with one horrible piece of code and mathematics. The other problem being that of bullet collisions not being detected when the bullet passes through the corner, never actually being contained by the rectangle that is an enemy. So to fix these I got to work, some horribly frustrating work.

I don't like work.

Okay, so what we want to do is check to see if the line made by the bullet ((x,y) to (x - xv, y - yv)) intersects with any sides of the rectangle. Using the equation for a line and some substitution and other crap we can get to some usefull stuff.

The equation for a line is:
$y = ax + c$

Lets say that a is the gradient of the line for the bullet. If the bullet's horizontal speed is 0 then the gradient can be 10000, it should be infinite. Otherwise:

$a = yv / xv$

where yv is the vertical speed of the bullet and xv the horizontal. Then

$c = y - ax$

works out the c value of the line by rearranging the equation for a line then substituting the bullets x and y locations as well as the gradient.

Now stuff gets annoying.

First we'll check the top side of the rectangle. To see if (well, when) the lines intersect we want to put the two lines equations against each other and find the x value that makes them equal:

$$ax + c = a'x + c'$$

Because the top is a horizontal line the equation for it will simply be the c of the line, this also happens to be the current y location of the part. So:

$$ax + c = c'$$

To get x we can rearrange to :

$$x = \frac{c' - c}{a}$$

This is where the lines intersect, and they will always intersect. Unless they are both parallel, in which case a would have to be zero and we would have a divide by zero error and well, crap. It hasn't happened yet so I'll leave it for now. Might fix that latter. With this we can check to see if it is between the left and right side of the rectangle, if it is then we have an intersection my friend so we say that the bullet hit the triangle.

This same method will work for the bottom side also, we just have to set c' to y + height rather than y.

The left and right side however are more fickle. Due to the fact that they are vertical the x location of the intersection will always be the same. So instead of all that annoying stull we'll just put an x location, let's say the left side, into the equation for the bullet and see what y location we get out of it. Like so:

$$y = ax + c$$

where x is the left side. Then we compare this y location to the top and bottom of our rectangle. If it's between these then we have an intersection.

COOOL.

But broken.

Why is it broken? Because nothing is simple. The problem is that by putting in the location of our part into the equation for the line we work out where the bullet will be at that x or y value, but is that where it is now? Sometimes. To fix this before each equation we check to see if the bullet is actually near that point. So I added a few variables:

$$px1 = x$$
$$py1 = y$$
$$px2 = x - xv$$
$$py2 = y - yv$$

Where x and y are the location of the bullet and xv and yv are its velocity. Then we need to check them and possibly swap them so if px1 is greater than px2 we swap them around so that px1 is less than px2. Same goes for the y ones. Then lets say for the top side we first check if py1 is less than the top and the top is less than py2. If it is then we do our equation and work shit out.

Now it works. Fantastically. It also works for the vertical laser that destroys everything without even moving. Couldn't have done that by checking to see if its location was inside of out square.

Sound To Harm The Ears:

I had nothing else to add and the game was rather non-immersive. So I've added sounds. They definitely make the game more interesting and give a lot more feedback, makes shooting things even more satisfying.

Currently most of them I've made by myself by just recording a sound I make. A few of them (the default bullet and friend shot) are that plus some editing in audacity. For the bullet I think it was mainly just getting rid of spare space. The friend shot however was a bit more interesting. I wanted a sound that would make the player think it possibly isn't a great idea to shoot this thing. I originally had it as a recording of me whistling because I couldn't think of anything better. With this I noticed that it created a horrible distorted scream sound. This was a good starting point. To make this even worse I recorded the playback adding to the distortion. This worked pretty well. Then I used audacity to make it fade out and whalla, a sufficiently horrible sound to stop people from shooting their own friends.

Background music however is proving hard. I tried to make some with generated sounds but it simply doesn't work. So next I tried to create my own sounds by making a drum set out of pieces of paper or fabric stretched over cups and cd container lids. This didn't work either. So now I'm coming back whimpering to the internet to find some royalty free sound samples. Let's see what horribleness I can create with these new resources.

Okay, that's pretty good. A short drum repeating every 0.23 seconds with a louder distorted drum every four of those. Then a loud distorted bass going

with that and then repeated a bit after so that there is never any silence.
Not bad at all.