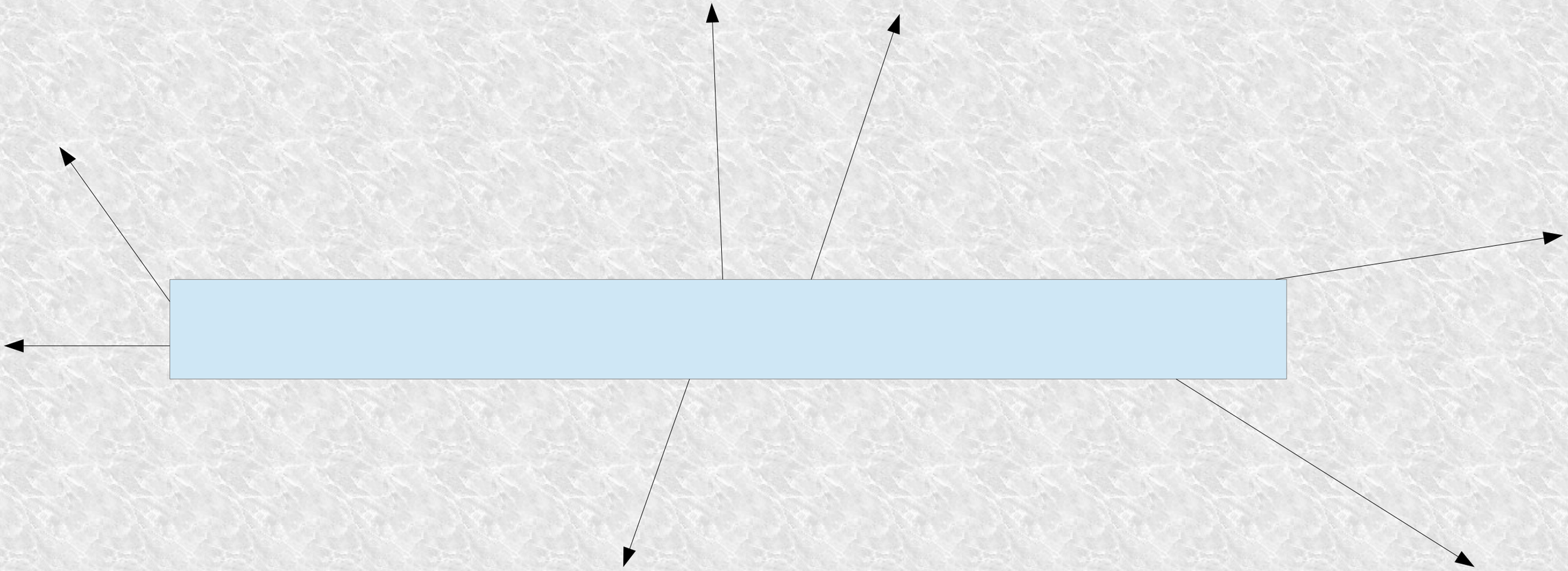
Ready Aim Fire

The Aiming Block



A draggable block will be an aiming block

Instead of just reversing direction, the direction depends on where you aim

- If the ball hits the center of a side, it bounces back perpendicular to that side
- The further the ball is from the center, the more to that side the ball bounces off
- The x and y speeds are added together, and that total gets divvied up depending on how far the ball bounces from the center
- A bounce to the center means all the speed goes in one direction only – x or y
- Otherwise, the x and y get part of the total speed

Changes to Ball

We changed Ball and moved the bounce code into 2 methods

- screenBounce
- blockBounce

We have 2 type of block bounces, so we will be maing two new methods

- normalBounce
- imBounce
- blockBounce will call one of these

normalBounce()

This one is easy, just move code from the old blockBounce

```
// bounce off of a normal block.  Just the direction changes
void normalBounce(Block block) {
  // see if we are closes to an x side of the block (left, right) or a y side (top, buttom)
  double closestX = min(x - block.position.left, block.position.right - x);
  double closestY = min(y - block.position.top, block.position.bottom - y);
  if (closestX < closestY) {
    // we are closest to the left/right of the block, so we hit a vertical edge
    speedX = -speedX;  // reverse x direction
  } else {
    // we are closest to the top/bottom of the block, so we hit a horizontal edge
    speedY = -speedY;  // reverse y direction
  }
}
```

If we are inside a normal (non-draggable) block, this method changes the direction as before.

aimBounce

This method is called if we are inside a draggable block

Start with this.

```
// bounce off of an aiming block.  You adjust the direction by where the ball
// hits the block
void aimBounce(Block block) {
  // see if we are closes to an x side of the block (left, right) or a y side (top,
buttom)
  double closestX = min(x - block.position.left, block.position.right - x);
  double closestY = min(y - block.position.top, block.position.bottom - y);
  double totalSpeed = speedX.abs() + speedY.abs();  // we divide up the speed by how
the block is hit
  … more code ...
  }
```

So far, the method is just like the other one. We need to know which side of the block we are closes to before calculating how our direction changes because of the bounce.

Notice we also get the total speed that we will divvy up later.

We now need to figure out which side we are closes to – top, left, bottom, right

This is because the directions change differently for each side

So we test for all 4 conditions

```
if (closestX < closestY) {
  // we are closest to the left/right of the block, so we hit a vertical edge
  if (x - block.position.topLeft.dx < block.position.topRight.dx - x) {
    // we hit the left side of the block
  } else {
    // we hit the right side of the block
  }
} else {
  // we are closest to the top/bottom of the block, so we hit a horizontal edge
  if (y - block.position.topLeft.dy < block.position.bottomLeft.dy - y) {
    // we hit the top of the block
  } else {
    // we hit the bottom of the block
  }
}
```

We have our 4 possibilities – now we add the code that changes the direction

Hitting the Left Side

Here is what we do if we bounce off the left side of the block

```
// make an aim number where the ball hit the left side of
// the block.  -1 means top left, 0 is middle left, 1 is bottom left.
// This number determines how much of the speed goes in the y direction.
double middle = (block.position.top + block.position.bottom) / 2;
double aim = (y - middle)/block.position.height*2;
speedY = totalSpeed * aim;  // ball can go up or down depending on aim
speedX = -(totalSpeed - speedY.abs()); // ball goes to the left
```

We get the y position that is in the middle of the top and bottom

If we bounce off the very top, aim will be 1

If we bounce off the very bottom, aim will be -1

If we bounce off the middle, aim will be 0

We will do it with an example

We have a block where the top is at y=100 and the bottom is at y=200

The middle will be 150 (100 + 200) / 2

So the height of the block is 100

We bounce off the very top left of the block: y=100

```
aim = (y - middle)/block.position.height*2;
aim = (100 - 150)/100*2;
aim = -50)/100*2;
aim = -1
```

Try it when you bounce off the middle of the left side (y = 150)
Or the bottom (y=200)

Anywhere between will always give you a number between -1 and 1

If we multiply the total speed by that number, we givethe x speed more the further away from the middle we are. If we are in the middle, x speed gets 0 and the rest goes to y
So you bounce back just to the right

So if we bounce off the top left, the direction is totally up (y is negative and gets the entire speed

If we bounce off the bottom left, the direction is totall down (y is positive and gets the entire speed)

If we bounce off the left middle, aim is 0 so y speed is 0 and it all goes to -x (goes completely to the left)

First we calculate the y speed, which can be positive or negative

Then we make it positive and subtract that from total speed to get what is left over

The left over speed goes to x

See if you can figure out the formulas for all the other sides we bounce off of

Each one has the same reasoning as the one we just did – just different directions

Here is the code for the other 3 sides of the block we can bounce off of

- Right side

```
double middle = (block.position.top + block.position.bottom) / 2;
double aim = (y - middle)/block.position.height*2;
speedY = totalSpeed * aim;  // ball can go up or down depending on aim
speedX = totalSpeed - speedY.abs(); // ball goes to the right
```

- Top side

```
double middle = (block.position.left + block.position.right) / 2;
double aim = (x - middle)/block.position.width*2;
speedX = totalSpeed * aim;  // ball can go left or right depending on aim
speedY = -(totalSpeed - speedX.abs()); // ball goes up
```

- Bottom side

```
double middle = (block.position.left + block.position.right) / 2;
double aim = (x - middle)/block.position.width*2;
speedX = totalSpeed * aim;  // ball can go left or right depending on aim
speedY = totalSpeed - speedX.abs(); // ball goes down
```

That's It

If you had trouble adding all of this to aimBounce(), check the code out here.

https://github.com/shawnlg/flutter_ball/tree/155_aim_block

All that's left is to fix up blockBounce to call the right methods

Replace the code that checks for the closest side and does the bounce with this logic.

```
// bounce off of block
if (block.draggableBlock == false) {
  normalBounce(block);
} else {  // a draggable block is one you use to aim with
  aimBounce(block);
}
```

Everything else stays the same. We still move outside of the block.

Run the game with the same blocks you had before. You can make the aim block wider so it is easier to use.

Next Time

We have just about everything for a complete game

We will need to add a few more things

- A game controller that starts and ends a game
- Text boxes to display information about the game
- Help text
- Different game levels