Game Levels

makeLevel()

This is the method that tests which level we are on and creates it

We are going to

- Make convenience methods to add blocks
- Make better level messages which explain to the user what they need to do for each level
- Come up with 10 levels with different types of game play

Instead of the simple level splash screen that just says the level number, we will add a second text box with details about the level

We will define a constant list of strings to contain this help text.

Level Messages

Add this constant list to makeLevelSplashScreen()

```
const List<String> LEVEL_TEXT = [
  "Practice aiming your ball. Hit the block by launching the ball at it.",
  "Practice aiming your ball using the red aim block. Hit the block 3 times.",
  "Break all of the blocks.",
  "Break all of the blocks. Watch out for the bottom!",
  "Break all of the blocks. They don't break so easily",
  "Break the square.",
  "Break the square. Can you do it without falling off the screen?",
  "Let's speed things up.",
  "Let the ball do the work.",
  "Time to face the challenge.",
];
```

Feel free to add to the text. Each level has its corresponding string in the list.

Display the Level Constant Text

Add a second text box under the one that displays the level number.

```
 style = TextStyle(fontSize: 12, color: Colors.blue,);
  span = TextSpan(text: LEVEL_TEXT[game.level - 1], style: style);
  textBox = TextDraw(Rect.fromLTWH(0, 50, gp.sizeX, 50), span,
    boxColor: null, borderColor: null,
  );
  game.add(textBox);
}
```

Try it out. Each level 1-10 should display its own text below the level title.

Next we add some more convenience methods

addBlockExact()

We already have a method addBlock() that lets us easily add game blocks

Everything is given in terms of screen width. If you want 10 blocks to fit width-wise on the screen, each one will have width 0.1

For more exact placement of blocks, we make a similar method that take pixels

```
// add a block using pixels
void addBlockExact(FlutterballGame game, GamePlay gp, double x, double y, double
width,
    {Color color : Colors.blue, int lives : 10, }) {
  Rect position = Rect.fromLTWH(x, y, width, width);
  Block block = Block(game, position: position,
    color: color, borderColor: Colors.black,
    draggableBlock: false, lives: lives,
  );
  game.add(block);
}
```

Aiming Block

We have another method for adding the aiming block. It is similar to our first method

It uses screen width and height in its calculations instead of pixels.

```
void addAimBlock(FlutterballGame game, GamePlay gp, double x, double y, double width, 
double height,
    {Color color : Colors.red, int lives : 10, }) {
  Rect position = Rect.fromLTWH(x*gp.width, y*gp.height, width*gp.width, 
height*gp.height);
  Block block = Block(game, position: position,
    color: color, borderColor: Colors.black,
    draggableBlock: true, lives: lives,
  );
  game.add(block);
}
```

The Levels

To add variety to the levels, we will use these game features in the various levels

- Number and placement of game blocks
- With and without aiming block
- Size of aiming block
- Different values for speedScale

  0.0 means the player chooses the speed at launch

  > 0.0 means the game has a specified speed
- Number of balls and number of bounces per ball
- Which sides are ignored

Our makeLevel() method will look like this:

```
void makeLevel(FlutterballGame game, GamePlay gp) {
  game.clearComponents();
  gp.speedScale = 0.0; // default
  switch (game.level) {
    case 1: // one block to hit with launcher, no aiming block
      break;
    case 2: // one block to hit using aiming block
      break;
    case 3: // one row of blocks and an aiming block
      break;
    case 4: // one row of blocks but ignore bottom
      break;
    case 5: // 1 row of blocks - 3 lives each
      break;
    case 6: // square of blocks
      break;
    case 7: // square no sides
      break;
    case 8: // speed it up
      break;
    case 9: // several rows
      break;
    case 10: // face
      break;
    default:
  }
```

Levels in General

In a real game, there may be hundreds or even thousands of levels

Many of the levels may be computer-generated

You could write methods to generate a certain type of level with various levels of difficulty to use them in different levels

Be careful with random blocks

Levels can look messy and boring with randomly-scattered blocks

They may be impossible to complete

We are hard-coding 10 levels that can be used as examples

Level 1

This lets you practice aiming with the launcher. You have 5 tries to hit the block since your ball will not bounce.

```
case 1: // one block to hit with launcher, no aiming block
  addBlock(game, gp, 0.4, 0.0, 0.1, lives: 1);
  gp.ballsLeft = 5;
  gp.ballBounces = 1;
  gp.ignoreTop = false;
  gp.ignoreBottom = false;
  gp.ignoreLeft = false;
  gp.ignoreRight = false;
  break;
```

Level 2

This level helps you practice aiming with the aiming block. You have 1 ball with lots of bounces to try and hit the block.

```
case 2: // one block to hit using aiming block
  addBlock(game, gp, 0.4, 0.0, 0.1, lives: 3);
  addAimBlock(game, gp, 0.4, 0.5, 0.2, 0.05, lives: 10);
  gp.ballsLeft = 1;
  gp.ballBounces = 25;
  gp.ignoreTop = false;
  gp.ignoreBottom = false;
  gp.ignoreLeft = false;
  gp.ignoreRight = false;
  break;
```

Level 3

This level gives you a row of blocks you have to break. Your aiming block has a limited number of uses, so you need to aim carefully.

```
case 3: // one row of blocks and an aiming block
  addBlock(game,gp, 0.0, 0.0, 0.1, lives: 1);
  addBlock(game,gp, 0.1, 0.0, 0.1, lives: 1);
  addBlock(game,gp, 0.2, 0.0, 0.1, lives: 1);
  addBlock(game,gp, 0.3, 0.0, 0.1, lives: 1);
  addBlock(game,gp, 0.4, 0.0, 0.1, lives: 1);
  addBlock(game,gp, 0.5, 0.0, 0.1, lives: 1);
  addBlock(game,gp, 0.6, 0.0, 0.1, lives: 1);
  addBlock(game,gp, 0.7, 0.0, 0.1, lives: 1);
  addBlock(game,gp, 0.8, 0.0, 0.1, lives: 1);
  addBlock(game,gp, 0.9, 0.0, 0.1, lives: 1);
  addAimBlock(game,gp, 0.4, 0.5, 0.2, 0.05);
  gp.ballsLeft = 3;
  gp.ballBounces = 25;
  gp.ignoreTop = false;
  gp.ignoreBottom = false;
  gp.ignoreLeft = false;
  gp.ignoreRight = false;
  break;
```

Level 4

At this point, the levels are getting more complicated

It doesn't make sense to copy the code from the screen

Get the code here

https://github.com/shawnlg/flutter_ball/tree/21_levels

This level is similar to level 3, but the bottom is ignored, so you need to keep the ball from dropping off the screen

Level 5

This level increases the number of lives for each block.

Be sure to play each level you create so that you know it is possible to win

Level 6 and 7 - shapes

This level makes a square shape from the blocks

Notice we have helper variables to position the squares in columns and rows

```
double width = 0.1 * gp.width;
double col1 = 0.35*gp.width;
double col2 = 0.45*gp.width;
double col3 = 0.55*gp.width;
double row1 = 30.0;
double row2 = row1 + width;
double row3 = row1 + 2*width;
```

If you display this level, you can see how the blocks stack on top of each other exactly using the row and column coordinates we defined.

Level 7 is like level 6 but it ignores the bottom

Level 8

This level gives a constant speed of 0.6 screen widths per second

It also ignores the bottom, so it is much harder.

When you ignore the bottom, be sure to give the aiming block a much higher life count since it needs to be used more often

Level 9

This level gives you several rows. You should launch the ball so it bounces between the rows and breaks the blocks for you.

You have control of the speed again.

Level 10

This is a fun level where you lay out the blocks in a face shape.

These type of levels scattered throughout the game make it more interesting and give the player a fun break.

You may notice that if you play on a slow emulator, the ball might do odd things

It might go through a block that you know you hit

This has to do with frame rate. If the rate is so slow that the ball actually jumps a long distance, it can actually jump across a block instead of bouncing into it

This should not happen on a phone or a good emulator like the iPhone simulator

Next Time

We are finished with the code for this course

We will discuss some possible improvements we could make to the game and to our components

Make sure the levels work for you. Think about things you would like to add to the game to make it more interesting

The code is here

https://github.com/shawnlg/flutter_ball/tree/21_levels