More Game Controller

Game States

Just like any complicated controller, we will define a bunch of states that our GamePlay component can be in

```
enum GameState {
  WAITING,  // waiting for screen size info
  STARTING, // putting up the game blocks
  SPLASH,   // showing splash screen
  LAUNCHING,// launching new ball
  PLAYING,// playing game
  BALL_OVER,  // ball is done, need to use next one
  COMPLETED, // completed level
  LOST, // no more balls but still blocks
  LEVEL, // showing level screen
  OVER, // game over screen
  DEAD, // destroy component
}
```

- SPLASH

  This allows us to put up an info screen and wait for a few seconds
- DEAD

  This is necessary when switching between the intro controller and the game controller
- LAUNCHING

  The ball launcher is up, so we only let that one drag
- Other states are for putting up certain screens and knowing where to resume the game

Here are some familiar instance variables for GamePlay

```
// instance variables
final FlutterballGame game;
GameState state = GameState.WAITING;
double width=0;  // size of the screen in the x direction
double height=0;  // size of the screen in the y direction
```

More instance variables

```
double splashOver;   // when to stop showing splash screen
```

This is a time variable. It tells the controller when it's time to take down the splash screen

```
int ballsLeft = 0;   // how many balls the player gets in the level
int ballBounces = 1;   // how many bounces each ball gets
TextDraw launchMessage;   // tell player to launch the ball
InteractiveBallReleaser launcher;   // ball launcher
double speedScale=0.0;   // speed of launch
```

These should be self-explanitory

Here's our normal constructor

```
// constructor
GamePlay(this.game, ) : super() {
}
```

We don't set instance variables like ballsLeft here, because they need to change depending on which level we are on.

Easy Methods

```
void render(Canvas c) {
}
```

We don't draw anything directly in our controller

```
bool destroy() => state == GameState.DEAD;
```

We've seen this before

```
void resize(Size size) {
  if (size.width <= 0) return;

  // save screen width and height
  width = size.width;
  height = size.height;
  state = GameState.STARTING;
}
```

And thhis

Splash Screens

We will put all of our splash screens in one place

Make a file called game_text.dart in the same folder as the game.

Not a game component

It's basically a bunch of functions that put up various screens

```
const double SPLASH_TIME = 5.0;

void addLaunchMessage(FlutterballGame game, GamePlay gp) {
  TextStyle messageStyle = TextStyle(fontSize: 15, color: Colors.white);
  TextSpan messageSpan = TextSpan(text: "Level ${game.level} Launch...", style: 
messageStyle);
  gp.launchMessage = TextDraw(Rect.fromLTWH(0, gp.height*0.4, gp.width, 50), 
messageSpan,
    boxColor: null, borderColor: null,
  );
  game.add(gp.launchMessage);
}
```

This message tells you to launch a ball

This one puts up the level finished screen

```
// set up splash screen for completed level
void makeCompletedSplash(FlutterballGame game, GamePlay gp) {
  game.clearComponents();
  TextStyle messageStyle = TextStyle(fontSize: 20, color: Colors.blue);
  TextSpan messageSpan = TextSpan(text: "Good Job!\n\n\nYou Finished\nLevel ${game.level}",
style: messageStyle);
  TextDraw message = TextDraw(Rect.fromLTWH(0, 0, gp.width, gp.height), messageSpan,
    boxColor: null, borderColor: null,
  );

  game.add(message);
  gp.splashOver = game.currentTime() + 3.0;
}
```

And one for losing the game

```
// set up splash screen for lost game
void makeLoseSplashScreen(FlutterballGame game, GamePlay gp) {
  game.clearComponents();
  TextStyle messageStyle = TextStyle(fontSize: 20, color: Colors.blue);
  TextSpan messageSpan = TextSpan(text: "You Lose!\n\n\nGame Over", style: messageStyle);
  TextDraw message = TextDraw(Rect.fromLTWH(0, 0, gp.width, gp.height), messageSpan,
    boxColor: null, borderColor: null,
  );
  game.add(message);
  gp.splashOver = game.currentTime() + SPLASH_TIME;
}
```

Levels

We'll make another file that has the code for making all of the levels. We're just making one for now. Make file game_levels.dart in the same folder as the game

We have a level splash screen

```
// set up splash screen for level
void makeLevelSplashScreen(FlutterballGame game, GamePlay gp) {
  game.clearComponents();

  TextStyle style = TextStyle(fontSize: 20, color: Colors.blue,);
  TextSpan span = TextSpan(text: "Level ${game.level}\n", style: style);
  TextDraw textBox = TextDraw(Rect.fromLTWH(0, 0, gp.width, gp.height), span,
    boxColor: null, borderColor: null,
  );
  game.add(textBox);
}
```

A handy function for creating blocks

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

And a function for creating a game level

Just one level – a very simple one

```
// set up game for a particular level
void makeLevel(FlutterballGame game, GamePlay gp) {
  game.clearComponents();
  addBlock(game, gp, 0.4, 0.0, 0.1, lives: 1);
  gp.ballsLeft = 5;
  gp.ballBounces = 5;
}
```

Back to the Controller

We will make a method in GamePlay that checks what is going on in the game.

It checks which components are on the screen

- If there are blocks but no ball, the player must have lost
- If there is a ball but no blocks, player completed a level
- If there are blocks and a ball, game is still being played
- It is called every frame and sets the state of the game based on which components are there

Create the method

```
// while game is playing, see if level is over and set status accordingly
// - player wins level
// - player loses level
// - player needs to launch new ball
void checkPlay() {
}
```

We loop through all the components seeing if we have balls or blocks in the game

```
// check for blocks and balls on screen
Block block; // found a block that ball can bounce on
Ball ball;  // found a ball that is bouncing
```

```
game.components.forEach((c) {
  if (c is Block) {
    if (!c.draggableBlock) {
      block = c;  // found a game block
    }
  } else if (c is Ball) {
    ball = c;
  }
});
```

We see if the player has cleared a level

```
if (block == null) {
  // level cleared
  makeCompletedSplash(game,this);
  if (game.level < MAX_LEVELS) {
    game.level++;  // go to next level
    state = GameState.COMPLETED;  // wait for splash
  } else {
    // reached last level
    state = GameState.OVER;
  }
}
```

Next, since there must be blocks still on the screen, we check if we are out of balls – lose game

```
else if (ball == null && ballsLeft <= 0) {  // no more balls left
  state = GameState.LOST;
  makeLoseSplashScreen(game,this);
```

Finally, we check if the ball is gone but the player still has more balls. In this case, we can launch another ball on the same level.

```
} else if (ball == null) {  // still balls left to launch
  // need to launch another ball, but don't put up a splash screen
  state = GameState.BALL_OVER;  // launch new ball
} else {
}
```

Update() will react to any of these conditions when it checks what state the game is in

Check for Ball Launch

We add the InteractiveBallLauncher whenever the player needs to start a new ball

While the user is launching, we must not allow any blocks to be dragged

We can tell that a ball has launched by checking if there is a ball in the game

Add this method.

```
// check if ball has been launched yet
bool checkLaunch() {
  bool launched = false;
  game.components.forEach((c) {
    if (c is Ball) {
      Ball ball = c;
      launched = true;  // found a bouncing one
    }
  });
  return launched;
}
```

The update() Method

This method is called once per frame.

It basically reacts to state changes

It is a giant switch statement which tests for each game state.

```
void update(double t) {
  switch (state) {

    default:
  }
}
```

We'll go over what to do for every game state

State STARTING

This state is when the controller is just added to the game

- When the start button is pressed
- After user completes a level
- Put up the level start splash screen for a few seconds
- Change the state to SPLASH so the game waits until the screen needs to be taken down

```
case GameState.STARTING:
  // put up level splash screen
  makeLevelSplashScreen(game,this);
  splashOver = game.currentTime() + SPLASH_TIME;
  state = GameState.SPLASH;
  break;
```

State SPLASH

We wait until it's time to take down the splash screen

When it's time to take it down, we

- create the game level (the various blocks)
- add the ball launcher so the player can launch a ball
- set the state to LAUNCH, indicating that the user needs to launch

We do the same thing for state BALL_OVER, except the game level is already on the screen, so we don't need to add that.

So we combine both states into one test

```
case GameState.SPLASH:
case GameState.BALL_OVER:
  if (game.currentTime() > splashOver) {
    // put up level screen if we just put up splash screen
    if (state == GameState.SPLASH) {
      makeLevel(game,this);
    }

    // add component that launches the ball
    launcher = InteractiveBallReleaser(game, speedScale: speedScale, lives:
ballBounces);
    game.add(launcher);
    addLaunchMessage(game,this);
    state = GameState.LAUNCHING;
  }
  break;
```

Notice that we can have 2 case lines together. This means the code will be done for either if the cases – state is SPLASH or BALL_OVER

State LAUNCHING

Here we use our check method to see if the launch is finished

```
case GameState.LAUNCHING:
  // see if ball launched yet
  if (checkLaunch()) {
    ballsLeft--;
    launchMessage.lives = 0;  // remove launch message
    launcher.lives = 0;  // remove launcher
    state = GameState.PLAYING;
    print("state = playing");
  }
  break;
```

If launching is over, we kill the ball launcher component and the launch message that was showing on the screen.

We reduce the number of balls left since the player just used one

We start playing the game – state = PLAYING

State PLAYING

We wrote a method to check for what is going on while playing the game, so all we have to do is call it. As long as the game is playing (there are blocks and a ball), nothing happens. The state stays the same.

```
case GameState.PLAYING:
  checkPlay();  // check if anything is over
  break;
```

State COMPLETED

This happens when a level is over

We had put up a level over splash screen

We wait for the splash time to be up

Then we restart the game

```
case GameState.COMPLETED:
  if (game.currentTime() > splashOver) {
    // restart next level
    state = GameState.STARTING;
  }
  break;
```

State LOST or OVER

Either way, the game is done. LOST means you ran out of balls. OVER means you completed all the levels.

The only difference is the splash screen that is displayed

We wait for the time for the splash screen to be removed

We then add the game intro controller and kill the game controller one

This puts the player back at the beginning with all the bouncing balls and buttons.

```
case GameState.LOST:
case GameState.OVER:
  if (game.currentTime() > splashOver) {
    // done showing lose screen, go back to start screen
    game.clearComponents();
    GameIntro gameIntro = GameIntro(game);
    game.add(gameIntro);
    state = GameState.DEAD;
  }
  break;
```

Try the Game

You should be able to play the game now

All the levels are the same

Not much text yet

Next time we will improve the screens

Finally, we will create 10 different levels

The code can be found here

`https://github.com/shawnlg/flutter_ball/tree/18_game_controller`