



## PART 1: GAME MECHANICS

@UnityCourse  
[facebook.com/UnityCourse](https://facebook.com/UnityCourse)



## Introduction To BowlMaster



### What BowlMaster Teaches

- Manipulating a 3D world.
- Test driven development.
- Introducing Unity 5.



## About the Game Design Doc

- This part of the section notes is a simple reference for the important specifications of the game.
- This is just for reference, we'll refer to it as needed during the videos.

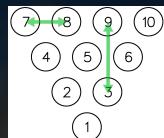
## Bowling Pin Specification

- Maple wood. Density about  $0.6 \text{ g cm}^{-3}$
- Mass 1.53 kg (3 lbs 6 oz).
- 38.0 cm (15 inches) tall.
- 12.1 cm (4.75 inches) at their widest point.

[http://en.wikipedia.org/wiki/Bowling\\_pin](http://en.wikipedia.org/wiki/Bowling_pin)

## Bowling Pin Layout

- 30.48 cm (12 inches) apart sideways (7-8)
- 52.71 cm (20.75 inches) every 2 rows (9-3)



## Ball Specification

- Mass <= 7.3 kg (16 lbs).
- Density <=  $3.80 \text{ g cm}^{-3}$
- But some float\* so call it  $1 \text{ g cm}^{-3}$
- Diameter: (21.59 to 21.83) cm (call it 21.7 cm)

\*[https://www.youtube.com/watch?v=FeKb\\_xfr608](https://www.youtube.com/watch?v=FeKb_xfr608)

## Bowling Lane Specifications

- 1829 cm from the foul line to the head pin.
- Make it 2000 cm long in total.
- 105 cm wide.
- Give some height e.g. 1cm

[http://en.wikipedia.org/wiki/Ten-pin\\_bowling](http://en.wikipedia.org/wiki/Ten-pin_bowling)

## **Rules 1 of 4**

The game consists of 10 frames as shown above. In each frame the player has two opportunities to knock down 10 pins. The score for the frame is the total number of pins knocked down, plus bonuses for strikes and spares.

---

---

---

---

---

## **Rules 2 of 4**

A spare is when the player knocks down all 10 pins in two tries. The bonus for that frame is the number of pins knocked down by the next roll. So in frame 3 above, the score is 10 (the total number knocked down) plus a bonus of 5 (the pins knocked down on the next roll.)

---

---

---

---

---

## **Rules 3 of 4**

A strike is when the player knocks down all 10 pins on his first try. The bonus for that frame is the value of the next two balls rolled.

---

---

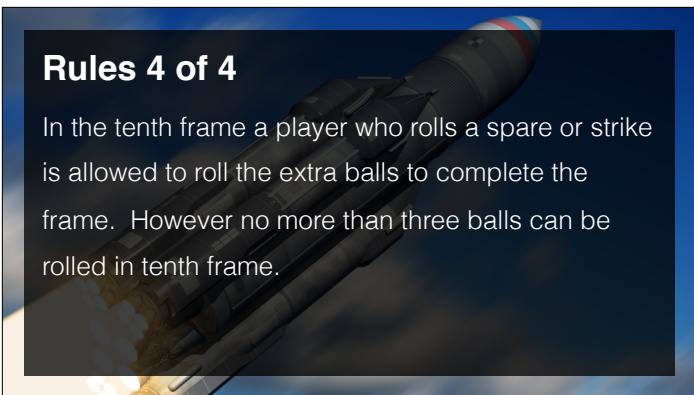
---

---

---

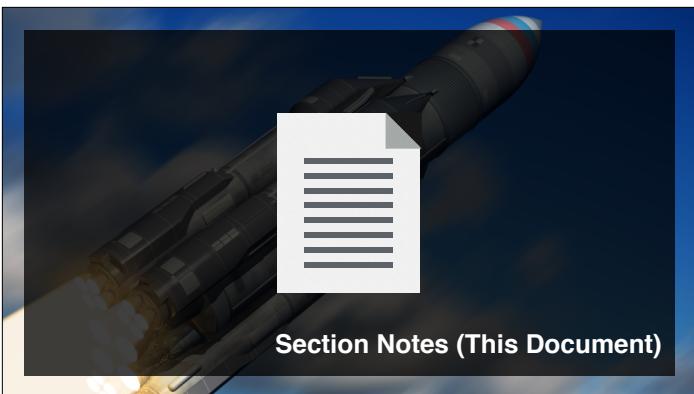
## Rules 4 of 4

In the tenth frame a player who rolls a spare or strike is allowed to roll the extra balls to complete the frame. However no more than three balls can be rolled in tenth frame.



## Your BowlMaster Assets

A screenshot of a Udemy course page titled "Your BowlMaster Assets". The page features a video thumbnail of two men, a "View resources" button circled in red with a number 1, and a download menu with a "Downloads Here" link circled in red with a number 2. A large yellow download icon is at the bottom right.



Section Notes (This Document)



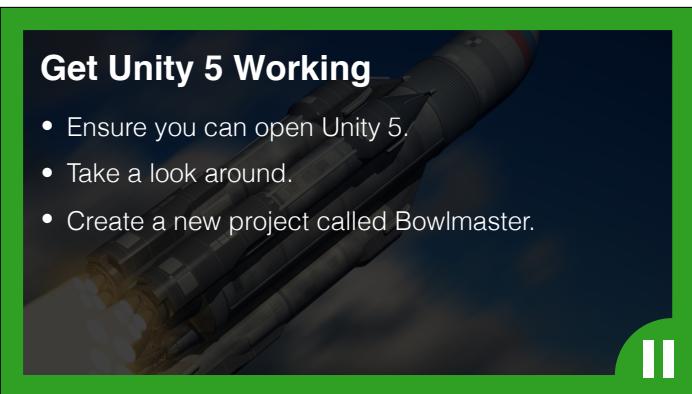
## Time To Install Unity 5

@UnityCourse  
[facebook.com/UnityCourse](https://facebook.com/UnityCourse)



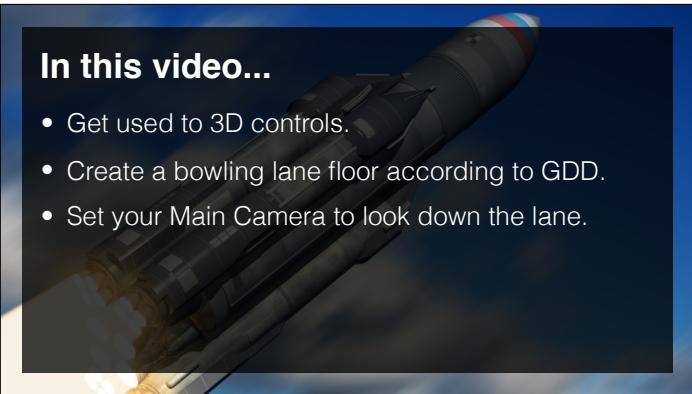
### In this video...

- How to keep Unity 4.6.3 as well.
- Installing the latest Unity 5.
- Backup before upgrading projects.
- A brief tour of what's different\*



### Get Unity 5 Working

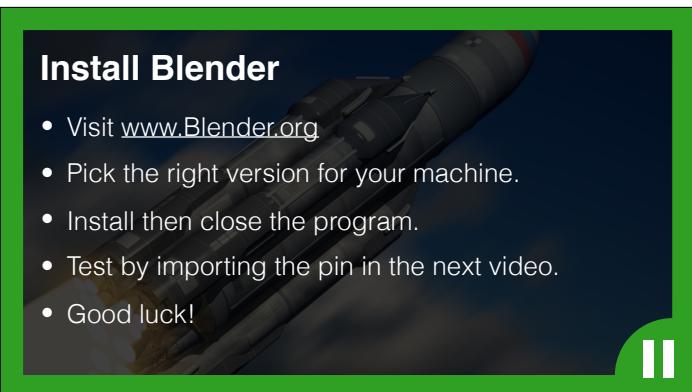
- Ensure you can open Unity 5.
- Take a look around.
- Create a new project called Bowlmaster.





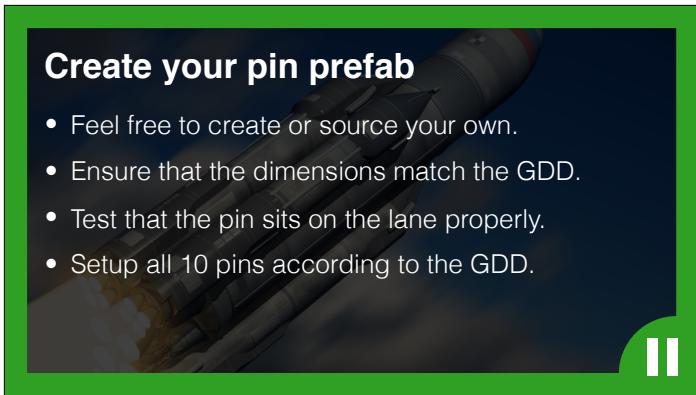
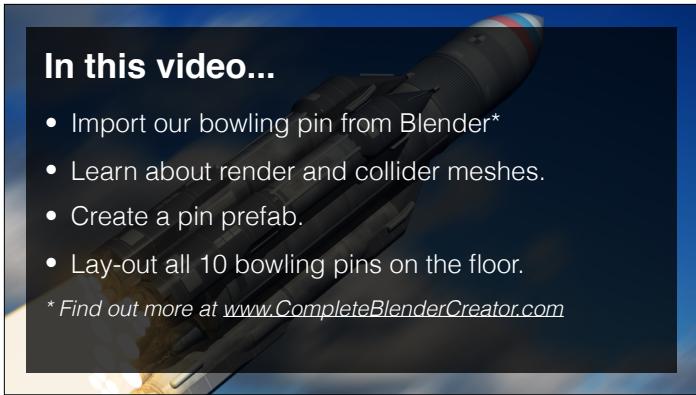
## In this video...

- Blender is a 3D art program.
- We need it to import **.blend** files.
- It's amazingly powerful, but tricky to get started.
- You only need to install for this course, not use.
- [www.CompleteBlenderCreator.com](http://www.CompleteBlenderCreator.com) to learn more.



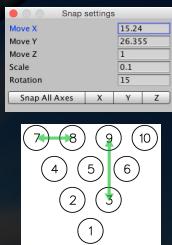
## Install Blender

- Visit [www.Blender.org](http://www.Blender.org)
- Pick the right version for your machine.
- Install then close the program.
- Test by importing the pin in the next video.
- Good luck!



## You could also use snap-to-grid

- Go to Edit > Snap Settings
- Use the settings shown here =>
- This is 1/2 of the pin spacing
- $30.48 / 2 = 15.24$  X spacing
- $52.71 / 2 = 26.355$  Y spacing



## 3D Sphere As Bowling Ball

@UnityCourse  
[facebook.com/UnityCourse](https://facebook.com/UnityCourse)

### In this video...

- Create a 3D sphere of the right size.
- Apply a temporary texture so we can see it roll.
- Add a sphere collider.
- Add a Rigidbody (3D) and set properties.
- Make the ball roll down the lane.

## Ball Specification

- Mass <= 7.3 kg (16 lbs).
- Density <=  $3.80 \text{ g cm}^{-3}$
- But some float\* so call it  $1 \text{ g cm}^{-3}$
- Diameter: (21.59 to 21.83) cm (call it 21.7 cm)

\*[https://www.youtube.com/watch?v=FeKb\\_xfr608](https://www.youtube.com/watch?v=FeKb_xfr608)

## Get the ball hitting the pins

- Give an appropriate initial velocity from script.
- Point so it passes through the pins\*
- Play a rolling sound.

\* *It won't hit them yet, that's next lecture*

## Control Camera To Track Ball

## In this video...

- Create **CameraControl.cs** for the Main Camera.
- Make the camera track the ball.
- Stop when it gets close to the headpin.

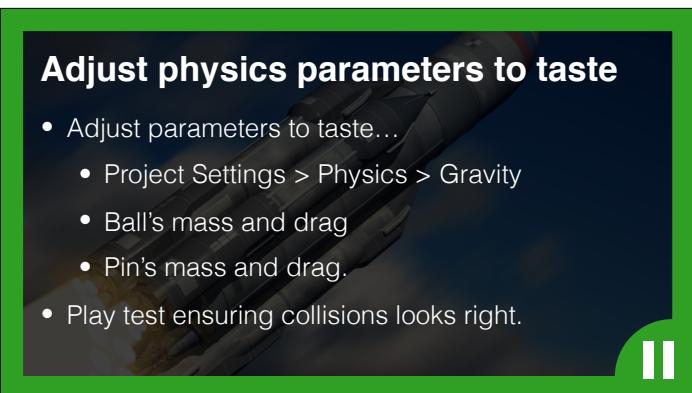
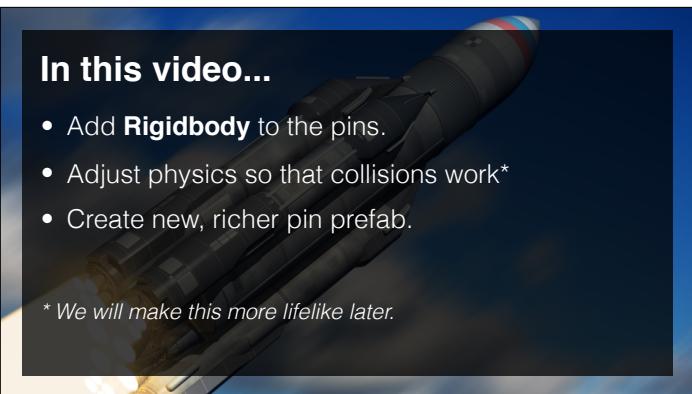
## Write CameraControl.cs script

- Keep the camera a fixed offset from the ball
- Use an **if** statement in your **Update()** loop
- Stop the camera at headpin ( $z = 1829$ )
- Test it works.

## How offset vectors work

- We want **Ball + Offset = Camera**
- Subtract **Ball** from both sides
- Gives: **Offset = Camera - Ball**
- Remember head - tail in future





## About adjusting gravity

- Sometimes in games gravity isn't realistic.
- For example fast cars jump too far if it is.
- In our case, we're using 1 World Unit = 1 cm.
- Unity default is 1 World Unit = 1 m.
- Therefore increase gravity to max  $-981 \text{ m s}^{-2}$ .

## About convex meshes

- Convex means curved like the exterior of a sphere
- Mesh Colliders must be convex to self-collide\*
- Maximum triangles in collider = 255.
- These are performance limitations of the engine

\* <http://docs.unity3d.com/Manual/class-MeshCollider.html>

## Top Camera Render Texture

## In this video...

- Create a render texture\* ready for camera output.
- Position a top-down camera looking at pins.
- Create a user interface panel.
- Embed a “Raw Image” with this render texture.

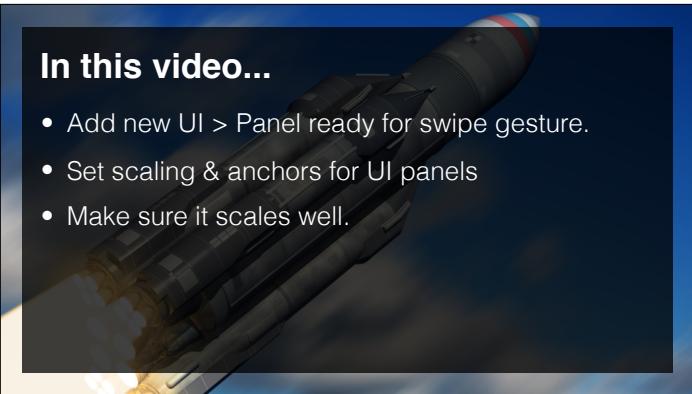
\* <http://docs.unity3d.com/Manual/class-RenderTexture.html>

## Create & position “Top View” camera

- Put the camera above the pins looking down.
- Make the pins fill about 80% of its view.
- Set “Target Texture” to TopCamera.renderTexture.
- Test game still runs (you won’t see the cam yet).

## Setup camera view UI

- Create GameObject > UI > Raw Image.
- Position it on the UI Left Panel.
- Set the Texture to “TopCamera.renderTexture”.
- Test you can see the top-down view in Game.



## In this video...

- Re-factor **Ball.cs** to create **public Launch ()**
- Write **DragLaunch.cs** component for ball.
- Wire the UI Panel's events to **DragLaunch.cs**
- Test new drag control system.

## Refactor Ball.cs

- Separate launch code into public method.
- Use signature **public Launch (Vector3 velocity)**
- **rigidBody.useGravity = false** on **Start ()**
- **rigidBody.useGravity = true** on **Launch ()**
- Test by calling from **Start ()**

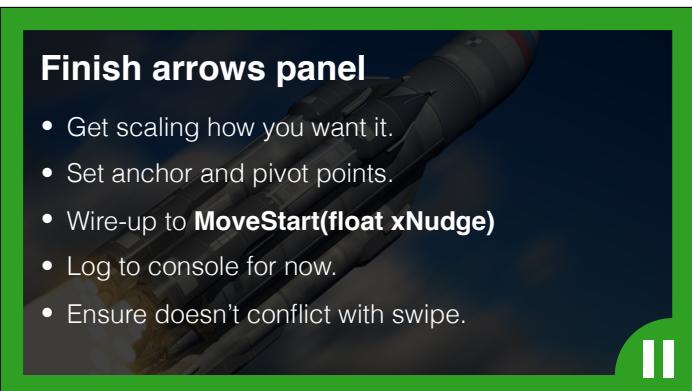
## Write DragLaunch.cs

- **public DragStart()** stores time & position.
- **public DragEnd()** launches ball.
- Launch depends on length & speed of drag.
- Should be no need for scaling factors.



### In this video...

- Add another UI panel at bottom of screen.
- Ensure this panel “occludes” the touch input.
- Add two arrows, for moving ball left and right.
- Write a simple method to allow this at start.



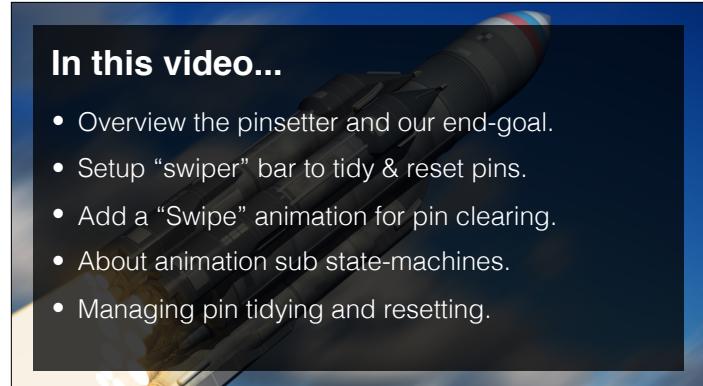
## Get start position working

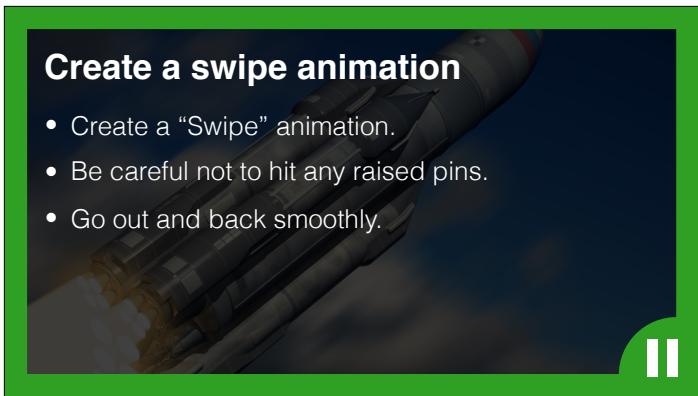
- Approach it however you like.
- Goal is to be able to drag the ball left and right.
- Only allow moving before the ball is launched.
- Clamp the ball to within the lane width.
- Test it feels right.



## In this video...

- Overview the pinsetter and our end-goal.
- Setup “swiper” bar to tidy & reset pins.
- Add a “Swipe” animation for pin clearing.
- About animation sub state-machines.
- Managing pin tidying and resetting.





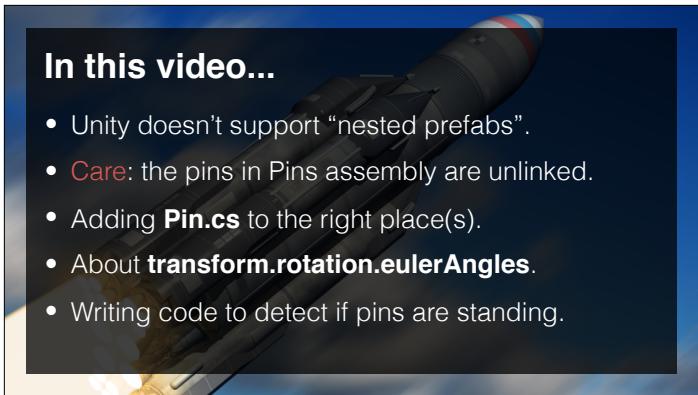
## Create a swipe animation

- Create a “Swipe” animation.
  - Be careful not to hit any raised pins.
  - Go out and back smoothly.



## Working Around Nested Prefabs

@UnityCourse  
[facebook.com/UnityCourse](https://facebook.com/UnityCourse)



## In this video...

- Unity doesn't support "nested prefabs".
  - **Care:** the pins in Pins assembly are unlinked.
  - Adding **Pin.cs** to the right place(s).
  - About **transform.rotation.eulerAngles**.
  - Writing code to detect if pins are standing.

## Detect if pins are standing

- Add **public float standingThreshold** to **Pin.cs**
- Write **public bool IsStanding ()** method
- Returns true if the pin's transform is rotated less than the threshold from vertical
- Returns false otherwise

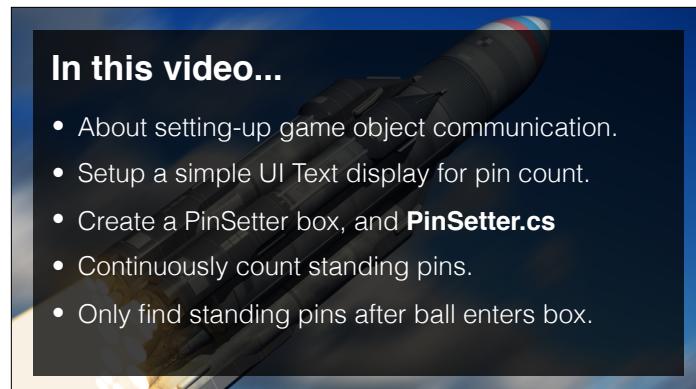


## Counting Upright Objects

@UnityCourse  
[facebook.com/UnityCourse](https://facebook.com/UnityCourse)

## In this video...

- About setting-up game object communication.
- Setup a simple UI Text display for pin count.
- Create a PinSetter box, and **PinSetter.cs**
- Continuously count standing pins.
- Only find standing pins after ball enters box.



## Count standing pins

- Write `int CountStanding()` method
- Loops through all pins in the scene
- Keeps track of number of standing pins
- Returns current number of standing pins

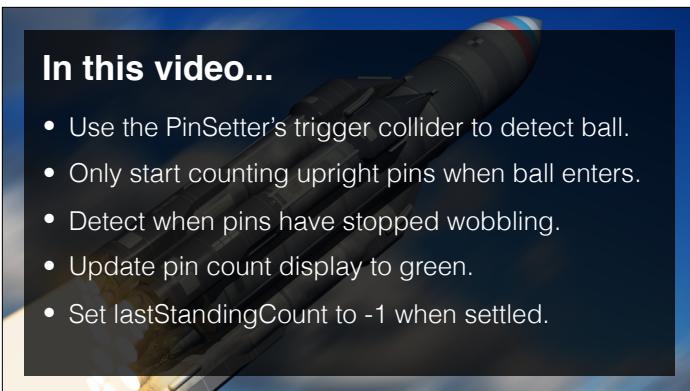


## Detecting Pins Have Settled

@UnityCourse  
[facebook.com/UnityCourse](https://facebook.com/UnityCourse)

## In this video...

- Use the PinSetter's trigger collider to detect ball.
- Only start counting upright pins when ball enters.
- Detect when pins have stopped wobbling.
- Update pin count display to green.
- Set lastStandingCount to -1 when settled.



## Setup Pin Setter Triggers

- Add to **PinSetter.cs** script.
- Set pin count to red when ball enters box.
- Set **ballEnteredBox** instance variable to true.
- Destroy pins when they leave the box.



## Write CheckStanding ()

- If **ballEnteredBox** then call **CheckStanding()**.
- Wait for standing count to stop changing for 3s.
- When pins settle call **PinsHaveSettled()**.
- Update display to green.



## Write Reset() in Ball.cs

- Create and manage **public bool inPlay;**
- Capture ball start position on **Start()**.
  - Reset position to start.
  - Set velocity & angular velocity to zero.
  - Prevent the ball falling before 2nd launch.





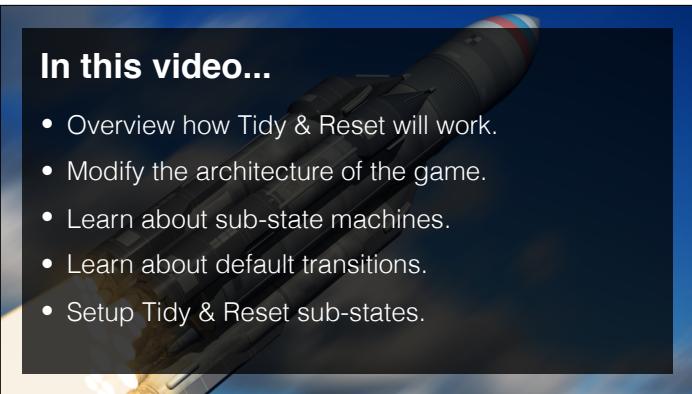
## Raise & Lower Standing Pins

@UnityCourse  
[facebook.com/UnityCourse](https://facebook.com/UnityCourse)



## Sub-states & Default States

@UnityCourse  
[facebook.com/UnityCourse](https://facebook.com/UnityCourse)

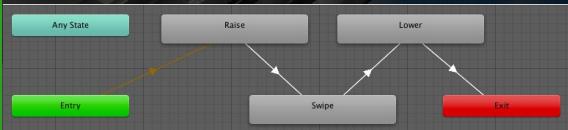


## In this video...

- Overview how Tidy & Reset will work.
- Modify the architecture of the game.
- Learn about sub-state machines.
- Learn about default transitions.
- Setup Tidy & Reset sub-states.

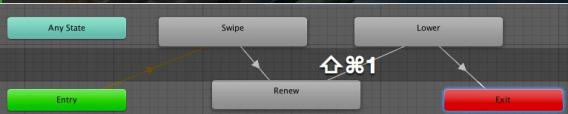
## Setup your Tidy animation

- Create Tidy sub-state, and it's inner states.
- Setup a "tidyTrigger" parameter
- Include all the following steps...



## Setup your Reset animation

- Create Reset sub-state, and it's inner states.
- Setup a "resetTrigger" parameter
- Include all the following steps...



## Calling Animator Helper Scripts

@UnityCourse  
facebook.com/UnityCourse



## In this video...

- Add temporary UI Buttons for Reset and Tidy.
- Write **RaisePins()** to lift all the standing pins up.
- Write **LowerPins()** to lower them down again.
- Test state machines work properly.

## Create Reset & Tidy UI Buttons

- Add two buttons to your UI.
  - Connect them to the Pin Setter game object.
  - Work out how to trigger animation states\*
- \* Note the UI system's *ResetTrigger* is different to ours!

## Finish RaisePins() and LowerPins()

- Flesh-out these two methods in **PinSetter.cs**.
- Create two temporary UI buttons, Tidy & Reset.
- Test that the game is now fully playable\*

\* Later the scoring system will "push" these buttons.

## Find the bug(s) in the system

- Save your work before you start fiddling.
- See if you can work-out what's going on.
- We'll address the debugging strategy next.



## Some Debugging Tips

@UnityCourse  
[facebook.com/UnityCourse](https://facebook.com/UnityCourse)

## In this video...

- A bit more about **Transform.Translate()**.
- Using the Step button to slow things down.
- The perils of moving static colliders.
- How default animation blends can cause issues.
- Warnings about moving motion clips around.



## Refactor raise / lower into Pin.cs

- Move **RaisePins()** and **LowerPins()** into **Pin.cs**
- Rename to **Raise()** & **Lower()**
- Create **distToRaise** instance variable.
- Disable and re-enable gravity as required.
- Test animation sequence works well.



## Common Physics Issues

@UnityCourse  
[facebook.com/UnityCourse](https://facebook.com/UnityCourse)

## In this video...

- A reminder about Fixed Timestep.
- About bounciness & default materials.
- Issues of scale & effect on Physics Settings.
- Stopping rigidbody's passing through each other.



## Ensure Your Pins Renew Properly

- No passing through the floor.
- No exploding outwards.
- No other funny business!



## Tidying & Refactoring Code

@UnityCourse  
[facebook.com/UnityCourse](https://facebook.com/UnityCourse)

### In this video...

- About re-factoring your code.
- Single Responsibility Principle\*
- Wider framework of **SOLID**.

\* [http://en.wikipedia.org/wiki/Single\\_responsibility\\_principle](http://en.wikipedia.org/wiki/Single_responsibility_principle)

## Refactor Your Code

- Save your current, working state.
- Go through all your methods.
- Add “why” comments as necessary.
- Move code between classes as required.
- Test it still works regularly.



## PART 2: CONTROL SYSTEM

@UnityCourse  
[facebook.com/UnityCourse](https://facebook.com/UnityCourse)



## How 10-Pin Bowling Scoring Works

@UnityCourse  
[facebook.com/UnityCourse](https://facebook.com/UnityCourse)

## In this video...

- Read how bowling scoring works\*
- Draw an Object Communication Diagram.

\*<https://www.youtube.com/watch?v=aBe71sD8o8c>

1	2	3	4	5	6	7	8	9	10	Total
8	7	34	X	2	X	X	8	X	8/9	170
17	30	37	57	77	105	123	131	151	170	170

[http://en.wikipedia.org/wiki/Ten-pin\\_bowling](http://en.wikipedia.org/wiki/Ten-pin_bowling)

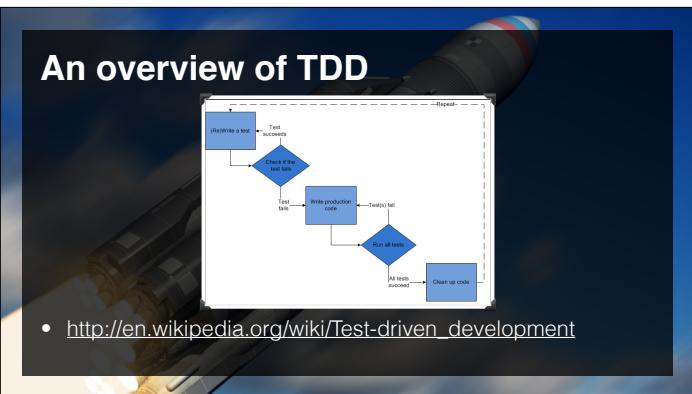
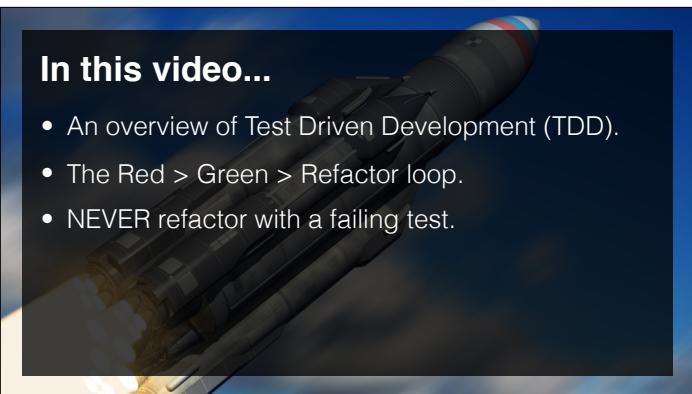
## Object Communication Diagram

- Draw boxes for PinSetter and ScoreMaster.
- Draw lines for API calls between them.
- Decide what messages will be passed.
- Decide what public properties (if any) you need.
- Mentally rehearse how it may work.

## Our Architecture

- Very simple API for **ScoreMaster.cs**
- Define **public enum Action {Reset, Tidy}**
- Frame list will grow as frame scores are finalised.





## “Red > Green > Refactor” Loop

- Write a failing test.
- Write code to make test pass.
- Rinse and repeat.



## Read around about TDD

- Have a look on YouTube.
- Read the Wikipedia article.
- Do a general Google search.
- Talk to your friends about it!



## Install Unity Test Tools

@UnityCourse  
[facebook.com/UnityCourse](https://facebook.com/UnityCourse)

## In this video...

- What Unity Test Tools are\*
- How to find and install them.
- Setting up your first failing test.

\*<https://www.assetstore.unity3d.com/#!/content/13802>

## Install Unity Test Tools

- Find Unity Test Tools in the Asset Store.
- Install them into your project.

## Making Your First Test Pass

@UnityCourse  
[facebook.com/UnityCourse](https://facebook.com/UnityCourse)

## In this video...

- Setup our **ActionMaster.cs** class.
- Write our first real test code.
- Code the solution until the test passes.
- Refactor.

## Get FirstStrikeReturnReset Passing

- Write the simplest code that makes this test pass.
- NO re-factoring at this stage.
- Just make something work in **ActionMaster.cs**.

Red > Green > Refactor

## In this video...

- Go through a few more TDD loops.
- Remember **Red > Green > Refactor**.
- Be strict with yourself, build the discipline.
- Save a version snapshot regularly.
- Have fun!

## Write one test of your own

- Write a single test of your own.
- Anything relating to correct return types.
- Ensure it leads to the need to write new code.
- Write the new code until the test passes.
- Refactor, and re-check all tests pass.

## Finishing Our Control Code

## Finish The Control Code

- Go through Red > Green > Refactor loop strictly.
- Focus your failing tests on edge cases.
- Keep your **ActionMaster.cs** looking clean.
- Stop when you're confident it's done.
- Challenge the community to write a failing test!



## The last frame

17	18	19	20	21
1	1	1	1	

No bowl 21 awarded  
endGame at bowl 20

1	1	1	/	?

Bowl 21 awarded  
Reset at bowl 20

1	1	X	-	?

Bowl 21 awarded  
Reset at bowl 19

## Failing Tests Challenge

@UnityCourse  
[facebook.com/UnityCourse](http://facebook.com/UnityCourse)

## In this video...

- Thanks to Daryl Kempthorne for this...

Bowl 20 tidy after strike

I think that I have found a failing test.

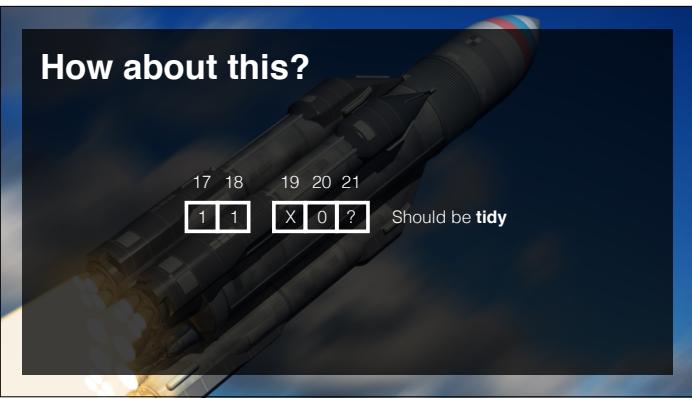
I think that in the last frame that if you roll a strike on bowl 19 and don't knock over all of the pins on bowl 20, then we should have a tidy. At the moment, we are getting a reset, not a tidy.

## Daryl's failing test

17 18 19 20 21  
1 1 X 5 ? Should be **tidy** (not reset)

## Write test, and make it pass

- **RED**: Write a test for Daryl's failing condition.
- **GREEN**: Write the code to make it pass.
- **REFACTOR**



---

---

---

---

---



---

---

---

---

---



---

---

---

---

---

## A possible bug reporting cycle

1. User submits bug report.
2. **QA** verify this bug is valid & can be reproduced.
3. **Developer 1** writes a failing test.
4. **Developer 2** writes code to make test pass.
5. User informed.

## Create a test for SYMPTOMS...

Nathan Robbert posted a discussion in

Knock down 10 pins on a second bowl in a frame.

In frames 1-9 if on your first bowl you knock down 0 (zero) pins, and then you knock down 10 pins on the second bowl, you trigger a EndTurn, but you are also adding 2 to bowl which kicks it into the 2nd bowl of the next frame, thus treating it as a strike and not a spare. It should EndTurn, but it should only increment bowl + 1, not 2.

## Nathans Bowl Index Test

1	2	3	4
0	10	!	?

Bowl 3 skipped over in this case

0	10	!	5
---	----	---	---

3rd bowl gets entered in wrong place.  
System wrongly returns a **tidy**.

0	10	5	1
---	----	---	---

Should return **endTurn**.

## Make Dondi's test pass

```
[Test]
public void Dondi10thFrameTurkey () {
    int[] rolls = {1,1, 1,1, 1,1, 1,1, 1,1, 1,1, 1,1, 1,1};
    foreach (int roll in rolls) {
        actionMaster.Bowl (roll);
    }
    Assert.AreEqual (reset, actionMaster.Bowl (10));
    Assert.AreEqual (reset, actionMaster.Bowl (10));
    Assert.AreEqual (endGame, actionMaster.Bowl (10));
}
```

<https://gist.github.com/BenTristem/958250aab975526adda8>



## Wire-Up ActionMaster.cs

@UnityCourse  
[facebook.com/UnityCourse](http://facebook.com/UnityCourse)

### In this video...

- Connect **PinSetter.cs** to **ActionMaster.cs**
- Refactor **PinSetter.cs** to report pin fall.
- Connect **PinSetter.cs** to the animator.
- Remove the Tidy and Reset test buttons.
- Ensure our game controls it's self now.



## Refactor PinSetter.cs for pin fall

- PinSetter.cs to call ActionMaster.Bowl()
- It must pass in pins fallen, not standing.



## Connect PinSetter.cs to animator

- Get PinSetter directly triggering the animator.
- Check it works.
- Remove UI test buttons.



## Using OnTriggerExit()

@UnityCourse  
[facebook.com/UnityCourse](https://facebook.com/UnityCourse)



## In this video...

- Recap the problem with the current setup.
- Use **OnTriggerExit ()** for objects leaving colliders.
- Refactor **PinSetter.cs** for the new paradigm.

## Change the pinsetter paradigm

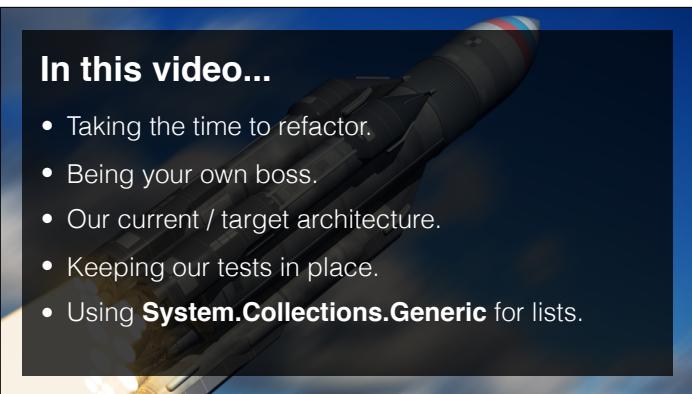
- Change ballEnteredBox to ballLeftBox.
- Create a new lane collider.
- Connect a **GutterBall.cs** script.
- **OnTriggerExit()** to set **pinSetter.ballLeftBox**
- Test it all still works.

## PART 3: FINISHING OFF

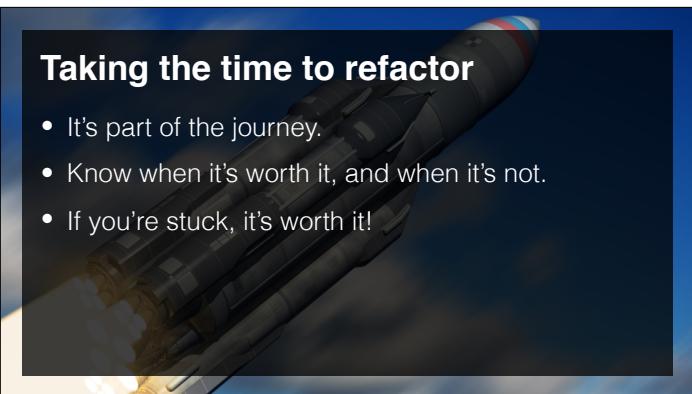
@UnityCourse  
[facebook.com/UnityCourse](https://facebook.com/UnityCourse)



@UnityCourse  
[facebook.com/UnityCourse](https://facebook.com/UnityCourse)



- Taking the time to refactor,
- Being your own boss.
- Our current / target architecture.
- Keeping our tests in place.
- Using **System.Collections.Generic** for lists.



- It's part of the journey.
- Know when it's worth it, and when it's not.
- If you're stuck, it's worth it!

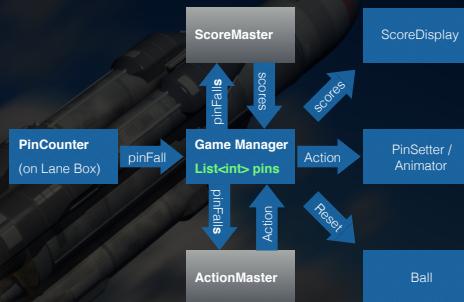
## Being your own boss

- You wear two hats.
- One is your boss, telling you what to do.
- The other is the worker, obeying the boss.
- Neither can improve if the lines are blurred.
- It takes discipline, and it's worth it.

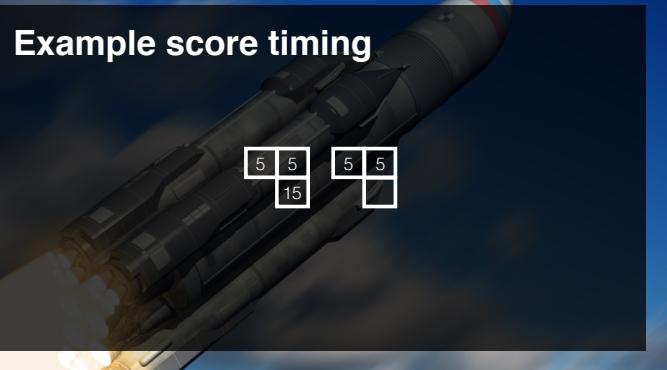
## Our current architecture...



## Target...



## Example score timing



5	5
15	

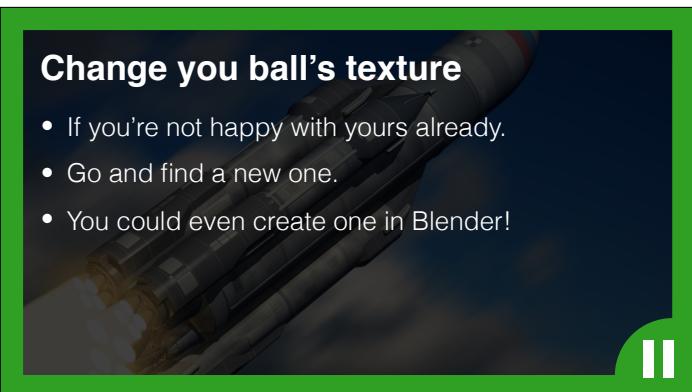
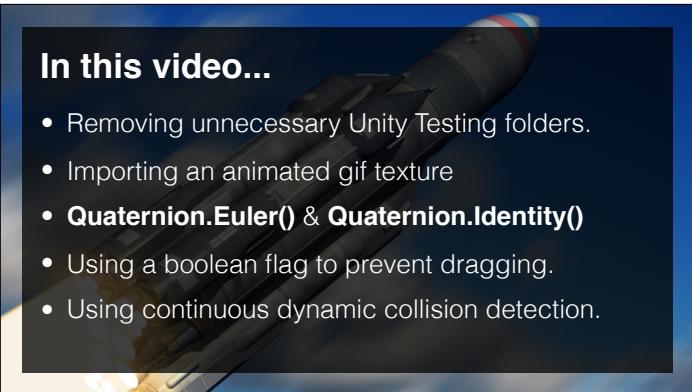
5	5
	5

## Keeping our tests in place

- We'll need to refactor the ActionMaster tests.
- Build ScoreMaster using TDD.
- We will have the option to merge these classes.
- The merge would be protected by the tests.

## Using System.Collections.Generic

- To use lists you need to import a new namespace.
- **using System.Collections.Generic;**
- Define a list like this...
- **List<T> listName = new List<T>();**
- ... where T is the type, **int** in our case.



## Reset rotation of ball and pins

- Pins should be reset to vertical when raised.
- Ball should be reset to neutral when reset.
- Consider `Quaternion.Euler()` for the pins.
- You could use `Quaternion.identity` for the ball.



## Prevent double dragging

- Make sure you can't drag the ball once launched.
- Similar solution already in...

*Thanks to Dondi for noticing this*



## Refactoring Code & Tests

@UnityCourse  
[facebook.com/UnityCourse](https://facebook.com/UnityCourse)

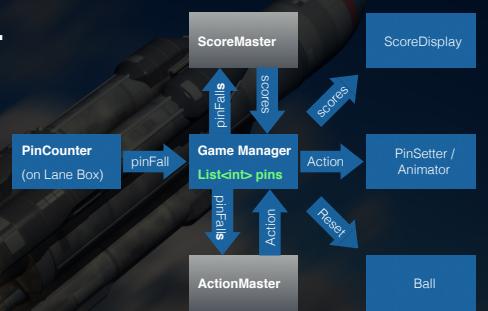


## In this video...

- Refactor your **ActionMaster** tests for new API.
- Refactor your code to single responsibility\*

\*[http://en.wikipedia.org/wiki/Single\\_responsibility\\_principle](http://en.wikipedia.org/wiki/Single_responsibility_principle)

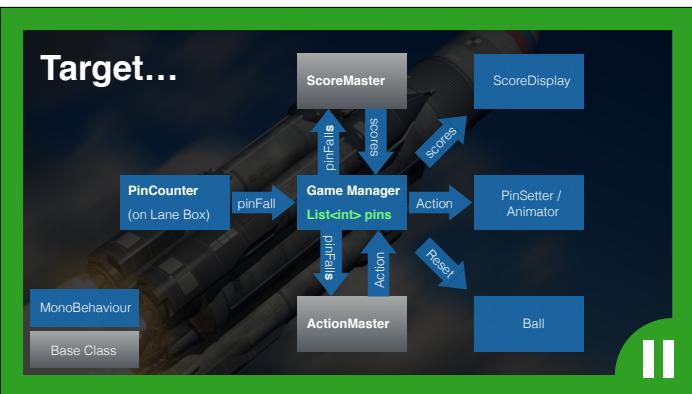
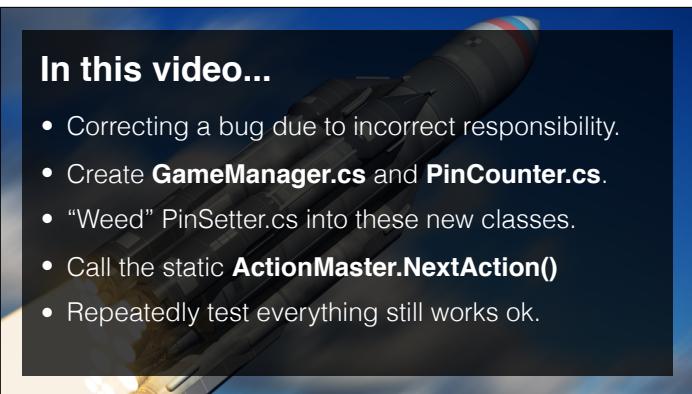
## Target...



## Refactor ActionMaster.cs

- Still return an **Action** enum.
- Create the new *static* method below.
- Call your existing **Bowl** method for each pinFall.
- New API passes a list of pitfalls, write this method

```
public static Action NextAction (List<int> pinFalls)
```





In this video...

- An overview of the challenge (and delight) ahead.
- Find & move **ScoreMasterTest.cs** to Editor folder.
- Temporarily disable **ActionMasterTest.cs**.
- Write cumulative scorer in **ScoreMaster.cs**.
- Set THE challenge.

Write cumulative scorer

- Write cumulative scorer in **ScoreMaster.cs**.
- We can write this now, without **ScoreFrames**.
- It's a simple loop, the signature is below...

```
public static List<int> ScoreCumulative (List<int> rolls)
```

||

## Get ScoreMaster.cs Working!

- This took me over 2 full days!
- It may take you much more, or less time.
- Use the TDD Red > Green > Refactor strictly.
- Un-comment the tests one at a time.
- (My solution has 18 new lines in **ScoreFrames**)



## Realtime Bowling Scoring Solution

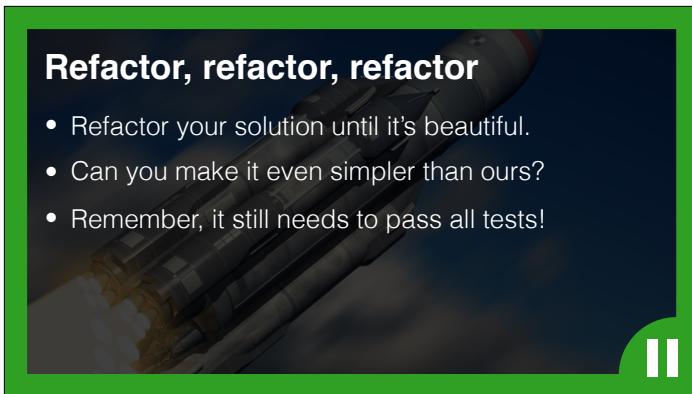
@UnityCourse  
[facebook.com/UnityCourse](https://facebook.com/UnityCourse)

### In this video...

- How we solved the scoring problem.
- A reminder that it's the destination that counts.
- How tests protect your refactoring.

*"Make things as simple as possible, but not simpler"*

Albert Einstein



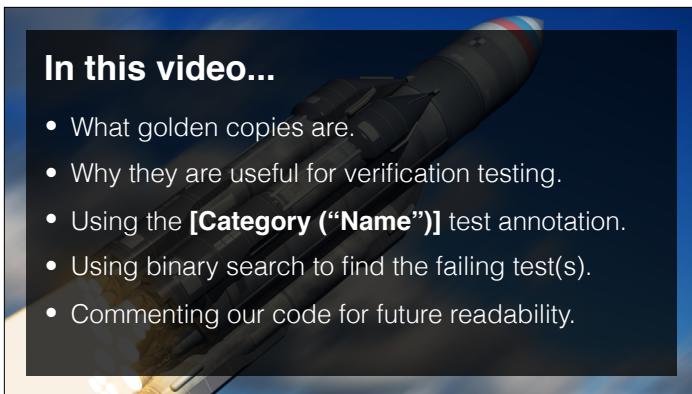
# Refactor, refactor, refactor

- Refactor your solution until it's beautiful.
  - Can you make it even simpler than ours?
  - Remember, it still needs to pass all tests!



# Golden Copy Testing

@UnityCourse  
[facebook.com/UnityCourse](https://facebook.com/UnityCourse)



## In this video...

- What golden copies are.
  - Why they are useful for verification testing.
  - Using the **[Category (“Name”)]** test annotation.
  - Using binary search to find the failing test(s).
  - Commenting our code for future readability.

## Comment your code

- Make your code as “pretty” as possible.
- The shorter the easier to reason about.
- Write comments about WHY you do things.
- Let the code speak for it’s self re the WHAT.
- Make pretty using <http://instacod.es> & share.



## An Array Of UI Text

@UnityCourse  
[facebook.com/UnityCourse](https://facebook.com/UnityCourse)

## In this video...

- Rearrange your UI to make space for scores.
- Create your scoreboard.
- Wire-up your scoreboard.



## Create your scoreboard

- Make space for up to 21 bowls (single character).
- Also ensure you have 10 frame score boxes.
- Frame scores need to take up to 3 digits.



## Wire-up your scoreboard

- Create an array of Text in **ScoreDisplay.cs**.
- Wire all the score boxes to this display.
- Do the same for the frame scores.



## Protecting Yourself From Bugs

@UnityCourse  
[facebook.com/UnityCourse](https://facebook.com/UnityCourse)



## In this video...

- Break your whole game!
- Trace your bug via console.
- Discuss handling undefined states.
- Discuss isolating and fixing bugs.

## Target...



## Break your whole game!

- Add a method to **ScoreDisplay.cs**
- Use the signature below.
- Call this from GameManager, before **ball.Reset()**;
- Make it error, for example invalid array access.

```
public void FillRollCard (List<int> rolls)
```

## Write down what's happening

- Use the console trace.
- Write 5+ bullet points as to what's going on.
- Think about how you may fix this.

---

---

---

---

---

## Tracing your bug via console

- Once the ball has settled **Pin Counter** loops.
- Every frame it calls **gameManager.Bowl ()**.
- **PinsHaveSettled ()** method never completes.
- **ballOutOfPlay** stays true, hence infinite loop.
- GameManager's pinfall list fills, leads to EndGame

---

---

---

---

---

## Handling undefined states

- Imagine this only happens 1 / 1000 games.
- Failing hard vs. failing soft & reporting error.
- Obviously we fix the bug and move on BUT...
- ... we also want to prevent show-stopping issues.
- We'll catch the errors next, *and* fix the bug.

---

---

---

---

---

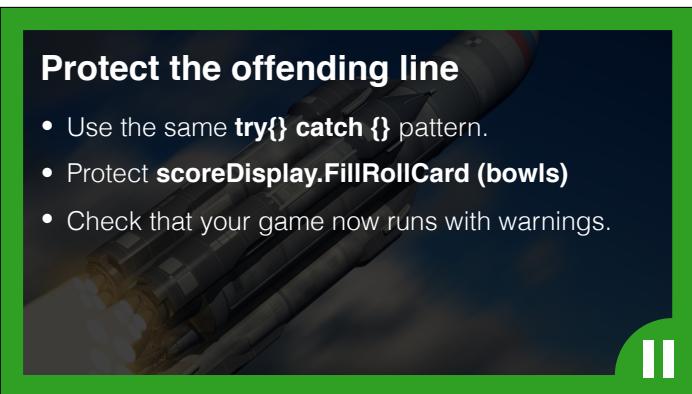


## Try, Catch For Error Handling

@UnityCourse  
[facebook.com/UnityCourse](https://facebook.com/UnityCourse)

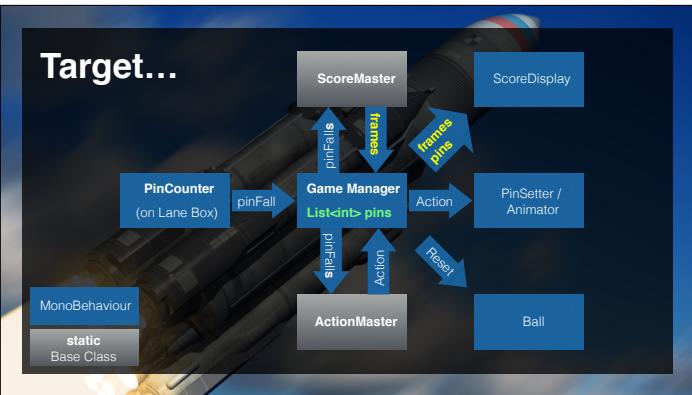
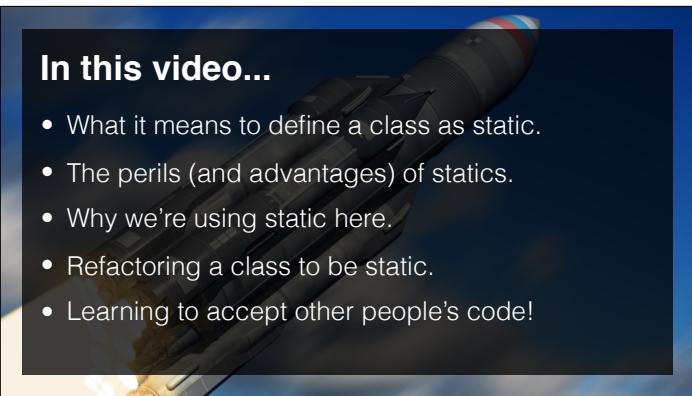
### In this video...

- How to decouple code with **try{}**, **catch{}**.
- Mainly used in file handling applications.
- Could be used in a team so one can move on.
- Isolating our **ScoreDisplay.cs** issues.
- Making our game fail gracefully.



### Protect the offending line

- Use the same **try{} catch {}** pattern.
- Protect **scoreDisplay.FillRollCard (bowls)**
- Check that your game now runs with warnings.



## Tom Butler's view...

### Conclusion

There are very few times when you should use static methods or variables and certainly never for locating dependencies. They should never be used by external classes and cause far more problems than they solve. They result in poorly designed spaghetti code and try to introduce procedural code into an object oriented world. They prevent objects from being able to guarantee they can fulfil their contracts and make code very difficult to debug and test. Code stops being self-documenting and a lot of the power of object-oriented programming is sacrificed entirely. Why would you ever want to impose those limitations on your code and anyone else who has to use it when there are almost always better methods that achieve the same thing?

Static variables always introduce global state (which is bad) and Static methods always break encapsulation (Which is also bad). Static should be avoided at all costs!

<https://r.je/static-methods-bad-practice.html>

## Optional: Refactor ActionMaster

- Integrate **ActionMaster2.cs** from Start Pack.
- Re-factor until beautiful.
- Ensure all tests pass.
- Paste your code in the discussions via PasteBin.
- The less lines the better, if still readable.



## Unit Testing Monobehaviours

@UnityCourse  
[facebook.com/UnityCourse](http://facebook.com/UnityCourse)

## In this video...

- Monobehaviour classes are hard to test\*
- Code a testable static helper method.
- Create our testing structure.

\* <http://blogs.unity3d.com/2014/06/03/unit-testing-part-2-unit-testing-monobehaviours/>

## Target...



## Write FillRolls Method

- **public void FillRolls (List<int> rolls)**
- Calls `FormatRolls` and iterates the returned string.
- Should be just 3 lines of code.



## Designing Your Own Tests

@UnityCourse  
[facebook.com/UnityCourse](https://facebook.com/UnityCourse)

A photograph of a rocket launching from a launch pad, with its engines glowing brightly at the base. The rocket has a white, blue, and red striped fairing. The background shows a clear blue sky with some wispy clouds.

**Use TDD To Write FormatRolls()**

- **public static string FormatRolls (List<int> rolls)**
- Write your own tests as you go.
- Stick to the red-green-refactor loop.
- Aim for 20 lines or less of beautiful code.
- Enjoy the process & share with [SPOILER].

||



## Final Fixes & Finishing Off

@UnityCourse  
[facebook.com/UnityCourse](https://facebook.com/UnityCourse)

## In this video...

- Stopping nudging ball off the floor\*
- Make ball collisions detection continuous.
- Suggestions for improvement.

\* Thanks Yang <https://db.tt/OLyfcZZr>

## Stop the ball nudging off

- Use whatever you've learnt so far.
- Make sure you can't nudge off floor.

## Make It Your Own

- Add a menu system (start, prefs, end game).
- Provide a 2nd ball, maybe a Death Star?
- Save the high score to Player Prefs.
- Make it portrait, and deploy to mobile.
- Make it multi-player?



@UnityCourse  
[facebook.com/UnityCourse](https://facebook.com/UnityCourse)

