**GMAPS**

[A picture containing text, table, vector graphics

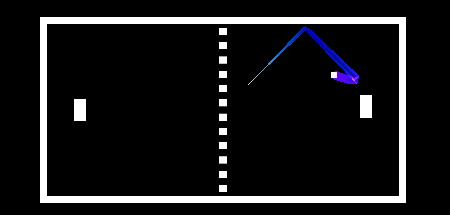
Description automatically generated](https://soundcloud.com/user-879892202)

**(Simple)**

**Collisions**

**Worksheet**

**To see the additional comments and resources, make sure you select All Markup in the Review/Tracking pane**



Important note

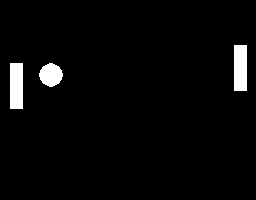
**PART 1: Unity Physics Pong Game**

Download **04\_SIMPLE\_COLLISIONS\_worksheet.zip** from LMS.

Add this to **GMAPS\_Oct\_2023\_Worksheets STUDENT** in the **Assets** folder.

In the **Dynamics & Kinematics worksheet**, you implemented shooting a pool ball by dragging the mouse to represent a pool cue. But the ball just rolls off into the distance! It would be nice if it could bounce off the edges of the table, like a real ball.

This is actually the same problem as PONG.

Diagram

Description automatically generated with low confidence

So, before we get started, we’ll take a quick diversion to see how collision detection works in a simple game like Pong.

A picture containing shape

Description automatically generated

But first, some review questions.

**Question 1**

Complete the **Practice Quiz: Simple Collisions** quiz in the LMS folder for this week. These questions review basic Unity Physics and math-related topics in the context of making a simple Pong game.

**Question 2**

Download **04\_SIMPLE COLLISIONS\_worksheet.zip** from LMSand add this to the Assets folder of your worksheet project.

Before looking at how to implement our own simple collision detection, we’ll remind ourselves how this is done using the Unity Physics Engine API.

So, in this lab, we will learn how the classic arcade game Pong can be implemented using physics. Note, most games involve objects firing, falling, bouncing, exploding, etc., and this is usually done with physics, too – so this is not just about Pong.

In **Part 2**, you will implement the classic Pong game *yourself*, where you have to figure out all the collisions and bounces yourself.

There is a very good online tutorial about a Unity physics-based Pong implementation. We’ll use this as a starting point – it doesn’t implement the full game, which you are encouraged to finish.

1. Before starting, watch [this video](https://www.youtube.com/watch?v=tNtOcDryKv4) about the different ways to move objects in Unity (this will also help you with the next section).
2. Find the 2D Pong tutorial on the **noobtuts** website.

noobtuts

<http://noobtuts.com/unity/2d-pong-game>

1. Go through the tutorial. The project files are available on the website, but it’s best to create the files yourself. Use the assets inside the project’s **04\_SIMPLE COLLISIONS\_worksheet/Pong/Pong Unity Physics** folder.
2. Add music and sound effects to your game. You may use the mp3 files on LMS, or find your own (make sure they are free and not copyrighted). This isn’t anything to do with math or physics, but it’s fun, and makes the game come alive!

Okay, onto creating our own simple collision system for Pong.

Text

Description automatically generated

**PART 2: Axis-Aligned Collisions**

**Question 3**

For this question, you must implement axis-aligned boundary collision detection and response for the Pong game.

The code in the Unity project will differ slightly from the slides, but it’s expected (actually, required) that you are able to handle such small differences.

1. Open **Pong\_NP** inside the **Pong/Pong No Unity Physics** folder and run the game.

Look through the code in **Scripts**. You don’t have to worry too much about the code for the paddles (though you should be able to understand it). Make sure you understand the existing code in **PongBall.cs**.

Complete the code for the **HandleBoundaryCollision()** function in PongBall.cs.

Make sure that you *comment* your code!

Chart, bar chart

Description automatically generated

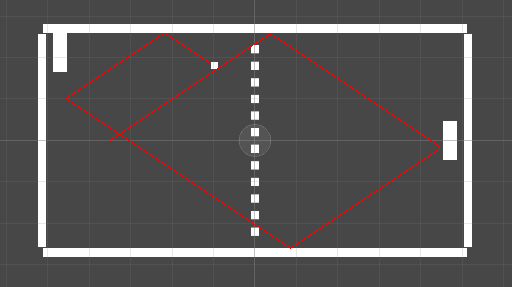
1. Explain what this code does in **HandleBoundaryCollision** , and why it is important:

A picture containing text

Description automatically generated

1. In **FixedUpdate()**, move the highlighted code below to the *top* of the function.

What happens? A bug!



Explain why this happens.

Graphical user interface, text, application

Description automatically generated

Text

Description automatically generated

**PART 3: Non-Axis-Aligned Collisions**

**Question 4**

Now we’ll look at how to detect and respond to collisions between a ball and a non-axis aligned boundary.

*As we go through these exercises, you’ll need to refer to your lecture slides a lot!*

*Note that these exercises use Unity’s Vector2 class.*

Open the **SimpleCollision** Unity scene in **04\_SIMPLE\_COLLISIONS\_worksheet/ SimpleCollision in GMAPS\_Oct\_2023\_Worksheets STUDENT.**

You should see these game objects:

A green rectangular object with a white circle and a white line

Description automatically generated

What we want is for the white ball to collide with a non-axis-aligned wall *without using Unity’s physics engine!*

The scripts folder should contain these scripts:

Icon

Description automatically generated with low confidence

These are all skeleton scripts. You’ll be adding more code soon!

First, we’ll look at **colliding with the wall**.

1. Click on **wall**.

You should see from the image above that the **wall** game object has two empty game objects as children, called **top** and **bottom**.

These allow us to calculate the vector from the bottom to the top of the wall. (See the diagram on the next page.)

We’ll call this vector **wallVec**. (Similar to lineVec in your lecture slides.)

This vector is obviously *parallel* to the wall. We can also calculate the centre position of the wall.

Diagram

Description automatically generated with medium confidence A picture containing diagram

Description automatically generated

From **wallVec**, we can get the normalised perpendicular vector to the wall, which we’ll call **normalVec**:

1. Complete the Wall2D script as shown below:

Graphical user interface, application

Description automatically generated

Again, note that we are using Unity’s Vector2 here, so we can call its **Perpendicular** function, and also use its **normalized** property.

1. Now we’ll switch attention to **Ball2D.cs**.

The ball needs to check if it has collided with a wall (which is basically a line, as in your lecture slides).

We’ll add a function: **IsCollidingWith(Wall2D other)**

Complete the code for the function below that checks for a collision between a wall and a ball.

Timeline

Description automatically generated

1. We should check that our code is working by printing a debug message if a collision is detected.

Rather than having to shoot the ball at the wall each time, it would be good to just drag the ball onto the wall to check for a collision.

First, we must stop the ball’s position from being updated every frame, so comment out the relevant line of code in **Ball2D.Update()**.

Graphical user interface, text, application

Description automatically generated

Add this code to **Table2D.cs**. Make sure that the **white\_ball** and **wall** object references are set in the Inspector (there is a **Walls** array).

A close-up of text

Description automatically generated

Run the project and drag the white ball onto the wall.

What happens?

If everything’s working, you should see the debug message printed out when you drag a ball onto the wall.

A screen shot of a computer

Description automatically generated

Now you know you can detect a collision. But what about the collision response?

Text

Description automatically generated

**PART 4: Collision response**

*Collision response* refers to calculating the velocity of the ball *after* the collision. We looked at this in the lecture slides. Remember that **vi** and **vf** are the initial and final velocities of the ball.

**Question 5**

1. Review this and the following slides from your lecture:

A math equation with a diagram

Description automatically generated with medium confidence

Note that the code is actually given to you in the slides.

The slides explain how to respond to a collision with a non-axis-aligned wall. You must understand the mathematics to be able to write the collision response code.

Let’s get started.

1. The first question is, where should the collision response code go?

**Table2D.Update()** checks for a collision between the white ball and the wall, so we’ll put our response code there.

A screenshot of a computer code

Description automatically generated

Remember, we must:

A picture containing diagram

Description automatically generated

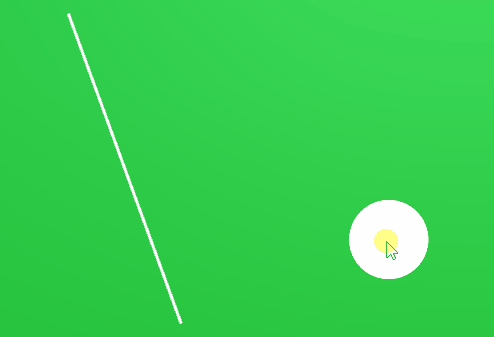
1. Find the perpendicular vector to the wall, **perp**
2. Normalise the perpendicular vector, **n**
3. Project (**Proj**) the negative of the ball’s velocity (**-vi**) onto **n**
4. Set the ball’s post-collision velocity (**vf**) to **2 \* P + vi**
5. Now all you have to do is complete the missing code!

A screenshot of a computer code

Description automatically generated

Notice how once you understand rather a lot of complex vector math, only a few lines of code are needed!

Test that everything works by shooting the white ball at the wall using the mouse cue stick.



The ball should move off in the correct direction after colliding with a wall.

You did all that without using a Unity Physics collider component or using any of Unity’s collision detection functions.

Yet another high-five! 😊

Text

Description automatically generated

**PART 3: Reflection**

1. What was the most difficult part of this worksheet. Explain why you found it difficult, and how you managed to overcome this (if at all).

*You might like to reference specific lecture slides and lines of code, give some URLs and diagrams as references, etc.*

(Min 150 words)

1. Choose a collision topic that are you still unsure about. Explain that topic, and the steps you will take to clarify your doubts. You will be graded on how well you analyse your problem with understanding that topic.

If you are confident about all the topics, select what you consider to be the most difficult topic and explain ONE more advanced concept related to the topic. You will be graded on the complexity of the concept, and how well you explain it.

*You might like to reference specific lecture slides and lines of code, give some URLs and diagrams as references, etc.*

(Min 150 words)

1. Write a reflection about this worksheet. You may like to include how effective it is to motivate and help you learn, how well it relates to games development, etc.

(Min 150 words)