**TEMASEK POLYTECHNIC**

**SCHOOL OF INFORMATICS & IT**

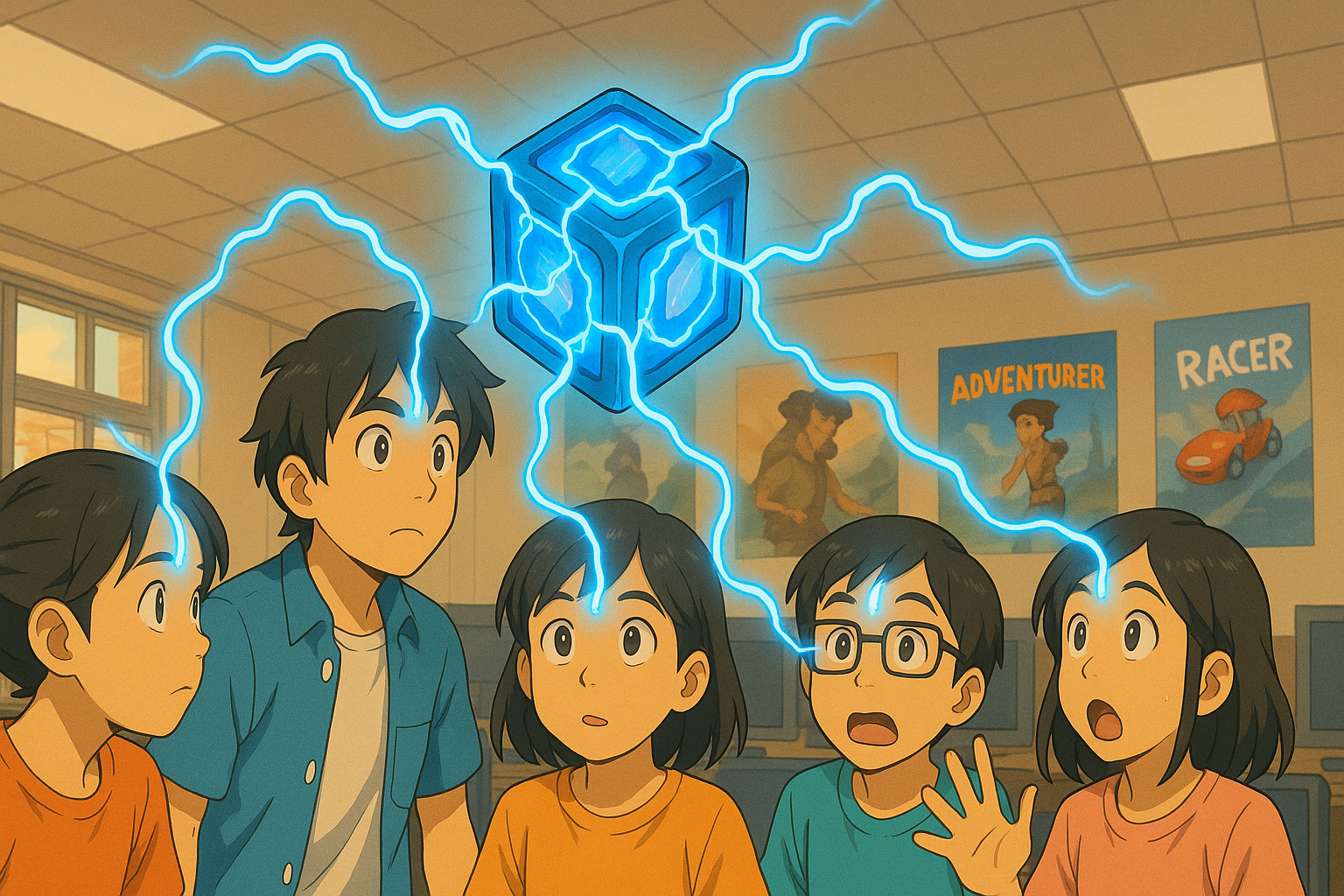
**DIPLOMA IN IMMERSIVE MEDIA & GAME DEVELOPMENT**

**AY2025/2026 APRIL SEMESTER**

**GADV (CGE2C25)**

**Unity Physics Worksheet**

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



**QUESTION 1 (basics of Unity physics)**

1. What is the core responsibility of a RigidBody component?

To provide physics to an object.

1. Why is the Pong game paddle mentioned as an example of an object whose rigidbody’s isKinematic property would be set to true? Hint: what controls the paddle movement?

This is because the physics would be controlled by the script, rather than unity’s physics engine

1. Explain what the rigidbody **constraints** are for. Give ONE example of when this might be useful.

Constraints as the name suggests, constraints the physics in someway. For example, you are making a 2d platformer, so you wouldn’t want your character to randomly rotate, you can use the freeze Z rotation constraint to prevent your character from rotating due to physics.

1. Watch [this video](https://www.youtube.com/watch?v=ixM2W2tPn6c).
2. What is one major problem with using the Translate function to move an object?   
   It would not take into account the collision of the object.
3. Why is the physics-based code put in the FixedUpdate function and not in Update?

Fixed Update runs at a Fixed rate, meaning that the physics based code would always run at a constant speed, unlike Update, where its dependant on the game’s framerate, having an inconsistent rate.

1. What are the THREE different methods covered in the video to move an object using physics?

Explain how each method works.

* Rigidbody.AddForce(): Applies a force to a rigidbody, like a push. Its great for games like moving in space. Eg: Rigidbody.AddForce(dir\*speed)
* Rigidbody.Velocity: Sets the velocity of the rigitbody, giving constant movement with no acceleration. This is good for rigidbodies that need to constantly move. Eg: Rigidbody.Velocity = dir\*speed
* Rigidbody.Moveposition: Moves to the exact location given, whilst respecting physics and collision. Eg: Rigidbody.MovePosition(position)

**QUESTION 2 (Colliders)**

1. What is the core responsibility of a collider component?

To facilitate collision detection

1. Look at the question in [this forum post](https://forums.oculusvr.com/developer/discussion/59641/problems-with-fast-moving-object-collision-in-unity-table-tennis-racket-vs-ball). What is the solution to the problem? Do some research to explain why this is the solution. Draw a diagram to illustrate your answer.

The rigidbody collision detection of the ball should be set to continuous, that way the game would constantly track collision, even if the ball is moving too fast.

1. What is the difference between collision *detection* and collision *resolution*?

Collision detection is the method in which the engine detects collision while the resolution is just how detailed the hitbox for the collider is.

**QUESTION 3 (Joints)**

1. What type of joint is used in the video? According to Unity’s documentation, what does this joint do? Give an example of how the joint could be used in a game.
2. What other types of joints are there? Briefly explain what FOUR of these joints does (do NOT include the type of joint you answered for part a above), and give an example of how each one might be used in a game.

**QUESTION 4 (Physic Material)**

1. What is the difference between dynamic friction and static friction?

Static friction applies when a rigid body is at rest, whilst dynamic friction applies while the body is moving, particularly sliding with another body.

1. Find a game that you have seen online or played yourself where physics is a core game mechanic.

You must:

* give a brief description of the game

Portal is a first-person puzzle game where you have to use portals and a fundamental understanding of physics to interact with chambers to open doors, from simply pressing buttons, to dropping cubes on pressure plates and more

* briefly explain how physics is used

In portal, you have to use physics to your advantage. One example is where you have to sometimes cross large gaps. By creating portals in a specific layout, you can translate your falling momentum into forward momentum to cross gaps.

* explain why physics is important

Without physics, the game would be very stagnant, which would be boring. The physics makes interaction with the puzzle chambers important.

* include a relevant screenshot of the game
* 

**QUESTION 5 (CharacterController)**

1. According to the video, the built-in CharacterController component does not interact with physics. So, a CharacterController does not react to gravity, and when a CharacterController collides with another object with a rigidbody it does not have any force added to it (like when being hit by a cannonball throws an object across the scene).

What gameplay-related reason(s) can you think of to explain why the CharacterController component was designed this way?

This is to give the developer more freedom over the character’s movement, and prevents any clashing with unity’s physics engine. For example, if you want a game where you fly around in space, you would rather want something more customisable than to fight against unity’s gravity.

1. To make a character controlled by a CharacterController experience gravity while moving, two options are given in the video. What are these?
2. Create a Unity scene with a capsule game object with a CharacterController. The script below will make the character jump when the space bar is pressed.

Fill in the missing parts of the code, then copy the whole Character class code here. Make sure it is formatted properly!

Text

Description automatically generated

*using* UnityEngine;

*using* UnityEngine.EventSystems;

*public* *class* Character : MonoBehaviour

{

*// Start is called once before the first execution of Update after the MonoBehaviour is created*

    CharacterController characterController;

*public* *float* speed = 6.0f;

*public* *float* jumpSpeed = 8.0f;

*public* *float* gravity = 20.0f;

*private* Vector3 moveDirection = Vector3.zero;

*void* Start()

    {

        characterController = GetComponent<CharacterController>();

    }

*// Update is called once per frame*

*void* Update()

    {

*if* (characterController.isGrounded)

        {

            moveDirection = *new* Vector3(Input.GetAxis("Horizontal"), 0.0f, Input.GetAxis("Vertical"));

            moveDirection \*= speed;

*if* (Input.GetButton("Jump"))

            {

                moveDirection.y = jumpSpeed;

            }

        }

*// Apply gravity*

        moveDirection.y -= gravity \* Time.deltaTime;

        characterController.Move(moveDirection \* Time.deltaTime);

    }

}

**QUESTION 6 (Constant Force component)**

1. For the Constant Force component, what is the difference between the Force and Relative Force properties, and between the Torque and Relative Torque properties?

Relative means that the force/torque is applied relative to the current position/rotation, while the normal Force/torque will apply based off the global position.

1. When the cube is in the air with a Force X of 1, it moves, but when it falls to the plane, it doesn’t move.

Why not? This is because of the way unity handles friction, a force of X may not be enough to overcome the friction of the plane or the cube.

1. An object has a constant force applied along its local positive Z axis. What happens to the object’s speed over time?

Explain your answer.

The object will accelerate overtime as more and more force is applied to the object, thus increasing its speed.

**QUESTION 7 (Forces)**

1. Paste your **Player** class code here. Make sure your code is readable, properly formatted, and commented.

*using* UnityEngine;

*public* *class* Player : MonoBehaviour

{

*private* CharacterController controller;

*// Your variables used for moving the player*

*public* *float* moveSpeed = 5.0F;

*public* *float* jumpForce = 10.0F;

*// Code to set the radius and force of the explosion*

*public* *float* radius = 5.0F;

*public* *float* power = 10000.0F;

*void* Start()

    {

        controller = GetComponent<CharacterController>();

        controller.detectCollisions = *false*; *// See comment about this!*

    }

*void* CheckExplosion()

    {

*if* (Input.GetKeyDown(KeyCode.E))

        {

*// Your code to generate the explosion*

            Vector3 explosionPosition = transform.position;

            Collider[] colliders = Physics.OverlapSphere(explosionPosition, radius);

*foreach* (Collider hit *in* colliders)

            {

                Rigidbody rb = hit.GetComponent<Rigidbody>();

*if* (rb != *null*)

                {

*// Apply explosion force*

                    rb.AddExplosionForce(power, explosionPosition, radius);

                }

            }

        }

    }

*void* MovePlayer()

    {

*float* horizontal = Input.GetAxis("Horizontal");

*float* vertical = Input.GetAxis("Vertical");

        Vector3 direction = *new* Vector3(horizontal, 0, vertical);

        direction.Normalize();

*// Apply movement*

        controller.Move(direction \* moveSpeed \* Time.deltaTime);

*// Jumping*

*if* (Input.GetButtonDown("Jump") && controller.isGrounded)

        {

            Vector3 jump = *new* Vector3(0, jumpForce, 0);

            controller.Move(jump \* Time.deltaTime);

        }

    }

*void* FixedUpdate()

    {

        CheckExplosion();

        MovePlayer();

    }

}

1. Paste your **Kick** function code here. Make sure your code is readable, properly formatted, and commented.

*if* (Input.GetKeyDown(KeyCode.K))

        {

            Collider[] hitColliders = Physics.OverlapSphere(transform.position, 1.0f);

*foreach* (Collider hit *in* hitColliders)

            {

                Rigidbody rb = hit.GetComponent<Rigidbody>();

*if* (rb != *null*)

                {

*// Apply kick force*

                    Vector3 direction = transform.forward;

                    rb.AddForce(direction.normalized \* kickForce, ForceMode.Impulse);

                }

            }

        }

1. What does the upwardsModifier argument of AddExplosiveForce do?



This adds a upwards force to the explosion, which can make objects fly higher when adding the explosion force

**QUESTION 8 (Forces cont.)**

1. What are the FOUR different values for ForceMode? Explain what each value does.

* Force: Applies forces continuously over time, and considers the mass of the object
* Impulse: Applies forces in one go, considers the mass of the object
* Acceleration: Applies forces continuously, but ignores the mass of the object
* Velocity change: Applies forces in one go, but ignores the mas of the object.

1. Paste your code for firing the sphere here.

*using* UnityEngine;

*public* *class* Impulse : MonoBehaviour

{

*// Start is called once before the first execution of Update after the MonoBehaviour is created*

*public* *float* impulseForce = 10f;

*private* Rigidbody rb;

*void* Start()

    {

        rb = GetComponent<Rigidbody>();

    }

*// Update is called once per frame*

*void* Update()

    {

        Vector2 movement = *new* Vector2(Input.GetAxis("Horizontal"), Input.GetAxis("Vertical"));

*if* (Input.GetKeyDown(KeyCode.Space))

        {

            ApplyImpulse(Vector3.up); *// Apply impulse in the upward direction when space is pressed*

        }

*if* (movement != Vector2.zero)

        {

            Vector3 direction = *new* Vector3(movement.x, 0, movement.y).normalized;

            ApplyImpulse(direction); *// Apply impulse in the direction of movement*

        }

    }

*void* ApplyImpulse(Vector3 *direction*)

    {

*if* (rb != *null*)

        {

            rb.AddForce(*direction* \* impulseForce, ForceMode.Impulse);

        }

    }

}

1. Paste your code for rotating the beam here.

*using* UnityEngine;

*public* *class* Beam : MonoBehaviour

{

*// Start is called once before the first execution of Update after the MonoBehaviour is created*

*public* *float* torqueForce = 10f;

*private* Rigidbody rb;

*void* Start()

    {

        rb = GetComponent<Rigidbody>();

    }

*// Update is called once per frame*

*void* Update()

    {

*if* (Input.GetKeyDown(KeyCode.Z))

        {

*// apply torque clockwise*

            ApplyTorque(Vector3.up);

        }

*if* (Input.GetKeyDown(KeyCode.X))

        {

*// apply torque counter-clockwise*

            ApplyTorque(Vector3.down);

        }

    }

*void* ApplyTorque(Vector3 *direction*)

    {

*if* (rb != *null*)

        {

            rb.AddTorque(*direction* \* torqueForce, ForceMode.Impulse);

        }

    }

}

**QUESTION 9 (Collision event-handling)**

1. Paste your code for **CosmicCube.cs** and **Orb.cs** here. Make sure your code is readable, properly formatted, and commented.

*using* UnityEngine;

*public* *class* CosmicCube : MonoBehaviour

{

*// Start is called once before the first execution of Update after the MonoBehaviour is created*

*public* Material colideEnterMaterial;

*public* Material colideExitMaterial;

*private* Renderer cubeRenderer;

*void* Start()

    {

        cubeRenderer = GetComponent<Renderer>();

    }

*// Update is called once per frame*

*void* Update()

    {

    }

*void* OnCollisionEnter(Collision *collision*)

    {

*if* (cubeRenderer != *null* && colideEnterMaterial != *null*)

        {

            cubeRenderer.material = colideEnterMaterial; *// Change material on collision enter*

        }

    }

*void* OnCollisionExit(Collision *collision*)

    {

*if* (cubeRenderer != *null* && colideExitMaterial != *null*)

        {

            cubeRenderer.material = colideExitMaterial; *// Change material on collision exit*

        }

    }

}

*using* UnityEngine;

*public* *class* Orb : MonoBehaviour

{

*// Start is called once before the first execution of Update after the MonoBehaviour is created*

*float* impulseForce = 10f;

*private* Rigidbody rb;

*void* Start()

    {

        rb = GetComponent<Rigidbody>();

    }

*// Update is called once per frame*

*void* Update()

    {

*if* (Input.GetKeyDown(KeyCode.Space))

        {

            ApplyImpulse(Vector3.forward); *// Apply impulse in the forward direction when space is pressed*

        }

    }

*void* ApplyImpulse(Vector3 *direction*)

    {

*if* (rb != *null*)

        {

            rb.AddForce(*direction* \* impulseForce, ForceMode.Impulse);

        }

    }

}

**QUESTION 10 (Raycasting)**

1. Paste your completed code for the CheckLineOfSight() function here.

*void* CheckLineOfSight()

    {

        GameObject[] enemies = GameObject.FindGameObjectsWithTag("Enemy");

        RaycastHit hitData;

*foreach* (GameObject enemy *in* enemies)

        {

            Vector3 vec = enemy.transform.position - transform.position;

            Debug.DrawRay(transform.position, vec, Color.red);

            RaycastHit hit;

*if* (Physics.Raycast(transform.position, vec, *out* hit, 30f))

            {

*if* (hit.collider.CompareTag("Enemy"))

                {

                    hit.collider.GetComponent<Renderer>().material.color = Color.red;

                }

            }

        }

    }

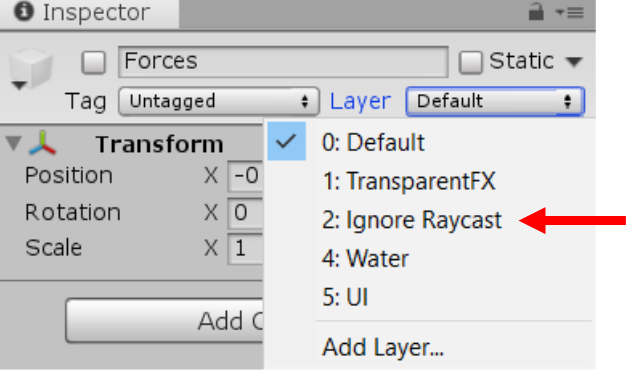
1. When checking for a hit using a ray, the code below is used:

if (Physics.Raycast(transform.position, vec, out hit, vec.magnitude))

Why is **vec.magnitude** used for the length of the ray?

Because magnitude gives the length of the vector

1. The layers dropdown in the Inspector has an entry for **Ignore Raycast**.



What does this mean? Why do you think this is important? Give an example.

This means that whatever is in that layer will not be detected in raycasts. Imagine you want to add windows, it would be unrealistic of the enemies to not see you through the window, so you can set the window to that layer, making it transparent to raycasts essentially.