# Pac-ball!

A game by Luiz

## Overview

The game is based off of PAC-MAN, wherein a player navigates through a maze, collecting points as he avoids all four ghosts. The game that I made is similar in terms of mechanics, basing my making of the game from Unity's Roll-a-ball tutorial.

The player is a ball. Using the arrow keys (or WASD keys) to navigate, you must collect all 80 points in the maze, while avoiding all four red enemy balls. A collision with an enemy ball will kill you. You will win if you manage to collect all points without dying.

Each ball has its own path it follows.

In the maze, there are green spots. These are safe spaces, because the enemy balls don't travel there.

## Components

- **Rigidbody.** This is a component that I added to all moving objects in the scene. The player ball and all enemy balls have this component. What does this is to allow the object to be affected by physics.
- **Script.** This is an important component, because essentially, we control the game from these. I put all code logic into scripts I attach to each object, that allow them to do certain things like, move when I press a key, do this when I click, or disappear upon collision, among many others.
- **Canvas Group.** I used this component for the UI. What this does is it allows me to hide or show a UI by changing its alpha and block raycasts option. By setting the alpha to 0 and set it to not block raycasts, all its child UI objects will effectively be hidden. Doing the opposite would show them.

# Scripts

## Camera

This script makes the camera follow the player ball as it moves in its x and z axis. I used Vector3.SmoothDamp so that the movement of the camera is smooth, with a bit of some elasticity to it.

```
public GameObject player;
private Vector3 velocity = Vector3.zero;
private float fixedY;

private void Start()
{
        fixedY = transform.position.y;
}

void LateUpdate()
{
        Vector3 newPos = Vector3.SmoothDamp(transform.position, player.transform.position, ref velocity,
            0.1f);
        newPos.y = fixedY;
        transform.position = newPos;
}
```

## Player

The player ball gets pushed with a force by GameObject.AddForce, but only if the player hasn't picked up all the pickups yet, and if it should be asking inputs, and if it isn't dead yet. If the player collides with a pickup, it adds to its points and makes the pickup disappear. If instead, it collides with an enemy ball, it dies.

```
private Rigidbody player;
private Vector3 move;
public float speed;
public int points;
```

```
private int totalPickups;
public bool acceptInputs;
public bool isDead;

void Start ()
{
        player = GetComponent<Rigidbody>();
        points = 0;
        acceptInputs = false;
        isDead = false;

        totalPickups = GameObject.FindGameObjectsWithTag("Pick Up").Length;
}

private void FixedUpdate()
{
        if (points < totalPickups && acceptInputs && !isDead)
        {
                float moveHorizontal = Input.GetAxisRaw("Horizontal");
                float moveVertical = Input.GetAxisRaw("Vertical");

                move = new Vector3(moveHorizontal, 0f, moveVertical);

                player.AddForce(move * speed);
        }
}

private void OnTriggerEnter(Collider other)
{
        if (other.gameObject.CompareTag("Pick Up"))
        {
                other.gameObject.SetActive(false);
                points++;
        }

        if ((other.gameObject.CompareTag("Enemy") || other.gameObject.CompareTag("Enemy2")))
        {
                isDead = true;
        }
}
```

## Enemy (1-4)

The enemy class mostly contains its code to move along its path. Based off of this code from the Unity Community Forums. Basically, its route is hardcoded. It starts a Coroutine wherein until a ball reaches its next waypoint, it must move straight to it. Once it has reached it, the waypoint will move to the next one, and the process repeats all over again. This script also makes it so that it ignores collisions with other enemy balls.

```
public float speed;
private Vector3[] waypoints;
private int i;
private const float minDistance = 0f;

private void Start()
{
        i = 0;

        waypoints = new Vector3[] {
                // List of all points to go through
                // Different for each ball
        };

        StartCoroutine(Move());
}

private void GetNewWaypoint()
{
        int newWaypoint = i + 1;

        if (newWaypoint > waypoints.Length - 1)
                newWaypoint = 0;

        i = newWaypoint;

        StartCoroutine(Move());
}

private IEnumerator Move()
```

```
{
        float distance = Vector3.Distance(transform.position, waypoints[i]);

        while (distance > minDistance)
        {
                distance = Vector3.Distance(transform.position, waypoints[i]);

                transform.position = Vector3.MoveTowards(transform.position, waypoints[i], speed *
                     Time.deltaTime);

                if (distance <= minDistance)
                {
                        GetNewWaypoint();
                }

                yield return null;
        }
}

private void OnCollisionEnter(Collision collision)
{
        if (collision.gameObject.CompareTag("Enemy2") || collision.gameObject.CompareTag("Enemy"))
        {
                Physics.IgnoreCollision(collision.collider, GetComponent<Collider>());
        }
}
```

## GUI Handler

This updates the UI. Since I'm not that familiar with how the UI works in Unity yet, I've decided to use states via a counter and if functions. The first state, state 0, is the start screen. On click, the instructions disappear, the state changes to 1, the playing state, the player starts accepting inputs, and the timer runs.

The playing state, state 1, checks for the number of points picked up by the player and the current time, then displays it. If it finds that it has indeed picked up all points, then it moves on to state 2.

State 2, the win-ending state, displays the ending message for when a player wins. State 3 is similar but displays the message for when the player loses instead.

```
public Text ui;
private Player player;
private float totalPickups;
private float pickedUp;
public Image overlay;
public Text endingText;
public Text endingDescription;
public CanvasGroup ending;
public CanvasGroup start;
private int state;
private float currentTime;
private float startTime;

private void Start()
{
        state = 0;
        player = GameObject.Find("Player").GetComponent<Player>();
        totalPickups = GameObject.FindGameObjectsWithTag("Pick Up").Length;
}

private void Update()
{
        if (player.isDead && state < 2)
        {
                state = 3;
        }

        if (state == 0)
        {
                if (Input.GetMouseButton(0))
                {
                        start.alpha = 0f;
                        start.blocksRaycasts = false;
                        state++;
                        player.acceptInputs = true;
                        startTime = Time.time;
```

```
            }
        }
        if (state == 1)
        {
            while (totalPickups <= 0)
                totalPickups = GameObject.FindGameObjectsWithTag("Pick Up").Length;

            pickedUp = player.points;
            currentTime = Time.time - startTime;

            ui.text = string.Format("Time: {0:0.00}s{1}Points: {2} / {3} ({4:0.00}%)", currentTime,
                Environment.NewLine, pickedUp, totalPickups, pickedUp / totalPickups * 100);

            if (pickedUp >= totalPickups)
                state = 2;
        }
        else if (state == 2)
        {
            endingDescription.text = string.Format("You finished the game in {0:0.00}
                seconds!{1}{1}Click to start a new game!", currentTime, Environment.NewLine);
            endingText.text = "You Won! :D";

            ending.alpha = 1f;
            ending.blocksRaycasts = true;
        }
        else if (state == 3)
        {
            endingDescription.text = string.Format("It's okay, you got {0} of {1} points in {2:0.00}
               seconds tho!{3}{3}Click to start a new game!", pickedUp, totalPickups, currentTime,
               Environment.NewLine);
            endingText.text = "You Died! :(";

            ending.alpha = 1f;
            ending.blocksRaycasts = true;
        }

        if (Input.GetMouseButton(0) && state >= 2)
        {
            SceneManager.LoadScene(SceneManager.GetActiveScene().name);
        }
}
```

## Pick up

This code makes the pickups rotate continuously.

```
void Update()
{
        transform.Rotate(new Vector3(15, 30, 45) * Time.deltaTime);
}
```

## Pick up Manager

Using the pickup prefab, I initialized all 80 pickups via code, instead of creating them directly in Unity

```
public GameObject pickupPrefab;

void Start()
{
        for (int x = -8; x <= 8; x += 2)
        {
                for (int z = -8; z <= 8; z += 2)
                {
                        if (x == 0 && z == 0)
                                continue;

                        GameObject pickupGeneric = Instantiate(pickupPrefab);
                        pickupGeneric.transform.position = new Vector3(x, 0.5f, z);
                        pickupGeneric.transform.Rotate(new Vector3(45, 45, 45));
                        pickupGeneric.transform.localScale = new Vector3(0.25f, 0.25f, 0.25f);
                }
        }
}
```

## Obstacles and Problems I've Encountered

- **This is my first time handling Unity.** The main obstacle is that Unity is brand new to me. I've touched it before briefly, but I didn't pursue it because it was hard for me back then. So everything is pretty much foreign to me. The Unity UI itself, its documentation, the methods it uses, among many, many others. Good thing that I was able to get the help of the Roll-a-Ball tutorial, documentation, forums, stackoverflow, and other people :)
- **Creating the play field.** The map is basically a maze, so I had to manually put in all the walls manually. It didn't help that the x/y/z coordinates are different from what I'm used to, so it definitely made it much more difficult.
- **Adding in the pickups.** In the tutorial, the tutor planted in the pickups manually. Unfortunately, my idea consisted of 80 pickups, which would be tedious if done manually. I had to find a way to do it using code. (Which is pretty helpful since I now know how to instantiate objects using prefab via code!)
- **Using Git.** I also consider myself a newbie in Git. Even after watching the videos, I still feel like actual Git experience is the key to learning Git. With that said:
- **I almost lost my project due to a Git mishap.** Yeah, I ended up changing my Github username while I was doing the project. I thought it was a good idea to rewrite all my previous commits to feature the new name and email. A couple of commands I didn't fully understand from stackoverflow later, my project ended up having TONS of conflicts. I couldn't even Save As a new project because of so much conflict. I had to salvage what I can get and create a new project and repository (hence, the v2 in the name)