

help me make multiple project ideas for my ten month program for game dev

Absolutely — a 10-month program is a great window for doing a series of projects that build your skills step-by-step and show a clear progression. Let's tailor your ideas so you come out with a strong portfolio.

Overall Strategy

You want a **mix of small prototypes + one major capstone**.

Here's a structure that fits 10 months:

Phase	Months	Focus	Output
1. Foundations	1–2	Core mechanics, 2D basics	2–3 mini games
2. Systems & Polish	3–5	Gameplay loops, UI, saving, sound	1 mid-size project
3. 3D & Advanced Design	6–8	3D worlds, physics, AI, shaders	1–2 ambitious prototypes
4. Capstone	9–10	Combine everything into a portfolio piece	1 polished full game

Project Ideas by Phase

Phase 1: 2D Foundations

Goal: Learn your engine (Unity, Unreal, Godot, etc.), player input, collisions, UI.

1. Platformer Prototype

- Simple 2D side-scroller with jumping, collectibles, and enemies.
- Add your own twist — e.g., gravity switching or wall running.

2. Top-Down Shooter

- Learn aiming, enemy spawns, score system.
- Add random level generation for replayability.

3. Puzzle Game

- a. Think *Sokoban*, *Tetris*, or *Portal*-style spatial puzzles.
- b. Great for learning logic systems and level design.

Phase 2: Systems & Polish

Goal: Learn persistence, menus, audio, polish.

4. Narrative Adventure / Visual Novel

- a. Branching dialogue, save/load, basic animations.
- b. Focus on storytelling and UI design.

5. Simulation / Tycoon Lite

- a. Simple economy loop (e.g., café sim, farm, factory).
- b. Emphasize systems thinking and user feedback loops.

Phase 3: 3D & Advanced Topics

Goal: Transition into 3D or deeper systems.

6. 3D Exploration Game

- a. Small environment, first-person or third-person camera.
- b. Experiment with lighting, terrain, and soundscapes.

7. AI Combat Arena

- a. Small map where bots fight or chase the player.
- b. Learn NavMesh, AI states, and combat balance.

8. Multiplayer Prototype

- a. Local or online (start with simple lobby + 2 players).
- b. Could remix one of your earlier 2D games.

Phase 4: Capstone Project

Goal: Combine everything into one standout game.

9. Capstone: “Showcase Game”

- a. Pick your strongest concept (from earlier) and fully polish it.
- b. Add:

- i. Custom art & sound
- ii. Menu + save system
- iii. Difficulty curve & tutorial
- iv. Trailer + itch.io/Steam demo page

Optional Thematic Tracks

If you want your projects to feel cohesive, you can pick a **theme**:

- **Retro Remix:** Each project reimagines an old genre (platformer, shooter, etc.).
- **World-Building:** Each game occurs in the same universe but in different genres.
- **Mechanic Focused:** Each project explores a specific mechanic (gravity, time, AI, music).

Portfolio Tips

- Document each project with screenshots, short gameplay clips, and what you learned.
- Keep your GitHub/itch.io updated monthly.
- Reflect: Write a short devlog post after each phase.

Would you like me to tailor this plan to a specific **engine (Unity, Unreal, Godot, etc.)** and **career goal** (e.g., indie dev, studio job, technical design, art-focused)?

That way I can give more concrete project scopes and tools for each phase.

Platformer Prototype Simple 2D side-scroller with jumping, collectibles, and enemies. Add your own twist — e.g., gravity switching or wall running. rearsch how i can do it and give me step by step how i'd do it

Perfect — let's break this down so you can *actually build* a **2D platformer prototype** with a cool twist (like **gravity switching** or **wall running**).

I'll assume you're using **Unity** since it's the most common for 2D dev, but if you're using **Godot** or **Unreal**, I can adjust it.

GOAL

Create a **2D side-scrolling platformer** where:

- Player moves and jumps.
- There are collectibles and enemies.
- You add a *unique mechanic* (we'll cover both **gravity switching** and **wall running** options).

PHASE 1: Setup the Project (1–2 hours)

1. Create a New 2D Project

- Open Unity Hub → “New Project” → choose **2D Core**.
- Name it: GravityPlatformerPrototype.

2. Organize Folders

Inside the Project window, create:

3. Create a Basic Scene

- Make a new Scene → call it Level_01.
- Create a **Tilemap** or use simple square sprites for ground/platforms.

PHASE 2: Player Controller (2–4 hours)

1. Create the Player Object

- Add a new **Sprite (Square or custom sprite)** → rename to Player.
- Add:
 - **Rigidbody2D** (for physics)
 - **BoxCollider2D** (for collisions)

2. Write the Movement Script

Create a new C# script: PlayerMovement.cs

Attach it to the Player.

```
using UnityEngine;
```

```
public class PlayerMovement : MonoBehaviour
```

```
{
```

```
    public float moveSpeed = 5f;
```

```
    public float jumpForce = 10f;
```

```
    private Rigidbody2D rb;
```

```
    private bool isGrounded;
```

```
    void Start()
```

```
    {
```

```
        rb = GetComponent<Rigidbody2D>();
```

```
    }
```

```
    void Update()
```

```
    {
```

```
        float move = Input.GetAxis("Horizontal");
```



```
rb.velocity = new Vector2(move * moveSpeed, rb.velocity.y);
```

```
if (Input.GetButtonDown("Jump") && isGrounded)
{
    rb.velocity = new Vector2(rb.velocity.x, jumpForce);
}
}
```

```
void OnCollisionEnter2D(Collision2D collision)
{
    if (collision.gameObject.CompareTag("Ground"))
        isGrounded = true;
}
```

```
void OnCollisionExit2D(Collision2D collision)
{
    if (collision.gameObject.CompareTag("Ground"))
        isGrounded = false;
}
}
```

3. Test & Adjust

- Tag your ground as Ground.
- Adjust moveSpeed and jumpForce until it feels right.

PHASE 3: Level Elements (2–3 hours)

1. Add Collectibles

- Create a coin sprite or placeholder.
- Add **CircleCollider2D** (set to “isTrigger”).
- Tag

make it more like a reasrch paper

1. Introduction

Platformers are among the most established genres in video game history, characterized by side-scrolling movement, jumping mechanics, and the navigation of obstacles. Games such as *Super Mario Bros.*, *Celeste*, and *VVVVV* demonstrate how player movement systems can provide depth and replayability through mechanical mastery.

This research project aims to prototype a platformer that retains core elements of the genre but introduces a **gravity-switching mechanic**. This twist challenges traditional movement expectations and level structure by allowing the player to invert gravity at will, creating vertical exploration and puzzle-like decision-making.

2. Objectives

The primary objectives of this prototype are:

1. To design and implement basic 2D platformer controls (movement, jumping, collisions).
2. To integrate a gravity-switching mechanic that dynamically changes the direction of gravity and affects all physics-based objects.
3. To analyze how this mechanic influences level design, player flow, and difficulty.
4. To create a functional, polished prototype suitable for playtesting and iteration.

3. Methodology

3.1 Tools and Framework

- **Game Engine:** Unity (Version 2022 LTS or later)
- **Programming Language:** C#
- **Art Assets:** 2D sprites from OpenGameArt or original pixel art
- **Physics System:** Unity’s 2D Physics (Rigidbody2D, Collider2D)
- **Version Control:** Git/GitHub for source management

3.2 Development Process

The project follows a lightweight iterative development process inspired by Agile prototyping:

Phase	Description	Duration
1. Planning & Research	Study reference games (e.g., <i>Celeste</i> , <i>VVVVV</i>). Outline core features and create basic design document.	Week 1
2. Prototype Mechanics	Implement player movement, jumping, collisions, and gravity switching in Unity.	Weeks 2–3
3. Level Design & Feedback	Build two short levels to test gameplay flow and player learning curve.	Weeks 4–5
4. Playtesting & Iteration	Gather feedback from peers, identify difficulty spikes or usability issues, refine mechanics.	Weeks 6–7
5. Polish & Documentation	Add audio, UI, particle effects, and compile development report.	Weeks 8–9

3.3 Step-by-Step Implementation

Step 1: Player Controller Setup

- Create a Player GameObject with:
 - Rigidbody2D (Dynamic)
 - BoxCollider2D
 - PlayerController.cs script
- Implement horizontal movement via `Input.GetAxis("Horizontal")`.
- Apply velocity to Rigidbody2D using `rb.velocity = new Vector2(move * speed, rb.velocity.y)`.

Step 2: Jump Mechanic

- Detect ground contact with a Raycast or `OnCollisionStay2D`.
- Apply upward force on jump press (`rb.AddForce(Vector2.up * jumpForce, ForceMode2D.Impulse)`).
- Limit to one jump unless player is grounded.

Step 3: Gravity Switching Mechanic

- Create a toggle input (e.g., Spacebar or “G” key).
- Invert the gravity scale of the Rigidbody2D when activated:

