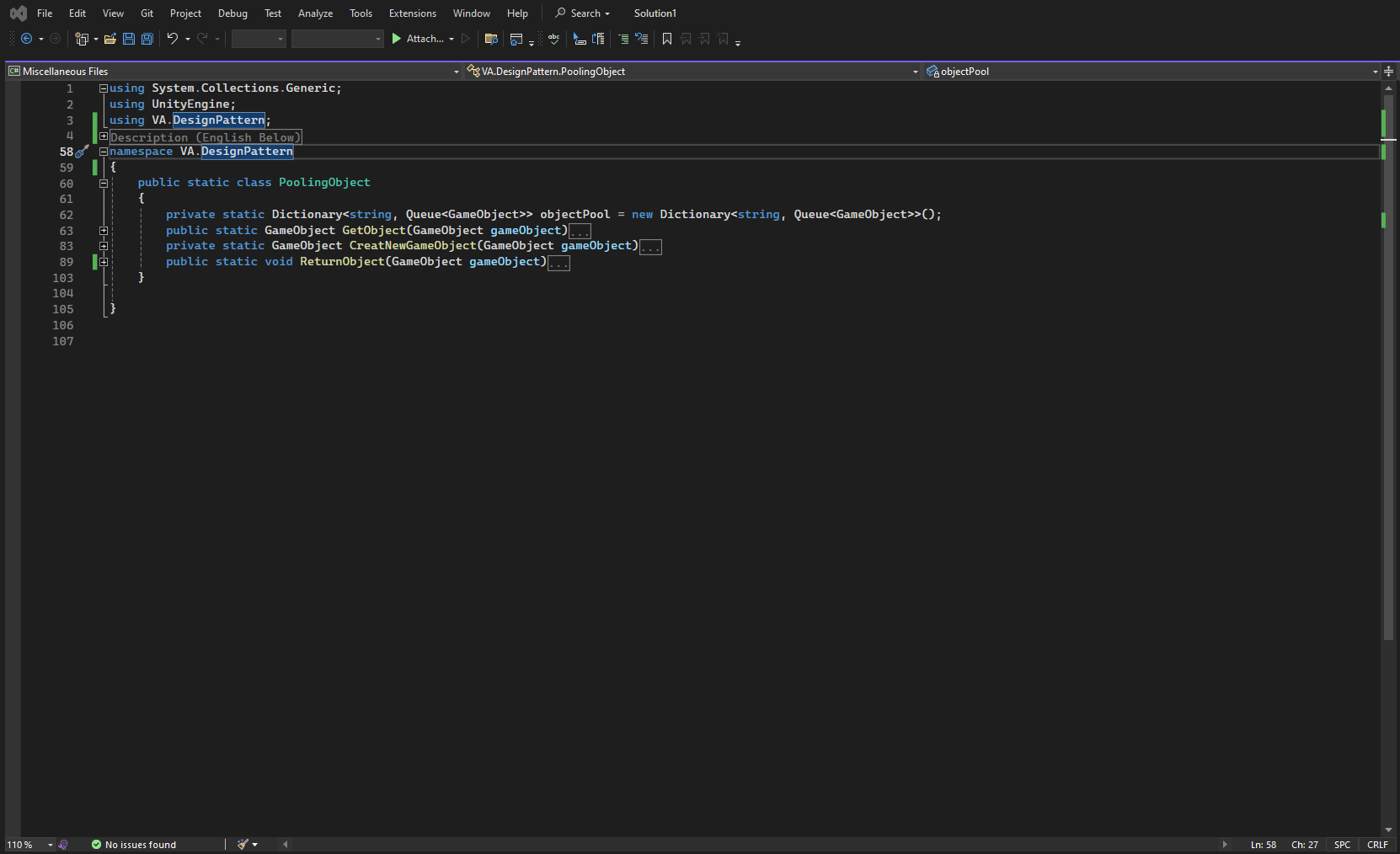
Tối ưu hiệu năng trong Unity là một chủ đề rộng, bắt đầu bằng cách tập trung vào các yếu tố chính sau:

1. Tối ưu hóa CPU

* Tránh sử dụng Update() ở mọi script:
  + Nguyên nhân:
    - Update được gọi mỗi frame (60 lần/giây nếu máy chạy ở 60 FPS), càng nhiều hàm update thì CPU sẽ phải duyệt qua tất cả các hàm update. Mỗi lần gọi một hàm update, unity sẽ thực hiện context switching giữa nhiều script (gây tiêu tốn tài nguyên của CPU). Các hàm update cũng là chạy song song với nhau, khó để theo dõi hàm nào đang gây tiêu tốn tài nguyên
    - context switching: mỗi lần Update của một script được gọi, CPU phải thực hiện quy trình Chuyển quyền xử lý sang Script đó (nạp bộ nhớ, lấy ttin hàm cần thực hiện, việc này khá tốn tài nguyên của CPU) -> thực thị logic trong update -> quay lại xử lý các script khác trong vòng lặp game.
  + Các khắc phục:
    - Dùng sự kiện (Event, Coroutine) thay vì kiểm tra liên tục.
    - Gộp các Update() vào một class quản lý duy nhất (ví dụ: dùng System.Action để quản lý callback).
* Giảm Draw Calls bằng Batching
  + Nguyên nhân:
    - Draw call là một lệnh CPU gửi đến GPU để yêu cầu render ra 1 đối tượng 3D trên màn hình. Một đối tượng có nhiều material khác nhau, mesh khác nhau thì sẽ tạo ra càng nhiều draw call
  + Cách khắc phục
    - Dùng **Static Batching** cho object tĩnh (Static checkbox).
    - Dùng **Dynamic Batching** cho object nhỏ (< 300 vertices).
    - Dùng **GPU Instancing** cho object có cùng material để giảm draw calls.
  + Hint:
    - **Static Batching** cho môi trường (tường, sàn, cây tĩnh).
    - **Dynamic Batching** nếu có nhiều object nhỏ di chuyển.
    - **GPU Instancing** cho object lặp lại như NPC, viên gạch.
* Object Pooling để tái sử dụng GameObject
  + Nguyên nhân:
    - Khi bạn liên tục tạo và hủy object bằng Instantiate() và Destroy(). **Tốn CPU** để khởi tạo và thiết lập object mớ. **Tạo rác bộ nhớ** → Unity sẽ kích hoạt Garbage Collector **(GC)** để dọn dẹp, GC sẽ gây giật lag khi chạy.
  + Cách khắc phục:
    - Tránh Instantiate() và Destroy() liên tục.
    - Dùng **Object Pooling**, tạo sẵn object và bật/tắt khi cần. (Đã có Pooling Object trong dự án, đọc mô tả trong script để biết thêm chi tiết)



* + Unity có OnDisable() và OnEnable() để quản lý tái sử dụng object.
* Tối ưu hóa vật lý (Physics Optimization)
  + Nguyên nhân:
    - Mỗi khi hai Collider tương tác, hệ thống vật lý phải kiểm tra xem chúng có va chạm không. Nếu không kiểm soát, Unity sẽ phải kiểm tra tất cả các Layer với nhau, gây quá tải CPU.
    - **FixedUpdate()** là nơi Unity xử lý hệ thống vật lý, nhưng nó **không chạy theo FPS**, mà chạy theo **Time.fixedDeltaTime** (mặc định là 0.02s, tức 50 lần/giây).
  + Giải pháp:
    - Điều chỉnh **Collision Matrix** để tránh tính toán va chạm không cần thiết.
    - Giảm tần suất FixedUpdate() nếu không cần cập nhật vật lý quá nhanh.
    - Dùng **Rigidbody Interpolation** thay vì tăng tần suất cập nhật vật lý.
* Giảm GC Alloc (Garbage Collection)
  + Nguyên nhân:
    - Trong Unity, mỗi khi bạn **cấp phát bộ nhớ mới trên heap**, nó sẽ tạo **GC Alloc**. Khi lượng bộ nhớ này vượt quá giới hạn, Unity **tạm dừng game** để dọn rác (**GC collection**), gây ra giật lag.
    - Những nguyên nhân phổ biến gây GC Alloc cao:
      * Tạo đối tượng mới liên tục trong Update()
      * Sử dụng string và string concatenation (+)
      * **Dùng Linq hoặc foreach** (có thể tạo bộ nhớ tạm)
      * Dùng Instantiate / Destroy liên tục thay vì Object Pooling
      * Dùng GetComponent, Find, SendMessage quá nhiều
  + Cách khắc phục:
    - Hạn chế dùng new trong Update(), thay vào đó dùng object tái sử dụng.
    - Dùng List<T>.Clear() thay vì tạo list mới.
    - Dùng StringBuilder thay vì nối chuỗi (+ hoặc string.Format), hoặc tối ưu hơn nữa, có thể dùng TextMeshPro do TMP có tối ưu riêng.
* HINT: Có thể mở **Profiler** (Window > Analysis > Profiler) và xem CPU Usage, tập trung vào Rendering và Scripts

1. Tối ưu hóa GPU

* Giảm Draw Calls bằng Batching:
  + Đã nói ở phần trên
  + Cách kiểm tra draw calls:
    - **Mở Stats** (trên Game View) → Xem “Batches” (số lượng draw call)
    - **Dùng Frame Debugger** (Window → Analysis → Frame Debugger) để xem chi tiết cách Unity vẽ từng object.
* Giảm sử dụng shader quá nặng:
  + Các loại shader khác nhau sẽ tốn tài nguyên khác nhau
  + Tối ưu shader bằng cách:
    - **Dùng Mobile Shader** thay vì Standard Shader
    - Giảm số lượng **texture sample** trong Shader
    - **Dùng Unlit Shader** cho đối tượng không cần ánh sáng.
* Tối ưu hệ thống LOD (Level of Detail) (Khi ở xa, không cần hiển thị chi tiết)
  + Dùng LOD Group (component LOD Group trong unity)
  + Dùng Impostor Texture (chỉ hiển thị ảnh tĩnh khi vật thể ở rất xa).
* Nén ảnh:
  + Các định dạng nén phổ biến: ASTC, ETC

1. Tối ưu hóa bộ nhớ (RAM & VRAM)
2. Tối ưu hóa hiệu suất tổng thể
3. ARN/Crash
4. Sử dụng bộ nhớ
5. Tối ưu bộ nhớ vật lý