BÁO CÁO ĐỒ ÁN

**Tên đồ án: Fox Brick Battle**

**Môn học:** Lập trình mạng căn bản

**Lớp:** NT106.P12

**GVHD:** Đặng Lê Bảo Chương

**THÀNH VIÊN THỰC HIỆN (Nhóm 18):**

|  |  |  |
| --- | --- | --- |
| **STT** | **Họ và tên** | **MSSV** |
| 1 | Phan Đình Khải | 23520678 |
| 2 | Lê Chánh Ân | 23520007 |
| 3 | Vũ Duy An | 23520034 |

|  |
| --- |
| **Điểm tự đánh giá** |
|  |

**ĐÁNH GIÁ KHÁC:**

|  |  |
| --- | --- |
| Tổng thời gian thực hiện (tính cả Báo cáo) | 21/10/2024 – 10/12/2024 |
| Phân chia công việc | Phan Đình Khải (GitHub: khaipd18): Phụ trách phần Authentication & Lobby trong UGS. Làm slide & *Sequence diagram.* Tổng hợp & viết báo cáo.  Lê Chánh Ân (GitHub: LeChanhAn): Hiệu ứng bắn, thanh máu của nhân vật. Thuyết trình slide, làm *Mô hình phân rã chức năng* và *Sequence diagram.* Tổng hợp báo cáo.  Vũ Duy An (GitHub: anvu5437): Phụ trách phần Relay (gồm cả NetCode for GameObject) trong UGS. UI trò chơi. Di chuyển, bắn, thanh máu của nhân vật. Tổng hợp báo cáo. |
| Ý kiến *(nếu có)*  + Khó khăn  + Đề xuất, kiến nghị |  |

Phần bên dưới của báo cáo này là báo cáo chi tiết của nhóm thực hiện

MỤC LỤC

[A. BÁO CÁO CHI TIẾT 2](#_Toc183546901)

[1. Giới thiệu Đồ án: 2](#_Toc183546902)

[a. Ý tưởng & Nội dung 2](#_Toc183546903)

[b. Nền tảng & Công cụ sử dụng 2](#_Toc183546904)

[2. Tổng quan về Unity Gaming Services (UGS) 3](#_Toc183546905)

[a. Authentication 3](#_Toc183546906)

[b. Lobby 3](#_Toc183546907)

[c. Relay & NetCode for GameObjects (NGO) 3](#_Toc183546908)

[3. Mô hình phân rã chức năng 3](#_Toc183546909)

[a. Chức năng 3](#_Toc183546910)

[b. Mô hình phân rã chức năng 4](#_Toc183546911)

[c. Mô tả 4](#_Toc183546912)

[4. UGS Sequence Diagram 4](#_Toc183546913)

[a. Chức năng 4](#_Toc183546914)

[b. Sequence Diagram 5](#_Toc183546915)

[c. Mô tả cách hoạt động của UGS trong Đồ án 5](#_Toc183546916)

[5. Demo sản phẩm 8](#_Toc183546917)

[B. TÀI LIỆU THAM KHẢO 8](#_Toc183546918)

# BÁO CÁO CHI TIẾT

## Giới thiệu Đồ án:

### Ý tưởng & Nội dung

* Ý tưởng: Lấy cảm hứng đồ họa 2D của các tựa game cổ điển với cơ chế bắn súng như game Gunny.
* Nội dung:
  + Là tựa game ném gạch 2 người chơi được tạo nên trên nền tảng Unity.
  + Sử dụng Unity Gaming Services (UGS) để thiết lập & đồng bộ kết nối giữa các người chơi.

### Nền tảng & Công cụ sử dụng

* Unity: Là nền tảng phát triển trò chơi mạnh mẽ, hỗ trợ tạo các ứng dụng đa nền tảng (PC, mobile, console) với giao diện đồ họa trực quan, tích hợp sẵn các công cụ hỗ trợ như hệ thống vật lý, renderer, và AI.
* Unity Gaming Services (UGS): cung cấp các dịch vụ backend toàn diện, bao gồm quản lý người chơi, hệ thống matchmaking, lưu trữ dữ liệu đám mây, và các công cụ phân tích, giúp hỗ trợ xây dựng trò chơi trực tuyến hiệu quả và tối ưu hóa trải nghiệm người chơi.

## Tổng quan về Unity Gaming Services (UGS)

### Authentication

* **Unity Authentication** cung cấp các hình thức đăng nhập khác nhau (ẩn danh, qua ứng dụng bên ngoài,…) giúp cho ứng dụng game có thể định danh được người chơi để cung cấp các tính năng và dịch vụ nhằm đảm bảo tính bảo mật, nhất quán và an toàn trong mọi hành động.

### Lobby

* **Dịch vụ Lobby** cung cấp một cách để người chơi khám phá và kết nối với nhau nhằm thực hiện nhiều kịch bản chơi game đa người. Một số ví dụ phổ biến:
  + Chia sẻ mã tham gia (join code) với bạn bè để họ có thể kết nối trực tiếp vào phiên chơi.
  + Tạo một lobby riêng tư và gửi lời mời đến danh sách bạn bè trong trò chơi.
  + Sử dụng tính năng Quick Join để tìm bất kỳ trận đấu nào có sẵn và tham gia ngay lập tức.
* Lobby có thể tồn tại trong suốt thời gian của phiên chơi, cung cấp cơ chế để người dùng tham gia lại phiên chơi hiện có,…

### Relay & NetCode for GameObjects (NGO)

* **Unity Relay** cung cấp cách làm để các nhà phát triển trò chơi tăng cường khả năng kết nối giữa người chơi một cách an toàn bằng cách sử dụng quy trình làm việc dựa trên mã tham gia (join code) mà không cần bên thứ ba. Dịch vụ Relay cung cấp kết nối thông qua một máy chủ Relay đóng vai trò như một proxy chung.
* Netcode for GameObjects (NGO) là một gói (package) Unity cung cấp các khả năng mạng cho quy trình làm việc dựa trên GameObject và MonoBehaviour.

## Mô hình phân rã chức năng

### Chức năng

* Mô hình này chia hệ thống thành các chức năng con, giúp người dùng dễ hiểu cách hoạt động của ứng dụng, hỗ trợ truy cập và thao tác trên từng chức năng một cách thuận tiện.
* Đảm bảo ứng dụng dễ bảo trì và mở rộng.

### Mô hình phân rã chức năng

Ảnh có chứa biểu đồ, bản phác thảo, Bản vẽ kỹ thuật, Kế hoạch

Mô tả được tạo tự động

### Mô tả

## UGS Sequence Diagram

### Chức năng

* Mô tả luồng tương tác giữa người chơi, client, và UGS server, bao gồm xác thực người chơi, tạo và quản lý phòng chơi, đồng bộ dữ liệu trò chơi, và gửi thông báo kết quả.
* Sơ đồ giúp trực quan hóa quy trình và đảm bảo các bước xử lý được thực hiện chính xác.

### Sequence Diagram

Ảnh có chứa văn bản, Song song, biểu đồ, số

Mô tả được tạo tự động

### Mô tả cách hoạt động của UGS trong Đồ án

* Ban đầu, khi người chơi bắt đầu vào game, sẽ tiến hành khởi tạo dịch vụ của Unity (Authentication, Lobby, Relay,…).

1. Authentication (Anonymous Authentication)

* Trong đồ án sử dụng Anonymous Authentication. Khi người chơi đăng nhập lần đầu vào game/quay trở lại game, Unity Authentication sẽ tạo các thông tin cho người chơi *PlayerId, Access Token.* Trong đó:
  + PlayerId:
    - Là mã định danh duy nhất cho người chơi.
    - Nếu đăng nhập ẩn danh (Anonymous):
      * PlayerId được liên kết với thiết bị hiện tại. Mỗi thiết bị sẽ có một PlayerId riêng nếu không sử dụng tài khoản.
      * Nếu người chơi gỡ cài đặt hoặc xóa dữ liệu game, PlayerId sẽ thay đổi.
    - Nếu đăng nhập bằng tài khoản (email, Google, Apple, v.v.):
      * PlayerId được liên kết với tài khoản người chơi. Người chơi có thể đăng nhập từ nhiều thiết bị và sử dụng cùng PlayerId.
  + Access Token:
    - Là một chuỗi mã hóa được tạo ra để xác thực người chơi trong các dịch vụ khác của Unity (ví dụ: Relay, Lobby, v.v.).
    - Có thời hạn sử dụng (3600 giây) và sẽ cần được làm mới (refresh token) sau khi hết hạn.

1. Lobby
   1. Tạo Lobby
   * Khi người chơi nhấn chọn nút Host, hệ thống sẽ khởi tạo dữ liệu người chơi, bao gồm *PlayerId, Gamertag*, và các thông tin liên quan. Sau đó, quá trình tạo Lobby được thực hiện với các bước sau:
     + Thông tin cấu hình Lobby bao gồm:
       - Tên Lobby.
       - Số lượng người chơi tối đa.
       - Trạng thái riêng tư (public/private).
       - Dữ liệu liên quan đến Lobby (ví dụ: màn chơi được chọn).
       - Dữ liệu người chơi (host).
     + Các thông tin này sẽ được gửi đến dịch vụ Lobby Service của Unity để tạo Lobby.
   * Sau khi tạo Lobby thành công, dịch vụ Lobby Service sẽ phản hồi các thông tin sau:
     + LobbyId: Mã định danh duy nhất của Lobby.
     + LobbyCode: Mã phòng để các người chơi khác tham gia.
     + PlayerId của Host: Dùng để xác định người tạo Lobby.
   * Khi Lobby được tạo thành công, hệ thống sẽ tự động thực hiện các cơ chế:
     + Heartbeat (phần 2.4): Gửi tín hiệu định kỳ để duy trì trạng thái hoạt động của Lobby.
     + Refresh (phần 2.5): Cập nhật dữ liệu Lobby (bao gồm thông tin người chơi và trạng thái Lobby).
   1. Tham gia Lobby
   * Khi người chơi nhấn chọn nút Join, hệ thống sẽ khởi tạo dữ liệu người chơi, bao gồm PlayerId, Gamertag, và các thông tin liên quan. Quá trình tham gia Lobby được thực hiện như sau:
     + Lấy LobbyCode do người chơi nhập cùng với dữ liệu người chơi và gửi đến dịch vụ Lobby Service của Unity để yêu cầu tham gia.
     + Nếu tham gia thành công, dịch vụ Lobby Service sẽ phản hồi các thông tin sau:
       - LobbyId: Mã định danh duy nhất của Lobby.
       - LobbyCode: Mã phòng để nhận diện Lobby.
       - Thông tin khác liên quan đến Lobby và người chơi hiện tại.
     + Nếu tham gia không thành công, người chơi sẽ không được vào Lobby.
   * Sau khi tham gia thành công, hệ thống sẽ tự động kích hoạt cơ chế Refresh (phần 2.5) để cập nhật dữ liệu của Lobby.
   1. Sẵn sàng trong Lobby
   * Khi người chơi nhấn nút Ready trong Lobby, giá trị IsReady trong dữ liệu của người chơi sẽ được đặt thành true. Thông tin này sau đó được gửi đến dịch vụ Lobby Service của Unity để cập nhật trạng thái của người chơi.
   * Sau khi dịch vụ Lobby Service cập nhật thành công, hệ thống sẽ phản hồi thông tin hiện tại của Lobby, bao gồm trạng thái Ready của từng người chơi trong phòng.
   * Tiếp theo, hệ thống sẽ kích hoạt cơ chế Refresh (phần 2.5) để đảm bảo các thông tin của Lobby được đồng bộ hóa và cập nhật kịp thời.
   * Về mặt giao diện, người chơi đã sẵn sàng sẽ được hiển thị bằng màu xanh lá cây. Khi toàn bộ người chơi trong Lobby ở trạng thái Ready, nút Start sẽ được hiển thị và chỉ khả dụng cho Host để bắt đầu trò chơi.
   1. Cơ chế Heartbeat
   * Trong Unity, Heartbeat là một cơ chế được sử dụng để duy trì trạng thái hoạt động của Lobby, tránh việc chuyển sang trạng thái không hoạt động (inactive) khi không có bất kỳ cập nhật nào trong vòng **30 giây**.
   * Cách hoạt động: Cơ chế Heartbeat được kích hoạt ngay từ thời điểm Lobby được tạo và sẽ duy trì cho đến khi Lobby bị đóng. Cứ mỗi **6 giây**, hệ thống sẽ gửi một tín hiệu Heartbeat tới dịch vụ Lobby Service của Unity để xác nhận rằng Lobby vẫn đang hoạt động.
   1. Cơ chế Refresh
   * Cơ chế Refresh là quá trình làm mới dữ liệu của Lobby, đảm bảo các thông tin như số lượng người chơi và trạng thái của từng người chơi luôn được cập nhật.
   * Cách hoạt động:
     + Cơ chế này gửi yêu cầu tới dịch vụ Lobby Service của Unity để lấy dữ liệu mới nhất của Lobby từ máy chủ (không phải dữ liệu cục bộ hiện tại).
     + Sau khi nhận được phản hồi, hệ thống kiểm tra xem dữ liệu nhận được có mới hơn dữ liệu hiện tại không. Nếu có, dữ liệu Lobby sẽ được cập nhật.
       - Duyệt qua toàn bộ thông tin của người chơi trong Lobby để kiểm tra và cập nhật các thay đổi, chẳng hạn như thêm người chơi mới, cập nhật trạng thái Ready, hoặc các thông tin khác.
       - Giao diện của Lobby cũng sẽ được điều chỉnh tương ứng (ví dụ: hiển thị người chơi mới, thay đổi trạng thái hiển thị của người chơi,...).
   * Tần suất hoạt động: Cơ chế này được thực hiện **mỗi giây một lần** để đảm bảo dữ liệu của Lobby luôn đồng bộ và chính xác.
2. Relay & NetCode for Gameobject:
   1. **Relay (Unity Relay)**

**Relay** là một dịch vụ do Unity cung cấp để hỗ trợ các game nhiều người chơi (multiplayer) kết nối dễ dàng và ổn định qua internet. Thay vì yêu cầu người chơi phải kết nối trực tiếp với nhau, Unity Relay giúp giảm thiểu vấn đề tường lửa (firewall) và NAT (Network Address Translation), giúp các client kết nối với server qua một dịch vụ trung gian (relay server). Điều này giúp giảm các vấn đề về kết nối mạng mà người chơi có thể gặp phải.

**Cách hoạt động của Relay trong dự án:**

* **Tạo phiên Relay:** Khi một người chơi bắt đầu làm host (máy chủ), dự án sẽ tạo một phiên Relay mới thông qua RelayManager.Instance.CreateRelay(). Điều này sẽ tạo ra một **allocation** (phân bổ tài nguyên trên server Relay), và từ đó nhận được một **join code** (mã tham gia), giúp người chơi khác có thể tham gia vào phiên.
* **Tham gia vào phiên Relay:** Khi người chơi là client, họ sử dụng mã tham gia (join code) để kết nối đến phiên đã được tạo. Phương thức RelayManager.Instance.JoinRelay() sẽ tham gia vào phiên đó và nhận thông tin kết nối cần thiết để kết nối đến server Relay.
* **Thông tin kết nối:** Mỗi phiên Relay sẽ cung cấp thông tin cần thiết như **IP**, **cổng**, **allocation ID**, **connection data**, và **key**. Những thông tin này sẽ được sử dụng trong quá trình thiết lập kết nối giữa client và server (host).
* **DTLS (Datagram Transport Layer Security):** Dự án sử dụng **DTLS** để bảo mật kết nối mạng giữa các client và server. DTLS là một giao thức bảo mật cho UDP, giúp mã hóa và xác thực dữ liệu trong quá trình truyền tải qua mạng. Relay chọn endpoint có kết nối kiểu **"dtls"** để đảm bảo an toàn trong kết nối.
  1. NetCode for GameObjects

**NetCode for GameObjects** là một phần của Unity Multiplayer, được thiết kế để hỗ trợ việc đồng bộ hóa game objects (đối tượng trong game) giữa các client và server trong các trò chơi nhiều người chơi. Nó cho phép xây dựng các trò chơi mạng với khả năng đồng bộ hóa trạng thái của game objects, xử lý các sự kiện mạng và đảm bảo tính chính xác trong game.

**Cách hoạt động của NetCode for GameObjects trong dự án:**

* **NetworkManager và UnityTransport:** **NetworkManager.Singleton** và **UnityTransport** (công cụ giao tiếp mạng) được sử dụng để quản lý kết nối mạng và các phiên trò chơi. Khi người chơi là **host**, sử dụng SetHostRelayData để thiết lập thông tin Relay cho server, và khi người chơi là **client**, sử dụng SetClientRelayData để thiết lập thông tin Relay cho client.
* **ConnectionApprovalCallback:**Phương thức **ConnectionApprovalCallback** giúp xử lý yêu cầu kết nối từ các client. Phương thức này kiểm tra xem client có thể kết nối hay không và tạo ra một đối tượng người chơi (player object) cho client khi kết nối thành công. Đây là cơ chế giúp quản lý người chơi và các đối tượng của họ trong game.
* **NetworkVariable:** Các giá trị mạng như vị trí, trạng thái di chuyển, hay các hành động của nhân vật (ví dụ: **networkPosition**, **networkIsGrounded**, **networkFacingRight**) được lưu trữ trong các **NetworkVariable**. Các giá trị này được đồng bộ hóa tự động giữa server và client. Khi một người chơi thực hiện hành động, server sẽ cập nhật giá trị của các **NetworkVariable** và gửi thông tin này tới các client.
* **NetworkObject:** Các game objects cần đồng bộ hóa giữa các client và server phải được đánh dấu là **NetworkObject**. Đây là lớp đại diện cho các đối tượng cần đồng bộ trong mạng. Trong dự án, nhân vật người chơi có thể là một **NetworkObject**, và trạng thái của nó (như vị trí, hướng di chuyển, trạng thái nhảy) được đồng bộ hóa giữa server và client thông qua các **NetworkVariable**.
* **ServerRpc và ClientRpc:** **ServerRpc** (như **RequestMoveServerRpc** và **RequestJumpServerRpc**) được sử dụng để gửi yêu cầu từ client đến server, ví dụ như khi client di chuyển hoặc nhảy. Các **ServerRpc** này giúp server xử lý các hành động và đồng bộ hóa các thay đổi đến tất cả các client. Ngược lại, **ClientRpc** có thể được dùng để gửi dữ liệu từ server đến client.
* **Câu hỏi không trả lời được trong buổi vấn đáp:**
  + **Chi tiết về kết nối mạng.**

Kết nối mạng trong dự án Unity này sử dụng **Unity Netcode** kết hợp với **Unity Relay**. Đây là cách kết nối được triển khai:

1. **Unity Netcode**: Cung cấp các chức năng để đồng bộ hóa các đối tượng, trạng thái và sự kiện giữa các máy khách (clients) và máy chủ (server) trong môi trường chơi mạng nhiều người. Netcode giúp thiết lập các kết nối giữa máy chủ và máy khách, và đảm bảo các thay đổi trạng thái của các đối tượng (như chuyển động nhân vật) được đồng bộ hóa trong suốt trò chơi.
2. **Unity Relay**: Cung cấp các giải pháp để kết nối máy khách và máy chủ mà không cần phải lo lắng về cấu hình mạng phức tạp (như NAT punch-through). Với Unity Relay, các máy khách có thể tham gia vào một trò chơi thông qua một mã tham gia (join code), và dữ liệu trò chơi có thể được truyền qua một máy chủ Relay trung gian. Điều này giúp giảm thiểu vấn đề về firewall và NAT trong việc kết nối giữa các máy tính.

**Các bước kết nối mạng trong dự án:**

1. **Khởi tạo NetworkManager**: Trong **GameManager.cs**, **NetworkManager.Singleton** được sử dụng để quản lý các kết nối mạng. Đầu tiên thiết lập **ConnectionApproval** thành true để bật tính năng kiểm tra kết nối trước khi đồng ý (**Connection Approval**).
2. **Xử lý Host và Client**:
   * Nếu máy chủ (IsHost), **NetworkManager.Singleton.StartHost()** sẽ được gọi để bắt đầu trò chơi ở chế độ host. Khi đó, Unity Transport sẽ sử dụng thông tin của Relay để thiết lập kết nối.
   * Nếu là máy khách, **NetworkManager.Singleton.StartClient()** sẽ được gọi để kết nối đến máy chủ đang chạy Relay.
3. **Cấu hình Relay**:
   * **Host**:
     + Đầu tiên, **RelayManager.Instance.GetHostConnectionInfo()** lấy thông tin kết nối của host, bao gồm IP, cổng, Allocation ID, và khóa (key).
     + **NetworkManager.Singleton.GetComponent** SetHostRelayData(...) thiết lập các thông số này cho máy chủ.
   * **Client**:
     + Tương tự, máy khách sử dụng **RelayManager.Instance.GetClientConnectionInfo()** để lấy thông tin kết nối và cấu hình cho UnityTransport.
4. **Quá trình kết nối Approval**: Phương thức **ConnectionApproval** trong **GameManager.cs** xử lý việc chấp nhận hoặc từ chối các kết nối từ các client. Nếu không có lý do gì để từ chối**, response.Approved = true** được thiết lập và một đối tượng người chơi sẽ được tạo ra (**response.CreatePlayerObject = true**).

**Các lớp và phương thức quan trọng trong kết nối mạng:**

1. **RelayManager.cs**:
   * Phương thức **CreateRelay(int maxConnection)** tạo ra một phiên Relay với số lượng kết nối tối đa.
   * Phương thức **JoinRelay(string joinCode)** cho phép một client tham gia vào một Relay session đã tồn tại thông qua mã tham gia (join code).
   * Phương thức **GetHostConnectionInfo()** và **GetClientConnectionInfo()** cung cấp thông tin kết nối của host và client, bao gồm IP, cổng, Allocation ID và các dữ liệu kết nối liên quan.
2. **GameManager.cs**:
   * Xử lý khởi tạo kết nối mạng và cấu hình Relay cho host và client.
   * Thiết lập **ConnectionApprovalCallback** để xác nhận các kết nối từ client.
   * Khởi tạo UnityTransport để thiết lập kết nối với Relay.
   * **Giao thức dtls là gì.**

**DTLS** (Datagram Transport Layer Security) là một giao thức được thiết kế để cung cấp bảo mật cho các ứng dụng sử dụng giao thức truyền tải không kết nối, chẳng hạn như UDP (User Datagram Protocol). DTLS được dựa trên giao thức TLS (Transport Layer Security), nhưng được điều chỉnh để hoạt động với các giao thức truyền tải không đáng tin cậy.

**Đặc điểm chính của DTLS:**

1. **Bảo mật**:
   * DTLS cung cấp các tính năng bảo mật tương tự như TLS, bao gồm mã hóa, xác thực, và tính toàn vẹn của dữ liệu.
2. **Tương thích với UDP**:
   * DTLS được thiết kế để xử lý các đặc tính của giao thức UDP như mất gói hoặc gói đến không theo thứ tự.
3. **Khả năng chống tấn công**:
   * DTLS bảo vệ khỏi các loại tấn công như nghe lén (eavesdropping), giả mạo (tampering), và phát lại (replay attack).
4. **Hiệu quả cao**:
   * Do sử dụng UDP, DTLS thường nhanh hơn so với các giao thức dựa trên TCP, đặc biệt trong các ứng dụng yêu cầu độ trễ thấp như VoIP, video streaming, hoặc các ứng dụng thời gian thực khác.

**Cách hoạt động:**

* DTLS thực hiện các bước bắt tay (handshake) tương tự TLS, nhưng có các cơ chế bổ sung để xử lý việc mất gói hoặc trùng lặp gói trong quá trình bắt tay.
* Sau khi bắt tay thành công, tất cả dữ liệu truyền qua đều được mã hóa và bảo mật.
  + **Thời gian mỗi lần Server đồng bộ cho tất cả người chơi (tại sao thời gian đó phù hợp?):**

Thời gian đồng bộ cho tất cả người chơi của dự án được để mặc định là **0.02s** hay **50Hz**  
- Thời gian trên phù hợp vì:

- Đảm bảo mượt mà và độ trễ thấp

- Cân bằng giữa hiệu suất và tài nguyên hệ thống

- Đồng bộ với Unity Netcode và NetworkManager

- Tối ưu băng thông mạng

- Phù hợp với các trò chơi hành động hoặc platformer

## Demo sản phẩm

* Link: [Video Demo sản phẩm](https://drive.google.com/file/d/1gqOqKfpikXe2x1tpXrQkSMYdYrgGhL1m/view?usp=sharing)

# TÀI LIỆU THAM KHẢO

* + 1. [Unity Gaming Services Docs](https://docs.unity.com/ugs/manual/overview/manual/unity-gaming-services-home)
    2. [Multiplayer with Unity's Netcode for Gameobject & UGS - Carl Boisvert Dev](https://www.youtube.com/playlist?list=PLxmtWA2eKdQSf2EXE-tv0lmqmmdDzs0fV)
    3. [Unity 2D Platformer Tutorial For Beginners - Antarsoft](https://www.youtube.com/playlist?list=PLjAb99vXJuCRD04EUp8p2az1ILZbq_ZfY)