ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN
KHOA KHOA HỌC MÁY TÍNH

BÁO CÁO ĐỒ ÁN MÔN THỰC TẠI ẢO

ĐỀ TÀI: Game thám hiểm mê cung thực tế ảo

Lớp: CS527.I21

GVLT: Nguyễn Hoàng Ngân

Nhóm Thực Hiện :

1.Lý Trung Dũng 15520136

2.Huỳnh Bảo Lâm 15520408

3.Vương Khánh Huy 15520323

4.Nguyễn Hoàng Phúc 15520644

TP.Hồ Chí Minh, tháng 05 năm 2018

Mục lục

I. ĐẶT VẤN ĐỀ CÁC MỤC TRONG PROPOSAL

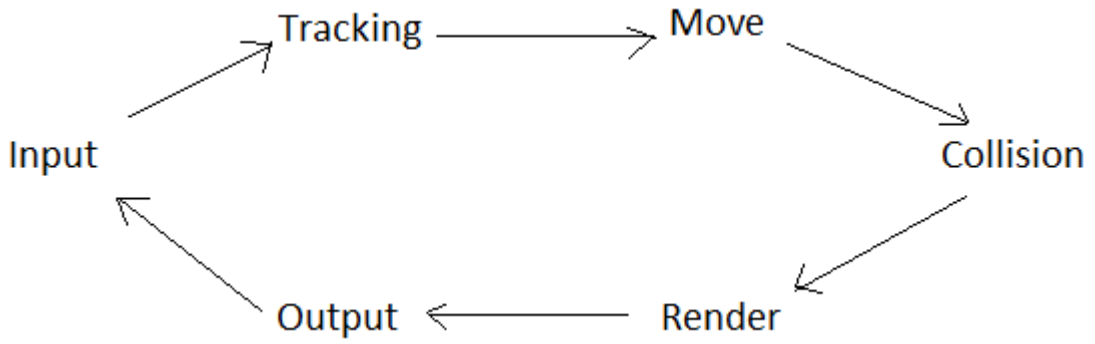
- Ban đầu, trò chơi hướng tới việc sử dụng controller để thực hiện thao tác di chuyển nhân vật và xoay camera, vì để thuận tiện hơn cho việc trải nghiệm mà không cần quá nhiều thiết bị nên người chơi chỉ cần sử dụng VR box đã có thể di chuyển, đứng yên và xoay camera. Trò chơi theo dạng tìm đường thoát với góc nhìn người thứ nhất, có thể bổ sung thêm một số tính năng như thời gian đếm ngược, điều kiện thắng và thua, tạo nhiều vật thể trong mê cung mà người chơi có thể tương tác được,...Thời lượng chơi không quá dài để tránh gây chóng mặt khi dùng kính lâu, vẫn bảo đảm trải nghiệm thú vị cho người dùng. Việc render, hiệu ứng đồ hoạ, số lượng vật thể, được tối giản để các thiết bị tầm thấp vẫn có thể sử dụng.

II. HIỆN THỰC & CÀI ĐẶT

1/ KIẾN TRÚC HỆ THỐNG

SƠ ĐỒ CHU KỲ PHÂN TÁN ĐỂ XỬ LÝ CÁC LUỒNG

- **Sơ đồ:**



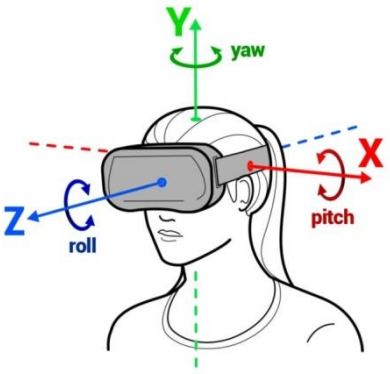
- **Giải thích sơ đồ:**

- Kiểm tra đầu vào của thiết bị chơi game, hệ điều hành có phù hợp hay không, điện thoại có hỗ trợ chế độ VR không.
- Input : các thông số Component của camera (góc quay, toạ độ, ...), âm thanh của trò chơi
- Tracking: kiểm tra component của camera, khi người dùng quay đầu. Sau một khoảng thời gian nhất định thì hệ thống sẽ kiểm tra vị trí mới và cập nhật lại vị trí, góc quay mới này cho camera (cũng như hướng nhìn của người chơi)
- Move: tính toán vị trí mới của người chơi để xử lý di chuyển, xử lý âm thanh di chuyển

- Collision: kiểm tra va chạm khi người chơi va chạm vào vật thể
- Render: dựng hình ảnh các đối tượng để hiện ra màn hình
- Output: trả kết quả component mới của camera, các đối tượng sau khi được render lên màn hình, các âm thanh tương ứng,...

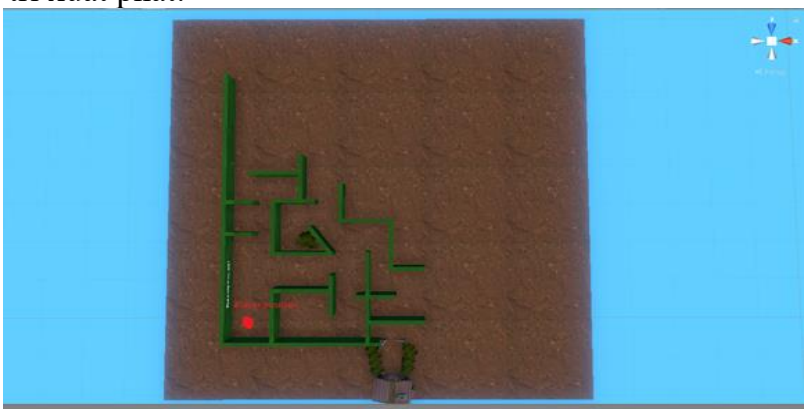
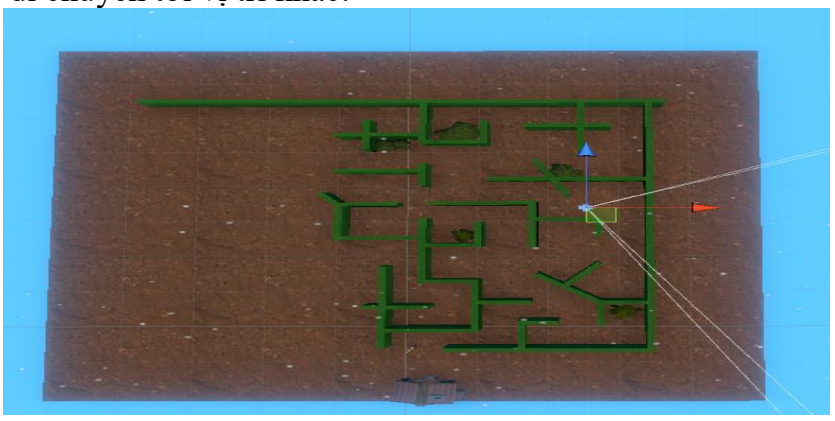
2/ THIẾT LẬP VR (Hỗ trợ Google VR)

- **Tracking:**

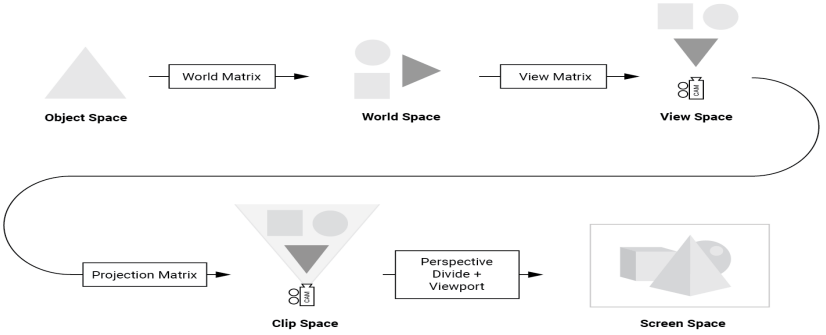


- Để xử lý kiểm tra head tracking thì ta sẽ kiểm tra camera. Trong Unity thì ta sẽ ấn giữ chuột và phím ALT hoặc CTRL để điều khiển sự di chuyển của cái đầu. Ngược lại nếu ta chơi trò chơi trên thiết bị di động thì việc điều khiển cái đầu vẫn chính xác bởi vì class VR.InputTracking sẽ lấy những giá trị sau khi xử lý để cập nhật trong hàm Update hoặc LateUpdate. Tracking bằng cách ta sẽ tạo 2 biến: Vector3 để lưu lại các thông số của đầu (camera) và Quaternion để lưu giá trị góc quay của đầu. Khi người chơi chuyển động cái đầu:
  - Nếu người chơi đang chuyển động cái đầu chuyển sang ngưng chuyển động thì ta sẽ xử lý thiết lập lại góc quay của đầu cho không bị nghiêng và cập nhật lại hướng quay.
  - Nếu người chơi đang trong quá trình chuyển động cái đầu thì ta sẽ tính toán các giá trị toạ độ X, Y, Z của các phép quay Roll hay Yaw sau đó cập nhật lại toạ độ và hướng quay của đầu.

- **Render:**

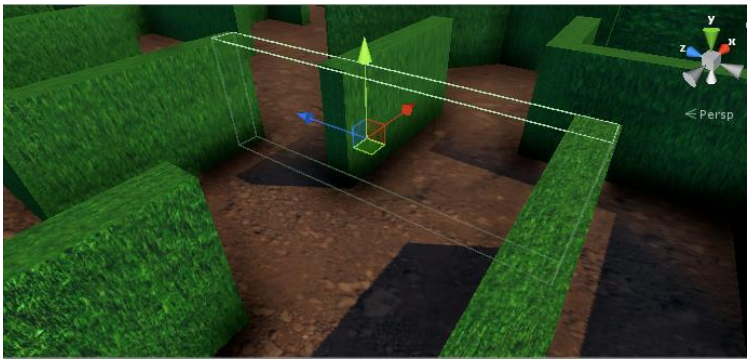
- Để tiến hành render ta cần xác định bán kính từ vị trí của camera đến vị trí của đối tượng(tường,cây,hang rào,…) sau đó so với khoảng cách mà ta thiết lập để render đối tượng nếu đối tượng nằm trong bán kính đó thì sẽ được render ra giao diện và ngược lại. Quá trình này thực hiện cho đến khi kết thúc trò chơi.
- Cách xác định khoảng cách giữa 2 vật trong engine có hổ trợ hàm Vector3.Distance với 2 tham số là vị trí của 2 vật
- Sau đây là 2 ví dụ các đối tượng được render trong bán kính của player
- Khi player ở vị trí xuất phát:
  - 
- Sau khi player di chuyển tới vị trí khác:
  - 

- Ta có thể tìm hiểu thêm quá trình chuyển đổi để đưa đối tượng ra màn hình



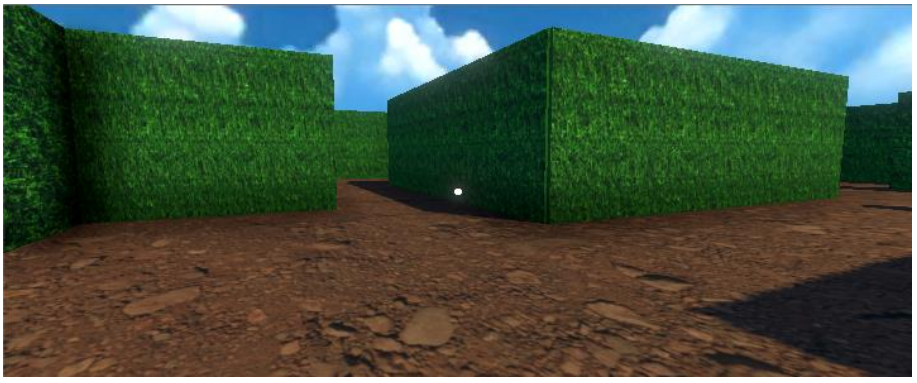
- Ban đầu các đối tượng ở trong ***Object Space*** của riêng chúng. Sau đó biến đổi các đối tượng bằng **World Matrix**, để đưa các vật thể vào ***World Space*** . ***World Space*** là một không gian chung cho vị trí ban đầu, tương đối của các đối tượng. Tiếp theo, biến đổi các đối tượng khỏi ***World Space*** để tới ***View Space*** , với **View Matrix**. Bây giờ các đối tượng được sắp xếp tương đối với quan điểm của chúng ta. Khi ở trong **View Space**, ta có thể chiếu chúng lên màn hình 2D với Projection Matrix, đưa các đối tượng đó vào ***Clip Space*** . Phối cảnh phân chia theo sau, dẫn đến *không gian NDC (tọa độ thiết bị đã chuẩn hóa)* và cuối cùng, **Viewport transform** được áp dụng, dẫn đến ***Screen Space***. Khi ta đang ở ***Screen Space***, ta có thể tạo ra các mảnh cho **render target**

- **Collision:**



- Ta sẽ tạo collider cho các vách tường, nền, cửa, nhà và player. Thiết lập các thông số Rigidbody cho player như mass (khối lượng) là 1, drag (lực kéo) là 50 để tránh tình trạng khi player va vào vách tường bị chịu tác động của các khối vật lí làm cho player di chuyển lệch khỏi các trục như thiết lập ban đầu thì ta sẽ giữ cho player thăng bằng lại.

- **Script (Autowalk):**



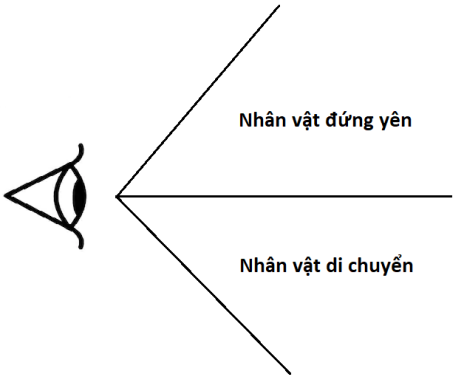
- Muốn player di chuyển để khi đeo kính thực tại ảo, người chơi sẽ trải nghiệm di chuyển trong mê cung. Để di chuyển ta sẽ khởi tạo file c# đặt là autowalk.cs.
- Trong file autowalk.cs sẽ sử dụng thư viện UnityEngine và UnityEngine.Audio
- Đầu tiên khởi tạo mô phỏng tiếng bước đi.
- Chúng ta sẽ khởi tạo tốc độ của người chơi ở Unity phần speed để ta có thể thay đổi tốc độ của player.
- Nếu chơi trên PC người chơi sẽ dừng lại khi click chuột, nếu không chọn thì người sẽ dừng khi chạm vào tường mê cung hay vượt qua ngưỡng góc di chuyển.
- Chúng ta sẽ tạo ra ngưỡng góc nếu nằm trong ngưỡng thì ta sẽ di chuyển tự động.

- Khi người chơi chạm tường hoặc một số vật thể được thiết lập collision thì người chơi sẽ dừng lại.
- Trường hợp là chạy có Triggered thì ta sẽ điều khiển bằng chuột muốn chuyển hướng thì bấm Alt + Xoay chuột.
- Khi di chuyển thì ta dùng Vector3 để cài đặt hướng cho player trong đó ta sẽ nhân giá trị của Vector3 với biến speed và lớp Time và deltaTime để tính khoảng thời gian player di chuyển . Ta sử dụng lớp Quaternion và Euler để điều chỉnh góc quay của player qua ba trục x,y,z của player. Hàm Vector3 và Quaternion giúp cho người chơi có thể di chuyển theo hướng xoay đầu và có thể nhìn theo các hướng để thấy toàn cảnh.

3/ KĨ THUẬT TÂM ĐẮC

- Kỹ thuật: Autowalk

- Việc không sử dụng controller sẽ gây khó khăn trong việc điều khiển di chuyển nhân vật và tương tác với các vật thể. Với thiết lập script Autowalk sẽ giúp nhân vật di chuyển theo ý muốn mà không cần dùng tay cầm. Giả sử có một mặt phẳng nằm ngang phía trước con mắt, nếu mắt hướng về phần mặt trên thì nhân vật sẽ đứng yên, ngược lại mặt dưới nhân vật sẽ di chuyển (có thể thiết lập góc độ khác), ngoài ra còn có thể chỉnh sửa tốc độ di chuyển, âm thanh khi di chuyển và một số setting khác.



- Hướng di chuyển: Di chuyển theo tọa độ x và z (mặt phẳng nằm ngang)

Vector3D direction dựa trên transform trục x và z của mainCamera * vận tốc * thời gian deltaTime = vector hướng di chuyển

- Hướng xoay:

- Một Quaternion rotation sẽ là một Quaternion được áp dụng thuật toán Euler vào Vector3D theo trục y.
- Và như vậy, việc thay đổi trạng thái khi di chuyển của player sẽ là (rotation * direction)

III. PHÂN CÔNG CÔNG VIỆC

- Lý Trung Dũng (leader): Đóng góp ý tưởng, thiết kế giao diện, xử lý hiệu ứng, xử lý một số script liên quan tới kết thúc game, collision, build ứng dụng, viết báo cáo phần I, II, IV
- Huỳnh Bảo Lâm: Đóng góp ý tưởng, Tracking, viết báo cáo phần III tracking
- Vương Khánh Huy: Đóng góp ý tưởng, Render, viết báo cáo phần III render
- Nguyễn Hoàng Phúc: Đóng góp ý tưởng, di chuyển player, viết báo cáo phần III di chuyển, test ứng dụng bằng điện thoại

Các mốc giai đoạn về tiến độ hoàn thành đồ án

- Giai đoạn 1(1/3/2018 đến ngày 15/3/2015): Tìm hiểu, lên ý tưởng về các loại maze trong game và chọn các texture, material để thiết kế maze.
- Giai đoạn 2 (16/3/2018 đến ngày 15/4/2018): Phác thảo maze trên paint. Sau đó thiết kế maze, thêm hiệu ứng, âm thanh,...
- Giai đoạn 3 (16/4/2018 đến ngày 20/5/2018): Xử lý code, hoàn thành phần còn lại của game