ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC BÁCH KHOA
KHOA KHOA HỌC & KỸ THUẬT MÁY TÍNH

- - - - - - - - - - - - - - -

Game Rising Steps

Thái Hoàng Nguyên 1612305

TP. HỒ CHÍ MINH, THÁNG 09/2020

LỜI NÓI ĐẦU

Căng thẳng thần kinh (stress) đang ngày càng trở thành nỗi ám ảnh không loại trừ một ai trong cuộc sống hiện đại. Hệ quả là tạo ra những gánh nặng ảnh hưởng đến chất lượng cuộc sống, công việc. “Rising Steps” ra đời sẽ giúp bạn giải tỏa những căng thẳng sau thời gian làm việc mệt mỏi. Ngoài yếu tố giải trí, tựa game này cũng sẽ giúp người chơi có sự phát triển về nhiều khía cạnh.

Trong báo cáo này, tôi xin trình bày những nghiên cứu, thiết kế và hiện thực cho tựa game “Rising Steps” dựa trên những công nghệ mới nhất. Trò chơi thuộc thể loại chiến thuật thời gian thực (Real-time Strategy).

MỤC LỤC

DANH MỤC HÌNH

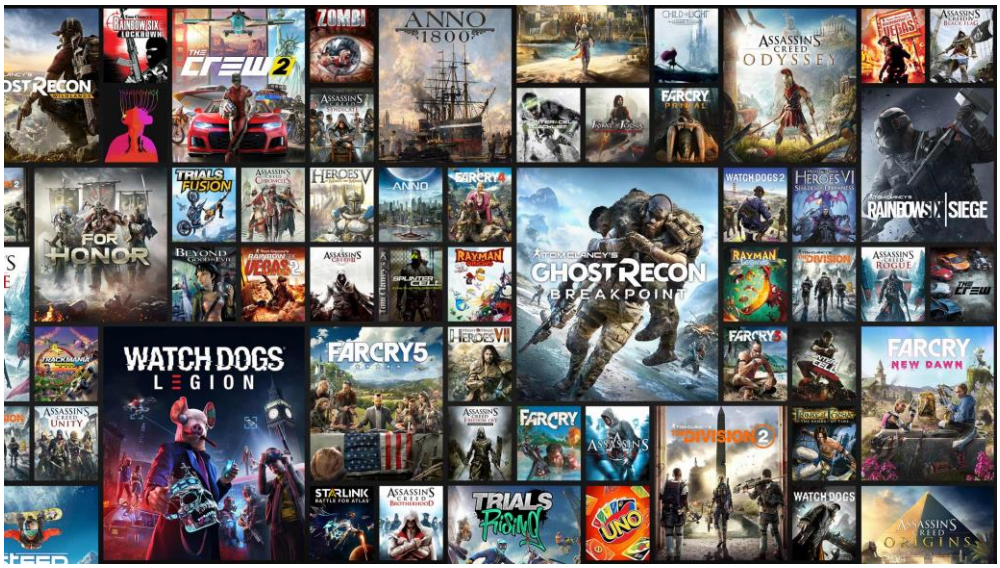
Chương I. Giới thiệu

Chương này giới thiệu một cách tổng quan về trò chơi, trình bày mục tiêu, nhiệm vụ, phạm vi và ý nghĩa của đề tài.

1. Tổng quan:

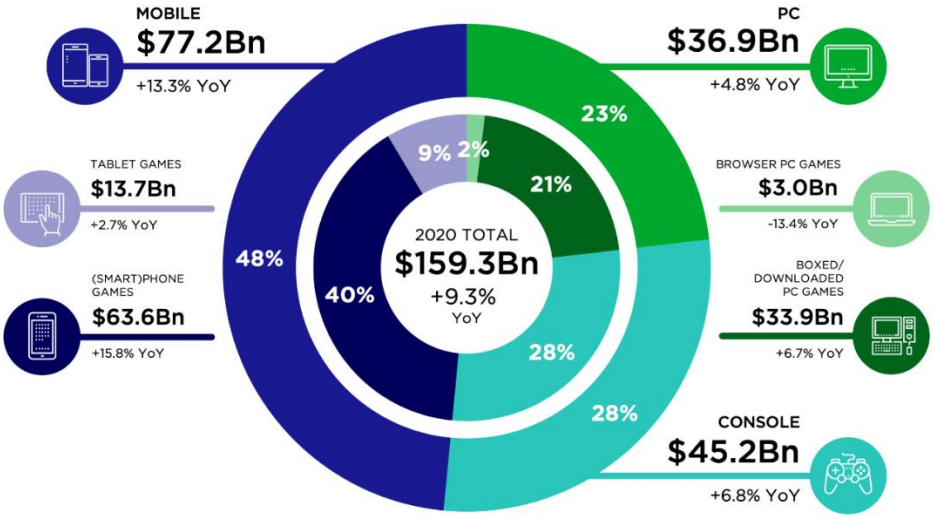
"Game là một hình thức chơi có cấu trúc, thường được thực hiện để vui chơi giải trí, và đôi khi được sử dụng như một công cụ giáo dục". Nhiều trò chơi giúp phát triển rất tốt các kỹ năng, chúng rất phổ biến và hầu như xuất hiện mọi nơi trên thế giới. Cờ vua, cờ tướng,… là những ví dụ điển hình.

Bắt đầu từ những năm 1980, khi thiết bị hiển thị video là các màn hình ra đời, video game xuất hiện và đã trở thành một phần ngày càng quan trọng trong ngành công nghiệp giải trí. Các hệ thống điện tử được sử dụng để chơi trò chơi video gọi là nền tảng.



Hình 1: Một số trò chơi phổ biến trên thị trường

Trong thế giới hiện đại ngày nay, vấn đề stress đang ngày càng trở thành nỗi ám ảnh khi mà con người liên tục bị cuốn vào guồng quay của cuộc sống. Cùng với đó, thị trường game cũng đang phát triển rất nhanh chóng để đáp ứng được phần nào nhu cầu về giải trí, giúp giải tỏa stress trong những thời gian rảnh.



Hình 2: Doanh thu dự báo ngành game năm 2020

Theo ước tính thì trong năm nay, doanh thu ngành game sẽ đạt được đến khoảng 159.3 tỉ $, chiếm một tỉ trọng khá lớn trong nền kinh tế.

Và trong số đó, các dòng game phổ biến nhất là casual game và puzzle game. Những game này thu hút được đông đảo người chơi bởi cách chơi đơn giản, dễ chơi, phù hợp cho mọi đối tượng người chơi và không phải đầu tư nhiều về mặt thời gian.

Trái ngược với những dòng game này thì strategy game ít được phổ biến hơn, mặc dù nó có thể giúp người chơi rèn luyện tư duy chiến thuật rất tốt. Vài ví dụ điển hình như Starcaft, Battle realms. Có thể nhận thấy rằng những ưu điểm của casual game và puzzle game cũng là những hạn chế của strategy game.

Trong đề tài này, tôi sẽ xây dựng một tựa game Real-time Strategy có tên là Rising Steps. Trò chơi được xây dựng bắt đầu từ niềm yêu thích trong lĩnh vực game; qua quá trình tìm hiểu, trải nghiệm và khảo sát các trò chơi chiến lược phổ biến, tôi đã tổng hợp và thiết kế ra nội dung đáp ứng được một số điểm còn hạn chế của strategy game.

Rising Steps sẽ có lối chơi đơn giản, nhưng vẫn mang tính chiến lược và tính giải trí cao, phù hợp với mọi đối tượng. Ngoài ra, trò chơi có thể chơi trên máy tính (windows) hoặc điện thoại (android). Người chơi cần phải tính toán đến nhiều yếu tố để đạt được mục tiêu chiến thắng cũng như tạo ra phong cách chơi của mình. Game lấy bối cảnh người chơi đóng vai trò là tộc

trưởng một bộ lạc, xây dựng lên ngôi làng hùng mạnh của mình, chinh phục các vùng đất khác.

2. Mục tiêu, phạm vi, nhiệm vụ đề tài:

Mục tiêu chính của đề tài là nghiên cứu, thiết kế và hiện thực một chương trình trò chơi chiến lược có tính giải trí cao, đồng thời có lối chơi gần gũi, đơn giản, phù hợp với mọi đối tượng. Qua đó phát hành để tạo ra một sản phẩm trên các cửa hàng ứng dụng trò chơi.

Phạm vi của đề tài là một chương trình trò chơi có thể kết nối mạng lưới nhiều người chơi, bao gồm các tính năng như: xây dựng nên ngôi làng từ các công trình, thu thập và quản lí tài nguyên, huấn luyện quân lính, đem quân đi chiến đấu, phòng thủ các cuộc tấn công, cập nhật theo thời gian thực, tìm đường đi cho quân lính.

Nhiệm vụ của đề tài:

- Tìm hiểu và phân tích một số trò chơi Real-Time Strategy
- Nghiên cứu một số kiến thức được áp dụng để giải quyết các bài toán trong trò chơi.
- Thiết kế nội dung trò chơi.
- Thiết kế kiến trúc chương trình.
- Tìm hiểu về game engine Unity và platform Firebase.
- Thiết kế các lớp đối tượng và hiện thực trò chơi
- Kiểm tra và đánh giá kết quả hiện thực

3. Giới thiệu chung:

a) Game và thể loại game:

Cho dù là game trên máy tính, máy chơi game hay smartphone thì chúng cũng được xếp vào nhiều thể loại dựa theo cách chơi và bản chất của game. Hầu hết mỗi game sẽ thuộc một hoặc nhiều thể loại: Action, Adventure, Role-playing, Simulation, Strategy, Sports, Other notable genres: board game or card game, casual games, digital collectible card game, logic game, MMO, party game, programming game, trivia game…

Game “Rising Steps” thuộc thể loại RTS-Tower Defense.

b) Giới thiệu về dòng game RTS (Real-Time Strategy):

RTS là dòng game *chiến lược thời gian thực*, một thể loại trò chơi điện tử chiến lược mà người chơi không phải đi theo lượt như các trò chơi chiến thuật theo lượt.

Với dòng game này, người chơi sẽ có khả năng xây dựng thành trì cũng như tạo nên 1 đội quân có thể mang đi chinh phạt thành trì của những người chơi khác. Các đơn vị và công trình sẽ tham gia vào một vị trí dưới sự kiểm soát của người chơi để bảo vệ các khu vực của bản đồ và/hoặc phá hủy tài sản của đối thủ.

Điểm nổi bật của RTS chính là yếu tố thời gian thực, ngay cả khi người chơi thoát khỏi game thì mọi hoạt động của trò chơi như thu thập tài nguyên, huấn luyện binh lính, xây công trình vẫn tiếp tục. Bởi vậy đòi hỏi thời gian vào game của người chơi ít hơn, phù hợp với những người có ít thời gian rảnh rỗi.

c) Dòng game TD (Tower Defense):

Game thủ thành là thể loại game mô tả một đám đông đang tìm cách tràn vào lâu đài, tháp chỉ huy,… Tuy nhiên, để đến được vị trí này, chúng sẽ phải vượt qua "trùng trùng lớp lớp" các phương tiện phòng thủ được bố trí tài tình.

4. Bố cục:

Báo cáo được tổ chức thành 10 chương:

- Chương 1: Giới thiệu. Trong chương này, tôi sẽ giới thiệu một cách tổng quan về trò chơi, trình bày mục tiêu, nhiệm vụ, phạm vi và ý nghĩa của đề tài.
- Chương 2: Các công trình liên quan. Trong chương này, tôi sẽ giới thiệu và phân tích một số trò chơi theo thể loại RTS và TD, một số bài toán được đặt ra khi xây dựng game theo thể loại này.
- Chương 3: Kiến thức nền tảng. Trong chương này, tôi sẽ trình bày các kiến thức trong lập trình game, giải thuật áp dụng, các kiến trúc thiết kế, những kiến thức về toán học và vật lí cần nắm vững để có

thể hiểu được kết cấu của trò chơi, cũng như áp dụng để giải quyết những vấn đề phát sinh khi hiện thực.

- Chương 4: Thiết kế trò chơi. Trong chương này, tôi sẽ trình bày những ý tưởng mô tả về game, bao gồm cốt truyện, cách chơi, các đối tượng trong game, các thuộc tính,…
- Chương 5: Thiết kế kiến trúc phần mềm. Trong chương này, tôi sẽ đưa ra bản thiết kế của hệ thống dựa trên mẫu kiến trúc mô hình MVC.
- Chương 6: Giới thiệu Unity engine và platform Firebase. Trong chương này, tôi sẽ giới thiệu về các công nghệ được sử dụng để hiện thực game.
- Chương 7: Hiện thực game. Trong chương này, tôi sẽ trình bày nội dung hiện thực cho các tính năng chính của chương trình.
- Chương 8: Kiểm thử và đánh giá.
- Chương 9: Tổng kết. Trong chương này, các kết quả đã thực hiện sẽ được tổng kết lại, được đánh giá và nhận xét chung, chương này cũng nêu ra hướng phát triển cho trò chơi trong tương lai.

Chương II. Các công trình liên quan

Trong chương này, tôi sẽ giới thiệu và phân tích một số trò chơi theo thể loại RTS và TD, một số bài toán được đặt ra khi xây dựng game theo thể loại này.

1. Khảo sát một số game thể loại RTS và Defense Tower:

a) Clash of Clans:

Hình 3: Một cảnh trong game Clash of Clans

Clash of Clans đặt trong bối cảnh là một thế giới kỳ ảo nơi người chơi là người đứng đầu một ngôi làng. ***Clash of Clans*** buộc người chơi hoàn thành các nhiệm vụ như sử dụng nguồn tài nguyên lấy được từ việc tấn công các người chơi khác thông qua chế độ chiến đấu.

Clash of Clans nhận được nhiều đánh giá tích cực cũng như có sự tăng trưởng mạnh mẽ nhất trong lịch sử trò chơi dành cho thiết bị di động.

b) Empire Warriors: Tower Defense:

Hình 4: Một cảnh trong game Empire Warrior: Tower Defense

Empire Warrior: Tower Defense là trò chơi kết hợp giữa game nhập vai và game chiến đấu thủ thành. ***Empire Warrior: Tower Defense*** đặt trong bối cảnh một thế giới mộng ảo với rất nhiều các chủng tộc. Một ngày nọ, Vương quốc Endia bị tấn công, người chơi sẽ đóng vai trò là một chiến thuật gia xây dựng các tòa tháp, huấn luyện các chiến binh, dẫn dắt các anh hùng và tiêu diệt kẻ thù.

c) Bloons TD Battles:

Hình 5: Một cảnh trong game Bloons TD Battles

Bloons TD Battles là game thủ thành của những chú khỉ ngăn chặn các quả bóng bay đang xâm chiếm vùng đất. Trò chơi có sự đối kháng giữa 2 người chơi, người chơi nào bảo vệ được vùng đất sau khi đối phương đã bị xâm chiếm sẽ là người thắng cuộc.

2. Các bài toán liên quan:

a) Tự động hóa các nhân vật trong game (NPC):

Trong việc tìm đường đi cho một NPC, các biến thể của giải thuật A* search rất thường được sử dụng để quyết định cách thức di chuyển từ điểm A tới điểm B. Không gian tìm kiếm thường là không gian vật lý trong trò chơi.

Để xác định nơi NPC sẽ đi, các phương pháp như cây hành vi hoặc máy trạng thái hữu hạn thường được sử dụng.

b) Tạo ra AI giống như một người chơi:

Trong các game strategy, người chơi cần phải đặt ra và thực hiện nhiều hành động phức tạp liên quan đến nhiều đơn vị. Thách thức này sẽ khó hơn đáng kể so với việc lập kế hoạch trong các board game cổ điển như chơi cờ.

Bởi vì mỗi turn bao gồm những hành động cho tất cả các đơn vị, không gian tìm kiếm có thể cực kì rộng lớn. Đây là vấn đề đối với hầu hết các giải thuật tìm kiếm.

Một cách để xử lí là phân rã vấn đề, để cho các đơn vị tự hành động; chúng ta thực hiện vấn đề tìm kiếm trên mỗi đơn vị riêng biệt. Điều này cũng có nhược điểm là các đơn vị sẽ không thể phối hợp với nhau. Tuy nhiên, việc áp dụng heuristic ở các đơn vị có thể được xây dựng cùng với cách thức này.

Giải pháp tiếp theo là thực hiện lấy mẫu con trong không gian các turn để các giải thuật tìm kiếm có thể áp dụng. Các giải thuật dựa trên sự phân rã này như Monte Carlo Tree Search, Non-linear Monte Carlo. Ý tưởng chính là lấy mẫu không gian turn một cách ngẫu nhiên, sau đó ước tính giá trị của mỗi turn đó. Dựa trên những ước tính này, một mạng neural được train để dự đoán giá trị của turn. Kĩ thuật regression được sử dụng sau đó để tìm kiếm turn với giá trị dự đoán cao nhất.

Chương III. Kiến thức nền tảng

Trong chương này, tôi sẽ trình bày các kiến thức trong lập trình game, giải thuật áp dụng, các kiến trúc thiết kế, những kiến thức về toán học và vật lí cần nắm vững để có thể hiểu được kết cấu của trò chơi, cũng như áp dụng để giải quyết những vấn đề phát sinh khi hiện thực.

1. Lập trình game:

a) Một số thành phần chính:

Assets: tất cả những tài nguyên được sử dụng trong việc phát triển game sẽ được gọi chung là assets. Assets có thể là hình ảnh, mô hình 3D, âm thanh, hiệu ứng,…

Scenes: scene là một cảnh game, chứa môi trường và menu trong game. Đây là nơi thiết lập bố cục cho các đối tượng.

Camera: camera được dùng để thể hiện khung hình, góc nhìn mà người chơi có thể nhìn thấy được.

Light: Light là ánh sáng trong game, ánh sáng xác định màu sắc và độ sáng của môi trường game.

GameObject: đối tượng đại diện cho các assets trong game.

Components: là các thuộc tính được thêm vào GameObject như mô hình 3D, animation, âm thanh,… để tạo ra được đối tượng game mong muốn.

Transform: lưu trữ vị trí, góc quay, tỉ lệ kích thước của một GameObject. Mỗi GameObject luôn có một transform để xác định vị trí của nó trong không gian game.

Rigidbody: GameObject nào chứa rigidbody sẽ chịu tác động của vật lí.

Collider: thành phần collider gắn vào GameObject với mục đích xử lí va chạm.

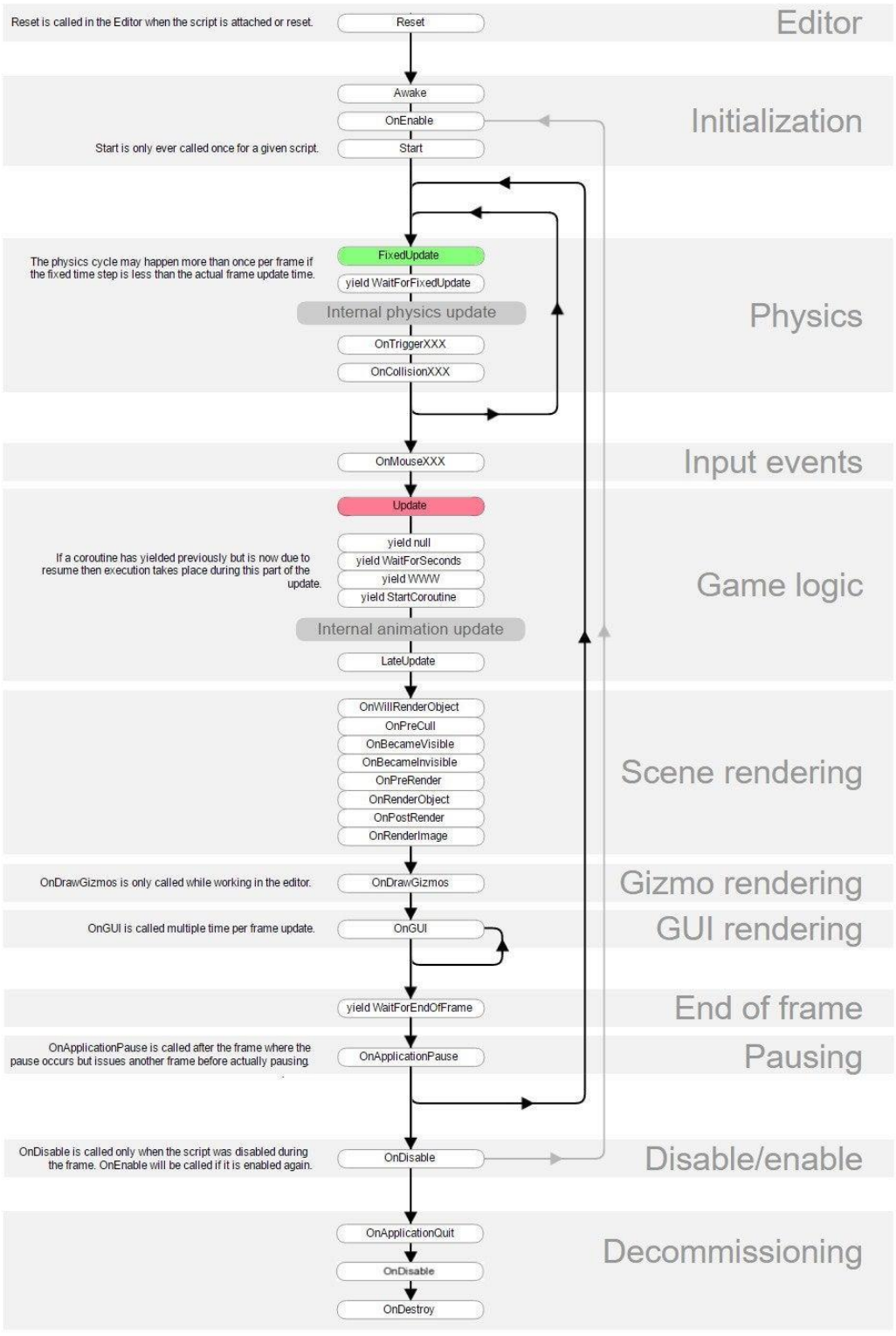
Scripts: chứa code của ngôn ngữ lập trình, được sử dụng để điều khiển mọi thành phần khác.

b) Vòng lặp game:

Phần cốt lõi của hầu hết các game chính là một vòng lặp chạy vô tận cho đến khi người dùng kết thúc trò chơi, được dùng để cập nhật và hiển thị trạng thái của game. Những thay đổi trong game sẽ được cập nhật rồi vẽ lại qua mỗi lần lặp.

Ở mỗi bước lặp, chương trình sẽ thực hiện rất nhiều tác vụ, tuy nhiên có ba hành động chính cần phải quan tâm:

- Đầu tiên chương trình nhận input từ các thiết bị bên ngoài như chuột, bàn phím, cảm ứng từ màn hình điện thoại,…
- Ở bước thứ hai, chương trình sẽ cập nhật các đối tượng trong scene dựa trên những input này và logic của game.
- Cuối cùng, chương trình xuất ra output đến các thiết bị bên ngoài như màn hình, loa,…



Hình 6: Workflow trong mỗi bước lặp được xử lí ở Unity

c) Tốc độ khung hình:

Qua mỗi bước lặp chương trình sẽ tạo ra một khung hình (frame). Một trò chơi với tốc độ frame là 60 FPS (frame per second) nghĩa là trong một giây trò chơi sẽ xuất ra 60 frame (lặp 60 lần một giây). FPS càng cao, hình

ảnh hiển thị sẽ mượt mà hơn. Giả sử tốc độ của game là 1 FPS, người chơi sẽ chỉ nhìn thấy một hình ảnh trên mỗi giây, điều này giống như là trình chiếu hơn là một đoạn video.

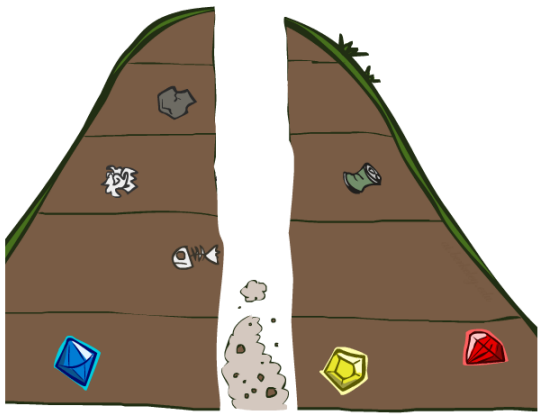
Để đảm bảo chương trình chạy ổn định thì thông thường trò chơi cần duy trì FPS ở mức tối thiểu 30 FPS.

2. Giải thuật Depth-first search (DFS):

Hầu hết mọi vấn đề AI đều có thể được quy về vấn đề tìm kiếm, đó có thể là tìm kế hoạch, đường đi, mô hình, hàm số… tốt nhất. Giải thuật tìm kiếm do đó thường được coi là trọng tâm của AI.

Ý tưởng của các giải thuật DFS là chúng ta sẽ xây dựng một cây tìm kiếm với node gốc là node đại diện cho trạng thái bắt đầu. Các cạnh đại diện cho các hành động mà agent thực hiện để chuyển trạng thái và các nút đại diện cho các trạng thái. Khi duyệt cho mỗi nhánh DFS sẽ khám phá xa hết mức có thể trước khi quay lại và thử một nhánh khác.

Ưu điểm của DFS là có thể dễ dàng áp dụng trong game với đặc tính có thể quan sát toàn bộ cây, số nhánh nhỏ và chuyển tiếp nhanh. DFS tìm kiếm trong không gian trạng thái mà ta không có bất kì thông tin nào về mục tiêu đến.



Hình 7: Minh họa trực quan giải thuật DFS

3. Giải thuật A*:

Trong khi giải thuật DFS không có bất kì thông tin nào về trạng thái đích đến thì giải thuật A* cần có một hàm heuristics để ước tính một trạng thái sẽ ở gần mục tiêu như thế nào. Đối với mỗi vấn đề khác nhau sẽ có mỗi hàm heuristics khác nhau để có thể đánh giá chính xác nhất.

A* Search sẽ khám phá đến node tiếp theo nào có f(n) = g(n) + h(n) là nhỏ nhất.

Hình 8: Minh họa trực quan giải thuật A*

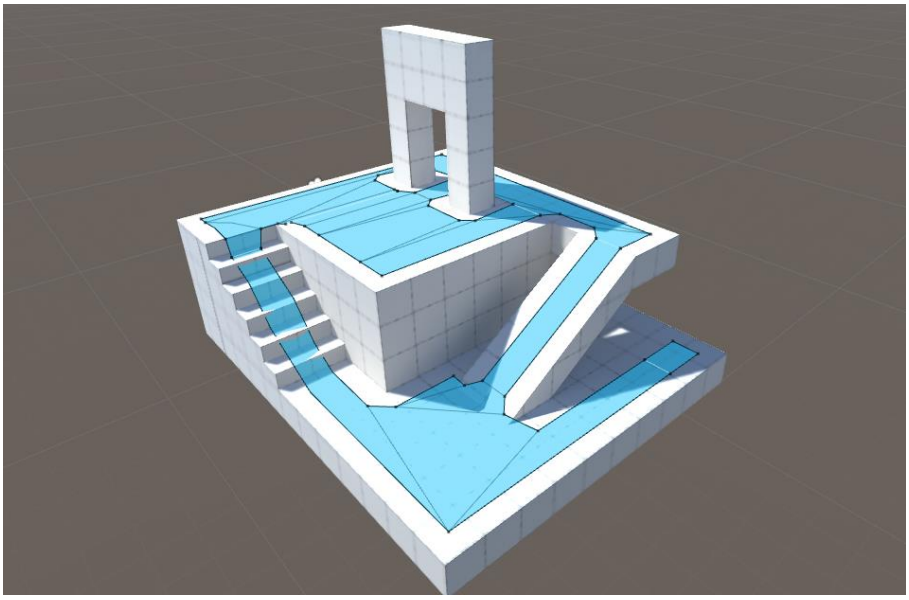
4. Tìm đường đi cho một nhóm đối tượng trong game:

a) Khu vực có thể đi được:

Hệ thống tìm đường cần phải có dữ liệu riêng đại diện cho các khu vực có thể đi được trong scene game. Các khu vực này là nơi mà agent có thể đứng hoặc di chuyển trên nó.

Trong Unity, khu vực này sẽ được xây dựng tự động từ hình dạng địa lý trong game. Cơ chế của nó đơn giản chỉ là thử nghiệm các vị trí mà agent có thể đứng. Và như vậy, các vị trí đó sẽ được kết nối với lớp lưới nằm trên bề mặt của địa hình. Lớp lưới này gọi là NavMesh (navigation mesh).

Chúng ta biết rằng, không có vật cản nào giữa hai điểm bất kỳ bên trong của một đa giác, vì vậy cách biểu diễn này rất hữu ích cho việc tìm đường đi.



Hình 9: NavMesh (vùng màu xanh)

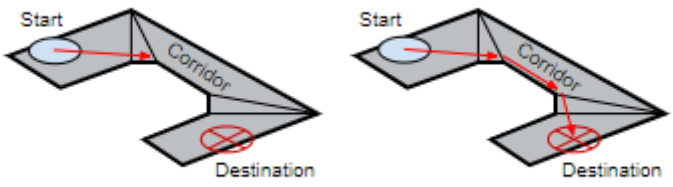
Các đa giác liền kề nhau sẽ được kết nối với nhau, điều này sẽ cho phép chúng ta tìm kiếm trên toàn bộ khu vực có thể đi được.

b) Tìm kiếm các đường đi:

Để tìm đường đi giữa hai vị trí, đầu tiên chúng ta cần tìm đa giác gần nhất ứng với mỗi điểm nguồn và điểm đích đến. Sau đó chúng ta bắt đầu tìm kiếm xuất phát từ vị trí nguồn, duyệt qua tất cả những đa giác liền kề cho tới khi chạm đến đa giác đích. Để thực hiện tác vụ này, giải thuật A* sẽ được sử dụng.

c) Theo một con đường:

Chuỗi đa giác mô tả đường đi từ đa giác khởi đầu đến đa giác đích được gọi là corridor. Agent sẽ đi đến đích bằng cách luôn hướng về phía góc có thể nhìn thấy tiếp theo của corridor.



Hình 10: Đường đi trên corridor

Khi nhiều agent cùng di chuyển tại một thời điểm, chúng sẽ cần phải đi chệch ra khỏi con đường ban đầu để tránh lẫn nhau.

d) Tránh chướng ngại vật:

Trong Unity, tính năng tránh chướng ngại vật chọn một vận tốc mới cân bằng giữa việc di chuyển theo hướng mong muốn và ngăn ngừa va chạm trong tương lai. Tính năng này sử dụng reciprocal velocity obstacles (RVO) để dự đoán và ngăn chặn va chạm.

e) Di chuyển agent:

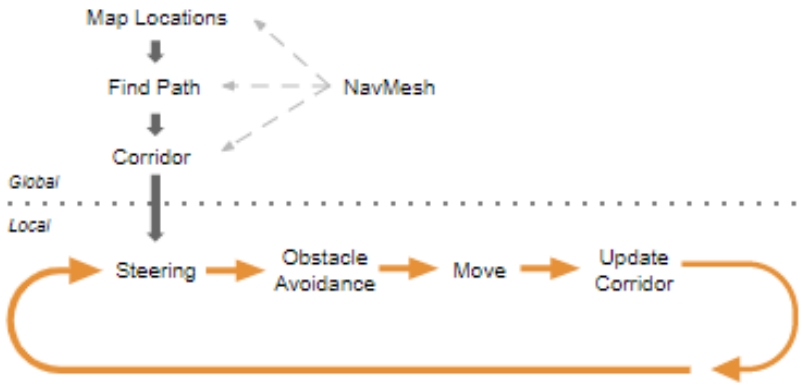
Sau khi hướng di chuyển và vận tốc cuối cùng đã được tính. Các agents trong Unity được mô phỏng sử dụng một mô hình động, bao gồm việc tính toán từ gia tốc để cho sự di chuyển trông tự nhiên và trơn tru hơn.

f) Tìm đường toàn cục và cục bộ:

Tìm đường toàn cục là việc tìm ra corridor trên toàn bản đồ, quá trình tìm đường này là một hoạt động tốn kém và đòi hỏi sức mạnh xử lí và bộ nhớ.

Danh sách các đa giác mô tả đường đi là một cấu trúc dữ liệu linh hoạt để điều hướng, và nó có thể được điều chỉnh cục bộ khi vị trí của agent thay đổi.

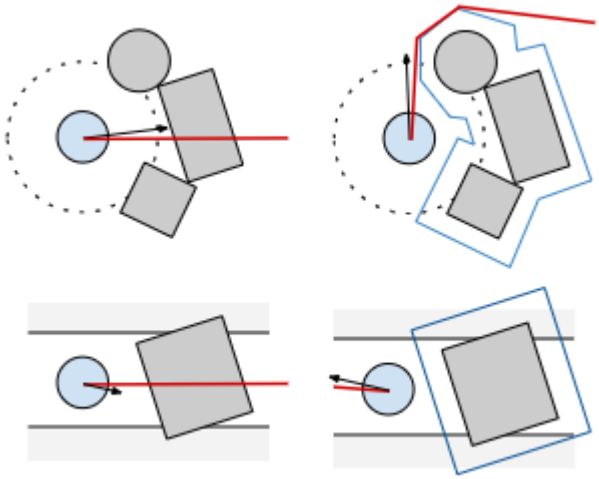
Tìm đường cục bộ là việc cố gắng tìm ra cách di chuyển hiệu quả về phía góc tiếp theo mà không va chạm với các agent hoặc đối tượng chướng ngại vật khác.



Hình 11: Tìm đường toàn cục và cục bộ

g) Hai trường hợp khi tránh chướng ngại vật:

Các chướng ngại vật có thể xử lí bằng tránh chướng ngại vật cục bộ, hoặc là tìm đường toàn cục.



Hình 12: Hai kịch bản cho việc tránh chướng ngại vật

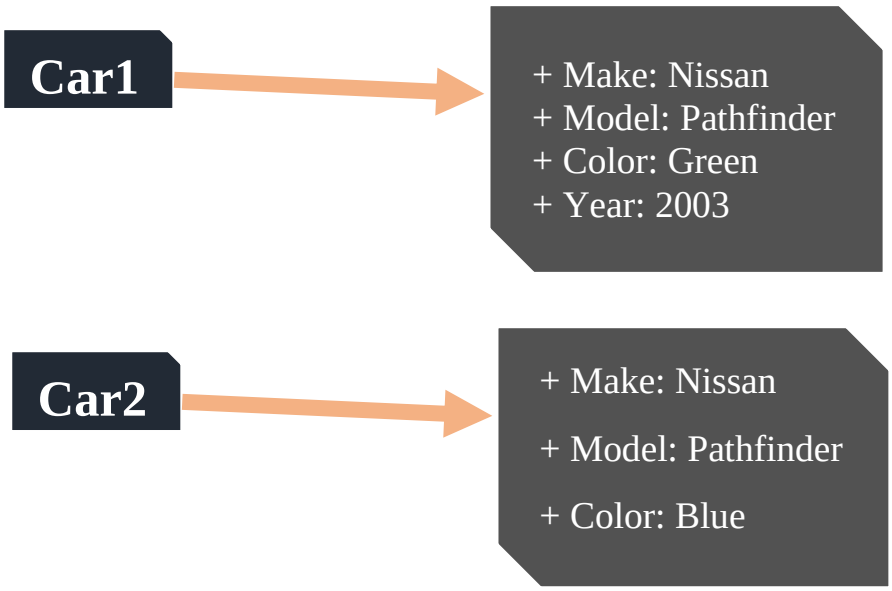
Đối với các chướng ngại vật di chuyển, cách tốt nhất là sử dụng tránh cục bộ. Bằng cách này agent có thể dự đoán trước các chướng ngại vật. Tuy nhiên, vì đặc tính cục bộ, nó sẽ chỉ xem xét các va chạm ngay sau đó và không thể điều hướng trong trường hợp chướng ngại vật chặn đường đi. Những trường hợp này có thể giải quyết bằng cách sử dụng carving.

Khi các chướng ngại vật đứng yên và chặn đường đi của agent. Chúng sẽ ảnh hưởng đến navigation mesh. Lúc này, ta cần xác định những chướng ngại vật nào chạm vào navigation mesh và làm thay đổi navigation mesh. Sự thay đổi navigation mesh được gọi là carving.

5. NoSQL database key-value:

NoSQL database có cách lưu trữ và truy vấn hoàn toàn khác với Relational Database lưu trữ thông tin theo dạng hàng-cột.

Các dữ liệu của NoSQL database loại key-value được lưu như một bảng hash, bao gồm một key liên kết với một value. Key là chuỗi duy nhất, được dùng để định danh và value chứa văn bản đơn giản hoặc các tập, danh sách dữ liệu phức tạp hơn. Quá trình tìm kiếm dữ liệu sẽ được thông qua key.



Hình 13: Ví dụ về cơ sở dữ liệu NoSQL dạng key-value

NoSQL database bỏ qua tính toàn vẹn của dữ liệu và transaction để đổi lấy hiệu suất nhanh và khả năng mở rộng. Ngoài ra NoSQL database key-value sử dụng ít bộ nhớ hơn Relational Database.

Về cơ bản, hầu hết các NoSQL database sẽ có các API sau:

- void Put(string key, byte[] data): đưa dữ liệu vào một key
- byte[] Get(string key): lấy dữ liệu từ một key
- void Remove(string key): xóa key và dữ liệu tương ứng

Dễ thấy rằng, ý tưởng của các NoSQL database là đơn giản hóa việc lưu trữ dữ liệu, nghĩa là không cần quan tâm đến nội dung cần lưu trữ là gì. Nói cách khác, chúng lưu trữ thông tin mà không cần phải xác định lược đồ. Việc để biết được dữ liệu thực tế như thế nào sẽ được định nghĩa ở phía client.

Sử dụng NoSQL database rất phù hợp với những ứng dụng trò chơi theo thời gian thực vì tốc độ truy xuất dữ liệu của nó rất nhanh.

6. Lập trình bất đồng bộ trong C#:

Lập trình bất đồng bộ lả một cách thức mà khi gọi nó chạy ở chế độ nền (task), trong khi tiến trình gọi nó không bị block. Từ .NET Framework 4.5, thư viện Task Parallel Library (TPL) được thêm vào giúp chương trình chạy đa luồng dễ dàng hơn.

Thông thường, khi lập trình để gọi một phương thức nào đó thì phương thức đó chạy và kết thúc thì các dòng code tiếp theo sau lời gọi đó mới được thực thi, đó là chạy đồng bộ, có nghĩa là thread gọi phương thức bị khóa lại cho đến khi phương thức kết thúc. Vấn đề này sẽ làm cho các dòng code tiếp theo phải chờ, nếu hàm đó thi hành mất nhiều thời gian trong khi tài nguyên vẫn đủ để làm các việc khác thì cả chương trình sẽ phải chờ phương thức trên kết thúc mới thi hành được tác vụ khác.

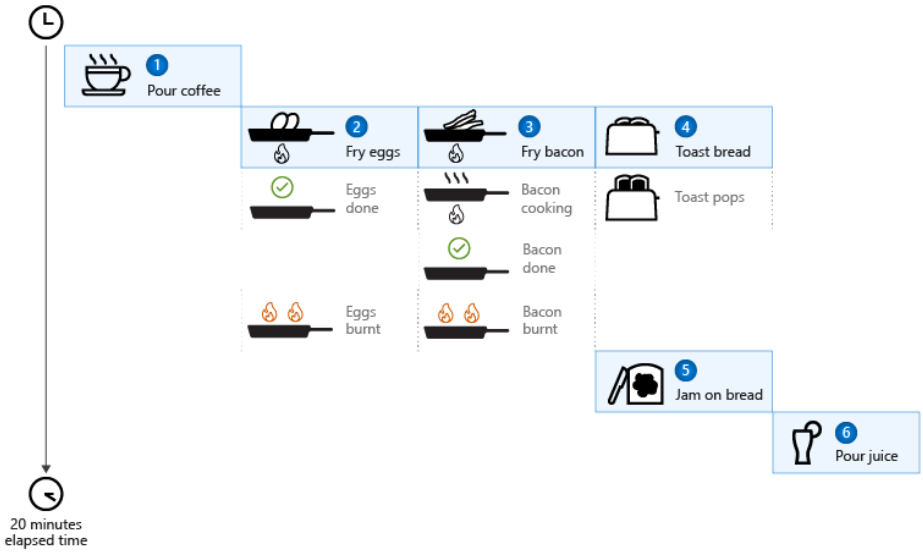
Để giải quyết vấn đề này, kỹ thuật bất đồng bộ giúp cho chương trình vẫn thực thi trong khi chờ cho phương thức được thực hiện xong.

Lớp Task biểu thị tác vụ bất đồng bộ, nếu tác vụ sau khi thực thi trả về kết quả thì Task<T> được sử dụng, với T là kiểu trả về. Khi muốn bắt đầu nhiều tác vụ độc lập cùng lúc, ta sẽ tạo một đối tượng Task nắm giữ công việc được giao. Sau đó có thể chờ đề làm việc với kết quả của nó mà không block chương trình.

Đoạn chương trình dưới đây minh họa cho việc nấu bữa sáng:

```
Coffee cup = PourCoffee();
Console.WriteLine("coffee is ready");

Task<Egg> eggsTask = FryEggsAsync(2);
Task<Bacon> baconTask = FryBaconAsync(3);
Task<Toast> toastTask = ToastBreadAsync(2);

Toast toast = await toastTask;
ApplyButter(toast);
ApplyJam(toast);
Console.WriteLine("toast is ready");
Juice oj = PourOJ();
Console.WriteLine("oj is ready");

Egg eggs = await eggsTask;
Console.WriteLine("eggs are ready");
Bacon bacon = await baconTask;
Console.WriteLine("bacon is ready");


Console.WriteLine("Breakfast is ready!");
```

Có thể thấy, ta có thể bắt đầu các tác vụ bất đồng bộ cùng một lúc. Sau đó chỉ việc đợi kết quả khi nào cần đến nó.



Hình 14: Minh họa trực quan ví dụ về lập trình bất đồng bộ cho việc nấu bữa sáng

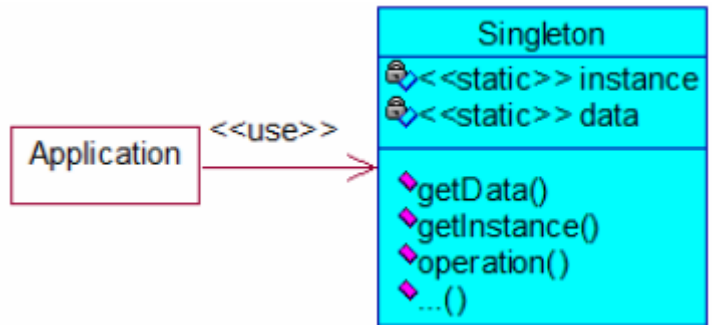
7. Các mẫu design pattern được áp dụng:

a) Mẫu Singleton:

Mẫu thiết kế Singleton là pattern đảm bảo rằng một lớp chỉ có thể có một instance duy nhất và nó sẽ cung cấp cổng giao tiếp để truy cập vào lớp đó ở bất kì thời điểm nào hoặc vị trí nào trong phần mềm.

Một số ví dụ có thể áp dụng mẫu Singleton:

- Những tài nguyên được chia sẻ như database để truy xuất, file vật lý được sử dụng chung… Những tài nguyên này chỉ nên có một instance tồn tại để sử dụng và theo dõi trạng thái của nó. Ví dụ để quản lí in ấn của các phần mềm ra một máy in được đúng đắn và nhất quán, ta chỉ được tạo một đối tượng “printer spooler” để quản lý máy in tương ứng.
- Những đối tượng không cần thiết phải tạo instance mới mỗi lần sử dụng như đối tượng logging…



Hình 15: Lược đồ class mẫu thiết kế Singleton

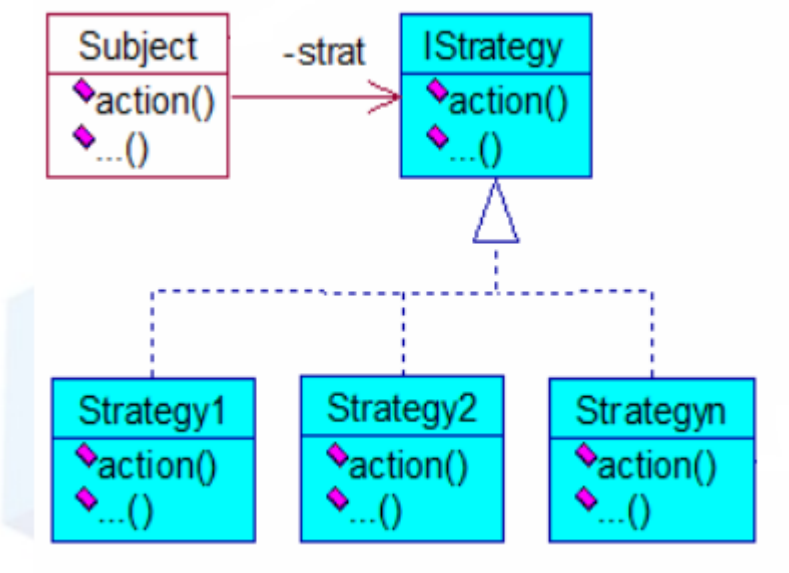
b) Mẫu Strategy:

Mẫu Strategy cung cấp một họ các giải thuật khác nhau để giải quyết cùng một vấn đề nào đó và cho phép Client chọn lựa linh động dễ dàng một giải thuật cụ thể tùy theo từng tình huống sử dụng.

Về nguyên lí chung, thường có nhiều giải thuật khác nhau cùng giải quyết được một bài toán. Mỗi giải thuật có những ưu khuyết điểm riêng và sẽ thích hợp trong ngữ cảnh sử dụng nào đó so với các giải thuật còn lại. Cách tốt nhất để giúp Client lựa chọn linh động và dễ dàng một giải thuật phù hợp tùy theo từng tình huống đó là dùng mẫu Strategy.

Một số ví dụ có thể áp dụng mẫu Strategy:

- Ví dụ để sắp xếp thứ tự các phần tử trong mảng, ta có nhiều giải thuật sắp xếp khác nhau như bubble sort, quick sort, merge sort… Cách tốt nhất để linh động giải thuật sắp xếp cho danh sách và giúp code của các tác vụ trong đối tượng danh sách độc lập với giải thuật sắp xếp đó là dùng mẫu Strategy.



Hình 16: Lược đồ class mẫu Strategy

8. Một số kiến thức toán học:

a) Hệ trục tọa độ Oxyz và vị trí của vật thể trong không gian:

Hệ trục tọa độ Oxyz gồm 3 trục Ox, Oy, Oz đôi một vuông góc nhau. Vị trí của vật thể sẽ được xác định dưới dạng tọa độ của vector đi từ gốc đến điểm đó.

b) Quaternion:

Quaternion được sử dụng trong phép quay không gian, được định nghĩa như một số phức có ba thành phần ảo:

$$q = w + xi + yj + zk$$

Các thành phần ảo i, j, k được coi như ba vector đơn vị của trục tọa độ Oxyz.

c) Finite State machine:

FSM (Finite State Machine) là một mô hình tính toán toán học để biểu diễn trạng thái của một hệ thống, trong đó số trạng thái là hữu hạn. Từ mỗi trạng thái, FSM có thể chuyển đổi qua một số trạng thái cố định khác, dựa trên event, input.

FSM có các đặc điểm:

- Tại mỗi thời điểm chỉ ở một trạng thái duy nhất.

- Tại mỗi trạng thái, chỉ có thể chuyển qua những trạng thái được cho phép.
- Từ trạng thái hiện tại, có thể biết được những trạng thái kế tiếp mà máy có thể chuyển qua.

FSM có thể được ứng dụng để quản lí trạng thái của một đối tượng hoặc workflow.

9. Một số kiến thức vật lí:

a) Vận tốc tức thời của vật:

Giá trị vận tốc tức thời được tính theo công thức:

$$v = \lim_{t \to 0} \frac{\Delta s}{\Delta t}$$

Δs là quãng đường vô cùng bé và Δt là thời gian vô cùng bé.

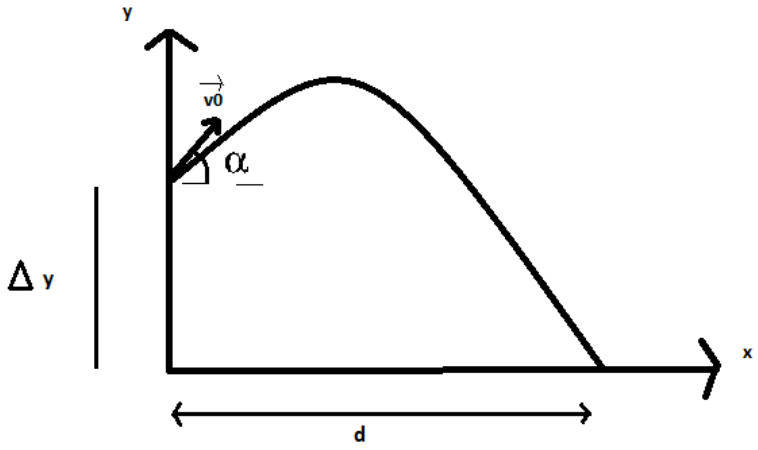
b) Phương trình tọa độ của vật thể chuyển động biến đổi đều:

Tọa độ của vật được xác định theo công thức:

$$x(t) = x_0 + v_0 t + \frac{at^2}{2}$$

Trong đó x_0 là tọa độ ban đầu, $v_0$là vận tốc ban đầu và a là gia tốc của vật, t là thời gian.

c) Tính vận tốc ban đầu của vật được ném lên theo phương xiên:



Hình 17: Bài toán vật ném lên theo phương xiên

Một vật được ném lên theo phương xiên với các dữ liệu đầu vào: α là góc nghiêng theo phương trục x, d là khoảng cách đến vị trí rơi, Δy là độ cao chênh lệch theo trục y từ vị trí ném xuống mặt đất. Ta sẽ tính được vận tốc ban đầu theo công thức:

$$v_0 = \frac{\sqrt{\frac{\frac{d^2 g}{2}}{d \tan \alpha + \Delta y}}}{\cos \alpha}$$

Và thời gian vật chuyển động:

$$t = \frac{d}{v_0 \cos \alpha}$$

Chương IV. Thiết kế trò chơi

Chương này trình bày những ý tưởng mô tả về game, bao gồm cốt truyện, cách chơi, các đối tượng trong game, các thuộc tính,… Mã hiện thực chương trình sẽ dựa trên những đặc tả này.

1. Cốt truyện:

Trong trò chơi, người chơi sẽ đóng vai trò là tộc trưởng của bộ tộc, tìm cách xây dựng một đế chế mới và chinh phục các miền đất khác..

2. Người chơi:

Trò chơi có thể được chơi kết nối giữa hai người thông qua mạng Internet hoặc chơi một mình với máy tính.

3. Các hoạt động chính trong game:

Bao gồm: thiết kế, bố trí các công trình; khai thác tài nguyên và phát triển kinh tế; xây dựng đội quân và chiến tranh với các bộ tộc khác.

a) Xây dựng các công trình:

Người chơi chọn loại công trình thích hợp và kéo thả đến vị trí theo mong muốn.

Một số loại công trình có thể được nâng cấp để mang lai hiệu quả cao hơn.

b) Phát triển kinh tế:

Các loại tài nguyên có thể thu thập sau mỗi khung thời gian từ một số loại công trình bên trong ngôi làng, hoặc khi chiến thắng các bộ tộc khác.

c) Tấn công các bộ tộc khác:

Người chơi chọn một bộ tộc khác làm mục tiêu tấn công (người chơi khác hoặc máy tính).

Từ các đơn vị binh lính và tướng, người chơi có thể bố trí vị trí cho mỗi đơn vị trước và trong khi cuộc chiến diễn ra, nhưng chỉ có thể đặt ở phạm vi bên ngoài ngôi làng, sau đó mỗi quân lính sẽ tự thực hiện hành vi của mình.

Các phương thức của quân lính gồm có: di chuyển, tấn công mục tiêu hoặc ở trạng thái đứng yên.

Nếu phá hủy được đối phương ≥ 50% các công trình, hoặc phá được nhà chính sẽ xem như thắng cuộc.

Xếp hạng mức độ chiến thắng từ 1-3 sao. Người chơi sẽ chiến thắng tuyệt đối khi phá được toàn bộ ngôi làng đối thủ.

d) Phòng thủ chống lại các cuộc tấn công:

Toàn bộ quân lính, tháp bên phòng thủ có nhiệm vụ chống lại các cuộc tấn công.

- Trại binh lính: các quân lính phòng thủ sẽ xuất phát từ công trình này để tiến ra chống lại quân địch.
- Tháp phòng thủ: lựa chọn tấn công đơn vị trong phạm vi.

4. Các đối tượng trong game:

a) Các loại công trình:

Nhà, nơi ở: giúp tăng số lượng cư dân.

Cửa hàng, các trạm khai thác tài nguyên: người chơi có thể thu thập được các tài nguyên từ loại công trình này theo mỗi khung thời gian.

Các tiện ích, công trình xã hội: giúp tăng mức độ hài lòng của cư dân.

Đường đi: cư dân sẽ di chuyển trên đó, đường đi tạo liên kết giữa các công trình.

Nhà chính/ lâu đài: trung tâm của ngôi làng, có thể dùng để nâng cấp các thuộc tính của ngôi làng.

Tháp phòng thủ: chống lại sự tấn công từ các bộ lạc khác. Có nhiều loại tháp phòng thủ với các đặc tính khác nhau.

Trại binh lính: nơi tập hợp và đào tạo một nhóm quân lính.

Tường thành, rào chắn: dùng để cản đường đi của quân lính địch.

Các công trình đặc biệt khác.

b) Các loại nhân vật:

Dân thường: tăng theo số đơn vị nhà ở, cần đạt được một số lượng nhất định để mở khóa xây dựng thêm loại công trình mới.

Quân lính: được tập hợp tại các trại lính, dùng để tấn công hoặc phòng ngự chống lại các bộ tộc khác.

c) Địa hình, vị trí, thời tiết:

Ban đầu người chơi sẽ tự chọn một vùng đất để xây dựng lên ngôi làng của mình.

Mỗi loại vùng đất sẽ có đặc điểm và tác động khác nhau lên các hoạt động trong game.

5. Thuộc tính của các đối tượng:

a) Lượng tài nguyên của người chơi:

Vàng, gỗ: sử dụng để xây dựng thêm các công trình mới, nâng cấp công trình, mua quân lính…

b) Chỉ số, thuộc tính của quân lính:

Điểm sức khỏe (HP): điểm sức khỏe thể hiện một phần khả năng chống chịu của quân lính, nếu điểm sức khỏe ≤ 0 thì quân lính sẽ chết.

Sát thương (damage): khi một quân lính gây sát thương lên đối tượng khác, HP của đối phương sẽ mất đi.

Giáp: Được dùng để giảm sát thương gây ra.

Xuyên giáp: xuyên qua lượng giáp của mục tiêu khi tấn công nó.

Tốc độ tấn công: thời gian giữa mỗi lần quân lính tấn công.

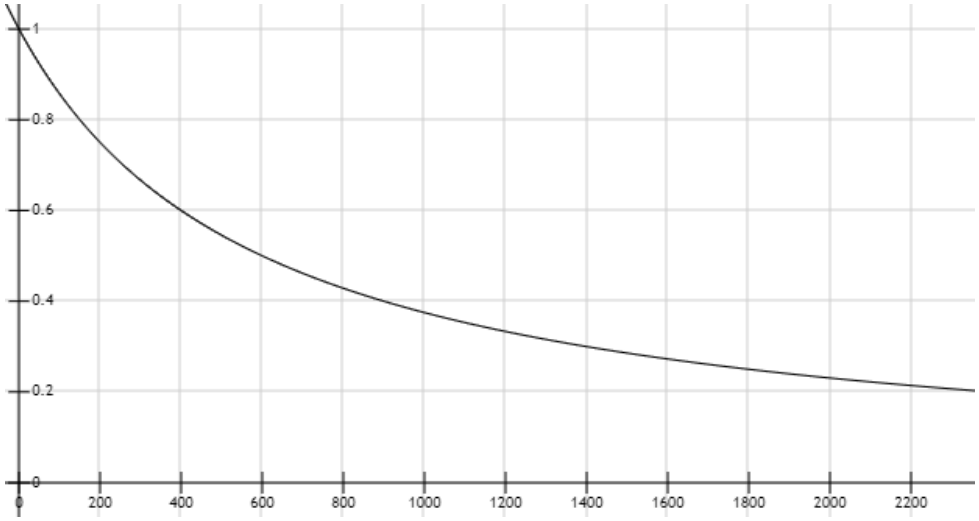
Phạm vi tấn công: khoảng cách tối thiểu để quân lính có thể tấn công.

Độ thông minh: quân lính có chỉ số này càng cao sẽ càng có hành vi phù hợp hơn đối với mỗi tình huống.

6. Cơ chế tấn công - nhận sát thương:

Những đối tượng bị tấn công sẽ bị trừ một lượng HP:

$$HP_{loss} = damage\ x \frac{600}{600 + armor}$$



Hình 18: đồ thị hàm số y = 600/(600+x)

Giá trị armor sẽ có tác dụng giảm đi sát thương theo hệ số thuộc đoạn (0;1). Khi armor càng thấp thì nó càng có giá trị, nếu armor bằng 0 thì toàn bộ sát thương sẽ chuyển thành lượng HP bị mất, còn armor càng lớn thì độ hữu ích của thuộc tính này lại càng giảm.

Trong giao tranh đối tượng sẽ chịu sát thương và mất HP theo thời gian, đến khi HP còn lại ≤ 0, đối tượng bị phá hủy.

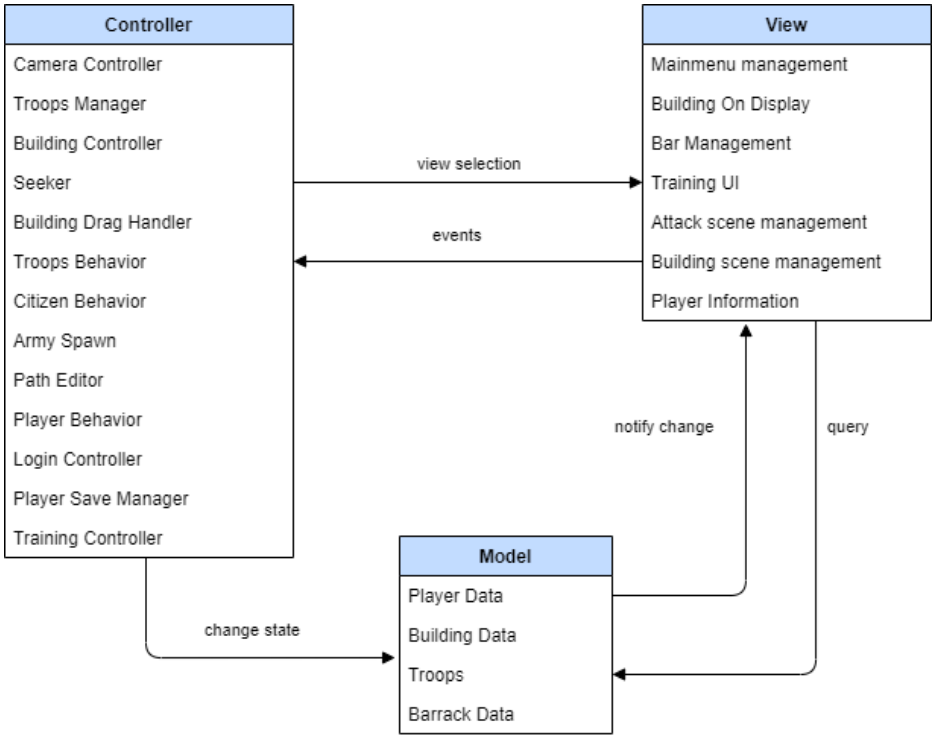
7. **Độ khó trong game:**

Người chơi cần vượt qua các cửa ải có độ khó tăng dần:

- Ở chế độ tấn công: độ khó được định nghĩa bởi số lượng quân địch, số lượng tháp phòng thủ và cách bố trí các công trình.
- Ở chế độ phòng thủ: độ khó của game tăng dần cùng với số lượng quân địch tấn công.

Chương V. Kiến trúc phần mềm

1. Mô hình kiến trúc MVC:



Hình 19: Mô hình kiến trúc MVC

Chương VI. Giới thiệu Unity engine và platform Firebase

Project sẽ được xây dựng trên công cụ Unity engine version: 2020.1, sử dụng ngôn ngữ lập trình VC# cùng với nền tảng Firebase.

1. Unity engine:

Unity là một công cụ phát triển game đa nền tảng, có thể sử dụng để phát triển game trên PC, consoles, thiết bị di động và trên websites. Đây là game engine phổ biến nhất hiện nay.

Một số ưu điểm của Unity: chức năng đa dạng, hỗ trợ đa nền tảng, dễ sử dụng, có tính kinh tế cao.

Các bước cài đặt và thiết lập môi trường Unity có thể tham khảo thêm tại:

https://docs.unity3d.com/2020.1/Documentation/Manual/GettingStartedInstallingUnity.html

2. Firebase:

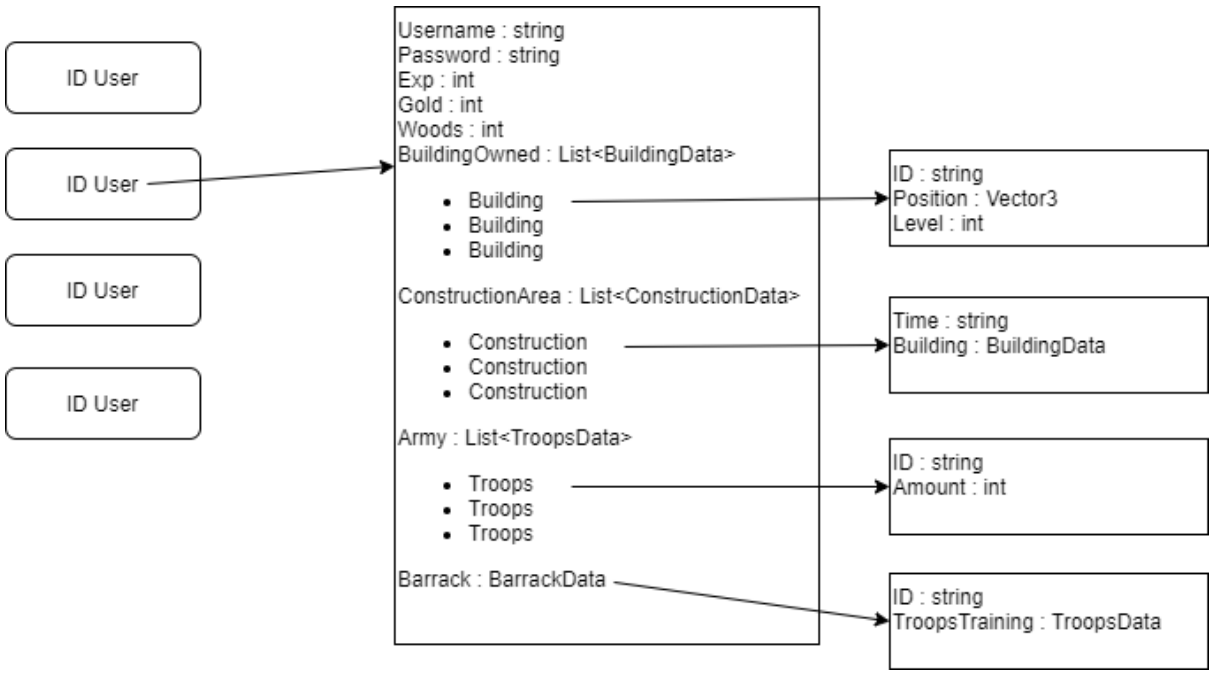
Firebase là dịch vụ cơ sở dữ liệu hoạt động trên nền tảng cloud. Kèm theo đó là hệ thống máy chủ cực kỳ mạnh mẽ của Google. Chức năng chính là giúp người dùng lập trình ứng dụng bằng cách đơn giản hóa các thao tác với cơ sở dữ liệu. Kèm theo đó Firebase còn là dịch vụ đa năng và bảo mật cực tốt, hỗ trợ cả hai nền tảng Android và IOS.

Các dịch vụ nổi bật của Firebase: Realtime Database, Crashlytics, Cloud Firestore, Authentication, Cloud Functions, Cloud Storage, Hosting, Test Lab for Android, Performance Monitoring, Google Analytics, Cloud Messaging, Predictions, Firebase Dynamic Links, Remote Config, Invites, App Indexing, AdMob, AdWords.

Cách cài đặt và tích hợp Firebase vào môi trường Unity có thể tham khảo thêm tại:

https://firebase.google.com/docs/unity/setup

Realtime database trong Firebase là NoSQL database, dữ liệu được lưu theo cấu trúc key-value.



Hình 20: Cơ sở dữ liệu trong Firebase

Chương VII. Phát triển trò chơi

Chương này trình bày nội dung thiết kế và hiện thực cho các tính năng chính của chương trình. Những nhóm asset model chính cần có để viết kịch bản cho game bao gồm: bản đồ, các công trình, hàng rào, các loại quân lính, hiệu ứng hình ảnh, âm thanh,… Các mô hình này có thể được import từ cửa hàng asset:

https://assetstore.unity.com/

Hình 21: Khung cảnh demo trong game

1. Điều khiển camera:

a) Giới thiệu chung:

Camera là thành phần đầu tiên không thể thiếu trong trò chơi. “Rising Steps” có camera chiếu bao quát từ trên xuống theo góc nhìn thứ 3 [x] cho phép người chơi có thể quan sát mọi thứ xung quanh với góc nhìn cực rộng.

Nếu muốn theo dõi chi tiết một góc nào đó, người chơi cũng có thể điều khiển camera lại gần để có cái nhìn cận cảnh.

Camera có thể được người chơi điều khiển di chuyển đến các vị trí khác nhau trong phạm vi của mặt phẳng song song với địa hình.

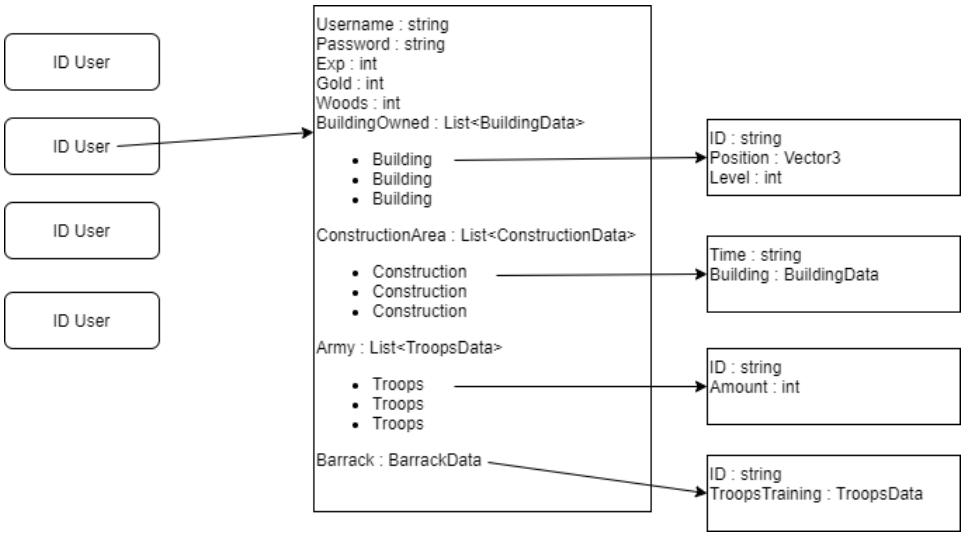
b) Hiện thực:

Việc di chuyển camera xung quanh đơn giản chỉ cần thay đổi tọa độ x, z. Để phóng to và thu nhỏ, ta sẽ dịch chuyển camera theo vector forward của nó.

2. Load dữ liệu Player:

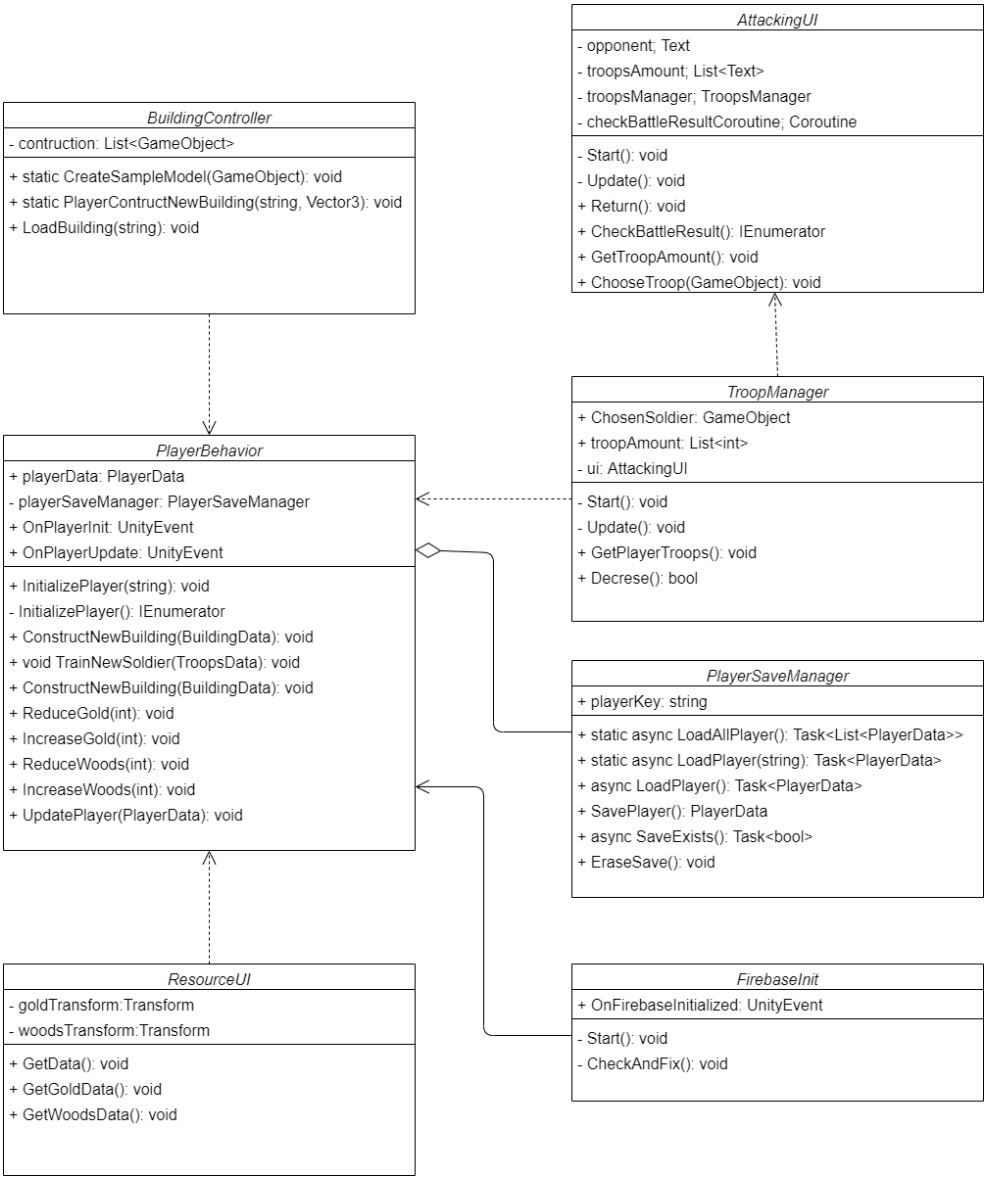
a) Giới thiệu chung:

Dữ liệu player được lưu trong struct PlayerData của lớp PlayerBehavior, bao gồm: tên đăng nhập, mật khẩu, điểm kinh nghiệm hiện tại, danh sách các công trình sở hữu, vàng, gỗ, danh sách quân lính, dữ liệu quân đang huấn luyện. Khi đăng nhập, hệ thống sẽ load dữ liệu player từ database.



Hình 22: Cấu trúc lưu trữ ở database

b) Thiết kế hướng đối tượng:



Hình 23: Class diagram hệ thống load dữ liệu

c) Hiện thực:

Khi người chơi chuyển đến scene, đối tượng Firebase được khởi tạo và kết nối đến server. Sau khi khởi tạo xong, sự kiện OnFirebaseInitialized được kích hoạt.

Đối tượng PlayerBehavior chứa PlayerData khi lắng nghe OnFirebaseInitialized sẽ lấy ID người chơi đang được lưu trên máy, sau đó gọi một hàm bất đồng bộ InitializePlayer() để load dữ liệu player có ID đó thông qua đối tượng PlayerSaveManager.

```
playerDataTask = playerSaveManager.LoadPlayer();
```

```
yield return new WaitUntil(predicate: () => playerDataTask.IsCompleted);
```

Đối tượng PlayerSaveManager gọi API truy vấn dữ liệu của player từ database và đợi kết quả trả về. Dữ liệu trả về tiếp theo sẽ được định dạng JSON và tạo ra PlayerData thông qua lớp JsonUtility.

```
return JsonUtility.FromJson<PlayerData>(dataSnapshot.GetRawJsonValue());
```

Cuối cùng PlayerData được cập nhật vào đối tượng PlayerBehavior để sử dụng và kích hoạt event OnPlayerInit.

```
var playerData = playerDataTask.Result;
UpdatePlayer(playerData);

UpdatePlayer(PlayerData newData) {
       if (newData not equals playerData) then
       begin
              playerData = newData;
          OnPlayerInit.Invoke();
   end
}
```

Đối tượng BuildingController lắng nghe event OnPlayerInit và khi có dữ liệu PlayerData sẽ bắt đầu load các công trình dựa trên danh sách BuildingOwned của người chơi.

BuildingOwned là một danh sách các BuildingData, mỗi BuildingData chứa ID công trình và transform mà người chơi đã xây được. Để load chính xác công trình, đối tượng BuildingController kiểm tra ID của mỗi BuildingData với các prefab hiện có, sau đó sinh ra prefab công trình với ID tương ứng nhận giá trị transform của nó.

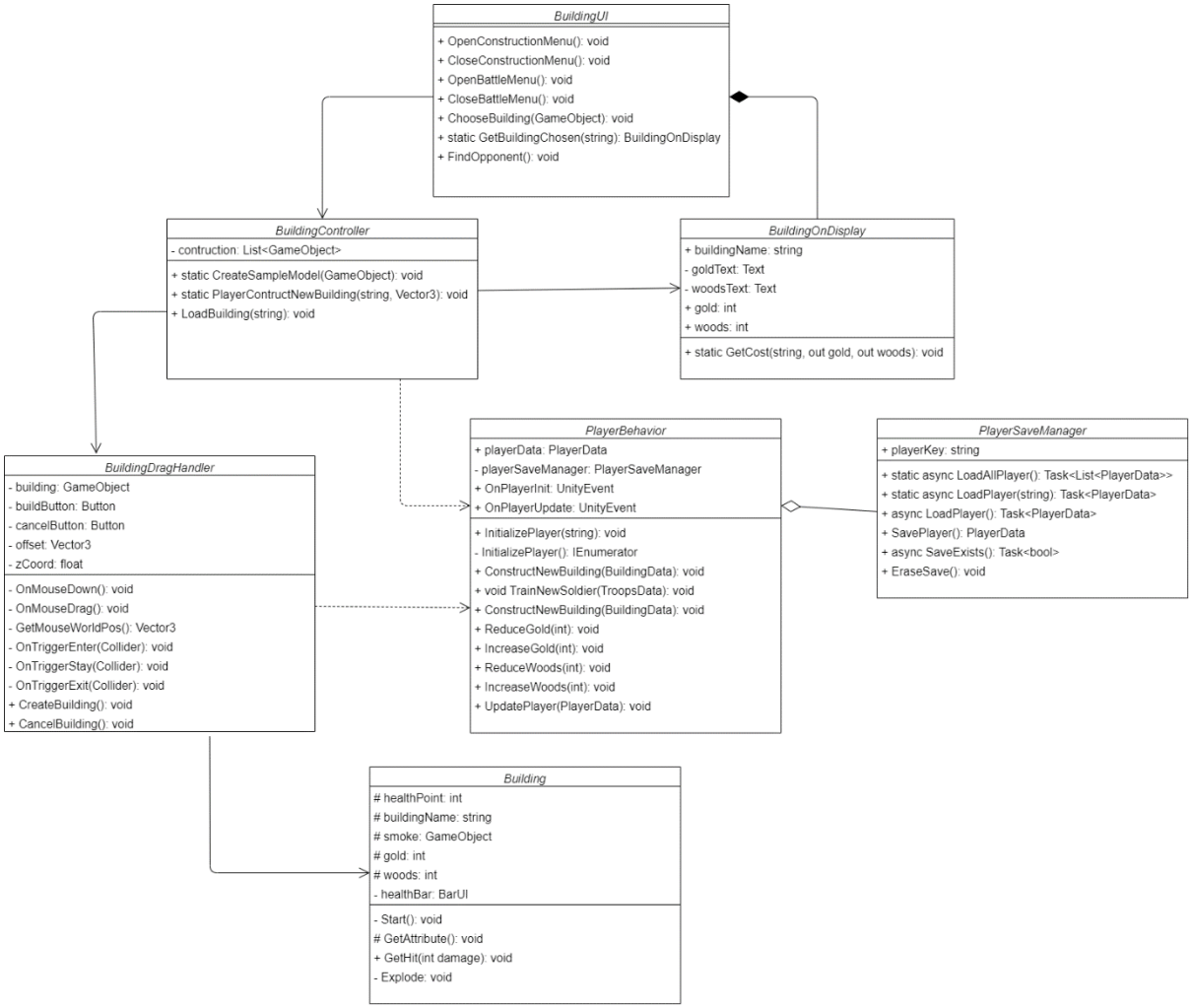
Cùng với BuildingController, đối tượng Resource cũng nhận dữ liệu PlayerData khi sự kiện OnPlayerInit được kích hoạt và đồng bộ các nội dung UI với dữ liệu tương ứng.

3. Tính năng xây công trình:

a) Giới thiệu chung:

Ngôi làng của người chơi bao gồm có nhiều loại công trình khác nhau. Trước tiên, người chơi cần chọn lựa và xây dựng nên chúng. Quá trình xây dựng bắt đầu từ việc người chơi click chọn cửa hàng để xem xét danh sách các công trình có thể xây. Sau đó chọn một công trình phù hợp và kéo thả đến vị trí mong muốn.

b) Thiết kế hướng đối tượng:



Hình 24: class diagram hệ thống xây công trình

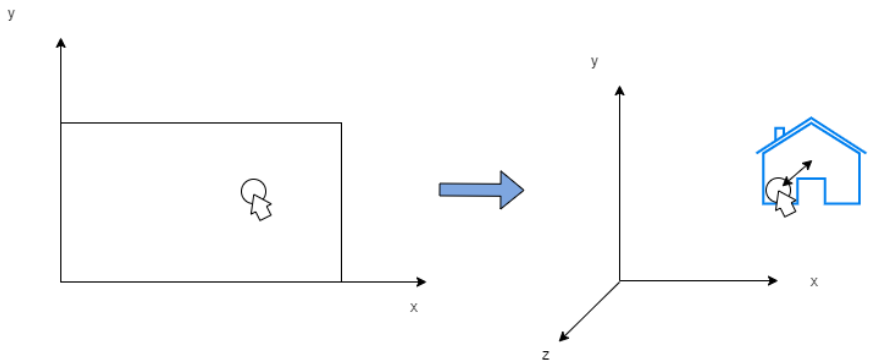
c) Hiện thực:

Trong UI cửa hàng, các đối tương Building On Display chứa component Button với sự kiện OnClick là phương thức ChooseBuilding(GameObject model) nhận đầu vào tương ứng với mỗi công trình đó.

Hình 25: Hiện thực tính năng xây dựng công trình

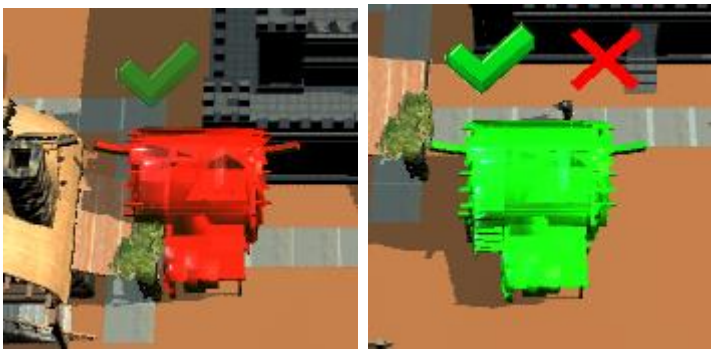
Khi được click chọn, sự kiện được thông báo cho lớp Building Controller thực thi phương thức CreateSampleModel. Lớp này kiểm tra tài nguyên của người chơi so với giá của công trình, nếu lượng tài nguyên lớn hơn, một đối tượng mẫu sampleBuilding sẽ được tạo ra trên bản đồ, ngược lại, dòng thông báo "không đủ chi phí" sẽ hiển thị.

Đối tượng mẫu sampleBuilding chứa lớp BuildingDragHandler được dùng để xử lí kéo thả và kiểm tra va chạm. Giải thuật kéo thả là khi được click vào, chương trình sẽ nhận input vị trí chuột tại screen space, sau đó chuyển tọa độ này thành world space cùng với việc tính toán vị trí độ lệch offset so với điểm tọa độ gốc của công trình. Mỗi khi kéo thả, công trình sẽ di chuyển theo vị trí chuột trong world space + offset.



Hình 26: Xử lí kéo thả

Đối tượng sampleBuilding cũng chứa thành phần Collider để kiểm tra va chạm với các công trình khác, nếu xảy ra va chạm, button xây sẽ bị vô hiệu hóa.

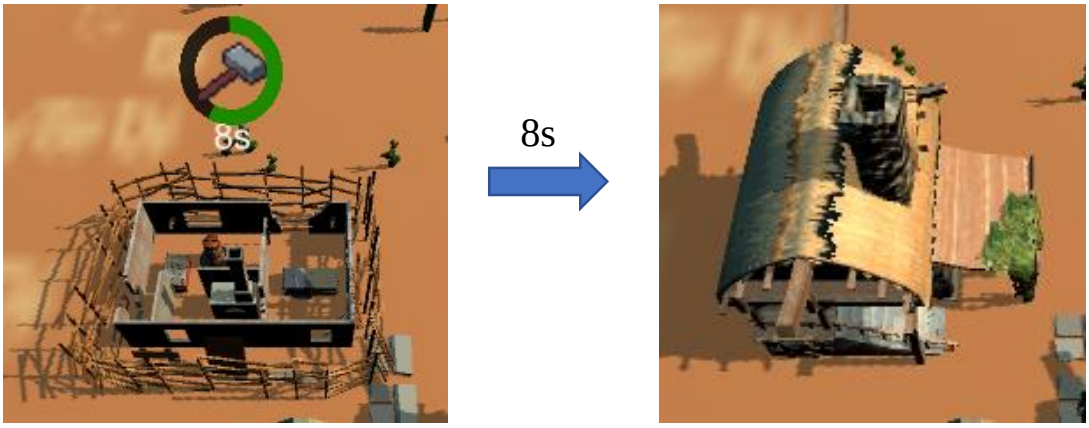


Hình 27: Kiểm tra va chạm mô hình mẫu

Nếu không xảy ra va chạm với công trình khác, người chơi có thể click chọn button Build để đồng ý xây hoặc button Cancel để hủy mô hình mẫu.

Khi công trình được xây thành công, tài nguyên của người chơi sẽ bị trừ đi theo chi phí của công trình và được cập nhật vào database.

Lúc này chương trình sẽ tạo một đối tượng ConstructionArea với một bộ đếm thời gian, đối tượng sẽ liên tục cập nhật thời gian trong game với thời gian thực, khi đã đủ thời gian yêu cầu, công t rình mới được hoàn thiện.



Hình 29: Công trình đang thi công

Hình 28: Công trình hoàn thiện

4. Xây dựng lớp tháp phòng thủ:

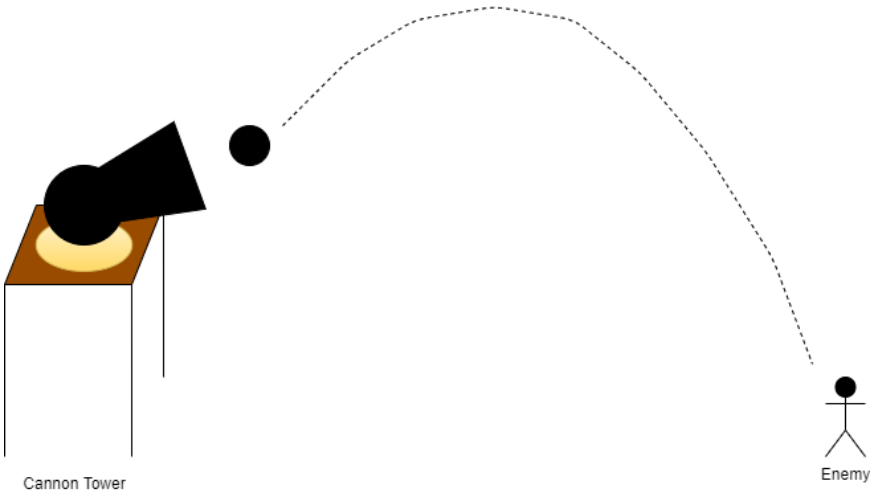
a) Giới thiệu chung:

Tháp phòng thủ là lớp đối tượng được kế thừa từ lớp Building. Đặc điểm của loại tháp này là nó có thể tấn công để tiêu diệt các quân lính đối phương. Trò chơi bao gồm nhiều loại tháp phòng thủ khác nhau với các chỉ số và công dụng rất đa dạng. Ba loại tháp cơ bản được giới thiệu là: cannon tower, mage tower, và the guardian.

- Cannon tower:
  - Cannon tower có thể xoay 360º để tìm diệt mục tiêu xung quanh. Điểm đặc biệt của công trình này là sự chống chịu tốt, phạm vi tấn công xa cùng với khả năng bắn lan, tức là đạn khi phát nổ sẽ trúng nhiều mục tiêu. Tuy nhiên nhược điểm của tháp là tốc độ tấn công khá thấp, và đạn cũng có xác suất không trúng địch.

Hình 30: Mô hình tháp Cannon

  - Đạn cannon được bắn ra sẽ bay rất chậm, vì vậy ta cần xác định được vị trí điểm rơi và vận tốc ban đầu để có thể bắn trúng mục tiêu



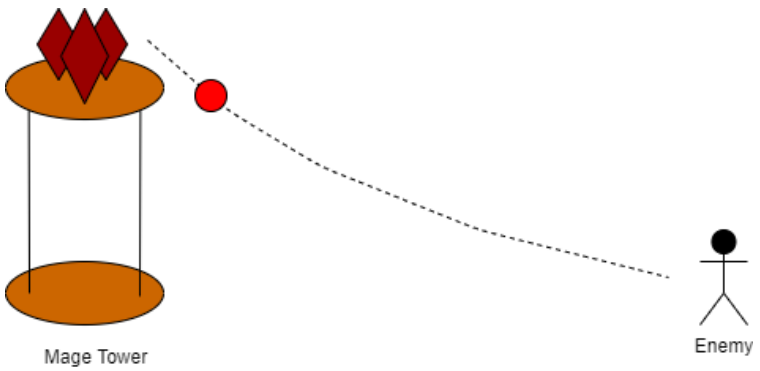
Hình 31: Minh họa hành vi tấn công của Cannon Tower

- Tháp cannon thích hợp để phòng thủ đối với một đám đông quân lính di chuyển cạnh nhau.

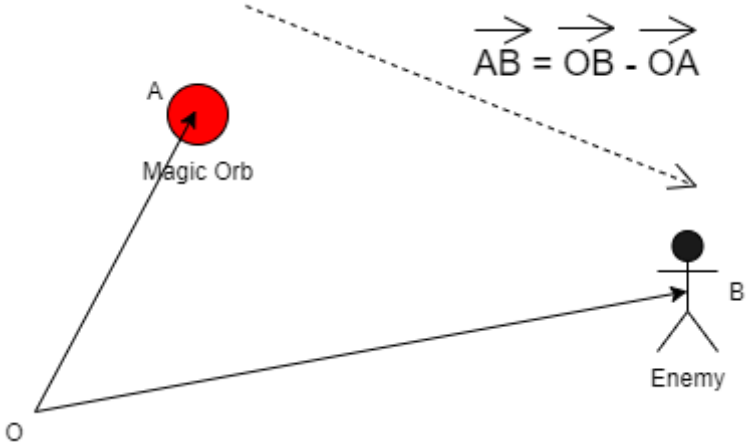
- Mage Tower:
  - Mage tower là loại tháp tấn công bằng cách bắn ra một quả cầu phép thuật tự hướng đến vị trí mục tiêu. Mage tower có sát thương cao, sức chống chịu và tầm tấn công ở mức trung bình.

Hình 32: Mô hình Mage Tower

  - Mage tower thích hợp để phòng thủ đối với những loại quân lính di chuyển đơn lẻ có sức chống chịu cao.
  - Mage tower tấn công bằng cách bắn ra một quả cầu phép thuật tự hướng đến vị trí mục tiêu.



Hình 33: Minh họa hành vi tháp Mage

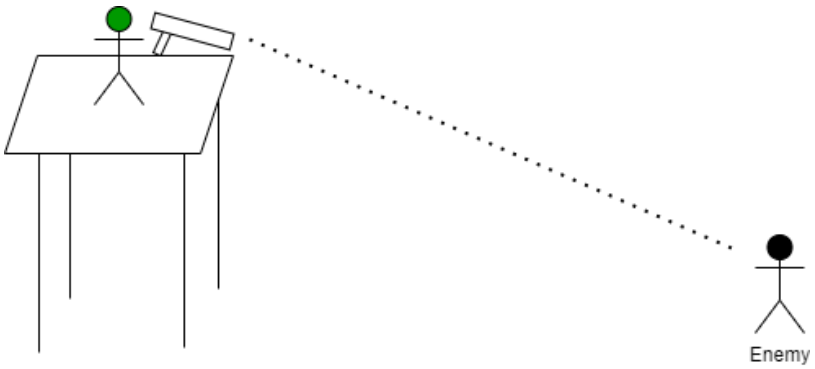


Hình 34: Magic Orb tự hướng đến mục tiêu

- The Guardian:
  - The Guardian là loại tháp chứa quân lính đứng canh gác phía trên, loại tháp này có tốc độ tấn công nhanh nhất, tuy nhiên sát thương thấp và tầm tấn công ngắn.

Hình 35: Mô hình The Guardian

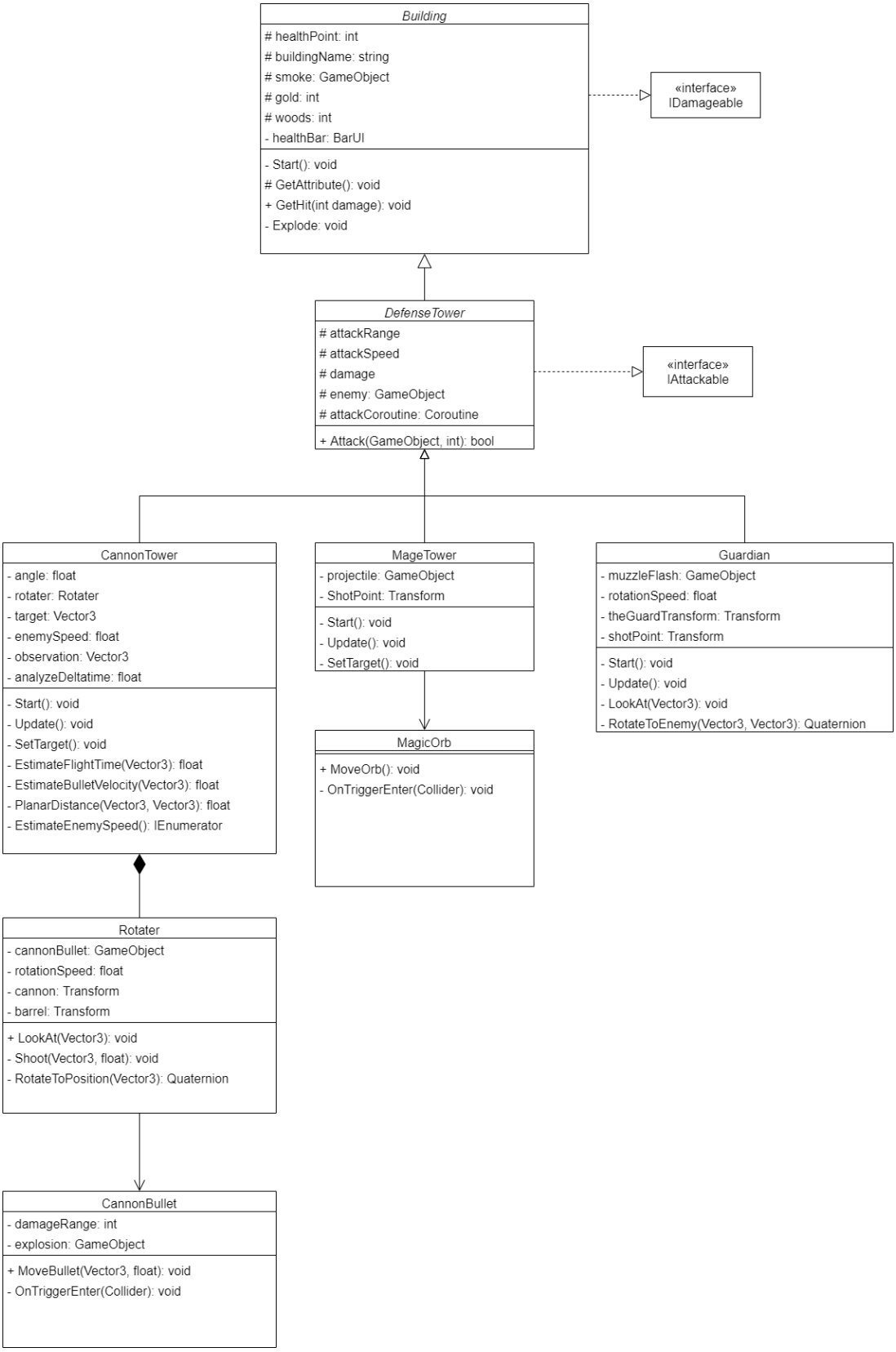
  - The Guardian phù hợp với nhiều loại chiến thuật khác nhau, tuy nhiên cần khéo léo chọn vị trí phù hợp vì khả năng chống chịu khá thấp.



Hình 36: Minh họa hành vi The Guardian

- The Guardian có tốc độ tấn công nhanh, khi bắn quân địch sẽ ngay lập tức chịu sát thương.

b) Thiết kế hướng đối tượng:



Hình 37: Class diagram hệ thống tháp phòng thủ

c) Hiện thực:

Giải thuật tấn công cho Cannon Tower:

- Đầu tiên Cannon Tower sẽ tính vận tốc ban đầu cần có để đạn bay đến vị trí quân lính hiện tại dựa vào góc nghiêng của cannon, khoảng cách đến mục tiêu và chiều cao tháp:

```
float EstimateBulletVelocity(Vector3 target) {
      float g = Physics.gravity.magnitude;
      Vector3 barrelPosition = rotater.barrel.position;
      float d = PlanarDistance(barrelPosition, target);
      // Distance along the y axis between objects
      float yOffset = barrelPosition.y - target.y;
      float initialVelocity = Mathf.Sqrt((0.5f * g * Mathf.Pow(d, 2)) / (d *
          Mathf.Tan(angle) + yOffset)) / Mathf.Cos(angle);
      return initialVelocity;
}
float PlanarDistance(Vector3 source, Vector3 destination) {
     Vector3 planarSource = new Vector3(source.x, 0, source.z);
     Vector3 planarTarget = new Vector3(destination.x, 0, destination.z);
     return (distance from planarSource to planarTarget);
}
```

- Sau đó tính thời gian đạn bay:

```
float EstimateFlightTime(Vector3 destination) {
     Vector3 barrelPosition = rotater.barrel.position;
     float d = PlanarDistance(barrelPosition, destination);
     float time = d / (EstimateBulletVelocity(destination) *
          Mathf.Cos(angle));
     return time;
}
```

- Bởi vì đạn cannon có tốc độ bay khá chậm, trong khoảng thời gian đạn di chuyển thì quân lính cũng đã đi được một quãng đường so với vị trí ban đầu. Để tính được quãng đường này, tháp cannon sẽ tạo một tác vụ bất đồng bộ ước lượng vận tốc tức thời của quân lính và cập nhật vào v_{enemy} trong từng khoảng thời gian Δt nhỏ:

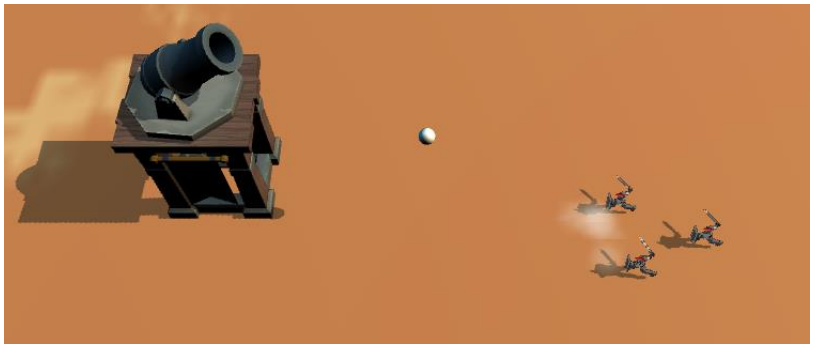
```
IEnumerator EstimateEnemySpeed() {
      yield return new WaitForSeconds(timeToGetSpeedSample);
      if (enemy != null){
         enemySpeed = Vector3.Distance(enemy.transform.position,
observation) / timeToGetSpeedSample;
         observation = enemy.transform.position;
      }
      else enemySpeed = 0;
```

```
        estimateEnemySpeedCoroutine = null;
    }
```

- Dựa vào vận tốc này, tháp sẽ tiếp tục tính quảng đường mục tiêu sẽ di chuyển được trong quãng thời gian đạn bay đến và cập nhật vào tọa độ được chọn làm mục tiêu:

```
float t = EstimateFlightTime(enemy.transform.position);
float s = t * enemySpeed;
Vector3 offset = s * enemy.transform.forward;
```

- Một tác vụ bất đồng bộ khác là Attack(), nó cho phép tháp cannon tấn công sau mỗi khoảng thời gian nhất định.
- Khi quân địch tiến lại gần, rotater sẽ xoay tháp hướng về phía tọa độ mục tiêu. Dựa vào thư viện Unity.Quaternion, ta sẽ tính toán được góc quay phù hợp trong mỗi frame game.
- Khi tháp đã xoay đến vị trí đó, và sau khoảng thời gian t cho phép, đạn cannon sẽ được bắn ra với một lực dựa trên khối lượng và vận tốc đạn đã tính được.
- Đạn cannon khi có va chạm sẽ phát nổ và tính toán những đối tượng ở gần nó, những quân lính nào nằm trong phạm vi đều chịu sát thương.



Hình 38: Ảnh chụp trong game

Giải thuật tấn công cho Mage Tower:

- Sau mỗi khoảng thời gian nhất định tỉ lệ với tốc độ tấn công của tháp, Mage Tower sẽ sinh ra một quả cầu phép thuật.
- Quả cầu sẽ luôn điều hướng về phía mục tiêu và di chuyển đều với vận tốc cố định mà không chịu tác động của trọng lực.

```
if (target != null){
        transform.LookAt(target.transform);
```

```
            transform.Translate(transform.forward * 10 * Time.deltaTime,
              Space.World);
            lastAttackPoint = target.transform.position;
        }
        else{
            transform.LookAt(lastAttackPoint);
            transform.Translate(transform.forward * 10 * Time.deltaTime,
              Space.World);
            if (Vector3.Distance(transform.position, lastAttackPoint) < 1)
                Destroy(gameObject);
        }
```

- Khi nhận được sự kiện va chạm với một mục tiêu đã xác định, quả cầu phép thuật sẽ gây sát thương lên nó.

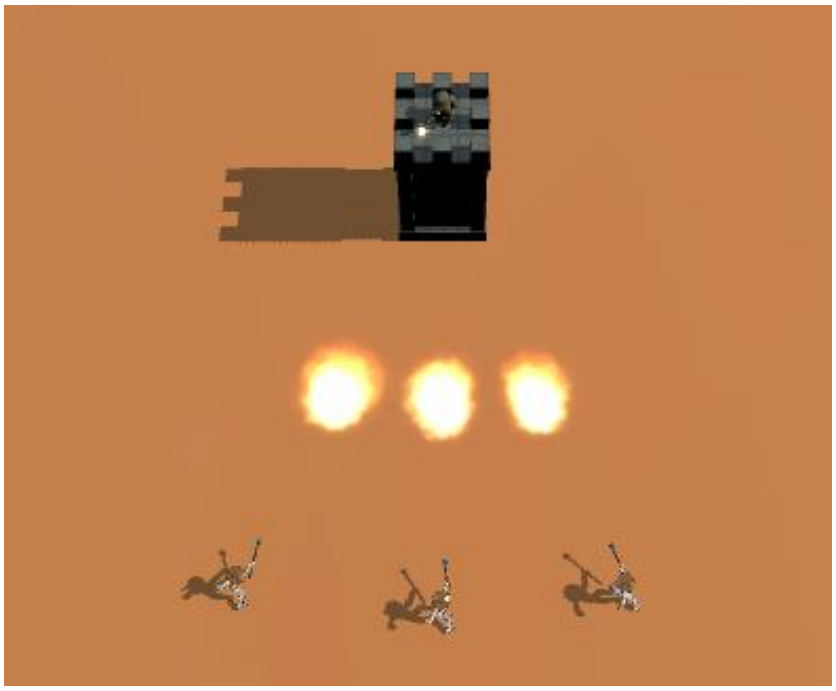
```
void OnTriggerEnter(Collider other) {
    if (target == other.gameObject) {
        target.GetComponent<IDamageable>().GetHit(damage);
        Destroy(gameObject);
    }
}
```

Hình 39: Ảnh chụp trong game

The Guardian có tốc độ tấn công khá nhanh, khi bắn quân địch sẽ ngay lập tức chịu sát thương.

```
IEnumerator Attack() {
        yield return new WaitForSeconds(attackSpeed);
        if (enemy != null) {
            enemy.GetComponent<IDamageable>().GetHit(damage);
        }
        attackCoroutine = null;
```

}



Hình 40: Ảnh chụp trong game

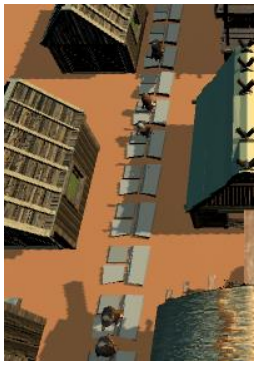
5. Hiện thực dân làng:

a) Giới thiệu chung:

Trong ngôi làng của người chơi sẽ có một số lượng nhất định các dân làng và sẽ tăng theo quy mô công trình nhà ở của người chơi.

Dân làng trong một thời điểm sẽ thực hiện 1 trong 5 loại hoạt động như: di chuyển xung quanh, đứng tại một vị trí, đi vào một ngôi nhà, đi đến cửa hàng hoặc cưỡi một phương tiện.

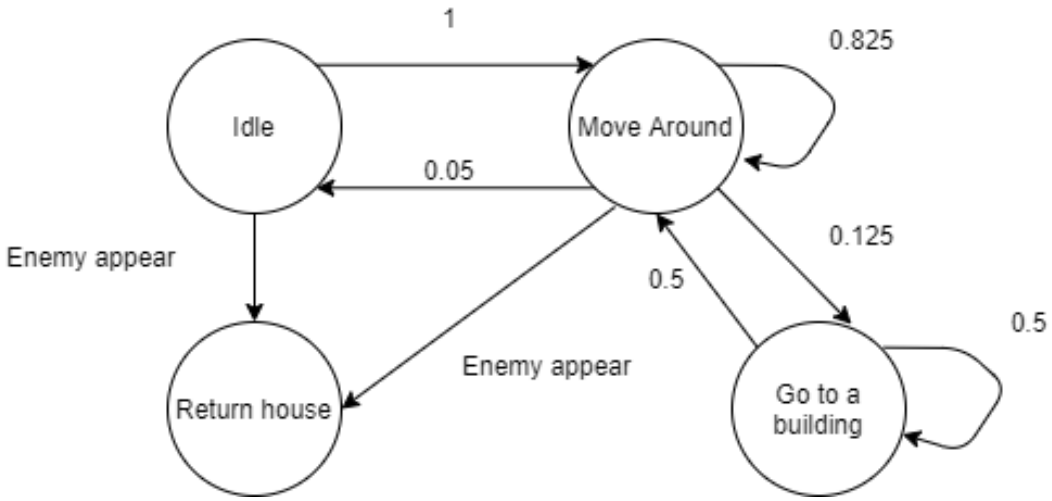
Khi di chuyển, các dân làng sẽ đi đến một điểm ngẫu nhiên trên các con đường được xây dựng bởi người chơi.



Hình 41: Dân cư trong ngôi làng

b) Hiện thực:

Hành vi của dân làng có thể được mô hình hóa thành một state machine như sau:



Hình 42: State machine biểu thị trạng thái dân làng

Trong trạng thái move around, vì không gian tìm kiếm khá nhỏ, việc tìm đường đi đơn giản sử dụng giải thuật DFS để duyệt đến một điểm ngẫu nhiên nào đó.

Khi bắt đầu trò chơi, mỗi đoạn đường có chứa đối tượng PathObject (những điểm có thể di chuyển tương ứng với các movable model trong game) sẽ tự tìm kiếm các PathObject lân cận và kết nối với nhau tạo thành một đồ thị.

```
Search(initial, goal):
        if (initial is goal) then return “Solution”
        initial.depth = 0
        open = new Stack;
        closed = new Set;
        insert(open, copy(initial))
        while(open is not empty) do
        begin
                n = pop(open)
                insert(closed, n)
                foreach valid move m at n do
                begin
                        next = state when playing m at n
                        if (next does not belong to closed) then
                        begin
                                next.depth = n.depth + 1
                                if (next is goal) then
```

```
                    return “Path”
                if (next.depth < maxDepth) then
                    insert(open, next)
            end
        end
    return “No path”
```

6. Xây dựng hệ thống quân lính:

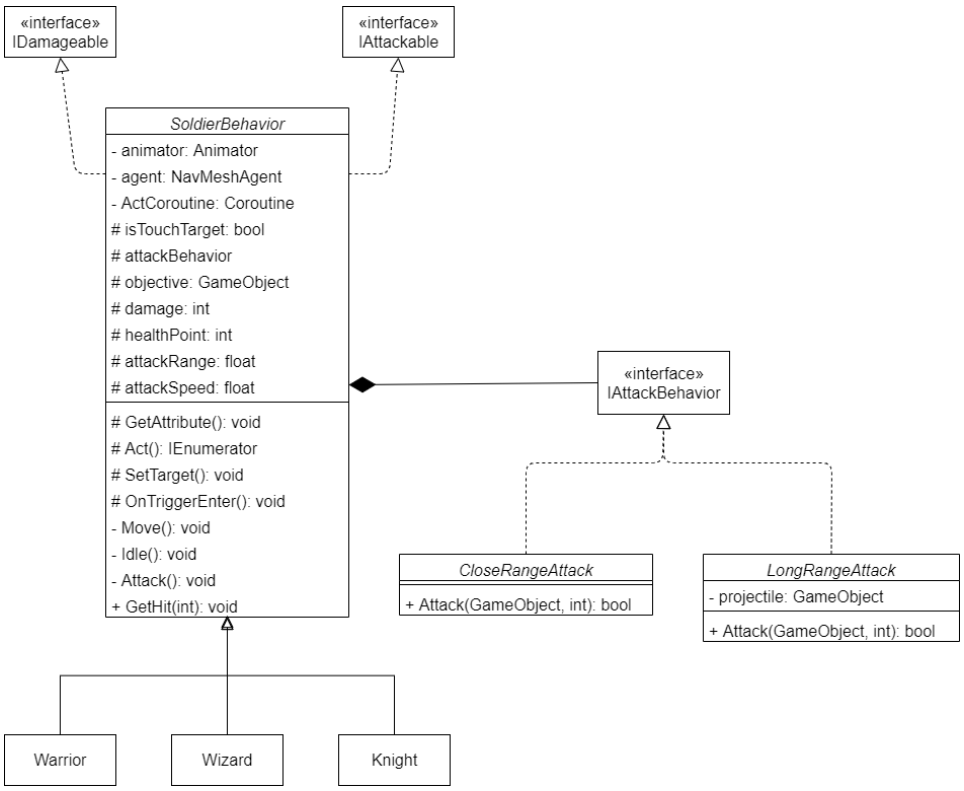
a) Giới thiệu chung:

Quân lính là thành phần không thể thiếu trong trò chơi, để chinh phục các ngôi làng khác, người chơi cần xây dựng được một đội quân hùng mạnh.

Có nhiều loại quân lính với các chỉ số khác nhau, tuy nhiên chúng đều được kế thừa từ lớp SoldierBehavior. Tại một thời điểm sẽ có hành vi thuộc một trong ba trạng thái: đứng yên, di chuyển, tấn công. Có hai kiểu tấn công là đánh cận chiến và tấn công tầm xa.

Người chơi click chọn quân lính từ đội quân của mình, sau đó click chọn một vị trí trên bản đồ bên ngoài ngôi làng của đối phương, quân lính sẽ được đặt ra và sau đó chúng tự tìm mục tiêu và tìm đường đi để di chuyển đến tấn công.

b) Thiết kế hướng đối tượng:



Hình 43: Class diagram quân lính

Lớp quân lính được thiết kế theo mẫu Design Pattern Strategy.

c) Hiện thực:

Khi bắt đầu giao tranh PlayerBehavior sẽ load dữ liệu của đối tượng Player và Opponent tương ứng. Nếu dữ liệu đội quân của Player đã được load, sự kiện OnPlayerInit được kích hoạt và đối tượng TroopsManager tiếp nhận thông tin quân lính của Player. Tiếp đến AttackingUI sẽ dựa vào đó xuất ra màn hình thông tin đội quân hiện tại.

Nếu người chơi chọn quân lính nào đó, instance của lớp TroopsManager sẽ cập nhật prefab của loại quân tương ứng.

Mỗi khi người chơi đặt quân lính ra bản đồ, đối tượng SpawnTroops quan sát input và tiếp nhận sự kiện này, nó thực hiện tác vụ Spawn(). Phương thức Spawn() chiếu một tia sáng từ camera xuống và khi cắt địa hình tại điểm nào, quân lính sẽ được tạo ra ở đó.

```
void Spawn() {
        Ray ray = Camera.main.ScreenPointToRay(Input.mousePosition);
        RaycastHit hit;
        if (Physics.Raycast(ray, out hit)) {
```

```
                    if (hit.transform.tag.Equals("Desert")) {
                        Instantiate(troopsManager.ChosenSoldier, hit.point,
                        Quaternion.identity);
                    }
                }
            }
```

Các quân lính chứa component NavMeshAgent khi được sinh ra sẽ tự tìm đường di chuyển đến mục tiêu, dựa trên NavMesh đã được bake từ trước.

Bởi vì đích đến được xác định ở tâm của mục tiêu, tức là điểm nằm ở bên trong của công trình, các quân lính sẽ không thể nào đến được đó. Như vậy, có hai kịch bản quân lính sẽ dừng lại và tấn công. Trường hợp thứ nhất là khi quân lính đã va chạm với Collider của mục tiêu (lính cận chiến), và trường hợp thứ hai là khi mục tiêu đã nằm trong tầm tấn công của chúng (lính đánh xa).

```
protected IEnumerator Act() {
    if (objective != null) {
            float distance = Vector3.Distance(transform.position,
objective.transform.position);
            if (distance <= agent.stoppingDistance || isTouchTarget == true) {
                Idle();
                Attack(objective);
            }
            else Move();
    }
    yield return new WaitForSeconds(attackSpeed);
     setStatusCoroutine = null;
}
```

ss

Chương VIII. Kiểm thử và đánh giá

1. Phạm vi kiểm tra:

Tên chương trình:	Rising Steps
Phiên bản:	v1.1
Môi trường	Thiết bị: laptop Dell Inspiron 5490, điện thoại thông minh Samsung Galaxy S10. Hệ điều hành: Windows 10, Android.
Các chức năng đã được test	Điều khiển camera Xây dựng công trình Load dữ liệu người chơi Lưu dữ liệu người chơi Tìm kiếm đối thủ Đặt quân lính Tìm đường đi cho quân lính Cơ chế tấn công Huấn luyện quân lính Hành vi tháp Cannon Hành vi tháp Mage Hành vi tháp Guardian Hành vi dân làng
Các chức năng chưa được test	

2. Kiểm tra các tính năng:

STT	Module	Số testcases	Pass
1	Điều khiển camera	6	6
2	Xây dựng công trình	10	10
3	Load dữ liệu người chơi	8	8
4	Lưu dữ liệu người chơi	8	8
5	Tìm kiếm đối thủ	4	4
6	Tìm đường đi cho quân lính	15	15
7	Cơ chế tấn công	9	9
8	Huấn luyện quân lính	4	4
9	Đặt quân lính	10	10
10	Hành vi tháp Cannon	13	13
11	Hành vi tháp Mage	13	13
12	Hành vi tháp Guardian	13	13
13	Hành vi dân làng	6	6

Chương IX. Tổng kết

1. Kết luận:

Qua quá trình khảo sát, tìm hiểu, tổng hợp các thông tin và kiến thức cần thiết; tôi đã xây dựng được trò chơi đạt hầu hết các mục tiêu đặt ra, bước đầu nhận được nhiều phản hồi tích cực và có thể đưa lên các cửa hàng ứng dụng.

Mặc dù chương trình đã được hoàn thiện, nó cũng có thể còn một số lỗi sai có thể gặp. Những lỗi nào được phát hiện sẽ được ghi nhận và sửa chữa trong các phiên bản sau.

2. Nhận xét chung:

a) Ưu điểm:

- Chương trình có thể chạy được trên nhiều nền tảng như windows, Android.
- Cách chơi đơn giản, dễ tiếp cận cho mọi đối tượng.
- Có thể kết nối mạng để nhiều người chơi với nhau.
- Trò chơi kích thích sức sáng tạo của người chơi cùng với sự đa dạng trong chiến thuật chơi.

b) Khuyết điểm:

- Các mô hình 3D chưa được phong phú và đa dạng.
- Một số hiệu ứng hình ảnh có chất lượng còn hạn chế.
- Hệ thống âm thanh trong trò chơi còn đơn giản.

3. Hướng phát triển tiếp theo:

Game “Rising Steps” đã hoàn thiện được phiên bản đầu tiên. Trong tương lai, các phiên bản sau sẽ cập nhật thêm một số tính năng mới, đồng thời tối ưu thêm các hiệu ứng đồ họa.

Một số định hướng cho việc xây dựng các chức năng tiếp theo:

- Xây dựng thêm nhiều màn chơi cho chế độ Single Player.

- Tăng cường số lượng loại quân lính và các công trình với các đặc điểm khác nhau.
- Bổ sung thêm một số thuộc tính phụ cho quân lính.
- Tối ưu hóa giao diện UI trong trò chơi.
- Thêm tính năng tự động tạo đội quân tấn công từ NPC.
- Tạo thời tiết trong game và ảnh hưởng từ thời tiết tới các hoạt động trong game.
- Thêm chức năng đánh giá cấu trúc ngôi làng của người chơi: người chơi có thiết kế ngôi làng tốt sẽ được nâng cao hiệu quả hoạt động bên trong ngôi làng.

TÀI LIỆU THAM KHẢO:

[1] Ernest Adams (2014). *Fundamentals of game design*, third edition.

[2] Jesse Schell (2015). *The Art of Game Design, A Book of Lenses*, second edition.

[3] Georgious N. Yannakakis, JulianTogelius (2018). *Artificial Intelligence and Games*, Springer.

[4] Nguyễn Quốc Bảo, Nguyễn Đình Hào, Phạm Quang Minh, Hoàng Lê Chánh Tú (2019). *Mô phỏng đám đông sử dụng Explicit Corridor Map*, Trường Đại học Bách Khoa – Đại học Quốc gia Thành Phố Hồ Chí Minh.

[5] Stuart Russell, Peter Norvig (2009). *Artificial Intelligence: A Modern Approach*, third edition.

[6] Nguyễn Văn Hiệp (2016). *Lập trình hướng đối tượng bằng VC#*, Nhà xuất bản Đại học Quốc gia Thành phố Hồ Chí Minh.

[7] Nguyễn Trung Trực (2016). *Kỹ thuật lập trình*, Nhà xuất bản Đại học Quốc gia Thành phố Hồ Chí Minh.

[8] Michelle Menard (2011). *Game Development with Unity*, first edition.

[9] Jeff W. Murray (2014). *C# Game Programming Cookbook for Unity 3D*.

[10] Jogn P. Doran (2017). *Unity 2017 Mobile Game Development: Build, deploy, and monetize game for Android and IOS with Unity*.

[11] *"https://docs.unity3d.com/2020.1/Documentation/Manual/index.html"*, truy cập vào 2020-08-04.

[12] *"https://docs.unity3d.com/2020.1/Documentation/ScriptReference/index.html"*, truy cập vào 2020-08-10.

[13] *"https://en.wikipedia.org/wiki/Game",* truy cập vào 2020-06-11.

[14] *"https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/operators/await"*, truy cập vào 2020-06-04.

[15] Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides (2012). *Design Patterns: Elements of Reusable Object-Oriented Software*.

[16] Nguyễn Thị Bé Bảy, Huỳnh Quang Linh, Trần Thị Ngọc Dung (2011). *Vật lý đại cương*.

[17] *"https://en.wikipedia.org/wiki/Quaternion"*, truy cập vào 2020-08-24

[18] Meier, Andreas, Kaufmann, Michal (2019). *SQL & NoSQL Databases: Model, Languages, Consistency Options and Architectures for Big Data Management*.

[19] Ian Sommerville (2016). *Software Engineering*, global edition.

[20] Houssem Yahiaoui (2017). *Firebase Cookbook.*

[21] Kevin D. Saunders, Jeannie Novak (2007). *Game Development Essentials Game Interface Design*, first edition.

[22] Rick Craig, Steve Jaskeil (2002). *Systematic Software Testing*, first edition.

[23] Martin Fowler (2004). *UML distilled: a brief guide to the standard object modeling language*, third edition.

[24] David M. Dikel (2001). *Software Architecture: Organizational Principles and Patterns.*

[25] Wendy Stahler, Dustin Clingman, Kaveh Kahrizi (2004). *Begin Math and Physics for Game Programmers*.