* UPS (Updates Per Second ) là một khái niệm trong lập trình game để đo tần suất cập nhật logic của trò chơi (Độ nhanh của game logic, vị trí, tốc độ, va chạm, ,…) trong một đơn vị thời gian cố định , thường là mỗi giây.
* UPS quyết định số lần mà logic của trò chơi được cập nhật và tính toán , không phải là số lần đồ họa được vẽ (Được thể hiện bằng FPS – Prams Per Second ).
* Ta cứ hiểu rằng UPS sẽ quyết định, liên quan đến số lần cập nhật trò chơi.Nó cũng liên quan đến khái niệm vòng lặp game(game loop) , là một vòng lặp lặp đi lặp lại để xử lí các sự kiện , cập nhật trạng thái và hiển thị hình ảnh của game. Vòng lặp game có thể được thiết kế theo nhiều cách khác nhau, tùy thuộc vào mục tiêu và các yêu cầu của game.
* Cũng giống như FPS thì cái UPS này chúng ta cũng có thể can thiệp được.

======================================================

* Tại sao chúng ta lại phải phân chia FPS và UPS là một chiến lược thường được sử dụng trong lập trình game để tách biệt giữa hai khía cạnh chính của trò chơi, hiển thị đồ họa và cập nhật logic. Như vậy thì cái code của chúng ta sẽ trở nên sạch sẽ hơn và game của chúng ta cũng sẽ dễ nâng cấp và bảo trì hơn.
* Tính hiệu suất :

+ FPS đo lường hiệu suất đồ họa, quyết định khả năng của hệ thống hiển thị hình ảnh mượt mà. Số FPS cao thường tạo ra trải nghiệm đồ họa tốt hơn

+ UPS đo lường hiệu suất logic và tính toán trong trò chơi. Số UPS cao đảm bảo rằng logic và tính toán của trò chơi được cập nhật một cách chính xác hơn là việc chúng ta gộp chung hai cái lại trông nó rất phèn😊

* Tách biệt giữa logic và đồ họa ra còn một số lí do nữa là do FPS và UPS thường không liên quan chặt chẽ với nhau. Một số trò chơi có thể có FPS cao nhưng UPS thấp hoặc ngược lại. Việc tách biệt giữa hai khía cạnh này giúp phân biệt vấn đề hiệu suất của hệ thống liên quan đến đồ họa và logic.
* Trong một số trường hợp, hiển thị đồ họa mượt có thể mà có thể quan trọng hơn việc cập nhật logic nhanh chóng và ngược lại. Tách biệt giữa FPS và UPS giúp các nhà phát triển quyết định ưu tiên cái nào quan trọng hơn tùy thuộc vào yêu cầu cụ thể của trò chơi.
* Khi FPS và UPS được tách biệt , chúng ta có thể tối ưu hóa và điều chỉnh mỗi khía cạnh một cách độc lập. Điều này giúp nâng cao hiệu suất toàn diện của trò chơi mà không làm ảnh hưởng quá mức đến khía cạnh khác.
* Việc giữ cho FPS và UPS riêng biệt giúp trong việc đánh giá hiệu suất và phân tích vấn đề khi cần thiết . Nếu FPS thấp, có thể là vấn đề về đồ họa, nếu UPS thấp, có thể là vấn đề về logic
* Tóm lại thì việc tách biệt FPS và UPS giúp các nhà phát triển dễ dàn quản lí và cải thiện hiệu suất của trò chơi một cách linh hoạt và chính xác hơn.

=======================================================

* Khai báo một biến lưu trữ số lần cập nhật logic (Updates Per Second – UPS ) mà trò chơi sẽ thực hiện trong mỗi giây.
* Giá trị của biến này là 200 , cho thấy bạn đang cố gắng cập nhật logic của trò chơi 200 lần mỗi giây.

private final int UPS\_S = 200;

* Điều này ám chỉ rằng mỗi giây, trò chơi của bạn sẽ thực hiện cập nhật logic 200 lần, nhằm đảm bảo rằng các thay đổi trong trạng thái và hành vi của trò chơi được xử lí một cách liền mạch và chính xác
* Giá trị của UPS\_S thường được lựa chọn dựa trên độ phức tạp của trò chơi và yêu cầu cụ thể về tần suất cập nhật logic.
* Chúng ta phải chuẩn bị trước một thực thể trò chơi player với phương thức update để có thể thực hiện các phương thức cho nhân vật đó.
* Chúng ta tạo thêm một biến để theo dõi số lần cập nhật logic đã được thực hiện trong khoảng 1 giây. Trong đoạn mã của bạn, updates được khởi tạo với giá trị bằng 0 và sau đó được tăng lên mỗi khi một lần cập nhật logic thực hiện.

long currentTime = System.nanoTime();

deltaU += (currentTime – previousTime) / timePerUpdate;

previousTime = currentTime;

if(deltaU >=1)

+ Gọi hàm update

+ Tăng biến đếm số lần cập nhật lên

+ Giảm deltaU –- Tức là nó sẽ trở về 0.

* Chạy lần đầu tiên :

+ currentTime là thời điểm hiện tại tính bằng nanoTime;

+ deltaU = 0;

+ previousTime được đặt là curentTime ban đầu.

A / B < 1 khi nào khi A < B

A / B >= 1 khi nào khi A >= B

(currentTime – previousTime) / timePerUpdate nghĩa là cái phép tính này mà cho kết quả nhỏ hơn 1 khi curentTime – previiousTime >= timePerUpdate = 16,67 mili giây với UPS\_S = 60;

* Ví dụ, nếu bạn muốn game cập nhật 60 lần mỗi giây, thì timePerUpdate sẽ bằng 1000000000 / 60 = 16666666.67 nano giây. Nếu thời gian trôi qua giữa hai lần cập nhật là 20000000 nano giây, thì **delta time** sẽ bằng 20000000 / 16666666.67 = 1.2 lần cập nhật.

========================================================

* Chương trình này là một ví dụ về gameloop , một vòng lặp liên tục trong quá trình chơi game.
* Mỗi lần lặp , nó xử lí đầu vào của người dùng mà không chặn , cập nhật các trạng thái của game và vẽ game lên màn hình. Nó theo dõi sự trôi qua của thời gian để điều khiển tốc độ của game và vẽ game lên màn hình.
* Nó theo dõi thời gian trôi qua của thời gian để điều khiển tốc độ của game. Mẫu này tách biệt sự tiến triển của thời gian trong game với đầu vào của người dùng và tốc độ của bộ xử lí.

+ TimeperFrame : Thời lượng cho mỗi khung hình.

+ TimePerUpdate: Thời lượng mỗi lần cập nhật kéo dài, tính bằng nano giây.

timePerFrame = 1 tỷ/ 60 = 16 666 666.67 nano giây, tức là mỗi khung hình sẽ kéo dài khoảng 16.7 mili giây.

timePerUpdate = 1 tỷ / 120 = 8 333 333.33 nano giây, tức là mỗi lần cập nhật sẽ kéo dài khoảng 8.3 mili giây.

deltaU và deltaF ban đầu bằng 0 , tức là chưa có thời gian nào trôi qua để cập nhật hay vẽ game.

previous là thời điểm bắt đầu mỗi lần lặp, giả sử là 0 nano giây.

Bây giờ bạn bắt đầu vòng lặp game , và giả sử thời gian hiện tại là 10 mili giây, tức là mỗi khung hình 10 000 000 nano giây

* deltaU = (10 triệu – 0 ) / (8 triệu 3 ) = 1,2 tức là đã đủ thời gian để cập nhật game một lần , và còn dư ra 0.2 lần nữa
* deltaF = (10 triệu – 0 ) / (16 triệu 7) = 0.6 tức là chưa đủ thời gian để vẽ khung hình , và còn thiếu 0.4 khung hình nữa.
* previousTime = 10 triệu , để chuẩn bị cho lần lặp tiếp theo.

-Do deltaU >=1 nên ….

-Do deltaF < 1 nên…

* Tiếp tục vòng lặp , và giả sử thời gian hiện tại là 20 mili giây, tức là 20 triệu nano giây , khi đó , bạn có:

- deltaU = 0.2 + (20000000 - 10000000) / 8333333.33 = 2.4, tức là đã đủ thời gian để cập nhật game hai lần, và còn dư ra 0.4 lần nữa.

- deltaF = 0.6 + (20000000 - 10000000) / 16666666.67 = 1.2, tức là đã đủ thời gian để vẽ game một khung hình, và còn dư ra 0.2 khung hình nữa. - previousTime = 20000000, để chuẩn bị cho lần lặp tiếp theo.

Chia cho timePerUpdate và timePerFrame giúp giữ lại phần thập phân dư thừa từ phép chia, như bạn đã mô tả, để đảm bảo rằng thời gian dư thừa sẽ được tính vào lần cập nhật hoặc vẽ khung hình tiếp theo, giúp đồng bộ hóa với chu kỳ thời gian mong muốn (UPS\_S và FPS\_SET). Điều này giúp tránh việc tính toán thừa trong trường hợp tần suất thực tế của vòng lặp không chính xác với tần suất mong muốn.