Table of Contents

[Thông tin 1](#_Toc181174701)

[Thêm tank mới 1](#_Toc181174702)

[Thêm map mới 2](#_Toc181174703)

[Cấu trúc 2](#_Toc181174704)

# Thông tin

Team.at(0) là team của người chơi.

Team.at(0).tanks.front() là tank của người chơi

Tốc độ bắn = tốc độ bắn của tank trong Bang Bang zing me \* 5 - 20;

Tầm bắn = tầm bắn của tank trong Bang Bang zing me - 20;

Tốc độ di chuyển = Tốc độ di chuyển của tank trong Bang Bang zing me - 20;

-Công thức tính máu mất khi nhận sát thương

Máu mất một lượng = TC của địch \* 100 / (100 + Giáp của mình - Xuyên giáp của địch)

# Thêm tank mới

-Thêm hàm collider của tank cùng bullet và chỉ số vào trong 2 namespace sau: namespace colliders, namespace spec

-bbg::loadTankCollection(): Thêm các thông tin cho tank

-Triển khai TankBehavior.

-Tam ban pegasus (Tank NL): 415

HP: 1948 (2267 BB kul)

TC: 192 (258)

Giap: 78 (131)

Khien 87 (138)

-Apocalypse II (Tank VL)

HP: 2608

TC: 271

Giap: 125

Khien: 132

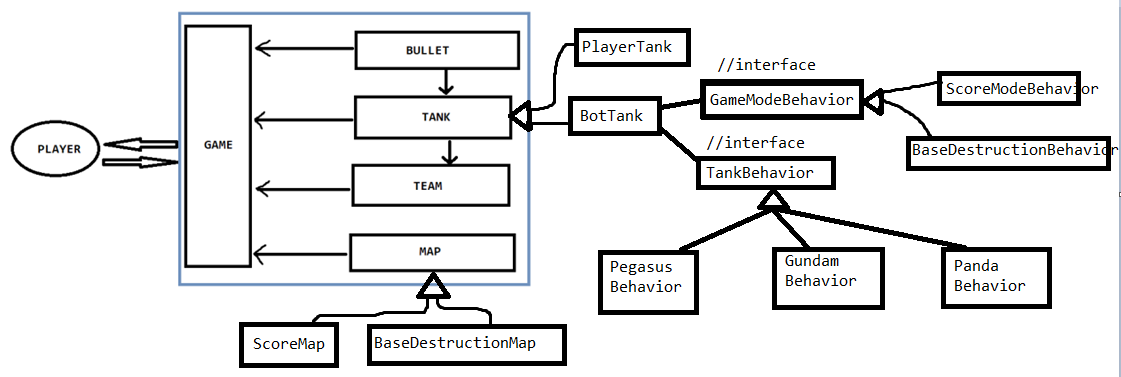
# Thêm map mới

-Thêm collider cho lớp obstacle trong namespace colliders

-Thêm các vị trí spawn cho mỗi map

-Map chưa thiết lập chướng ngại vật xong

# Cấu trúc



Chú thích:

Các dấu mũi tên là hướng được gọi: VD Player được Game gọi (lấy input), hay game gọi các class khác như bullet, tank,…

Dấu mũi tên là đầu hình tam giác là quan hệ kế thừa

Còn đường thẳng bình thường thì là quan hệ Association như sơ đồ usecase. VD: Bottank và GameModeBehavior có quan hệ Association, tức là trong BotTank có thuộc tính có kiểu dữ liệu GameModeBehavior.

-Trong Map: triển khai Strategy Pattern với Map là interface (trong C++ ko có, nên Map là abstract class để thay thế), có Concrete Strategy là: ScoreMap, BaseDestructionMap,.. (Context: Game). Mỗi lớp hiện thực đó sẽ hiện thực các method như checkVictoryCondition(),…



-BotTank: có 2 thuộc tính GameModeBehavior và TankBehavior

Vấn đề: Khi random 1 map, và random tank cho bottank, thì làm sao bottank sẽ chọn được GameModeBehavior và TankBehavior cho tank đó?

Đơn giản là if else hay switch-case:  
TankBehavior\* tankBehavior = nullptr;

switch (randomTank) {

case PEGASUS:

tankBehavior = new PegasusBehavior();

break;

case GUNDAM:

tankBehavior = new GundamBehavior();

break;

case PANDA:

tankBehavior = new PandaBehavior();

break;

// Thêm các loại tank khác nếu có

}

Nhưng nếu là GameModeBahavior thì làm vậy được, vì chế độ game thì có vài cái: tính điểm, phá căn cứ,…

Còn Tank thì nếu thiết kế cỡ 100 tank thì không ổn.

Giải pháp: sử dụng factory pattern (trong design pattern)

Nó cung cấp một cách thức để tạo ra các đối tượng mà không cần phải biết cụ thể loại đối tượng đó là gì. Thay vào đó, bạn sẽ sử dụng một "nhà máy" để tạo ra các đối tượng cho bạn. Nó giải quyết vấn đề là ta muốn tạo đối tượng TankBehavior nhưng không xác định chính xác rằng phải tạo ra PegasusBehavior hay GundamBehavior hay v.v.. thì có phải sử dụng if else hay không?

Cái pattern này sẽ đưa ra giải pháp là có 1 nhà máy sản xuất các đối tượng cùng họ (các đối tượng con của 1 interface)

Lớp TankBehaviorFactory sẽ quản lý một danh sách các loại TankBehavior khác nhau. Khi cần tạo TankBehavior cho một loại tank nào đó, chỉ cần gọi factory, và nó sẽ trả về đúng đối tượng TankBehavior.