

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



**BÁO CÁO
LUẬN VĂN TỐT NGHIỆP ĐẠI HỌC**

GAME SURVIVAL2D

Ngành: Khoa học Máy tính

Hội đồng: Khoa học Máy tính
GVHD: ThS. Vương Bá Thịnh
GVPB: KS. Trần Huy
—oo—

SVTH 1: Trần Minh Quân (1712831)
SVTH 2: Nguyễn Minh Tiên (1713484)

TP. Hồ Chí Minh, Tháng 05/2023



LỜI CAM ĐOAN

Nhóm xin cam đoan đề tài này là thành quả nghiên cứu của riêng mình, dưới sự hướng dẫn của ThS. Vương Bá Thịnh. Các nội dung nghiên cứu, phân tích và kết quả trong đề tài này là trung thực và chưa công bố dưới bất kỳ hình thức nào trước đây.

Nếu phát hiện có bất kỳ sự gian lận nào, nhóm hoàn toàn chịu trách nhiệm về nội dung đề tài luận văn của mình và chịu mọi hình thức kỷ luật trước Ban chủ nhiệm Khoa và Ban giám hiệu nhà trường.



LỜI CẢM ƠN

Trước hết, nhóm xin cảm ơn thầy Vương Bá Thịnh đã đồng hành cùng nhóm trong suốt thời gian thực hiện đề tài. Cảm ơn thầy Trần Huy vì những góp ý quý giá của thầy trong buổi phản biện đề tài, đã giúp nhóm nhận ra nhiều thiếu sót trước đó và chỉnh sửa lại hoàn thiện hơn. Cảm ơn các thầy, cô trong khoa Khoa học và Kỹ thuật Máy tính, trường Đại học Bách Khoa, Đại học Quốc gia Thành phố Hồ Chí Minh đã truyền đạt kiến thức trong những năm học tập. Những tri thức quý báu ấy đã giúp nhóm có thể vận dụng để làm nên đề tài này.

Nhóm cũng xin cảm ơn các sếp, các bạn đồng nghiệp tại ZingPlay Game Studios, quá trình làm việc ở đây đã giúp các thành viên trong nhóm trau dồi kinh nghiệm thực tế trong quá trình làm ra một game. Các bạn cũng là những người tạo nguồn cảm hứng cho việc lựa chọn một chủ đề về game làm đề tài luận văn tốt nghiệp.

Vì khả năng của bản thân còn hạn chế, trong quá trình làm đề tài nhóm không tránh khỏi những sai sót, kính mong nhận được ý kiến đóng góp từ thầy cô và các bạn để đề tài được hoàn thiện hơn.

Cuối cùng, nhóm xin chân thành cảm ơn thầy cô và các bạn đã dành thời gian đọc tài liệu này.



TÓM TẮT ĐỀ TÀI

Survival2D là game bắn súng Battle Royale lấy ý tưởng từ game Player Unknown Battle Ground (PUBG). **Survival2D** sử dụng đồ họa 2D, không yêu cầu phần cứng cao; thích hợp chơi trên trình duyệt web, điện thoại, có thể tiếp cận với các nền tảng không thường được sử dụng để chơi game nhiều như Linux và MacOS.

Báo cáo này gồm 9 chương, bao gồm: giới thiệu đề tài, thiết kế game, công nghệ sử dụng, phân tích hệ thống, thiết kế hệ thống, kiểm thử, đánh giá kết quả đạt được, định hướng phát triển và công cụ sử dụng.

Chương giới thiệu đề tài trình bày tổng quan về nội dung, mục tiêu và giới hạn của đề tài.

Chương thiết kế game trình bày về luật game và các phần liên quan đến thiết kế của game.

Chương công nghệ sử dụng trình bày các công nghệ đã được sử dụng khi triển khai hệ thống.

Chương phân tích hệ thống trình bày các vấn đề lớn gấp phải để hiện thực được game.

Chương thiết kế hệ thống trình bày định hướng giải quyết các vấn đề lớn kể trên.

Chương kiểm thử trình bày các kết quả kiểm thử sau khi hiện thực đề tài.

Chương đánh giá kết quả đạt được trình bày các kết quả mà nhóm đã đạt được sau quá trình thực hiện luận văn, và đánh giá kết quả này.

Chương định hướng phát triển trình bày ưu điểm, nhược điểm và hướng phát triển của đề tài.

Chương công cụ sử dụng liệt kê các công cụ mà nhóm đã sử dụng trong quá trình thực hiện đề tài, tạo thành sản phẩm.

Mục lục

1 Giới thiệu đề tài	9
1.1 Sơ lược về thể loại game Battle Royale	9
1.2 Vấn đề của thể loại game Battle Royale	9
1.3 Mục tiêu của Survival2D	10
1.4 Giới hạn	10
2 Thiết kế game	12
2.1 Luật game	12
2.2 Bản đồ	12
2.3 Vùng an toàn (Safe zone)	12
2.4 Vật cản	13
2.5 Người chơi	13
2.6 Vũ khí	14
2.7 Trang bị	14
2.8 Vật phẩm	15
3 Công nghệ sử dụng	16
3.1 Client	16
3.1.1 Cocos2d-x	16
3.2 Server	18
3.2.1 Java	18
3.2.2 Netty	19
3.2.3 WebSocket	19
3.2.4 FlatBuffers	20
4 Phân tích hệ thống	26
4.1 Bản đồ (Map)	26
4.1.1 Map lớn	27
4.1.2 Các đối tượng trên map (map objects)	28
4.1.3 Thiết kế (design) map	28
4.2 Người chơi máy (bot)	29
4.3 Hạn chế hack	29
4.3.1 Hạn chế hack ESP (<i>Extra Sensory Perception</i>)	29
4.3.2 Hạn chế các hack khác	31
5 Thiết kế hệ thống	32
5.1 Thiết kế hệ thống quản lý map	32
5.1.1 Thiết kế map	32
5.1.2 Thiết kế và quản lý các objects	33
5.1.3 Quản lý các object trong map	46
5.1.4 Sinh map ngẫu nhiên	47
5.2 Thiết kế mô hình client-server	49
5.2.1 Thiết kế cơ bản	49

5.2.2	Kỹ thuật dự đoán phía client	52
5.3	Thiết kế người chơi máy (bot)	52
5.3.1	Thuật toán tìm đường đi cho bot	52
5.3.2	Xây dựng cây hành vi (behavior tree) quản lý hành vi cho bot	53
5.3.3	Thiết kế độ khó cho bot	61
6	Kiểm thử	63
6.1	Thuật toán tạo bản đồ	63
6.2	Thuật toán tìm đường A*	66
6.3	Kiểm thử tối ưu tìm kiếm xung quanh một khu vực bằng quadtree	68
6.4	Kiểm thử vùng nhìn của Player	68
6.5	Kiểm thử trận đấu thực tế	69
7	Đánh giá kết quả đạt được	76
8	Định hướng phát triển	77
9	Công cụ sử dụng	78
9.1	Chung	78
9.2	Client	78
9.3	Server	78
Tài liệu tham khảo		79

Danh sách hình ảnh

1	Biểu đồ PUBG	9
2	Vùng an toàn (Safe zone) trong Survival2D	12
3	Vật cản trong Survival2D: cây, đá, thùng, tường	13
4	Người chơi trong Survival2D: tay không và cầm súng	13
5	Các loại súng trong Survival2D. Từ trái qua: súng ngắn, súng săn và súng bắn tỉa	14
6	Nón và giáp trong Survival2D	15
7	Bộ sơ cứu và băng cứu thương trong Survival2D	15
8	So sánh FlatBuffers so với các phương thức khác ¹	21
9	Biểu đồ độ trễ của các gói tin khi so sánh trên localhost	24
10	Biểu đồ độ trễ của các gói tin khi so sánh trên server thật đặt tại Singapore	25
11	Bản đồ tổng thể trong Survival2D [4]	27
12	Các objects trong Survival2D	28
13	Người chơi cheat ESP trong game PUBG Mobile ²	30
14	Mô tả cơ chế chống hack ESP bằng vùng nhìn trong Survival2D	31
16	Diagram của class GameConfig, chứa các config cần thiết trong game	35
17	Diagram của class GameConstant, chứa các thông số sẽ không bao giờ đổi trong game	36
18	Diagram của các class cơ bản	37
19	Diagram của các item	38
20	Diagram của trận đấu	39
22	Diagram của player	40
21	Diagram của các vật cản	41
23	Diagram của quadtree	42
24	Diagram của các enum	43
25	Diagram của các class hỗ trợ	44
26	Diagram của các class vũ khí trong Survival2D	45
27	Mẫu thiết kế Phân vùng không gian sử dụng cấu trúc dữ liệu QuadTree ³	47
28	Mô tả thuật toán tạo bản đồ	48
29	Tổng thể map được tạo ra bởi thuật toán tạo bản đồ	49
30	Minh họa mô hình client-server	51
31	Minh họa bài toán tìm đường[5]	53
32	Minh họa kết quả tìm đường áp dụng A*[5]	53
33	Minh họa các nhiệm vụ đơn giản của behavior tree[8]	54
34	Minh họa một nhiệm vụ phức tạp hơn của behavior tree[8]	55
35	Cách hoạt động của bot sử dụng behavior tree	55
36	Ví dụ về nút Sequence trong Survival2D	56
37	Ví dụ về nút Selector trong Survival2D	56
38	Ví dụ về nút Inverter trong Survival2D	57
39	Ví dụ về nút Succeeder trong Survival2D	57
40	Ví dụ về nút Repeat Until Success trong Survival2D	58
41	Behavior tree rút gọn của Survival2D	58
42	Thiết kế của nhiệm vụ kiểm tra sẵn sàng chiến đấu	59
43	Thiết kế của nhiệm vụ kiểm tra sẵn sàng phản công	59



44	Thiết kế của nhiệm vụ tìm đích	60
45	Thiết kế của nhiệm vụ tấn công	60
46	Thiết kế của nhiệm vụ bảo đảm an toàn	61
47	Thiết kế của nhiệm vụ nhặt vật phẩm	61
48	Kết quả một lần kiểm thử thuật toán tạo tường	64
49	Kết quả một lần kiểm thử thuật toán tạo các vật thể trên bản đồ	65
50	Kết quả một lần kiểm thử thuật toán tìm đường A*	67
51	Kết quả một lần kiểm thử hiệu năng khi tìm kiếm và quản lý các vật thể sử dụng quadtree so với sử dụng danh sách	68
52	Kiểm thử vùng nhìn của người chơi	69
53	Gameplay của Survival2D: Giữ an toàn, tìm kiếm trang bị đầy đủ	69
54	Gameplay của Survival2D: Bắt gặp và giao chiến với đối thủ	70
55	Gameplay của Survival2D: Tiêu diệt thành công đối thủ	70
56	Gameplay của Survival2D: Chiến thắng khi là người sống sót cuối cùng	71
57	Phản hồi: Bao lâu bạn chơi game một lần?	71
58	Phản hồi: Mỗi lần chơi bạn chơi bao lâu?	72
59	Phản hồi: Thể loại game mà bạn chơi	72
60	Phản hồi: Mức độ yêu thích thể loại game bắn súng sinh tồn	73
61	Phản hồi: Thiết bị bạn chơi game	73
62	Phản hồi: Bạn có thích chơi Survival2D không?	74



Danh sách bảng

1	Chi tiết các loại vật cản trong Survival2D	13
2	Thông số các loại vũ khí trong Survival2D	14
3	Chỉ số các loại trang bị trong Survival2D	14
4	Chỉ số các loại vật phẩm trong Survival2D	15
5	So sánh mức độ đáp ứng yêu cầu giữa các game engine	18
6	Bảng đánh giá hiệu năng của FlatBuffers ⁴	22
7	Kích thước của các gói tin quan trọng khi so sánh giữa Json và FlatBuffers	23
8	Độ trễ (ping time) trung bình của các gói tin quan trọng khi so sánh giữa Json và FlatBuffers trên môi trường localhost	24
9	Độ trễ (ping time) trung bình của các gói tin quan trọng khi so sánh giữa Json và FlatBuffers trên server thật đặt tại Singapore	25
10	Phản hồi: Điều gì làm bạn thích Survival2D?	74
11	Phản hồi: Điều gì bạn không thích ở Survival2D?	74
12	Phản hồi: Bạn có góp ý gì cho nhóm để hoàn thiện game hơn không?	75

1 Giới thiệu đề tài

1.1 Sơ lược về thể loại game Battle Royale

Battle Royale là thể loại game bắn súng sinh tồn nhiều người chơi (multiplayer), với yếu tố sống còn được đặt lên hàng đầu. Trong mỗi trận đấu, người chơi phải thu thập vũ khí, trang bị để trở nên mạnh hơn, khi thời gian trôi qua thì khu vực an toàn sẽ thu hẹp lại buộc các người chơi phải chiến đấu với nhau, và người sống sót cuối cùng là người chiến thắng. Bên cạnh tính hành động, chiến thuật, Battle Royale còn đòi hỏi người chơi tinh thần đồng đội khi chơi theo nhóm, hỗ trợ lẫn nhau để giành chiến thắng cuối cùng.

1.2 Vấn đề của thể loại game Battle Royale

Trên thị trường hiện tại đã có một số tựa game Battle Royale nổi tiếng và thành công như PUBG, CS:GO, Valorant, PUBG Mobile, Free Fire,... nhưng tất cả đều có những nhược điểm nhất định. Những tựa game kể trên đều có đồ họa 3D, yêu cầu cấu hình máy trung bình đến cao, khó tiếp cận những người dùng phổ thông. Một vài game khi đã có được lượng người chơi khổng lồ lại không giữ được lâu.

Lấy ví dụ với PUBG, tựa game đã từng rất hot ở thời điểm năm 2018 với hơn 3 triệu người chơi nhưng hiện nay chỉ còn khoảng 1/10 con số này:



Hình 1: Biểu đồ lượng người chơi của PUBG⁵

⁵Nguồn: <https://steamcharts.com/app/578080#All>



Hai trong số nhiều lý do dẫn đến việc PUBG đi xuống đó là nạn hack cheat và gameplay nhảm chán không có nhiều thay đổi đáng kể từ lúc phát hành.⁶ Nạn hack/cheat⁷ là điều khá thường thấy ở các tựa game online nói chung và game bắn súng Battle Royale nói riêng. Nhiều tựa game đã "chết" vì lý do chính là nạn hack quá nhiều. Nạn hack vừa làm mất cân bằng game, gây bất lợi cho người chơi không dùng hack, vừa làm mất uy tín của nhà phát hành.

Một trong những thể loại hack phổ biến của dòng game Battle Royale là cheat ESP. Đây là tình trạng người chơi có thể sử dụng công cụ bên ngoài, truy cập trái phép vào dữ liệu của game; để có thể nhìn xuyên địa hình, vật thể, thấy được người chơi khác nhằm chiếm lợi thế trong game.

1.3 Mục tiêu của Survival2D

Nhận ra thiếu sót của các tựa game Battle Royale hiện có, nhóm mong muốn tạo ra một tựa game mới, thể loại chiến đấu sinh tồn, tên là **Survival2D**. **Survival2D** sẽ khắc phục những điểm chưa tốt của những tựa game đi trước, giải quyết được các vấn đề của dòng game Battle Royale, với các đặc điểm sau:

- *Thể loại Battle Royale multiplayer, đồ họa 2D*: Game nhiều người chơi cùng lúc, đồ họa đơn giản, không yêu cầu cấu hình cao, có thể chơi trên đa nền tảng, dễ tiếp cận với nhiều người chơi ở các tập người chơi khác nhau.
- *Hạn chế hack/cheat ESP*: Như đã nói ở trên, các dòng game Battle Royale thường có hạn chế là bị hack ESP. **Survival2D** sẽ phát triển cơ chế bảo mật để không thể hack cheat ESP, mang lại trải nghiệm tốt hơn cho người chơi.
- *Có người chơi máy*: Có người chơi máy để đấu với người chơi khi số lượng người chơi online thấp, dẫn đến không thể ghép trận với các người chơi khác. Bot đảm bảo có thể thực hiện các hành động một player thông thường có thể làm: di chuyển, nhặt vũ khí, vật phẩm, chiến đấu,... và có thể cạnh tranh chiến thắng với người chơi thật.

1.4 Giới hạn

Trong quá trình phát triển **Survival2D**, nhóm nhận thấy một số chức năng hay, có thể đưa vào mở rộng game để tăng trải nghiệm người chơi nhưng vì giới hạn về nhân lực và thời gian, nhóm chưa hoàn thành được trong đề tài này, như là: chơi theo nhóm, cho âm thanh vào game, phương tiện di chuyển, cho phép người chơi tương tác nhiều hơn với đồng đội (tin nhắn, thoại,...). Thay vào đó, nhóm sẽ ưu tiên phát triển đầy đủ các tính năng cơ bản của một game chiến đấu, và sau đó tập trung vào những vấn đề chính để hoàn thành mục tiêu của đề tài, bao gồm:

- Hiện thực game đồ họa 2D trên nền tảng web (và có khả năng mở rộng lên đa nền tảng), thỏa mãn yêu cầu:
 - Gameplay đầy đủ tính năng của một game Battle Royale

⁷PUBG và những lý do trở thành dead game: <https://www.youtube.com/watch?v=i6KC0xFBn5U>

⁷Những phương thức để giết chết một game online (GameK): <https://gamek.vn/game-online/nhung-phuong-thuc-de-giet-chet-mot-game-online-20130531061110655.chn>



- Đáp ứng được 5-10 người chơi cùng lúc
- Đồng bộ dữ liệu giữa các client giống nhau theo thời gian thực
- Ping trung bình dưới 50ms trong điều kiện mạng phổ thông ở Việt Nam
- Hiện thực phương pháp giải quyết nạn cheat ESP
- Hiện thực người chơi máy áp dụng các giải thuật AI, có đầy đủ các hành vi chơi game giống với người chơi thật.

2 Thiết kế game

Dưới đây là các thiết kế của game **Survival2D**:

2.1 Luật game

Trò chơi sinh tồn: người chơi lập nhóm gồm 1 hoặc nhiều người, sau đó được đưa vào một bản đồ, chiến đấu với nhau đến khi chỉ còn 1 người hoặc 1 nhóm cuối cùng.

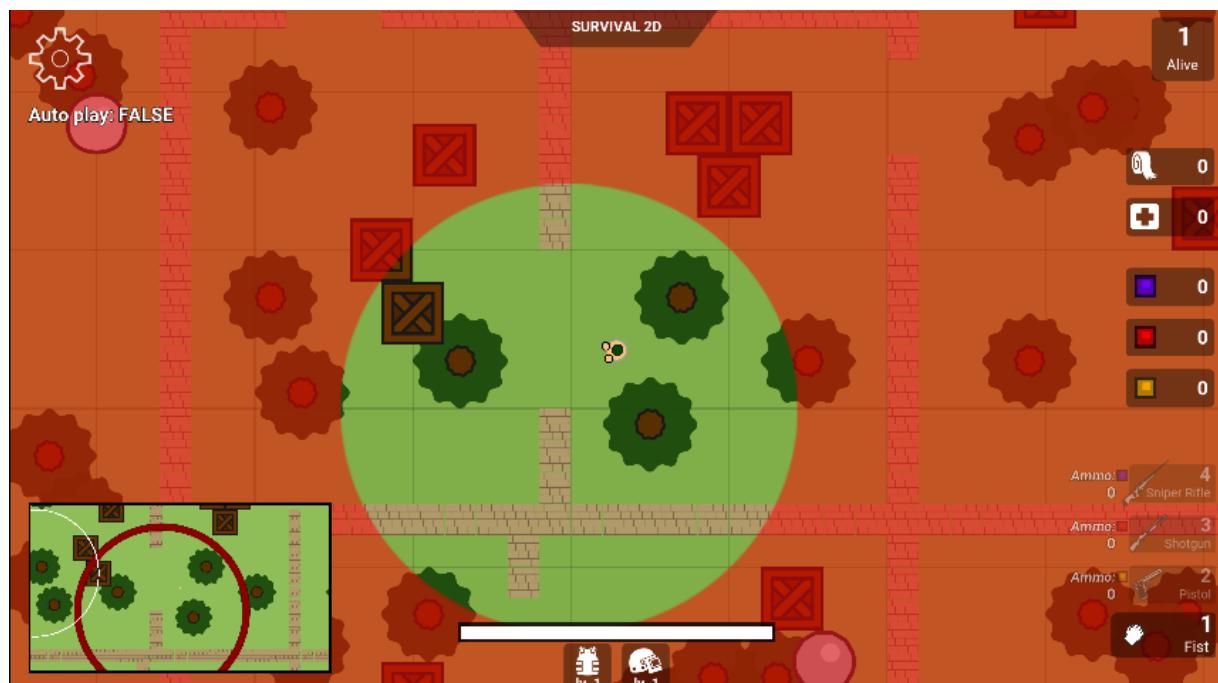
Khi vào một trận đấu, người chơi sẽ xuất hiện ngẫu nhiên trên map, nếu là nhóm thì vị trí các thành viên sẽ gần nhau. Sau đó người chơi tìm nhặt vũ khí, vật phẩm trên map, chiến đấu với nhau để sinh tồn cho đến khi chỉ còn 1 người hoặc 1 nhóm; nhóm cuối cùng này sẽ là nhóm chiến thắng.

Người chơi có một lượng máu nhất định, máu sẽ giảm khi người chơi bị bắn trúng. Có thể dùng vật phẩm hồi máu để tăng máu trở lại. Nếu chơi đơn thì hết máu sẽ chết, nếu chơi nhóm thì hết máu sẽ bị vào trạng thái "knock", trạng thái này người chơi sẽ có "thanh máu knock", thanh máu này giảm dần theo thời gian, hết thanh máu knock sẽ chết hẳn. Trong trạng thái knock, đồng đội có thể cứu, sau đó người chơi sẽ trở lại trạng thái bình thường với rất ít máu.

2.2 Bản đồ

Bản đồ hình chữ nhật, trên bản đồ rải rác các vật cản, vật phẩm. Các vật cản, vật phẩm trên bản đồ và vị trí ban đầu của người chơi được sắp xếp để đảm bảo cân bằng giữa các người chơi.

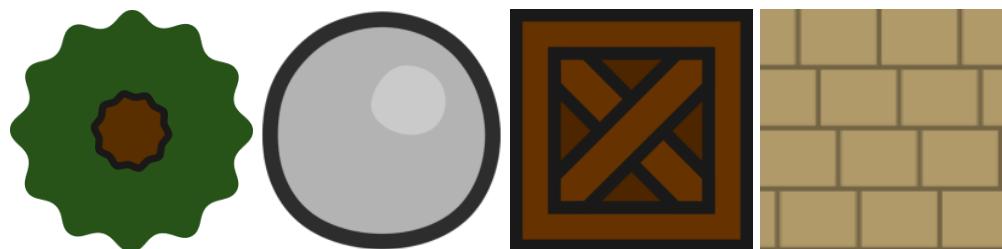
2.3 Vùng an toàn (Safe zone)



Hình 2: Vùng an toàn (Safe zone) trong **Survival2D**

Lấy ý tưởng từ PUBG, **Survival2D** cũng có một vùng bao quanh được gọi là vùng an toàn (safe zone), còn có tên khác là *bo*. Người chơi ở ngoài vòng *bo* này sẽ bị mất máu mỗi giây. Đây là biện pháp để sau khi người chơi đã tiêu diệt bớt đối thủ thì sẽ nhanh chóng tìm được đối thủ khác (do đứng bên ngoài *bo* sẽ bị bắt lợi mất máu, người chơi sẽ cố gắng tập trung lại bên trong vòng *bo* này để được lợi thế), và cũng hạn chế người chơi thích ẩn nấp, rình rập, tránh chiến đấu để sinh tồn. *Bo* sẽ thu hẹp lại sau một thời gian, và đến một lúc sẽ không còn nữa, đồng nghĩa tất cả người chơi đều sẽ bị mất máu đến khi trò chơi kết thúc.

2.4 Vật cản



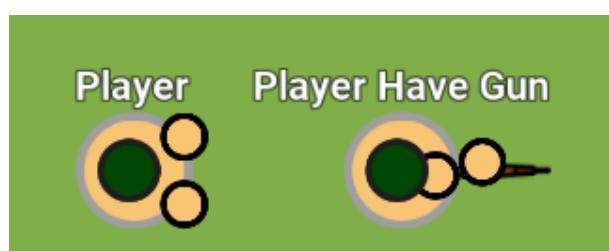
Hình 3: Vật cản trong **Survival2D**: cây, đá, thùng, tường

Survival2D có các loại vật cản sau:

Vật cản	Hình dạng	Kích thước	Đặc tính
Cây	Tròn	Thân cây: 50x50 Tán lá: 150x150	Có thể bị phá huỷ Bao gồm thân cây và tán lá Có thể nấp dưới tán lá
Thùng	Vuông	200x200	Có thể bị phá huỷ Khi bị phá huỷ rơi ra vật phẩm
Đá	Tròn	100x100	Có thể bị phá huỷ Chủ yếu để tránh né đạn của địch
Tường	Vuông	100x100	Không thể bị phá huỷ Liên kết với nhau để phân chia bản đồ Tạo hình dạng giống mê cung

Bảng 1: Chi tiết các loại vật cản trong **Survival2D**

2.5 Người chơi



Hình 4: Người chơi trong **Survival2D**: tay không và cầm súng

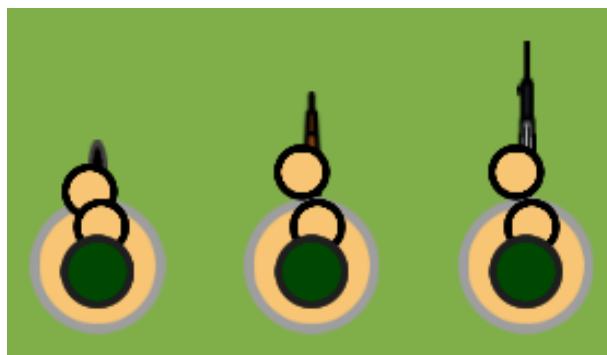
Khi vừa vào trận đấu, người chơi sẽ có sẵn 100 máu. Người chơi được biểu diễn là 2 hình tròn đồng tâm gồm hình tròn ngoài bán kính 30 thể hiện *cơ thể*, hình tròn trong bán kính 10 thể hiện *đầu*, cùng với 2 hình tròn nhỏ ở phía trước thể hiện 2 *tay*. Khi *cơ thể* bị đâm trúng hoặc bị đạn xuyên qua, người chơi sẽ nhận sát thương và mất máu. Sát thương trúng vào *đầu* sẽ tăng lên nhiều lần so với trúng vào *cơ thể*.

2.6 Vũ khí

Survival2D có các loại vũ khí sau:

Vũ khí	Tầm bắn	Sát thương lên cơ thể	Sát thương lên đầu	Sức chứa	Đặc điểm
Tay không	10	5	15		Không cần đạn để tấn công Không cần thay đạn
Súng ngắn	500	10	30	20	Lượng đạn súng ngắn trên bản đồ nhiều
Súng săn	500	20	60	5	Khi bắn tạo ra 5 luồng đạn với độ lệch không xác định
Súng bắn tỉa	1500	30	90	10	Tầm bắn xa, sát thương lớn

Bảng 2: Thông số các loại vũ khí trong **Survival2D**



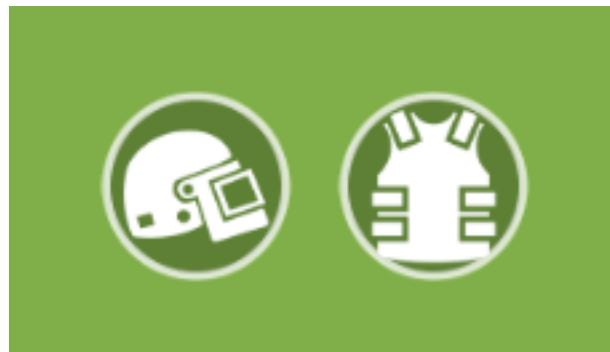
Hình 5: Các loại súng trong **Survival2D**. Từ trái qua: súng ngắn, súng săn và súng bắn tỉa

2.7 Trang bị

Survival2D có các loại trang bị sau:

Trang bị	Tác dụng
Nón	Giảm 30% sát thương vào đầu
Giáp	Giảm 30% sát thương lên cơ thể

Bảng 3: Chỉ số các loại trang bị trong **Survival2D**



Hình 6: Nón và giáp trong Survival2D

2.8 Vật phẩm

Survival2D có các loại vật phẩm sau:

Vật phẩm	Tác dụng
Bộ sơ cứu	Hồi 80% máu đã mất
Băng cứu thương	Hồi 20% máu đã mất

Bảng 4: Chỉ số các loại vật phẩm trong Survival2D



Hình 7: Bộ sơ cứu và băng cứu thương trong Survival2D

3 Công nghệ sử dụng

3.1 Client

3.1.1 Cocos2d-x

Survival2D là game trực tuyến nhiều người chơi, và hướng đến nền tảng trình duyệt; vì vậy client của **Survival2D** yêu cầu một game engine phải gọn nhẹ và có khả năng hỗ trợ đa nền tảng. Cocos2d-x là một engine phát triển game nổi tiếng, ra mắt từ lâu nhưng vẫn đang được sử dụng phổ biến hiện nay, đáp ứng được những yêu cầu trên.

Ưu điểm

- Miễn phí và có mã nguồn mở.
- Gọn nhẹ, dễ dàng cài đặt và sử dụng, có thể chạy mượt mà trên các máy tính với cấu hình không cần quá cao.
- Hỗ trợ nhiều ngôn ngữ lập trình khác nhau, trong đó có C++ để tối ưu hiệu năng, hay JavaScript để lập trình nhanh chóng, dễ dàng.
- Hỗ trợ phát triển chủ yếu game đồ họa 2D, Cocos2d-x vẫn là một engine mạnh mẽ, tốc độ phát triển và dựng game nhanh.
- Hỗ trợ đa nền tảng, đáp ứng mục tiêu của game là tiếp cận được nhiều người chơi, bên cạnh đó cũng mang lại tiện lợi trong quá trình phát triển game, giúp nhóm dễ dàng test game mà không sợ thiếu thiết bị.

Nhược điểm

- Engine cũ, không có nhiều tiện ích như các engine khác.
- Tài liệu khó tìm, ít nhận được sự hỗ trợ từ cộng đồng.
- Còn tồn tại một số lỗi nhưng đã không còn được bảo trì, không được nâng cấp và vá lỗi.

Tuy có vài nhược điểm như đã nêu trên nhưng các nhược điểm này không quá lớn; bên cạnh đó những ưu điểm của Cocos2d-x là rất phù hợp với mục tiêu của đề tài: game 2D, gọn nhẹ, dễ tiếp cận nhiều tập người chơi.

Một số engine khác để so sánh

Unity

- Ưu điểm:
 - Miễn phí.
 - Phổ biến, có sẵn nhiều tài liệu để học và cộng đồng đông đảo dễ tìm kiếm sự hỗ trợ, có nhiều asset và plugin.

- Hỗ trợ đa nền tảng.
- Cung cấp nhiều tính năng có sẵn: editor mạnh mẽ, trực quan, hỗ trợ xây dựng scene,... hỗ trợ yếu tố vật lý, âm thanh,...
- Nhược điểm:
 - Mã nguồn đóng.
 - Có quá nhiều thành phần khiến người mới tiếp cận bị choáng ngợp.
 - Hỗ trợ game 3D nên khá nặng, cần máy tính đủ mạnh để sử dụng, không cần thiết đổi với quy mô của đề tài.
 - Khó kiểm thử do không mở được 2 instance game cùng lúc trong khi đang phát triển.

Unreal Engine

- Ưu điểm:
 - Miễn phí
 - Hiệu năng tuyệt vời, do tận dụng sự tối ưu của C++
 - Mã nguồn có thể truy cập được
- Nhược điểm:
 - Chỉ hỗ trợ C++, khó dùng.
 - Khá nặng, chú trọng đồ họa 3D (thích hợp cho game AAA hơn), không cần thiết đổi với quy mô và mục tiêu của **Survival2D**.
 - Asset store chưa bằng Unity.

Cocos Creator

- Ưu điểm:
 - Là phiên bản engine thế hệ mới của Cocos2d-x.
 - Khắc phục lỗi từ Cocos2d-x, cung cấp thêm những tính năng mới, có editor mạnh, trực quan như Unity.
 - Có tài liệu tham khảo tốt hơn, cộng đồng phát triển hơn.
- Nhược điểm:
 - Tuy mới nhưng chưa hoàn thiện, vẫn còn khó sử dụng, khó tích hợp với các library của NodeJS để mở rộng, chưa đạt được hiệu quả cao.
 - Tương tự Unity, Cocos Creator đã chú trọng hơn về game 3D, không cần thiết đổi với quy mô của đề tài.

Tổng hợp lại từ ưu nhược điểm của các engine phổ biến kể trên, bảng so sánh dưới đây thể hiện mức độ hiệu quả, đáp ứng yêu cầu về 3 tiêu chí khi chọn một engine để phát triển **Survival2D** cơ bản như sau:

	Cocos2d-x	Unity	Unreal Engine	Cocos Creator
Hỗ trợ phát triển tốt game đồ họa 2D	Có	Có	Có	Có
Engine gọn nhẹ	Có	Không	Không	Không
Dễ sử dụng	Có	Có	Không	Có

Bảng 5: So sánh mức độ đáp ứng yêu cầu giữa các game engine

Đa số các engine tuy mạnh và nhiều tiện ích nhưng đều hướng đến phát triển game 3D, game cầu hình cao, rất dư thừa vì **Survival2D** không dùng đến những tiện ích đó, hơn nữa các phần không cần thiết này có thể ảnh hưởng đến hiệu năng của **Survival2D**. Vì vậy Cocos2d-x là sự lựa chọn vừa đủ và phù hợp nhất để nhóm phát triển client của **Survival2D**.

3.2 Server

3.2.1 Java

Tổng quan

Là một game nhiều người chơi, server của **Survival2D** cần hỗ trợ nhiều người chơi cùng lúc, xử lý nhiều trận đấu cùng lúc, đồng thời có thể tối ưu hóa bộ nhớ bởi vì lượng object trong một trận đấu là cực kỳ nhiều. Nhóm lựa chọn Java vì Java có thể thoả mãn những yêu cầu trên.

Ưu điểm

- Ngôn ngữ bậc cao: Java là ngôn ngữ bậc cao, hướng đối tượng dễ tiếp cận.
- Đa nền tảng: Java có thể chạy trên nhiều kiến trúc máy tính và hệ điều hành khác nhau.
- Bảo mật cao: Java có tính bảo mật cao, bao gồm kiểm soát quyền truy cập, phát hiện và ngăn chặn các lỗ hổng bảo mật, và mã hóa dữ liệu để bảo vệ thông tin của người dùng.
- Xử lý đa luồng hiệu quả: Java hỗ trợ xử lý đa luồng, giúp tăng hiệu suất và độ tin cậy của các ứng dụng phát triển trên nền tảng Java.
- Quản lý bộ nhớ tự động: Java có bộ thu gom rác tự động, giúp quản lý bộ nhớ và giải phóng bộ nhớ không còn sử dụng nữa.
- Độ tin cậy cao: Java có tính năng kiểm tra kiểu dữ liệu trực tiếp trong quá trình biên dịch, giúp giảm thiểu khả năng xảy ra lỗi liên quan đến kiểu dữ liệu.
- Kho thư viện phong phú: Java có kho thư viện của bên thứ 3 rất đa dạng, hỗ trợ giải quyết nhiều vấn đề khác nhau mà không phải code lại từ đầu.

Nhược điểm

- Java chạy trên máy ảo Java Virtual Machine (JVM) nên tốc độ sẽ kém hơn khi so sánh với các ngôn ngữ compile ra ngôn ngữ máy như C++.
- Để hiện thực một hàm thực thi một tác vụ giống nhau, code của Java có thể dài hơn khi so sánh với một số ngôn ngữ khác, ví dụ Python.
- Không hỗ trợ tốt phát triển các ứng dụng desktop cần đồ họa.

Những nhược điểm bên trên không ảnh hưởng quá nhiều đến **Survival2D**, nhưng những ưu điểm lại hoàn toàn đáp ứng yêu cầu **Survival2D** đặt ra. Vì vậy nhóm chọn Java làm ngôn ngữ để hiện thực server.

3.2.2 Netty

Lập trình socket là một lĩnh vực khó khăn và phức tạp, nhưng lại rất cần thiết cho các ứng dụng network, đặc biệt là game nhiều người chơi như **Survival2D**. Để lập trình socket, chúng ta cần phải biết nhiều thứ từ mạng máy tính, các giao thức cho đến các kỹ thuật lập trình. Những phần này không phải là mục tiêu đề tài này muôn chung trọng, vì vậy nhóm sử dụng một framework có sẵn hỗ trợ lập trình socket nhanh chóng, dễ dàng, đó là **Netty**.

Netty là một NIO (Non-blocking Input Output) client server framework cho Java, cho phép phát triển ứng dụng mạng nhanh chóng và dễ dàng. Netty rất đơn giản và thuận tiện cho lập trình mạng, hỗ trợ các giao thức kết nối TCP và UDP.

Tính năng của Netty

- Thiết kế:** Netty có thiết kế thông nhất về api, hỗ trợ nhiều kiểu truyền tải, blocking và non-blocking, dựa trên mô hình sự kiện linh hoạt và có thể mở rộng.
- Dễ dàng sử dụng:** Netty có document, hướng dẫn và ví dụ cụ thể; không cần cài thêm thư viện phụ nào.
- Hiệu năng:** Tăng tốc độ lưu thông, giảm độ trễ, tối thiểu hóa tài nguyên sử dụng và hạn chế copy bộ nhớ không cần thiết.
- Bảo mật:** Netty hỗ trợ SSL/TLS và StartTLS.
- Cộng đồng:** Netty có cộng đồng sử dụng to lớn, các bản cập nhật được phát hành thường xuyên.

3.2.3 WebSocket

Survival2D thuộc dạng game trực tuyến nhiều người chơi (Massively Multiplayer Online - MMO), mỗi hành động của người chơi này phải được người chơi khác biết trong thời gian sớm nhất để tạo cảm giác gần như là đồng thời. Vì vậy việc trao đổi thông tin giữa client và server phải đảm bảo nhanh nhất có thể. Với mục tiêu phát triển trên nền tảng trình duyệt, nhóm lựa chọn giao thức phổ biến, dễ sử dụng và được nhiều trình duyệt hỗ trợ, đó là **WebSocket**.



WebSocket là một giao thức truyền tin dựa trên kết nối TCP, cho phép tạo phiên giao tiếp tương tác hai chiều giữa máy khách và máy chủ. Sử dụng **WebSocket**, ta có thể gửi tin nhắn đến máy chủ và nhận các phản hồi theo hướng sự kiện mà không cần phải gửi gói tin thăm dò đến máy chủ để đợi hồi đáp. **WebSocket** hỗ trợ gói tin giao tiếp ở cả dạng text và binary; là một giao thức truyền tin dễ sử dụng, có độ trễ thấp và dễ xử lý lỗi.

Ưu điểm

- Giao thức phổ biến, dễ sử dụng, được hỗ trợ bởi nhiều trình duyệt web
- Hỗ trợ gói tin giao tiếp ở dạng text và binary, thích hợp kết hợp với FlatBuffers
- Độ trễ thấp do không phải gửi gói tin thăm dò để đợi hồi đáp
- Dựa trên kết nối TCP nên việc gửi nhận đảm bảo gói tin không bị thất lạc, dễ xử lý lỗi

Nhược điểm

- Vì dựa trên kết nối TCP nên tốc độ gửi nhận chưa thể bằng UDP

3.2.4 FlatBuffers

Để đảm bảo trải nghiệm game trực tuyến nhiều người chơi được mượt mà, cần phải đảm bảo độ trễ khi gửi/nhận gói tin là thấp nhất có thể. Phương án mà nhóm đưa ra là cố gắng giảm thiểu kích thước gói tin sao cho nhỏ nhất, để thời gian gửi ngắn đi. Và **Survival2D** có sử dụng FlatBuffers để thực hiện điều này.

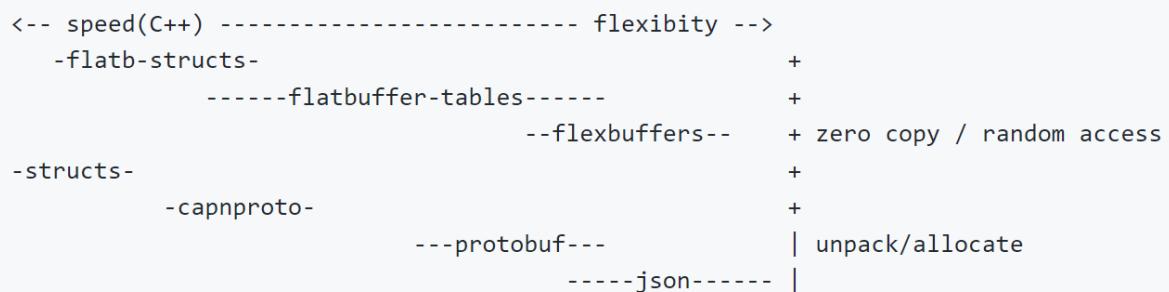
FlatBuffers là một thư viện thực hiện việc serialize và deserialize tương tự như Protocol Buffer (Protobuf) nhưng được đánh giá là tốt hơn.

Ưu điểm

- Nhanh hơn khi so sánh với các phương thức khác.
- Chỉ cần định nghĩa schema, không cần parse, unpack phức tạp như Protobuf.
- Hỗ trợ những trường optional.
- Hỗ trợ nhiều ngôn ngữ lập trình, trong đó có Java và JavaScript - 2 ngôn ngữ đang sử dụng cho server và client của **Survival2D**.

Nhược điểm

- File schema cần được định dạng, compile mới sử dụng được.
- Data dạng binary, không thể đọc hiểu trực tiếp. Dẫn đến khó debug hơn.



Hình 8: So sánh FlatBuffers so với các phương thức khác ⁸

Các bảng so sánh cho thấy FlatBuffers có hiệu năng vượt trội so với các phương thức khác, chỉ kém raw structs; lại linh động, dễ truy xuất, khắc phục được nhược điểm của raw structs.

⁸Nguồn: <https://github.com/google/flatbuffers/wiki/Why-FlatBuffers-vs-other-options%3F>

	FlatBuffers (binary)	Protocol Buffers LITE	Rapid JSON	FlatBuffers (JSON)	pugixml	Raw structs
Decode + Traverse + Dealloc (1 million times, seconds)	0.08	302	583	105	196	0.02
Decode/Traverse/Dealloc (breakdown)	0 / 0.08 / 0	220 / 0.15 / 81	294 / 0.9 / 287	70 / 0.08 / 35	41 / 3.9 / 150	0 / 0.02 / 0
Encode (1 million times, seconds)	3.2	185	650	169	273	0.15
Wire format size (normal / zlib, bytes)	344 / 220	228 / 174	1475 / 322	1029 / 298	1137 / 341	312 / 187
Memory needed to store decoded wire (bytes / blocks)	0 / 0	760 / 20	65689 / 4	328 / 1	34194 / 3	0 / 0
Transient memory allocated during decode (KB)	0	1	131	4	34	0
Generated source code size (KB)	4	61	0	4	0	0
Field access in handwritten traversal code	typed accessors	typed accessors	manual error checking	typed accessors	manual error checking	typed but no safety
Library source code (KB)	15	some subset of 3800	87	43	327	0

Bảng 6: Bảng đánh giá hiệu năng của FlatBuffers⁹

⁹Nguồn: https://google.github.io/flatbuffers/flatbuffers_benchmarks.html
Bảng so sánh FlatBuffers với các phương thức serialization khác: https://en.wikipedia.org/wiki/Comparison_of_data-serialization_for_mats#Overview

Ứng dụng vào Survival2D

Json là phương thức serialize phổ biến, dễ sử dụng khi phát triển các ứng dụng liên quan đến javascript và trình duyệt; vì vậy nhóm tiến hành so sánh FlatBuffers và json khi ứng dụng vào **Survival2D** để cho thấy sự hiệu quả của FlatBuffers:

Các thông số được đo gồm có: kích thước gói tin và thời gian độ trễ khi truyền tải (ping time).

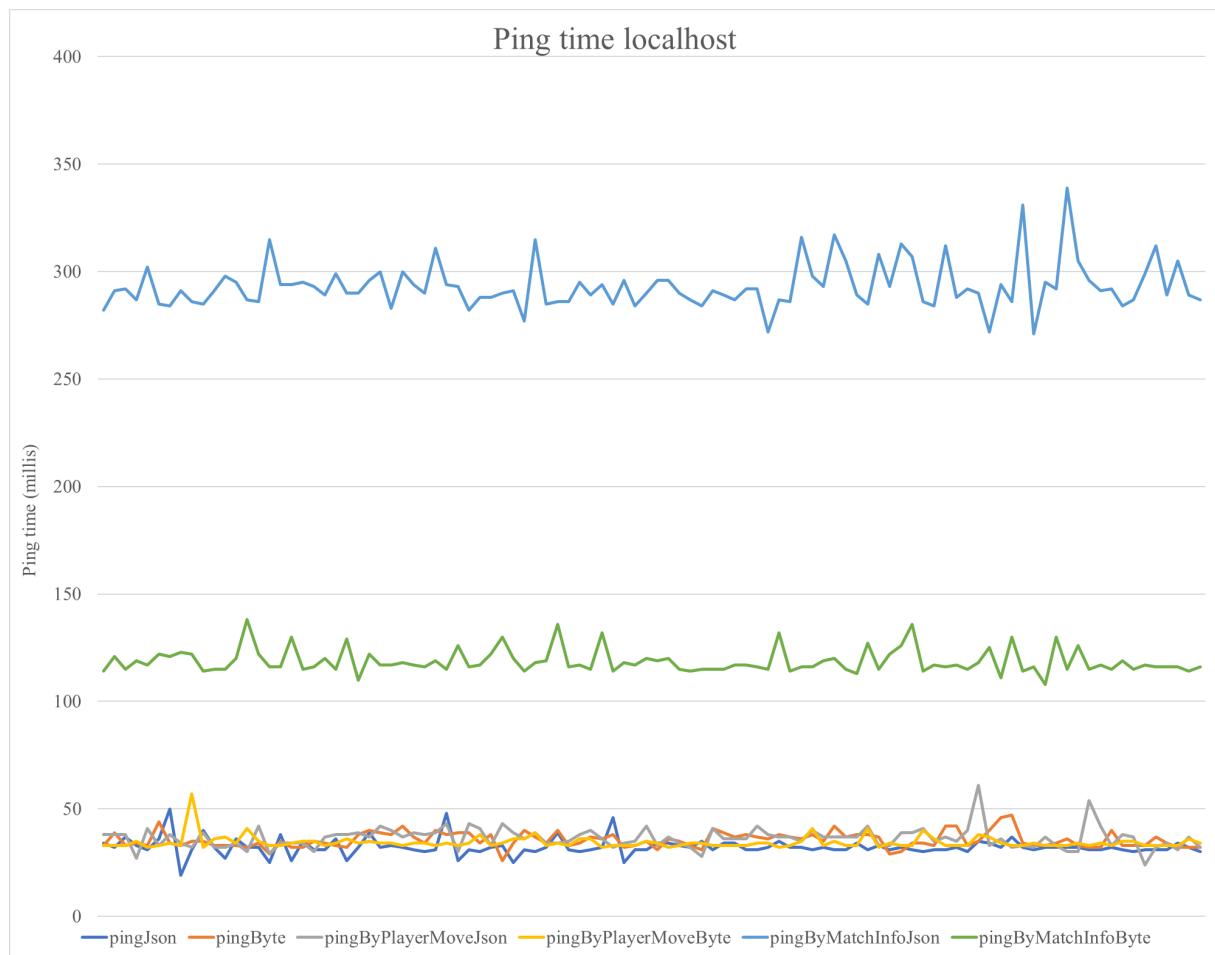
Các gói tin được đo gồm có: Gói ping, gói player di chuyển (gói được gửi/nhận nhiều nhất game) và gói match info (gói tin có kích thước to nhất game, với thông tin của 1000 map objects và 100 players).

Việc đo đạc ping time được thực hiện trên 2 môi trường: môi trường dev localhost và môi trường live (một server thật đặt tại Singapore).

Thông số đo đạc được như sau:

Gói tin	Kích thước (đơn vị byte)		Tỉ lệ kích cỡ của FlatBuffers khi so sánh với Json
	Json	FlatBuffers	
Ping	29	36	124.14%
PlayerMove	115	64	55.65%
MatchInfo	22809	5396	23.66%

Bảng 7: Kích thước của các gói tin quan trọng khi so sánh giữa Json và FlatBuffers
(Kích thước nhỏ hơn là tốt hơn)

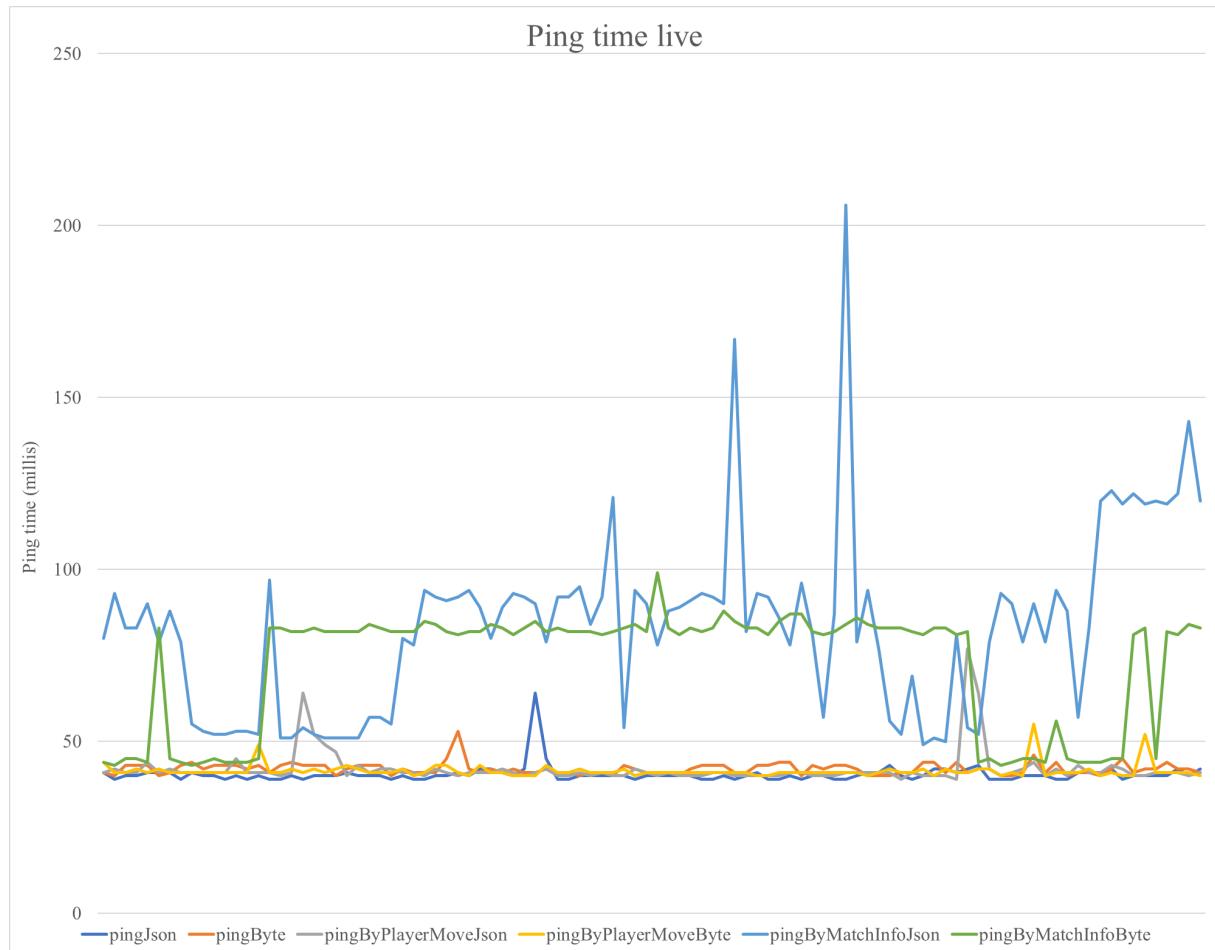


Hình 9: Biểu đồ độ trễ của các gói tin khi so sánh trên localhost

Gói tin	Ping time trung bình (đơn vị millis)		Tỉ lệ ping time của FlatBuffers khi so sánh với Json
	Json	FlatBuffers	
Ping	32.28	35.53	110.07%
PlayerMove	36.35	34.41	94.66%
MatchInfo	293.24	118.7	40.48%

Bảng 8: Độ trễ (ping time) trung bình của các gói tin quan trọng khi so sánh giữa Json và FlatBuffers trên môi trường localhost

(Độ trễ nhỏ hơn là tốt hơn)



Hình 10: Biểu đồ độ trễ của các gói tin khi so sánh trên server thật đặt tại Singapore

Gói tin	Ping time trung bình (đơn vị millis)		Tỉ lệ ping time của FlatBuffers khi so với Json
	Json	FlatBuffers	
Ping	40.4	42.03	104.03%
PlayerMove	42	41.44	98.67%
MatchInfo	83.39	71.93	86.26%

Bảng 9: Độ trễ (ping time) trung bình của các gói tin quan trọng khi so sánh giữa Json và FlatBuffers trên server thật đặt tại Singapore

(Độ trễ nhỏ hơn là tốt hơn)

Các so sánh trên cho thấy được FlatBuffers đã giúp **Survival2D** giảm packet size và ping time đáng kể. Gói tin càng lớn thì độ hiệu quả của FlatBuffers càng cao. Điều này rất thích hợp để phát triển **Survival2D** thêm, vì khi đó số lượng gói tin lắn kích thước gói tin sẽ ngày càng tăng.



4 Phân tích hệ thống

Tổng quan về hệ thống game, muốn phát triển **Survival2D** với những tính năng đáp ứng mục tiêu đề ra, trước hết ta cần phải có một môi trường gameplay cơ bản. Một gameplay chiến đấu sẽ có các phần chính đó là bản đồ (map), người chơi (player) và các đối tượng khác (objects), vật phẩm người chơi như súng, đạn,...

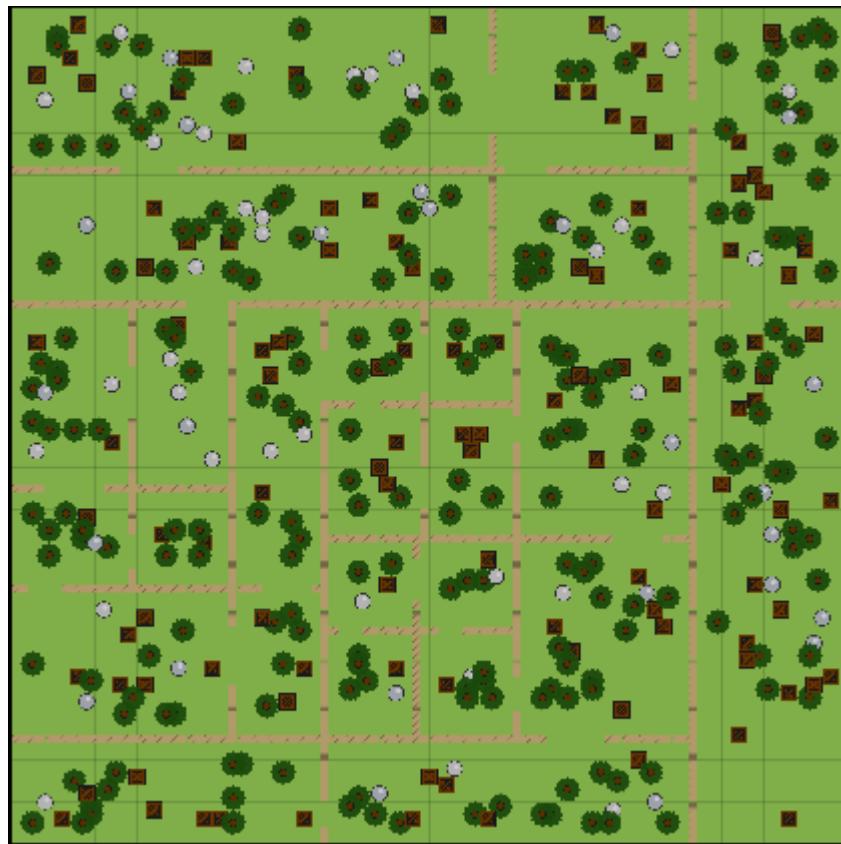
Từ đó, nhóm đưa ra phân tích về hệ thống cần thiết kế của game có những vấn đề quan trọng bao gồm:

- Bản đồ (Map), và các đối tượng trên map: cách tạo ra map và quản lý các đối tượng hiệu quả.
- Giải pháp để hạn chế hack/cheat.
- Hiện thực người chơi máy (bot).

4.1 Bản đồ (Map)

Bản đồ (map) là một mô phỏng của chiến trường, nơi mà các người chơi sẽ đối đầu với nhau trực tiếp ở trên đó. Là game chiến đấu thời gian thực nhiều người chơi nên map là một phần rất quan trọng ảnh hưởng đến chất lượng và sự hấp dẫn của game. Do đó, việc có một hệ thống tạo và quản lý map hiệu quả là rất cần thiết.

Một số vấn đề của khâu thiết kế và quản lý map có thể kể đến như là độ rộng của map, quản lý logic map, hiển thị map, thiết kế vật thể trên map (map object),...



Hình 11: Bản đồ tổng thể trong Survival2D [4]

4.1.1 Map lớn

Để mô phỏng chiến trường 2D mà trong đó có 100 người chơi được phân bố vị trí đảm bảo cách xa nhau, ta cần có một map với diện tích đủ rộng. Ví dụ một người chơi chiếm diện tích $1 m^2$ được quy đổi vào game là 1 cell, người chơi được sinh ra ở vị trí an toàn là trong phạm vi bán kính 100 cells không có địch, ta tạm xem như một vùng an toàn như vậy có diện tích 100×100 cells, 100 vùng như vậy tương đương $100 \times 100 = 1000000$ cells, vậy có nghĩa là map có diện tích khoảng 1000000 cells. Lúc này game sẽ gấp hai vấn đề cần giải quyết:

- Hiển thị map (client): Với máy tính sử dụng 16-bit màu, mỗi pixel sẽ dùng 4 bits để hiển thị màu sắc của chúng, tương tự sẽ là 8 bits đối với 32-bit màu. Ví dụ mỗi cell tương ứng 32×32 pixels, 1000000 cells tương ứng $32 \times 32 \times 1000000 = 1024000000$ pixels, lúc này để hiển thị được toàn bộ map, ta cần khoảng 488 megabytes đối với 16-bit màu và khoảng gần 1 gigabytes đối với 32-bit màu, thực sự lớn.
- Xử lý logic trên map (chủ yếu với server): Nếu xử lý logic trên toàn map, ta phải xử lý cho hàng trăm người chơi và rất nhiều những đối tượng khác trên map như đạn, súng, vật phẩm, vật cản,... cùng một lúc. Ví dụ trong một game có 100 người chơi, tương tác nhặt súng nhặt đạn, sau đó giả sử mỗi người bắn 2-3 viên đạn \rightarrow số lượng object lên đến 150-200 objects (chưa tính vật cản), vậy số phép tính toán là cực lớn.

4.1.2 Các đối tượng trên map (map objects)

Trong game, ngoài người chơi thì ta còn có những objects khác bao gồm đạn, vật phẩm, vật cản như cây, đá, tường,...(hình 12)



Hình 12: Các objects trong Survival2D

Tuỳ vào loại object mà chúng sẽ có những logic cần phải xử lý khác nhau. Để quản lý tốt chúng, ta cũng sẽ phải có một hệ thống quản lý map objects, hệ thống đó cần đáp ứng được những vấn đề sau:

- Nhiều loại object, số lượng objects lớn.
 - Các object trong game sẽ được phân thành nhiều lớp và có những thuộc tính, hành vi nhất định.
 - Một số lớp sẽ có số object cố định, nhưng một số lớp thì không, và thậm chí là số lượng còn là rất lớn, sinh ra, tồn tại, mất đi liên tục và có thể gần như cùng lúc trong một số trường hợp.
- Các objects tương tác với nhau chặt chẽ.

4.1.3 Thiết kế (design) map

Bản thân của map cũng là một object của game, cũng sẽ được thiết kế logic và có hệ thống quản lý như các objects khác. Tuy nhiên vẫn còn vấn đề về design map nữa đó là việc ta phải thiết kế tất cả các thuộc tính của map như là hình dạng, độ rộng, vị trí objects phân bố trên map, thời gian chúng xuất hiện,...

Để game thu hút người chơi, map còn đòi hỏi các vật cản được sắp xếp tự nhiên, hợp lý; các vật phẩm xuất hiện phải cân bằng, không thiên vị cho một player nào. Bên cạnh đó map cũng phải liên thông được, tránh player bị kẹt, không có cách nào để di chuyển đến khu vực khác trên bản đồ.

Với map có diện tích lớn như 10000x10000 pixels của **Survival2D**, việc thiết kế đảm bảo tất cả các yêu cầu trên là một vấn đề khó.

4.2 Người chơi máy (bot)

Người chơi máy (bot) là một phần hầm như không thể thiếu trong khá nhiều thể loại game và đặc biệt là game chiến đấu. Trong nhiều trường hợp, sự thể hiện của người chơi máy có ảnh hưởng rất lớn tới sự thành công của game. Lấy ví dụ, khi game không có nhiều người chơi đang chơi cùng lúc do nhiều lý do khác nhau (game chưa thu hút nhiều người chơi, thời điểm ít người chơi trong ngày,...) thì bot sẽ giúp cho game vẫn giữ được sự nhộn nhịp và đảm bảo cho trận đấu vẫn có thể diễn ra hấp dẫn.

Có nhiều vấn đề về việc thiết kế bot, ví dụ như làm sao để bot di chuyển thông minh, tự nhiên, hành vi giống như người chơi thật trong vài tình huống, bên cạnh đó lại không quá áp đảo người chơi để giữ cho người chơi cảm giác hứng phấn,... Thiết kế bot cũng là một vấn đề khó.

4.3 Hạn chế hack

Một trong những vấn đề lớn của game online là nạn hack/cheat. Hack cheat làm mất cân bằng game, gây bất lợi cho người chơi không dùng hack, từ đó người chơi dễ nản chí và bỏ game; khi đó tổng lượng người chơi giảm, người chơi hack cũng sẽ không gắn bó với game lâu dài, lại làm mất uy tín của nhà phát hành, ảnh hưởng đến các sản phẩm khác. Vì vậy cơ chế bảo mật để chống hack cũng là một vấn đề khó và quan trọng không kém.

4.3.1 Hạn chế hack ESP (*Extra Sensory Perception*)

Hack map, hack ESP, hack xuyên tường,... là các thể loại hack phổ biến trên các game bắn súng. Những tựa game bắn súng nổi tiếng như PUBG, CS:GO, Valorant,... đều bị các vấn nạn hack kể trên.

Nguyên nhân:

Do các thông tin cần biết để biểu diễn, render ở client quá lớn, ví dụ người chơi ở bên kia tường khi di chuyển ta sẽ nghe tiếng chân; người chơi nấp sau tường nhưng vẫn sẽ có đổ bóng; người chơi ở quá xa, khi bắn súng ta vẫn nhìn thấy tia lửa,... Client cần biết thông tin như người chơi đó đang đứng hay ngồi, quay mặt về hướng nào,... để biểu diễn cho chính xác. Đến server phải gửi rất nhiều thông tin người chơi khác (mà về mặt logic người chơi không được biết) để client hiển thị cho đúng. Các modder tận dụng lỗ hổng này khai thác, lấy những thông tin bị che dấu đi này để làm các bản hack cho người chơi biết trạng thái của người chơi khác, gọi là hack ESP.

Ví dụ trong hình 13, người chơi thấy được vị trí đối thủ dù bị khuất tầm nhìn, có được lợi thế khi giao tranh.



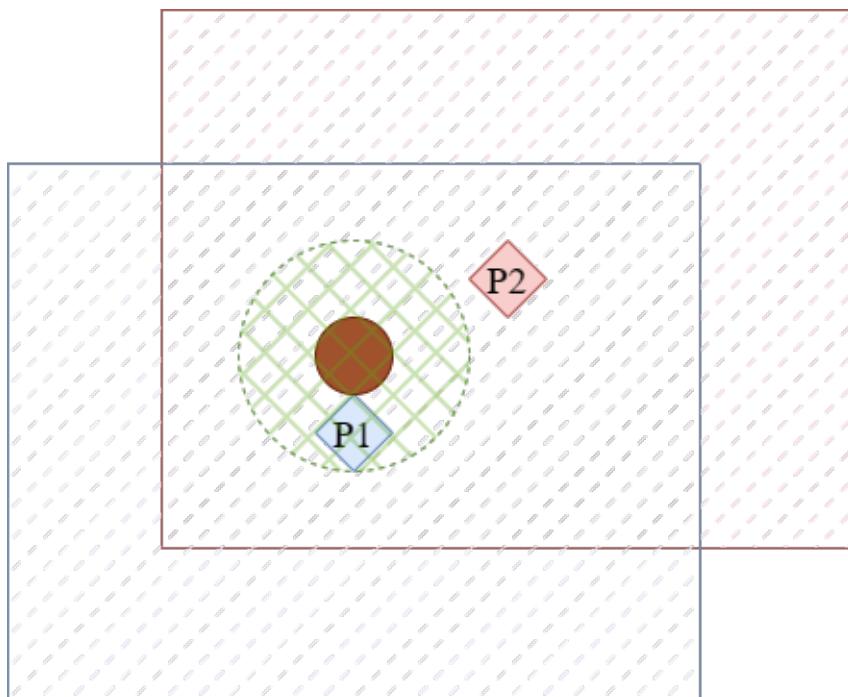
Hình 13: Người chơi cheat ESP trong game PUBG Mobile¹⁰

Cách khắc phục ở Survival2D:

Do Survival2D là game 2D đơn giản, những logic quản lý thông tin về vùng nhìn thấy của người chơi rất dễ thực hiện, nên server sẽ tính toán luôn phần này thay client.

Ví dụ trong hình 14: Người chơi P1 đứng dưới tán cây, có vùng nhìn màu xanh. Người chơi P2 đứng ngoài tán cây, có vùng nhìn màu cam. Vùng nhìn của người chơi P2 bị che mắt đi các vật thể ở bên dưới tán cây. Lúc server gửi thông tin trong vùng nhìn về cho người chơi P2 sẽ lược bỏ các thông tin nằm dưới tán cây, do đó người chơi P2 sẽ không biết gì về người chơi P1. Còn khi server thông tin về cho người chơi P1, thì sẽ gửi luôn thông tin của người chơi P2 do không bị cản trở. Điều này cũng áp dụng cho các công trình gây cản trở tầm nhìn khác ví dụ nhà, hầm,... và các object di chuyển trên map (ví dụ người chơi, đạn,...). Cơ chế này làm cho modder không thể hack ESP được.

¹⁰Nguồn: <https://www.youtube.com/watch?v=v8Cwrn-waf4>



Hình 14: Mô tả cơ chế chống hack ESP bằng vùng nhìn trong Survival2D

4.3.2 Hạn chế các hack khác

Các trường thông tin người chơi như tên, máu, lượng đạn,... của người chơi đều được xác thực thông qua session của người chơi, vì vậy sẽ không thể có việc người chơi B có thể can thiệp vào thông tin của người chơi A.

Các thông tin hiển thị ở client, về máu, súng, lượng đạn, sát thương của súng... của bản thân và đối thủ đều là do server trả về, vì vậy các giá trị trong bộ nhớ của client chỉ mang tính chất dùng để hiển thị trên game. Dù modder có thay đổi giá trị này thì cũng chỉ có thể làm hiển thị ở client của bản thân sai khác đi chứ không ảnh hưởng đến logic thật sự của server gửi cho những client khác.

5 Thiết kế hệ thống

Từ những phân tích ở phần trước, nhóm đưa ra các giải pháp tương ứng cho các vấn đề đặt ra như sau:

- Thiết kế hệ thống quản lý map
 - Thiết kế map
 - Thiết kế objects và hệ thống quản lý objects
 - Sinh map ngẫu nhiên
- Thiết kế mô hình client-server
 - Thiết kế cơ bản
 - Kỹ thuật dự đoán phía client
- Thiết kế người chơi máy (bot)
 - Thuật toán tìm đường đi cho bot
 - Xây dựng cây hành vi (behavior tree) quản lý hành vi cho bot
 - Thiết kế độ khó cho bot

5.1 Thiết kế hệ thống quản lý map

5.1.1 Thiết kế map

Map được thiết kế theo góc nhìn từ trên xuống (top-down), và các objects map cũng sẽ được chiếu lên map theo góc nhìn như vậy.

- Quản lý map và các objects map bằng ma trận hai chiều.

Một ma trận 2 chiều tương ứng với hệ trục tọa độ Oxy , được sử dụng để mô phỏng map và quản lý vị trí cũng như kích thước của toàn bộ các đối tượng có trên map.

- Load map từng phần.

Thay vì load toàn bộ map (objects map) ngay từ đầu, map sẽ được load từng phần theo từng khu vực mà người chơi đang nhìn thấy, điều này sẽ giúp tăng tốc độ tính toán hơn và giảm thời gian chờ khởi tạo trận đấu. Khi người chơi vào trận, ta có thể biết vị trí hiện tại của người chơi và dựa vào đó, phần map đầu tiên sẽ được load, các objects ở quanh khu vực đó được tạo ra. Khi người chơi di chuyển, tùy vào vị trí của người chơi mà các phần khác sẽ được load tiếp theo. Khi load từng phần như vậy, phần map được load là một vùng lớn hơn vùng nhìn tối đa của người chơi, để không gây hiện tượng các objects tạo ra bị chập khi người chơi đã di chuyển đến đó.

- Chia nhỏ map, quản lý theo từng đoạn (chunk).

Tương tự hiển thị map trên client, phần xử lý logic ta cũng có thể chia nhỏ map theo chunks, và chỉ quản lý logic trên từng vùng rồi kết nối chúng lại với nhau. Mỗi người chơi

ở trong chunk sẽ đăng ký theo dõi các sự kiện của chunk đó, khi di chuyển qua khu vực khác thì sẽ huỷ đăng ký với chunk cũ và lại đăng ký vào chunk mới. Với phương án này, khi có các sự kiện diễn ra ở mỗi chunk, ta chỉ cần xử lý sự kiện đó trong chính chunk đó mà không cần duyệt toàn map hay broadcast cho toàn bộ người chơi khác trên map, vừa giảm được khối lượng tính toán mà vừa giúp game có tính bảo mật, chống cheat khiến người chơi không thể biết được những sự kiện xảy ra ở ngoài vùng nhìn thấy của người chơi đó.

5.1.2 Thiết kế và quản lý các objects

Thiết kế class diagram của các objects

Các object trong một trận đấu của Survival2D được thiết kế theo hướng đối tượng (OOP), kết hợp với design patterns thích hợp.

Nhận thấy một số thuộc tính tồn tại ở nhiều class, và các thuộc tính này có tính chất, hành động giống nhau. Nhóm đã định nghĩa ra nhiều interface phổ biến để các class khác kế thừa. Và do Java không cho phép đa kế thừa, nhóm đã hiện thực những method dùng chung này bằng tính năng default method của Java.

Ví dụ: Interface *HasHp* và *Destroyable* không liên quan đến nhau. class *Container* do có thể bị phá huỷ nên sẽ hiện thực interface *Destroyable*; nhưng *Container* khi bị đánh sẽ hao mòn dần chứ không bị phá huỷ ngay, nên sẽ hiện thực interface *HasHp* để thể hiện "độ bền". Khi bị đánh, *Container* sẽ gọi *reduceHp* của *HasHp*, và để hiện thực *isDestroyed* của *Destroyable*, *Container* sẽ gọi *isDestroyed* của *HasHp*, chứ không phải hiện thực lại. Code bên dưới mô tả cho ví dụ này, các class, field và method không liên quan đã được ẩn đi để tránh nhầm lẫn.

```
1  public interface HasHp {  
2  
3      double getHp();  
4  
5      void setHp(double hp);  
6  
7      default void reduceHp(double hp) {  
8          setHp(getHp() - hp);  
9      }  
10  
11     default boolean isAlive() {  
12         return getHp() > 0;  
13     }  
14  
15     default boolean isDestroyed() {  
16         return !isAlive();  
17     }  
18 }  
  
1  public interface Destroyable {  
2      void setDestroyed(boolean destroyed);  
3  
4      boolean isDestroyed();  
5 }
```

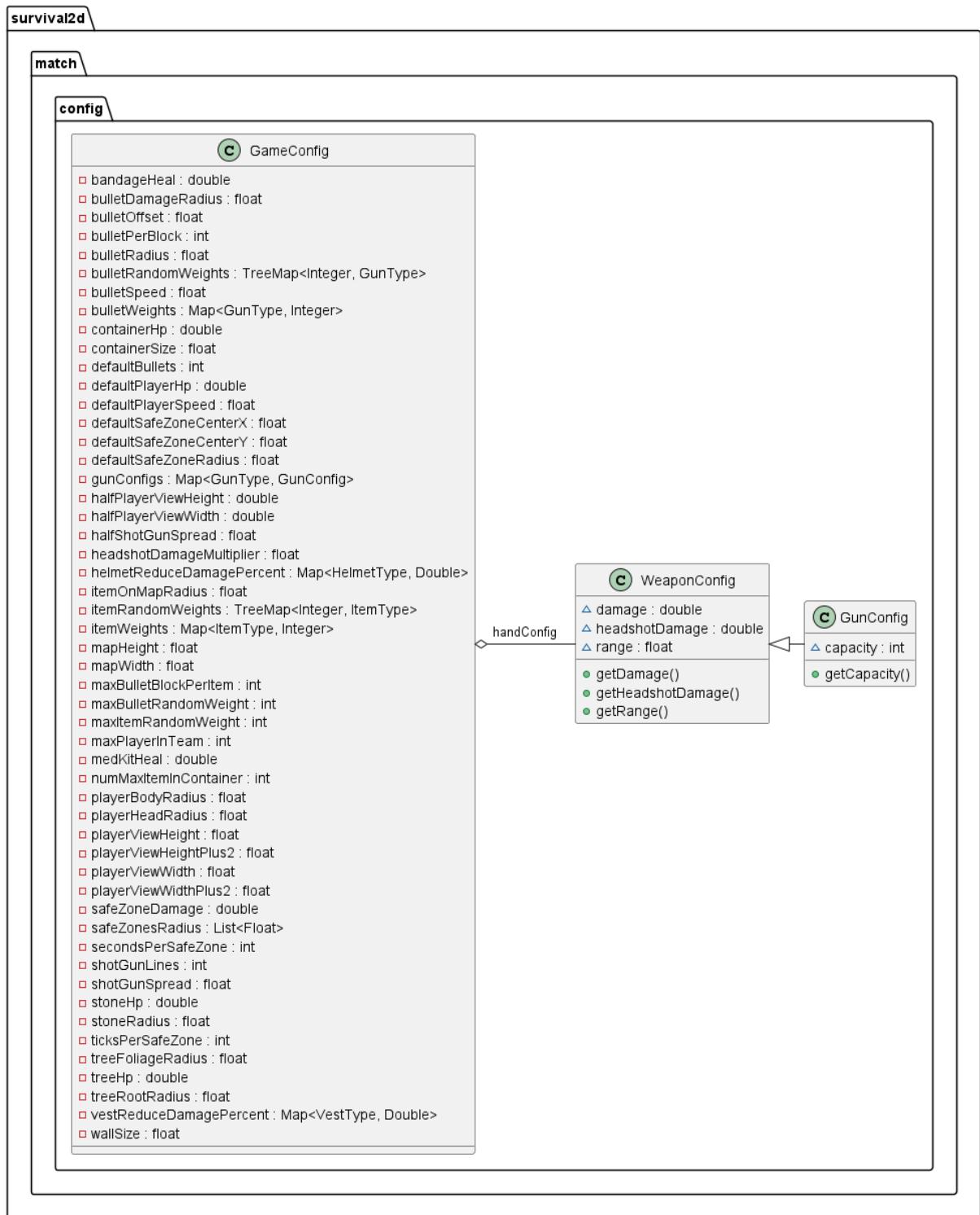


```
6     default void markDestroyed() {
7         setDestroyed(true);
8     }
9 }

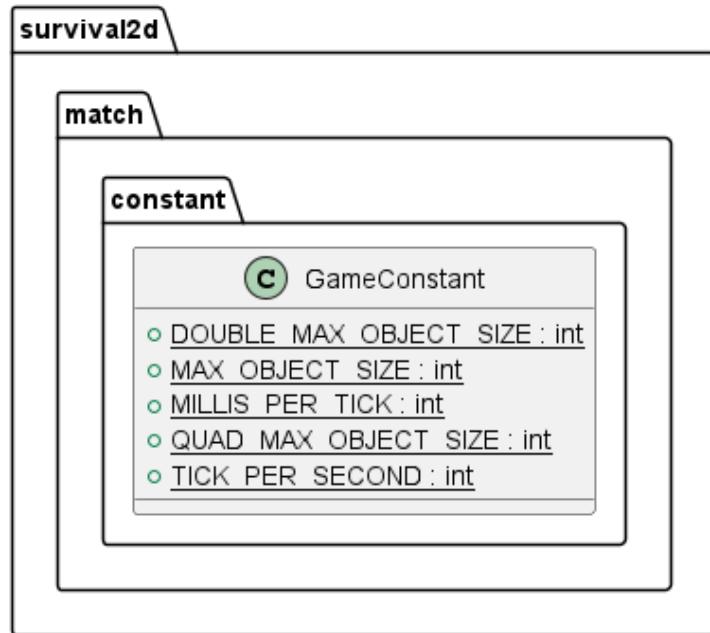
1 //Irrelevant classes, fields and methods have been hidden
2 public class Container implements Destroyable, HasHp {
3     double hp;
4
5     @Override
6     public boolean isDestroyed() {
7         return HasHp.super.isDestroyed();
8     }
9 }
```

Nhờ mô hình trên mà **Survival2D** chỉ hiện thực logic 1 lần ở interface chung, tránh bị trùng lắp code.

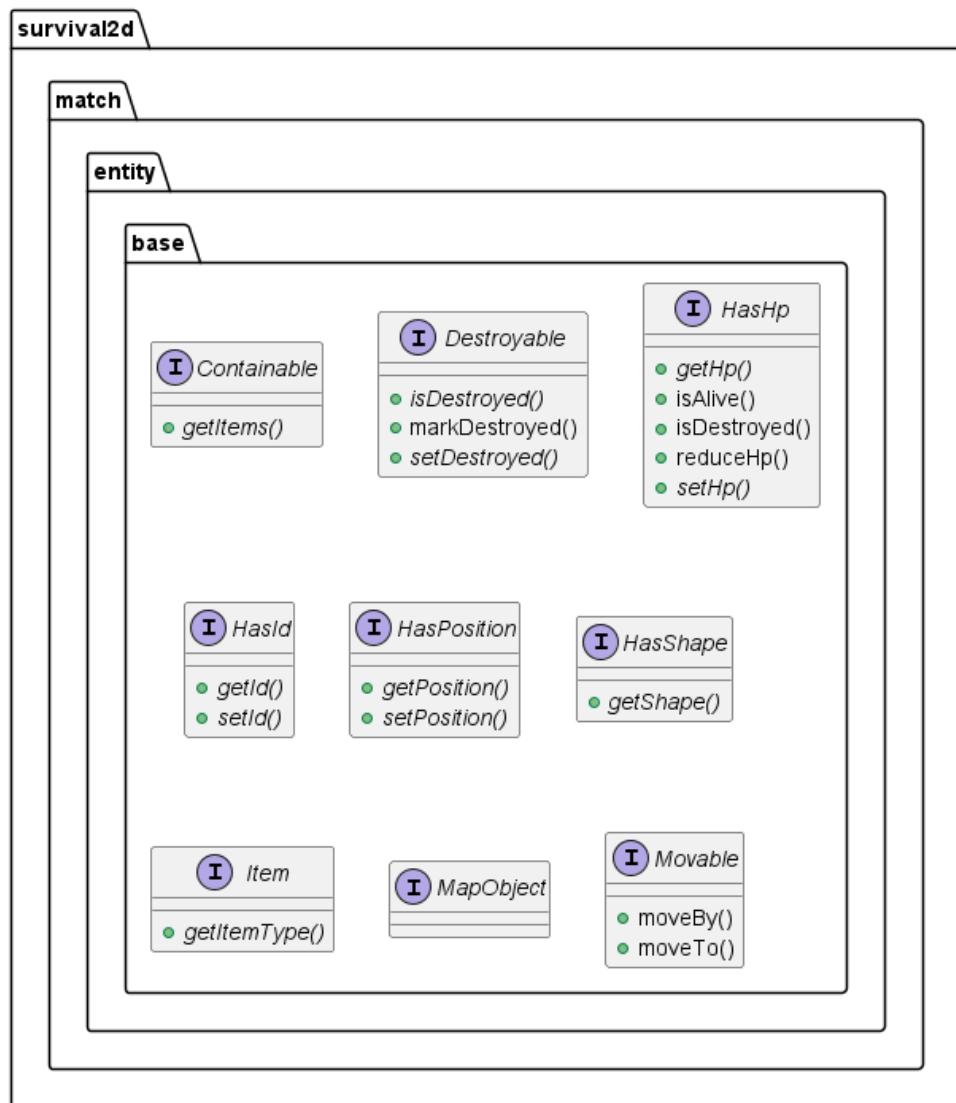
Class diagram cho các lớp (class) quan trọng trong một trận đấu như sau:



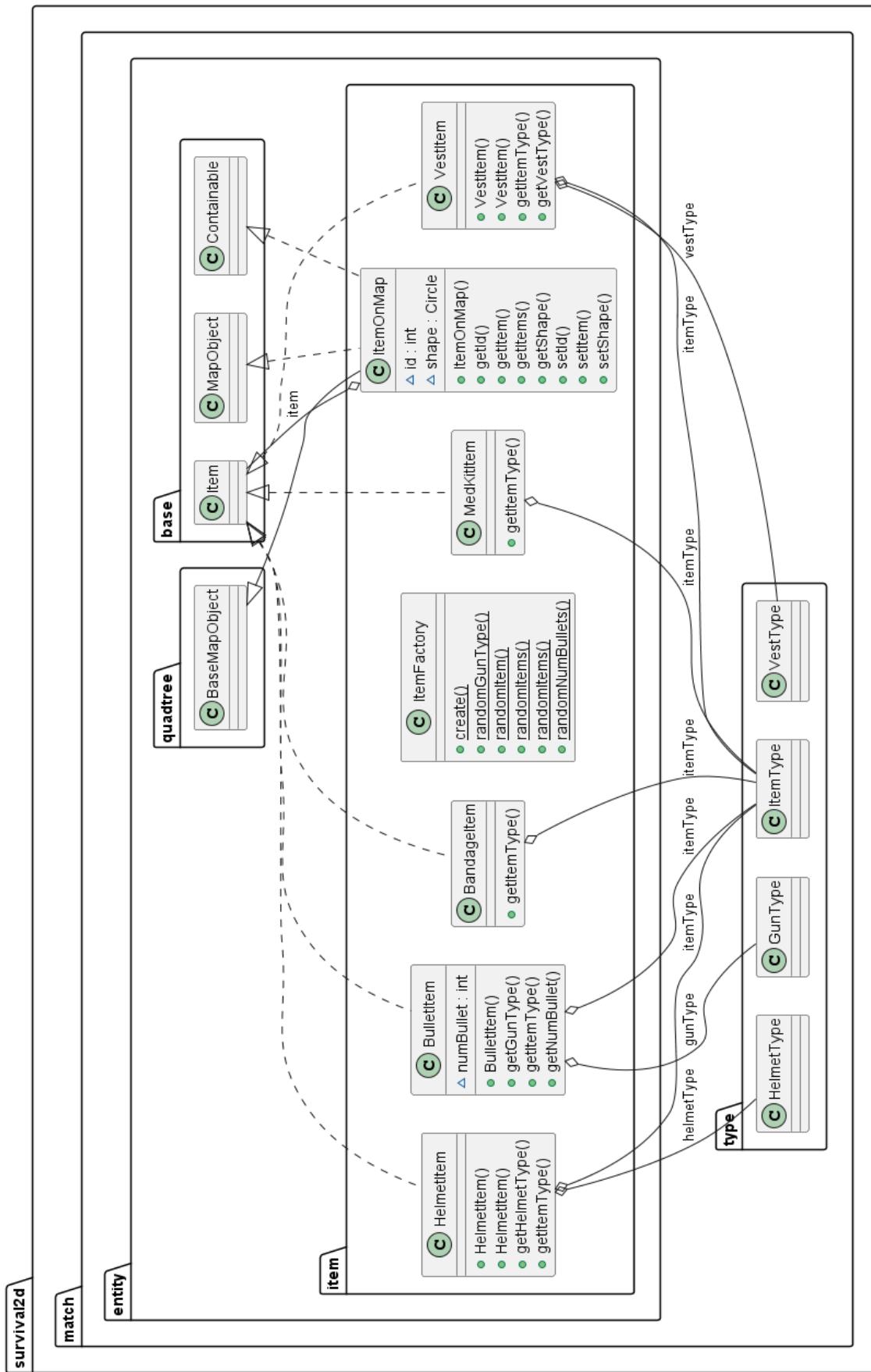
Hình 16: Diagram của class GameConfig, chứa các config cần thiết trong game



Hình 17: Diagram của class GameConstant, chứa các thông số sẽ không bao giờ đổi trong game



Hình 18: Diagram của các class cơ bản
Các interface này sẽ được các class khác kế thừa và tận dụng logic đã hiện thực.

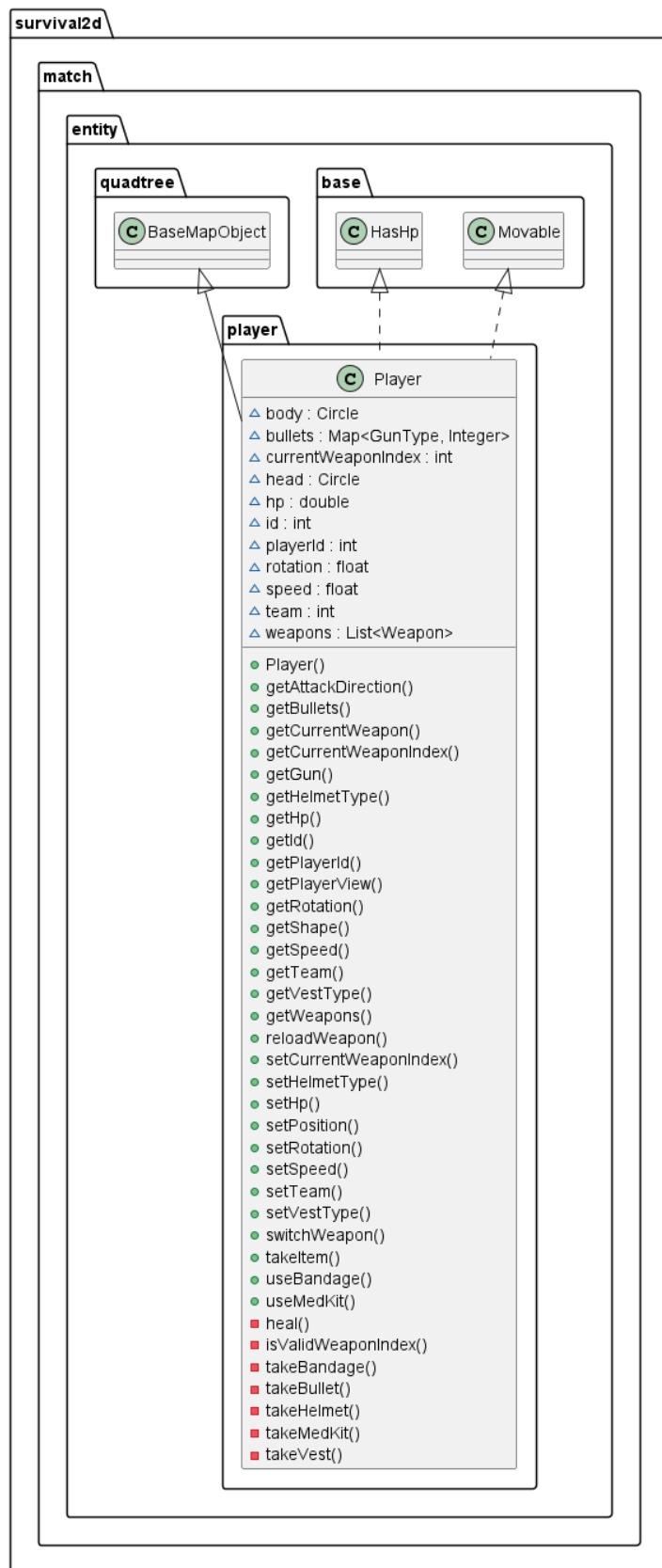


Các item sẽ có method `getItemType` để biết loại item; và có thêm type cụ thể của từng loại item (như `GunType`, `BulletType`...).

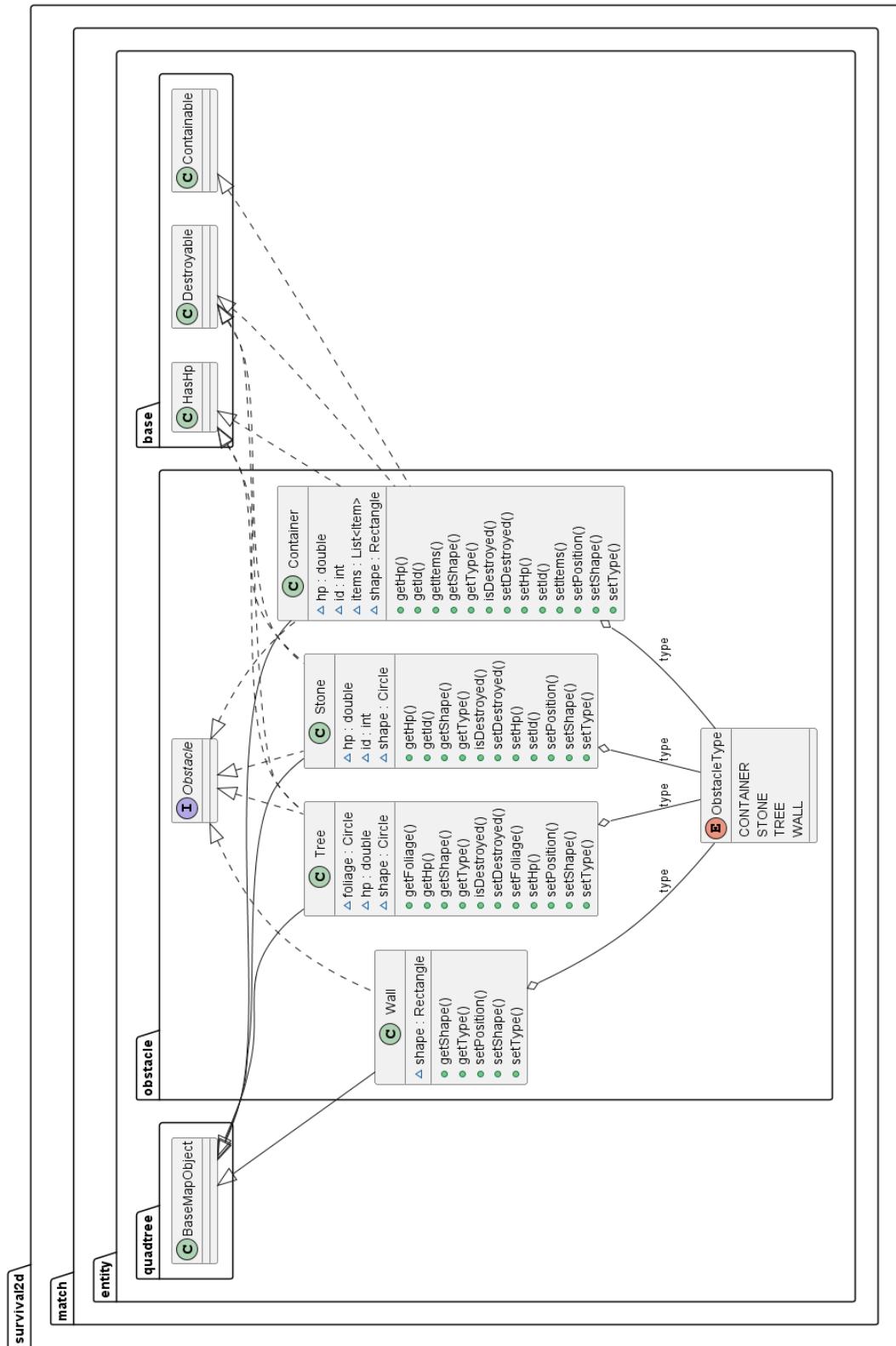


Hình 20: Diagram của trận đấu

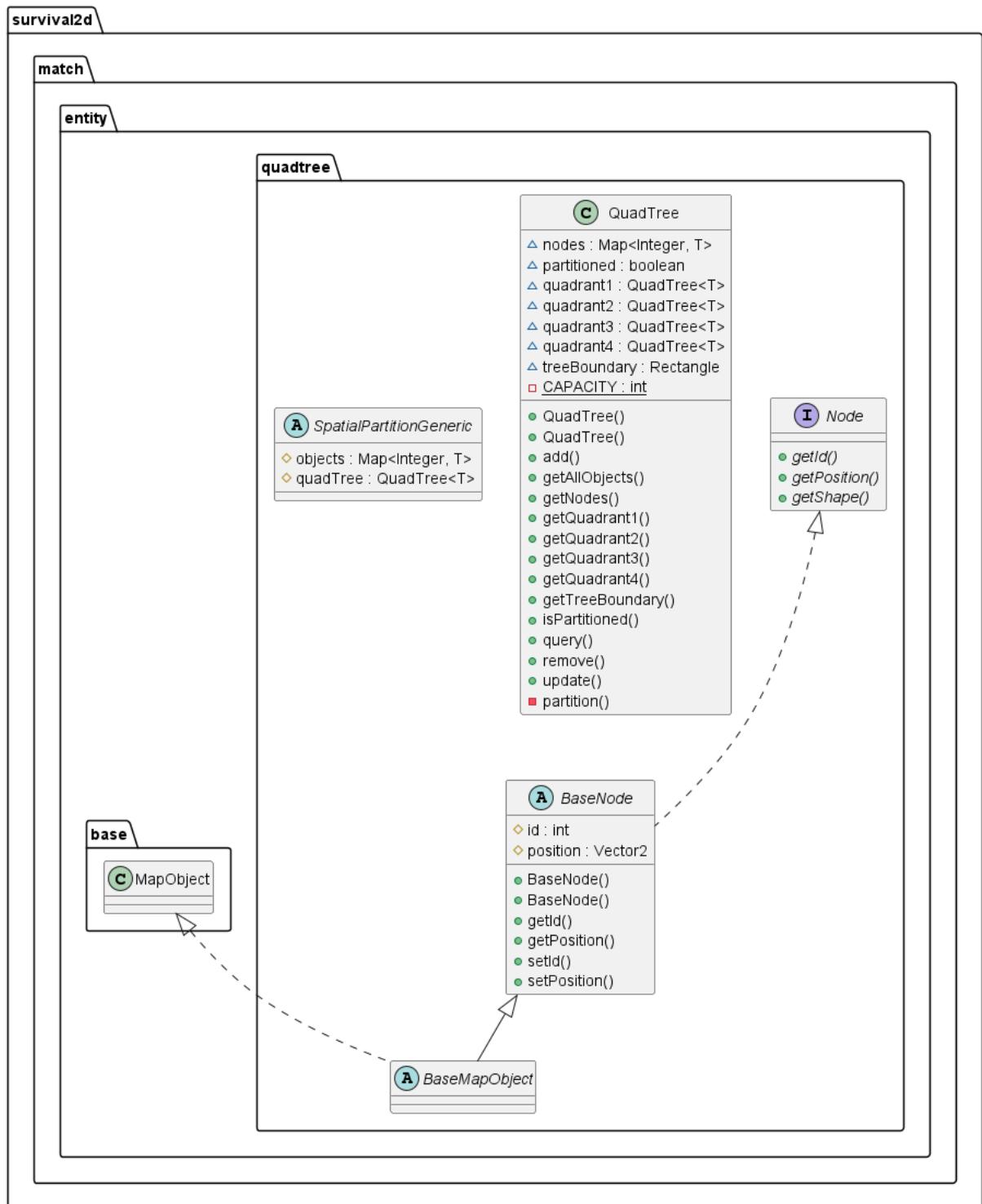
Trận đấu chứa thông tin của toàn bộ objects trong một game, cũng là nơi thực thi logic game.



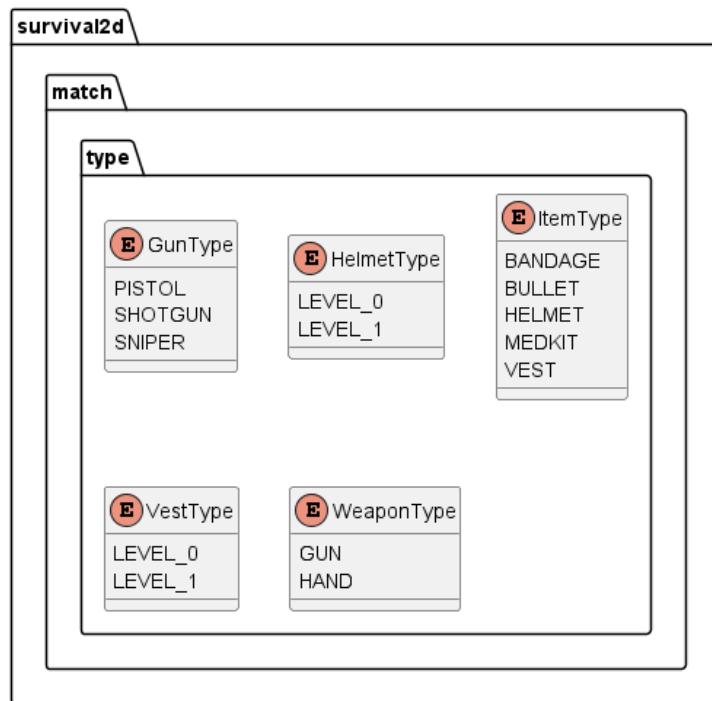
Hình 22: Diagram của player
Class Player chứa nhiều thông tin về vũ khí, trang bị mà người chơi đang nắm giữ.



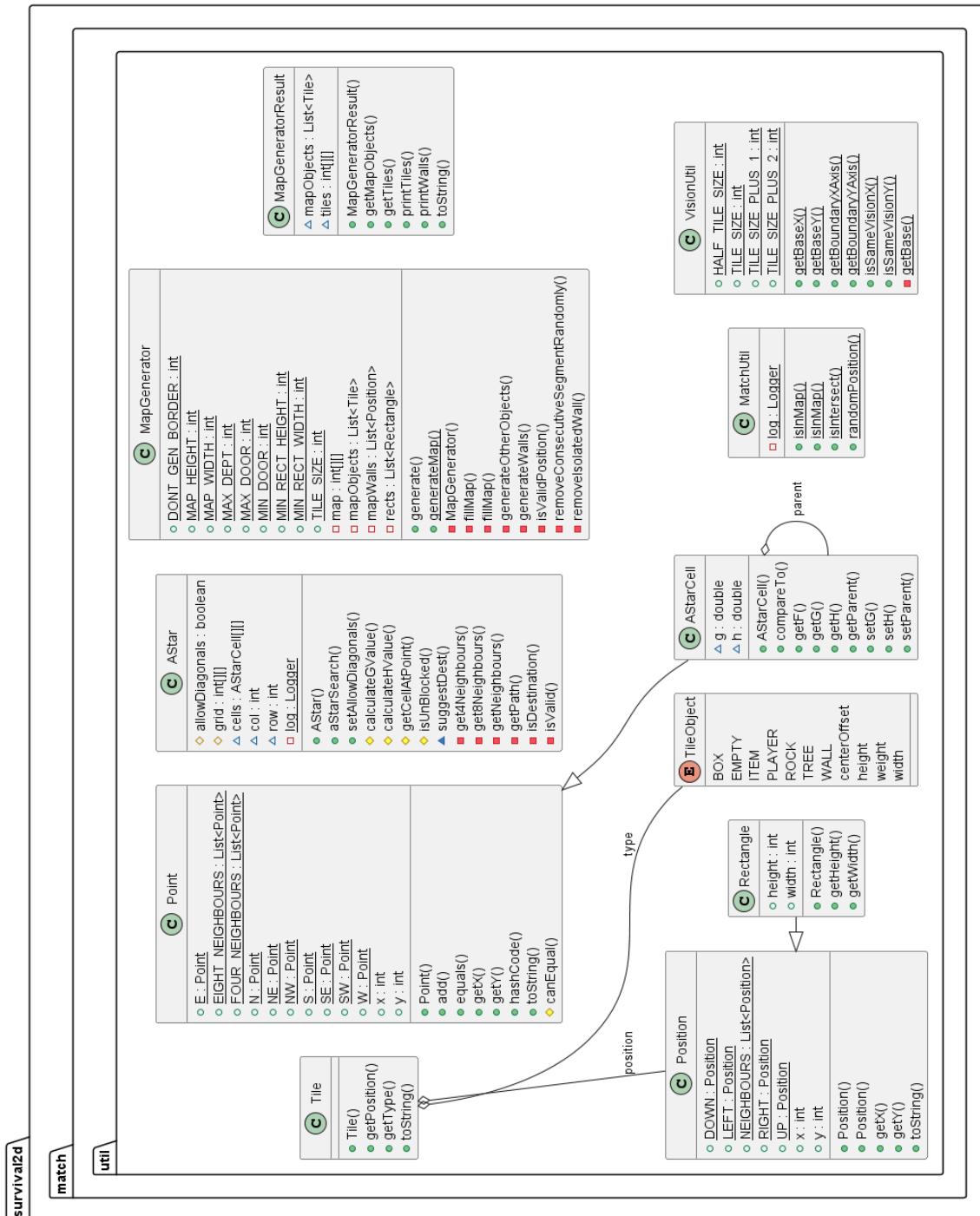
Hình 21: Diagram của các vật cản
Các vật cản của Survival2D tuy ít nhưng mỗi loại vật cản lại mang vai trò riêng biệt.



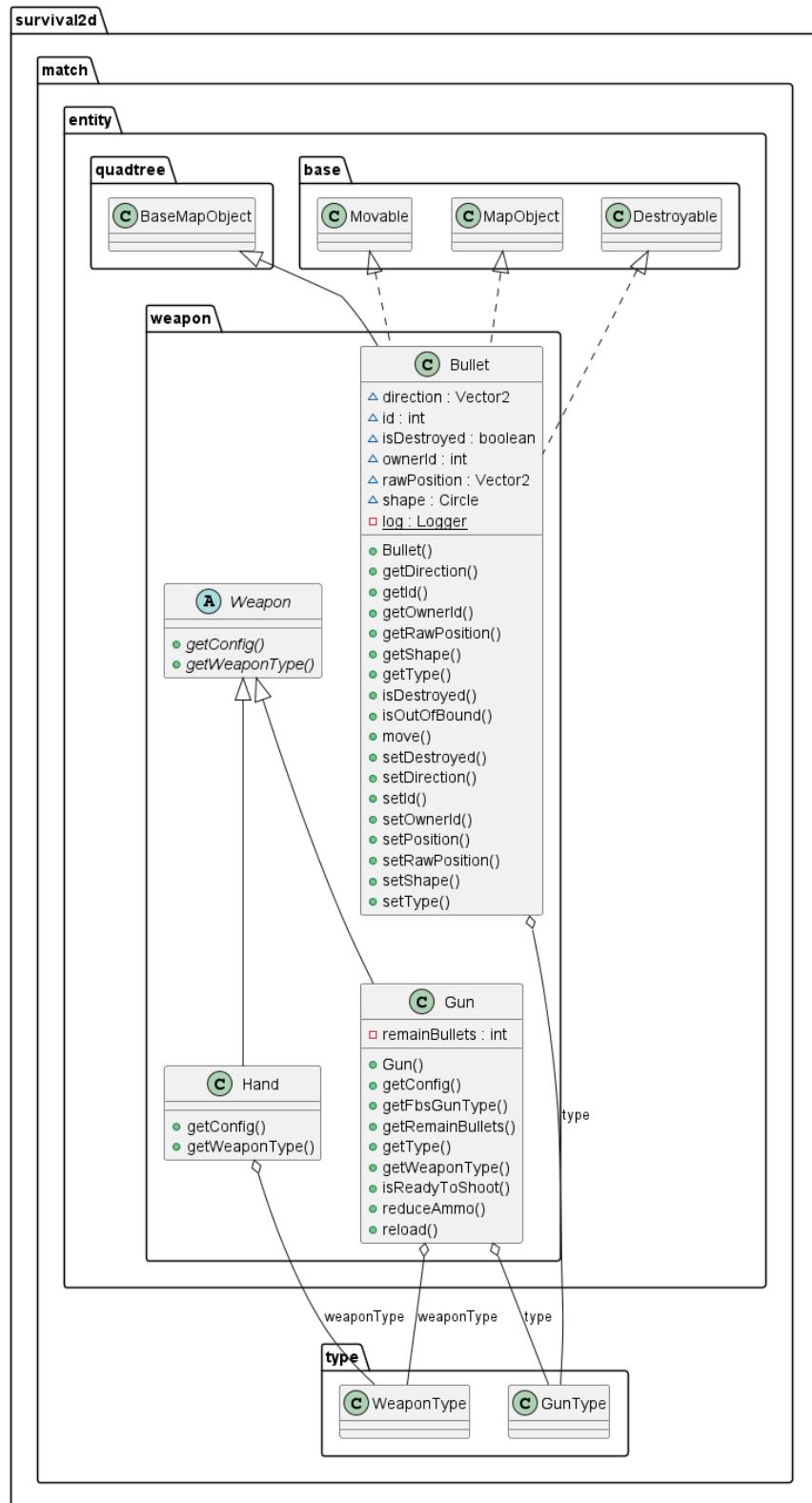
Hình 23: Diagram của quadtree
Các class này dùng để cải thiện truy xuất object trong match.



Hình 24: Diagram của các enum
Các enum này chứa thông số để config cho game.



Hình 25: Diagram của các class hỗ trợ
Các class này thực hiện những việc phổ biến, sử dụng lại nhiều trong game.



Hình 26: Diagram của các class vũ khí trong **Survival2D**
Các class này cho biết trạng thái của vũ khí và cho phép lấy chỉ số của vũ khí trong game dễ dàng hơn.

5.1.3 Quản lý các object trong map

Vấn đề

Trong một map lớn, rất nhiều objects, và thời gian cập nhật của mỗi vòng lặp logic cũng rất nhanh, nên khi các object tương tác với nhau, nếu phải duyệt qua toàn bộ các objects khác thì chi phí vòng lặp tăng lên nhiều lần. Ví dụ để khi đạn di chuyển, nếu ta phải duyệt qua toàn bộ map objects và players để kiểm tra va chạm và tính toán sát thương thì sẽ tốn chi phí n^2 , với lượng map objects lên đến hàng nghìn của **Survival2D**, và không chỉ mỗi việc kiểm tra va chạm của đạn, còn có va chạm của player để cản player di chuyển,... thì đây là một chi phí lớn. Vì vậy cần có giải pháp để có thể tìm ra các vật thể xung quanh một object nhanh hơn, đảm bảo vòng lặp xử lý logic của game xử lý nhanh, đáp ứng được yêu cầu realtime.

Và đó lí do nhóm áp dụng mẫu thiết kế *Spatial Partition* (tạm dịch "*Phân vùng không gian*") để quản lý các object.

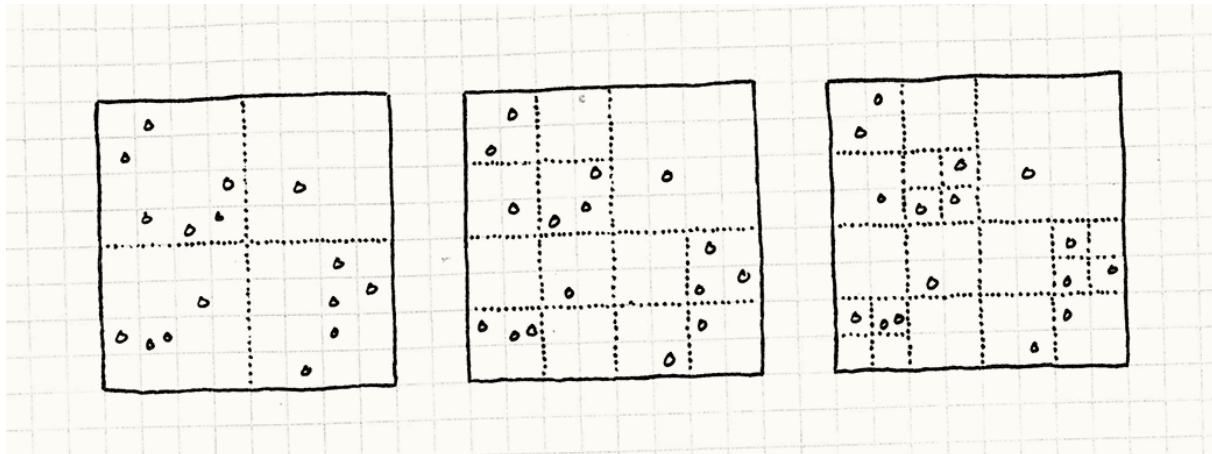
Giải pháp

Ý tưởng của mẫu thiết kế *Phân vùng không gian* cũng giống với Binary search trên mảng 1 chiều. Đơn giản hóa bài toán "Tìm các vật thể xung quanh bán kính x của một vị trí cho trước trên một bản đồ 2D" thành "Tìm các số có khoảng cách so với một số cho trước $\leq x$ trong một mảng 1 chiều". Để giải quyết bài toán này, nếu đảm bảo các phần tử trong mảng đã được sắp xếp, ta có thể dùng Binary search để tìm các số thích hợp, thay vì phải duyệt qua toàn bộ mảng.

Mẫu thiết kế Phân vùng không gian cũng giống vậy, nhưng mở rộng ra cho không gian 2 chiều. Nó cho phép xác định nhanh các đối tượng xung quanh một vị trí cho trước; bằng cách sử dụng cấu trúc dữ liệu được sắp xếp theo vị trí 2 chiều của chúng. Nhờ vậy khi thực hiện một thao tác như kiểm tra va chạm ở trên, không cần phải kiểm tra vị trí của *mọi* đối tượng so với vị trí của đối tượng cần xét va chạm.

Cấu trúc dữ liệu có thể được sử dụng để lưu trữ các đối tượng chuyển động và hoặc đứng yên; với các đối tượng chuyển động, chúng sẽ được gỡ ra và chèn lại mỗi khi vị trí của chúng thay đổi. Trong **Survival2D**, nhóm sử dụng Quadtree, cấu trúc này sẽ giảm độ phức tạp về thời gian của việc tìm kiếm các đối tượng trong một phạm vi nhất định từ $O(n^2)$ còn $O(n * \log(n))$, giảm đáng kể các tính toán cần thiết trong trường hợp có số lượng lớn đối tượng.

Hình 27 biểu diễn mẫu thiết kế Phân vùng không gian sử dụng cấu trúc dữ liệu QuadTree, map sẽ được chia thành nhiều vùng nhỏ cho đến khi mỗi vùng chỉ chứa tối đa 2 objects.



Hình 27: Mẫu thiết kế Phân vùng không gian sử dụng cấu trúc dữ liệu QuadTree¹¹

5.1.4 Sinh map ngẫu nhiên

Để có được một cấu trúc map (map config) phù hợp, thiết kế thủ công sẽ giúp ta tinh chỉnh chính xác được những gì ta muốn trên một map để map cân bằng và thú vị. Tuy nhiên, việc này sẽ tốn rất nhiều thời gian, nên sẽ khó tạo ra được nhiều map. Nếu game chỉ có một hoặc hai map lặp đi lặp lại thì người chơi sẽ cảm thấy nhảm chán, do đó một thuật toán sinh map ngẫu nhiên sẽ được xây dựng trong game để đáp ứng được sự đa dạng về trải nghiệm của người chơi.

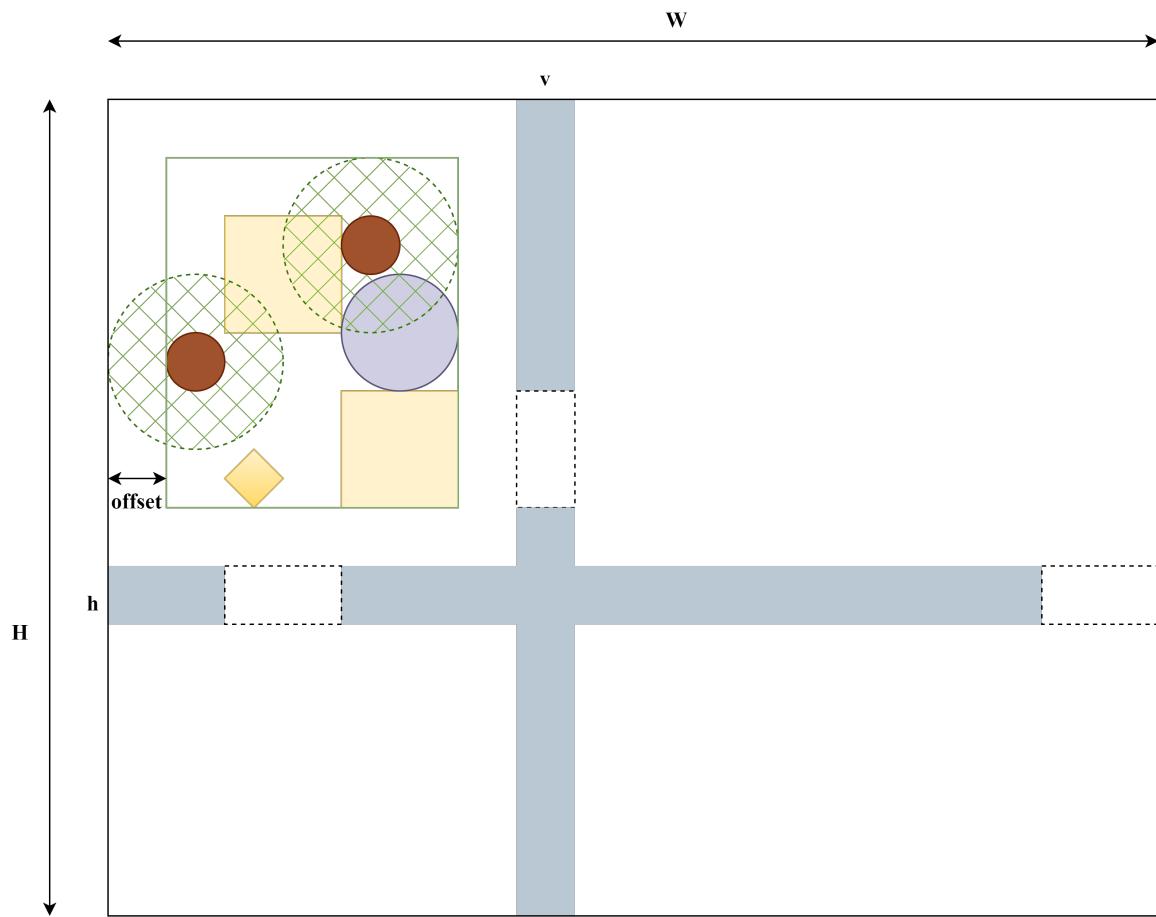
Thuật toán sẽ sinh ra một config chứa thông tin của map gồm kích thước, hình dạng (dựa theo từng tile của tilemaps), vị trí các objects trên map,... sau đó game sẽ đọc config, tạo logic map và render map.

Thuật toán tạo bản đồ

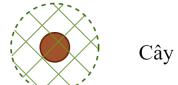
Chia map 10000x10000 pixels ban đầu thành nhiều ô, mỗi ô có kích thước 100x100 pixels. Map ban đầu sẽ trở thành một map 100x100 ô, tổng quát là một hình chữ nhật có chiều dài W , chiều rộng H . Để đảm bảo những "khu vực" được ngăn cách bởi tường đủ lớn, ta sẽ gọi khoảng chiều dài, chiều rộng nhỏ nhất này là $minSpace$. Sau đó chọn một giá trị v sao cho $minSpace \leq v \leq W$ và một giá trị h sao cho $minSpace \leq h \leq H$. Sau đó thêm một hàng tường có $x = v$ hoặc $y = h$. Ta được 4 đoạn tường nhỏ chia cắt map. Để đảm bảo map liên thông với nhau, ta sẽ "đục" 3 trên 4 tường này để tạo lối đi. Ngoài ra để đảm bảo dễ dàng di chuyển giữa các vùng, ta cũng tiến hành đục nếu tường quá dài (có độ dài lớn hơn L cho trước). Và đệ quy hàm này cho 4 hình chữ nhật mới (sau khi bị chia cắt bởi 2 đường kẻ); cho đến khi $W, H < minSpace/2$ (để đảm bảo khu vực mới tạo ra không quá nhỏ như đã nói ở trên).

Sau đó ta sẽ tạo ra những vật cản khác trên map cho từng "khu vực". Để đảm bảo vật cản mới sinh ra không bị kín lối đi, ta sẽ gen vật cản cách tường một khoảng là $offset$. Các vật cản này được tạo ra theo tỉ lệ trong config cho trước. Với mỗi "khu vực" ta cũng chọn ra một ô trống để đánh dấu có thể cho người chơi xuất hiện tại ô này; để sau khi gen map, ta có một list những vị trí có thể cho người chơi xuất hiện. Bằng cách này, có thể đảm bảo mỗi player ban đầu sẽ có được một khu vực an toàn, nhưng vẫn có vật phẩm, đảm bảo cân bằng game. Hình 28 mô tả cho thuật toán này.

¹¹Nguồn: <https://gameprogrammingpatterns.com/spatial-partition.html>



Chú thích



Cây



Thùng



Tường



Đá



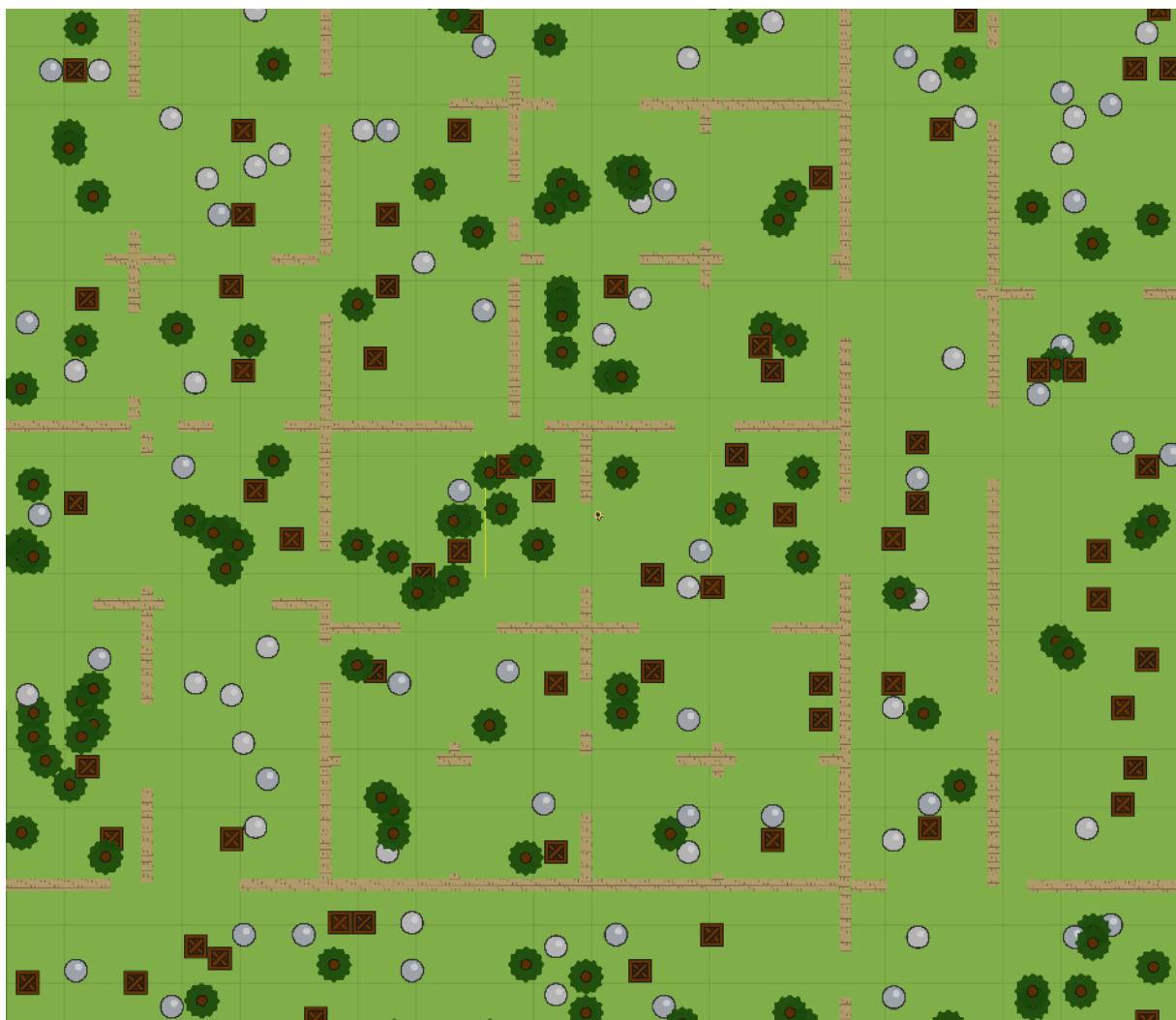
Vị trí có thể cho
người chơi xuất hiện



Mảng tường sau khi
loại bỏ để tạo lối đi

Hình 28: Mô tả thuật toán tạo bản đồ

Sử dụng thuật toán này cho map ban đầu, ta có thể đạt được kết quả là map như trong hình 29.



Hình 29: Tổng thể map được tạo ra bởi thuật toán tạo bản đồ

Chú thích

- Các đường màu vàng cam, liên kết với nhau là tường.
- Các hình tròn màu xanh lá có viền răng cưa, bao quanh một hình tròn màu nâu là cây (phần màu nâu là gốc cây, phần màu xanh là tán cây).
- Các hình tròn màu xám là đá.
- Các hình vuông có dấu chéo là các thùng gỗ có chứa item.

5.2 Thiết kế mô hình client-server

5.2.1 Thiết kế cơ bản

Hầu hết các trò chơi trực tuyến ngày nay đều được thiết kế theo mô hình client-server. Trong thiết kế này, có một server duy nhất chịu trách nhiệm xử lý các logic chính của game. Server này



sau đó được kết nối đến một hoặc nhiều client. Các client này có nhiệm vụ thu thập các input từ người chơi, sau đó gửi đến server để xử lý, server sử dụng các input đó để cập nhật lại các thông số của các vật thể trong trò chơi (như vị trí, trạng thái, ...) rồi gửi lại các cập nhật đó cho client hiển thị. Mỗi client và server đều chạy các vòng game loop của riêng mình và xử lý các công việc khác nhau trong vòng lặp đó.

Đối với client, vòng lặp sẽ được chạy mỗi frame và làm các công việc như sau:

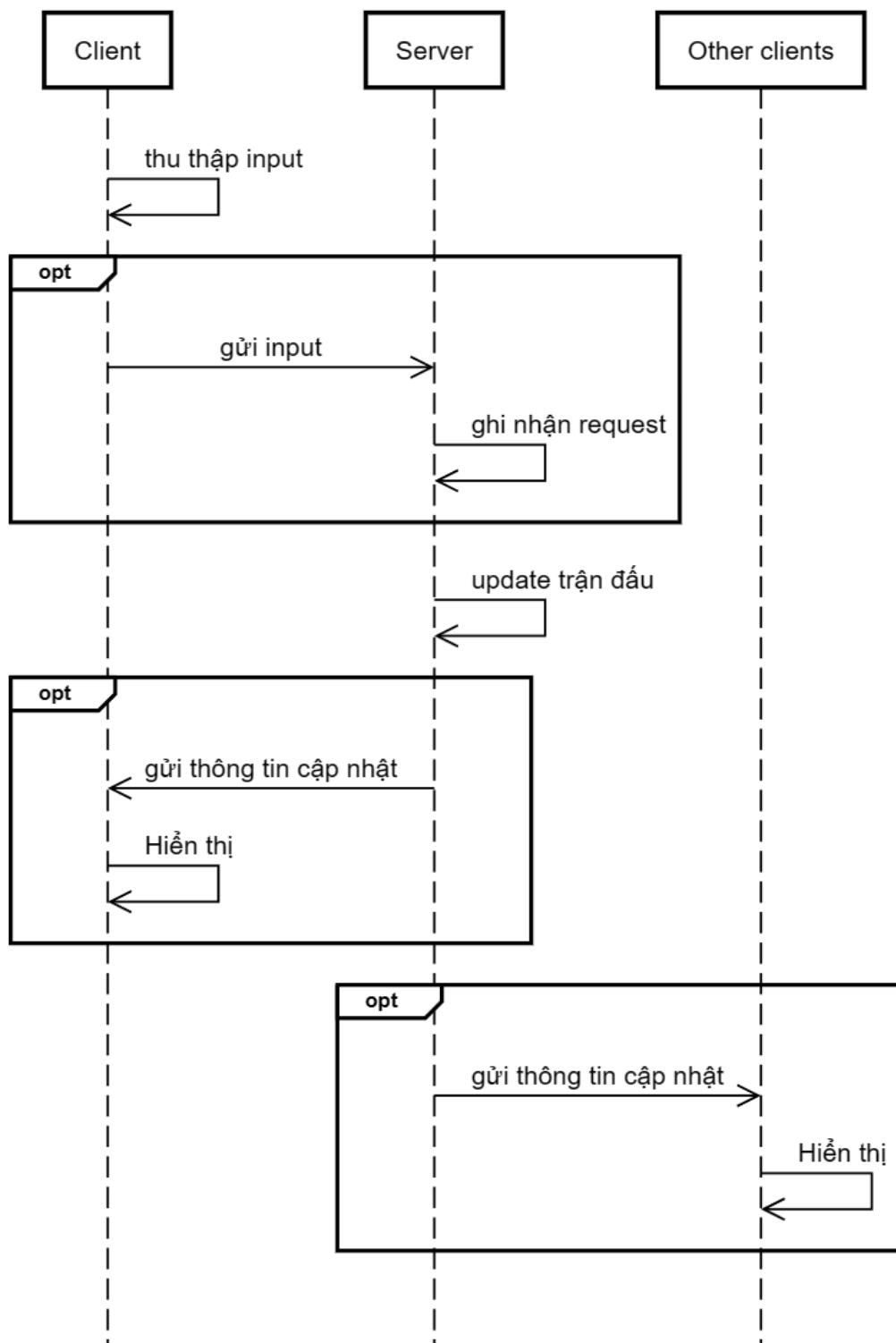
1. Thu thập các input của người chơi.
2. Đóng gói và gửi các input của người chơi cùng với frametime.
3. Đọc các gói tin trả về từ server và cập nhật lại thông tin của các vật thể cần hiển thị.
4. Render game

Đối với server, vòng lặp sẽ có dạng như sau:

1. Đọc các input từ client.
2. Xử lý các input từ client.
3. Mô phỏng game sử dụng frame time.
4. Tạo các gói cập nhật của game tương ứng với từng client và gửi đi.

Với cấu trúc client-server đơn giản như trên, về cơ bản hệ thống sẽ hoạt động như sau: Đầu tiên client tạo và gửi các input từ người dùng đến cho server, server sau đó xử lý và cập nhật lại trạng thái tất cả các vật thể cho client để client hiển thị.

Mô hình client server



Hình 30: Minh họa mô hình client-server

Thiết kế trên tuy khá đơn giản và dễ hiện thực nhưng lại gặp nhiều vấn đề khi triển khai trên môi trường mạng. Vấn đề lớn nhất chính là sự ảnh hưởng của độ trễ khi gửi gói tin đến trái



nghiệm chơi game của người chơi. Giả sử một gói tin input của người chơi mất một khoảng thời gian 50ms để đi từ client đến server, đợi server xử lý rồi trả ngược kết quả về cho client, và gói tin trả về này cũng mất 50ms mới đến nơi, thì tất cả các hành động của người chơi đều sẽ phải đợi 100ms sau mới thấy được kết quả của hành động đó trên màn hình. Nếu trong một trò chơi một người, ta có thể tạm chấp nhận độ trễ đó, nhưng trong một trò chơi nhiều người với diễn biến nhanh và chỉ cho phép sai lệch tối đa vài chục mili giây, ta cần có các kỹ thuật để hạn chế sự ảnh hưởng của vấn đề độ trễ này đến trải nghiệm người chơi.

5.2.2 Kỹ thuật dự đoán phía client

Một kỹ thuật để hạn chế ảnh hưởng của độ trễ mạng là client thay vì chỉ thực hiện các chức năng đơn giản là thu thập input và render theo kết quả trả về từ server thì sẽ tự thực hiện hành động của người chơi ngay sau khi nhận được input một cách địa phương trên client đó mà không cần đợi kết quả trả về từ server.

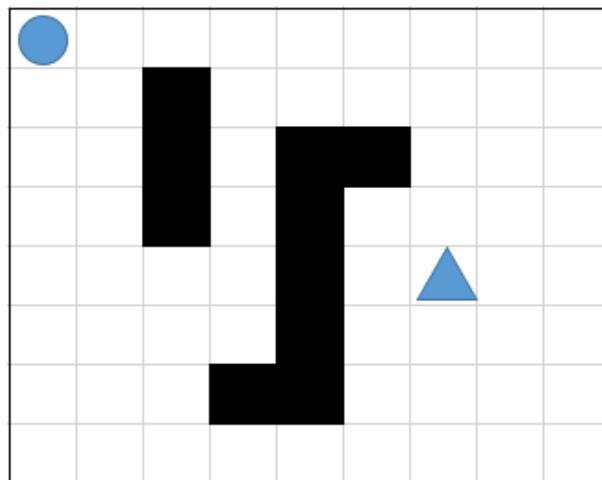
Điều này không có nghĩa là mỗi client sẽ chạy một logic khác nhau mà server vẫn sẽ xử lý input của người chơi như bình thường và sau đó khi client nhận được gói tin kết quả, client sẽ sửa lại trạng thái của các object trong game cho đúng với kết quả trong gói tin đó. Điểm trừ của kỹ thuật này là khi client sửa lại trạng thái các vật thể theo kết quả, có thể sẽ gây ra một sự "giật" nhẹ; nếu việc này xảy ra quá thường xuyên sẽ ảnh hưởng đến trải nghiệm game của người chơi.

5.3 Thiết kế người chơi máy (bot)

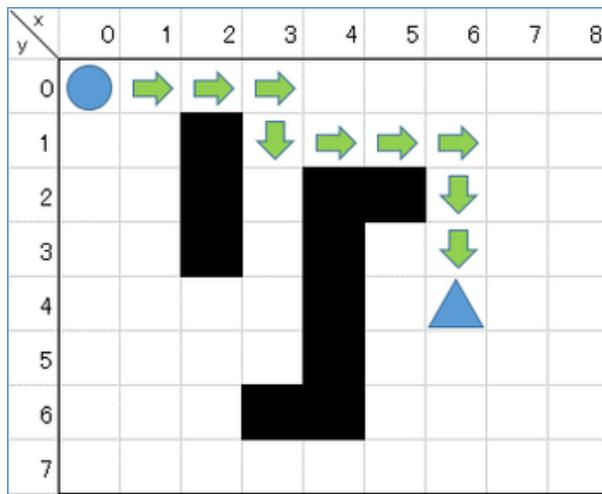
5.3.1 Thuật toán tìm đường đi cho bot

Thuật toán tìm đường đi ngắn nhất cần được áp dụng cho bot khi bot cần di chuyển đến khu vực an toàn, đi nhặt trang bị hay chạy trốn một cách nhanh nhất. Tuy nhiên ở trường hợp tìm địch và chiến đấu, đường đi của bot cần phải linh hoạt và tự nhiên hơn (đi vòng, né tránh,...), do đó giải thuật cần được cải tiến, hoặc thậm chí sẽ phải dùng thêm giải thuật khác để đáp ứng điều này.

Trước tiên, giải pháp mà **Survival2D** sử dụng là giải thuật A*. Trong khoa học máy tính, A* là thuật toán tìm kiếm trong đồ thị và là một thuật toán tìm kiếm theo lựa chọn tốt nhất (best-first search). Thuật toán này tìm một đường đi từ một nút khởi đầu đến một nút đích cho trước (hoặc đến một nút thỏa mãn một điều kiện đích). Thuật toán này sử dụng một hàm heuristic để đánh giá và xếp loại từng nút theo ước lượng về tuyến đường tốt nhất đi qua nút đó.



Hình 31: Minh họa bài toán tìm đường[5]



Hình 32: Minh họa kết quả tìm đường áp dụng A*[5]

Mô tả thuật toán

A* lưu giữ một tập các lời giải chưa hoàn chỉnh trong một hàng đợi ưu tiên (priority queue), nghĩa là các đường đi qua đồ thị, bắt đầu từ nút xuất phát. Thứ tự ưu tiên gán cho một đường đi x được quyết định bởi hàm $f(x) = g(x) + h(x)$, trong đó:

- $g(x)$: là chi phí của đường đi cho đến thời điểm hiện tại, nghĩa là tổng trọng số của các cạnh đã đi qua
- $h(x)$: là hàm heuristic ước lượng về chi phí nhỏ nhất để đến đích nếu đi qua x

5.3.2 Xây dựng cây hành vi (behavior tree) quản lý hành vi cho bot

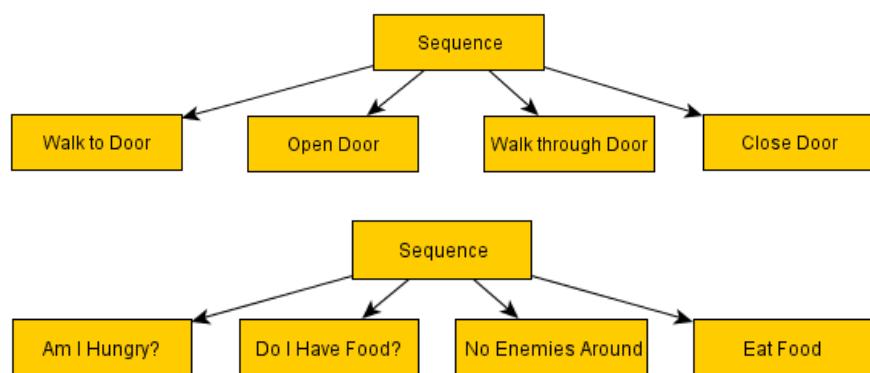
Cây hành vi (behavior tree) là một mô hình toán học được sử dụng phổ biến trong các lĩnh vực như khoa học máy tính (computer science), người máy (robotics), hệ thống điều khiển (control systems) và đặc biệt là trong công nghiệp game (video game industry). Behavior tree

mô tả việc chuyển đổi giữa một tập hợp hữu hạn các nhiệm vụ theo kiểu module, mỗi module là một nhiệm vụ phức tạp được tạo thành từ các nhiệm vụ nhỏ đơn giản hơn, và cái hay của behavior tree là không cần biết các nhiệm vụ nhỏ đó được hiện thực thế nào.

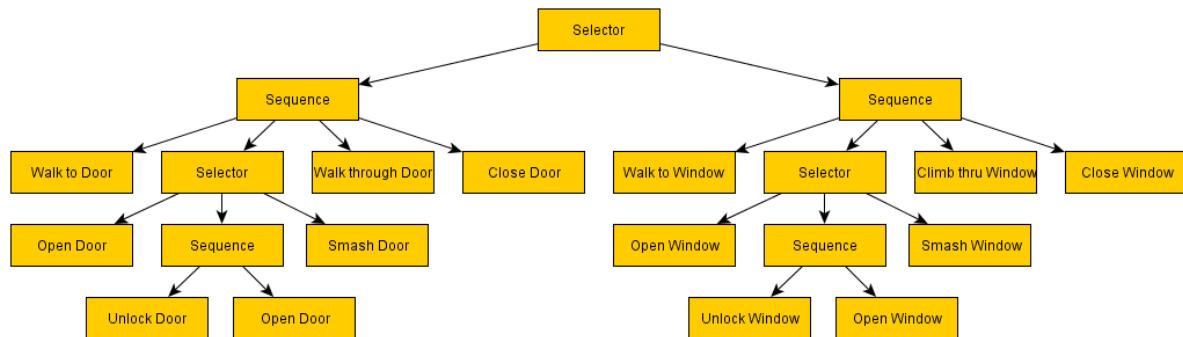
So sánh với một mô hình khác là máy trạng thái (state machine)[6] phổ biến không kém trong lĩnh vực khoa học máy tính và cũng có thể dùng để quản lý trạng thái cho bot, behavior tree cho thấy sự trực quan hơn, thân thiện, dễ hiểu hơn với đời sống thực khi quản lý hành vi bằng các module nhiệm vụ, quyết định hành động thực tế của nhân vật xuyên suốt game thay vì quản lý trạng thái của chúng. Không dừng lại ở đó, behavior tree thực sự mạnh khi có khả năng thực thi nhiều nhiệm vụ cùng lúc bằng cách thêm các nhiệm vụ song song, trong khi state machine chỉ có thể có một trạng thái trong một thời điểm. Bên cạnh đó tính linh hoạt và khả năng mở rộng cũng tốt hơn state machine nhờ vào cấu trúc cây. Các điểm mạnh kể trên cho thấy behavior tree cực kỳ phù hợp trong việc thiết kế bot dành cho cộng đồng phát triển game nói chung và cho **Survival2D** nói riêng.[7]

Khái quát về ý tưởng của behavior tree

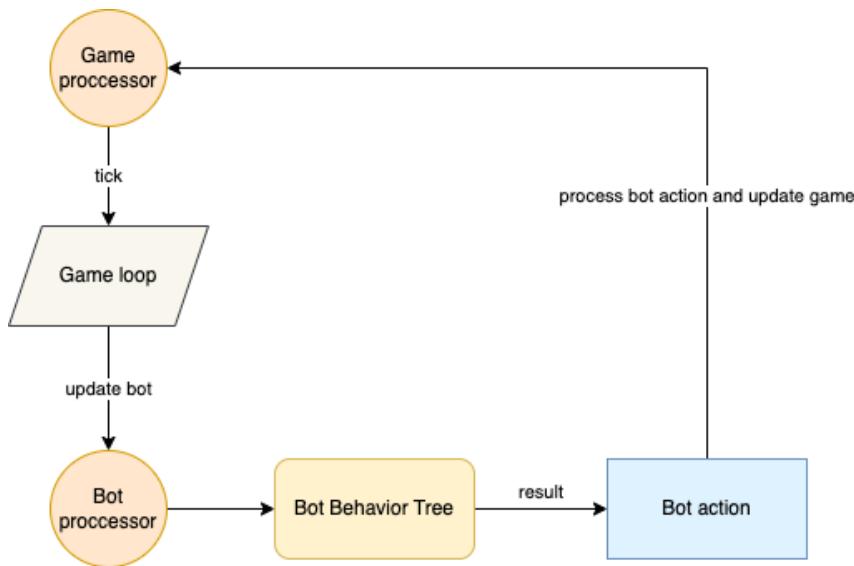
Như đã nêu trên, behavior tree có cấu trúc cây và là một cây gồm các nút con phân tầng, kiểm soát chuỗi các hành vi của bot. Các nút trong hay còn gọi là các cây con của behavior tree đại diện cho một nhóm các nhiệm vụ, nút càng sâu thì nhiệm vụ càng rõ ràng. Duyệt đến tầng sâu nhất ta sẽ gặp các nút lá, chính là nhiệm vụ thực sự mà bot cần phải thực hiện. Nói dễ hiểu hơn khi xét cây từ dưới lên là: các nút lá là các lệnh thực điều khiển hành vi, các nút trong "gộp nhóm" các nút lá lại theo từng nhóm nhiệm vụ. Hình 33 và 34 là một ví dụ về việc "gộp nhóm" các nhóm nhiệm vụ trong cấu trúc của behavior tree.



Hình 33: Minh họa các nhiệm vụ đơn giản của behavior tree[8]



Hình 34: Minh họa một nhiệm vụ phức tạp hơn của behavior tree[8]



Hình 35: Cách hoạt động của bot sử dụng behavior tree

Luồng thực thi của behavior tree cũng rất đơn giản. Ở mỗi thời điểm trong game (tick), ta duyệt cây tiền thứ tự (preorder), thực thi nút lá và trả về kết quả. Kết quả từ các nút con sẽ quyết định kết quả cho cha của nó theo một số quy luật cụ thể, từ đó quyết định nhiệm vụ tiếp theo sau được duyệt đến và thực thi hay là bị cắt tỉa và chuyển sang nhiệm vụ khác. Cứ thế cho đến nhiệm vụ cuối cùng theo thứ tự duyệt cây. Tập các nút lá mà ta duyệt đến được chính là các nhiệm vụ mà bot thực thi trong tick đó.

Các kết quả mà một nút có thể trả về bao gồm:

- SUCCESS - phản ánh nhiệm vụ thành công.
- FAILURE - phản ánh nhiệm vụ thất bại.
- RUNNING - phản ánh nhiệm vụ đang trong quá trình thực thi, chưa có kết quả sau cùng.

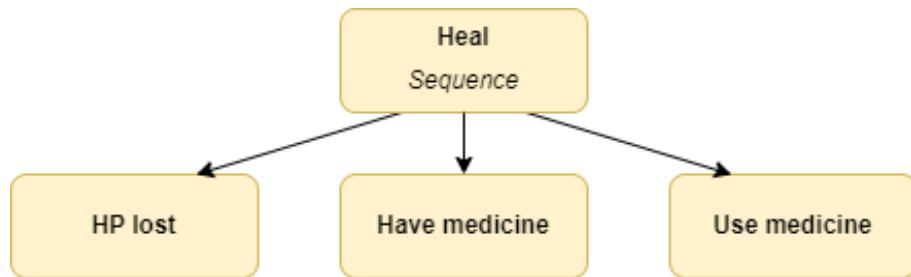
Behavior tree sử dụng cho Survival2D

Có thể thấy quy luật quyết định kết quả của một nút trong là rất quan trọng. Áp dụng vào đề tài, behavior tree của **Survival2D** sử dụng những loại nút trong cơ bản sau đây:

- *Sequence*

Thực thi các nút con theo thứ tự, FAILURE nếu có bất cứ nút con nào FAILURE.

Nút Sequence thường dùng trong các trường hợp nhiệm vụ có nhiều bước, cần phải hoàn thành hết các bước, hay đơn giản có thể dùng để kiểm tra điều kiện trước khi thực hiện nhiệm vụ.



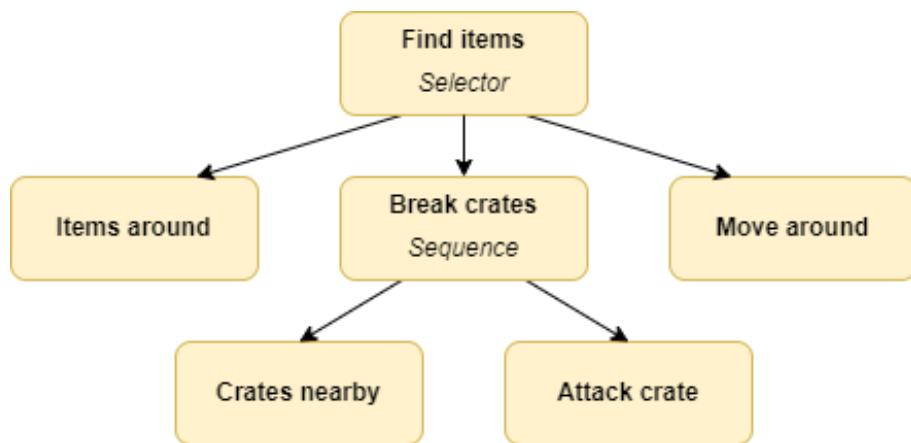
Hình 36: Ví dụ về nút Sequence trong **Survival2D**

Bot sẽ thực hiện các nhiệm vụ kiểm tra có bị mất máu hay không, sau đó kiểm tra có thuốc hồi máu hay không, cuối cùng là dùng thuốc để hồi máu.

- *Selector*

Thực thi các nút con theo thứ tự, SUCCESS nếu có bất cứ nút con nào SUCCESS.

Nút Selector thường dùng trong các trường nhiệm vụ có nhiều cách để hoàn thành, và có thể dùng để kiểm tra điều kiện trước khi thực hiện nhiệm vụ (có thể xem Selector là đối nghịch với Sequence, Selector là phép OR và Sequence là phép AND).



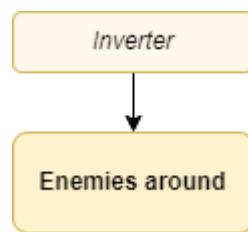
Hình 37: Ví dụ về nút Selector trong **Survival2D**

Có nhiều cách để tìm kiếm một vật phẩm, bao gồm kiểm tra vật phẩm xung quanh, đập thùng tìm và di chuyển đến chỗ khác nếu cả hai nhiệm vụ trước đều FAILURE.

- *Inverter*

Thực thi nút con (Inverter chỉ có một con) sau đó trả về kết quả ngược lại với con của nó.

Nút Inverter cơ bản thường dùng trong kiểm tra điều kiện, có thể tận dụng những nút có sẵn rồi đảo kết quả lại, giảm số lượng nút cần phải hiện thực.

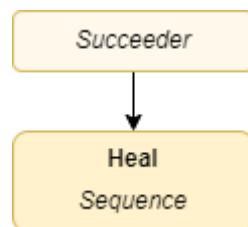


Hình 38: Ví dụ về nút Inverter trong **Survival2D**
Đảo nghịch kết quả của nhiệm vụ kiểm tra địch xung quanh thay vì định nghĩa nhiệm vụ mới.

- *Succeeder*

Thực thi nút con (Succeeder chỉ có một con) sau đó luôn trả về SUCCESS.

Nút Succeeder dùng cho những nhiệm vụ không bắt buộc, giúp chúng được thực thi nếu có thể và không quan tâm về kết quả có thành công hay không, không làm ảnh hưởng đến luồng thực thi các nhiệm vụ tiếp theo.

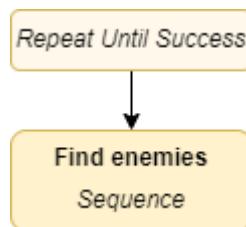


Hình 39: Ví dụ về nút Succeeder trong **Survival2D**
Hồi máu là một hành vi có thể xảy ra song song với nhiều hành vi khác (không để phí thời gian và đảm bảo lượng máu là tốt nhất có thể), vì vậy Succeeder giúp nhiệm vụ hồi máu được thực thi cùng với những nhiệm vụ khác.

- *Repeat Until Success*

Ngoài ra còn một loại kết quả nữa là RUNNING. Khi một nút con trả về RUNNING, kết quả của nút cha cũng sẽ là RUNNING và nhiệm vụ đó được hiểu là đang thực thi, vào tick tiếp theo nó sẽ được tiếp tục.

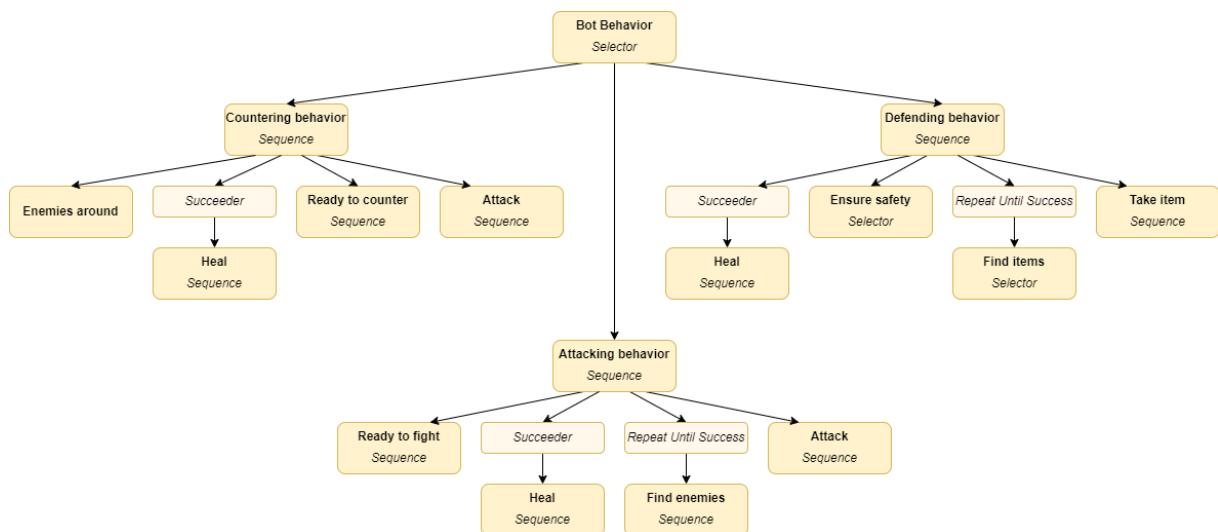
Repeat Until Success thực thi nút con (chỉ có một con) lặp lại ở mỗi tick cho đến khi nút con SUCCESS, nếu nút con FAILURE nó sẽ trả về RUNNING. Ta cũng có thể biến đổi điều kiện một chút để có thêm giới hạn số lần lặp, dành cho các nhiệm vụ chỉ cho phép thử thực hiện lại trong một khoảng thời gian nhất định.



Hình 40: Ví dụ về nút Repeat Until Success trong Survival2D

Trong khi đang ở trạng thái tấn công, bot sẽ đi tìm địch nếu không thấy địch, và sẽ tìm đến khi thấy địch mới chuyển sang hành động tấn công địch.

Từ những nút cơ bản kể trên, cấu trúc behavior tree rút gọn quản lý hành vi của bot trong Survival2D được xây dựng như sau:



Hình 41: Behavior tree rút gọn của Survival2D

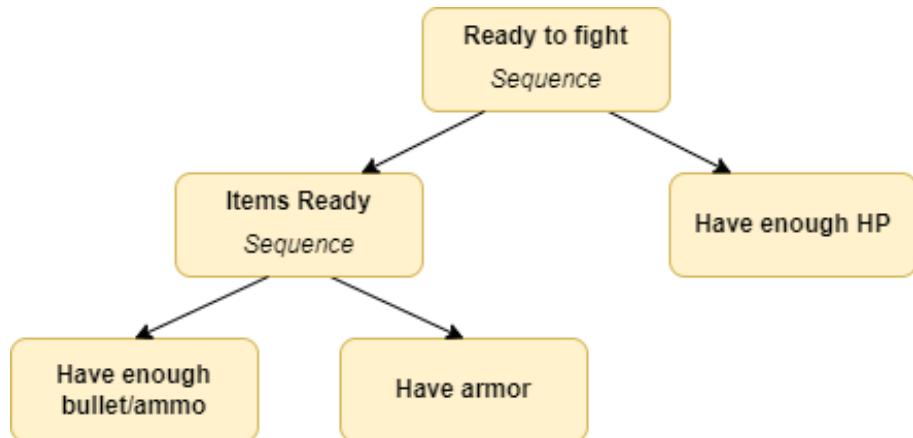
Sau mỗi tick, bot sẽ mang một trạng thái là phản công, tấn công hoặc phòng thủ, tùy vào nhóm hành vi tương ứng nào sẽ SUCCESS. Sở dĩ 3 trạng thái này có thể sắp xếp theo một thứ tự khác, tuy nhiên cách hiện thực các nhiệm vụ con sẽ phải thay đổi đôi chút để vừa đảm logic vừa không làm cấu trúc cây trở nên quá rối.

Theo cách hiện thực trên, nhóm hành vi phản công được duyệt đến đầu tiên. Khi duyệt qua nhóm này, hành vi đầu tiên mà bot thực hiện là kiểm tra có địch xung quanh hay không, nếu có thì sẽ tiếp tục thực hiện nhiệm vụ tiếp theo để phản công bao gồm hồi máu, kiểm tra có đủ trang bị để phản công hay không, và sau đó tấn công kẻ địch. Nếu không có địch xung quanh, hoặc có nhưng lại không đủ trang bị, cây sẽ duyệt tiếp sang nhóm hành vi tấn công.

Ở nhóm hành vi tấn công, đầu tiên bot kiểm tra có trang bị, máu đã sẵn sàng chiến đấu hay chưa, hồi máu nếu có thể, tìm kiếm địch và tấn công tiêu diệt địch sau khi tìm thấy. Khi không đủ sẵn sàng để tấn công nữa, cây duyệt tiếp sang nhóm hành vi cuối là nhóm phòng thủ.

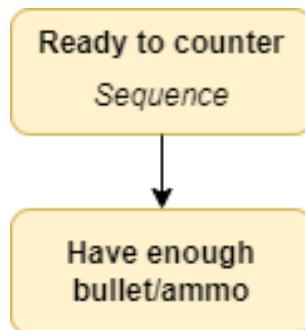
Khi chuyển sang phòng thủ, đầu tiên dĩ nhiên lại là hồi máu nếu có thể, sau thực hiện nhiệm vụ đảm bảo an toàn cho bản thân. Khi đã an toàn rồi, bot sẽ đi tìm vật phẩm để tăng cường trang bị cũng như có thêm thuốc để hồi máu, và khi đã đủ điều kiện sẵn sàng quay lại chiến đấu, bot hoàn toàn có thể quay lại nhóm hành vi phản công hoặc chủ động tấn công ở tick tiếp theo.

Ngoài nút nhiệm vụ hồi máu (Heal) và nhiệm vụ tìm vật phẩm (Find items) đã được mô tả trong ví dụ về Sequence và Selector, cấu trúc cây trên còn có các nút nhiệm vụ khác đã được rút gọn, chưa mô tả rõ là: sẵn sàng phản công (Ready to counter), sẵn sàng chiến đấu (Ready to fight), tìm địch (Find Enemies), tấn công (Attack), bảo đảm an toàn (Ensure safety) và nhặt vật phẩm (Take item).



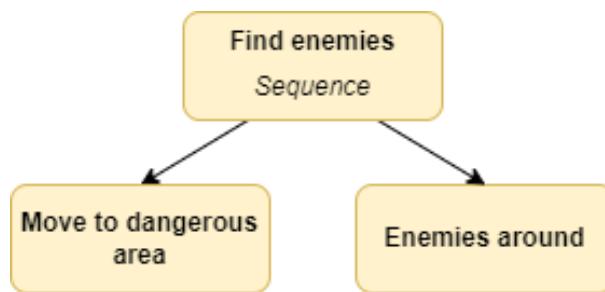
Hình 42: Thiết kế của nhiệm vụ kiểm tra sẵn sàng chiến đấu

Bot trong **Survival2D** kiểm tra tình trạng của chính nó có sẵn sàng chiến đấu hay không bằng cách: kiểm tra vũ khí trang bị đầy đủ bao gồm đầy đủ các loại đạn, giáp, và kiểm tra máu có ở mức an toàn chưa.



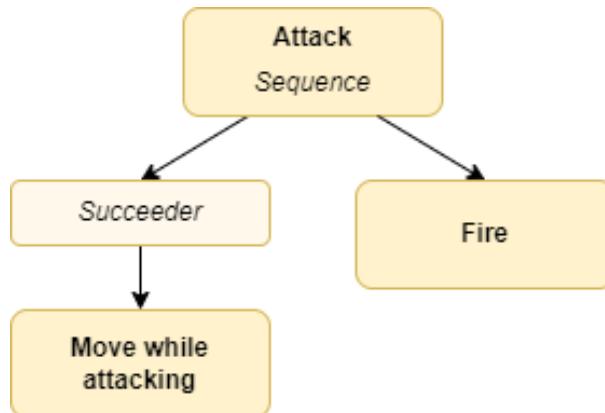
Hình 43: Thiết kế của nhiệm vụ kiểm tra sẵn sàng phản công

Điều kiện sẵn sàng phản công thì ít hơn so với sẵn sàng chiến đấu vì lúc đó có địch ở gần nên tình hình cấp bách hơn, chỉ cần có đủ đạn là đã phản công được.



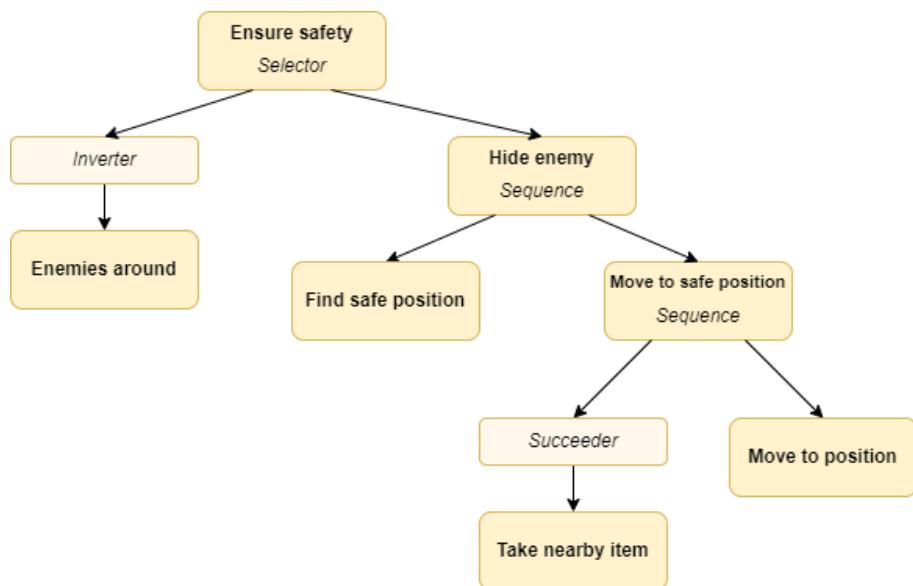
Hình 44: Thiết kế của nhiệm vụ tìm địch

Vì không biết được địch đang ở những vị trí nào trên map nên việc xác định vị trí chính xác để di chuyển đến là không thể, thay vào đó bot sẽ thực hiện hành vi như người chơi thông thường là sẽ di chuyển đến vùng an toàn (Safe Zone (2.3)) - khu vực có khả năng cao tìm thấy địch, vừa di chuyển vừa kiểm tra xem đã thấy địch xung quanh hay chưa.)



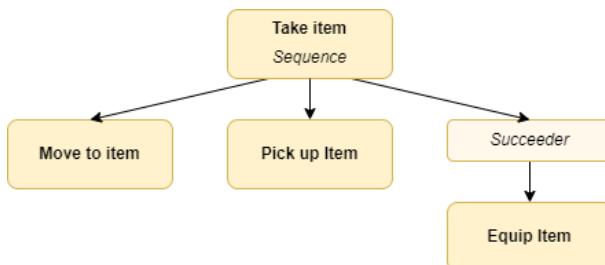
Hình 45: Thiết kế của nhiệm vụ tấn công

Sau khi đã xác định được mục tiêu, bot tiến hành tấn công bằng cách bắn và cố gắng di chuyển linh hoạt để vừa bắn vừa né nếu địch phản công, hoặc cũng có thể tránh không bị các vật cản chặn đường bắn.



Hình 46: Thiết kế của nhiệm vụ bảo đảm an toàn

Hành vi chính của việc bảo đảm an toàn là chạy trốn địch khi phát hiện có địch ở gần, trong lúc chạy thì nhặt luôn các vật phẩm trên đường nếu có thể.



Hình 47: Thiết kế của nhiệm vụ nhặt vật phẩm

Nhiệm vụ nhặt vật phẩm đơn giản chỉ là việc di chuyển tới vật phẩm mục tiêu, nhặt vật phẩm lên và sử dụng luôn nếu cần thiết.

Behavior tree được thiết kế cho bot của **Survival2D** là tương đối đơn giản, dễ hiểu nhưng đủ để bot có những hành vi thông minh, kết hợp với khả năng tìm đường đi có thể còn tốt hơn người chơi thật, giúp game không bị nhàm chán, tạo thêm nhiều sự khó khăn để người chơi trải nghiệm.

5.3.3 Thiết kế độ khó cho bot

Các cấp độ khó của bot thật ra cũng không có thước đo chính xác. Trong **Survival2D**, để thay đổi phong cách chơi cho bot, tạo sự đa dạng thì có một hệ số được đặt ra, tạm gọi là *hệ số tự tin* α ($0 \leq \alpha \leq 1$). Hệ số này sẽ ảnh hưởng đến hàm đánh giá cho những nhiệm vụ kiểm tra, khiến cho hành vi của bot sẽ khác nhau theo từng phong cách chơi.

Ví dụ, bot có phong cách liều lĩnh, thích tấn công có α cao, điều kiện để sẵn sàng chiến đấu (về cả trang bị và máu) sẽ yêu cầu ít khắt khe hơn bot có phong cách dè chừng, thiên về phòng



thủ.

$$readyToFightHp = (1 - \alpha) * maxHp$$

$$readyToFightAmmo = (1 - \alpha) * maxAmmo$$

Tuy nhiên nếu chỉ dùng mỗi α thì chưa đủ cho sự đa dạng, vì như ta có thể thấy ở hai phương trình trên, tỉ lệ máu cần và tỉ lệ số lượng đạn cần sẽ bằng nhau, do đó ở mỗi điều kiện sử dụng thêm một hệ số độ dời ngẫu nhiên nữa, lúc này game sẽ có bot yêu cầu ít máu nhưng nhiều đạn và bot yêu cầu nhiều máu nhưng ít đạn.

$$readyToFightHp = (1 - \alpha) * maxHp * randomOffsetHp$$

$$readyToFightAmmo = (1 - \alpha) * maxAmmo * randomOffsetAmmo$$

Bên cạnh $hệ số tự tin \alpha$, việc chọn độ ưu tiên sử dụng loại súng (và loại đạn đi kèm) sẽ tạo ra thêm những phong cách chơi khác nhau cho bot, do đó mỗi loại đạn sẽ có một trọng số khác nhau khi đánh giá số lượng đạn, và các trọng số của cùng một loại đạn cũng sẽ khác nhau đối với những con bot khác nhau.

Một biện pháp nữa cũng làm thay đổi hành vi của bot là sử dụng Selector thay cho Sequence ở một số nút, thậm chí là random thứ tự của những nút con để đẩy sự đa dạng lên cao nhất. Tuy nhiên ở thiết kế hiện tại của behavior tree trong **Survival2D** vẫn chưa có các nút có thể áp dụng biện pháp này.

6 Kiểm thử

Để đảm bảo game hoạt động đúng với mong muốn, nhóm đã thực hiện kiểm thử cho một số logic quan trọng. Do các logic ngày thường có tính chất ngẫu nhiên, nên nhóm đã kiểm thử bằng việc tự kiểm tra lại các output bằng tay; chứ không theo quy nguyên tắc truyền đầu vào (input) sau đó kiểm tra kết quả (actual) có đúng với mong đợi (expect) không.

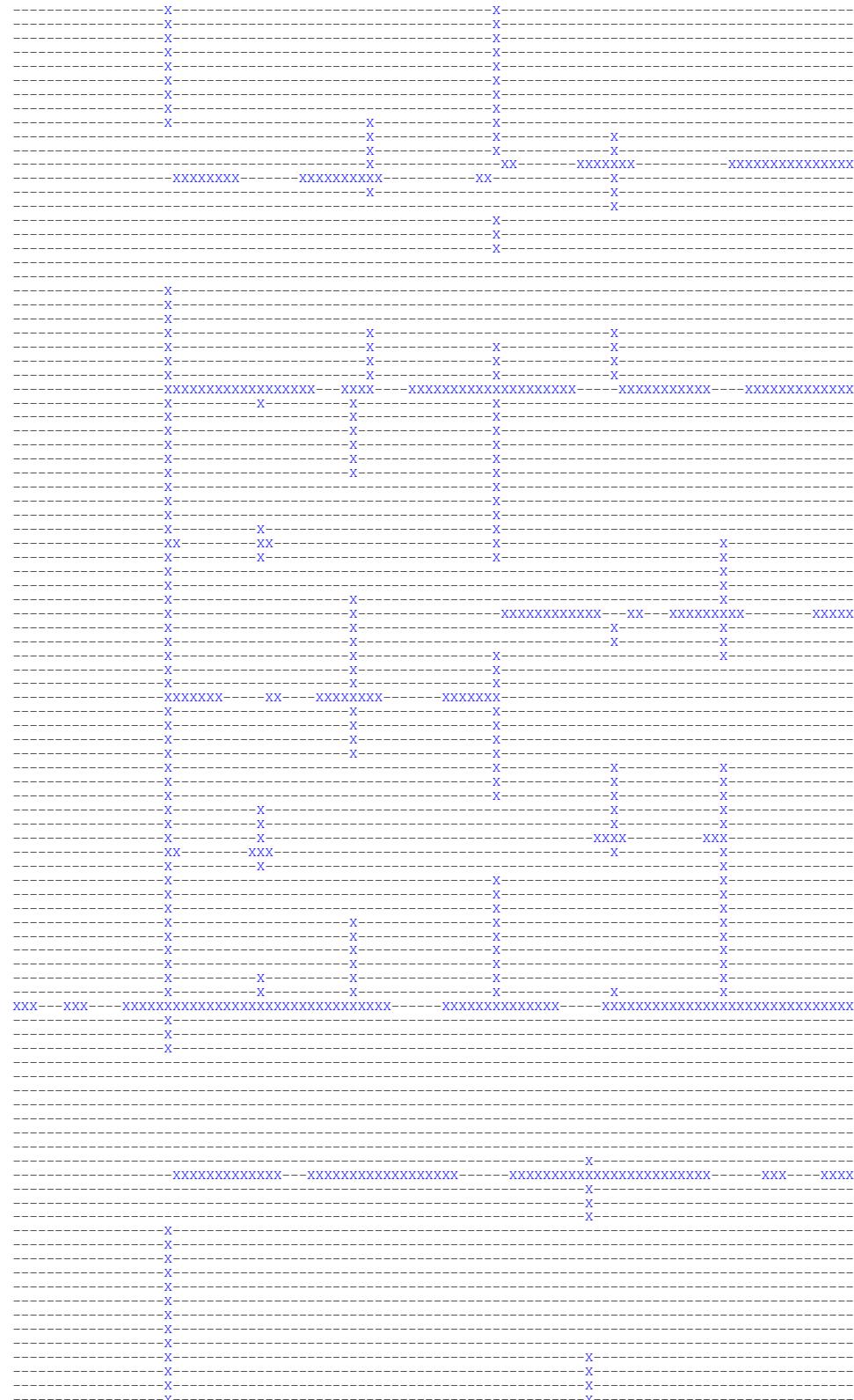
6.1 Thuật toán tạo bản đồ

Hình 48 là kết quả của một lần kiểm thử bước tạo tường của thuật toán tạo bản đồ, với ký tự X biểu thị một ô tường và ký tự – biểu thị một ô trống. Kết quả cho thấy thuật toán tạo ra được các vùng không quá nhỏ liên thông với nhau, đáp ứng được yêu cầu ban đầu.

Hình 49 là kết quả kiểm thử thuật toán tạo các vật thể trên map, với thông tin tường là output bên trên. Kết quả cho thấy xung quanh vị trí sinh ra player có các item và vật cản đảm bảo cân bằng.

Chú thích:

- Ký tự -: các ô trống
- Ký tự 1: vị trí sinh ra player
- Ký tự 2: vị trí sinh ra item
- Ký tự 3: tường
- Ký tự 4: cây
- Ký tự 5: thùng
- Ký tự 6: đá



Hình 48: Kết quả một lần kiểm thử thuật toán tạo tường

Hình 49: Kết quả một lần kiểm thử thuật toán tạo các vật thể trên bản đồ

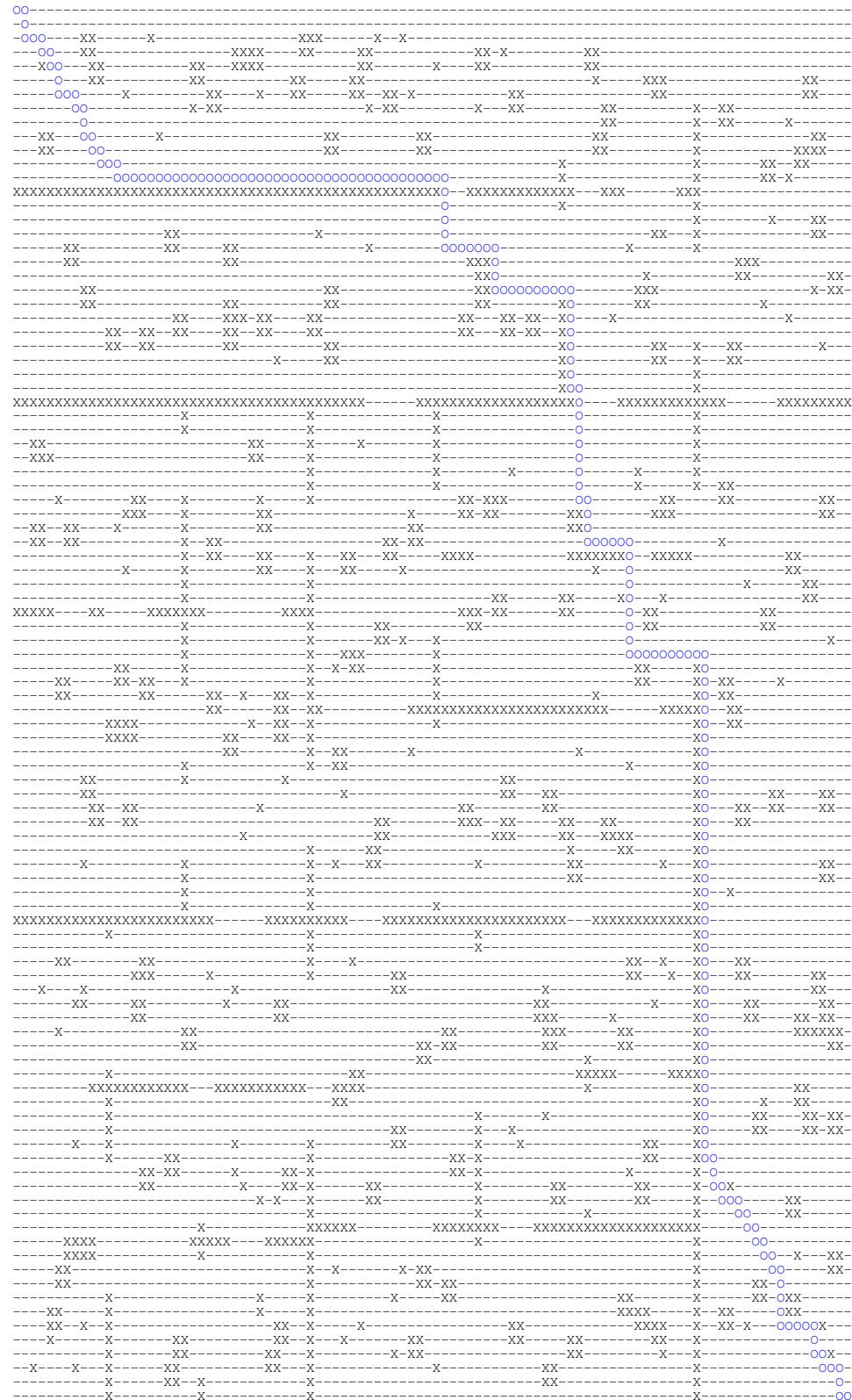


6.2 Thuật toán tìm đường A*

Hình 50 thể hiện một lần thực thi giải thuật tìm đường A* từ góc trên bên trái bản đồ đến góc dưới bên phải bản đồ.

Chú thích:

- Ký tự -: các ô trống
- Ký tự X: vị trí các vật cản: tường, cây, thùng, đá
- Ký tự O: vị trí mà thuật toán tìm đường đi qua



Hình 50: Kết quả một lần kiểm thử thuật toán tìm đường A*

6.3 Kiểm thử tối ưu tìm kiếm xung quanh một khu vực bằng quadtree

Kết quả khi thực hiện performance test 1000 lần với map có 1000 object di chuyển trong 1 tick được kết quả như sau:

Without QuadTree	
Min: 6.0	
Max: 31.0	
Avg: 8.455	(Đơn vị đo <i>millis</i> , kết quả được làm tròn xuống)
With QuadTree	
Min: 0.0	
Max: 7.0	
Avg: 0.611	

Hình 51: Kết quả một lần kiểm thử hiệu năng khi tìm kiếm và quản lý các vật thể sử dụng quadtree so với sử dụng danh sách

Có thể thấy với cách duyệt qua objects truyền thông, cần trung bình khoảng 8.46ms, gần 50% thời gian của một tick ($\approx 16.67\text{ms}$) để thực hiện tìm kiếm object gần đó; chưa kể đến việc xử lý logic. Thậm chí trường hợp xấu nhất còn tồn gần gấp đôi thời gian một tick, điều này sẽ ảnh hưởng rất lớn đến khả năng xử lý của **Survival2D**. Trong khi đó bằng biện pháp ứng dụng Mẫu thiết kế Phân vùng không gian với QuadTree, hiệu năng được tăng đáng kể, trung bình chưa đến 1ms, trường hợp tệ nhất cũng chưa đến 50% thời gian của 1 tick, tốt hơn cả trường hợp trung bình của cách sử dụng danh sách.

6.4 Kiểm thử vùng nhìn của Player

Nhóm đã tạo cheat zoom ở client để kiểm thử vùng nhìn logic của người chơi. Kết quả ghi nhận đúng mong muốn: người chơi không biết gì về các vật thể nằm ngoài vùng nhìn của mình.



Hình 52: Kiểm thử vùng nhìn của người chơi

6.5 Kiểm thử trận đấu thực tế

Để đảm bảo game chạy đúng logic và đầy đủ flow, nhóm cũng đã tự chơi với nhau, ghi nhận một số hình ảnh trong trận đấu:



Hình 53: Gameplay của Survival2D: Giữ an toàn, tìm kiếm trang bị đầy đủ



Hình 54: Gameplay của Survival2D: Bắt gặp và giao chiến với đối thủ



Hình 55: Gameplay của Survival2D: Tiêu diệt thành công đối thủ



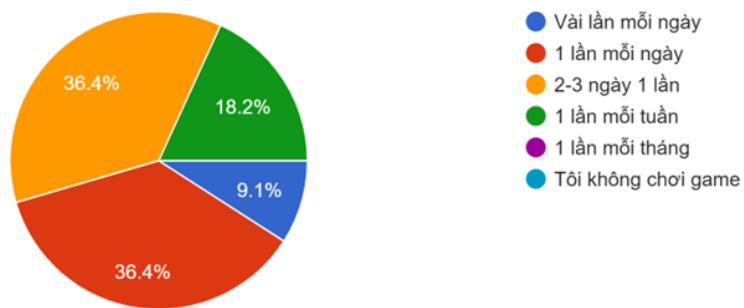
Hình 56: Gameplay của Survival2D: Chiến thắng khi là người sống sót cuối cùng

Ngoài ra nhóm cũng đã có kiểm thử tốc độ mạng thực tế, như đã nêu ở mục 3.2.4 đảm bảo game có thể chơi ổn định, không bị giật lag.

Nhóm cũng mời một số bạn bè tham gia trải nghiệm và thu thập phản hồi từ các bạn để có nhiều góc nhìn hơn và cải thiện game tốt hơn. Kết quả tổng hợp được từ 11 phản hồi như sau:

Bao lâu bạn chơi game một lần?

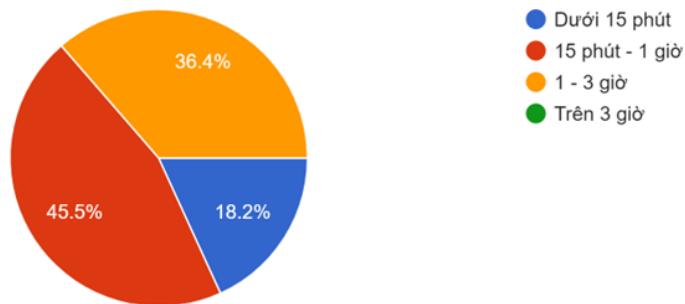
11 responses



Hình 57: Phản hồi: Bao lâu bạn chơi game một lần?

Mỗi lần chơi bạn chơi bao lâu?

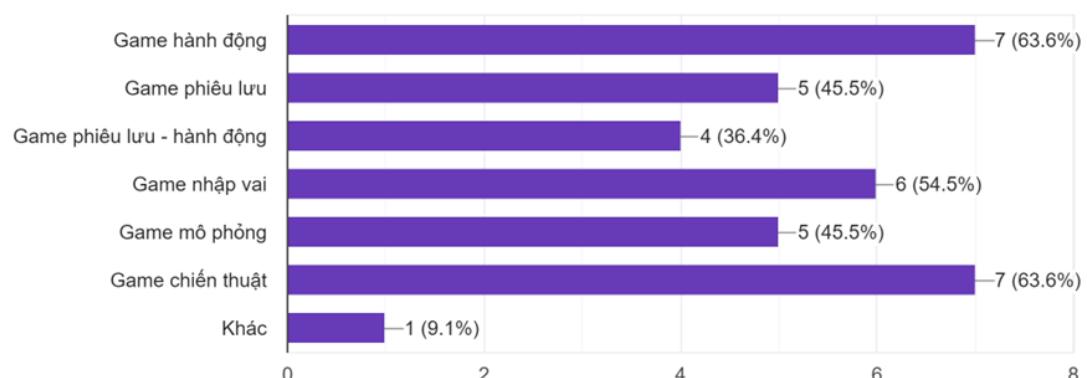
11 responses



Hình 58: Phản hồi: Mỗi lần chơi bạn chơi bao lâu?

Thể loại game mà bạn chơi

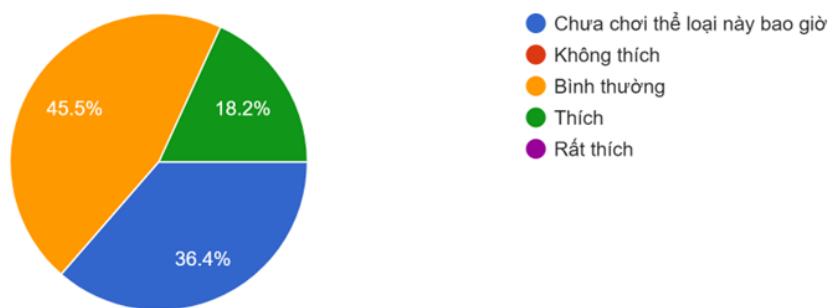
11 responses



Hình 59: Phản hồi: Thể loại game mà bạn chơi

Mức độ yêu thích thể loại game bắn súng sinh tồn

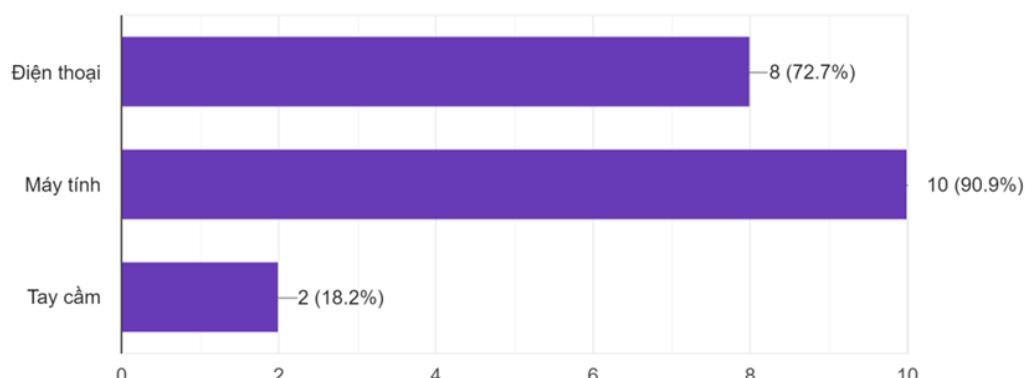
11 responses



Hình 60: Phản hồi: Mức độ yêu thích thể loại game bắn súng sinh tồn

Thiết bị bạn chơi game

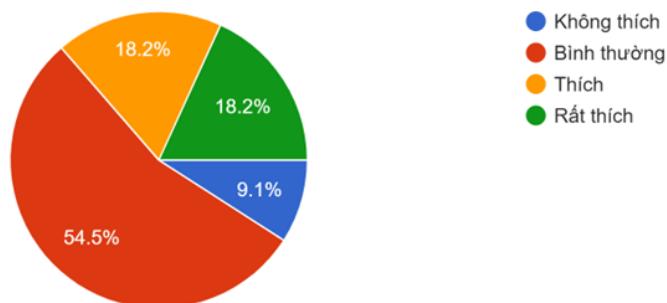
11 responses



Hình 61: Phản hồi: Thiết bị bạn chơi game

Bạn có thích chơi Survival2D không?

11 responses



Hình 62: Phản hồi: Bạn có thích chơi Survival2D không?

Game bắn nhau kịch tính, vui
Đồ họa vui nhộn, không có cảm giác máu me kinh dị
Được chơi PUBG mà không cần mua card đồ họa
Sản phẩm người quen làm nên khá thích và tự hào về game
Chơi mượt, không thấy giật lag
Mượt, chơi luôn trên web khá là nhanh và tiện
Game chơi vui, nhanh, chơi nhiều trận liên tiếp không sập server

Bảng 10: Phản hồi: Điều gì làm bạn thích Survival2D?

Chơi với bot khá chán
Chưa có art đẹp, ít tính năng không tạo nhóm chơi chung được với bạn bè
Súng đơn giản, chơi chán
Bot lộ liễu, chạy rất cứng nhắc có khi còn đứng yên
Chơi thua cả bot, không vui
Xem mini map không biết được đi hướng nào để vượt tường nhanh hơn
Di chuyển đôi khi vướng tường, vướng vật cản
Bot chơi rập khuôn quá, di chuyển dễ bắt bài
Bot bắn súng quá nhanh, không thể né
Game mới ở mức prototype, không nhiều tính năng
Bản đồ đơn điệu, các vật thể ngay hàng thẳng lối như được sắp đặt, không nhiều tinh tiết như cây mọc sát nhau thành rừng,...
Di chuyển gần vật thể không mượt
Súng đạn không đa dạng

Bảng 11: Phản hồi: Điều gì bạn không thích ở Survival2D?



Thiết kế tường có khuôn mẫu hơn
Phát triển nhiều tính năng hơn nữa
Thêm súng và nhiều loại vũ khí khác
Vẽ game đẹp hơn hoặc nâng lên 3D
Cơ chế di chuyển sát tường
Làm nhà máy, hầm mỏ, nhiều chỗ nấp
Ra phiên bản trên điện thoại
Cải thiện bot bắn thông minh hơn
+ Gameplay chán, chơi 1 trận sẽ chẳng muốn chơi lại
+ Design thêm nhiều vũ khí
+ Bot không thông minh, chơi dễ thắng
+ Không thân thiện người dùng, di chuyển sát tường không có "trượt", Minimap vô dụng, còn che khuất một góc bản đồ
+ Thông báo về play zone vô dụng, không thể xem chính xác play zone
+ Tường cản trở nhiều
+ Không có hiệu ứng, hoạt họa sinh động
Sửa các phần nêu trong mục không thích

Bảng 12: Phản hồi: Bạn có góp ý gì cho nhóm để hoàn thiện game hơn không?

Nhận xét

- Game chưa đủ cuốn hút người chơi.
- Game có mặt tích cực là gọn nhẹ, không cần cài đặt, và ít bị giật lag.
- Game sơ sài, không có nhiều tính năng.
- Một số thao tác, hiển thị không thân thiện với người chơi: di chuyển sát vật cản, thông báo về play zone khó nắm bắt.
- Bot chưa tốt.

7 Đánh giá kết quả đạt được

Trong giới hạn của đề tài, nhóm đã được hoàn thành các yêu cầu ban đầu đề ra, là một game battle royale đầy đủ gameplay cơ bản, có thể chơi multiplayer tốt. Cụ thể như sau:

- **Multiplayer online game**

Server hoạt động ổn với 5 người chơi kết nối chơi cùng lúc, đã deploy được lên VPS đặt ở Singapore. Mức ping dao động của gói gửi thường xuyên nhất là *PlayerMove* nằm trong khoảng 40-50ms (8), được cho là mức ping hợp lý¹². Cá biệt có gói tin lớn như gói *MatchInfo* mức ping dao động 80-100ms, nhưng gói này chỉ gửi 1 lần ở đầu game nên không quá ảnh hưởng.

- **Thuật toán sinh map ngẫu nhiên**

Thuật toán hoạt động tốt. Map được sinh ra có các objects phân bố đều nhau, đảm bảo cân bằng cho người chơi. Vị trí xuất hiện của người chơi được tách xa nhau, đảm bảo người chơi được an toàn trong giai đoạn đầu của trận đấu. Các khu vực của map được liên thông với nhau, người chơi không gặp trở ngại khi chạy vào safezone. Tuy nhiên hình dạng tường chưa đẹp, có những mảng tường xuất hiện rời rạc, chưa thể hiện được mục đích cản trở người chơi.

- **Hệ thống quản lý map objects**

Có đầy đủ những thành phần cơ bản: vật cản, người chơi, vũ khí, trang bị và các vật phẩm. Các object trên map hoạt động đúng logic, tương tác và chạm với nhau chính xác, bao gồm va chạm giữa người chơi với vật cản, giữa đạn với vật cản, giữa đạn với người chơi.

- **Gameplay**

Flow gameplay hoạt động tốt từ bắt đầu trận đấu cho đến kết thúc trận đấu, đầy đủ các tính năng bao gồm di chuyển, tấn công bằng tay, tấn công bằng súng, phá thùng, nhặt vật phẩm, sử dụng vật phẩm. Một vấn đề nhỏ là do thời gian gửi nhận gói tin giữa client và server không khớp được với nhau từng tick, nên khi sử dụng kĩ thuật dự đoán phía client để diễn trước hành động thì hiện tượng giật xảy ra khá thường xuyên. Tuy nhiên đây chỉ là phương án giúp cảm nhận của người chơi tốt hơn trong trường hợp mạng không tốt và cũng không ảnh hưởng đến logic game, nên đối với môi trường mạng ổn định thì không cần dùng tới kĩ thuật dự đoán và game vẫn diễn ra bình thường.

- **Bot**

Bot hoạt động ổn, cơ bản thực hiện được các hành vi tấn công, phòng thủ khác nhau, tuy nhiên vài trường hợp bot vẫn chưa có hành vi như mong đợi, trong đó cần cải thiện nhất là khả năng di chuyển linh hoạt để giống với người chơi thật hơn.

Do chỉ mới được một số bạn bè trải nghiệm, chưa có người chơi thật nên nhóm chỉ mới test được với số lượng người chơi ít, kết quả đánh giá thu được sẽ chưa thật sự khách quan, chưa chính xác so với khi có số lượng lớn người chơi.

¹²What is a good ping? <https://hyperoptic.com/blog/what-is-a-good-ping-for-gaming/>



8 Định hướng phát triển

Vì thời gian phát triển đề tài không nhiều, gói gọn trong đề tài luận văn tốt nghiệp nên **Survival2D** đã được giới hạn chỉ còn một số tính năng chính. Nhưng với thế mạnh là thuộc thể loại bắn súng Battle Royale, đã có nhiều game khác với những tính năng đa dạng để tham khảo; **Survival2D** có rất nhiều ý tưởng để có thể mở rộng. Ngoài ra, cũng có nhiều chủ đề học thuật rất hay có thể áp dụng vào game. Định hướng trong tương lai nếu được phát triển hoàn chỉnh để tung ra thị trường, **Survival2D** có thể mở rộng và cập nhật thêm những tính năng hay, mới lạ như:

- Tracking hành vi của người chơi, lưu trữ và phân tích những dữ liệu thu thập được, để tìm hiểu thói quen của người chơi, các tính năng hấp dẫn, các mặt thiếu sót cần cải thiện.
- Thêm khả năng tự học cho bot áp dụng Machine Learning, kết hợp với data đã tracking để dạy cho bot có nhiều hành vi ngày càng giống với người chơi thật.
- Tối ưu server, tăng khả năng chịu tải hàng nghìn người chơi cùng lúc.
- Phát triển game trên nhiều nền tảng khác để tiếp cận nhiều người chơi hơn.
- Phát triển hệ thống Machine Learning để phát hiện người chơi dùng hack/cheat, dựa vào những hành vi bất thường, có nét tương đồng với những trường hợp hack/cheat đã phát hiện trước đó.
- Chế độ chơi tổ đội, chiến đấu theo nhóm nhiều người.
- Thêm những chế độ chơi khác để tăng độ thú vị của game như: chơi nhanh, sử dụng vũ khí hạn chế, chế độ zombie, chiến tranh...
- Thêm hiệu ứng âm thanh, hình ảnh bắt mắt hơn.
- Mở rộng loại nhiều loại súng, nhiều loại vật phẩm, vũ khí mới như lựu đạn, vũ khí cận chiến,...
- Thêm nhiều dạng địa hình: sông suối, núi đồi,...
- Tương tác với nhau: tin nhắn, voice chat,...

9 Công cụ sử dụng

Trong thời gian hoàn thành đề tài này, từ phần lên ý tưởng, trao đổi, hoàn thiện, deploy lên server thật, những công cụ sau đã giúp đỡ nhóm rất nhiều:

9.1 Chung

Quản lý repository:

- Host: [GitHub](#)
- Client: [GitHub Desktop](#), [SourceTree](#)

Vẽ biểu đồ: [Draw.io](#)

Ghi chú, quản lý tiến độ: [HackMD](#)

Liên lạc: [Telegram](#), [Google Meet](#)

Chia sẻ files: [Google Drive](#)

9.2 Client

- [Cocos2d-x v3.17.2](#)
- Cocos Studio v3.10
- [Visual Studio 2017 v15.9.51](#)
- [Jetbrains WebStorm v2022.1.4](#)
- [Cloudflare Pages](#)

9.3 Server

- [Oracle JDK v17.0.6](#)
- [Oracle OpenJDK v17.0.6](#)
- [Jetbrains IntelliJ IDEA v2022.1.4](#)
- [FlatBuffers v1.12.0](#)
- [Oracle Cloud Infrastructure \(OCI\)](#)
- [Ubuntu Server v22.04](#)

Tài liệu tham khảo

- [1] Tài liệu khoá học Lập trình game (CO3045) - GV: Vương Bá Thịnh (HK202)
- [2] *Netty in Action* (2015), Norman Maurer and Marvin Allen Wolfthal
- [3] AI for game developers: 7 ways AI can take your game to the next level
<https://hub.packtpub.com/ai-for-game-developers>.
Truy cập ngày 06-12-2021.
- [4] Tham khảo thiết kế game Surviv.io
<https://survivio.fandom.com/wiki/>
Truy cập ngày 06-12-2021.
- [5] Giải thuật tìm kiếm A*
https://vi.wikipedia.org/wiki/Gi%e1%ba%a3i_thu%e1%ba%adt_t%C3%ACm_ki%e1%ba%bfm_A%2a
Truy cập ngày 06-12-2021.
- [6] Finite-state machine
https://en.wikipedia.org/wiki/Finite-state_machine
Truy cập ngày 06-12-2021.
- [7] Behavior tree (artificial intelligence, robotics and control)
[https://en.wikipedia.org/wiki/Behavior_tree_\(artificial_intelligence,_robotics_and_control\)](https://en.wikipedia.org/wiki/Behavior_tree_(artificial_intelligence,_robotics_and_control))
Truy cập ngày 06-12-2021.
- [8] Behavior tree áp dụng cho game development
<https://www.gamedeveloper.com/programming/behavior-trees-for-ai-how-they-work>
Truy cập ngày 06-12-2021.
- [9] Các design pattern phổ biến trong java
<https://java-design-patterns.com/>
Truy cập ngày 10-12-2022
- [10] Các design pattern phổ biến trong lập trình game
<https://gameprogrammingpatterns.com/>
Truy cập ngày 10-12-2022