ỦY BAN NHÂN DÂN TP.HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC SÀI GÒN
KHOA CÔNG NGHỆ THÔNG TIN

——————o0o——————

BÁO CÁO

PHÁT TRIỂN HỆ ĐIỀU HÀNH MÃ NGUỒN MỞ

ĐỀ TÀI:

XÂY DỰNG GAME POKER VÀ GAME BOMB

Giảng viên hướng dẫn: **ThS.Từ Lãng Phiêu**

Nhóm: **4**

Sinh viên: **3120410091 - Lữ Nhựt Duy**
3120410241 - Nguyễn Tuấn Anh Khanh
3120410431 - Đỗ Linh Quân
3120410279 - Nguyễn Thị Linh

Lớp: **DCT1215**

TP.Hồ Chí Minh, 5/2024

MỤC LỤC

LỜI NÓI ĐẦU

Trong thời đại công nghệ phát triển như hiện nay, các trò chơi trực tuyến đang ngày càng trở thành một phần không thể thiếu trong cuộc sống hàng ngày của chúng ta. Trong số đó, trò chơi Poker là một trong những trò chơi phổ biến nhất và được yêu thích trên toàn thế giới.

Việc xây dựng một game Poker trực tuyến không chỉ đem lại sự giải trí mà còn là một thách thức đối với các nhà phát triển. Trò chơi này đòi hỏi một hệ thống phức tạp để quản lý luật chơi, tương tác giữa người chơi và máy chủ, cũng như các yếu tố đồ họa và âm thanh để tạo ra trải nghiệm chân thực nhất cho người chơi.

Trong khóa luận này, chúng tôi tập trung vào việc xây dựng một game Poker trực tuyến sử dụng ngôn ngữ lập trình Python và các công nghệ web hiện đại như WebSocket và Flask. Chúng tôi sẽ trình bày cách thiết kế và triển khai hệ thống, cùng với các chi tiết về cách thức hoạt động của game và các tính năng mà chúng tôi đã thực hiện.

Hy vọng rằng khóa luận này sẽ cung cấp cho bạn cái nhìn tổng quan về quy trình phát triển một game Poker trực tuyến, cũng như cung cấp ý tưởng và kiến thức để bạn có thể tiếp tục nghiên cứu và phát triển sản phẩm của riêng mình trong tương lai.

LỜI CẢM ƠN

Chúng em xin gửi lời cảm ơn chân thành đến với giảng viên môn Phát triển phần mềm mã nguồn mở – ThS.Từ Lãng Phiêu, người đã nhiệt tình giảng dạy và hướng dẫn chúng em trong suốt quá trình học tập cũng như hoàn thiện bài báo cáo đề tài môn học này. Trong quá trình nghiên cứu và làm báo cáo đề tài, vì trình độ lý luận cũng như kinh nghiệm chuyên môn còn nhiều hạn chế nên bào báo cáo tiểu luận sẽ có sai sót, chúng em mong là sẽ nhận được ý kiến đóng góp, đánh giá từ thầy để có thể rút kinh nghiệm, học hỏi thêm để những đề tài tiếp theo được hoàn thiện hơn.

Chúng em xin chân thành cảm ơn thầy!

CHƯƠNG 1. TỔNG QUAN VỀ ĐỀ TÀI

1.1 Lịch sử phát triển

1.1.1 Lịch sử phát triển của game poker

Poker là game bài cá cược bằng tiền thật, nếu chơi trực tiếp tại sòng bài thì là tiền tươi còn nếu chơi trực tuyến tại các nhà cái thì là tiền nạp vào tài khoản rồi chơi.

Trò chơi này hiện nay đang thu hút được sự chú ý của nhiều người chơi bởi sức hút ma mị của nó, vừa là trò giải trí nhưng cũng vừa là một trò vô cùng hách não. Đến với game chơi này bạn sẽ được trải qua đủ cung bậc cảm xúc.

Đầu tiên là tò mò, khó hiểu, chán nản, buồn bực, bực tức cho tới vui sướng, mừng rỡ,...Người chơi đến với game này thì không bao giờ lo bị nhàm chán vì chơi mãi một trò.

Với những trường hợp bài khác nhau và với số lượng đối thủ đông đảo, bạn sẽ không bao giờ lo bị chơi trùng lặp kiểu, gặp mãi một đối thủ. Đây hoàn toàn không phải một game đỏ đen, dựa vào sự may mắn, phù hộ của thần linh để chiến thắng, muốn thắng game này cần tới sự vận dụng của trí não.

Game này không chỉ là một trong những game được yêu thích nhất hiện nay đâu mà nó còn có quá trình phát triển cực kì nhanh chóng.

Đầu tiên là được biết đến và truyền bá rộng rãi ở Mỹ sau đó là được lưu truyền, phổ biến thông qua đường miệng và sách vở những năm 1830. Đây là điểm nhấn đầu tiên trong lịch sử phát triển của game này.

Tuy nhiên đang trong quá trình phát triển thì game này vướng phải một số scandal liên quan đến việc dính dáng với các băng đảng nên mất tới 30 năm sau thì game này mới được người dân thực sự đón nhận và truyền bá.

Với độ phủ sóng của mình, ngày nay game này có rất nhiều các biến thể khác nhau nhưng thể loại phổ biến và được yêu thích nhất hiện nay đó là Texas Hold'em. Thể loại này chúng ta vẫn thường chơi ở các nhà cái và các sòng bài.

Năm 1970 với giải đấu WSOP tại Las Vegas đã đánh dấu bước ngoặt và sự phát triển lớn của game này. Tên tuổi của nó đã được vươn ra trên trường quốc tế, sánh vai với nhiều game khác.

Một dấu ấn nữa mà chúng ta không thể bỏ qua chính là sự vào cuộc của truyền thông những năm cuối thập niên 90 đã góp phần rất lớn đưa Poker đến với mọi người trên Thế

Giới và ngày nay được biết tới là một trong những game chơi trí tuệ được yêu thích và săn đón nhất.

1.1.2 Lịch sử phát triển của game đặt Bomb

Trò chơi đặt bomb, hoặc còn được gọi là bomberman, là một thể loại game phổ biến với nguồn gốc từ Nhật Bản. Dưới đây là một số giai đoạn quan trọng trong lịch sử phát triển của trò chơi này:

Nguồn gốc ban đầu (1983):

Bomberman ra đời vào năm 1983 do Hudson Soft phát triển và phát hành cho hệ máy MSX. Trong trò chơi này, người chơi điều khiển một nhân vật trong một mê cung và đặt bomb để phá hủy các vật thể và quái vật.

Sự phát triển và mở rộng (1985 - 1990):

Bomberman trở thành một phần không thể thiếu trong nhiều hệ máy game khác nhau, bao gồm NES, SNES, và Sega. Các phiên bản mới được phát triển với nhiều chế độ chơi đa người, nhiều loại vũ khí và môi trường mê cung đa dạng hơn.

Sự ra đời của Bomberman 64 (1997):

Bomberman 64 là một bước ngoặt lớn, đưa Bomberman từ góc nhìn trên xuống sang góc nhìn 3D. Trò chơi này cũng mở đầu cho nhiều phiên bản Bomberman 3D sau này.

Thế hệ mới và Bomberman Online (1999):

Bomberman tiếp tục phát triển trên các nền tảng mới như PlayStation và Nintendo 64. Phiên bản Bomberman Online cho Sega Dreamcast được phát hành, mở ra khả năng chơi trực tuyến.

Bomberman trên các hệ máy di động và console (2000 - nay):

- Bomberman tiếp tục mở rộng ra nhiều nền tảng khác nhau, bao gồm các phiên bản trên điện thoại di động và các hệ máy console hiện đại như PlayStation, Xbox, và Nintendo Switch.
- Các phiên bản Bomberman sau này tiếp tục đổi mới với chế độ chơi trực tuyến, nhiều chế độ chơi mới, và đồ họa cải tiến.

Từ sự xuất hiện đầu tiên trên MSX cho đến những phiên bản hiện đại trên các nền tảng console và di động, Bomberman đã trở thành một trong những trò chơi kinh điển và được yêu thích trên toàn thế giới.

1.2 Mục tiêu

Phát triển kỹ năng lập trình Python thông qua việc xây dựng game Poker và game Bomb đơn giản nhưng hấp dẫn.

Tiếp cận và hiểu cơ bản về lập trình game, bao gồm xử lý sự kiện, đồ họa, và tương tác người chơi. Khuyến khích sự sáng tạo thông qua việc tự do thiết kế và mở rộng trò chơi dựa trên ý tưởng cá nhân.

Áp dụng kiến thức đã học trong việc giải quyết vấn đề thực tế, từ việc xử lý vật lý đến tạo ra trải nghiệm người chơi tốt.

1.3 Phạm vi đề tài

1.3.1 Đối với game Poker

Triển khai các tính năng cơ bản của trò chơi Poker như phân phối bài, xác định thứ hạng của các tay, quy tắc chơi và các bước thực hiện trong mỗi vòng.

Thiết kế và phát triển giao diện người dùng cho trò chơi, bao gồm bàn cờ, các nút điều khiển, hiển thị thông tin về người chơi và các thẻ bài.

Phát triển trò chơi Poker trên nhiều nền tảng như ứng dụng di động (Android, iOS), ứng dụng web hoặc các nền tảng khác nếu có yêu cầu.

1.3.2 Đối với game đặt Bomb

Nghiên cứu về cơ chế gameplay trong trò chơi đặt bom, bao gồm cách di chuyển, đặt bom, và chiến lược.

Triển khai, tiết kế và phát triển giao diện người dùng cho trò chơi, bao gồm trận địa game, các nút điều khiển, hiển thị thông tin về người chơi và các đối tượng, vật thể,...

Phát triển trò chơi đặt bomb trên nhiều nền tảng như ứng dụng di động (Android, iOS), ứng dụng web hoặc các nền tảng khác nếu có yêu cầu.

1.4 Kết hoạch triển khai

- Sử dụng nền tảng công nghệ: Python.

- Môi trường phát triển: Visual Studio Code.

- Công cụ quản lý mã nguồn như Git.

- Sử dụng một số Framework như: Pygame để lập trình logic game, Socket để kết nối các người chơi lại với nhau,...

- Phát triển các tính năng chính của game Poker:

+ Tạo phòng (tạo nhiều phòng)

+ Vào phòng

+ Chế độ chơi nhiều người

+ Chức năng phân lượt

* Các chức năng nâng cao: đáng bài poker kiểu call, fold, reise, xác định thắng thua, chung tiền cược,...

- Phát triển các tính năng chính của game đặt Bomb:

+ Bắt đầu game

+ Đợi người chơi

+ 2 người chơi

+ Chức năng đặt bomb

+ Phá vật thể

+ Phân thắng thua

- Tối ưu hóa giao diện người dùng, đảm báo trải nghiệm về game cho người chơi.

CHƯƠNG 2. GIỚI THIỆU VỀ PYGAME, SOCKET VÀ CÁC ỨNG DỤNG, CÔNG NGHỆ

2.1 Giới thiệu về Pygame

2.1.1 Khái niệm về Pygame

Pygame là một thư viện Python được sử dụng để phát triển trò chơi và ứng dụng đa phương tiện. Dựa trên thư viện SDL (Simple DirectMedia Layer), Pygame cung cấp các công cụ và chức năng cần thiết để tạo ra các ứng dụng đồ họa đơn giản đến phức tạp.

Dưới đây là một số điểm quan trọng về Pygame:

Đồ Họa và âm Thanh: Pygame cung cấp các chức năng mạnh mẽ để vẽ đồ họa và xử lý âm thanh. Bạn có thể tạo các cửa sổ đồ họa, vẽ hình ảnh, quản lý sprite, và phát lại âm thanh một cách dễ dàng.

Xử lý sự kiện: Pygame giúp bạn quản lý và xử lý sự kiện người dùng như nhấn phím, di chuyển chuột, và các sự kiện khác. Điều này rất quan trọng khi xây dựng trò chơi tương tác.

Đối tượng Sprite: Pygame có hỗ trợ mô hình đối tượng Sprite, giúp tổ chức và quản lý đối tượng trong trò chơi của bạn.

Dễ học và sử dụng: Pygame được thiết kế để dễ học và sử dụng, đặc biệt là cho những người mới bắt đầu với lập trình trò chơi.

Di Động và Nền Tảng Đa Dạng: Pygame có khả năng chạy trên nhiều nền tảng, bao gồm Windows, macOS và Linux. Điều này giúp bạn phát triển ứng dụng của mình một cách linh hoạt.

Ngoài ra còn nhiều hàm hỗ trợ khác giúp giảm độ phức tạp khi phát triển trò chơi. Mỗi trò chơi có thể có cấu trúc và logic riêng, nhưng các bước cơ bản trên đây giúp bạn hiểu cách Pygame hoạt động.

2.1.2 Lịch sử hình thành và phát triển pygame

Pygame ban đầu được viết bởi Pete Shinners để thay thế PySDL sau khi quá trình phát triển của nó bị đình trệ.

Khởi Đầu (2000):

Pygame được tạo ra bởi Pete Shinners vào năm 2000. Ban đầu, nó là một dự án cá

nhân của Shinners với mục đích tạo ra một thư viện dễ sử dụng cho việc phát triển trò chơi với ngôn ngữ Python.

Dự Án Mã Nguồn Mở:

Pygame nhanh chóng trở thành một dự án mã nguồn mở và được công bố dưới giấy phép LGPL (GNU Lesser General Public License). Điều này có nghĩa là bất kỳ ai đều có thể sử dụng, sửa đổi và phân phối mã nguồn Pygame mà không cần phải công bố mã nguồn của trò chơi của họ.

Sự Phát Triển Trong Cộng Đồng (2000 - Nay):

Pygame ngày càng trở nên phổ biến trong cộng đồng Python, đặc biệt là đối với những người mới bắt đầu làm quen với lập trình game. Nó được sử dụng rộng rãi để tạo ra các trò chơi đơn giản, các ứng dụng đa phương tiện và các dự án sáng tạo khác.

Cập Nhật và Bảo Trì Liên Tục:

Cộng đồng người dùng và những người đóng góp vào dự án đã giữ cho Pygame được cập nhật và bảo trì liên tục. Các phiên bản mới đưa vào những tính năng và cải tiến mới, giúp thư viện thích nghi với các tiêu chuẩn và công nghệ mới.

Tích Hợp Với Các Công Nghệ Khác:

Pygame không chỉ là một thư viện đơn lẻ mà còn thường được tích hợp với các công nghệ và thư viện khác trong cộng đồng Python. Việc này mở rộng khả năng của Pygame và giúp nó tích hợp tốt với nhiều dự án Python khác nhau.

Sự Hỗ Trợ Đa Nền Tảng:

Pygame được thiết kế để hỗ trợ đa nền tảng, cho phép người phát triển tạo ra trò chơi có thể chạy trên nhiều hệ điều hành khác nhau mà không cần phải viết lại mã nguồn.

2.1.3 Ưu – nhược điểm của pygame

- ***Ưu điểm***:

Dễ học và sử dụng: Pygame làm cho việc phát triển trò chơi trở nên dễ dàng hơn cho những người mới bắt đầu với lập trình game hoặc Python. Cú pháp của nó thân thiện với người dùng và có nhiều tài nguyên học trực tuyến.

Cộng đồng lớn và hỗ trợ: Cộng đồng người sử dụng Pygame rất lớn, và có nhiều tài nguyên hỗ trợ trực tuyến, diễn đàn, và ví dụ mã nguồn.

Mã nguồn mở và linh hoạt: Pygame là một dự án mã nguồn mở, cho phép người dùng sửa đổi và tùy chỉnh mã nguồn theo nhu cầu của họ.

Tương thích đa nền tảng: Pygame hỗ trợ nhiều hệ điều hành khác nhau, bao gồm Windows, macOS và Linux.

Sử dụng cùng Python: Pygame là một phần của ngôn ngữ lập trình Python, điều này giúp người lập trình tận dụng những ưu điểm của Python như đơn giản, dễ đọc, và nhanh chóng.

- ***Nhược điểm***:

Hiệu suất không cao: So với một số thư viện và ngôn ngữ lập trình game khác, Pygame có thể có hiệu suất không cao đối với các trò chơi phức tạp và đòi hỏi nhiều tài nguyên.

Thiếu các tính năng nâng cao: Pygame không cung cấp nhiều tính năng nâng cao mà một số dự án lớn và phức tạp có thể yêu cầu. Điều này có thể làm giảm khả năng mở rộng của nó trong một số trường hợp.

Khả năng tương thích hạn Chế: Pygame có thể không phải là lựa chọn tốt nhất cho các dự án yêu cầu tích hợp sâu với các công nghệ và thư viện khác ngoài Python.

Đang Trong Quá Trình Phát Triển (nếu có): Mặc dù Pygame có một cộng đồng đông đảo, nhưng đôi khi nó vẫn đang trong quá trình phát triển, điều này có thể tạo ra sự không ổn định trong các phiên bản cụ thể.

2.2 Giới thiệu về Socket

2.2.1 Khái niệm về Socket

Socket là điểm cuối (end-point) trong liên kết truyền thông hai chiều (two-way communication) biểu diễn kết nối giữa máy khách – máy chủ (Client – Server).

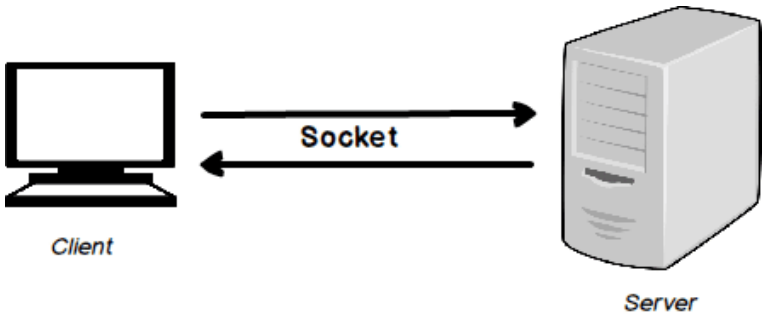
Các lớp Socket được ràng buộc với một cổng port (thể hiện qua con số cụ thể) để các tầng TCP (TCP Layer) định danh ứng dụng mà dữ liệu sẽ được gửi tới.

Socket là giao diện lập trình ứng dụng mạng được dùng để truyền và nhận dữ liệu trên internet. Giữa hai chương trình chạy trên mạng cần có một liên kết giao tiếp hai chiều, hay còn gọi là two-way communication nhằm kết nối 2 process trò chuyện với nhau. Điểm cuối (endpoint) của liên kết giữa 2 process này được gọi là Socket.

2.2.2 Vai trò của Socket

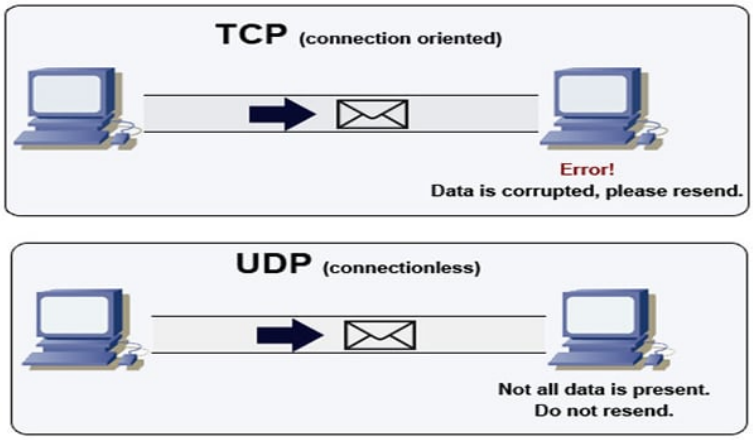
Nhiều socket được sử dụng liên tục để tiết kiệm thời gian và nâng cao hiệu suất làm việc.

Ưu điểm lớn nhất của socket là hỗ trợ hầu hết các hệ điều hành bao gồm MS Windows, Linux,... Ngoài ra, nó cũng được sử dụng với nhiều ngôn ngữ lập trình, gồm C, C++, Java,



Visual Basic hay Visual C++,... Vì thế nó tương thích với hầu hết mọi đối tượng người dùng với những cấu hình máy khác nhau.

2.2.3 Cách thức hoạt động của Socket



Chức năng của socket là kết nối giữa máy khách và máy chủ thông qua TCP/IP và UDP để truyền và nhận dữ liệu qua Internet. Giao diện lập trình ứng dụng mạng này chỉ có thể hoạt động khi biết thông số IP và số hiệu cổng của 2 ứng dụng cần trao đổi dữ liệu cho nhau.

Điều kiện cần giữa 2 ứng dụng truyền thông tin để socket hoạt động:

- 2 ứng dụng nằm cùng trên một máy hoặc 2 máy khác nhau.
- Trường hợp 2 ứng dụng nằm trên cùng một máy, số hiệu cổng không trùng nhau.

Socket được chia làm 4 loại khác nhau, cụ thể:

- Stream Socket

- Datagram Socket
- Websocket
- Unix socket

2.2.4 Lịch sử phát triển của Socket

Lịch sử phát triển của socket được bắt đầu từ những năm đầu của Internet, khi mà nhu cầu kết nối và truyền thông giữa các máy tính trở nên quan trọng. Dưới đây là một số giai đoạn quan trọng trong lịch sử phát triển của socket:

Đầu những năm 1970: Trong giai đoạn này, ARPANET (Advanced Research Projects Agency Network) đã phát triển các giao thức ban đầu cho mạng Internet. Giao thức đầu tiên được sử dụng là NCP (Network Control Protocol), và sau đó là TCP (Transmission Control Protocol) và IP (Internet Protocol).

Giai đoạn đầu của TCP/IP (1970-1980): Các giao thức TCP/IP được phát triển và trở thành tiêu chuẩn cho việc truyền thông trên mạng. Trong giai đoạn này, TCP đã giới thiệu khái niệm về "cổng" (port) để định danh ứng dụng trên máy tính.

BSD Unix (1980s): Các phiên bản của BSD Unix (Berkeley Software Distribution) đã bổ sung và cải tiến thêm các tính năng cho TCP/IP stack, bao gồm cơ chế socket. BSD socket API (Application Programming Interface) đã trở thành một chuẩn de facto cho việc phát triển ứng dụng mạng.

Sự phát triển của Internet (1990s): Internet đã trở nên phổ biến hơn, và sự phát triển của các ứng dụng truyền thông như email, web và chat đã tạo nên nhu cầu lớn hơn cho việc sử dụng socket. Các ngôn ngữ lập trình như C và C++ đã có thư viện socket hỗ trợ.

Java (1990s-2000s): Java có một thư viện socket tích hợp trong Java API, cho phép các nhà phát triển dễ dàng tạo ra các ứng dụng mạng. Điều này đã giúp socket trở nên phổ biến hơn trong cộng đồng phát triển.

Phát triển các giao thức và công nghệ mới: Cùng với sự phát triển của Internet, có nhiều giao thức và công nghệ mới được phát triển như UDP (User Datagram Protocol), WebSocket, và các framework như Node.js cung cấp các cách tiếp cận khác nhau cho việc xử lý kết nối mạng.

Hiện đại (từ năm 2000 đến nay): Socket vẫn là một phần quan trọng của lập trình mạng ngày nay. Các ngôn ngữ lập trình hiện đại như Python, JavaScript, và Ruby cũng cung cấp các thư viện socket mạnh mẽ để xây dựng các ứng dụng mạng phức tạp.

2.2.5 Ưu - nhược điểm của Socket

- ***Ưu điểm của socket***:

Đơn giản và linh hoạt: Sockets cung cấp một giao diện lập trình đơn giản cho việc truyền thông giữa các máy tính trên mạng. Các ứng dụng có thể dễ dàng gửi và nhận dữ liệu thông qua các kết nối socket.

Hiệu suất cao: Sockets cho phép truyền dữ liệu trực tiếp giữa các máy tính mà không cần thông qua các tầng giao thức trung gian. Điều này giúp giảm thiểu độ trễ và tăng hiệu suất.

Hỗ trợ nhiều giao thức: Sockets không chỉ hỗ trợ TCP mà còn hỗ trợ UDP và các giao thức khác, cho phép phát triển ứng dụng với nhiều mô hình truyền thông khác nhau.

Cross-platform: Sockets được hỗ trợ trên nhiều hệ điều hành và nền tảng phần cứng, từ máy tính cá nhân đến thiết bị nhúng.

Tích hợp trong các ngôn ngữ lập trình: Các ngôn ngữ lập trình phổ biến như C/C++, Java, Python, JavaScript đều cung cấp các API hoặc thư viện cho việc sử dụng socket.

- ***Nhược điểm của socket***:

Khó sử dụng: Socket có thể khá phức tạp và khó hiểu đối với người mới bắt đầu. Việc xử lý lỗi và quản lý kết nối có thể gặp phải nhiều thách thức.

Cần phải tự xử lý nhiều vấn đề: Socket đòi hỏi người phát triển phải tự xử lý các vấn đề như quản lý kết nối, bảo mật và đồng bộ hóa dữ liệu.

Thiếu tính linh hoạt: Socket không cung cấp một số tính năng cao cấp như đa luồng (multithreading) hoặc quản lý trạng thái kết nối.

Khả năng xảy ra lỗi: Trong môi trường mạng không ổn định, việc sử dụng socket có thể dẫn đến các vấn đề như mất kết nối hoặc dữ liệu bị mất.

2.3 Các ứng dụng, công nghệ và công cụ thực hiện

2.3.1 Các ứng dụng, công nghệ

Pygame: Pygame là một thư viện mã nguồn mở trong Python được thiết kế để hỗ trợ việc phát triển game 2D. Nó cung cấp nhiều chức năng và tính năng cho việc vẽ đồ họa, xử lý sự kiện, và âm thanh, giúp bạn tập trung vào logic game.

Panda: Panda là một engine game mã nguồn mở cung cấp nền tảng cho phát triển game. Nó có thể sử dụng Python làm ngôn ngữ kịch bản và hỗ trợ nhiều tính năng như xử lý đồ họa, âm thanh, và vật lý.

IDE (Integrated Development Environment): Sử dụng một IDE như VSCode giúp tăng cường trải nghiệm phát triển bằng cách cung cấp tính năng như gợi ý mã nguồn, gỡ lỗi thuận tiện, và quản lý dự án hiệu quả.

Tìm Hiểu về Thư viện âm thanh và hình ảnh: Sử dụng thư viện như Pygame Mixer để xử lý âm thanh và thư viện PIL (Python Imaging Library) hoặc Pillow để xử lý hình ảnh.

Git và GitHub: Sử dụng Git để theo dõi phiên bản và GitHub để lưu trữ mã nguồn của dự án. Điều này giúp bạn duy trì lịch sử của dự án và cung cấp khả năng làm việc nhóm.

2.3.2 Công cụ thực hiện

Sử dụng Visual Studio Code (VSCode) để thực hiện đề tài "Xây dựng game đua xe".

Một số đặc điểm của VSCode bao gồm:

Hỗ trợ Nhiều Ngôn Ngữ: VSCode hỗ trợ nhiều ngôn ngữ lập trình, bao gồm Python, JavaScript, Java, C++, và nhiều ngôn ngữ khác.

Gỡ lỗi Tích hợp: Cung cấp khả năng gỡ lỗi tích hợp cho nhiều ngôn ngữ, giúp bạn dễ dàng theo dõi và sửa lỗi trong mã nguồn của mình.

Mở Rộng và Cộng đồng: VSCode hỗ trợ các extension mở rộng, cho phép bạn tùy chỉnh và mở rộng chức năng của nó. Cộng đồng VSCode cũng rất lớn và tích cực, có nhiều extension và theme sẵn có.

Quản lý Dự án: Cung cấp các tính năng quản lý dự án như Git Integration để theo dõi thay đổi mã nguồn và quản lý phiên bản.

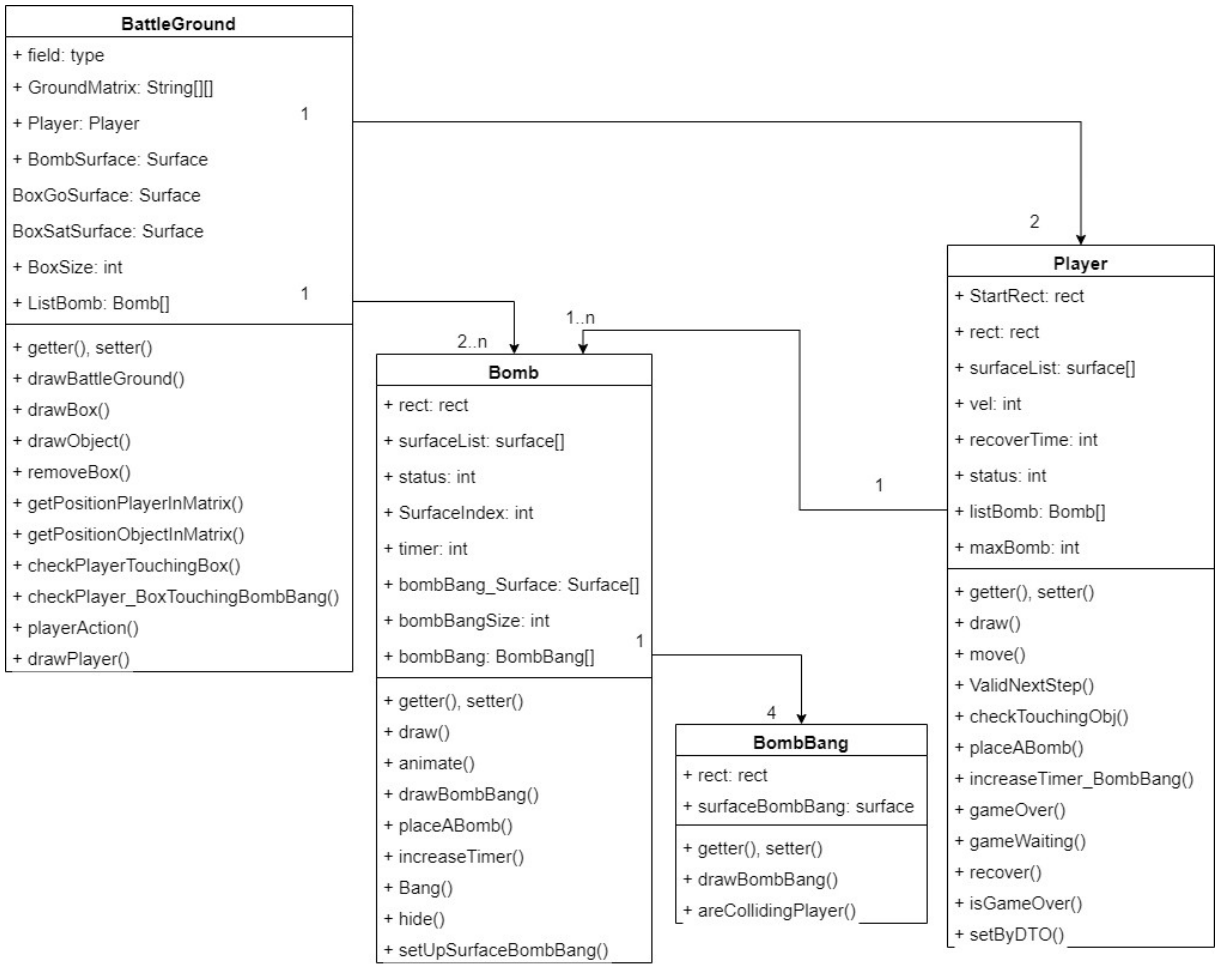
IntelliSense: Hỗ trợ tính năng gợi ý mã nguồn thông minh để tăng tốc quá trình viết mã và giảm lỗi.

Đa Nền Tảng: VSCode có sẵn trên nhiều hệ điều hành như Windows, macOS, và Linux.

Và còn nhiều tính năng khác, VSCode thường được lựa chọn cho nhiều loại dự án phần mềm và lập trình, bao gồm cả phát triển game bằng Python.

CHƯƠNG 3. PHÂN TÍCH THIẾT KẾ

3.1 Sơ đồ class Game đặt Boom

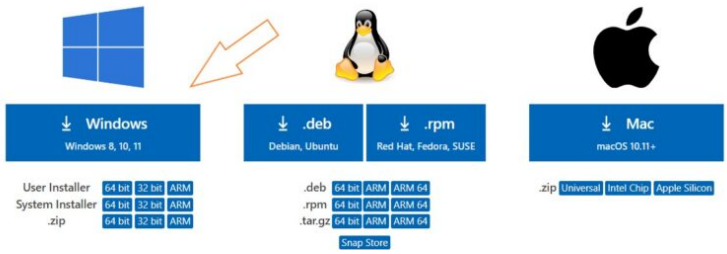


CHƯƠNG 4. CÀI ĐẶT MÔI TRƯỜNG

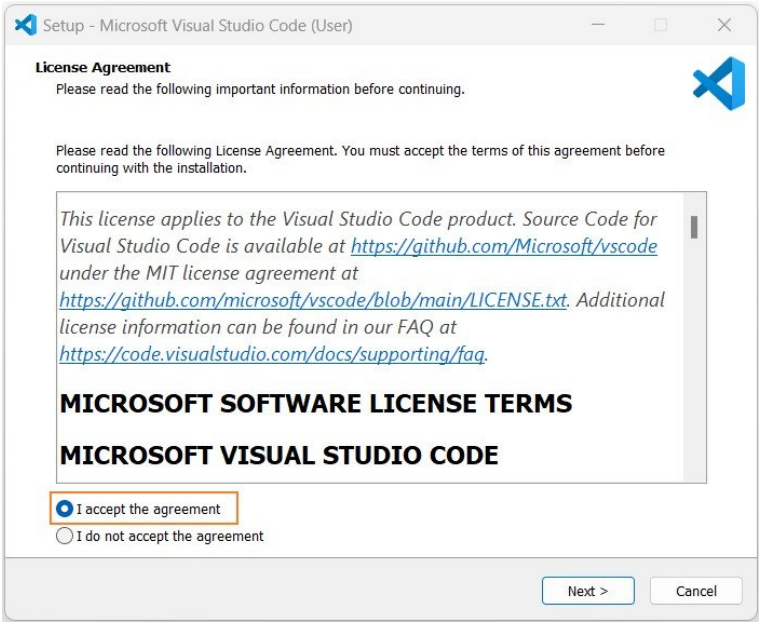
4.1 Cài đặt Visual Studio Code

Visual Studio Code là một trình soạn thảo source code được phát triển bởi Microsoft. Nó rất gọn nhẹ, hỗ trợ nhiều ngôn ngữ lập trình và có các chức năng thú vị như debugging, syntax highlighting, intelligent code completion, snippets, code refactoring và embedded Git.

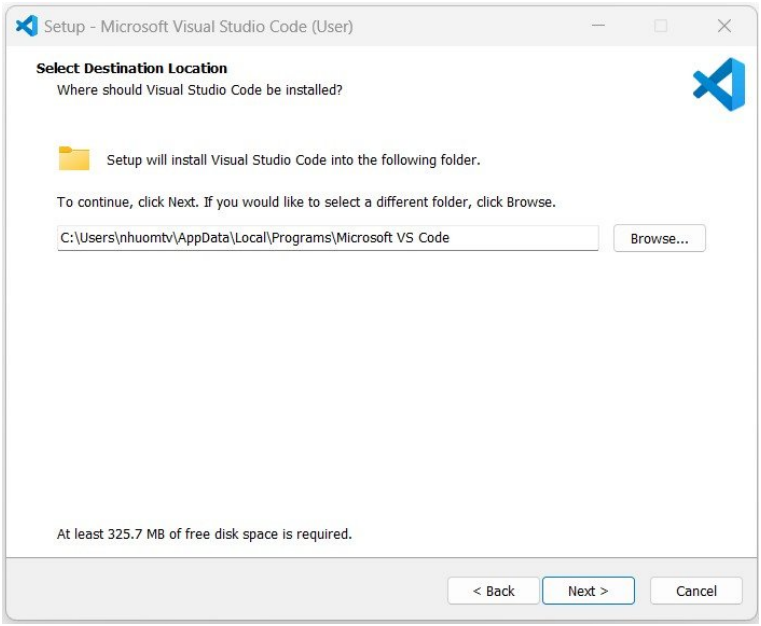
Bước 1: Các bạn download Visual Studio Code



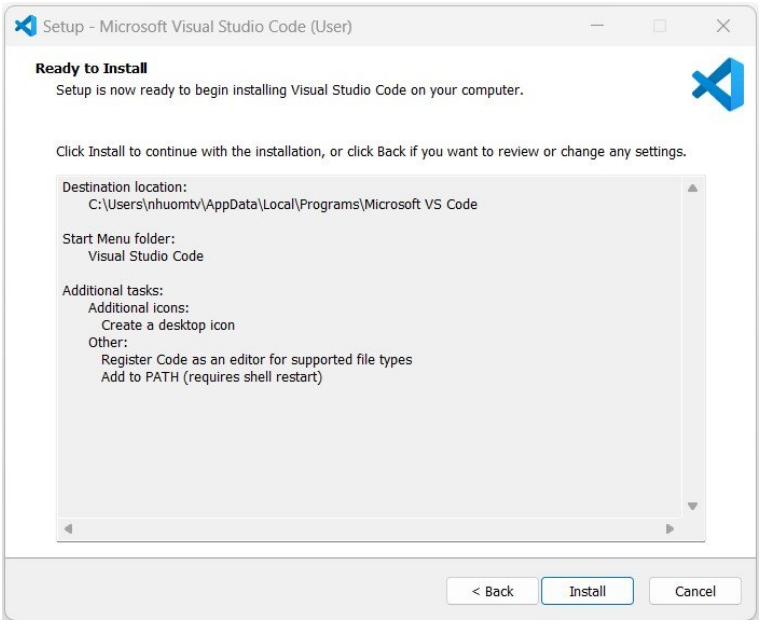
Bước 2: Các bạn double click vào file cài đặt Visual Studio Code để bắt đầu cài đặt.



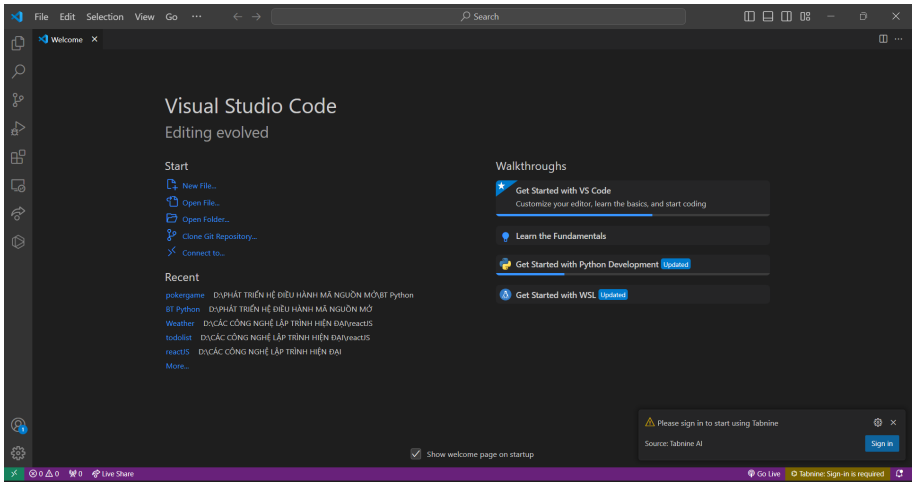
Bước 3: Nhấn Next rồi chọn đường dẫn cài đặt Visual Studio Code.



Bước 4: Nhấn Install để cài đặt Visual Studio Code.



Bước 5: Sau khi cài đặt thành công Visual Studio Code có giao diện như sau:



4.2 Thiết lập môi trường lập trình Python với Visual Studio Code

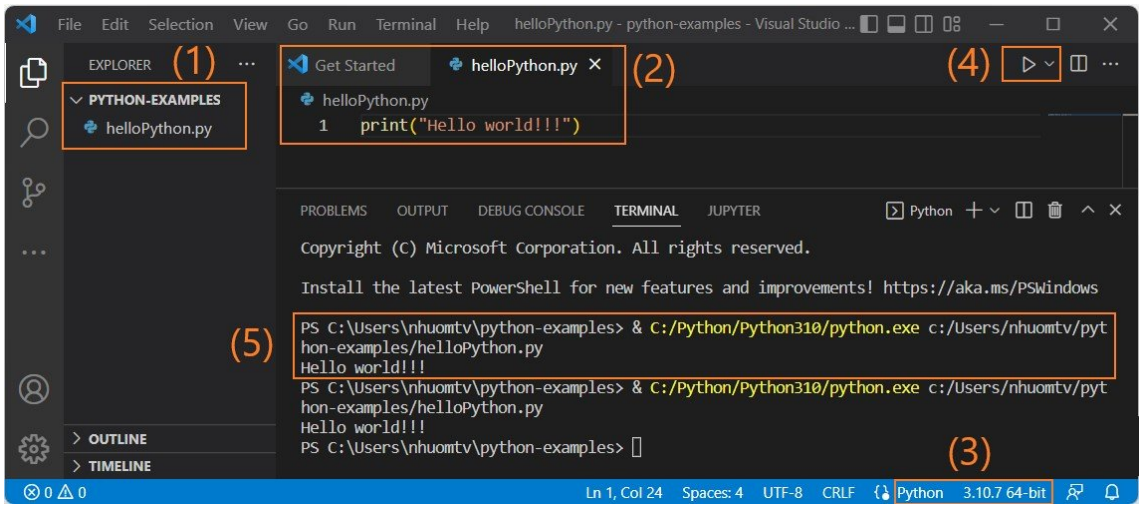
Để lập trình Python với Visual Studio Code, chúng ta cần cài extension Python cho Visual Studio Code. Extension Python được phát hành bởi Microsoft giúp tự động nhận biết trình thông dịch Python, hỗ trợ gợi nhớ code, gợi ý lỗi, debug, format code,...

Để cài đặt extension Python trong Visual Studio Code, các bạn tìm đến phần Extensions hoặc nhấn tổ hợp phím Ctrl+Shift+X rồi gõ python và tiến hành cài đặt extension Python.

Bước 1:

Trong Visual Studio Code, chúng ta tìm đến phần Explorer hoặc nhấn tổ hợp phím Ctrl+Shift+E để open một folder mà các bạn sẽ tạo các file source code Python. Trong ví dụ bên dưới, một folder python-examples trong phân vùng C được tạo và chứa các file có đuôi mở rộng là .py. Các file này là các file chứa source code của Python.

Bước 2:



(1) – Khu vực các file source code của Python.

(2) – Khu vực soạn thảo source của Python.

(3) – Trình thông dịch sẽ tự động được nhận diện khi đang mở file .py

(4) – Nhấn để thực thi source code của Python.

(5) – Khu vực Terminal để hiển thị kết quả chương trình Python.

Và cài thêm thư viện Pygame (pip install pygame) và các thư viện cần thiết khác bằng terminal.

CHƯƠNG 5. XÂY DỰNG VÀ PHÁT TRIỂN GAME POKER VÀ GAME ĐẶT BOOM

5.1 Xây dựng Game Poker

5.1.1 In ra tên của các lá bài trên bàn sau khi ra Flop

```
def print_name_of_flop(flop):
    print("-" * 30)
    return (f"Flop: {[card.name for card in flop]}")
```

Hàm print_name_of_flop nhận đầu vào là danh sách flop chứa các lá bài trên bàn sau khi ra Flop trong trò chơi Poker.

In ra một dòng gạch ngang có độ dài 30 ký tự để tạo đường phân cách trước khi in tên các lá bài.

flop: Là danh sách các lá bài trên bàn sau khi ra Flop.

Tạo một danh sách mới chứa tên của mỗi lá bài trong flop.

Trả về một chuỗi kết quả có dạng "Flop: [tên của các lá bài trong flop]" để in ra tên của các lá bài trên bàn sau khi ra Flop.

5.1.2 In ra vòng đấu hiện tại

```
def print_current_round(round_number):
    if round_number in poker_rounds:
        return f"Current Round:  {poker_rounds[round_number]}"
    else:
        return ("Invalid round number.")
```

Hàm print_current_round nhận đầu vào là round_number, là số thứ tự của vòng đặt cược hiện tại trong trò chơi Poker.

round_number: Số thứ tự của vòng đặt cược hiện tại.

poker_rounds: Một từ điển lưu trữ thông tin về các vòng đặt cược trong trò chơi Poker.

Hàm này thực hiện các bước sau:

- Kiểm tra xem round_number có tồn tại trong poker_rounds không.
- Nếu round_number tồn tại trong poker_rounds, hàm trả về một chuỗi thông báo về vòng đặt cược hiện tại dựa vào giá trị của round_number.
- Nếu round_number không tồn tại trong poker_rounds, hàm trả về một thông báo "Invalid round number." để thông báo rằng số vòng đặt cược không hợp lệ.

5.1.3 In ra các lựa chọn hành động của người chơi và thông tin về tay bài và số tiền của họ

```
def print_action(player, current_bet):
    s = ""
    if current_bet == 0:
        s = "1.Bet  2.Fold  3.Check \n"
    else:
        s = "1.Call  2.Raise  3.Fold \n"
    s += player.print_name_of_hand() + (f"Your Money: {player.money}")
    return s
```

Hàm print_action nhận hai tham số đầu vào là player và current_bet. Dưới đây là giải thích từng phần trong hàm:

player: Đối tượng người chơi trong trò chơi Poker.

current_bet: Giá trị của cược hiện tại trong ván đấu.

Đối với hàm này:

- Nếu current_bet == 0, tức là đến lượt người chơi đặt cược, hàm sẽ trả về chuỗi lựa chọn cho người chơi, bao gồm "1.Bet", "2.Fold", và "3.Check".
- Sau đó, hàm sẽ thêm vào chuỗi s thông tin về bộ bài của người chơi và số tiền mà người chơi đó có.
- Cuối cùng, hàm trả về chuỗi s đó, chứa các lựa chọn và thông tin về bộ bài và số tiền của người chơi.

5.1.4 Khởi tạo một vòng chơi Poker mới

```
def game(player_list):
    deck = Deck()
    deck.shuffle()
    small_blind = 5
    global state
    player_in_room = list(player_list)
    state = game_state(deck, reset_players_hand(player_list), small_blind * 2)
    dealCards(player_list, state.deck)
    state.pot = make_pot(state.player_list,state.first_index, small_blind, state.pot)
```

Hàm game(player_list) được thiết kế để thực hiện các bước chuẩn bị cho một vòng chơi mới trong trò chơi Poker, bao gồm tạo bộ bài mới, xáo trộn nó, phân phát các quân bài cho các người chơi, và tạo pot cho vòng chơi.

Tạo một bộ bài (deck) mới, chưa xáo trộn.

deck.shuffle(): Xáo trộn bộ bài để chuẩn bị cho ván chơi mới.

small_blind = 5: Xác định giá trị cược mù nhỏ (small blind) là 5 (giả sử).

global state: Khai báo biến state là một biến toàn cục, có thể truy cập từ bất kỳ đâu trong chương trình.

player_in_room = list(player_list): Tạo một danh sách mới sao chép từ player_list, danh sách này chứa các người chơi trong phòng.

Gọi hàm game_state() để khởi tạo trạng thái của trò chơi với các thông tin cần thiết như bộ bài, bộ bài của các người chơi đã được đặt lại (reset), và giá trị cược mù lớn (big blind). Kết quả của hàm này được gán cho biến state.

dealCards(player_list, state.deck): Phân phát các quân bài cho các người chơi trong player_list từ bộ bài đã được xáo trộn state.deck.

state.pot = make_pot(state.player_list,state.first_index, small_blind, state.pot): Tạo pot (nơi chứa các phần cược) cho ván chơi hiện tại bằng cách gọi hàm make_pot() với các tham số như danh sách người chơi, chỉ số người chơi đầu tiên, giá trị cược mù nhỏ, và pot hiện tại. Kết quả được gán lại vào biến state.pot.

5.1.5 Đặt tiền cược

```
def put_money_into_pot(state):
        for player in state.player_list:
            if player.initial_bet!=-1:
                state.pot += player.initial_bet
                player.initial_bet=0
```

Hàm này được sử dụng để đặt tiền cược của các người chơi vào nồi tiền của trò chơi.

Duyệt qua danh sách các người chơi trong trạng thái hiện tại.

Nếu người chơi đã đặt cược ban đầu và chưa fold (initial_bet khác -1), thì số tiền đó sẽ được thêm vào nồi tiền của trò chơi (state.pot).

Sau đó, cược ban đầu của người chơi sẽ được thiết lập lại về 0.

5.1.6 Thiết lập lại cược ban đầu của người chơi

```
def reset_players_bet(player_list):
    for player in player_list:
        player.initial_bet = 0
    return player_list
```

Hàm reset_players_bet được thiết kế để đặt lại giá trị cược ban đầu của tất cả các người chơi trong danh sách player_list. Dưới đây là các bước được thực hiện trong hàm:

Duyệt qua danh sách người chơi (player_list): Hàm duyệt qua mỗi phần tử trong danh sách player_list, mỗi phần tử đại diện cho một người chơi.

Đặt lại cược ban đầu của mỗi người chơi: Đối với mỗi người chơi, hàm đặt giá trị của thuộc tính initial_bet (cược ban đầu) về 0. Điều này đảm bảo rằng sau khi gọi hàm, tất cả các người chơi sẽ có giá trị cược ban đầu là 0.

Trả về danh sách người chơi đã được cập nhật: Sau khi hoàn thành việc đặt lại giá trị cược ban đầu của tất cả các người chơi trong danh sách, hàm trả về danh sách player_list đã được cập nhật.

5.1.7 Thiết lập lại bộ bài của người chơi

```
def reset_players_hand(player_list):
    for player in player_list:
        player.hand = []
    return player_list
```

Hàm reset_players_hand được sử dụng để đặt lại bộ bài của tất cả các người chơi trong danh sách player_list về trống.

Tham số đầu vào của hàm, là danh sách các người chơi trong trò chơi.

Vòng lặp duyệt qua từng phần tử trong danh sách người chơi.

Gán bộ bài của người chơi về một danh sách rỗng, tức là loại bỏ tất cả các lá bài mà người chơi đó đã có.

Trả về danh sách các người chơi sau khi đã đặt lại bộ bài của họ về trạng thái trống.

5.1.8 Thiết lập lại bộ bài của người chơi

```
def send_available_action(player, player_list, state):
    s = f"{player.name} TURN\n"
    if state.round != 0:
        s += print_name_of_flop(state.flop) + "\n"
    s += print_current_round(state.round) + "\n"
    s += print_action(player, state.current_bet) + "\n"
    s += (f"Current bet: {state.current_bet}") + "\n"
    s += (f"Current pot: {state.pot}") + "\n"
    player.conn.sendall(s.encode())
    if player.host:
        choice = player.conn.recv(1024).decode().split()[2]
        process_action(player, player_list, choice)
```

Hàm send_available_action được sử dụng để gửi các lựa chọn hành động có sẵn cho một người chơi cụ thể trong trò chơi Poker.

Hàm bắt đầu bằng việc tạo một chuỗi thông điệp s bằng cách kết hợp tên của người chơi với từ "TURN", chỉ ra rằng đến lượt của người chơi đó.

Thêm thông tin về bài Flop (nếu có): Nếu state.round khác 0 (nghĩa là Flop đã được ra), hàm thêm thông tin về bài Flop bằng cách gọi hàm print_name_of_flop(state.flop) và thêm vào chuỗi s.

Thêm thông tin về vòng đặt cược hiện tại: Hàm thêm thông tin về vòng đặt cược hiện tại bằng cách gọi hàm print_current_round(state.round) và thêm vào chuỗi s.

Thêm thông tin về hành động có sẵn cho người chơi: Hàm thêm thông tin về các hành động có sẵn cho người chơi bằng cách gọi hàm print_action(player, state.current_bet) và thêm vào chuỗi s.

Thêm thông tin về cược hiện tại và số tiền trong pot: Hàm thêm thông tin về cược hiện tại (state.current_bet) và số tiền trong pot (state.pot) vào chuỗi s.

Gửi thông điệp cho người chơi: Sau khi tạo chuỗi thông điệp s, hàm gửi chuỗi này cho người chơi thông qua kết nối của họ (player.conn).

Nhận và xử lý hành động từ người chơi host (nếu có): Nếu người chơi là host, hàm nhận hành động từ người chơi thông qua kết nối của họ, sau đó gọi hàm process_action để xử lý hành động đó.

Hàm này là một phần quan trọng của quá trình giao tiếp giữa máy chủ và người chơi trong trò chơi Poker, cho phép người chơi thực hiện các hành động của mình dựa trên trạng thái hiện tại của trò chơi.

5.1.9 Thiết lập lại bộ bài của người chơi

```
def processing_game():
    global state

    if len(state.flop)==0:
        flop_deal(state.flop, state.deck)
        put_money_into_pot(state)
        state.current_bet = 0
    elif(len(state.flop)==5):
        winner = showdown(state.player_list, state.flop)
        if type(winner) is list:
            for player in winner:
                player.money += state.pot // len(winner)
        else:
            winner.money += state.pot
        state.winner=winner
    else:
        state.flop.append(state.deck.draw_card())
        put_money_into_pot(state)
        state.current_bet=0
```

Hàm processing_game được sử dụng để xử lý quá trình diễn ra của trò chơi Poker. Dưới đây là giải thích từng bước của hàm:

Kiểm tra xem Flop có rỗng không: Đầu tiên, hàm kiểm tra xem danh sách state.flop có rỗng không. Nếu không, nghĩa là bàn đã ra bài Flop.

- Nếu Flop rỗng, hàm tiến hành chia bài Flop, đặt tiền vào pot và thiết lập current_bet về 0.
- Nếu không, hàm tiếp tục sang bước tiếp theo.

Kiểm tra xem đã ra cả Flop chưa: Nếu số lượng lá bài trên bàn (Flop) đã đủ 5 lá, hàm tiến hành xử lý trường hợp đến showdown (kết thúc ván đấu).

- Hàm gọi hàm showdown để xác định người chơi chiến thắng.
- Nếu có nhiều người chơi chiến thắng, tiền trong pot sẽ được chia đều cho các người chơi đó.

- Nếu chỉ có một người chiến thắng, toàn bộ pot sẽ được trao cho người đó.
- Sau đó, hàm gán người chiến thắng vào thuộc tính state.winner.

Nếu chưa ra đủ Flop: Nếu số lượng lá bài trên bàn chưa đủ 5 lá, hàm tiếp tục ra thêm 1 lá bài trên bàn, đặt tiền vào pot và thiết lập current_bet về 0.

Hàm này chịu trách nhiệm quản lý các bước tiến của trò chơi, bao gồm việc chia bài, đặt cược, và xác định người chiến thắng.

5.1.10 Thiết lập lại bộ bài của người chơi

```
def make_pot(player_list,first_index, small_blind, pot):
    small = player_list[first_index]
    big = player_list[first_index+1]
    small.money -= small_blind
    big.money -= small_blind * 2
    small.initial_bet = small_blind
    big.initial_bet = small_blind * 2
    pot += small_blind * 3
    return pot
```

Hàm make_pot được sử dụng để tạo và cập nhật pot (kho chứa tiền cược) cho một vòng đặt cược trong trò chơi Poker.

player_list: Danh sách các người chơi trong trò chơi.

first_index: Chỉ mục của người chơi đặt cược nhỏ (small blind) trong danh sách người chơi.

small_blind: Giá trị tiền cược nhỏ.

pot: Giá trị hiện tại của pot.

Người chơi đặt cược nhỏ là người chơi đứng ở vị trí đầu tiên trong danh sách người chơi.

Người chơi đặt cược lớn là người chơi đứng ở vị trí tiếp theo sau người chơi đặt cược nhỏ trong danh sách người chơi.

Trừ tiền cược nhỏ từ số tiền của người chơi đặt cược nhỏ.

Trừ tiền cược lớn từ số tiền của người chơi đặt cược lớn (thường là gấp đôi của tiền cược nhỏ).

Gán giá trị tiền cược ban đầu cho người chơi đặt cược nhỏ.

Gán giá trị tiền cược ban đầu cho người chơi đặt cược lớn.

Cộng thêm vào pot tổng tiền cược đã được đặt, bao gồm tiền cược nhỏ, tiền cược lớn và tiền cược đặt ban đầu của người chơi đặt cược nhỏ.

Cuối cùng, Trả về giá trị mới của pot sau khi đã cập nhật.

5.1.11 So sánh các lá bài cao nhất của các người chơi

```
def compare_high_cards(highest_players, flop, number_of_high_cards):
    # calculate high card by removing best hand combination
    for player in highest_players:
        if player.highest[0] == 1:
            pass
        else:
            player.highest = [1, player.highcard(player.highest[1], flop, number_of_high_cards)]
    for i in range(number_of_high_cards):
        highest_players = find_player_with_n_highest_value(highest_players, i)
        if len(highest_players) == 1:
            return highest_players
    return highest_players
```

Hàm compare_high_cards được sử dụng để so sánh các lá bài cao nhất của các người chơi trong trò chơi Poker.

Tham số đầu vào:

- highest_players: Danh sách các người chơi có lá bài cao nhất.
- flop: Danh sách các lá bài flop trên bàn.
- number_of_high_cards: Số lượng lá bài cao nhất cần so sánh.

Dùng Vòng lặp qua từng người chơi trong danh sách highest_players.

Nếu lá bài cao nhất của người chơi không phải là lá bài cao, thì cập nhật lá bài cao nhất của họ thành lá bài cao.

Sau đó dùng Vòng lặp để so sánh từng lá bài cao nhất của người chơi. Hàm find_player_with_n_hig được sử dụng để tìm người chơi có lá bài cao nhất ở vị trí thứ i trong danh sách.

Nếu chỉ còn một người chơi có lá bài cao nhất tại vị trí i, thì trả về danh sách đó.

Trả về danh sách các người chơi có lá bài cao nhất.

player.highcard: Hàm này trả về lá bài cao nhất của người chơi sau khi loại bỏ các bộ bài có điểm số cao hơn.

find_player_with_n_highest_value: Hàm này tìm người chơi có lá bài cao nhất ở vị trí thứ n trong danh sách.

5.1.12 Tìm người chơi có giá trị cao nhất ở vị trí thứ n trong danh sách người chơi

```
def find_player_with_n_highest_value(players, n=0):
    highest_value = 0
    for player in players:
        if type(player.highest[1][n]) is int:
            if player.highest[1][n] >= highest_value:
                highest_value = player.highest[1][n]
        else:
            if player.highest[1][n].value >= highest_value:
                highest_value = player.highest[1][n].value
    if type(players[0].highest[1][n]) is int:
        return [p for p in players if p.highest[1][n] >= highest_value]
    return [p for p in players if p.highest[1][n].value >= highest_value]
```

Hàm find_player_with_n_highest_value được sử dụng để tìm người chơi có lá bài cao nhất ở vị trí thứ n trong danh sách.

Tham số đầu vào:

- players: Danh sách các người chơi.
- n: Vị trí của lá bài cao nhất cần tìm trong danh sách các lá bài cao nhất.

Dùng Vòng lặp qua từng người chơi trong danh sách.

Kiểm tra xem giá trị của lá bài cao nhất ở vị trí n của người chơi là một số nguyên hay một lá bài có giá trị. Nếu đó là một số nguyên, so sánh trực tiếp giá trị của lá bài. Nếu là một lá bài, so sánh giá trị của lá bài.

Tìm giá trị cao nhất của lá bài ở vị trí n trong danh sách người chơi.

Kiểm tra lại loại dữ liệu của lá bài ở vị trí n của người chơi đầu tiên trong danh sách. Nếu là một số nguyên, thì trả về danh sách các người chơi có giá trị bằng hoặc lớn hơn giá trị cao nhất. Ngược lại, trả về danh sách các người chơi có giá trị của lá bài bằng hoặc lớn hơn giá trị cao nhất.

Lưu ý:

- Đối với trường hợp lá bài là số nguyên, so sánh trực tiếp giá trị của lá bài.
- Đối với trường hợp lá bài có giá trị, so sánh giá trị của lá bài

5.1.13 Lọc người chơi theo giá trị của lá bài cao nhất

```
def filter_by_value(players, value):
    return [player for player in players if player.highest[1][0] == value]
```

Hàm filter_by_value được sử dụng để lọc ra danh sách các người chơi có lá bài cao nhất có giá trị bằng value. Tham số đầu vào:

- players: Danh sách các người chơi cần lọc.
- value: Giá trị cần lọc.

Sử dụng list comprehension để tạo ra một danh sách mới chỉ chứa các người chơi có lá bài cao nhất có giá trị bằng value.

Vòng lặp for player in players duyệt qua từng người chơi trong danh sách players.

Biểu thức điều kiện if player.highest[1][0] == value kiểm tra xem giá trị của lá bài cao nhất của người chơi có bằng value không.

Lưu ý: Hàm này giả định rằng chỉ có một lá bài cao nhất được lựa chọn và so sánh với value. Trong trường hợp có nhiều lá bài cao nhất có cùng giá trị, nó chỉ trả về danh sách các người chơi có lá bài đầu tiên có giá trị bằng value.

5.1.14 Phát bài cho người chơi từ một bộ bài

```
def dealCards(p_list, deck):
    for player in p_list:
        for _ in range(2):
            player.hand.append(deck.draw_card())
```

Hàm dealCards được sử dụng để phát bài cho mỗi người chơi trong danh sách p_list. Tham số đầu vào:

- p_list: Danh sách các người chơi cần phát bài.
- deck: Bộ bài để lấy các lá bài từ đó.

Duyệt qua mỗi người chơi trong danh sách p_list.

Dùng vòng lặp để phát 2 lá bài cho mỗi người chơi.

Hàm deck.draw_card() được gọi để lấy một lá bài từ bộ bài.

Lá bài được thêm vào tay của người chơi bằng cách gọi player.hand.append().

Lưu ý: Mỗi người chơi trong danh sách p_list sẽ được phát 2 lá bài.

5.1.15 Phát các lá bài flop từ một bộ bài

```
def flop_deal(flop, deck):
    for _ in range(3):
        flop.append(deck.draw_card())
```

Hàm flop_deal được sử dụng để "ra flop" trong trò chơi Poker, nghĩa là phát 3 lá bài lên bàn. Tham số đầu vào:

- flop: Danh sách chứa các lá bài trên bàn (flop).
- deck: Bộ bài để lấy các lá bài từ đó.

Sử dụng vòng lặp để thực hiện 3 lần lặp, tương đương với việc phát 3 lá bài lên bàn.

Gọi deck.draw_card() để lấy một lá bài từ bộ bài và thêm vào danh sách flop bằng phương thức flop.append().

Lưu ý: Hàm này được sử dụng sau khi phát bài cho người chơi, để phát 3 lá bài chung lên bàn.

5.1.16 Lọc người chơi theo loại lá bài cao nhất mà họ có

```
def filter_by_highest_hand(players, target):
    return [player for player in players if player.highest[0] == target]
```

Hàm filter_by_highest_hand được sử dụng để lọc ra những người chơi có bộ bài cao nhất. Tham số đầu vào:

- players: Danh sách các người chơi.
- target: Giá trị mục tiêu, thường là loại bộ bài cao nhất.

Sử dụng biểu thức list comprehension để lặp qua mỗi người chơi trong danh sách players. Kiểm tra nếu giá trị của player.highest[0] (loại bộ bài cao nhất của người chơi) bằng target. Trả về danh sách các người chơi mà loại bộ bài cao nhất của họ là target.

Lưu ý: Hàm này hữu ích khi cần lọc ra các người chơi có bộ bài cao nhất theo một loại nhất định, chẳng hạn như thắng theo thứ tự high card, flush, full house,...

5.1.17 Lấy danh sách lá bài

```
def getHands(hands):
    myCards=[f"{card}" for card in hands]
    return pickle.dumps(myCards)
```

Hàm getHands nhận danh sách các lá bài và trả về một chuỗi được mã hóa dạng pickle của danh sách đó.

Tham số đầu vào:

hands: Danh sách các lá bài.

Sử dụng một biểu thức list comprehension để chuyển đổi mỗi lá bài trong danh sách hands thành một chuỗi, trong đó mỗi lá bài được biểu diễn bằng chuỗi của nó.

Sử dụng pickle.dumps để chuyển đổi danh sách các chuỗi lá bài thành một đối tượng byte-string đã được mã hóa bằng giao thức pickle. Điều này giúp lưu trữ và truyền tải danh sách các lá bài dễ dàng hơn.

Lưu ý: Giao thức pickle cho phép lưu trữ và truyền tải đối tượng Python dưới dạng chuỗi byte, giúp dễ dàng lưu trữ và truyền tải dữ liệu phức tạp như danh sách, từ điển,...

5.1.18 Thu thập thông tin người chơi

```
def getPlayers(player,player_list):
    player_index = player_list.index(player)
    player_list.pop(player_index)
    myList=[]
    for player1 in player_list:
        myList.append((player1.name,player1.initial_bet,player1.money,player1.hand))
    return pickle.dumps(myList)
```

Hàm getPlayers nhận vào một đối tượng người chơi player và một danh sách player_list chứa tất cả các người chơi còn lại trong trò chơi.

Trước tiên, nó tìm vị trí của player trong player_list và loại bỏ player ra khỏi danh sách.

Sau đó, nó tạo một danh sách mới myList chứa thông tin của tất cả các người chơi còn lại trong trò chơi, bao gồm tên, mức cược ban đầu, số tiền, và bộ bài của mỗi người chơi.

Cuối cùng, nó chuyển đổi danh sách này thành một chuỗi bytes bằng cách sử dụng module pickle và trả về kết quả.

5.1.19 Xác định người chiến thắng và số tiền trong bàn

```
def getWinner(winner,pot):
    s="Winner "
    if type(winner) is list:
        for w in winner:
            s+=f" {w.name}"
    else:
        s+= winner.name
    s+= f" win the pot : {pot}"
    return s
```

Hàm getWinner nhận hai đối số là winner và pot, trong đó winner là người chiến thắng hoặc danh sách các người chiến thắng, và pot là số tiền trong bàn.

Nếu winner là một danh sách, nó sẽ lặp qua danh sách này và thêm tên của mỗi người chiến thắng vào chuỗi s. Nếu winner không phải là một danh sách, nó chỉ đơn giản thêm tên của người chiến thắng vào chuỗi s.

Sau đó, hàm trả về một chuỗi thông báo về người chiến thắng và số tiền trong bàn.

5.1.20 Phát sóng hành động tới người chơi khác

```
def broadcast_to_others(current_player, player_list, action):
    if type(current_player) is not list:
        for player in player_list:
            message = f"{current_player.name} {action}"
            player.conn.sendall(message.encode())
    else:
        player_names = ', '.join([player.name for player in current_player])
        message = f"{player_names} {action}"
        for player in player_list:
            player.conn.sendall(message.encode())
```

Hàm broadcast_to_others được sử dụng để gửi thông điệp về hành động của người chơi hiện tại tới các người chơi khác trong danh sách player_list.

Nếu current_player không phải là một danh sách, tức là chỉ có một người chơi thực hiện hành động, hàm sẽ gửi thông điệp về hành động của người chơi đó tới tất cả các người chơi trong danh sách player_list.

Nếu current_player là một danh sách, tức là có nhiều người chơi cùng thực hiện hành động, hàm sẽ gửi thông điệp về hành động của các người chơi trong danh sách current_player tới tất cả các người chơi trong danh sách player_list.

Thông điệp được tạo dựa trên tên của người chơi và hành động thực hiện. Đối với trường hợp nhiều người chơi thực hiện hành động, tên của các người chơi sẽ được nối với nhau bằng dấu phẩy. Thông điệp sau đó được gửi đến tất cả các kết nối của người chơi trong player_list.

5.1.21 Hàm main của máy chủ

```
def main():
    """Starts the server."""
    with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
        s.bind((HOST, PORT))
        s.listen()
        print(f'Server listening on {HOST}:{PORT}')

        while True:
            conn, addr = s.accept()
            client_thread = threading.Thread(target=handle_client, args=(conn, addr))
            client_thread.start()
```

Hàm main() là hàm chính của chương trình, được sử dụng để khởi động máy chủ. Trong hàm này:

- Một socket được tạo và liên kết với địa chỉ và cổng được chỉ định.
- Máy chủ lắng nghe kết nối đến từ các client.
- Khi một client kết nối, máy chủ chấp nhận kết nối và bắt đầu một luồng mới để xử lý client đó bằng cách gọi hàm handle_client().

- Trong khi chương trình đang chạy, máy chủ sẽ tiếp tục lắng nghe kết nối mới từ các client và tạo luồng xử lý cho mỗi kết nối.

5.1.22 Khởi tạo Người chơi và Phương thức In bài

```
def __init__(self,name, conn, money = 5000):
    self.name = name
    self.highest = ()
    self.hand = []
    self.money = money
    self.initial_bet = 0
    self.host = False
    self.conn = conn
    self.room = None
def __str__(self):
    return f"{self.name}"
def print_value_of_hand(self,cards):
    values = [card.value for card in self.hand]
    print(values)
```

Hai phương thức này đều là phương thức khởi tạo (__init__) và phương thức chuỗi (__str__) của một lớp người chơi trong trò chơi Poker.

**** Phương thức khởi tạo (__init__):***

- Tham số:

- name: Tên của người chơi.
- conn: Kết nối của người chơi.
- money: Số tiền ban đầu của người chơi (mặc định là 5000).

- Công việc:

Khởi tạo các thuộc tính của người chơi bao gồm tên (name), bộ bài cao nhất (highest), bộ bài (hand), số tiền (money), mức cược ban đầu (initial_bet), trạng thái chủ (host), kết nối (conn), và phòng (room).

Lưu ý:

+ Thuộc tính highest được khởi tạo là một tuple rỗng.

+ Thuộc tính host được thiết lập là False mặc định.

**** Phương thức chuỗi (__str__):***

Công việc: Trả về tên của người chơi khi được chuyển đổi thành chuỗi.

Lưu ý: Phương thức này cho phép chúng ta in ra tên của người chơi một cách thuận tiện khi cần.

**** Phương thức print_value_of_hand:***

Tham số:

- cards: Danh sách các lá bài trong tay của người chơi.

Công việc: In ra giá trị của các lá bài trong tay của người chơi.

5.1.23 In tên của các lá bài trong tay và sắp xếp theo giá trị

```
def print_name_of_hand(self):
    return (f"Your hand: {[card.name for card in self.hand]}\n")

def sort_hand_by_value(self,cards):
    return sorted(cards, key=lambda card: card.value,reverse=True)
```

print_name_of_hand(self): Phương thức này trả về một chuỗi mô tả các lá bài trong tay của người chơi. Nó sử dụng list comprehension để lặp qua danh sách các lá bài trong tay (self.hand) và tạo ra một danh sách các tên của chúng. Kết quả cuối cùng là một chuỗi có dạng "Your hand: [tên lá bài1, tên lá bài2, ...]".

sort_hand_by_value(self, cards): Phương thức này nhận vào một danh sách các lá bài và sắp xếp chúng theo giá trị giảm dần. Nó sử dụng hàm sorted() và tham số key là một lambda function để sắp xếp các lá bài dựa trên giá trị của chúng. Kết quả là một danh sách các lá bài đã được sắp xếp theo giá trị giảm dần.

5.1.24 Xác định lá bài cao nhất của người chơi

```
def highcard(self, highest, flop,ncard = None):
    all_cards = self.hand + flop
    high_cards =  [card for card in all_cards if card not in highest]
    high = self.sort_hand_by_value(high_cards)
    if ncard is not None:
        return high[:ncard+1]
    else:
        return high
```

Hàm highcard trong đoạn code trên được sử dụng để tìm các lá bài cao nhất trong tay của một người chơi sau khi loại bỏ các lá bài tạo thành cặp hoặc bộ bài tốt nhất.

Tham số self: Đây là tham số đại diện cho đối tượng người chơi hiện tại.

Tham số highest: Là một danh sách chứa các lá bài đã được sắp xếp thành một cặp hoặc bộ bài tốt nhất của người chơi.

Tham số flop: Là một danh sách chứa các lá bài trên bàn.

Tham số ncard: Là số lượng lá bài cao nhất mà người chơi muốn trả về.

Các bước hoạt động của hàm:

- Kết hợp các lá bài trong tay của người chơi và các lá bài trên bàn thành một danh sách all_cards.
- Loại bỏ các lá bài tạo thành cặp hoặc bộ bài tốt nhất (highest) khỏi all_cards, và lưu vào danh sách high_cards.
- Sắp xếp danh sách high_cards theo giá trị từ cao đến thấp bằng cách sử dụng hàm sort_hand_by_value.
- Nếu ncard được cung cấp, hàm trả về ncard + 1 lá bài cao nhất từ danh sách high, ngược lại trả về tất cả các lá bài cao nhất từ danh sách high.

5.1.25 Kiểm tra thùng sảnh

```
def hasStraightFlush(self, hand, flop):
    flush_value = self.hasFlush(hand, flop)
    if flush_value:
        flush_cards = self.filter_by_suit(hand + flop, flush_value[0].suit)
        straight_value = self.hasStraight(flush_cards, [])
        if straight_value:
            return straight_value  # Return the value of the highest card in
    return None
```

Hàm hasStraightFlush kiểm tra xem người chơi có tụ hạng không.

Kiểm tra Flush: Đầu tiên, hàm kiểm tra xem người chơi có Flush không bằng cách gọi hàm hasFlush().

Kiểm tra Straight trong Flush: Nếu người chơi có Flush, hàm tiếp tục kiểm tra xem trong các lá bài của Flush có tạo thành một Straight không, bằng cách gọi hàm hasStraight().

Trả về kết quả: Nếu người chơi có cả Flush và Straight trong Flush, hàm sẽ trả về giá trị của lá bài cao nhất trong Straight Flush. Nếu không, nó trả về None để chỉ ra rằng người chơi không có Straight Flush.

5.1.26 Kiểm tra cặp bài

```
def hasPair(self,hand, flop):
    hand_copy = hand[:]
    hand_copy.extend(flop)
    values = [card.value for card in hand_copy]
    value_counts = Counter(values)
    pairs = [value for value, count in value_counts.items() if count >= 2]
    if pairs:
        return pairs
    else:
        return None
```

Hàm hasPair kiểm tra xem người chơi có cặp bài hay không trong tay và flop của họ.

Đầu tiên, nó tạo một bản sao của tay để tránh sửa đổi danh sách gốc.

Sau đó, nó kết hợp bản sao này với các lá bài trên bàn (flop).

Tiếp theo, nó tạo một danh sách các giá trị của các lá bài.

Sử dụng Counter để đếm số lần xuất hiện của mỗi giá trị.

Cuối cùng, nó trả về danh sách các giá trị xuất hiện ít nhất 2 lần, nghĩa là các cặp bài nếu có. Nếu không có cặp nào, hàm trả về None.

5.1.27 Kiểm tra hai cặp bài

```
def has2pairs(self, hand, flop):
    hand_copy = hand[:]
    hand_copy.extend(flop)

    values = [card.value for card in hand_copy]
    value_counts = Counter(values)

    pairs = [value for value, count in value_counts.items() if count >= 2]

    if len(pairs) >= 2:
        return sorted(pairs, reverse=True)[:2] # Return the two pairs
    else:
        return None
```

Hàm has2pairs kiểm tra xem người chơi có hai cặp bài không trong tay và trên bàn (flop).

Đầu tiên, nó tạo một bản sao của tay để tránh sửa đổi danh sách ban đầu.

Sau đó, nó mở rộng bản sao này bằng cách thêm các lá bài trên bàn (flop).

Tiếp theo, nó đếm số lần xuất hiện của mỗi giá trị lá bài trong tay và trên bàn.

Sau đó, nó tìm các cặp bài bằng cách xem xét các giá trị có số lần xuất hiện ít nhất là 2.

Nếu có ít nhất hai cặp bài, nó trả về hai cặp bài có giá trị cao nhất, được sắp xếp giảm dần.

Nếu không có cặp bài nào, nó trả về None.

5.1.28 Kiểm tra Bộ ba

```
def hasThree(self,hand, flop):
    hand_copy = hand[:]
    hand_copy.extend(flop)
    values = [card.value for card in hand_copy]
    value_counts = Counter(values)
    three_of_a_kind = [value for value, count in value_counts.items() if count >= 3]
    if three_of_a_kind:
        return three_of_a_kind
    else:
        return None
```

Hàm này kiểm tra xem có tồn tại một bộ ba bài nào đó trong bộ bài của người chơi và trên bàn chung sau khi ra Flop hay không.

Để làm điều này, hàm tạo một bản sao của bộ bài của người chơi và mở rộng nó bằng cách thêm các lá bài trên bàn chung (flop).

Sau đó, nó đếm số lần xuất hiện của mỗi giá trị bài trong bộ bài và kiểm tra xem có ba lá bài nào có cùng một giá trị không.

Nếu có, hàm trả về giá trị của các lá bài trong bộ ba. Nếu không tìm thấy, hàm trả về None.

5.1.29 Kiểm tra tứ quý

```
def hasSquad(self, hand, flop):
    hand_copy = hand[:]
    hand_copy.extend(flop)
    values = [card.value for card in hand_copy]
    value_counts = Counter(values)
    squad = [value for value, count in value_counts.items() if count >= 4]
    if squad:
        return squad
    else:
        return None
```

Hàm hasSquad kiểm tra xem người chơi có bộ tứ quý không.

Tạo một bản sao của bộ bài của người chơi và thêm vào các lá bài trên bàn (flop).

Tạo một danh sách chứa các giá trị của các lá bài trong bộ bài.

Đếm số lần xuất hiện của mỗi giá trị bằng cách sử dụng Counter.

Tìm các giá trị mà xuất hiện ít nhất 4 lần trong bộ bài, đây chính là bộ tứ quý.

Nếu tìm thấy bộ tứ quý, trả về danh sách chứa giá trị của bộ tứ quý, ngược lại trả về None.

5.1.30 Phát hiện full house

```
def hasFullHouse(self, hand, flop):
    hand_copy = hand[:]
    hand_copy.extend(flop)
    value_counts = Counter(card.value for card in hand_copy)
    if {3, 2}.issubset(set(value_counts.values())):
        three_of_a_kind = None
        pair = None
        for value, count in value_counts.items():
            if count == 3:
                three_of_a_kind = value
            elif count == 2:
                pair = value
        return [three_of_a_kind, pair]
    else:
        return None
```

Hàm hasFullHouse kiểm tra xem một bộ bài có tổ hợp là Full House không.

Đầu tiên, hàm tạo một bản sao của bộ bài để tránh thay đổi danh sách gốc.

Sau đó, nó tính số lần xuất hiện của mỗi giá trị lá bài.

Nếu có chính xác hai giá trị phân biệt: một giá trị xuất hiện 3 lần và một giá trị xuất hiện 2 lần, hàm trả về danh sách gồm ba lá bài giống nhau và một cặp lá bài khác.

Ví dụ:

Nếu bộ bài chứa ba lá bài với giá trị 7 và hai lá bài với giá trị 3, thì hàm sẽ trả về [7,

3].

Nếu không có tổ hợp Full House, hàm trả về None.

5.1.31 Tìm bộ dây (Straight - một bộ năm con liên tiếp)

```
def hasStraight(self,hand, flop):
    hand_copy = hand[:]
    hand_copy.extend(flop)
    hand_copy = self.sort_hand_by_value(hand_copy)
    straight_list = []
    for i in range(len(hand_copy) - 1):
        current_value = hand_copy[i].value
        next_value = hand_copy[i + 1].value
```

```
        if next_value == current_value - 1:  # Cards are consecutive
            if hand_copy[i] not in straight_list and hand_copy[i+1] not in straight_list:
                straight_list.append(hand_copy[i])
                straight_list.append(hand_copy[i+1])
        elif next_value == current_value:  # Allow duplicates within a straight
            continue
        else:  # Cards are not consecutive, reset straight tracking
            straight_list = []

        if len(straight_list) >= 5:
            return straight_list[:5]
    return None  # No straight found
```

Hàm hasStraight kiểm tra xem người chơi có bộ dây không trong tay và trên bàn.

Trước tiên, hàm tạo một bản sao của tay người chơi để không làm thay đổi danh sách gốc.

Tiếp theo, nó mở rộng bản sao đó với các lá bài trên bàn.

Sau đó, nó sắp xếp các lá bài theo giá trị và tạo ra một danh sách rỗng để lưu các lá bài liên tiếp.

Hàm duyệt qua danh sách đã sắp xếp và kiểm tra xem hai lá bài liên tiếp có giá trị lần lượt nhỏ hơn nhau 1 đơn vị không. Nếu có, nó thêm chúng vào danh sách liên tiếp.

Nếu giá trị của hai lá bài liên tiếp bằng nhau, nó bỏ qua và tiếp tục duyệt.

Nếu không, nó đặt lại danh sách liên tiếp.

Nếu độ dài của danh sách liên tiếp đạt 5, nó trả về danh sách đó là bộ dây.

Nếu không tìm thấy bộ dây, hàm trả về None.

Hàm này sử dụng một cách tiếp cận tương tự để kiểm tra xem có bộ dây hay không, và nó cho phép các lá bài trùng lặp trong một bộ dây.

5.1.32 Kiểm tra xấp bài và Gập lại bài

```
def hasFlush(self,hand, flop,nmax = 0):
    hand_copy = hand[:]
    hand_copy.extend(flop)
    suit_counts = {}
    for card in hand_copy:
        suit = card.suit
        suit_counts[suit] = suit_counts.get(suit, 0) + 1
    for suit, count in suit_counts.items():
        if count >= 5:
            flush_cards = self.filter_by_suit(hand_copy, suit)
            if nmax == 0:
                return  self.sort_hand_by_value(flush_cards)
            else:
                flush_cards = self.sort_hand_by_value(flush_cards)
                return flush_cards[-nmax].value
    return None

def fold(self):
    self.money -= self.initial_bet
```

Hàm hasFlush kiểm tra xem một người chơi có Flush hay không trong bộ bài của mình

và trên bàn (flop). Flush là một tình huống trong Poker khi một người chơi có năm lá bài cùng một suit.

- Đầu tiên, hàm tạo một bản sao của bộ bài người chơi để tránh làm thay đổi danh sách ban đầu. Sau đó, thêm các lá bài trên bàn vào bản sao này.
- Tiếp theo, hàm đếm số lần xuất hiện của mỗi suit trong bộ bài.
- Tiếp theo, hàm đếm số lần xuất hiện của mỗi suit trong bộ bài.
- Sau đó, hàm kiểm tra xem có bất kỳ suit nào xuất hiện ít nhất năm lần không. Nếu có, nó trả về danh sách các lá bài trong Flush hoặc giá trị lớn thứ nmax trong Flush (nếu có).
- Nếu không có Flush nào, hàm trả về None.

Hàm fold đơn giản là giảm số tiền cược ban đầu của người chơi từ số tiền của họ. Điều này thường được gọi khi một người chơi quyết định gấp bài của mình.

5.1.33 Kiểm tra và thực hiện cược của người chơi

```
def call(self, current_bet):
    if self.money >= current_bet - self.initial_bet:
        print(f"{self.name} calls the bet of {current_bet}.")
        print("-"*20)
        self.money -= current_bet - self.initial_bet
        self.initial_bet = current_bet
        return True
    else:
        print(f"{self.name} doesn't have enough chips to call.
              They go all-in with {self.money} chips.")
        self.money = 0
        return False
```

Hàm call kiểm tra xem người chơi có đủ chip để gọi cược hiện tại không. Nếu có, người chơi sẽ gọi cược và trừ số chip tương ứng từ số chip ban đầu của mình. Nếu không đủ, người chơi sẽ đặt tất cả chip của mình vào cuộc (all-in) và số chip của họ sẽ trở thành 0.

- Nếu người chơi có đủ chip để gọi cược, hàm sẽ in ra thông báo "tên người chơi calls the bet of current_bet." và trừ số chip cần thiết.
- Nếu không đủ chip để gọi cược, hàm sẽ in ra thông báo "tên người chơi doesn’t have enough chips to call. They go all-in with số chip của họ chips" và đặt số chip của người chơi thành 0.

5.1.34 Cược Tiền

```
def bet(self, amount):
    if self.money >= self.initial_bet+amount:
        print(f"{self.name} bets {amount} money.")
        print("-" * 20)
        self.money -= amount
        self.initial_bet = self.initial_bet+amount
        return self.initial_bet
    else:
        print(f"{self.name} doesn't have enough money to bet.
              They go all-in with {self.money} chips.")
        bet_all_in = self.money
        self.money = 0
        return self.initial_bet+bet_all_in
```

Hàm bet trong đoạn code trên thực hiện việc đặt cược của người chơi.

Nếu số tiền người chơi có đủ để đặt cược, hàm sẽ thực hiện việc trừ số tiền đặt cược từ số tiền hiện có của người chơi và cập nhật số tiền đã đặt cược của người chơi.

Nếu số tiền không đủ để đặt cược, người chơi sẽ đặt cược toàn bộ số tiền có, và hàm sẽ trả về tổng số tiền đã đặt cược của người chơi.

5.1.35 Lọc bài theo bộ

```
def filter_by_suit(self,cards, target_suit):
        return [card for card in cards if card.suit == target_suit]
```

Hàm này lọc các lá bài từ một danh sách các lá bài theo suit được chỉ định.

cards: Danh sách các lá bài cần lọc.

target_suit: Suit cần lọc.

Hàm sẽ trả về một danh sách mới chỉ chứa các lá bài có suit là target_suit.

5.1.36 Khởi tạo client và tham gia server

```
def __init__(self):
    self.client=socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    self.client.settimeout(5)
    self.id=self.connect_and_join()
```

def __init__(self):: Đây là phương thức khởi tạo của lớp. Khi một đối tượng mới của lớp được tạo, phương thức này được gọi tự động.

self.client=socket.socket(socket.AF_INET, socket.SOCK_STREAM): Đoạn code này tạo một socket client mới với domain là socket.AF_INET (IPv4) và kiểu là socket.SOCK_STREAM (TCP).

self.client.settimeout(5): Đặt timeout cho socket là 5 giây. Nếu không nhận được phản hồi từ server sau khoảng thời gian này, socket sẽ timeout.

self.id=self.connect_and_join(): Gọi phương thức connect_and_join() để kết nối và tham gia vào một server.

Sau đó, kết quả được gán vào thuộc tính id của đối tượng. Đây có thể là một mã định danh hoặc một giá trị khác tùy thuộc vào cài đặt cụ thể của ứng dụng.

5.1.37 Thiết lập thông tin client khi tham gia phòng

```python
def setNAR(self,room_id,name):
    self.room_id=room_id
    self.name=name
    message = f"JOIN {room_id} AS {name}"
    self.client.send(message.encode())
    data = self.client.recv(1024).decode()
    if data.find("host") != -1:
        return True
    else:
        return False
```

Hàm setNAR được sử dụng để thiết lập thông tin của client khi tham gia vào một phòng. Nó nhận hai đối số là room_id (ID của phòng) và name (tên của client).

- self.room_id và self.name được gán bằng các giá trị tương ứng.
- Một thông điệp được tạo để gửi đến server với nội dung "JOIN room_id AS name" để thông báo rằng client muốn tham gia vào phòng với room_id và tên là name.
- Thông điệp được gửi đi qua kết nối của client với phương thức send.
- Client nhận dữ liệu từ server với recv. Nếu dữ liệu chứa từ "host" (tức là client được phân làm chủ phòng), hàm trả về True, ngược lại trả về False.

5.1.38 Kết nối và tham gia vào server

```python
def connect_and_join(self):
    try:
        self.client.connect((HOST,PORT))
        return 1
    except:
        pass
```

Đoạn code này là phương thức để kết nối và tham gia vào server. Nó cố gắng kết nối với máy chủ (host) tại địa chỉ và cổng đã được chỉ định.

Nếu kết nối thành công, nó trả về 1, ngược lại nếu gặp lỗi hoặc không thể kết nối, nó sẽ không trả về gì cả.

5.1.39 Gửi và nhận dữ liệu qua socket

```
def sendData(self,data):
    try:
        self.client.send(data.encode())
        data=self.client.recv(1024).decode()
        return data
    except:
        pass
```

Hàm setData gửi dữ liệu đến máy chủ thông qua kết nối socket. Nó nhận dữ liệu từ tham số đầu vào và gửi đi sau khi mã hóa thành dạng bytes.

Sau đó, nó chờ và nhận phản hồi từ máy chủ, sau đó giải mã và trả về dữ liệu nhận được.

Nếu có lỗi xảy ra trong quá trình gửi hoặc nhận dữ liệu, hàm sẽ bỏ qua và không trả về bất kỳ giá trị nào.

5.1.40 Tương tác với Server và Nhận Dữ liệu

```
def GetObs(self,data):
    try:
        self.client.send(data.encode())
        data = self.client.recv(1024)
        data = pickle.loads(data)
        return data
    except:
        return None
```

Hàm này được sử dụng để gửi dữ liệu và nhận kết quả từ server thông qua socket.

Trước tiên, nó gửi dữ liệu data đã được mã hóa sang byte qua kết nối socket.

Sau đó, nó nhận phản hồi từ server.

Phản hồi được giải mã từ dạng byte sang dữ liệu gốc bằng cách sử dụng pickle.loads().

Kết quả được trả về cho người dùng hoặc ứng dụng. Nếu có bất kỳ lỗi nào trong quá trình này, hàm sẽ trả về None.

5.1.41 Vẽ các lá bài trên màn hình

```
def drawCards(screen,cards):
    if cards:
        for ind,card in enumerate(cards):
            card_game=pygame.image.load(f"IMGCards/{card}.png")
            card_image = pygame.transform.scale(card_game, (75, 100))
            screen.blit(card_image, (350+55*ind, 360))
```

Hàm này vẽ các lá bài trên màn hình game. Đầu vào của hàm là screen, đại diện cho màn hình của trò chơi và cards, là danh sách các lá bài cần vẽ.

Nếu danh sách cards không rỗng, vòng lặp for sẽ duyệt qua từng lá bài trong danh sách. Mỗi lá bài sẽ được load từ tệp ảnh trong thư mục "IMGCards", sau đó được chuyển đổi kích thước thành 75x100 pixels và được vẽ lên màn hình game tại các vị trí được tính toán dựa trên chỉ số ind của lá bài trong danh sách.

5.1.42 Vẽ các lá bài trên bàn (Flop)

```
def drawflop(screen,flop):
    if flop:
        for ind,card in enumerate(flop):
            card_game=pygame.image.load(f"IMGCards/{card}.png")
            card_image = pygame.transform.scale(card_game, (75, 100))
            screen.blit(card_image, (200+80*ind, 200))
```

Hàm này được sử dụng để vẽ các lá bài trên bàn (Flop) lên màn hình trong trò chơi. Nó nhận đầu vào là một danh sách chứa các lá bài trên bàn và một đối tượng màn hình của Pygame.

Sau đó, nó lặp qua từng lá bài trong danh sách và tải hình ảnh tương ứng từ thư mục

IMGCards, sau đó chia nhỏ hình ảnh đó về kích thước 75x100 pixels và vẽ chúng lên màn hình tại vị trí được xác định.

5.1.43 Vẽ người chiến thắng

```
def drawWinner(winner,screen,font):
    color =(231, 201, 13)
    Winner_rect = pygame.Rect(150, 30, 500, 30)
    name_surface1 = font.render(f'{winner}', True, (0,0,0))
    name_rect1 = name_surface1.get_rect(midleft=(Winner_rect.left + 50, Winner_rect.centery))
    pygame.draw.rect(screen, color, Winner_rect)
    screen.blit(name_surface1, name_rect1)
```

Hàm này vẽ thông báo người chiến thắng lên màn hình trong trò chơi.

winner: Tên người chiến thắng.

screen: Màn hình trò chơi.

font: Font được sử dụng để hiển thị văn bản.

Hàm sử dụng thư viện Pygame để vẽ một hình chữ nhật và hiển thị tên người chiến thắng bên trong nó.

- Tạo một hình chữ nhật để đặt văn bản thông báo người chiến thắng.
- Tạo bề mặt văn bản cho tên người chiến thắng.
- Vẽ hình chữ nhật trên màn hình với màu nền và vị trí được xác định trước.
- Vẽ văn bản lên màn hình tại vị trí cụ thể trong hình chữ nhật.

5.1.44 Tạo bộ bài

Hàm generate_deck này tạo một bộ bài bằng cách tạo ra các lá bài từ 2 đến A của các bộ bài Hearts, Tiles, Clovers và Pikes.

suits: Mảng chứa các loại quân bài (Hearts, Tiles, Clovers, Pikes).

ranks: Mảng chứa các giá trị của quân bài (từ 2 đến A).

```
def generate_deck(self):
    suits = ['Hearts', 'Tiles', 'Clovers', 'Pikes']
    ranks = ['2', '3', '4', '5', '6', '7', '8', '9', '10',
             'Jack', 'Queen', 'King', 'A']
    for suit in suits:
        for rank in ranks:
            if rank.isdigit():
                value = int(rank)
            elif rank in ['Jack', 'Queen', 'King']:
                value = 10 + ranks.index(rank) - ranks.index('Jack') + 1
            else:
                value = 14  # Ace
            name = f"{rank}"
            card = Card(name, value, suit)
            self.cards.append(card)
```

Vòng lặp lồng nhau đi qua từng loại quân bài và từng giá trị của quân bài để tạo ra các lá bài.

Nếu giá trị của quân bài là một số, giá trị được đặt là chính giá trị đó.

Nếu giá trị của quân bài là Jack, Queen hoặc King, giá trị được đặt là 10 cộng với vị trí của nó trong mảng ranks trừ đi vị trí của Jack và cộng thêm 1.

Nếu giá trị của quân bài là Ace, giá trị được đặt là 14. Tên của lá bài được đặt bằng cách kết hợp giá trị và loại của lá bài.

Mỗi lá bài được tạo ra sẽ được thêm vào danh sách self.cards.

5.1.45 Xáo trộn bộ bài và rút lá bài từ bộ bài

```
def shuffle(self):
    random.shuffle(self.cards)

def draw_card(self):
    if len(self.cards) > 0:
        return self.cards.pop()
    else:
        print("Deck is empty!")
```

Hàm shuffle được sử dụng để xáo trộn bộ bài.

Hàm draw_card được sử dụng để rút một lá bài từ bộ bài. Nếu bộ bài không còn lá bài nào, nó sẽ thông báo rằng "Deck is empty!".

5.1.46 Định nghĩa các lá bài trong bộ bài

```
class Card:
    def __init__(self, name, value,suit):
        self.name = name
        self.value = value
        self.suit = suit

    def __str__(self):
        return f"{self.suit}_{self.name}_white"
```

Đây là một class Card trong Python, đại diện cho một lá bài trong bộ bài.

__init__(self, name, value, suit): Phương thức khởi tạo của class, nhận các thông tin cần thiết để tạo một lá bài mới, bao gồm tên (name), giá trị (value) và suit (nước bài).

__str__(self): Phương thức trả về một chuỗi mô tả lá bài, gồm suit, name và màu sắc của lá bài.

Ví dụ: Nếu tạo một lá bài với name="Ace", value=14, suit="Hearts", thì khi gọi phương thức __str__, nó sẽ trả về chuỗi "Hearts_Ace_white".

5.2 Xây dựng Game Đặt Boom

5.2.1 Khởi tạo đối tượng Player

```
def __init__(self, rect, surfaceList, bomb):
    self.startRect = [rect.centerx,rect.centery]
    self.rect = rect
    self.vel = 4
    self.surfaceList = surfaceList
    self.surface = surfaceList[0]
    self.maxBomb = 1
    self.listBomb = [bomb]
    self.status = 1
    self.recoverTime = 0
```

Định nghĩa một lớp Player trong game Bomb SocketPython.

rect: Đây là hình chữ nhật mô tả vị trí và kích thước của player trên màn hình.

surfaceList: Là danh sách các hình ảnh của player tương ứng với các hướng di chuyển khác nhau.

bomb: Đối tượng Bomb của player.

startRect: Lưu trữ vị trí ban đầu của player.

vel: Tốc độ di chuyển của player.

surface: Hình ảnh của player hiện tại.

maxBomb: Số lượng bom tối đa mà player có thể đặt.

listBomb: Danh sách các đối tượng bom của player.

status: Trạng thái của player, có thể là 1 (còn sống) hoặc 0 (chết).

recoverTime: Thời gian hồi phục sau khi chết của player.

Hàm __init__ khởi tạo một đối tượng Player với các thuộc tính như trên.

5.2.2 Các phương thức vẽ và cập nhật của Player

Hàm draw: Vẽ hình ảnh của người chơi lên màn hình.

```
def draw(self, screen):
    screen.blit(self.surface, self.rect)

def update(seft, playerRect):
    seft.rect = playerRect

def setSurface_Rect(self, playerRect, indexSurface):
    self.rect = playerRect
    self.surface = self.surfaceList[indexSurface]
```

Hàm update: Cập nhật vị trí của người chơi dựa trên hộp giới hạn mới.

Hàm setSurface_Rect: Cập nhật hình ảnh và hộp giới hạn mới cho người chơi.

5.2.3 Phương thức di chuyển của Player

```
def move(self, direction):
    if self.status == 1:
        if direction == "left":
            if self.ValidNextStep("left"):
                self.rect.centerx -= self.vel
        if direction == "right":
            if self.ValidNextStep("right"):
                self.rect.centerx += self.vel
        if direction == "up":
            if self.ValidNextStep("up"):
                self.rect.centery -= self.vel
        if direction == "down":
            if self.ValidNextStep("down"):
                self.rect.centery += self.vel
```

Phương thức move trong đoạn mã này cho phép di chuyển Player theo hướng chỉ định.

Nó kiểm tra trạng thái của Player (self.status) để đảm bảo rằng Player có thể di chuyển trước khi thực hiện hành động. Sau đó, nó kiểm tra hướng di chuyển được chỉ định và kiểm tra xem bước tiếp theo có hợp lệ không thông qua phương thức ValidNextStep.

- Nếu hướng là "left" (trái), nó sẽ giảm giá trị centerx của rect đi self.vel nếu bước tiếp theo là hợp lệ.
- Tương tự, nếu hướng là "right" (phải), nó sẽ tăng centerx.
- Nếu hướng là "up" (lên), nó sẽ giảm giá trị centery của rect đi self.vel.
- Nếu hướng là "down" (xuống), nó sẽ tăng centery.

Trong mỗi trường hợp, nó kiểm tra xem bước tiếp theo có hợp lệ không trước khi di chuyển.

5.2.4 Kiểm tra hướng di chuyển hợp lệ

```
def ValidNextStep(self, direction):
    if direction == "left":
        if self.rect.centerx - self.vel <= 50:
            return False
    if direction == "right":
        if self.rect.centerx + self.vel >= 800-50:
            return False
    if direction == "up":
        if self.rect.centery - self.vel <= 0+50:
            return False
    if direction == "down":
        if self.rect.centery + self.vel >= 800-50:
            return False
    return True
```

Phương thức này kiểm tra xem người chơi có thể di chuyển vào hướng tiếp theo được không.

Đối với mỗi hướng di chuyển (trái, phải, lên, xuống), nó kiểm tra xem việc di chuyển sẽ đưa ra khỏi ranh giới của màn hình hoặc vào trong vùng cấm (ví dụ như khu vực hiển thị trò chơi).

Nếu điều kiện nào đó không thỏa mãn, nó sẽ trả về False, ngược lại sẽ trả về True, cho phép người chơi di chuyển.

5.2.5 Kiểm tra va chạm và lấy thông tin vị trí và hình ảnh của player

```
def checkTouchingObj(self, Obj):
    if Obj == 0:
        return False
    if self.rect.colliderect(Obj):
        return True
    return False

def getRect(self):
    return self.rect

def getSurface(self):
    return self.surface
```

Hàm checkTouchingObj kiểm tra xem player có va chạm với một object nào đó không. Nếu player va chạm với object, hàm trả về True, ngược lại trả về False.

Hàm getRect trả về hình chữ nhật đại diện cho vị trí của player.

Hàm getSurface trả về hình ảnh hiển thị của player.

5.2.6 Đặt bom và lấy danh sách bom

Định nghĩa phương thức placeABomb() để đặt một quả bom tại vị trí xác định trên màn hình. Phương thức này nhận vào hai đối số bombRectx và bombRecty, là tọa độ x và y của vị trí muốn đặt bom, cũng như một âm thanh sound_PlaceBomb được phát khi đặt bom.

Trong phương thức placeABomb(), vòng lặp duyệt qua danh sách bom của người chơi để tìm một quả bom chưa được sử dụng.

Nếu tìm thấy một quả bom có trạng thái là 0 (chưa được sử dụng), phương thức đặt trạng thái của nó thành 1 (đã được đặt).

```
def placeABomb(self, bombRectx, bombRecty, sound_PlaceBomb):
    for i in range(len(self.listBomb)):
        if self.listBomb[i].getStatus() == 0:
            self.listBomb[i].setStatus(1)
            self.listBomb[i].setRectWithPosition(bombRectx,bombRecty)
            sound_PlaceBomb.play()
            break

def getListBomb(self):
    return self.listBomb
```

Cập nhật vị trí của quả bom dựa trên bombRectx và bombRecty.

Phát âm thanh sound_PlaceBomb. Kết thúc vòng lặp và thoát khỏi phương thức.

Phương thức getListBomb() trả về danh sách các quả bom của người chơi.

5.2.7 Xử lý thời gian đếm ngược và nổ bom của người chơi

```
def increaseTimer_BombBang(self, value, sound_BombBang):
    for bomb in self.listBomb:
        if bomb.getStatus()==1 or bomb.getStatus()==2:
            bomb.increaseTimer(value)
            if bomb.getTimer() == 2000:
                bomb.Bang()
                sound_BombBang.play()
            if bomb.getTimer() == 2500:
                bomb.hide()
```

increaseTimer_BombBang(self, value, sound_BombBang): Phương thức này được sử dụng để tăng thời gian đếm ngược cho các quả bom của người chơi và phát âm thanh khi quả bom nổ.

value: Đại diện cho giá trị thời gian được tăng lên cho mỗi vòng lặp. sound_BombBang: Âm thanh được phát khi quả bom nổ.

Trong hàm này:

- Duyệt qua danh sách các quả bom trong trò chơi.
- Nếu trạng thái của quả bom là 1 hoặc 2 (tức là quả bom đang chờ nổ hoặc đang trong quá trình nổ):
  - Tăng thời gian của quả bom lên value.
  - Nếu thời gian của quả bom đạt đến 2000 (2 giây):
    * Gọi hàm Bang() của quả bom để nổ.
    * Phát âm thanh của quả bom nổ.
  - Nếu thời gian của quả bom đạt đến 2500 (2.5 giây): Ẩn quả bom.

5.2.8 Xử lý kết thúc và chờ đợi trong trò chơi

```
def gameOver(self):
    self.status = 0
    self.surface = self.surfaceList[1]

def gameWaiting(self):
    self.status = 2
    self.surface = self.surfaceList[1]
```

Hàm gameOver và gameWaiting được sử dụng để thay đổi trạng thái của người chơi khi trò chơi kết thúc hoặc khi đang chờ đợi.

Trong hàm gameOver, trạng thái của người chơi được đặt thành 0, và bề mặt (surface) của người chơi được thay đổi thành một hình ảnh khác (thường là hình ảnh của người chơi bị thua cuộc).

Trong hàm gameWaiting, trạng thái của người chơi được đặt thành 2, và bề mặt của người chơi cũng được thay đổi thành một hình ảnh khác (thường là hình ảnh của người chơi đang chờ đợi).

5.2.9 Khôi phục và kiểm tra kết thúc game cho người chơi

```
def recover(self):
    self.surface = self.surfaceList[0]
    self.rect.centerx = self.startRect[0]
    self.rect.centery = self.startRect[1]
    self.status = 1

def isGameOver(self):
    if self.status == 0:
        return True
    return False
```

Phương thức recover: Được sử dụng để khôi phục trạng thái của người chơi sau khi kết thúc một ván chơi. Khi gọi phương thức này, người chơi sẽ được di chuyển về vị trí ban đầu của mình, hình ảnh của người chơi cũng sẽ được chuyển về hình ảnh bình thường, và trạng thái của người chơi sẽ được đặt lại thành 1 (đang hoạt động).

Phương thức isGameOver: Được sử dụng để kiểm tra xem trạng thái của người chơi có phải là kết thúc game hay không. Nếu trạng thái của người chơi là 0 (kết thúc game), phương thức trả về True, ngược lại trả về False.

5.2.10 Truy xuất thuộc tính người chơi

```
def getCenterx(self):
    return self.rect.centerx

def getCentery(self):
    return self.rect.centery

def getVel(self):
    return self.vel
```

Định nghĩa các phương thức để truy xuất các thuộc tính của đối tượng người chơi.

Phương thức getCenterx: Trả về tọa độ x của trung tâm của hình chữ nhật đại diện cho

người chơi.

Phương thức getCentery: Trả về tọa độ y của trung tâm của hình chữ nhật đại diện cho người chơi.

Phương thức getVel: Trả về vận tốc di chuyển của người chơi.

5.2.11 Phương thức truy xuất thông tin người chơi

```
def getMaxBomb(self):
    return self.maxBomb

def getStatus(self):
    return self.status

def getRecoverTime(self):
    return self.recoverTime
```

*** getMaxBomb(self):** Chức năng: Trả về số lượng bom tối đa mà người chơi có thể giữ được.

Biến trả về: maxBomb - Số lượng bom tối đa.

getStatus(self):

Chức năng: Trả về trạng thái hiện tại của người chơi (0 - Game over, 1 - Đang hoạt động, 2 - Đang chờ).

Biến trả về: status - Trạng thái của người chơi.

getRecoverTime(self):

Chức năng: Trả về thời gian hồi phục của người chơi sau khi bị loại khỏi trò chơi (đối với trạng thái chờ).

Biến trả về: recoverTime - Thời gian hồi phục của người chơi.

5.2.12 Cập nhật thông tin người chơi từ DTO

```
def setByDTO(self, playerDTO):
    self.rect.centerx = playerDTO.get_centerx()
    self.rect.centery = playerDTO.get_centery()
    self.status = playerDTO.get_status()
    self.vel = playerDTO.get_vel()
    self.maxBomb = playerDTO.get_maxBomb()
    self.recoverTime = playerDTO.get_recoverTime()
    self.listBomb = []
    for bombDTO in playerDTO.get_ListBomb():
        self.listBomb.append(bombConvert().toBomb(bombDTO))
```

Được sử dụng để cập nhật thông tin của đối tượng Player từ một đối tượng DTO (Data Transfer Object).

playerDTO: Là một đối tượng chứa thông tin của người chơi, bao gồm vị trí, trạng thái, tốc độ, số lượng bom, thời gian hồi phục và danh sách bom.

self.rect.centerx: Cập nhật vị trí trục x của người chơi từ playerDTO.

self.rect.centery: Cập nhật vị trí trục y của người chơi từ playerDTO.

self.status: Cập nhật trạng thái của người chơi từ playerDTO.

self.vel: Cập nhật tốc độ của người chơi từ playerDTO.

self.maxBomb: Cập nhật số lượng bom tối đa của người chơi từ playerDTO.

self.recoverTime: Cập nhật thời gian hồi phục của người chơi từ playerDTO.

self.listBomb: Khởi tạo danh sách bom của người chơi từ playerDTO, bằng cách chuyển đổi từng đối tượng bombDTO trong danh sách bom của playerDTO sang đối tượng bom và thêm vào danh sách listBomb của người chơi.

5.2.13 Khởi tạo hiệu ứng nổ bom

```
def __init__(self, surfaceBombBang):
    self.rect = 1000
    self.surfaceBombBang = surfaceBombBang
    self.bombExplodeSize = 1
```

Khởi tạo của một lớp bombBang trong game.

surfaceBombBang: Đây là hình ảnh hoặc bề mặt đại diện cho hiệu ứng nổ bom.

rect: Đây là vị trí ban đầu của hiệu ứng nổ bom trên màn hình. Trong đoạn code này, nó được đặt là 1000, có thể là một giá trị không hợp lệ.

bombExplodeSize: Đây là kích thước của hiệu ứng nổ bom, thường được sử dụng để xác định phạm vi tác động của hiệu ứng nổ. Trong đoạn code này, giá trị ban đầu là 1.

5.2.14 Vẽ hiệu ứng nổ bom

```
def drawBombBang(self, screen):
    screen.blit(self.surfaceBombBang, self.rect)
```

Hàm drawBombBang dùng để vẽ hiệu ứng nổ bom lên màn hình.

screen: Màn hình pygame.

self.surfaceBombBang: Hình ảnh của hiệu ứng nổ bom.

self.rect: Vị trí của hiệu ứng nổ bom trên màn hình.

5.2.15 Thiết lập tọa độ cho hiệu ứng nổ bom

```
def setRectCenterx(self, value):
    self.rect.centerx=value

def setRectCentery(self, value):
    self.rect.centery=value

def setRectCenterX_Y(self, centerx, centery):
    self.rect = self.surfaceBombBang.get_rect(center=(centerx, centery))
```

Phương thức setRectCenterx(self, value) được sử dụng để đặt tọa độ x của tâm của hình chữ nhật.

Phương thức setRectCentery(self, value) được sử dụng để đặt tọa độ y của tâm của hình chữ nhật.

Phương thức setRectCenterX_Y(self, centerx, centery) đặt tọa độ của tâm của hình chữ nhật theo hai giá trị centerx và centery cho trước.

5.2.16 Quản lý hiệu ứng nổ bom

```
def setRect(self, rect):
    self.rect = rect

def getRect(self):
    return self.rect

def hide(self):
    self.rect = self.surfaceBombBang.get_rect(center=(-1000, 0))
```

Hàm setRect: Thiết lập tọa độ và kích thước cho hiệu ứng nổ bom.

Hàm getRect: Trả về hình chữ nhật đại diện cho hiệu ứng nổ bom.

Hàm hide: Ẩn hiệu ứng nổ bom bằng cách đặt tọa độ ngoài màn hình.

5.2.17 Kiểm tra va chạm với người chơi

```
def areCollidingPlayer(self, obj):
    if self.rect.colliderect(obj):
        return True
    return False
```

Hàm areCollidingPlayer kiểm tra xem 2 đối tượng này có va chạm với nhau không. Đối tượng được truyền vào qua tham số obj.

Nếu hai đối tượng va chạm, tức là phần diện tích của chúng giao nhau, hàm trả về True, ngược lại trả về False.

5.2.18 Khởi tạo trường đánh và các đối tượng trong trò chơi

```
def __init__(self, groundMatrix, player, player2, bombSurface, boxGoSurface, boxSatSurface):
    self.groundMatrix = groundMatrix
    self.player = player
    self.player2 = player2
    self.bombSurface = bombSurface
    self.boxGoSurface = boxGoSurface
    self.boxSatSurface = boxSatSurface
    self.boxSize = 50
    self.groundRect = [[0 for _ in range(17)] for _ in range(17)]
    self.listBomb = []
    self.listBomb.append(player.getListBomb())
    self.listBomb.append(player2.getListBomb())
    self.rectTemp = self.boxGoSurface.get_rect(center=(-1000, 0))
```

Phương thức khởi tạo (__init__):

- groundMatrix: Là ma trận đại diện cho mặt đất trong trò chơi. Mỗi phần tử của ma trận này đại diện cho một ô trên mặt đất, và có thể là các giá trị như 'g' (mặt đất), 's' (hộp không di chuyển được), '-' (ô trống) và các giá trị khác tùy thuộc vào quy ước của trò chơi.
- player: Đối tượng của người chơi 1.

- player2: Đối tượng của người chơi 2.
- bombSurface: Là bề mặt của quả bom.
- boxGoSurface: Là bề mặt của hộp di chuyển được.
- boxSatSurface: Là bề mặt của hộp không di chuyển được.
- boxSize: Kích thước của mỗi ô trên mặt đất, cũng là kích thước của hộp.
- groundRect: Một mảng hai chiều, chứa các hình chữ nhật đại diện cho từng ô trên mặt đất. Đối tượng này được sử dụng để vẽ hình chữ nhật đại diện cho mặt đất trên màn hình.
- listBomb: Danh sách chứa tất cả các quả bom trong trò chơi, bao gồm cả quả bom của cả hai người chơi.
- rectTemp: Là hình chữ nhật tạm thời được sử dụng trong quá trình thực hiện các thao tác về hộp.

5.2.19 Vẽ trường đánh

```
def drawBattleGround(self, screen):
    for i in range(len(self.groundMatrix)):  # Duyệt qua các hàng
        for j in range(len(self.groundMatrix)):  # Duyệt qua các cột của hàng đó
            value = self.groundMatrix[i][j]
            if value == 's':
                self.groundRect[i][j] = self.boxSatSurface.get_rect(center=((j+1)*self.boxSize,(i+1)*self.boxSize))
                self.drawBox(self.boxSatSurface, self.groundRect[i][j], screen)
            if value == 'g':
                self.groundRect[i][j] = self.boxGoSurface.get_rect(center=((j+1)*self.boxSize, (i+1)*self.boxSize))
                self.drawBox(self.boxGoSurface, self.groundRect[i][j], screen)
            if value == '-':
                if self.groundRect[i][j] !=0:
                    self.groundRect[i][j].centerx=-1000
            # Lấy giá trị và chỉ mục của phần tử tại hàng i, cột j
            index = (i, j)
    self.drawObject(screen)
```

Hàm drawBattleGround nhận một màn hình (screen) làm đối số và vẽ trường đánh trên màn hình đó.

Hai vòng lặp được sử dụng để duyệt qua từng ô của ma trận groundMatrix, biểu diễn các ô trên trường đánh.

Vòng lặp bên ngoài duyệt qua các hàng của ma trận.

Vòng lặp bên trong duyệt qua các cột của hàng hiện tại.

Trong mỗi vòng lặp, giá trị của từng ô trong ma trận được kiểm tra: Nếu giá trị là 's' (đại diện cho ô chứa hộp sắt), hàm drawBox được gọi để vẽ hộp sắt tại vị trí tương ứng trên màn hình.

Nếu giá trị là 'g' (đại diện cho ô trống), hàm drawBox được gọi để vẽ ô trống tại vị trí tương ứng.

Nếu giá trị là '-' (đại diện cho ô không có gì), vị trí của ô trên màn hình được đặt xa ra (x = -1000) để ẩn nó đi.

Sau khi duyệt qua toàn bộ ma trận, hàm drawObject được gọi để vẽ các đối tượng (người chơi và bom) trên trường đánh.

5.2.20 Xóa ô hộp khỏi ma trận và màn hình

```
def removeBox(self, i, j, screen):
    if self.groundMatrix[i][j]=='g':
        # print(self.groundRect[i][j].centerx, '-', self.grou
        self.groundRect[i][j] = self.rectTemp
        screen.blit(self.boxGoSurface, self.groundRect[i][j])
        self.groundMatrix[i][j] = '-'
```

Xử lý việc loại bỏ ô hộp (box) khỏi ma trận đất (ground matrix) và cập nhật trạng thái của ma trận và hiển thị trên màn hình.

removeBox(self, i, j, screen): Phương thức này nhận các tham số i và j, đại diện cho vị trí của ô trong ma trận đất, và screen, đại diện cho màn hình trò chơi.

Nếu giá trị của ô trong ma trận là 'g' (ô chứa hộp):

self.groundRect[i][j] = self.rectTemp: Cập nhật hình dạng của ô chứa hộp về giá trị mặc định (self.rectTemp).

screen.blit(self.boxGoSurface, self.groundRect[i][j]): Vẽ hộp trên màn hình với hình

dạng mới.

self.groundMatrix[i][j] = '-': Cập nhật giá trị của ô trong ma trận về '-' để chỉ ra rằng ô này không còn chứa hộp nữa.

5.2.21 Vẽ các phần tử trong trò chơi

```
def drawBox(self, boxSurface, boxRect, screen):
    # Vẽ Box
    screen.blit(boxSurface, boxRect)

def drawObject(self, screen):
    # Vẽ Bomb
    for bomb in self.player.getListBomb():
        if bomb.getStatus() <= 1:
            bomb.draw(screen)
        else:
            bomb.drawBombBang(screen)
    for bomb in self.player2.getListBomb():
        if bomb.getStatus() <= 1:
            bomb.draw(screen)
        else:
            bomb.drawBombBang(screen)

    # Vẽ Player
    self.drawPlayer(screen)
```

Tất cả các hình vẽ trong trò chơi, bao gồm cả các ô đất, quả bom và người chơi, được vẽ ra trên màn hình trong hàm drawObject.

Trong hàm drawBox, mỗi ô đất hoặc ô box được vẽ ra trên màn hình bằng cách sử dụng phương thức blit của pygame.

Hàm drawObject sau đó gọi draw của mỗi quả bom trong danh sách bom của cả hai người chơi. Nếu trạng thái của quả bom là nhỏ hơn hoặc bằng 1, nghĩa là nó đang chờ hoặc đang đếm ngược để nổ, thì quả bom được vẽ ra bằng hàm draw của quả bom. Ngược lại, nếu trạng thái là lớn hơn 1, nghĩa là quả bom đã nổ hoặc đang nổ, thì hàm drawBombBang được gọi để vẽ hiệu ứng nổ.

Cuối cùng, hàm drawObject cũng vẽ ra người chơi bằng cách gọi hàm drawPlayer.

5.2.22 Chỉ số vị trí của người chơi trong ma trận

```
def getPositionPlayerInMatrix(self):
    rectPlayer = self.player.getRect()
    indexI = round(rectPlayer.centery/50)
    indexJ = round(rectPlayer.centerx/50)
    return indexI, indexJ
```

Hàm getPositionPlayerInMatrix trả về vị trí của người chơi trong ma trận của trò chơi.

Đầu tiên, nó lấy hình chữ nhật biểu diễn người chơi bằng cách gọi phương thức getRect() từ đối tượng player.

Tiếp theo, nó tính chỉ số hàng indexI bằng cách chia tọa độ y của trung tâm của hình chữ nhật cho kích thước ô (50 pixel).

Tương tự, chỉ số cột indexJ được tính bằng cách chia tọa độ x của trung tâm của hình chữ nhật cho kích thước ô.

Cuối cùng, hàm trả về cặp chỉ số hàng và cột của người chơi trong ma trận.

5.2.23 Tính vị trí đối tượng trong ma trận

```
def getPositionObjectInMatrix(self, ObjectRect):
    indexI = round(ObjectRect.centery/50)
    indexJ = round(ObjectRect.centerx/50)
    return indexI, indexJ
```

Hàm này nhận vào một hình chữ nhật đại diện cho một đối tượng và trả về chỉ số hàng và cột tương ứng của nó trong ma trận game. Đối tượng này có thể là một ô vuông trên màn hình hoặc một đối tượng khác như bom hoặc người chơi.

Cách tính chỉ số hàng và cột:

- Đối với chỉ số hàng (indexI): Lấy tọa độ y của tâm của hình chữ nhật và chia cho kích thước của ô vuông.

- Đối với chỉ số cột (indexJ): Lấy tọa độ x của tâm của hình chữ nhật và chia cho kích thước của ô vuông.

Sau đó, hàm trả về cặp chỉ số hàng và cột của đối tượng trong ma trận game.

5.2.24 Debugging và In thông tin ma trận

```
def printTest(self):
    print("GroundMatrix: \n")
    for i in self.groundMatrix:
        for j in i:
            print(j, end=",")
        print()
    print("GroundRect:")
    for i in self.groundRect:
        for j in i:
            print(j.centerx, end=",")
        print()
```

In ra ma trận và các hình chữ nhật tương ứng trong ma trận.

Vòng lặp đầu tiên (dòng 2-7) in ra các giá trị của ma trận groundMatrix. Mỗi hàng của ma trận được in trên một dòng và các giá trị của mỗi hàng được phân tách bằng dấu phẩy.

Vòng lặp thứ hai (dòng 9-14) in ra tọa độ trung tâm của các hình chữ nhật trong groundRect. Mỗi hàng của groundRect tương ứng với một hàng trong ma trận, và các tọa độ trung tâm của các hình chữ nhật trong hàng đó được in trên cùng một dòng và phân tách bằng dấu phẩy.

Đây là một hàm debug, giúp theo dõi và kiểm tra dữ liệu trong quá trình phát triển ứng dụng.

5.2.25 Animate Bom

```
def animate(self):
    if self.surfaceIndex == 0:
        self.rect.centery -=3
        self.surfaceIndex = 1
    else:
        self.rect.centery +=3
        self.surfaceIndex = 0
    self.surface = self.surfaceList[self.surfaceIndex]
```

Tạo hiệu ứng chuyển động cho bom trước khi nổ.

Phương thức animate thực hiện việc thay đổi vị trí và hình ảnh của bom.

Nếu surfaceIndex là 0, tức là đang sử dụng hình ảnh đầu tiên trong danh sách surfaceList, bom sẽ di chuyển lên trên (self.rect.centery -= 3) và chuyển sang hình ảnh thứ hai trong danh sách (self.surfaceIndex = 1).

Nếu surfaceIndex là 1, tức là đang sử dụng hình ảnh thứ hai trong danh sách surfaceList, bom sẽ di chuyển xuống dưới (self.rect.centery += 3) và chuyển sang hình ảnh đầu tiên trong danh sách (self.surfaceIndex = 0).

Cuối cùng, cập nhật surface của bom thành hình ảnh mới tương ứng với surfaceIndex.

5.2.26 Vẽ các đối tượng lên màn hình

```
def draw(self, screen):
    screen.blit(self.surface, self.rect)

def drawBombBang(self, screen):
    for BombBang in self.bombBang:
        BombBang.drawBombBang(screen)
```

Phương thức draw: Dùng để vẽ bom lên màn hình. Nó nhận đối số screen, là đối tượng màn hình pygame, và sử dụng phương thức blit để vẽ hình ảnh của bom lên vị trí được chỉ định bởi thuộc tính rect.

Phương thức drawBombBang: Dùng để vẽ các hiệu ứng nổ bom lên màn hình. Nó

nhận đối số screen và sử dụng một vòng lặp để vẽ mỗi hiệu ứng nổ bom trong danh sách bombBang. Mỗi hiệu ứng nổ bom được vẽ bằng cách gọi phương thức drawBombBang của đối tượng hiệu ứng nổ bom đó.

5.2.27 Các phương thức đặt bom

```
def placeABomb(seft, playerRect):
    seft.status = 1
    seft.rect = playerRect

def setSurface_Rect(self, playerRect, indexSurface):
    self.rect = playerRect
    self.surface = self.surfaceList[indexSurface]

def setStatus(self, status):
    self.status = status
    if status == 1:
        self.timer=0
```

Định nghĩa một số phương thức trong một lớp (hoặc một đối tượng) để thực hiện các hành động liên quan đến việc đặt bom trong trò chơi.

placeABomb: Phương thức này được sử dụng để đặt bom vào vị trí của người chơi. Khi bom được đặt, trạng thái của bom sẽ chuyển sang 1, đồng thời cập nhật vị trí của bom bằng vị trí của người chơi.

setSurface_Rect: Phương thức này cập nhật vị trí và hình ảnh của bom. Nó nhận vào vị trí của người chơi và chỉ số của hình ảnh bom trong danh sách hình ảnh của bom.

setStatus: Phương thức này được sử dụng để đặt trạng thái của bom. Khi trạng thái được đặt thành 1 (đã đặt bom), đồng hồ đếm thời gian của bom được đặt lại.

5.2.28 Kích hoạt vụ nổ của bomb

```
def Bang(self):
    self.status = 2
    i=0
    for bang in self.bombBang:
        if i==0:
            bang.setRectCenterX_Y(self.getRect().centerx,self.getRect().centery-35*self.bombBangSize)
        elif i==1:
            bang.setRectCenterX_Y(self.getRect().centerx,self.getRect().centery+35*self.bombBangSize)
        elif i==2:
            bang.setRectCenterX_Y(self.getRect().centerx-35*self.bombBangSize,self.getRect().centery)
        elif i==3:
            bang.setRectCenterX_Y(self.getRect().centerx+35*self.bombBangSize,self.getRect().centery)
        i+=1
```

Phương thức Bang trong đoạn mã này được sử dụng để kích hoạt bomb, khi nó đã đạt đến thời gian nổ. Cách nó hoạt động:

Thay đổi trạng thái của bomb: Đặt trạng thái của bomb thành 2 để chỉ ra rằng nó đã nổ.

Tạo và đặt vị trí cho các vụ nổ của bomb: Duyệt qua danh sách các vụ nổ của bomb và đặt vị trí cho mỗi vụ nổ dựa trên vị trí hiện tại của bomb và kích thước vụ nổ.

- Nếu i == 0, vụ nổ đặt ở phía trên bomb.
- Nếu i == 1, vụ nổ đặt ở phía dưới bomb.
- Nếu i == 2, vụ nổ đặt ở bên trái của bomb.
- Nếu i == 3, vụ nổ đặt ở bên phải của bomb.

Kết quả là, khi bomb nổ, nó sẽ tạo ra các vụ nổ ở các hướng khác nhau xung quanh vị trí của nó.

5.2.29 Ẩn bomb sau khi nổ

```
def hide(self):
    self.rect = self.surface.get_rect(center=(-1000, 0))
    for bang in self.bombBang:
        bang.hide()
    self.status = 0
```

Hàm này được sử dụng để ẩn bomb khỏi màn hình khi nó đã nổ.

Đầu tiên, nó thiết lập lại vị trí của bomb ra xa màn hình bằng cách đặt centerx của rect của nó thành -1000. Điều này khiến cho bomb không còn được vẽ trên màn hình nữa.

Tiếp theo, nó gọi hàm hide() trên các vụ nổ của bomb (nếu có), để cũng ẩn chúng khỏi màn hình.

Cuối cùng, nó đặt trạng thái của bomb thành 0 để chỉ đánh dấu rằng bomb đã bị ẩn và không còn hoạt động nữa.

5.2.30 Tạo hiệu ứng nổ của bom

```
def setUpSurfaceBombBang(self):
    for i in range(1, 5):
        arrayTemp = []
        if i==1:
            direction = 'up'
        if i==2:
            direction = 'down'
        if i==3:
            direction = 'left'
        if i==4:
            direction = 'right'
        for j in range(1, 11):
            skin = pygame.image.load('./img/Bomb/bombbang_'+str(direction)+''+str(j)+'.png')
            if i==1 or i==2:
                arrayTemp.append(pygame.transform.scale(skin, (50,45*(j+1))))
            else:
                arrayTemp.append(pygame.transform.scale(skin, (45*(j+1),50)))
        self.bombBang_Surface.append(arrayTemp)
```

Hàm setUpSurfaceBombBang tạo ra các hình ảnh để vẽ hiệu ứng nổ của bom. Mỗi

hiệu ứng nổ sẽ có một hướng cụ thể (lên, xuống, trái, phải) và sẽ bao gồm một chuỗi các frame hoạt hình để tạo ra hiệu ứng chuyển động.

Đầu tiên, vòng lặp for chạy từ 1 đến 4 để tạo ra hiệu ứng nổ cho mỗi hướng (lên, xuống, trái, phải). Trong mỗi vòng lặp, biến direction được gán giá trị tương ứng với hướng nổ: 'up', 'down', 'left', 'right'.

Tiếp theo, vòng lặp for tiếp theo chạy từ 1 đến 10 để tạo ra 10 frame hoạt hình cho mỗi hướng.

Mỗi frame hoạt hình được tạo bằng cách tải hình ảnh từ file có định dạng 'bomb-bang_directionX.png', trong đó X là chỉ số của frame.

Kích thước của các hình ảnh được điều chỉnh tùy thuộc vào hướng nổ: các frame ở hướng dọc (lên, xuống) có chiều cao tăng dần, còn các frame ở hướng ngang (trái, phải) có chiều rộng tăng dần.

Mỗi list các frame hoạt hình tương ứng với một hướng nổ sẽ được lưu vào list self.bombBang_Surf
để sử dụng sau này khi vẽ hiệu ứng nổ của bom.

5.2.31 Khởi tạo kết nối và luồng xử lý cho client

```
currentPlayer = 0
while True:
    conn, addr = s.accept()
    print("Connected to:", addr)

    start_new_thread(threaded_client, (conn, currentPlayer))
    currentPlayer += 1
```

Thực hiện việc chấp nhận kết nối từ client và khởi tạo một luồng xử lý cho mỗi kết nối. Biến currentPlayer được sử dụng để gán một chỉ mục duy nhất cho mỗi client kết nối, để mỗi client có thể tương tác với trận đấu của họ một cách độc lập.

s.accept(): Chờ đợi và chấp nhận một kết nối từ client. Trả về một cặp giá trị conn (một đối tượng socket mới để giao tiếp với client) và addr (địa chỉ của client).

print("Connected to:", addr): In ra địa chỉ của client đã kết nối.

start_new_thread(threaded_client, (conn, currentPlayer)): Khởi tạo một luồng mới để xử lý client vừa kết nối. Tham số đầu tiên là hàm threaded_client, tham số thứ hai là một tuple chứa conn (đối tượng socket) và currentPlayer (chỉ mục của người chơi hiện tại).

currentPlayer += 1: Tăng chỉ mục của người chơi hiện tại để sử dụng cho kết nối tiếp theo.

5.2.32 Giao tiếp Client-Server cho Game nhiều người chơi

```
def threaded_client(conn, battleGround_Idex):
    conn.send(pickle.dumps(battleGrounds[battleGround_Idex]))
    reply = ""
    while True:
        try:
            data = pickle.loads(conn.recv(2048))
            battleGrounds[battleGround_Idex] = data

            if not data:
                print("Disconnected")
                break
            else:
                if battleGround_Idex == 1:
                    reply = battleGrounds[0]
                else:
                    reply = battleGrounds[1]

            conn.sendall(pickle.dumps(reply))
        except:
            break

    print("Lost connection")
    conn.close()
```

Hàm này quản lý giao tiếp giữa server và client để cập nhật và đồng bộ hóa trạng thái của battleground trong một trò chơi nhiều người chơi

Hàm này nhận hai đối số: conn, là socket kết nối, và battleGround_Idex, một chỉ mục chỉ định battleground nào được xử lý.

Nó gửi trạng thái hiện tại của battleground được chỉ định bởi battleGround_Idex tới client. Hàm pickle.dumps() được sử dụng để serialize dữ liệu trước khi gửi đi.

Nó bắt đầu một vòng lặp trong đó liên tục lắng nghe dữ liệu từ client.

Khi nhận được dữ liệu từ client, nó deserialize dữ liệu đó bằng cách sử dụng pickle.loads() và cập nhật trạng thái battleground tương ứng.

Nếu dữ liệu nhận được từ client là trống, nó chỉ ra rằng client đã ngắt kết nối, và vòng lặp sẽ kết thúc.

Nếu client không ngắt kết nối, nó chuẩn bị một phản hồi dựa trên trạng thái hiện tại của battleground khác. Nếu battleGround_Idex là 1, nó sẽ trả lời với trạng thái của battleground 0, và ngược lại.

Phản hồi được gửi trở lại cho client bằng cách sử dụng conn.sendall(pickle.dumps(reply)).

Nếu xảy ra một ngoại lệ trong quá trình giao tiếp với client, nó sẽ thoát khỏi vòng lặp và đóng kết nối.

Cuối cùng, nó in một thông báo chỉ ra rằng kết nối đã bị mất và đóng kết nối lại.

5.2.33 Khởi tạo Client Socket

```
def __init__(self):
    self.client = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    self.server = SystemVariable.serverIP
    self.port = SystemVariable.port
    self.addr = (self.server, self.port)
    self.battleGround = self.connect()
```

Hàm này khởi tạo một client socket để kết nối đến một server socket với địa chỉ và cổng nhất định.

Tạo một đối tượng socket cho client với kiểu AF_INET (đại diện cho IPv4) và SOCK_STREAM (đại diện cho TCP socket).

Lấy địa chỉ IP của server từ biến serverIP trong một module hoặc class khác.

Lấy số cổng của server từ biến port trong cùng module hoặc class đó.

Tạo tuple chứa địa chỉ IP và số cổng của server.

Gọi phương thức connect() để thiết lập kết nối tới server và nhận dữ liệu về trạng thái của battleground.

5.2.34 Kết nối đến server và nhận dữ liệu từ server

```
def connect(self):
    try:
        self.client.connect(self.addr)
        return pickle.loads(self.client.recv(2048))
    except:
        pass
```

Phương thức connect trong lớp này được sử dụng để kết nối với server.

Đầu tiên, nó cố gắng kết nối đến địa chỉ được chỉ định (self.addr) bằng cách sử dụng phương thức connect của socket.

Nếu kết nối thành công, nó nhận dữ liệu từ server bằng cách gọi self.client.recv(2048). Dữ liệu nhận được được giải pickled (deserialize) bằng cách sử dụng pickle.loads.

Nếu có bất kỳ lỗi nào xảy ra trong quá trình này (ví dụ: kết nối thất bại hoặc không thể nhận dữ liệu), nó chỉ đơn giản là bỏ qua và tiếp tục.

5.2.35 Phương thức gửi dữ liệu và nhận phản hồi

```
def send(self, data):
    try:
        self.client.send(pickle.dumps(data))
        return  pickle.loads(self.client.recv(2048))
    except socket.error as e:
        print(e)
```

Phương thức send dùng để gửi dữ liệu lên máy chủ và nhận phản hồi từ máy chủ. Nó nhận một đối số là dữ liệu cần gửi, sau đó nó sẽ thực hiện các bước sau:

Gửi dữ liệu đã được đóng gói bằng pickle lên máy chủ.

Nhận phản hồi từ máy chủ, cũng đã được đóng gói bằng pickle.

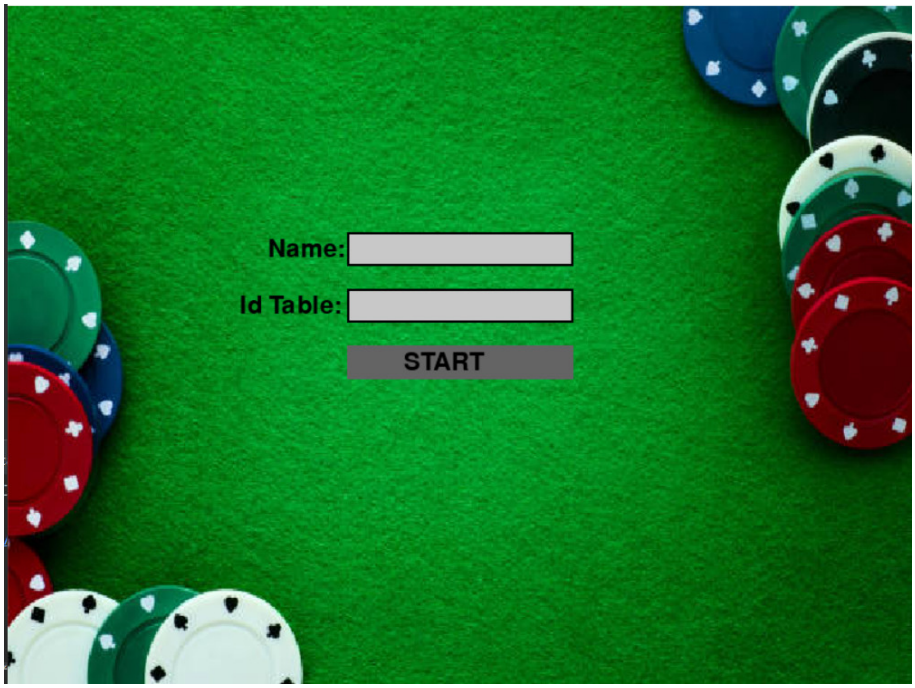
Trả về dữ liệu đã được giải nén từ máy chủ.

Nếu có lỗi xảy ra trong quá trình gửi hoặc nhận, nó sẽ in ra lỗi đó.

CHƯƠNG 6. GIAO DIỆN GAME

6.1 Giao diện Game Poker

6.1.1 Giao diện khi bắt đầu vào Game

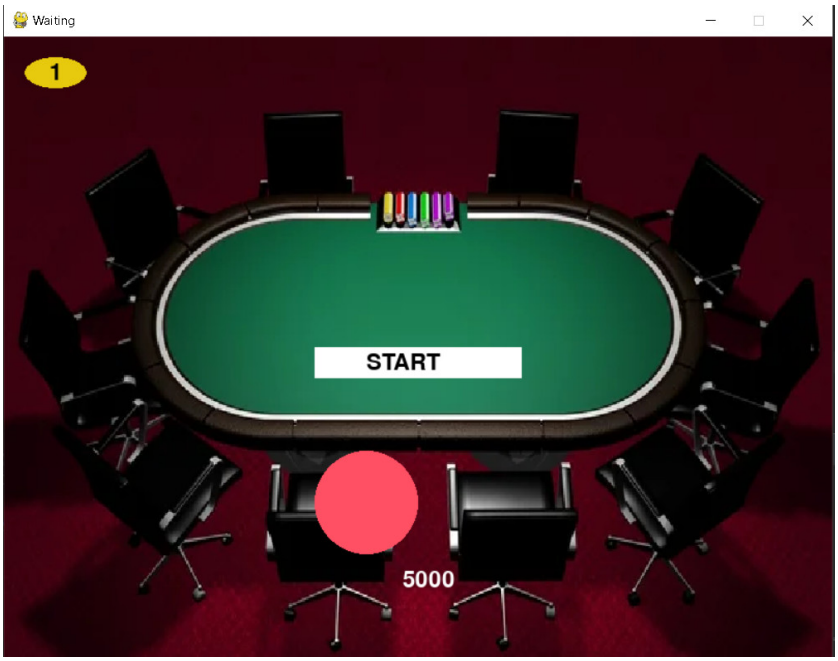


Người chơi sẽ phải thực hiện nhập tên và mã phòng để tham gia vào trò chơi, sau khi nhấn vào nút "START" thì người chơi sẽ đựng đưa vào phòng chờ.

6.1.2 *Giao diện màn hình đợi*

Mỗi phòng sẽ có tối đa là 5 người chơi.

6.1.3 *Giao diện màn hình host khi có hai người chơi trở lên*



6.1.4 Giao diện màn hình vô game trong lượt

6.1.5 Giao diện màn hình flop

6.1.6 Giao diện màn hình turn

6.1.7 Giao diện màn hình khi rise

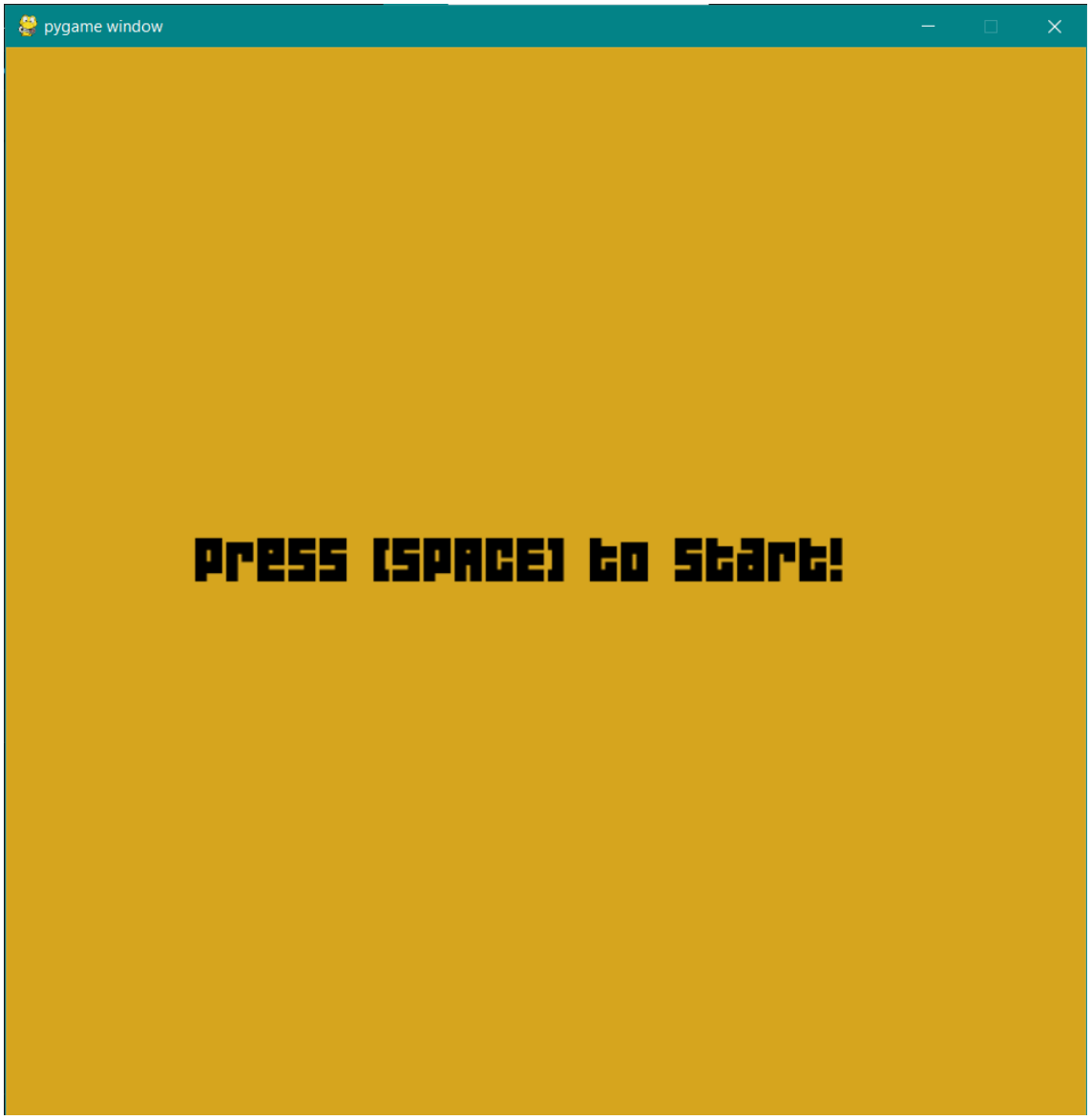
6.1.8 Giao diện màn hình 5 lá

6.1.9 Giao diện màn hình thông báo người chơi win và show bài

6.1.10 Giao diện màn hình fold

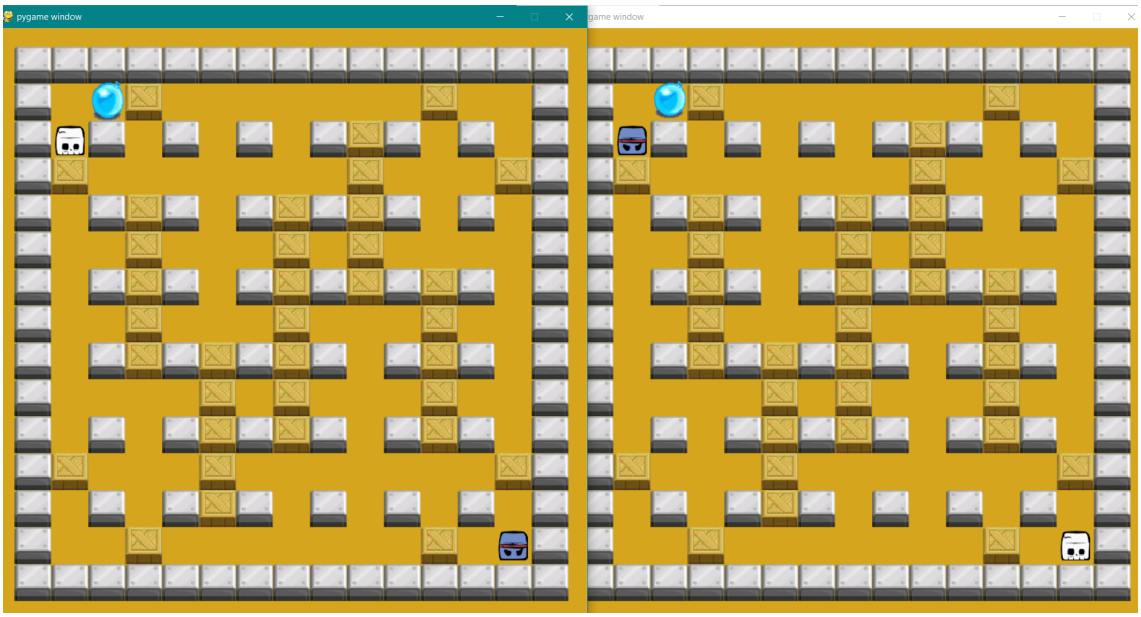
6.2 Giao diện Game đặt Boom

6.2.1 Giao diện khi bắt đầu vào Game



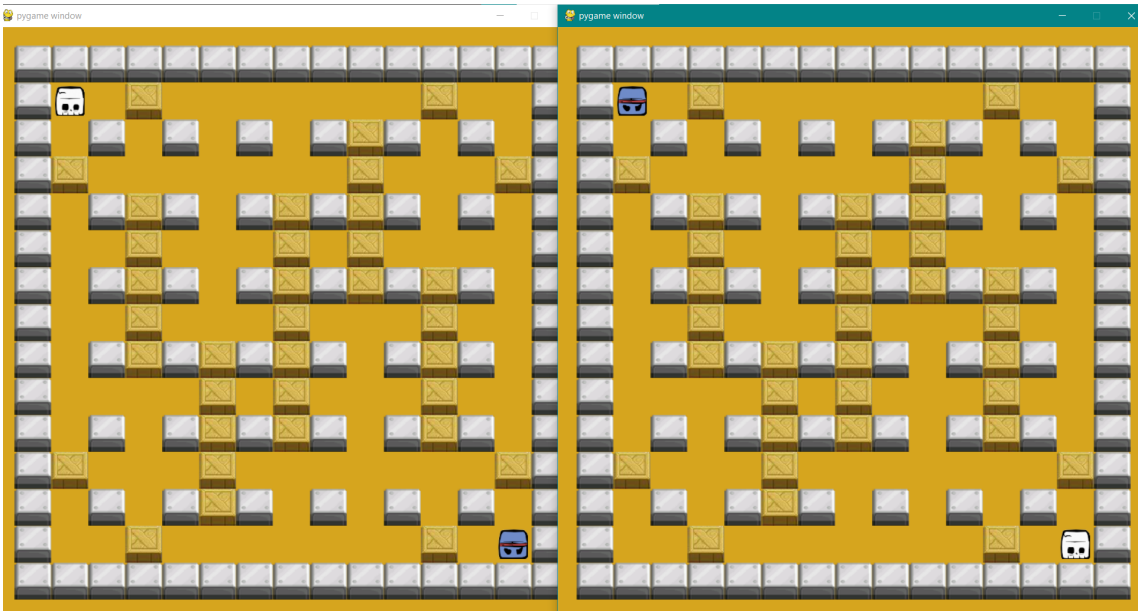
Đây là giao diện khi bắt đầu game.

6.2.2 Giao diện đặt Boom



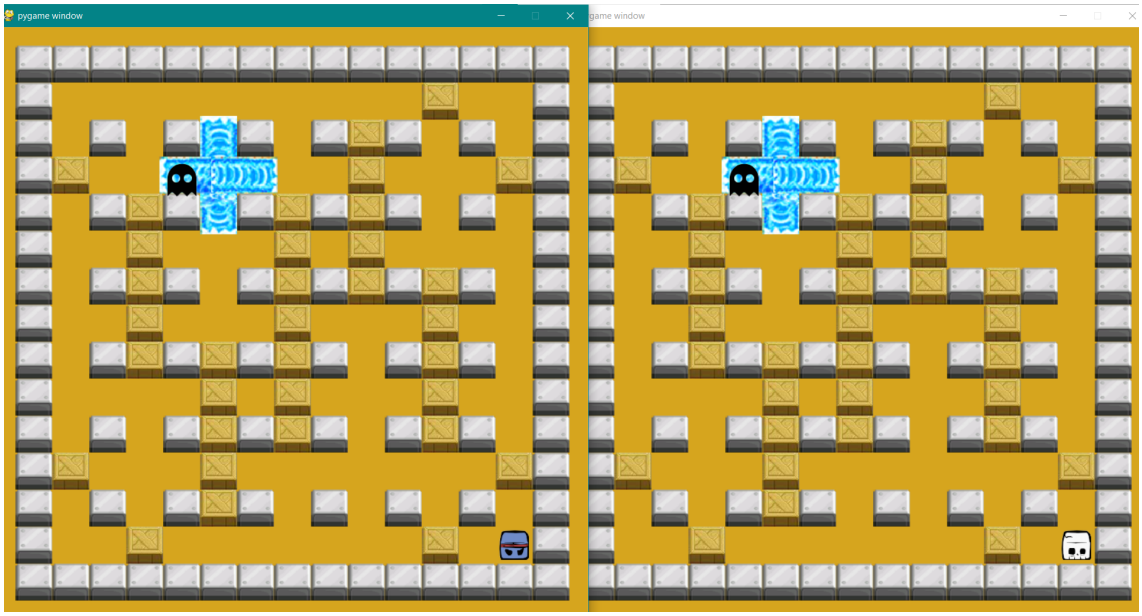
Đây là phần của trò chơi nơi người chơi có thể đặt bom tại vị trí hiện tại của họ trên bản đồ.

6.2.3 Giao diện game 2 người chơi



Đây là chế độ chơi của trò chơi mà hai người chơi có thể tham gia cùng một lúc và cạnh tranh với nhau.

6.2.4 Giao diện khi Boom nổ và người chơi chết



Khi bom nổ, nó có thể gây ra sự kiện người chơi chết nếu họ ở trong phạm vi tác động của nó.

CHƯƠNG 7. TÀI LIỆU THAM KHẢO

1. https://www.pygame.org/wiki/Contribute
2. https://vi.wikipedia.org/wiki/Pygame
3. https://hostingviet.vn/socket-la-gi