

TRƯỜNG ĐẠI HỌC SÀI GÒN
KHOA CÔNG NGHỆ THÔNG TIN



PHÁT TRIỂN PHẦN MỀM MÃ NGUỒN MỞ

Đồ án cuối kỳ

Xây dựng game Caro 3x3 trên PYTHON

GVHD: Từ Lãng Phiêu
SV: Trần Nguyễn Việt Thái - 3120560088
Phan Thái Hòa - 3120410176
Dỗ Minh Hiếu - 3120410167
Nguyễn Đức Nhâm - 3120410356

TP. HỒ CHÍ MINH, THÁNG 5/2024

Mục lục

1	Tổng quan đề tài	3
1.1	Dặt vấn đề	3
1.2	Mục tiêu	3
1.3	Phạm vi của ứng dụng	4
2	Giới thiệu về các công nghệ sử dụng	4
2.1	Ngôn ngữ lập trình Python	4
2.1.1	Lịch sử và nguồn gốc của python	4
2.1.2	Điểm nổi bật của ngôn ngữ lập trình Python	5
2.1.3	Ưu nhược điểm	5
2.1.3.a	Ưu điểm	5
2.1.3.b	Nhược điểm	6
2.1.3.c	Kết luận	6
2.2	Các thư viện dùng trong đồ án	6
2.2.1	Tkinter	6
2.2.2	Threading	6
2.3	Python WebSocket	7
2.3.1	Nguyên lý hoạt động cơ bản của Socket	7
2.3.2	7
2.4	Môi trường làm việc	8
2.4.1	Pycharm	8
2.4.2	Visual Studio Code	8
3	Phân tích và xây dựng game	9
3.1	Giới thiệu về trò chơi Tic tac toe	9
3.2	Phân tích luật chơi đơn giản Tic tac toe	9
3.3	Xây dựng game	10
3.3.1	Xây dựng đối tượng Bàn cờ	10
3.3.2	Cài đặt giao tiếp giữa Server socket và Client socket	14
3.3.3	Mô phỏng các nước đi giữa 2 player	17
3.3.4	Các kịch bản khi kết thúc ván chơi	19
3.3.5	Xây dựng GUI trò chơi	21
3.4	Flow chart	23
4	Thiết kế giao diện	24
4.1	Player 1	24
4.2	Player2	25
5	Cài đặt và khởi chạy game	28
5.1	Hướng dẫn cài đặt game	28
5.2	Khởi động game	31
5.3	Kết quả chạy game	34
6	Kết luận	36
6.1	Kết quả thu được	36
6.2	Ưu và nhược điểm	36
6.3	Hướng phát triển nâng cấp tương lai	36



Danh sách hình vẽ

1	Đối tượng Board	11
2	Các hàm xử lý logic của Bàn cờ	12
3	Hàm kiểm tra state trò chơi	13
4	Hàm computeStats	14
5	Tạo Server socket phía player1	14
6	Hàm clickConnect()	15
7	Hàm acceptConnection	15
8	Hàm rececive() của player2	16
9	Hàm rececive() của player1	16
10	Hàm createThread()	17
11	Hàm receciveMove()	17
12	Hàm play bên player1()	18
13	Tạo btn các ô cờ trên UI	18
14	Các hàm nhận sự kiện click của player	18
15	Hàm play bên player2()	19
16	Hàm receiveMove() bên player1	19
17	Hàm reset()	20
18	Hàm gameOver()	20
19	Hàm clickQuit()	20
20	Xây dựng đối tượng cửa sổ màn hình	21
21	Tạo các Frame chia bố cục trên màn hình	21
22	Bản phác thảo các bố cục trên giao diện	21
23	Dựng UI hiển các ô cờ	22
24	Dựng UI hiển thị thông tin player	22
25	Dựng UI hiển thị các tỉ số trận	22
26	Dựng UI các nút lựa chọn cho player	22
27	Dựng UI hiển thị tên trò chơi	23
28	Flow chart của chương trình chạy	23
29	Dialog nhập tên player1	24
30	Các thành phần giao diện của cửa sổ player1	24
31	Dialog thông báo kết nối từ player2	25
32	Giao diện player2 có thêm nút Connect	26
33	Dialog điền ip address	26
34	Dialog điền port	27
35	Dialog điền tên player2	27
36	Quy định ký tự đánh của 2 player	28

Danh sách bảng

1	Bảng thành phần giao diện player 1	25
---	--	----



Lời Mở Đầu

Chúng em xin chân thành gửi lời cảm ơn đến tập thể quý Thầy Cô trong khoa Công nghệ thông tin Trường Đại Học Sài Gòn đã giúp cho chúng em có những kiến thức cơ bản làm nền tảng để thực hiện đề tài này. Đặc biệt, chúng em xin gửi lời cảm ơn và lòng biết ơn sâu sắc nhất tới thầy Từ Lâng Phiêu, giảng viên môn Phát triển phần mềm mã nguồn mở. Thầy đã trực tiếp hướng dẫn tận tình, sửa chữa và đóng góp nhiều ý kiến quý báu giúp nhóm chúng em hoàn thành tốt báo cáo môn học của mình. Trong thời gian một học kỳ thực hiện đề tài, nhóm chúng em đã vận dụng những kiến thức nền tảng đã được tích lũy đồng thời kết hợp với việc học hỏi và nghiên cứu những kiến thức mới. Từ đó nhóm đã vận dụng tối đa những gì đã thu thập được để hoàn tất một báo cáo đồ án tốt nhất.

Tuy nhiên, trong quá trình thực hiện, nhóm chúng em không tránh khỏi những thiếu sót. Chính vì vậy, nhóm chúng em rất mong muốn nhận được những sự góp ý từ phía Thầy, nhằm hoàn thiện những kiến thức để nhóm chúng em có thể học tập thêm và đó cũng là hành trang tốt nhất để chúng em thực hiện tiếp các đề tài khác trong tương lai.

Nhóm chúng em xin chân thành cảm ơn Thầy!

Nhóm 7

1 Tổng quan đề tài

1.1 Đặt vấn đề

Trong các loại trò chơi trí tuệ sử dụng “bàn cờ” phổ biến hiện nay chúng ta có thể kể tên rất nhiều. Trong đó có một trò có thể giết thời gian và giải trí rất hiệu quả với thời gian nhanh – gọn, luật chơi đơn giản dễ làm quen và có thể chơi được hầu như mọi lúc – mọi nơi, đó chính là Tic tac toe. Với cương vị là những sinh viên theo học ngành công nghệ thông tin có hứng thú với xây dựng app, ý tưởng của bọn em là biến một trò chơi tuổi thơ gắn liền với tuổi học trò và trang giấy tập vở này thành một mini game có thể chơi trên môi trường desktop (nhằm tiết kiệm đĩa lăng phí giấy vở).

1.2 Mục tiêu

Mục tiêu của dự án này là phát triển một trò chơi Tic Tac Toe hoàn chỉnh bằng ngôn ngữ lập trình Python. Trò chơi này sẽ không chỉ là một phiên bản đơn giản của Tic Tac Toe, mà còn là một ứng dụng có khả năng cung cấp khả năng kết nối 2 người chơi, cùng với trải nghiệm tốt nhất cho người chơi thông qua các tính năng và giao diện người dùng tối ưu. Với các chức năng tối thiểu như:

- Chơi đối kháng giữa 2 người chơi.
- Chơi lại màn liên tục.



1.3 Phạm vi của ứng dụng

Về giao diện:

- Tạo ra một giao diện người dùng thân thiện và dễ sử dụng, cho phép người chơi tương tác một cách trực quan với trò chơi. Sử dụng thư viện như Tkinter để phát triển giao diện.
- Hiển thị bảng trò chơi, các nút điều khiển và thông báo cho người chơi về kết quả của mỗi trận đấu.

Về dữ liệu:

- Dữ liệu được khởi tạo và cập nhật liên tục giữa 2 người chơi đang đối kháng nhằm đảm bảo tính toàn vẹn và đồng bộ.
- Dữ liệu được dùng và truyền tải chỉ giới hạn vòng đồi trong một phiên chạy chương trình game. Sẽ không lưu cache hay dùng cơ sở dữ liệu.

Về phần xử lý:

- Thực hiện các logic cơ bản nhất của trò chơi Tic tac toe, bao gồm việc kiểm tra chiến thắng, kiểm tra hòa, và cập nhật trạng thái của bảng sau mỗi nước đi của người chơi.
- Ngoài ra còn xử lý việc truyền nhận dữ liệu giao tiếp giữa 2 người chơi nhằm đưa ra các kịch bản phù hợp với mỗi thông điệp.

2 Giới thiệu về các công nghệ sử dụng

2.1 Ngôn ngữ lập trình Python

Python là một trong những ngôn ngữ lập trình phổ biến nhất trên thế giới, với một cộng đồng đông đảo và đa dạng của các nhà phát triển và người sử dụng. Được thiết kế với mục đích đơn giản và dễ sử dụng, Python nhanh chóng trở thành công cụ ưa thích cho nhiều mục đích, từ viết script đến phát triển các ứng dụng web và khoa học dữ liệu.

2.1.1 Lịch sử và nguồn gốc của python

Python được sáng tạo bởi Guido van Rossum, một nhà lập trình người Hà Lan, vào cuối những năm 1980 và đầu những năm 1990. Guido ban đầu bắt đầu dự án Python trong thời gian làm việc tại Trung tâm Nghiên cứu Math và Computer (CWI) ở Amsterdam. Ông cảm thấy cần có một ngôn ngữ lập trình mới, dễ đọc và dễ hiểu hơn, cho các dự án lớn hơn mà ông đang làm việc.

- 1989: Guido van Rossum bắt đầu dự án Python tại CWI.
- 1991: Phiên bản đầu tiên của Python, Python 0.9.0, được phát hành vào tháng 2 năm 1991. Python được thiết kế với cú pháp đơn giản và dễ đọc, cũng như khả năng làm việc với các thư viện C và C++.



- 1994: Python 1.0 được phát hành vào tháng 1 năm 1994, mang lại một số tính năng mới như cơ chế module và hỗ trợ exception.
- 2000: Python 2.0 được phát hành vào tháng 10 năm 2000, với nhiều cải tiến như garbage collection tự động và hỗ trợ Unicode.
- 2008: Python 3.0, phiên bản lớn đầu tiên kể từ Python 2.0, được phát hành vào tháng 12 năm 2008. Python 3.0 không tương thích ngược với Python 2.x, nhưng cung cấp nhiều cải tiến và sửa lỗi.
- Hiện tại: Python 3.x vẫn tiếp tục phát triển và là phiên bản được khuyến khích sử dụng. Python 2.x đã chính thức kết thúc vòng đời vào ngày 1 tháng 1 năm 2020.

Python đã trở thành một trong những ngôn ngữ lập trình phổ biến nhất trên thế giới, được sử dụng rộng rãi trong nhiều lĩnh vực khác nhau như phát triển web, khoa học dữ liệu, trí tuệ nhân tạo, và nhiều hơn nữa.

2.1.2 Điểm nổi bật của ngôn ngữ lập trình Python

- Dễ đọc và dễ hiểu: Python có cú pháp rất đơn giản và gần gũi với ngôn ngữ tự nhiên, làm cho nó dễ đọc và dễ hiểu.
- Đa mục đích: Python có thể được sử dụng cho nhiều mục đích khác nhau, từ viết script đến phát triển ứng dụng web và phần mềm máy tính.
- Hỗ trợ mạnh mẽ cho Mảng Rộng Các Thư Viện: Python có một hệ sinh thái phong phú của các thư viện và framework, giúp giải quyết các vấn đề từ xử lý dữ liệu đến trí tuệ nhân tạo và khoa học dữ liệu.
- Cộng đồng lớn mạnh: Python có một cộng đồng lớn mạnh, nơi người dùng có thể chia sẻ kiến thức, hỏi đáp vấn đề, và đóng góp vào các dự án mã nguồn mở.

2.1.3 Ưu nhược điểm

2.1.3.a Ưu điểm

- Dễ đọc và dễ hiểu: Python có cú pháp đơn giản và gần gũi với ngôn ngữ tự nhiên, giúp giảm thời gian và công sức khi viết mã.
- Đa mục đích: Python là một ngôn ngữ đa mục đích, có thể được sử dụng cho nhiều mục đích khác nhau, từ phát triển web và ứng dụng di động đến khoa học dữ liệu và trí tuệ nhân tạo.
- Hệ sinh thái mạnh mẽ của thư viện và framework: Python có một hệ sinh thái phong phú của các thư viện và framework, giúp giải quyết các vấn đề từ đơn giản đến phức tạp.
- Cộng đồng lớn mạnh và hỗ trợ tích cực: Python có một cộng đồng lớn mạnh, nơi người dùng có thể chia sẻ kiến thức, hỏi đáp vấn đề, và đóng góp vào các dự án mã nguồn mở.
- Tính di động và khả năng tương thích: Python hoạt động trên nhiều hệ điều hành khác nhau và có khả năng tương thích với nhiều ngôn ngữ lập trình khác.
- Dễ học: Python được coi là một trong những ngôn ngữ lập trình dễ học nhất, phù hợp cho cả người mới bắt đầu và người có kinh nghiệm.



2.1.3.b Nhược điểm

- Tốc độ chậm hơn: Python thường không nhanh bằng các ngôn ngữ như C và C++ do việc thông dịch mã nguồn.
- Ghi nhớ dữ liệu lớn: Python có thể gặp khó khăn trong việc xử lý dữ liệu lớn do quản lý bộ nhớ không hiệu quả.
- Khó mở rộng đa luồng: Python không tốt cho các ứng dụng đa luồng và đa xử lý, do GIL (Global Interpreter Lock) hạn chế việc thực thi đồng thời của các luồng.
- Phiên bản không tương thích ngược: Sự không tương thích giữa Python 2 và Python 3 có thể gây khó khăn cho các dự án chuyển đổi từ Python 2 sang Python 3.

2.1.3.c Kết luận

Mặc dù có nhược điểm nhất định, Python vẫn là một ngôn ngữ lập trình mạnh mẽ và phổ biến, được sử dụng rộng rãi trên toàn thế giới. Điều quan trọng là hiểu rõ ưu điểm và nhược điểm của nó để có thể lựa chọn sử dụng một cách hợp lý trong các dự án cụ thể.

2.2 Các thư viện dùng trong đồ án

2.2.1 Tkinter

Tkinter là một giao diện người dùng đồ họa (GUI) tiêu chuẩn cho Python. Nó được tích hợp sẵn trong Python và cung cấp các thành phần để xây dựng các ứng dụng GUI. Trong đồ án, Tkinter sẽ được sử dụng để xây dựng giao diện chơi Caro 3x3. Nhờ vào Tkinter, ta có thể tạo các thành phần như ô cờ, nút chơi, và các hộp thoại để tương tác với người dùng một cách trực quan.

Một số điểm nổi bật của Tkinter:

- Đơn giản và dễ sử dụng: Tkinter cung cấp một giao diện dễ hiểu và cú pháp đơn giản, phù hợp cho cả người mới bắt đầu và người có kinh nghiệm.
- Đa nền tảng: Tkinter hoạt động trên nhiều hệ điều hành khác nhau bao gồm Windows, macOS và Linux, giúp cho việc phát triển ứng dụng đa nền tảng trở nên dễ dàng.
- Thành phần giao diện đầy đủ: Tkinter cung cấp một loạt các thành phần giao diện như nút, hộp văn bản, nhãn, ô nhập liệu, vv., giúp cho việc tạo ra giao diện người dùng đa dạng và phong phú.
- Tích hợp tốt với Python: Tkinter được tích hợp sẵn trong Python, không cần cài đặt thêm, giúp giảm thiểu sự phức tạp và tăng tính tương thích với các phiên bản Python khác nhau.

2.2.2 Threading

Threading là một thư viện trong Python cho phép thực hiện đa luồng, cho phép các tác vụ được thực thi đồng thời. Trong đồ án, ta sẽ sử dụng Threading để xử lý các



tác vụ đa nhiệm, như lắng nghe các yêu cầu từ người chơi, xử lý logic của trò chơi và cập nhật giao diện người dùng một cách song song và hiệu quả.

Dưới đây là một số điểm nổi bật của thư viện threading:

- Да luồng: Threading cho phép thực hiện đa luồng trong Python, giúp tận dụng tối đa các tài nguyên hệ thống và tăng hiệu suất thực thi của các ứng dụng.
- Phù hợp cho các tác vụ I/O Động: Threading thích hợp cho các tác vụ I/O động như đọc và ghi file, kết nối mạng, vv., giúp cho việc xử lý các tác vụ này trở nên hiệu quả và nhanh chóng.
- Tiện ích trong lập trình đa nhiệm: Threading là một công cụ hữu ích trong lập trình đa nhiệm, giúp quản lý và thực thi các tác vụ cùng một lúc một cách linh hoạt và hiệu quả.

2.3 Python WebSocket

WebSocket trong Python là một giao thức truyền tải hai chiều và đa luồng, được sử dụng để thiết lập kết nối giữa máy chủ và các máy khách. Nó cho phép truyền tải dữ liệu theo thời gian thực, làm cho các ứng dụng web trở nên tương tác và động hơn.

Trong đồ án, WebSocket sẽ được sử dụng để tạo một giao tiếp hai chiều thời gian thực giữa máy chủ và máy khách. Thay vì sử dụng phương thức truyền thống như HTTP, WebSocket cho phép chúng ta thiết lập một kết nối liên tục, cho phép truyền tải dữ liệu một cách hiệu quả và không giới hạn.

2.3.1 Nguyên lý hoạt động cơ bản của Socket

- Khởi tạo kết nối: Một máy khách khởi tạo kết nối đến máy chủ thông qua một yêu cầu kết nối.
- Xác nhận kết nối: Máy chủ nhận yêu cầu kết nối và xác nhận nó, tạo ra một kết nối socket duy trì trạng thái kết nối.
- Truyền tải dữ liệu: Sau khi kết nối được thiết lập, cả máy chủ và máy khách có thể truyền tải dữ liệu qua kết nối socket.
- Đóng kết nối: Khi hoàn thành truyền tải dữ liệu, hoặc khi không cần thiết đến kết nối nữa, cả máy chủ và máy khách có thể đóng kết nối.

2.3.2

Nguyên lý hoạt động cơ bản của WebSocket

- Handshake (Bắt tay): Một máy khách khởi tạo kết nối WebSocket bằng cách gửi một yêu cầu HTTP đặc biệt đến máy chủ. Máy chủ sau đó phản hồi bằng cách xác nhận yêu cầu và thực hiện việc chuyển đổi từ HTTP sang WebSocket.
- Kết nối hai chiều và đa luồng: Sau khi kết nối được thiết lập, cả máy chủ và máy khách có thể truyền tải dữ liệu hai chiều qua kết nối WebSocket, mà không cần phải thiết lập lại kết nối.
- Truyền tải dữ liệu thời gian thực: WebSocket cho phép truyền tải dữ liệu theo thời gian thực, giúp làm cho các ứng dụng web trở nên tương tác và động hơn.
- Giữ kết nối mở: Kết nối WebSocket được duy trì mở trong suốt thời gian hoạt động của ứng dụng, cho phép truyền tải dữ liệu một cách liên tục và hiệu quả.



2.4 Môi trường làm việc

2.4.1 Pycharm

PyCharm là một IDE phổ biến được phát triển bởi JetBrains, dành riêng cho việc lập trình Python. Với giao diện thân thiện và nhiều tính năng mạnh mẽ, PyCharm là lựa chọn ưa thích của nhiều nhà phát triển Python. PyCharm cung cấp các tính năng như:

- Môi trường phát triển tích hợp đầy đủ: từ việc viết mã, debug đến quản lý mã nguồn.
- Hỗ trợ kiểm thử và lập trình hiệu suất cao.
- Tích hợp với các công cụ quản lý phiên bản như Git.
- Cung cấp tính năng kiểm tra lỗi và gợi ý cải thiện mã.

PyCharm là một IDE mạnh mẽ và linh hoạt, thích hợp cho việc phát triển ứng dụng mã nguồn mở nhờ vào giao diện dễ sử dụng, tính năng đa dạng, và sự hỗ trợ từ cộng đồng lập trình viên lớn.

- Giao diện và tính năng mạnh mẽ: PyCharm cung cấp một giao diện người dùng thân thiện và dễ sử dụng, đi kèm với nhiều tính năng mạnh mẽ như tự động hoàn thành mã, debugging thông minh, kiểm tra lỗi, refactor mã, và nhiều tính năng khác giúp tăng tốc độ phát triển ứng dụng.
- Hỗ trợ đa nền tảng và ngôn ngữ: PyCharm hỗ trợ phát triển trên nhiều nền tảng như Windows, macOS và Linux, đồng thời hỗ trợ nhiều ngôn ngữ lập trình khác nhau ngoài Python như JavaScript, HTML, CSS, và nhiều ngôn ngữ khác.
- Tích hợp với các công cụ quản lý phiên bản: PyCharm tích hợp sâu với các hệ thống quản lý mã nguồn như Git, SVN, Mercurial, và Perforce, giúp quản lý mã nguồn dễ dàng và hiệu quả.
- Hỗ trợ công cụ kiểm thử và phát triển hiệu suất cao: PyCharm đi kèm với các công cụ mạnh mẽ như Unit Testing, Profiler, và Jupyter Notebooks, giúp bạn kiểm thử và tối ưu hóa hiệu suất của ứng dụng một cách dễ dàng.
- Cộng đồng lớn và hỗ trợ mở rộng: PyCharm được hỗ trợ bởi một cộng đồng lập trình viên lớn và nhiều tài nguyên mở rộng như plugins và themes, giúp bạn tùy chỉnh và mở rộng chức năng của IDE theo nhu cầu cụ thể của dự án.

2.4.2 Visual Studio Code

Visual Studio Code là một IDE mã nguồn mở được Microsoft phát triển, hỗ trợ nhiều ngôn ngữ lập trình, bao gồm Python. Với giao diện đơn giản và linh hoạt, VSC được nhiều người ưa chuộng. Các tính năng chính của VSC bao gồm:

- Hỗ trợ đa ngôn ngữ và tích hợp với nhiều công cụ mở rộng.
- Debugging và kiểm thử mã mạnh mẽ.
- Tích hợp Git và các công cụ quản lý phiên bản khác.
- Cung cấp tính năng IntelliSense giúp việc viết mã dễ dàng hơn.



Visual Studio Code là một IDE linh hoạt, nhẹ nhàng và mạnh mẽ, thích hợp cho việc phát triển ứng dụng mã nguồn mở nhờ vào tính linh hoạt, tích hợp với Git, và khả năng mở rộng thông qua plugins và themes.

- Tính nhẹ nhàng và linh hoạt: VSC có giao diện đơn giản và nhẹ nhàng, tối ưu cho việc phát triển ứng dụng. Đi kèm với đó là khả năng linh hoạt trong cài đặt và tùy chỉnh theo nhu cầu của từng dự án cụ thể.
- Hỗ trợ nhiều ngôn ngữ và framework: VSC hỗ trợ nhiều ngôn ngữ lập trình khác nhau và tích hợp với các framework phổ biến như Node.js, Python, C++, và nhiều ngôn ngữ khác, giúp bạn dễ dàng phát triển ứng dụng mã nguồn mở trên nhiều nền tảng.
- Tích hợp Git và quản lý mã nguồn: VSC tích hợp sâu với hệ thống quản lý mã nguồn Git, cung cấp các tính năng như commit, push, pull, và xem lịch sử thay đổi một cách dễ dàng. Điều này giúp quản lý mã nguồn và cộng tác trong dự án một cách hiệu quả.
- Cộng đồng mở rộng và hỗ trợ plugins: VSC có một cộng đồng lập trình viên lớn và sôi động, cùng với hàng ngàn plugins và themes miễn phí trên Marketplace. Điều này cho phép bạn tùy chỉnh và mở rộng chức năng của IDE theo nhu cầu cụ thể của dự án.
- Tính di động và đa nền tảng: VSC hoạt động trên nhiều nền tảng như Windows, macOS và Linux, cho phép bạn phát triển ứng dụng trên bất kỳ hệ điều hành nào mà không gặp rào cản.

3 Phân tích và xây dựng game

3.1 Giới thiệu về trò chơi Tic tac toe

Trong thế giới của trò chơi, có một tựa game kinh điển với sự đơn giản nhưng lại ẩn chứa bài học vô giá về chiến lược và tư duy - Tic Tac Toe. Dưới vẻ ngoài giản dị của bảng vuông 3x3 và hai loại ký hiệu, Tic Tac Toe là một bài toán phức tạp về tối ưu hóa chiến thuật và dự đoán.

3.2 Phân tích luật chơi đơn giản Tic tac toe

Trò chơi bắt đầu với hai người chơi, mỗi người lần lượt đặt một ký hiệu của mình vào một ô trống trên bảng. Mục tiêu là tạo ra một chuỗi gồm ba ký hiệu giống nhau theo hàng ngang, hàng dọc hoặc chéo. Dường như đơn giản nhưng bên dưới cả vẻ đơn giản đó là một thế giới rộng lớn của chiến thuật và tư duy chiến lược. Trong mỗi nước đi, người chơi phải đánh giá và dự đoán những hành động tiếp theo của đối thủ. Họ cần phải xem xét các khả năng và lựa chọn nước đi tối ưu nhất dựa trên những gì họ biết về trò chơi và về đối thủ của mình. Từ việc đánh giá các biến thể có thể xảy ra đến việc tối ưu hóa chiến lược của bản thân, Tic Tac Toe là một bài học về quyết định đúng đắn và khả năng tư duy logic. Hơn nữa, Tic Tac Toe cũng là một bài học về quản lý thời gian và tài nguyên. Người chơi phải biết khi nào nên tấn công và khi nào nên phòng thủ, khi nào nên tạo ra các cơ hội và khi nào nên chờ đợi. Sự cân nhắc và tính toán kỹ lưỡng là chìa khóa để chiến thắng trong trò chơi này. Tóm



lại, Tic Tac Toe không chỉ là một trò chơi giải trí, mà còn là một bài học sâu sắc về chiến lược, tư duy và quản lý tài nguyên. Bằng cách tham gia vào cuộc phiêu lưu tinh thần này, người chơi có thể rèn luyện và phát triển những kỹ năng quan trọng trong cuộc sống hàng ngày.

3.3 Xây dựng game

Theo vai trò, có thể chia lĩnh vực của game thành 3 thành phần chính: **Bàn cờ**: là một đối tượng có các thuộc tính cần có của trò chơi như điểm số, lượt đánh,... ; sử dụng các function để tính toán, cập nhật bàn cờ, nắm giữ và quyết định logic cho các luật lệ cơ bản của trò Tic tac toe **Player1 (Server)**: đóng vai trò như một Host server của Websocket. Là bên khởi tạo socket trước để người chơi sau kết nối tới. Cùng với các function giả định các hành động (exit game, accept challenge) và nước đánh có thể của người chơi 1. **Player2 (Client)**: đóng vai trò như một Client kết nối socket với Host. Cùng với các function giả định các hành động (exit, connect) và nước đánh có thể của người chơi 2.

3.3.1 Xây dựng đối tượng Bàn cờ

Tạo file gameboard.py đảm nhiệm thành phần Bàn cờ. Xây dựng một đối tượng Board gồm các thuộc tính như sau:

- board: array đại diện cho 9 ô cờ (3x3) trong bàn cờ. Mỗi ô gồm 3 trạng thái có thể có được 0: ô chưa bị chiếm; “X”: ô bị chiếm bởi player2; “O”: ô bị chiếm bởi player 1
- user1, user2: tên của 2 người chơi và là đại diện nhận dạng cho mỗi nước đi và hành động của 2 người chơi khi thao tác trong quá trình chơi.
- lastMove: quản lý lượt đánh của người chơi.
- numGames: chứa thông tin số trận đã diễn ra giữa 2 người chơi trong 1 phiên chạy chương trình.
- wins, loss: chứa thông số tỉ số thắng thua giữa 2 người chơi.
- ties: dùng trong trường hợp đặc biệt - đếm số lần hòa giữa 2 người chơi.



```
class Board:

    def __init__(self, user2):
        self.board = [0,0,0,0,0,0,0,0,0]
        self.user1 = "player1"
        self.user2 = user2
        self.lastMove = ""
        self.numGames = 1
        self.wins = {"X": 0, "O": 0}
        self.loss = {"X": 0, "O": 0}
        self.ties = 0
```

Hình 1: Đôi tượng Board

Tiếp theo xây dựng các function cần thiết cho một bàn cờ: Hàm **recordGamePlayed()** sẽ được gọi để cập nhật số game đã chơi giữa 2 người chơi trong một phiên trong trường hợp ván “game over” và người chơi chọn “play again”.

Cùng lúc thời điểm game over, hàm **resetGameBoard()** cũng sẽ được gọi để đặt lại trạng thái trống ban đầu cho 9 ô bàn cờ. Khi một trong 2 người chơi thực hiện một nước đi trên bàn cờ, hàm **playMoveOnBoard()** sẽ được gọi nhằm kiểm tra xem người chơi nào đã invoke hàm (luợt đánh thuộc về ai) từ đó cập nhật trạng thái cho đúng ô cờ vị trí nước đánh với flag X hoặc O của người chơi tương ứng. Sau đó ghi nhận lại lượt đánh cuối cùng thuộc về player đó.

Vì bàn cờ có giới hạn 9 ô (3x3) nên đôi khi sẽ xảy ra trường hợp 2 người chơi đánh hết số ô trên bàn cờ nhưng vẫn chưa ai chiến thắng. Lúc này sẽ invoke hàm **isBoardFull()** để kiểm tra tất cả ô cờ đã được đánh hay chưa. Nếu kết quả = True thì sẽ cập nhật tỉ số tương ứng.



```
def recordGamePlayed(self):
    self.numGames = self.numGames + 1

def resetGameBoard(self):
    self.board = [0,0,0,0,0,0,0,0,0]

def playMoveOnBoard(self,position,player):
    if player == "player1":
        self.board[position] = "O"
        self.lastMove = "player1"
    else:
        self.board[position] = "X"
        self.lastMove = self.user2

def isBoardFull(self):
    for cell in self.board:
        if cell == 0:
            return False

    return True
```

Hình 2: Các hàm xử lý logic của Bàn cờ

Tiếp theo cần xây dựng một hàm **isGameFinished()** để kiểm tra trạng thái của bàn cờ. Hàm này sẽ liên tục được gọi sau mỗi lượt đánh của người chơi nhằm check các trường hợp game over khả thi trên ô 3x3:

Lấy trạng thái từng ô cờ hiện tại qua thuộc tính gameboard.board.
Check 3 flag X hoặc O giống nhau trên 1 hàng ngang, dọc và chéo và thuộc flag của người chơi nào. Từ đó cập nhật các thuộc tính cho tỉ số tương ứng cho bên thắng và bên thua.
Nếu kiểm tra thấy trường hợp hòa từ hàm **isBoardFull()** thì cập nhật tỉ số của ván hòa.



```
def isGameFinished(self):  
  
    finish = False  
    b1 = self.board[0]; b2 = self.board[1]; b3 = self.board[2]  
    b4 = self.board[3]; b5 = self.board[4]; b6 = self.board[5]  
    b7 = self.board[6]; b8 = self.board[7]; b9 = self.board[8]  
  
    if (b1==b2 and b1==b3 and b1=="O") or (b4==b5 and b4==b6 and b4=="O") or \  
        (b7==b8 and b7==b9 and b7=="O"):  
        finish = True  
        self.wins["O"] = self.wins["O"] + 1  
        self.loss["X"] = self.loss["X"] + 1  
    elif (b1==b2 and b1==b3 and b1=="X") or (b4==b5 and b4==b6 and b4=="X") or \  
        (b7==b8 and b7==b9 and b7=="X"):  
        finish = True  
        self.wins["X"] = self.wins["X"] + 1  
        self.loss["O"] = self.loss["O"] + 1  
    elif (b1==b4 and b1==b7 and b1=="O") or (b2==b5 and b2==b8 and b2=="O") or \  
        (b3==b6 and b3==b9 and b3=="O"):  
        finish = True  
        self.wins["O"] = self.wins["O"] + 1  
        self.loss["X"] = self.loss["X"] + 1  
    elif (b1==b4 and b1==b7 and b1=="X") or (b2==b5 and b2==b8 and b2=="X") or \  
        (b3==b6 and b3==b9 and b3=="X"):  
        finish = True  
        self.wins["X"] = self.wins["X"] + 1  
        self.loss["O"] = self.loss["O"] + 1  
    elif (b1==b5 and b1==b9 and b1=="O") or (b7==b5 and b7==b3 and b7=="O"):  
        finish = True  
        self.wins["O"] = self.wins["O"] + 1  
        self.loss["X"] = self.loss["X"] + 1  
    elif (b1==b5 and b1==b9 and b1=="X") or (b7==b5 and b7==b3 and b7=="X"):  
        finish = True  
        self.wins["X"] = self.wins["X"] + 1  
        self.loss["O"] = self.loss["O"] + 1  
    elif self.isBoardFull():  
        finish = True  
        self.ties = self.ties + 1  
  
    return finish
```

Hình 3: Hàm kiểm tra state trò chơi

Cuối cùng là hàm **computeStats()** sẽ được gọi sau khi “game over” để trả về object chứa đầy đủ các dữ liệu tỉ số giữa 2 người chơi trong một phiên để cập nhật lên GUI hiển thị.



```
def computeStats(self):
    stats = {}
    stats["board"] = self.board
    stats["user1"] = self.user1
    stats["user2"] = self.user2
    stats["lastMove"] = self.lastMove
    stats["numGames"] = self.numGames
    stats["wins"] = self.wins
    stats["loss"] = self.loss
    stats["ties"] = self.ties

    return stats
```

Hình 4: Hàm computeStats

3.3.2 Cài đặt giao tiếp giữa Server socket và Client socket

Dầu tiên sẽ xây dựng một server của socket. Đây sẽ là phía bên player 1. Để khởi tạo một server socket đơn giản bằng python, ta sử dụng thư viện socket như sau:

Hàm **socket()** sẽ mở một máy chủ bằng một đối tượng socket. Ở đây chỉ rõ 2 tham số truyền vào hàm là 2 hằng số address family và socket type, AF-INET cho biết tham số ip được sử dụng sẽ có dạng IPv4, SOCK-STREAM sẽ chỉ ra giao thức kết nối và giao tiếp mà socket dùng sẽ là TCP.

Sau đó hàm **bind()** sẽ gán địa chỉ cho socket, giá trị tham số truyền vào hàm này phụ thuộc vào address family mà ta đã khai báo cho đối tượng socket trước đó. Ở đây là IPv4 nên truyền vào 1 tuple (địa chỉ IP, Port)

Tiếp theo cần cho socket server lắng nghe các kết nối tới máy chủ với hàm **listen()**. Ở đây ta chỉ định chỉ chấp nhận chờ 1 kết nối trong hàng chờ các client vì chỉ cần 2 người chơi cùng lúc trong 1 phiên (nếu 1 trong 2 out game thì sẽ coi như phiên mới).

```
serverAddress = '192.168.2.11'
port = 8000

serverSocket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
serverSocket.bind((serverAddress, port))
serverSocket.listen(1)

clientSocket, clientAddress = None, None
```

Hình 5: Tạo Server socket phía player1

Sau khi đã khởi tạo một server, cần xây dựng phía bên client để kết nối tới máy chủ đó. Vậy player2 sẽ đảm nhiệm vai trò làm client này. Hàm **clickConnect()** sẽ được invoke khi player2 chọn bắt đầu kết nối.

Để kết nối tới server thì client cần thông tin địa chỉ IPv4 và port của server. Tương tự như phía player1, để kết nối thì cần khởi tạo một đối tượng socket bằng hàm **socket()**

Sau đó gọi hàm **connect()** và truyền thông tin địa chỉ của server (player1) vào.



```
def clickConnect():
    global serverAddress,serverPort,connectionSocket,isCon
    if not isCon:
        serverAddress = simpledialog.askstring("Input", "What is host name/ip address?",parent=window)
        serverPort = simpledialog.askstring("Input", "What is the port?",parent=window)
        if serverAddress != "" and serverPort != "":
            try:
                connectionSocket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
                connectionSocket.connect((serverAddress,int(serverPort)))
                isCon = True
```

Hình 6: Hàm clickConnect()

Bên phía Server (player1) sẽ chạy sẵn hàm **acceptConnection()** để chờ kết nối từ client. Sau khi đã khởi tạo đối tượng socket như bước trên, gọi hàm **accept()** để sẵn sàng chấp nhận kết nối từ player2. Lúc này chương trình sẽ bị treo cho tới khi nào nhận được request kết nối từ client. Hàm **accept()** này trả về 2 giá trị: đối tượng kết nối với client và địa chỉ của client. Từ lúc này trở đi ta sẽ sử dụng đối tượng kết nối client để gửi và nhận dữ liệu với bên client.

Trường hợp từ chối kết nối, sử dụng hàm **close()** sẽ đóng lại kết nối giữa 2 bên.

```
def acceptConnection():
    print("Thread created")
    global clientSocket,clientAddress,start
    clientSocket,clientAddress = serverSocket.accept()
    message = "Incoming play request from "+str(clientAddress)+" client?"
    answer = messagebox.askyesno("Question",message)
    if answer:
        sendData = '{}'.format("accepted").encode()
        clientSocket.send(sendData)
        start = True
        reset()
        resetStat()
        print(sendData)
        print("Client connected from: ",clientAddress)
        receive()
    else:
        sendData = '{}'.format("Not accepted").encode()
        clientSocket.send(sendData)
        clientSocket.close()
        print("Client socket closes")
```

Hình 7: Hàm acceptConnection

Sau khi kết nối 2 phía thành công, player1 sẽ gửi tín hiệu đồng ý “accepted” cho player2. Phía player2 thông qua hàm **receive()** đang chạy sẽ nhận được data gửi và xử lý.

Player2 tiến hành điền username để định danh đầy đủ cho cả 2 người chơi. Gửi data username về lại cho player1.



```
def receive():
    global start,user,board,username,player1,connectionSocket,isCon,close
    while True:
        if isCon:
            if (not start or not user):
                data,addr = connectionSocket.recvfrom(1024)
                data = data.decode()
                print(data)
                if data == "Not accepted":
                    isCon = True
                    connectionSocket.close()
                elif data == "accepted":
                    username = simpledialog.askstring("Input", "Enter your name?",parent=window)
                    sendData = '{}'.format(username).encode()
                    connectionSocket.send(sendData)
                    lblUsername["text"] = username
                    print(sendData)
                    start = True
                    close = False
                elif data == "player1":
                    player1 = data
                    board = gb.Board(username)
                    board.lastMove = player1
                    user = True
                    break
```

Hình 8: Hàm receive() của player2

Bên player1 cũng đang chạy sẵn hàm receive() để nhận data username từ player2, cập nhật lại trên UI tên của player2. Sau đó gửi data username của player1 cho phía player2 để player2 làm điều tương tự.

```
def receive():
    global start,user,player2,board
    while True:
        if start and not user:
            try:
                data,addr = clientSocket.recvfrom(1024)
                name = data.decode()
                print(data)
                board = gb.Board(name)
                player2 = name
                sendData = '{}'.format("player1").encode()
                clientSocket.send(sendData)
                user = True
            except:
                pass
```

Hình 9: Hàm receive() của player1

Bên phía player2 nhận được username của player1, cập nhật lại trên UI. Tất cả các bước kết nối đã hoàn tất và có thể sẵn sàng bắt đầu trò chơi.



```
close = raise
elif data == "player1":
    player1 = data
    board = gb.Board(username)
    board.lastMove = player1
    user = True
    break
```

3.3.3 Mô phỏng các nước đi giữa 2 player

Dầu tiên, để đảm bảo quá trình giao tiếp giữa 2 player được liên tục, cần phải đảm bảo chức năng gửi nhận dữ liệu qua socket luôn chạy. Thay vì sử dụng vòng lặp, ta sẽ cài đặt và cho phép quá trình này diễn ra trên các luồng chạy song song với chương trình game.

Sử dụng thư viện **threading** của python. Và khai báo một hàm **createThread()** nhằm tạo ra các luồng chạy song song. Khai báo **thread.daemon=True** : sử dụng thread kiểu deamon đảm bảo thread này luôn chạy ngầm và sẽ tự động kết thúc khi chương trình kết thúc.

```
def createThread(con):
    thread = threading.Thread(target=con)
    thread.daemon = True
    thread.start()
```

Hình 10: Hàm createThread()

Lúc này cho chạy hàm **createThread()** với tham số truyền vào là một hàm khác **receiveMove()** để cho hàm này được chạy ngầm trong một luồng song song.

```
createThread(receiveMove)
```

Hàm chạy ngầm **receiveMove()** chính là hàm đảm nhận vai trò gửi và nhận dữ liệu giao tiếp giữa 2 player thông qua socket: Nhận data được gửi từ đối thủ qua hàm **recvfrom()**. Data được gửi theo format pos: vị trí nước đi trên ô cờ, board.lastMove: lượt đổi thủ.

Gọi hàm **playMoveOnBoard()** để cập nhật dữ liệu nước đi trên ô bàn cờ, và hàm **updateBoard()** để cập nhật lại trên UI hiển thị cho player.

```
def receiveMove():
    global pos,start,user
    while True:
        if isCon and start and user:
            try:
                data,addr = connectionSocket.recvfrom(1024)
                data = data.decode()
                print(data)
                if data != "":
                    pos,board.lastMove = data.split("-")
                    board.playMoveOnBoard(int(pos),board.lastMove)
                    updateBoard()
                    lblTurn["text"] = "You"
                    play()
            except:
                lblTurn["text"] = "you"
                pass
```



Hình 11: Hàm receiveMove()

Hàm **play()** sẽ được gọi sau mỗi lượt đánh của player đối với lượt của bản thân, hoặc sau mỗi lần nhận được dữ liệu từ đối thủ sau mỗi lượt đánh của đối thủ. Sẽ kiểm tra trạng thái của bàn cờ xem đã “game over” chưa.

Nếu thỏa điều kiện “game over”, các hàm **recordGamePlayed()** và **resetGameBoard()** sẽ được gọi để làm các nhiệm vụ tương ứng trên bàn cờ, đồng thời update lại lượt đánh cuối cho player1.

```
def play():
    finish = board.isGameFinished()
    if finish:
        board.recordGamePlayed()
        board.resetGameBoard()
        board.lastMove = "player1"
```

Hình 12: Hàm play bên player1()

Tiếp theo là cài đặt mô phỏng cho các nước đánh của player.

Đầu tiên dựng UI các ô cờ bằng thư viện tkinter : các ô là các component Button trên UI, có 9 ô tất cả.

Mỗi ô từ 1 đến 9 đều được gán event khi click vào sẽ invoke hàm **clickedX()** bằng lệnh **command=clickedX**.

```
btn1 = Button(leftFrame, text=" ",bg="yellow",
               fg="Black",width=8,height=3,font=('Times', 26),command=clicked1)
btn1.grid(column=1, row=1, sticky = S+N+E+W)
```

Hình 13: Tạo btn các ô cờ trên UI

Ví dụ trong hàm **clicked1()**, đầu tiên sẽ kiểm tra điều kiện xem ô cờ thứ 1 có tọa độ (0;0) trên bàn cờ 3x3 – đã được đánh hay chưa (X hoặc O), nếu ô này còn trống thì mới có thể tiếp tục xử lý.

Sau đó kiểm tra tiếp điều kiện lượt đánh cuối có phải của đối thủ hay không, nhằm đảm bảo không phạm luật 1 player đánh 2 lượt liên tiếp.

Nếu thỏa cả 2 điều kiện trên, cập nhật UI và gửi data thông tin lượt đánh cho đối thủ qua socket với format vị trí đánh-lượt player. Sau đó gọi hàm **playMoveOnBoard()** để cập nhật lại dữ liệu cho đối tượng bàn cờ và gọi hàm **play()** để tính toán trạng thái bàn cờ.

```
def clicked1():
    global finish,board
    if start and user and btn1["text"]==" ":
        if board.lastMove == player2:
            btn1["text"]="0"
            board.lastMove = "player1"
            sendData = '{}-{}'.format("0",board.lastMove).encode()
            clientSocket.send(sendData)
            board.playMoveOnBoard(0,board.lastMove)
            lblTurn["text"] = "opponent"
            print(sendData)
            play()
```

Hình 14: Các hàm nhận sự kiện click của player

3.3.4 Các kịch bản khi kết thúc ván chơi

Khi một trong 2 bên thắng hoặc hòa, sẽ có 2 lựa chọn: chơi tiếp hoặc kết thúc phiên đối kháng giữa 2 player. Sau khi hàm **play()** được gọi và trả về kết quả, phía Client (player2) sẽ là bên lựa chọn và gửi request cho phía Server.

Nếu client chọn chơi lại, message “Play Again” sẽ được gửi qua socket cho phía Server. Nếu không chơi lại, message “Fun Times” sẽ được gửi đi, đồng thời đóng kết nối socket qua hàm **close()** và cập nhật lại bảng tỉ số.

```
def play():
    global isCon,start,user,close
    finish = board.isGameFinished()
    if finish:
        answer = messagebox.askyesno("Question","Do you want to play again?")
        if answer:
            sendData = '{}'.format("Play Again").encode()
            connectionSocket.send(sendData)
            board.recordGamePlayed()
            reset()
        else:
            isCon = False
            sendData = '{}'.format("Fun Times").encode()
            connectionSocket.send(sendData)
            connectionSocket.close()
            start = False
            user = False
            close = True
            comData = board.computeStats()
            lblGames["text"] = comData["numGames"]
            lblWon["text"] = comData["wins"]["X"]
            lblLost["text"] = comData["loss"]["X"]
            lblTies["text"] = comData["ties"]
            lblTurn["text"] = "You"
            reset()
```

Hình 15: Hàm play bên player2()

Bên phía Server (player1) sẽ nhận được message tương ứng và tùy vào trường hợp, sẽ gọi các hàm **reset()** để chơi lại hoặc **gameOver()** để kết thúc phiên chơi giữa 2 player.

```
def receiveMove():
    global pos,start
    while True:
        if start and user:
            try:
                data,addr = clientSocket.recvfrom(1024)
                data = data.decode()
                print(data)
                if data == "Play Again":
                    reset()
                elif data == "Fun Times":
                    gameOver()
```



Hình 16: Hàm receiveMove() bên player1

Làm mới lại bàn cờ bằng hàm **reset()**, clear dữ liệu các nút ô cờ trong đối tượng Bàn cờ cũng như clear phần hiển thị trên UI, đặt lượt chơi cuối về player1 để player2 luôn đi đầu tiên.

```
def reset():
    board.resetGameBoard()
    board.lastMove = player1
    btn1["text"] = " "
    btn2["text"] = " "
    btn3["text"] = " "
    btn4["text"] = " "
    btn5["text"] = " "
    btn6["text"] = " "
    btn7["text"] = " "
    btn8["text"] = " "
    btn9["text"] = " "
```

Hình 17: Hàm reset()

Trong hàm **gameOver()**, update lại giá trị các biến boolean quan trọng để check điều kiện: **isCon**, **start**, **user = False**; **close = True**

Đóng kết nối socket bằng hàm **close()**, tính toán lại tỉ số và hiển thị, cuối cùng là gọi hàm **reset()** để làm mới lại bàn cờ.

```
def gameOver():
    isCon = False
    start = False
    user = False
    close = True
    clientSocket.close()
    board.numGames = board.numGames - 1
    comData = board.computeStats()
    lblGames["text"] = comData["numGames"]
    lblWon["text"] = comData["wins"]["0"]
    lblLost["text"] = comData["loss"]["0"]
    lblTies["text"] = comData["ties"]
    lblTurn["text"] = "opponent"
    reset()
```

Hình 18: Hàm gameOver()

Trường hợp tự thoát trò chơi, trước khi đóng cửa sổ game bằng hàm **window()**, gọi hàm **close()** để đóng kết nối.

```
def clickQuit():
    if start or user:
        connectionSocket.close()
    window.destroy()
```

Hình 19: Hàm clickQuit()



3.3.5 Xây dựng GUI trò chơi

Tạo cửa sổ ứng dụng và một số thuộc tính cho nó

```
window=Tk()
window.title("TiC-Tac-Toe Player1")
window.geometry("920x650+0+0")
window.configure(background = 'Cadet Blue')
```

Hình 20: Xây dựng đối tượng cửa sổ màn hình

Chia cửa sổ ra làm từng phần sử dụng Frame() : top, main, left, right.

```
tops = Frame(window, bg = 'Cadet Blue', pady =2, width = 1350, height=100, relief = RIDGE)
tops.grid(row=0, column =0)

mainFrame = Frame (window, bg = 'Powder Blue', bd=10, width = 1350, height=600, relief=RIDGE)
mainFrame.grid(row=1,column=0)

leftFrame = Frame (mainFrame ,bd=10, width =750, height=500, pady=2, padx=10, bg="Cadet Blue", relief=RIDGE)
leftFrame.pack(side=LEFT)

rightFrame = Frame (mainFrame,bd=10, width =560, height=500, padx=10, pady=2, bg="Cadet Blue", relief=RIDGE)
rightFrame.pack(side=RIGHT)

rightFrame1=Frame(rightFrame ,bd=10, width=560, height=200, padx=10, pady=2, bg="Cadet Blue", relief=RIDGE)
rightFrame1.grid(row=0, column=0)

rightFrame2 = Frame(rightFrame,bd=10, width =560, height=250, padx=10, pady=2, bg="Cadet Blue", relief=RIDGE)
rightFrame2.grid(row=1,column=0)

rightFrame3=Frame(rightFrame ,bd=10, width=560, height=150, padx=10, pady=2, bg="Cadet Blue", relief=RIDGE)
rightFrame3.grid(row=2, column=0)
```

Hình 21: Tạo các Frame chia bố cục trên màn hình

Đoạn code trên xây dựng GUI cho cửa sổ theo bố cục đại khái như sau:



Hình 22: Bản phác thảo các bố cục trên giao diện

Ở Frame bên trái (LEFT), đựng 9 ô tương ứng các ô bàn cờ.



```
btn1 = Button(leftFrame, text=" ",bg="yellow", fg="Black",width=8,height=3,font=('Times', 26),command=clicked1)
btn1.grid(column=1, row=1, sticky = S+N+E+W)
btn2 = Button(leftFrame, text=" ",bg="yellow", fg="Black",width=8,height=3,font=('Times', 26),command=clicked2)
btn2.grid(column=2, row=1, sticky = S+N+E+W)
btn3 = Button(leftFrame, text=" ",bg="yellow", fg="Black",width=8,height=3,font=('Times', 26),command=clicked3)
btn3.grid(column=3, row=1, sticky = S+N+E+W)
btn4 = Button(leftFrame, text=" ",bg="yellow", fg="Black",width=8,height=3,font=('Times', 26),command=clicked4)
btn4.grid(column=1, row=2, sticky = S+N+E+W)
btn5 = Button(leftFrame, text=" ",bg="yellow", fg="Black",width=8,height=3,font=('Times', 26),command=clicked5)
btn5.grid(column=2, row=2, sticky = S+N+E+W)
btn6 = Button(leftFrame, text=" ",bg="yellow", fg="Black",width=8,height=3,font=('Times', 26),command=clicked6)
btn6.grid(column=3, row=2, sticky = S+N+E+W)
btn7 = Button(leftFrame, text=" ",bg="yellow", fg="Black",width=8,height=3,font=('Times', 26),command=clicked7)
btn7.grid(column=1, row=3, sticky = S+N+E+W)
btn8 = Button(leftFrame, text=" ",bg="yellow", fg="Black",width=8,height=3,font=('Times', 26),command=clicked8)
btn8.grid(column=2, row=3, sticky = S+N+E+W)
btn9 = Button(leftFrame, text=" ",bg="yellow", fg="Black",width=8,height=3,font=('Times', 26),command=clicked9)
btn9.grid(column=3, row=3, sticky = S+N+E+W)
```

Hình 23: Dụng UI hiển các ô cờ

Ở Frame bên phải hàng 1 (RIGHT1), dụng phần hiển thị tên người chơi, lượt đánh.

```
lbl=Label(rightFrame1, font=('arial', 20, 'bold'), text="Player2:",padx=2, pady=2, bg="Cadet Blue")
lbl.grid (row=0, column=0, sticky=W)
lblUsername=Label(rightFrame1, font=('arial', 20, 'bold'), text="",padx=2, pady=2, bg="Cadet Blue",width=10)
lblUsername.grid (row=0, column=1, sticky=W)
lbl=Label(rightFrame1, font=('arial', 20, 'bold'), text="Turn: ",padx=2, pady=2, bg="Cadet Blue")
lbl.grid (row=1, column=0, sticky=W)
lblTurn=Label(rightFrame1, font=('arial', 20, 'bold'), text="You",padx=2, pady=2, bg="Cadet Blue",width=10)
lblTurn.grid (row=1, column=1, sticky=W)
```

Hình 24: Dụng UI hiển thị thông tin player

Frame bên phải hàng 2 (RIGHT2), hiển thị các thông tin như: số trận, tỉ số thắng-thua-hòa.

```
lbl=Label(rightFrame2, font=('arial', 20, 'bold'), text="Games:",padx=2, pady=2, bg="Cadet Blue")
lbl.grid (row=0, column=0, sticky=W)
lblGames=Label(rightFrame2, font=('arial', 20, 'bold'), text=" ",padx=2, pady=2, bg="Cadet Blue",width=10)
lblGames.grid (row=0, column=1, sticky=W)
lbl=Label(rightFrame2, font=('arial', 20, 'bold'), text="Won:",padx=2, pady=2, bg="Cadet Blue")
lbl.grid (row=1, column=0, sticky=W)
lblWon=Label(rightFrame2, font=('arial', 20, 'bold'), text=" ",padx=2, pady=2, bg="Cadet Blue",width=10)
lblWon.grid (row=1, column=1, sticky=W)
lbl=Label(rightFrame2, font=('arial', 20, 'bold'), text="Lost:",padx=2, pady=2, bg="Cadet Blue")
lbl.grid (row=2, column=0, sticky=W)
lblLost=Label(rightFrame2, font=('arial', 20, 'bold'), text=" ",padx=2, pady=2, bg="Cadet Blue",width=10)
lblLost.grid (row=2, column=1, sticky=W)
lbl=Label(rightFrame2, font=('arial', 20, 'bold'), text="Ties:",padx=2, pady=2, bg="Cadet Blue")
lbl.grid (row=3, column=0, sticky=W)
lblTies=Label(rightFrame2, font=('arial', 20, 'bold'), text=" ",padx=2, pady=2, bg="Cadet Blue",width=10)
lblTies.grid (row=3, column=1, sticky=W)
```

Hình 25: Dụng UI hiển thị các tỉ số trận

Frame bên phải hàng 3 (RIGHT3), hiển thị các nút lựa chọn Quit game, Connect to Server.

```
btnConnect=Button(rightFrame3, text="Connect", font=('arial', 17, 'bold'), height = 1, width =20,command=clickConnect)
btnConnect.grid (row=2, column=0 ,padx=6, pady=11)

btnQuit=Button (rightFrame3, text="Quit", font=('arial', 17, 'bold'), height = 1, width =20, command = clickQuit)
btnQuit.grid(row=3, column=0 ,padx=6, pady=10)
```

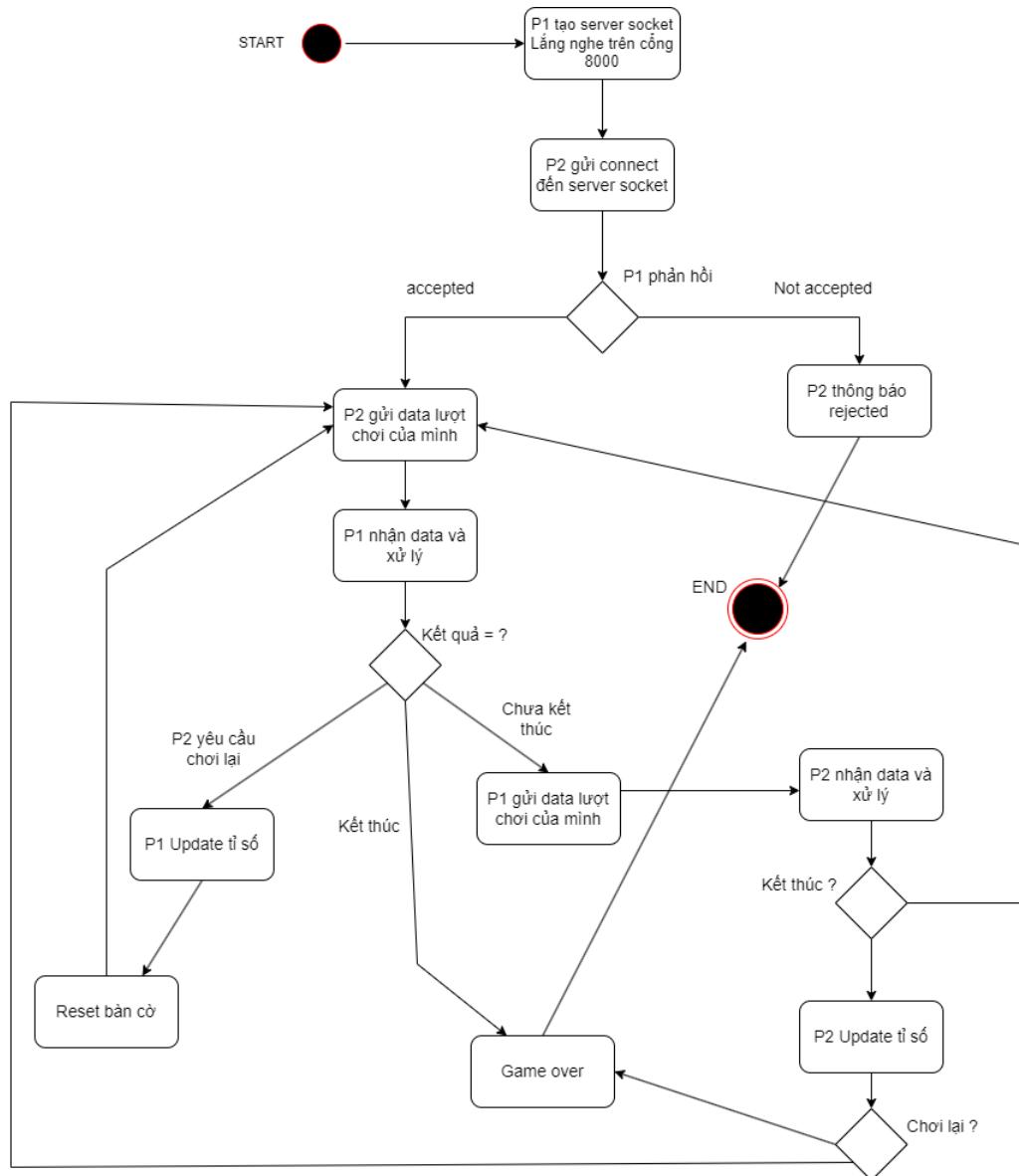
Hình 26: Dụng UI các nút lựa chọn cho player

Frame bên trên cùng (TOP), hiển thị label tên trò chơi.

```
lblTitle = Label(tops, font=('arial',50,'bold'),text="Tic Tac Toe Game", bd=21,
                 bg='Cadet Blue',fg='Cornsilk',justify = CENTER)
lblTitle.grid(row=0,column = 0)
```

Hình 27: Dụng UI hiển thị tên trò chơi

3.4 Flow chart

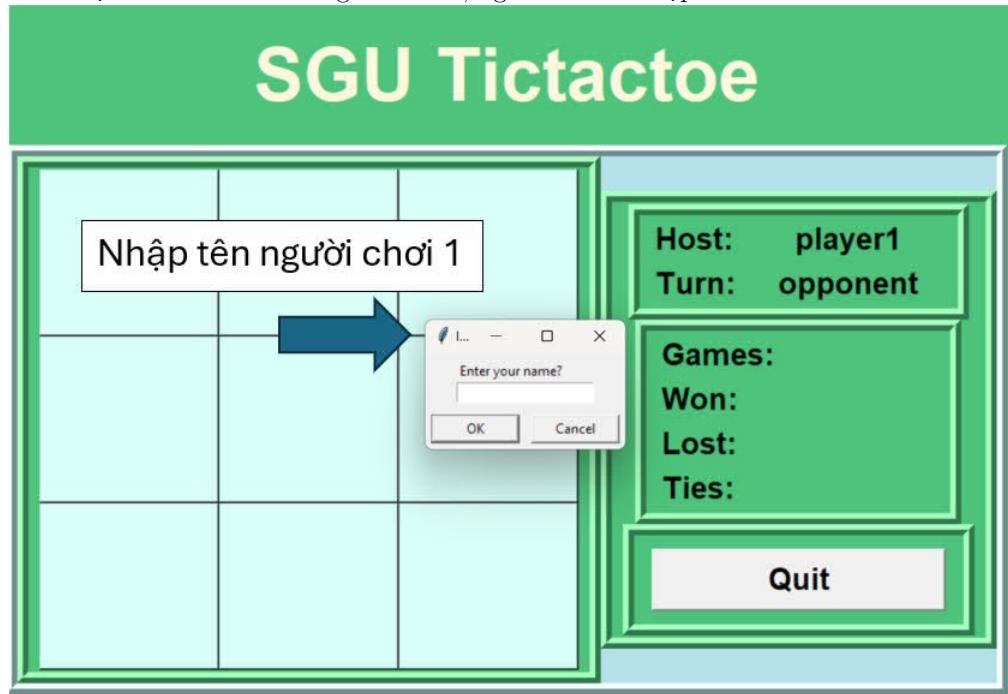


Hình 28: Flow chart của chương trình chạy

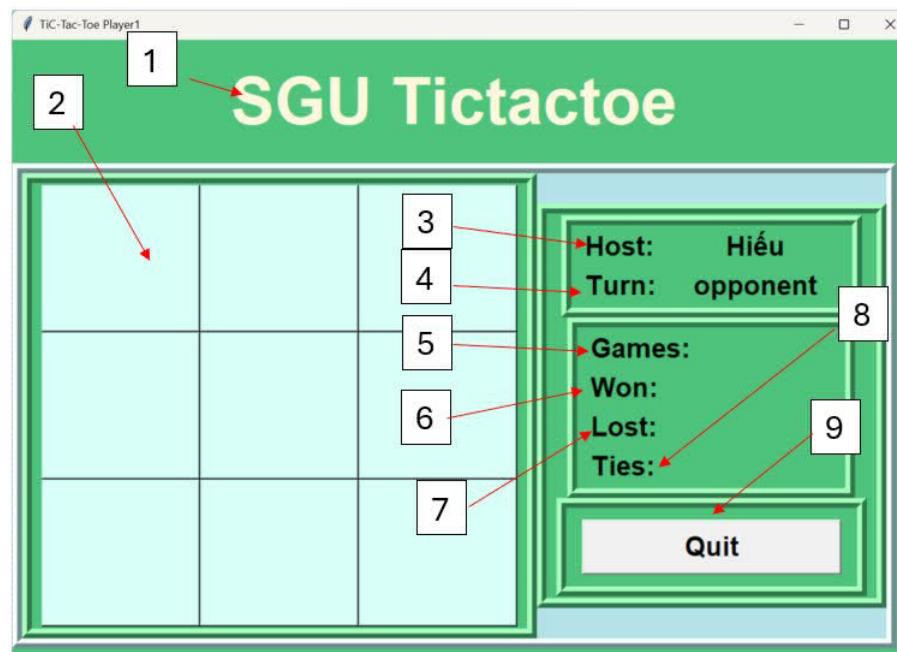
4 Thiết kế giao diện

4.1 Player 1

Giao diện khi bắt đầu của người chơi 1, người chơi sẽ nhập tên



Hình 29: Dialog nhập tên player1

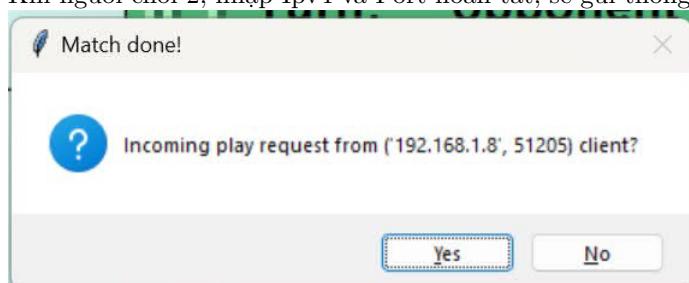


1	Tên tựa game
2	Khu vực để đánh caro
3	Tên người chơi 1
4	Hiển thị lượt chơi hiện tại, Opponent: lượt của đối thủ, You: lượt của bạn
5	Hiển thị số lượt trận đấu
6	Số trận đấu bạn đã thắn
7	Số trận đấu bạn đã thua
8	Số trận hòa
9	Nút thoát

Bảng 1: Bảng thành phần giao diện player 1

Hình 30: Các thành phần giao diện của cửa sổ player1

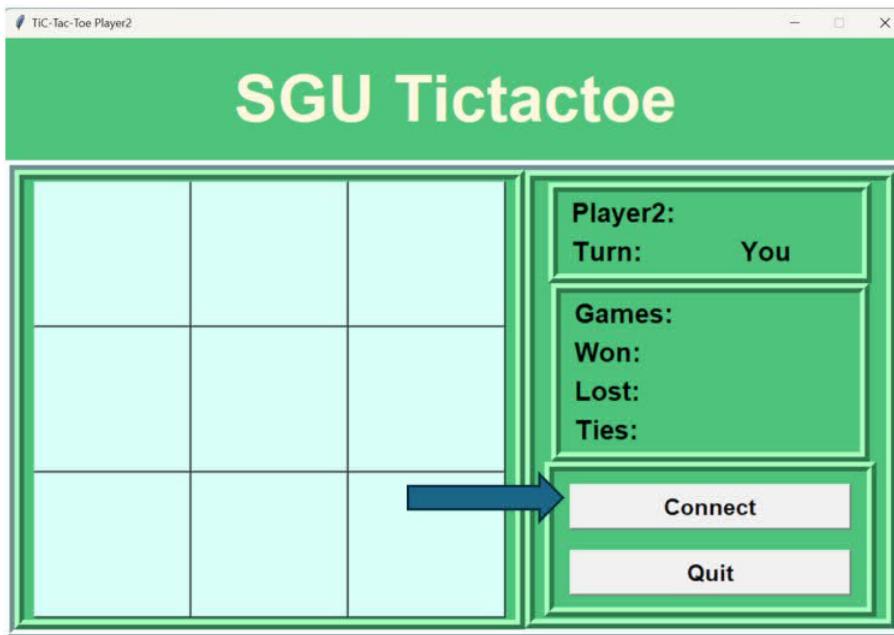
Khi người chơi 2, nhập Ipv4 và Port hoàn tất, sẽ gửi thông báo và đợi phản hồi từ người chơi 1.



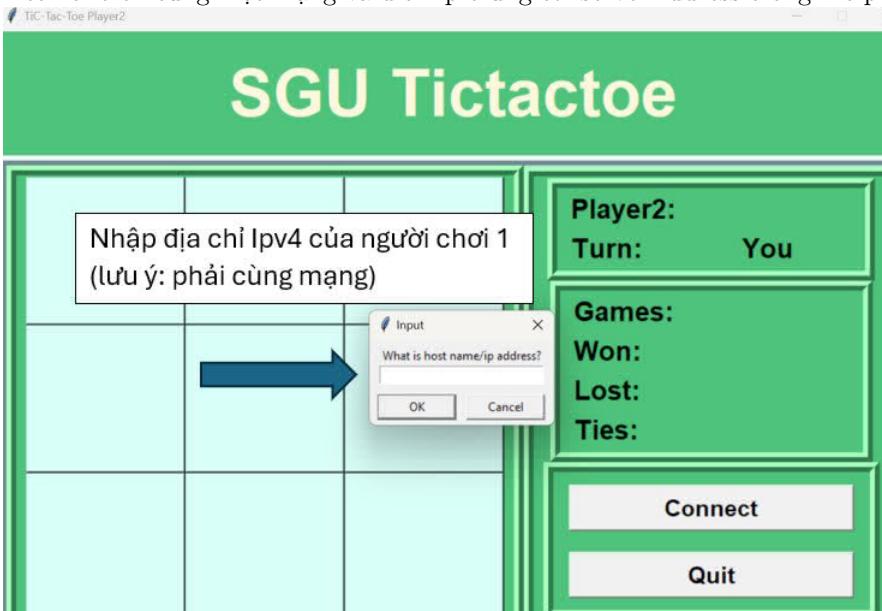
Hình 31: Dialog thông báo kết nối từ player2

4.2 Player2

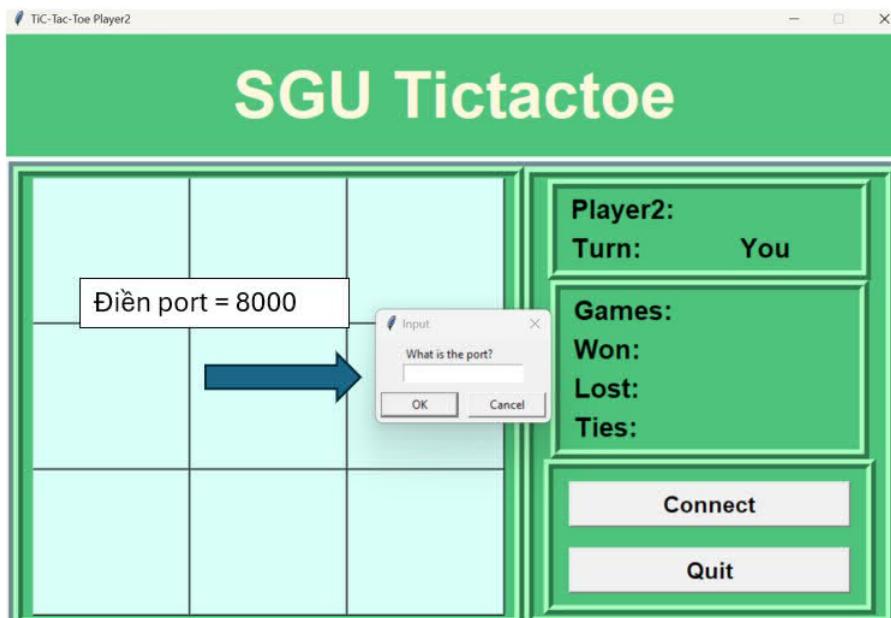
Giao diện của người chơi 2, tương tự người chơi 1, tuy nhiên có thêm nút “Connect” để có thể kết nối tới người chơi 1



Hình 32: Giao diện player2 có thêm nút Connect
Kết nối trên cùng một mạng và điền ip trùng tới serverAddress trong file player1.py

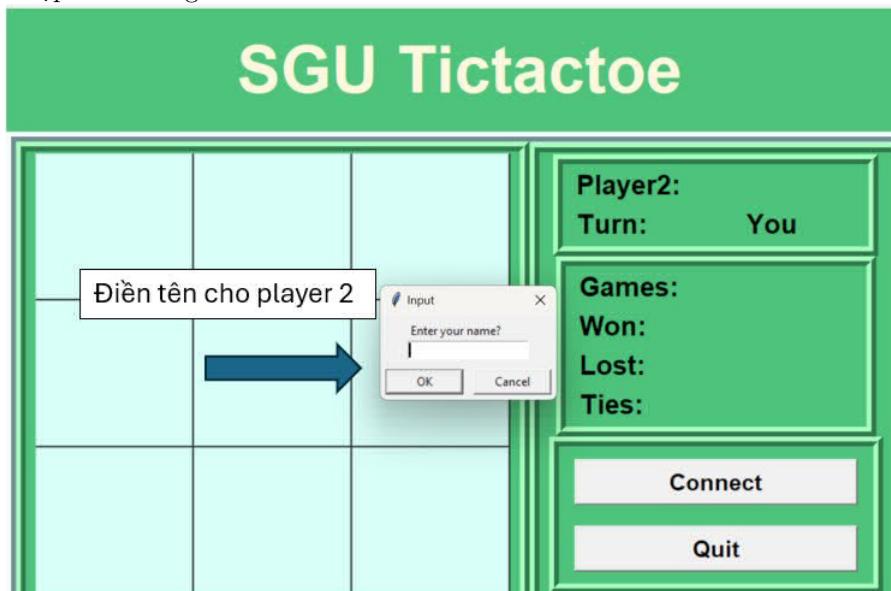


Hình 33: Dialog điền ip address
Điền port = 8000



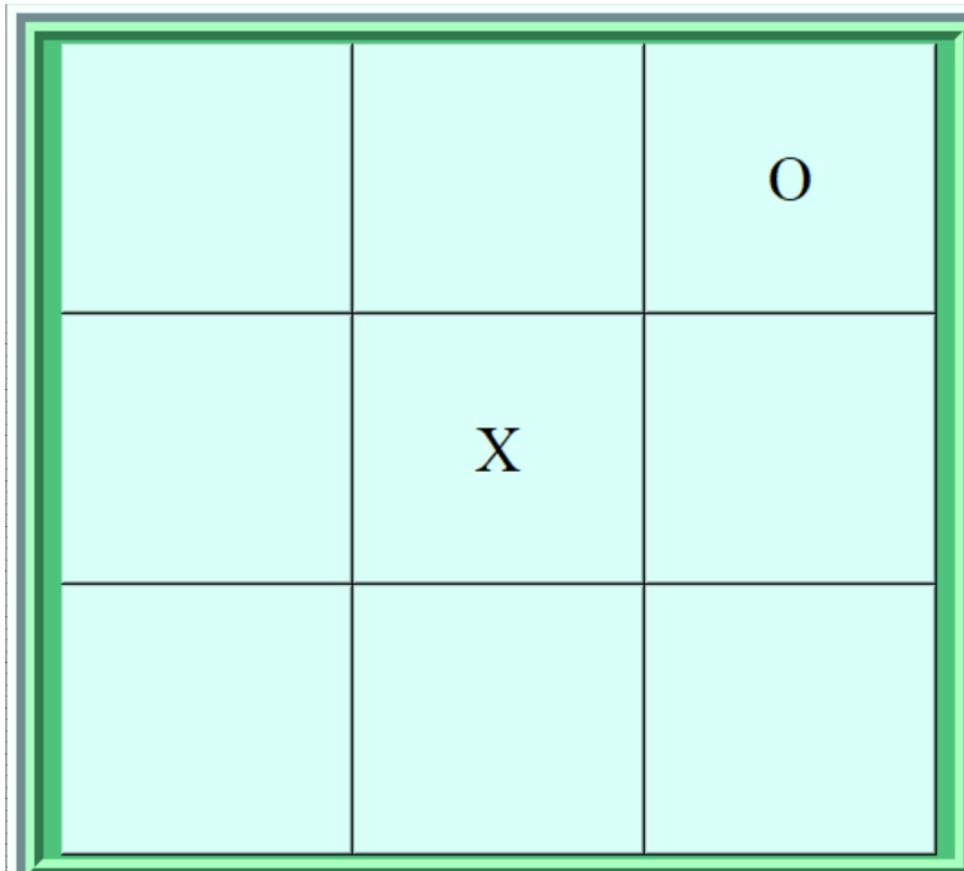
Hình 34: Dialog điền port

Nhập tên cho người chơi 2



Hình 35: Dialog điền tên player2

Người chơi 2 sẽ là “X” và người chơi 1 sẽ là “O”



Hình 36: Quy định ký tự đánh của 2 player

5 Cài đặt và khởi chạy game

5.1 Hướng dẫn cài đặt game

Chuẩn bị môi trường chạy được game:

- Tải và cài đặt python trên máy. <https://www.python.org/downloads/>
+ Bằng lệnh: sudo apt-get install python
- Tải và cài đặt git trên máy. <https://git-scm.com/downloads>
+ Bằng lệnh: sudo apt install git

Trên Windows

Cách 1:

B1: Truy cập đường dẫn sau để điện repo trên Github. https://github.com/baseboy1102002/sgu_ossd_tictactoe

B2: Copy đường dẫn trên hoặc tham khảo ở hình dưới



The screenshot shows a GitHub repository page for 'sgu_ossd_tictactoe'. The repository has 1 branch and 1 tag. The file list includes 'baseboy1102002 Update README.md', '_pycache_ (v1.0 release)', 'LICENSE (add LICENSE)', 'README.md (Update README.rst)', 'gameboard.py (v1.0 release)', 'player1.py (v1.0 release)', and 'player2.py (v1.0 release)'. On the right, there's a 'Clone' section with options for Local, Codespaces, HTTPS, SSH, and GitHub CLI. The HTTPS URL 'https://github.com/baseboy1102002/sgu_ossd_tictactoe' is highlighted with a red box.

B3: Chọn thư mục mà bạn muốn cài đặt game vào, chuột phải chọn Open in Terminal (Hoặc bằng cách thao tác trên Terminal để với lệnh mkdir và cd để tạo và truy cập đến thư mục)

B4: Trong terminal bấm lệnh sau: git init

```
C:\Users\MinhHieu\Desktop\TicTacToe>git init
Initialized empty Git repository in C:/Users/MinhHieu/Desktop/TicTacToe/.git/
C:\Users\MinhHieu\Desktop\TicTacToe>
```

B5: Sau đó bấm lệnh: git clone “đường dẫn”

```
Administrator: Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS C:\Users\MinhHieu\Desktop\TicTacToe> git clone "https://github.com/baseboy1102002/sgu_ossd_tictactoe"
Cloning into 'sgu_ossd_tictactoe'...
remote: Counting objects: 10, done.
remote: Compressing objects: 100% (10/10), done.
remote: Total 10 (delta 0), reused 0 (delta 0)
Unpacking objects: 100% (10/10), done.
```

Cách 2:

B1: Tương tự như trên

B2: Chọn Download file.zip



The screenshot shows a GitHub repository page for 'sgu_ossd_tictactoe'. The repository has 1 branch and 1 tag. The file list includes: __pycache__ (v1.0 release), LICENSE (add LICENSE), README.md (Update README.rst), gameboard.py (v1.0 release), player1.py (v1.0 release), and player2.py (v1.0 release). At the bottom, there are links for README and MIT license. On the right, there's a 'Clone' section with options for HTTPS, SSH, and GitHub CLI, and a 'Download ZIP' button which is highlighted with a red dashed box.

B3: Tải và giải nén file tại thư mục bạn muốn

Trên Linux

Cách 1:

B1: Tải file.zip

The screenshot shows the same GitHub repository page for 'sgu_ossd_tictactoe'. The file list and clone options are identical to the first screenshot. The 'Download ZIP' button at the bottom right is again highlighted with a red dashed box.

B2: Chọn giải nén vào folder mong muốn (Hoặc bằng cách thao tác trên Terminal để với lệnh mkdir và cd để tạo và truy cập đến thư mục)

Cách 2:

B1: mở Terminal

B2: Truy cập vào thư mục muốn tải bằng lệnh cd (nếu tạo dùng lệnh mkdir <tên thư mục>)



B3: dùng lệnh git clone https://github.com/baseboy1102002/sgu_ossd_tictactoe.git

```
minhhieu@HIEUD:~$ ls
test.txt
minhhieu@HIEUD:~$ mkdir TicTacToe
minhhieu@HIEUD:~$ ls
TicTacToe test.txt
minhhieu@HIEUD:~$ cd TicTacToe
minhhieu@HIEUD:~/TicTacToe$ git clone "https://github.com/baseboy1102002/sgu_ossd_tictactoe.git"
Cloning into 'sgu_ossd_tictactoe'...
remote: Enumerating objects: 14, done.
remote: Counting objects: 100% (14/14), done.
remote: Compressing objects: 100% (12/12), done.
remote: Total 14 (delta 3), reused 7 (delta 1), pack-reused 0
Receiving objects: 100% (14/14), 9.72 KiB | 1.94 MiB/s, done.
Resolving deltas: 100% (3/3), done.
minhhieu@HIEUD:~/TicTacToe$ █
```

5.2 Khởi động game

B1: Bạn mở Terminal, gõ lệnh: ipconfig

```
Wireless LAN adapter Wi-Fi:

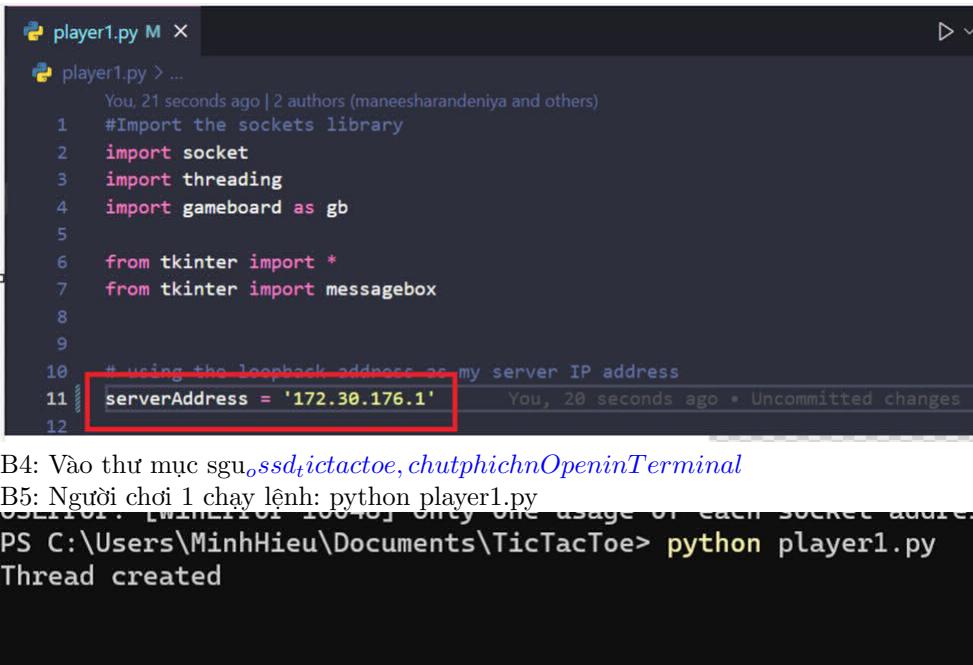
Connection-specific DNS Suffix . :
IPv6 Address . . . . . : 2405:4802:90db:8a50:47fb:b8e0:b472:163e
Temporary IPv6 Address . . . . . : 2405:4802:90db:8a50:ac1c:1bbb:cbl1:c5b2
Link-local IPv6 Address . . . . . : fe80::2dc6:e663:cb8c:8c82%8
IPv4 Address . . . . . : 192.168.1.8
Subnet Mask . . . . . : 255.255.255.0
Default Gateway . . . . . : fe80::1%8
                           192.168.1.1

Ethernet adapter vEthernet (WSL):

Connection-specific DNS Suffix . :
Link-local IPv6 Address . . . . . : fe80::1e9d:5874:52b0:81df%37
IPv4 Address . . . . . : 172.30.176.1
Subnet Mask . . . . . : 255.255.240.0
Default Gateway . . . . . :
```

B2: Truy cập vào thư mục TicTacToe

B3: Lấy địa chỉ Ipv4 trên và gán vào dòng serverAddress = '172.30.176.1' trong file player1.py



```
player1.py M X
player1.py > ...
You, 21 seconds ago | 2 authors (maneesharandeniya and others)
1 #Import the sockets library
2 import socket
3 import threading
4 import gameboard as gb
5
6 from tkinter import *
7 from tkinter import messagebox
8
9
10 # using the loopback address as my server IP address
11 serverAddress = '172.30.176.1' You, 20 seconds ago * Uncommitted changes
12
```

B4: Vào thư mục sgu_ossdictactoe, chutphichnOpeninTerminal

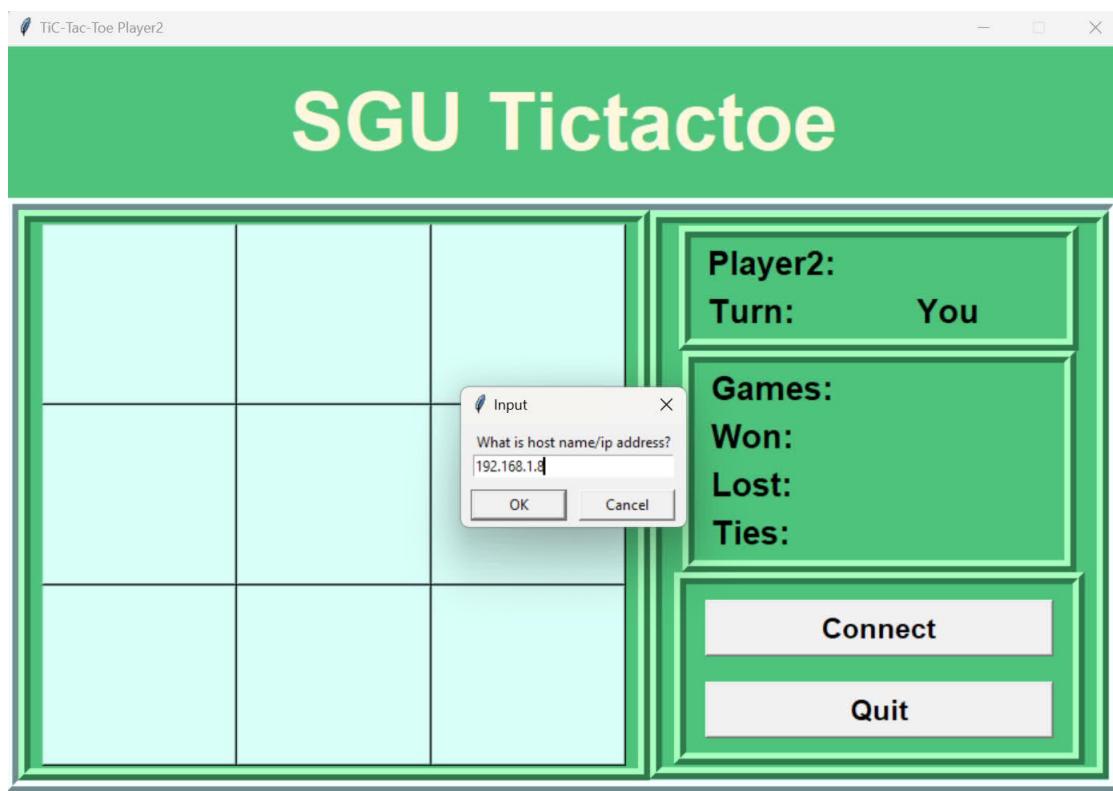
B5: Người chơi 1 chạy lệnh: python player1.py

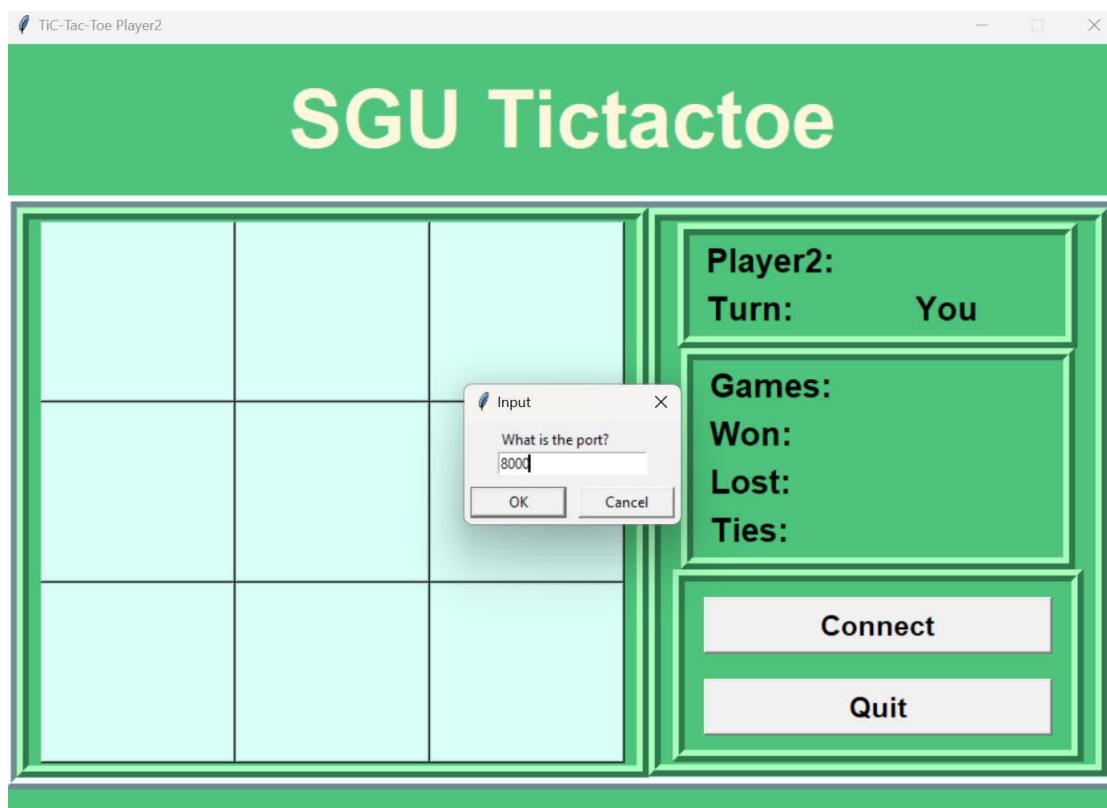
```
PS C:\Users\MinhHieu\Documents\TicTacToe> python player1.py
Thread created
```

B6: Người chơi 2 chạy lệnh: python player2.py

```
PS C:\Users\MinhHieu\Documents\TicTacToe> python player2.py
```

Để 2 thiết bị có thể chơi với nhau, cả hai phải kết nối trên cùng một mạng và player1 phải đổi serverAddress = '172.30.176.1' bằng Ipv4 của máy player 1 và player 2 sẽ kết nối tới địa chỉ '172.30.176.1' và port = 8000





5.3 Kết quả chạy game

Thắng

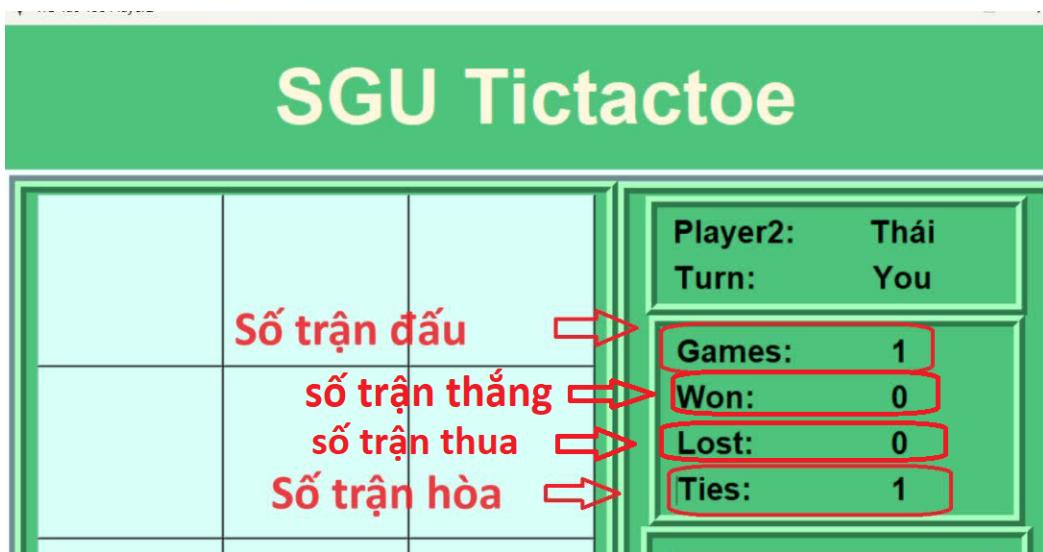
Khi người chơi đạt đủ 3 hàng: đường thẳng hay đường ngang hay đường chéo thì sẽ chiến thắng.
Khi người chơi chiến thắng, sẽ tự động reset lại trận đấu, và hỏi bạn có muốn tiếp tục hay không.
Khi người chơi đồng ý, trận đấu sẽ tiếp tục trận mới

Hòa:

Người chơi khi thua, trận đấu sẽ tự reset lại.

Hòa:

Khi tất cả các ô đều đã được đánh, mà cả hai người chơi đều vẫn không có hàng 3 của đường dọc hay đường ngang hay đường chéo, thì trận đấu sẽ có kết quả hòa





6 Kết luận

6.1 Kết quả thu được

Qua thời gian nghiên cứu và học tập trên lớp học, khảo sát các game tương tự trên thị trường,... Nhóm cuối cùng cũng tự hào “phát hành” sản phẩm game Tic tac toe. Trải qua nhiều lần góp ý sửa đổi, bàn bạc và nâng cấp, phiên bản game đầu tiên bao gồm những chức năng sau:

Game cờ caro giới hạn 9 ô (3x3).

Cơ chế 2 người chơi đối kháng.

Chức năng lưu tì số, phục vụ cho 1 phiên chơi nhiều ván giữa 2 player.

6.2 Ưu và nhược điểm

Ưu điểm

- Phần giao diện: UI vừa mắt, trực quan hiển thị đầy đủ thông tin.
- Trải nghiệm gameplay: mượt mà, đủ đem lại trải nghiệm tốt cho 2 người chơi.
- Hỗ trợ điều hướng các thao tác cho người dùng dễ hiểu, dễ làm quen.
- Thao tác chơi lại ván mới nhanh chóng, tiện dụng.

Nhược điểm

- Phần nhìn còn khá thô sơ do sử dụng các thư viện cơ bản của Python và không có nhiều sự tùy biến, đem lại cảm giác hiện đại như các app, game desktop trên thị trường hiện nay.
- Phần kết nối (matching) giữa 2 người chơi còn vài phần phức tạp, chương trình chưa hỗ trợ tối đa giúp người dùng ở bước này.

6.3 Hướng phát triển nâng cấp tương lai

Trên cơ sở những chức năng đã thực hiện, đề tài có một số hướng phát triển trong tương lai như sau:

- Nâng cấp giao diện.
- Xây dựng chức năng matching đơn giản, nhanh chóng hơn cho phía end user.
- Phát triển và đưa vào trò chơi chế độ chơi với máy (dùng AI đơn giản).
- Phát triển chế độ nhiều hơn 2 người chơi: hàng đợi khiêu chiến, khán giả,...



7 Tài liệu tham khảo

- [1]. Python Documentation. <https://docs.python.org/3/>
- [2]. Socket Programming in Python. <https://realpython.com/python-sockets/#background>
- [3]. An Intro to Threading in Python. <https://realpython.com/intro-to-python-threading/#starting-a-thread>
- [4]. Lập trình socket bằng Python. <https://viblo.asia/p/lap-trinh-socket-bang-python-jvEla084Zkw>
- [5]. Tic-tac-toe. <https://vi.wikipedia.org/wiki/Tic-tac-toe>
- [6]. Python Tic Tac Toe game. <https://www.youtube.com/watch?v=V9MbQ2Xl4CE>