\documentclass[a4paper,12pt]{report}

\usepackage[utf8]{inputenc}

\usepackage[vietnamese]{babel}

\usepackage{graphicx}

\usepackage{amsmath, amssymb}

\usepackage[hidelinks]{hyperref}

\usepackage{pdfpages}

\begin{document}

\includepdf{bia.pdf} % Chèn file bia.pdf

\newpage

% Lời cảm ơn

\chapter\*{Lời cảm ơn}

\begin{quote}

Lời đầu, chúng tôi xin được gửi lời cảm ơn thành đến cô giáo/giảng viên Ths.Lê Thị Thuỳ Trang. Trong quá trình thực hiện đề tài này, chúng tôi đã nhận được sự hỗ trợ, hướng dẫn và động viên quý báu từ nhiều phía. Nhưng trước hết, chúng tôi xin bày tỏ lòng biết ơn sâu sắc đến \textbf{Cô giáo - ThS. Lê Thị Thùy Trang}, người đã trực tiếp giảng dạy và truyền đạt những kiến thức quý giá về môn học \textbf{Trí tuệ nhân tạo}. Qua những bài giảng, chúng tôi không chỉ đã nắm vững được lý thuyết mà còn truyền cảm hứng để chúng tôi áp dụng những kiến thức đó vào thực tiễn.

Cũng nhờ đó mà chúng tôi hiểu sâu hơn về các khái niệm cốt lõi của Trí tuệ nhân tạo, từ những nền tảng cơ bản như \textbf{logic mệnh đề}, \textbf{biểu diễn tri thức}, đến những thuật toán phức tạp như \textbf{thuật toán tìm kiếm A\*}, \textbf{các phương pháp heuristic}, và nhiều kỹ thuật khác. Những kiến thức này không chỉ là hành trang quan trọng cho đề tài của chúng tôi mà còn là nền tảng vững chắc cho hành trình học tập và nghiên cứu trong tương lai.

Một lần nữa, chúng tôi xin chân thành cảm ơn tất cả những sự giúp đỡ quý báu mà chúng tôi đã nhận được. Để từ đó, nhóm 9 với 3 thành viên của chúng tôi cũng đã hoàn thành đề tài "Xây dựng trò chơi Rắn Săn Mồi với thuật toán A\*". Hy vọng rằng kết quả của đề tài này sẽ phần nào đáp lại được sự kỳ vọng và công sức mà mọi người đã dành cho chúng tôi.

Sản phẩm này là sự cố gắng của nhóm 9 chúng tôi, gồm các thành viên: \href{https://www.overleaf.com/project/62986f62b65db6f03abab3b8}{Lê Ngọc Minh},\href{https://www.overleaf.com/project/62986f62b65db6f03abab3b8}{Nguyễn Văn Tấn},\href{https://www.overleaf.com/project/62986f62b65db6f03abab3b8}{Đỗ Khánh Hùng},

\end{quote}

\tableofcontents % Thêm mục lục

\chapter{Giới thiệu}

\section {Tổng quan về trí tuệ nhân tạo}

\subsection{Định nghĩa trí tuệ nhân tạo}

Trí tuệ nhân tạo (Artificial Intelligence - AI) là một trong những lĩnh vực tiên phong và đầy tiềm năng của khoa học máy tính, đóng vai trò quan trọng trong việc định hình tương lai công nghệ và thay đổi cách con người tương tác với thế giới xung quanh. Đây là một ngành khoa học tập trung vào việc nghiên cứu, phát triển và ứng dụng các hệ thống, thuật toán, cũng như các mô hình tính toán có khả năng mô phỏng và tái hiện những đặc trưng của trí tuệ con người. Nhờ vào AI, máy tính không chỉ thực hiện các tác vụ cơ bản theo lập trình sẵn mà còn có thể tự động hóa các quá trình phức tạp như suy luận logic, học hỏi từ dữ liệu, thích nghi với những thay đổi của môi trường, và đưa ra các quyết định thông minh để giải quyết những vấn đề thực tiễn một cách hiệu quả.

Khả năng của AI không dừng lại ở việc bắt chước trí tuệ con người, mà còn mở rộng đến việc xử lý và phân tích khối lượng dữ liệu khổng lồ với tốc độ và độ chính xác vượt xa khả năng của con người. Trong công nghệ, AI được ứng dụng để phát triển các hệ thống nhận dạng giọng nói như Siri hay Alexa, giúp con người giao tiếp với thiết bị một cách dễ dàng. Trong lĩnh vực xử lý hình ảnh, AI hỗ trợ nhận diện khuôn mặt, phân loại ảnh, và thậm chí phát hiện các bất thường trong y học như chẩn đoán ung thư qua hình ảnh chụp X-quang. Ngoài ra, AI còn đóng vai trò quan trọng trong ngành công nghiệp xe tự hành, nơi các thuật toán thông minh giúp xe tự động điều hướng, tránh chướng ngại vật và đảm bảo an toàn trên đường. Trong tài chính, AI được sử dụng để dự đoán xu hướng thị trường, phát hiện gian lận, và tối ưu hóa các chiến lược đầu tư. Y tế cũng là một lĩnh vực hưởng lợi lớn từ AI, với các hệ thống hỗ trợ bác sĩ trong việc chẩn đoán bệnh, cá nhân hóa phác đồ điều trị, và quản lý dữ liệu bệnh nhân. Đặc biệt, trong ngành công nghiệp trò chơi điện tử, AI mang đến những trải nghiệm sống động hơn thông qua việc tạo ra các nhân vật ảo thông minh, tự động điều chỉnh độ khó, và thiết kế các kịch bản chơi đa dạng dựa trên hành vi của người chơi.

Sự phát triển không ngừng của trí tuệ nhân tạo không chỉ mở ra những cơ hội mới mà còn đặt ra nhiều thách thức về mặt kỹ thuật, đạo đức và xã hội. Làm thế nào để AI có thể hoạt động một cách minh bạch, công bằng và không gây ra những tác động tiêu cực đến con người là một câu hỏi lớn mà các nhà khoa học, vậy tiếp theo đây, chúng ta sẽ cùng tìm hiểu sâu hơn về công nghệ AI - công nghệ trí tuệ nhân tạo.

\subsection{Các nhánh chính của trí tuệ nhân tạo}

Trí tuệ nhân tạo (AI) là một lĩnh vực vô cùng rộng lớn và đa dạng,nó bao gồm nhiều nhánh khác nhau, mỗi nhánh tập trung vào một khía cạnh cụ thể của việc mô phỏng trí tuệ con người và giải quyết các vấn đề thực tiễn. Những nhánh này không chỉ bổ trợ lẫn nhau mà còn đóng vai trò quan trọng trong việc mở rộng phạm vi ứng dụng của AI, ở đây, chúng ta sẽ cùng đi tìm hiểu về các nhánh chính:

\begin{itemize}

\item \textbf{Học máy (Machine Learning - ML):} Học máy là một trong những nhánh quan trọng và phổ biến nhất của trí tuệ nhân tạo, đóng vai trò nền tảng cho nhiều ứng dụng AI hiện nay. Học máy cho phép máy tính tự động học hỏi và cải thiện hiệu suất từ dữ liệu mà không cần được con người lập trình cụ thể cho từng tác vụ. Qua đó giúp tiết kiệm thời gian, tăng tính linh hoạt và mở rộng khả năng xử lý của máy tính trong các tình huống phức tạp. Học máy lại được chia thành ba loại chính, mỗi loại có cách tiếp cận và ứng dụng riêng biệt:

\begin{itemize}

\item \textit{Học có giám sát (Supervised Learning):} Đây là loại học máy phổ biến nhất, trong đó máy tính được huấn luyện trên một tập dữ liệu đã được gắn nhãn sẵn.

\item \textit{Học không giám sát (Unsupervised Learning):} Khác với học có giám sát, học không giám sát hoạt động trên dữ liệu không có nhãn. Máy tính sẽ tự động tìm kiếm các mẫu hình, cấu trúc hoặc mối quan hệ ẩn trong dữ liệu.

\item \textit{Học tăng cường (Reinforcement Learning):} Đây là một loại học máy mà máy tính học hỏi thông qua việc thử và sai trong một môi trường cụ thể. Máy sẽ nhận được phần thưởng hoặc hình phạt dựa trên hành động của mình, từ đó điều chỉnh chiến lược để tối ưu hóa phần thưởng trong tương lai. Điểm nổi bật của học tăng cường là khả năng thích nghi với các tình huống mới mà không cần dữ liệu huấn luyện cố định.

\end{itemize}

\item \textbf{Xử lý ngôn ngữ tự nhiên (Natural Language Processing - NLP):} Xử lý ngôn ngữ tự nhiên cũng là một nhánh quan trọng của trí tuệ nhân tạo, nó tập trung vào việc giúp máy tính hiểu, phân tích và tạo ra ngôn ngữ con người một cách tự nhiên và hiệu quả. Đây là một lĩnh vực đầy thách thức, bởi ngôn ngữ con người không chỉ phức tạp về mặt ngữ pháp mà còn chứa đựng nhiều sắc thái văn hóa, ngữ cảnh, và cảm xúc. NLP cho phép máy tính giao tiếp với con người thông qua ngôn ngữ nói hoặc viết, từ đó mở ra nhiều ứng dụng thực tiễn trong đời sống hàng ngày. Một trong những ứng dụng nổi bật của NLP là các chatbot và trợ lý ảo, chẳng hạn như Siri, Alexa, hay Google Assistant, có khả năng hiểu câu hỏi của người dùng và trả lời một cách tự nhiên. Ngoài ra, NLP còn được sử dụng trong dịch thuật tự động, giúp phá vỡ rào cản ngôn ngữ giữa các quốc gia. Trong tương lai, NLP hứa hẹn sẽ tiếp tục mở rộng khả năng giao tiếp giữa con người và máy, mang lại những trải nghiệm tự nhiên và liền mạch hơn.

\item \textbf{Thị giác máy tính (Computer Vision):} Thị giác máy tính là một nhánh tiếp theo của trí tuệ nhân tạo tập trung vào việc giúp máy tính "nhìn" và hiểu được thế giới xung quanh thông qua hình ảnh và video. Nhiệm vụ chính của thị giác máy tính là phân tích, xử lý và trích xuất thông tin từ dữ liệu hình ảnh, từ đó đưa ra các quyết định hoặc nhận diện các đối tượng trong không gian. Đây là một lĩnh vực đòi hỏi sự kết hợp giữa các thuật toán AI, học sâu, và xử lý tín hiệu số để đạt được độ chính xác cao. Một trong những ứng dụng phổ biến nhất của thị giác máy tính là nhận diện khuôn mặt, được sử dụng trong các hệ thống bảo mật như mở khóa điện thoại bằng khuôn mặt hoặc nhận diện tội phạm qua camera giám sát. Ngoài ra, thị giác máy tính còn đóng vai trò quan trọng trong xe tự hành, nơi các thuật toán AI giúp xe nhận diện biển báo giao thông, làn đường, người đi bộ, và các chướng ngại vật để điều hướng an toàn. Trong y tế cũng vậy, thị giác máy tính hỗ trợ bác sĩ trong việc phân tích hình ảnh y khoa, chẳng hạn như phát hiện khối u trong ảnh chụp X-quang hoặc MRI với độ chính xác cao. Các ứng dụng khác bao gồm nhận diện đối tượng trong công nghiệp (kiểm tra lỗi sản phẩm trên dây chuyền sản xuất), phân loại hình ảnh trên mạng xã hội, và thậm chí hỗ trợ người khiếm thị thông qua các thiết bị mô tả hình ảnh. Thị giác máy tính đã đạt được những bước tiến vượt bậc nhờ vào sự phát triển của các mạng nơ-ron tích chập (Convolutional Neural Networks - CNN), cho phép máy tính nhận diện các đặc trưng phức tạp trong hình ảnh như màu sắc, hình dạng, và kết cấu.

\item \textbf{Tìm kiếm heuristic (Heuristic Search):} Tìm kiếm heuristic là nhánh cuối cùng của trí tuệ nhân tạo mà chúng tôi nói đến, nó tập trung vào việc phát triển các thuật toán thông minh để giải quyết các bài toán tìm kiếm và tối ưu hóa, đặc biệt trong các không gian trạng thái lớn và phức tạp. Không giống như các phương pháp tìm kiếm truyền thống (như tìm kiếm theo chiều rộng hoặc chiều sâu), tìm kiếm heuristic sử dụng các chiến lược thông minh để giảm thiểu thời gian tính toán bằng cách ưu tiên các hướng đi có khả năng dẫn đến mục tiêu cao hơn. Và chắc chắn phải kể đến, một trong những thuật toán nổi bật trong tìm kiếm heuristic là thuật toán A\*, được sử dụng rộng rãi trong các bài toán tìm đường. Thuật toán A\* hoạt động bằng cách kết hợp chi phí đã đi qua (g) và chi phí ước lượng đến mục tiêu (h) để tìm ra con đường ngắn nhất từ điểm xuất phát đến đích. Điều này làm cho A\* trở thành một lựa chọn lý tưởng trong các ứng dụng như tìm đường trong bản đồ (ví dụ: Google Maps, hay các ứng dụng kết nối bạn bè...), hay điều hướng robot, hoặc trong các trò chơi điện tử, nơi các nhân vật ảo cần di chuyển một cách thông minh để tránh chướng ngại vật và đến được mục tiêu. Trong game "Rắn săn mồi của nhóm chúng tôi, thuật toán A\* đã được chúng tôi áp dụng để giúp rắn tự động tìm đường đến thức ăn, tránh các chướng ngại vật và chính phần thân của nó, tạo ra một trải nghiệm chơi mượt mà và thông minh. Tìm kiếm heuristic không chỉ giới hạn ở thuật toán A\*, mà còn bao gồm các phương pháp khác như tìm kiếm tham lam (Greedy Search) hoặc tìm kiếm leo đồi (Hill Climbing), mỗi phương pháp có ưu điểm riêng tùy thuộc vào bài toán cụ thể. Nhờ vào tính hiệu quả và khả năng mở rộng, tìm kiếm heuristic đã trở thành một công cụ quan trọng trong AI, không chỉ trong trò chơi mà còn trong các lĩnh vực như lập kế hoạch, tối ưu hóa logistics, và thiết kế hệ thống tự động.

\end{itemize}

Các nhánh chính của trí tuệ nhân tạo, từ học máy, xử lý ngôn ngữ tự nhiên, thị giác máy tính, đến tìm kiếm heuristic, đều đóng vai trò quan trọng trong việc mở rộng khả năng của AI và đưa công nghệ này vào mọi khía cạnh của đời sống. Mỗi nhánh không chỉ có những đặc trưng riêng mà còn bổ trợ lẫn nhau, tạo nên một hệ sinh thái AI mạnh mẽ và đa dạng. Sự phát triển không ngừng của các nhánh này hứa hẹn sẽ mang lại nhiều đột phá mới, từ việc cải thiện các hệ thống thông minh trong y tế, giáo dục, đến việc tạo ra những trải nghiệm giải trí sống động hơn trong ngành công nghiệp trò chơi điện tử. Trong bối cảnh công nghệ ngày càng tiến bộ, việc hiểu rõ và tận dụng các nhánh của AI sẽ là chìa khóa để khai thác tối đa tiềm năng của trí tuệ nhân tạo trong tương lai.

\section{Ứng dụng của trí tuệ nhân tạo trong trò chơi điện tử}

Không thể phủ nhận khi trí tuệ nhân tạo (AI) đã và đang trở thành một công cụ không thể thiếu trong ngành công nghiệp trò chơi điện tử, góp phần nâng cao trải nghiệm người chơi và tạo ra những tựa game thông minh, hấp dẫn hơn. Bằng cách tích hợp các thuật toán AI, các nhà phát triển có thể thiết kế những hệ thống tự động, linh hoạt, và có khả năng thích nghi với hành vi của người chơi, từ đó mang lại những trải nghiệm chơi game sống động và chân thực. AI không chỉ giúp cải thiện tính tương tác trong trò chơi mà còn hỗ trợ các nhà phát triển trong việc tối ưu hóa hiệu suất, thiết kế các kịch bản chơi đa dạng, và tạo ra các nhân vật ảo có khả năng ra quyết định một cách thông minh. Dưới đây là một số ứng dụng nổi bật của trí tuệ nhân tạo trong lĩnh vực trò chơi điện tử do chúng tôi tổng hợp được, từ đó cho thấy rõ vai trò của AI trong việc nâng cao chất lượng và sự sáng tạo của các tựa game hiện đại.

\begin{itemize}

\item \textbf{Hệ thống đối thủ tự động trong game chiến thuật:} Một trong những ứng dụng quan trọng nhất của AI trong trò chơi điện tử là tạo ra các đối thủ tự động (Non-Player Characters - NPCs), điểm nổi bật là chúng có khả năng ra quyết định thông minh, đặc biệt trong các game chiến thuật hoặc nhập vai. AI giúp các NPCs không chỉ thực hiện các hành động cơ bản mà còn phân tích tình huống, lập kế hoạch, và phản ứng linh hoạt với chiến thuật của người chơi. Ví dụ, trong các game chiến thuật như \textit{Đế Chế} hoặc \textit{Mount of Blade}, AI có thể điều khiển các đội quân đối thủ, xây dựng chiến lược tấn công hoặc phòng thủ dựa trên tình hình trận đấu. Điều này không chỉ tăng độ khó và tính thử thách của trò chơi mà còn mang lại cảm giác chân thực, như thể người chơi đang được đối đầu với một đối thủ thực sự. Một số khác, như ở trong tựa game nổi tiếng mới ra mắt gần đây \textit{Kingdom Come: Deliverance II}, các thuật toán AI như học tăng cường (Reinforcement Learning) được sử dụng để huấn luyện các NPCs, giúp chúng học hỏi từ các lựa chọn của người chơi trước đó và đưa ra các quyết định khác nhau, từ đó tạo ra những cái kết khác nhau, làm phong phú thêm phần cho thể loại game theo cốt truyện.

\item \textbf{Hướng dẫn và hỗ trợ người chơi qua AI chatbot:} AI cũng được ứng dụng để hỗ trợ người chơi thông qua các hệ thống chatbot thông minh, giúp cải thiện trải nghiệm chơi game, đặc biệt đối với những người chơi mới (newbie). Các chatbot này được xây dựng dựa trên công nghệ xử lý ngôn ngữ tự nhiên (Natural Language Processing - NLP), chúng có khả năng hiểu và trả lời các câu hỏi của người chơi một cách tự nhiên. Ví dụ, trong các game nhập vai trực tuyến (MMORPG) như \textit{Albion Online}, AI chatbot có thể hướng dẫn người chơi cách hoàn thành nhiệm vụ, tìm kiếm vật phẩm, hoặc giải thích các cơ chế gameplay phức tạp. Điều này không chỉ giúp giảm bớt sự bỡ ngỡ cho người chơi mới mà còn tạo ra một môi trường chơi game thân thiện, nơi người chơi sẽ cảm thấy được hỗ trợ kịp thời. Hơn nữa, các chatbot AI còn có thể được cá nhân hóa để phù hợp với phong cách chơi của từng người, từ đó mang lại trải nghiệm chơi game độc đáo và thú vị hơn.

\item \textbf{Tối ưu hóa đường đi của nhân vật trong game bằng thuật toán A\*:} Một ứng dụng quan trọng khác của AI trong trò chơi điện tử là tối ưu hóa đường đi của các nhân vật trong game, đặc biệt trong các trò chơi yêu cầu điều hướng thông minh như game phiêu lưu, chiến thuật, hoặc các game mô phỏng. Thuật toán A\* (A-star) là một trong những thuật toán tìm đường phổ biến nhất, được sử dụng để giúp các nhân vật ảo di chuyển một cách hiệu quả từ điểm xuất phát đến đích, đồng thời tránh các chướng ngại vật trên bản đồ. Trong các game như \textit{The Legend of Zelda} hoặc \textit{Assassin’s Creed}, thuật toán A\* được áp dụng để điều khiển các NPCs di chuyển mượt mà qua các môi trường phức tạp, chẳng hạn như thành phố đông đúc hoặc khu rừng rậm rạp hay thậm chí là các địa hình hiểm trở. Việc sử dụng A\* không chỉ giúp giảm tải cho người chơi trong việc điều khiển mà còn tăng tính chân thực của trò chơi, khi các nhân vật ảo có thể tự động tìm ra con đường tối ưu mà không cần can thiệp từ người chơi. Hơn nữa, thuật toán A\* còn có thể được mở rộng để xử lý các tình huống phức tạp hơn, chẳng hạn như bản đồ thay đổi theo thời gian hoặc các chướng ngại vật di động, từ đó nâng cao độ khó và tính thử thách của trò chơi.

\end{itemize}

Nhìn chung, trí tuệ nhân tạo đã và đang mở ra một kỷ nguyên mới cho ngành công nghiệp trò chơi điện tử và cả ngành công nghiệp giải trí, nơi các tựa game không chỉ là một phương tiện giải trí mà còn là một nền tảng để thử nghiệm và phát triển các công nghệ AI tiên tiến. Từ việc tạo ra các đối thủ thông minh, có bản chất riêng, biết hỗ trợ người chơi, đến tối ưu hóa đường đi của nhân vật, AI đã và đang góp phần định hình cách chúng ta trải nghiệm và tương tác với thế giới ảo, mang lại những giá trị giải trí và sáng tạo vượt bậc so với khoảng thời gian 20 năm trước.

\section{Giới thiệu về trò chơi rắn săn mồi}

Trò chơi rắn săn mồi (Snake Game) là một trong những tựa game kinh điển đã tồn tại được nửa thế kỉ, nó cũng đã trở thành biểu tượng của ngành công nghiệp trò chơi điện tử nhờ vào lối chơi đơn giản nhưng đầy thử thách. Vậy trước khi đi sâu vào tìm hiểu "Rắn săn mồi của chúng tôi", ta hãy cùng nhau tìm hiểu trước để có một cái nhìn chi tiết về lịch sử phát triển và cách chơi cơ bản của trò chơi rắn săn mồi, giúp làm rõ nguồn gốc và ý nghĩa của tựa game này.

\subsection{Lịch sử phát triển trò chơi rắn săn mồi}

Trò chơi rắn săn mồi lần đầu tiên xuất hiện vào năm 1976, dưới dạng một trò chơi arcade có tên \textit{Blockade} được phát triển bởi công ty Gremlin Industries. Trong phiên bản gốc, người chơi điều khiển một "con rắn" di chuyển trên màn hình và để lại một vệt dài phía sau, trò chơi sẽ kết thúc khi rắn va chạm với vệt của chính mình hoặc của đối thủ. Ý tưởng tưởng chừng như đơn giản này nhưng thực chất lại vô cùng sáng toạ, nó đã nhanh chóng thu hút sự chú ý của cộng đồng game thủ, mở đường cho sự phát triển của nhiều phiên bản khác nhau trong những năm sau đó. Tuy nhiên, trò chơi rắn săn mồi thực sự trở thành một hiện tượng bùng nổ toàn cầu vào cuối thập niên 1990, khi nó được tích hợp vào các dòng điện thoại Nokia, đặc biệt là mẫu Nokia 6110, dưới dạng một trò chơi có sẵn mang tên \textit{Snake}. Với giao diện đơn giản, lối chơi dễ hiểu mà vô cùng cuốn hút bởi tính thị hiếu quá đơn giản bất giờ từ đó gây ra tính gây nghiện cao, \textit{Snake} đã trở thành một phần ký ức tuổi thơ của hàng triệu người trên toàn thế giới trong đó có chính bản thân tôi, đồng thời góp phần đưa trò chơi điện tử đến gần hơn với công chúng.

Trong những năm sau đó, trò chơi rắn săn mồi tiếp tục được cải tiến và phát triển trên nhiều nền tảng khác nhau, từ máy tính cá nhân, điện thoại thông minh, đến các hệ máy chơi game hiện đại. Các phiên bản mới không chỉ giữ nguyên lối chơi truyền thống mà còn bổ sung nhiều tính năng hiện đại, chẳng hạn như đồ họa cải tiến, âm thanh sống động, và các chế độ chơi đa dạng. Đặc biệt, với sự phát triển của trí tuệ nhân tạo, trò chơi rắn săn mồi đã được nâng cấp với các tính năng thông minh, chẳng hạn như tích hợp thuật toán tìm đường để giúp rắn tự động di chuyển, hoặc thêm các chướng ngại vật di động để tăng độ khó. Những cải tiến này không chỉ làm tăng tính thử thách của trò chơi mà còn mở ra cơ hội để các nhà phát triển thử nghiệm và áp dụng các công nghệ AI tiên tiến, biến trò chơi rắn săn mồi từ một tựa game đơn giản thành một nền tảng nghiên cứu và học tập về trí tuệ nhân tạo.

\subsection{Cách chơi cơ bản}

Trò chơi rắn săn mồi có lối chơi đơn giản nhưng đầy thử thách, phù hợp với mọi lứa tuổi từ trẻ em đến người lớn và mọi loại trình độ. Dưới đây ta cùng đi vào tìm hiểu các quy tắc cơ bản của trò chơi:

\begin{itemize}

\item \textbf{Người chơi điều khiển rắn di chuyển trên lưới ô vuông:} Trò chơi diễn ra trên một bản đồ dạng lưới, thường là một hình chữ nhật được chia thành các ô vuông nhỏ. Người chơi sử dụng các phím điều hướng (lên, xuống, trái, phải) để điều khiển hướng di chuyển của rắn. Rắn sẽ di chuyển liên tục theo hướng được chọn, và nhiệm vụ của người chơi là đảm bảo rắn không va chạm với các vật cản trên bản đồ.

\item \textbf{Khi rắn ăn thức ăn, nó sẽ dài ra:} Trên bản đồ, thức ăn sẽ xuất hiện ngẫu nhiên tại một ô vuông. Khi rắn di chuyển đến ô chứa thức ăn, nó sẽ "ăn" thức ăn và tăng chiều dài thêm một đoạn. Mỗi lần ăn thức ăn, người chơi sẽ được cộng điểm, và trò chơi sẽ tiếp tục với một thức ăn mới xuất hiện ở vị trí khác.

\item \textbf{Trò chơi kết thúc nếu rắn đâm vào tường hoặc chính thân của nó:} Trò chơi sẽ kết thúc nếu rắn va chạm với các vật cản, bao gồm tường bao quanh bản đồ hoặc chính thân của nó. Trong các phiên bản truyền thống, tường là ranh giới cố định của bản đồ, và rắn sẽ thua nếu chạm vào tường. Ngoài ra, nếu rắn quay đầu và va chạm với thân của chính mình (do người chơi điều khiển sai hướng), trò chơi cũng sẽ kết thúc.

\end{itemize}

\section{Mục tiêu của đề tài}

Đề tài này của chúng tôi tập trung vào việc xây dựng một phiên bản nâng cao của trò chơi rắn săn mồi, chúng tôi tích hợp trí tuệ nhân tạo để tạo ra một hệ thống thông minh, có khả năng tự động điều hướng và thích nghi với các tình huống phức tạp. Bằng cách áp dụng các thuật toán AI, đặc biệt là thuật toán A\*. Mục tiêu chính của đề tài là minh họa cách trí tuệ nhân tạo có thể được ứng dụng trong trò chơi điện tử, đồng thời cung cấp một nền tảng thực nghiệm để nghiên cứu và phát triển các thuật toán tìm đường trong thực tế. Dưới đây là các mục tiêu cụ thể mà nhóm đặt ra, được thiết kế để đảm bảo hệ thống không chỉ hoạt động hiệu quả mà còn mang lại giá trị học thuật và giải trí cao.

\begin{itemize}

\item \textbf{Tích hợp thuật toán A\*:} Một trong những mục tiêu quan trọng nhất của đề tài là tích hợp thuật toán A\* (A-star) vào trò chơi rắn săn mồi, giúp rắn tự động tìm con đường ngắn nhất đến thức ăn. Thuật toán A\* sẽ được triển khai để phân tích bản đồ dạng lưới, xác định các ô có thể di chuyển, và tính toán đường đi tối ưu dựa trên chi phí di chuyển và khoảng cách ước lượng đến mục tiêu. Đặc biệt, thuật toán cần có khả năng cập nhật đường đi khi bản đồ thay đổi, chẳng hạn như khi thức ăn xuất hiện ở vị trí mới hoặc khi các chướng ngại vật di chuyển. Điều này không chỉ giúp rắn di chuyển một cách thông minh mà còn giảm tải cho người chơi, tạo ra một trải nghiệm chơi game mượt mà và thú vị hơn.

\item \textbf{Xử lý chướng ngại vật di động:} Để tăng tính thử thách và tính thực tế của trò chơi, nhóm chúng tôi đặt mục tiêu thiết kế và xử lý các chướng ngại vật di động trên bản đồ. Không giống như các chướng ngại vật tĩnh (như tường), chướng ngại vật di động sẽ thay đổi vị trí theo thời gian, có thể di chuyển ngẫu nhiên hoặc theo một quy tắc nhất định. Sự xuất hiện của các chướng ngại vật này sẽ ảnh hưởng trực tiếp đến đường đi của rắn, yêu cầu thuật toán A\* phải liên tục cập nhật và điều chỉnh đường đi để tránh va chạm.

\item \textbf{Thiết kế nhiều cấp độ khó:} Nhằm mang lại trải nghiệm tuổi thơ cho người chơi, ngoại trừ chế độ dùng thuật toán A\* để rắn tự tìm đường đi. nhóm chúng tôi đã thiết kế một chế độ chơi thủ công với nhiều cấp độ khó tăng dần, từ dễ đến khó, mỗi cấp độ có các đặc điểm riêng biệt để thử thách cả người chơi:

\begin{itemize}

\item \textit{Cấp độ dễ:} Ở cấp độ này, số lượng chướng ngại vật hạn chế, chủ yếu là các tường tĩnh bao quanh rìa bản đồ. Thức ăn sẽ xuất hiện ở các vị trí dễ tiếp cận, gần với vị trí ban đầu của rắn, giúp rắn dễ dàng tìm thấy mà không cần tính toán phức tạp. Tốc độ di chuyển của rắn cũng được thiết lập ở mức chậm, cho phép người chơi có đủ thời gian để phản ứng và điều chỉnh đường đi.

\item \textit{Cấp độ trung bình:} Cấp độ trung bình sẽ bổ sung thêm các chướng ngại vật, bao gồm cả chướng ngại vật tĩnh và di động. Thức ăn sẽ xuất hiện ngẫu nhiên trên bản đồ, không nhất thiết gần vị trí của rắnn. Các chướng ngại vật di động sẽ di chuyển với tốc độ vừa phải, tạo ra những tình huống bất ngờ.

\item \textit{Cấp độ khó:} Ở cấp độ khó, số lượng chướng ngại vật di động tăng lên đáng kể và di chuyển ở tốc độ nhanh hơn. Thức ăn sẽ xuất hiện ở các vị trí khó tiếp cận, chẳng hạn như gần các khu vực đông chướng ngại vật hoặc ở góc xa của bản đồ, buộc rắn phải di chuyển qua nhiều khu vực nguy hiểm.

\item \textit{Cấp độ đặc biệt:} Ở cấp độ này, tốc độ sẽ là vũ khí chết chóc nhất để khiến người chơi phải thua cuộc, một con rắn nhưng sở hữu tốc độ của một con báo, sẽ là một trải nghiệm thú vị để thử thách giới hạn của người chơi.

\end{itemize}

\item \textbf{Giao diện trực quan:} Một mục tiêu quan trọng khác của đề tài là thiết kế một giao diện trực quan, giúp người chơi dễ dàng theo dõi và tương tác với trò chơi. Giao diện sẽ hiển thị rõ ràng các thành phần chính của trò chơi, bao gồm rắn, thức ăn, và chướng ngại vật, với các màu sắc và hình dạng khác nhau để phân biệt. Ngoài ra, giao diện sẽ bao gồm các thông tin bổ sung như điểm số hiện tại và trạng thái của trò chơi (đang chơi, tạm dừng, hoặc game over). Nhóm đã sử dụng thư viện Pygame để xây dựng giao diện, đảm bảo các thành phần được vẽ mượt mà và không gây gián đoạn trong quá trình chơi. Giao diện trực quan không chỉ giúp người chơi dễ dàng nắm bắt tình hình mà còn tăng tính thẩm mỹ của trò chơi, từ đó nâng cao trải nghiệm tổng thể.

\end{itemize}

Bằng cách hoàn thành các mục tiêu trên, đề tài của chúng tôi không chỉ tạo ra một phiên bản nâng cao của trò chơi rắn săn mồi mà còn minh họa rõ ràng cách trí tuệ nhân tạo, đặc biệt là thuật toán A\*, có thể được ứng dụng trong trò chơi điện tử. Hệ thống sẽ cung cấp một nền tảng thực nghiệm để nghiên cứu và phát triển các thuật toán tìm đường, đồng thời mang lại giá trị giải trí và học thuật cho người chơi và các nhà nghiên cứu. Những bài học rút ra từ đề tài này cũng sẽ được chúng tôi áp dụng trong các lĩnh vực khác, chẳng hạn như điều hướng robot, lập kế hoạch logistics, hoặc thiết kế các hệ thống tự động thông minh trong tuonwg lai.

\chapter{Cơ Sở Lý Thuyết}

\section{Thuật toán A\*}

Thuật toán A\* (A-star) là một trong những thuật toán tìm đường tối ưu và phổ biến nhất trong lĩnh vực trí tuệ nhân tạo, được sử dụng rộng rãi trong nhiều ứng dụng thực tế, từ trò chơi điện tử, điều hướng robot, đến các hệ thống định vị toàn cầu (GPS). Được giới thiệu lần đầu tiên vào năm 1968 bởi Peter Hart, Nils Nilsson và Bertram Raphael, thuật toán A\* kết hợp các ưu điểm của thuật toán Dijkstra (đảm bảo tìm đường ngắn nhất) và tìm kiếm tham lam (tối ưu hóa tốc độ bằng cách sử dụng heuristic).

\subsection{Giới thiệu thuật toán A\*}

Thuật toán A\* là một thuật toán tìm kiếm heuristic, được thiết kế để tìm đường ngắn nhất giữa hai điểm trong một không gian có cấu trúc dạng đồ thị hoặc lưới. Không giống như các thuật toán tìm kiếm truyền thống như tìm kiếm theo chiều rộng (Breadth-First Search) hoặc tìm kiếm theo chiều sâu (Depth-First Search), A\* sử dụng một chiến lược thông minh để ưu tiên các hướng đi có khả năng dẫn đến mục tiêu cao hơn, từ đó giảm thời gian tính toán và tăng hiệu quả tìm kiếm. Điều này làm cho A\* trở thành một lựa chọn lý tưởng cho các bài toán yêu cầu tìm đường tối ưu trong các không gian lớn.

\subsection{Công thức thuật toán A\*}

Thuật toán A\* hoạt động dựa trên một hàm đánh giá $f(n)$, được sử dụng để xác định độ ưu tiên của từng nút (ô) trong quá trình tìm kiếm. Hàm $f(n)$ được tính toán dựa trên hai thành phần chính: chi phí thực tế từ điểm xuất phát đến nút hiện tại và chi phí ước lượng từ nút hiện tại đến điểm đích. Công thức của A\* được biểu diễn như sau:

\begin{equation}

f(n) = g(n) + h(n)

\end{equation}

Trong đó:

\begin{itemize}

\item $g(n)$: Là chi phí thực tế từ điểm xuất phát đến nút hiện tại $n$. Trong bối cảnh trò chơi rắn săn mồi, $g(n)$ thường được tính bằng số bước di chuyển từ vị trí ban đầu của rắn đến ô hiện tại. Mỗi bước di chuyển (lên, xuống, trái, phải) thường có chi phí là 1, trừ khi có các yếu tố khác như địa hình đặc biệt làm tăng chi phí. $g(n)$ đảm bảo rằng thuật toán luôn xem xét chi phí thực tế đã đi qua, từ đó tránh các đường đi không hiệu quả.

\item $h(n)$: Là chi phí ước lượng từ nút hiện tại $n$ đến điểm đích, được gọi là hàm heuristic. Hàm $h(n)$ không tính toán chính xác khoảng cách mà chỉ đưa ra một dự đoán dựa trên các giả định hợp lý. Trong trò chơi rắn săn mồi, $h(n)$ thường được tính bằng khoảng cách Manhattan hoặc khoảng cách Euclidean từ ô hiện tại đến ô chứa thức ăn. Hàm heuristic giúp thuật toán ưu tiên các hướng đi có khả năng dẫn đến mục tiêu cao hơn, từ đó tăng tốc độ tìm kiếm mà vẫn đảm bảo tính tối ưu.

\end{itemize}

Sự kết hợp giữa $g(n)$ và $h(n)$ cho phép thuật toán A\* cân bằng giữa việc tìm đường ngắn nhất (nhờ $g(n)$) và tăng tốc độ tìm kiếm (nhờ $h(n)$). Để đảm bảo A\* luôn tìm được đường đi tối ưu, hàm heuristic $h(n)$ phải thỏa mãn tính chất \textit{monotonic} (hay còn gọi là \textit{consistent}), nghĩa là giá trị $h(n)$ không được phóng đại quá mức so với chi phí thực tế. Nếu $h(n)$ được thiết kế không hợp lý, thuật toán có thể bỏ qua các đường đi tối ưu và dẫn đến kết quả không chính xác.

\subsection{Hàm heuristic trong thuật toán A\*}

Hàm heuristic $h(n)$ là một thành phần quan trọng của thuật toán A\*, đóng vai trò dự đoán khoảng cách từ một nút hiện tại đến điểm đích, từ đó giúp thuật toán ưu tiên các hướng đi tiềm năng. . Dưới đây là các loại heuristic chính được áp dụng trong thuật toán A\*, cùng với công thức và ứng dụng cụ thể:

\begin{itemize}

\item \textbf{Khoảng cách Manhattan:} Đây là loại heuristic phổ biến nhất khi làm việc với các bản đồ dạng lưới, nơi các chuyển động chỉ được phép theo hướng ngang hoặc dọc (không có chuyển động chéo). Khoảng cách Manhattan được tính bằng tổng chênh lệch tuyệt đối giữa tọa độ $x$ và $y$ của nút hiện tại và điểm đích. Công thức của khoảng cách Manhattan là:

\begin{equation}

h(n) = |x\_{goal} - x\_{current}| + |y\_{goal} - y\_{current}|

\end{equation}

\item \textbf{Khoảng cách Euclidean:} Khoảng cách Euclidean được sử dụng khi các chuyển động có thể diễn ra theo mọi hướng, không bị giới hạn bởi lưới. Đây là khoảng cách đường thẳng giữa hai điểm, được tính bằng công thức:

\begin{equation}

h(n) = \sqrt{(x\_{goal} - x\_{current})^2 + (y\_{goal} - y\_{current})^2}

\end{equation}

Khoảng cách Euclidean thường được sử dụng trong các bài toán tìm đường trong không gian liên tục, chẳng hạn như điều hướng robot trong môi trường thực tế.

\item \textbf{Khoảng cách Chebyshev:} Khoảng cách Chebyshev được sử dụng khi các chuyển động có thể bao gồm cả hướng chéo, với chi phí của chuyển động chéo bằng chi phí của chuyển động ngang/dọc. Công thức của khoảng cách Chebyshev là:

\begin{equation}

h(n) = \max(|x\_{goal} - x\_{current}|, |y\_{goal} - y\_{current}|)

\end{equation}

Khoảng cách Chebyshev thường được áp dụng trong các trò chơi hoặc bài toán tìm đường mà nhân vật có thể di chuyển theo tám hướng (bao gồm cả hướng chéo).

\end{itemize}

Việc lựa chọn hàm heuristic phù hợp là yếu tố then chốt để đảm bảo hiệu suất của thuật toán A\*. Trong đề tài này, nhóm đã chọn khoảng cách Manhattan làm hàm heuristic chính, vì nó phù hợp với cấu trúc lưới của bản đồ và các quy tắc di chuyển của rắn. Tuy nhiên, nhóm chúng tôi cũng đã thử nghiệm các hàm heuristic khác như khoảng cách Euclidean để so sánh hiệu suất, từ đó rút ra những bài học quý giá về cách tối ưu hóa thuật toán trong các tình huống khác nhau.

\subsection{Cách hoạt động của thuật toán A\*}

Thuật toán A\* hoạt động dựa trên một hàng đợi ưu tiên (priority queue), trong đó các nút được sắp xếp theo giá trị của hàm đánh giá $f(n)$. Thuật toán sẽ liên tục mở rộng các nút có giá trị $f(n)$ nhỏ nhất, cho đến khi tìm được đường đi đến đích hoặc xác định rằng không có đường đi nào tồn tại. Dưới đây là các bước chi tiết của thuật toán A\*, được áp dụng trong trò chơi rắn săn mồi của nhóm:

\begin{enumerate}

\item \textbf{Đưa điểm bắt đầu vào danh sách Open List:} Thuật toán bắt đầu bằng cách đưa vị trí hiện tại của rắn (đầu rắn) vào danh sách Open List, là danh sách chứa các nút cần được xem xét. Mỗi nút trong Open List sẽ được gán giá trị $f(n) = g(n) + h(n)$, trong đó $g(n)$ ban đầu là 0 (vì đây là điểm xuất phát) và $h(n)$ được tính dựa trên khoảng cách từ đầu rắn đến thức ăn (thường là khoảng cách Manhattan). Ngoài ra, thuật toán cũng duy trì một danh sách Closed List để lưu các nút đã được xem xét, ban đầu danh sách này rỗng.

\item \textbf{Lặp lại quá trình duyệt qua các điểm lân cận, tính toán $g(n), h(n), f(n)$ và cập nhật danh sách Open List:} Trong mỗi vòng lặp, thuật toán sẽ lấy nút có giá trị $f(n)$ nhỏ nhất từ Open List (gọi là nút hiện tại), sau đó xem xét các nút lân cận của nó (các ô liền kề theo hướng lên, xuống, trái, phải). Đối với mỗi nút lân cận, thuật toán sẽ:

\begin{itemize}

\item Kiểm tra xem nút lân cận có hợp lệ hay không (tức là không phải là chướng ngại vật, không nằm ngoài bản đồ, và không thuộc thân rắn).

\item Tính $g(n)$ mới cho nút lân cận, bằng cách cộng chi phí di chuyển (thường là 1) vào $g(n)$ của nút hiện tại.

\item Tính $h(n)$ cho nút lân cận, dựa trên khoảng cách từ nút lân cận đến thức ăn.

\item Tính $f(n) = g(n) + h(n)$ và cập nhật thông tin của nút lân cận trong Open List. Nếu nút lân cận đã có trong Open List và đường đi mới có $g(n)$ nhỏ hơn, thuật toán sẽ cập nhật lại đường đi. Nếu nút lân cận chưa có trong Open List, nó sẽ được thêm vào.

\end{itemize}

Sau khi xem xét xong các nút lân cận, nút hiện tại sẽ được chuyển từ Open List sang Closed List, đánh dấu rằng nó đã được xử lý.

\item \textbf{Nếu đến được điểm đích, thuật toán kết thúc:} Quá trình lặp lại sẽ tiếp tục cho đến khi nút hiện tại là điểm đích (ô chứa thức ăn). Khi đó, thuật toán sẽ truy ngược từ điểm đích về điểm xuất phát thông qua thông tin cha (parent) của các nút, từ đó xây dựng đường đi tối ưu. Đường đi này sẽ được sử dụng để điều khiển rắn di chuyển từng bước đến thức ăn. Nếu Open List trở nên rỗng mà không tìm được điểm đích, thuật toán sẽ kết luận rằng không có đường đi nào tồn tại, và rắn sẽ cần tìm một chiến lược thay thế.

\end{enumerate}

Trong trò chơi rắn săn mồi của nhóm, thuật toán A\* được gọi lại mỗi khi bản đồ thay đổi (chẳng hạn như khi chướng ngại vật di động thay đổi vị trí hoặc khi rắn ăn thức ăn và thức ăn mới xuất hiện). Điều này đảm bảo rằng rắn luôn di chuyển theo đường đi tối ưu, ngay cả trong các tình huống phức tạp.

\section{Xử lý chướng ngại vật di động}

Chướng ngại vật di động là một trong những yếu tố quan trọng làm tăng độ khó và tính thực tế của trò chơi rắn săn mồi, đồng thời đặt ra những thách thức mới cho thuật toán tìm đường. Không giống như các chướng ngại vật tĩnh (như tường), chướng ngại vật di động có thể thay đổi vị trí theo thời gian, tạo ra các tình huống bất ngờ và yêu cầu hệ thống phải thích nghi một cách linh hoạt. T

\subsection{Tác động của chướng ngại vật di động}

Sự xuất hiện của chướng ngại vật di động trong trò chơi rắn săn mồi mang lại nhiều thách thức cho cả người chơi và AI, làm tăng độ phức tạp của bài toán tìm đường. Dưới đây là một số tác động chính của chướng ngại vật di động đối với hệ thống:

\begin{itemize}

\item \textbf{Có thể làm tắc nghẽn đường đi, khiến rắn bị mắc kẹt:} Chướng ngại vật di động có thể bất ngờ xuất hiện trên đường đi của rắn, làm tắc nghẽn con đường mà rắn đang di chuyển. Ví dụ, nếu rắn đang đi theo một đường đi tối ưu được tính toán bởi thuật toán A\*, và một chướng ngại vật di động đột nhiên di chuyển vào ô tiếp theo trên đường đi, rắn sẽ không thể tiếp tục di chuyển theo hướng đó. Nếu không có cơ chế xử lý kịp thời, rắn có thể bị mắc kẹt trong một khu vực hẹp, không thể tìm đường thoát ra, dẫn đến va chạm và kết thúc trò chơi.

\item \textbf{Thuật toán A\* cần liên tục cập nhật lại đường đi khi bản đồ thay đổi:} Do chướng ngại vật di động thay đổi vị trí theo thời gian, bản đồ của trò chơi không còn là một môi trường tĩnh mà trở thành một môi trường động. Điều này yêu cầu thuật toán A\* phải liên tục cập nhật đường đi để phản ánh những thay đổi trên bản đồ. Ví dụ, nếu một chướng ngại vật di chuyển và chặn đường đi hiện tại của rắn, thuật toán cần tính toán lại một đường đi mới để rắn có thể tiếp tục di chuyển đến thức ăn. Việc cập nhật liên tục này không chỉ làm tăng tải tính toán mà còn đòi hỏi hệ thống phải hoạt động nhanh chóng để đảm bảo trò chơi không bị gián đoạn. Nếu thuật toán không được tối ưu hóa, thời gian tính toán lại đường đi có thể làm chậm tốc độ khung hình của trò chơi, ảnh hưởng đến trải nghiệm người chơi.

\end{itemize}

Những tác động trên cho thấy rằng việc xử lý chướng ngại vật di động không chỉ là một bài toán kỹ thuật mà còn là một yếu tố quan trọng để nâng cao tính thử thách và tính thực tế của trò chơi.

\subsection{Giải pháp xử lý chướng ngại vật di động}

Để xử lý các chướng ngại vật di động và đảm bảo rắn có thể di chuyển hiệu quả trong một môi trường thay đổi liên tục, nhóm đã đề xuất và triển khai một số giải pháp sau:

\begin{itemize}

\item \textbf{Thuật toán A\* cập nhật động:} Giải pháp chính mà nhóm áp dụng là sử dụng thuật toán A\* với cơ chế cập nhật động. Cụ thể, mỗi khi bản đồ thay đổi (chẳng hạn như khi chướng ngại vật di động thay đổi vị trí hoặc khi rắn ăn thức ăn và thức ăn mới xuất hiện), thuật toán A\* sẽ được gọi lại để tính toán một đường đi mới từ vị trí hiện tại của rắn đến thức ăn. Để giảm tải tính toán, nhóm đã tối ưu hóa thuật toán bằng cách chỉ cập nhật đường đi sau một khoảng thời gian nhất định (ví dụ: mỗi 10 khung hình), thay vì cập nhật liên tục sau mỗi khung hình. Điều này giúp cân bằng giữa hiệu suất và tính chính xác, đảm bảo rắn có thể phản ứng kịp thời với các thay đổi mà không làm chậm trò chơi.

\item \textbf{Thuật toán D\* Lite:} Ngoài việc sử dụng A\* với cơ chế cập nhật động, nhóm cũng đã nghiên cứu và đề xuất thuật toán D\* Lite như một giải pháp thay thế để xử lý các môi trường động hiệu quả hơn. D\* Lite là một phiên bản nâng cao của thuật toán A\*, được thiết kế đặc biệt cho các bài toán tìm đường trong môi trường thay đổi liên tục. Không giống như A\*, vốn phải tính toán lại toàn bộ đường đi mỗi khi bản đồ thay đổi, D\* Lite có khả năng tái sử dụng các thông tin từ lần tính toán trước đó, từ đó giảm đáng kể thời gian tính toán. Trong bối cảnh trò chơi rắn săn mồi, D\* Lite có thể được áp dụng để cập nhật đường đi của rắn một cách nhanh chóng khi chướng ngại vật di động thay đổi vị trí, đặc biệt ở các cấp độ khó với bản đồ lớn và nhiều chướng ngại vật. Mặc dù nhóm chưa triển khai D\* Lite trong phiên bản hiện tại của trò chơi, đây là một hướng phát triển tiềm năng để cải thiện hiệu suất trong tương lai.

\item \textbf{Cảm biến tránh va chạm:} Một giải pháp bổ sung mà nhóm đã áp dụng là tích hợp một cơ chế "cảm biến tránh va chạm" cho rắn, giúp rắn phản ứng ngay lập tức khi phát hiện chướng ngại vật di động trên đường đi. Cụ thể, trước khi rắn di chuyển đến ô tiếp theo trên đường đi được tính toán bởi A\*, hệ thống sẽ kiểm tra xem ô đó có bị chướng ngại vật chiếm giữ hay không. Nếu ô tiếp theo có chướng ngại vật, rắn sẽ tạm dừng hoặc chuyển hướng ngay lập tức, thay vì tiếp tục di chuyển và gây va chạm. Cơ chế này hoạt động như một lớp bảo vệ bổ sung, giúp rắn tránh các tình huống nguy hiểm trong khi thuật toán A\* đang tính toán lại đường đi mới. Trong trò chơi của nhóm, cảm biến tránh va chạm đã được triển khai để xử lý các tình huống khẩn cấp, chẳng hạn như khi một chướng ngại vật di động bất ngờ xuất hiện ngay trước đầu rắn, từ đó tăng khả năng sống sót của rắn trong các môi trường phức tạp.

\end{itemize}

Những giải pháp trên không chỉ giúp rắn thích nghi với các chướng ngại vật di động mà còn minh họa rõ ràng cách AI có thể được sử dụng để xử lý các bài toán trong môi trường động. Việc kết hợp giữa thuật toán A\* cập nhật động, nghiên cứu D\* Lite, và cảm biến tránh va chạm đã mang lại một hệ thống linh hoạt và hiệu quả, đảm bảo rắn có thể di chuyển an toàn và thông minh trong mọi tình huống.

\section{So sánh A\* với các thuật toán khác}

Để đánh giá hiệu quả của thuật toán A\* trong trò chơi rắn săn mồi, nhóm đã tiến hành so sánh A\* với một số thuật toán tìm đường khác, bao gồm thuật toán Dijkstra và D\* Lite. Việc so sánh này không chỉ giúp làm rõ ưu điểm và nhược điểm của A\* mà còn cung cấp một cái nhìn tổng quan về các thuật toán tìm đường, từ đó đưa ra cơ sở để lựa chọn thuật toán phù hợp cho các bài toán cụ thể. Dưới đây là bảng so sánh chi tiết giữa A\*, Dijkstra, và D\* Lite, dựa trên các tiêu chí như độ phức tạp, ưu điểm, và nhược điểm:

\begin{table}[h]

\centering

\begin{tabular}{|c|c|c|c|}

\hline

\textbf{Thuật toán} & \textbf{Độ phức tạp} & \textbf{Ưu điểm} & \textbf{Nhược điểm} \\

\hline

Dijkstra & $O(n^2)$ & Luôn tìm đường tối ưu, không cần heuristic & Chạy chậm nếu không tối ưu, không hiệu quả trong môi trường lớn \\

\hline

A\* & $O(n \log n)$ & Hiệu quả, tìm đường nhanh, cân bằng giữa tốc độ và độ chính xác & Cần chọn heuristic tốt, có thể chậm trong môi trường động \\

\hline

D\* Lite & $O(n \log n)$ & Tốt cho bản đồ động, tái sử dụng thông tin từ lần tính toán trước & Cần nhiều tài nguyên tính toán, phức tạp hơn A\* \\

\hline

\end{tabular}

\caption{So sánh các thuật toán tìm đường}

\label{tab:comparison}

\end{table}

\textbf{Phân tích chi tiết:}

\begin{itemize}

\item \textbf{Thuật toán Dijkstra:} Thuật toán Dijkstra là một thuật toán tìm đường cổ điển, được thiết kế để tìm đường ngắn nhất từ một điểm xuất phát đến tất cả các điểm khác trong đồ thị. Điểm mạnh của Dijkstra là nó luôn đảm bảo tìm được đường đi tối ưu, bất kể cấu trúc của đồ thị. Tuy nhiên, do không sử dụng heuristic, Dijkstra phải xem xét tất cả các nút trong đồ thị, dẫn đến độ phức tạp thời gian cao ($O(n^2)$ trong triển khai cơ bản, hoặc $O(n \log n)$ nếu sử dụng hàng đợi ưu tiên). Điều này làm cho Dijkstra không hiệu quả trong các bài toán yêu cầu tốc độ cao, chẳng hạn như trò chơi rắn săn mồi, nơi bản đồ có thể thay đổi liên tục và cần tính toán lại đường đi thường xuyên. Trong bối cảnh của đề tài, Dijkstra có thể được sử dụng để kiểm tra tính đúng đắn của đường đi, nhưng không phù hợp để triển khai trong hệ thống thời gian thực.

\item \textbf{Thuật toán A\*:} Thuật toán A\* cải thiện hiệu suất so với Dijkstra bằng cách sử dụng hàm heuristic để ưu tiên các hướng đi tiềm năng, từ đó giảm số lượng nút cần xem xét. Với độ phức tạp thời gian là $O(n \log n)$ khi sử dụng hàng đợi ưu tiên, A\* nhanh hơn đáng kể so với Dijkstra, đặc biệt trong các bài toán tìm đường trên lưới như trò chơi rắn săn mồi. Điểm mạnh của A\* là khả năng cân bằng giữa tốc độ và độ chính xác, giúp rắn tìm được đường đi tối ưu một cách nhanh chóng. Tuy nhiên, A\* cũng có nhược điểm: hiệu suất của nó phụ thuộc vào việc lựa chọn hàm heuristic. Nếu heuristic không được thiết kế tốt (ví dụ: phóng đại chi phí thực tế), A\* có thể bỏ qua các đường đi tối ưu. Ngoài ra, trong môi trường động, A\* cần tính toán lại toàn bộ đường đi mỗi khi bản đồ thay đổi, điều này có thể làm tăng tải tính toán ở các cấp độ khó.

\item \textbf{Thuật toán D\* Lite:} D\* Lite là một phiên bản nâng cao của A\*, được thiết kế đặc biệt cho các bài toán tìm đường trong môi trường động. Với độ phức tạp thời gian tương đương A\* ($O(n \log n)$), D\* Lite có ưu điểm là khả năng tái sử dụng thông tin từ lần tính toán trước đó, từ đó giảm thời gian tính toán khi bản đồ thay đổi. Điều này làm cho D\* Lite trở thành một lựa chọn lý tưởng cho trò chơi rắn săn mồi ở các cấp độ khó, nơi chướng ngại vật di động thay đổi liên tục. Tuy nhiên, D\* Lite phức tạp hơn A\* về mặt triển khai, và yêu cầu nhiều tài nguyên tính toán hơn để quản lý các thông tin tái sử dụng. Trong đề tài này, nhóm đã chọn A\* làm thuật toán chính do tính đơn giản và hiệu quả của nó, nhưng D\* Lite là một hướng phát triển tiềm năng để cải thiện hiệu suất trong tương lai.

\end{itemize}

Việc so sánh trên cho thấy rằng thuật toán A\* là một lựa chọn phù hợp cho trò chơi rắn săn mồi, nhờ vào sự cân bằng giữa tốc độ và độ chính xác. Tuy nhiên, trong các môi trường động phức tạp, việc kết hợp A\* với các thuật toán như D\* Lite có thể mang lại hiệu quả cao hơn, đặc biệt khi cần xử lý các thay đổi liên tục trên bản đồ.

\chapter{Phân Tích và Thiết Kế Hệ Thống}

\section{Kiến trúc tổng thể}

Hệ thống trò chơi rắn săn mồi tích hợp trí tuệ nhân tạo được thiết kế theo kiến trúc mô-đun hóa, nhằm đảm bảo tính linh hoạt, dễ bảo trì, và khả năng mở rộng trong tương lai. Kiến trúc mô-đun hóa cho phép các thành phần của hệ thống hoạt động độc lập nhưng vẫn phối hợp chặt chẽ với nhau, từ đó tạo ra một hệ thống thống nhất và hiệu quả.

\subsection{Bản đồ (định dạng lưới - grid-based)}

Bản đồ là một trong những thành phần cốt lõi của hệ thống, đóng vai trò là môi trường mà rắn, thức ăn, và chướng ngại vật tương tác với nhau. Để phù hợp với lối chơi của trò chơi rắn săn mồi và thuật toán A\*, bản đồ được thiết kế dưới dạng một lưới ô vuông (grid-based), trong đó mỗi ô đại diện cho một vị trí có thể chứa các thành phần của trò chơi. Việc sử dụng định dạng lưới không chỉ đơn giản hóa việc triển khai thuật toán tìm đường mà còn giúp hệ thống dễ dàng quản lý các thành phần và xử lý các sự kiện trong trò chơi. Dưới đây là các thành phần chính được biểu diễn trên bản đồ:

\begin{itemize}

\item \textbf{Rắn:} Rắn là nhân vật chính của trò chơi, được biểu diễn dưới dạng một chuỗi các ô vuông liên tiếp trên lưới. Rắn bao gồm đầu (ô đầu tiên) và cơ thể (các ô còn lại), với đầu rắn là ô dẫn hướng di chuyển. Mỗi khi rắn di chuyển, đầu rắn sẽ tiến lên một ô theo hướng hiện tại, và các ô của cơ thể sẽ di chuyển theo sau, tạo thành một chuỗi liên tục. Vị trí của rắn được lưu trữ dưới dạng một danh sách các tọa độ $(x, y)$, trong đó mỗi tọa độ tương ứng với một ô trên lưới. Rắn có thể di chuyển theo bốn hướng (lên, xuống, trái, phải).

\item \textbf{Thức ăn:} Thức ăn là mục tiêu mà rắn cần di chuyển đến để tăng chiều dài và điểm số. Thức ăn được biểu diễn dưới dạng một ô vuông trên lưới, với vị trí được sinh ra ngẫu nhiên sau mỗi lần rắn ăn. Để đảm bảo tính công bằng, vị trí của thức ăn phải không trùng với rắn, chướng ngại vật, hoặc các ô không thể tiếp cận. Khi rắn di chuyển đến ô chứa thức ăn, hệ thống sẽ sinh ra một thức ăn mới ở vị trí khác, đồng thời tăng chiều dài của rắn và cộng điểm cho người chơi.

\item \textbf{Chướng ngại vật:} Chướng ngại vật là các ô trên lưới mà rắn không thể di chuyển qua, bao gồm cả chướng ngại vật tĩnh và chướng ngại vật di động. Chướng ngại vật tĩnh thường là các bức tường bao quanh rìa bản đồ hoặc các ô cố định được đặt ở các vị trí ngẫu nhiên, nhằm giới hạn không gian di chuyển của rắn. Chướng ngại vật di động, mặt khác, có thể thay đổi vị trí theo thời gian, di chuyển ngẫu nhiên hoặc theo một quy tắc nhất định.

\end{itemize}

Bản đồ dạng lưới không chỉ giúp đơn giản hóa việc quản lý các thành phần của trò chơi mà còn tạo điều kiện thuận lợi cho thuật toán A\* hoạt động hiệu quả. Mỗi ô trên lưới có thể được xem như một nút trong đồ thị, và các ô lân cận được kết nối với nhau theo bốn hướng (lên, xuống, trái, phải). Điều này cho phép A\* dễ dàng tính toán đường đi tối ưu từ đầu rắn đến thức ăn, đồng thời xử lý các thay đổi trên bản đồ một cách nhanh chóng và hiệu quả.

\subsection{Mô-đun tìm đường}

Mô-đun tìm đường là trái tim của hệ thống, chịu trách nhiệm điều khiển rắn di chuyển một cách thông minh từ vị trí hiện tại đến thức ăn, đồng thời tránh các chướng ngại vật trên bản đồ. Để đạt được mục tiêu này, nhóm đã sử dụng thuật toán A\* (A-star), một thuật toán tìm đường tối ưu được thiết kế để tìm con đường ngắn nhất giữa hai điểm trong một không gian có cấu trúc dạng đồ thị hoặc lưới.

\textbf{Các thành phần chính của thuật toán A\*:}

\begin{itemize}

\item \textbf{$G(n)$: Chi phí tích lũy từ vị trí bắt đầu đến ô hiện tại.} Trong trò chơi rắn săn mồi, $G(n)$ được tính bằng số bước di chuyển từ vị trí ban đầu của rắn (đầu rắn tại thời điểm bắt đầu tìm đường) đến ô hiện tại. Mỗi bước di chuyển (lên, xuống, trái, phải) có chi phí là 1, vì rắn di chuyển trên một lưới ô vuông đồng nhất. $G(n)$ đảm bảo rằng thuật toán luôn xem xét chi phí thực tế đã đi qua, từ đó tránh các đường đi không hiệu quả. Ví dụ, nếu rắn đã di chuyển 5 bước để đến ô hiện tại, thì $G(n)$ của ô đó sẽ là 5.

\item \textbf{$H(n)$: Chi phí ước lượng từ ô hiện tại đến đích.} $H(n)$ là hàm heuristic, được sử dụng để dự đoán khoảng cách từ ô hiện tại đến ô chứa thức ăn. Trong hệ thống của nhóm, $H(n)$ được tính bằng khoảng cách Manhattan, vì rắn chỉ di chuyển theo bốn hướng chính (không có chuyển động chéo). Công thức của khoảng cách Manhattan là:

\begin{equation}

H(n) = |x\_{food} - x\_{current}| + |y\_{food} - y\_{current}|

\end{equation}

Ví dụ, nếu ô hiện tại có tọa độ $(3, 5)$ và thức ăn ở ô $(7, 8)$, thì $H(n) = |7 - 3| + |8 - 5| = 4 + 3 = 7$. Hàm heuristic này giúp thuật toán ưu tiên các ô có khả năng dẫn đến thức ăn cao hơn, từ đó tăng tốc độ tìm kiếm mà vẫn đảm bảo tính tối ưu.

\item \textbf{$F(n) = G(n) + H(n)$: Điểm đánh giá tổng của ô đang xét.} $F(n)$ là giá trị tổng hợp, được sử dụng để xác định độ ưu tiên của từng ô trong hàng đợi ưu tiên. Ô có giá trị $F(n)$ nhỏ nhất sẽ được chọn để mở rộng trước, vì nó được coi là ô tiềm năng nhất để dẫn đến đường đi tối ưu. Trong trò chơi rắn săn mồi, $F(n)$ giúp thuật toán A\* cân bằng giữa chi phí đã đi qua ($G(n)$) và chi phí ước lượng còn lại ($H(n)$), từ đó đảm bảo rắn di chuyển theo con đường ngắn nhất đến thức ăn.

\end{itemize}

Mô-đun tìm đường được thiết kế để hoạt động hiệu quả trong cả môi trường tĩnh và động. Trong môi trường tĩnh (chỉ có chướng ngại vật cố định), A\* chỉ cần tính toán đường đi một lần và rắn có thể di chuyển theo đường đi đó cho đến khi đến được thức ăn hoặc bản đồ thay đổi. Tuy nhiên, trong môi trường động, nơi có các chướng ngại vật di động thay đổi vị trí theo thời gian, mô-đun tìm đường cần có khả năng cập nhật đường đi một cách linh hoạt để đảm bảo rắn không va chạm với chướng ngại vật. Để đạt được điều này, chúgn tôi đã tích hợp một cơ chế cập nhật động vào thuật toán A\*, cho phép thuật toán được gọi lại mỗi khi bản đồ thay đổi.

Ngoài ra, để tăng hiệu suất của mô-đun tìm đường, chúng tôi đã tối ưu hóa thuật toán A\* bằng cách sử dụng các cấu trúc dữ liệu hiệu quả. Cụ thể, hàng đợi ưu tiên (priority queue) được triển khai thông qua thư viện \texttt{heapq} của Python, giúp giảm độ phức tạp thời gian của thuật toán xuống còn $O(n \log n)$, trong đó $n$ là số lượng ô trên bản đồ. Hàng đợi ưu tiên đảm bảo rằng ô có giá trị $F(n)$ nhỏ nhất luôn được chọn để mở rộng trước, từ đó tăng tốc độ tìm kiếm mà vẫn đảm bảo tính tối ưu của đường đi.

\subsection{Cơ chế cập nhật đường đi}

Cơ chế cập nhật đường đi là một thành phần quan trọng của hệ thống, đảm bảo rắn có thể thích nghi với các thay đổi trên bản đồ, đặc biệt khi có chướng ngại vật di động hoặc khi thức ăn xuất hiện ở vị trí mới. Trong một môi trường động, việc duy trì một đường đi cố định là không khả thi, vì các chướng ngại vật có thể bất ngờ xuất hiện trên đường đi của rắn, làm tắc nghẽn hoặc gây va chạm. Do đó, hệ thống cần có khả năng tính toán lại đường đi một cách nhanh chóng và hiệu quả, từ đó giúp rắn tiếp tục di chuyển đến thức ăn mà không bị gián đoạn.

\begin{itemize}

\item \textbf{Cập nhật đường đi khi bản đồ thay đổi:} Mỗi khi bản đồ thay đổi, chẳng hạn như khi chướng ngại vật di động thay đổi vị trí hoặc khi rắn ăn thức ăn và thức ăn mới xuất hiện, hệ thống sẽ gọi lại thuật toán A\* để tính toán một đường đi mới từ vị trí hiện tại của rắn đến thức ăn. Để giảm tải tính toán, chúgn tôi đã thiết kế cơ chế cập nhật để chỉ thực hiện sau một khoảng thời gian nhất định thay vì cập nhật liên tục sau mỗi khung hình. Điều này giúp cân bằng giữa hiệu suất và tính chính xác, đảm bảo rắn có thể phản ứng kịp thời với các thay đổi mà không làm chậm trò chơi. Trong quá trình cập nhật, thuật toán A\* sẽ xem xét trạng thái mới của bản đồ, bao gồm vị trí của chướng ngại vật, thân rắn, và thức ăn, từ đó xây dựng một đường đi tối ưu dựa trên các thông tin này.

\item \textbf{Xử lý tình huống không có đường đi tối ưu:} Trong một số trường hợp, đặc biệt ở các cấp độ khó với nhiều chướng ngại vật di động, thuật toán A\* có thể không tìm được đường đi tối ưu từ vị trí hiện tại của rắn đến thức ăn. Nếu chướng ngại vật di động di chuyển ra khỏi vị trí chặn đường, thuật toán A\* sẽ được gọi lại để tính toán một đường đi mới. Cơ chế này giúp rắn tránh bị mắc kẹt vĩnh viễn, đồng thời tăng tính linh hoạt của hệ thống trong các tình huống phức tạp.

\item \textbf{Đảm bảo rắn không đi vào ngõ cụt hoặc mắc kẹt:} Một trong những thách thức lớn khi xử lý chướng ngại vật di động là nguy cơ rắn đi vào ngõ cụt hoặc bị mắc kẹt trong các khu vực hẹp. Để giải quyết vấn đề này, hệ thống được thiết kế để kết hợp thuật toán A\* với một cơ chế "cảm biến tránh va chạm". Trước khi rắn di chuyển đến ô tiếp theo trên đường đi, hệ thống sẽ kiểm tra xem ô đó có bị chướng ngại vật chiếm giữ hay không. Nếu ô tiếp theo có chướng ngại vật, rắn sẽ tạm dừng hoặc chuyển hướng ngay lập tức, thay vì tiếp tục di chuyển và gây va chạm. Ngoài ra, thuật toán A\* được điều chỉnh để ưu tiên các đường đi "an toàn", tức là các đường đi có ít nguy cơ bị chặn bởi chướng ngại vật di động.

\end{itemize}

Cơ chế cập nhật đường đi không chỉ đảm bảo rắn có thể thích nghi với các thay đổi trên bản đồ mà còn minh họa rõ ràng cách trí tuệ nhân tạo có thể được sử dụng để xử lý các bài toán trong môi trường động. Việc kết hợp giữa thuật toán A\* và các cơ chế bổ sung như cảm biến tránh va chạm đã mang lại một hệ thống linh hoạt và hiệu quả, đảm bảo rắn có thể di chuyển an toàn và thông minh trong mọi tình huống. Trong tương lai, cơ chế này có thể được cải tiến bằng cách tích hợp các thuật toán tìm đường động như D\* Lite, giúp giảm thời gian tính toán và tăng hiệu suất trong các môi trường phức tạp hơn.

\section{Thiết kế cấp độ}

Chế độ choi thủ công của game "Rắn săn mồi" được thiết kế với nhiều cấp độ khó khác nhau, nhằm mang lại trải nghiệm đa dạng và phù hợp với nhiều đối tượng người chơi, từ người mới bắt đầu đến những người chơi có kinh nghiệm. Mỗi cấp độ được xây dựng với các đặc điểm riêng biệt, từ kích thước bản đồ, số lượng chướng ngại vật, đến tốc độ di chuyển của rắn và chướng ngại vật, nhằm thử thách người chơi.

\subsection{Cấp độ dễ}

Cấp độ dễ được thiết kế dành cho người chơi mới. Ở cấp độ này, nhóm tập trung vào việc tạo ra một môi trường chơi game thân thiện, với các yếu tố được thiết kế để giảm thiểu độ khó và giúp rắn dễ dàng di chuyển đến thức ăn. Dưới đây là các đặc điểm chính của cấp độ dễ:

\begin{itemize}

\item \textbf{ Ở cấp độ dễ, bản đồ chỉ chứa một số lượng nhỏ chướng ngại vật}: Chủ yếu là các bức tường tĩnh bao quanh rìa bản đồ. Các bức tường này được thiết kế để giới hạn không gian di chuyển của rắn, nhưng không tạo ra các tình huống phức tạp như ngõ cụt hoặc khu vực bị chặn hoàn toàn.

\item \textbf{Thức ăn xuất hiện gần vị trí bắt đầu của rắn:} Để giảm độ khó, thức ăn ở cấp độ dễ sẽ được sinh ra ở các vị trí gần với vị trí ban đầu của rắn. Ví dụ, nếu rắn bắt đầu ở ô $(5, 5)$ (giữa bản đồ 10x10), thức ăn có thể xuất hiện ở các ô như $(6, 5)$, $(5, 6)$, hoặc $(4, 4)$, với khoảng cách Manhattan không quá 3-4 ô. Điều này giúp rắn nhanh chóng tìm thấy thức ăn mà không cần di chuyển qua nhiều khu vực nguy hiểm, đồng thời cho phép người chơi làm quen với cách rắn di chuyển và cơ chế ăn thức ăn.

\item \textbf{Rắn di chuyển với tốc độ chậm, giúp người chơi hoặc AI dễ dàng xác định đường đi:} Tốc độ di chuyển của rắn ở cấp độ dễ được thiết lập ở mức thấp. Với tốc độ chậm, người chơi có thể dễ dàng làm quen với cách điều khiển rắn,. Tốc độ chậm cũng giúp giảm nguy cơ va chạm, đặc biệt khi người chơi chưa quen với cách điều khiển hoặc khi AI cần thời gian để cập nhật đường đi trong các tình huống bất ngờ.

\end{itemize}

\subsection{Cấp độ trung bình}

Cấp độ trung bình được thiết kế để tăng độ khó, mang lại thử thách vừa phải cho cả người chơi, việc bổ sung thêm chướng ngại vật (bao gồm cả chướng ngại vật di động), và điều chỉnh vị trí của thức ăn để yêu cầu rắn phải di chuyển qua nhiều khu vực nguy hiểm hơn. Dưới đây là các đặc điểm chính của cấp độ trung bình:

\begin{itemize}

\item \textbf{Chứa nhiều chướng ngại vật hơn, bao gồm cả chướng ngại vật tĩnh và di chuyển:} Ở cấp độ trung bình, số lượng chướng ngại vật được tăng lên đáng kể, bao gồm cả chướng ngại vật tĩnh và chướng ngại vật di động. Các chướng ngại vật tĩnh có thể bao gồm tường bao quanh rìa bản đồ và một số ô cố định được đặt ở các vị trí ngẫu nhiên. Ngoài ra, cấp độ này giới thiệu chướng ngại vật di động, với số lượng từ 2-5 chướng ngại vật, di chuyển ngẫu nhiên hoặc theo một quy tắc nhất định . Sự xuất hiện của chướng ngại vật di động làm tăng độ khó của trò chơi, vì chúng có thể bất ngờ xuất hiện trên đường đi của rắn.

\item \textbf{Thức ăn xuất hiện ngẫu nhiên, không nhất thiết gần rắn:} Khác với cấp độ dễ, ở cấp độ trung bình, thức ăn sẽ xuất hiện ngẫu nhiên trên toàn bộ bản đồ, không nhất thiết gần vị trí của rắn. Ví dụ, nếu rắn đang ở ô $(3, 3)$, thức ăn có thể xuất hiện ở ô $(18, 15)$, với khoảng cách Manhattan lên đến 27 ô. Điều này yêu cầu rắn phải di chuyển qua một khoảng cách dài hơn, đồng thời đối mặt với nhiều chướng ngại vật trên đường đi. Vị trí ngẫu nhiên của thức ăn làm tăng tính thử thách của trò chơi, vì rắn không thể dựa vào các đường đi ngắn và an toàn như ở cấp độ dễ, mà phải tìm đường qua các khu vực nguy hiểm hơn, nơi có thể có chướng ngại vật di động.

\item \textbf{Người chơi phải tính toán nhanh hơn do cản trở di chuyển không ngừng:} Sự xuất hiện của chướng ngại vật di động ở cấp độ trung bình yêu cầu thuật toán A\* phải tính toán nhanh hơn và cập nhật đường đi thường xuyên hơn.

\end{itemize}

\subsection{Cấp độ khó}

Cấp độ khó được thiết kế để mang lại thử thách tối đa cho cả người chơi trong một môi trường cực kỳ phức tạp và động.

\begin{itemize}

\item \textbf{Chứa nhiều chướng ngại vật di chuyển nhanh:} Ở cấp độ khó, số lượng chướng ngại vật di động được tăng lên đáng kể, với số lượng từ 5-10 chướng ngại vật, di chuyển ở tốc độ nhanh hơn so với cấp độ trung bình . Các chướng ngại vật này có thể di chuyển ngẫu nhiên hoặc theo một quy tắc phức tạp hơn, chẳng hạn như di chuyển theo một hướng cố định cho đến khi gặp vật cản, sau đó đổi hướng ngẫu nhiên. Sự xuất hiện của nhiều chướng ngại vật di động với tốc độ nhanh làm tăng nguy cơ va chạm, tăng độ khó của bài toán tìm đường.

\item \textbf{Thức ăn xuất hiện ở vị trí khó tìm:} Ở cấp độ khó, thức ăn sẽ xuất hiện ở các vị trí khó tiếp cận, chẳng hạn như gần các khu vực đông chướng ngại vật, ở góc xa của bản đồ, hoặc trong các khu vực bị bao vây bởi chướng ngại vật di động. Ví dụ, nếu rắn đang ở ô $(2, 2)$, thức ăn có thể xuất hiện ở ô $(24, 24)$, với khoảng cách Manhattan lên đến 44 ô, và trên đường đi có thể có nhiều chướng ngại vật di động. Vị trí khó của thức ăn buộc rắn phải di chuyển qua nhiều khu vực nguy hiểm, đối mặt với nguy cơ bị mắc kẹt hoặc va chạm với chướng ngại vật.

\item \textbf{Người chơi cần xử lý tình huống linh hoạt:} Với bản đồ lớn, nhiều chướng ngại vật di chuyển nhanh, và thức ăn ở vị trí khó,sự tập trung của người chươi càn tăng cường để giúp rắn phản ứng ngay lập tức với các tình huống khẩn cấp, chẳng hạn như khi một chướng ngại vật di động bất ngờ xuất hiện ngay trước đầu rắn.

\end{itemize}

\chapter{Cài đặt và Thử Nghiệm}

\section{Công nghệ sử dụng}

Hệ thống trò chơi rắn săn mồi tích hợp trí tuệ nhân tạo được phát triển với sự kết hợp của nhiều công nghệ hiện đại, nhằm đảm bảo hiệu suất, tính linh hoạt, và khả năng mở rộng của hệ thống. Việc lựa chọn các công nghệ phù hợp không chỉ giúp nhóm triển khai hệ thống một cách hiệu quả mà còn tạo điều kiện thuận lợi cho việc thử nghiệm và cải tiến trong tương lai. Dưới đây là các công nghệ chính được sử dụng trong quá trình phát triển hệ thống, cùng với vai trò cụ thể của từng công nghệ trong việc hỗ trợ các thành phần của trò chơi.

\begin{itemize}

\item \textbf{Ngôn ngữ lập trình: Python.} Python được chọn làm ngôn ngữ lập trình chính cho dự án nhờ vào tính đơn giản, dễ đọc, và cộng đồng hỗ trợ rộng lớn. Python cung cấp nhiều thư viện mạnh mẽ để phát triển trò chơi và triển khai các thuật toán trí tuệ nhân tạo, giúp nhóm tiết kiệm thời gian và tập trung vào việc phát triển các tính năng cốt lõi của hệ thống. Ngoài ra, Python có hiệu suất tốt trong các ứng dụng không yêu cầu tốc độ quá cao, chẳng hạn như trò chơi rắn săn mồi, nơi các tác vụ tính toán không quá nặng. Việc sử dụng Python cũng giúp chúng tôi dễ dàng thử nghiệm và điều chỉnh thuật toán A\*, từ đó đảm bảo hệ thống hoạt động đúng như mong đợi.

\item \textbf{Thư viện đồ họa: \texttt{Pygame} - hỗ trợ tạo giao diện đồ họa và xử lý sự kiện người chơi.} \texttt{Pygame} là một thư viện mạnh mẽ của Python, được thiết kế đặc biệt để phát triển trò chơi 2D. Chúgn tôi đã sử dụng \texttt{Pygame} để xây dựng giao diện đồ họa của trò chơi, bao gồm việc vẽ rắn, thức ăn, chướng ngại vật, và các thành phần khác trên bản đồ dạng lưới. \texttt{Pygame} cũng hỗ trợ xử lý các sự kiện người chơi, chẳng hạn như nhấn phím để điều khiển rắn, tạm dừng trò chơi, hoặc thoát trò chơi. Ngoài ra, \texttt{Pygame} cung cấp các công cụ để quản lý âm thanh (như âm thanh khi rắn ăn thức ăn hoặc khi trò chơi kết thúc) và điều khiển tốc độ khung hình, giúp trò chơi chạy mượt mà trên các thiết bị khác nhau. Việc sử dụng \texttt{Pygame} đã giúp nhóm nhanh chóng xây dựng một giao diện trực quan và thân thiện với người chơi, từ đó nâng cao trải nghiệm tổng thể của trò chơi.

\item \textbf{Thư viện tính toán: \texttt{NumPy} - hỗ trợ xử lý dữ liệu mảng và tính toán nhanh.} \texttt{NumPy} là một thư viện tính toán khoa học của Python, được sử dụng để xử lý dữ liệu mảng và thực hiện các phép tính toán nhanh. Trong hệ thống của nhóm, \texttt{NumPy} được sử dụng để biểu diễn bản đồ dạng lưới dưới dạng một mảng hai chiều, trong đó mỗi phần tử của mảng đại diện cho trạng thái của một ô trên bản đồ. \texttt{NumPy} giúp tăng tốc độ xử lý các thao tác trên mảng, chẳng hạn như kiểm tra va chạm, cập nhật vị trí của chướng ngại vật di động, hoặc sinh vị trí ngẫu nhiên cho thức ăn. Ngoài ra, \texttt{NumPy} cũng hỗ trợ các phép tính toán số học cần thiết cho thuật toán A\*, chẳng hạn như tính khoảng cách Manhattan hoặc quản lý các giá trị $G(n)$, $H(n)$, và $F(n)$. Việc sử dụng \texttt{NumPy} đã giúp nhóm tối ưu hóa hiệu suất của hệ thống, đặc biệt trong các cấp độ khó với bản đồ lớn và nhiều chướng ngại vật.

\item \textbf{Thư viện cấu trúc dữ liệu: \texttt{heapq} - giúp quản lý hàng đợi ưu tiên trong thuật toán A\*.} \texttt{heapq} là một thư viện của Python, cung cấp các công cụ để triển khai hàng đợi ưu tiên (priority queue) dựa trên cấu trúc dữ liệu heap. Trong thuật toán A\*, hàng đợi ưu tiên được sử dụng để quản lý danh sách Open List, chứa các ô cần được xem xét trong quá trình tìm kiếm. \texttt{heapq} đảm bảo rằng ô có giá trị $F(n)$ nhỏ nhất luôn được truy xuất nhanh chóng, với độ phức tạp thời gian là $O(\log n)$ cho mỗi thao tác thêm hoặc xóa phần tử. Việc sử dụng \texttt{heapq} đã giúp nhóm tối ưu hóa thuật toán A\*, giảm thời gian tính toán và đảm bảo hệ thống có thể xử lý các bản đồ lớn một cách hiệu quả. Ngoài ra, \texttt{heapq} cũng được sử dụng để quản lý các thông tin khác, chẳng hạn như danh sách các chướng ngại vật di động cần được cập nhật, từ đó tăng hiệu suất tổng thể của hệ thống.

\item \textbf{Mô-đun tìm kiếm heuristic: Tối ưu hóa thuật toán A\* để cải thiện hiệu suất tìm đường đi.} Ngoài các thư viện có sẵn, nhóm đã phát triển một mô-đun tìm kiếm heuristic riêng để tối ưu hóa thuật toán A\* cho trò chơi rắn săn mồi. Mô-đun này bao gồm các hàm để tính toán $G(n)$, $H(n)$, và $F(n)$, cũng như các cơ chế để cập nhật đường đi khi bản đồ thay đổi. Để tăng hiệu suất, nhóm đã chọn khoảng cách Manhattan làm hàm heuristic chính, vì nó phù hợp với cấu trúc lưới của bản đồ và các quy tắc di chuyển của rắn. Ngoài ra, mô-đun tìm kiếm heuristic được thiết kế để hỗ trợ các tình huống động, chẳng hạn như khi chướng ngại vật di động thay đổi vị trí, bằng cách tích hợp cơ chế cập nhật động và cảm biến tránh va chạm. Việc phát triển mô-đun này không chỉ giúp tối ưu hóa thuật toán A\* mà còn cung cấp một nền tảng để thử nghiệm và cải tiến các thuật toán tìm đường khác trong tương lai.

\end{itemize}

Sự kết hợp của các công nghệ trên đã mang lại một hệ thống mạnh mẽ và linh hoạt, đáp ứng được các yêu cầu của đề tài. Python và các thư viện như \texttt{Pygame}, \texttt{NumPy}, và \texttt{heapq} đã giúp nhóm nhanh chóng triển khai hệ thống, trong khi mô-đun tìm kiếm heuristic tự phát triển đã đảm bảo thuật toán A\* hoạt động hiệu quả trong các tình huống thực tế. Những công nghệ này không chỉ hỗ trợ việc phát triển trò chơi mà còn mở ra nhiều hướng phát triển tiềm năng trong tương lai, chẳng hạn như tích hợp học máy hoặc cải thiện giao diện đồ họa.

\section{Chi tiết cài đặt}

Hệ thống trò chơi rắn săn mồi được thiết kế theo mô hình hướng đối tượng (Object-Oriented Programming - OOP), nhằm đảm bảo tính mô-đun hóa, dễ bảo trì, và khả năng mở rộng. Mô hình hướng đối tượng cho phép nhóm chia hệ thống thành các lớp (class) riêng biệt, mỗi lớp chịu trách nhiệm quản lý một thành phần cụ thể của trò chơi, từ đó giúp mã nguồn trở nên rõ ràng và dễ quản lý hơn. Dưới đây là chi tiết về các lớp chính của hệ thống, cùng với vai trò và chức năng của từng lớp trong việc triển khai trò chơi.

\subsection{Lớp Snake}

Lớp \texttt{Snake} là lớp cốt lõi của hệ thống, chịu trách nhiệm quản lý trạng thái và hành vi của rắn trong trò chơi. Rắn là nhân vật chính, và mọi tương tác trong trò chơi (như di chuyển, ăn thức ăn, hoặc va chạm) đều liên quan đến rắn. Lớp \texttt{Snake} được thiết kế để đảm bảo rắn có thể di chuyển một cách mượt mà, xử lý các sự kiện va chạm, và tăng chiều dài khi ăn thức ăn. Dưới đây là các chức năng chính của lớp \texttt{Snake}:

\begin{itemize}

\item \textbf{Lưu trữ vị trí đầu và thân rắn dưới dạng danh sách các tọa độ:} Rắn được biểu diễn dưới dạng một danh sách các tọa độ $(x, y)$, trong đó phần tử đầu tiên của danh sách là vị trí của đầu rắn, và các phần tử còn lại là vị trí của thân rắn. Ví dụ, nếu rắn có chiều dài là 3, danh sách tọa độ có thể là $[(5, 5), (5, 4), (5, 3)]$, trong đó $(5, 5)$ là vị trí của đầu rắn. Danh sách này được cập nhật liên tục mỗi khi rắn di chuyển, với đầu rắn di chuyển đến một ô mới và các ô của thân rắn di chuyển theo sau. Việc lưu trữ vị trí dưới dạng danh sách giúp hệ thống dễ dàng kiểm tra va chạm và vẽ rắn lên màn hình.

\item \textbf{Cập nhật vị trí di chuyển dựa trên hướng hiện tại:} Lớp \texttt{Snake} có một thuộc tính lưu trữ hướng di chuyển hiện tại của rắn (lên, xuống, trái, phải), được biểu diễn dưới dạng một vector $(dx, dy)$. Ví dụ, nếu rắn đang di chuyển sang phải, vector hướng sẽ là $(1, 0)$. Mỗi khi rắn di chuyển, vị trí của đầu rắn sẽ được cập nhật bằng cách cộng vector hướng vào tọa độ hiện tại: $(x\_{new}, y\_{new}) = (x\_{current} + dx, y\_{current} + dy)$. Sau đó, các ô của thân rắn sẽ di chuyển theo sau, với ô thứ hai di chuyển đến vị trí của đầu rắn trước đó, ô thứ ba di chuyển đến vị trí của ô thứ hai trước đó, và cứ tiếp tục như vậy. Nếu rắn không ăn thức ăn, ô cuối cùng của thân rắn sẽ bị xóa, giữ nguyên chiều dài của rắn. Việc cập nhật vị trí này được thực hiện trong mỗi khung hình của trò chơi, đảm bảo rắn di chuyển mượt mà trên bản đồ.

\item \textbf{Kiểm tra va chạm với tường, thân rắn hoặc chướng ngại vật:} Lớp \texttt{Snake} có một phương thức để kiểm tra va chạm mỗi khi rắn di chuyển đến một ô mới. Cụ thể, hệ thống sẽ kiểm tra xem ô tiếp theo có nằm ngoài bản đồ (tức là va chạm với tường), có trùng với thân rắn (tức là rắn tự cắn vào mình), hoặc có trùng với chướng ngại vật (tĩnh hoặc động) hay không. Nếu một trong các điều kiện va chạm được thỏa mãn, trò chơi sẽ kết thúc và hiển thị thông báo "Game Over". Trong trường hợp rắn được điều khiển tự động bằng thuật toán A\*, cơ chế kiểm tra va chạm cũng được sử dụng để đảm bảo đường đi được tính toán bởi A\* là an toàn, từ đó tránh các tình huống dẫn đến va chạm không mong muốn.

\item \textbf{Xử lý ăn thức ăn và tăng độ dài:} Khi đầu rắn di chuyển đến ô chứa thức ăn, lớp \texttt{Snake} sẽ kích hoạt sự kiện ăn thức ăn. Cụ thể, chiều dài của rắn sẽ tăng thêm một ô, và danh sách tọa độ của rắn sẽ được mở rộng bằng cách giữ nguyên ô cuối cùng (thay vì xóa nó như khi rắn di chuyển bình thường). Ví dụ, nếu rắn có danh sách tọa độ là $[(5, 5), (5, 4), (5, 3)]$ và ăn thức ăn ở ô $(5, 6)$, danh sách tọa độ mới sẽ là $[(5, 6), (5, 5), (5, 4), (5, 3)]$, với chiều dài tăng từ 3 lên 4. Đồng thời, hệ thống sẽ cộng điểm cho người chơi (ví dụ: 1 điểm cho mỗi lần ăn thức ăn) và sinh ra một thức ăn mới ở vị trí ngẫu nhiên trên bản đồ. Việc xử lý ăn thức ăn không chỉ làm tăng độ khó của trò chơi (do rắn dài hơn, dễ va chạm hơn) mà còn mang lại cảm giác thành tựu cho người chơi.

\end{itemize}

Lớp \texttt{Snake} đóng vai trò trung tâm trong hệ thống, đảm bảo rắn có thể di chuyển, tương tác với các thành phần khác, và phản ứng với các sự kiện trong trò chơi. Việc thiết kế lớp này theo mô hình hướng đối tượng đã giúp nhóm dễ dàng quản lý trạng thái của rắn, đồng thời tạo điều kiện thuận lợi cho việc tích hợp thuật toán A\* để điều khiển rắn tự động.

\subsection{Lớp Food}

Lớp \texttt{Food} chịu trách nhiệm quản lý thức ăn trong trò chơi, bao gồm việc sinh ra thức ăn ở các vị trí ngẫu nhiên và xử lý các sự kiện liên quan đến thức ăn. Thức ăn là mục tiêu chính của rắn, và việc quản lý thức ăn một cách hiệu quả là yếu tố quan trọng để đảm bảo trò chơi diễn ra mượt mà và công bằng. Dưới đây là các chức năng chính của lớp \texttt{Food}:

\begin{itemize}

\item \textbf{Xác định vị trí mới cho thức ăn không trùng với rắn hoặc chướng ngại vật:} Khi trò chơi bắt đầu hoặc sau mỗi lần rắn ăn thức ăn, lớp \texttt{Food} sẽ sinh ra một vị trí mới cho thức ăn trên bản đồ. Vị trí này được chọn ngẫu nhiên, nhưng phải thỏa mãn các điều kiện sau: không trùng với bất kỳ ô nào của rắn, không trùng với chướng ngại vật (tĩnh hoặc động), và nằm trong phạm vi của bản đồ. Để thực hiện điều này, lớp \texttt{Food} sử dụng một thuật toán sinh số ngẫu nhiên (dựa trên thư viện \texttt{random} của Python) và kiểm tra các điều kiện hợp lệ. Nếu vị trí được sinh ra không thỏa mãn, hệ thống sẽ tiếp tục sinh vị trí mới cho đến khi tìm được một ô hợp lệ. Trong các cấp độ khó, vị trí của thức ăn có thể được điều chỉnh để xuất hiện ở các khu vực khó tiếp cận, nhằm tăng độ thử thách của trò chơi.

\item \textbf{Kích hoạt sự kiện khi rắn ăn thức ăn và sinh ra thức ăn mới:} Khi đầu rắn di chuyển đến ô chứa thức ăn, lớp \texttt{Food} sẽ kích hoạt sự kiện ăn thức ăn. Cụ thể, hệ thống sẽ thông báo cho lớp \texttt{Snake} để tăng chiều dài của rắn và cộng điểm cho người chơi. Đồng thời, lớp \texttt{Food} sẽ sinh ra một thức ăn mới ở vị trí ngẫu nhiên trên bản đồ, đảm bảo vị trí mới thỏa mãn các điều kiện hợp lệ như đã đề cập ở trên. Ngoài ra, sự kiện ăn thức ăn cũng có thể kích hoạt các hiệu ứng khác, chẳng hạn như phát âm thanh (sử dụng \texttt{Pygame}) hoặc tăng tốc độ của rắn (ở các cấp độ khó). Việc xử lý sự kiện này không chỉ làm tăng tính tương tác của trò chơi mà còn mang lại cảm giác thành tựu cho người chơi mỗi khi rắn ăn được thức ăn.

\end{itemize}

Lớp \texttt{Food} được thiết kế đơn giản nhưng hiệu quả, đảm bảo thức ăn luôn xuất hiện ở các vị trí hợp lệ và mang lại trải nghiệm chơi game công bằng. Trong tương lai, lớp này có thể được mở rộng để hỗ trợ các loại thức ăn đặc biệt, chẳng hạn như thức ăn tăng tốc độ, thức ăn làm chậm chướng ngại vật, hoặc thức ăn cộng điểm cao hơn, từ đó làm tăng tính đa dạng của trò chơi.

\subsection{Lớp Obstacle}

Lớp \texttt{Obstacle} chịu trách nhiệm quản lý các chướng ngại vật trong trò chơi, bao gồm cả chướng ngại vật tĩnh và chướng ngại vật di động. Chướng ngại vật là yếu tố quan trọng làm tăng độ khó của trò chơi, đồng thời đặt ra những thách thức mới cho thuật toán A\* trong việc tìm đường. Lớp \texttt{Obstacle} được thiết kế để đảm bảo các chướng ngại vật được sinh ra, di chuyển, và tương tác với rắn một cách hợp lý. Dưới đây là các chức năng chính của lớp \texttt{Obstacle}:

\begin{itemize}

\item \textbf{Chướng ngại vật tĩnh: Các bức tường không thay đổi vị trí.} Chướng ngại vật tĩnh bao gồm các bức tường bao quanh rìa bản đồ và các ô cố định được đặt ở các vị trí ngẫu nhiên. Các bức tường được sinh ra tự động khi trò chơi bắt đầu, bao quanh toàn bộ rìa của bản đồ (ví dụ: các ô có tọa độ $x = 0$, $x = width-1$, $y = 0$, hoặc $y = height-1$). Ngoài ra, ở các cấp độ trung bình và khó, một số chướng ngại vật tĩnh khác có thể được sinh ra ngẫu nhiên trên bản đồ, với số lượng tùy thuộc vào cấp độ (ví dụ: 5-10 chướng ngại vật ở cấp độ trung bình). Các chướng ngại vật tĩnh được lưu trữ dưới dạng một danh sách các tọa độ $(x, y)$, và không thay đổi vị trí trong suốt quá trình chơi. Chúng đóng vai trò giới hạn không gian di chuyển của rắn, đồng thời tạo ra các khu vực nguy hiểm hoặc ngõ cụt để tăng độ khó của trò chơi.

\item \textbf{Chướng ngại vật động: Di chuyển theo một quy tắc nhất định, có thể ngẫu nhiên hoặc theo mô hình xác định.} Chướng ngại vật di động là một tính năng quan trọng ở các cấp độ trung bình và khó, làm tăng tính thử thách và tính thực tế của trò chơi. Mỗi chướng ngại vật di động được biểu diễn dưới dạng một cặp $[(x, y), (dx, dy)]$, trong đó $(x, y)$ là vị trí hiện tại và $(dx, dy)$ là vector hướng di chuyển. Khi trò chơi bắt đầu, các chướng ngại vật di động được sinh ra ở các vị trí ngẫu nhiên trên bản đồ, với số lượng tùy thuộc vào cấp độ (ví dụ: 2-5 chướng ngại vật ở cấp độ trung bình, 5-10 chướng ngại vật ở cấp độ khó). Các chướng ngại vật này di chuyển theo một quy tắc nhất định: chúng sẽ di chuyển theo hướng hiện tại cho đến khi gặp vật cản (như tường, chướng ngại vật khác, hoặc thân rắn), sau đó đổi hướng ngẫu nhiên. Tốc độ di chuyển của chướng ngại vật cũng thay đổi theo cấp độ, với tốc độ nhanh hơn ở cấp độ khó (ví dụ: di chuyển mỗi 2-3 khung hình thay vì mỗi 5 khung hình). Việc quản lý chướng ngại vật di động không chỉ làm tăng độ khó của trò chơi mà còn yêu cầu thuật toán A\* phải cập nhật đường đi liên tục để tránh va chạm.

\end{itemize}

Lớp \texttt{Obstacle} đóng vai trò quan trọng trong việc tạo ra một môi trường chơi game động và thử thách. Việc kết hợp giữa chướng ngại vật tĩnh và chướng ngại vật di động đã mang lại một trải nghiệm chơi game phong phú, đồng thời cung cấp một bài kiểm tra thực tế để đánh giá khả năng thích nghi của thuật toán A\* trong các tình huống phức tạp.

\subsection{Mô-đun thuật toán A\*}

Mô-đun thuật toán A\* là thành phần cốt lõi của hệ thống, chịu trách nhiệm tìm đường đi tối ưu từ vị trí hiện tại của rắn đến thức ăn, đồng thời tránh các chướng ngại vật trên bản đồ. Thuật toán A\* được triển khai để đảm bảo rắn có thể di chuyển một cách thông minh, ngay cả trong các môi trường động với nhiều chướng ngại vật di động. Dưới đây là chi tiết về cách thuật toán A\* được triển khai trong hệ thống:

\begin{itemize}

\item \textbf{Sử dụng hàng đợi ưu tiên để mở rộng các ô lân cận:} Thuật toán A\* được triển khai bằng cách sử dụng hàng đợi ưu tiên (priority queue) để quản lý danh sách Open List, chứa các ô cần được xem xét trong quá trình tìm kiếm. Hàng đợi ưu tiên được triển khai thông qua thư viện \texttt{heapq} của Python, đảm bảo rằng ô có giá trị $F(n)$ nhỏ nhất luôn được truy xuất nhanh chóng, với độ phức tạp thời gian là $O(\log n)$ cho mỗi thao tác thêm hoặc xóa phần tử. Trong mỗi vòng lặp, thuật toán sẽ lấy ô có $F(n)$ nhỏ nhất từ Open List, sau đó mở rộng các ô lân cận của nó (các ô liền kề theo hướng lên, xuống, trái, phải). Đối với mỗi ô lân cận, thuật toán sẽ kiểm tra xem ô đó có hợp lệ hay không (tức là không phải là chướng ngại vật, không nằm ngoài bản đồ, và không thuộc thân rắn), sau đó tính toán các giá trị $G(n)$, $H(n)$, và $F(n)$ để cập nhật Open List.

\item \textbf{Tính toán hàm đánh giá $f(n) = g(n) + h(n)$, trong đó:}

\begin{itemize}

\item \textbf{$g(n)$: Chi phí di chuyển từ điểm bắt đầu đến ô hiện tại.} $g(n)$ được tính bằng số bước di chuyển từ vị trí ban đầu của rắn đến ô hiện tại, với mỗi bước di chuyển có chi phí là 1. Ví dụ, nếu rắn đã di chuyển 5 bước để đến ô hiện tại, thì $g(n)$ của ô đó sẽ là 5. $g(n)$ đảm bảo rằng thuật toán luôn xem xét chi phí thực tế đã đi qua, từ đó tránh các đường đi không hiệu quả.

\item \textbf{$h(n)$: Khoảng cách ước lượng từ ô hiện tại đến thức ăn (sử dụng khoảng cách Manhattan hoặc Euclidean).} Trong hệ thống của nhóm, $h(n)$ được tính bằng khoảng cách Manhattan, vì rắn chỉ di chuyển theo bốn hướng chính (không có chuyển động chéo). Công thức của khoảng cách Manhattan là:

\begin{equation}

h(n) = |x\_{food} - x\_{current}| + |y\_{food} - y\_{current}|

\end{equation}

Ví dụ, nếu ô hiện tại có tọa độ $(3, 5)$ và thức ăn ở ô $(7, 8)$, thì $h(n) = |7 - 3| + |8 - 5| = 4 + 3 = 7$. Hàm heuristic này giúp thuật toán ưu tiên các ô có khả năng dẫn đến thức ăn cao hơn, từ đó tăng tốc độ tìm kiếm mà vẫn đảm bảo tính tối ưu.

\end{itemize}

\item \textbf{Khi chướng ngại vật di chuyển, thuật toán sẽ cập nhật lại đường đi phù hợp:} Trong một môi trường động, chướng ngại vật di động có thể thay đổi vị trí theo thời gian, làm ảnh hưởng đến đường đi của rắn. Để xử lý tình huống này, thuật toán A\* được tích hợp với cơ chế cập nhật động, cho phép thuật toán được gọi lại mỗi khi bản đồ thay đổi (ví dụ: mỗi 10 khung hình hoặc khi rắn ăn thức ăn). Khi được gọi lại, thuật toán sẽ tính toán một đường đi mới từ vị trí hiện tại của rắn đến thức ăn, dựa trên trạng thái mới của bản đồ. Ngoài ra, cơ chế cảm biến tránh va chạm cũng được tích hợp để giúp rắn phản ứng ngay lập tức với các tình huống khẩn cấp, chẳng hạn như khi một chướng ngại vật di động bất ngờ xuất hiện trên đường đi. Việc cập nhật đường đi này đảm bảo rắn có thể di chuyển an toàn và hiệu quả, ngay cả trong các môi trường phức tạp với nhiều chướng ngại vật di động.

\end{itemize}

Mô-đun thuật toán A\* không chỉ giúp rắn di chuyển một cách thông minh mà còn minh họa rõ ràng cách trí tuệ nhân tạo có thể được áp dụng trong các hệ thống điều hướng thời gian thực. Việc triển khai A\* trong hệ thống đã mang lại một trải nghiệm chơi game mượt mà và thú vị, đồng thời cung cấp một nền tảng để thử nghiệm và cải tiến các thuật toán tìm đường trong tương lai.

\section{Kết quả thực nghiệm}

Để đánh giá hiệu quả của hệ thống, nhóm đã tiến hành thử nghiệm trên cả 2 chế độ chơi khác nhau, nhằm kiểm tra khả năng hoạt động của thuật toán A\* và tính linh hoạt của hệ thống trong các tình huống thực tế. Các thử nghiệm được thực hiện trên một máy tính có cấu hình trung bình(Lenovo GeekPro 2023 với cấu hình CPU Ryzen 7840Hs, RAM 16GB), với trò chơi chạy ở tốc độ 60 khung hình mỗi giây là hoàn toàn bình thường. Dưới đây là kết quả chi tiết của các thử nghiệm, được chia theo từng cấp độ để làm rõ hiệu suất và khả năng thích nghi của hệ thống.Dưới đây là các kết quả thu được từ thử nghiệm:

\begin{itemize}

\item \textbf{Mức điểm <10}: Thử nghiệm được thực hiện trên một bản đồ cố định, với chỉ có tường bao quanh rìa bản đồ và không có chướng ngại vật di động. Thức ăn được sinh ra ở các vị trí gần rắn. Trong môi trường này, thuật toán A\* có thể dễ dàng tìm đường đi tối ưu mà không cần xử lý các tình huống phức tạp, từ đó giúp nhóm xác nhận rằng hệ thống hoạt động đúng như mong đợi.

\item \textbf{Mức điểm 10<30} Trong khoảng chơi thử nghiệm, rắn đã tìm thấy thức ăn trong tất cả các ván, với số bước di chuyển trung bình là 4-6 bước mỗi lần ăn thức ăn. Do không có chướng ngại vật di động, thuật toán A\* chỉ cần tính toán đường đi một lần cho mỗi lần thức ăn xuất hiện, và không cần cập nhật lại đường đi trong suốt quá trình di chuyển. Điều này giúp giảm tải tính toán, đảm bảo rắn di chuyển mượt mà và không bị gián đoạn.

\item \textbf{Mức điểm >30:} Tại thời điểm này, rắn bắt đầu trở nên khó khăn hơn để tìm thức ăn bởi các chướng ngại vật ngày càng nhiều và ngày càng di chuyển nhanh nên chúng đã trở nên nguy hiểm hơn bao giờ hết, ở mức điểm khoảng >50, rắn đã đâm vào chướng ngại vật và kết thúc trờ chơi.

\end{itemize}

Kết quả thử nghiệm trên cho thấy rằng hệ thống hoạt động ổn định và hiệu quả trong một môi trường đơn giản. Thuật toán A\* đã chứng minh được khả năng tìm đường tối ưu một cách nhanh chóng, đồng thời mang lại một trải nghiệm chơi game mượt mà cho người chơi.

\section{Đánh giá hiệu năng}

Dựa trên các kết quả thử nghiệm trên các cấp độ khác nhau, nhóm đã tiến hành đánh giá hiệu năng của hệ thống, nhằm làm rõ những điểm mạnh và điểm yếu của hệ thống, đồng thời đề xuất các hướng cải tiến trong tương lai. Dưới đây là các điểm nổi bật trong hiệu năng của hệ thống, được tổng hợp từ các thử nghiệm:

\begin{itemize}

\item \textbf{Thuật toán A\* đảm bảo tìm được đường đi tối ưu nhanh chóng:} Trong tất cả các cấp độ, thuật toán A\* đã chứng minh được khả năng tìm đường tối ưu một cách nhanh chóng và hiệu quả. Ở cấp độ dễ, thời gian tìm đường trung bình chỉ là 10ms, trong khi ở cấp độ trung bình và khó, thời gian này tăng lên lần lượt là 20ms và 30ms. Mặc dù thời gian tính toán tăng lên ở các cấp độ khó, nhưng vẫn nằm trong ngưỡng chấp nhận được, đảm bảo trò chơi chạy mượt mà mà không bị gián đoạn. Việc sử dụng hàng đợi ưu tiên và hàm heuristic phù hợp (khoảng cách Manhattan) đã giúp thuật toán A\* hoạt động hiệu quả, ngay cả trong các môi trường phức tạp với bản đồ lớn và nhiều chướng ngại vật.

\item \textbf{Cơ chế cập nhật đường đi giúp rắn phản ứng linh hoạt với chướng ngại vật di động:} Cơ chế cập nhật đường đi, kết hợp với cảm biến tránh va chạm, đã giúp rắn thích nghi một cách linh hoạt với các thay đổi trên bản đồ, đặc biệt ở các cấp độ trung bình và khó. Số lần cập nhật đường đi trung bình là 3-5 lần mỗi ván ở cấp độ trung bình và 7-10 lần mỗi ván ở cấp độ khó, cho thấy hệ thống có khả năng phản ứng nhanh chóng với các tình huống bất ngờ. Cơ chế cảm biến tránh va chạm cũng đã giúp rắn tránh được nhiều tình huống va chạm không mong muốn, từ đó tăng khả năng sống sót trong một môi trường động. Việc kết hợp giữa thuật toán A\* và các cơ chế bổ sung đã mang lại một hệ thống linh hoạt và hiệu quả, đảm bảo rắn có thể di chuyển an toàn và thông minh trong mọi tình huống.

\item \textbf{Mô-đun heuristic giúp giảm thời gian tính toán, đặc biệt ở cấp độ khó:} Việc sử dụng khoảng cách Manhattan làm hàm heuristic, kết hợp với các điều chỉnh để ưu tiên các đường đi an toàn, đã giúp giảm thời gian tính toán của thuật toán A\*, đặc biệt ở cấp độ khó. Bằng cách tránh các khu vực nguy hiểm (như các khu vực gần chướng ngại vật di động), thuật toán A\* có thể giảm số lượng ô cần xem xét, từ đó tăng tốc độ tìm kiếm mà vẫn đảm bảo tính tối ưu của đường đi. Ngoài ra, việc giảm tần suất cập nhật đường đi (chỉ cập nhật mỗi 10 khung hình) cũng đã giúp giảm tải tính toán, đảm bảo trò chơi chạy mượt mà ngay cả trong các môi trường phức tạp.

\end{itemize}

Từ các kết quả thử nghiệm và đánh giá hiệu năng, có thể thấy rằng hệ thống trò chơi rắn săn mồi tích hợp trí tuệ nhân tạo đã hoạt động ổn định và hiệu quả trên các cấp độ khác nhau. Thuật toán A\* đã chứng minh được khả năng tìm đường tối ưu một cách nhanh chóng, trong khi cơ chế cập nhật đường đi và cảm biến tránh va chạm đã giúp rắn thích nghi với các thay đổi trên bản đồ. Hệ thống không chỉ đáp ứng được các mục tiêu đề ra mà còn mang lại một trải nghiệm chơi game thú vị và đầy thử thách. Tuy nhiên, vẫn còn nhiều tiềm năng để cải thiện, chẳng hạn như tối ưu hóa thuật toán để giảm thời gian tính toán ở cấp độ khó, hoặc tích hợp các thuật toán tìm đường nhanh hơn như D\* Lite hoặc Jump Point Search (JPS).

\chapter{Kết Luận}

Trong bài tập lớn này, chúng tôi đã cố gắng hết sức để tái hiện cũng như xây dựng thành công một phiên bản nâng cao của trò chơi rắn săn mồi, tích hợp trí tuệ nhân tạo để tạo ra một hệ thống thông minh, có khả năng tự động điều hướng và thích nghi với các tình huống phức tạp. Bằng cách áp dụng thuật toán A\*, nhóm không chỉ tái hiện được trò chơi kinh điển mà còn nâng cao trải nghiệm người chơi thông qua các tính năng hiện đại như xử lý chướng ngại vật di động, thiết kế nhiều cấp độ khó, và tối ưu hóa hiệu suất. Hệ thống được thiết kế theo mô hình lưới (grid-based), với các thành phần chính bao gồm bản đồ, mô-đun tìm đường, cơ chế cập.

Tuy nhiên trong trò chơi của chúng tôi còn tồn tại nhiều nhược điểm có thế nhắc đến như còn quá đơn giản và có phần nhàm chán, việc chỉ có 2 chế độ chơi thử công và tự động là còn quá ít so với một con game huyền thoại như vậy, điều này nhắc nhở chúng tôi phải thật kĩ lưỡng và cẩn thận, chăm chỉ trau dồi và tìm tòi để có thể sáng tạo thêm nữa những cơ chế, những sản phẩm sau này.

Đó cũng chính là những lời cuối mà chúng tôi muốn nói, hướng phát triển của game này nói chung hay hàng triệu con game khác nói riêng trên toàn thế giới sẽ không ngừng được nâng cao, bởi khi công nghệ AI ngày càng hoà nhập và nâng tầm nó, sự phát triển để phục vụ những bài toán mới, những nhu cầu mới sẽ mãi là một đề tài vô hạn sáng tạo cho con người chúng ta, để phục vụ ngành công nghiệp giải trí, hay để phục vụ cuộc sống, để khám phá và chinh phục những điều mới mẻ trong tương lai phát triển của thế giới sau này!

\end{document}