Ho Chi Minh City University of Technology
Faculty of Computer Science and Engineering

ADVANCED PROGRAMMING

PROJECT

Báo cáo Học phần mở rộng

Student: Bùi Trọng Hiển - 2310993
Advisor: Trương Tuấn Anh

April, 2025

Mục lục

1 So sánh Lập Trình Hướng Đối Tượng trong Java và Ruby

1.1 Giới thiệu về Lập Trình Hướng Đối Tượng (OOP)

Lập trình hướng đối tượng (Object-Oriented Programming – OOP) là một mô hình lập trình tập trung vào **đối tượng** thay vì chức năng. Mỗi đối tượng là sự kết hợp giữa *dữ liệu (thuộc tính)* và *hành vi (phương thức)*.

Bốn đặc trưng cơ bản của OOP:

- **Encapsulation (Đóng gói)**: Giấu thông tin chi tiết bên trong đối tượng.
- **Inheritance (Kế thừa)**: Cho phép tạo lớp mới từ lớp hiện có.
- **Polymorphism (Đa hình)**: Hành vi của đối tượng phụ thuộc vào ngữ cảnh.
- **Abstraction (Trừu tượng)**: Chỉ hiển thị phần cần thiết, ẩn đi chi tiết không quan trọng.

1.2 OOP trong Java

Java là một ngôn ngữ lập trình hướng đối tượng thuần túy, nơi mọi thứ đều được tổ chức dưới dạng `class` (lớp) và `object` (đối tượng). Java sử dụng kiểu tĩnh (static typing), tức là kiểu dữ liệu của biến và đối tượng phải được xác định tại thời điểm biên dịch.

Class và Object Trong Java, một lớp là bản thiết kế cho đối tượng. Đối tượng là một thể hiện (instance) cụ thể của lớp. Các thuộc tính (fields) lưu trữ trạng thái của đối tượng, còn các phương thức (methods) định nghĩa hành vi.

```java
public class Person {
    private String name;

    public Person(String name) {
        this.name = name;
    }

    public void greet() {
        System.out.println("Hello, my name is " + name);
    }
}
```

Ví dụ đơn giản về class và object

Tính kế thừa (Inheritance) Java hỗ trợ kế thừa thông qua từ khóa `extends`, cho phép lớp con sử dụng và ghi đè các phương thức của lớp cha.

```java
public class Student extends Person {
    public Student(String name) {
        super(name);
    }

    @Override
    public void greet() {
        System.out.println("Hi, I'm a student and my name is " + name);
    }
}
```

Tính đa hình (Polymorphism) Java hỗ trợ đa hình ở cả mức compile-time (method overloading) và runtime (method overriding), cho phép các đối tượng có hành vi khác nhau khi gọi cùng một phương thức.

Interface Interface là một tập hợp các phương thức trừu tượng. Các lớp có thể thực thi nhiều interface cùng lúc, cho phép mô phỏng đa kế thừa.

```
public interface Flyable {
    void fly();
}

public class Bird implements Flyable {
    public void fly() {
        System.out.println("Bird flies");
    }
}
```

Ví dụ về interface

Interface trong Java giúp định nghĩa hợp đồng hành vi, thúc đẩy thiết kế hướng giao diện (interface-driven design).

1.3 OOP trong Ruby

Ruby là ngôn ngữ hướng đối tượng thuần túy, kiểu động (dynamic typing), trong đó tất cả mọi thứ, bao gồm số nguyên, chuỗi, thậm chí cả `nil`, đều là đối tượng. Điều này mang lại sự linh hoạt lớn khi lập trình.

Class và Object Ruby cho phép định nghĩa lớp một cách đơn giản và linh hoạt. Thuộc tính được định nghĩa với `@` và phương thức được định nghĩa bằng từ khóa `def`.

```
class Animal
  def initialize(name)
    @name = name
  end

  def speak
    puts "#{@name} speaks"
  end
end
```

Lớp Animal trong Ruby

Tính kế thừa Ruby hỗ trợ kế thừa thông qua dấu `<`, và chỉ hỗ trợ đơn kế thừa (single inheritance).

```
class Dog < Animal
  def speak
    puts "#{@name} barks"
  end
end
```

Duck Typing Duck typing là một đặc điểm nổi bật của Ruby. Nó dựa trên nguyên lý: *"If it walks like a duck and quacks like a duck, it is a duck"*. Tức là, một đối tượng được xem là hợp lệ nếu nó có phương thức cần thiết, bất kể lớp của nó là gì.

```ruby
def make_it_speak(animal)
  animal.speak
end

make_it_speak(Dog.new("Lucky"))
```

Ví dụ duck typing

Trong ví dụ trên, hàm `make_it_speak` không cần biết đối tượng có phải là Dog hay Animal, chỉ cần nó có phương thức `speak` là đủ.

Module và Mixin Ruby không hỗ trợ đa kế thừa, nhưng giải quyết bằng cách sử dụng `module` và `mixin`. Module là một tập hợp phương thức có thể được chèn vào lớp bằng `include` hoặc `extend`.

```ruby
module Flyable
  def fly
    puts "I can fly!"
  end
end

class Bird
  include Flyable
end

Bird.new.fly
```

Ví dụ module và mixin

include sẽ thêm phương thức vào như thể chúng là instance method. **extend** sẽ thêm như class method.

Tính linh hoạt Ruby cho phép mở rộng lớp tại runtime, định nghĩa lại phương thức, hoặc sử dụng metaprogramming để tự động sinh mã. Tuy nhiên, điều này cần được dùng cẩn trọng để tránh lỗi khó phát hiện.

1.4 So sánh OOP trong Java và Ruby

Tiêu chí	**Java**	**Ruby**
Kiểu dữ liệu	Tĩnh (static typing)	Động (dynamic typing)
Độ nghiêm ngặt cú pháp	Cao	Thoải mái
Interface	Có interface rõ ràng	Dùng module/mixin
Duck typing	Không hỗ trợ	Hỗ trợ mạnh
Kiểm tra lỗi	Compile-time	Runtime
Môi trường thực thi	JVM (Java Virtual Machine)	Thông dịch (interpreter)
Phù hợp cho	Hệ thống lớn, doanh nghiệp	Ứng dụng nhanh, startup

So sánh OOP giữa Java và Ruby

1.5 Nhận xét

Java và Ruby đều hỗ trợ lập trình hướng đối tượng nhưng có cách tiếp cận khác nhau:

- Java nghiêm ngặt, rõ ràng và an toàn hơn trong các dự án lớn.
- Ruby linh hoạt, ngắn gọn, phù hợp để phát triển nhanh và thử nghiệm ý tưởng.

- Nếu ưu tiên **kiểm tra kiểu dữ liệu**, **hiệu năng**, Java là lựa chọn tốt.
- Nếu ưu tiên **tốc độ phát triển** và **đơn giản hóa code**, Ruby là lựa chọn phù hợp.

2 So sánh Lập trình Hàm trong Haskell và Python

2.1 Giới thiệu về Lập trình Hàm

Lập trình hàm là một mô hình lập trình trong đó việc tính toán được xem như việc đánh giá các hàm toán học và tránh sử dụng trạng thái thay đổi hay dữ liệu có thể biến đổi. Mô hình này nhấn mạnh vào các hàm thuần túy, tính bất biến, và cách tiếp cận khai báo.

2.2 Lập trình Hàm trong Haskell

Haskell là một ngôn ngữ lập trình thuần hàm (purely functional programming language), trong đó mọi biểu thức đều là hàm thuần túy. Ngôn ngữ này được thiết kế với mục tiêu chính là hỗ trợ tối đa các đặc trưng của lập trình hàm.

- **Hàm thuần túy (Pure Functions):** Trong Haskell, hàm thuần túy là nền tảng. Một hàm thuần túy là hàm mà kết quả chỉ phụ thuộc vào các đối số truyền vào, không phụ thuộc vào trạng thái bên ngoài và không gây ra tác dụng phụ (side effects). Điều này giúp chương trình dễ kiểm thử, dễ suy luận và dễ song song hóa.
- **Tính bất biến (Immutability):** Mọi giá trị trong Haskell là bất biến. Khi một biến được gán giá trị, giá trị đó không thể thay đổi. Điều này loại bỏ hoàn toàn lỗi do thay đổi trạng thái ngoài ý muốn.
- **Đánh giá lười (Lazy Evaluation):** Haskell sử dụng đánh giá lười, nghĩa là biểu thức chỉ được tính toán khi thật sự cần thiết. Điều này cho phép định nghĩa các cấu trúc dữ liệu vô hạn, cải thiện hiệu suất bằng cách tránh các phép tính không cần thiết.
- **Hàm bậc cao (Higher-Order Functions):** Haskell hỗ trợ mạnh mẽ hàm bậc cao, tức là hàm có thể nhận hàm khác làm tham số hoặc trả về một hàm khác. Đây là một yếu tố quan trọng trong việc trừu tượng hóa và tái sử dụng mã nguồn.
- **Hệ thống kiểu tĩnh mạnh (Strong Static Typing):** Haskell có hệ thống kiểu tĩnh mạnh kết hợp với khả năng suy diễn kiểu (type inference), giúp phát hiện lỗi tại thời điểm biên dịch mà không cần chỉ định rõ ràng kiểu ở mọi nơi.

2.2.1 Ví dụ biến bất biến và hàm thuần túy trong Haskell:

```haskell
let x = 5
-- x cannot be changed after declaration

square :: Int -> Int
square n = n * n
-- This is a pure function: no side effects, output depends only on input
```

Biến không thay đổi và hàm thuần túy trong Haskell

2.2.2 Ví dụ hàm bậc cao:

```haskell
applyTwice :: (a -> a) -> a -> a
applyTwice f x = f (f x)
-- applyTwice applies function f twice to input x
```

Hàm bậc cao applyTwice trong Haskell

2.3 Lập trình Hàm trong Python

Python là một ngôn ngữ lập trình đa mô hình (multi-paradigm), hỗ trợ lập trình hướng đối tượng, thủ tục và cả lập trình hàm. Tuy nhiên, khác với Haskell, lập trình hàm trong Python không phải là thuần túy.

- **Hàm là đối tượng bậc nhất (First-Class Functions):** Trong Python, hàm có thể được gán cho biến, truyền làm tham số cho hàm khác, và được trả về từ hàm — nghĩa là chúng có thể xử lý như bất kỳ đối tượng nào khác.
- **Hàm bậc cao (Higher-Order Functions):** Python cung cấp sẵn các hàm như `map()`, `filter()`, `reduce()` và cho phép người dùng tự định nghĩa hàm nhận hoặc trả về hàm khác. Điều này hỗ trợ viết mã ngắn gọn và biểu cảm hơn.
- **Biểu thức lambda (Lambda Expressions):** Python cho phép khai báo hàm ẩn danh (anonymous function) ngắn gọn bằng từ khóa `lambda`, thích hợp dùng trong các biểu thức ngắn.
- **Không đảm bảo tính thuần túy:** Các hàm trong Python có thể truy cập và thay đổi biến bên ngoài, in ra màn hình, đọc/ghi file — nghĩa là có thể có tác dụng phụ (side effects). Do đó, Python không ép buộc hàm phải thuần túy như Haskell.
- **Bất biến tuỳ thuộc kiểu dữ liệu:** Python hỗ trợ cả kiểu bất biến (immutable) như `tuple`, `str` và kiểu có thể thay đổi (mutable) như `list`, `dict`. Tuy nhiên, biến trong Python có thể gán lại giá trị, tức là tên biến không bị ràng buộc cố định như trong Haskell.

2.3.1 Ví dụ hàm bậc cao và lambda trong Python:

```python
# Define a function that applies another function twice
def apply_twice(f, x):
    return f(f(x))

square = lambda n: n * n  # anonymous function to square a number

result = apply_twice(square, 3)
print(result)  # Output: 81
```

Hàm bậc cao và biểu thức lambda trong Python

2.3.2 Ví dụ dữ liệu bất biến với tuple:

```python
point = (2, 3)  # tuple is immutable
# point[0] = 5  # This will raise an error
```

Sử dụng tuple - kiểu dữ liệu bất biến trong Python

2.4 So sánh giữa Haskell và Python trong Lập trình Hàm

2.4.1 Tính thuần túy và tác dụng phụ

Trong Haskell, mọi hàm đều là *hàm thuần túy* — nghĩa là đầu ra của hàm chỉ phụ thuộc vào tham số đầu vào, không làm thay đổi trạng thái chương trình bên ngoài (không có tác dụng phụ). Điều này giúp đảm bảo độ tin cậy và dễ kiểm soát trong các chương trình lớn.

Trong khi đó, Python không bắt buộc tính thuần túy. Các hàm trong Python có thể ghi ra màn hình, đọc/ghi file, thay đổi biến toàn cục,... Việc này mang lại tính linh hoạt nhưng cũng có thể gây lỗi nếu không kiểm soát chặt.

2.4.2 Hệ thống kiểu

Haskell sử dụng hệ thống *kiểu tĩnh mạnh* với cơ chế suy diễn kiểu. Các kiểu dữ liệu được kiểm tra tại thời điểm biên dịch, giúp phát hiện lỗi sớm. Ngoài ra, Haskell hỗ trợ các khái niệm mạnh như kiểu hàm, kiểu tham số hóa, và lớp kiểu (*type classes*).

Python sử dụng hệ thống *kiểu động*, nghĩa là kiểu dữ liệu được xác định tại thời điểm chạy. Điều này làm Python dễ tiếp cận hơn nhưng có thể dẫn đến lỗi nếu thiếu kiểm soát.

2.4.3 Tính bất biến

Trong Haskell, tất cả các giá trị đều bất biến theo mặc định. Khi cần thay đổi trạng thái, lập trình viên phải sử dụng các kỹ thuật riêng như *monad*. Điều này giúp chương trình dễ kiểm soát hơn và tránh các lỗi không mong muốn.

Python hỗ trợ cả kiểu dữ liệu bất biến (như tuple, string) và kiểu dữ liệu có thể thay đổi (list, dict, set). Ngoài ra, biến có thể gán lại. Điều này linh hoạt hơn nhưng dễ gây ra lỗi do thay đổi trạng thái ngoài ý muốn.

2.4.4 Đánh giá biểu thức

Haskell sử dụng cơ chế *đánh giá lười* (lazy evaluation), nghĩa là biểu thức chỉ được tính toán khi thật sự cần đến giá trị của nó. Điều này cho phép tạo và xử lý các cấu trúc dữ liệu vô hạn một cách hiệu quả.

Python sử dụng *đánh giá eager* (tức thì), nghĩa là biểu thức được tính toán ngay khi được khai báo. Một số tính năng như `generator` có thể giả lập đánh giá lười, nhưng không toàn diện như Haskell.

2.4.5 Hàm bậc cao và cú pháp

Cả hai ngôn ngữ đều hỗ trợ hàm bậc cao — tức là hàm có thể nhận hàm khác làm tham số hoặc trả về hàm. Tuy nhiên, cú pháp trong Haskell ngắn gọn và nhất quán hơn vì được thiết kế dành riêng cho lập trình hàm. Trong khi đó, Python cần nhiều cú pháp bổ sung và đôi khi khó đọc hơn trong các biểu thức lồng nhau.

2.4.6 Ví dụ minh họa sự khác biệt

Ví dụ sau minh họa cách tính tổng bình phương các số chẵn trong một danh sách.

Haskell:

```haskell
sumOfSquares :: [Int] -> Int
sumOfSquares xs = sum [x^2 | x <- xs, even x]

-- sumOfSquares [1,2,3,4,5] => 20
```

Tính tổng bình phương các số chẵn trong Haskell

Python:

```python
def sum_of_squares(lst):
    return sum([x**2 for x in lst if x % 2 == 0])

# sum_of_squares([1,2,3,4,5]) => 20
```

Tính tổng bình phương các số chẵn trong Python

Phân tích: Cả hai đoạn mã đều cho kết quả như nhau. Tuy nhiên:

- Haskell sử dụng cú pháp list comprehension một cách rất tự nhiên và ngắn gọn, không cần biến trung gian.

- Trong Haskell, hàm chắc chắn là thuần túy và bất biến. Trong Python, nếu thêm dòng `print()` vào hàm, tính thuần túy không còn được đảm bảo.
- Haskell kiểm tra kiểu tại thời điểm biên dịch, giúp phát hiện lỗi sớm. Python thì không.

2.5 Nhận xét

Mặc dù cả Haskell và Python đều hỗ trợ lập trình hàm, Haskell là ngôn ngữ thuần hàm, đảm bảo tính bất biến và không tác dụng phụ, rất phù hợp cho các bài toán toán học và tính toán đòi hỏi độ chính xác cao. Python linh hoạt hơn, hỗ trợ kết hợp đa mô hình, phù hợp với phát triển đa dạng ứng dụng, có thể kết hợp lập trình hàm với các mô hình khác.

3 Nghiên cứu các Hướng Lập Trình Mới

3.1 Data Wrangling – Làm sạch và xử lý dữ liệu

3.1.1 Khái niệm và vai trò

Data Wrangling (còn gọi là làm sạch và chuẩn bị dữ liệu) là quá trình chuyển đổi dữ liệu thô – thường không được tổ chức, thiếu thông tin, hoặc định dạng không thống nhất – thành dạng dữ liệu có cấu trúc, nhất quán và dễ phân tích. Đây là một bước tiền xử lý cực kỳ quan trọng trong lĩnh vực Khoa học Dữ liệu (Data Science), Trí tuệ nhân tạo (AI) và Học máy (Machine Learning), bởi vì chất lượng của dữ liệu đầu vào ảnh hưởng trực tiếp đến độ chính xác của mô hình học.

Theo một thống kê phổ biến, các nhà khoa học dữ liệu dành khoảng 60–80% thời gian làm việc cho quá trình Data Wrangling, điều này cho thấy tầm quan trọng của kỹ năng này trong thực tế.

3.1.2 Các bước cơ bản trong quá trình Data Wrangling

Quá trình Data Wrangling thường bao gồm các bước sau:

- **1. Khai thác dữ liệu (Data Acquisition):** Thu thập dữ liệu từ các nguồn khác nhau như file CSV, Excel, cơ sở dữ liệu SQL, API, hoặc web scraping. Dữ liệu đầu vào có thể đa dạng về định dạng và cấu trúc.
- **2. Làm sạch dữ liệu (Data Cleaning):** Loại bỏ các giá trị thiếu (missing), không hợp lệ (invalid), hoặc ngoại lai (outliers). Ngoài ra, cần xử lý dữ liệu trùng lặp, chuẩn hóa định dạng (ví dụ: ngày tháng), và phát hiện lỗi logic trong dữ liệu.
- **3. Biến đổi dữ liệu (Data Transformation):** Chuyển đổi định dạng dữ liệu hoặc tái cấu trúc lại dữ liệu để phù hợp với yêu cầu phân tích. Có thể bao gồm chuẩn hóa dữ liệu (normalization), chuẩn hóa kiểu dữ liệu, phân loại dữ liệu thành nhóm (binning), hoặc tạo ra các biến mới từ dữ liệu cũ (feature engineering).
- **4. Kết hợp dữ liệu (Data Integration):** Gộp dữ liệu từ nhiều nguồn lại với nhau bằng cách nối (merge), ghép nối (join), hoặc liên kết bảng dữ liệu theo khoá chính/phụ (key).
- **5. Tái định dạng dữ liệu (Reshaping):** Thay đổi cấu trúc dữ liệu từ dạng rộng sang dạng dài hoặc ngược lại (wide to long), pivot table, stack/unstack, nhằm phục vụ mục đích trực quan hóa hoặc phân tích.
- **6. Xác thực dữ liệu (Validation):** Kiểm tra độ chính xác và nhất quán của dữ liệu sau xử lý để đảm bảo rằng nó có thể được sử dụng an toàn trong các bước tiếp theo như phân tích hoặc huấn luyện mô hình.

3.1.3 Ngôn ngữ và thư viện phổ biến

Python là ngôn ngữ lập trình phổ biến nhất trong lĩnh vực Data Wrangling nhờ hệ sinh thái thư viện mạnh mẽ:

- `pandas`: Thư viện nền tảng hỗ trợ thao tác dữ liệu dưới dạng DataFrame (tương tự bảng tính Excel), giúp đọc, xử lý, tổng hợp, biến đổi và phân tích dữ liệu hiệu quả.
- `numpy`: Cung cấp các thao tác với mảng số học hiệu năng cao, hỗ trợ xử lý dữ liệu số lượng lớn.
- `openpyxl`, `csv`, `json`: Dùng để tương tác với các định dạng dữ liệu khác nhau như Excel, CSV, hoặc JSON.
- `matplotlib`, `seaborn`: Dùng để trực quan hóa dữ liệu trong quá trình kiểm tra và xác thực.

Ví dụ thực tiễn với pandas trong Python

```python
import pandas as pd

# Read data from CSV file
df = pd.read_csv("du_lieu_nhan_vien.csv")

# Show first 5 lines
print(df.head())

# Delete rows with missing values
df = df.dropna()

# Convert data type
df["tuoi"] = df["tuoi"].astype(int)

# Normalize text data (uppercase -> lowercase)
df["ten_phong_ban"] = df["ten_phong_ban"].str.lower()

# Calculate average salary by department
luong_tb = df.groupby("ten_phong_ban")["luong"].mean()

print(luong_tb)
```

Ví dụ làm sạch và phân tích dữ liệu với pandas

Thách thức trong Data Wrangling

- **Dữ liệu chất lượng kém:** Thường xuyên gặp thiếu dữ liệu, dữ liệu lỗi, hoặc không thống nhất.
- **Định dạng dữ liệu không chuẩn:** Dữ liệu có thể được lưu trữ dưới nhiều dạng không tương thích.
- **Khối lượng dữ liệu lớn:** Khi làm việc với dữ liệu hàng triệu dòng, các thao tác wrangling cần được tối ưu hóa.
- **Thiếu tài liệu:** Nhiều nguồn dữ liệu không có tài liệu giải thích, gây khó khăn trong việc hiểu và xử lý.

3.1.4 Tổng kết

Data Wrangling là kỹ năng nền tảng nhưng cực kỳ quan trọng đối với mọi nhà phân tích dữ liệu. Một quy trình xử lý dữ liệu tốt giúp tiết kiệm thời gian, tăng độ chính xác và đảm bảo rằng các mô hình học máy hoặc phân tích thống kê sẽ hoạt động hiệu quả hơn. Việc sử dụng lập trình – đặc biệt là trong các ngôn ngữ như Python – cho phép tự động hóa quá trình này, tăng hiệu suất và độ tin cậy trong công việc thực tế.

3.2 Smart Contract – Hợp đồng thông minh

3.2.1 Khái niệm và định nghĩa

Smart Contract (hợp đồng thông minh) là một chương trình máy tính tự động thực hiện, kiểm soát hoặc ghi nhận các sự kiện và hành động theo điều khoản của một hợp đồng hoặc thỏa thuận đã được mã hóa sẵn. Các hợp đồng này chạy trên nền tảng blockchain, một cơ sở dữ liệu phân tán, phi tập trung và bất biến, giúp đảm bảo tính minh bạch, an toàn và không thể chỉnh sửa sau khi được triển khai.

Thuật ngữ "Smart Contract" được giới thiệu lần đầu bởi nhà khoa học Nick Szabo vào năm 1994, nhằm mô tả các hợp đồng tự động, có thể thực thi mà không cần bên trung gian, giảm thiểu rủi ro và chi phí giao dịch.

3.2.2 Cách hoạt động của Smart Contract

Smart Contract hoạt động theo nguyên tắc "nếu... thì..." (if... then...), trong đó các điều kiện được xác định rõ ràng trong mã lệnh. Khi các điều kiện được đáp ứng, hợp đồng sẽ tự động thực thi các hành động được định trước, chẳng hạn chuyển tiền, cập nhật trạng thái, hoặc cấp quyền truy cập.

Một điểm quan trọng là các Smart Contract chạy trên blockchain nên:

- **Tính bất biến:** Mã nguồn và dữ liệu hợp đồng không thể bị sửa đổi sau khi được triển khai.
- **Tính minh bạch:** Mọi giao dịch và trạng thái hợp đồng đều được ghi lại trên blockchain công khai, ai cũng có thể kiểm tra.
- **Tự động hóa:** Giảm bớt sự can thiệp của con người, tránh sai sót và gian lận.
- **Phân quyền:** Không có bên thứ ba kiểm soát hợp đồng, giảm nguy cơ tham nhũng hoặc thao túng.

3.2.3 Ngôn ngữ lập trình và nền tảng phổ biến

Ngôn ngữ lập trình Smart Contract thường đặc thù cho từng nền tảng blockchain:

- **Solidity:** Ngôn ngữ phổ biến nhất trên nền tảng Ethereum, cú pháp tương tự JavaScript, dễ học và phát triển.
- **Vyper:** Ngôn ngữ an toàn hơn, ngắn gọn hơn, cũng cho Ethereum nhưng hạn chế các tính năng phức tạp để giảm rủi ro bảo mật.
- **Rust, C++, Go:** Được sử dụng trong các nền tảng blockchain khác như Solana, EOS, hoặc Hyperledger.

3.2.4 Ứng dụng của Smart Contract

Smart Contract đã mở rộng ứng dụng trên nhiều lĩnh vực khác nhau, ví dụ:

- **Tài chính phi tập trung (DeFi):** Tự động hóa vay, cho vay, giao dịch, bảo hiểm mà không cần ngân hàng hoặc trung gian.
- **Quản lý chuỗi cung ứng:** Theo dõi sản phẩm từ nguồn gốc đến tay người tiêu dùng một cách minh bạch và không thể làm giả.
- **Bỏ phiếu điện tử:** Tăng tính minh bạch và bảo mật trong các hệ thống bầu cử.
- **Bản quyền và quyền sở hữu trí tuệ:** Ghi nhận, phân phối tiền bản quyền tự động.
- **Quản lý tài sản kỹ thuật số:** NFT (Non-Fungible Token) sử dụng Smart Contract để xác nhận quyền sở hữu duy nhất.

3.2.5 Ưu điểm và hạn chế

Ưu điểm:

- **Tự động và nhanh chóng:** Thực thi hợp đồng nhanh chóng, không phụ thuộc bên trung gian.
- **Minh bạch và an toàn:** Mọi giao dịch đều được ghi lại, bảo mật nhờ công nghệ blockchain.
- **Giảm chi phí:** Loại bỏ hoặc giảm đáng kể chi phí cho trung gian, luật sư, giấy tờ.

Hạn chế:

- **Khó sửa đổi:** Sau khi triển khai, việc sửa lỗi hoặc cập nhật rất khó khăn vì tính bất biến của blockchain.
- **Rủi ro bảo mật:** Lỗi lập trình trong Smart Contract có thể dẫn đến thiệt hại tài chính lớn (ví dụ: vụ hack DAO trên Ethereum).
- **Khả năng mở rộng:** Hiện nay các blockchain phổ biến vẫn còn giới hạn về tốc độ xử lý và chi phí giao dịch.
- **Pháp lý chưa rõ ràng:** Hợp đồng thông minh còn chưa được luật pháp nhiều nước công nhận hoặc điều chỉnh đầy đủ.

Ví dụ đơn giản về Smart Contract bằng Solidity

```
pragma solidity ^0.8.0;

contract SimpleStorage {
    uint256 private data;
    // Assign value to data variable
    function set(uint256 _data) public {
        data = _data;
    }
    // Get current value of data
    function get() public view returns (uint256) {
        return data;
    }
}
```

Ví dụ hợp đồng thông minh đơn giản bằng Solidity

Hợp đồng này lưu trữ một giá trị số nguyên, cho phép người dùng ghi và đọc giá trị đó mà không cần bên trung gian nào.

3.2.6 Tổng kết

Smart Contract là một bước tiến quan trọng trong công nghệ chuỗi khối, mang lại sự minh bạch, tự động hóa và giảm thiểu chi phí trong các giao dịch và hợp đồng. Tuy nhiên, vẫn còn nhiều thách thức về bảo mật, khả năng mở rộng và khung pháp lý cần được giải quyết. Việc nghiên cứu và áp dụng Smart Contract ngày càng phổ biến, đặc biệt trong các lĩnh vực tài chính, logistics, và quản lý quyền sở hữu kỹ thuật số.

4 Giới thiệu Ngôn Ngữ GoLang và Cách Lập Trình

4.1 Tổng quan về GoLang

GoLang, hay còn gọi là Go, là một ngôn ngữ lập trình mã nguồn mở được phát triển bởi Google vào năm 2007 và công bố chính thức vào năm 2009. Go được thiết kế nhằm mục đích kết hợp hiệu suất và sự đơn giản của ngôn ngữ lập trình hệ thống với sự tiện lợi và năng suất của ngôn ngữ lập trình cấp cao.

Go đặc biệt phù hợp cho phát triển các ứng dụng có quy mô lớn, có tính đồng thời cao như hệ thống mạng, dịch vụ web, và các ứng dụng đám mây. Ngôn ngữ này nổi bật với cú pháp đơn giản, hiệu năng cao, và hỗ trợ tích hợp tốt cho lập trình đa luồng (concurrency).

4.2 Đặc điểm nổi bật của Go

GoLang được thiết kế nhằm giải quyết nhiều hạn chế của các ngôn ngữ lập trình hiện đại đồng thời giữ lại sự đơn giản và hiệu quả. Một số đặc điểm nổi bật của Go bao gồm:

- **Cú pháp đơn giản, rõ ràng:** Go loại bỏ nhiều phức tạp không cần thiết thường gặp ở các ngôn ngữ khác như C++ hay Java. Cú pháp Go dễ đọc, dễ học và giúp lập trình viên tập trung vào logic thay vì chi tiết ngôn ngữ.
- **Biên dịch nhanh, hiệu suất cao:** Go là ngôn ngữ biên dịch (compiled language), cho phép chuyển mã nguồn trực tiếp thành mã máy nhanh và tối ưu. Thời gian biên dịch rất nhanh, giúp tăng năng suất phát triển.
- **Hỗ trợ lập trình đồng thời (concurrency) đơn giản và mạnh mẽ:** Một trong những điểm nổi bật nhất của Go là cơ chế đồng thời tích hợp sẵn với goroutines và channels. Goroutines là các luồng nhẹ có chi phí rất thấp, có thể chạy hàng nghìn goroutine đồng thời mà không ảnh hưởng đến hiệu suất. Channels cho phép giao tiếp và đồng bộ dữ liệu giữa các goroutine một cách an toàn và hiệu quả.
- **Quản lý bộ nhớ tự động (Garbage Collection):** Go có cơ chế quản lý bộ nhớ tự động giúp giải phóng bộ nhớ không còn sử dụng mà không cần lập trình viên can thiệp thủ công, giảm thiểu lỗi về rò rỉ bộ nhớ.
- **Hệ thống kiểu tĩnh mạnh mẽ:** Go có hệ thống kiểu tĩnh, giúp phát hiện lỗi kiểu dữ liệu ngay khi biên dịch, đảm bảo tính an toàn và ổn định cho chương trình. Đồng thời Go hỗ trợ kiểu dữ liệu cơ bản (int, float, bool, string), cấu trúc dữ liệu phức tạp (struct, array, slice, map) và interface.
- **Thư viện tiêu chuẩn phong phú và đa dạng:** Go cung cấp thư viện chuẩn lớn, hỗ trợ đầy đủ các chức năng cần thiết cho lập trình mạng, xử lý file, mã hóa, web, đa luồng, ... giúp lập trình viên không cần phải phụ thuộc quá nhiều vào thư viện bên ngoài.
- **Hỗ trợ đa nền tảng:** Go có thể biên dịch cho nhiều hệ điều hành và kiến trúc phần cứng khác nhau như Windows, Linux, macOS, ARM,... giúp việc phát triển và triển khai ứng dụng đa dạng hơn.
- **Không có kế thừa lớp (class inheritance):** Go không hỗ trợ kế thừa theo kiểu hướng đối tượng truyền thống mà thay vào đó sử dụng interface và composition để xây dựng các đối tượng phức tạp, giúp đơn giản hóa thiết kế và tránh những vấn đề phức tạp của kế thừa đa lớp.

4.3 Cách lập trình cơ bản trong Go

Cấu trúc chương trình

Mỗi chương trình Go bắt đầu với việc khai báo `package`. Package `main` được sử dụng để định nghĩa chương trình thực thi chính.

Các gói (package) bên ngoài được nhập bằng từ khóa `import`, cho phép sử dụng các hàm và kiểu dữ liệu đã được định nghĩa trong các gói đó.

Hàm `main` là điểm bắt đầu thực thi chương trình.

```
package main

import "fmt"

func main() {
    fmt.Println("Hello, Go!")
}
```

Khai báo biến và kiểu dữ liệu

Go là ngôn ngữ có kiểu tĩnh, nhưng cho phép khai báo biến ngắn gọn bằng toán tử `:=` trong phạm vi hàm.

Một số kiểu dữ liệu cơ bản phổ biến:

- `int, int8, int16, int32, int64`: số nguyên có kích thước khác nhau
- `uint, uint8, uint16, uint32, uint64`: số nguyên không dấu
- `float32, float64`: số thực
- `bool`: kiểu logic đúng/sai
- `string`: chuỗi ký tự

Ví dụ khai báo biến:

```
var age int = 30        // Declare variables with explicit types
name := "Nguyen Van A"  // Short variable declaration, self-inferring type
```

Go cũng hỗ trợ các cấu trúc dữ liệu phức tạp như mảng (array), danh sách động (slice), bản đồ (map) và cấu trúc (struct).

Cấu trúc điều khiển

Go có các cấu trúc điều khiển tương tự các ngôn ngữ lập trình truyền thống:

- `if-else` dùng để kiểm tra điều kiện
- `for` là vòng lặp duy nhất trong Go, có thể dùng như `while` hoặc `do-while`
- `switch` hỗ trợ lựa chọn nhiều trường hợp

Ví dụ:

```
for i := 0; i < 5; i++ {
    fmt.Println(i)
}

if age >= 18 {
    fmt.Println("Adult")
```

```
} else {
    fmt.Println("Children")
}

switch day := 3; day {
case 1:
    fmt.Println("Monday")
case 2:
    fmt.Println("Tuesday")
default:
    fmt.Println("Other day")
}
```

Định nghĩa hàm

Hàm trong Go được định nghĩa bằng từ khóa `func`. Hàm có thể nhận nhiều tham số và trả về một hoặc nhiều giá trị.

Ví dụ:

```
// Function to calculate the sum of two integers
func sum(a int, b int) int {
    return a + b
}

// Function returning multiple values: swapping two strings
func swap(x, y string) (string, string) {
    return y, x
}
```

Lập trình đồng thời (Concurrency)

Một điểm đặc biệt của Go là hỗ trợ lập trình đồng thời rất đơn giản với goroutines và channels.

- **Goroutine** là các luồng nhẹ, được khởi chạy bằng từ khóa `go` trước lời gọi hàm, giúp chạy các tác vụ song song.
- **Channels** cho phép giao tiếp và đồng bộ dữ liệu an toàn giữa các goroutine, tránh các vấn đề về tranh chấp dữ liệu.

Ví dụ tạo goroutine:

```
func hello() {
    fmt.Println("Hello from goroutine")
}

func main() {
    go hello()        // Run goroutine
    fmt.Println("Main function")
    // Add timeout or use sync to ensure goroutine completes
}
```

Tuy nhiên, vì goroutine chạy song song không đảm bảo thứ tự, nên cần sử dụng các công cụ đồng bộ như `sync.WaitGroup` hoặc channel để quản lý.

4.4 Ứng dụng của Go

Go được sử dụng rộng rãi trong các lĩnh vực:

- Phát triển hệ thống phân tán, dịch vụ backend.
- Xây dựng các công cụ dòng lệnh (CLI).
- Phát triển microservices và các ứng dụng đám mây.
- Các hệ thống cần xử lý đồng thời cao như server web, proxy, network tools.

4.5 Tổng kết

GoLang là ngôn ngữ hiện đại, đơn giản nhưng mạnh mẽ, đặc biệt thích hợp cho các ứng dụng cần hiệu năng cao và hỗ trợ lập trình đồng thời. Với cú pháp dễ học và thư viện chuẩn phong phú, Go đang ngày càng trở thành lựa chọn phổ biến trong phát triển phần mềm hiện đại, nhất là trong môi trường điện toán đám mây và hệ thống phân tán.

5 Trò chơi con rắn

5.1 Giới thiệu trò chơi

5.1.1 Tên game

Trò chơi con rắn (Snake Game) là một trò chơi cổ điển, trong đó người chơi điều khiển một con rắn di chuyển trên màn hình để ăn thức ăn. Mỗi khi ăn được thức ăn, rắn sẽ dài ra và tốc độ có thể tăng dần theo độ khó. Mục tiêu của trò chơi là đạt được điểm số cao nhất mà không để rắn tự va vào thân mình hoặc ra ngoài biên màn hình.

Ở phiên bản hiện thực trò chơi con rắn này, ta sử dụng ngôn ngữ lập trình C++ kết hợp thư viện SDL3 (Simple DirectMedia Layer) để xử lý đồ họa và nhập xuất. Chương trình được thiết kế theo hướng lập trình hướng đối tượng (OOP), giúp việc mở rộng, bảo trì và quản lý mã nguồn dễ dàng hơn.

5.1.2 Mục tiêu của game

Mục tiêu của trò chơi là điều khiển con rắn để ăn càng nhiều thức ăn càng tốt, từ đó tăng điểm số. Khi ăn thức ăn, con rắn sẽ dài ra, và tốc độ di chuyển sẽ tăng dần theo mức độ khó đã chọn. Trò chơi kết thúc khi rắn va chạm với tường (ở chế độ khó) hoặc chính thân mình.

5.2 Cách hoạt động

5.2.1 Luật chơi

- Người chơi điều khiển con rắn di chuyển trên màn hình bằng các phím mũi tên hoặc phím WASD.
- Khi rắn ăn thức ăn, điểm số tăng lên, và rắn dài ra.
- Ở chế độ dễ, rắn có thể xuyên qua tường mà không bị thua.
- Ở chế độ trung bình và khó, rắn sẽ thua nếu va chạm với tường.
- Trò chơi kết thúc khi rắn va chạm với chính thân mình hoặc tường (trong chế độ trung bình và khó).

5.2.2 Cách tương tác với người dùng

- Người chơi chọn mức độ khó (EASY, MEDIUM, HARD) từ menu trước khi bắt đầu trò chơi.
- Trong trò chơi, người chơi sử dụng các phím điều hướng để điều khiển rắn.
- Sau khi thua, người chơi có thể nhấn phím ENTER để chơi lại hoặc ESC để thoát.

5.3 Giải thích mã nguồn

5.3.1 Tổng quan kiến trúc chương trình

Chương trình bao gồm các thành phần chính như sau:

Thành phần	Mô tả
`Snake`	Lớp chính đại diện cho con rắn, bao gồm các thuộc tính và hành vi như di chuyển, phát triển, kiểm tra va chạm.
`Position`	Cấu trúc đại diện cho một vị trí trên lưới, gồm tọa độ `x` và `y`.
`Direction`	Kiểu `enum` định nghĩa hướng di chuyển của rắn (trên, dưới, trái, phải).
`Difficulty`	`enum` định nghĩa các mức độ khó của trò chơi (EASY, MEDIUM, HARD).
Các hàm tiện ích	Bao gồm hàm tạo vị trí thức ăn, thiết lập tốc độ, giới hạn tốc độ, xử lý menu lựa chọn độ khó, ...
`main()`	Hàm chính điều phối các phần còn lại, xử lý đầu vào, cập nhật và hiển thị game.

Các thư viện sử dụng

- `SDL3/SDL.h`: thư viện SDL để xử lý đồ họa, âm thanh, và nhập xuất.
- `iostream`: nhập/xuất dữ liệu qua console.
- `vector`, `deque`: dùng để lưu các phần tử như thân rắn hoặc lịch sử di chuyển.
- `ctime`, `random`: tạo số ngẫu nhiên để sinh ra vị trí thức ăn.
- `string`: làm việc với chuỗi ký tự.

Hằng số và cấu hình ban đầu

- `SCREEN_WIDTH = 640`, `SCREEN_HEIGHT = 480`: kích thước cửa sổ trò chơi.
- `GRID_SIZE = 20`: độ lớn của mỗi ô trên lưới trò chơi.
- `SNAKE_SPEED = 10`: tốc độ ban đầu của rắn (số lần cập nhật mỗi giây).

Kiểu `enum` và cấu trúc dữ liệu

- `Direction`: gồm `UP`, `RIGHT`, `DOWN`, `LEFT` – đại diện cho 4 hướng di chuyển.
- `Difficulty`: gồm `EASY`, `MEDIUM`, `HARD` – mức độ khó ảnh hưởng tới tốc độ rắn.
- `Position`: cấu trúc lưu vị trí `x`, `y` và định nghĩa toán tử so sánh `==` để kiểm tra vị trí trùng nhau.

Cấu hình tốc độ theo độ khó

- Tốc độ cập nhật theo độ khó
- Mức tăng tốc mỗi lần ăn
- Giới hạn tốc độ tối đa (cập nhật nhanh nhất)

Các hàm tiện ích đã khai báo

- `generateFood(const Snake& snake)`: tạo vị trí thức ăn ngẫu nhiên không trùng thân rắn.
- `setDifficultySettings(Difficulty, Uint32&)`: thiết lập tốc độ cập nhật ban đầu theo độ khó.
- `getMinUpdateInterval(Difficulty)`: lấy tốc độ tối đa tương ứng với độ khó.

- `getSpeedIncrease(Difficulty)`: lấy mức độ tăng tốc ứng với mỗi lần ăn.
- `showDifficultyMenu(SDL_Renderer*, Difficulty&)`: hiển thị menu cho người chơi chọn độ khó.

Hàm `main()`

Hàm `main()` là trung tâm điều phối hoạt động của toàn bộ trò chơi. Nó thực hiện:

- Khởi tạo SDL và các thành phần đồ họa.
- Hiển thị menu để người chơi chọn độ khó.
- Tạo vòng lặp trò chơi: xử lý đầu vào, cập nhật trạng thái, kiểm tra va chạm, và hiển thị lên màn hình.
- Thoát game khi rắn chết hoặc người chơi thoát.

5.3.2 Lớp `Snake`

Lớp `Snake` đại diện cho con rắn trong trò chơi. Nó quản lý trạng thái và hành vi của rắn, bao gồm vị trí, hướng di chuyển, khả năng tăng trưởng, kiểm tra va chạm và hiển thị rắn lên màn hình.

- **Thuộc tính:**
  - `body`: là một `deque` chứa các vị trí (kiểu `Position`) đại diện cho từng đốt thân rắn.
  - `direction`: lưu hướng hiện tại của rắn (trái, phải, lên, xuống).
  - `growing`: cờ đánh dấu rắn có đang lớn lên hay không (khi ăn thức ăn).
- **Phương thức khởi tạo:**
  - `Snake(int x, int y)`: khởi tạo rắn tại vị trí `(x, y)` với 3 đốt ban đầu theo chiều ngang.
- **Phương thức chính:**
  - `setDirection(Direction dir)`: thay đổi hướng đi, tránh quay đầu 180 độ.
  - `move(Difficulty difficulty)`: di chuyển rắn theo hướng hiện tại và xử lý giới hạn màn hình tùy thuộc vào độ khó.
  - `grow()`: đánh dấu rắn sẽ phát triển thêm một đốt ở lượt di chuyển kế tiếp.
  - `checkWallCollision(Difficulty difficulty)`: kiểm tra xem rắn có đâm vào tường hay không (tùy chế độ).
  - `checkSelfCollision()`: kiểm tra rắn có cắn vào chính mình không.
  - `checkFoodCollision(const Position& food)`: kiểm tra rắn có ăn trúng thức ăn không.
  - `render(SDL_Renderer* renderer, bool gameOver)`: vẽ rắn lên màn hình, màu đầu và thân rắn khác nhau.
  - `size()`: trả về độ dài hiện tại của rắn.
  - `getBodyPosition(size_t index)`: trả về vị trí một đốt thân của rắn theo chỉ số.

Lớp này đóng vai trò trung tâm trong việc điều khiển hành vi của rắn, hỗ trợ xử lý logic trò chơi như di chuyển, va chạm, ăn thức ăn và vẽ rắn lên màn hình. sectionCác hàm tiện ích liên quan đến trò chơi Bên cạnh lớp `Snake`, chương trình còn sử dụng một số hàm tiện ích quan trọng để phục vụ cho quá trình điều khiển trò chơi. Dưới đây là mô tả sơ bộ về chức năng của từng hàm:

- `generateFood(const Snake& snake)`: Hàm này có nhiệm vụ tạo ra vị trí ngẫu nhiên cho thức ăn trên màn hình. Đảm bảo rằng vị trí tạo ra không trùng với thân của rắn. Nếu sau nhiều lần thử không tìm được vị trí hợp lệ, hàm sẽ đặt thức ăn tại vị trí cách đầu rắn một khoảng nhất định. Việc này tránh gây ra vòng lặp vô hạn nếu thân rắn chiếm quá nhiều không gian.
- `setDifficultySettings(Difficulty difficulty, Uint32& updateInterval)`: Hàm này thiết lập tốc độ ban đầu của rắn (số lần cập nhật mỗi giây) dựa trên mức độ khó được chọn: DỄ, TRUNG BÌNH, hoặc KHÓ. Tốc độ càng cao thì rắn di chuyển càng nhanh, và `updateInterval` càng nhỏ.
- `getMinUpdateInterval(Difficulty difficulty)`: Trả về giá trị cập nhật tối thiểu (nhỏ nhất có thể) tương ứng với mức độ khó, dùng để giới hạn tốc độ tối đa mà rắn có thể đạt được trong suốt quá trình chơi.
- `getSpeedIncrease(Difficulty difficulty)`: Trả về mức độ tăng tốc (số lượng cập nhật thêm mỗi lần ăn được thức ăn) tương ứng với mức độ khó. Mức độ khó cao thì rắn tăng tốc nhanh hơn.
- `showDifficultyMenu(SDL_Renderer* renderer, Difficulty& selectedDifficulty)`: Hàm hiển thị menu lựa chọn độ khó cho người chơi và xử lý các sự kiện bàn phím để ghi nhận lựa chọn. Menu được hiển thị bằng SDL với ba tuỳ chọn chính: DỄ, TRUNG BÌNH, và KHÓ. Người chơi có thể dùng phím 1, 2, 3 hoặc ESC để thoát. Hàm này đảm bảo trải nghiệm tương tác ban đầu trước khi bắt đầu trò chơi.

5.3.3 Giải thích hàm `main`

Hàm `main` là điểm bắt đầu của chương trình và thực hiện các bước sau:

- Hiển thị hướng dẫn điều khiển game ra màn hình console để người chơi nắm được cách chơi.
- Khởi tạo SDL bằng `SDL_Init`, tạo cửa sổ (`SDL_CreateWindow`) và renderer (`SDL_CreateRenderer`) để phục vụ cho việc hiển thị đồ họa của game.
- Hiển thị menu để người chơi chọn mức độ khó (`EASY`, `MEDIUM`, hoặc `HARD`) bằng cách gọi hàm `showDifficultyMenu`. Nếu người chơi thoát tại đây, chương trình sẽ kết thúc.
- Tạo đối tượng `Snake` đại diện cho rắn và đặt nó ở vị trí giữa màn hình. Đồng thời, tạo thức ăn đầu tiên bằng hàm `generateFood`.
- Thiết lập các biến điều khiển vòng lặp game như `isRunning`, `gameOver`, thời gian cập nhật, điểm số, v.v.
- Vòng lặp chính của game xử lý:
  - Nhận và xử lý sự kiện bàn phím (di chuyển rắn, thoát game, khởi động lại, v.v.).
  - Cập nhật trạng thái game theo thời gian:
    - Di chuyển rắn.
    - Kiểm tra va chạm với tường hoặc chính nó.
    - Kiểm tra va chạm với thức ăn, cập nhật điểm số, tăng tốc độ nếu cần.
  - Vẽ lại toàn bộ màn hình: nền, lưới, tường (nếu có), rắn, thức ăn, và các thông báo như `Game Over`.
- Khi kết thúc, giải phóng tài nguyên đã cấp phát (renderer và window), và gọi `SDL_Quit()` để đóng SDL một cách an toàn.

5.4 Cách chơi trò chơi

5.4.1 Cài đặt và biên dịch

Để chơi trò chơi, trước tiên cần biên dịch mã nguồn bằng trình biên dịch `g++` và thư viện SDL3. Sau khi đã cài đặt đầy đủ SDL3, tiến hành theo các bước sau:

- Mở terminal và điều hướng đến thư mục chứa mã nguồn.
- Dùng lệnh sau để biên dịch chương trình:

```
g++ -o snake main.cpp -I. -L. -lSDL3
```

- Sau khi biên dịch thành công, ta sẽ có tệp thực thi .\snake.exe

5.4.2 Cách chơi

- Thực thi chương trình bằng lệnh:

```
.\snake.exe
```

- Khi chạy chương trình, một cửa sổ game sẽ hiện lên, đồng thời sẽ có 3 chế độ khó khác nhau ta có thể chọn, đó là EASY, MEDIUM, HARD. Trong đó bấm phím 1,2,3 sẽ chọn tương ứng độ khó từ dễ đến khó.
- Sau khi chọn độ khó, trò chơi bắt đầu. Rắn sẽ di chuyển theo hướng đã chọn, người chơi điều khiển để ăn thức ăn và tránh va vào tường hoặc chính thân rắn.
- Tiếp theo để chơi trò chơi:
  - Di chuyển lên: `W hoặc` ↑
  - Di chuyển xuống: `S hoặc` ↓
  - Di chuyển sang trái: `A hoặc` ←
  - Di chuyển sang phải: `D hoặc` →
  - Thoát game: `ESC`
  - Chơi lại sau khi thua: `Enter`
- Nếu va chạm xảy ra, màn hình sẽ hiển thị thông báo `Game Over`. Người chơi có thể nhấn `Enter` để chơi lại hoặc `ESC` để thoát.

6 Tổng kết

Qua quá trình xây dựng trò chơi **Snake** bằng thư viện **SDL3**, em đã có cơ hội áp dụng nhiều kiến thức lập trình đã học, đặc biệt là về *lập trình hướng đối tượng*, *quản lý tài nguyên*, *xử lý sự kiện*, và *vẽ đồ họa 2D thời gian thực*.

Việc triển khai các mức độ khó khác nhau, logic tăng tốc độ theo thời gian, và xử lý va chạm chính xác đã giúp em hiểu rõ hơn về cách tổ chức chương trình game một cách linh hoạt và dễ mở rộng.

Bên cạnh đó, việc sử dụng thư viện SDL không chỉ giúp tăng hiệu suất vẽ hình mà còn mang đến cho em kỹ năng làm việc với API đồ họa nền tảng thấp. Dự án cũng giúp em rèn luyện khả năng *debug*, *tối ưu hiệu năng*, và viết mã có thể bảo trì được.

Dù trò chơi vẫn còn một số hạn chế như giao diện đơn giản và chưa có âm thanh, em tin rằng nền tảng hiện tại hoàn toàn có thể tiếp tục phát triển và cải tiến trong tương lai, chẳng hạn như:

- Thêm hiệu ứng âm thanh, nhạc nền.
- Lưu điểm cao.
- Chế độ chơi 2 người.
- Tùy biến skin hoặc bản đồ.

Dự án này không chỉ giúp em củng cố kiến thức, mà còn truyền cảm hứng để tiếp tục khám phá sâu hơn vào lĩnh vực phát triển game và lập trình đồ họa.

7 Tài liệu tham khảo

1. SDL Official Documentation. https://wiki.libsdl.org
2. Lazy Foo' Productions – SDL Tutorials. https://lazyfoo.net/tutorials/SDL/
3. Cplusplus.com – STL và cú pháp C++. https://www.cplusplus.com