**ĐẠI HỌC QUỐC GIA, TP. HỒ CHÍ MINH**

**TRƯỜNG ĐẠI HỌC BÁCH KHOA**

**KHOA KHOA HỌC VÀ KỸ THUẬT MÁY TÍNH**



**BÁO CÁO BÀI TẬP LỚN**

**LẬP TRÌNH NÂNG CAO (MỞ RỘNG)**

**(CO2039)**

**GVHD: Trương Tuấn Anh**

**Sinh viên thực hiện: Phan Trần Minh Đạt**

**MSSV: 2111025**

TP. Hồ Chí Minh, 12/2022

**MỤC LỤC**

[**1.** **LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG VỚI JAVA VÀ RUBY** 3](#_Toc137323867)

[**1.1.** **Lập trình hướng đối tượng (Object-Oriented Programming)** 3](#_Toc137323868)

[***1.1.1.*** ***Khái niệm*** 3](#_Toc137323869)

[***1.1.2.*** ***Các tính chất của Lập trình hướng đối tượng*** 3](#_Toc137323870)

[**1.2.** **Ngôn ngữ lập trình Java** 4](#_Toc137323871)

[***1.2.1.*** ***Giới thiệu*** 4](#_Toc137323872)

[***1.2.2.*** ***Các đặc điểm và thế mạnh của Java*** 4](#_Toc137323873)

[***1.2.3.*** ***Các hạn chế của Java*** 5](#_Toc137323874)

[**1.3.** **Ngôn ngữ lập trình Ruby** 6](#_Toc137323875)

[***1.3.1.*** ***Giới thiệu*** 6](#_Toc137323876)

[***1.3.2.*** ***Các đặc điểm và thế mạnh của Ruby*** 6](#_Toc137323877)

[***1.3.3.*** ***Các hạn chế của Ruby*** 7](#_Toc137323878)

[**1.4.** **So sánh ngôn ngữ lập trình Java và Ruby** 8](#_Toc137323879)

[***1.4.1.*** ***Giống nhau*** 8](#_Toc137323880)

[***1.4.2.*** ***Khác nhau*** 8](#_Toc137323881)

[**2.** **LẬP TRÌNH HÀM VỚI HASKELL VÀ SML** 9](#_Toc137323882)

[**2.1.** **Lập trình hàm (Functional Programming)** 9](#_Toc137323883)

[***2.1.1.*** ***Khái niệm*** 9](#_Toc137323884)

[***2.1.2.*** ***Các tính chất của Lập trình hàm*** 9](#_Toc137323885)

[**2.2.** **Ngôn ngữ lập trình Haskell** 9](#_Toc137323886)

[***2.2.1.*** ***Giới thiệu*** 9](#_Toc137323887)

[***2.2.2.*** ***Các đặc điểm và thế mạnh của Haskell*** 10](#_Toc137323888)

[***2.2.3.*** ***Các hạn chế của Haskell*** 11](#_Toc137323889)

[**2.3.** **Ngôn ngữ lập trình SML** 11](#_Toc137323890)

[***2.3.1.*** ***Giới thiệu*** 11](#_Toc137323891)

[***2.3.2.*** ***Các đặc điểm và thế mạnh của SML*** 12](#_Toc137323892)

[***2.3.3.*** ***Các hạn chế của SML*** 12](#_Toc137323893)

[**2.4.** **So sánh ngôn ngữ lập trình Haskell và SML** 13](#_Toc137323894)

[***2.4.1.*** ***Giống nhau*** 13](#_Toc137323895)

[***2.4.2.*** ***Khác nhau*** 13](#_Toc137323896)

[**3.** **DATA WRANGLING VÀ SMART CONTRACT** 15](#_Toc137323897)

[**3.1.** **Data wrangling** 15](#_Toc137323898)

[***3.1.1.*** ***Khái niệm*** 15](#_Toc137323899)

[***3.1.2.*** ***Quy trình, cách thức hoạt động*** 15](#_Toc137323900)

[***3.1.3.*** ***Vai trò*** 15](#_Toc137323901)

[**3.2.** **Smart contract** 16](#_Toc137323902)

[***3.2.1.*** ***Khái niệm*** 16](#_Toc137323903)

[***3.2.2.*** ***Quy trình, cách thức hoạt động*** 16](#_Toc137323904)

[***3.2.3.*** ***Ứng dụng*** 16](#_Toc137323905)

[**4.** **HIỆN THỰC GAME ĐƠN GIẢN BẰNG C++ VỚI LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG** 17](#_Toc137323906)

[**4.1.** **Giới thiệu game** 17](#_Toc137323907)

[**4.2.** **Luật chơi** 17](#_Toc137323908)

[***4.2.1.*** ***Giải thích từ ngữ*** 18](#_Toc137323909)

[***4.2.2.*** ***Thành phần bộ bài*** 18](#_Toc137323910)

[***4.2.3.*** ***Thiết lập ban đầu*** 18](#_Toc137323911)

[***4.2.4.*** ***Lượt chơi*** 18](#_Toc137323912)

[***4.2.5.*** ***Labyrinth*** 19](#_Toc137323913)

[***4.2.6.*** ***Rút bài*** 19](#_Toc137323914)

[***4.2.7.*** ***Prophecy*** 19](#_Toc137323915)

[***4.2.8.*** ***Incantation*** 20](#_Toc137323916)

[***4.2.9.*** ***Ác mộng*** 20](#_Toc137323917)

[***4.2.10.*** ***Limbo*** 20](#_Toc137323918)

[***4.2.11.*** ***Xào bài*** 20](#_Toc137323919)

[***4.2.12.*** ***Kết thúc trò chơi*** 20](#_Toc137323920)

[**4.3.** **Cấu trúc chương trình** 21](#_Toc137323921)

[**4.4.** **Mô tả giao diện game** 21](#_Toc137323922)

[**TÀI LIỆU THAM KHẢO** 23](#_Toc137323923)

1. **LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG VỚI JAVA VÀ RUBY**
   1. **Lập trình hướng đối tượng (Object-Oriented Programming)**
      1. ***Khái niệm***

**- Lập trình hướng đối tượng:** là một mẫu hình lập trình dựa trên khái niệm “đối tượng” mà trong đó, đối tượng chứa đựng các dữ liệu thông qua các trường, thường được gọi là các *thuộc tính*; và mã nguồn, được tổ chức thành các *phương thức*.

- **Lớp:** là một kiểu dữ liệu bao gồm các thuộc tính và các phương thức được định nghĩa từ trước. Đây là sự trừu tượng hóa của đối tượng. Khác với kiểu dữ liệu thông thường, một lớp là một đơn vị trừu tượng bao gồm sự kết hợp giữa các phương thức và các thuộc tính.

- **Đối tượng:** là một đơn vị đầy đủ kết hợp các khuôn mẫu dữ liệu và chỉ thị được tạo ra bởi một lớp. Đơn vị này tương đương với một chương trình con và vì thế, các đối tượng sẽ được chia thành hai bộ phận chính: gồm các phương thức (*method*) và các thuộc tính (*attribute*/*properties*).

* + 1. ***Các tính chất của Lập trình hướng đối tượng***

- **Tính trừu tượng (Abstraction):** Tính chất này cho phép chương trình bỏ qua hay không chú ý đến một số khía cạnh của thông tin mà nó đang trực tiếp làm việc trên, nghĩa là nó có khả năng tập trung vào những cốt lõi cần thiết. Mỗi đối tượng phục vụ như là một “động tử” có thể hoàn tất các công việc một cách nội bộ, báo cáo, thay đổi trạng thái của nó và liên lạc với các đối tượng khác mà không cần cho biết làm cách nào đối tượng tiến hành được các thao tác.

- **Tính bao đóng (Encapsulation):** Tính chất này không cho phép người sử dụng các đối tượng thay đổi trạng thái nội tại của một đối tượng. Chỉ có các phương thức nội tại của đối tượng cho phép thay đổi trạng thái của nó. Việc cho phép môi trường bên ngoài tác động lên các dữ liệu nội tại của một đối tượng theo cách nào là hoàn toàn tùy thuộc vào người viết mã thông qua các *chỉ định truy cập* (access modifier). Đây là tính chất đảm bảo sự toàn vẹn của đối tượng.

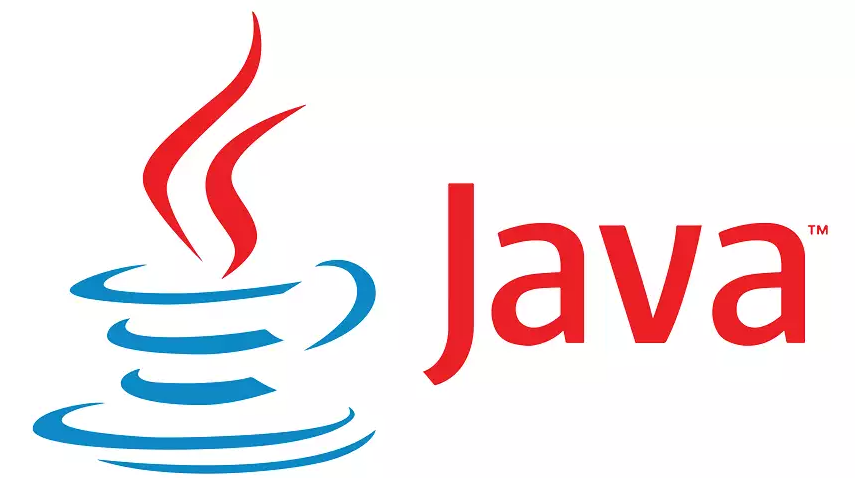
- **Tính đa hình (Polymorphism):** Tính chất này cho phép người lập trình có thể định nghĩa một đặc tính (chẳng hạn thông qua tên của các phương thức) cho một loạt các đối tượng gần nhau, nhưng khi thi hành thì dùng cùng một tên gọi mà sự thi hành của mỗi đối tượng sẽ tự động xảy ra tương ứng với đặc tính của từng đối tượng mà không bị nhầm lẫn.

- **Tính kế thừa (Inheritance):** Đặc tính này cho phép một đối tượng có thể có sẵn các đặc tính mà đối tượng khác đã có thông qua kế thừa. Điều này cho phép các đối tượng chia sẻ hay mở rộng các đặc tính sẵn có mà không phải tiến hành định nghĩa lại. Tuy nhiên, không phải ngôn ngữ định hướng đối tượng nào cũng có tính chất này.

* 1. **Ngôn ngữ lập trình Java**
     1. ***Giới thiệu***

**Java** là một ngôn ngữ lập trình *hướng đối tượng*, dựa trên lớp được thiết kế để có càng ít phụ thuộc vào thực thi càng tốt. Đây là ngôn ngữ lập trình có mục đích là cho phép các nhà phát triển ứng dụng “viết một lần, chạy ở mọi nơi” (WORA), nghĩa là mã Java đã biên dịch có thể chạy trên tất cả các nền tảng hỗ trợ Java mà không cần biên dịch lại.

Cú pháp của Java tương tự như C và C++, nhưng có ít cơ sở cấp thấp hơn các ngôn ngữ trên. Java Runtime cung cấp các khả năng động (chẳng hạn như phản ánh và sửa đổi mã thời gian chạy) thường không có sẵn trong các ngôn ngữ biên dịch truyền thống. Tính đến năm 2019, Java là một trong những ngôn ngữ lập trình phổ biến nhất được sử dụng theo GitHub, đặc biệt cho các ứng dụng web máy khách-máy chủ (server-client), với khoảng 9 triệu nhà phát triển đã được báo cáo.



* + 1. ***Các đặc điểm và thế mạnh của Java***

- **Simple (Đơn giản):** Java là một ngôn ngữ khá dễ học với cấu trúc ngữ nghĩa đơn giản hơn các ngôn ngữ tiền nhiệm như C hay C++. Nhờ vậy, lập trình viên có thể dễ dàng hiện thực chương trình hơn mà không phải quá lo về các lỗi cú pháp, chẳng hạn như các lỗi liên quan đến việc quản lý bộ nhớ. Một ví dụ điển hình là khái niệm con trỏ, vốn là đặc sản của C++, đã được lược bỏ ở Java, khiến cho code đọc dễ hiểu và “sạch” hơn.

- **Robust (Mạnh mẽ):** Java không đi sâu vào việc quản lý bộ nhớ như C và C++ nhờ có cơ chế *Garbage Collector*, giúp trình biên dịch có thể tự động làm sách bộ nhớ dựa theo vòng đời của từng đối tượng. Ngoài ra, Java còn có cơ chế *Automatic* *Exception Handling*, giúp xử lý một số ngoại lệ mà người dùng chưa lập trình sẵn.

- **Portable (Di động):** Một chương trình được viết bằng ngôn ngữ lập trình Java sẽ được biên dịch ra mã của máy ảo Java (Java Virtual Machine – JVM), gọi là mã Java bytecode. Sau đó, máy ảo Java sẽ chịu trách nhiệm thông dịch mã Java bytecode mã máy tương ứng. Nhờ vậy, một chương trình viết bằng Java có thể chạy được trên bất kỳ máy tính hay hệ điều hành nào có cài đặt máy ảo Java.

- **Interpreted (Thông dịch):** Trình thông dịch Java là một phần mềm dùng để thực thi JVM, đọc Java bytecode và chạy chương trình. Tuy nhiên, tốc độ xử lý của Java cao hơn các ngôn ngữ thông dịch khác (ví dụ như Python) bởi có cơ chế *Just-In-Time compiler* (trình biên dịch runtime, nghĩa là biên dịch trong khi chương trình đang thực thi) giúp biên dịch bytecode sang mã máy. Nói cách khác, thay vì phải thông dịch từng dòng rồi chạy lần lượt, JIT biên dịch cả chương trình trong runtime.

- **Secured (Bảo mật):** Java là ngôn ngữ được thiết kế để tối ưu hóa việc bảo mật. Java hỗ trợ một lượng lớn các API, công cụ bảo mật với độ tin cậy cao. Bên cạnh đó, chương trình Java luôn chạy trong JRE và gần như không tương tác với hệ điều hành bên ngoài, nhờ vậy giảm khả năng tiếp xúc với virus.

- **Multithreading (Đa luồng):** Java cho phép lập trình đa luồng, giúp tận dụng tối đa nguồn tài nguyên sẵn có để giúp chương trình chạy nhiều tác vụ một cách đồng thời.

- **Architectural Neutral (Trung lập với kiến trúc):** Không giống như C và C++ với cơ chế biên dịch riêng cho từng platform, Java đảm bảo tính năng “viết một lần, chạy ở mọi nơi”. Mã Java Bytecode sau khi được tạo có thể chạy ở bất kì máy nào có JRE (Java Runtime Environment) với bất kì hệ điều hành.

- **Object-Oriented (Hướng đối tượng):** Đây là một đặc điểm hết sức quan trọng của Java. Trong ngôn ngữ này, hầu hết mọi thứ đều là đối tượng thuộc một class nào đó (ngoại trừ một số ít kiểu nguyên thủy), bên trong chứa các thuộc tính và phương thức. Java hỗ trợ đầy đủ từ khái niệm cơ bản đến các nguyên tắc của lập trình hướng đối tượng.

- **Dynamic (Động):** Với tính chất thừa kế và trừu tượng của lập trình hướng đối tượng, nhà phát triển có thể tái sử dụng những đoạn code đã được lập trình sẵn, cũng như sử dụng các thư viện hàm của Java.

* + 1. ***Các hạn chế của Java***

- **Hiệu năng thấp:** Tuy nhanh hơn so với các ngôn ngữ thông dịch khác, Java lại có tốc độ chậm hơn so với các ngôn ngữ biên dịch, do tính chất của một ngôn ngữ thông dịch.

- **Chi phí:** Bởi Java cần bộ xử lý và bộ nhớ mạnh, khi lập trình một ứng dụng phức tạp, người ta thường chuyển sang các ngôn ngữ khác tiện lợi hơn.

- **Giao diện không bắt mắt:** Mặc dù có các GUIs và thư viện hỗ trợ như Swing, JavaFX, SWT, giao diện của Java nhìn khá lỗi thời và phải cần nghiên cứu lâu mới có thể thành thục một công cụ UI trong Java.

- **Không có cơ chế backup:** Java chủ yếu làm việc trên kho lưu trữ và không quan tâm đến việc backup dữ liệu ở một nơi nào đó. Đây có thể xem là điểm yếu lớn nhất của Java.

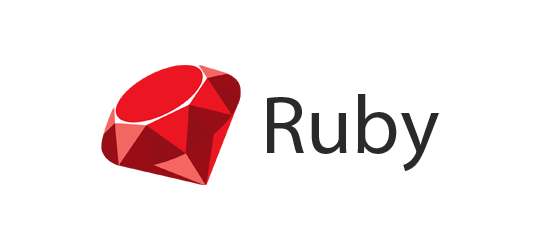
- **Tốn nhiều không gian bộ nhớ:** Do phải chạy trên máy ảo, Java sử dụng lượng bộ nhớ khá lớn so với C và C++. Trong quá trình Garbage Collection, hiệu năng của bộ nhớ sẽ bị ảnh hưởng.

- **Câu lệnh phức tạp:** Tuy đã tinh giản và dễ hiễu hơn C/C++, Java còn khó hiểu và cấu trúc câu lệnh còn khá dài dòng so với một số ngôn ngữ lập trình được ưa chuộng hiện nay. Ví dụ như tác vụ in trên console, trong khi Python chỉ cần lệnh print(), thì ở Java, ta phải gõ System.out.println().

* 1. **Ngôn ngữ lập trình Ruby**
     1. ***Giới thiệu***

**Ruby** là một ngôn ngữ lập trình hướng đối tượng (OOP) được phát triển bởi Yukihiro “Maz” Matsumoto tại Nhật Bản vào giữa thập niên 90. Với chủ trương thiết kế một ngôn ngữ giúp lập trình viên có thể tăng hiệu suất công việc và sự hứng thú, Ruby nhấn mạnh tầm quan trọng của việc một phần mềm phải tường minh dưới góc độ con người trước, sau đó mới là máy tính.

Bất kỳ loại phần mềm nào cũng có thể được tạo ra với sự trợ giúp của Ruby, từ web đến các ứng dụng di động. Trong Ruby, mọi giá trị, thậm chí là class, đều là đối tượng. Ngôn ngữ Ruby được thiết kế theo nguyên tắc ít gây ngạc nhiên nhất (principle of least surprise-POLS), nghĩa là ngôn ngữ hoạt động theo một cách trực quan hay ít nhất đó cũng là nhận xét mà các nhà lập trình đưa ra.



* + 1. ***Các đặc điểm và thế mạnh của Ruby***

- **Open source (Mã nguồn mở):** Ruby hỗ trợ Ruby License, cung cấp mã nguồn của Ruby miễn phí trên GitHub. Đồng thời người dùng có thể đóng góp xây dựng các phiên bản sau.

- **Interpreted (Thông dịch):** Ruby là một ngôn ngữ thông dịch. Do đó, mã nguồn không cần biên dịch mà vẫn có thể chạy ngay.

- **Dynamic + duck typing (Động):** Ruby là ngôn ngữ động, tức là kiểu dữ liệu các biến chỉ được xét tại runtime. Các biến có thể thay đổi kiểu trong runtime, ví dụ nhưng biến a đang ở kiểu string có thể chuyển sang kiểu integer mà trình thông dịch không báo lỗi. Nhờ vậy, chương trình sẽ chạy nhanh hơn và việc viết mã cũng diễn ra dễ dàng.

- **Garbage Collector (Bộ dọn rác):** Ruby hỗ trợ cơ chế dọn bộ nhớ của đối tượng khi không cần thiết, cùng với các giải thuật thông minh được Ruby sử dụng để tối ưu hóa vùng nhớ.

- **Pure** **Object-Oriented (Hướng đối tượng):** Ruby là một ngôn ngữ lập trình hướng đối tượng thuần túy, hỗ trợ các khái niệm của OOP như class, thừa kế, trừu tượng, đa hình và đóng gói. Các đặc trưng của OOP giúp việc bảo trì và phát triển code dễ dàng hơn.

- **Embedding (Nhúng):** Ruby là một trong số các ngôn ngữ có thể nhúng vào HTML (eRuby), tương tự các ngôn ngữ script server-side như PHP, có ích trong triển khai mô hình MVC (Model-View-Controller).

- **Applicable (Tính ứng dụng):** Ruby có thể dùng để lập trình CGI script (Common Gateway Interface, giao thức tương tác giữa web server và client), xây dựng website và các ứng dụng Intranet.

- **Supportive (Tính hỗ trợ):** Ruby hỗ trợ các công cụ GUI khác nhau như OpenGL, Tcl/Tk, GTK, ... Bên cạnh đó, ngôn ngữ này cũng hỗ trợ tương tác kết nối với một số database như MySQL, DB2 và kiến trúc serverless như AWS Lambda.

* + 1. ***Các hạn chế của Ruby***

- **Popularity (Độ phổ biến):** Ruby vẫn là một ngôn ngữ chưa có độ phổ biến cao, dù vẫn nằm trong top 10. Do đó, Ruby hay Ruby on Rails thường không phải là lựa chọn đầu tiên của các lập trình viên. Thay vào đó, ta dễ tìm một người thạo ngôn ngữ Python hay Java hơn.

- **Slow runtime and Scalability (Tốc độ thực thi chậm và Khả năng mở rộng):** Đây là điểm yếu lớn nhất của Ruby. Vì là ngôn ngữ dùng trình thông dịch, Ruby chậm hơn các ngôn ngữ biên dịch như C/C++ và ngay cả Java. Do đó khi mở rộng đến lượng người dùng lớn hơn, vấn đề từ server hay database sẽ cản trở hiệu suất của chương trình. Còn xét nền tảng framework Ruby on Rails, nó chậm hơn các backend framework khác như NodeJS hay Go.

- **Flexibility (Độ linh hoạt):** bản chất của Ruby là xây dựng ứng dụng web, vì vậy nếu sản phẩm được ứng dụng cho những tác vụ khác thì việc điều chỉnh có phần phức tạp.

- **Development (Phát triển):** so với các ngôn ngữ lập trình khác, Ruby tung bản cập nhật ít và chậm hơn. Ví dụ như PHP, dù ra đời trễ hơn 2 năm, hiện có phiên bản 8.2 (08/12/2022), trong khi Ruby vẫn ở bản 3.2.2 (30/03/2023).

* 1. **So sánh ngôn ngữ lập trình Java và Ruby**
     1. ***Giống nhau***

- **Garbage Collection:** Khác với C/C++ phải quản lý bộ nhớ một cách thủ công, cả Java và Ruby (và một số ngôn ngữ cấp cao khác) đều hỗ trợ cơ chế dọn bộ nhớ tự động. Mặc dù mỗi ngôn ngữ có cơ chế xử lý vùng nhớ khác nhau, chúng đều bảo đảm an toàn cho vùng nhớ (không bao giờ xảy ra trường hợp xóa vùng nhớ có thể được sử dụng) cũng như tối ưu phần bộ nhớ trống.

- **Multithreading:** Cả hai ngôn ngữ đều có các thư viện hỗ trợ lập trình đa luồng, xử lí song song các tác vụ. Trong Java, đa luồng được thể hiện rõ bằng việc tận dụng tất cả nguồn tài nguyên CPU. Dù Ruby không tận dụng tốt như vậy, nó vẫn hỗ trợ chạy chương trình theo từng thread.

- **Security:** Java và Ruby được hầu hết các lập trình viên công nhận là có tính bảo mật cao. Không có khái niệm con trỏ, người dùng bên ngoài không thể nào tùy tiện xâm nhập vùng nhớ được khai báo private . Ngoài ra, cả hai đều có các thư viện hỗ trợ việc bảo mật rất tốt.

* + 1. ***Khác nhau***

|  |  |
| --- | --- |
| **Java** | **Ruby** |
| - Vừa thông dịch, vừa biên dịch  - Cơ chế biên dịch JIT | - Thông dịch |
| - Sử dụng *static typing*, tức là kiểu của biến đã được biên dịch từ trước và không thể thay đổi trong thời gian chạy. | - Sử dụng *dynamic typing*, tức là không cần phải định nghĩa kiểu biến, và kiểu của một biến có thể thay đổi nhiều lần trong suốt chương trình. |
| - Java có 8 kiểu dữ liệu nguyên thủy, là một *Hybrid object-oriented programming language*. | - Tất cả các dữ liệu trong Ruby đều là các đối tượng, là một *Pure object-oriented programming language*. |
| - Java có hỗ trợ *Nạp chồng phương thức* (Method overloading) nhưng không cho phép *Nạp chồng toán tử* (Operator overloading). | - Ruby không hỗ trợ *Nạp chồng phương thức* nhưng cho phép *Nạp chồng toán tử*. |

1. **LẬP TRÌNH HÀM VỚI HASKELL VÀ SML**
   1. **Lập trình hàm (Functional Programming)**
      1. ***Khái niệm***

- **Lập trình hàm:** là một mẫu hình lập trình mà chương trình được xây dựng dựa trên việc sử dụng và tạo các hàm, tức lấy các hàm là đơn vị thao tác cơ bản.

- Khác với lập trình thủ tục nhận mạnh vào sự thay đổi trạng thái, lập trình hàm nhấn mạnh vào việc ứng dụng các phép tính lambda, hàm số và đệ quy.

* + 1. ***Các tính chất của Lập trình hàm***

- **Pure function (Hàm thuần túy):** Lập trình hàm sử dụng *hàm thuần túy*, thường có tính chất như các hàm toán học, và trả về kết quả độc lập với các biến có tầm vực rộng hơn.

- **Immutable data (Dữ liệu bất biến):** Những biến đã được khai báo một lần thì sẽ không bao giờ thay đổi giá trị đã được khởi gán. *Immutable* là một trong các đặc trưng cơ bản nhất của lập trình hàm, và thường được đảm bảo bằng đệ quy.

- **Higher-order function (Hàm bậc cao):** tương tự với khái niệm hàm của toán học, *higher-order function* là một hàm nhận đầu vào là một hàm khác hoặc trả về kết quả là một hàm. Trong lập trình hàm, hầu như mọi hàm đều là hàm bậc cao, vì chúng đều có thể nhận vào và trả về các hàm. Điều này cho phép việc gọi hàm và trừu tượng hóa hành động diễn ra linh hoạt hơn.

- **Composition:** Mỗi hàm đại diện cho một thao tác và có thể ghép một chuỗi có thứ tự các hàm cơ bản thành hàm phức tạp hơn. Đây là khái niệm bắt nguồn từ hàm hợp của Toán học. Composition đơn thuần chỉ là sự lồng ghép các hàm với nhau, tạo thành một hàm lớn với chức năng phức tạp.

* 1. **Ngôn ngữ lập trình Haskell**
     1. ***Giới thiệu***

**Haskell** là ngôn ngữ lập trình hàm thuần túy (Pure Functional Programming), là một ngôn ngữ biên dịch và có kiểu tĩnh, tức là các kiểu dữ liệu được khai báo trong mã nguồn tại thời điểm dịch. Tuy nhiên, trình biên dịch có thể hỗ trợ người lập trình trong việc tự động suy kiểu.

Haskell có tính lười biếng (lazy evaluation), nghĩa là trừ khi buộc phải làm, không thì Haskell sẽ không bao giờ thực thi tính toán. Do tính chất thiên về toán học và đệ quy, kết quả của một hàm Haskell thường khó đoán hơn so với các ngôn ngữ lập trình thủ tục, cũng như cần nắm vững các nền tảng toán học để có thể hiện thực thành công một hàm.



* + 1. ***Các đặc điểm và thế mạnh của Haskell***

- **Open-source (Mã nguồn mở):** Haskell được xây dựng để tất cả những ai có đam mê về lập trình hàm đều có thể đóng góp và sử dụng mà không cần tốn chi phí. Nhờ vậy, số lượng người dùng Haskell tương đối đông đảo và tạo tiềm năng phát triển mạnh.

- **Quản lý bộ nhớ:** Không giống như các chương trình C/C++, nơi mà người dùng phải tự dọn dẹp vùng nhớ heap sau khi khai báo, Haskell có cơ chế *Garbage Collector* giúp tự động phân bố tối ưu vùng nhớ.

- **Tái sử dụng:** Do các hàm trong Haskell mang tính chất Purity, tức là các hàm thuần túy, chúng có thể được dùng lại cho các tác vụ khác nhau với cùng mục đích mà không lo bị ảnh hưởng bởi những hiệu ứng phụ (như thay đổi trạng thái chương trình thông qua các biến toàn cục hay biến tĩnh …). Bên cạnh đó, Haskell còn cho phép lồng ghép các hàm nhỏ lẻ thành hàm phức tạp hơn và có thể tái sử dụng các hàm trong một module để áp dụng vào một module khác bằng việc export các định nghĩa hàm và import khi cần.

- **Static typing:** Kiểu dữ liệu của các biến trong Haskell được xác định tường minh ở compile-time, nhờ vậy trình biên dịch có thể phát hiện lỗi sai và giảm tính phức tạp khi mở rộng quy mô lập trình. Đồng thời, khi chương trình Haskell đã được biên dịch thành công, nó không cần phải lo lắng về kiểu của các biến, do đó tăng hiệu năng ở runtime.

- **Lazy evaluation:** ở chế độ mặc định, Haskell dùng chiến thuật thực thi code mang tên lazy evaluation. Cụ thể, một biểu thức sẽ không bao giờ được định lượng cho đến khi nó buộc phải được dùng định lượng một biểu thức lớn hơn khác. Sau khi đã được định lượng, kết quả sẽ được lưu trữ hoặc xóa bỏ nếu chương trình không còn chạy. Nhờ vậy, chỉ các thao tác thực sự cần thiết mới được thực thi, giúp giảm đáng kể khối lượng tính toán.

- **Thuần lập trình hàm:** tất cả những thao tác tính toán được quy về đơn vị là các hàm số trong toán học (first-order), nhờ đó mà sẽ không có side effect (một đầu vào chỉ có thể cho ra một kết quả mà không bị tác động bên ngoài). Nhờ vậy, người dùng dễ phát hiện lỗi sai trong chương trình, đồng thời tạo tính thống nhất cho hàm.

- **Multhreading (Lập trình đa luồng):** Việc đánh giá các hàm thuần túy không phụ thuộc vào thứ tự hoàn tất, nhờ vậy ta có thể tối ưu hiệu năng lập trình bằng xử lý đa luồng mà không cần lo lắng gặp phải các vấn đề về concurrency.

- **Type Inference (Suy luận kiểu):** Đây là một cách thực thi cho phương pháp Implicit Typing, tức thông tin về kiểu biến không được cung cấp trong mã nguồn. Với Type Inference, các kiểu biến được cấu trúc dựa trên cú pháp của giá trị của nó.

* + 1. ***Các hạn chế của Haskell***

**- Tính phức tạp:** cộng đồng xây dựng là các nhà toán học, do đó mã và các kí hiệu có thể trở nên quá trừu tượng đối với dân không chuyên hoặc nhập môn. Lập trình viên khi học Haskell phải có kiến thức vững về lập trình, ngay cả là lập trình hướng đối tượng.

**- Tính rộng rãi:** Chưa có quá nhiều thư viện hỗ trợ Haskell hay các tài liệu hướng dẫn, và cộng đồng người dùng ngôn ngữ này ở thời điểm hiện tại còn khá nhỏ so với các ngôn ngữ lập trình khác như C++, Java hay Python. Do vậy, người dùng thường cảm thấy nản chí khi học ngôn ngữ này.

**- Debug (Gỡ lỗi):** Cơ chế *lazy evaluation* đôi khi sẽ bỏ qua một số đoạn mã, do đó việc gỡ lỗi với ngôn ngữ này thường khá khó khắn. Bên cạnh đó, việc không có *stack traces* cho Haskell cũng khiến việc dò tìm lỗi sai càng khó khăn hơn.

**- Hiệu năng:** Các chương trình được viết bằng ngôn ngữ lập trình Haskell thường có hiệu năng thấp hơn so với nhiều ngôn ngữ lập trình khác.

* 1. **Ngôn ngữ lập trình SML**
     1. ***Giới thiệu***

Standard ML là một trong các ngôn ngữ lập trình họ ML, bắt nguồn từ Meta Language của định lý LCF, phát triển bởi Robin Milner và cộng sự vào thập niên 70 với mục đích xây dựng một ngôn ngữ có logic chặt chẽ. Đây là một ngôn ngữ lập trình hàm không thuần túy, do SML cho phép side-effect xảy ra.

Về nguyên tắc, Standard ML nhấn mạnh tính an toàn trong lập trình, bao gồm tính chặt chẽ trong suy diễn kiểu và quản lý bộ nhớ. Đồng thời, SML có các thư viện và cơ chế biên dịch, I/O phức tạp nhằm tối ưu hiệu suất khi mở rộng quy mô ứng dụng.



* + 1. ***Các đặc điểm và thế mạnh của SML***

- **Mostly functional:** SML dựa trên mô hình đánh giá các hàm là đơn vị tương tác nhỏ nhất (first-order) và từ các hàm đơn lẻ ta có thể xây dựng một hàm phức tạp hơn (higher-order). Hầu như các biến và cấu trúc dữ liệu trong ngôn ngữ đều không thể thay đổi sau khi khởi gán. Ngoài ra, khác với ngôn ngữ thuần lập trình hàm như Haskell, SML cho phép side-effect xảy ra.

- **Strict:** tất cả các tham số đều được định giá trước khi đi vào thân hàm, giúp ta phát hiện dễ phát hiện lỗi sai và kiểm soát sự tiêu tốn bộ nhớ của toàn bộ dữ liệu.

- **Modular:** SML hỗ trợ các module và bản thiết kế (tương tự như khái niệm *class* và *interface* trong Java, ngoại trừ việc tất cả các thuộc tính đều có thể tự do truy cập từ bên ngoài), tạo ra các cấu trúc linh hoạt và phức tạp. Mỗi module có một hay nhiều bản thiết kế khác nhau. Ngoài ra, functor (một module nhận bản thiết kế của module khác là tham số) được kiểm tra trong compile-time, đảm bảo tính module hóa tốt hơn.

- **Strong and Static Typing:** Mỗi câu lệnh trong SML được gán một kiểu và kiểm tra tính hợp lệ trong compile-time để đảm bảo rằng chỉ những thao tác thích hợp được cho phép. Quá trình này hỗ trợ việc dò lỗi dễ dàng và theo dõi sự cập nhật mã tốt hơn.

- **Complex and Abstract Types (Kiểu trừu tượng và phức tạp):** Bên cạnh các kiểu dữ liệu cơ bản như số nguyên, số thực, chuỗi, tuple, list,... ta có thể định nghĩa các đối tượng dữ liệu phức tạp không có sẵn và che giấu cách hiện thực bên trong.

- **Type inference and consistency (Suy diễn kiểu và tính nhất quán):** SML không yêu cầu lập trình viên ghi rõ các kiểu của biến và tham số hàm, vì trình biên dịch có thể suy diễn được kiểu dữ liệu. Tuy nhiên cần lưu ý rằng, SML không hỗ trợ chuyển đổi kiểu dữ liệu, ngay cả khi đó là phép tính cộng giữa số nguyên và số thực.

- **Polymorphism (Đa hình):** Mỗi hàm đều hoạt động theo cơ chế đa hình, do đó nó có thể hoạt động với nhiều kiểu dữ liệu khác nhau nếu như các toán tử trong hàm hợp lệ với kiểu dữ liệu đó. Ngoài ra, SML cũng hỗ trợ đa hình theo kiểu biến, chằng hạn như một biến kiểu list có thể chứa list dạng số nguyên hay list dạng string.

- **Exception Handler (Bộ xử lí ngoại lệ):** Standard ML có cơ chế xử lý ngoại lệ dễ dàng và đa dạng. Ta có thể ném ra một ngoại lệ với bất kì giá trị nào, kể cả hàm.

* + 1. ***Các hạn chế của SML***

- **Kiểu dữ liệu:** Như đã nói, tất cả kiểu dữ liệu trong SML không được chuyển đổi tự dộng, kể cả là từ kiểu số nguyên sang số thực. Điều này sẽ gây khó chịu và phức tạp cho người dùng vì ta cần phải nhiều thao tác hơn cho cùng một tác vụ so với các ngôn ngữ khác. Chưa kể, kiểu dữ liệu được nới lỏng hơn so với các ngôn ngữ như Haskell (Type inference), dễ gây ra lỗi trong runtime.

- **Recompile (Tái biên dịch):** Khác với C hay Java chỉ cần compile một lần, SML không hỗ trợ tốt quá trình biên dịch. Mỗi lần chạy SML khi có cập nhật ta hầu như phải tái biên dịch toàn bộ chương trình, do đó mất nhiều thời gian hơn.

- **Support (Hỗ trợ):** tài liệu hỗ trợ sử dụng Standard ML khá ít, số người sử dụng ngôn ngữ này cũng ở một con số khiêm tốn (chủ yếu là vì mục đích nghiên cứu). Ngoài ra, tính tương thích với một số hệ điều hành không cao, các thư viện bên thứ 3 hỗ trợ chưa thật sự mạnh mẽ, do đó hầu hết các tính năng phải lập trình từ nền tảng, mất thời gian và chi phí vận hành.

* 1. **So sánh ngôn ngữ lập trình Haskell và SML**
     1. ***Giống nhau***

- **Polymorphism and Type inference:** Haskell và SML đều có hệ thống suy diễn kiểu dựa trên định lý toán học Hilney-Milner, do đó chúng khá hoàn chỉnh. Đây cũng là cơ sở để xây dựng các kiểu dữ liệu phức tạp hơn và tăng tính linh động.

- **Pattern Matching:** cả hai ngôn ngữ đều cho phép xây dựng hàm dựa trên một khuôn mẫu có sẵn (tính đa hình của datatype).

- **Cú pháp mang tính chất toán học:** do đều được xây dựng là các ngôn ngữ lập trình hàm, các hàm trong Haskell và Standard ML đều khá giống với hàm số trong toán học, dù Haskell có phần nhỉnh hơn về độ tương tự so với SML (bù lại là các cú pháp phức tạp).

- **Syntactic Sugar and Wildcard:** Cả hai ngôn ngữ đều cố gắng tạo ra các cú pháp giúp cho người dùng dễ đọc hiểu, với thiên hướng là sự kết hợp của ngôn ngữ tự nhiên và toán học (cô đọng, súc tích). Ngoài ra, *wildcard* giúp người lập trình bỏ qua một số giá trị ít quan trọng và dễ gỡ lỗi hơn.

* + 1. ***Khác nhau***

|  |  |
| --- | --- |
| **Haskell** | **SML** |
| - Sử dụng cơ chế *lazy evaluation*, chỉ tính toán các giá trị cần thiết khi mà hàm bắt buộc phải sử dụng các tham số đó. | - Sử dụng cơ chế *strict evaluation*, nghĩa là các tham số trong một hàm phải được định giá trước khi gọi, ngay cả khi hàm đó có được gọi trong suốt quá trình thực thi hay không. |
| - Thuần lập trình hàm, do đó bảo đảm không có side-effect. | - Không thuần lập trình hàm: trong một số trường hợp, Standard ML cho phép side-effect xảy ra. |
| - Module trong Haskell định nghĩa một tập hợp các giá trị, kiểu dữ liệu và mối quan hệ với các module khác. | - Signature và Structure trong SML xử lý cơ chế quản lý vùng dữ liệu cho kiểu dữ liệu phức tạp, bao gồm các kiểu dữ liệu cơ bản hay structure con bên trong, có cơ chế tương tự interface trong Java. |
| - Sử dụng monad để xử lý ngoại lệ, làm cho quá trình trở nên phức tạp hơn. | - Cho phép side-effect nên thao tác xử lí ngoại lệ diễn ra dễ dàng hơn. |

1. **DATA WRANGLING VÀ SMART CONTRACT**
   1. **Data wrangling**
      1. ***Khái niệm***

**Data Wrangling (Sắp xếp dữ liệu)** là quá trình chuyển đổi dữ liệu từ dạng thô sang dạng sẵn sàng để phân tích. Sắp xếp dữ liệu là một bước quan trọng trong tiền xử lý dữ liệu và bao gồm một số quy trình như nhập dữ liệu, làm sạch dữ liệu, cấu trúc dữ liệu, xử lý chuỗi, phân tích cú pháp HTML, xử lý ngày và giờ, xử lý dữ liệu bị thiếu và khai thác văn bản.

* + 1. ***Quy trình, cách thức hoạt động***

Một quá trình sắp xếp dữ liệu thường sẽ trải qua các công đoạn sau:

- **Data discovery:** Đây là bước đầu trong cả quá trình, bao gồm việc mô tả dữ liệu để hiểu rõ và làm quen với các thông tin mà bộ dữ liệu cung cấp.

- **Structuring:** Bước kế tiếp là tổ chức, tái cấu trúc dữ liệu. Đây là bước cấu trúc lại dữ liệu thô sao cho chúng được sắp xếp thành các dạng chuẩn, phân chia các trường dữ liệu rõ ràng nhằm tối ưu hóa quá trình phân tích.

- **Cleaning:** Hầu hết các dữ liệu thô đều chứa các số liệu lỗi, không xác định, hoặc không không phải dạng cần thiết trong quá trình phân tích (ví dụ như dữ liệu về ngày tháng).

- **Enriching:** Đây là một bước tự chọn. Ta sẽ xác định có cần thiết thêm thông tin mới và hữu ích cho bộ dữ liệu hay không, nhằm phục vụ cho công tác phân tích.

- **Validating:** Đây là một bước khá tương đồng với structuring và cleaning. Ở bước này, ta sẽ sử dụng một loạt công đoạn kiểm tra để đảm bảo tính nhất quán của dữ liệu, song song là chất lượng và tính bảo mật.

- **Publishing:** Chuẩn bị tập dữ liệu sẵn sàng, có thể thêm vào phần mềm hoặc các công đoạn phân tích kế tiếp. Bước này cũng bao gồm việc viết tài liệu hướng dẫn cách sử dụng bộ dữ liệu một cách chi tiết.

* + 1. ***Vai trò***

Quá trình sắp xếp dữ liệu là một bước quan trọng đối với bất kỳ nhà khoa học dữ liệu nào. Rất hiếm khi dữ liệu dễ dàng truy cập trong một dự án khoa học dữ liệu để phân tích. Nhiều khả năng dữ liệu nằm trong tập tin, trong cơ sở dữ liệu hoặc được trích xuất từ các tài liệu như trang web, mạng xã hội hoặc PDF. Biết cách sắp xếp và làm sạch dữ liệu sẽ cho phép bạn thu được thông tin chi tiết quan trọng từ dữ liệu của mình mà nếu không các dữ liệu đó sẽ bị ẩn.

* 1. **Smart contract**
     1. ***Khái niệm***

**Smart Contract (Hợp đồng thông minh)** là các chương trình chạy trên blockchain, tự động thực hiện, kiểm soát và ghi lại những sự kiện, hành động có liên quan đến nhau về mặt pháp lý dựa theo những điều khoản của hợp đồng hoặc thỏa thuận.

Hai bên trong hợp đồng có thể đưa ra những cam kết thông qua blockchain mà không cần phải biết về danh tính hay tin tưởng lẫn nhau mà vẫn đảm bảo nếu các điều kiện của hợp đồng không được thỏa mãn thì hợp đồng sẽ không được thực thi, đồng thời có thể tự động hóa quy trình, kích thích hành động tiếp theo nếu đáp ứng được các điều kiện.

* + 1. ***Quy trình, cách thức hoạt động***

Smart Contract hoạt động tuân theo các câu lệnh “nếu… thì…” nên sẽ thực thi một tác vụ cụ thể trong trường hợp thỏa mãn các điều kiện nhất định. Hợp đồng thông minh đề cập đến một thuật toán được thiết kế để hình thành, kiểm soát và cung cấp thông tin về chủ sở hữu nội dung, nó là một chương trình chạy trên blockchain để xác minh hoặc thực hiện các giao dịch đáng tin cậy một cách độc lập.

Trong mỗi Smart Contract bao gồm: Chữ ký giữa hai hoặc nhiều bên, chủ thể hợp đồng, các điều khoản. Tất cả các điều khoản này được mã hóa, khi hệ thống kiểm tra, xác minh các điều kiện đã hoàn thành thì hợp đồng sẽ chuyển sang bước tiếp theo.

Dựa vào những điều trên, nhà phát triển sẽ lập trình ra Smart Contract để doanh nghiệp và cá nhân sử dụng theo nhu cầu cụ thể. Chỉ bên được cấp quyền mới có thể xem kết quả và không thể tự ý thay đổi giao dịch trong Smart Contract.

* + 1. ***Ứng dụng***

Smart Contract được ứng dụng nhiều, đặc biệt là đối với lĩnh vực tiền điện tử. Bitcoin là đồng tiền điện tử đầu tiên đặt nền tảng cơ bản cho việc thiết lập Smart Contract trên Blockchain, tuy còn sơ khai. Phải đến khi Ethereum phát triển Smart Contract trở thành nghiệp vụ nền tảng chạy trên Ethereum một cách chính xác và không bị lỗi gián đoạn, giả mạo cũng như bị can thiệp bởi bên thứ ba thì loại hợp đồng này mới trở nên phổ biến.

Hợp đồng thông minh loại bỏ các bên giao dịch trung gian, khắc phục những bất cập về vấn đề bảo mật, pháp lý, gian lận, kiểm duyệt … tiết kiệm thời gian và nhân lực.

Bên cạnh đó,hợp đồng thông minh còn được ứng dụng vào các hoạt động của nhiều tổ chức như: Bảo vệ hiệu quả thuốc Sonoko và IBM, tăng cường mối quan hệ giữa nhà bán lẻ và nhà cung cấp của Home Depot, WE.Trade tổ chức các hoạt động thương mại quốc tế hiệu quả hơn.

1. **HIỆN THỰC GAME ĐƠN GIẢN BẰNG C++ VỚI LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG**
   1. **Giới thiệu game**

- Tên game: **Onirim**

- Thể loại: Board game, Chiến thuật

- Tác giả: Shadi Torbey, Philippe Guérin, Élise Plessis

- Tổng quan: Onirim là một trò chơi thẻ bài một người chơi, lấy bối cảnh là một vùng đất của những giấc mơ đầy huyền bí và cũng nhiều cạm bẫy. Nhiệm vụ của người chơi là khám phá ra tất cả cánh cửa thuộc về 4 khu vực khác nhau của thế giới giấc mơ và trở về thế giới thực trước khi các quyền năng cạn kiệt. Tuy nhiên, người chơi sẽ không bao giờ cảm thấy cô đơn trên chặng đường phiêu lưu của mình, bởi sẽ luôn có những *cơn ác mộng* đang chực chờ tấn công, trừng phạt họ…

- Trong phiên bản này, sinh viên thiết kế chương trình dựa trên trò chơi gốc kết hợp với bản mở rộng *The Glyph*.



* 1. **Luật chơi**

Bạn là một nhà du hành giấc mơ và vô tình lạc vào một *Mê cung* bí ẩn (Labyrinth).

Để thoát khỏi đó, bạn cần tìm và mở khóa tất cả 12 cánh cửa dẫn đến 4 địa điểm khác nhau của thế giới giấc mơ: **Đài quan sát** (Obsevatory), **Thủy cung** (Aquarium), **Khu vườn** (Garden), **Thư viện** (Library).

Có 3 cách để mở khóa các cánh cửa, sẽ được nhắc đến ở các mục 4.2.5, 4.2.6 và 4.2.8 bên dưới.

* + 1. ***Giải thích từ ngữ***

Do giao diện chơi game được viết bằng tiếng Anh, để tiện cho việc đọc hiểu luật cũng như chơi trò chơi, sinh viên sẽ giải nghĩa một số từ khóa quan trọng như sau:

|  |  |  |  |
| --- | --- | --- | --- |
| **Deck** | Chồng bài rút | **Discard pile** | Chồng bài hủy |
| **Labyrinth** | Mê cung | **Limbo pile** | Vùng lơ lửng |
| **Play card** | Thả bài | **Discard card** | Hủy bài |
| **Draw card** | Rút bài | **Discover door** | Mở khóa cửa |
| **Nightmare** | Ác mộng | **Trigger** | Kích hoạt |
| **Prophecy, Incantation** | Hiệu ứng đặc biệt (xem mục 4.2.7 và 4.2.8) | **Sun, Moon,**  **Glyph, Key** | Các biểu tượng trên Location cards |

* + 1. ***Thành phần bộ bài***

- **Location cards (66 lá):** Đây là những lá bài sẽ nằm trên tay người chơi khi họ rút lên từ *chồng bài rút*. *Thả* hoặc *Hủy* những lá bài trên tay một cách thông minh sẽ giúp người chơi nhanh chóng mở khóa các cánh cửa.

+ **Đài quan sát:** 9 Sun, 4 Moon, 2 Glyph, 3 Key.

+ **Thủy cung:** 8 Sun, 4 Moon, 2 Glyph, 3 Key.

+ **Khu vườn:** 7 Sun, 4 Moon, 2 Glyph, 3 Key.

+ **Thư viện:** 6 Sun, 4 Moon, 2 Glyph, 3 Key.

- **Door cards (12 lá):** Mở khóa tất cả 12 cánh cửa được ẩn trong chồng bài rút là điều kiện thắng duy nhất của trò chơi. Mỗi địa điểm sẽ tương ứng có 3 cánh cửa.

- **Dream cards (10 lá):** *Ác mộng* là những lá bài giấc mơ duy nhất trong bản mở rộng này. Chúng cũng sẽ được xào lên trong chồng bài rút, và người chơi sẽ buộc phải xử lí chúng như một hình phạt nếu rút phải.

* + 1. ***Thiết lập ban đầu***

Để bắt đầu trò chơi, toàn bộ 88 là bài sẽ được xào lên, đặt úp trước mặt và người chơi sẽ rút 5 lá trên đỉnh chồng bài. Nếu rút phải *Door card* hay *Dream card*, đặt chúng vào *Limbo* và rút tiếp đến khi đủ 5 lá bài trên tay, sau đó đưa những lá bài trong *Limbo* trở lại chồng bài rút và xào bài trở lại.

* + 1. ***Lượt chơi***

Ở mỗi lượt chơi, người chơi có quyền lựa chọn 1 trong 2 hành động: *Thả* hoặc *Hủy* một lá bài bất kỳ trên tay. Nếu người chơi chọn *hủy* lá bài có biểu tượng *Glyph* hoặc *Key*, một hiệu ứng đặc biệt sẽ được kích hoạt (xem ở mục 4.2.7 và 4.2.8).

Lưu ý: Một lượt chơi mới sẽ chỉ bắt đầu khi người chơi có đủ 5 lá bài trên tay.

* + 1. ***Labyrinth***

Các lá bài trong khu vực *Labyrinth* sẽ do người chơi xây dựng nên trong suốt quá trình chơi. Bất cứ khi nào người chơi *thả* bài hợp lệ, lá bài sẽ được xếp vào cuối dãy bài trong Labyrinth. Hành động thả bài được xem là hợp lệ chỉ khi lá bài được thả không có cùng biểu tượng với lá bài cuối cùng nằm trên chuỗi bài trong Labyrinth.

Nếu lá bài vừa thả là lá thứ 3 trong một chuỗi bộ ba lá bài liên tiếp cùng màu vào *Labyrinth*, người chơi sẽ mở khóa được **một** cánh cửa tương ứng với màu đó. Một *Door card* sẽ được lấy ra khỏi chồng bài rút và chồng bài được xào lại ngay sau đó.

Lưu ý: Nếu lá bài vừa thả là lá thứ 4 trong một chuỗi bài liên tiếp cùng màu, nó sẽ được tính là lá bài đầu tiên của một chuỗi bộ ba mới.

* + 1. ***Rút bài***

Sau khi thực hiện một hành động, người chơi sẽ lần lượt rút bài từ chồng bài rút cho đến khi đủ 5 lá bài trên tay. Ở mỗi lần rút, sẽ có 3 trường hợp xảy ra:

- Nếu rút phải một *Location card*, đặt nó lên tay như bình thường.

- Nếu rút phải một *Dream card*, cụ thể là *Ác mộng*, người chơi cần hóa giải nó ngay lập tức (xem ở mục 4.2.9).

- Nếu rút phải một *Door card*, người chơi sẽ gặp 1 trong 2 trường hợp sau:

+ Nếu người chơi có sẵn một lá bài với biểu tượng *Key* **cùng màu** trên tay, người chơi được quyền chọn mở khóa cánh cửa đó ngay lập tức và *hủy* lá bài *Key* đó, hoặc chọn gửi *Door card* đó đến *Limbo*.

+ Ngược lại, gửi *Door card* đó đến *Limbo*.

Ngay khi có đủ 5 lá bài trên tay, thực hiện các hành động kết thúc lượt (nếu có) và bắt đầu lượt chơi mới.

* + 1. ***Prophecy***

Nếu ở lượt của mình, người chơi chọn *hủy* lá bài trên tay có biểu tượng *Key*, người chơi sẽ kích hoạt hiệu ứng Prophecy ngay lập tức. Khi đó, người chơi sẽ:

- Được xem 5 lá bài trên đỉnh chồng bài rút,

- *Hủy* **một** trong số chúng (bắt buộc),

- Đặt những lá còn lại lên **đỉnh** chồng bài theo thứ tự tùy ý.

Nếu chồng bài rút còn ít hơn 5 lá, vẫn thực hiện tương tự trên tất cả những lá bài đó.

Lưu ý: Nếu tất cả lá bài được tiết lộ đều là *Door card*, người chơi sẽ THUA ngay lập tức do việc *hủy* một trong số chúng là bắt buộc.

* + 1. ***Incantation***

Nếu ở lượt của mình, người chơi chọn hủy lá bài trên tay có biểu tượng *Glyph*, người chơi sẽ kích hoạt hiệu ứng Incantation ngay lập tức. Khi đó, người chơi sẽ:

- Được xem 5 lá bài trên đỉnh chồng bài rút,

- *Mở khóa* **một** trong số các cánh cửa (nếu có thì bắt buộc phải mở khóa),

- Đặt những lá còn lại dưới **đáy** chồng bài theo thứ tự tùy ý.

Nếu chồng bài rút còn ít hơn 5 lá, vẫn thực hiện tương tự trên tất cả những lá bài đó.

Màu sắc của *Door card* được mở khóa và màu sắc của lá bài *Glyph* được hủy trước đó không nhất thiết phải trùng nhau.

* + 1. ***Ác mộng***

Nếu rút phải một lá bài *Ác mộng* trong khi rút bài, người chơi sẽ phải thực hiện một trong 4 hình phạt sau:

1. Hủy một lá bài trên tay có biểu tượng *Key*.

2. Hủy toàn bộ lá bài đang có trên tay và rút 5 lá bài mới như cách rút bài lần đầu (xem ở mục 4.2.3).

3. Hủy toàn bộ 5 lá bài trên đỉnh chồng bài rút, trừ những lá bài *Dream card* và *Door card*, thứ được gửi vào *Limbo*.

4. Gửi một trong số những *Door card* ứng với cánh cửa đã mở khóa trở lại Limbo.

Lưu ý: Một hoặc nhiều hình phạt sẽ không thể thực hiện được tại các thời điểm khác nhau (ví dụ: người chơi không thể chọn hình phạt 1 nếu trên tay không có lá bài *Key* nào).

* + 1. ***Limbo***

Limbo, hay *Vùng lơ lửng*, là vùng không gian tạm thời, dùng để chứa những lá bài sẽ được xào lại cùng với chồng bài rút vào cuối mỗi lượt.

* + 1. ***Xào bài***

Chồng bài rút sẽ được xào lại ngay sau khi người chơi mở khóa một cánh cửa bằng bộ ba lá bài cùng màu ở *Labyrinth*, hoặc khi trên tay đã đủ 5 lá bài mà ở *Limbo* vẫn còn bài. Những lá bài ở *Limbo* đó sẽ được hợp nhất với chồng bài rút và xào lại cùng nhau.

* + 1. ***Kết thúc trò chơi***

- **Chiến thắng:** Người chơi sẽ chiến thắng khi tại bất kỳ thời điểm nào, tất cả 12 cánh cửa đều đã được mở khóa.

- **Thua cuộc:** Người chơi sẽ thua cuộc khi trên tay chưa đủ 5 lá bài nhưng chồng bài rút không còn lá bài nào để rút thêm, hoặc rơi vào trường hợp như đã đề cập ở mục 4.2.7.

* 1. **Cấu trúc chương trình**

Để hiện thực game, sinh viên chia các class thành phần thành 3 nhóm:

**1. components:** chứa các class là những thành phần trong game (bao gồm: *Card*).

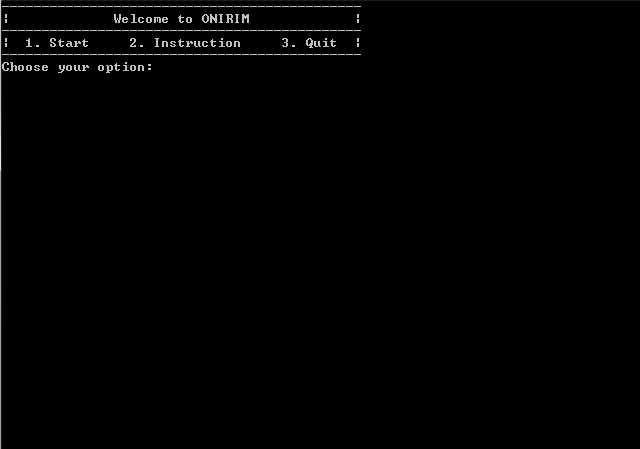
**2. container:** chứa các class là những khu vực thẻ bài riêng lẻ trong một màn chơi (bao gồm: *Deck*, *Discard*, *Hand*, *Labyrinth*, *Limbo*).

**3. gameplay:** chứa các class là những tương tác liên quan đến components và container, đồng thời thể hiện ra giao diện (bao gồm: *Action*, *Display*).

Ngoài ra, để phân khu giao diện một cách trực quan và dễ dàng hơn, class *Display* còn chứa 2 class con là *Point* và *Area*.

* 1. **Mô tả giao diện game**

Ngay khi khởi động game, màn hình sẽ hiện ra một Menu lựa chọn như sau:



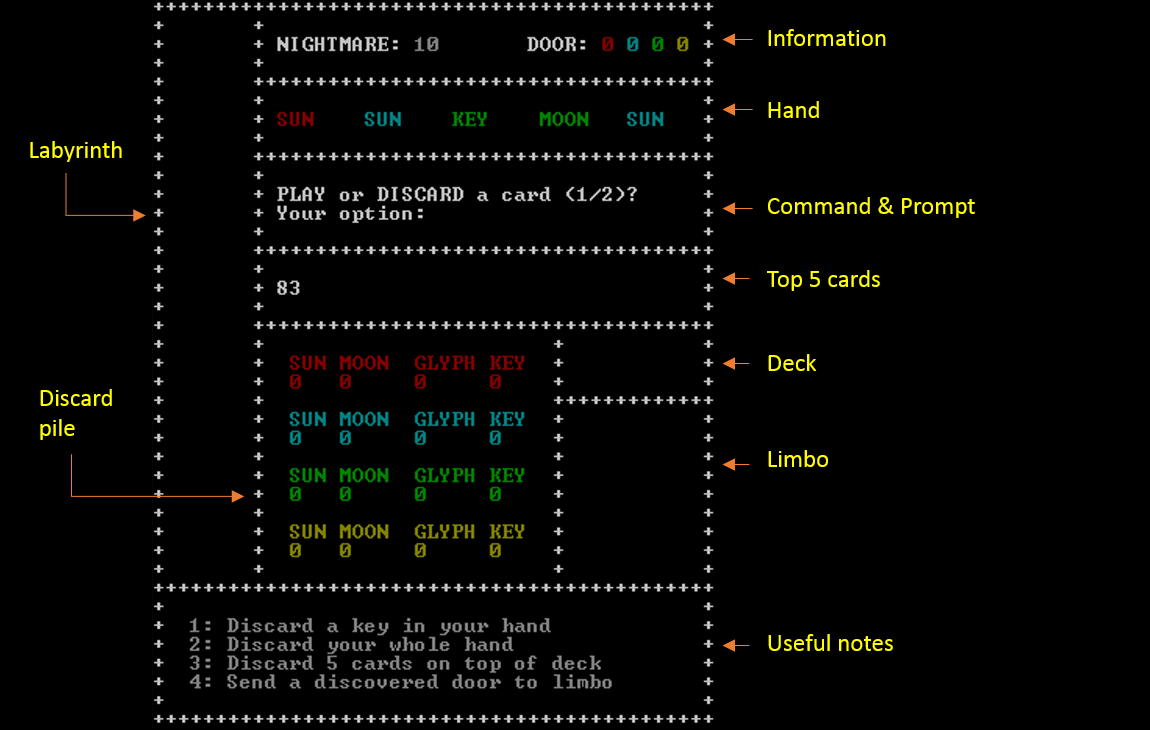
Người chơi có 3 sự lựa chọn:

**1. Bắt đầu trò chơi:** Màn hình sẽ chuyển qua giao diện chơi game như bên dưới.

**2. Hướng dẫn:** Một file text hướng dẫn luật chơi sẽ được xuất ra ổ đĩa D.

**3. Thoát:** Tắt chương trình.

Để bắt đầu trò chơi, người chơi nhập **1** và bấm Enter.



Theo giao diện trên,

- **Labyrinth:** Chứa các lá bài đã được người chơi *thả* trong suốt màn chơi.

- **Discard pile:** Chứa thông tin về số lượng các lá bài mà người chơi đã *hủy* trong suốt màn chơi, chia theo từng loại.

- **Information:** Chứa thông tin về số *Ác mộng* chưa được hóa giải còn lại trong chồng bài rút và số cánh cửa đã được mở khóa trong suốt màn chơi.

- **Hand:** Chứa 5 lá bài trên tay người chơi ở thời điểm hiện tại.

- **Command & Prompt:** Chứa các lệnh và các thông báo từ game đến người chơi.

- **Top 5 cards:** Chứa 5 lá bài rút ra từ đỉnh chồng bài rút khi kích hoạt hiệu ứng Prophecy hoặc Incantation. Con số nằm bên trái chính là số lượng bài còn lại trong chồng bài rút.

- **Deck:** Chứa lá bài vừa rút ra từ đỉnh chồng bài rút ở mỗi lượt rút bài.

- **Limbo:** Chứa những lá bài đang được đưa vào Limbo.

- **Useful notes:** Chứa ghi chú về các lựa chọn hình phạt khi gặp phải *Ác mộng*.

Để chơi trò chơi, người chơi sẽ thực hiện các câu lệnh ở Command & Prompt tùy theo hoàn cảnh.

**TÀI LIỆU THAM KHẢO**

1. Wikipedia (last edited 27/5/2023), “*Object-oriented programming*”. https://en.wikipedia.org/wiki/Object-oriented\_programming

2. Kyle Biniasz (12/11/2021), “*What Are OOP Concepts in Java? How They Work and More*”. https://stackify.com/oops-concepts-in-java/

3. Thanoshan MV (1/5/2020), “*Object-Oriented Programming Principles in Java: OOP Concepts for Beginners*”. https://www.freecodecamp.org/news/java-object-oriented-programming-system-principles-oops-concepts-for-beginners/

4. Dong Nguyen (8/8/2019), “*Functional Programming - Phần 1 - Con đường sáng*”. https://kipalog.com/posts/Functional-Programming---Phan-1---Con-duong-sang

5. Ryan Thelin (15/10/2020), “*What is functional programming? Explained in Python, JS, and Java*”. https://www.educative.io/blog/what-is-functional-programming-python-js-java

6. Oracle, “*Java Documentation*”. https://docs.oracle.com/en/java/

7. James Britt, “*Documentation for Ruby*”. https://ruby-doc.org/

8. Equinix, “*Documentation - Haskell.org*”. https://www.haskell.org/documentation/

9. SML/NJ Fellowship, “*Standard ML of New Jersey User’s Guide*”. http://www.smlnj.org/doc/

10. Hackerrank (22/2/2021), “*Difference Between Java and Ruby*”. https://www.geeksforgeeks.org/difference-between-java-and-ruby

11. Vahid Dejwakh (13/11/2020), “*Encapsulation, Polymorphism, and Abstraction in Ruby*”. https://dev.to/dej/encapsulation-polymorphism-and-abstraction-in-ruby-41ia

12. Wikipedia (last edited 30/4/2023), “*Data wrangling*”. https://en.wikipedia.org/wiki/Data\_wrangling

13. Topi.vn (22/12/2022), “*Smart Contract là gì? Vai trò và ứng dụng của hợp đồng thông minh trong đầu tư*”. https://topi.vn/smart-contract-la-gi.html

14. Ultra BoardGames, “*Onirim Fan Site*”, https://www.ultraboardgames.com/onirim/