**ĐẠI HỌC ĐÀ NẴNG**

**TRƯỜNG ĐẠI HỌC BÁCH KHOA**

**KHOA CÔNG NGHỆ THÔNG TIN**



**PBL2: DỰ ÁN CƠ SỞ LẬP TRÌNH**

**Đề tài V: XÂY DỰNG TRÒ CHƠI ĐẬP RUỒI**

**SINH VIÊN THỰC HIỆN:**

**Nguyễn Văn Hoàng Giáp LỚP: 22T\_DT1 NHÓM: 22.Nh10**

**Nguyễn Công Thành LỚP: 22T\_DT1 NHÓM: 22.Nh10**

**GIẢNG VIÊN HƯỚNG DẪN: NGUYỄN VĂN NGUYÊN**

**Đà Nẵng ngày 23-09-2023**

NHẬN XÉT CỦA GIÁO VIÊN HƯỚNG DẪN

LỜI MỞ ĐẦU

Trong thế giới ngày nay, trò chơi giải trí không chỉ là một phần giải trí mà còn là một cách để thể hiện sự sáng tạo và kỹ năng lập trình. Đối với người chơi, trò chơi không chỉ mang lại giây phút thư giãn mà còn là cơ hội để tận hưởng thế giới ảo đầy màu sắc. Trong bối cảnh này, chúng ta sẽ bắt đầu hành trình xây dựng một trò chơi đơn giản nhưng thú vị - "Trò Chơi Đập Ruồi".

Mục tiêu là xây dựng một trò chơi đơn giản nhưng hấp dẫn, nơi người chơi có nhiệm vụ đập những con ruồi xuất hiện trên màn hình bằng cách sử dụng công cụ đập sử dụng ngôn ngữ lập trình C++ kết hợp với thư viện SDL2 để tạo ra trò chơi. SDL2 là một thư viện đa nền tảng cung cấp các công cụ cần thiết để xây dựng các ứng dụng đa phương tiện, bao gồm cả trò chơi.

Chọn lựa sử dụng C++ vì đó là một ngôn ngữ lập trình mạnh mẽ và phổ biến, kết hợp với thư viện đa nền tảng SDL2. SDL2 không chỉ cung cấp các chức năng cơ bản như tạo cửa sổ và xử lý sự kiện mà còn giúp chúng ta dễ dàng tích hợp âm thanh, đồ họa và tương tác người chơi.

MỤC LỤC

[1](#_Toc151792518)

[LỜI MỞ ĐẦU 3](#_Toc151792519)

[MỤC LỤC 4](#_Toc151792520)

[GIỚI THIỆU ĐỀ TÀI 7](#_Toc151792521)

[CHƯƠNG 1: BÀI TOÁN 8](#_Toc151792522)

[1.1. PHÂN TÍCH CẤU TRÚC HỆ THỐNG 8](#_Toc151792523)

[1.1.1. Giao diện chính của chương trình 8](#_Toc151792524)

[1.1.2. Phần lựa chọn màn chơi 8](#_Toc151792525)

[1.1.3. Phần chơi 8](#_Toc151792526)

[1.1.4. Phần setting 8](#_Toc151792527)

[1.2. CẤU TRÚC DỮ LIỆU 8](#_Toc151792528)

[1.2.1. Phát biểu bài toán 8](#_Toc151792529)

[1.2.2. Phân tích và ứng dụng cấu trúc dữ liệu trong hệ thống 9](#_Toc151792530)

[a) Danh sách liên kết 9](#_Toc151792531)

[b) Cây 9](#_Toc151792532)

[1.3. Thuật toán áp dụng 10](#_Toc151792533)

[1.3.1. Cách hoạt động của chương trình 10](#_Toc151792534)

[1.3.2. Thuật toán được sử dụng cho việc đập rồi. 10](#_Toc151792535)

[1.4. ÁP DỤNG HƯỚNG ĐỐI TƯỢNG TRONG CHƯƠNG TRÌNH 11](#_Toc151792536)

[1.4.1. Cấu trúc hệ thống hướng đối tượng 11](#_Toc151792537)

[a) Abstract Class Scene: 11](#_Toc151792538)

[b) Class Object. 11](#_Toc151792539)

[c) Các class khác 11](#_Toc151792540)

[CHƯƠNG 2: TRIỂN KHAI VÀ KẾT QUẢ 13](#_Toc151792541)

[2.1. Giao diện chính của chương trình 13](#_Toc151792542)

[2.1.1. Kết quả thực thi của chương trình 13](#_Toc151792543)

[2.1.2. Nhận xét 16](#_Toc151792544)

[CHƯƠNG 3: KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN 17](#_Toc151792545)

[3.1. Đạt được 17](#_Toc151792546)

[3.2. Chưa đạt được 17](#_Toc151792547)

[3.3. Hướng phát triển 17](#_Toc151792548)

[TÀI LIỆU THAM KHẢO 18](#_Toc151792549)

[PHỤ LỤC 19](#_Toc151792550)

**DANH MỤC HÌNH VẼ**

[Hình 1 13](#_Toc150538439)

[Hình 2 13](#_Toc150538440)

[Hình 3 14](#_Toc150538441)

[Hình 4 14](#_Toc150538442)

[Hình 5 15](#_Toc150538443)

[Hình 6 15](#_Toc150538444)

[Hình 7 16](#_Toc150538445)

# GIỚI THIỆU ĐỀ TÀI

Các trò chơi giải trí là một phần không thể thiếu trong cuộc sống của chúng ta, từ những game đơn giản, game giải đố trí thông minh cho đến các game đối kháng có đồ hoạ phức tạp đang dần len lỏi vào đời sống của chúng ta.

Ruồi là một loài động vật đáng ghét trong cuộc sống và hành động đập ruồi rất phổ biến ở các gia đình hiện nay. Dựa vào đó nhóm chúng em đã chọn đề tài làm game đập ruồi là một game giải trí tương đối đơn giản cho bài tập PBL2 – dự án cơ sở lập trình.

Trò chơi bao gồm 1 chiếc vợt điều khiển bằng chuột và màn hình chứa các con ruồi đang di chuyển. Nhiệm vụ của người chơi là đánh trúng càng nhiều con ruồi đang đâu càng tốt. Mỗi lần đánh chết một con ruồi, màn hình sẽ cộng điểm tương ứng và nếu để ruồi bay đi người chơi sẽ bị trừ điểm. Sau khi số ruồi trên màn hình bị giết hết, màn hình trò chơi sẽ hiển thị điểm và trở lại màn hình menu.

# BÀI TOÁN

## PHÂN TÍCH CẤU TRÚC HỆ THỐNG

### Giao diện chính của chương trình

Ở giao diện chính của chương trình có 3 lựa chọn là Play, End và Setting( hình bánh răng ).

### Phần lựa chọn màn chơi

Ở phần lựa chọn màn chơi, hệ thống sẽ in ra một danh sách màn chơi từ level 1 trở đi, người chơi có thể thao tác bằng hình thức cuộn chuột và click để lựa chọn màn chơi.

Người chơi có thể chọn autorun hoặc không. Nếu người chơi không chọn mặc định sẽ là không autorun.

Người chơi cũng có thể lựa chọn quay lại bằng mũi tên quay lui ở phía góc phải bên dưới màn hình.

### Phần chơi

Người chơi có thể lựa chọn pause bằng nút pause ở góc phải bên dưới màn hình đẻ quay trở lại phần lựa chọn màn chơi.

### Phần setting

Người chơi có thể tự do custom màn chơi theo ý thích của mình. Người chơi có thể thêm số lượng ruồi vào màn chơi, điều chỉnh sửa đổi các thuộc tính của nó. Nhấn lưu để lưu lại màn chơi custom và thoát ra ngoài để tiếp tục chơi tiếp.

## CẤU TRÚC DỮ LIỆU

### Phát biểu bài toán

Chương trình có đầu vào là file txt để đọc các dữ liệu màn chơi bao gồm số lượng màn chơi, số lượng ruồi và các thuộc tính gồm điểm bắt đầu, kết thúc, độ lớn, tốc độ bay, điểm số và thời điểm xuất hiện.

Các file ảnh rời rạc như background, các buttons lựa chọn, các đối tượng trong game, khung viền và các file âm thanh game bao gồm âm nền, âm đánh trúng ruồi và âm thanh kết thúc màn chơi.

### Phân tích và ứng dụng cấu trúc dữ liệu trong hệ thống

#### Danh sách liên kết

Chương trình sử dụng cấu trúc dữ liệu danh sách liên kết đôi để lưu và quản lý các đối tượng thuộc class Fly – Giảm độ phức tạp so với sử dụng Mảng.

Dùng class Node để lưu giá trị, và con trỏ Node lưu địa chỉ của Node phía trước và Node phía sau, dùng con trỏ Node head để lưu nối đầu tiên, dùng con trỏ Node tail để lưu nối cuối, dùng con trỏ Node current để lưu vị trí Node hiện tại của để lấy giá trị.

Trong danh sách liên kết đôi có 2 hàm chính đó là deleteNode(), hàm pushData().

* **PushData():** thêm Node mới vào cuối danh sách liên kết, con trỏ Node next của Node mà con trỏ Node tail đang chỉ đến sẽ trỏ đến địa chỉ Node mới, con trỏ Node privious của Node mới sẽ trỏ đến địa chỉ của Node tail đang trỏ đến, con trỏ Node next của Node mới của trỏ đến NULL, con trỏ Node tail sẽ trỏ đến địa chỉ của Node mới.
* **DeleteNode():** xoá nốt mà con trỏ Node current đang trỏ đến. 2 Node phía trước Node mà con trỏ Node current đang trỏ sẽ liên kết với nhau, giải bộ nhớ con trỏ Node current trỏ đến. con trỏ Node curent trỏ đến Node nằm phía trước Node bị xoá.
* **IsEmpty():** xét xem danh sách có rỗng hay không trả về tru nếu danh sách rỗng, trả về false nếu danh sách không rông.

#### Cây

Chương trình sử dụng cấu trúc dữ liệu cây có nhiều nút con lưu trữ và quản lý các đối tượng của mỗi class kế thừa từ class Scene, quản lý chuyển cảnh màn hinh qua lại lẫn nhau.

Dùng class Node lưu giá trị, con trỏ Node để lưu địa chỉ Node cha, một mảng con trỏ Node để lưu địa chỉ Node con.

* **PushNode():** thêm lần lượt địa chỉ các Node con vào mảng con trỏ Node.
* **PreviousIndex():** chuyển con trỏ Node current đến địa chỉ Node cha của Node hiện tại mà con trỏ Node current đang trỏ đến.
* **PreviousIndex(int index):** con trỏ Node current gán giá trị thứ index của mảng con trỏ Node của Node hiện tại mà con trỏ Node current đang trỏ đến.
* **getNodeIndex():** trả về giá trị mà con trỏ Node current đang trỏ đến.

## Thuật toán áp dụng

### Cách hoạt động của chương trình

Mọi hoạt đông của trò chơi đều thực hiên trong vòng lặp, theo chu trình:

- HandleEvent() để nhận sự kiện dùng để xử lý các đối tượng.

- Update() để cập nhật lại tất cả các đối tượng của chương trình.

- Render() hiển thị tất cả các đối tượng ra màn hình.

Quá trình sẽ được lặp lại người chơi sẽ sử dụng chuột để chơi game cho đến khi gặp điều kiện dừng thì trò chơi kết thúc.

### Thuật toán được sử dụng cho việc đập rồi.

Mỗi con ruồi đều có một hình chữ nhật bao quanh con ruồi được gọi là hitbox(gồm có tạo độ x, y, chiều rộng, chiều dài), khi ruồi được cấp tạo độ mới thì hitbox cũng cấp một tọa độ giống như tọa độ của ruồi.

Tương tự với vợt đập ruồi cũng có một hitbox, vợi sẽ được cấp tọa độ trùng với tọa độ của chuột trên màn hình.

Sử dụng thuật toán AABB dùng để xét xem hai hitbox của ruồi và vợt có va chạm với nhau hay không. Khi hitbox của ruồi và vợt va chạm thì nếu có nhận sự kiện bấm click chuột thì ruồi đã được tính là đã bị đập.

Mô tả thuật toán bằng hình dưới:

A diagram of a diagram

Description automatically generated

## ÁP DỤNG HƯỚNG ĐỐI TƯỢNG TRONG CHƯƠNG TRÌNH

### Cấu trúc hệ thống hướng đối tượng

#### Abstract Class Scene:

Class Scene là một asbtract class gồm các hàm init(), handeEvent(), update(), render(), destroy(), loop(), mục đích của class này để định nghĩa giao diện người dùng.

Từ đó class menu, class gameplay, class listmap và class tool kế thừa từ class Scene. Các class này sẽ overiding lại các hàm của class Scene để phù hợp chức năng của từng class.

#### Class Object.

Class Object là 1 lớp cơ sở đại diện cho 1 đối tượng trong trò chơi với các thuộc tính cơ bản: toạ độ trên cửa sổ(pos\_x, pox\_y), texture, renderer, 1 hình vuông chứa đối tượng với SDL\_FRect.

Class Fly và Racket kế thừa lại class Object với các thuộc tính riêng. Đối với Fly là hướng di chuyển, tốc độ, điểm số, animation vỗ cánh. Còn với Racket là toạ độ chuột, bool click, animation click chuột (hành động đánh).

#### Các class khác

Class LoadImage, LoadMusic, LoadDocument xử lý các yếu tố hình ảnh, âm thanh và phần tiêu đề cho game.

Class Animation xử lý các chuyển động hoạt ảnh của các đối tượng trong game.

Class ManageObject kiểm soát các hoạt động của game, các đối tượng được xử lý như thế nào, xử lý điểm số, va chạm, cũng như khởi tạo các đối tượng trong mỗi màn chơi. Gồm các hàm chính:

* Hàm add() được đa năng hoá để có thể sử dụng cho nhiều đối tượng, mục đích để thêm vào danh sách liên kết các đối tượng trong màn chơi. Cụ thể là những con ruồi, cây vợt, sự kiện và thời gian.
* Hàm ReadMap() được dùng để đọc các thông tin màn chơi từ file txt được lưu ở ngoài thư mục.
* Hàm ManageFly() dùng để quản lý các đối tượng ruồi trong game, xử lý va chạm, xử lý điểm.
* Hàm Update() được dùng để cập nhật các đối tượng theo thời gian trong game.

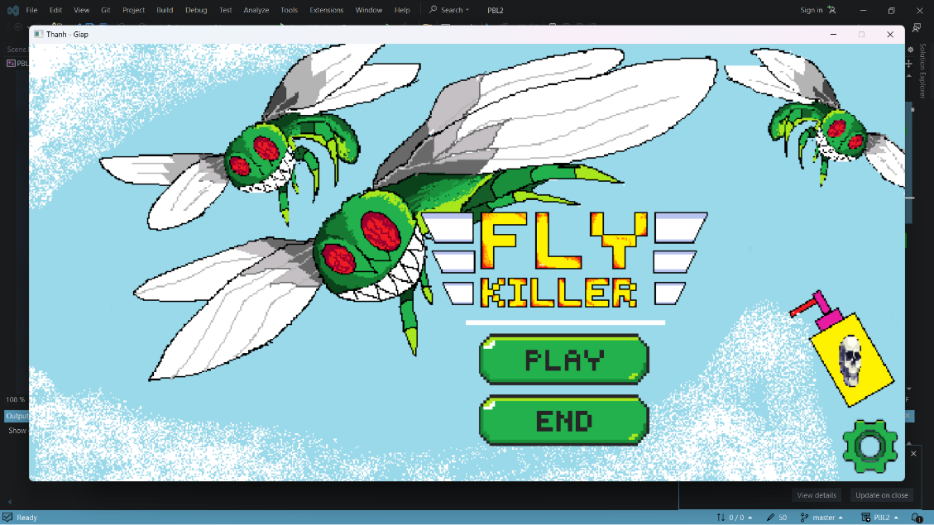
Class Timer dùng để khởi tạo, cập nhật và điều khiển thời gian trong game.

Ngoài ra còn có các Class LinkList và Tree dùng để quản lý các đối tượng theo kiểu danh sách liên kết.

# TRIỂN KHAI VÀ KẾT QUẢ

## Giao diện chính của chương trình

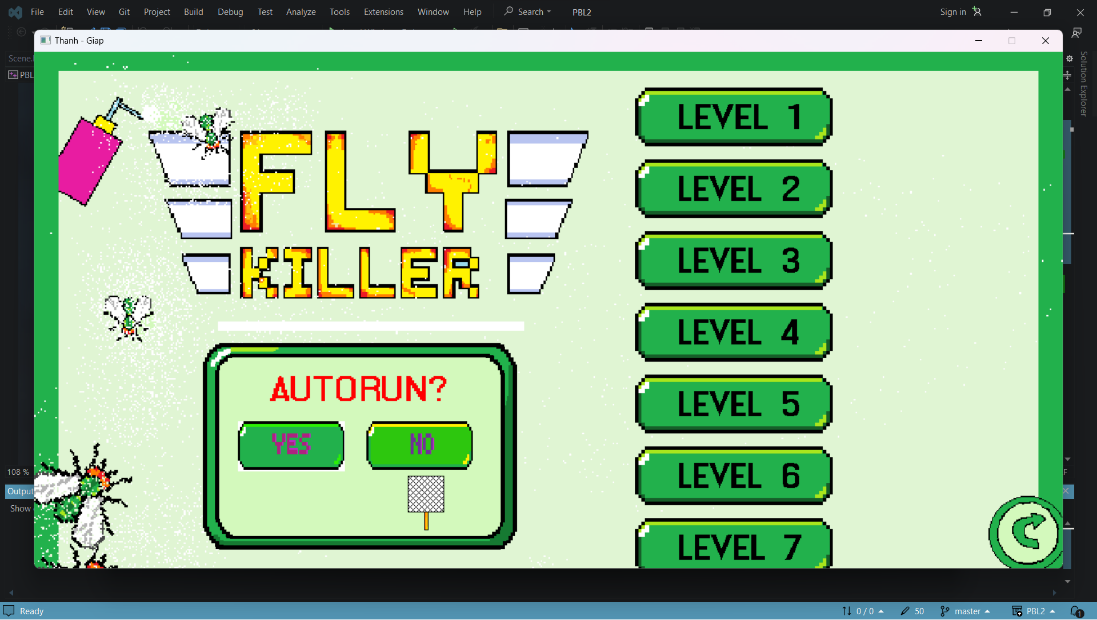
Giao diện chính của chương trình sau khi khởi động như hình với 3 lựa chọn là bắt đầu chơi, thoát và mục setting.



*(Hình 1)*

### Kết quả thực thi của chương trình

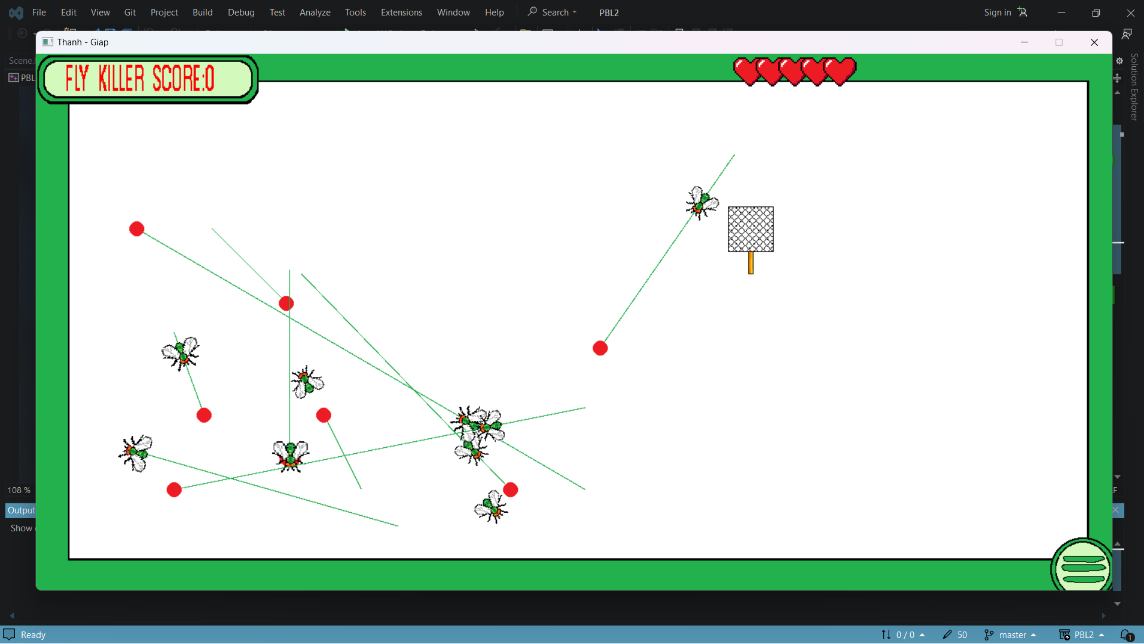
Nếu nhấn vào play, chương trình sẽ chuyển qua một menu để người chơi lựa chọn màn chơi ( list map ). Sau đó người chơi chọn màn chơi và chọn chế độ để bắt đầu. Có hai chế độ là autorun và tự chơi. Nếu không chọn thì mặc định sẽ là tự chơi như hình.



*(Hình 2)*

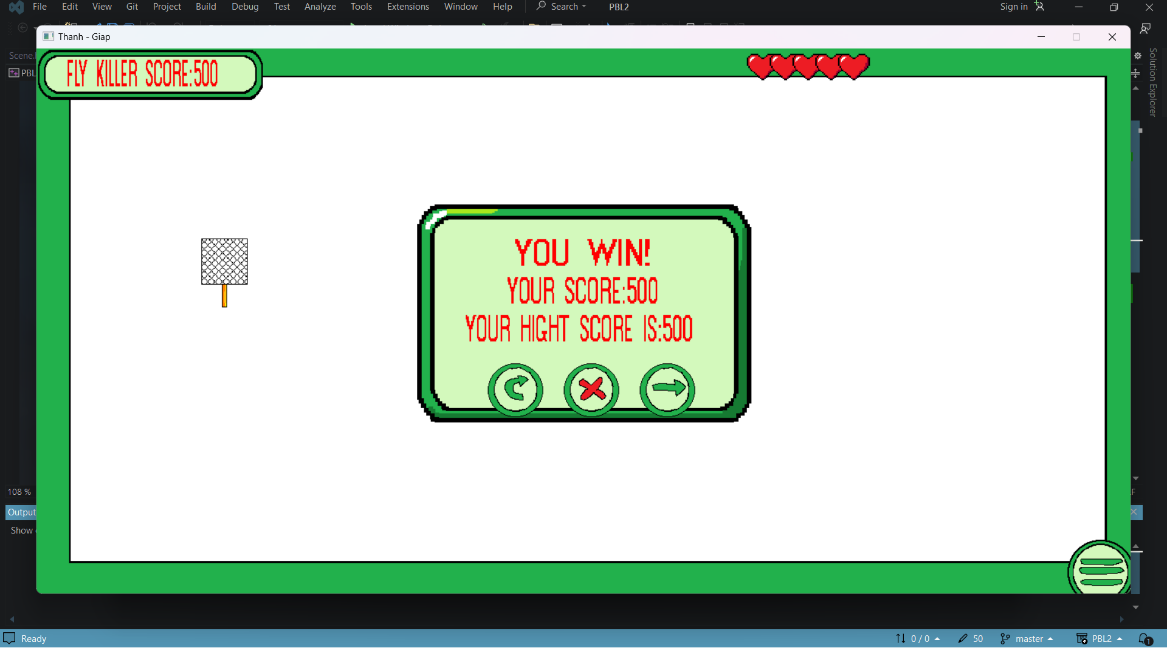
Người chơi cũng có thể lựa chọn quay lại màn hình chính với dấu quay lại ở góc phải bên dưới màn hình.

Sau khi người chơi lựa chọn màn chơi thì chương trình sẽ tiến vào chế độ chơi với số mạng chơi mặc định là 5. Các con ruồi sẽ lần lượt xuất hiện và sau khi con ruồi dừng lại thì người chơi tiến hành đập, lưu ý nếu đánh vào lúc con ruồi đang di chuyển sẽ bị tính mất 1 mạng chơi. Số mạng chơi là các trái tim màu đỏ ở góc phải phía trên màn hình.



*(Hình 3)*

Mỗi một con ruồi đập được chương trình sẽ tiến hành cộng điểm cho người chơi ứng với số điểm mà con ruồi đang mang và sau khi hoàn thành màn chơi hoặc số ruồi trên màn hình biến mất với số điểm lớn hơn 0 thì người chơi chiến thắng.



*(Hình 4)*

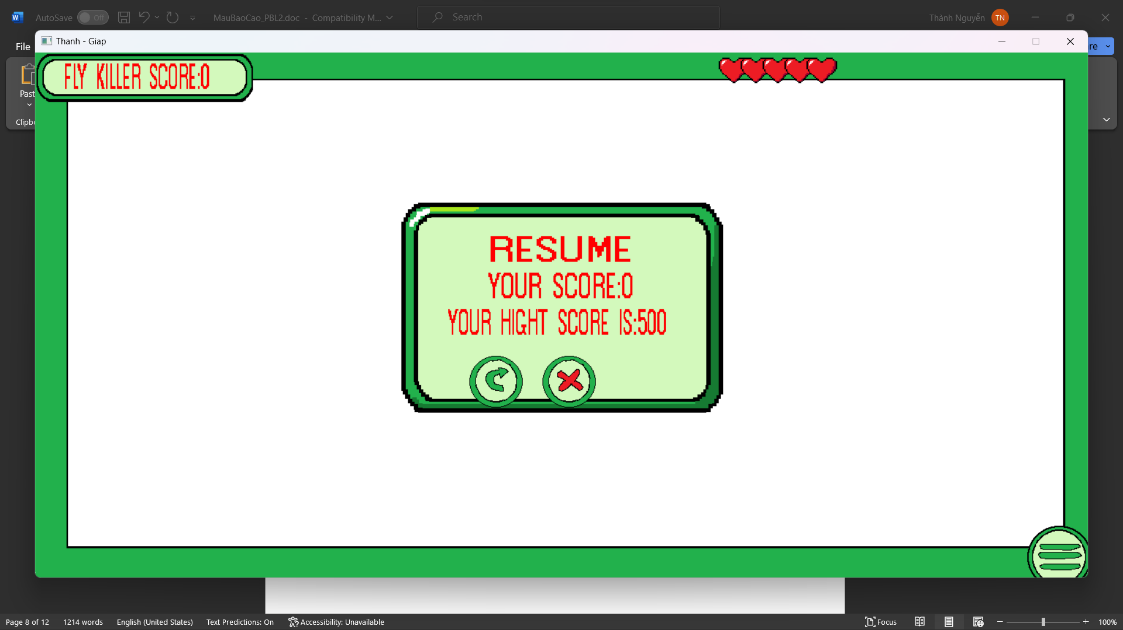
Lúc này người chơi có thể lựa chọn chuyển sang màn chơi tiếp theo khi nhấn dấu mũi tên, quay lại phần menu lựa chọn màn với dấu mũi tên hoặc là chơi lại màn chơi với dấu mũi tên quay lui.

Nếu người chơi không thể vượt qua màn chơi – số điểm sau khi kết thúc màn chơi là 0 hoặc số mạng chơi mất hết thì người chơi sẽ được cho là thua và có thể chọn chơi lại màn hoặc là thoát ra menu lựa chọn màn. Người chơi thua sẽ không được lựa chọn qua màn chơi tiếp theo.



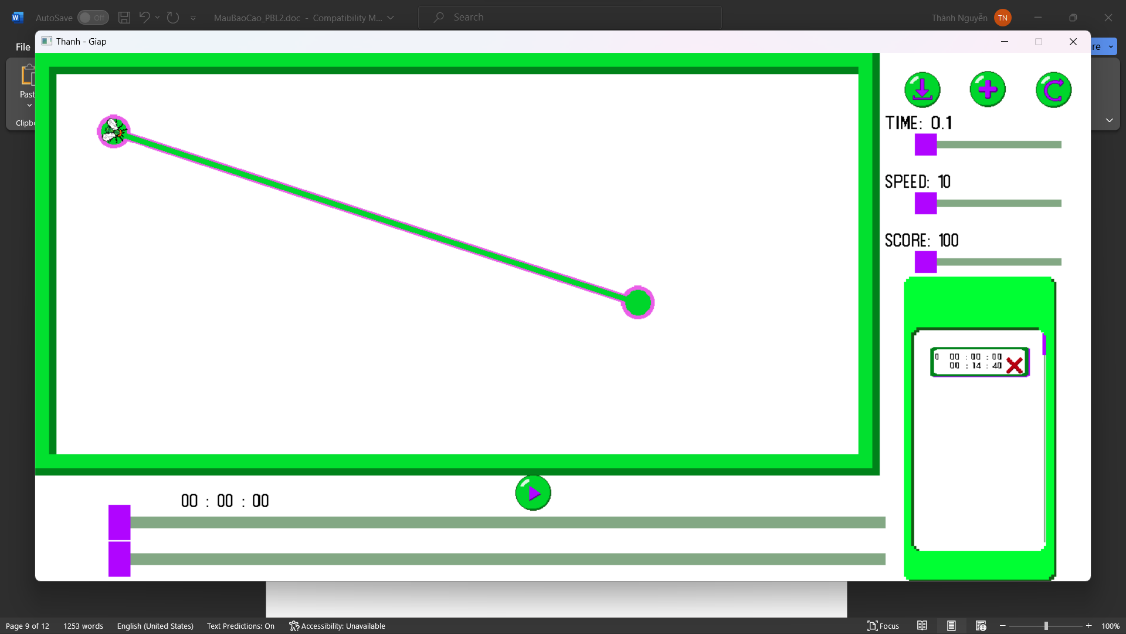
*(Hình 5)*

Người chơi cũng có thể pause lại màn chơi. Lúc này người chơi có 2 lựa chọn là quay lại chơi tiếp hoặc là thoát ra ngoài.



*(Hình 6)*

Ở menu chính nếu người chơi lựa chọn setting thì chương trình sẽ chuyển sang phần setting. Tại đây người chơi sẽ tự do để custom 1 màn chơi cho mình.



*(Hình 7)*

Người chơi cũng có thể tuỳ chỉnh số lượng ruồi, tốc độ bay cho ruồi, thời điểm xuất hiện, biến mất cũng như điểm số của từng chú ruồi. Sau khi custom xong thì người chơi nhấn nút lưu rồi thoát ra, chương trình sẽ lưu màn chơi của người chơi custom ở vị trí cuối cùng trong list màn chơi.

Ngoài ra, chương trình còn có âm thanh sống động, hiện đại mang chút âm hưởng vui nhộn phù hợp với đa số trẻ em và người muốn giải trí.

### Nhận xét

Chương trình tuy vẫn còn thô sơ nhưng cũng đã tổ chức sử dụng rất tốt phần hướng đối tượng trong lập trình C++ cũng như sử dụng khá tốt phần thư viện đồ hoạ SDL2.

Các nhân vật, giao diện hoàn toàn được vẽ thủ công và phối màu đẹp, thích mắt tạo sự hứng thú cho người chơi.

# KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN

## Đạt được

Chương trình đã vẫn dụng các kiến thức như cấu trúc dữ liệu, lập trình hướng đối tượng. Sử dụng được ở mức cơ bản thư viện đồ hoạ SDL2.

Game tương đối hoàn thiện, game dễ tiếp cận với người chơi nhưng vẫn đầy cuốn hút và thú vị.

## Chưa đạt được

Phần đồ hoạ thiết kế chưa được đẹp và còn đơn giản, phần animation chưa được mượt mà.

Phần âm thanh của game còn sử dụng âm thanh có sẵn.

## Hướng phát triển

Game vẫn cần nâng cấp về cấu hình, đồ hoạ và âm thanh.

Có thể phát triển để người chơi có thể tạo và chia sẻ màn chơi của mình cho những người chơi khác.

TÀI LIỆU THAM KHẢO

[1] Lê Minh Hoàng. “*Giải thuật và lập trình*”. Đại học sư phạm Hà Nội, 1999-2002.

[2] Đỗ Xuân Lôi. "*Cấu trúc dữ liệu và giải thuật*". Đại học Quốc gia Hà Nội, 2010.

[3]− GS. Phạm Văn Ất. “*Giáo trình c++ & lập trình hướng đối tượng*”. Bách Khoa Hà Nội, 2023.

[4] Phùng Đức Hòa. "*Giáo trình lập trình hướng đối tượng với C++*". Khoa học kỹ thuật, 2013.

[5]Trương Hải Bằng . “*Giáo Trình Lập Trình Hướng Đối Tượng*”. Đại học Quốc gia TPHCM, 2007.

[6] Nguyễn Tuấn Anh. “*Giáo trình kỹ thuật hướng đối tượng bằng C++*”. Giáo dục Việt Nam, 2016.

[7] Lê Thị Mỹ Hạnh. “*Giáo trình Lập trình hướng đối tượng*”. Giáo trình lưu hành nội bộ, 2012.

[8] Nguyễn Thanh Bình, Phạm Minh Tuấn, Đặng Thiên Bình. “*Phân tích và thiết kế giải thuật*”. NXB GD, 2016.

[9] T.H. Cormen, C.E. Leiserson, R.R. Rivest. “*Introduction to algorithms*”. Mit Press, 1990.

PHỤ LỤC

*Phần code của chương trình*

* Class LoadImage

#ifndef LOAD\_IMAGE\_H

#define LOAD\_IMAGE\_H

#include <SDL.h>

#include <SDL\_image.h>

#include <iostream>

class LoadImage

{

public:

// ham load anh cho doi tuong co bien "static"

// nen co the goi ham o bat cu dau ma khong can phai

// tao doi tuong mien la da #include "LoadImage"

static SDL\_Texture\* Load(const char\* file, SDL\_Renderer\* render)

{

SDL\_Texture\* texture;

SDL\_Surface\* image = IMG\_Load(file);

if (image)

{

texture = SDL\_CreateTextureFromSurface(render, image);

SDL\_FreeSurface(image);

return texture;

}

else std::cout << "Dell tim thay duong dan " << file << std::endl;

return nullptr;

}

};

#endif // !LOAD\_IMAGE\_H

* Class ManageObject
* #include <Manager/ManagerObject.h>
* #include <Load/LoadMusic.h>
* LoadMusic\* music;
* LoadMusic\* music2;
* FramesObject\* Round = nullptr;
* Manager::Manager() {
* FlyLinkList = new ObjectLinkList<Fly>();
* music = new LoadMusic(2);
* music2 = new LoadMusic(5);
* music->addSound("Data\\Sound\\hit3.wav");
* music2->addSound("Data\\Sound\\miss.mp3");
* }
* void Manager::Add(Fly\* a) {
* FlyLinkList->push(a);
* }
* void Manager::Add(Racket\* racKet) {
* this->racKet = racKet;
* }
* void Manager::Add(float\* timegame)
* {
* this->timegame = timegame;
* }
* void Manager::ReadMap(SDL\_Renderer\* renderer, int level)
* {
* std::string path = "Data//Map-dif//Level" +std::to\_string(level) + ".txt";
* ReadFile(FlyLinkList, renderer, timegame, path.c\_str());
* Round = new FramesObject(FlyLinkList->getIndex()->getData()->GetENd(), "Data//Edit//Goal\_50\_50\_50\_50.png", renderer, false);
* sizeStart = FlyLinkList->getSize();
* sizeEnd = 0;
* }
* void Manager::Update(bool set, int& heart, bool autorun, float \*timegame) {
* UpdatePositionAndVector();
* FlyLinkList->resetIndex();
* ManagerFly(set, heart, autorun, timegame);
* if (!autorun) {
* racKet->UpdatePositionOfMouse();
* }
* else {
* //Do nothing
* }
* racKet->UpdateRacket();
* }
* void Manager::render(SDL\_Renderer\* renderer) {
* while (!FlyLinkList->setIndex()&& !FlyLinkList->isEmpty())
* {
* FlyLinkList->getIndex()->getData()->Render();
* FlyLinkList->GoNext();
* }
* FlyLinkList->resetIndex();
* racKet->Render();
* }
* int Manager::getSize() { // hàm lấy giá trị vector
* return 0;// obJect.size();
* }
* void Manager::UpdatePositionAndVector() {
* while (!FlyLinkList->setIndex() && !FlyLinkList->isEmpty())
* {
* FlyLinkList->getIndex()->getData()->UpdateFly();
* Fly\* fly = FlyLinkList->getIndex()->getData();
* fly->move();
* FlyLinkList->GoNext();
* }
* FlyLinkList->resetIndex();
* }
* void Manager::ManagerFly(bool set, int& heart, bool Autorun, float \*timegame) {
* if (set) {
* while (!FlyLinkList->setIndex() && !FlyLinkList->isEmpty()) {
* if (!Autorun) {
* Fly\* fly = FlyLinkList->getIndex()->getData();
* if (fly->getStatus() && Collision(fly->GetArea(), racKet->GetArea()) && racKet->GetHit()) {
* if (!FlyLinkList->getIndex()->getData()->status) {
* music2->playSound(0);
* heart--; // tru mang
* FlyLinkList->getIndex()->getData()->MISS();
* break;
* }
* else {
* music->playSound(0);
* scored += FlyLinkList->getIndex()->getData()->Getscore();
* FlyLinkList->getIndex()->getData()->HIT();
* FlyLinkList->deleteNode();
* FlyLinkList->resetIndex();
* sizeEnd++;
* break;
* }
* }
* if (CollisionBlockWidth(fly->GetArea()) || CollisionBlockHeight(fly->GetArea())) {
* scored -= FlyLinkList->getIndex()->getData()->Getscore();
* if (scored < 0) scored = 0;
* FlyLinkList->deleteNode();
* FlyLinkList->resetIndex();
* break;
* }
* }
* else {
* if (FlyLinkList->getIndex()->getData()->TimeLand(timegame)) {
* racKet->Updateifautorun(FlyLinkList->getIndex()->getData()->GetEnd());
* if (Collision(racKet->GetArea(), FlyLinkList->getIndex()->getData()->GetArea()) && FlyLinkList->getIndex()->getData()->status){
* racKet->AutoHit();
* scored += FlyLinkList->getIndex()->getData()->Getscore();
* FlyLinkList->deleteNode();
* FlyLinkList->resetIndex();
* break;
* }
* }
* }
* FlyLinkList->GoNext();
* }
* FlyLinkList->resetIndex();
* }
* }
* bool Manager::IsEmty() {
* return FlyLinkList->isEmpty();
* }
* void Manager::Reset() {
* scored = 0;
* while (!FlyLinkList->setIndex() && !FlyLinkList->isEmpty()) {
* FlyLinkList->deleteNode();
* FlyLinkList->resetIndex();
* }
* }
* float Manager::getPercent()
* {
* return 100 \* ((float)sizeEnd) / (float)sizeStart;
* }

Class Fly:

#ifndef FLY\_H\_

#define FLY\_H\_

#include "Object.h"

#include <Time\_vector/Vector.h>

#include <Vacham/CollisionHandling.h>

#define pi 3.1415926

class Fly:public Object {

protected:

float\* timegame;

Vector direction;

double angle;

FramesObject\* FlyNormal = nullptr;

FramesObject\* FlySuper = nullptr;

FramesObject\* Goal = nullptr;

FramesObject\* Line = nullptr;

FramesObject\* Miss = nullptr;

FramesObject\* Hit = nullptr;

SDL\_FRect area;

SDL\_FRect goal;

SDL\_FRect line;

SDL\_Point linePoint;

int score;

Vector Start;

Vector End;

float speed;

float tHit;

float tMiss;

float t\_to\_a; //time to apear

float t\_to\_land;

public:

Fly

(

float pos\_x, float pos\_y, float width, float height,

float end\_x, float end\_y, float t\_to\_a, float t\_to\_land,

float speed, SDL\_Renderer\* Render, int score, float\* timegame)

: Object(pos\_x, pos\_y, width, height, Render) {

area.x = pos\_x;

area.y = pos\_y;

area.h = height;

area.w = width;

this->Start = Vector{ pos\_x, pos\_y };

this->End = Vector{ end\_x, end\_y };

this->t\_to\_a = t\_to\_a;

this->t\_to\_land = t\_to\_land;

this->speed = speed;

direction.x = End.x - Start.x;

direction.y = End.y - Start.y;

float r = sqrt(direction.x \* direction.x + direction.y \* direction.y);

direction.x = direction.x / r;

direction.y = direction.y / r;

status = false;//Trangj thái di chuyển

this->score = score;

FlyNormal = new FramesObject(&area, "Data//Picture//fly\_100\_100\_200\_100.png", renderer, true);

this->timegame = timegame;

// dat vi tri chi line va goal

line.w = 10;

line.h = r;

line.x = Start.x + width / 2.0f - line.w / 2.0f ;

line.y = Start.y - line.h + area.h/2;

goal.x = End.x + 25;

goal.y = End.y + 25;

goal.h = height - 50;

goal.w = width - 50;

Miss = new FramesObject(&area, "Data//Edit//Delete\_50\_50\_100\_50.png", renderer, true);

// dat vi tri xoay cua hinh chu Line

linePoint.x = line.w / 2;

linePoint.y = line.h;

Line = new FramesObject(&line, "Data//Picture//Line\_100\_100\_100\_100.png", renderer, false);

Goal = new FramesObject(&goal, "Data//Picture//Goal\_100\_100\_100\_100.png", renderer, false);

tHit = -1;

tMiss = -1;

}

bool status;

// hàm cập nhật điểm cho đối tượng (chú ý không gán trực tiếp lúc khai báo)

SDL\_FRect GetArea()

{

SDL\_FRect tmp;

tmp.x = area.x + area.w / 2;

tmp.y = area.y + area.h / 2;

tmp.h = area.h;

tmp.w = area.w;

return tmp;

}

void UpdateScore(int score) {

this->score = score;

}

void UpdateFly()

{

if(\*timegame > t\_to\_a) FlyNormal->UpdateFrames();

}

void HIT()

{

tHit = \*timegame + 0.5;

tMiss = -1;

}

void MISS()

{

tMiss = \*timegame + 0.1;

}

void Render()

{

if (\*timegame > t\_to\_a)

{

double angle = atan((double)direction.y / (double)direction.x);

if (direction.x < 0) angle += pi;

// ve line va goal

Line->Get\_Texture((angle \* 180) / pi + 90, linePoint);

Goal->Get\_Texture();

FlyNormal->Get\_Texture((angle \* 180) / pi + 90);

if (tHit > \*timegame) Hit->Get\_Texture();

if (tMiss > \*timegame) Miss->Get\_Texture();

}

}

int Getscore() {

return score;

}

void move() {

if (\*timegame > t\_to\_a)

{

float t = Timer::sInit->DeltaTime();

if (!status)

{

area.x += direction.x \* speed \* t;

area.y += direction.y \* speed \* t;

}

if (!status && setFly() && t\_to\_land > 0) status = true;

if (status)

{

t\_to\_land -= t;

if (t\_to\_land <= 0) status = false;

}

}

}

bool setFly()

{

Vector v1{ area.x - Start.x, area.y - Start.y };

Vector v2{ area.x - End.x, area.y - End.y };

float dot = v1.x \* v2.x + v1.y \* v2.y;

if (dot < 0) return false;

else return true;

}

void setStatus(bool set) {

this->status = set;

}

bool getStatus()

{

return \*timegame > t\_to\_a;

}

bool Collison(SDL\_FRect Racket) {

return Collision(area, Racket);

}

bool TimeLand(float \*timegame) {

float r = sqrt((End.x - Start.x) \* (End.x - Start.x) + (End.y - Start.y) \* (End.y - Start.y));

float t = r / speed;

if (t - \*timegame + t\_to\_a < 0.1) return true;

else return false;

}

Vector GetEnd() {

return End;

}

SDL\_FRect \*GetENd() {

return &goal;

}

};

#endif

* Phần cấu trúc dữ liệu
* #ifndef LINKLIST\_H
* #define LINKLIST\_H
* #include <iostream>
* template <typename T> class Node
* {
* T\* data;
* Node<T>\* previous;
* Node<T>\* next;
* public:
* Node(T\* data)
* {
* this->data = data;
* }
* void setPrevious(Node\* node)
* {
* previous = node;
* }
* Node<T>\* getPrevious() { return previous; }
* Node<T>\* getNext() { return next; }
* void setNext(Node<T>\* node)
* {
* next = node;
* }
* void setData(T\* data)
* {
* this->data = data;
* }
* T\* getData()
* {
* return data;
* }
* };
* template <typename T> class ObjectLinkList
* {
* private:
* Node<T>\* nodeHead;
* Node<T>\* nodeTail;
* Node<T>\* nodeIndex;
* int size;
* public:
* ObjectLinkList()
* {
* nodeHead = NULL;
* nodeTail = NULL;
* nodeIndex = NULL;
* size = 0;
* }
* bool isEmpty()
* {
* return nodeHead == NULL;
* }
* void push(T\* data)
* {
* if (isEmpty())
* {
* nodeHead = new Node<T>(data);
* nodeIndex = nodeHead;
* nodeTail = nodeHead;
* nodeIndex->setPrevious(NULL);
* nodeIndex->setNext(NULL);
* }
* else
* {
* Node<T>\* newNode = new Node<T>(data);
* newNode->setPrevious(nodeTail);
* newNode->setNext(NULL);
* nodeTail->setNext(newNode);
* nodeTail = newNode;
* }
* size++;
* }
* void deleteNode() // lay va xoa
* {
* if (isEmpty()) {
* }
* else {
* if (nodeIndex->getPrevious() == NULL && nodeIndex->getNext() == NULL) {
* nodeIndex = NULL;
* nodeHead = NULL;
* nodeTail = NULL;
* }
* else if (nodeIndex->getPrevious() == NULL) {
* nodeIndex->getNext()->setPrevious(nullptr);
* nodeHead = nodeIndex->getNext();
* nodeIndex = nodeIndex->getNext();
* }
* else if (nodeIndex->getNext() == NULL) {
* nodeIndex->getPrevious()->setNext(NULL);
* nodeTail = nodeIndex->getPrevious();
* nodeIndex = NULL;
* }
* else {
* nodeIndex->getPrevious()->setNext(nodeIndex->getNext());
* nodeIndex->getNext()->setPrevious(nodeIndex->getPrevious());
* nodeIndex = nodeIndex->getNext();
* }
* size--;
* }
* }
* void resetIndex()
* {
* nodeIndex = nodeHead;
* }
* bool setIndex()
* {
* return nodeIndex == NULL;
* }
* Node<T>\* getIndex() // chi lay ko xoa
* {
* return nodeIndex;
* }
* void GoNext() {
* nodeIndex = nodeIndex->getNext();
* }
* void GoPrevious() {
* nodeIndex = nodeIndex->getPrevious();
* }
* int getSize() { return size; }
* };
* #endif // !LINKLIST\_H
* #ifndef TREE\_H
* #define TREE\_H
* #include <iostream>
* #include <vector>
* template <typename T> class NodeTree {
* T\* data;
* NodeTree<T>\* previous;
* std::vector <NodeTree<T>\*> NodeData;
* public:
* NodeTree(T\* data) {
* this->data = data;
* }
* void setPrevious(NodeTree<T>\* node)
* {
* previous = node;
* }
* NodeTree<T>\* getPrevious()
* {
* return previous;
* }
* void PushNode(NodeTree<T>\* node)
* {
* NodeData.push\_back(node);
* }
* NodeTree<T>\* getNodeIndex(int i) {
* return NodeData[i];
* }
* void setIndex(int i, T\* data)
* {
* NodeData[i]->SetData(data);
* }
* void SetData(T\* data) {
* this->data = data;
* }
* T\* getData() {
* return data;
* }
* int getSize() {
* return NodeData.size();
* }
* };
* template <typename T> class Tree {
* NodeTree<T>\* NodeHead;
* NodeTree<T>\* NodeIndex;
* public:
* Tree() {
* NodeHead = NULL;
* NodeIndex = NULL;
* }
* bool isEmpty() {
* return NodeHead == NULL;
* }
* void Push(T\* data)
* {
* if (isEmpty())
* {
* NodeHead = new NodeTree<T>(data);;
* NodeIndex = NodeHead;
* NodeIndex->setPrevious(NULL);
* }
* else
* {
* NodeTree<T>\* node = new NodeTree<T>(data);
* NodeIndex->PushNode( node);
* node->setPrevious(NodeIndex);
* }
* }
* void NextIndex(int i)
* {
* NodeIndex = NodeIndex->getNodeIndex(i);
* }
* void PreviousIndex()
* {
* NodeIndex = NodeIndex->getPrevious();
* }
* NodeTree<T>\* getNodeIndex()
* {
* return NodeIndex;
* }
* void Reset() {
* NodeIndex = NodeHead;
* }
* };
* #endif