

BỘ GIÁO DỤC VÀ ĐÀO TẠO
TRƯỜNG ĐẠI HỌC NHA TRANG
KHOA CÔNG NGHỆ THÔNG TIN



ĐỒ ÁN TỐT NGHIỆP

**ỨNG DỤNG GODOT XÂY DỰNG BÀI THỰC HÀNH
KÍNH HIÊN VI ẢO CHO PHÒNG THÍ NGHIỆM SINH HỌC,
ĐẠI HỌC NHA TRANG**

Giảng viên hướng dẫn: Ths. Đoàn Vũ Thịịnh

Sinh viên thực hiện: Phạm Ngọc Ân

Mã số sinh viên: 61130013

Khánh Hòa – 2023

BỘ GIÁO DỤC VÀ ĐÀO TẠO
TRƯỜNG ĐẠI HỌC NHA TRANG
KHOA CÔNG NGHỆ THÔNG TIN



ĐỒ ÁN TỐT NGHIỆP

**ỨNG DỤNG GODOT XÂY DỰNG BÀI THỰC HÀNH
KÍNH HIỂN VI ẢO CHO PHÒNG THÍ NGHIỆM SINH HỌC,
ĐẠI HỌC NHA TRANG**

GVHD: Ths. Đoàn Vũ Thịnh

SVTH: Phạm Ngọc Ân

MSSV: 6130013

Khánh Hòa, Tháng 06/2023

LỜI CAM ĐOAN

Tôi xin cam đoan kết quả của đề tài “*Ứng dụng Godot xây dựng bài thực hành kính hiển vi ảo cho phòng thí nghiệm sinh học, Đại học Nha Trang*” là công trình nghiên cứu của riêng tôi và chưa từng được công bố trong bất kỳ nghiên cứu nào khác.

Khánh Hòa, ngày 05 tháng 06 năm 2023

Tác giả luận văn

(kí và ghi rõ họ tên)

LỜI CẢM ƠN

Trong quá trình nghiên cứu và xây dựng đồ án “Ứng dụng Godot xây dựng bài thực hành kính hiển vi ảo cho phòng thí nghiệm sinh học, Đại học Nha Trang”, tôi xin gửi lời cảm ơn chân thành tới thầy Đoàn Vũ Thịnh (Khoa Công nghệ Thông tin) và cô Văn Hồng Cầm (Viện CNSH&MT, Trường Đại học Nha Trang), hai người đã trở thành nguồn động lực và đồng hành quan trọng trong hành trình này.

Đầu tiên, tôi muốn gửi lời cảm ơn sâu sắc tới thầy Đoàn Vũ Thịnh, người đã chia sẻ kiến thức vô cùng quý báu về cách xây dựng chương trình ứng dụng và phương pháp giảng dạy. Thầy đã truyền đạt những khái niệm và kỹ năng quan trọng giúp tôi xây dựng chương trình dạy học tối ưu cho sinh viên. Những lời động viên và sự khích lệ từ thầy đã giúp tôi vượt qua những khó khăn và tìm ra giải pháp sáng tạo trong quá trình nghiên cứu. Tôi vô cùng biết ơn sự tận tâm và sự hỗ trợ không ngừng từ thầy.

Xin gửi lời cảm ơn đến cô Văn Hồng Cầm, người không chỉ đóng góp quan trọng trong việc tạo tài liệu kịch bản chuyên môn về lĩnh vực sinh học, cung cấp hình ảnh vi sinh, dụng cụ phòng thí nghiệm, mà còn là người đề xuất và hướng dẫn đề tài của tôi. Sự chuyên nghiệp và chính chu của cô đã giúp tôi có nguồn tài liệu đáng tin cậy và hấp dẫn cho chương trình dạy học. Bên cạnh đó, sự hỗ trợ kinh phí từ cô cũng đã đóng góp quan trọng để hoàn thành đề tài một cách thành công. Tôi trân trọng những đóng góp và sự hỗ trợ to lớn của cô Cầm trong quá trình nghiên cứu.

Cuối cùng, tôi xin chân thành cảm ơn tất cả các thầy cô trong Khoa Công nghệ Thông tin. Nhờ nền tảng vững chắc về lập trình và kiến thức chuyên môn mà tôi đã được học trong suốt quá trình đào tạo, tôi đã có cơ hội tiếp thu và ứng dụng kiến thức để xây dựng thành công đồ án này. Sự đồng hành và sự hỗ trợ không ngừng của các thầy cô trong Khoa đã tạo nên nền tảng vững chắc và định hướng cho sự phát triển của tôi.

Xin chân thành cảm ơn tất cả các thầy cô đã đóng góp và hỗ trợ để tôi có thể hoàn thành đồ án này.

Khánh Hòa, ngày 07 tháng 06 năm 2023

Tác giả luận văn

(kí và ghi rõ họ tên)

MỤC LỤC

LỜI CAM ĐOAN	i
LỜI CẢM ƠN.....	ii
DANH MỤC HÌNH	v
DANH MỤC BẢNG	viii
DANH MỤC CÁC KÝ HIỆU, TỪ VIẾT TẮT	i
LỜI MỞ ĐẦU	1
CHƯƠNG 1. TỔNG QUAN VỀ ĐỀ TÀI.....	3
1.1. THỰC TRẠNG.....	3
1.2. CÁC SẢN PHẨM NỔI BẬT VỀ CHỦ ĐỀ PHÒNG THÍ NGHIỆM ẢO	3
CHƯƠNG 2. CƠ SỞ LÝ THUYẾT.....	7
2.1. THÍ NGHIỆM ẢO	7
2.2. PHẦN MỀM GAME (GAME ENGINE)	8
2.3. GODOT ENGINE (PHIÊN BẢN 3.5.1).....	9
2.3.1. Tổng quan về các khái niệm chính trong Godot	10
2.3.2. Tổng quan giao diện của Godot	13
2.3.3. Màn hình chỉnh sửa kịch bản của Godot.....	14
2.4. NGÔN NGỮ LẬP TRÌNH GODOT (GDSCRIPT)	15
2.5. BLENDER (PHIÊN BẢN 3.4.1)	16
2.6. ADOBE PHOTOSHOP (Phiên bản 19.1.3 2018).....	17
2.7. CHUYỂN VĂN BẢN THÀNH GIỌNG NÓI VOICE MAKER CỦA FPT.AI18	
CHƯƠNG 3. XÂY DỰNG CHƯƠNG TRÌNH MÔ PHỎNG	19
3.1. THIẾT KẾ GIAO DIỆN CÁC BÀI THỰC HÀNH	19
3.2. THIẾT KẾ GIAO DIỆN KÍNH HIỂN VI 3D	22
3.3. XÂY DỰNG CÁC MÀN HÌNH TRONG BÀI THỰC HÀNH ẢO	25
3.3.1. Frame 1 – Màn hình đăng nhập	25
3.3.2. Frame 2 – Màn hình chính	30
3.3.3. Frame 3 – Màn hình vật liệu và hóa chất.....	38
3.3.4. Frame 4 – Màn hình cấu tạo kính hiển vi	45

3.3.5. Frame 5 – Màn hình hướng dẫn thực hành	49
3.3.6. Frame 6 – Màn hình thực hành	57
3.3.7. Frame 7 – Màn hình kiểm tra.....	64
3.3.8. Frame 8 – Màn hình chứng chỉ	69
3.3.9. Frame Loading – Màn hình tải.....	73
CHƯƠNG 4. KẾT QUẢ	76
4.1. MÀN HÌNH ĐĂNG NHẬP	76
4.2. MÀN HÌNH CHÍNH.....	76
4.3. MÀN HÌNH VẬT LIỆU VÀ HÓA CHẤT.....	76
4.4. MÀN HÌNH CẨU TẠO KÍNH HIỂN VI.....	77
4.5. MÀN HÌNH HƯỚNG DẪN THỰC HÀNH	78
4.6. MÀN HÌNH THỰC HÀNH.....	82
4.7. MÀN HÌNH KIỂM TRA	83
CHƯƠNG 5. TỔNG KẾT	85
5.1. KẾT LUẬN	85
5.2. HƯỚNG PHÁT TRIỂN ĐỀ TÀI.....	85
TÀI LIỆU THAM KHẢO	87

DANH MỤC HÌNH

Hình 1.1. Các bài thí nghiệm ảo môn hóa học của NoBook	3
Hình 1.2. Các bài thí nghiệm của PhET	4
Hình 1.3. Giao diện Chương trình mô phỏng “microscope” của NCBioNetwork.....	6
Hình 2.1 Một số game engine phổ biến hiện nay	8
Hình 2.2. Logo của Godot Engine.....	10
Hình 2.3. Ví dụ về scene trong Godot.....	11
Hình 2.4. Giao diện tạo mới một Node trong Godot.....	12
Hình 2.5. Giao diện màn hình Signals của Godot.....	13
Hình 2.6. Giao diện màn hình chỉnh sửa của Godot Engine	14
Hình 2.7. Giao diện màn hình kịch bản của Godot Engine.....	15
Hình 2.8. Ví dụ một số cú pháp GDScript trong Godot Engine	16
Hình 2.9. Giao diện chuyển đổi văn bản thành giọng nói Voice Maker của FPT.AI ...	18
Hình 3.1. Giao diện thiết kế các nút có hình ảnh minh họa trong Photoshop.....	19
Hình 3.2. Giao diện thiết kế các thanh trượt trong Photoshop	20
Hình 3.3. Giao diện thiết kế hộp thoại trong Photoshop	20
Hình 3.4. Tài nguyên hình ảnh các vật mẫu được soi dưới kính hiển vi.....	21
Hình 3.5. Giao diện thiết kế tài liệu hướng dẫn trong Photoshop	21
Hình 3.6. Kết quả sau khi xây dựng mô hình 3D trong Blender.....	22
Hình 3.7. Giao diện Shading khi thiết kế vật liệu cho vật kính trong Blender	23
Hình 3.8. Giao diện Texture Paint cho công tắc bật đèn cả kính hiển vi	24
Hình 3.9. Mô hình kính hiển vi hoàn chỉnh.....	24
Hình 3.10. Các thông số khi xuất (export) mô hình kính hiển vi.....	25
Hình 3.11. Kết quả sau khi import mô hình kính hiển vi vào Godot Engine.....	25
Hình 3.12. Giao diện scene login thiết kế ở Photoshop	26
Hình 3.13. Sơ đồ cấu trúc cây của màn hình đăng nhập	27
Hình 3.14. Sơ đồ luồng xử lý của Scene Login.....	28
Hình 3.15. Giao diện màn hình chính thiết kế trong Photoshop	31
Hình 3.16. Sơ đồ cấu trúc cây của màn hình chính.....	31
Hình 3.17. Scene Main thiết kế trong Godot Engine	32
Hình 3.18. Scene btn_menu trong Godot	37

Hình 3.19. Màn hình vật liệu và hóa chất thiết kế trong Photoshop	39
Hình 3.20. Sơ đồ cấu trúc cây của màn hình vật liệu và hóa chất.....	40
Hình 3.21. Màn hình vật liệu và hóa chất trong Godot Engine.....	41
Hình 3.22. Sơ đồ luồng chuyển đổi các trạng thái ở màn hình vật liệu và hóa chất....	42
Hình 3.23. Màn hình cấu tạo kính hiển vi được thiết kế trên Photoshop.....	46
Hình 3.24. Sơ đồ cấu trúc cây của màn hình cấu tạo kính hiển vi	47
Hình 3.25. Màn hình cấu tạo kính hiển vi được thiết kế trên Godot Engine	48
Hình 3.26. Sơ đồ luồng chuyển đổi trạng thái của màn hình cấu tạo kính hiển vi	48
Hình 3.27. Giao diện hướng dẫn thực hành thiết kế ở Photoshop	51
Hình 3.28. Sơ đồ các trạng thái trong màn hình hướng dẫn thực hành.....	52
Hình 3.29. Màn hình thực hành được thiết kế trog Godot Engine	59
Hình 3.30. Giao diện trình xử lý shader của GodotEngine	63
Hình 3.31. Màn hình kiểm tra được thiết kế trên Godot Engine.....	66
Hình 3.32. Màn hình chứng chỉ được thiết kế trong Godot Engine	70
Hình 3.33. Giấy chứng chỉ được thiết kế trong Godot Engine.....	70
Hình 3.34. Sơ đồ cấu trúc cây của màn hình chứng chỉ	71
Hình 3.35. Màn hình loading được thiết kế trong Godot Engine.....	74
Hình 4.1. Giao diện đăng nhập.....	76
Hình 4.2. Giao diện thông báo khi nhập sai thông tin mã số sinh viên.....	76
Hình 4.3. Dialog giới thiệu bài thực hành kính hiển vi.....	76
Hình 4.4. Thẻ tên của bài học vật liệu và hóa chất.....	76
Hình 4.5. Giới thiệu nội dung bài học vật liệu và hóa chất	77
Hình 4.6. Giao diện màn hình vật liệu và hóa chất	77
Hình 4.7. Dialog hiển thị thông tin và chức năng của vật liệu và hóa chất.....	77
Hình 4.8. Sau khi hoàn thành bài học vật liệu và hóa chất	77
Hình 4.9. Giao diện giới thiệu bài học cấu tạo kính hiển vi.....	78
Hình 4.10. Giao diện bài thực hành cấu tạo kính hiển vi	78
Hình 4.11. Giao diện chi tiết của một bộ phần trong kính hiển vi	78
Hình 4.12. Giao diện thông báo hoàn thành bài học	78
Hình 4.13. Giao diện giới thiệu của bài hướng dẫn thực hành.....	78
Hình 4.14. Giao diện giới thiệu bước 1 của màn hình hướng dẫn thực hành	78
Hình 4.15. Giao diện yêu cầu cảm điện của bài hướng dẫn thực hành	79

Hình 4.16. Giao diện yêu cầu bật công tắc điện của bài hướng dẫn thực hành	79
Hình 4.17. Đường truyền ánh sáng của kính hiển vi (tài liệu hướng dẫn).....	79
Hình 4.18. Giao diện giới thiệu bước 2 của màn hình hướng dẫn thực hành	79
Hình 4.19. Giao diện yêu cầu người dùng đặt tiêu bản vào kính hiển vi.....	79
Hình 4.20. Giao diện yêu cầu người dùng nhìn vào thị kính	79
Hình 4.21. Những lưu ý khi thực hành điều chỉnh ánh sáng trong kính hiển vi	80
Hình 4.22. Yêu cầu tăng ánh sáng của kính hiển vi	80
Hình 4.23. Giao diện thông báo người dùng đã điều chỉnh ánh sáng quá mạnh.....	80
Hình 4.24. Giao diện yêu cầu điều chỉnh lại ánh sáng và nhấn nút kiểm tra	80
Hình 4.25. Giao diện giới thiệu bước 3 của bài hướng dẫn thực hành	81
Hình 4.26. Cách tính độ phóng đại của kính hiển vi	81
Hình 4.27. Giao diện hướng dẫn điều chỉnh lấy nét kính hiển vi	81
Hình 4.28. Giao diện kiểm tra lấy nét mẫu vật của kính hiển vi.....	81
Hình 4.29. Giao diện giới thiệu thực hành với vật kính 100X	82
Hình 4.30. Tài liệu hướng dẫn nhỏ dầu soi kính hiển vi	82
Hình 4.31. Yêu cầu người dùng nhỏ dầu soi kính.....	82
Hình 4.32. Animation nhỏ dầu soi kính	82
Hình 4.33. Giao diện thực hiện điều chỉnh lấy nét.....	82
Hình 4.34. Giao diện thông báo đã hoàn thành bài học hướng dẫn thực hành	82
Hình 4.35. Giao diện danh sách mẫu vật có trong hộp đựng tiêu bản	83
Hình 4.36. Giao diện bảng điều khiển các chức năng trong kính hiển vi	83
Hình 4.37. Giao diện màn hình kiểm tra kiến thức	83
Hình 4.38. Giao diện màn hình khi người dùng lựa chọn các câu hỏi	83
Hình 4.39. Giao diện nộp bài khi điền hết 5 câu hỏi trắc nghiệm.....	84
Hình 4.40. Giao diện thông báo điểm bài kiểm tra thực hành của người dùng	84
Hình 4.41. Giao diện đã hoàn thành bài kiểm tra kiến thức.....	84
Hình 4.42. Giao diện nhận giấy chứng nhận đã hoàn thành bài thực hành ảo	84

DANH MỤC BẢNG

Bảng 2.1. So sánh một số tính năng của các game Engine phổ biến9

DANH MỤC CÁC KÝ HIỆU, TỪ VIẾT TẮT

2D	Two Dimensions (Không gian 2 chiều)
3D	Three Dimensions (Không gian 3 chiều)
E-learning	Electronic Learning (Đào tạo trực tuyến)
GPU	Graphics Processing Unit (Đơn vị xử lý đồ họa)
JPG	Joint Photographic Experts Group (Tiêu chuẩn chung cho định dạng ảnh)
HDD	Hard Disk Drive (Ổ cứng cơ học)
MP3	MPEG-1 Audio Layer III (một dạng âm thanh được mã hóa PCM - pulse-code modulation)
PC	Personal Computer (Máy tính cá nhân)
PNG	Portable Network Graphics (Đồ họa mạng di động)
RAM	Random Access Memory (Bộ nhớ tạm thời)
SSD	Solid State Drive (Ổ cứng bán dẫn)

LỜI MỞ ĐẦU

Trong thời đại 4.0 hiện nay, việc áp dụng khoa học công nghệ vào giảng dạy đã trở thành một xu hướng không thể thiếu. Sự phát triển của Công nghệ Thông tin và truyền thông đã mở ra nhiều cơ hội mới để nâng cao hiệu quả trong quá trình học tập và giảng dạy. Ví dụ, việc sử dụng Internet cho học tập và làm việc trực tuyến [1], ứng dụng công nghệ thực tế ảo tăng cường AR trong dạy học trực tuyến [2]. Công nghệ đã mang đến cho người dùng những tiện ích, khả năng tương tác và khám phá vô cùng mới mẻ.

Điều đáng chú ý là đại dịch COVID-19 diễn ra vào đầu năm 2020 đã tác động thúc đẩy khoa học công nghệ nhanh hơn bao giờ hết, đặc biệt là trong lĩnh vực giáo dục. Dịch bệnh đã gây ảnh hưởng lớn đến quá trình học tập của học sinh và sinh viên, khiến việc tiếp cận trường lớp và các phòng thí nghiệm trở nên khó khăn. Học trực tuyến và làm việc từ xa đã trở thành phương pháp thay thế chính để duy trì quá trình giảng dạy trong thời gian giãn cách xã hội. Điều này đã đặt ra một thách thức lớn đối với việc cung cấp môi trường thực hành thích hợp cho học tập. Việc xây dựng các bài thực hành ảo đáp ứng nhu cầu học tập và thực hành của học sinh và sinh viên ngày càng trở nên quan trọng hơn bao giờ hết.

Nắm bắt được nhu cầu cấp bách đối với bài thực hành ảo, đề tài nghiên cứu “Ứng dụng Godot xây dựng bài thực hành kính hiển vi ảo cho phòng thí nghiệm sinh học, đại học Nha Trang” ra đời, tập trung vào việc thiết kế mô phỏng bài thực hành kính hiển vi ảo sử dụng Godot Engine (phiên bản 3.5.1) - một công cụ phát triển ứng dụng đa nền tảng mạnh mẽ. Đề tài sẽ trình bày về các công cụ và kỹ thuật được sử dụng để xây dựng môi trường thực hành ảo, cùng với những lợi ích mà việc áp dụng công nghệ này mang lại. Đề tài hướng đến mục tiêu tạo ra một phòng thí nghiệm sinh học ảo chất lượng cao, cung cấp một cách học tập mới và sáng tạo cho học sinh và sinh viên.

Kết quả thu được từ việc thực hiện chương trình mô phỏng “Ứng dụng godot xây dựng bài thực hành kính hiển vi ảo cho phòng thí nghiệm sinh học, đại học Nha Trang” đã đáp ứng hoàn toàn các yêu cầu đã đề ra. Chương trình này đã được thiết kế với sự chính xác cao trong việc tái hiện các vật liệu và hóa chất cần thiết, đồng thời mô phỏng đầy đủ các bước quan trọng trong kịch bản thí nghiệm. Giao diện tương tác của chương trình cũng được thiết kế một cách rõ ràng và dễ sử dụng, đảm bảo rằng sinh viên có thể tiếp cận nhanh chóng và thuận tiện.

Cuối cùng, mục tiêu thực sự của đồ án muốn nhấn mạnh rằng bài thực hành ảo mang lại nhiều lợi ích vượt xa so với các bài thực hành truyền thống. Việc sử dụng bài thực hành ảo không chỉ nâng cao hiệu quả học tập, mà còn tiết kiệm thời gian, tài nguyên và mở ra cơ hội học tập không giới hạn. Đây là một công cụ giáo dục đầy tiềm năng, giúp người học phát triển kỹ năng và hiểu sâu hơn về các môn học khác nhau.

CHƯƠNG 1. TỔNG QUAN VỀ ĐỀ TÀI

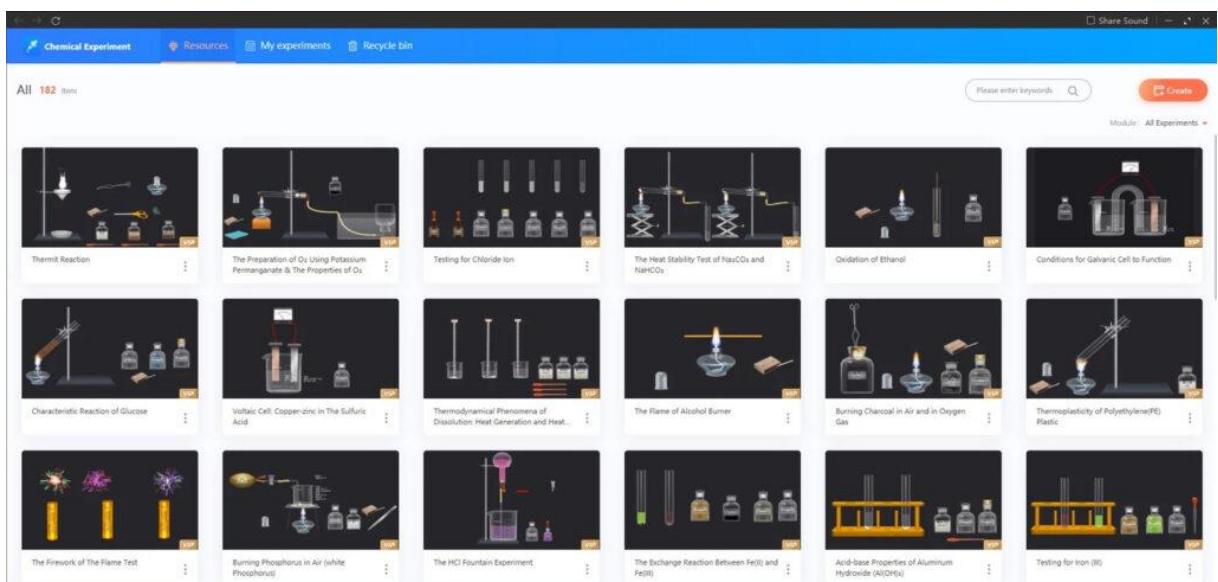
1.1. THỰC TRẠNG

Ngày nay, các chương trình mô phỏng và thí nghiệm ảo đang được áp dụng rộng rãi trong hệ thống đào tạo trực tuyến (E-learning) của các trường phổ thông và Đại học trên toàn cầu. Việc kết hợp giữa bài học và thí nghiệm ảo giúp tăng tính sinh động và theo dõi tiến độ học tập của sinh viên. Các chương trình mô phỏng cũng giúp sinh viên hứng thú và hình dung rõ hơn về kiến thức đã học. Sử dụng môi trường ảo không đòi hỏi thiết bị phức tạp và giúp giảm chi phí. Ngoài ra, việc sử dụng các bài thực hành ảo giúp sinh viên tích lũy kinh nghiệm trước khi thực hành thực tế. Phương pháp này đang thay thế cách học một chiều, khuyến khích sự tương tác và tính chủ động của người học. Việc ứng dụng các chương trình mô phỏng, thí nghiệm ảo vào giảng dạy là một xu hướng mới trong lĩnh vực giáo dục [3], [4].

1.2. CÁC SẢN PHẨM NỔI BẬT VỀ CHỦ ĐỀ PHÒNG THÍ NGHIỆM ẢO

Phòng Thí Nghiệm Ảo Nobook

Nobook là một sản phẩm thuộc công ty ClassIn, cung cấp nền tảng Virtual Lab (phòng thí nghiệm ảo) cho giáo viên thực hiện các thí nghiệm ảo. Với hiệu ứng hình ảnh và chuyển động, NoBook là phần mềm thí nghiệm ảo chuyên nghiệp và hỗ trợ dạy học tương tác.



Hình 1.1. Các bài thí nghiệm ảo môn hóa học của NoBook (<https://classin.com.vn/>)

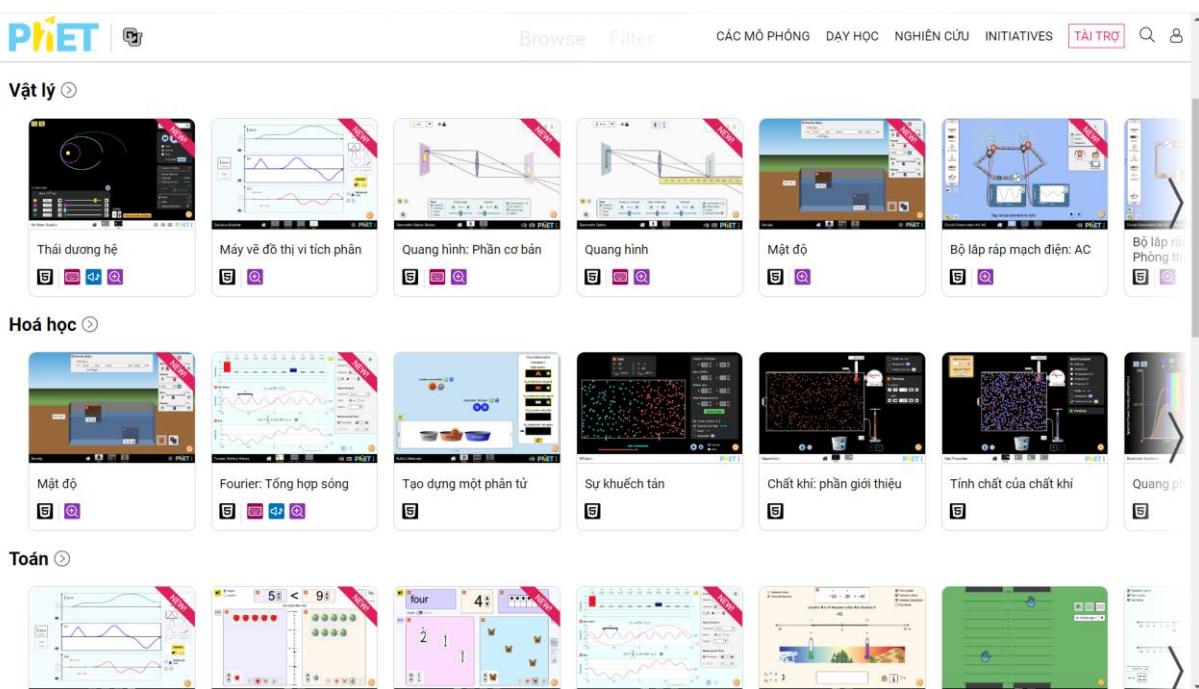
Phần mềm NoBook bao gồm vật lý, hóa học và sinh học trong chương trình học từ cấp tiểu học đến trung học cơ sở. Trong lĩnh vực vật lý, NoBook hỗ trợ nhiều loại thí

nghiệm và minh họa bằng ảnh động, với hơn 1.000 tài nguyên thí nghiệm và 300 công cụ thí nghiệm. Về hóa học, NoBook cung cấp hình ảnh trực quan về các phản ứng hóa học để giúp học sinh hiểu rõ hơn về khái niệm và quy trình hoạt động. Đồng thời, tính năng này cũng giúp giáo viên và học sinh tạo và thực hiện các thí nghiệm hóa học mà không cần lo lắng về vấn đề an toàn.

NoBook có nhiều ưu điểm về chức năng và tính hoàn thiện. Tuy nhiên, hỗ trợ tiếng Việt của nó còn hạn chế. Để sử dụng NoBook, người dùng cần tải chương trình về và cài đặt trên máy tính, với dung lượng yêu cầu gần 1GB bộ nhớ ngoài. Phần mềm chủ yếu tập trung vào vật lý và hóa học. Hơn nữa, NoBook là một sản phẩm thương mại và để sử dụng các chức năng đầy đủ, người dùng phải trả một khoản phí hàng tháng là 1 triệu 300 nghìn đồng.

Dự Án PhET Interactive Simulations

Dự án PhET Interactive Simulations ra đời vào năm 2002, được khởi xướng bởi Carl Wieman - một nhà Vật lý vinh dự đoạt giải Nobel. Mục tiêu chính của dự án là tạo ra những mô phỏng tương tác miễn phí trong lĩnh vực Toán học và các ngành khoa học khác. PhET mang đến cho người sử dụng một loạt mô phỏng đa dạng trong các lĩnh vực như Toán học, Vật lý, Hóa học, Sinh học và Khoa học Trái đất. Những mô phỏng này giúp hình dung các khái niệm một cách trực quan, cho phép người học tương tác trực tiếp trên màn hình thông qua các hoạt động như kéo-thả, thanh trượt và lựa chọn.



Hình 1.2. Các bài thí nghiệm của PhET (<https://phet.colorado.edu/vi/>)

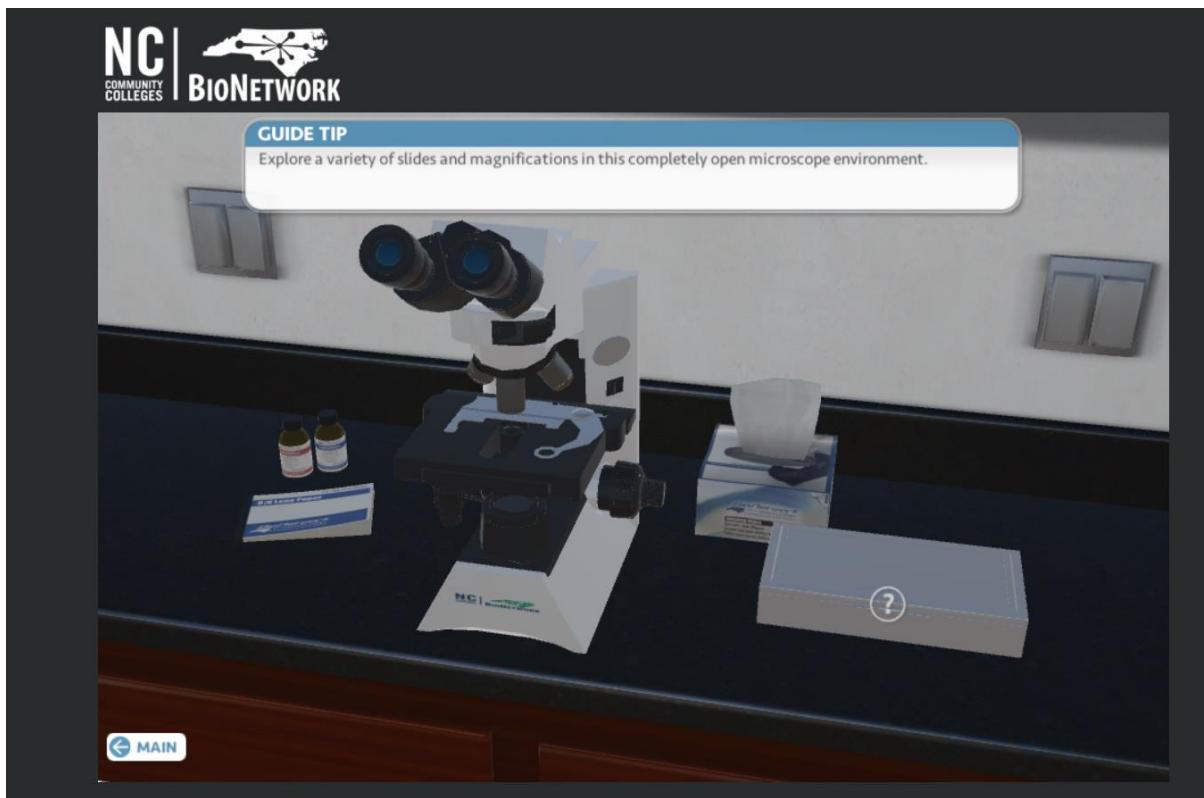
Đối với học sinh ở cấp trung học cơ sở và trung học phổ thông, họ có thể điều chỉnh các thông số và sử dụng các công cụ đo tương tự như trong thực tế, như thước đo, đồng hồ bấm giờ, volt kế, nhiệt kế... để hiểu về các quá trình có tính định lượng. Người dùng có thể truy cập vào trang web <https://phet.colorado.edu> hoặc tải ứng dụng PhET lên các thiết bị di động như iPad. Tuy nhiên, việc tải ứng dụng lên các thiết bị di động sẽ đòi hỏi một khoản phí là 22.000 đồng.

PhET được coi là một điểm sáng trong việc cung cấp các bài thực hành ảo miễn phí, nhanh chóng, tiện lợi, có hỗ trợ tiếng Việt nhưng cũng còn hạn chế. Giao diện đồ họa vẫn còn đơn giản, thô sơ. Các hình ảnh 2D không được chụp từ các vật mẫu thật, phần lớn là được vẽ lại. Các phần thực hành và lý thuyết được tách rời, yêu cầu người học phải đọc qua lý thuyết trước khi vào thực hành.

Chương trình mô phỏng “microscope” của NCBioNetwork

NCBioNetwork là tổ chức được thành lập vào năm 2004 thông qua sự tài trợ của Golden LEAF Foundation - một trong những tổ chức cung cấp các khóa học, hội thảo phát triển kỹ năng và các chương trình chứng chỉ về công nghệ sinh học, sản xuất dược phẩm và hóa chất. Chương trình mô phỏng “kính hiển vi” của NCBioNetwork là một trong những chương trình mô phỏng nổi tiếng được thiết kế và lập trình bởi đội ngũ của NCBioNetwork. Chương trình mô phỏng “Sử dụng kính hiển vi” cung cấp một cái nhìn chi tiết về cách hoạt động của kính hiển vi thông qua việc tương tác với một kính hiển vi 3D ảo có đầy đủ chức năng.

Đây là một công cụ rất hữu ích để chuẩn bị cho công việc làm việc trong phòng thí nghiệm khoa học. Có thể tìm hiểu thuật ngữ, cách sử dụng và cách chăm sóc kính hiển vi thông qua chương trình mô phỏng này. Một trong những ưu điểm của chương trình mô phỏng này là cung cấp một môi trường an toàn để thực hành mà không cần thực tế sử dụng một kính hiển vi thực tế. Có thể tương tác với kính hiển vi ảo và thực hiện các thao tác như điều chỉnh thấu kính, thay đổi độ phóng đại và di chuyển mẫu.



*Hình 1.3. Giao diện Chương trình mô phỏng “microscope” của NCBioNetwork
(<https://www.ncbionetwork.org/iet/microscope/>)*

Tuy có rất nhiều ưu điểm, nhưng chương trình mô phỏng vẫn còn một số hạn chế. Độ phân giải của các mô hình 3D vẫn còn thấp, chương trình có tông màu u tối, lý thuyết khô khanganh, không hỗ trợ giọng nói để tránh nhảm chán, không thể phóng to toàn màn hình để dễ quan sát, khi chỉnh thông số của kính để quan sát vật mẫu vẫn còn dễ đoán và các thông số qua các vật mẫu vẫn còn cố định.

Tổng kết, phần lớn các chương trình mô phỏng hiện nay đều được phát triển bởi các tổ chức nước ngoài và chỉ một số ít trong số đó hỗ trợ ngôn ngữ Tiếng Việt. Điều này tạo ra khó khăn gián tiếp cho người học trong việc tiếp cận các chương trình mô phỏng và thí nghiệm ảo. Các bài học chất lượng cao yêu cầu phải trả một mức giá khá cao hàng tháng để được học, trong khi đó các bài học miễn phí hỗ trợ nền tảng web thì có chất lượng khá thấp. Để giải quyết vấn đề này, việc xây dựng các phòng thí nghiệm ảo có thể đóng vai trò quan trọng trong việc đáp ứng nhu cầu giảng dạy và học tập của người học trong nước. Các phòng thí nghiệm này sẽ được phát triển với mục tiêu hỗ trợ ngôn ngữ Tiếng Việt và đáp ứng được đa dạng các yêu cầu và nhu cầu học tập. Qua đó, sẽ tạo điều kiện thuận lợi hơn cho việc ứng dụng chương trình mô phỏng, thí nghiệm ảo vào giảng dạy và nghiên cứu tại Việt Nam.

CHƯƠNG 2. CƠ SỞ LÝ THUYẾT

2.1. THÍ NGHIỆM ẢO

Thí nghiệm ảo là một khái niệm chỉ quá trình thực hiện các thí nghiệm hoặc mô phỏng các hiện tượng trong một môi trường ảo, thay vì thực hiện trực tiếp trong thực tế. Thí nghiệm ảo sử dụng các công nghệ ảo hóa để tạo ra một môi trường giả lập, trong đó các điều kiện và thông số có thể điều chỉnh và mô phỏng một cách linh hoạt. Thí nghiệm ảo có thể được thực hiện trong các lĩnh vực khác nhau như khoa học, kỹ thuật, y học, giáo dục, và nhiều lĩnh vực khác. Thông qua thí nghiệm ảo, người ta có thể nghiên cứu và kiểm tra các giả định, mô hình, và kịch bản khác nhau mà không cần đến các thiết bị và tài nguyên vật lý. Điều này có thể giúp tiết kiệm thời gian, chi phí và giảm rủi ro trong quá trình thí nghiệm.

Trong quá trình giảng dạy, Thí nghiệm ảo đem lại nhiều lợi ích đáng kể. Đầu tiên, Thí nghiệm ảo cung cấp một môi trường học tập tương tác và hấp dẫn cho sinh viên. Thông qua các chương trình mô phỏng và thí nghiệm ảo, sinh viên có thể tương tác trực tiếp với các hiện tượng và quá trình phức tạp, giúp họ hiểu rõ hơn và hứng thú với các khái niệm khoa học.

Thứ hai, thí nghiệm ảo cung cấp một phạm vi thực hành rộng hơn và linh hoạt hơn. Thay vì chỉ có thể thực hiện các thí nghiệm trong giới hạn của phòng thí nghiệm vật lý, sinh viên có thể thực hành trên các môi trường ảo mô phỏng nhiều lĩnh vực khác nhau như hóa học, sinh học, vật lý hạt nhân và nhiều hơn nữa. Điều này mở ra cơ hội cho sinh viên khám phá và áp dụng kiến thức vào các tình huống thực tế.

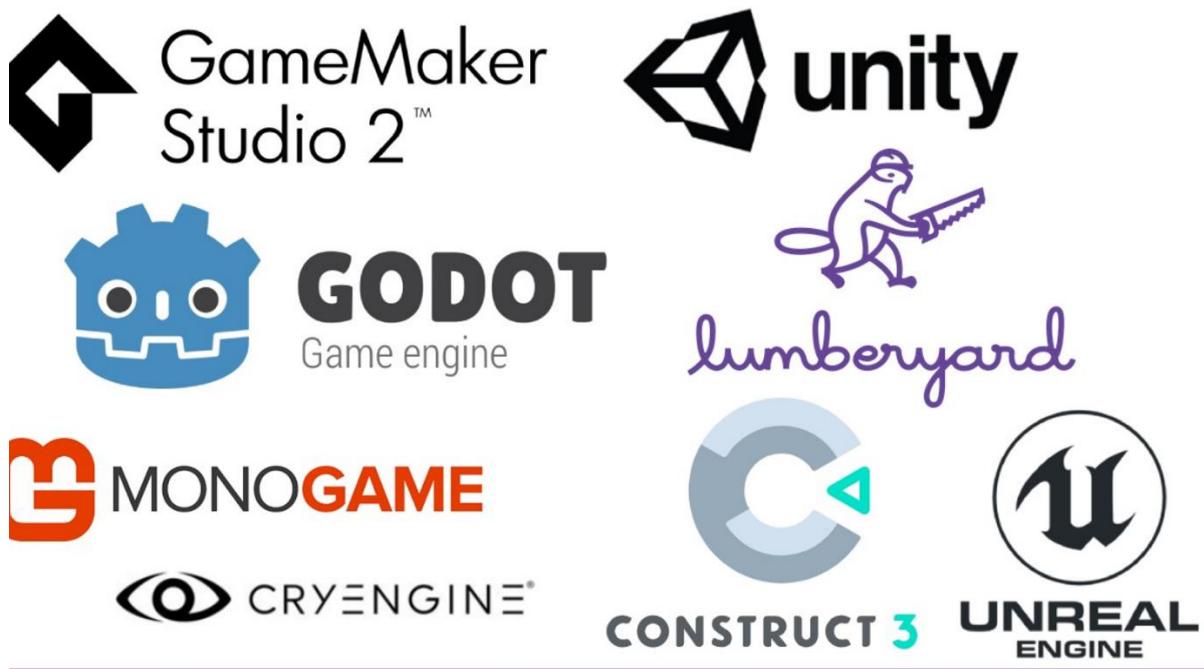
Thứ ba, thí nghiệm ảo giúp sinh viên nắm vững và áp dụng kiến thức một cách chính xác. Qua việc tương tác với các đối tượng và thực hiện các thí nghiệm trong môi trường ảo, sinh viên có thể thấy được kết quả và hậu quả của các hành động và thay đổi các thông số. Điều này giúp họ hiểu rõ hơn về các khái niệm và quá trình khoa học, cũng như phát triển kỹ năng vận dụng và tư duy phản biện.

Cuối cùng, thí nghiệm ảo tiết kiệm chi phí và tài nguyên. Thay vì phải mua và duy trì các thiết bị và vật liệu phục vụ thí nghiệm, giảng viên và sinh viên chỉ cần truy cập vào chương trình mô phỏng hoặc môi trường ảo thông qua máy tính kết nối Internet.

Tóm lại, thí nghiệm ảo là một công cụ hữu ích trong giảng dạy, mang lại lợi ích tương tác, phạm vi thực hành rộng hơn, sự hiểu biết sâu sắc và tiết kiệm chi phí.

2.2. PHẦN MỀM GAME (GAME ENGINE)

Game Engine là một phần mềm hoặc bộ công cụ được sử dụng để phát triển và xây dựng các trò chơi điện tử, cung cấp một nền tảng cho các nhà phát triển để tạo ra các trò chơi bao gồm các chức năng và tính năng quan trọng như đồ họa, âm thanh, vật lý và cơ chế trò chơi (GamePlay). Lịch sử phát triển Game Engine bắt đầu từ những năm 1980, nhưng chỉ khi công nghệ đồ họa 3D phát triển vào những năm 1990, các Game Engine trở nên quan trọng và phổ biến trong ngành công nghiệp trò chơi. Mục đích chính của việc tạo ra Game Engine là cung cấp một nền tảng để phát triển trò chơi một cách hiệu quả (Hình 1.1). Mỗi công cụ phát triển có ưu và nhược điểm khác nhau và được liệt kê ở Bảng 1.1.



Hình 2.1 Một số game engine phổ biến hiện nay

(<https://www.gamesindustry.biz/what-is-the-best-game-engine-for-your-game>)

Để lựa chọn Game Engine phù hợp cho dự án, cần xem qua các tiêu chí như ngôn ngữ lập trình, mức độ hỗ trợ, đa nền tảng và sự dễ tiếp cận, chi phí bắt đầu thấp. Godot Engine là một lựa chọn tối ưu và phù hợp với các tiêu chí đặt ra.

Đầu tiên, ngôn ngữ lập trình trong Godot là GDScript, một ngôn ngữ dễ hiểu và dễ học, tương tự với Python, điều này giúp giảm thiểu thời gian và công sức cần thiết để phát triển trò chơi.Thêm vào đó Godot Engine có cả trình chỉnh sửa tệp lệnh tích hợp (Built-in script editor) và trình gỡ lỗi GDScript (GDScript debugger).

Mức độ hỗ trợ của Godot Engine cũng là một yếu tố quan trọng. Cộng đồng người dùng của Godot rất lớn và tích cực, với nhiều diễn đàn, tài liệu và nguồn tài nguyên trực

tuyến. Bất kỳ khi nào gặp vấn đề hoặc cần giúp đỡ, có thể dễ dàng tìm thấy sự hỗ trợ từ cộng đồng Godot (<https://www.facebook.com/groups/godotengine>).

Bảng 2.1. So sánh một số tính năng của các game Engine phổ biến

Tính năng	Unity	Unreal Engine	Godot
Hỗ trợ 2D/3D	Có	Có	Có
Ngôn ngữ lập trình	C#	C++, Python	GDScript, Visual Scripting, C#, C++
Hỗ trợ đa nền tảng	PC, smartphone, console, web	PC, di động, console	PC, smartphone, console, web
Giấy phép phát hành	Phiên bản thương mại và miễn phí	Phiên bản thương mại và miễn phí	Mã nguồn mở, miễn phí
Cấu hình tối thiểu	4GB RAM, 20GB HDD/SSD	8GB RAM, 50GB HDD/SSD	4GB RAM, 1GB+ HDD/SSD

Đa nền tảng là một yêu cầu quan trọng cho dự án game ngày nay, và Godot Engine đáp ứng tốt yêu cầu này. Godot hỗ trợ phát triển trò chơi cho nhiều nền tảng như Windows, MacOS, Linux, iOS, Android và cả các hệ máy chơi game như PlayStation và Xbox. Điều này cho phép tiếp cận một đối tượng khán giả rộng lớn trên nhiều thiết bị khác nhau.

Một lợi thế khác của Godot Engine là sự dễ tiếp cận. Giao diện người dùng được thiết kế đơn giản và thân thiện với người dùng, giúp cho người mới bắt đầu dễ dàng làm quen với công cụ này. Cung cấp một bộ công cụ đồ họa mạnh mẽ cho việc xây dựng và chỉnh sửa các tài nguyên, từ hình ảnh, âm thanh đến các cấu trúc logic trong trò chơi.

Godot Engine được phát hành miễn phí, mã nguồn mở giúp tiết kiệm chi phí phát triển và cung cấp khả năng tùy chỉnh linh hoạt. Không cần phải đầu tư một số tiền lớn vào công cụ phát triển trò chơi, điều này đặc biệt quan trọng đối với các nhà phát triển độc lập hoặc các dự án với nguồn lực hạn chế.

2.3. GODOT ENGINE (PHIÊN BẢN 3.5.1)

Godot là một game engine được khởi đầu bởi Juan 'reduz' Linietsky và Ariel 'punto' Manzur vào năm 2007. Đến ngày 15 tháng 12 năm 2014, Godot đã đạt đến phiên bản 1.0, đánh dấu bản phát hành ổn định đầu tiên.



Hình 2.2. Logo của Godot Engine (<https://godotengine.org/>)

Ngày nay, Godot Engine là một công cụ phát triển trò chơi đa nền tảng, tích hợp đầy đủ tính năng để tạo ra cả trò chơi 2D và 3D từ một giao diện thống nhất. Godot Engine đi kèm với một trình soạn thảo game hoàn chỉnh với các công cụ tích hợp để đáp ứng các yêu cầu phổ biến nhất bao gồm: một trình soạn thảo mã, một trình soạn thảo hoạt hình (animation), một trình soạn thảo đồ họa (shader), một trình gỡ lỗi (debugger), một trình phân tích và nhiều hơn nữa. Godot cung cấp cho người dùng một bộ công cụ phong phú, giúp họ tập trung vào việc tạo ra trò chơi mà không phải làm lại những công việc đã có sẵn. Godot yêu cầu cấu hình tối thiểu bao gồm RAM 4GB, Bộ nhớ Trò chơi phát triển trong Godot có thể được xuất bản lên nhiều nền tảng khác nhau, bao gồm máy tính cá nhân (Linux, macOS, Windows), thiết bị di động (Android, iOS), cũng như các nền tảng dựa trên web và máy chơi game.

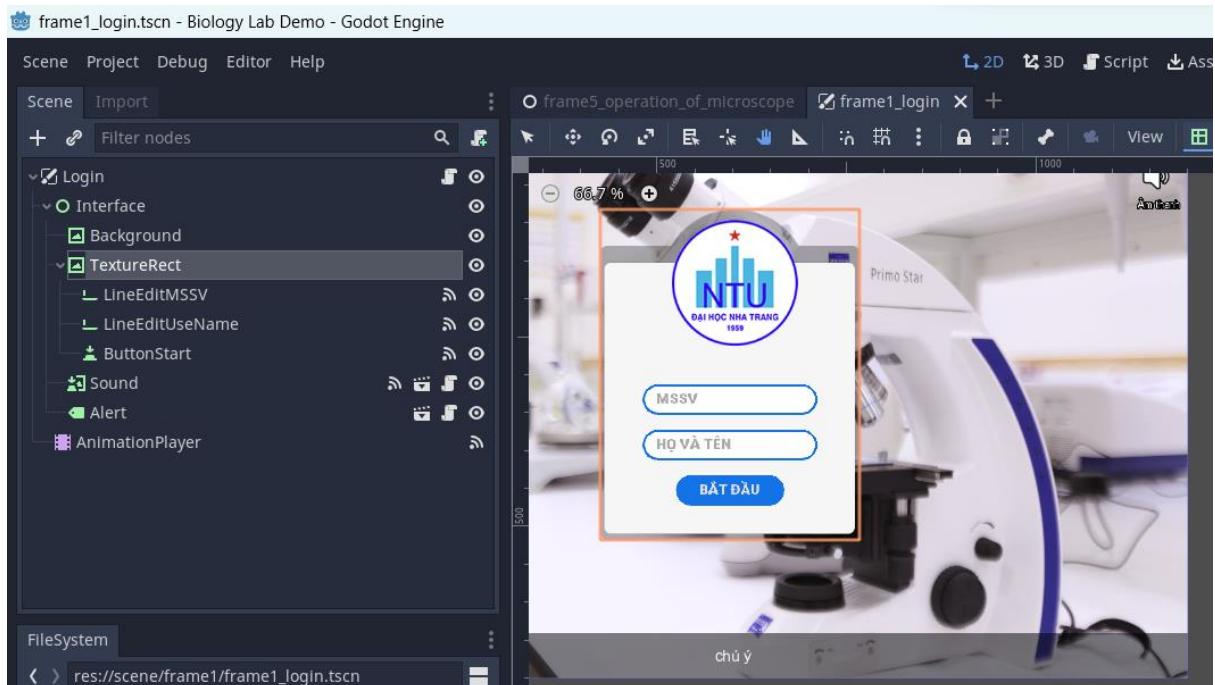
Đặc biệt, Godot là một công cụ hoàn toàn miễn phí và mã nguồn mở dưới giấy phép MIT, cho phép người dùng sử dụng mà không bị ràng buộc. Sự phát triển của Godot hoàn toàn độc lập và được hỗ trợ bởi cộng đồng, giúp người dùng tham gia hình thành công cụ phù hợp với mong đợi của mình. Godot cũng được hỗ trợ bởi tổ chức phi lợi nhuận Godot Foundation.

2.3.1. Tổng quan về các khái niệm chính trong Godot

Mỗi Game Engine xoay quanh các định nghĩa trừu tượng (abstractions) được sử dụng để xây dựng ứng dụng. Trong Godot, một trò chơi được biểu diễn dưới dạng cây (tree) các nút (nodes) khi các node được nhóm lại thành các cảnh (scenes). Có thể kết nối các nút này để chúng có thể giao tiếp với nhau bằng các tín hiệu (signals).

Scenes: Trong Godot, có thể chia nhỏ trò chơi thành các cảnh có thể tái sử dụng. Một cảnh có thể là một nhân vật, một vũ khí, một menu trong giao diện người dùng, một ngôi nhà. Có thể lồng ghép các cảnh lại với nhau để tạo một cảnh lớn hơn. Ở hình 2.3 là

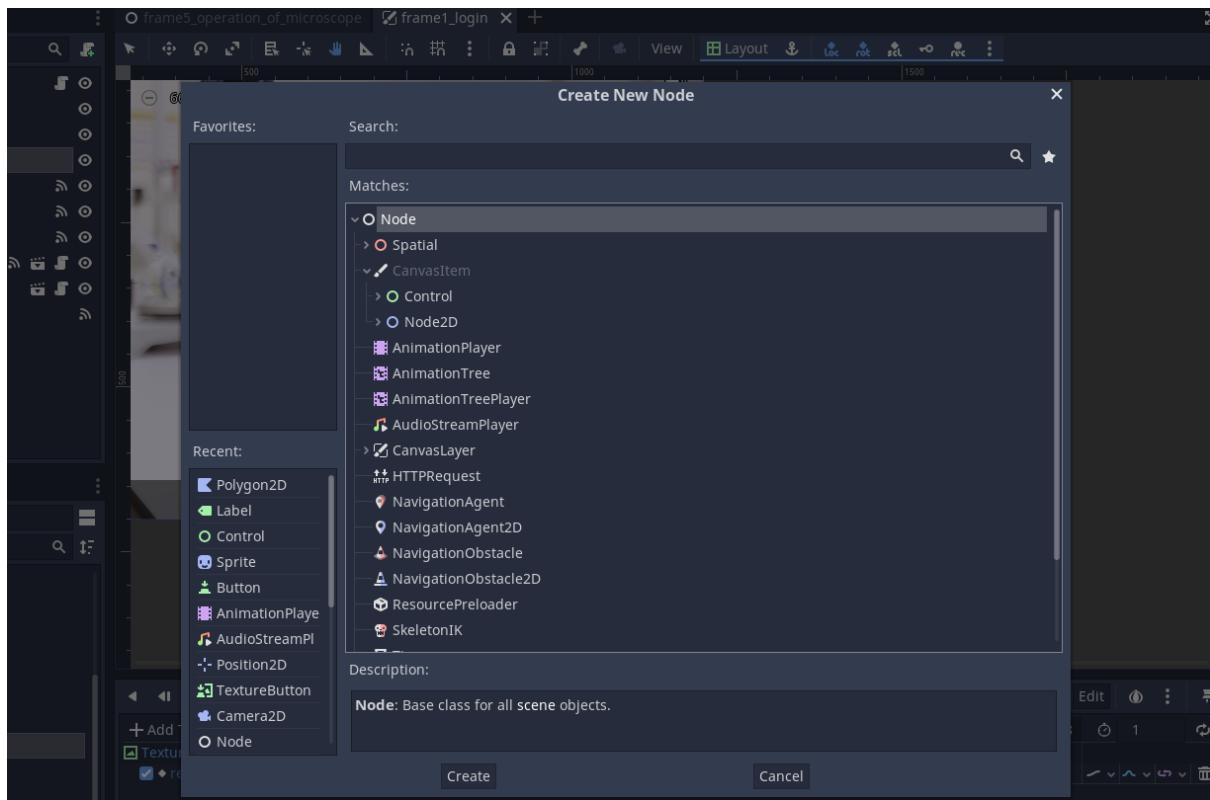
một ví dụ về màn hình đăng nhập (scenes login), trong scenes login sẽ có các đối tượng như: Interface (giao diện), TextureRect (đối tượng chủ biểu mẫu đăng nhập), Sound (âm thanh), Alert (thông báo), AnimationPlayer (hoạt hình). Sound và Alert là 2 scene được thêm vào scene login. Sound và Alert cũng được sử dụng ở các màn hình khác ở trong bài thực hành ảo “Kính hiển vi”, việc tạo 2 đối tượng thành scenes sẽ giúp tái sử dụng lại ở các scene khác.



Hình 2.3. Ví dụ về scene trong Godot

Scene Tree: Tất cả các cảnh của trò chơi được tổ chức thành cây cảnh (scene tree), nghĩa là một cây gồm các cảnh. Và vì các cảnh là các cây nút, cây cảnh cũng là một cây nút. Tuy nhiên, việc nghĩ về trò chơi dưới dạng các cảnh là dễ dàng hơn vì chúng có thể đại diện cho nhân vật, vũ khí, cửa ra vào hoặc giao diện người dùng.

Nodes: Một cảnh được tạo thành từ một hoặc nhiều nút. Các nút là những khối xây dựng nhỏ nhất của trò chơi để sắp xếp thành cây (tree).



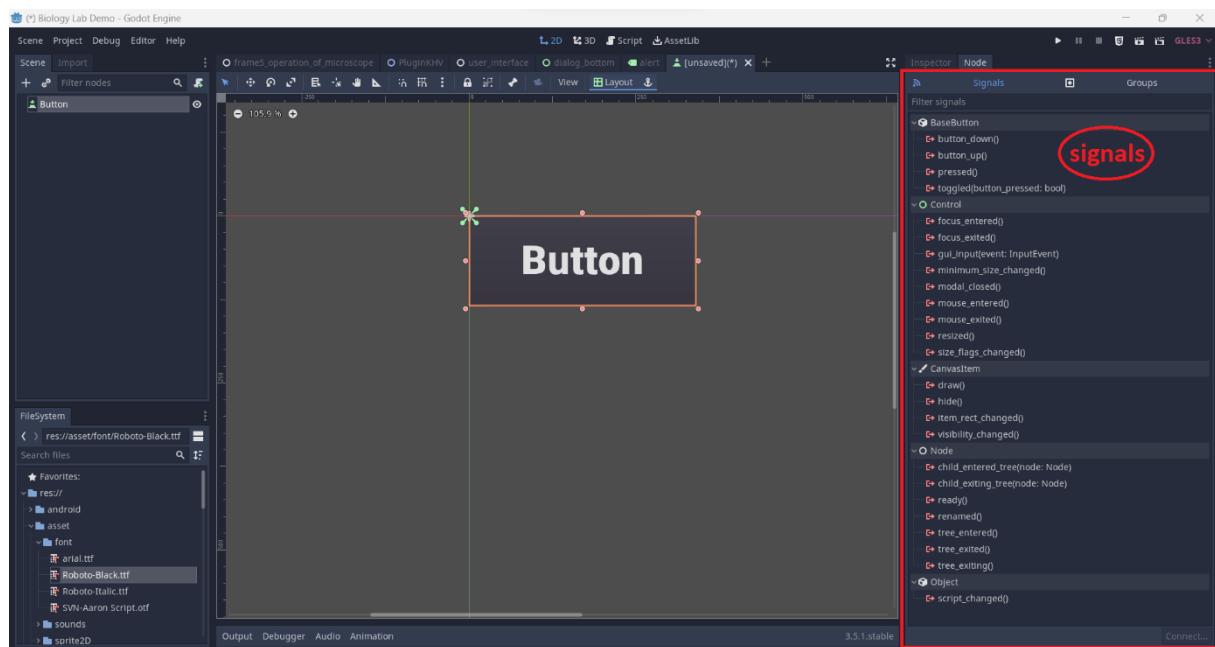
Hình 2.4. Giao diện tạo mới một Node trong Godot

Một số Node thông dụng:

- **TextureButton:** có chức năng tương tự như một nút bấm (button), nhưng TextureButton sử dụng họa tiết (texture) để hiển thị. Nói cách khác một nút TextureButton khi được khởi tạo sẽ không có hình dáng, phải nhập (import) hình ảnh vào với từng trạng thái của nút. Có 4 trạng thái của nút bao gồm: Normal (không nhấn), Pressed (đã nhấn nút), Hover (khi chuột di chuyển vào khu vực của nút), Disabled (nút bị tắt, không nhấn được). Mỗi trạng thái sẽ có một hình để tượng trưng cho trạng thái đó.
- **Sprite:** có chức năng hiển thị hình ảnh 2D, bao gồm các thao tác với hình ảnh như di chuyển (move), xoay (rotate), thu phóng (scale).
- **AudioStreamPlayer:** nút phát âm thanh. Hỗ trợ các hàm xử lý âm thanh: play (phát âm thanh), stop (dừng phát âm thanh), finished (trả về true khi kết thúc đoạn âm thanh đang phát), playing (trả về true khi âm thanh đang phát).
- **AnimationPlayer:** có chức năng chạy các hoạt hình được tạo sẵn. Các đoạn hoạt hình được tham chiếu bằng tên. VD: Muốn gọi đoạn hoạt hình login (đăng nhập) thì dùng lệnh - play('login').
- **CanvasPlayer:** được sử dụng để hiển thị giao diện người dùng trong game, giao diện này luôn hiển thị qua các màn chơi. CanvasPlayer có chỉ số lớp (layer index)

cho phép đánh số nguyên để định nghĩa thứ tự hiển thị. VD: nếu một nút CanvasPlayer có layer bằng -1 thì sẽ hiển thị ở dưới Node có layer bằng 1.

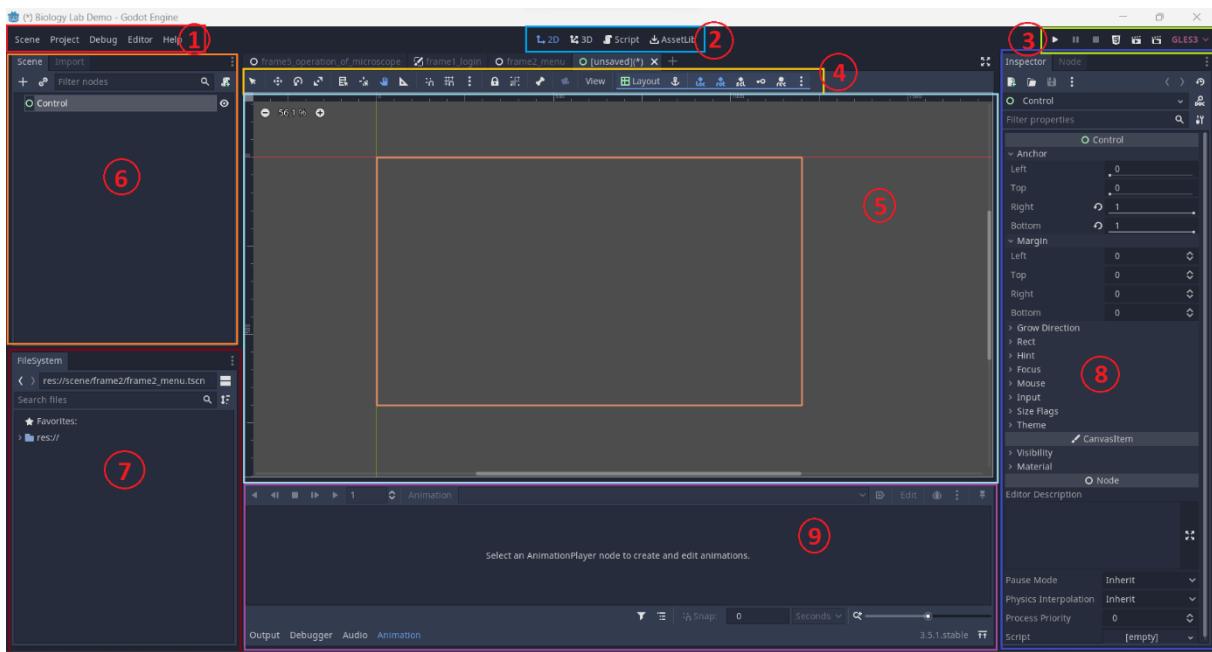
Signals: Các nút phát ra tín hiệu khi một sự kiện nào đó xảy ra. Tính năng này cho phép kết nối các nút để giao tiếp mà không cần viết mã cứng. Signals rất linh hoạt trong cách cấu trúc các cảnh. Các signals được tích hợp sẵn, có thể thông báo với các sự kiện hoặc hành động xảy ra trong chương trình như di chuyển chuột vào vùng chọn, sự kiện nhấp chuột (click), hoặc di chuyển chuột thoát khỏi vùng chọn, hoặc phát tín hiệu khi các vật thể chạm vào nhau. Một node sẽ có các signal khác nhau cho đặc trưng của mỗi node.



Hình 2.5. Giao diện màn hình Signals của Godot

2.3.2. Tổng quan giao diện của Godot

Godot hỗ trợ giao diện kéo thả trực quan, vị trí các công cụ được bố cục hợp lý, hiển thị đầy đủ thông tin, có thể thay đổi di chuyển vị trí các công cụ tùy ý.



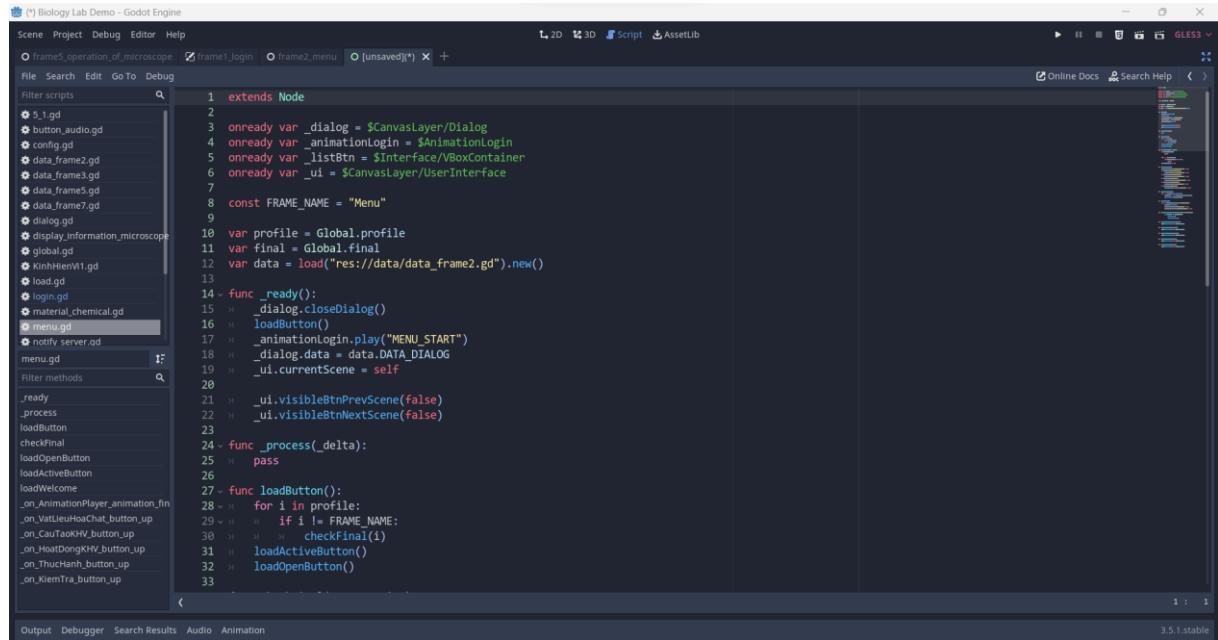
Hình 2.6. Giao diện màn hình chỉnh sửa của Godot Engine

Khi mở một dự án mới hoặc dự án hiện có, giao diện của trình soạn thảo sẽ xuất hiện **Hình 2.6**. Các thành phần chính trong giao diện chỉnh sửa của Godot:

- (1) **Menus**: chỉnh sửa các cài đặt, thiết lập, cấu hình môi trường lập trình game. Cho phép lưu game, xuất game ra các nền tảng khác, tài liệu hướng dẫn, v.v.
- (2) **Main screens**: chứa các nút dẫn tới các màn hình chính của trình chỉnh sửa bao gồm: màn hình chỉnh sửa 2D, màn hình chỉnh sửa 3D, màn hình chỉnh sửa code (Script), màn hình cung cấp các plugin, tài nguyên của Godot (AssetLib).
- (3) **Playtest buttons**: gồm các nút thực hiện chạy các màn hình game.
- (4) **Toolbar**: bao gồm các công cụ để chỉnh sửa, di chuyển các đối tượng trong màn hình game (lựa chọn – select, di chuyển - move, xoay - rotate, thu phóng - scale).
- (5) **Viewport**: nơi thao tác, chỉnh sửa, hiển thị, xóa các đối tượng trong game.
- (6) **Scene**: chứa các phân cảnh và các node đang hoạt động trong viewport.
- (7) **FileSystem**: quản lý các file của hệ thống, các phân cảnh (scenes), các kịch bản (scripts), âm thanh, hình ảnh, video, v.v.
- (8) **Inspector**: kiểm tra cho phép tùy chỉnh thuộc tính (properties) các node đã chọn.
- (9) **Bottom panel**: bao gồm các công cụ như màn hình gỡ lỗi (debug console), trình chỉnh sửa hoạt hình (animation editor), trình hòa trộn âm thanh (audio mixer).

2.3.3. Màn hình chỉnh sửa kịch bản của Godot

Màn hình kịch bản (Script screen) là một trình chỉnh sửa mã hoàn chỉnh với bộ gỡ lỗi (debugger), tính năng tự động hoàn thành mã phong phú và tham chiếu mã tích hợp sẵn.



Hình 2.7. Giao diện màn hình kịch bản của Godot Engine

Godot cung cấp bốn ngôn ngữ lập trình trò chơi: GDScript, Visual Scripting, C# và C++. Tuy nhiên Script editor chỉ hỗ trợ GDScript nên để sử dụng các ngôn ngữ khác cần chỉ dẫn godot chuyển sang một trình soạn thảo code khác để hỗ trợ (Visual Studio Code, Visual Studio, JetBrains Rider,...).

2.4. NGÔN NGỮ LẬP TRÌNH GODOT (GDSCRIPT)

GDScript là một ngôn ngữ lập trình hướng đối tượng và cấu trúc dành cho Godot. GDScript được tạo ra dành cho nhà phát triển game để tiết kiệm thời gian lập trình trò chơi. Các tính năng của GDScript bao gồm:

- Cú pháp đơn giản dẫn đến việc có các file mã ngắn gọn.
- Thời gian biên dịch và tải nhanh chóng.
- Tích hợp chặt chẽ với trình chỉnh sửa, với tính năng hoàn thành mã cho các nút, tín hiệu và thông tin khác từ cảnh mà được gắn kết.
- Có sẵn các loại vector và biến đổi (transform), GDScript giúp tăng hiệu suất khi sử dụng đại số tuyến tính.
- Hỗ trợ đa luồng một cách hiệu quả như các ngôn ngữ kiểu tĩnh (static).

GDScript có các khối mã sắp xếp bằng cách thuật lè tương tự như Squirrel, Lua và Python.

```

# Function

func some_function(param1, param2):
    var local_var = 5

    if param1 < local_var:
        print(param1)
    elif param2 > 5:
        print(param2)
    else:
        print("Fail!")

    for i in range(20):
        print(i)

    while param2 != 0:
        param2 -= 1

    var local_var2 = param1 + 3
    return local_var2

```

Hình 2.8. Ví dụ một số cú pháp GDScript trong Godot Engine

2.5. BLENDER (PHIÊN BẢN 3.4.1)

Blender là một phần mềm mã nguồn mở, miễn phí dùng để tạo và chỉnh sửa đồ họa 3D và hoạt ảnh. Blender cung cấp nhiều công cụ và chức năng mạnh mẽ giúp người dùng tạo ra các mô hình 3D, hiệu ứng đặc biệt và nhiều hơn nữa.

Blender cung cấp chế độ mô hình hóa (modeling) cho phép người dùng tạo ra các hình dạng và bề mặt 3D từ đơn giản đến phức tạp. Người dùng có thể sử dụng các công cụ như Extrude, Bevel và Boolean để tạo ra các khối hình và chi tiết.

Ngoài ra, Blender còn cung cấp chế độ chạm trổ (sculpting) cho phép người dùng tạo ra các chi tiết tự nhiên trên mô hình 3D. Công cụ này giúp tạo ra các hiệu ứng như chạm trổ đá, da và nhiều hơn nữa.

Blender cũng hỗ trợ tạo vật liệu (texturing), cho phép người dùng áp dụng các bản vẽ, ảnh và hiệu ứng shader lên các đối tượng 3D. Điều này giúp tạo ra các vật liệu như gỗ, kim loại và thủy tinh.

Công cụ hoạt ảnh (animation) của Blender cho phép người dùng tạo ra các hoạt ảnh chuyển động. Người dùng có thể tạo ra các khung hình chuyển động, điều khiển sự di chuyển và biểu cảm của các đối tượng.

Blender cung cấp tính năng kết xuất (rendering) để tạo ra hình ảnh và video chất lượng cao từ các mô hình 3D và hoạt ảnh. Người dùng có thể tùy chỉnh ánh sáng, vật liệu và các cài đặt khác để tạo ra hình ảnh sống động và chân thực.

Ngoài ra, Blender còn có các công cụ và tính năng để tạo ra các hiệu ứng đặc biệt (Visual Effects - VFX) như hỏa tiễn, nước, khói và nhiều hiệu ứng động khác. Công cụ này sử dụng hệ thống phân tử, mô phỏng vật lý và xử lý hình ảnh để tạo ra các hiệu ứng đa dạng.

Blender là một công cụ đồ họa 3D đa năng và mạnh mẽ, được sử dụng rộng rãi trong ngành công nghiệp đồ họa 3D và hoạt ảnh. Với sự hỗ trợ mạnh mẽ từ đông đảo người dùng, Blender đã trở thành một lựa chọn hàng đầu cho những người yêu thích sáng tạo và làm việc trong lĩnh vực thiết kế đồ họa 3D.

2.6. ADOBE PHOTOSHOP (PHIÊN BẢN 19.1.3 2018)

Adobe Photoshop là một phần mềm biên tập ảnh và thiết kế đồ họa hàng đầu trong ngành công nghiệp. Photoshop cung cấp một loạt các chức năng và công cụ mạnh mẽ để tạo ra và chỉnh sửa các hình ảnh chất lượng cao và thiết kế đồ họa sáng tạo.

Một trong những chức năng chính của Photoshop là chỉnh sửa ảnh. Người dùng có thể điều chỉnh màu sắc, độ sáng, độ tương phản và sắc nét của ảnh để tạo ra hiệu ứng tùy chỉnh. Ngoài ra, Photoshop cung cấp công cụ khử nhiễu, làm mờ, làm rõ và sửa chữa các khuyết điểm của ảnh.

Photoshop cũng cho phép người dùng thực hiện các chỉnh sửa nâng cao như cắt, xoay, thay đổi kích thước và biến đổi hình dạng của các đối tượng trong ảnh. Photoshop cung cấp công cụ lựa chọn chính xác để xử lý các phần tử cụ thể trong ảnh.

Ngoài ra, Photoshop cung cấp các công cụ để thêm và chỉnh sửa văn bản trong hình ảnh. Người dùng có thể tạo ra các hiệu ứng chữ viết tay, đặt văn bản theo đường cong hoặc áp dụng các hiệu ứng đặc biệt như nổi bật, đổ bóng và đồ họa.

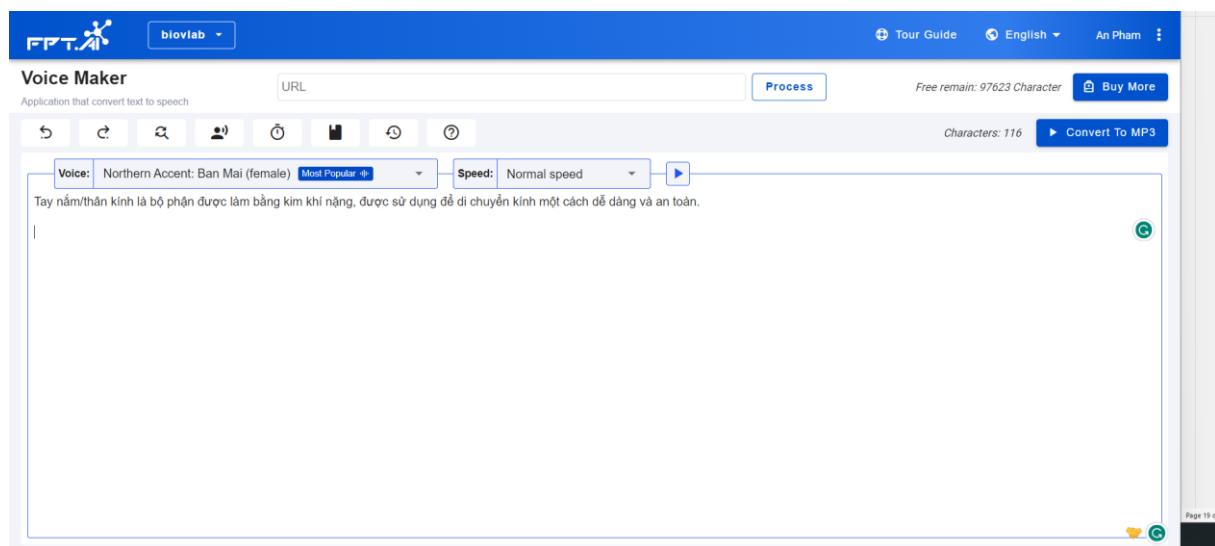
Photoshop là một công cụ mạnh mẽ cho thiết kế đồ họa. Người dùng có thể tạo ra các poster, banner, hình nền và biểu đồ sáng tạo bằng cách sử dụng các công cụ vẽ, kẻ, gradient và các hiệu ứng đặc biệt. Photoshop cũng hỗ trợ việc tạo và chỉnh sửa các lớp và mặt nạ, cho phép người dùng tạo ra các hiệu ứng đa chiều và sáng tạo.

Bên cạnh đó, Photoshop cung cấp tính năng xử lý ảnh RAW để chỉnh sửa và tái tạo màu sắc chính xác cho ảnh chụp từ máy ảnh chuyên nghiệp. Hỗ trợ công cụ đo màu, viền và công cụ pha trộn màu sắc để tạo ra các hiệu ứng độc đáo.

2.7. CHUYỂN VĂN BẢN THÀNH GIỌNG NÓI VOICE MAKER CỦA FPT.AI

Voice Maker cung cấp một giao diện thân thiện với người dùng cho phép dễ dàng thao tác và phù hợp cho cả cá nhân và tổ chức với mọi quy mô. Với ứng dụng này, người dùng có thể dễ dàng thực hiện công việc chuyển văn bản thành giọng nói và tải kết quả xuống dưới định dạng MP3.

Ứng dụng cung cấp tổng cộng 9 giọng từ các khu vực khác nhau bao gồm: giọng Bắc (Ban Mai, Thu Minh, Lê Minh), giọng Trung (Gia Huy, Mỹ An, Ngọc Lam), giọng Nam (Lan Linh, Linh San, Minh Quang). Với 13 cấp độ tốc độ khác nhau (-3, -2.5, -2, -1.5, -1, -0.5, bình thường, 0.5, 1, 1.5, 2, 2.5, 3).



Hình 2.9. Giao diện chuyển đổi văn bản thành giọng nói Voice Maker của FPT.AI

Người dùng Voice Maker có thể tạo ra những bài phát biểu tự nhiên phục vụ cho các ứng dụng trong đời sống thực như lòng tiếng cho video, sách nói, và nhiều ứng dụng khác. Voice Maker cho phép người dùng sử dụng miễn phí với tổng số lượng ký tự có thể chuyển đổi thành văn bản là 100000 ký tự. Ngoài ra, Voice Maker cũng cung cấp các gói nâng cấp giọng nói và tăng số lượng ký tự có thể chuyển đổi hành giọng nói. Các văn bản được chuyển đổi thành giọng nói được lưu lại trong lịch sử chuyển đổi và có thể tải các file âm thanh thành các định dạng khác nhau.

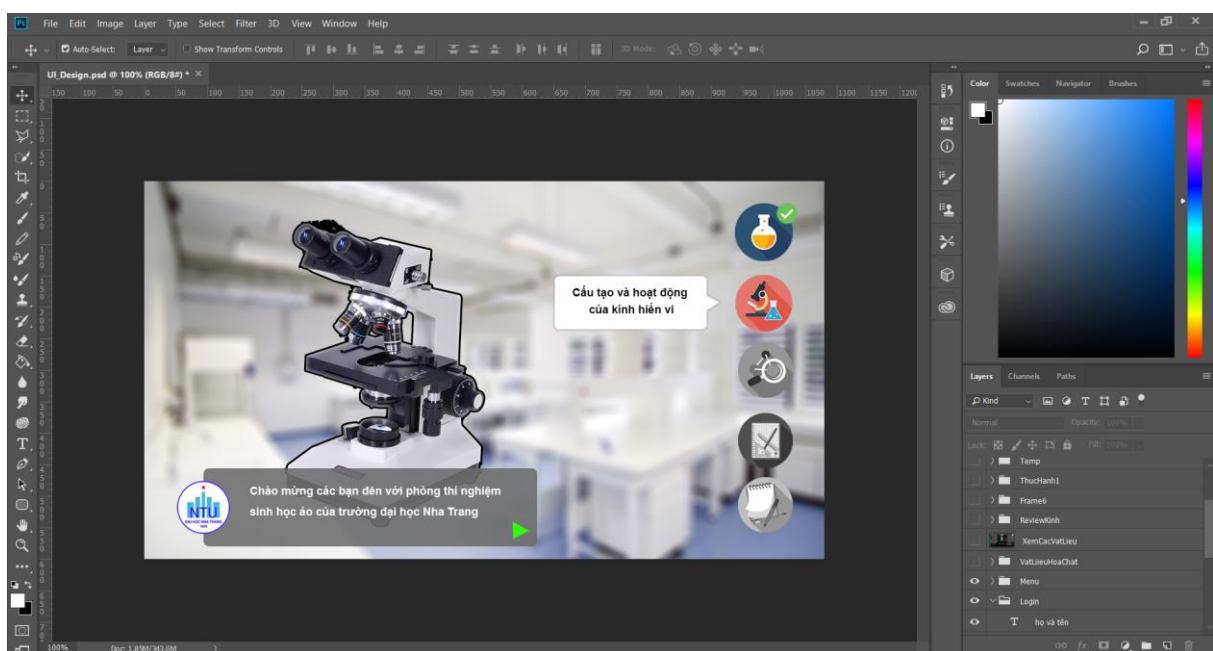
Giọng nói được tạo ra bởi Voice Maker được sử dụng trong bài thực hành ảo “Kính hiển vi” để chuyển căn bản thành giọng nói được sử dụng trong các hộp thoại (dialog) hay các hướng dẫn trong chương trình.

CHƯƠNG 3. XÂY DỰNG CHƯƠNG TRÌNH MÔ PHỎNG

3.1. THIẾT KẾ GIAO DIỆN CÁC BÀI THỰC HÀNH

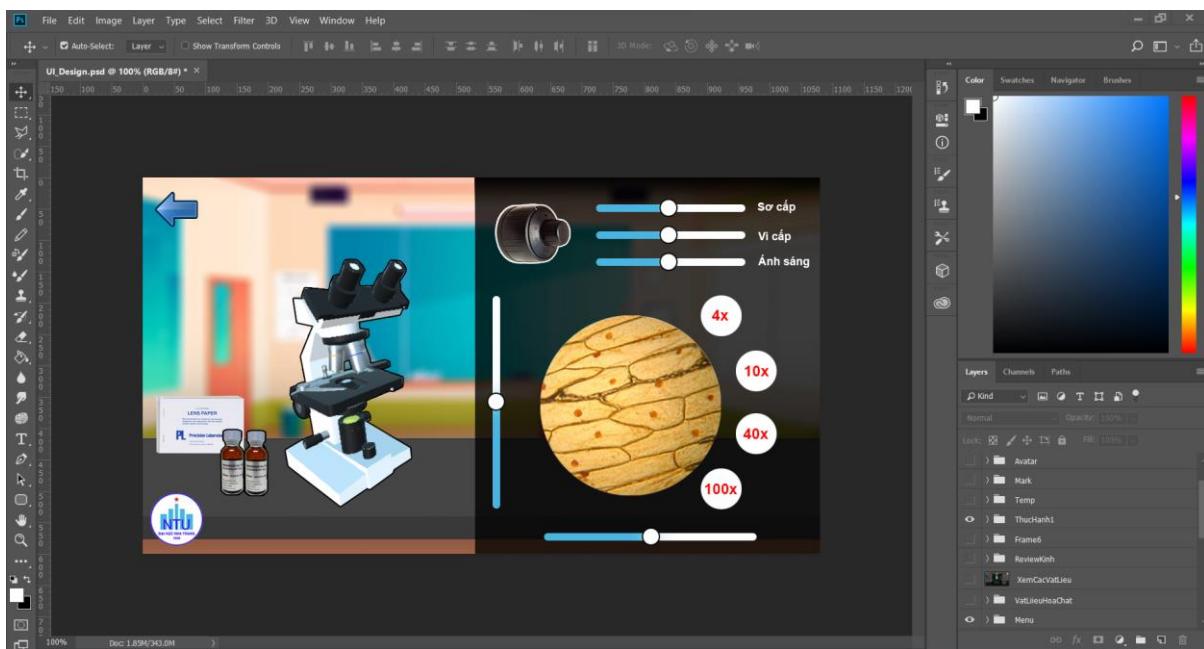
Giao diện sử dụng trong bài thực hành mô phỏng kính hiển vi đóng vai trò quan trọng trong việc đảm bảo người dùng có thể tương tác và sử dụng kính hiển vi một cách hiệu quả. Giao diện trong bài thực hành kính hiển vi ảo được thiết kế và xây dựng bằng phần mềm photoshop trước để đảm bảo các khung hình (frame), bố cục, màu sắc, kích thước các nút (button), thông báo (dialog) được đồng bộ và thống nhất với nhau. Có được cái nhìn tổng quan trước khi ứng dụng được đưa vào lập trình đồng thời tính toán được các đối tượng cần xây dựng, số màn hình (scene) sẽ được tạo ra.

Phong cách xây dựng trong bài thực hành kính hiển vi được thiết kế đơn giản và dễ sử dụng để người dùng có thể hiểu và tương tác với bài thực hành một cách nhanh chóng. Điều này đặc biệt quan trọng trong trường hợp của sinh viên hoặc người mới bắt đầu sử dụng và thao tác trên bài thực hành ảo, vì họ cần phải nắm bắt các chức năng và điều chỉnh một cách dễ dàng. Các nút (button) được thiết kế có hình ảnh minh họa để người dùng có thể dùng và hiểu ngay ý nghĩa chức năng của các nút.



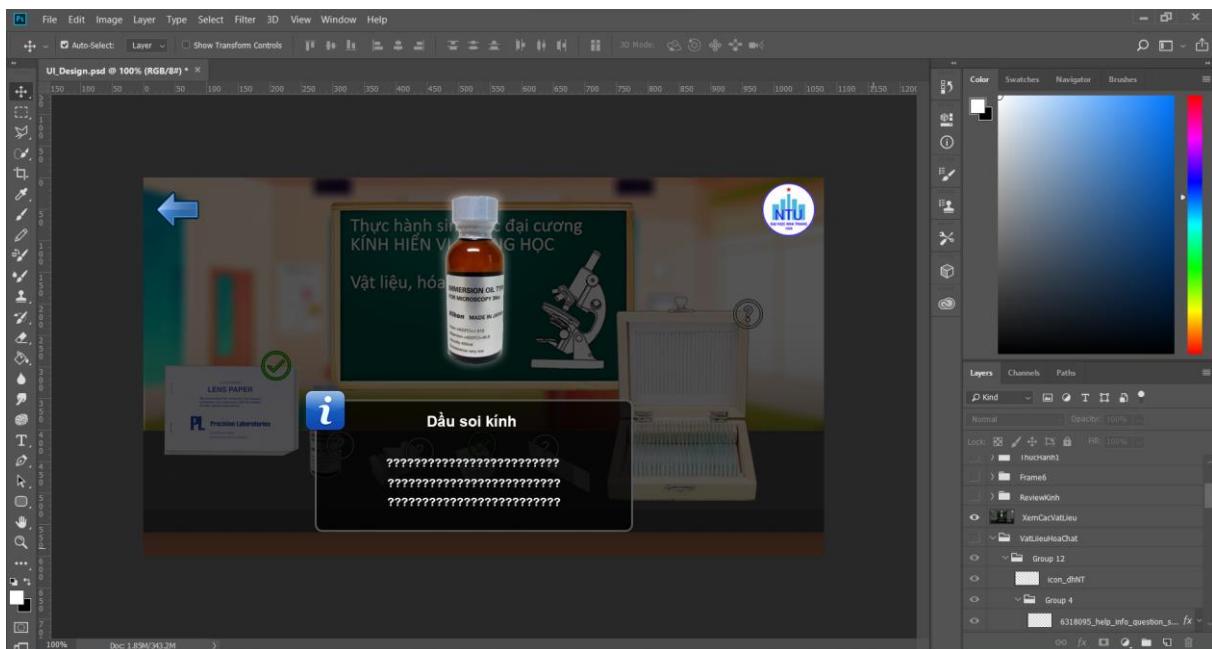
Hình 3.1. Giao diện thiết kế các nút có hình ảnh minh họa trong Photoshop

Giao diện cung cấp đầy đủ các chức năng cần thiết để người dùng có thể thực hiện các thao tác quan trọng. Ví dụ, người dùng cần có khả năng điều chỉnh độ phóng đại, lấy nét, thay đổi màu sắc và điều chỉnh ánh sáng trong mô phỏng kính hiển vi.



Hình 3.2. Giao diện thiết kế các thanh trượt trong Photoshop

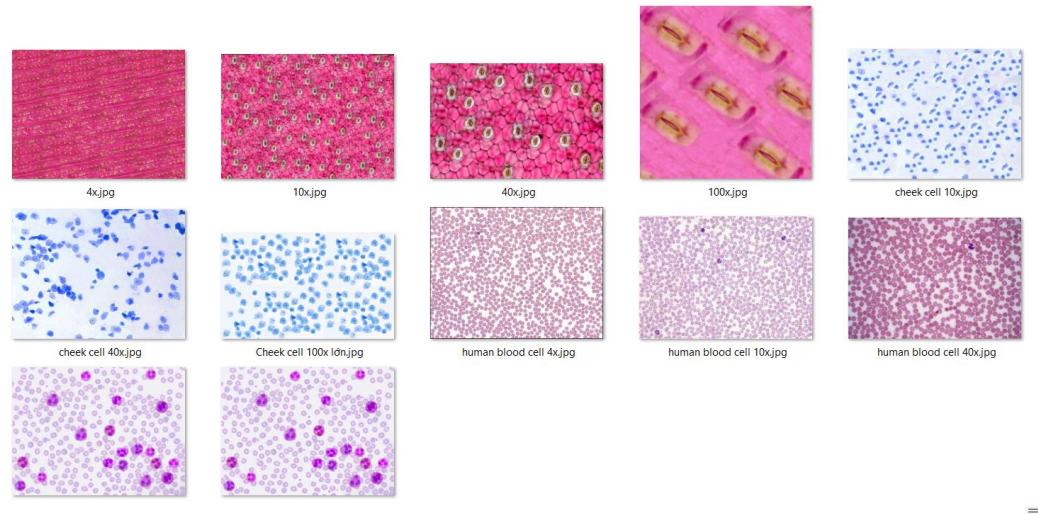
Giao diện cung cấp hướng dẫn rõ ràng và thông tin hữu ích để giúp người dùng hiểu và sử dụng các tính năng của kính hiển vi một cách hiệu quả. Hướng dẫn này có thể bao gồm các gợi ý, lời nhắc, thông báo và mô tả cho từng chức năng hoặc thao tác.



Hình 3.3. Giao diện thiết kế hộp thoại trong Photoshop

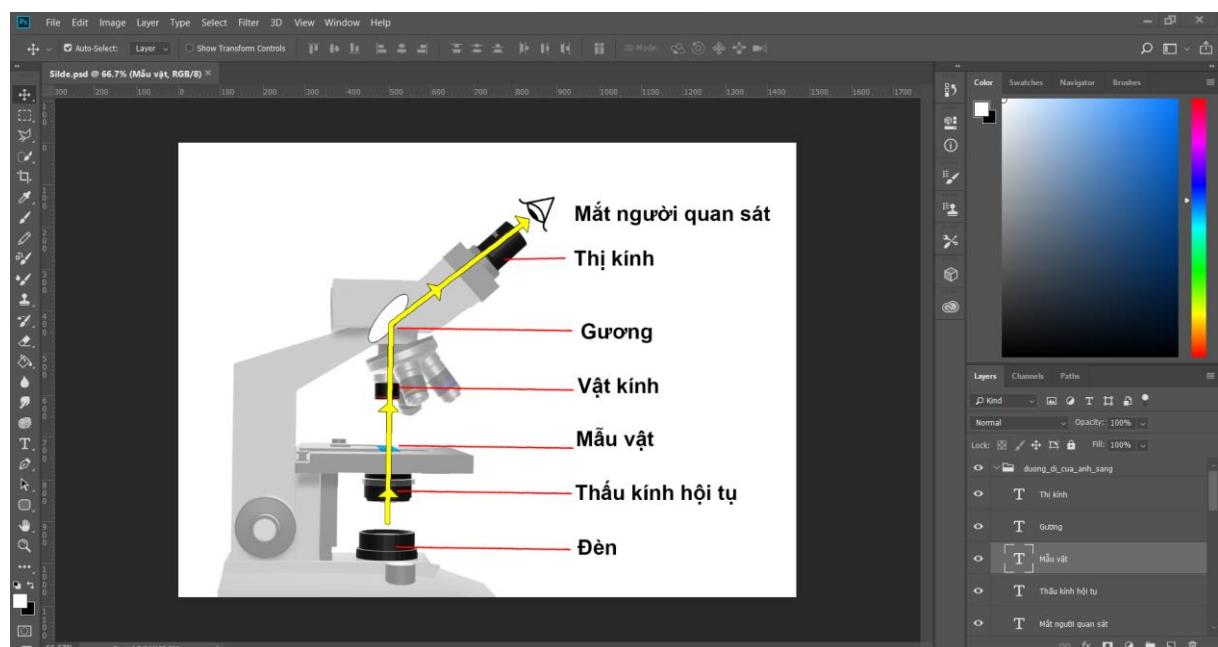
Để tối ưu hóa thời gian, công sức, hầu hết các biểu tượng, hình vẽ trong bài thực hành ảo đều sử dụng các tài nguyên có sẵn được cung cấp miễn phí trên mạng (<https://www.freeimages.com/icon>). Sau khi chọn được các biểu tượng phù hợp, các hình ảnh đó được chỉnh sửa, thay đổi màu sắc, bố cục hay cắt ghép từ nhiều hình ảnh khác để phù hợp hơn với các tiêu chí được đặt ra trong khâu thiết kế.

Ngoài các hình ảnh được cung cấp trên mạng, bài thực hành ảo cần hình ảnh của các vật liệu, hóa chất, dụng cụ trong phòng thí nghiệm và các vật mẫu đã được quan sát và chụp lại trên kính hiển vi điện tử đã được cô Văn Hồng Cầm (Viện CNSH&MT, Trường Đại học Nha Trang) cung cấp.



Hình 3.4. Tài nguyên hình ảnh các vật mẫu được soi dưới kính hiển vi

Phần cuối cùng không thể thiếu là tài liệu hướng dẫn thực hành của mỗi bài học trong bài thực hành ảo. Các thông tin dạng văn bản sẽ được thêm vào bài học. Để tránh nhảm chán, các phần hướng dẫn sẽ có các hình ảnh để minh họa cho người học dễ hình dung quy trình cần thực hiện.



Hình 3.5. Giao diện thiết kế tài liệu hướng dẫn trong Photoshop

Tất cả các hình ảnh được xuất bản với định dạng JPEG và PNG.

3.2. THIẾT KẾ GIAO DIỆN KÍNH HIỂN VI 3D

Áp dụng mô hình 3D vào các dự án mô phỏng mang lại nhiều lợi ích và cải tiến so với mô hình 2D truyền thống. Mô hình 2D truyền thống trong các dự án mô phỏng giới hạn khả năng tái tạo và truyền tải thông tin. Hạn chế chính của mô hình 2D là không thể đồng thời tái tạo đầy đủ các chiều không gian và các góc nhìn khác nhau. Điều này dẫn đến sự thiếu thực tế và hạn chế trong việc hiểu và tương tác với mô hình. Mô hình 3D cho phép quan sát và tương tác với các đối tượng từ nhiều góc nhìn khác nhau. Người dùng có thể quay, xoay và phóng to/thu nhỏ mô hình, giúp họ khám phá chi tiết các khía cạnh khác nhau của đối tượng.

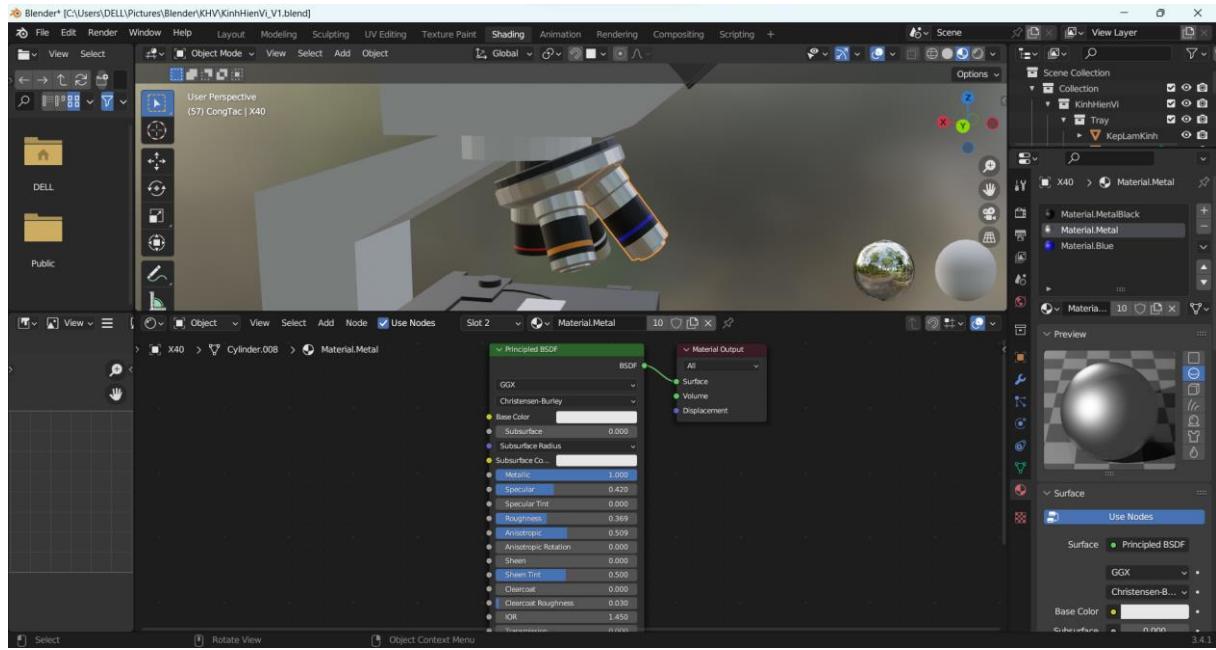
Nhận thấy nhưng ưu điểm và lợi ích của việc mô hình hóa 3D các vật thể. Trong bài thực hành ảo này đã sử dụng công cụ Blender để tạo mô hình 3D của kính hiển vi. Qua đặc điểm và cấu tạo của kính hiển vi, các hình ảnh tham khảo từ kính hiển vi “Microscope Omax Om88 40x-1600x Clinical Compound Microscope”, các thông tin về thân kính, thị kính, vật kính, nút vi cấp và sơ cấp được sử dụng để mô phỏng lại từng phần nhỏ của kính hiển vi hoàn chỉnh.



Hình 3.6. Kết quả sau khi xây dựng mô hình 3D trong Blender

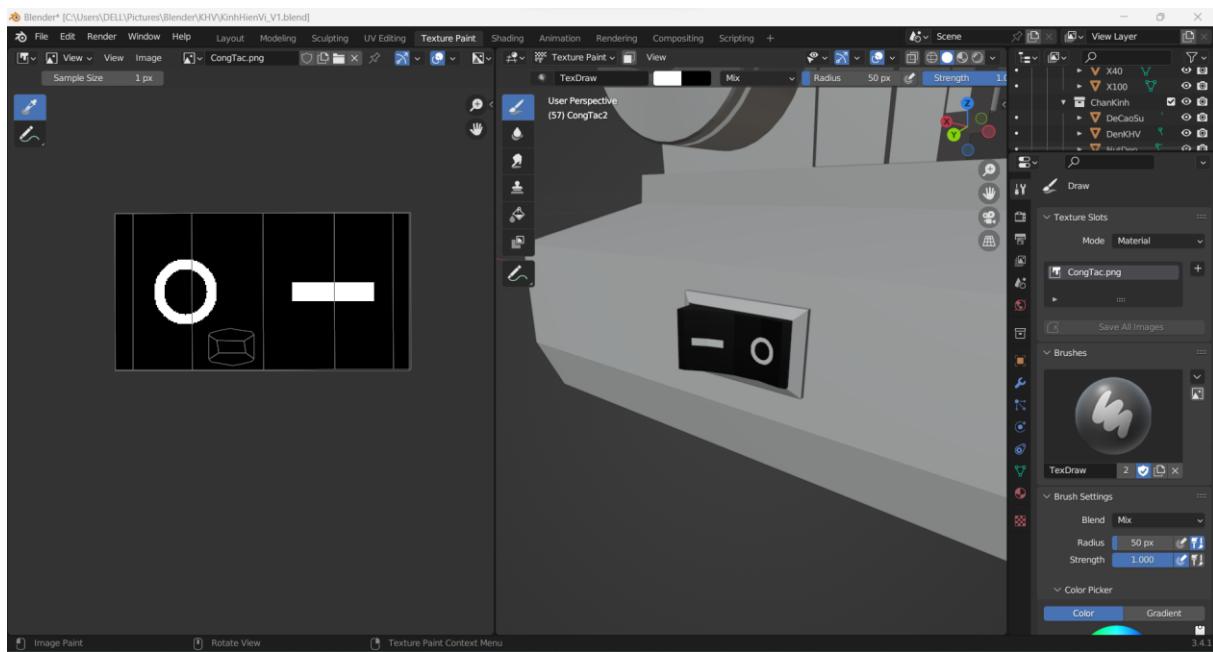
Tuy nhiên việc chỉ xây dựng mô hình 3D thôi thì chưa đủ. Để kính hiển vi 3D đạt được độ chân thực và gần giống với kính hiển vi thật. Đầu tiên cần thiết kế tạo ra các vật liệu phù hợp với từng bộ phận của kính hiển vi, sau đó phủ lớp vật liệu đó vào các bề mặt của các bộ phận kính hiển vi. Ví dụ: vật kính làm bằng thép không gỉ và có

bề mặt được đánh bóng, sử dụng công cụ đổ bóng (Shading) để mô phỏng lại vật liệu của vật kính, sau đó phủ lớp vật liệu đó vào các bề mặt của bộ phận vật kính.

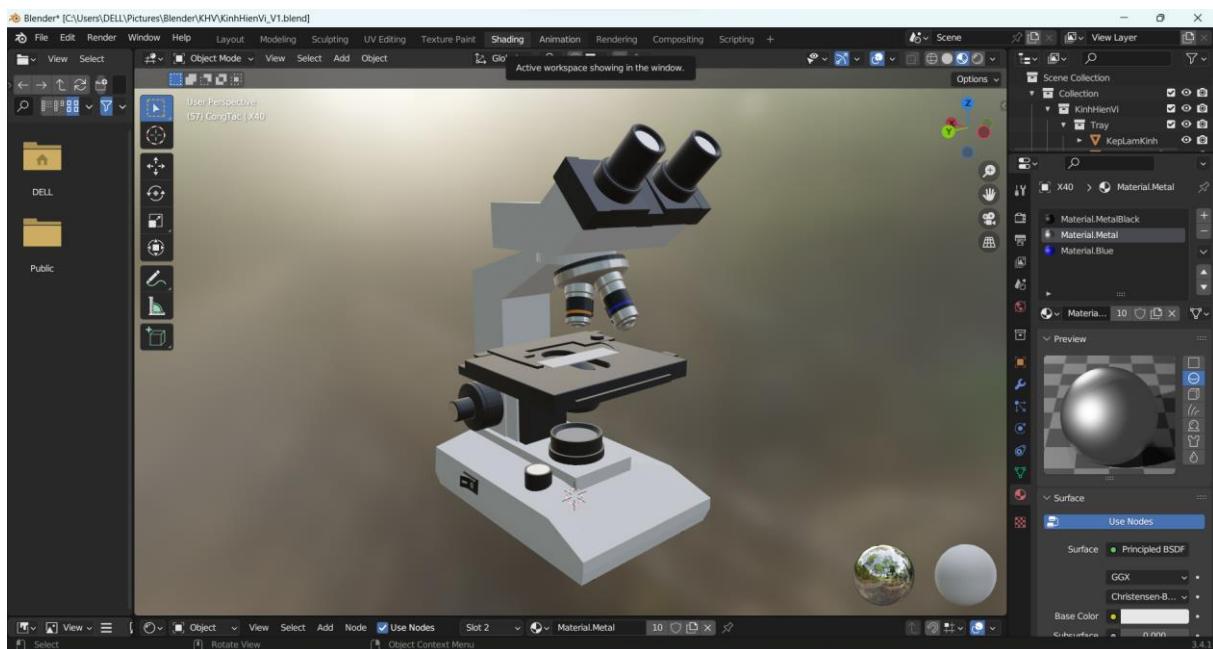


Hình 3.7. Giao diện Shading khi thiết kế vật liệu cho vật kính trong Blender

Bước cuối cùng là trang trí kính hiển vi bằng cách thêm họa tiết (texture) để hiển thị thông tin như độ phóng đại 4x, 10x, 40x, 100x hoặc các ký hiệu tắt/bật cho công tắc. Trong Blender, chúng ta có thể sử dụng chức năng UV Editing (chức năng UV-mapping) cho phép người dùng chỉnh sửa và ánh xạ mô hình 3D vào một mặt phẳng 2D để ánh xạ mô hình 3D thành một mặt phẳng, sau đó vẽ các hình vẽ hoặc chữ lên mặt phẳng tương ứng ở màn hình Texture Paint (Vẽ kết cấu).

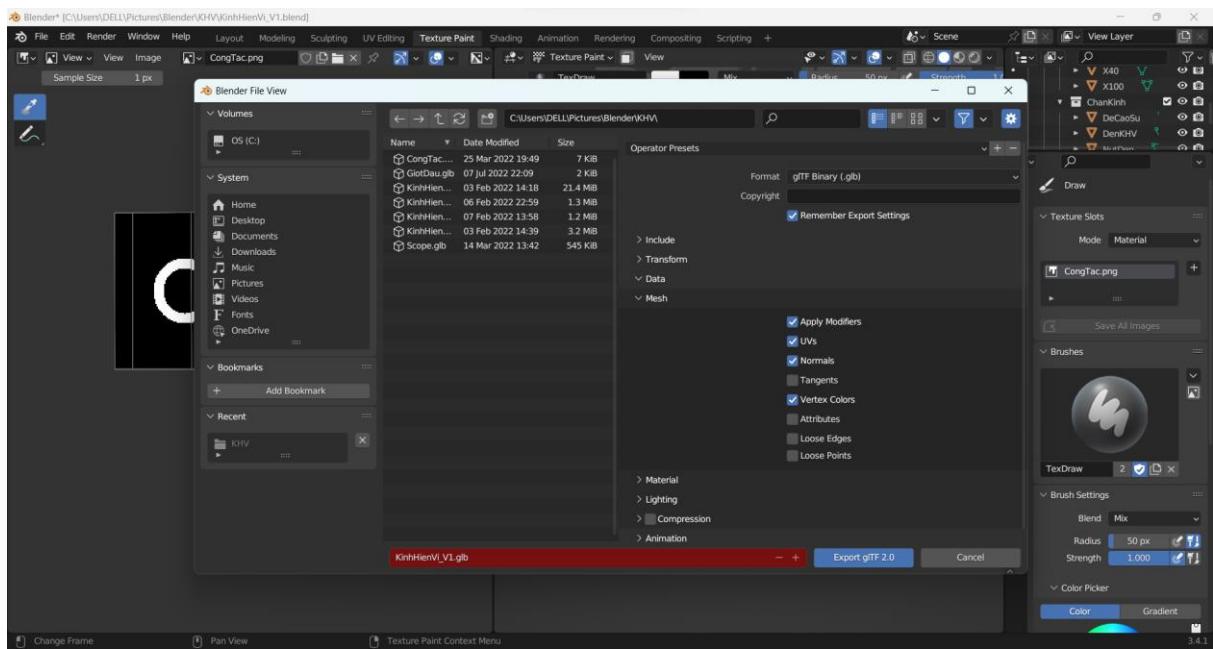


Hình 3.8. Giao diện Texture Paint cho công tắc bật đèn cả kính hiển vi



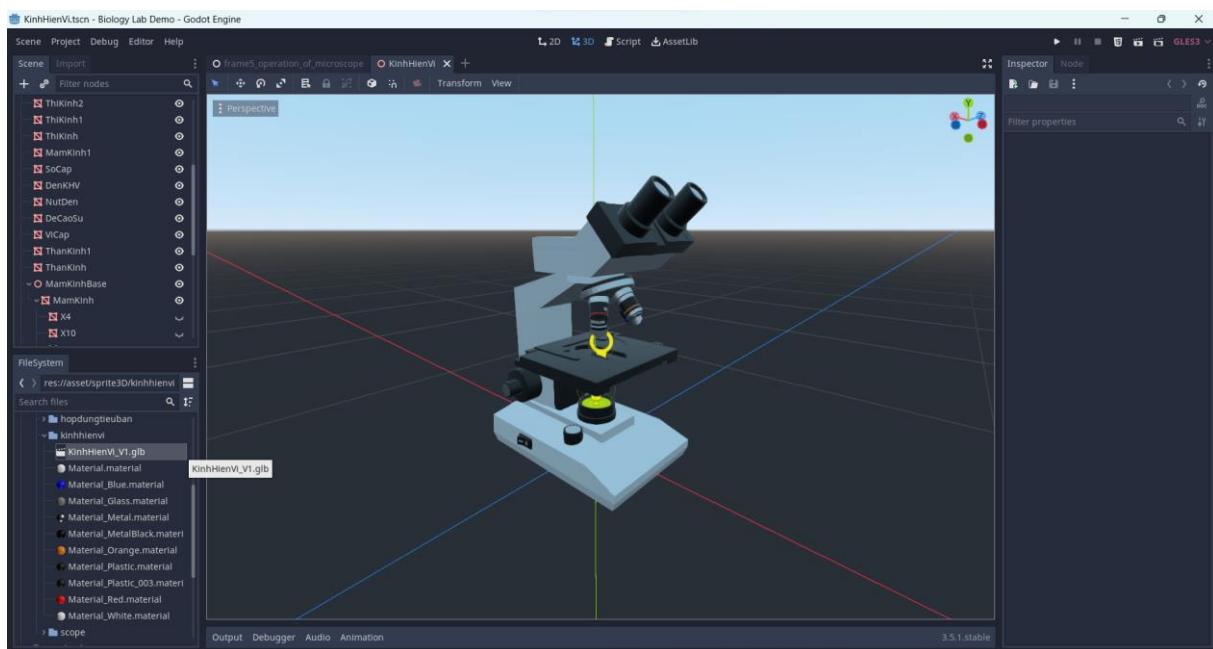
Hình 3.9. Mô hình kính hiển vi hoàn chỉnh

Sau khi hoàn thiện kính hiển vi 3D, mô hình vẫn chưa thể sử dụng trực tiếp để thêm (Import) vào dự án Godot mà phải được xuất sang định dạng glTF. glTF™ (GL Transmission Format) được sử dụng để truyền và tải các mô hình 3D trong ứng dụng web và ứng dụng native. glTF giảm kích thước của các mô hình 3D và thời gian xử lý cần thiết để giải nén và hiển thị các mô hình đó. Định dạng này thường được sử dụng trên web đồ họa (Web GL) và được hỗ trợ trong các engine 3D khác nhau như Unity3D, Unreal Engine 4 và Godot.



Hình 3.10. Các thông số khi xuất (export) mô hình kính hiển vi

Sau khi xuất (export) mô hình kính hiển vi ra thành tập tin có định dạng .glb (VD: KinhHienVi_V1.glb) để có thể Import trực tiếp vào Godot và sử dụng.



Hình 3.11. Kết quả sau khi import mô hình kính hiển vi vào Godot Engine

3.3. XÂY DỰNG CÁC MÀN HÌNH TRONG BÀI THỰC HÀNH ẢO

3.3.1. Frame 1 – Màn hình đăng nhập

Chức năng

Màn hình đăng nhập được sử dụng để nhận thông tin cung cấp từ sinh viên. Màn hình yêu cầu nhập các thông tin bao gồm: MSSV (mã số sinh viên) và tên sinh viên.

Các thông tin đó được lưu trữ để sau khi sinh viên hoàn thành xong khóa học thì sẽ cung cấp chứng chỉ gồm các thông tin trên để làm bằng chứng đã hoàn thành khóa học.

Yêu cầu

❖ Giao diện

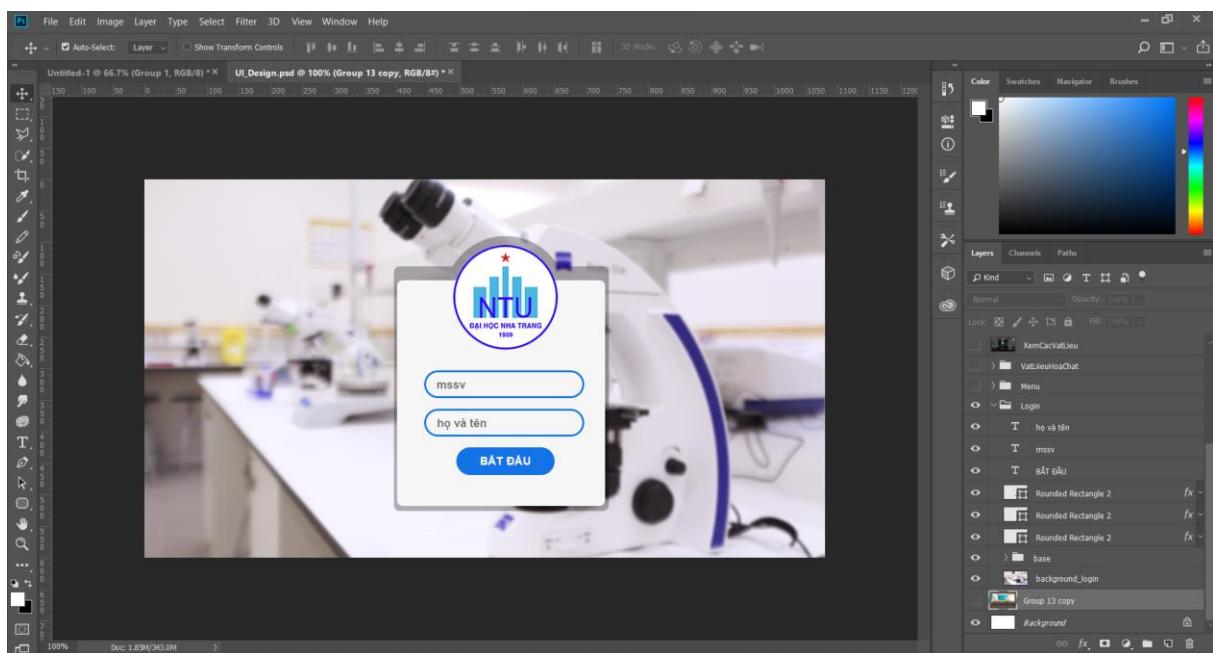
- Có đầy đủ các trường thông tin cần sinh viên nhập vào.
- Có nút bắt đầu để bắt đầu bài học.
- Có nút bật tắt âm lượng trước khi vào bài học.
- Có biểu tượng trường Đại học Nha Trang.
- Hiển thị thông báo sau nếu sinh viên nhập sai các trường thông tin.

❖ Đặc tả hệ thống

- MSSV phải đủ 8 chữ số, không chứa các kí tự đặc biệt hoặc khoảng trắng.
- Tên sinh viên không được bỏ trống.

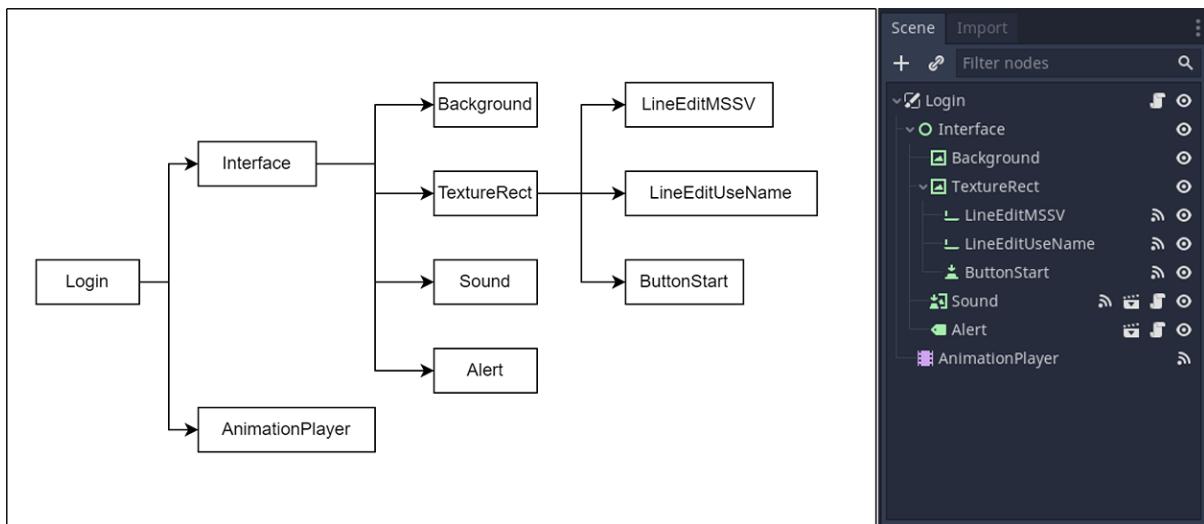
Triển khai

Giao diện được thiết kế với các tông màu sáng, màu xanh làm chủ đạo và là điểm nhấn. Các ô nhập liệu và nút bấm được làm màu xanh trên nền màu trắng được thể hiện nổi bật dễ nhìn. Biểu tượng của trường Đại học Nha Trang nằm ở giữa. Ánh nền là kính hiển vi được tạo hiệu ứng làm mờ để tạo chiều sâu.



Hình 3.12. Giao diện scene login thiết kế ở Photoshop

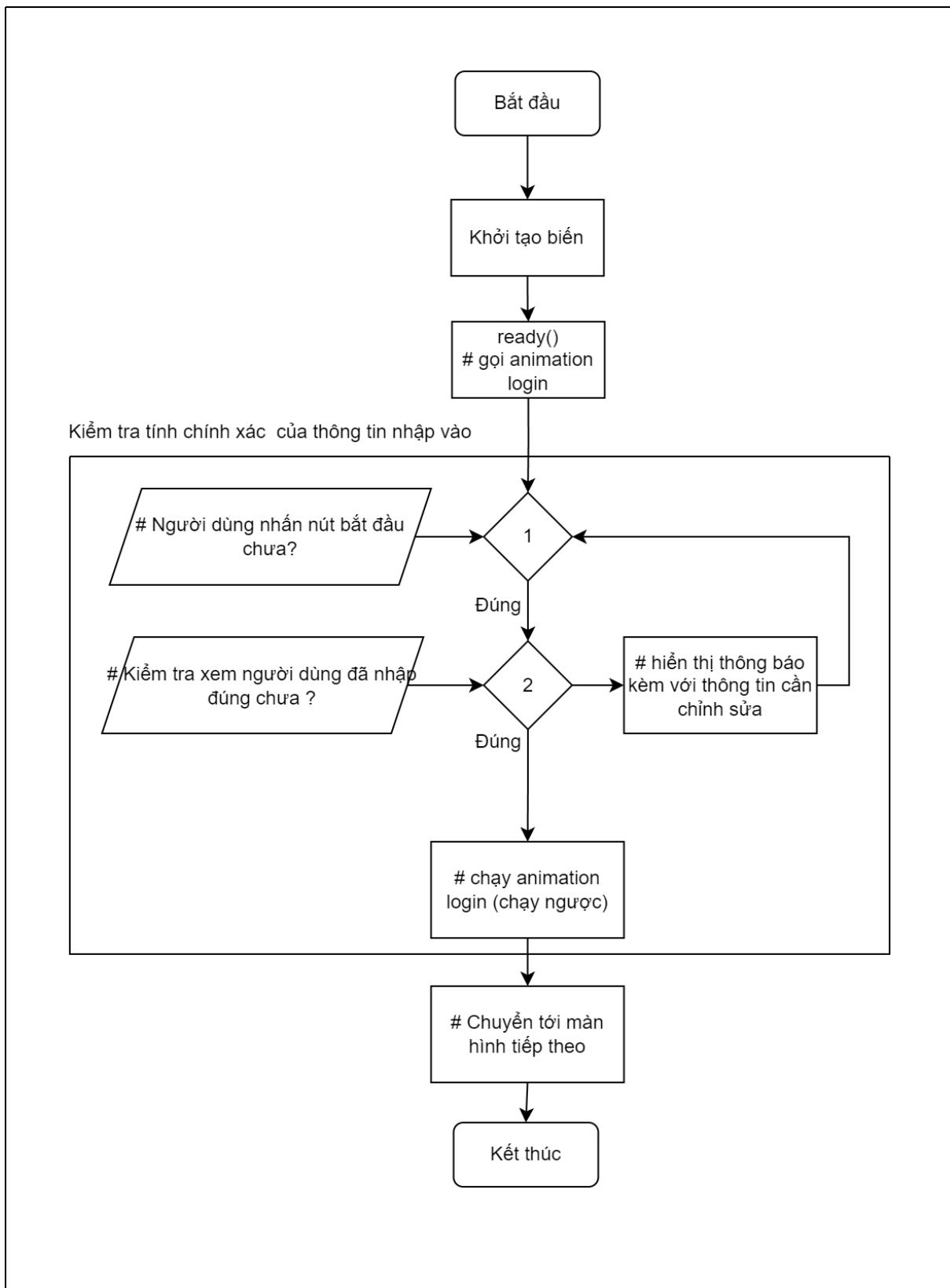
Khi thiết kế giao diện được hoàn thành, ta cần xuất các đối tượng ra các file hình ảnh để nhập khẩu vào thư mục tài nguyên (asset) của dự án. Qua giao diện ta sẽ phân tích ra các đối tượng cần tạo trong dự án. Sau đây là sơ đồ cây của các đối tượng trong màn hình đăng nhập.



Hình 3.13. Sơ đồ cấu trúc cây của màn hình đăng nhập

Trong sơ đồ cấu trúc cây này cần lưu ý các đối tượng:

- LineEditMSSV: đối tượng nhận MSSV từ sinh viên nhập vào.
- LineEditUsername: đối tượng nhận tên của sinh viên.
- ButtonStart: nút bắt đầu.
- Sound: nút bật tắt âm thanh trong bài thực hành.
- Alert: thẻ thông báo, hiển thị ở dưới màn hình khi SV nhập sai thông tin.
- AnimationPlayer: chạy hoạt hình đăng nhập.



Hình 3.14. Sơ đồ luồng xử lý của Scene Login

Giải thích sơ đồ:

- ❖ Khởi tạo biến: Khi bắt đầu vào màn hình, hệ thống sẽ khởi tạo các biến được khai báo bên trong kịch bản. Từ khóa onready var được sử dụng để gán giá trị cho biến khi các node được thêm vào scene tree, theo sau là tên biến (tên biến: _animationLogin,

_alert...). Trong khi từ khóa var được hiểu là khởi tạo biến, biến sử dụng từ khóa var được tải trước khi các node chưa được tải vào scene tree.

```
# _animationLogin ánh xạ tới AnimationPlayer đảm nhiệm animation
onready var _animationLogin = $AnimationPlayer
# _alert ánh xạ tới Alert (hiển thị thông báo khi nhập sai)
onready var _alert = $Interface/Alert
# isLoadMenu: true -> nhập thông tin chính xác | false -> nhập sai
var isLoadMenu: bool = false
```

❖ Hàm ready: Đây là một hàm được gọi tự động khi đối tượng được tạo. Trong trường hợp này, hàm này chạy khi đối tượng được tạo sẵn và sẽ chơi hoạt cảnh “Login” bằng cách gọi phương thức play trên đối tượng AnimationPlayer.

```
func _ready():
    _animationLogin.play("Login")
```

❖ Kiểm tra tính chính xác của thông tin nhập vào: đây là một khối mã kiểm tra bao gồm 3 tín hiệu (signals). Trong đó, Global là biến được khởi tạo từ cửa sổ autoload và biến Global ảnh hưởng đến tất cả các scene, các kịch bản được tải từ màn hình autoload được khởi tạo theo mẫu thiết kế singleton.

```
# Phương thức kiểm tra hợp lệ của MSSV
func checkMSSV():
    if Global.mssv.length() != 8:
        return false
    if not Global.mssv.is_valid_integer():
        return false
    if Global.mssv.find("+", 0) != -1:
        return false
    if Global.mssv.find("-", 0) != -1:
        return false
    return true

# Khi sinh viên nhấp nút bắt đầu sẽ thực hiện kiểm tra các giá trị
func _on_ButtonStart_button_up():
    if not Global.userName.empty() and checkMSSV():
        isLoadMenu = true
        # Chạy animation login (chạy ngược)
        _animationLogin.play_backwards("Login")
    else:
        if Global.mssv.length() != 8:
            _alert.setText("Mã số sinh viên cần 8 ký tự")
        elif not Global.mssv.is_valid_integer():
            _alert.setText("Mã số sinh viên chỉ được chứa chữ số")
        elif Global.userName.empty():
```

```

        _alert.setText("Họ và tên không được bỏ trống")
        _alert.showAlert()

# Thực hiện hành động khi sinh viên nhập giá trị vào ô input
func _onLineEditMSSV_text_changed(new_text:String):
    Global.mssv = new_text # Lưu mssv vào biến toàn cục mssv

# Thực hiện hành động khi sinh viên nhập giá trị vào ô input
func _onLineEditUserName_text_changed(new_text:String):
    # Lưu tên sinh viên vào biến toàn cục userName
    Global.userName = new_text

```

- ❖ Chuyển đến màn hình tiếp theo: sau khi animation login (chạy ngược) được hoàn thành thì sẽ chuyển sang màn hình tiếp theo ở đây là FRAME_2 (Màn hình chính).

```

# Chạy hoạt hình hiển thị giao diện của bảng thông tin đăng nhập
func _onAnimationPlayer_animation_finished(anim_name):
    if anim_name == "Login" and isLoadMenu:
        Load.load_scene(self, Global.FRAME_2)

```

3.3.2. Frame 2 – Màn hình chính

Chức năng

Màn hình chính (scene main), giới thiệu bài học, hiển thị các nút chuyển cảnh giúp người dùng có thể di chuyển sang các màn hình khác. Các nút chuyển cảnh cần kích hoạt thì mới tới được các màn hình tiếp theo.

Yêu cầu

❖ Giao diện:

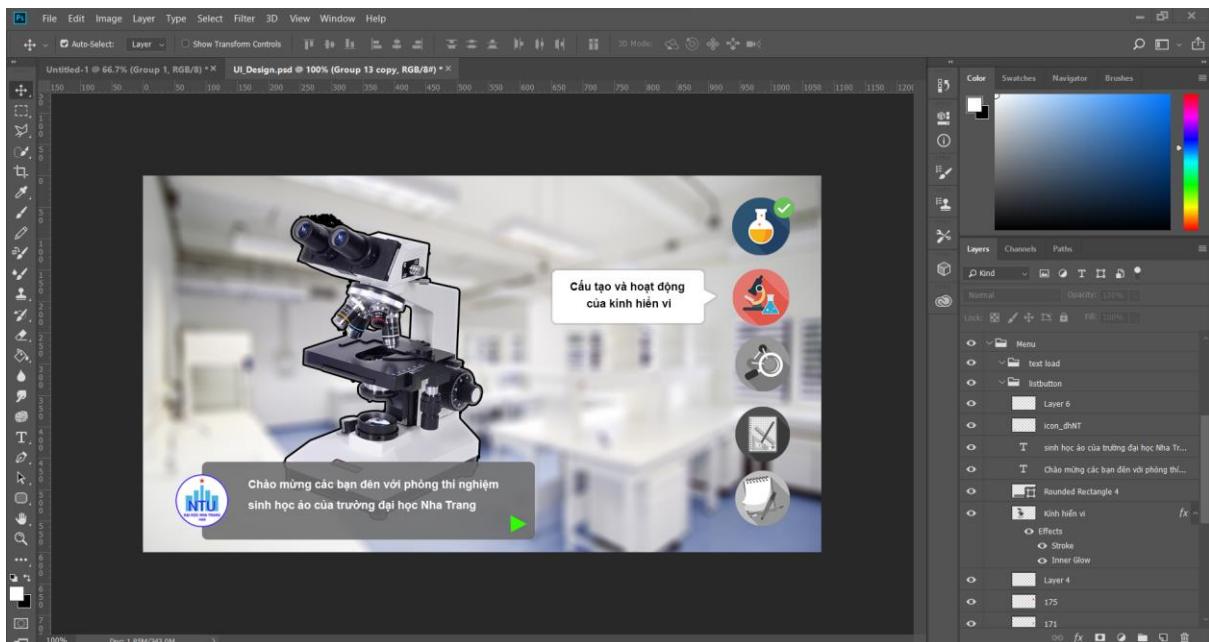
- Bao gồm 5 nút chuyển cảnh: Tìm hiểu vật liệu và hóa chất, tìm hiểu cấu tạo kính hiển vi, thực hành mẫu quan sát hiển vi, thực hành và cuối cùng là kiểm tra. Tương ứng với 4 bài học và 1 bài kiểm tra trắc nghiệm để tổng kết kiến thức. Các nút chuyển cảnh cần thiết kế có hình ảnh minh họa cụ thể với từng nội dung bài học. Có thêm các hộp thoại (dialog) để hiển thị tên của các nút.
- Các nút được mở sẽ có màu, các nút bị tắt sẽ có màu đen trắng. Các bài học hoàn thành tương ứng sẽ có dấu tích xanh để hiểu rằng bài đó đã hoàn thành.
- Nút bật tắt âm lượng.

❖ Đặc tả hệ thống:

- Khi sinh viên bắt đầu bài học đầu tiên, một hộp thoại hiện ra để chào mừng sinh viên đến với khóa học. Hộp thoại này chỉ xuất hiện một lần duy nhất.
- Nút chuyển cảnh đầu tiên luôn bật (Tìm hiểu vật liệu và hóa chất) để người dùng vào bài học đầu tiên. Chỉ sau khi hoàn thành bài học ở nút đó sẽ hiện lên dấu tích xanh (đã hoàn thành bài học). Sau đó mở bài học tiếp theo.

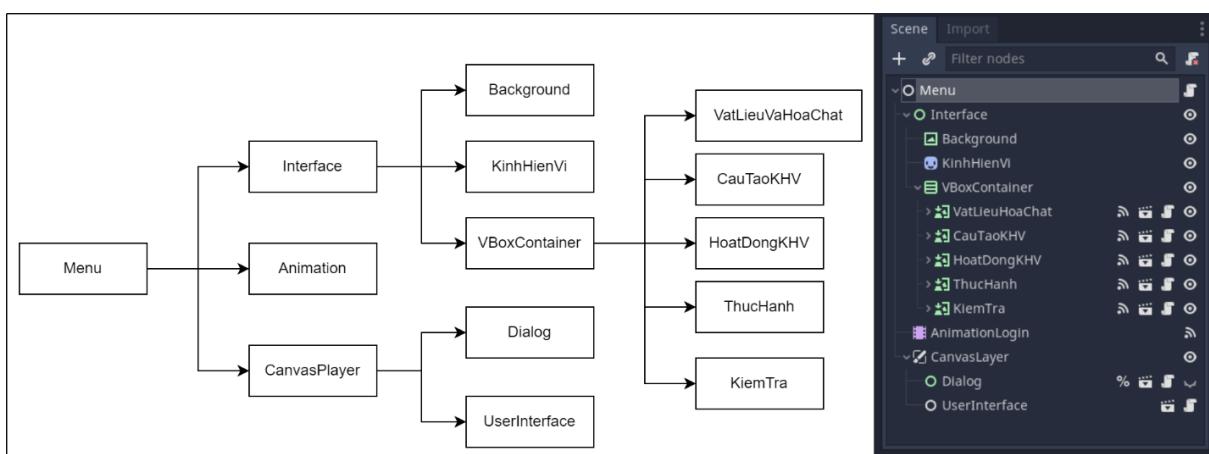
Triển khai

Giao diện màn hình được thiết kế thuận tay của người dùng, các nút chuyển cảnh được đặt ở phía bên phải gần chuột để dễ dàng lựa chọn và quan sát. Các nút chuyển cảnh to, hình ảnh bắt mắt. Các hộp thoại chứa thông tin của từng bài học có nền trắng chữ đen, kích thước lớn, giúp người dùng dễ dàng đọc được.



Hình 3.15. Giao diện màn hình chính thiết kế trong Photoshop

Cấu trúc cây của các đối tượng trong màn hình chính:

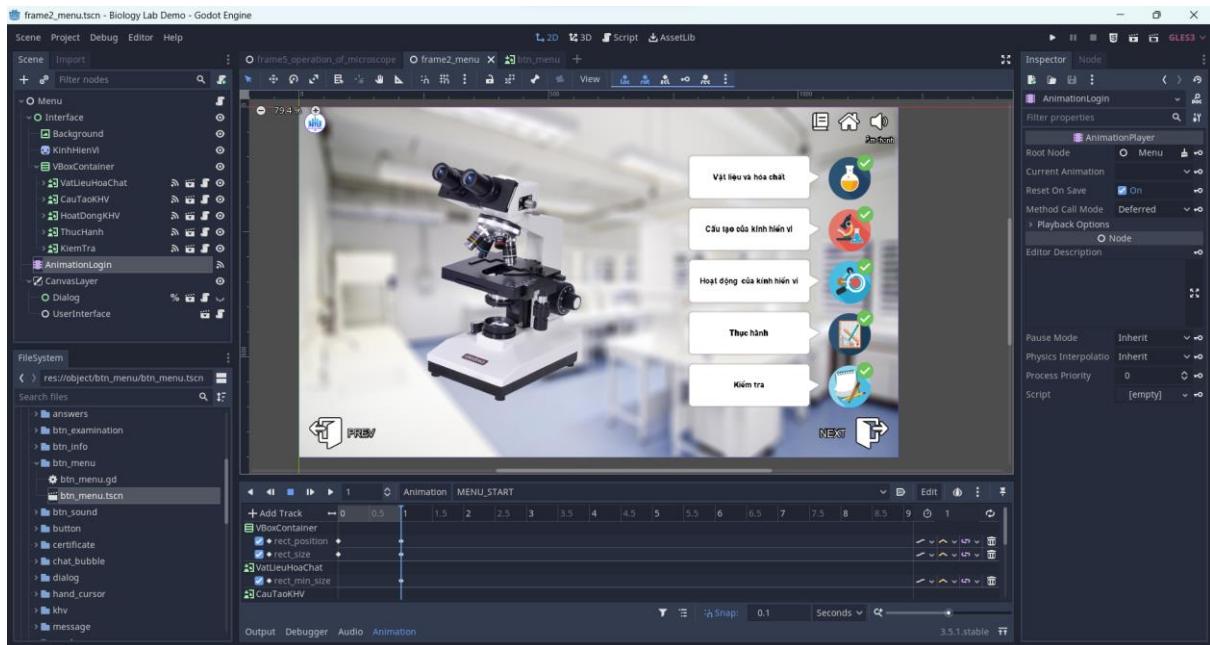


Hình 3.16. Sơ đồ cấu trúc cây của màn hình chính

Giải thích chức năng của các đối tượng:

- VboxContainer: Chứa các nút chuyển cảnh VatLieuVaHoaChat, CauTaoKHV, HoatDongKHV, ThucHanh, KiemTra.
- AnimationLogin: chạy animation hiển thị các nút chuyển cảnh.
- Dialog: scene hiển thị các thông tin, hướng dẫn.

- UserInterface: scene chứa giao diện dùng chung có tất cả các màn hình khác bao gồm các nút: nút trang chủ, nút âm thanh, logo trường Đại học Nha Trang, nút chuyển tới bài học tiếp theo, nút quay về bài học cũ.



Hình 3.17. Scene Main thiết kế trong Godot Engine

Kịch bản xử lý (menu.gd)

- ❖ Khai báo lớp kế thừa của màn hình chính là lớp Node.

```
extends Node
```

- ❖ Khởi tạo các biến đối tượng:

```
# _dialog ánh xạ tới scene Dialog (Hiển thị chỉ dẫn hoặc thông tin)
onready var _dialog = $CanvasLayer/CanvasLayer/Dialog
# _animationLogin ánh xạ tới AnimationLogin (chạy animation login)
onready var _animationLogin = $AnimationLogin
# _listBtn ánh xạ tới VBoxContainer (đối tượng chứa các nút chuyển cảnh)
onready var _listBtn = $Interface/VBoxContainer
# _ui ánh xạ tới scene UserInterface (giao diện người dùng: nút trang chủ, âm thanh,...)
onready var _ui = $CanvasLayer/UserInterface
```

- ❖ Khởi tạo các hằng số

```
# ID của Frame 2 ("Menu")
const ID_FRAME = Global.ID_FRAME_2
# Mảng tên của các màn hình
const FRAMES = ["VatLieuHoaChat", "CauTaoKHV", "HoatDongKHV",
"ThucHanh", "KiemTra"]
```

Đây là các hằng số được định nghĩa. ID_FRAME là hằng số cho ID_FRAME_2 từ biến toàn cục Global, và FRAMES là một mảng chứa các chuỗi tên các frame. Các hằng này dùng để truy xuất các giá trị ở biến toàn cục.

❖ Khởi tạo các biến:

```
# profile ánh xạ tới biến toàn cục profile (chứa data người dùng)
var profile = Global.profile
# final ánh xạ tới biến toàn cục final (chứa thông tin các bài học đã
hoàn thành)
var final = Global.final
# data ánh xạ tới tệp lệnh có data của dialog thuộc frame 2
var data = load("res://data/data_frame2.gd").new()
```

Trong biến toàn cục Gobal.profile có kiểu từ điển (Dictionary: khá giống với json) cho phép lưu trữ các cặp khóa – giá trị (key - value), muốn truy xuất các giá trị trong biến profile ta sẽ gọi thông qua các khóa. VD: print(profile["Menu"][GioiTieu]) kết quả sẽ bằng false. Chức năng của các biến này sẽ được giải thích rõ hơn ở hàm loadButton()

```
var profile: Dictionary = {
    "Menu": {
        "GioiTieu": false,
        "VatLieuHoaChat": false,
        "CauTaoKHV": false,
        "HoatDongKHV": false,
        "ThucHanh": false,
        "KiemTra": false,
    },
    <...>
}
```

Biến final cũng chứa các giá trị giống với biến profile tuy nhiên chức năng của final là lưu trữ các bài học đã hoàn thành.

```
var final: Dictionary = {
    "VatLieuHoaChat": false,
    "CauTaoKHV": false,
    "HoatDongKHV": false,
    "ThucHanh": false,
    "KiemTra": false,
}
```

❖ Hàm _ready:

```

func _ready():
    loadUI()
    loadButton()
    loadDialog()
    _animationLogin.play("MENU_START")

```

Đây là một hàm được gọi tự động khi đối tượng được tạo. Trong hàm này, các phương thức loadUI(), loadButton(), và loadDialog() được gọi để tải giao diện, các nút và hộp thoại. Hàm _animationLogin.play("MENU_START") được gọi để chơi hoạt cảnh "MENU_START".

❖ Hàm loadUI():

```

func loadUI():
    _ui.currentScene = self
    _ui.visibleBtnPrevScene(false)
    _ui.visibleBtnNextScene(false)

```

Đây là một hàm để tải giao diện. Trong hàm này, self được gán cho currentScene của _ui, và visibleBtnPrevScene(false) và visibleBtnNextScene(false) được gọi để ẩn các nút điều hướng tới bài học trước và sau.

❖ Hàm loadButton():

```

func loadButton():
    for i in profile:
        if i != ID_FRAME:
            checkFinal(i)
    loadOpenButton()
    loadActiveButton()

```

Đây là một hàm để tải các nút và kiểm tra trạng thái của chúng. Trong hàm này, một vòng lặp kiểm tra các frame trong biến profile và gọi checkFinal() để kiểm tra xem frame đó đã hoàn thành hay chưa. Sau đó, các nút mở và nút hoàn thành được tải lên ở 2 hàm là loadOpenButton và loadActiveButton.

❖ Hàm loadDialog():

```

func loadDialog():
    _dialog.closeDialog()
    _dialog.data = data.DATA_DIALOG

```

Đây là một hàm để tải hộp thoại. Trong hàm này, closeDialog() được gọi để đóng hộp thoại và data.DATA_DIALOG được gán cho data của _dialog.

❖ Hàm checkFinal(name: String):

```
func checkFinal(name: String):
    if data.final[name]:
        return

    for i in profile[name]:
        if profile[name][i] == false:
            return
    data.final[name] = true
```

Đây là một hàm để kiểm tra xem frame đã hoàn thành hay chưa. Nếu frame chưa hoàn thành, các mục con của frame sẽ được kiểm tra trong biến profile[name]. Nếu có mục con nào chưa hoàn thành, hàm sẽ trả về ngay lập tức. Nếu tất cả mục con đã hoàn thành, data.final[name] sẽ được đặt thành true.

❖ Hàm loadOpenButton():

```
func loadOpenButton():
    profile[ID_FRAME][FRAMES[0]] = true
    for i in FRAMES.size() - 1:
        if data.final[FRAMES[i]]:
            profile[ID_FRAME][FRAMES[i+1]] = true
```

Đây là một hàm để tải các nút mở. Biến profile[ID_FRAME][FRAMES[0]] được đặt thành true và vòng lặp kiểm tra các frame tiếp theo để đặt chúng thành true nếu các frame trước đó đã hoàn thành.

❖ Hàm loadActiveButton():

```
func loadActiveButton():
    for i in _listBtn.get_children():
        i.disabled = not profile[ID_FRAME][i.name]
        i.setTickFinal(final[i.name])
```

Đây là một hàm để tải các nút hoàn thành. Trong hàm này, một vòng lặp lấy các mục con của _listBtn và vô hiệu hóa nút nếu chúng không có trong profile[ID_FRAME]. Ngoài ra, setTickFinal() được gọi để thiết lập trạng thái của nút hoàn thành.

❖ Hàm loadWelcome():

```

func loadWelcome():
    if not data.profile[ID_FRAME][“GioiThieu”]:
        _dialog.showDialog()
        profile[ID_FRAME][“GioiThieu”] = true
        profile[ID_FRAME][“VatLieuHoaChat”] = true

```

Đây là một hàm để tải nội dung chào mừng. Hàm này được gọi khi sự kiện khi animation hiển thị các nút kết thúc. Trong hàm này, nếu người dùng lần đầu sử dụng ứng dụng thì data.profile[ID_FRAME][“GioiThieu”] là false. Hộp thoại dialog sẽ xuất hiện. Hai biến có khóa là “GioiThieu” và “VatLieuHoaChat” bằng true, để lần sau khi quay lại màn hình chính sẽ không chạy dialog nữa.

- ❖ func _on_AnimationPlayer_animation_finished(anim_name):

```

func _on_AnimationPlayer_animation_finished(anim_name):
    if anim_name == “MENU_START”:
        loadWelcome()

```

Đây là một hàm xử lý sự kiện được gọi khi hoạt cảnh trên đối tượng AnimationPlayer kết thúc. Trong trường hợp này, nếu hoạt cảnh kết thúc là “MENU_START”, loadWelcome() được gọi để tải nội dung chào mừng.

- ❖ Các hàm xử lý chuyển cảnh khi nhấn vào nút chuyển cảnh:

```

func _on_VatLieuHoaChat_button_up():
    Load.load_scene(self, Global.FRAME_3)

func _on_CauTaoKHKH_button_up():
    Load.load_scene(self, Global.FRAME_4)

func _on_HoatDongKHKH_button_up():
    Load.load_scene(self, Global.FRAME_5)

func _on_ThucHanh_button_up():
    Load.load_scene(self, Global.FRAME_6)

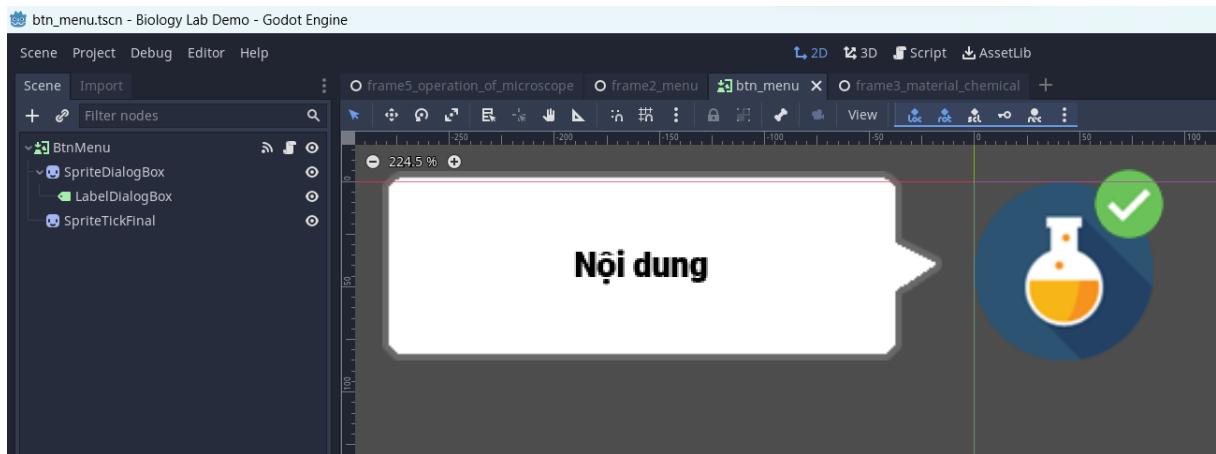
func _on_KiemTra_button_up():
    Load.load_scene(self, Global.FRAME_7)

```

Các scene đặc biệt chỉ có trong màn hình chính

Trong màn hình chính sẽ có 5 nút chuyển cảnh để chuyển người dùng sang các bài học khác nhau. Vì vậy để tiết kiệm thời gian các nút này sẽ được thiết kế thành các

scene để có thể tái sử dụng, tiết kiệm thời gian lập trình, tránh lặp lại các đoạn mã giống nhau.



Hình 3.18. Scene *btn_menu* trong Godot

Cấu trúc cây của các đối tượng trong scene *btn_menu*:

- **SpriteDialogBox:** hiển thị hình ảnh icon của nút chuyển cảnh. Như trong hình là bình thuốc hóa học (biểu tượng của nút Vật liệu và hóa chất).
- **LabelDialogBox:** hiển thị nội dung của màn hình sẽ chuyển cảnh. Di chuyển chuột vào mới xuất hiện.
- **SpriteTickFinal:** dấu tích xanh khi người dùng hoàn thành bài học.

Kịch bản

```
# Lớp kế thừa từ button_V2
# button_V2: xử lý âm thanh cho các nút
extends "res://object/button/button_v2.gd"

# _dialogBox ánh xạ tới SpriteDialogBox
onready var _dialogBox = $SpriteDialogBox
# _spriteTickFinal ánh xạ tới SpriteTickFinal
onready var _spriteTickFinal = $SpriteTickFinal

# Trạng thái chuột có hover hay ko
var isEnter: bool = false

# Tự động gọi khi màn hình load xong
func _ready():
    # Tắt nút, chờ cài đặt từ bên ngoài để mở nút
    self.disabled = true
    #Ẩn bảng thông báo (dialog kích thước = 0)
    _dialogBox.scale = Vector2.ZERO

# Khi chuột hover vào nút
func _process(_delta):
    if isEnter:
```

```

# Phóng to dialog để hiện rõ thông báo
_dialogBox.scale = lerp(_dialogBox.scale, Vector2.ONE, 0.15)
else:
    # Thu nhỏ dialog (biến mất)
    _dialogBox.scale = lerp(_dialogBox.scale, Vector2.ZERO, 0.15)

# Hàm hiển thị dấu tích xanh
func setTickFinal(value: bool):
    _spriteTickFinal.visible = value

# Phát âm thanh và cho biến isEnter = true
# Khi có tín hiệu chuột di chuyển vào nút
func _on_TextureButton_mouse_entered():
    hover_audio_play()
    isEnter = true

# Cho biến isEnter = false
# Khi có tín hiệu chuột di chuyển vào nút
func _on_TextureButton_mouse_exited():
    isEnter = false

```

3.3.3. Frame 3 – Màn hình vật liệu và hóa chất

Chức năng

Màn hình vật liệu và hóa chất, giới thiệu thông tin và chức năng của các dụng cụ và hóa chất được sử dụng trong phòng sinh học.

Yêu cầu

❖ Giao diện:

- Có hình ảnh minh họa các loại vật liệu và hóa chất trong phòng thí nghiệm.
- Trên mỗi vật liệu sẽ có một nút nhấn để hiển thị thông tin chi tiết khi người dùng nhấn vào.

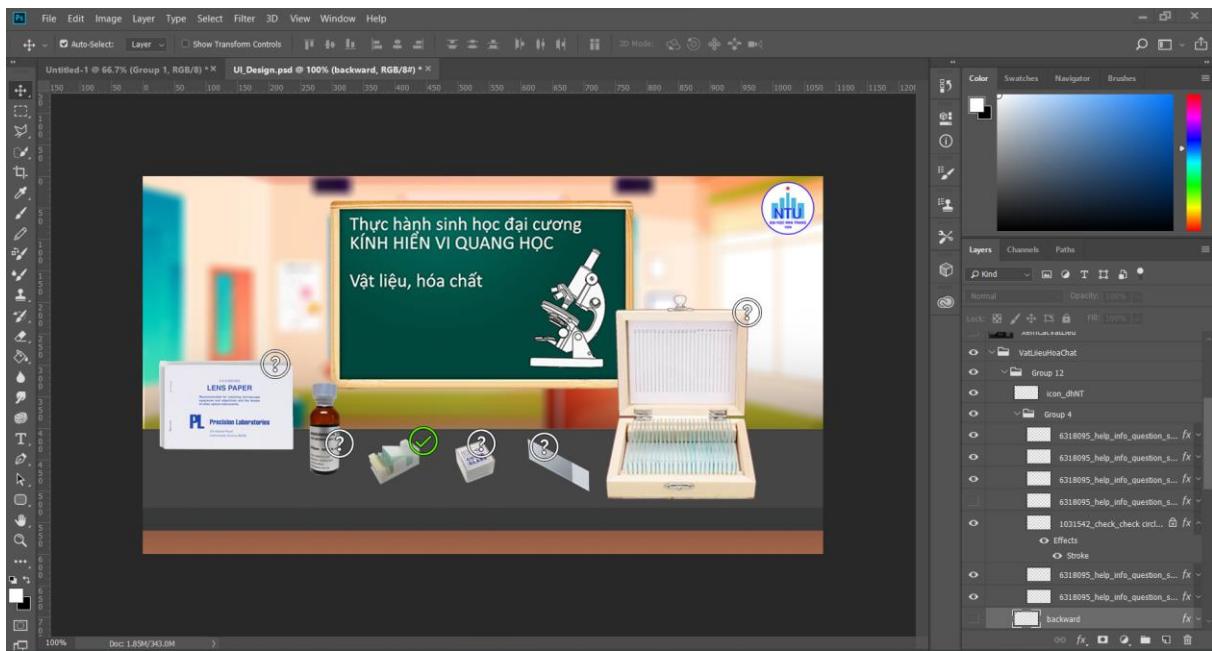
❖ Đặc tả hệ thống:

- Khi bắt đầu vào bài học sẽ có một dialog hiện ra giới thiệu về bài học. Chỉ hiện ra khi người dùng lần đầu vào bài học.
- Người dùng nhấn vào nút hiển thị chi tiết (dấu chấm hỏi), sẽ xuất hiện một dialog chứa thông tin chi tiết của vật liệu và bao gồm cả hình ảnh phóng to của vật liệu đó.

- Khi người dùng xem hết tất cả các vật liệu thì sẽ hiển thị thông báo để người dùng biết được mình đã hoàn thành bài học.
- Khi người dùng chưa hoàn thành xong bài học và thoát ra màn hình chính thì những nội dung mà người dùng học vẫn được lưu lại. Sau khi trở lại bài học, các mục đã hoàn thành sẽ được đánh dấu tích.

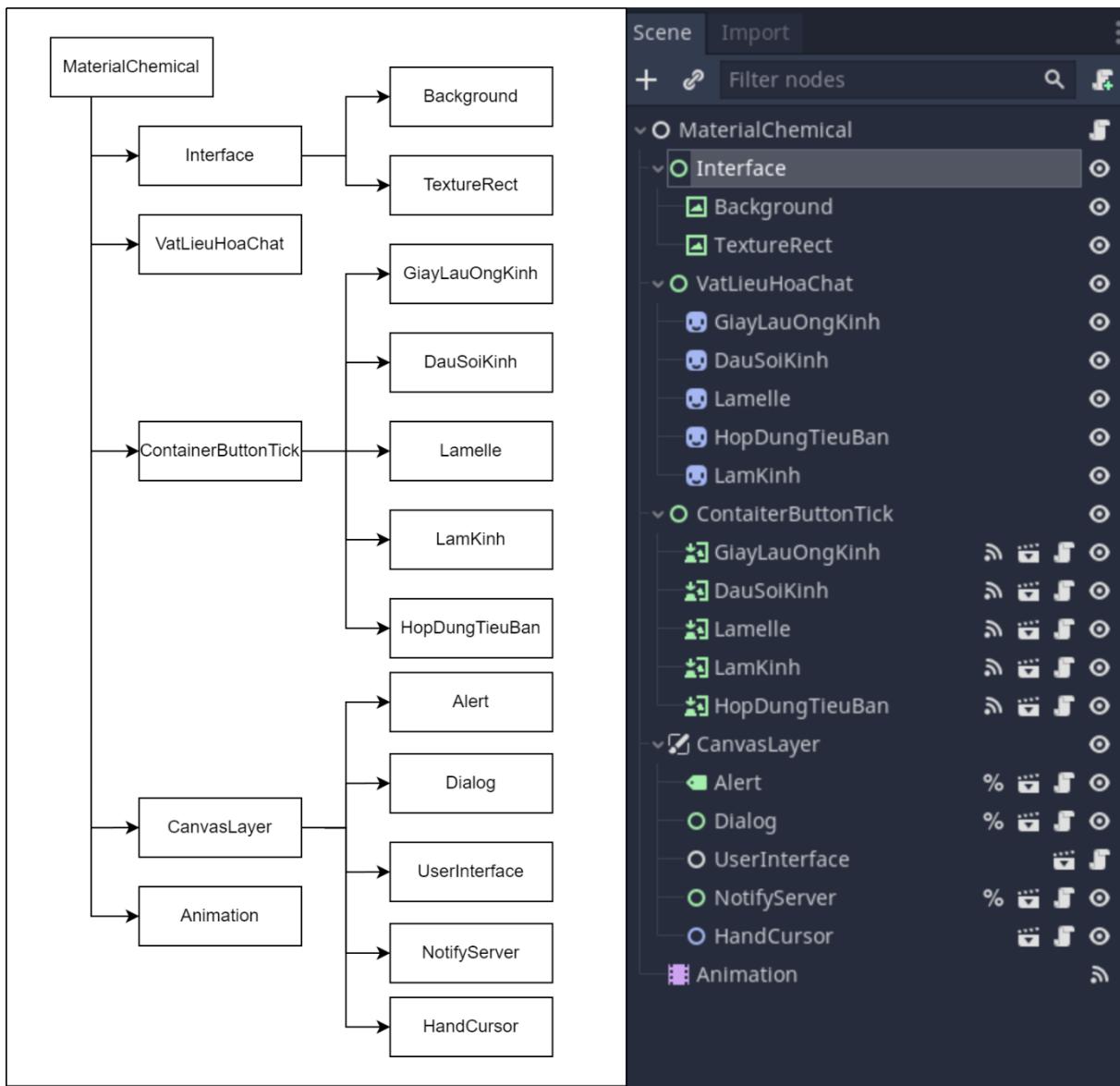
Triển khai

Giao diện màn hình vật liệu và hóa chất được thiết kế giống với khi đang ở phòng thực hành. Các hóa chất và dụng cụ thí nghiệm được đặt trên bàn, phía sau là bảng ghi chủ đề bài học. Các dấu chấm hỏi để chỉ dẫn người dùng nhập vào để hiển thị thông tin chi tiết chức năng và cách sử dụng của từng loại dụng cụ và hóa chất.



Hình 3.19. Màn hình vật liệu và hóa chất thiết kế trong Photoshop

Sau khi phân tích các đối tượng được mô tả trên giao diện của photoshop, màn hình vật liệu và hóa chất sẽ có cấu trúc cây của các đối tượng như sau:



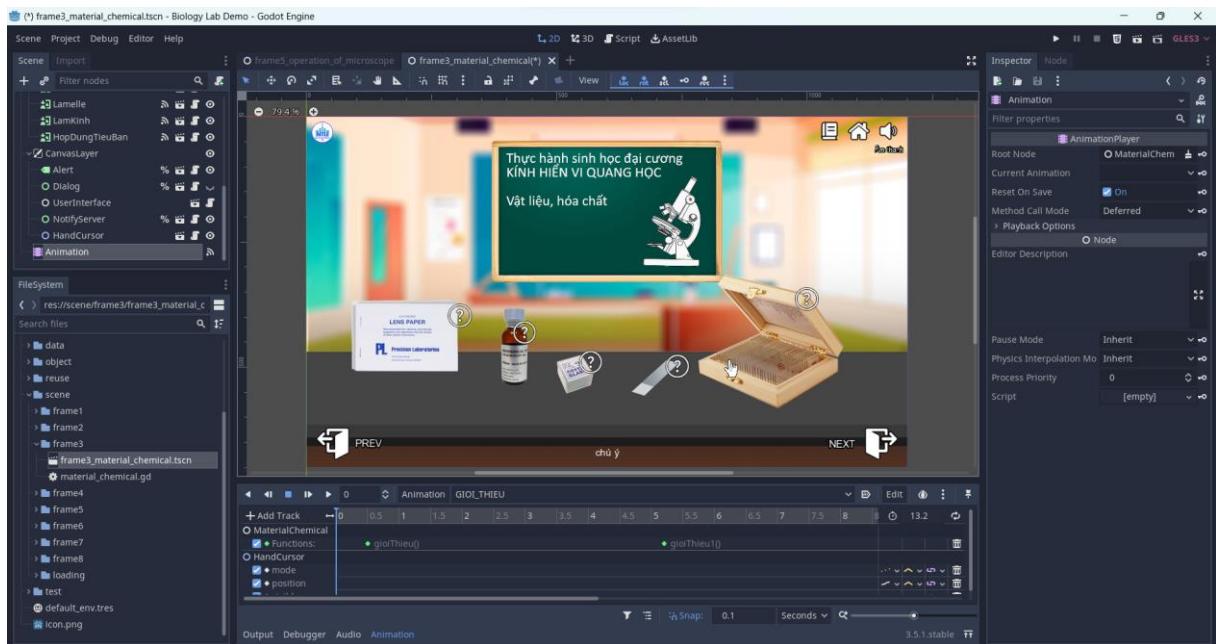
Hình 3.20. Sơ đồ cấu trúc cây của màn hình vật liệu và hóa chất

Giải thích chi tiết các đối tượng trong sơ đồ:

- Interface: bao gồm hình nền (Background) và tấm bảng đen ghi tên của bài học (TextureRect).
- VatLieuhoaChat: gồm 5 đối tượng kiểu Sprite (đối tượng dùng để hiển thị hình ảnh 2D trong Godot), hiển thị 5 đối tượng tương ứng với tên của các đối tượng.
- ContainerButtonTick: đối tượng chứa danh sách các nút thông tin chi tiết (dấu chấm hỏi). Khi nhấn vào các nút này và thả ra sẽ phát ra tín hiệu (signal) nút đã được nhấn (button_down) và đã được thả ra (button_up). Các nút nhấn này là một scene được kế thừa từ lớp button_V2.gd.
- CanvasLayer: các đối tượng giao diện tương tác với người dùng bao gồm: Alert (thông báo dạng thẻ mở ở dưới cùng hoặc ở trên màn hình), Dialog

(thông tin, thông báo, to hơn, dễ nhìn hơn), UserInterface (giao diện người dùng, dùng cho tất các frame có bà học), NotifyServer (thông báo kèm âm thanh, dạng popup, ở góc trên bên trái), HandCusor (khi cần gợi ý người dùng nên nhấn vào vị trí nào thì một icon – biểu tượng bàn tay được hiển thị và gợi ý cho người dùng).

- Animation: đối tượng thực hiện các đoạn hoạt hình.



Hình 3.21. Màn hình vật liệu và hóa chất trong Godot Engine

Kịch bản

Lấy ý tưởng từ mẫu thiết kế State (<https://refactoring.guru/design-patterns/state>), mỗi một bài hành động trong bài thực hành được xem là một trạng thái (state). Khi một giai đoạn trong bài học được hoàn thành sẽ chuyển sang giai đoạn tiếp theo. Theo yêu cầu đặc tả hệ thống, màn hình vật liệu và hóa chất sẽ chia thành 5 trạng thái: kiểm tra ban đầu, giới thiệu, vật liệu và hóa chất, kết thúc, hoàn thành. Tham khảo sơ đồ luồng chuyển đổi các trạng thái của màn hình vật liệu và hóa chất (Hình 3.22).

Khi bắt đầu hệ thống sẽ thực hiện “kiểm tra ban đầu” các giá trị đầu vào như dữ liệu các bài học mà sinh viên đã học, nếu sinh viên hoàn thành bài học rồi thì chuyển tới màn hình “hoàn thành”, nếu sinh viên chưa hoàn thành thì chuyển sinh viên tới bài “giới thiệu”. Sinh viên sẽ phải xem hết phần giới thiệu, sau đó là tự do xem chi tiết thông tin các “vật liệu và hóa chất”, khi hoàn thành sẽ hiển thị dialog với lời thông báo đã hoàn thành bài học (trạng thái kết thúc). Người dùng vẫn có thể xem lại bài học (trạng thái hoàn thành).



Hình 3.22. Sơ đồ luồng chuyển đổi các trạng thái ở màn hình vật liệu và hóa chất

Giải thích kịch bản

- ❖ Khai báo lớp kế thừa: kịch bản của màn hình vật liệu và hóa chất thuộc đối tượng Node, vì thế kịch bản sẽ kế thừa từ lớp Node.

extends Node

- ❖ Khai báo các biến và hằng:

```

onready var _dialog = $CanvasLayer/Dialog
onready var _containerButtonTick = $ContainerButtonTick
onready var _ui = $CanvasLayer/UserInterface
onready var _alert = $CanvasLayer/Alert
onready var _notify = $"%NotifyServer"
onready var _animation = $Animation
onready var _handCursor = $CanvasLayer/HandCursor

enum StateEnum {
    KIEM_TRA_BAN_DAU,
    GIOI_THIEU,
    VAT_LIEU_VA_HOA_CHAT,
    KET_THUC,
    HOAN_THANH
}

var idFrame = Global.ID_FRAME_3

var data = load("res://data/data_frame3.gd").new()

var profile = Global.profile[idFrame]
var final = Global.final[idFrame]

var state = StateEnum.KIEM_TRA_BAN_DAU

```

- ❖ Hàm _ready(): đây là hàm tự động gọi sau khi các đối tượng đã được nạp đầy đủ vào bộ nhớ ram. Hàm này thực hiện cài đặt các setting ban đầu cho bài học, như tài điểm số, tải dữ liệu thuyết trình, ẩn một vài nút, đặt màn hình tiếp theo sau khi học xong.

```

func _ready():
    loadHandCursor()
    loadDialog()
    loadTicks()
    loadUI()

func loadHandCursor():
    _handCursor.visible = false

func loadDialog():
    _dialog.current = self
    _dialog.methodClose = "actionCloseDialog"
    _dialog.visible = false
    _dialog.data = data.DATA_DIALOG
    _dialog.loadEditable(false)
    pass

func loadTicks():
    for i in _containerButtonTick.get_children():
        i.setIsTrueView(profile[i.name])
        i.loadTickTexture()
        i.current = self

func loadUI():
    _ui.currentScene = self
    _ui.setPrevScene(Global.NAME_FRAME_2, Global.FRAME_2)
    _ui.setNextScene(Global.NAME_FRAME_4, Global.FRAME_4)
    if not final:
        _ui.visibleBtnNextScene(false)
        _ui.visibleBtnPrevScene(false)

```

- ❖ Hàm _process(_delta): hàm _process được gọi tổng mỗi khung hình (frame) để xử lý logic của bài học.

```

func _process(_delta):
    match state:
        StateEnum.KIEM_TRA_BAN_DAU:
            if final == false:
                if profile["GioiThieu"] == false:
                    state = StateEnum.GIOI_THIEU
                else:
                    state = StateEnum.VAT_LIEU_VA_HOA_CHAT

```

```

        pass
    else:
        state = StateEnum.HOAN_THANH
        pass
    pass
StateEnum.GIOI_THIEU:
    callAnimationByName("GioiTieu")
    pass
StateEnum.VAT_LIEU_VA_HOA_CHAT:
    for i in _containerButtonTick.get_children():
        if i.isHover:
            showAlert(i.name)
        if i.getIsTick():
            if profile[i.name]:
                callDialogByName(i.name)
            else:
                callAnimationByName(i.name)
        pass
StateEnum.KET_THUC:
    callAnimationByName("KetThuc")
    pass
StateEnum.HOAN_THANH:
    _ui.visibleBtnNextScene(true)
    _ui.visibleBtnPrevScene(true)
    _alert.setText("Bạn đã tìm hiểu xong các hóa chất và vật
liệu cần thiết trong phòng thí nghiệm")
    _alert.setSticker(true)
    _alert.showAlert()

    for i in _containerButtonTick.get_children():
        if i.getIsTick():
            showDialog(i.name)
    pass
pass

```

Sử dụng câu lệnh match để kiểm tra giá trị của biến state (trạng thái) và thực hiện các hành động phù hợp với từng trạng thái. Các hành động này bao gồm hiển thị dialog, chạy animation, kiểm tra trạng thái hoàn thành, và điều hướng giao diện. VD: khi giá trị của state là “GIOI_THIEU” thì hệ thống sẽ thực hiện lệnh gọi chạy animation “GioiTieu”. Animation sẽ được chạy và sau khi animation kết thúc thì sẽ phát ra tín hiệu là animation “GioiTieu” đã kết thúc rồi, state (trạng thái) sẽ được chuyển thành “VAT_LIEU_VA_HOA_CHAT”, sau đó sẽ thực hiện các hoạt động của trạng thái tiếp theo.

- ❖ Cách xử lý khi mỗi một animation kết thúc:

```

func _on_Animation_animation_finished(anim_name):
    var name: String = convertNameAnimationToString(anim_name)
    match anim_name:
        “GIOI_THIEU”:
            profile[“GioiThieu”] = true
            state = StateEnum.VAT_LIEU_VA_HOA_CHAT
            pass
        “KET_THUC”:
            _notify.fire(“SUCCESS”, “Chúc mừng bạn đã hoàn thành bài
học này”)
            state = StateEnum.HOAN_THANH
            pass
        _:
            profile[name] = true
            handleTickView(name)
            actionFinalAnimation()
            pass
    _dialog.loadBtnReplay(true)
    _dialog.loadBtnClose(true)
    _dialog.loadEditable(true)

func actionFinalAnimation():
    match state:
        StateEnum.VAT_LIEU_VA_HOA_CHAT:
            _animation.stop()
            for i in profile:
                if profile[i] == false:
                    return
            final = true
            if final and state == StateEnum.VAT_LIEU_VA_HOA_CHAT:
                state = StateEnum.KET_THUC
                pass
        StateEnum.HOAN_THANH, StateEnum.KET_THUC:
            _notify.fire(“SUCCESS”, “Chúc mừng bạn đã hoàn thành bài
học này”)

```

3.3.4. Frame 4 – Màn hình cấu tạo kính hiển vi

Chức năng

Màn hình cấu tạo kính hiển vi sẽ cung cấp thông tin chi tiết về từng bộ phận của kính hiển vi. Vị trí, chức năng và đặc điểm các bộ phận của kính hiển vi.

Yêu cầu

❖ Giao diện:

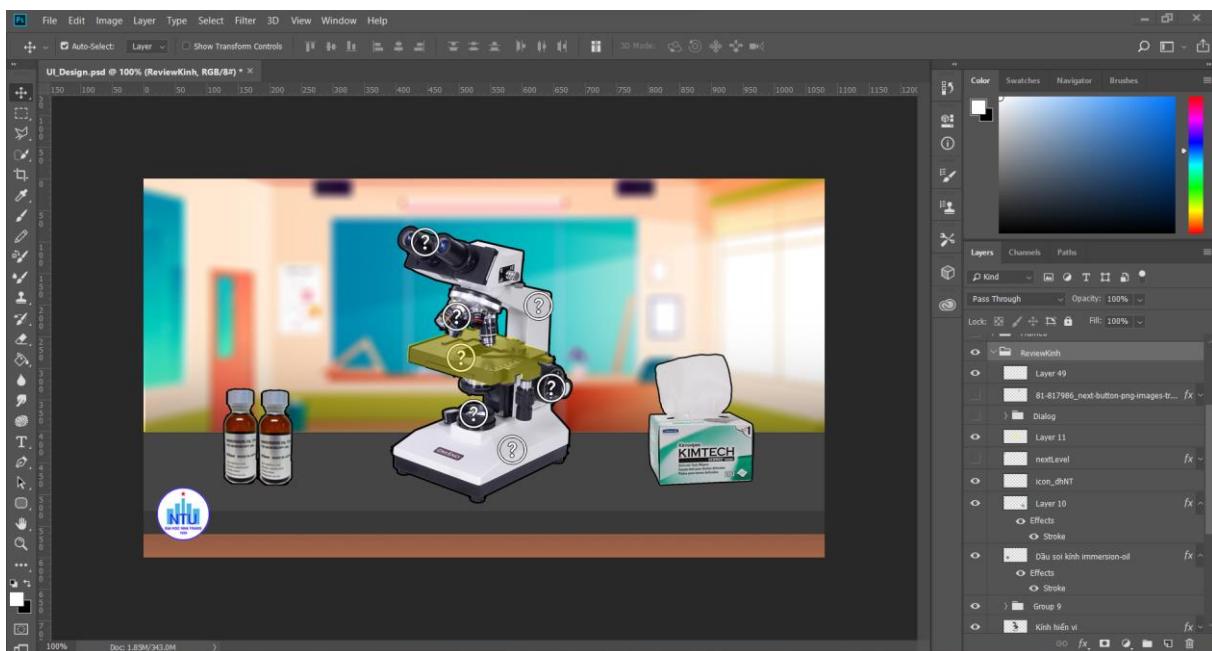
- Đặt kính hiển vi ở trung tâm màn hình.
- Ở mỗi bộ phận của kính hiển vi sẽ có một nút thông tin hiển thị chi tiết.
- Khi người dùng nhấp vào sẽ có một dialog hiện ra giới thiệu chi tiết của bộ phận được chọn.
- Khi chọn một bộ phận, bộ phận đó sẽ phát sáng hoặc đổi màu so với các bộ phận khác để dễ nhận biết.

❖ Đặc tả hệ thống:

- Khi vừa vào bài học lần đầu, một dialog hiển thị lời giới thiệu và chào mừng. Người dùng bắt buộc phải xem hết để tiếp tục bài học.
- Đảm bảo bộ phận được chọn sẽ được thể hiện nổi bật với các bộ phận khác.
- Sau khi học xong sẽ có phần thông báo đã hoàn thành bài học.

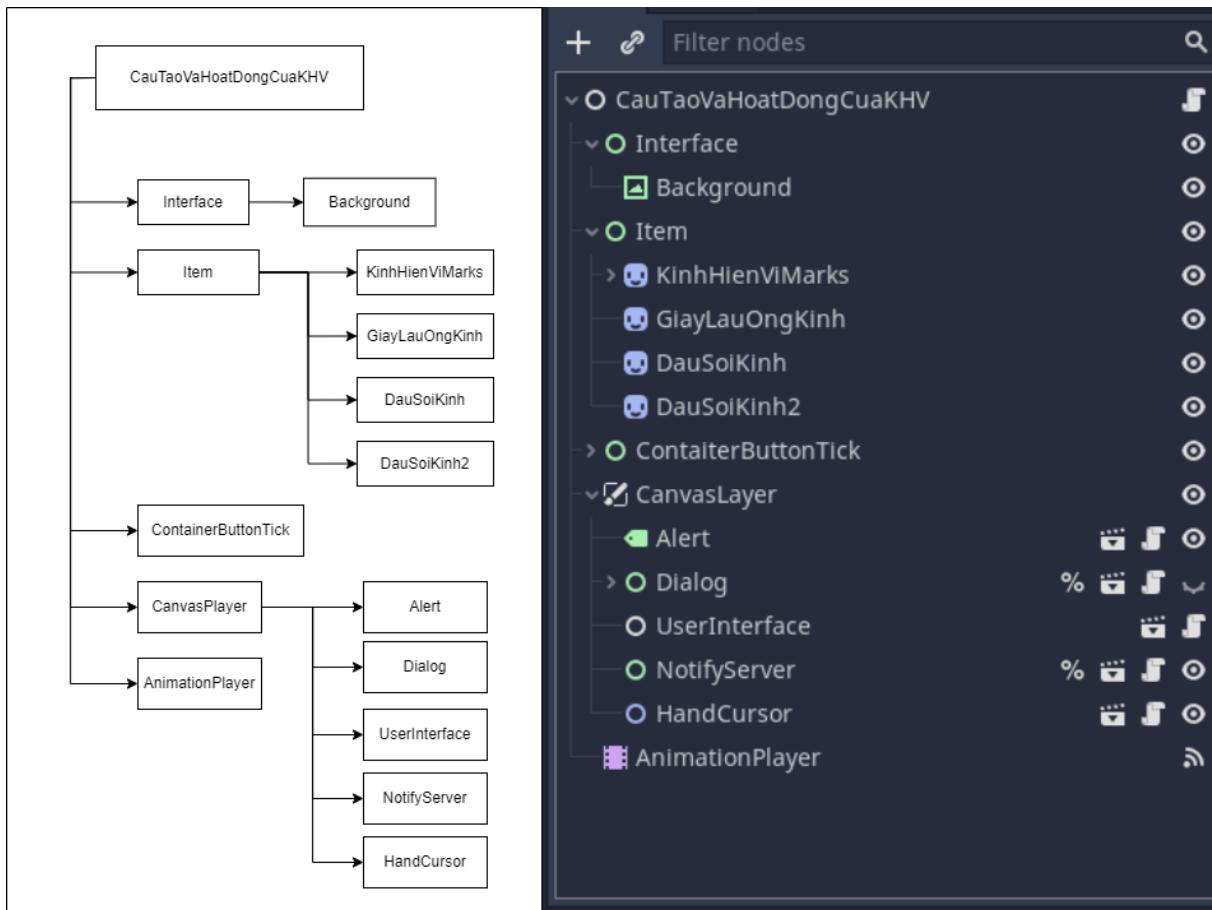
Triển khai

Qua yêu cầu về mô tả giao diện, màn hình kính hiển vi sẽ có giao diện, một kính hiển vi nằm ở giữa màn hình, kê bên sõi là 2 lọ chứa dầu soi kính và giấy lau kính để làm cho khung hình không bị trống. Lớp phủ màu vàng được làm mờ và chồng lên cùng với kích thước của các bộ phận sẽ làm rõ được vị trí của bộ phận đó mà không làm ảnh hưởng đến tầm nhìn.



Hình 3.23. Màn hình cấu tạo kính hiển vi được thiết kế trên Photoshop

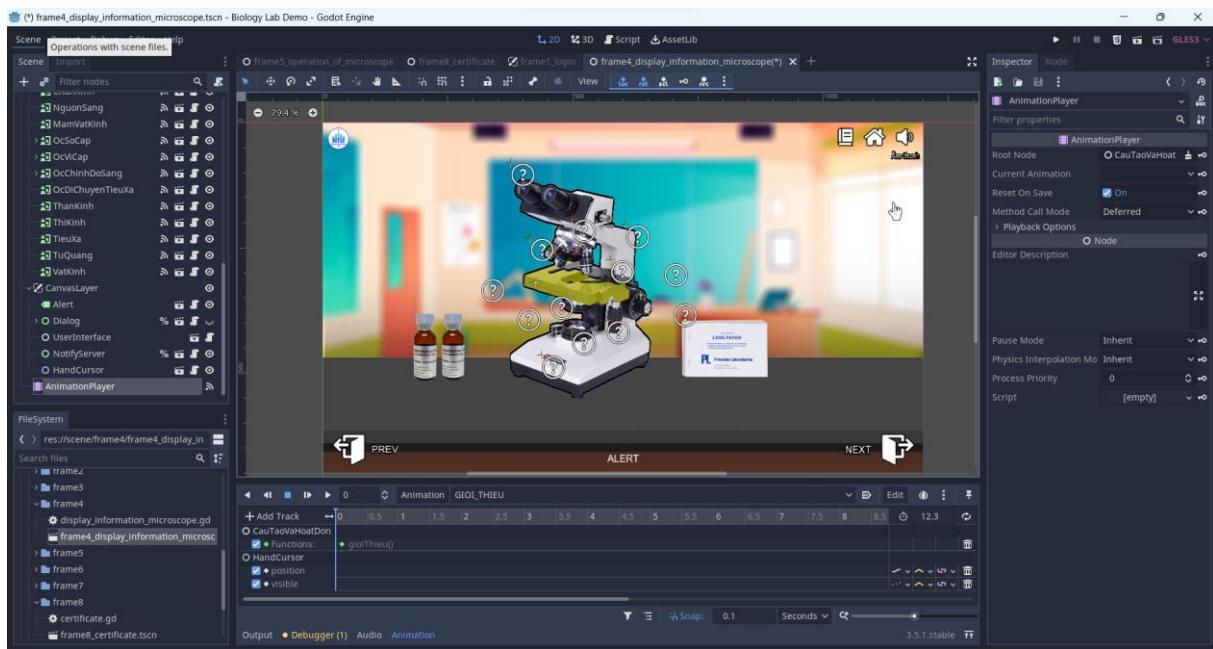
Cấu trúc cây của màn hình cấu tạo kính hiển vi có rất nhiều đối tượng được thêm vào chủ yếu là các lớp mark (lớp phủ màu vàng lên các bộ phận của kính) và nút thông tin của từng mark.



Hình 3.24. Sơ đồ cấu trúc cây của màn hình cấu tạo kính hiển vi

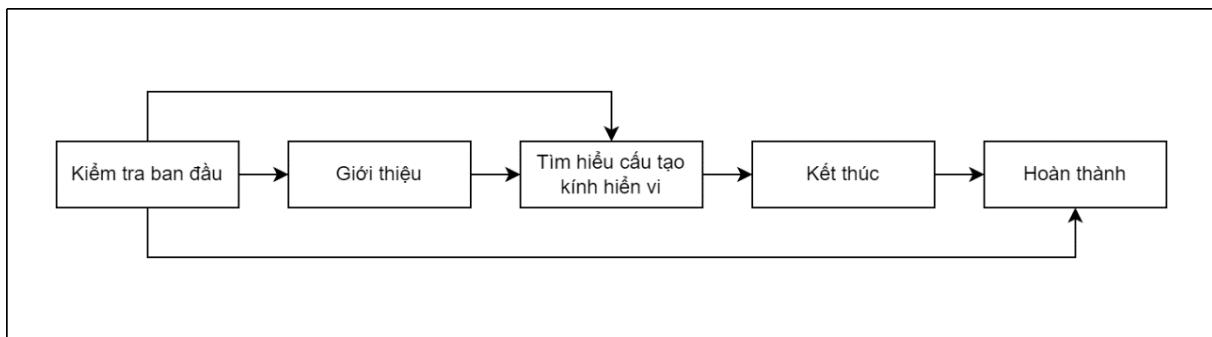
Trong scene tree trên cần chú ý đến các đối tượng sau:

- Item: đối tượng quản lý các hình ảnh của các dụng cụ trên màn hình.
- KinhHienViMarks: là đối tượng sprite chứa ảnh của kính hiển vi, các nút con (child) của KinhHienViMarks là các lớp phủ màu vàng (mark) được đặt đúng với từng vị trí của các bộ phận của kính.
- ContainerButtonTick: chứa các nút thông tin (hình chấm dấu hỏi). Tên của các button (nút) được đặt cùng tên với các bộ phận trên kính hiển vi. Khi nhấn vào một nút sẽ dùng đúng tên của nút đó để tìm ra mark cùng tên để hiển thị.
- CanvasLayer: chứa các đối tượng thuộc giao diện người dùng.
- AnimationPlayer: đối tượng chạy các hoạt hình được tạo sẵn.



Hình 3.25. Màn hình cấu tạo kính hiển vi được thiết kế trên Godot Engine
Kịch bản

Theo đặc tả hệ thống, người dùng sẽ phải xem hết phần giới thiệu, sau đó xem hết các chi tiết thông tin các bộ phận của kính hiển vi, khi hoàn thành sẽ hiển thị dialog với lời thông báo đã hoàn thành bài học. Người dùng vẫn có thể xem lại bài học. Quy trình xử lý các state tương tự như ở frame 3, sau khi một nhiệm vụ hoàn thành sẽ chuyển đến nhiệm vụ tiếp theo cho đến khi state cuối cùng và kết thúc.



Hình 3.26. Sơ đồ luồng chuyển đổi trạng thái của màn hình cấu tạo kính hiển vi
Theo sơ đồ luồng chuyển đổi các trạng thái của màn hình cấu tạo kính hiển vi (Hình 3.26), cần chú ý mã khi state ở trạng thái tìm hiểu cấu tạo kính hiển vi.

```

StateEnum.TIM_HIEU_CAU_TAO_KHV:
    for i in _ticks.get_children():
        if i.isHover:
            showAlert(i.name)
        if i.getIsTick():
            loadMarkActive(i.name)
            if profile[i.name]:
                callDialogByName(i.name)
  
```

```

else:
    _animation.stop()
    callPlayAnimationByName(i.name)

```

Khi state bằng TIM_HIEM_CAU_TAO_KHV sẽ thực hiện chạy vòng lặp for để duyệt qua tất cả các phần tử nút thông tin. Khi chuột di chuyển vào khu vực hiển thị của một nút (i.isHover bằng true) thì sẽ hiện thẻ thông tin là tên bộ phận của kính hiển vi (showAlert(i.name)). Nếu nhấp vào nút thông tin (i.getIsTick) thì sẽ hiển thị lớp phủ màu vàng (mark) lên bộ phận tương ứng của kính hiển vi đồng thời kiểm tra xem người dùng đã học qua bộ phận này của kính hay chưa, nếu chưa thì chạy hoạt hình của bộ phận tương ứng. Nếu đã xem rồi thì sẽ chỉ chạy hộp nội dung của bộ phận đó.

3.3.5. Frame 5 – Màn hình hướng dẫn thực hành

Chức năng

Hướng dẫn người dùng từng bước sử dụng kính hiển vi quang học. Các bước chuẩn bị trước khi dùng kính, những điểm cần chú ý, và lý thuyết cần biết trong quá trình thực hành. Ngoài ra trong bài hướng dẫn sẽ có các bài tập nhỏ để người dùng thực hành quan sát mẫu vật.

Yêu cầu

❖ Giao diện

- Hiển thị kính hiển vi 3D ở bên phải, giấy lau ống kính và dầu soi kính nằm ở bên phải của kính hiển vi, công tắc và ổ cắm điện nằm ở trên trái kính.
- Có một bong bóng ảnh đại diện của giáo viên (avatar) để hướng dẫn tới người dùng các thao tác.
- Thiết kế giao diện các nút bấm và các thanh trượt để thể hiện thao tác điều chỉnh nút sơ cấp, nút vi cấp, độ sáng của đèn và các nút chuyển đổi các thấu kính hiển vi.

❖ Đặc tả hệ thống

- Đảm bảo người dùng học đúng theo trình tự: chuẩn bị, cắm điện, bật điện, điều chỉnh ánh sáng, thực hành quan sát vật mẫu với thấu kính 4x, nhỏ dầu, thực hành điều chỉnh vật kính 100x.
- Song song với trình tự thực hiện cần có các thông tin về lý thuyết và phương pháp thực hiện, những điểm cần lưu ý khi thực hiện từng bước trong quá trình thực hành quan sát mẫu vật bằng kính hiển vi.

- Bóng bóng thông tin có thể kéo thả tùy ý, đảm bảo thông tin không bị che khuất khi kéo thả ở các góc màn hình.
- Khi điều chỉnh các thông tin ở bảng điều khiển (panel) của kính hiển vi, các thao tác điều chỉnh sẽ đồng bộ với các bộ phận của kính hiển vi. VD: khi điều chỉnh nút sơ cấp thì bàn đế vật mẫu của kính hiển vi phải di chuyển lên xuống theo cường độ của thanh trượt khi trượt sang phải hay sang trái.
- Có các hoạt hình (animation) minh họa cho hoạt động nhỏ đầu, lau vật kính bằng giấy lau kính.
- Có các bài kiểm tra nhỏ để người dùng tự thao tác điều chỉnh các vật kính để học quan sát mẫu như thế nào là đúng. Có nút kiểm tra để kiểm tra xem người dùng đã thao tác chính xác hay chưa.
- Khi kết thúc bài học sẽ hiển thị nút chuyển sang bài học tiếp theo và nút trở lại bài học trước.
- Những hoạt động có thông tin hiển thị nhiều sẽ sử dụng hội thoại (dialog) để thể hiện đầy đủ thông tin. Những thông tin như các bước thực hiện hay các đoạn hướng dẫn nhấn nút sẽ có các bong bóng chat hiển thị để người dùng nắm được thông tin.
- Khi kết thúc một dialog cần có một biểu tượng (icon) hình ngón tay chỉ vào nút đóng (close) dialog để tắt dialog đi.

Triển khai

Qua mô tả giao diện hướng dẫn thực hành sử dụng kính hiển vi, màn hình hướng dẫn (Hình 3.27) sẽ có bảng điều khiển thao tác kính hiển vi ở phía bên phải, phía bên trái là kính hiển vi và các dụng cụ để thực hiện bài thực hành.

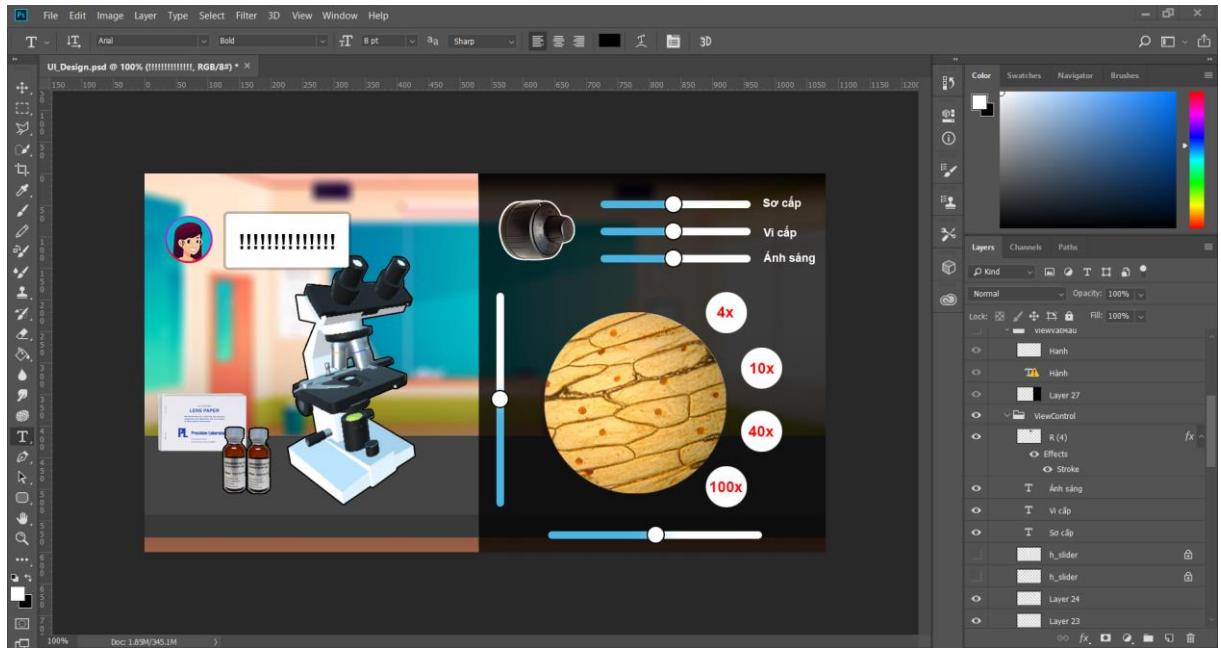
❖ Bảng điều khiển thao tác kính hiển vi ở bên phải sẽ bao gồm:

- 5 thanh trượt: nút sơ cấp, nút vi cấp, nút chỉnh ánh sáng, 2 thanh trượt còn lại điều chỉnh bàn kính di chuyển vào trong hoặc sang trái, sang phải.
- 4 nút bấm: trên mỗi nút bấm thông tin độ phóng đại của vật kính.
- Góc trên bên trái sẽ là hình của vật kính tương ứng được lựa chọn.
- Ở giữa bảng điều khiển thao tác là một tấm ảnh hình tròn, đại diện cho vùng quan sát được trên vật mẫu khi nhìn vào thị kính.

❖ Bên trái màn hình hướng dẫn thực hành sẽ bao gồm:

- Mô hình kính hiển vi 3D.

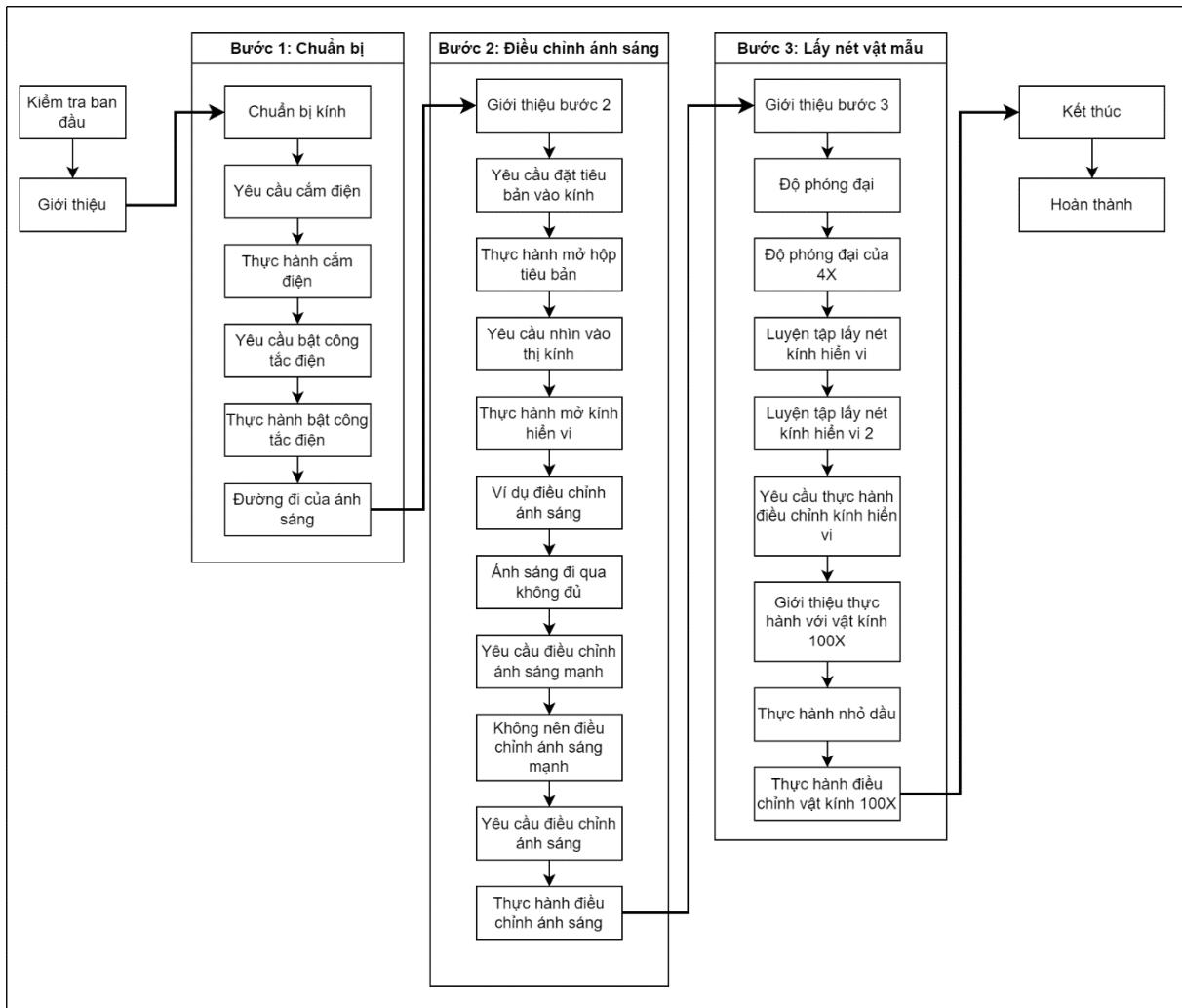
- Bên trái kính hiển vi là dây cắm điện và ổ điện (bổ sung ở màn hình Godot)
- Bên phải kính hiển vi là các vật liệu để thực hành như: hộp đựng tiêu bản, dầu soi kính, giấy lau ống kính.



Hình 3.27. Giao diện hướng dẫn thực hành thiết kế ở Photoshop

Kịch bản

Ý tưởng viết script tương tự như ở frame 3 và frame 4 sử dụng mẫu thiết kế state để thực hiện chuyển đổi từ trạng thái này đến trạng thái khác để hoàn thành từng nhiệm vụ một. Ở frame 5 – màn hình hướng dẫn thực hành sẽ được lên kịch bản bởi tệp lệnh operation_of_microscope.gd.



Hình 3.28. Sơ đồ các trạng thái trong màn hình hướng dẫn thực hành

❖ **Khởi tạo chương trình:** khi màn hình tải, tải xong các tài nguyên cho màn hình hướng dẫn bao gồm các node, hình ảnh, âm thanh, các dữ liệu cho dialog, và mô hình kính hiển vi. Logic chương trình sẽ chạy hàm `_ready` để cài đặt các thông số ban đầu cho các biến của bài thực hành. Ngoài ra biến state được khởi tạo có giá trị là `KIEM_TRA_BAN_DAU` từ khi tải chương trình.

```

func _ready():
    # Ẩn con trỏ hình bàn tay
    loadHandCursor()
    # Nạp file dữ liệu chứa âm thanh và văn bản cho dialog
    loadDialog()
    # Nạp file dữ liệu chứa âm thanh và văn bản cho bong bóng chat
    loadChatBubble()
    # Nạp danh sách vật mẫu vào menu lựa chọn vật mẫu
    loadItemList()
    # Tải giao diện người dùng
    loadUI()
    # Đặt vật kính ban đầu của kính hiển vi là 4X có chỉ số là 0
    loadKHV(0)
    # Cấu hình các đối tượng khác như ẩn nút kiểm tra, cấu hình KHV
    loadOther()

```

❖ **Kiểm tra ban đầu:** sau khi thực hiện cấu hình các trạng thái ban đầu cho các đối tượng ở hàm _ready. Chương trình tiếp tục thực hiện hàm _process để thực hiện logic trò chơi, kiểm tra biến state ở trạng thái nào và thực hiện các hành động ở trạng thái tương ứng. Hàm _process sẽ sử dụng câu lệnh match để kiểm tra các trường hợp của state. VD: Nếu state có giá trị là KIEM_TRA_BAN_DAU thì hàm _process sẽ thực hiện lấy dữ liệu profile của đối tượng Global và gán cho biến profile (chứa thông tin các bài học của người dùng). Sau đó chuyển trạng thái state thành GIOI_THIEU.

```

func _process(_delta):
    match state:
        StateEnum.KIEM_TRA_BAN_DAU:
            profile = Global.profile[FRAME_ID]
            state = StateEnum.GIOI_THIEU
            pass
        StateEnum.GIOI_THIEU:
            if profile["GioiTieu"]:
                state = StateEnum.CHUAN_BI_KINH
            else:
                if not stopAnimation:
                    callAnimationByName("GioiTieu")
            pass

```

❖ **Giới thiệu:** Khi state ở trạng thái giới thiệu sẽ thực hiện câu lệnh if kiểm tra xem biến profile[“GioiTieu”] có bằng true hay không. Biến profile có kiểu dữ liệu từ điển (Dictionary), với khóa là “GioiTieu” kết quả trả về là true/false. Nếu bằng true, có nghĩa là người dùng đã xem qua đoạn giới thiệu, thực hiện gán cho biến state bằng

StateEnum.CHUAN_BI_KINH, để chuyển sang trạng thái tiếp theo là chuẩn bị kính. Nếu profile[“GioiThieu”] bằng false sẽ thực hiện kiểm tra xem biến stopAnimation có bằng false hay không, nếu bằng false có nghĩa được phép chạy animation có tên “GioiThieu” bằng cuộc gọi hàm callAnimationByName(“GioiThieu”). Nếu bằng biến stopAnimation bằng true thì không thực hiện hành động nào. Logic chương trình ở các trạng thái khác cũng sẽ sử dụng cấu trúc tương tự, sẽ kiểm tra xem trong biến profile với khóa tương ứng có bằng true hay không nếu bằng true có nghĩa người dùng đã học qua, nếu bằng false có nghĩa là chưa học. Nếu học rồi thì chuyển sang trạng thái mới, nếu chưa học thì thực hiện các hoạt động trong state hiện tại.

```
StateEnum.GIOI_THIEU:
    if profile["GioiThieu"]:
        state = StateEnum.CHUAN_BI_KINH
    else:
        if not stopAnimation:
            callAnimationByName("GioiThieu")
        pass
```

- ❖ Bước 1 – Chuẩn bị: trong bước chuẩn bị sẽ bao gồm 6 trạng thái cần thực hiện bao gồm: chuẩn bị kính, yêu cầu cầm điện, yêu cầu bật công tắc điện, thực hành bật công tắc điện, đường đi của ánh sáng.

Sau khi kết thúc trạng thái giới thiệu, biến state sẽ có trạng thái là CHUAN_BI_KINH. Sau khi kiểm tra profile xem người dùng đã học nội dung ở state chuẩn bị kính chưa, nếu chưa sẽ gọi animation “ChuanBiKinh”.

```
StateEnum.CHUAN_BI_KINH:
    if profile["ChuanBiKinh"]:
        state = StateEnum.YEU_CAU_CAM_DIEN
    else:
        if not stopAnimation:
            callAnimationByName("ChuanBiKinh")
        pass
```

Sau khi người dùng xem hết đoạn hoạt hình có chứa nội dung của chuẩn bị kính sẽ gọi một signal là _on_AnimationPlayer_animation_finished(anim_name) khi đoạn hoạt hình kết thúc. Với tham số truyền vào là anim_name là tên của đoạn animation vừa kết thúc. Ở đoạn code bên dưới, khi animation có tên là CHUAN_BI_KINH kết thúc sẽ được thực hiện các hành động sau. Đặt biến stopAnimation = true, animation sẽ không được chạy lại ở hàm _process. Gọi hàm handleEndAnimation(), hàm này sẽ thực hiện

cấu hình các nút trên dialog như: hiển thị nút replay (lặp lại) – Gọi hàm loadBtnReplay có tham số là true, hiển thị nút close (nút đóng) - loadBtnClose(true), có cho phép tua đoạn âm thanh được phát trên dialog - loadEditable(true).

```
func _on_AnimationPlayer_animation_finished(anim_name):
    match anim_name:
        "GIOI_THIEU":
            stopAnimation = true
            handleEndAnimation()
        "CHUAN_BI_KINH":
            stopAnimation = true
            handleEndAnimation()

func handleEndAnimation():
    _dialog.loadBtnReplay(true)
    _dialog.loadBtnClose(true)
    _dialog.loadEditable(true)
```

Khi xem đến hết nội dung trong dialog, animation sẽ hiện icon ngón tay chỉ vào nút close trên dialog để gợi ý người dùng đóng dialog. Khi người dùng thực hiện nhấn vào nút close thì hàm actionCloseDialog() được gọi để xử lý sự kiện đóng dialog. Ứng với giá trị của state thì sẽ có các hành động phù hợp. Biến state hiện đang có giá trị là CHUAN_BI_KINH, khi đóng dialog hàm actionCloseDialog sẽ thực hiện gán giá trị true cho biến profile[“ChuanBiKinh”] = true có nghĩa là người dùng đã xem hết nội dung ở dialog, sau đó đặt biến stopAnimation = false (có thể chạy animation trong hàm _process), cuối cùng chuyển state thành YEU_CAU_CAM_DIEN.

```
func actionCloseDialog():
    match state:
        StateEnum.GIOI_THIEU:
            profile["GioiThieu"] = true
            stopAnimation = false
            state = StateEnum.CHUAN_BI_KINH
        StateEnum.CHUAN_BI_KINH:
            profile["ChuanBiKinh"] = true
            stopAnimation = false
            state = StateEnum.YEU_CAU_CAM_DIEN
```

Khi state có giá trị là YEU_CAU_CAM_DIEN (yêu cầu cảm điện), người dùng sẽ được hiển thị cho xem bong bóng chat, yêu cầu cảm điện. Quá trình xử lý logic của trạng thái yêu cầu cảm điện cũng tương tự như trạng thái chuẩn bị kính.

Kết thúc state yêu cầu cảm điện sẽ là thực hành cảm điện. Biến state sẽ có giá trị là THUC_HANH_CAM_DIEN, câu lệnh if profile[“ThucHanhCamDien”] để kiểm tra

xem người dùng đã cắm điện cho kính hiển vi hay chưa, nếu bằng true thì người dùng đã cắm điện rồi, thực hiện câu lệnh `_pluginKHZ.snapFull = true` để đặt trạng thái của dây cắm điện là đã cắm. Đối tượng `_pluginKHZ` là một scene chứa 2 đối tượng là ô điện và dây cắm điện. Khi `_pluginKHZ.snapFull` bằng true thì có nghĩa ô điện đã có điện, bằng false thì chưa cắm điện. Sau khi đặt `_snapFull = true` thì chuyển trạng thái thực hành cắm điện sang trạng thái yêu cầu bật công tắc điện. Nếu người dùng chưa cắm điện thì sẽ thực hiện kiểm tra xem người dùng đã thực hiện click chọn vào ô dây điện (`_pluginKHZ.selected`) và đã cắm vào ô (`_pluginKHZ.isSnap`) nếu cả 2 đều bằng true thì có nghĩa là người dùng đã cắm điện. Gọi animation “ThucHanhCamDien” chạy animation hướng dẫn người dùng cắm điện. Kiểm tra lại `_pluginKHZ.snapFull` có bằng true không, nếu bằng true thì ấn đi con trỏ hình ngón tay, dừng animation “ThucHanhCamDien” lại và gọi ra một thông báo `_notify.fire("SUCCESS", "Kính hiển vi đã có điện")` hiển thị dòng thông báo “Kính hiển vi đã có điện”. Sau đó đặt giá trị true cho biến `profile["ThucHanhCamDien"]` và chuyển trạng thái của state thành yêu cầu bật công tắc điện ở câu lệnh `state = StateEnum.YEU_CAU_BAT_CONG_TAC_DIEN`.

```
StateEnum.THUC_HANH_CAM_DIEN:
    if profile["ThucHanhCamDien"]:
        _pluginKHZ.snapFull = true
        state = StateEnum.YEU_CAU_BAT_CONG_TAC_DIEN
    else:
        if _pluginKHZ.isSnap and _pluginKHZ.selected==false:
            _pluginKHZ.snapFull = true
        if not stopAnimation:
            callAnimationByName("ThucHanhCamDien")
        if _pluginKHZ.snapFull == true:
            _handCursor.visible = false
            _animation.stop()
            _notify.fire("SUCCESS", "Kính hiển vi đã có điện")
            profile["ThucHanhCamDien"] = true
            state = StateEnum.YEU_CAU_BAT_CONG_TAC_DIEN
    pass
```

Qua quá trình minh họa và giải thích code ở các trạng thái: kiểm tra ban đầu, giới thiệu, chuẩn bị kính, yêu cầu cắm điện, thực hành cắm điện. Logic chương trình ở mỗi trạng thái đều khá tương đồng nhau ở khâu kiểm tra giá trị đầu vào là biến `profile` nếu giá trị của khóa có trong biến `profile` bằng true thì bài học đã được học và chuyển sang bài học tiếp theo, nếu bằng false thì thực hiện các hoạt động cho người dùng thực hành

ví dụ như: hiển thị dialog để người dùng đọc tài liệu hướng dẫn, yêu cầu thực hiện một thao tác với kính, thực hiện kiểm tra kết quả sau khi thực hành.

3.3.6. Frame 6 – Màn hình thực hành

Chức năng

Sau khi hoàn thành bài hướng dẫn thực hành ở frame 5, màn hình thực hành sẽ được mở khóa. Ở màn hình thực hành người dùng sẽ dựa theo những kiến thức đã học để áp dụng vào bài thực hành.

Yêu cầu

❖ Giao diện

- Màn hình thực hành sẽ có giao diện giống với giao diện của frame 5. Tuy nhiên ở frame 6 – màn hình thực hành sẽ không có các dialog hay bong bóng chat.

❖ Đặc tả hệ thống

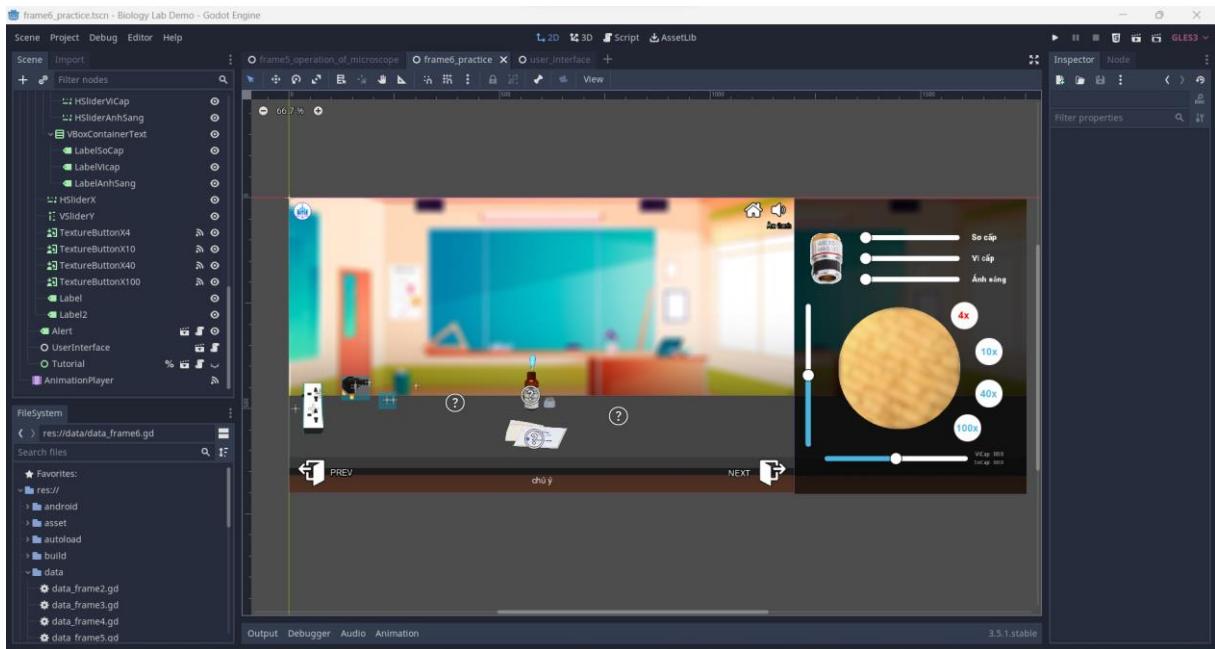
- Cho phép người dùng tự do thao tác thử nghiệm với các hóa chất, vật liệu, kính hiển vi.
- Hộp đựng tiêu bản: khi click vào dấu chấm hỏi nằm trên hộp đựng tiêu bản, danh sách các mẫu vật để quan sát dưới kính hiển vi sẽ hiện ra. Người dùng có thể lựa chọn các mẫu vật để đặt lên bàn kính. Các mẫu vật sẽ bao gồm: máu người, khí khổng (một lớp mỏng bề mặt của lá cây), khoang miệng (các tế bào mô được lấy từ bên trong khoang miệng).
- Để mở giao diện thao tác kính hiển vi, người dùng cần đặt tiêu bản vào bàn kính của kính hiển vi. Sau đó click vào nút chấm hỏi được đặt ở gần thị kính của kính hiển vi, giao diện thao tác sẽ xuất hiện. Nếu người dùng chưa đặt tiêu bản lên bàn kính thì sẽ xuất hiện dòng thông báo “Chưa bỏ tiêu bản vào kính hiển vi”.
- Lựa chọn độ phóng đại ở bảng điều khiển kính hiển vi: kính hiển vi sẽ có 4 độ phóng đại là 4x, 10x, 40x và 100x. Khi chuyển đổi từ vật kính 4x sang 100x cần chuyển từ 4x sang 10x rồi tới 40x và cuối cùng là 100x. Không thể thực hiện đổi vật kính nhảy bước, ví dụ không thể chuyển trực tiếp từ 4x sang 40x hoặc ngược lại. Khi nhấn vào vật kính 4x yêu cầu ở kính hiển vi sẽ thực hiện animation quay mâm kính để vật kính 4x vuông góc với bàn kính, và

- hướng vào vật mẫu. Khi quay vật kính tới độ phóng đại 4x thì hình ảnh của vật mẫu cũng sẽ có độ phóng đại 4x và tương tự với các vật kính khác.
- Điều chỉnh ánh sáng kính hiển vi ở bản điều khiển kính hiển vi: nếu người dùng đã cảm điện và bật công tắc, khi kéo nút điều chỉnh ánh sáng (dạng thanh trượt) từ trái sang phải, đèn ở chân kính hiển vi sẽ sáng càng mạnh. Nếu không cảm điện hoặc chưa bật công tắc thì đèn sẽ không sáng và hình ảnh của vật mẫu sẽ không hiển thị được và có màu đen do thiếu ánh sáng.
 - Điều chỉnh nút sơ cấp và vi cấp: nút sơ cấp (ly độ cao, độ chính xác thấp) và nút vi cấp (ly độ thấp, độ chính xác cao) có dạng thanh trượt, khi kéo thanh trượt từ trái sang phải thì bàn kính sẽ di chuyển từ dưới lên gần với vật kính. Độ mờ của hình ảnh từ mẫu vật kính sẽ thay đổi theo tỉ lệ được cho từ trước, ví dụ: cho tỉ lệ là 0.6 thì khi kéo tới 60% của nút sơ cấp thì hình ảnh của mẫu vật sẽ rõ nhất.
 - Ốc chỉnh bàn kính: 2 thanh trượt không có tên, một thanh nằm đứng song song với cạnh trái/phải của màn hình máy tính và 1 thanh nằm ngang song song với cạnh trên/dưới của màn hình máy tính. Thanh nằm đứng di chuyển hình ảnh vật mẫu lên trên hoặc xuống dưới, tương ứng với di chuyển lên trước hoặc lùi sau của bàn kính. Thanh nằm ngang di chuyển hình ảnh mẫu vật sang trái hoặc sang phải tương ứng bàn kính sẽ di chuyển qua trái hoặc qua phải.
 - Nhỏ dầu: vì thấu kính 100x có độ phóng đại rất lớn, yêu cầu đặt sát vật mẫu, để tránh ánh sáng tán xạ vào mặt kính khi quan sát, làm giảm chất lượng hình ảnh thu được, cần nhỏ dầu để giảm tán xạ ánh sáng khi đi qua vật mẫu. Để thực hiện được thao tác nhỏ dầu, người dùng cần chỉnh vật kính về 100x, nếu không thì sẽ hiển thị cảnh báo “Chỉ sử dụng dầu soi cho vật kính 100X”, ngoài ra nếu chưa bỏ vật mẫu vào kính mà nhấn nút nhỏ dầu thì cũng hiện ra thông báo “Chưa bỏ mẫu vật vào kính hiển vi”. Khi đã bỏ vật mẫu và chỉnh kính về 100x thì thực hiện animation nhỏ dầu, trên vật mẫu sẽ có một giọt dầu, để dễ nhìn thì giọt dầu sẽ có màu xanh.
 - Lau kính: dầu soi kính chỉ sử dụng được cho vật kính 100x nên sau khi sử dụng vật kính 100x, muốn chuyển sang các vật kính khác cần lau đi dầu soi kính để tránh làm hỏng các vật kính khác. Muốn thực hiện hoạt động lau kính,

trên vật mẫu phải có dầu, nếu không có dầu thì hiển thị thông báo “Tiêu bản không có dầu, không cần sử dụng giấy lau”.

Triển khai

Giao diện frame 6 giống với giao diện của frame 5, có thể nói bài hướng dẫn của frame 5 là để người dùng hiểu rõ các thao tác để sử dụng ở frame 6.



Hình 3.29. Màn hình thực hành được thiết kế trong Godot Engine

Ý tưởng viết mã

❖ Xử lý ban đầu: khi các dữ liệu hình ảnh, các biến và các node được đưa vào scene tree. Hàm `_ready` được gọi để gọi các hàm khởi tạo và cài đặt các giá trị ban đầu cho bài thực hành. Hàm `loadScene` có chức năng ẩn giọt dầu soi kính, gán chỉ số cho nút sơ cấp và vi cấp. Hàm `loadKHZ(0)` chuyển kính hiển vi về vật kính 4x tương ứng với tham số truyền vào bằng 0. Hàm `loadChangeScene`, hiển thị nút chuyển cảnh sang màn hình kiểm tra. Hàm `loadItemList` tải dữ liệu các hình ảnh mẫu vật vào bảng lựa chọn tiêu bản sau khi mở hộp tiêu bản.

```
func _ready():
    loadScene()
    loadKHZ(0)
    loadChangeScene()
    loadItemList()
```

❖ Hộp đựng tiêu bản: khi click vào nút mở hộp đựng tiêu bản, hàm `_on_BtnHopTieuBan_button_up` được gọi và gán biến `isOpen` bằng phủ định của biến `isOpen`, nghĩa là khi biến `isOpen` bằng `true` sẽ thành `false` và ngược lại. Hàm

loadUIListMauVat sẽ thực hiện di chuyển danh sách tiêu bản từ rìa màn hình di chuyển và hiển thị ở phía bên phải màn hình. Nếu isOpen bằng false thì sẽ di chuyển danh sách vật mẫu di chuyển ra khỏi màn hình thực hành.

```
func _on_BtnHopTieuBan_button_up():
    _hopDungTieuBan.isOpen = !_hopDungTieuBan.isOpen

func loadUIListMauVat():
    if _hopDungTieuBan.isOpen:
        _viewListMauVat.rect_position.x =
lerp(_viewListMauVat.rect_position.x, 970, 0.05)
    else:
        _viewListMauVat.rect_position.x =
lerp(_viewListMauVat.rect_position.x, 1200, 0.05)
```

❖ Lựa chọn mẫu vật: khi lựa chọn một vật mẫu trong danh sách, tương ứng với mẫu vật click vào thì hàm `_onItemList_item_selected` sẽ trả về một biến index. Biến indexVatMau sẽ được gán bằng index, biến indexVatMau là biến toàn cục, được sử dụng để tải hình ảnh từ danh sách mẫu vật. Câu lệnh `_khv.layLamKinh` gọi animation bô lam kính vào kính hiển vi. Hàm `loadKHZ(_khv.scope)` sẽ lấy hình ảnh mẫu vật tương ứng với độ phóng đại của vật kính.

```
func _onItemList_item_selected(index):
    indexVatMau = index
    _khv.layLamKinh()
    loadKHZ(_khv.scope)
```

❖ Điều chỉnh ánh sáng: Khi người dùng cảm điện biến Global.isLight sẽ được gán bằng true, hoặc khi người dùng bật công tắc thì biến `_khv.isCongTac` sẽ được gán bằng true. Hàm `loadLight` sẽ kiểm tra xem 2 biến Global.isLight và biến `_khv.isCongTac` có bằng true hay không ở câu lệnh if. Nếu điều kiện đáp ứng thì đèn sẽ được bật, độ sáng của đèn được xử lý bằng câu lệnh `_khv.valuePower = _hSliderAnhSang.value` với `_hSliderAnhSang.value` là giá trị của thanh kéo trong khoảng giá trị từ -0.5 đến 0.5. Khi giá trị bằng -0.5 thì ánh sáng yếu nhất, khi giá trị bằng 0.5 thì đạt ánh sáng mạnh nhất. Câu lệnh điều chỉnh ánh sáng của hình ảnh mẫu vật được xử lý bởi câu lệnh `_viewMauVat.material.set_shader_param("light_adjust", _hSliderAnhSang.value)`. Cuối cùng là câu lệnh `_black.visible = false`, sẽ tắt đi tấm nền màu đen che khi kính hiển vi không có ánh sáng.

```

func loadLight() -> void:
    if Global.isLight == true && _khv.isCongTac:
        _viewMauVat.material.set_shader_param("light_adjust",
_hSliderAnhSang.value)
        _khv.valuePower = _hSliderAnhSang.value
        _black.visible = false
    else:
        _viewMauVat.material.set_shader_param("light_adjust", -0.1)
        _khv.valuePower = -0.5
        _black.visible = true

```

❖ Chọn vật kính: khi chọn vật kính biến `_khv.scope` sẽ được gán vào một số nguyên từ 0 đến 3 tương ứng là: số 0 là vật kính 4x, số 1 là 10x, số 2 là 40x và số 100 là 100x. Lấy chỉ số vật kính muốn chuyển đổi trừ cho chỉ số vật kính hiện tại sau đó trị tuyệt đối nếu đáp án bằng 1 thì sẽ chuyển đổi sang vật kính muốn chọn. Khi đang ở vật kính 10x muốn chuyển sang vật kính 4x, hàm `_on_TextureButtonX4_button_up` sx được gọi, ở câu lệnh if sẽ lấy chỉ số vật kính 10x là 1 (`_khv.scope`) trừ cho chỉ số vật kính 4x là 0 sau đó trị tuyệt đối, kết quả là $|1 - 0| = 1$. Thực hiện chuyển đổi sang vật kính 4x ở câu lệnh `loadKHV(0)`. Khi từ vật kính 10x chuyển sang vật kính 100x, hàm `_on_TextureButtonX100_button_up` được gọi, ở câu lệnh if sẽ lấy chỉ số kính hiện vi 10x là 2 trừ đi chỉ số của vật kính 100x là 3, kết quả $|1 - 3| = 2$. Điều kiện if bị sai, không thực hiện chuyển đổi vật kính từ 10x sang 100x. Xử lý điều kiện ở vật kính 40x sẽ khác các nút chọn vật kính khác, nếu có dầu lau kính ở mẫu vật thì sẽ hiển thị thông báo “Lau dầu trước, để tránh làm hỏng các vật kính khác”, nếu không có dầu soi kính thì thực hiện chuyển đổi sang vật kính 40x.

```

# Nút chọn vật kính 4X
func _on_TextureButtonX4_button_up():
    if abs(_khv.scope - 0) == 1:
        loadKHV(0)

# Nút chọn vật kính 10X
func _on_TextureButtonX10_button_up():
    if abs(_khv.scope - 1) == 1:
        loadKHV(1)

# Nút chọn vật kính 40X
func _on_TextureButtonX40_button_up():
    if abs(_khv.scope - 2) == 1:
        if _khv.isNhoDau:
            _alert.setText("Lau dầu trước, để tránh làm hỏng các vật
kính khác")

```

```

        _alert.showAlert(2)
    else:
        loadKHV(2)

# Nút chọn vật kính 100X
func _on_TextureButtonX100_button_up():
    if abs(_khv.scope - 3) == 1:
        loadKHV(3)

```

❖ Dầu soi kính: khi nhấn vào nút nhỏ dầu thì hàm `_on_BtnDauSoiKinh_button_up()` được gọi. Thực hiện kiểm tra biến `isAnimation = false` thì có nghĩa là không có hoạt hình nào đang chạy, biến `is100X` có nghĩa là vật kính 100x đang được chọn, cuối cùng là biến `_khv.isLamKinh` được sử dụng để kiểm tra xem vật kính đã được đặt vào kính hiển vi chưa. Nếu điều kiện được thỏa mãn thì sẽ chạy animation “NhoDau”. Nếu không thì hiển thị thông báo, nếu không có mẫu vật thì sẽ hiển thị thông báo “Chưa bỏ mẫu vật vào kính hiển vi”, nếu không phải là vật kính 100x thì hiển thị thông báo “Chỉ sử dụng dầu soi cho vật kính 100X”.

```

func _on_BtnDauSoiKinh_button_up():
    if not isAnimation and is100X and _khv.isLamKinh:
        _animation.play("NhoDau")
    else:
        if not _khv.isLamKinh:
            _alert.setText("Chưa bỏ mẫu vật vào kính hiển vi")
            _alert.showAlert()
        elif not is100X:
            _alert.setText("Chỉ sử dụng dầu soi cho vật kính 100X")
            _alert.showAlert()

```

❖ Giấy lau kính: `_on_BtnGiayLauKinh_button_up()` khi nhấn vào nút lau ống kính, điều kiện của để chạy animation “LauDau” cần đảm bảo các điều kiện không có animation nào đang chạy, đang ở vật kính 100x, trên bàn kính có mẫu vật, và vật mẫu đã được nhô dầu. Nếu điều kiện bị sai thì sẽ hiển thị báo khi dầu soi kính chưa được nhô “Tiêu bản không có dầu, không cần sử dụng giấy lau”

```

func _on_BtnGiayLauKinh_button_up():
    if not isAnimation and is100X and _khv.isLamKinh and
    _khv.isNhoDau:
        _animation.play("LauDau")
    else:
        if not _khv.isNhoDau:
            _alert.setText("Tiêu bản không có dầu, không cần sử dụng
giấy lau")
            _alert.showAlert(2)

```

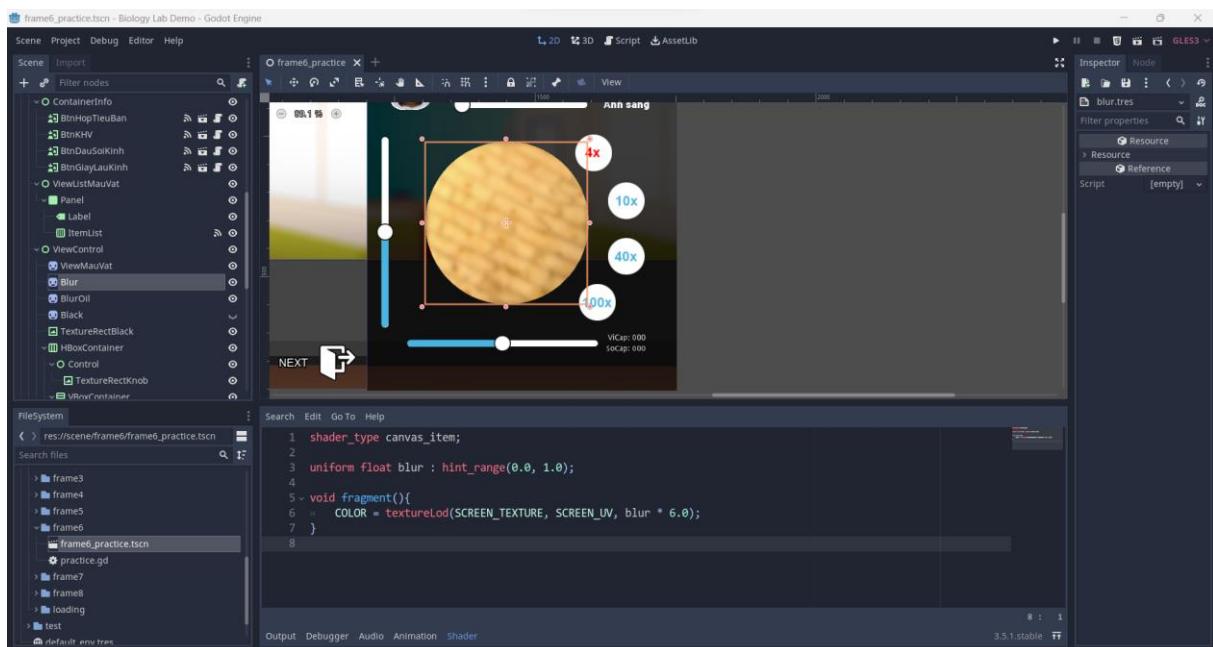
- ❖ Nút chỉnh sơ cấp và vi cấp: hàm loadScopeValue lấy dữ liệu độ lớn của nút sơ cấp và vi cấp ở biến data đưa vào hàm loadScopeSoCap và loadScopeViCap để thực hiện xử lý làm mờ cho hình ảnh mẫu vật.

```

func loadScopeValue(s: String):
    loadScopeSoCap(data.CONFIG_SCOPE[indexVatMau][s]["socap"])
    loadScopeViCap(data.CONFIG_SCOPE[indexVatMau][s]["vicap"])

```

Để xử lý làm mờ hình ảnh của mẫu vật, dùng kỹ thuật shader (trình đồ bóng), xử lý bằng bộ xử lý đồ họa GPU (Graphics Processing Unit). Godot hỗ trợ xử lý shader ở màn hình Shader.



Hình 3.30. Giao diện trình xử lý shader của Godot Engine

Đoạn mã xử lí shader trong Godot Engine:

```

shader_type canvas_item;

uniform float blur : hint_range(0.0, 1.0);

void fragment(){

```

```
COLOR = textureLod(SCREEN_TEXTURE, SCREEN_UV, blur * 6.0);  
}
```

Dòng đầu tiên shader_type canvas_item; chỉ định rằng đây là một shader dùng để vẽ trên một đối tượng dạng canvas_item trong Godot Engine. Các đối tượng canvas_item bao gồm các Node2D, Control và một số loại khác.

Dòng tiếp theo uniform float blur : hint_range(0.0, 1.0); định nghĩa một biến uniform tên là "blur" có kiểu dữ liệu float. Uniforms là các biến được gửi từ bên ngoài vào shader và giá trị của chúng có thể được điều chỉnh từ script hoặc qua giao diện của trình chỉnh sửa shader trong Godot. Trong trường hợp này, "blur" là một tham số cho mức độ mờ được áp dụng lên hình ảnh.

Dòng void fragment() khai báo hàm "fragment". Hàm này sẽ được gọi cho mỗi pixel trên màn hình hoặc canvas_item mà shader được áp dụng vào.

Dòng COLOR = textureLod(SCREEN_TEXTURE, SCREEN_UV, blur * 6.0); gán giá trị cho biến màu "COLOR" của pixel hiện tại. Hàm textureLod được sử dụng để lấy màu từ một texture. Trong trường hợp này, SCREEN_TEXTURE là texture của màn hình hoặc canvas_item mà shader được áp dụng, SCREEN_UV là tọa độ texture tương ứng với pixel hiện tại, và blur * 6.0 là mức độ mờ được nhân với 6.0 để tăng cường mức độ mờ theo giá trị của biến "blur". Kết quả cuối cùng được gán vào biến "COLOR", làm thay đổi màu sắc của pixel đó.

Shader blur áp dụng hiệu ứng mờ lên màn hình hoặc canvas_item, với mức độ mờ được điều chỉnh thông qua biến uniform "blur".

Để có thể điều chỉnh tham số "blur" của shader sử dụng câu lệnh ở hàm _process:

```
_blur.material.set_shader_param("blur", valueSoCap + valueViCap)
```

Hàm set_shader_param từ đối tượng material, sẽ lấy tổng giá trị của biến vi cấp (valueViCap) cộng với biến sơ cấp (valueSoCap), tổng của 2 biến này bằng 1 thì sẽ hình ảnh kính hiển vi sẽ rõ nhất và khi khác 1 thì hình ảnh của kính hiển vi sẽ bị mờ.

3.3.7. Frame 7 – Màn hình kiểm tra

Chức năng

Giúp người dùng ôn tập lại kiến thức đã học với 5 câu hỏi trắc nghiệm đã được chuẩn bị từ trước. Người dùng cần hoàn thành tất cả các câu hỏi để nhận được chứng chỉ chứng nhận đã hoàn thành bài học thực hành.

Yêu cầu

❖ Giao diện:

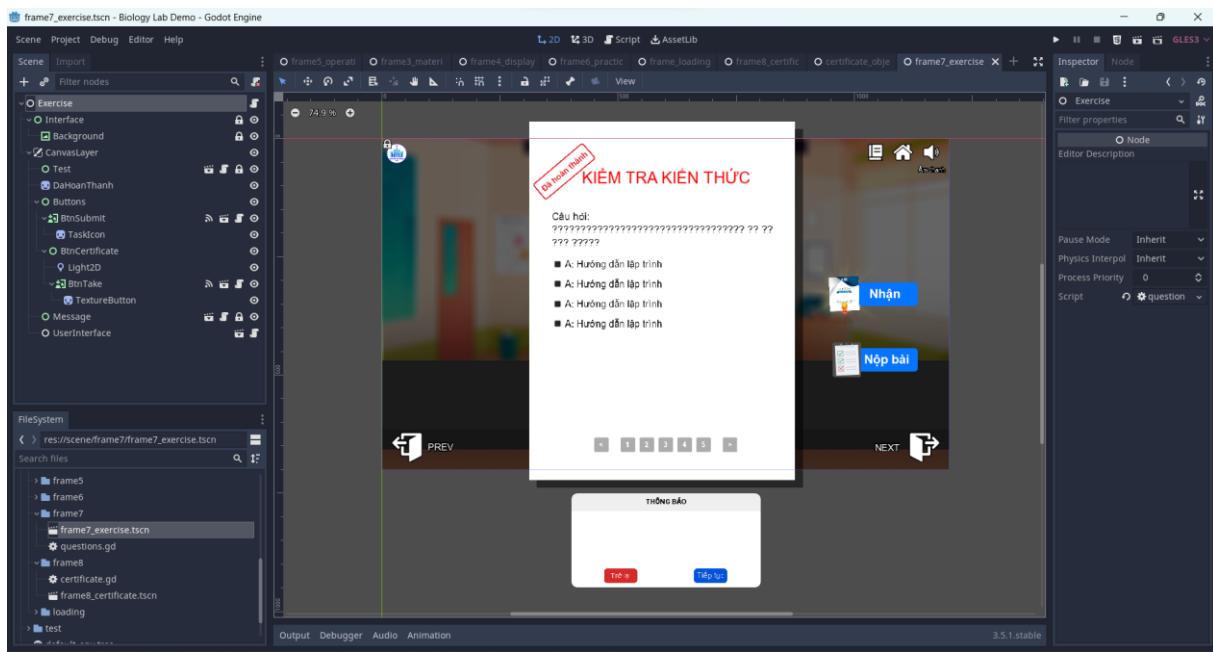
- Thiết kế giống khi đang làm một bài kiểm tra trên lớp, các câu hỏi cần có các checkbox (hộp chọn) nằm kế bên để người dùng chọn vào.
- Bên phải có nút nộp bài. Nếu làm đúng sẽ hiển thị một tin nhắn thông báo hoàn thành kèm với số điểm và nút nhận chứng chỉ sẽ hiện ra để người dùng nhấn vào và nhận chứng chỉ. Nếu làm sai thì tin nhắn vẫn hiển thị để thông báo điểm và gợi ý người dùng làm lại.
- Ngoài ra, bên dưới bài làm sẽ có các nút điều hướng tới các câu hỏi tương ứng. Khi tích chọn thì câu hỏi được chọn sẽ có màu khác với câu khác.

❖ Đặc tả hệ thống:

- Chọn ngẫu nhiên 5 câu hỏi trong bộ đề đã chuẩn bị từ trước. Đảo trộn cả câu hỏi và đáp án.
- Khi người dùng làm đúng tất cả các câu hỏi thì sẽ hiển thị dấu mốc đỏ là “Đã hoàn thành” ở góc trên bên trái để người dùng biết là đã hoàn thành.
- Khi nhấn vào nút nộp bài sẽ có một hộp thoại thông báo sẽ hiện ra để sinh viên biết được số điểm của mình. Sẽ có 2 lựa chọn, nếu chọn “Trở lại” sẽ trở lại bài kiểm tra để sinh viên xem lại các kết quả đã chọn. Nếu làm sai khi nhấn nút tiếp tục sẽ quay trở lại màn hình trắc nghiệm để làm lại bài trắc nghiệm khác. Nếu làm đúng thì khi nhấn nút tiếp tục sẽ dẫn người dùng qua màn hình nhận chứng chỉ.

Triển khai

Màn hình kiểm tra được thiết kế ngay trên Godot bởi vì đây là màn hình khá đơn giản, số lượng hình ảnh cần tạo và sử dụng khá ít. Hầu hết là các đoạn text và các button nên Godot đảm bảo có thể thực hiện được.



Hình 3.31. Màn hình kiểm tra được thiết kế trên Godot Engine

Kịch bản

Đầu để thực hiện màn hình bài kiểm tra cần có cách tổ chức các câu hỏi và các câu trả lời để khi truy cập, hệ thống có thể hiểu và biết đâu là câu trả lời đúng. Sử dụng cách lưu trữ dữ liệu dạng từ điển (Dictionary giống với Json). Xem một câu hỏi như 1 đối tượng, trong mỗi đối tượng sẽ có 4 câu trả lời. Để biết câu trả lời đó là gì sẽ cần 1 biến dạng text để lưu trữ và 1 biến kiểu true/false để biết câu trả lời đó đúng hay sai. Kết quả ta sẽ có một danh sách các câu hỏi nằm trong script có tên data_frame7.gd, trong đó sẽ có một hằng số DATA_QUESTION lưu trữ danh sách các câu hỏi.

```
const DATA_QUESTION = [
  {
    "question": "Dầu soi kính là gì và tại sao cần sử dụng dầu soi kính?",  

    "answers": [
      {
        "content": "Dầu soi kính là dung dịch trong suốt, được sử dụng và làm bóng các vật kính.",  

        "correct": false
      },
      {
        "content": "Dầu soi kính là môi trường để giúp mầm vật bám dính vào lâm kính.",  

        "correct": false
      },
      {
        "content": "Dầu soi kính là chất bôi trơn được sử dụng để giúp mâm vật kính có thể xoay một cách dễ dàng.",  

        "correct": true
      }
    ]
  }
]
```

```

        "correct": false
    },
    {
        "content": "Đầu soi kính là chất lỏng đặc biệt chỉ sử
được dùng khi cần soi mẫu vật ở vật kính 100x nhằm giúp ánh sáng tập
trung vào mẫu, và nhờ vậy tăng độ phân giải và độ rõ ảnh của vật.",
        "correct": true
    }
],
},
}

```

Đoạn mã cốt lõi để thực hiện việc xáo trộn các câu hỏi và câu trả lời đã chuẩn bị sẵn ở trên.

```

func randomAnswer():
    randomize()
    questions.shuffle()
    for i in questions:
        randomize()
        i[“answers”].shuffle()

```

Đầu tiên, sử dụng hàm randomize() để khởi tạo một trạng thái ngẫu nhiên cho việc sinh số ngẫu nhiên. Điều này đảm bảo rằng mỗi lần chạy hàm, các số ngẫu nhiên sẽ có một giá trị khác nhau. Áp dụng phương thức shuffle() lên mảng questions để xáo trộn các phần tử trong mảng theo một thứ tự ngẫu nhiên. Kết quả là các câu hỏi sẽ được sắp xếp lại một cách ngẫu nhiên. Tiếp theo, sử dụng vòng lặp for để lặp qua từng câu hỏi trong mảng questions. Trong mỗi lần lặp, sử dụng lại hàm randomize() để tạo một trạng thái ngẫu nhiên mới cho việc sinh số ngẫu nhiên. Điều này đảm bảo rằng mỗi lần xáo trộn câu trả lời của một câu hỏi, chúng sẽ được sắp xếp lại theo một thứ tự ngẫu nhiên. Cuối cùng, sử dụng phương thức shuffle() trên thuộc tính “answers” của mỗi câu hỏi để xáo trộn các câu trả lời trong mảng “answers”. Kết quả là các câu trả lời của mỗi câu hỏi sẽ được sắp xếp lại một cách ngẫu nhiên.

Để xác định các đáp án đúng trong bộ đề, chúng ta sẽ duyệt qua tất cả các câu hỏi sau đó, tìm ra đâu là đáp án đúng và lưu trữ đáp án đó lại dưới dạng chỉ số (index) ở đây là j, nếu a[j][“correct”] bằng true thì vị trí j sẽ được lưu vào biến correctAnswer.

```

func createArrayCorrectAnswer():
    for i in size:
        var a = questions[i][“answers”]
        for j in a.size():

```

```

if a[j][“correct”]:
    correctAnswer.append(j)

```

Khi người dùng đã chọn đủ tất cả các đáp án thì nút nộp bài sẽ hiện ra bên gốc phải màn hình. Hàm showSubmitBtn sẽ liên tục kiểm tra và đếm xem số câu hỏi lựa chọn đã đủ chưa. Nếu biến dem có giá trị bằng với giá trị của biến size (số câu hỏi) thì có nghĩa là người dùng đã chọn hết tất cả câu hỏi.

```

func showSubmitBtn():
    var choiceAnswer = _test.getChoiceAnswers()
    var dem = 0
    for i in size:
        if choiceAnswer[i] != -1:
            dem += 1
    if dem == size:
        _btnSubmit.visible = true
    else:
        _btnSubmit.visible = false
    dem = 0

```

Khi người dùng nhấn vào nút “nộp bài”, hệ thống sẽ tự động tính điểm và đưa ra kết quả tiếp theo.

```

func checkSubmit():
    _test._answers.disabledCheckbox(true)
    var choiceAnswer = _test.getChoiceAnswers()
    for i in questions.size():
        if correctAnswer[i] == choiceAnswer[i]:
            count += 1
    if count == size:
        _message.setMessage(self, “closeFinalMessage”, “““Bạn đã trả lời chính xác tất cả các câu hỏi.
Nhấn tiếp tục để nhận bằng chứng nhận”““)
        _message.show()
        _btnCertificate.visible = true
        _finalSprite.visible = true
        profile[“ChinhXac”] = true
        final = true
    else:
        _message.setMessage(self, “closeMessage”, “““Bạn làm đúng được: ”““ + str(count) + ”““/5 câu.
Nhấn tiếp tục để làm lại.”““)
        _message.show()
    count = 0

```

Hàm checkSubmit bắt đầu bằng việc vô hiệu hóa các checkbox để ngăn người dùng thay đổi câu trả lời sau khi đã nộp. Tiếp theo, chúng ta lấy danh sách câu trả lời

được chọn từ đối tượng _test và gán vào biến choiceAnswer. Sau đó, chúng ta sử dụng một vòng lặp để kiểm tra từng câu hỏi trong danh sách câu hỏi. Nếu câu trả lời được chọn cho mỗi câu hỏi khớp với câu trả lời đúng tương ứng, biến count sẽ tăng lên.

Tiếp theo, chúng ta kiểm tra nếu count bằng với kích thước của danh sách câu hỏi (biến size). Nếu điều này xảy ra, tức là tất cả câu trả lời đều chính xác, chúng ta hiển thị một thông báo thông báo cho người dùng rằng họ đã trả lời đúng tất cả câu hỏi. Sau đó, hiển thị nút chứng chỉ và hình ảnh cuối cùng, và cập nhật trạng thái của biến final và profile.

Nếu không, tức là có câu trả lời không chính xác, chúng ta hiển thị một thông báo cho người dùng cho biết số câu trả lời đúng và khuyến nghị họ làm lại. Sau đó, chúng ta đặt count về 0 để chuẩn bị cho lần nộp kế tiếp.

3.3.8. Frame 8 – Màn hình chứng chỉ

Chức năng

Khi người dùng hoàn thành kết tất cả các bài học và hoàn thành bài kiểm tra cuối cùng thì sẽ truy cập được vào màn hình chứng chỉ. Ở màn hình chứng chỉ sẽ bao gồm tên người dùng (người học), mssv và ngày hoàn thành bài học (ngày hệ thống).

Yêu cầu

❖ Giao diện:

- Giao diện bao gồm một nút tải cho phép người dùng tải hình ảnh của chứng chỉ có chứa tên của người học và ngày – tháng – năm hoàn thành chứng chỉ.
- Giao diện có đầy đủ các nút trang chủ, nút âm thanh và logo trường Đại học Nha Trang.

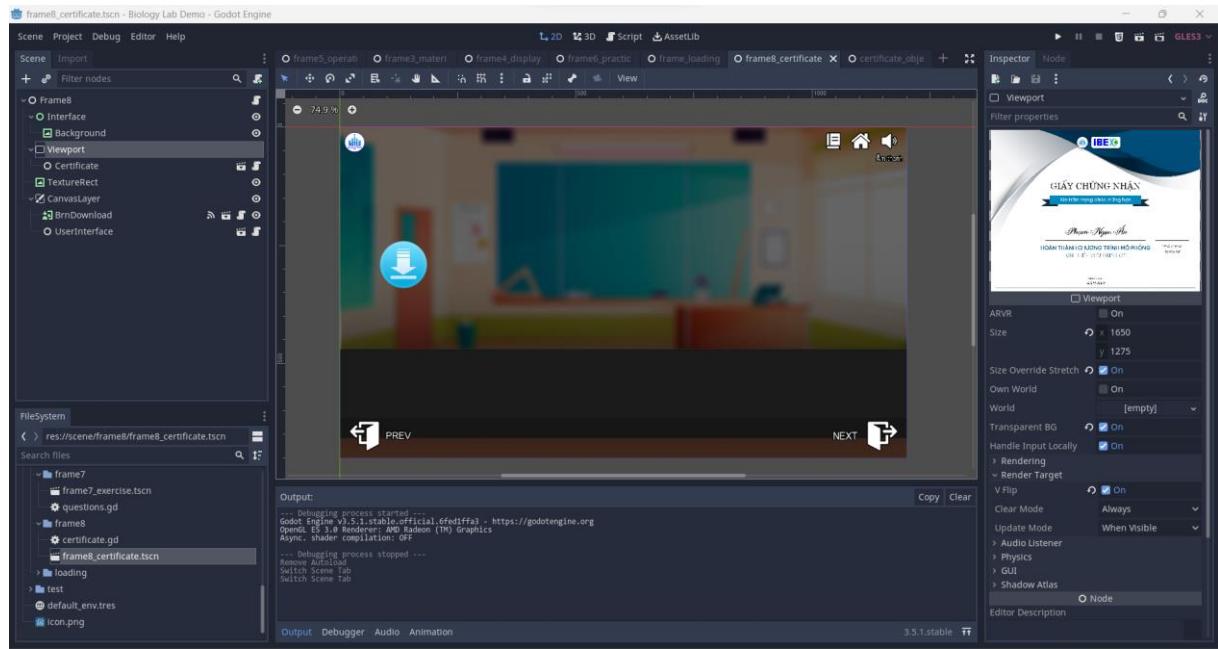
❖ Đặc tả hệ thống:

- Cho phép người dùng tải chứng chỉ từ trình duyệt về bộ nhớ máy tính.
- Chứng chỉ tải về phải có tên là cert và có định dạng là PNG.

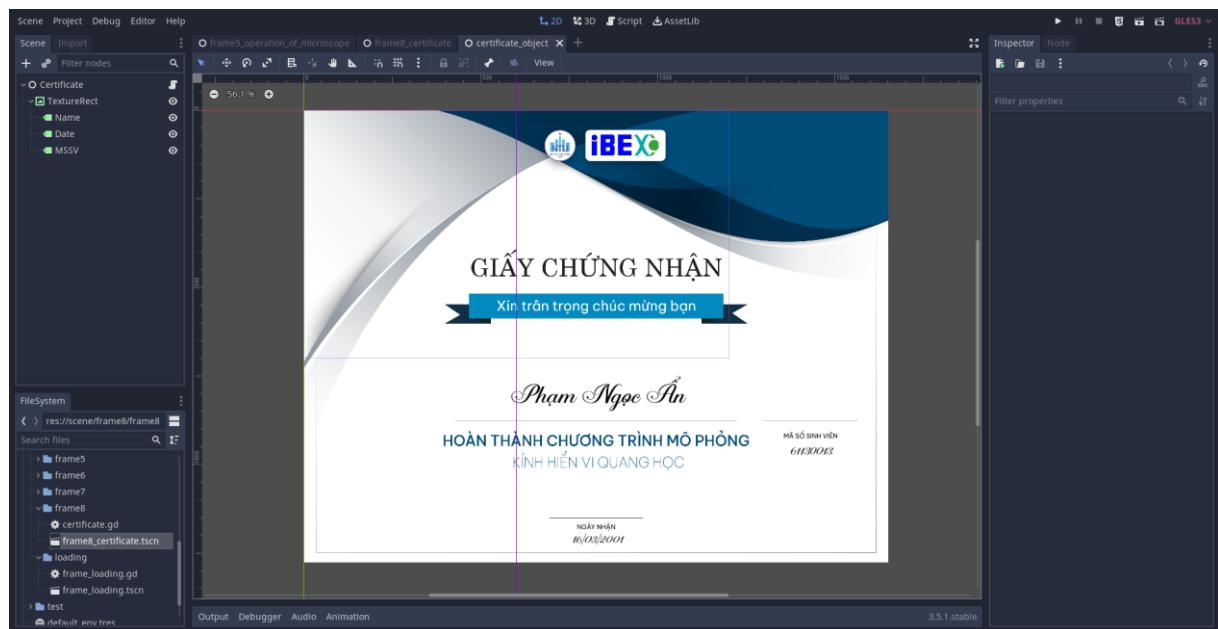
Triển khai

Màn hình chứng chỉ được xây dựng giao diện ngay trên Godot Engine. Đây là một giao diện đơn giản, chỉ bao gồm hình ảnh chứng chỉ và nút tải (download). Để chứng chỉ in ra có đầy đủ thông tin của sinh viên bao gồm họ tên, MSSV, và ngày hoàn thành khóa học. Chứng chỉ sẽ không có các thông tin vừa kể trên. Các thông tin

đó sẽ được thêm vào sau khi người dùng nhập các thông tin cần thiết ở màn hình login.

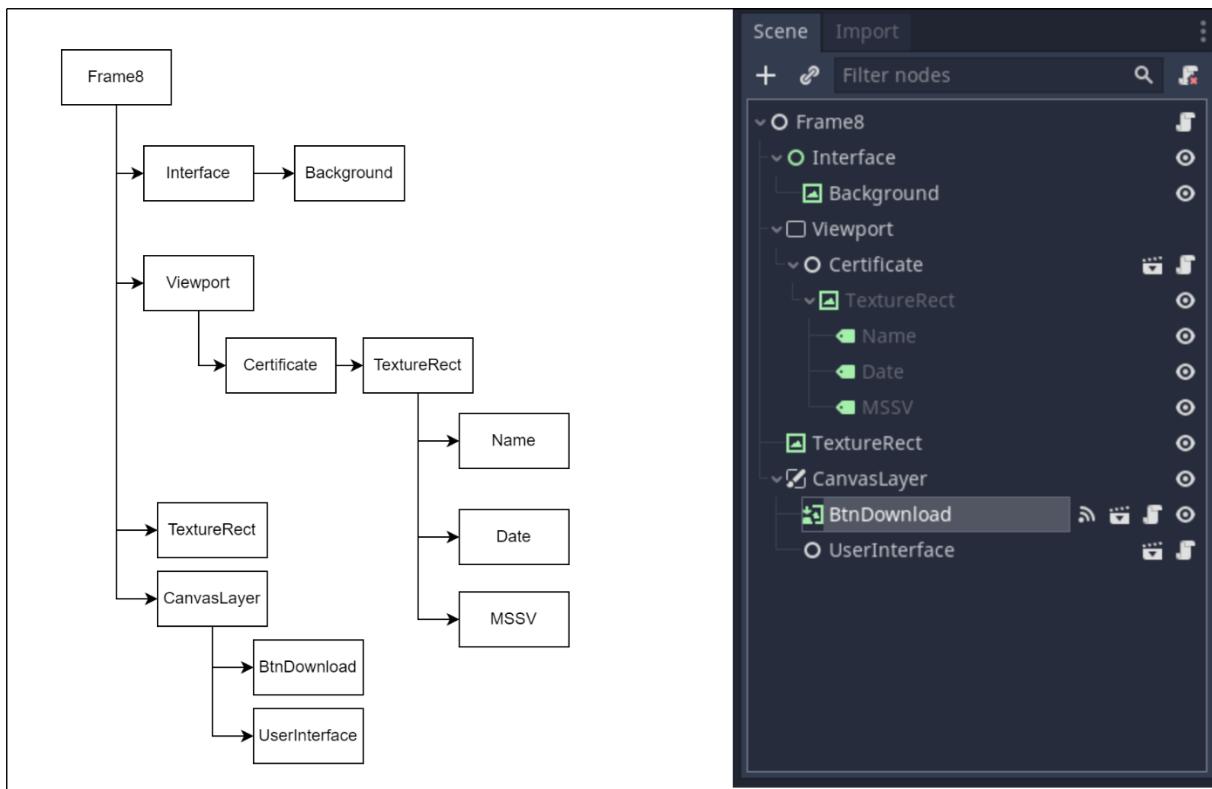


Hình 3.32. Màn hình chứng chỉ được thiết kế trong Godot Engine



Hình 3.33. Giấy chứng chỉ được thiết kế trong Godot Engine

Sơ đồ cấu trúc cây của màn hình chứng chỉ



Hình 3.34. Sơ đồ cấu trúc cây của màn hình chứng chỉ

Qua sơ đồ cấu trúc cây, cần lưu ý các đối tượng nút chính sau đây:

- Viewport: đối tượng kết xuất hình ảnh chứng chỉ có đầy đủ thông tin sinh viên được nhập ở màn hình đăng nhập: Tên sinh viên, MSSV, ngày hoàn thành chứng chỉ.
- Certificate: đối tượng chứng chỉ, bao gồm các nút con (child node): Name – Tên sinh viên, Date – Ngày hoàn thành chứng chỉ, MSSV – Mã số sinh viên. Thông tin trên các node con thay đổi thì kết quả kết xuất (render) sẽ khác. Certificate được xử lý bởi kịch bản certificate_object.gd. Đây là kịch bản xử lý lấy thông tin sinh viên lưu ở toàn cục, xử lý và hiển thị lên màn hình.
- TextureRect (cùng cấp với Viewport): xuất hình ảnh hiển thị của chứng chỉ lên màn hình.
- BtnDownload: nút tải, bắt sự kiện nhấn nút sẽ tải hình ảnh về máy tính.

Kịch bản

Certificate.gb xử lý trích xuất hình ảnh từ _viewport2D để lưu hình ảnh từ trình duyệt về máy tính.

extends Node

```
# _ui - Giao diện UI của người dùng (Trang chủ, âm thanh, logo)
onready var _ui = $CanvasLayer/UserInterface
```

```

# _texture - chứa hình ảnh chứng chỉ có đầy đủ
# thông tin của sinh viên khi được kết xuất từ viewport
onready var _texture = $TextureRect
# _viewport2D - Scene kết xuất ra chứng chỉ
onready var _viewport2D = $Viewport

func _ready():
    loadUI()

# Load giao diện người dùng
func loadUI():
    _ui.currentScene = self
    _ui.visibleBtnNextScene(false)
    _ui.visibleBtnPrevScene(false)

func _process(_delta):
    # Lấy hình ảnh được kết xuất từ _viewport2D hiển thị
    # lên giao diện
    _texture.texture = _viewport2D.get_texture()

# Bắt tín hiệu khi người dùng nhấn nút tải
func _on.BtnDownload_button_down():
    # Lấy dữ liệu hình ảnh từ _viewport2D lưu vào đối tượng image
    var image: Image = _viewport2D.get_texture().get_data()
    # Định nghĩa hình ảnh được tải xuống máy tính
    JavaScript.download_buffer(image.save_png_to_buffer(),
    "cert.png", "image/png")

```

Certificate_object.gd lấy thông tin từ biến toàn cục và hiển thị lên giao diện, định dạng lại các tên, mssv và ngày hoàn thành.

```

extends Node

# _date ánh xạ tới nút label hiển thị ngày hoàn thành
onready var _date = $TextureRect/Date
# _name ánh xạ tới nút label hiển thị tên sinh viên hoàn thành
onready var _name = $TextureRect/Name
# _mssv ánh xạ tới nút label hiển thị mssv hoàn thành
onready var _mssv = $TextureRect/MSSV

# Tạo biến date lưu dữ liệu ngày
var date: Dictionary
# Tạo biến userName lấy tên sinh viên từ biến toàn cục userName
var userName: String = Global.userName
# Tạo biến mssv lấy mssv sinh viên từ biến toàn cục mssv
var mssv: String = Global.mssv

```

```

func _ready():
    date = OS.get_date()
    _date.text = formatDay()
    _name.text = formatUserName(userName)
    _mssv.text = mssv

# Hàm định dạng lại tên sinh viên
# VD: pHạm nGỌC ẨN => Phạm Ngọc Ẩn
func formatUserName(value: String) -> String:
    var s = ""
    s = value.to_lower()
    s = s.capitalize()
    return s

# Hàm định dạng hiển thị thời gian
# VD: 11/06/2023
func formatDay() -> String:
    var s = ""
    if date.day < 10:
        s += "0" + str(date.day)
    else:
        s += str(date.day)

    if date.month < 10:
        s += "/0" + str(date.month)
    else:
        s += "/" + str(date.month)

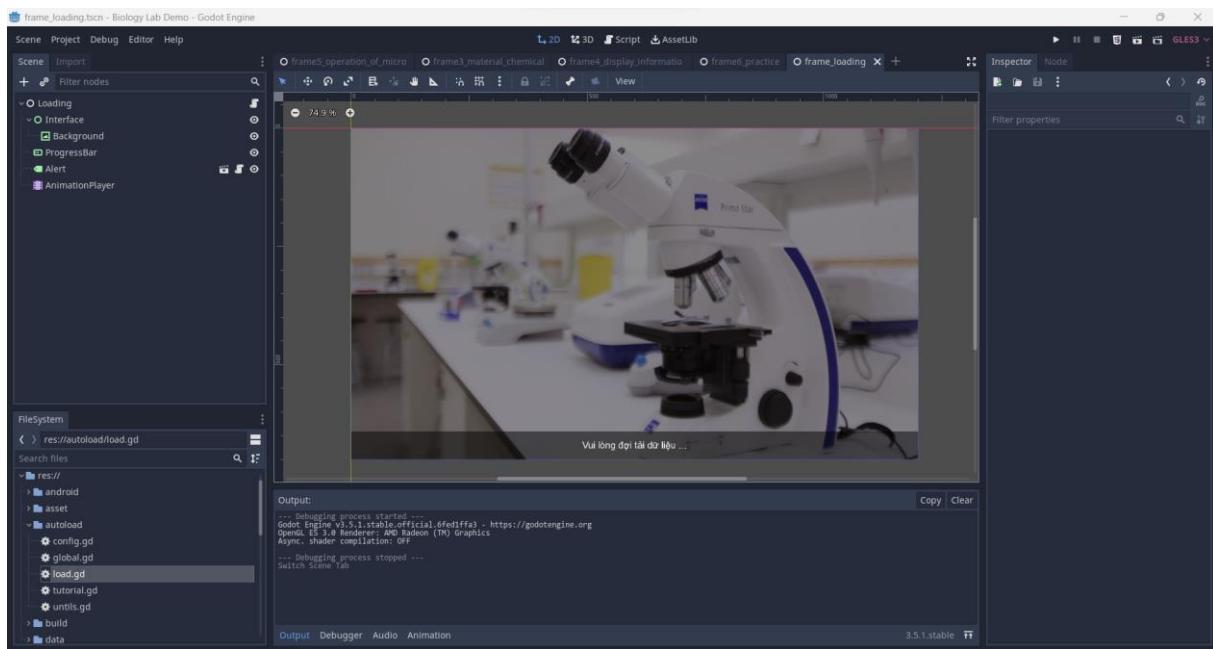
    s += "/" + str(date.year)
    return s

```

3.3.9. Frame Loading – Màn hình tải

Khi chơi game hay sử dụng các ứng dụng có dung lượng lớn, hay đơn giản hơn là duyệt web thì hầu hết người chơi đều quen thuộc với màn hình tải (loading). Ứng dụng có thể cần tải các tài nguyên, như hình ảnh, âm thanh, mô hình 3D, v.v. từ ổ cứng hoặc từ Internet. Quá trình tải dữ liệu có thể mất thời gian, và màn hình tải ứng dụng cung cấp các thông tin như đã tải được bao nhiêu phần trăm (%), thời gian tải game tốn bao nhiêu thời gian,... để người dùng biết rằng ứng dụng đang trong quá trình tải dữ liệu.

Bài mô phỏng thực hành ảo cũng có màn hình tải, màn hình tải sẽ chồng lên các màn hình khác, đợi cho màn hình bên dưới tải hết tất cả tài nguyên thì sẽ tự động biến mất và nhường lại khung hình cho màn hình vừa tải xong.



Hình 3.35. Màn hình loading được thiết kế trong Godot Engine

Kịch bản

Chạy một script (kịch bản) ngầm, script này sẽ chạy xuyên suốt các màn hình, khi thực hiện chuyển cảnh, tệp lệnh này sẽ được gọi, màn hình tải sẽ hiện lên và che đi màn hình cũ, màn hình cũ sẽ được giải phóng (xóa đi) và tải màn hình tiếp theo vào bộ nhớ, sau khi tải đầy đủ sẽ thêm màn hình mới vào nút gốc (root) và giải phóng màn hình tải. Trên màn hình tải sẽ có đối tượng ProgressBar (thanh tải), khi thanh tải đầy sẽ tải màn hình mới. Kịch bản sẽ có một hàm load_scene(current_scene, next_scene) với current_scene là màn hình hiện tại, và next_scene là màn hình tiếp theo cần tải.

```
extends Node

onready var loading_scene =
preload("res://scene/loading/frame_loading.tscn")

func load_scene(current_scene, next_scene):
    # Thêm màn hình loading vào gốc của cây scene
    var loading_scene_instance = loading_scene.instance()
    get_tree().get_root().call_deferred("add_child",
loading_scene_instance)

    # Tìm scene cần tải
    var loader = ResourceLoader.load_interactive(next_scene)

    if loader == null:
        # Xử lý lỗi
        print("Đã xảy ra lỗi khi lấy scene")
        return
```

```

current_scene.queue_free()
# Tạo độ trễ nhỏ để màn hình loading hiển thị.
yield(get_tree().create_timer(1.0),"timeout")

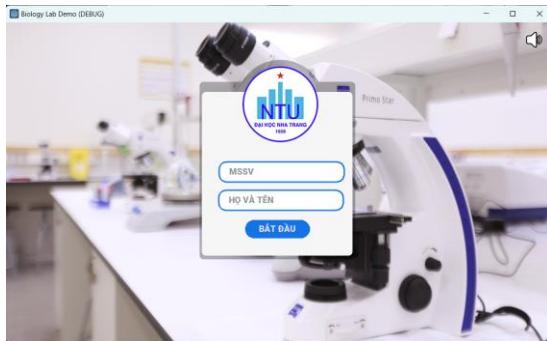
# Tải next_scene sử dụng hàm poll()
# Vì hàm poll() tải dữ liệu theo từng phần nên chúng ta cần đặt
trong vòng lặp
while true:
    var error = loader.poll()
    # Khi nhận được một phần dữ liệu
    if error == OK:
        # Cập nhật thanh tiến trình theo lượng dữ liệu đã tải
        var progress_bar =
loading_scene_instance.get_node("ProgressBar")
        progress_bar.value =
float(loader.get_stage())/loader.get_stage_count() * 100
        yield(get_tree().create_timer(0.01),"timeout")
        pass
    # Khi tất cả dữ liệu đã được tải
    elif error == ERR_FILE_EOF:
        # Tạo instance scene từ dữ liệu đã tải
        var scene = loader.get_resource().instance()
        # Thêm scene vào gốc của cây scene
        get_tree().get_root().call_deferred("add_child", scene)
        # Loại bỏ màn hình loading
        loading_scene_instance.queue_free()
        return
    else:
        # Xử lý lỗi
        print("Đã xảy ra lỗi khi tải từng phần dữ liệu")
        return

```

CHƯƠNG 4. KẾT QUẢ

4.1. MÀN HÌNH ĐĂNG NHẬP

Khi bắt đầu vào bài học, hệ thống yêu cầu người dùng nhập thông tin bao gồm mã số sinh viên và tên sinh viên (Hình 4.1). Khi nhập vào mssv mà có tồn tại ký tự đặc biệt hoặc số thì sẽ thông báo “Mã số sinh viên chỉ được chứa chữ số” (Hình 4.2), hoặc khi bỏ trống tên sinh viên thì sẽ thông báo “Họ tên sinh viên không được bỏ trống”.



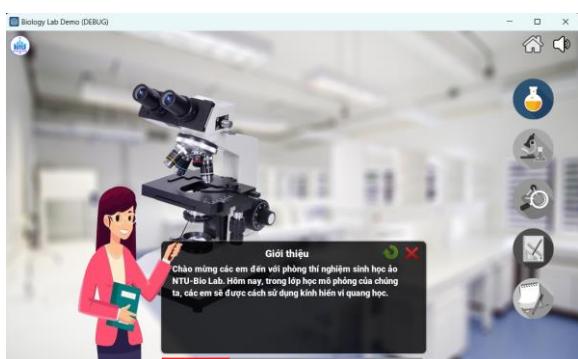
Hình 4.1. Giao diện đăng nhập



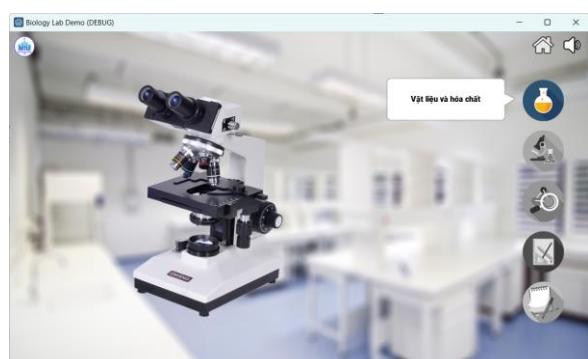
Hình 4.2. Giao diện thông báo khi nhập sai thông tin mã số sinh viên

4.2. MÀN HÌNH CHÍNH

Sau khi nhập thông tin thành công ở màn hình đăng nhập, người dùng được đưa vào màn hình chính. Khi vào màn hình chính sẽ có một dialog xuất hiện và xin chào, giới thiệu mục tiêu của bài thực hành (Hình 4.3). Khi người dùng di chuyển chuột đi qua các bài học các thẻ chứa tên của bài học sẽ hiện ra (Hình 4.4). Để bắt đầu bài học người dùng cần nhấn vào icon của bài học.



Hình 4.3. Dialog giới thiệu bài thực hành kính hiển vi



Hình 4.4. Thẻ tên của bài học vật liệu và hóa chất

4.3. MÀN HÌNH VẬT LIỆU VÀ HÓA CHẤT

Khi bắt đầu bài học về vật liệu và hóa chất trong phòng thí nghiệm, dialog xuất hiện giới thiệu nội dung bài học (Hình 4.5). Sau khi xem xong người dùng nhấn nút đóng dialog để đến với màn hình vật liệu và hóa chất (Hình 4.6). Người dùng nhấn vào

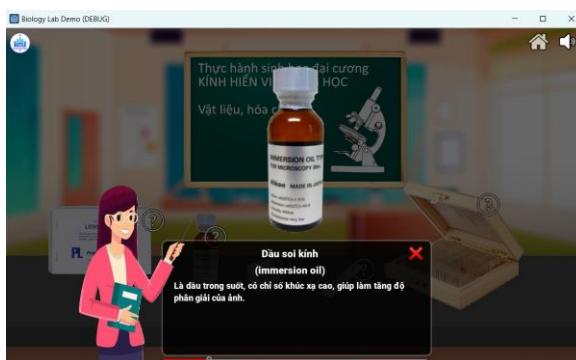
các nút thông tin trên màn hình để xem chi tiết chức năng và hình ảnh của từng loại vật liệu và dụng cụ (Hình 4.7). Sau khi xem hết tất cả các loại vật liệu, trên màn hình sẽ hiển thị thông báo “Bạn đã tìm hiểu xong các hóa chất và vật liệu trong phòng thí nghiệm”, đồng thời 2 nút chuyên màn hình về trang chủ và tới bài học tiếp theo cũng sẽ hiện ra (Hình 4.8).



Hình 4.5. Giới thiệu nội dung bài học vật liệu và hóa chất



Hình 4.6. Giao diện màn hình vật liệu và hóa chất



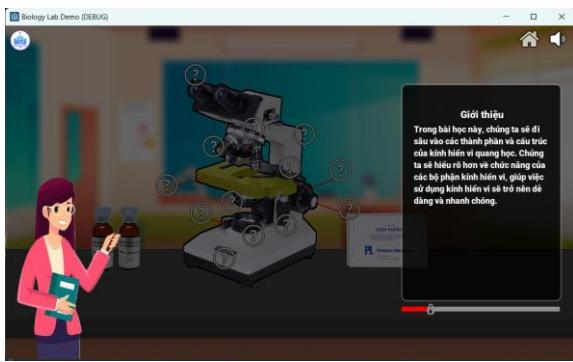
Hình 4.7. Dialog hiển thị thông tin và chức năng của vật liệu và hóa chất



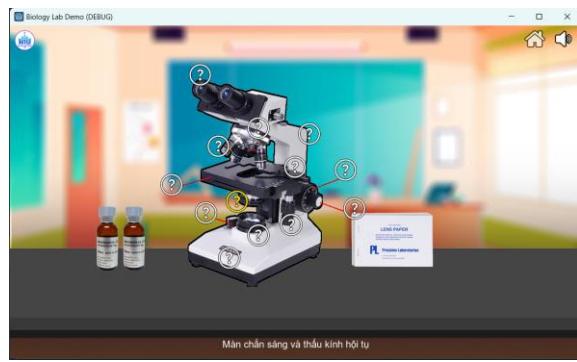
Hình 4.8. Sau khi hoàn thành bài học vật liệu và hóa chất

4.4. MÀN HÌNH CẤU TẠO KÍNH HIỂN VI

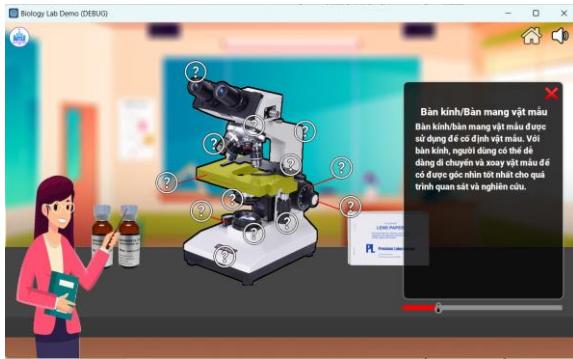
Khi bắt đầu vào bài học cấu tạo kính hiển vi, chương trình sẽ hiển thị dialog giới thiệu cho người dùng chức năng và nội dung của bài học (Hình 4.9). Người dùng cần nhấn vào nút close để đóng dialog, màn hình bài học sẽ xuất hiện (Hình 4.10). Ở giao diện bài học người dùng click vào các dấu chấm hỏi để xem chi tiết các bộ phận của kính hiển vi (Hình 4.11). Khi nghe hết đoạn âm thanh của người hướng dẫn thì dấu chấm hỏi sẽ chuyển thành dấu tích xanh. Dialog thông báo người dùng đã học xong bài học xuất hiện khi tất cả các dấu chấm hỏi chuyển thành dấu tích xanh (Hình 4.12).



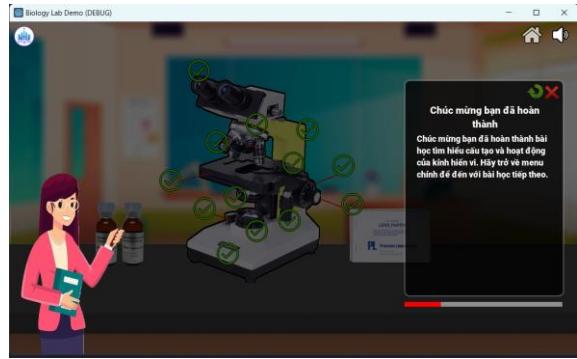
Hình 4.9. Giao diện giới thiệu bài học
cấu tạo kính hiển vi



Hình 4.10. Giao diện bài thực hành cấu
tạo kính hiển vi



Hình 4.11. Giao diện chi tiết của một bộ
phần trong kính hiển vi



Hình 4.12. Giao diện thông báo hoàn
thành bài học

4.5. MÀN HÌNH HƯỚNG DẪN THỰC HÀNH

Khi bắt đầu vào màn hình hướng dẫn thực hành, dialog giới thiệu bài học được tự động xuất hiện để giới thiệu tóm tắt nội dung sẽ được học (Hình 4.13). Sau đó, chương trình gợi ý người dùng click vào nút close trên dialog để hiển thị dialog giới thiệu bước 1 – chuẩn bị kính (Hình 4.14).



Hình 4.13. Giao diện giới thiệu của bài
hướng dẫn thực hành



Hình 4.14. Giao diện giới thiệu bước 1
của màn hình hướng dẫn thực hành

Sau khi xem xong nội dung của bước 1, và close dialog thì bong bóng chat sẽ xuất hiện, gợi ý người dùng cắm điện cho kính hiển vi (Hình 4.15). Sau khi người dùng cắm điện, bong bóng chat sẽ gợi ý người dùng bật công tắc điện (Hình 4.16).

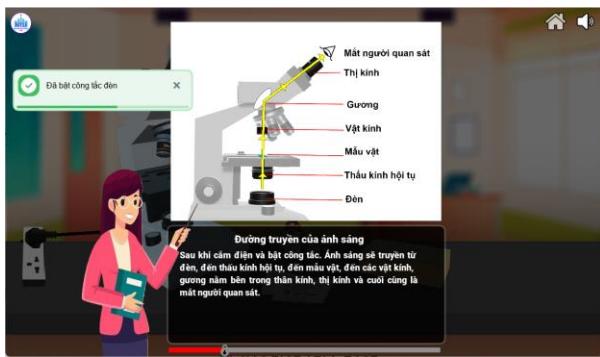


Hình 4.15. Giao diện yêu cầu cắm điện của bài hướng dẫn thực hành

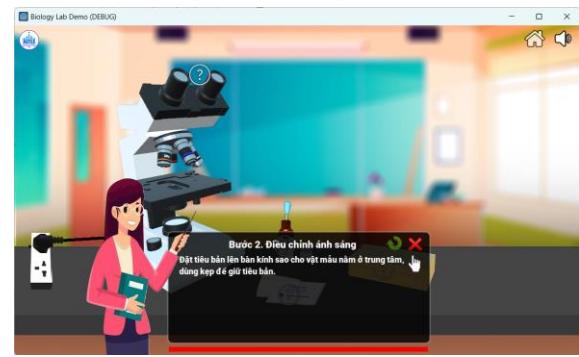


Hình 4.16. Giao diện yêu cầu bật công tắc điện của bài hướng dẫn thực hành

Kết thúc nội dung hướng dẫn bước 1 bằng một dialog chưa nội dung là đường truyền của ánh sáng (Hình 4.17). Sau khi xem xong chương trình sẽ đưa người dùng đến với bước 2 của bài hướng dẫn thực hành kính hiển vi (Hình 4.18).

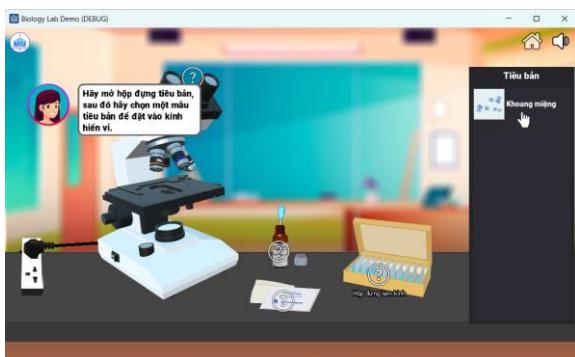


Hình 4.17. Đường truyền ánh sáng của kính hiển vi (tài liệu hướng dẫn)

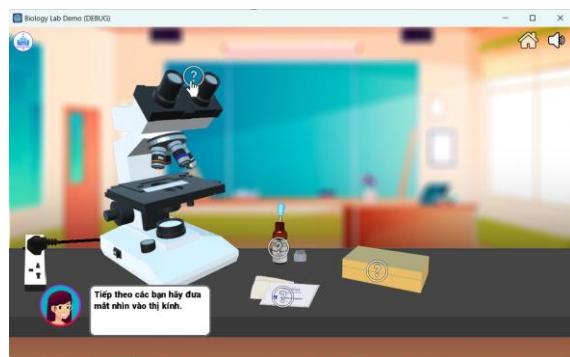


Hình 4.18. Giao diện giới thiệu bước 2 của màn hình hướng dẫn thực hành

Chương trình yêu cầu người dùng mở hộp đựng tiêu bản và lựa chọn một mẫu tiêu bản để bỏ vào kính hiển vi (Hình 4.19). Sau khi đặt tiêu bản vào kính hiển vi, người dùng cần đưa mắt nhìn vào thị kính, ở trong chương trình người dùng sẽ nhấn vào nút ở gần thị kính (Hình 4.20).



Hình 4.19. Giao diện yêu cầu người dùng đặt tiêu bản vào kính hiển vi



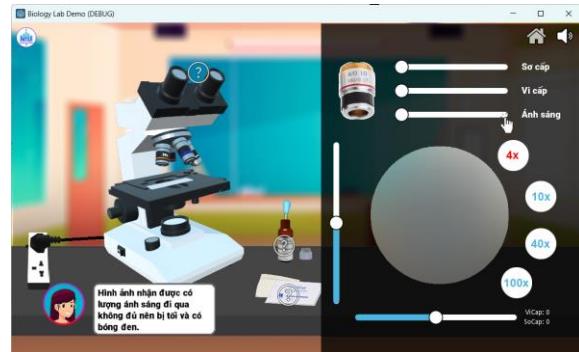
Hình 4.20. Giao diện yêu cầu người dùng nhìn vào thị kính

Trước khi thực hiện điều chỉnh ánh sáng người dùng cần tìm hiểu những điều cần lưu ý và xem qua các hình ảnh ví dụ trước khi thực hành điều chỉnh ánh sáng, đồng thời

màn hình điều chỉnh kính hiển vi cũng xuất hiện để chuẩn bị cho nội dung bài học tiếp theo (Hình 4.21). Theo ví dụ trong bài học, hình ảnh của tiêu bản khi hiển thị trên màn hình điều khiển kính hiển vi sẽ bị thiếu ánh sáng, chương trình sẽ yêu cầu người dùng tăng ánh sáng bằng cách gõ ý kéo nút ánh sáng để điều chỉnh (Hình 4.22).

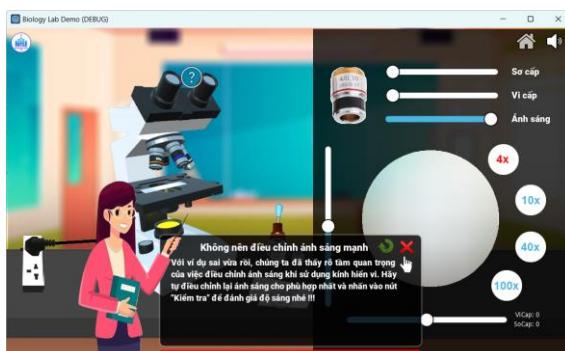


Hình 4.21. Những lưu ý khi thực hành điều chỉnh ánh sáng trong kính hiển vi

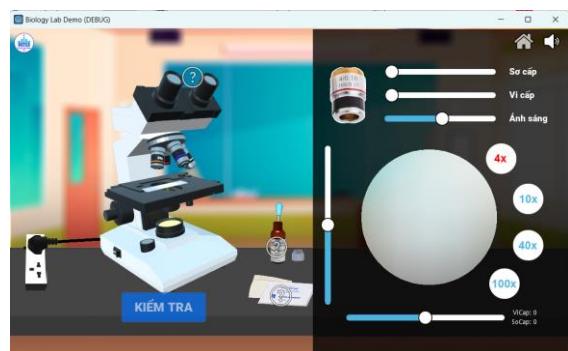


Hình 4.22. Yêu cầu tăng ánh sáng của kính hiển vi

Sau khi thực hiện điều chỉnh ánh sáng theo hướng dẫn, một dialog hiện ra thông báo rằng ánh sáng quá mạnh, cần điều chỉnh cho phù hợp với mắt của người quan sát (Hình 4.23). Để biết được ánh sáng được điều chỉnh đúng hay sai, người dùng nhấn vào nút kiểm tra (Hình 4.24). Nếu ánh sáng đã điều chỉnh phù hợp với mắt thì sẽ hiển thị thông báo “Ánh sáng vừa đủ” và chuyển sang bước 3 (Hình 4.25), nếu ánh sáng chưa phù hợp thì sẽ thông báo “Ánh sáng chưa phù hợp”. Để bắt đầu điều chỉnh quan sát mẫu vật, người dùng cần nắm được cách tính độ phóng đại của kính hiển vi (Hình 4.26).



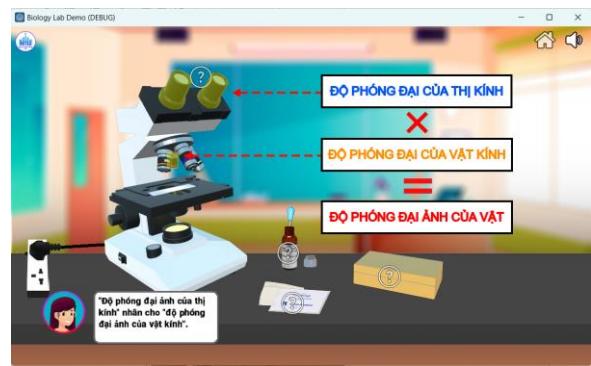
Hình 4.23. Giao diện thông báo người dùng đã điều chỉnh ánh sáng quá mạnh



Hình 4.24. Giao diện yêu cầu điều chỉnh ánh sáng và nhấn nút kiểm tra

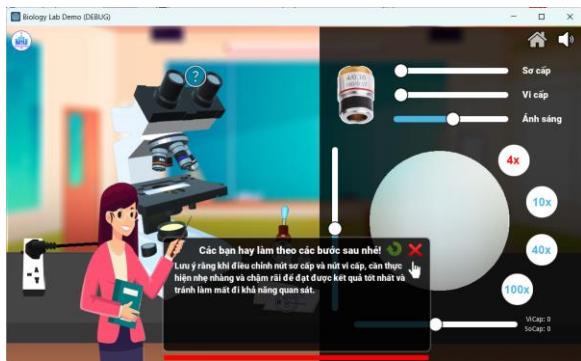


Hình 4.25. Giao diện giới thiệu bước 3 của bài hướng dẫn thực hành

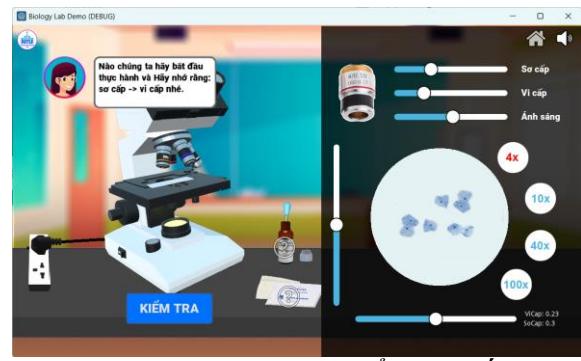


Hình 4.26. Cách tính độ phóng đại của kính hiển vi

Sau khi nắm được cách tính độ phóng đại của kính hiển vi, chương trình sẽ tiếp tục đưa ra các thông tin về các thao tác và những điểm cần lưu ý khi thực hiện lấy nét kính hiển vi (Hình 4.27). Người dùng bắt đầu điều chỉnh lấy nét mẫu vật đối với vật kính 4x, sau đó nhấn nút kiểm tra để xem kết quả lấy nét đã chính xác hay chưa (Hình 4.28).

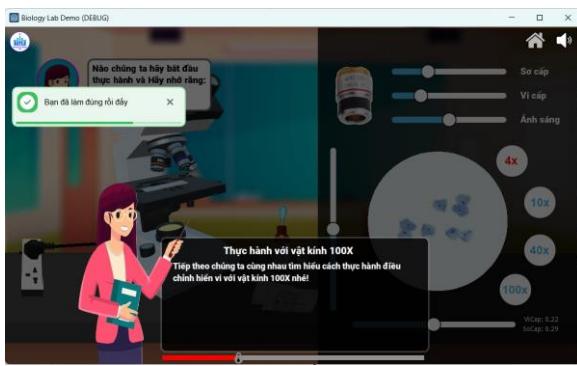


Hình 4.27. Giao diện hướng dẫn điều chỉnh lấy nét kính hiển vi



Hình 4.28. Giao diện kiểm tra lấy nét mẫu vật của kính hiển vi

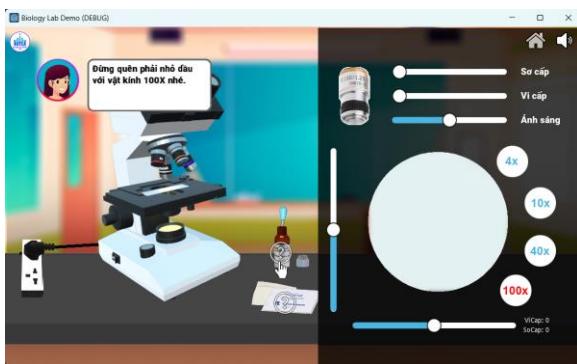
Tiếp theo người dùng sẽ học cách sử dụng vật kính 100x (Hình 4.29), và cách nhỏ dầu soi kính cho vật kính 100x (Hình 4.30). Sau khi tìm hiểu nội dung thực hành với kính hiển vi và dầu soi kính. Người dùng được yêu cầu thực hiện nhỏ dầu soi kính lên mẫu vật (Hình 4.31, Hình 4.32).



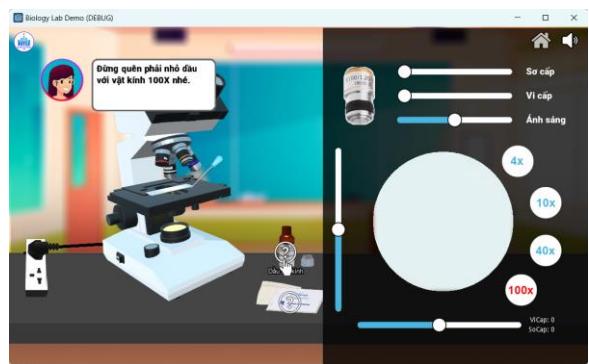
Hình 4.29. Giao diện giới thiệu thực hành với vật kính 100X



Hình 4.30. Tài liệu hướng dẫn nhỏ đầu soi kính hiển vi

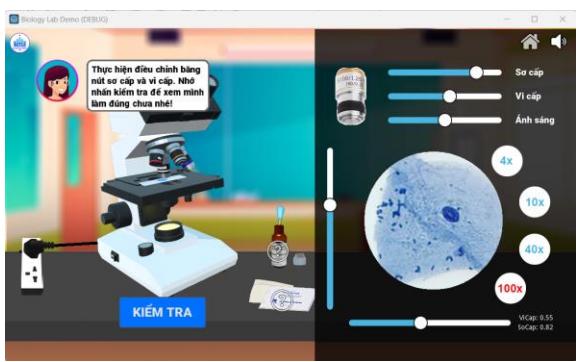


Hình 4.31. Yêu cầu người dùng nhỏ đầu soi kính



Hình 4.32. Animation nhỏ đầu soi kính

Người dùng cần điều chỉnh nút sơ cấp và vi cấp để lấy nét mẫu vật, sau đó nhấn nút kiểm tra (Hình 4.33) nếu thành công thì sẽ xuất hiện thông báo “Bạn đã làm đúng rồi đấy”, sau đó dialog thông báo bạn đã hoàn thành bài học.



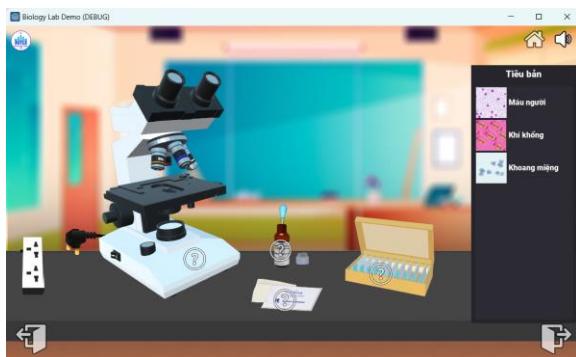
Hình 4.33. Giao diện thực hiện điều chỉnh lấy nét



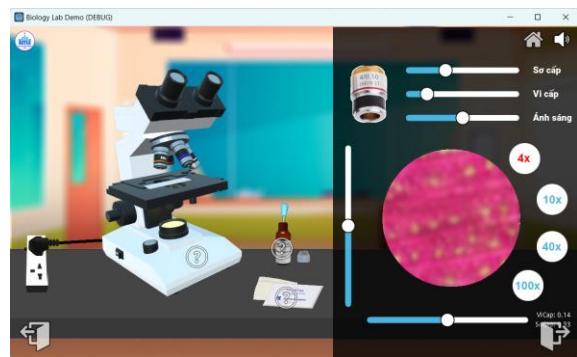
Hình 4.34. Giao diện thông báo đã hoàn thành bài học hướng dẫn thực hành

4.6. MÀN HÌNH THỰC HÀNH

Màn hình thực hành sẽ giống màn hình hướng dẫn thực hành, tuy nhiên danh sách mẫu vật sẽ nhiều hơn.



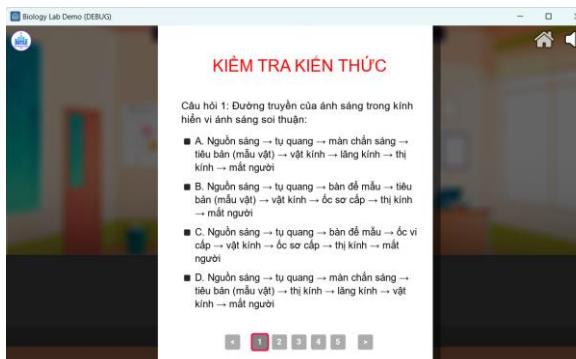
Hình 4.35. Giao diện danh sách mẫu vật có trong hộp đựng tiêu bản



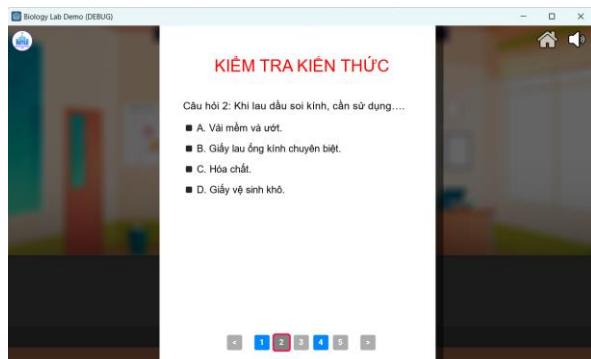
Hình 4.36. Giao diện bảng điều khiển các chức năng trong kính hiển vi

4.7. MÀN HÌNH KIỂM TRA

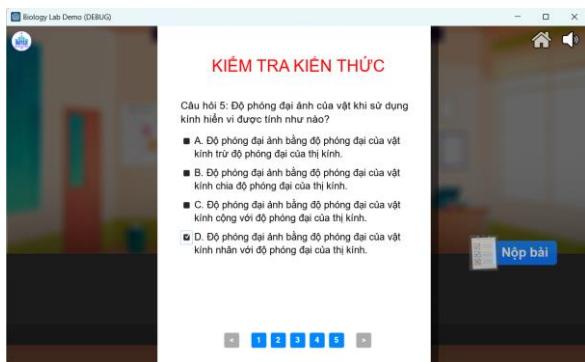
Sau khi người dùng hoàn thành tất cả các bài học, để kiểm tra lại kiến thức người dùng cần hoàn thành 5 câu hỏi trắc nghiệm (Hình 4.37). Người dùng lựa chọn các câu trả lời, khi một câu trả lời được lựa chọn thì nút chọn câu hỏi sẽ từ màu xám chuyển thành màu xanh da trời (Hình 4.38). Khi điền đầy đủ tất cả các câu hỏi, nút nộp bài sẽ hiển thị để người dùng nộp bài (Hình 4.39). Khi nhấn nút nộp bài thì sẽ hệ thống sẽ chấm điểm và hiển thị thông báo điểm của người dùng (Hình 4.40).



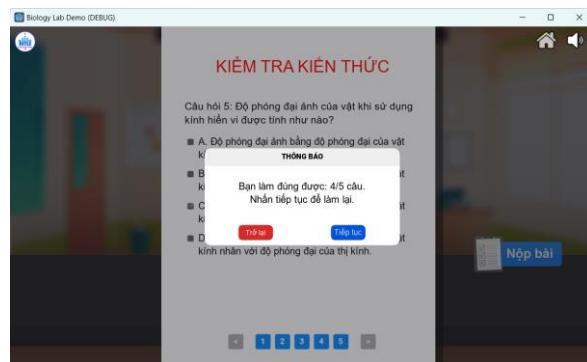
Hình 4.37. Giao diện màn hình kiểm tra kiến thức



Hình 4.38. Giao diện màn hình khi người dùng lựa chọn các câu hỏi

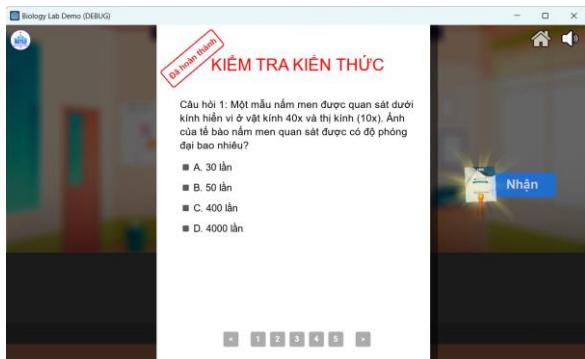


Hình 4.39. Giao diện nộp bài khi điền hết 5 câu hỏi trắc nghiệm



Hình 4.40. Giao diện thông báo điểm bài kiểm tra thực hành của người dùng

Người dùng khi làm đúng hết 5 câu hỏi trắc nghiệm thì sẽ đóng dấu mộc đã hoàn thành trên tờ kiểm tra và nút nhận bằng chứng nhận sẽ hiển thị ở bên phải màn hình (Hình 4.41). Sau khi vào giao diện nhận giấy chứng nhận thì người dùng có thể nhận giấy chứng nhận bằng cách nhấn vào nút tải xuống (Hình 4.42).



Hình 4.41. Giao diện đã hoàn thành bài kiểm tra kiến thức



Hình 4.42. Giao diện nhận giấy chứng nhận đã hoàn thành bài thực hành ảo

CHƯƠNG 5. TỔNG KẾT

5.1. KẾT LUẬN

Đồ án “Ứng dụng Godot xây dựng bài thực hành kính hiển vi ảo cho phòng thí nghiệm sinh học, Đại học Nha Trang” là sản phẩm mô phỏng nhằm giúp người dùng thực hành và nắm vững quy trình trong lĩnh vực kính hiển vi quang học. Đồ án này sử dụng môi trường Godot làm nền tảng Game Engine chính. Quá trình xây dựng đồ án đã bắt đầu bằng việc tìm hiểu sâu về Giáo dục số, bài giảng số và thí nghiệm ảo. Sau đó, đã tiến hành nghiên cứu và phân tích các công cụ phù hợp với yêu cầu, trong đó Godot được chọn làm công cụ chính. Ngoài ra, các phần mềm như Blender đã được sử dụng để tạo và tùy chỉnh vật thể 3D, và voicemaker để tạo ra âm thanh giọng nói AI.

Đồ án đã được trang bị một loạt tính năng hữu ích. Đầu tiên, hỗ trợ ngôn ngữ Tiếng Việt, giúp người dùng dễ dàng tiếp cận và hiểu rõ hơn về quy trình thực hành kính hiển vi quang học. Tương tác đối tượng trong môi trường 3D đã được thực hiện, mang lại sự chân thực và tương tác cho người dùng. Câu hỏi trắc nghiệm đã được tích hợp để kiểm tra kiến thức, trong khi khả năng tùy chỉnh vật thể 3D cho phép người dùng tạo ra các mô hình tùy chỉnh. Cuối cùng, hỗ trợ âm thanh giúp tăng cường trải nghiệm người dùng.

Mặc dù đồ án đã đạt được những thành tựu, nhưng vẫn còn một số phần chưa hoàn thiện, chẳng hạn như chưa hỗ trợ các dịch vụ trực tuyến như lưu trữ lại thông tin người dùng để có thể đăng nhập và học lại, trang quản lý các tài nguyên hệ thống giúp tùy biến các mẫu vật kính hiển vi khi cần cập nhập, hoặc đơn giản hơn là thay đổi các slide lý thuyết hay giọng nói một cách nhanh chóng. Tuy nhiên, với sự cố gắng và nỗ lực, đồ án “Ứng dụng Godot xây dựng bài thực hành kính hiển vi ảo cho phòng thí nghiệm sinh học, Đại học Nha Trang” hi vọng sẽ đem lại giá trị và hỗ trợ cho người học trong quá trình thực hành kính hiển vi.

5.2. HƯỚNG PHÁT TRIỂN ĐỀ TÀI

Đồ án “Ứng dụng Godot xây dựng bài thực hành kính hiển vi ảo cho phòng thí nghiệm sinh học, Đại học Nha Trang” không chỉ dừng lại ở mức đáp ứng các yêu cầu ban đầu về mô phỏng thực hành kính hiển vi, mà còn có thể mở rộng với nhiều tính năng hấp dẫn để cung cấp trải nghiệm tốt hơn cho người dùng trong quá trình sử dụng chương trình mô phỏng.

Các tính năng tiềm năng để bổ sung vào đồ án bao gồm tính năng Gửi thông tin và kết quả cho giảng viên, tạo môi trường mở để người dùng có thể tự do tương tác, xây dựng hệ thống chấm điểm đúng/sai theo từng bước thực hiện, cùng với bảng điểm để liệt kê những người thực hiện tốt nhất. Ngoài ra, việc bổ sung các hiện tượng sai khi làm sai bước sẽ giúp người dùng nhận biết và sửa lỗi một cách chính xác và hiệu quả. Đồ án “Ứng dụng Godot xây dựng bài thực hành kính hiển vi ảo cho phòng thí nghiệm sinh học, Đại học Nha Trang” là một trong những đồ án đáng chú ý trong lĩnh vực mô phỏng thí nghiệm. Bằng việc sử dụng môi trường Game Engine, đồ án đã tạo ra một mô phỏng ảo chân thực, mang đến cho người dùng trải nghiệm tương tự như trong thực tế. Điều này rất quan trọng trong bối cảnh gia tăng nhu cầu sử dụng các nền tảng giáo dục số tại các trường học.

Hơn nữa, các đồ án “Ứng dụng Godot xây dựng bài thực hành kính hiển vi ảo cho phòng thí nghiệm sinh học, Đại học Nha Trang” như một sự động viên đến những lập trình viên trẻ hãy cố gắng đổi mới, trao đổi kỹ năng lập trình. Các công nghệ và kỹ năng được áp dụng trong đồ án này đóng góp vào sự phát triển của ngành công nghiệp công nghệ thông tin, tạo ra một môi trường mở cho sự sáng tạo và đổi mới.

Tổng kết lại, đề tài “Ứng dụng Godot xây dựng bài thực hành kính hiển vi ảo cho phòng thí nghiệm sinh học, Đại học Nha Trang” không chỉ đạt được những thành tựu trong dạy học, mà còn mang trong mình tiềm năng để phát triển và nâng cao trải nghiệm người dùng trong tương lai. Chúng ta hi vọng rằng trong thời gian tới, các đồ án liên quan đến bài giảng số và chương trình mô phỏng sẽ được triển khai rộng rãi, góp phần mạnh mẽ vào sự phát triển bền vững của giáo dục trong nước và cung cấp cơ hội việc làm cho các lập trình viên trẻ.

TÀI LIỆU THAM KHẢO

- [1] Trần Quốc Trung, “Hệ thống quản lý học tập trực tuyến trong giáo dục đại học”, *Tạp chí giáo dục Việt Nam*, tr 66–72, 2021.
- [2] N. Thị Thanh Tú và c.s., “Ứng dụng công nghệ thực tế ảo tăng cường AR trong dạy học trực tuyến theo hình thức Microlearning”, *Tập*, vol 18, 2022, doi: 10.15625/2615-8957/12220206.
- [3] T. Văn Biều, “Tạp chí KHOA HỌC ĐHSP TPHCM MỘT SỐ VẤN ĐỀ VỀ ĐÀO TẠO TRỰC TUYẾN (E-LEARNING)”, 2012.
- [4] N. M. Hương, “GIẢI PHÁP CÔNG NGHỆ CHO DẠY HỌC THỰC HÀNH TRONG ĐÀO TẠO TRỰC TUYẾN TẠI TRƯỜNG ĐẠI HỌC MỞ HÀ NỘI”.