

MỤC LỤC

GIỚI THIỆU	7
CHƯƠNG 1 TỔNG QUAN NGÀNH CÔNG NGHIỆP TRÒ CHƠI ĐIỆN TỬ	9
1.1. LỊCH SỬ NGÀNH CÔNG NGHIỆP TRÒ CHƠI ĐIỆN TỬ 10	
1.1.1. Thời kỳ trước sự ra đời của Space War	10
1.1.2. Thời kỳ từ SpaceWar đến Atari.....	14
1.1.3. Thời kỳ máy chơi game chuyên dụng và máy tính cá nhân 17	
1.1.4. Giai đoạn lung lay và củng cố	22
1.1.5. Sự ra đời của Game Engine	24
1.1.6. Cuộc cách mạng của các thiết bị cầm tay	25
1.1.7. Hiện tượng điện thoại thông minh.....	27
1.1.8. Game nhiều người chơi	29
1.1.9. Tương lai ở phía trước	31
1.2. NHÂN LỰC VÀ VAI TRÒ TRONG PHÁT TRIỂN GAME 32	
1.2.1. Chiến lược gia (vision)	32
1.2.2. Đội ngũ quản trị sản xuất- Production.....	34
1.2.3. Đội thiết kế	43
1.2.4. Đội ngũ kỹ thuật	50
1.2.5. Đội ngũ mỹ thuật	55
1.2.6. Đội ngũ kiểm tra chất lượng sản phẩm.....	59
1.3. QUY TRÌNH PHÁT TRIỂN GAME VÀ CÁC TÀI LIỆU LIÊN QUAN	64
1.3.1. Xây dựng khái niệm	64

1.3.2. Giai đoạn Tiền sản xuất (Preproduction – Proof of Concept)	69
1.3.3. Quá trình phát triển.....	75
CHƯƠNG 2 NHẬP MÔN LẬP TRÌNH WINDOWS VÀ DIRECTX.	81
2.1. LẬP TRÌNH WINDOWS BẰNG C++	82
2.1.1. Hệ thống thông điệp của Windows (Windows Messaging) 82	
2.1.2. Đa nhiệm	83
2.1.3. Đa tiểu trình (multi-threading)	85
2.1.4. Xử lý sự kiện	87
2.2. TỔNG QUAN VỀ DIRECTX	88
2.3. CẤU TRÚC CHƯƠNG TRÌNH WINDOWS BẰNG C++ ...	90
2.3.1. Tạo một project Win32.....	91
2.3.2. Các tham số của WinMain	94
2.3.3. Khung hàm WinMain đầy đủ.....	95
2.4. VÒNG LẬP CHÍNH CỦA GAME (GAME LOOP)	98
2.5. KHỞI TẠO DIRECTX.....	101
2.5.1. Direct3D interface	103
2.5.2. Tạo đối tượng Direct3D và device	103
2.5.3. Game_Run.....	104
2.5.4. Game_End	105
2.5.5. Chạy trong chế độ toàn màn hình.....	106
2.5.6. Kiểm tra phím để thoát game khi ở chế độ fullscreen...	107
2.6. Surface và Bitmaps	107
2.6.1. Bề mặt chính.....	108
2.6.2. Bề mặt phụ ngoài (offscreen surface).....	109

2.6.3.	Tạo surface	109
2.6.4.	Vẽ lên Surface	109
2.6.5.	Ví dụ Create_Surface	111
2.6.6.	Tải ảnh lên từ ổ cứng.....	119
2.6.7.	Chương trình Load_Bitmap.....	120
CHƯƠNG 3 KỸ THUẬT TẠO CHUYỂN ĐỘNG.....		125
3.1.	Cách vẽ một sprite chuyển động.....	126
3.1.1.	Project Anim_Sprite	126
3.1.2.	Cài đặt project.....	126
3.1.3.	File source code	129
3.1.4.	Đồ họa của Sprite	141
3.2.	Vẽ Sprite Trong Suốt	154
3.2.1.	Tạo Đối Tượng Sprite Handler.....	155
3.2.2.	Bắt đầu Sprite Handler	155
3.2.3.	Vẽ Sprite	156
3.2.4.	Dừng Sprite Handler.....	157
3.2.5.	Tải Sprite Image	157
3.2.6.	Chương trình Trans_Sprite	160
3.2.7.	Tạo Project Trans_Sprite	161
3.3.	Vẽ Tiled Sprite.....	170
3.3.1.	Hiểu về Tile	170
3.3.2.	Chương trình Tiled_Sprite.....	171
CHƯƠNG 4 CÁC PHÉP BIẾN ĐỔI		177
4.1.	Biến đổi tọa độ (Transform) là gì?	179
4.1.1.	Flip.....	179
4.1.2.	Tịnh tiến.....	180

4.2. Biến đổi từ hệ toạ độ thế giới (World) sang View Port trong Game.....	186
4.1.3. Xác định toạ độ của Object trong View Port.....	188
4.1.4. Code biến đổi từ world sang view port	190
CHƯƠNG 5 LẬP TRÌNH CHUỘT VÀ BÀN PHÍM	192
5.1. Lập trình bàn phím.....	193
5.1.1. Đối tượng DirectInput và DirectInput Device.....	193
5.1.2. Khởi tạo đối tượng bàn phím.....	195
5.1.3. Thiết lập Định dạng Dữ liệu.....	195
5.1.4. Thiết lập mức độ hợp tác	196
5.1.5. Giành kiểm soát thiết bị.....	196
5.1.6. Đọc Phím nhấn	197
5.2. Lập trình chuột.....	198
5.2.1. Khởi tạo chuột	199
5.2.2. Thiết lập Định dạng Dữ liệu.....	199
5.2.3. Thiết lập Mức độ Hợp tác.....	199
5.2.4. Giành kiểm soát thiết bị.....	200
5.2.5. Đọc chuột.....	200
5.3. Ví dụ minh hoạ Paddle Game	202
5.3.1. Framework code mới cho DirectInput.....	203
5.3.2. Mã nguồn Paddle Game	209
5.3.3. Giải thích Paddle Game	219
CHƯƠNG 6 LẬP TRÌNH ÂM THANH.....	221
6.1. Tổng quan DirectSound	222
6.1.1. Khởi tạo DirectSound	223
6.1.2. Tạo bộ nhớ đệm âm thanh	224

6.1.3.	Nạp file định dạng wave	224
6.1.4.	Chạy hiệu ứng âm thanh.....	225
6.1.5.	Kiểm tra DirectSound	226
6.2.	Xây dựng ứng dụng minh họa	226
6.2.1.	Sao chép tái sử dụng mã nguồn.....	227
6.2.2.	Sao chép file dsutil	228
6.2.3.	Thêm những file copy vào project.....	229
6.2.4.	Thêm liên kết tới thư viện DirextX	231
6.2.5.	Tạo những file hỗ trợ cho DirectX Audio	233
6.2.6.	Tinh chỉnh hệ thống.....	236
6.3.	Chạy chương trình	244
CHƯƠNG 7 ĐẠI CƯƠNG XỬ LÝ VA CHẠM		245
7.1.	Giới thiệu	246
7.1.1.	Tại sao cần xử lý va chạm?	246
7.1.2.	Vấn đề đối với thuật toán “ngây thơ”	246
7.2.	Cài đặt thuật toán Swept AABB	249
7.3.	Xử lý va chạm.....	255
7.3.1.	Phản xạ	256
7.3.2.	Đẩy	257
7.3.3.	Trượt.....	258
7.4.	Các bài toán phụ trợ	259
7.4.1.	Kiểm tra diện rộng (Broad-Phasing)	259
7.4.2.	Giới hạn của thuật toán.....	262
CHƯƠNG 8 BÀI TOÁN PHÂN HOẠCH KHÔNG GIAN		264
8.1.	Tại sao cần phải phân hoạch không gian?	265
8.2.	Một số tiếp cận phân hoạch không gian trong game 2D	267

<i>Phân chia theo nhóm:</i>	267
<i>Cây nhị phân</i>	268
<i>Cây tứ phân</i>	268
8.3. Cài đặt kỹ thuật Quad tree	269
8.3.1. Build Tree	270
8.3.2. Lưu Quadtree vào file	273
8.3.3. Load QuadTree vào game	275
8.3.4. Lấy danh sách những đối tượng trong màn hình.....	277
8.4. Tóm tắt.....	278
TÀI LIỆU THAM KHẢO	280

GIỚI THIỆU

Công nghiệp trò chơi điện tử (video game) cùng với ngành công nghệ thông tin nói chung đã thay đổi sâu sắc đến sự nhu cầu giải trí của con người. Dù có chấp nhận hay không, thực tế vẫn cho thấy video game đã trở thành một phương tiện giải trí thiết yếu và kinh tế nhất của một phần lớn dân số trong xã hội hiện đại ngày nay. Ngành công nghiệp trò chơi điện tử đến nay đã là một ngành công nghiệp then chốt – đem lại một nguồn lợi nhuận khổng lồ - của nhiều quốc gia - đặc biệt là các quốc gia phát triển như Mỹ, Nhật và Trung Quốc.

Việt Nam chúng ta đã khá chậm chân trong cuộc cách mạng của ngành công nghiệp trò chơi điện tử. Chúng ta chỉ mới nhận ra tiềm năng to lớn của ngành này từ những năm 2000 và hoàn toàn bị tụt hậu so với thế giới. Tuy vậy, cơ hội vẫn chưa hoàn toàn trôi đi. Cuộc cách mạng di động và sự ra đời của các hệ sinh thái phần mềm như AppStore hay GooglePlay đã tạo một cơ hội công bằng cho những “kẻ chậm chân” như chúng ta có cơ hội bắt kịp với những “tay chơi” lớn trên thế giới.

Giáo trình Nhập môn Phát triển game sẽ giúp Sinh viên có một cái nhìn tổng quan về ngành công nghệ phần mềm và trang bị cho sinh viên những kiến thức *nền tảng* trong việc lập trình game nói chung (như xử lý va chạm, vật lý, tạo chuyển động, phân hoạch không gian, v...v) và kỹ thuật cần thiết để xây dựng một game 2D đơn giản (theo thể loại casual) trên nền tảng DirectX sử dụng C++.

Đây không phải là một môn có thể giúp Sinh viên ngay lập tức có thể làm ra các game thương mại có thể đưa lên các Appstore để kiếm tiền. Thay vì vậy, để phù hợp với mục tiêu đào tạo Kỹ Sư, chúng tôi trang bị cho các em những khởi đầu tốt để các em có thể nhìn thấu đáo những bài toán trong lập trình game để từ đó tự tìm hiểu và phát triển một cách bền vững trong tương lai.

Giáo trình được biên soạn với bài hướng dẫn chi tiết có tổng số trang là

TP HCM, ngày tháng năm 2013

Ban biên soạn:

1. PGS.TS. Vũ Thanh Nguyên
2. ThS. Đinh Nguyễn Anh Dũng

CHƯƠNG 1

TỔNG QUAN NGÀNH CÔNG NGHIỆP TRÒ CHƠI ĐIỆN TỬ

1.1. LỊCH SỬ NGÀNH CÔNG NGHIỆP TRÒ CHƠI ĐIỆN TỬ

Công nghệ và cả tính nghệ thuật của ngành công nghiệp game đã có những tiến bộ vượt bậc ngay từ những ngày đầu tiên. Sự phát triển như vũ bão của phần cứng đã làm độ phức tạp và tính phong phú của game “bùng nổ” dữ dội. Để có cái nhìn sâu sắc hơn về lập trình game ngày nay, chúng ta nên dành một ít thời gian nhìn lại lịch sử của lập trình game. Đây là một chủ đề khá dài, phong phú và có nhiều quan điểm khác nhau. Ở đây, chúng ta sẽ chia quá trình này thành 7 giai đoạn quan trọng nhất, có ý nghĩa lịch sử trong việc hình thành nên “điện mạo” của công nghiệp game ngày nay.

1.1.1. Thời kỳ trước sự ra đời của SpaceWar

Công nghiệp video game không bắt đầu bằng bất kỳ một chiến lược tổng thể nào. Nó dần hình thành khi các nhóm người làm việc tại những quốc gia khác nhau vô tình có chung ý tưởng gần như cùng lúc. Họ là các công ty kinh doanh những trò chơi bình thường tìm cách làm phong phú hướng kinh doanh; các công ty kỹ thuật cao tìm cách kinh doanh từ các công nghệ mới và một vài cá nhân có tầm nhìn xa đã tưởng tượng ra một hình thức giải trí mới sau này sẽ phát triển thành một ngành công nghiệp khổng lồ như chúng ta đã biết ngày nay.

Buổi bình minh của công nghiệp video game bắt đầu vào những năm đầu 1970. Tuy vậy các công ty đóng vai trò then chốt trong thời gian này lại ra đời trước đó khá lâu. Họ là các công ty kinh doanh trong các lĩnh vực khác nhưng sau cùng nhận ra rằng trò chơi trên các thiết bị điện tử sẽ giúp họ kiếm được nhiều tiền và sẽ trở thành một thị trường độc lập.

Một ví dụ là người khổng lồ Nintendo, một công ty kinh doanh trò chơi truyền thống được sáng lập từ năm 1889 bởi Fusajiro Yamauchi. Ban đầu hoạt động với cái tên Marafuku, lĩnh vực kinh doanh chính của họ là sản xuất và bán loại bài Hanafuda của người Nhật. Vào năm 1951,

Marafuku được đổi tên lại thành Nintendo. Sau này, khi các trò chơi điện tử đầu tiên xuất hiện, Nintendo thành lập một bộ phận kinh doanh khác để kinh doanh trò chơi điện tử. Thời gian trôi qua, thị trường trò chơi truyền thống ngày càng mai một dần và Nintendo trở thành công ty mà chúng ta đều đã biết ngày nay. Như vậy, Nintendo là một ví dụ điển hình của một công ty chuyển đổi hướng kinh doanh để đi vào công nghiệp video game.

Sony lại theo một hướng hoàn toàn khác. Sony ban đầu được tạo ra để tập trung vào thị trường điện tử tiêu dùng. Sony được Akio Morita và Masaru Ibuka sáng lập dưới cái tên Tokyo Telecommunications Engineering Corporation (Tập đoàn kỹ thuật viễn thông Tokyo) vào năm 1946. Lĩnh vực kinh doanh chính của công ty là các máy chơi nhạc cát sét. Khi các sản phẩm của công ty bắt đầu vào thị trường châu Âu và Mỹ, các nhà sáng lập đã quyết định đổi tên để nó dễ nhớ hơn đối với các khách hàng không phải người Nhật. Sony nhanh chóng trở thành một trong các nhà cung cấp hàng đầu trong thị trường hàng điện tử tiêu dùng, đặc biệt là trong lĩnh vực nghe nhìn. Nhưng Sony vẫn đứng ngoài thị trường video cho đến cuối những năm 1980 khi máy chơi game PlayStation ra đời. Ngày nay, Sony sản xuất các máy chơi trò chơi điện tử (Playstation2, Playstation3, PSP) cũng như sản xuất bản thân các trò chơi điện tử từ khắp các chi nhánh trên thế giới. Hệ thống máy Playstation hiện vẫn đang đóng vai trò trung tâm trong chiến lược kinh doanh tổng thể của Sony.

Nhóm thứ 3 chiếm thiểu số là các công ty trung gian giữa các công ty chuyên về công nghệ và công ty trò chơi truyền thống. Sega là một ví dụ. Câu chuyện của Sega bắt đầu vào năm 1940 khi Martin Bromely, Irving Bromberg và Hames Humpert sáng lập công ty Standard Games, một công ty chuyên sản xuất máy chơi game thùng (máy chơi game bằng cách bỏ đồng xu vào) đặt trụ sở tại Honolulu. Vào năm 1951, công ty dời sang Tokyo và sau đó đổi tên lại thành Service Games (gọi tắt là Sega). Lý do dời chỗ là để cho công ty dễ dàng cung cấp các máy game thùng cho lực lượng Mỹ đóng tại Nhật. Vài năm sau, 1965, Sega sát nhập với Rosen Enterprise, một công ty kinh doanh tổng hợp. Rosen

Enterprise được sáng lập vào năm 1954 bởi David Rosen - một cựu chiến binh trong cuộc chiến Nam Triều Tiên. Rosen nhận thấy sự phổ biến của các máy game thùng (như game pinball) trong các doanh trại quân đội Mỹ tại Nhật. Rosen bắt đầu xuất khẩu các máy này sang thị trường Nhật dưới cái tên Sega. Khi công việc kinh doanh phát đạt, Rosen bắt đầu sản xuất các trò chơi của chính họ bằng cách mua lại một công ty gốc từ Tokyo chuyên sản máy chơi nhạc đồng xu (Jukebox). Cuối cùng, Sega tập trung sản xuất game thùng và sau đó mở rộng sang máy chơi game chuyên dụng tại nhà và phát triển game.

Tuy nhiên, không phải Nintendo, Sony hay ngay cả Sega là người dẫn đường của giải trí điện tử. Những công ty này chỉ bước chân vào công nghiệp game theo sau những nhà tiên phong thực sự; những người đã tạo ra những thiết kế đầu tiên cũng như những mô hình kinh doanh đầu tiên. Dĩ nhiên, cần có một ai đó hình dung ra cách chơi game trên các thiết bị điện tử. Đó chính là các nhà nghiên cứu làm việc tại các trường đại học và trong quân đội bởi vì họ là những người có điều kiện tiếp xúc với những thiết bị phần cứng tối tân nhất (vào thời điểm đó).

Người đầu tiên trong số những nhà tiên phong này là William Higinbotham - một người yêu thích pinball và là một nhà vật lý hạt nhân tại phòng thí nghiệm quốc gia Brookhaven tại New York. Trong những năm 1950, Brookhaven là một trung tâm nghiên cứu được quốc gia bảo trợ, chủ yếu nghiên cứu ứng dụng năng lượng hạt nhân. Trung tâm thường xuyên đón khách thăm quan đến tìm hiểu những ứng dụng hòa bình của năng lượng nguyên tử. Trung tâm trưng bày các hình ảnh, các mẫu thiết bị để minh họa mọi thứ từ nhà máy điện đến các loại thuốc



Hình 1.1. “Trò chơi” Tennis for two

dựa vào năng lượng phóng xạ. Higinbotham cho rằng khách tham quan có thể bị nhầm chán nên đã thiết kế ra một thiết bị kỳ lạ sử dụng các thiết bị còn dư của phòng thí nghiệm: một máy đo dao động điện (oscilloscope), vài cái tụ điện và một máy tính analog nhỏ. Ông gọi phát minh này là “Tennis for two”. Đó là một trò chơi bóng bàn đơn giản hai người chơi trong đó cái bàn và trái banh được hiển thị trên màn hình của thiết bị đo dao động điện. Người chơi có thể đổi góc đánh banh bằng cách chỉnh cái điện kế. Trò chơi hầu như chỉ bao gồm phần cứng nên không thể xem là “lập trình”.

Cũng như nhiều thiên tài khác, Higinbotham đã không nhận ra ông đã đạt được điều gì, ngay cả khi người ta bắt đầu xếp hàng dài để chờ chơi

trò chơi của ông tại Brookhaven. Đó là vào năm 1958, cũng vào khoảng thời gian đó, nhiều người khác trên thế giới cũng đạt được cùng kết quả tương tự. Vào đầu năm 1952, A.S. Douglas trình luận án tiến sĩ về lý thuyết tương tác người-máy tại trường đại học Cambridge, Anh. Để minh họa, ông đã lập trình game tic-tac-toe trên một máy tính EDSAC để minh họa lý thuyết của mình.

Vẫn còn nhiều câu chuyện tương tự như vậy diễn ra vào khoảng những năm 1950, tuy vậy phát minh của Higinbotham được xem là một trong số những công trình hoàn chỉnh nhất thời bấy giờ.

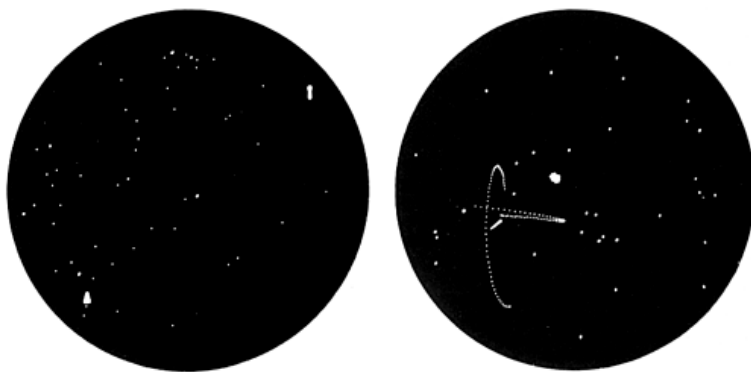
Một nhà tiên phong nữa cũng đáng ghi nhớ là Ralph Baer, người đã phát minh ra máy chơi game chuyên dụng tại nhà (console) vào năm 1951. Trong khi làm việc, Baer nhận được nhiệm vụ thiết kế một bộ thiết bị ti-vi mới. Ông đã đề nghị với công ty tăng cường tính năng của thiết bị này bằng một vài trò chơi đơn giản. Lãnh đạo công ty đã phớt lờ ý tưởng này. 15 năm sau, Baer thử ý tưởng của mình một lần nữa khi làm việc cho một công ty khác. Ông thành công lần này và bắt đầu làm việc để xây dựng nên thiết bị chơi game chuyên dụng tại nhà đầu tiên – Magnavox Odyssey.

Trong thời kỳ đầu tiên này, tuy đã có một số game thử nghiệm xuất hiện nhưng tất cả đều được cài đặt bằng những phần cứng chuyên biệt. Lúc này, phần cứng chính là game. Do đó, lập trình game chưa thể được xem là đã ra đời vào thời gian này. Vào năm 1960, cần có một chất xúc tác giữa các game truyền thống, các nhà cung cấp công nghệ và các nhà nghiên cứu để tạo ra được một game thành công để chứng tỏ rằng 3 yếu tố này có thể phối hợp với nhau tạo nên một ngành kinh doanh mới.

1.1.2. Thời kỳ từ SpaceWar đến Atari

Điểm sang trang của công nghiệp game bắt đầu vào năm 1961 khi Steve Russell, một sinh viên MIT, lập trình một trò chơi đơn giản cho hai người chơi trên một máy tính mini Digital PDP-1. Trò chơi này được

gọi là SpaceWar. Trò chơi hiển thị hai con tàu vũ không gian trên màn hình. Cả hai con tàu có thể được điều khiển để di chuyển độc lập và bắn vào tàu đối phương.



Hình 1.2. Trò chơi SpaceWar

Trò chơi không đến tay người dùng cuối nhưng nó đã tạo cảm hứng cho nhiều người khác. SpaceWar thực sự là một trò chơi đúng nghĩa đen của nó. Nó sử dụng công nghệ mới và khai thông con đường cho nhiều công nghệ khác. Trò chơi có các đặc tính:

- Đối kháng giữa hai người chơi
- Luật chơi
- Định nghĩa rõ ràng về thắng thua.

Cấu trúc trên không khác mấy so với các trò chơi truyền thống như cờ vua. Điểm khác biệt chính là công nghệ hỗ trợ cho việc chơi. Tuy với thời gian, công nghệ này đã phát triển vượt bậc và bản thân các trò chơi cũng trở nên đặc biệt phong phú, nhưng 3 đặc tính cơ bản trên của các trò chơi vẫn không thay đổi mấy.

Nhiều người đã chịu tác động sâu sắc từ SpaceWar, nhưng chúng ta chỉ đề cập đến hai nhà tiên phong có dấu ấn rõ rệt nhất. Nolan Bushnell có dịp tiếp xúc với SpaceWar trong khi đang học kỹ sư tại trường Đại Học

Utah. Ông hình dung ra viễn cảnh các trò chơi trên máy tính như SpaceWar được cài đặt trên các máy giải trí, hộp hồn nhiều người chơi. Vài năm sau, tiên đoán của ông đã thành sự thật khi ông sáng lập Ataria và tạo ra chiếc máy trò chơi điện tử chạy bằng đồng xu đầu tiên.

Sau khi tiếp xúc với SpaceWar, Bushnell bắt đầu thiết kế một chiếc máy chơi chuyên dụng với giá cả phải chăng. Game đầu tiên của ông, trước khi Atari ra đời nhiều năm, được gọi là Computer Space. Đó là một phiên bản của SpaceWar được “cứng hóa” và gắn vào bộ điều khiển TV trong phòng ngủ của con gái ông. Nutting Associates, một công ty chuyên sản xuất máy giải trí, mua lại ý tưởng về Computer Space và mướn Bushnell để giám sát dây chuyền sản xuất. Năm 1971, 1500 máy chơi Computer Space đã được xuất xưởng và lắp đặt tại Hoa Kỳ. Nhưng người chơi tỏ ra lạnh lùng với trò chơi này vì nó quá khó chơi. Bushnell cố gắng xây dựng những ý tưởng trò chơi mới nhưng sau vài lần đàm phán, ông nghỉ việc ở Nutting. Kết quả là ông sáng lập ra Atari năm 1972 cùng với Ted Dabney. Atari được lấy từ tên một trò chơi của người Nhật.

Ở “phía bên kia chiến tuyến” là Ralph Baer, người tiên phong trong ý tưởng trò chơi trên TV từ năm 1950. Vào năm 1966, ông đã nghỉ việc ở Loral và đang làm việc cho Sanders Associates, một nhà thầu quân đội. Ông được công ty bật đèn xanh để tiếp tục nghiên cứu ý tưởng của mình. Năm 1968, ông đăng ký bản quyền ý tưởng của mình. Kết quả là đã có rất nhiều thiết bị đã được mua và lắp đặt tại nhà. Vào lúc đó, ông đã tạo ra được hai trò chơi trên TV và một thiết kế đầu tiên về khẩu súng ánh sáng. Vào năm 1970, nhà sản xuất TV Magnavox mua bản quyền công nghệ của Baer. Dưới sự hướng dẫn của Baer, chiếc máy chơi game chuyên dụng đầu tiên có tên là Magnavox Odyssey đã ra đời vào năm 1972, cùng năm với sự xuất hiện của Atari.

Cuối năm 1972, ngành công nghiệp trò chơi điện tử bắt đầu “xâm lấn” thế giới như vũ (bắt đầu từ Hoa Kỳ). Trò chơi Atari đầu tiên có tên là *Pong* trở thành ví dụ đầu tiên về sự thành công của máy chơi game

đồng xu. Trong vòng hai tuần, nhiều máy chơi Pong ở bang California bị hư hỏng vì có quá nhiều đồng xu $\frac{1}{4}$ đô la trong máy làm nghẽn máy, một việc mà Bushnell không bao giờ mơ tới. Cùng lúc đó, Magnavox đạt được doanh số 100 000 máy, đây là một chiến tích đáng nể vì máy Magnavox chỉ được bán tại các cửa hàng phân phối của chính hãng Magnavox.

Như vậy, trong thời kỳ này, hai mô hình kinh doanh mới đã xuất hiện: máy chơi game tiền xu (arcade) chơi đến khi “chết” và hệ thống chơi game tại nhà cho phép người chơi có thể chơi bao lâu tùy thích. Cả hai hình thức này đều phát triển và đều còn tồn tại đến nay với hình thức chơi game tại nhà chiếm ưu thế hơn hẳn. Hình thức thứ 3 lúc này chưa xuất hiện. Đó là hình thức mà trò chơi không cần phải chơi trên các phần cứng chuyên biệt nữa.

1.1.3. Thời kỳ máy chơi game chuyên dụng và máy tính cá nhân

Trò chơi điện tử, dù chơi trên các hệ chơi game tại nhà hoặc trên máy bỏ đồng xu, đều nhanh chóng thành công. Các nhà sản xuất khác như Coleco, Fairchild, ... bắt đầu giới thiệu các hệ thống chơi game tại nhà của mình. Như một quy luật tất yếu, cạnh tranh dẫn đến sáng tạo. Kết quả là sự ra đời của băng trò chơi điện tử (cartridge) vào năm 1976, các loại trackball vào năm 1978 và nhiều phát minh khác nữa.

Các máy chơi game (console) và các nhà phát triển độc lập

Cũng chính vào thời điểm phát triển nhanh chóng này, đầu đó trong thập kỷ 1970, mà sự phân biệt rạch ròi giữa nhà sản xuất thiết bị chơi game (console) và nhà phát triển game đã được hình thành. Các nhà sản xuất console sẽ đầu tư một số tiền lớn vào việc xây dựng phần cứng và bán cho dân chơi game với giá rất hấp dẫn. Thậm chí nhiều lúc còn thấp hơn cả giá thực sự của phần cứng của thiết bị đó. Các nhà sản xuất cũng xây dựng các bộ công cụ (vào thời điểm này là rất thô sơ) cho phép các

công thi bên ngoài thiết kế trò chơi cho console. Lý do để các nhà sản xuất cho phép các công ty khác “chia sẻ” khá hiển nhiên: khi thị trường có nhiều máy chơi game, hệ máy nào có nhiều trò chơi hơn sẽ có khả năng cạnh tranh lớn hơn. Lý do này đã nâng cao vai trò của các nhà phát triển game hơn vì không một nhà sản xuất nào tự mình đủ sức xây dựng được một lượng game đủ lớn để thu hút người chơi.

Như vậy, các công ty “bên ngoài” bắt đầu lập trình game cho các hệ console. Trong giai đoạn này, hầu như các nhà sản xuất ít hoặc không thực hiện quy trình kiểm tra chất lượng của game. Lúc này số lượng quan trọng hơn chất lượng. Cho mỗi bản sao của game bán được, các nhà phát triển chấp nhận trả cho nhà sản xuất console một khoảng tiền gọi là *phí đồng hành* (royalty fee) để bù đắp chi phí phát triển console cũng như tạo doanh thu cho nhà sản xuất. Một số nhà sản xuất chỉ tập trung nghiên cứu phát triển console, một số khác lại chọn giải pháp vừa là nhà sản xuất console, vừa là nhà phát triển game. Mô hình này thường gây tranh cãi về việc các nhà phát triển game “gà nhà” có được “ưu tiên” hơn các nhà phát triển game khác hay không? Tuy vậy, cả hai mô hình này vẫn tồn tại cho đến ngày nay. Điểm khác biệt với mô hình ngày nay là các console được bán với giá thấp hơn hẳn giá trị thật của nó, đặc biệt là lúc nó mới được tung ra thị trường. Đây là một quyết định rất nguy hiểm vì nhà sản xuất phải gồng mình chịu lỗ một thời gian. Sau đó, khi nhận được *phí đồng hành* từ các nhà phát triển game, các nhà sản xuất mới bắt đầu giảm lỗ và cuối cùng là có lời.

Thiết bị console tiêu biểu trong thời gian này là Atari VCS (hay Atari2600) được giới thiệu vào năm 1977 với giá 247 USD. Thiết bị này dùng CPU 6507 có **128 bytes** được dùng để chứa các biến trạng thái như “máu” hay số đạn được còn lại. Chương trình trò chơi được chứa trong một “băng trò chơi” bên ngoài, có thể gắn vào tháo ra khỏi thiết bị được. Khi gắn vào thiết bị, “băng trò chơi” trở thành một phần cứng thuộc về thiết bị, thông thường là bộ nhớ để chứa mã chương trình và dữ liệu.

Phương pháp dùng “băng trò chơi” dùng làm bộ nhớ được sử dụng cho đến khi Nintendo 64 và Gameboy Advance ra đời cho phép dùng các loại thiết bị lưu trữ khác như các loại đĩa. Lưu trữ game trong đĩa CD-ROMs và DVDs giúp giảm đáng kể chi phí sản xuất. Đổi lại, điều này làm tăng khả năng bị sao chép lậu và phải tăng cường bộ nhớ của thiết bị để chứa mã chương trình. Trở lại Atari 2600, mỗi “băng trò chơi” có chứa 6KB ROM (để chứa mã chương trình và dữ liệu) và một chip nhớ. CPU 6507 hoạt động ở tần số 1.19MHz và có thể định vị tối đa 8KB bộ nhớ.

Hỗ trợ cho CPU là một loại card màn hình gọi là TIA hoặc Stella và một chip I/O gọi là RIOT. Stella được truy cập thông qua các thanh ghi và cực kỳ bị giới hạn vì không có bộ nhớ màn hình để lưu trữ nội dung hiện tại của màn hình. Các card màn hình ngày nay đều có bộ nhớ màn hình để lưu trữ tất cả các điểm ảnh đang được hiển thị trên màn hình. Ghi vào bộ nhớ này chính là vẽ lên màn hình. Máy Atari 2600 không có bộ nhớ màn hình như vậy. Việc vẽ lên màn hình được thực hiện bằng cách đọc tuần tự một vài thanh ghi của Stella một cách đồng bộ theo chùm electron của TV. Bản thân CPU cũng phải đồng bộ với chùm electron của TV và phải ghi dữ liệu vào các thanh ghi với một tốc độ hợp lý để đảm bảo hình ảnh được hiển thị đúng và không bị nháng.

Atari 2600, cũng như mọi loại console vào thời kỳ này, được lập trình bằng hợp ngữ. Cả chương trình và dữ liệu đều nằm chung trong một khối bộ nhớ. Dữ liệu thường được lưu trữ dưới dạng một chuỗi bit nằm ngay sau mã chương trình, sao cho chương trình không bao giờ “chạy” đến. Nếu điều này xảy ra, đơn giản là toàn bộ hệ thống sẽ “bị điên”.

Thời kỳ này, đa số các game đều do một người làm ra. Người này phải cẩn thận xây dựng bản đồ bộ nhớ (nơi nào chứa code, nơi nào chứa dữ liệu và chứa dữ liệu gì), viết chương trình, thiết kế đồ họa và thậm chí tạo cả âm thanh. Khái niệm tái sử dụng mã mà ngày nay quá quen thuộc với chúng ta hầu như không tồn tại. Thật ra thì đôi lúc các lập trình viên cũng sao chép lại một vài đoạn lệnh để các game sau được viết nhanh

hơn. Nhưng nói chung, lập trình cho một chiếc máy như vậy đòi hỏi kỹ năng cao và sự tỉ mỉ cao độ.

Máy tính cá nhân

Trong lúc các game thủ đang say sưa chơi game trên console, cuộc cách mạng máy tính cá nhân chuẩn bị diễn ra. Máy tính xuất hiện như là một hệ quả tất yếu của việc phát minh ra mạch tích hợp của công ty Texas Instrument vào năm 1959. Máy tính đầu tiên, Digital PDP-1, xuất hiện vào năm 1960 có một bàn phím và màn hình. Nó được tung ra thị trường với giá 120.000USD, với mục tiêu rõ ràng là nhắm vào thị trường doanh nghiệp với kế toán, kỹ thuật và cơ sở dữ liệu là 3 mục tiêu chính. Vào năm 1964, ngôn ngữ BASIC ra đời đã giúp lập trình trở nên dễ dàng hơn.

Douglas Engelbart phát minh ra con chuột máy tính. Năm 1968 (một năm trước khi nhân loại đặt chân lên mặt trăng), máy tính cá nhân đầu tiên, Honeywell H316 ra đời với giá bán 10600USD. Kế đến là sự ra đời của máy tính dẫn đường cho Apollo, một thành tựu rất ấn tượng. Máy tính này có tốc độ xung 2MHz, 64KB ROM, 4KB RAM, có thể thực hiện 50000 phép cộng một giây.

Năm 1972, khi Atari và Magnavox thành công, máy tính đã bắt đầu trở nên phổ biến. Ngôn ngữ C ra đời, ý tưởng đầu tiên về máy tính xách tay xuất hiện tại trung tâm nghiên cứu của Xerox đặt tại Palo Alto và Intel giới thiệu bộ vi xử lý 8008 – một chip giá thành thấp, định vị được 6KB bộ nhớ, xung nhịp 200KHz, thực hiện được 60000 lệnh mỗi giây. Từ thời điểm này trở đi, máy tính phát triển như vũ bão. Vào năm 1974, Intel ra đời 8080, xung nhịp 2MHz, định vị 16KB bộ nhớ, 640 000 lệnh mỗi giây. Cùng năm đó, máy MITS Altair 8800 được công bố tại hội chợ Popular Electronics với giá 400USD.

Một năm sau, Microsoft được sáng lập. Sản phẩm chủ lực là trình thông dịch BASIC cho máy Altair. Đó là ngôn ngữ lập trình đầu tiên cho máy

tính cá nhân. Vài năm sau, Microsoft xây dựng hệ điều hành cho máy IBM PC. Tiếp theo là lịch sử mà ai cũng biết.

Năm 1976, một trong những công ty ra đời từ ga-ra nổi tiếng nhất xuất hiện và ngành công nghiệp máy tính cá nhân chính thức được ra đời. Apple được sáng lập bởi Steven Jobs và Steve Wozniak. Jobs với con mắt nhà nghề trong thiết kế và kinh doanh đã làm việc trong Atari từ năm 1974. Wozniak, một thiên tài về phần cứng, đang làm việc cho Hewlett-Packard. Sản phẩm đầu tiên của họ, máy Apple-I, bán theo bộ với giá 666USD và được giới thiệu tại câu lạc bộ máy tính Homebrew. Một vài trò chơi đơn giản đã xuất hiện trên Apple I nhưng chủ yếu vẫn là bản sao từ các game tương tự trên console.

Máy Apple II ra đời năm 1977 đánh dấu sự phát triển nhanh chóng của game trên máy tính cá nhân. Yếu tố quyết định trong định hướng này là việc máy Apple II dùng CPU 6502, cực kỳ giống CPU của máy Atari 2600. Máy có 16KB RAM và có thể đạt tới 48KB RAM. Bộ “đồ nghề” của máy gồm có trình thông dịch BASIC cho phép người dùng tự tạo chương trình của mình, một bàn phím QWERTY như ngày nay, một giao tiếp với đầu đọc đĩa cát-sét, một cổng I/O để nối tay cầm chơi game, joysticks. Đặc biệt là nó có thể hiển thị một hình ảnh rộng 280x192 16 màu, một điều kiện rất lý tưởng để thể hiện trò chơi.

Lúc bấy giờ, Apple II là đỉnh cao của công nghệ, vượt trội các sản phẩm khác của các đối thủ cạnh tranh, kể cả các máy console. Apple II có thể được dùng để lập trình, để chơi game và các ứng dụng văn phòng. Máy có một lượng bộ nhớ lớn (so với thời bấy giờ) và có đầy đủ màu. Nhược điểm rõ ràng là giá cả. Máy với cấu hình cơ bản có giá 600USD, trong khi máy với cấu hình tối đa, 48KB (có thể xem là một “siêu máy tính” thời đó) có giá 2275USD. Giá cao như vậy chủ yếu là do giá RAM thời đó còn quá cao. Tuy vậy, máy Apple II vẫn được bán với số lượng được đo bằng đơn vị xe tải! Hàng trăm game khác nhau đã được ra đời trên máy này – trong đó có trò chơi cực kỳ phổ thông Ultima.

5 năm sau, IBM ra đời máy IBM-PC, chạy trên bộ vi xử lý Intel 8086 và hệ điều hành MS-DOS. Điểm lợi chính của PC là kiến trúc mở. Điều này cho phép các công ty khác có thể thiết kế và bán các máy PC có kiến trúc tương thích với IBM-PC. Một lần nữa, cạnh tranh thúc đẩy sáng tạo và một kiến trúc mở là cách tốt nhất để tạo ra cuộc cách mạng. Kỷ nguyên của máy tính cá nhân bắt đầu.

1.1.4. Giai đoạn lung lay và củng cố

Với Ataria 2600 và Apple II trên thị trường, định hướng cho ngành kinh doanh trò chơi điện tử đã có những định hướng rõ ràng. Do vậy, thời kỳ giữa cuối thập kỷ 1970 và đầu 1980 được gọi là thời kỳ vàng. Tất cả những gì cần làm trong lúc này là cải tiến các sản phẩm ban đầu, tạo các thiết bị phần cứng cũng như những game tốt hơn tại mỗi chu kỳ sản phẩm. Trên mặt trận console, Atari vẫn chiếm lĩnh thị trường trong nhiều năm, ngay cả khi nhà sáng lập Nolan Bushnell rời công ty vào năm 1978.

Theo sự dẫn đường của các nhà sáng lập, Atari đã tận hưởng sự giàu sang nhờ vào số lượng game khổng lồ đã được phát hành. Thời kỳ này cũng đánh dấu sự tham gia của các nhà phát triển game Nhật Bản. Taito giới thiệu game Space Invaders; Nintendo bắt đầu bán game Othello trên máy chơi game tiền xu và sau đó là Donkey Kong (trò chơi đầu tay của nhà thiết kế game bây giờ đã thành huyền thoại – Shigeru Miyamoto) vào năm 1981; Namco giới thiệu các game cổ điển như Galaxian, Pac-man, Galaga và Pole Position.

Sau một thời gian, những ngày tháng vàng son của Ataria đã bắt đầu kết thúc. Công ty đã không cẩn thận kiểm soát chất lượng các trò chơi trên hệ 2600. Nhiều trò chơi được đầu tư lớn trên máy 2600 chỉ là những chuỗi thất bại. Pac-man và E.T là hai trò chơi được ra mắt công chúng với nhiều tầng bậc đến mây xanh nhưng lại nhận được sự từ chối thẳng thừng của game thủ vì chất lượng kém của chúng. Người ta kháo nhau rằng hàng ngàn bản sao của trò chơi này đã được dùng để lấp đất tại

New Mexico. Tháng 12 năm 1982, Atari công bố rằng doanh số của 2600, lần đầu tiên, thấp hơn mong đợi. Điều này đã làm giá chứng khoán của công ty mẹ của Atari, Warner Communications, giảm 32% trong vòng chỉ trong một ngày. Vào đầu năm 1983, cùng với sự vật vờ của Ataria để phục hồi sau thất bại của E.T, thị trường trò chơi điện tử đã vấp phải một cú ngã đầu tiên. Các nhà bán lẻ tồn kho hàng đồng các game không bán được, trong đó có rất nhiều game kém chất lượng. Giá game tụt dốc thảm hại. Nhiều game từng được bán với giá 40USD chỉ còn 1USD, khiến cho nhiều công ty phá sản. Nhiều công ty khác trụ lại được nhưng hầu như không thể hoàn toàn phục hồi sau cú ngã này.

Phải một thời gian sau, ngành công nghiệp game mới phục hồi lại được từ kinh nghiệm xương máu này. May thay, năm 1985, một số các công ty đã tạo ra làn sóng mới các sản phẩm cùng với các chính sách kiểm tra chất lượng nghiêm ngặt. Nintendo giới thiệu máy console NES với sự “bảo kê” của một loạt game chất lượng cao do chính công ty sản xuất.. Một thời gian ngắn sau, một vài nhà phát triển cho Ataria như Namco chuyển sang đầu quân cho Nintendo. Vào năm 1988, các trò chơi cổ điển như Super Mario Bros và Legend of Zelda được giới thiệu biến NES thành máy console bán chạy nhất. Sega cũng không chịu ngồi yên, giới thiệu Master System vào năm 1986. Về kiến trúc, cả hai hệ này đều dùng bộ vi xử lý 8-bit, NES dùng 6502 và Master dùng Zilog Z80). So sánh chi li thì Master System hơi mạnh hơn: nó hỗ trợ nhiều sprites hơn, RAM dung lượng lớn hơn. Nhưng NES có lợi thế là người ra quân đầu tiên với một bộ sưu tập game đầy uy tín và ấn tượng.

Cũng trong thời gian này, mô hình kinh doanh trong lĩnh vực console dần dần định hình rõ. Nhà sản xuất console xây dựng phần cứng và viết một vài trò chơi đầu tiên (tự làm hoặc hợp đồng với các công ty khác) để trình diễn khả năng của thiết bị. Động thái này sẽ giúp nhà sản xuất bán được một lượng hàng ban đầu. Sau đó, các nhà phát triển game muốn tham gia sẽ phải đăng ký để có quyền viết game cho thiết bị này. Các game do nhà phát triển làm ra sẽ được chính nhà sản xuất console kiểm tra chất lượng kỹ lưỡng. Đây là cách duy nhất để nhà sản xuất

console có được một bộ sưu tập game có chất lượng. Điều này sẽ giúp các game thủ hài lòng với số tiền mình đã bỏ ra để mua console. Các nhà phát triển sẽ được nhà sản xuất cung cấp tài liệu và hỗ trợ kỹ thuật. Đồng thời họ cũng phải trả cho nhà sản xuất tiền “bản quyền” (royalty fee) để bù đắp cho chi phí sản xuất console. Đối với Nintendo, bất kỳ thiết bị console nào do Nintendo làm ra cũng được “chống lưng” mạnh mẽ bằng một loạt các game chất lượng cao do chính Nintendo tự xây dựng. Vào năm 1988, Nintendo trở thành một thương hiệu đầu đàn với một chính sách hợp tác với các nhà sản xuất game rất chặt chẽ.

Cuộc cách mạng máy tính cũng không đứng yên trong thập kỷ này. Máy Apple II đã vạch đường cho nền công nghiệp. Máy tính với giá phải chăng ngoài việc là một công cụ làm việc hữu dụng còn là một ứng cử viên sáng giá cho thiết bị chơi game tại nhà. Cứ mỗi vòng đời, máy tính ngày càng có phần cứng mạnh hơn, môi trường lập trình dễ dàng hơn và càng phù hợp hơn với thế giới trò chơi điện tử. Sự ra đời của kiến trúc mở của máy IBM-PC vào năm 1981 đã tạo nên cuộc cách mạng đối với nhiều nhà sản xuất thiết bị ngoại vi. Các card đồ họa và ổ cứng bắt đầu được sản xuất đại trà bởi các nhà sản xuất khác nhau, cạnh tranh gay gắt trên từng phân khúc thị trường. Sự thay đổi này rất kinh khủng: chỉ trong vòng 5 năm, những card màn hình CGA thô sơ chỉ có 4 màu đã được thay thế bằng các card VGA 256 màu với độ phân giải 320x200. Chính VGA đã khiến cho các trò chơi trên máy tính trở nên cực kỳ phổ biến vào cuối thập kỷ 1980. Nó cho phép các trò chơi thể hiện hình ảnh trên toàn màn hình với độ phân giải và số màu không hề thua kém hình ảnh từ các máy console. Các game “vượt thời gian” như Day of the Tentacle của Lucas Arts hoặc dòng game “Alone in the Dark” là những đại diện tiêu biểu của thời kỳ VGA.

1.1.5. Sự ra đời của Game Engine

Ngay khi khái niệm hệ thống file xuất hiện trong các máy tính cá nhân, nó nhanh chóng trở thành một phương tiện cực kỳ thích hợp để tổ chức, sắp xếp các dữ liệu của trò chơi. Đã qua rồi thời kỳ một tập tin

ôm đồm cả mã chương trình và dữ liệu làm cho việc kiểm soát và theo dõi dự án trở nên cực kỳ khó khăn. Với một hệ thống tập tin tốt, mã trò chơi có thể được lưu trong tập tin thực thi, và dữ liệu game sẽ được lưu trữ trên những tập tin khác nhau. Bằng cách này, sự “hỗn loạn” sẽ được kiểm soát, công việc có thể dễ dàng được chia sẻ cho nhiều người hơn, mỗi người chỉ lo nội dung của tập tin mà họ kiểm soát. Lập trình viên chỉ chú tâm vào tập tin thực thi, các họa sĩ thì làm việc trên các tập tin ảnh, Hệ quả tất yếu cho cách tổ chức này (dùng hệ thống file) sẽ nhanh chóng xuất hiện: một tập tin thực thi sẽ có khả năng sử dụng dữ liệu khác nhau từ các tập tin dữ liệu khác nhau. Điều này nghĩa là ta có thể tạo ra các game khác nhau với nội dung khác nhau nhưng cách thức chơi giống nhau. Điều này trông có vẻ hiển nhiên với các game ngày nay nhưng bạn hãy cứ thử tưởng tượng xem điều này có ý nghĩa như thế nào đối với một “thế giới” mà trong đó mọi thứ từ mã chương trình, âm thanh hình ảnh đều được “nhét” chung trong cùng một file. Rõ ràng đây là một hướng đi tự nhiên của ngành công nghiệp. Cũng từ đó, các game engines theo mô hình ở trên cũng xuất hiện ngày càng nhiều vì tính kinh tế của nó so với các game “cứng” ở thời kỳ trước.

1.1.6. Cuộc cách mạng của các thiết bị cầm tay

Máy NES chưa phải là máy trò chơi đầu tiên của Nintendo. Vào năm 1970, công ty này nổi tiếng nhờ một hệ thống trò chơi điện tử đơn giản gọi là Game & Watch. Đây là một thiết bị cầm tay kích thước cỡ một cuộn băng cát sét, sử dụng màn hình LCD rẻ tiền, giá khoảng 10 USD đến 25 USD có cài một trò chơi duy nhất. Trò chơi nổi tiếng Donkey Kong được ra đời trên một hệ thống như vậy. Tuy vậy nhưng Game & Watch đã bán được hàng triệu đơn vị và trở nên cực kỳ phổ biến, đặc biệt là đối với các game thủ nhỏ tuổi vì các em có thể mang nó đi khắp mọi nơi và chơi chung với các bạn mình.

Sự quan tâm của công chúng dành cho Game & Watch dần biến mất khi thị trường console bùng nổ. Các máy cầm tay đơn giản như vậy không thể cạnh tranh với các trò chơi đầy màu sắc và âm thanh trên các hệ

NES và Sega Master System. Tuy nhiên, năm 1989, khi mọi người hầu như đã quên mất Game & Watch, Nintendo cho ra đời một máy console đem lại nhiều lợi nhuận nhất trong lịch sử công ty: máy Nintendo GameBoy. Đó là một máy console cầm tay đen-trắng có thể chơi hàng trăm trò chơi bằng cách thay “băng” y như hệ NES.

GameBoy thành công vang dội ngay từ khi ra đời. Sự việc này chỉ có thể được giải thích phần nào bởi lý do Game Boy có bán kèm trò chơi Tetris. Sản phẩm này nhắm vào các game thủ trẻ tuổi hơn, dùng cùng một chiến lược như Game & Watch. Các trò chơi nổi tiếng trên GameBoy có thể kể là Pokemon và một phiên bản đặc biệt của Super Mario Bros (hái nấm). Bên trong, GameBoy lấy cảm hứng từ thiết kế của NES. Nó sử dụng một CPU 8-bit tương tự như Intel 8080 hoặc Zilog Z80 và được trang bị 8KB bộ nhớ chứa mã lệnh và 8KB cho bộ nhớ video. CPU chạy với tốc độ khoảng 4MHz. Độ phân giải màn hình khoảng 160x144.

GameBoy có thể được xem là tương đối mạnh so với chuẩn năm 1989. Đó là một phiên bản đen trắng thu nhỏ của NES, có thể sản xuất với giá rẻ hơn. Nhưng điều khiến GameBoy vượt lên trên các đối thủ là độ dẻo dai của pin và số lượng khổng lồ các game hay trên hệ máy này. Lý do nữa là việc phát triển game cho GameBoy là một ngành kinh doanh béo bở. Chi phí phát triển cho GameBoy chỉ bằng phần lẻ so với các hệ console khác (đặc biệt là khi PlayStation và N64 ra đời) nhưng giá bán game lại không cao hơn là bao so với các hệ console “lớn”. GameBoy tạo ra lợi nhuận cao cho cả nhà sản xuất thiết bị và các công ty phát triển game.

Trong số các hệ máy chơi game, GameBoy có “tuổi thọ” cao nhất – 11 năm. Sự ra đời của GameBoy Color vào năm 1998 cùng với nhiều game “truyền thống” của Nintendo đã thổi một làn gió mới vào sản phẩm này khiến GameBoy Color phá vỡ vô số kỷ lục. Tuy vậy, ngay cả một sản phẩm lớn như GameBoy cũng phải “già” đi, và do đó, vào năm 2000, Nintendo cho ra đời một thiết bị mới hơn, mạnh mẽ hơn. Máy GameBoy

Advanced (GBA) được thiết kế với mục tiêu thay thế hoàn toàn GameBoy. GBA dùng CPU ARM 32-bit chạy ở tần số 16.7MHz, 32KB RAM, 96KB video RAM và 16KB RAM cho âm thanh. Độ phân giải màn hình là 244x160, thể hiện được 32 768 màu cùng lúc, gần bằng phân nửa của hệ máy SuperNES và không thua kém độ phân giải của máy Apple II.

Thêm vào đó, các “băng” có thể lưu trữ 64MB dữ liệu. Điều này làm cho máy này cách xa hàng năm ánh sáng so với GameBoy. Kết quả của cấu hình phần cứng này là nhiều trò chơi cổ điển huyền thoại như Mario và Legend of Zelda có thể dễ dàng được chuyển sang GBA. Điều này đảm bảo cho Nintendo có một lượng khách ổn định ngay từ đầu.

1.1.7. Hiện tượng điện thoại thông minh

Đáng ngạc nhiên là đối thủ cạnh tranh của GBA lại là một thiết bị không ai ngờ tới. Thiết bị nào được bán với số lượng hàng triệu, giá rẻ và cũng là thiết bị cầm tay? Điện thoại thông minh dĩ nhiên là đáp án!. Ngày nay đã có hàng triệu triệu điện thoại di động trên khắp thế giới với cấu hình ngày càng mạnh hơn. Tỷ lệ giữa số máy điện thoại và số máy di động ngày càng trở nên lớn hơn, cho đến thời điểm cuốn sách này được biên soạn, tỷ lệ đã là 6:1 (6 máy điện thoại di động cho một máy console).

Điện thoại di động thế hệ đầu rất to và nặng. Điều này không mang lại chút lợi ích nào cho việc chơi game. Lúc này, điện thoại di động chỉ đơn thuần là một thiết bị liên lạc, cung cấp thêm chức năng nhắn tin SMS. Tuy nhiên, mỗi một chu kỳ phát triển, điện thoại lại càng trở nên mạnh mẽ hơn với nhiều chức năng hơn vì sự cạnh tranh khốc liệt trên thị trường này. Chẳng có gì đáng ngạc nhiên khi điện thoại di động lại phát triển với tốc độ kinh khủng như vậy khi có quá nhiều nhà sản xuất và cung cấp dịch vụ cùng tham gia thị trường. Trong vài năm, Nintendo vẫn còn “một mình một chợ” khi các nhà sản xuất game trên điện thoại chưa thống nhất được một chuẩn chung. Các máy điện thoại di động từ

châu Âu và châu Mỹ đa phần là khác nhau nên tạo rất nhiều khó khăn cho việc các trò chơi thâm nhập sâu vào thị trường quốc tế đang bị thống trị bởi các máy console di động.

Tuy nhiên, các điện thoại di động trang bị Java bắt đầu xuất hiện vào năm 2000 với cấu hình phần cứng ngày càng phù hợp. Ngày nay, các máy điện thoại di động có 64MB RAM với năng lực xử lý tương đương CPU Pentium không còn là điều bất thường. Bạn có thể làm gì với một thiết bị như vậy? Bạn có thể chơi Quake, Age of Empires và rất nhiều nhiều game khác nữa. Thật ra, chơi game trên điện thoại di động có nhiều lợi thế hơn so với PC hoặc các thiết bị console di động. Đơn giản là vì bản thân điện thoại di động là một thiết bị viễn thông từ trong trứng nước nên việc chơi nối mạng trở nên rất dễ dàng.

Một ví dụ tiêu biểu về sự thành công của trò chơi trên điện thoại di động là dịch vụ I-Mode của NTT DoCoMo, xuất hiện tại Nhật vào năm 1999. Đây là một dịch vụ trả phí hàng tháng trong đó người dùng trả phí để truy cập vào các nội dung khác nhau dành cho điện thoại di động như các ứng dụng tiện ích nhỏ (bản đồ, lịch biểu, ...) và các trò chơi. Vì chi phí quá rẻ nên việc dùng dịch vụ này là cực kỳ hấp dẫn. I-Mode ngay lập tức thành công. Chỉ đến năm 2002, I-Mode có hơn 32 triệu người đăng ký (với mỗi ngày có thêm 28000 tài khoản mới). Yếu tố chính mang đến thành công của mô hình I-Mode là nội dung đặc sắc, mô hình kinh doanh lạ (người dùng trả tiền cho mỗi một lần download thay vì phải trả phí cho từng phút) và chi phí phát triển cực thấp dành cho các nhà phát triển. I-Mode dựa vào các tiêu chuẩn như HTML và Java. Với sự thành công của mô hình I-Mode, nhiều công ty khác cũng nhanh chóng nhập cuộc.

Một ví dụ điển hình cho loại nội dung mà người dùng có thể download từ I-Mode là game Samurai Romanesque, một dạng game nhập vai thời trung cổ. Điểm đặc sắc là dựa vào dịch vụ thời tiết, trò chơi có thể cảm nhận được tình hình thời tiết ngay tại nơi người chơi đang đứng. Nếu trời đang mưa, thuốc súng của người chơi trong thế giới ảo sẽ bị ướt và

do đó, người chơi sẽ không dùng súng được. Đây là một ví dụ chứng minh lợi thế độc quyền của điện thoại di động so với các thiết bị chơi game khác. Nó có thể truy cập được tất cả các thông tin trực tuyến và cả vị trí của người chơi.

1.1.8. Game nhiều người chơi

Game nhiều người chơi có mặt từ lâu hơn bạn vẫn biết. Game đầu tiên đã xuất hiện từ năm 1980 và sau đó nhanh chóng được điều chỉnh để thích ứng với sự xuất hiện của Internet và World Wide Web vào giữa thập kỷ 1990. Game nhiều người chơi có đầy đủ đặc tính của những game chơi đơn đồng thời có thêm sự hấp dẫn đặc biệt của việc chơi chung hoặc đối kháng với những người chơi khác. Nhiều hình thức chơi nhiều người cũng ra đời: chơi phối hợp đồng đội (squad-based) được giới thiệu đầu tiên trong game Essex Multi-User Dungeon (MUD) hoặc chơi đối kháng được giới thiệu trong game Multiple User Labor Element (M.U.L.E) do hãng EA phát hành trong năm 1983.

Lý do khiến game nhiều người chơi thành công nằm ở chỗ việc chơi cùng hoặc chống lại trí thông minh nhân tạo của máy tính là không thể so sánh được với một con người khác. Hơn nữa, game nhiều người chơi còn mang lại tính cộng đồng và xã hội, một yếu tố không thể tồn tại trong game một người chơi.

Ví dụ tiêu biểu về game nhiều người chơi là Essex MUD được phát triển bởi Richard Bartle và Roy Trubshaw tại trường Đại học Essex vào năm 1979. Trò chơi ban đầu bao gồm 20 phòng được mô tả bằng từ ngữ. Tại một thời điểm, có thể có tới đa đến 250 người chơi trong thế giới của Essex MUD. Những người chơi này có thể “thấy” nhau dựa vào các câu mô tả như “Peter đang ở đây”. Các game thủ nối vào máy chủ MUD thông qua mạng EPSS - mạng này kết nối 6 trường đại học của Anh quốc. Từ năm 1980, các game thủ kết nối đến MUD server thông qua mạng ARPA (tiền thân của Internet). Các game hiện đại ngày

nay có thể “chứa” vài ngàn người chơi cùng lúc trên cùng một máy chủ (ví dụ như Planetside của Sony có thể chứa 3500 trên một máy).

Essex MUD đã tạo niềm cảm hứng lớn cho nhiều nhà phát triển khác. Thực tế là cộng đồng MUD đã hoạt động rất tích cực từ những năm 1980 cho đến giữa thập kỷ 1990. MUD dần đi vào quên lãng với sự xuất hiện của các game nhiều chơi trực quan như Everquest.

Tuy thành công nhưng MUD không phải là dạng game nhiều người chơi duy nhất. Các dạng game đối kháng đơn giản giữa người chơi với nhau cũng tạo nên nhiều kịch tính cho người chơi. Ví dụ cho thể loại này là game M.U.L.E cho phép 4 người chơi cùng lúc có thể tranh giành nhau để chinh phục một hành tinh mới trong dãy ngân hà. Dạng này không khác mấy so với dạng game chiến thuật ngày nay. Nhưng đáng tiếc là thành công của game này cũng khá hạn chế. Lý do chính là hạ tầng mạng vào những năm 1983 chưa phù hợp với hình thức game thời gian thực.

Điểm sang trang quan trọng đến từ năm 1993, khi Internet và đặc biệt là WWW bùng nổ. Trong vòng chưa tới 6 tháng, WWW từ con số không đã xâm lăng toàn bộ hành tinh như vũ bão. Một hệ quả tất yếu là tốc độ kết nối ngày càng được cải thiện. Từ các modem “rùa bò” 9600bps, đến 56Kbps rồi đến ISDN, DSL và cuối cùng là cáp riêng. Tốc độ mạng tăng rất có ý nghĩa đối với game nhiều người chơi vì nó cho phép trò chơi chuyển tải nhiều thông tin hơn giữa các máy tính từ đó làm cho game chạy “mượt mà” hơn. Ví dụ như Age of Empires, có thể chơi thoải mái bằng modem 56kbps với hàng chục loại đơn vị khác nhau chiến đấu cùng lúc.

Các game nhiều người chơi ngày nay đã ổn định xung quanh 2 quỹ đạo: dựa trên MUD hoặc M.U.L.E. MUD cho vạch ra con đường cho các trò chơi nhập vai trực tuyến như Ultima Online, Everquest, Asheron’s Call và nhiều game khác. Các game này cho phép game thủ đắm chìm trong một thế giới ảo rộng lớn. Hàng ngàn người chơi đã thực sự sống cuộc

sống ảo của họ tại một trong số những game này. Thực tế cho thấy đây có lẽ là loại game dễ gây nghiện nhất. Những “tín đồ” trung thành có thể dành hơn 1600 giờ chơi game trong một năm (nghĩa là hơn 4 giờ mỗi ngày, kể cả ngày Chủ Nhật). Và hiệu quả tất yếu, xuất phát từ Ultima/Everquest là sự khai sinh ra nền kinh tế ảo. Các đồ vật trong game được rao bán trên các website với một số tiền không nhỏ. Các hoạt động này làm chúng ta phải suy nghĩ về ảnh hưởng của game đến cuộc sống thật của con người.

Dạng khác của game nhiều người chơi cũng khá thành công là dạng game nhiều người chơi thu gọn theo kiểu của M.U.L.E. Theo dạng này, các người chơi phải phân biệt rạch ròi với nhau ai là bạn ai là thù. Một game cực kỳ nổi tiếng (kể cả ở Việt Nam) theo dạng này là CounterStrike, một game được hiệu chỉnh từ game Half-Life. Trong CounterStrike, người chơi lập thành hai phe: cảnh sát và khủng bố chiến đấu với nhau. Người chơi vừa chơi đối kháng (đối kháng với phe kia) lẫn hợp tác (hợp tác với các người chơi thuộc phe mình).

Tương lai của game nhiều người chơi rất tươi sáng. Hạ tầng mạng ngày càng phát triển. Hầu hết người chơi ngày nay ít nhiều đều có kết nối tốc độ hợp lý. Nhiều công ty phần mềm (middleware) đã bắt đầu xuất hiện cho phép tạo ra các game lớn hơn và hấp dẫn hơn. Thêm vào đó, các hình thức chơi mới xuất hiện mỗi ngày. Nhiều người lạc quan còn cho rằng các game nhiều người chơi sẽ “hất cẳng” tất cả thể loại game một người chơi. Mặc dù đây là ý kiến cực đoan nhưng nó cho thấy người ta kỳ vọng thế nào vào thể loại game nhiều người chơi. Nói gì đi nữa, các loại game một người chơi sẽ không thể biến mất vì các dạng trò chơi một người chơi và chơi đồng đội ngoài đời thật vẫn tồn tại trong hòa bình.

1.1.9. Tương lai ở phía trước

Công nghiệp game đang trải qua một giai đoạn chuyển mình quan trọng. Sự xuất hiện của PlayStation 3 và Microsoft XBOX 360 (và gần

đây là PS4 và XBOX One) đã mở ra một kỷ nguyên mới. Các game với chất lượng đẹp như phim có vẻ chỉ còn cách tay ta vài năm nữa thôi. Các máy console trong tương lai sẽ không chỉ bị giới hạn trong chức năng chơi game. Nó rồi sẽ bao gồm cả dàn âm thanh, TV, video tương tác, duyệt Web, thương mại điện tử ... tất cả chỉ trong một chiếc máy cầm tay “đơn giản”, giá cả phải chăng và dễ dùng. Các hệ console sẽ biến thành một hệ thống giải trí gia đình hoàn thiện với đầy đủ chức năng.

Ở mặt khác, thị trường thiết bị di động cũng đang bùng nổ. 3 đối thủ truyền kiếp (điện thoại di động, máy tính bỏ túi và máy chơi game di động) có lẽ sẽ được hợp nhất thành một. Lý do đơn giản là vì người dùng sẽ chẳng muốn đem cả 3 thứ cùng lúc trong khi chức năng của loại nào cũng na ná nhau. Trong tương lai, liệu thiết bị tổng hợp này hay một loại thiết bị mới sẽ chiếm lĩnh thị trường? Mặc dù khả năng thứ hai khó có thể xảy ra nhưng điều bất ngờ vẫn chờ đón chúng ta ở những ngõ ngách ít ai chú ý đến.

Sự thành bại của bất kỳ thiết bị nào rốt cuộc vẫn do người tiêu dùng quyết định.

1.2. NHÂN LỰC VÀ VAI TRÒ TRONG PHÁT TRIỂN GAME

1.2.1. Chiến lược gia (vision)

Bất kỳ dự án nào cũng phải có một nhà hoạch định chiến lược – chiến lược gia. Đây không phải là một chức vụ mà người ta lúc nào cũng có thể tìm thấy trong sơ đồ tổ chức của một công ty. Đó đúng hơn là một công việc thường được giao cho người thiết kế game (game designer) nhưng đôi khi cũng rơi vào tay producer (người chịu trách nhiệm chính của toàn bộ dự án game), trưởng nhóm kỹ thuật (tech lead) hoặc giám đốc nghệ thuật (art director). Đôi lúc công việc này được đảm nhiệm

bởi hai người khác nhau nhưng luôn luôn làm việc chặt chẽ với nhau. Tuy vậy, đây vẫn là một tình thế tế nhị và thường nên tránh.

Nhà chiến lược tổng thể là người, do đã từng trải qua nhiều kinh nghiệm thương đau của việc phát triển game, biết cách phối hợp tất cả các công việc riêng rẽ và nắm bắt được cách thức người chơi “thường thức” một trò chơi. Mặc dù không phải là chuyên gia trong bất kỳ một công đoạn phát triển game nào, nhà chiến lược phải nắm vững cách vận hành của tất cả mọi công đoạn. Nhà chiến lược không nhất thiết phải là một lập trình viên nhưng anh ta/cô ta ít nhất phải hiểu được sự ảnh hưởng của những vấn đề kỹ thuật lên trên các điều kiện của dự án. Nhà chiến lược không cần phải là một họa sĩ nhưng phải hiểu cách thức phân chia công việc trong việc giữa các họa sĩ để tạo nên một hình ảnh tổng thể hấp dẫn trong game. Nhà chiến lược cũng không cần là người thiết kế game nhưng anh ta/cô ta phải cảm nhận được thế nào là một game thú vị. Nhà chiến lược cũng không cần phải là một nhà tâm lý học nhưng phải là một người biết đối nhân xử thế để có thể “lôi kéo” nhiều con người với nhiều cá tính, sở thích khác nhau cùng làm việc để đạt đến một mục tiêu chung.

Nhà chiến lược là một kim chỉ nam và cũng là người gác cổng qua đó mọi ý tưởng mới được sàng lọc. Anh ta/cô ta là “quan tòa” quyết định cái gì nên cho đi và nên giữ lại. Nếu anh là nhà chiến lược của dự án, cần phải biết rất chắc chắn rằng cái gì là cốt lõi khiến cho trò chơi của anh thành công. Đó là tập các chức năng tối thiểu và không thể thiếu trước khi được chính thức phát hành trò chơi của mình. Trong suốt quá trình phát triển, sẽ có hàng ngàn ý tưởng xuất hiện và được “cầu xin” để được đưa vào game. Cùng lúc, thời gian và kế hoạch sẽ tạo áp lực buộc phải bỏ bớt một hoặc nhiều chức năng trong game. Lúc này, mọi việc sẽ tùy thuộc vào nhà chiến lược để quyết định rằng một ý tưởng mới sẽ có giá trị, vô giá trị hay thậm chí là phản giá trị (làm mất đi những giá trị cơ bản của game đang được xây dựng).

Có lẽ sẽ dễ dàng hơn để “chống” lại sự cám dỗ của việc đưa ý tưởng mới vào game khi biết rằng hầu hết các game hay nhất ngày nay thường không có nhiều chức năng nổi đình nổi đám tăn mác mà chỉ tập trung như tia laser vào những gì họ làm tốt nhất. Một nhà thiết kế hàng đầu đã phát biểu “Nếu bạn muốn bán được game, hãy học hỏi từ Mario64, Half-Life và Diablo cách họ tinh giản đến mức tối đa tập chức năng của game cũng như cách họ thể hiện rất sâu sắc và tinh tế tập chức năng này”.

Khi đánh giá một ý tưởng mới, hãy luôn nhớ rằng, “mục tiêu cao nhất chính là trò chơi”. Nguồn gốc của ý tưởng không quan trọng. Nếu ý tưởng này có giá trị đối với game, chúng ta nên chấp nhận nó cho dù nó từ một nhân viên kiểm tra chất lượng (tester), cháu của chính mình hay từ sếp lớn của mình. Hãy trải lòng với những cách khác nhau để đạt được cùng một mục đích. Nếu có ai đó tranh cãi dữ dội để loại bỏ hoặc thay đổi một chức năng mà nó không ảnh hưởng lắm đến giá trị gốc thì chiến lược gia cũng nên linh động. Game sẽ tốt hơn nếu chúng ta biết tôn trọng những ý kiến từ chính những người làm ra nó. Tuy nhiên, nếu một chức năng chính đang bị mọi người tấn công, phải sẵn sàng chuẩn bị đầy đủ “vũ khí” để bảo vệ nó.

1.2.2. Đội ngũ quản trị sản xuất- Production

Có hai công việc khác nhau thường được gọi bằng cái tên *producer*. Công việc đầu tiên là *external producer*. Đây là người làm việc cho nhà phát hành game và bao quát công việc của một đội phát triển ngoài công ty phát hành game. Công việc thứ hai gọi là *internal producer* là nhân viên của chính đội phát triển. Người này sẽ quản lý đội ngũ phát triển và “giới thiệu” nó cho thế giới bên ngoài (bao gồm giới thiệu cho ban quản lý của nhà xuất bản cũng như các bộ phận marketing, quan hệ công chúng –PR và bộ phận bán hàng. Những người này cũng được gọi là *project manager* (trưởng dự án), *project lead* (lãnh đạo dự án) hoặc đơn giản là (director) giám đốc.

Dù thuộc loại “nội” hay “ngoại”, producer luôn là nhân vật quan trọng của dự án game đối với phần còn lại của công ty. Đây là người sẽ giải thích những điểm nổi bật của trò chơi cho bộ phận PR, marketing và bán hàng. Anh ta nắm được tại sao game phù hợp với mục đích của công ty và có thể giải thích cho các ban bộ khác tại sao cần phải phát triển game này. Đây cũng là người kết nối toàn bộ đội ngũ của game. Anh ta minh họa và báo cáo tình hình của game (đang nhanh hơn hay chậm hơn dự kiến, các vấn đề đã xảy ra và cách giải quyết chúng) tại các buổi họp.

Producer đồng thời cũng là nhân vật quan trọng trong đội ngũ quản lý đối với nhóm phát triển. Anh ta giải thích cho nhóm phát triển biết tại sao game đang làm phù hợp với chiến lược của công ty. Anh ta cũng cập nhật thông tin cho nhóm phát triển về tình hình PR, kế hoạch marketing đang diễn ra cho game. Producer cung cấp cho nhóm phát triển các tài nguyên cần thiết để làm việc (phần mềm, phần cứng và các trang thiết bị khác). Anh ta cũng thường xuyên cập nhật tình hình công ty cho nhóm phát triển.

Một công việc khá quan trọng đối với producer là việc theo dõi các mối đe dọa đối với dự án (risk management). Việc phát triển game không bao giờ trơn tru. Lúc nào cũng sẽ có vấn đề, cả từ bên trong lẫn bên ngoài. Đối thủ cạnh tranh có thể có một chức năng mới, văn phòng của nhóm trở nên quá chật chội, trưởng nhóm lập trình yêu một cô nhân viên kiểm tra chất lượng sản phẩm. Về mặt nguyên tắc, hàng ngàn sự cố có thể xảy ra nên việc một vài trong số đó thực sự xảy ra là không thể tránh khỏi. Công việc của producer không phải là tránh không cho các vấn đề xảy ra (điều này là nhiệm vụ bất khả thi) mà đối diện với nó một cách thông minh để giảm thiểu thiệt hại do nó gây ra.

Vì vậy, producer định kỳ phải liệt kê ra những điều tồi tệ có thể xảy ra đối với dự án, cố gắng nghĩ cách để tránh chúng không xảy ra và lên kế

hoạch ứng phó khi nó thực sự xảy ra. Danh sách các mối đe dọa bao gồm các khả năng như:

- Phải làm gì khi bộ công cụ phát triển không đến tay nhóm phát triển đúng thời hạn?
- Phải làm gì khi người phát triển engine không hoàn thành một chức năng chính đúng thời hạn?
- Phải làm gì khi không thể tuyển được một nhân viên nòng cốt?
- Phải làm gì khi công đoạn lập trình trí tuệ nhân tạo cho các nhân vật kéo dài hơn kế hoạch?

Danh sách này không nên bao gồm các khả năng như:

- Phải làm gì khi tòa nhà bị thiên thạch rơi trúng?
- Phải làm gì khi nhà xuất bản sập tiệm?
- Phải làm gì khi công ty bị cúp điện trong tuần cuối cùng trước hạn chót?

3 khả năng cuối là các mối đe dọa không thể giải quyết được. Producer không cần phải làm gì để giảm thiểu nguy cơ xảy ra của các mối đe dọa này cũng như không cần phải lên kế hoạch ứng phó. Nói tóm lại, Producer không nên tốn thời gian cho những mối đe dọa kiểu này.

Producer ngoại

External producer là người (nhân viên) của công ty phát hành game, chịu trách nhiệm quản lý công ty phát triển game nhằm đảm bảo game hoàn tất đúng thời hạn, trong ngân sách cho phép và có đầy đủ tính năng cần thiết. Người giữ vai trò này (cũng như vai trò phụ tá hoặc producer tập sự) phải theo dõi tiến độ dự án, phê duyệt các chi phí cho đội ngũ phát triển và nhiều việc khác - nói chung đảm bảo cho đội ngũ phát triển được chăm sóc tốt để yên tâm làm việc.

Một external producer thường phải làm việc với nhiều dự án cùng lúc. Trong điều kiện như vậy, nghệ thuật là phải tìm cách sao cho các dự án không vào cao điểm cùng lúc. Giai đoạn cuối của các dự án thường đòi

hỏi sự tập trung cao độ của mọi người. Do đó, nếu cùng lúc cả hoặc nhiều game cùng vào cao điểm thì anh ta sẽ không đủ thời gian "chăm sóc" tất cả cùng lúc.

Tuy vai trò này không yêu cầu người này phải có tầm nhìn chiến lược nhưng người này nhất định phải đảm bảo game làm ra đáp ứng được mục tiêu mà công ty đặt ra. Nếu người này phải đứng giữa sự chọn lựa về tiền, thời gian hoặc tính năng, external producer phải dành nhiều công sức để thảo luận với tất cả mọi người. External producer cần biết tính năng nào là sống còn cho sự thành công của game và tính năng nào mang tính bổ trợ (có thì tốt, không có cũng không sao - nice to have). Cần phải biết rõ công ty cần game xin hay cần game ra mắt đúng thời điểm.

Khi dự án mới bắt đầu, external producer sẽ có thể phải tham gia quá trình đánh giá và chọn lựa nhà phát triển. Cần tìm hiểu càng kỹ càng tốt về đội ngũ này. Họ đã chinh chiến chung với nhau bao lâu rồi? Họ đã cùng nhau làm những game gì rồi? Các game họ làm có bán chạy không? Game họ sắp làm với bạn có tương tự như những gì họ đã làm trước đây không? Họ có kinh nghiệm ở nền tảng phần cứng đã chọn không? Họ có quen với engine hoặc công cụ nào có sẵn không? Họ có đảm bảo được thời gian phát triển hoặc tuân thủ ngân sách? Ngoài dự án với bạn, họ có đang làm dự án nào khác cùng lúc nữa không?

Thông thường, nếu công ty phát hành game của external producer là nguồn thu nhập duy nhất của nhà phát triển - đó là dấu hiệu không tốt. Nếu họ đứt gánh tài chính ở 3/4 đoạn đường với anh ta, thông thường anh ta phải cắn răng chi thêm tiền hoặc đứng nhìn toàn bộ tiền đầu tư của nhà phát hành vào dự án với họ ra đi không đường trở lại. Ngược lại, nếu công ty phát triển này có nhiều sản phẩm, nhiều nguồn thu nhập, đó là dấu hiệu cho thấy họ đang làm tốt. Anh ta chỉ cần lưu ý đừng để họ "vay mượn" nhân sự từ dự án của anh ta cho những dự án khác.

Sau khi đã chọn lựa xong nhà phát triển và làm việc với họ, external producer này cần phải hiểu rõ họ kỹ càng hơn nữa. Chẳng có cách nào

khác cho anh ta ngoài việc phải thân chinh đến làm việc với những con người này thường xuyên. Hãy nói chuyện và tìm hiểu kỹ từng thành viên của đội ngũ phát triển. Kiểm tra xem họ có đam mê sản phẩm đang làm không hay là họ chỉ làm cho có (trong khi dành tâm huyết cho những dự án nào khác?). Họ có tin vào kế hoạch đã được thống nhất hay không? Hay chỉ đồng ý để anh ta vui lòng mà chẳng quan tâm hoàn thành nó hay không? Họ có biết chia nhỏ công việc thành các mốc nhỏ để đảm bảo hoàn thành các mốc lớn hơn? Họ có tiền để dành hay không hay chỉ sống bữa nào hay bữa đó (cần nhận tiền khi hoàn thành một mốc nào đó để duy trì công ty)

Tuy là người làm công cho công ty nhưng external producer (và công ty của họ) cần đối xử tốt với nhà phát triển: tiếp nhận và phản hồi đầy đủ mỗi khi hoàn thành mốc kế hoạch, trả lời các câu hỏi và thắc mắc nhanh chóng, phản ứng kịp thời trước những yêu cầu hợp lý và quan trọng nhất - chi trả tiền cho họ ! External producer cần họ yên tâm để tập trung làm việc để tạo ra sản phẩm chất lượng từ đó làm giàu cho chính công ty của bạn.

Ngắn gọn hơn, external producer là dầu bôi trơn cho hai bánh răng là công ty phát hành game và nhà phát triển game. Thiếu external producer, mọi thứ sẽ hoạt động khập khiễng và nhiều điều tồi tệ sẽ xảy ra. Người này cần phải trung thực, quyết tâm đứng giữa 2 tổ chức dàn xếp ổn thỏa mọi bất đồng. Nên lưu ý thiết lập những mục tiêu nhỏ nhỏ giữa 2 bên ngay từ đầu để mỗi bên dễ dàng ứng yêu cầu cho bên kia. Nếu có khó khăn xảy ra, những thành công nhỏ nhỏ ban đầu sẽ giúp đôi bên tin tưởng và hỗ trợ nhau hơn.

Ngoài ra, external producer cũng sẽ phải đóng góp không nhỏ vào thiết kế game. External producer đưa ra các hướng dẫn cho team về những quy định của nhà phát hành game - những yếu tố nào được hay không thể được chấp nhận. Cần phải điều chỉnh họ để họ có thể tạo ra những sản phẩm thoả mãn mục tiêu của công ty bạn trong việc tài trợ cho dự án - cho dù các mục tiêu này là để tạo ra các game đỉnh cao hay chỉ để tạo ra doanh số bù lấp cho khoảng trống trong kế hoạch. Tuy vậy, đóng

góp lớn nhất vẫn nằm ở quyết định của bạn trong việc đánh giá tính năng, chi phí và kế hoạch.

Khi làm việc với nhà phát triển, anh ta cũng rất cần lưu ý về những ý kiến, bình luận anh ta dành cho họ. Nếu external producer thực sự muốn điều gì, external producer cần phải nhấn mạnh đủ để họ hiểu đó là cái bắt buộc phải có; ngược lại, khi chỉ muốn đưa góp ý anh ta cũng phải nói thật rõ để họ không phí thời gian đầu tư cho một ý tưởng thoáng qua của anh ta.

Ở chiều ngược lại, external producer cũng phải biết cách lắng nghe nhà phát triển. Dùng miệng đôi lúc dễ gây hiểu nhầm hoặc bỏ sót thông tin. Cách tốt nhất là nên viết xuống những ý tóm tắt và cần tìm cách viết sao cho không thể hiểu nhầm được. Cơ bản là làm sao để khi hữu sự, chúng ta có bằng chứng là "lúc đó, cả anh và tôi đã thống nhất làm thế này mà bây giờ kết quả là thế kia"

Khi có sự thay đổi diễn ra, trao đổi bằng email cũng tạm ổn đối với những vấn đề nhỏ nhưng những vấn đề lớn nên được đưa thành phụ lục của hợp đồng. Cụ thể là bất kỳ điều chỉnh nào ảnh hưởng đến kết quả của một mốc kế hoạch, thời gian giao hàng, thời gian thanh toán đều khá quan trọng và cần phải được cập nhật vào hợp đồng. Các biến động trong kinh doanh là bình thường trong khi các dự án lại thường kéo dài. Chẳng ai muốn đặt mình vào vị trí khó chịu phải đảm bảo cam kết do những người đã nghỉ việc tạo ra. Cuối cùng, external producer là người trả tiền cho nhà phát triển, có quyền lực khá lớn. Đừng lạm dụng nó! Nhiệm vụ của external producer là người hỗ trợ cho nhà phát triển để họ tạo ra những game ngon lành nhất.

Internal Producer

Internal producer sẽ quản lý team phát triển trực tiếp và báo cáo tiến độ dự án cho nhà đầu tư (là công ty đang làm việc hoặc là nhà phát hành bên ngoài).

Một trong những nhiệm vụ đầu tiên vào lúc bắt đầu dự án là làm việc với trưởng nhóm họa sĩ và trưởng nhóm kỹ thuật để tìm đúng người cho game của internal producer. Nếu công ty không có đủ người, cần phải lên kế hoạch tuyển dụng người mới hoặc tìm cộng tác viên (tìm cộng tác viên cho đội ngũ họa sĩ thường dễ hơn so với đội ngũ kỹ thuật)

Nếu không có đủ nhân lực, hoặc là internal producer sẽ tìm cách tăng cường đội ngũ hoặc là internal producer sẽ phải thảo luận lại với đội ngũ thiết kế game để điều chỉnh phạm vi lại cho phù hợp với đội ngũ sẵn có. Đừng cố chịu đấm ăn xôi để lên kế hoạch bắt mọi người phải làm việc 60 giờ / tuần! Mọi kế hoạch nên dựa trên 40 giờ / tuần trong đó phải tính luôn cả thời gian họp hành, demo, ngày lễ, nghỉ phép. Tất nhiên, khó ai tránh được các cao trào nhưng là nhà quản lý, internal producer có trách nhiệm không để mọi người phải "chiến đấu đến hơi thở cuối cùng".

Trong quá trình phát triển, đóng góp của internal producer đối với thiết kế game xuất phát từ những tương tác hằng ngày với dự án. Thiết kế game không phải là một tài liệu bất biến. Ngay cả team dự án cũng không bất biến. Ngành quân sự đã từng phát biểu "không có kế hoạch chiến tranh nào diễn ra 100% đúng dự đoán kể từ khi viên đạn đầu tiên được bắn ra". Điều này có nghĩa là một tướng tài phải biết cách ứng phó với sự hỗn loạn đang diễn ra xung quanh anh ta. Điều này rất đúng đối với producer. Ngay khi quá trình phát triển bắt đầu, mục tiêu là phải dẫn dắt team đi qua "làn khói chiến tranh", giữ mọi người bên nhau, di chuyển về một mục tiêu chung để khi "làn khói" tan đi, mọi người đều đạt được mục tiêu của mình.

Với từng mỗi ý tưởng phát sinh, người này sẽ giúp nhận định độ ảnh hưởng của nó với thiết kế của game và quan trọng hơn là độ ảnh hưởng đến toàn dự án : ai sẽ thực hiện nó? bao nhiêu kế hoạch cần phải được thay đổi? ý tưởng này có giúp mang game đến gần hơn mục tiêu của nhà đầu tư? Trả lời các câu hỏi này sẽ quyết định ý tưởng đó có được chấp nhận hay không. Quá trình sẽ dễ hơn nếu người này là người có vai trò xây dựng tầm nhìn cho dự án. Nếu không phải, internal producer

phải làm việc sâu sát với người có vai trò này để đảm bảo các ý tưởng đưa ra tương thích với định hướng của dự án. Internal producer luôn bị sức ép "vi phạm" từ nhiều hướng khác nhau. Game đến tay người dùng cuối cùng thường là sự tổng hợp từ bản thiết kế ban đầu cộng với vô số những "vi phạm" xảy ra trong suốt quá trình phát triển.

Đến cuối chu kỳ phát triển, internal producer sẽ phải làm việc với trưởng nhóm kiểm tra chất lượng sản phẩm và trưởng nhóm kỹ thuật để tiêu diệt bug! Sẽ phải đánh giá độ nghiêm trọng của từng bug, ước lượng công sức để sửa nó và lường trước khả năng sinh thêm bug từ việc sử dụng bug. Đôi lúc internal producer sẽ thấy mình giống trọng tài đứng giữa việc test và lập trình. Khi sức ép gia tăng thì con người sẽ dễ nóng tính và gây chuyện. Người này cần phải "lạnh lùng" phân xử các tranh cãi và nhắc nhở mọi người hướng đến mục tiêu hoàn tất game.

Cuối cùng, internal producer phải là người quyết định game đã sẵn sàng ra thị trường hay chưa. Không thể có game không có bug, internal producer cần hỏi tất ý kiến từ tất cả các phía để giúp bạn quyết định những tồn đọng còn lại có quá nghiêm trọng hay không. Tuy vậy, internal producer vẫn là người phải nói với mọi người "xong!"

Trợ lý producer hoặc producer tập sự

Nếu một người là trợ lý cho producer hoặc producer tập sự, nhiệm vụ của anh ta sẽ biến đổi tùy theo điểm mạnh và yếu của producer của dự án cũng như của đội ngũ phát triển. Anh ta có thể chuyên làm việc quản lý trong một phân khúc cụ thể nào đó đóng vai trò hỗ trợ chung. Về cơ bản, anh ta sẽ phải sẵn sàng cho những loại công việc khá chi tiết.

Các dạng việc dành cho trợ lý producer bao gồm

Quản lý tài sản: Một khi dự án cũng như đội ngũ phát triển phình to ra, lượng dữ liệu phát sinh ra sẽ bùng nổ. Một dự án game thông thường có hàng trăm ngàn đến cả triệu tập tin cần phải được quản lý. Người quản lý tài sản có trách nhiệm phải theo dõi từng tập tin một! Version mới

nhất của nó là gì ? Tập tin này nằm ở server nào? Tập tin này có bị ghi đè bằng một version trên máy nội bộ hay bị xóa nhầm không? Chuyện gì xảy ra nếu ai đó cập nhật nội dung tập tin nhưng chưa ghi nhận lại trên hệ thống? Nếu tập tin không còn hữu dụng thì nó có được đưa vào build không (để dự phòng) ? Hầu hết các công ty đều có bộ công cụ để làm việc này nhưng việc đảm bảo mọi thứ hoạt động hoàn hảo sẽ cần đến khá nhiều thời gian và công sức của trợ lý producer.

Quản lý các build hằng ngày cũng như backup hệ thống: Khi dự án đang diễn ra suông sẻ, thông thường trách nhiệm của trợ lý producer là phải luôn luôn có một phiên bản chơi được sẵn sàng trên mạng. Anh ta cũng có trách nhiệm đảm bảo một kế hoạch backup định kỳ (ngày, tuần, tháng) toàn bộ hệ thống được nghiêm ngặt thi hành.

Quản lý website chứa tài liệu thiết kế: đội phát triển sẽ nhanh chóng bị ngập mặt trong đồng tài liệu về design nên việc mở một website riêng (dùng hệ thống Wiki) để quản lý thông tin là điều gần như bắt buộc. Điều này sẽ giúp mọi người biết được người khác đang làm gì. Dĩ nhiên là website không tự nhiên xuất hiện và chạy trơn tru nếu không có ai đó (là trợ lý producer) phải nhọc công thu thập, sắp xếp, đăng thông tin lên website chung này.

Tạo screenshots và hỗ trợ hoạt động PR: khi dự án được công bố, gần như ngay lập tức, nhu cầu về screenshot của game sẽ xuất hiện. Tạo ra screenshot để giới thiệu cho công chúng gần như là một "nghệ thuật". Bạn phải khéo léo chọn những góc hình sao cho game trở lên lung linh dưới mắt người chơi game. Thêm vào đó, bạn sẽ phải là tay chuyên đi demo để "show hàng" game cho các nhà báo đến viết bài về dự án của bạn.

Đánh giá các mốc dự án: khi một mốc dự án được team dự án tuyên bố là hoàn thành, anh ta phải kiểm tra xem các yêu cầu của mốc đó có được đáp ứng đầy đủ hay chưa trước khi chấp nhận là xong mốc đó. Việc này đòi hỏi anh ta phải nhúng tay chơi thử game hoặc đọc kỹ các tài liệu liên quan mới có thể đảm bảo chắc chắn mọi thứ đều ổn.

Viết lách và giấy tờ: vai trò trợ lý producer thường phải viết và quản lý hầu hết các giấy tờ (thường có tính chất pháp lý) có liên quan đến dự án và cũng có trách nhiệm gửi nó đến các bên liên quan (như nhà sản xuất console) để giấy tờ được ký duyệt đầy đủ.

Các nhiệm vụ khác: bao gồm hàng trăm việc nhỏ "không tên" đột nhiên xuất hiện trong quá trình phát triển cũng thường được giao cho trợ lý producer. Các việc này có thể là bất kỳ việc gì từ việc khẩn cấp chạy đến phi trường để nhận hàng, gọi điện thoại kêu thức ăn cho team khi team đang cày buổi tối hay thu dọn máy tính để đi demo tại các hội chợ game như E3

Bên cạnh đó, anh ta còn có cơ hội tác động đến thiết kế game thông qua việc anh ta dính líu đến khá nhiều khía cạnh của dự án ở vai trò trợ lý producer. Đóng góp của bạn thường sẽ không mang đánh đập của những quyết định mang tính sống còn hoặc có ảnh hưởng sâu rộng mà thường liên quan đến những vấn đề nhỏ mà team phải gặp hằng ngày. Khi các lập trình viên, họa sĩ những tay thiết kế cảnh (level designer) tạo ra những phần nhỏ của game hằng ngày, bạn sẽ là người đầu tiên được thấy và góp ý cho các phần này. Nếu ý kiến của bạn được tôn trọng, bạn sẽ là người góp phần thay đổi "hình dáng" cuối cùng của game.

1.2.3. Đội thiết kế

Một đội thiết kế chính thức sẽ bao gồm người thiết kế hệ thống, người thiết kế cảnh chơi và người viết kịch bản. Mặc dù mọi người trong dự án đều có thể có ít nhiều ảnh hưởng đến thiết kế game nhưng đây sẽ là nhóm chính tạo nên bản thiết kế "nguyên thủy". Người thiết kế hệ thống (cũng thông thường kiêm luôn vai trò viết kịch bản) sẽ tạo ra tài liệu thiết kế và cập nhật nó trong suốt quá trình phát triển. Anh ta sẽ thiết kế gameplay cơ bản, các cơ chế game chính và cũng là người có vai trò tầm nhìn để sàng lọc các ý tưởng mới để quyết định các ý tưởng này là có lợi hay hại cho game.

Người thiết kế hệ thống sẽ làm việc với một họa sĩ vẽ *story board* - mạch truyện (một hoặc nhiều hình vẽ phát họa sơ lược một cơ chế, một quy trình cụ thể nào đó trong game) để thiết kế ra các đoạn giới thiệu game, kết thúc game và các đoạn phim giữa game (cutscenes). Nếu người thiết kế này cũng đóng vai trò viết kịch bản, anh ta sẽ xây dựng các mẫu đối thoại và viết luôn cả tài liệu hướng dẫn sử dụng. Nếu anh ta không làm điều này, anh ta sẽ đi thuê lại một người chuyên viết lách để làm công việc này.

Người thiết kế hệ thống cũng thường sẽ chỉ đạo nhóm xây dựng cảnh chơi. Anh ta sẽ tạo ra quy trình của game và hướng dẫn các người thiết kế cảnh chơi để tạo ra các phân khúc game phù hợp với quy trình đó. Nhà thiết kế cũng hợp tác với team PR khi họ đang xây dựng website quảng bá cho game; hợp tác với team marketing khi họ tạo ra các mẫu quảng cáo và vỏ hộp đựng đĩa game; hợp tác với nhóm bán hàng khi họ tạo ra các mẫu in ấn, tờ rơi phát cho game thủ. Anh ta cũng phải thiết kế các bản demo cho cả 3 nhóm này để họ quảng bá game cũng như sẵn sàng xuất hiện trước ống kính phóng viên.

Vai trò thiết kế game

Khi một producer đang xây dựng đội ngũ phát triển, anh ta cần phải tìm một người thiết kế có đam mê và tình yêu với công việc anh ta đang làm. Thiết kế là một công việc tinh tế, nếu không yêu nghề, chúng ta sẽ khó tránh khỏi sai sót và chủ quan. Đảm nhận vai trò thiết kế nghĩa là ta phải quyết định game thủ sẽ làm gì trong game của mình. Chúng ta sẽ là "cội nguồn" của niềm vui của game thủ ! Chúng ta có trách nhiệm phải làm game thủ của mình vui vẻ trong suốt quá trình họ chơi game của mình.

Nhà thiết kế, thông thường (nhưng không phải luôn luôn) cũng sẽ là người đóng vai trò chiến lược. Viết ra tài liệu thiết kế rồi xông vào cài đặt ngay thành game là điều hiếm xảy ra. Trong suốt quá trình phát triển, sẽ có hàng ngàn các đề nghị / ý tưởng nhỏ được đưa ra từ tất cả mọi thành viên của nhóm, người này sẽ là người sàng lọc toàn bộ ý

tưởng này. Cần phải so sánh từng ý tưởng một đối với mục tiêu/tâm nhìn ban đầu của game để xem chúng khớp nhau không.

Trong suốt quá trình này, nhà thiết kế phải rất mềm dẻo. Tâm nhìn của nhà thiết kế không phải là thứ bất biến, độc đoán cần phải được thực hiện bằng mọi giá. Thiết kế game luôn đòi hỏi ta phải biết "hy sinh". Mọi nền tảng đều có giới hạn. Nhà thiết kế sẽ luôn luôn đối diện với câu hỏi có thể làm việc này khác đi ít hay nhiều vì một giới hạn nào đó. Phải luôn thực tế và sẵn sàng điều chỉnh tâm nhìn của mình để nó khả thi đối với năng lực của đội ngũ phát triển.

Đừng quyết định chớp nhoáng! Nếu ai đó hỏi nhà thiết kế để thay đổi một tính năng, cần dành chút thời gian để suy nghĩ xem tại sao ban đầu chúng ta lại thiết kế tính năng đó như vậy. Game là một mạng lưới phức tạp các mối ràng buộc. Một số ràng buộc là bắt buộc một số không. Nhà thiết kế sẽ không muốn mình thực hiện một thay đổi mà sau đó vài tháng một phần lớn game của anh ta mất đi mục đích ban đầu của nó chỉ vì một yếu tố đã bị thay đổi.

Nếu yêu cầu thay đổi là hợp lý, chúng ta hãy cố gắng chấp thuận nó dù có thể không thích nó. Một thay đổi nhỏ trong thiết kế game có thể tiết kiệm được rất nhiều công sức lập trình cũng như giảm thiểu được nhiều nguy cơ; thậm chí có thể tiết kiệm được hàng tháng công sức dựng hình 3D và tạo chuyển động của các họa sĩ! Hãy biết lắng nghe các thỉnh cầu từ các thành viên trong đội của mình!

Nếu game có dùng ngôn ngữ kịch bản - *scripting language* (thường là có vì đa số các engine hiện đại đều cung cấp scripting language kèm theo - scripting language là một dạng ngôn ngữ lập trình đơn giản để ta có thể điều khiển các yếu tố trong game rất nhanh mà không cần phải tốn công "chọt" vào mã nguồn chính của game hay biên dịch lại game), nên học nó. Vào thời kỳ game còn được làm bởi một người, người thiết kế game cũng là tay lập trình ra game và ngược lại. Quá trình chuyên biệt hoá đã tạo ra 2 vai trò khác nhau như hiện nay nên việc một người thiết kế không biết tí gì về lập trình là chuyện bình thường. Tuy nhiên,

nếu không biết tí gì, nhà thiết kế sẽ mất đi cơ hội được vọc sâu vào game - một phần rất quan trọng của thiết kế game. Đây là lý do mà các ngôn ngữ kịch bản được tạo ra. Nó cho phép một người thiết kế ít rành kỹ thuật có thể đưa ý tưởng của anh ta trực tiếp vào game và quan sát xem nó có hiệu quả không. Một hệ thống ngôn ngữ kịch bản tốt cũng cho phép các họa sĩ và thiết kế cảnh chơi đạt được mục tiêu tương tự.

Trở trêu thay, một trong những phần khó của việc trở thành người thiết kế game là chơi những game khác. Như mọi ngành nghề khác, bạn phải luôn luôn cập nhật thời thế! Nghe có vẻ dễ nhưng không! Mỗi năm có hàng ngàn game mới xuất hiện. Trong số đó, liên quan đến game của bạn cũng đã hàng trăm. Để chơi đầy đủ một game ngày nay phải dành từ 20 giờ đến hàng trăm giờ. Nhà thiết kế sẽ khó thu xếp đủ thời gian để chơi hết các game này trong lúc phải thiết kế game của chính anh ta.

Không có một giải pháp hoàn hảo cho vấn đề này nhưng cũng có vài cách hữu hiệu để tấn công nó.

Một là tìm cách bỏ qua các game không quan trọng bằng cách đọc các bài đánh giá. Các tay đánh giá game trên các tạp chí thường bỏ ra hàng ngàn giờ để chơi game rồi nên nhà thiết kế không cần phải làm thế nữa. Nếu nhà thiết kế chịu khó dùng những tạp chí này thường xuyên, nhà thiết kế sẽ luôn nắm bắt được xu thế của những game có liên quan đến game của anh ta.

Mỗi tháng, các tạp chí game thường kèm theo các đĩa CD với hàng đồng demo game. Nhà thiết kế cũng có thể download các bản demo chơi thử từ Internet từ các website game. Các bản demo chơi thử thường thu hẹp trải nghiệm game so với bản đầy đủ nhưng vẫn thể hiện được đầy đủ "linh hồn" của game. Đây chính là điều người thiết kế muốn. Nếu anh ta đang có ý định làm một tính năng mới mà chưa biết thiên hạ đã làm nó hay chưa, chỉ cần duyệt qua các bản demo hoặc bản chơi thử trước (previews) là nhà thiết kế có thể có ngay câu trả lời.

Ba là nhìn vòng vòng trong văn phòng :) Nếu nhà thiết kế đang làm cho một công ty phát triển game, thường thì nhà thiết kế sẽ thấy các đồng nghiệp chơi game mỗi ngày. Một số chơi vào giờ nghỉ trưa, một số chơi lúc nghỉ giải lao và hầu như ai cũng chơi vào cuối ngày làm việc. Đây là cơ hội vàng cho nhà thiết kế vì người ta đã làm giúp nhà thiết kế việc cài đặt game, học cách chơi game. Hãy đứng sau lưng và quan sát họ. Hỏi họ là họ đang chơi gì và tại sao. Hỏi anh ta là anh ta thích hay không thích chơi game này (hay một tính năng cụ thể nào đó) vì cái gì.

Kết quả của 3 bước trên là nhà thiết kế sẽ có một danh sách game ngắn hơn rất nhiều mà anh ta thực sự cần phải chơi để tìm hiểu. Bước cuối cùng là đừng cảm thấy tội lỗi khi chơi game (trong khi đồng nghiệp bạn đang cày cuốc :)). Đây là một phần công việc của người thiết kế game để biết được cái gì là vui. Nếu nhà thiết kế ngừng chơi game hoàn toàn, cảm nhận của nhà thiết kế về niềm vui trong game sẽ biến mất! Không cảm nhận được niềm vui thì làm sao nhà thiết kế mang lại niềm vui cho người khác?

Thiết kế cảnh chơi

Thiết kế cảnh chơi là một lĩnh vực còn mới mẻ trong ngành công nghiệp. Trong những ngày đầu của ngành công việc, nghề này chưa tồn tại và cũng chẳng có tài liệu nào nói về nghề này! Trước kia, đây là nghề dành riêng cho một số tay nghiệp dư có tài nhưng giờ nó đã trở thành một vị trí then chốt trong nhiều đội phát triển game.

Công việc cơ bản của người thiết kế cảnh chơi là xây dựng nên các giai đoạn nhỏ của game - tuân thủ theo các quy trình cơ chế gameplay có sẵn - sao cho game thủ cảm nhận được game một cách trọn vẹn nhất có thể. Nhà thiết kế cảnh chơi sẽ giống như một tay đầu bếp vậy - với một số nguyên liệu có sẵn, bạn phải biết nấu và sắp xếp từng món từ khai vị, món phụ, món chính, tráng miệng sao cho thực khách của bạn cảm thấy ngon miệng và thỏa mãn nhất.

Đầu tiên anh ta sẽ được sắp xếp làm việc trong một môi trường mở thay vì trong một văn phòng kín. Điều này giúp mọi người chia sẻ ý tưởng nhanh chóng và giữ mọi người đồng bộ với nhau. Môi trường mở cũng ta ra không khí thân thiện - vừa có lợi mà cũng có phần hại. Vấn đề thông thường là anh ta đôi khi bị phân tâm khi đồng nghiệp xung quanh của anh ta làm ồn. Đầu tư cho một cặp headphone là hành động khôn ngoan!

Một điểm kỳ lạ ở vị trí của nhà thiết kế cảnh chơi là đôi khi được "nhờ" đi demo game. Đây không phải là công việc thường nhật nên cần tập luyện và chuẩn bị. Nhiều lúc nhà thiết kế cảnh chơi sẽ phải xuất hiện để chạy game trong văn phòng chủ tịch công ty, các nhà đầu tư tiềm năng, những người đánh giá game đến từ các tạp chí game hay trước đám đông tại một hội chợ (đôi khi trong đó có một nhân vật khá quan trọng mà chúng ta không hề biết!)

Ngay cả khi nhà thiết kế cảnh chơi thuộc tuýp người kín đáo và chỉ thích làm việc trong 4 bức tường suốt ngày với cái máy tính, một khi đã chọn nghề này, cần phải rèn luyện kỹ năng giới thiệu game và nói chuyện trước "công chúng" - ít nhất đủ để tóm tắt những tính năng chính và trình diễn những tính năng "độc" của game. Nhà thiết kế cảnh chơi không bắt buộc phải đi học khoá về kỹ năng thuyết trình nhưng cũng nên lên kế hoạch demo 5 phút, 20 phút và chi tiết hơn nữa.

Viết lách

Nếu chỉ là một freelancer hoặc chỉ đóng vai trò viết lách mà không phải là người thiết kế game, có thể người này sẽ không cần phải toàn thời gian ở dự án game. Anh ta chỉ cần xuất hiện lúc cần thiết ở một số giai đoạn và công việc nhất định. Các loại việc này bao gồm: lời thoại của nhân vật, lời bình luận của phóng viên (trong các tựa game thể thao), lời dẫn trong các cảnh cutscene, tài liệu hướng dẫn sử dụng, hệ thống gợi ý trong game (như trong hệ thống nhiệm vụ) hoặc bất kỳ đâu trong game cần đến câu chữ.

Khi mới tham gia dự án, điều quan trọng là anh ta cần phải làm việc sát sao với người thiết kế game để có cái nhìn toàn diện về game. Người này sẽ phải đọc toàn bộ tài liệu thiết kế game, nói chuyện với những thành viên trong team để hiểu rõ mục tiêu của game. Ngay cả khi quay lại dự án sau một thời gian tạm ngưng (vì anh ta không làm toàn thời gian cho dự án) - anh ta cũng ít nhiều cần lặp lại quá trình này - vì mọi thứ có thể thay đổi. Bỏ qua quá trình "đồng bộ" chính mình với dự án là khá nguy hiểm vì khi thiếu thông tin, anh ta có thể viết ra những câu chữ không gắn kết, rời rạc hoặc tệ hơn là hoàn toàn lạc lõng với game.

Viết lách cho game hơi khác so với viết lách cho báo chí, dưới đây là một số điểm người viết nội dung cho game cần lưu tâm

- Người chơi có thể đi qua một phần của game nhiều lần. Điều này có nghĩa là nếu phần đó của game có liên quan đến chữ nghĩa, anh ta phải đọc đi đọc lại câu chữ của người viết nhiều lần. Do vậy, đừng làm phiền người chơi! Ngay cả một câu chữ thú vị cũng trở nên phiền phức khi đọc lại cả chục lần. Người viết lách cần luôn lưu ý đến tính lặp này và thảo luận với lập trình viên để tạo ra các biến thể khác nhau.
- Các game thủ khác nhau sẽ đi theo những con đường khác nhau trong game, do đó không ai sẽ tiếp nhận thông tin theo thứ tự giống hệt nhau. Nếu có thể, người viết cần cố gắng tách các câu chữ thành từng đoạn rời rạc sao cho khi đọc các đoạn này theo thứ tự nào cũng hiểu được.
- Cố gắng giữ cho các đoạn đối thoại ngắn gọn. Game thủ không thích nghe đối thoại nhiều. Họ muốn hành động trong game (nếu game thủ thích nghe đối thoại họ đã đi xem phim rồi :))
- Tương tự, cố gắng giữ các đoạn thông tin trong game ngắn. Nếu không thể làm ngắn đi được, hãy tìm cách đưa thông tin đến với game thủ theo từng đoạn một thay vì quẳng hết tất cả câu chữ

lên màn hình cùng lúc. Một lần nữa, game thủ muốn hành động chứ không phải muốn đọc. Nếu muốn đọc họ đã đi vào thư viện.

- Ngữ điệu và chọn lựa từ vựng là quan trọng. Người viết lách cần phải thống nhất ngữ điệu của nhân vật hoặc môi trường và cần phải chọn lựa từ vựng phù hợp với đối tượng người chơi. Chúng ta không thể làm một game cho teen mà sử dụng bộ từ vựng của các vị phụ huynh nghiêm nghị !

1.2.4. Đội ngũ kỹ thuật

Game cũng là một sản phẩm phần mềm nên vai trò của đội ngũ kỹ thuật khá tương đồng với đội ngũ của một team phần mềm. Tuy nhiên, cũng có một số điểm cần lưu ý như sau.

Trưởng nhóm kỹ thuật

Là trưởng nhóm kỹ thuật, người trưởng nhóm thường tham gia dự án rất rất sớm cùng với producer, thiết kế game và trưởng nhóm mỹ thuật. Một trong những nhiệm vụ ban đầu là tạo cảm hứng cho cả nhóm về những gì ta có thể làm được, một mặt giữ lại những mong muốn bất khả thi, mặt kia tìm ta những việc hào hứng, cách mạng. Cố gắng đừng chọn hơn 2 mối nguy kỹ thuật chính trên mỗi dự án. Tuy năng lực của chúng ta (và của đội ngũ làm game) có thể đủ "du hành tới mặt trăng" hoặc hơn thế nữa nhưng ai cũng có nguy cơ đi trượt mục tiêu (trễ kế hoạch, vượt kinh phí ...)

Hãy ước lượng nền tảng mà game sẽ chạy và xây dựng một kiến trúc game tận dụng được tối đa sức mạnh của nền tảng, bù đắp lại những điểm yếu của nó. Nếu trưởng nhóm kỹ thuật dự định làm game đa nền tảng, hãy lên kế hoạch cho những tính năng đặc biệt chuyên biệt trên từng nền tảng để tận dụng tối đa sức mạnh đặc biệt của nền tảng đó.

Trong quá trình preproduction (tiền sản xuất), trưởng nhóm kỹ thuật sẽ xây dựng kế hoạch kỹ thuật bao gồm những công việc chính và ước lượng chi phí nhân công và thời gian cần thiết để hoà tất. Khi đưa ra kế hoạch này, anh ta phải ý thức rằng nó không thể chính xác tuyệt đối và

cần phải nắm rõ được các điểm "tương đối" trong kế hoạch của mình để điều chỉnh sau này.

Trong quá trình preproduction, trưởng nhóm kỹ thuật cũng sẽ xây dựng đội ngũ của mình - bằng cách tuyển mới từ bên ngoài hoặc chọn từ những đội ngũ khác. Xây một đội ngũ mạnh là một trong những nhiệm vụ rất quan trọng nhưng thường bị xem nhẹ trong phát triển game. Tìm được người tốt (giỏi và phù hợp) rất khó - tốn kém cả thời gian và tiền bạc. Sẽ tốn khá nhiều thời gian để sơ duyệt ứng viên bằng các bài test, chấm bài rồi phỏng vấn những ứng viên có tiềm năng.

Trưởng nhóm kỹ thuật cũng cần lưu ý trang bị đủ đồ nghề làm việc cho team của mình. Các nhân viên lập trình game thường cần những cỗ máy tính mạnh mẽ để làm việc - thậm chí là mạnh nhất trên thị trường. Cũng đừng quên nâng cấp trang bị làm việc cho đội ngũ của mình vì dự án có thể kéo dài vài năm, các thiết bị hiện đại ban đầu có thể dễ dàng lỗi thời sau 1-2 năm. Ngoài máy tính ra, cơ sở hạ tầng mạng, hệ thống lưu trữ và sao lưu cũng cần được đầu tư đúng mức.

Một việc quan trọng nữa là cần đánh giá các công nghệ cần thiết cho game và quyết định nên tự xây dựng hoặc mua lại từ bên ngoài. Công nghệ không chỉ giới hạn trong game engine mà còn liên quan đến các bộ công cụ mà mọi người dùng trong suốt quá trình phát triển. Trưởng nhóm kỹ thuật cũng nên ưu tiên cho việc mua những công cụ có sẵn để tăng tốc quá trình phát triển thay vì phải tự xây dựng. Ý tưởng chung là dùng team này vào việc mà họ giỏi nhất và thích làm nhất: làm game.

Trưởng nhóm kỹ thuật cũng phải đặt ra các tiêu chuẩn về code (coding standards), khuyến khích mọi người tuân thủ các "best practices", thiết lập quy trình quản lý mã nguồn, thực thi kế hoạch sao lưu dự phòng dữ liệu trong trường hợp gặp thảm họa!

Trong quá trình sản xuất, trưởng nhóm quản lý các công việc và tiến độ của các lập trình viên. Có một sự khác biệt giữa giao việc và lập kế hoạch cho nó. Mặc dù anh ta cần quyết định ai sẽ làm gì nhưng kế hoạch thực hiện thường do chính người được giao việc ước lượng. Cần theo sát thời gian ước lượng và thời gian hoàn thành thực tế để giúp các

thành viên ngày càng ước lượng chính xác hơn. Khi đã đi xa hơn, các ước lượng (ngày càng chính xác) này sẽ giúp trưởng nhóm kỹ thuật cân đối và đưa ra quyết định có thể làm hoặc bỏ đi tính năng nào.

Đối với những người không rành kỹ thuật, thế giới kỹ thuật trông có vẻ rất huyền bí. Trưởng nhóm kỹ thuật phải biết cách giải thích những vấn đề kỹ thuật cho họ theo cách đơn giản và dễ hiểu. Đừng ra vẻ ta đây là người duy nhất có tư cách quyết định về kỹ thuật. Thay vì vậy, hãy chia sẻ thông tin của bạn với đội ngũ để mọi người có thể cùng nhau đưa ra những quyết định thông minh nhất.

Trưởng nhóm kỹ thuật đặc biệt cần phải biết cách giải thích các chọn lựa mang tính "đánh đổi" với producer. Game thường phải chọn giữa các yếu tố: đúng thời hạn, chi phí, chất lượng (ít bug) và mức độ thoả mãn của game thủ. Hiếm có game nào thoả mãn cả 4 tiêu chí cùng lúc. Các chọn lựa này thậm chí cần được thực hiện hằng ngày và phải biết cách giải thích (một cách dễ hiểu) cho mọi người tại sao lại quyết định như vậy.

Có lẽ yếu tố khó chịu nhất của trưởng nhóm lập trình là để đến được vị trí này đã phải trải nghiệm, học hỏi nhiều để trở thành một coder chuyên nghiệp nhưng giờ anh ta lại dành phần lớn thời gian làm những việc khác. Team càng lớn bạn càng phải bỏ nhiều thời gian cho việc quản lý, lên kế hoạch, kiểm soát, cân bằng, review code và càng có ít thời gian để ngồi lập trình! Nếu team nhỏ, có thể dành thời gian code nhưng nếu team có hơn 5 developers, sẽ là một sai lầm tai hại nếu tự dành quyền code một tính năng lớn nào đó của game!

Một giải pháp cho vấn đề này là dành ít thời gian cho cho việc R&D một số tính năng hay nhưng không quá quan trọng trong game. Làm vậy sẽ giúp anh ta giữ được lửa. Nếu hoàn thành được các tính năng này, anh ta sẽ cảm thấy khá là thoả mãn!

Lập trình viên

Là một lập trình viên trong một dự án game, người này sẽ ít nhiều phải gõ code ở một hoặc nhiều trong số các đầu việc (tùy theo độ lớn của đội, đội càng lớn ta càng làm ít đầu việc và làm càng sâu hơn)

- Rendering engine
- Trí tuệ nhân tạo
- Vật lý
- Công cụ
- Cơ sở dữ liệu
- Network & multiplayer
- Hiệu ứng đồ hoạ
- Âm thanh
- Scripting
- Game logic
- Giao diện game
- Tích hợp các tài nguyên game

Cuộc sống của một lập trình viên game đôi lúc cũng khá "khó khăn". Lập trình viên là người hiện thực hoá mọi ý tưởng - hiểu rõ về nó nhất - nhưng lại có ít quyết định nhất về nó nhất. Do vậy, lập trình viên cũng ít nhiều cần nâng cao hơn kỹ năng đối thoại để có thể giải thích một cách cặn kẽ và dễ hiểu về chi tiết kỹ thuật của tính năng mình đang làm để cung cấp thêm thông tin cho những người cần quyết định.

Sau đây là một số gợi ý mà một lập trình viên game nên lưu ý

- Hãy thường xuyên đọc các bài post mortem (rút kinh nghiệm sau khi hoàn thành dự án) của những dự án khác để tìm hiểu những gì người khác thất bại. Hãy suy nghĩ xem trường hợp của họ có thể xảy ra với mình không. Nếu có, hãy nhanh chóng thông báo cho mọi người và tìm cách chuẩn bị đối phó. Đừng tự huỷ hoại hoặc "điều này không bao giờ xảy ra với mình" hoặc tệ hơn là "Chà, chính nó đã xảy ra với chúng ta nhưng nó buộc phải thế, bó tay rồi"

- Liên tục cập nhật kiến thức và kỹ năng trong ngành. Tự học liên tục là chìa khoá. Nếu lập trình viên không theo kịp ngành, anh ta sẽ nhanh chóng bị bỏ lại.
- Tránh "cám dỗ" của ý tưởng trở thành "siêu nhân" (làm việc như trâu suốt ngày). Thỉnh thoảng điều đó cũng cần thiết nhưng không đến nỗi phải liên tục. Hãy nên nhớ là cây trâu luôn có cái giá là ta sẽ bị giảm hiệu quả công việc giữa chừng hoặc sẽ phải tốn thời gian để hồi phục về sau.
- Cần mở rộng tầm lòng để đồng nghiệp đánh giá công việc của mình.
- Cần trung thực với công việc của mình, với chính mình và với người khác.
- Dành thời gian để lên kế hoạch công việc.
- Hoà đồng với đồng nghiệp. Ngày nay thậm chí cả hoạ sĩ và người viết kịch bản thỉnh thoảng cũng cần biết chút ít về kỹ thuật để hiểu rõ việc mình làm hơn. Hãy làm và giúp đỡ họ.
- Tránh bị "lạm" quá sâu vào một công việc / tính năng. Dĩ nhiên game cần phải hay nhưng đừng quên là chúng ta có những ưu tiên khác quan trọng hơn.
- Mọi tính năng - dù đơn giản cũng cần phải tích hợp, test và debug đầy đủ. Đừng xem thường những tính năng nhỏ. Đôi lúc một tính năng nhỏ bị hỏng cũng gây ra hậu quả nghiêm trọng tới toàn game. Ví dụ: chỉ cần quên tính năng đóng một hộp thoại trong game sẽ gây thảm hoạ, khi game thủ mở hộp thoại lên và không cách nào tắt nó đi được - mạch chơi của họ sẽ bị gián đoạn và sự giận dữ với chúng ta là không thể tránh khỏi.

Một cách rất quan trọng để lập trình viên đóng góp vào thiết kế game là khi họ làm việc với game designer về những chi tiết nhỏ mà anh ta có thể quên chưa đề cập đến trong tài liệu thiết kế của mình. Do lập trình viên phải viết code rất chi tiết nên họ có thể phát hiện ra các chi tiết này. Designer thường suy luận theo kiểu "nếu ... thì ..." mà quên mất về

else. Do đó lập trình viên có thể giúp designer tìm ra về "else" bị bỏ quên đó từ đó gán tiếp đóng góp vào gameplay.

1.2.5. Đội ngũ mỹ thuật

Hình ảnh đẹp đã trở thành một trong những tiêu chuẩn đánh giá game. Mặc dù nhiều ý kiến chuyên môn vẫn cho rằng chúng ta không thể đánh giá một cuốn sách thông qua bìa của nó hay người Việt ta thường nói “tốt gỗ hơn tốt nước sơn” – sự thật là hàng triệu game thủ ngày nay vẫn nhìn vào đồ họa game đầu tiên để quyết định có mua hay gắn bó với nó hay không.

Mỹ thuật ảnh hưởng đến mọi khía cạnh thiết kế game – từ giao diện người dùng đến cách thể hiện thế giới game và các hiệu ứng đồ họa đặc biệt trong game. Việc tạo ra các tài nguyên mỹ thuật đang ngày càng trở nên quan trọng và phức tạp khi các công cụ mỹ thuật đang ngày càng trở nên hoàn thiện và tiện lợi. Nhiều công ty trước kia từng xem nhẹ mỹ thuật game (bằng cách outsource công việc mỹ thuật cho các công ty chuyên về mỹ thuật) nay đã nhận ra tầm quan trọng của nó trong việc tạo ra hiệu quả cạnh tranh của game nên đã đầu tư xây dựng bộ phận mỹ thuật thường trực ngay trong công ty của mình.

Trưởng nhóm mỹ thuật

Trưởng nhóm mỹ thuật là người chịu trách nhiệm cho “mặt mũi” của game. Thông thường trưởng nhóm mỹ thuật cũng sẽ kiêm luôn vai trò họa sỹ ý tưởng (concept artist) hoặc nếu không giữ vai trò này, trưởng nhóm mỹ thuật sẽ hướng dẫn những người trong vai trò này tạo ra các “hình ảnh” thể hiện được hình dung của người thiết kế về thế giới game.

Trưởng nhóm mỹ thuật sẽ “sống” tại điểm giao nhau giữa thiết kế, lập trình và quản lý. Trưởng nhóm mỹ thuật cần phân tích được nhu cầu của designer, làm việc với trưởng nhóm kỹ thuật để lên kế hoạch tổng thể và phạm vi công việc của nhóm mỹ thuật, tính toán nguồn nhân sự,

kỹ năng của nhân sự và ước lượng thời gian để lắp ghép mọi thứ thành một bức tranh mỹ thuật tổng thể của game.

Khi làm việc với designer, một trong những mục tiêu quan trọng của vai trò này là phải xây dựng được một phong cách game thống nhất xuyên suốt toàn bộ game, từ màn hình giới thiệu đến nhân vật và môi trường game – thậm chí cả giao diện menu game. Sau khi anh ta đã thiết lập xong phong cách mỹ thuật, trưởng nhóm mỹ thuật cần ghi xuống thành cẩm nang hướng dẫn “phong cách game” cho tất cả mọi họa sĩ trong nhóm mỹ thuật tra cứu hằng ngày.

Khi làm việc với trưởng nhóm kỹ thuật, ta cần phải tạo ra một quy trình xuyên suốt để đưa các tài nguyên mỹ thuật từ các công cụ vẽ của các họa sĩ vào trong game một cách nhanh nhất. Hoàn hảo nhất là làm sao để các họa sĩ có thể thấy ngay kết quả của họ trong game sau khi vẽ xong mà không cần sự hỗ trợ của các lập trình viên. Cần thảo luận với trưởng nhóm lập trình về việc sử dụng một ngôn ngữ kịch bản hoặc công cụ có sẵn nào đó để các họa sĩ có thể cắt dán kết quả của họ vào trong một build của game và ngay lập tức thấy nó nhìn ra sao trong game.

Điều này sẽ giúp tránh được các thời gian “chết” trong dự án khi các họa sĩ vẽ xong các hình ảnh hay chuyển động rồi đưa cho một “ai đó” để đưa vào game – hy vọng rằng nó sẽ tự nhiên ổn. Làm việc kiểu này sẽ thường dẫn đến tình huống “bi kịch” khi “ai đó” quên bỏ vào game cho đến vài tuần hoặc tháng sau khi bạn chợt phát hiện ra các hình ảnh đó không đạt chất lượng – và mọi người phải lục lại làm lại từ đầu. Cho phép các họa sĩ tự đưa kết quả công việc của họ trực tiếp vào game sẽ làm mọi thứ nhanh và hiệu quả hơn rất nhiều cho dự án.

Cũng với tư cách là nhà quản lý, trưởng nhóm mỹ thuật cũng sẽ phải tuyển dụng và điều phối hoạt động của team, lên kế hoạch công việc, đánh giá các công nghệ (mỹ thuật) cần dùng, chọn lựa công cụ và quyết định xem tài nguyên mỹ thuật nào cần tự xây dựng hay outsource ra bên ngoài.

Cũng như mọi công việc quản lý nào khác, bạn sẽ nhanh chóng nhận ra rằng các việc quản lý sẽ chiếm gần hết quỹ thời gian của bạn. Hãy cố

gắng tìm một công việc trong team mang tính chất nghiên cứu hoặc huấn luyện các thành viên mới.

Họa sĩ

Không có lĩnh vực nào trong ngành công nghiệp game phát triển nhanh như mỹ thuật. Là họa sĩ, cần phải sẵn sàng để liên tục cập nhật kiến thức và kỹ năng hay là “chết”. Họa sĩ không thể nói không với công nghệ. Họa sĩ không những cần các máy tính đời mới nhất cùng với những phần mềm tiên tiến nhất để vẽ mà còn phải có những kiến thức nhất định về nền tảng của game để có thể điều chỉnh hình ảnh phù hợp với ưu nhược điểm của nền tảng đó.

Đóng góp của họa sĩ đối với thiết kế game chính là mọi thứ mọi người nhìn thấy trên màn hình. 90% thứ mọi người cảm nhận được trong game ít nhiều liên quan đến hình ảnh.

Công việc của họa sĩ trong dự án game liên quan đến một trong những hướng sau: khái niệm, dự hình nhân vật, tạo chuyển động, ảnh nền hoặc textures.

Họa sĩ khái niệm

Họa sỹ khái niệm làm việc với designer để xây dựng nên “mặt mũi” của game. Họa sỹ khái niệm sẽ phải vẽ ra nhiều bản phác thảo của nhân vật và các tùy biến liên quan với mục tiêu trực quan hóa ý tưởng của designer. Các phiên bản hoàn chỉnh của các phác thảo bạn tạo ra sẽ trở thành “tiêu chí” hướng dẫn cho các thành viên khác trong đội mỹ thuật để hình ảnh, thể hiện của game thống nhất chứ không phải chỉ là tập hợp của những hình ảnh lạc lõng.

Họa sỹ khái niệm cũng có thể làm việc với designer trong các mạch truyện và các họa cảnh của game để khi quá trình xây dựng game bắt đầu thì mọi người đều hiểu ý nhau để không ai tạo nên những hình ảnh thừa thãi.

Dựng hình nhân vật

Họa sĩ dựng hình nhân vật tạo ra các nhân vật trong game như nhân vật chính, quái vật, NPC cũng như các đối tượng 3D khác sử dụng các công cụ dựng hình 3D như : 3D Studio Max, Maya (hiện nay đã được 3DSMax mua lại). Xuất phát từ các hình ảnh khái niệm (concept art), bạn sẽ tạo ra mô hình lưới 3D và sau đó áp texture vào lưới 3D này để tạo ra “da” của nhân vật. Điểm thú vị là bạn có thể dựng lưới 3D từ đầu hoặc dùng đất sét để tạo ra hình dáng rồi “scan” vào máy tính.

Tạo chuyển động

Người tạo chuyển động (animator) tạo ra sự sống cho mọi thứ bằng cách bắt chúng di chuyển. Vai trò này sẽ nhận được một danh sách các yêu cầu chỉ ra các hoạt động (đi, chạy, nhảy, ngồi, né đạn, v...v) của từng nhân vật trong game và anh ta sẽ phải tạo ra một chuỗi các chuyển động ứng với từng hoạt động đó. Tạo ra các hoạt động đơn lẻ không quá khó nhưng tạo ra các bước chuyển mượt mà giữa các hành động là không đơn giản. Gần đây các gói phần mềm đều có công cụ hỗ trợ để người dùng có thể “trộn” (blend) các chuyển động với nhau để giúp ta xây dựng các chuyển động trung gian dễ hơn.

Người tạo chuyển động cũng cần có ý kiến thức về sinh học – đặc biệt về sự liên hệ giữa cơ và xương. Công nghệ chuyển động dựa vào xương (skeleton animation) ra đời đã giúp cho nhân vật chuyển động tự nhiên hơn bao giờ hết đồng thời cũng giúp cho các họa sĩ tạo chuyển động không cần phải dựng hình thủ công chi tiết từng chuyển động nữa.

Một cách để tạo chuyển động rất tự nhiên là dùng công nghệ “motion capture”. Công nghệ này đặc biệt hữu dụng trong việc thu lại chi tiết các chuyển động đặc trưng của một số vận động viên thể thao nổi tiếng – quá khó để có thể tái lập lại chỉ bằng cách quan sát và mô phỏng lại.

Trong một studio thực hiện motion capture, đạo diễn gắn các cảm biến vào những điểm quan trọng trên cơ thể của diễn viên. Diễn viên sẽ thực hiện các động tác hay hành động cần thiết và quá trình di chuyển của các điểm này sẽ được ghi nhận lại. Motion capture là một quá trình không hề rẻ nhưng kết quả mang lại thường rất chính xác trong việc mô phỏng lại hoạt động của cơ thể con người. Công nghệ này thường được

sử dụng trong các game thể thao và cũng đã trở nên khá quen thuộc đối với các dòng game đánh nhau và hành động.

Mô hình thế giới nền (background modelling)

Người làm vai trò này sẽ xây dựng thế giới mà trong đó người chơi sẽ “sống” trong đó. Anh ta sẽ thường bắt đầu từ những hình học cơ bản (như hình khối, hình trụ, vv) sau đó kết hợp chúng lại, biến đổi chúng để tạo ra các căn phòng / khu vực và những đối tượng trong đó tạo nên môi trường của game. Sau khi tạo xong khối đa giác (mesh), anh ta sẽ thêm vào hiệu ứng tô bóng phẳng (flat shading) rồi áp texture và cuối cùng là hiệu ứng ánh sáng để thế giới nhìn giống thực.

Textures

Họa sĩ texture tạo ra “da” để lợp lên mô hình 3D “trơ xương” của nhân vật. Đối với các texture background, anh ta sẽ vẽ lên bề mặt 2D để sau khi lợp lên các đối tượng nó sẽ tạo cảm giác nổi lên và nhìn xa trông giống thật như gạch, đá, kim loại. Có lúc anh ta sẽ tạo các texture này từ đầu rồi xây dựng từng lớp, từng lớp lên. Tuy nhiên, nhiều lúc để hiệu quả hơn, chúng ta nên chụp hình các bề mặt thật ở ngoài đời thật rồi chỉnh sửa nó. Đối với texture cho nhân vật thì các gói phần mềm hỗ trợ càng ngày càng phổ biến. Người dùng có thể vẽ theo cách WYSIWYG – nghĩa là vẽ trực tiếp lên đối tượng 3D chứ không cần phải chuyển qua chuyển lại giữa đối tượng 3D và bề mặt 2D của texture.

1.2.6. Đội ngũ kiểm tra chất lượng sản phẩm

Kiểm tra chất lượng không phải chỉ đơn giản là tìm lỗi trước khi game được phát hành. Các tester (nhân viên kiểm tra chất lượng sản phẩm) bắt đầu đóng vai trò sống còn trong team ngay khi dòng code đầu tiên được viết. Tester là người cuối cùng quyết định game có nên được phát hành hay không.

Trưởng nhóm kiểm tra chất lượng sản phẩm (test lead)

Vai trò của *test lead* thường bị được hiểu sai và hay bị đánh giá thấp nhất trong đội ngũ phát triển!

Trách nhiệm chính của trưởng nhóm không chỉ là đảm bảo game không bị lỗi mà còn phải đảm bảo game chơi vui vẻ.

Những ngày đầu của dự án, team thường nhỏ và mục tiêu của team là cung cấp những phản hồi về chất lượng chính xác cho đội ngũ lập trình. Mỗi ngày các lập trình viên sẽ viết ra một số đoạn code, kiểm tra nhanh xem nó có chạy không rồi giao lại cho nhóm test để mọi người rà soát kỹ lưỡng xem nó có lỗi hay vấn đề nguy hiểm nào không. Chính quá trình này là cơ hội để test lead đóng góp hay gây ảnh hưởng đến thiết kế game vì test lead có thể góp ý về tất cả những tính năng trong lúc nó còn chưa định hình.

Khi dự án tiến đến giai đoạn alpha (giai đoạn game có thể ít nhiều chơi được từ đầu đến cuối), test lead sẽ phải xây dựng team đầy đủ và viết kế hoạch test (test plan). Đây là thời điểm mà vai trò của trưởng nhóm vượt qua việc báo lỗi game. Trưởng nhóm phải đồng bộ với designer và các lập trình viên vì anh ta cần phải biết chính xác game hoạt động ra sao ở từng thời điểm cụ thể.

Trưởng nhóm có thể sử dụng tài liệu thiết kế (giả định rằng tài liệu này lúc nào cũng được cập nhật) như là điểm khởi đầu để xây dựng test plan nhưng một tài liệu viết không bao giờ có thể ghi nhận đầy đủ hàng ngàn những quyết định / thay đổi nho nhỏ trong suốt quá trình phát triển. Để bù đắp cho những chỗ còn thiếu trong tài liệu thiết kế, test lead cần phải hiểu một cách thấu đáo tầm nhìn của game.

Khi những tester khác đang làm việc trong dự án, test lead sẽ thấy mình đóng vai trò trung gian giữa họ và các lập trình viên. Anh ta nên nhớ rằng không có team nào có thể đọc được suy nghĩ của người khác và việc đối thoại với team lập trình về các vấn đề của game (thường được hiểu sai là lỗi làm hay khuyết điểm của chính team lập trình) là cực kỳ nhạy cảm. Chẳng hạn: khi một tester thấy một hình ảnh bất thường trên màn hình, anh ta thường không chắc hình ảnh nằm chỗ đó có phải là có

chủ ý hay không nên anh ta sẽ báo bug. Báo bug không đúng chỗ sẽ làm “phật ý” lập trình viên – vì anh ta đã làm tốt nhiệm vụ của mình.

Một vấn đề thường gặp khác là lập trình viên thông báo đã sửa xong lỗi và sau đó nhận lại một báo lỗi về chính lỗi vừa sửa. Tại thời điểm đó, rất khó cho anh ta để biết là lỗi đó là lỗi cũ chưa sửa hay lỗi mới phát sinh. Nhiệm vụ của trưởng nhóm là phải liên tục nắm rõ tình hình lỗi và việc sửa lỗi – loại bỏ những báo lỗi trùng lặp cũng như báo cho lập trình viên biết anh có sửa lỗi thành công thay không.

Một test lead giỏi là người phải biết kiểm chế và giải tỏa các tình huống tế nhị trước khi căng thẳng bắt đầu dâng cao. Bạn cần phải điều chỉnh các ngôn ngữ mang tính chỉ trích từ các báo cáo lỗi, bắt tester sửa lại các báo lỗi chưa rõ ràng cho đến khi nó có đủ thông tin cho lập trình viên. Ở chiều ngược lại, test lead cũng cần phải bảo vệ tester của mình trước những lập trình viên “hung hãn” khi họ bị chạm tự ái. Anh ta cần nhớ rằng, nhiệm vụ của tester là báo lỗi (và có thể trình bày quan điểm của họ) nhưng chỉ có producer và designer là người quyết định cuối cùng là cần làm gì với các báo cáo lỗi.

Một hệ thống quản lý lỗi tốt sẽ giúp ích nhiều thứ. Nếu một bug report có kèm theo version, lập trình viên sẽ nhanh chóng thấy được bug được báo trước hay sau khi hệ thống có thay đổi. Hệ thống cho phép bạn sắp xếp danh sách bug theo lập trình viên cũng khá hữu ích vì nó sẽ giúp cho lập trình viên đó theo dõi chính xác những bug mà anh ta phải sửa – thay vì bị “chìm” trong đồng bug của cả team.

Khi sản phẩm tiến đến giai đoạn beta, team của bạn sẽ lớn hơn nữa và bạn sẽ phải lên kế hoạch chi tiết mỗi ngày cho từng tester. Phản hồi của bạn về chất lượng sản phẩm cho team lập trình lúc này cực kỳ quan trọng vì mọi người lúc này đều phải đưa ra nhiều quyết định cuối cùng liên quan đến việc “hy sinh” hay không một vài tính năng. Đừng ngại đứng lên bảo vệ những tính năng chính yếu của game và thẳng thắn khi cần phải bỏ đi những tính năng không quan trọng.

Hãy xoay vòng những tester mới vào dự án nếu có thể. Những nhân tố mới này có thể tìm ra những bug mới hoặc có thể đưa ra những góc nhìn mới lạ về những vấn đề “cũ” mà mọi người đã quá quen với nó.

Các lỗi thường được xếp hạng dựa trên ước lượng của test lead về độ quan trọng của chúng. Lỗi hạng A là lỗi gây treo hoặc lỗi nghiêm trọng ảnh hưởng đến khả năng phát hành của game. Lỗi loại B là các lỗi liên quan đến chất lượng rất cần phải sửa nhưng nếu game buộc phải được phát hành vì lý do nào đó, những lỗi này sẽ không được liệt vào hàng “thảm họa” (showstopper) – Tuy nhiên, nếu có nhiều lỗi hạng B quá cũng sẽ tương đương với có lỗi hạng A. Lỗi hạng C thường là những lỗi dạng “sửa được thì tốt” hoặc liên quan đến những vấn đề hiếm gặp. Danh sách lỗi dạng C thường chỉ được sửa khi mọi thứ khác đã xong.

Vào cuối chu kỳ sản phẩm, khi áp lực hoàn tất và phát hành game dâng cao, test lead sẽ càng có cảm nhận rõ ràng hơn về việc có nên phát hành game hơn chính cả designer hoặc tech lead. Anh ta sẽ phải gặp gỡ hằng ngày với những nhân vật quan trọng để thảo luận về những vấn đề còn tồn đọng và quyết định cần phải giải quyết vấn đề nào và tạm bỏ qua vấn đề nào. Trong một số công ty, vai trò của test lead quan trọng tới mức người này được quyền quyết định cuối cùng trong việc có nên đưa game ra thị trường hay không!

Nhân viên kiểm tra chất lượng - Tester

Việc nhiều *game designer* bắt đầu sự nghiệp với vai trò tester là điều hoàn toàn không ngẫu nhiên! Ở vai trò này, anh ta là người thấy rõ nhất các lỗi làm được tạo ra và được giải quyết như thế nào. Làm tester một năm có lẽ là phương pháp huấn luyện designer tốt nhất.

Khi test một game, tester cần phải lưu ý những điều sau cùng lúc

- **Game chơi vui không?** Đây là câu hỏi cần được đặt ra liên tục trong suốt thời gian đầu của game. Cơ chế gameplay cơ bản của game có đáng thưởng thức không? Ngay cả khi game chưa được cân bằng hay tinh chỉnh, bạn có thể tìm được niềm vui khi chơi game không? Phản hồi của bạn tới game designer và cả lập trình viên trong suốt quá trình pre-alpha sẽ có tầm ảnh hưởng kinh khủng đến thiết kế game.

- **Mọi thứ có dễ dàng không?** Điều khiển mọi thứ trong game là đơn giản hay rối rắm? Giao diện game có dễ hiểu, tiện lợi không? Tài liệu hướng dẫn sử dụng có chính xác không?
- **Mọi thứ trông có lý không?** Khi chơi theo mạch game, game thủ có được thứ mà ta muốn anh ta cảm nhận được không?
- **Game chơi có vui không (tập 2):** khi dự án tiến đến giai đoạn alpha, câu hỏi này cần được tiếp tục lặp lại. Trước kia, tester chỉ soi vào những yếu tố cơ bản để xem chúng có đáng thưởng thức không – còn bây giờ tester đang test chính cái game để đảm bảo niềm vui mà anh ta tìm thấy ban đầu vẫn còn nguyên vẹn sau một thời gian dài. Mọi thứ có khó quá? Hay dễ quá? Có chỗ nào khiến game thủ bị lạc lối hay không hiểu nổi mình cần phải làm gì tiếp theo không?
- **Mọi thứ có chạy không?** Đây là việc mà mọi người đều nghĩ đến khi nghe nói đến vai trò tester. Khi tester chơi qua game, thực hiện những việc mà game thủ được định hướng sẽ phải làm, anh ta có thể đến đích được không? Nếu anh ta không làm theo những định hướng đó, mọi thứ có diễn ra suông sẽ không? Tester có thể làm game treo không?

Khi tester tiếp cận công việc này, cần cố gắng làm mọi thứ khác đi một chút mỗi khi anh ta chơi game. Mục tiêu của anh ta là phải chơi như 5000 game thủ khác nhau chơi game, mỗi người chơi theo một cách hơi khác nhau. Tránh đừng lặp lại theo một mẫu nào đó – chẳng hạn luôn đi cùng một con đường trong môi trường game. Hãy xáo trộn mọi thứ. Hãy thử những việc mà những người bình thường thường không làm! Tester sẽ trở nên mau chán khi cứ chơi đi chơi lại một kiểu giống nhau. Hãy tự thưởng cho mình (và cũng là một cách làm việc hiệu quả hơn) bằng cách đưa vào nhiều kiểu chơi game khác nhau.

Viết báo cáo lỗi tốt cũng là một nghệ thuật. Tester cần phải viết mọi thứ thật chính xác. Tester càng mô tả bug chi tiết thì lập trình viên càng dễ sửa nó. Anh ta không nên chỉ báo về lỗi xảy ra ra sao mà còn phải đưa càng nhiều thông tin chi tiết về cái gì và làm sao để lỗi xảy ra. Tester

cũng cần phải cung cấp mọi thông báo lỗi mà anh ta thấy trên màn hình (nếu có).

Khả năng tái tạo lỗi (reproducibility / repeatability) là “chén thánh” của mọi báo cáo lỗi. Nếu tester có khả năng tái lập lỗi chính xác bất cứ lúc nào, 90% là lập trình viên sẽ sửa được nó.

Tester cũng không nên ngại cho mọi người biết ý kiến của anh ta về lỗi nhưng đừng nhạy cảm về những phản ứng của mọi người về ý kiến của anh ta. Nhiệm vụ của tester là nói cho mọi người biết về nó còn nhiệm vụ của họ là quyết định cần làm gì với nó. Hãy thể hiện tác phong chuyên nghiệp mọi lúc và cần nhận thức rõ mọi người đang ở đâu trong quá trình phát triển. Một đề xuất có ảnh hưởng sâu rộng sẽ được trân trọng trong giai đoạn pre-alpha nhưng sẽ không được trân trọng 2 ngày trước khi đến ngày phát hành game!

1.3. QUY TRÌNH PHÁT TRIỂN GAME VÀ CÁC TÀI LIỆU LIÊN QUAN

Một số game được xây dựng trong vòng vài tháng nhưng cũng có những game được xây dựng trong vòng vài năm – thậm chí cả thập kỷ. Bất chấp độ dài của dự án, game nào cũng phải trải qua một quy trình phát triển khá tiêu chuẩn. Phần này mô tả các giai đoạn trong quy trình phát triển và các tài liệu liên quan.

1.3.1. Xây dựng khái niệm

Xây dựng khái niệm game là một bước khá “mò” đối với thiết kế game. Quá trình này diễn ra từ khi ai đó chợt nảy ra một ý tưởng game cho đến khi game đến giai đoạn tiền sản xuất. Trong giai đoạn này, đội ngũ sẽ rất nhỏ - chỉ bao gồm designer, tech lead, họa sĩ khái niệm và producer (có thể chỉ là parttime).

Mục tiêu của giai đoạn xây dựng khái niệm là để quyết định game làm gì và viết xuống rõ ràng để mọi người có thể hiểu nó ngay lập tức.

Trong lúc này, mọi người sẽ quyết định các yếu tố gameplay cơ bản, tạo ra các hình ảnh khái niệm (concept arts) để minh họa game trong thế nào trong màn hình và định hình cốt truyện (nếu có).

Nếu chúng ta làm việc cho một nhà phát triển độc lập, giai đoạn này sẽ không được ai cấp kinh phí. Trừ khi công ty của chúng ta có một lịch sử đầy tự hào về làm game còn không thì hiếm khi chúng ta sẽ tìm được một nhà phát hành nào chịu chi tiền cho mình chỉ để nghe thuyết trình về ý tưởng!.

Các tài liệu có được sau giai đoạn khái niệm là tài liệu “khái niệm tổng quát” (high-concept), đề xuất game (game proposal – hay pitch doc) và tài liệu khái niệm (concept document)

Khái niệm tổng quan

High-concept là một hoặc hai câu mô tả cho biết game chúng ta sắp làm là cái gì. Đây là “cần câu” để làm game này trở nên thú vị và tách biệt hẳn khỏi các đối thủ.

Một khái niệm tổng quan mạnh mẽ cũng rất có giá trị trong suốt quá trình phát triển vì nó giúp chúng ta quyết định tính năng nào sẽ được giữ lại hoặc bỏ đi. Nếu ví việc phát triển game như việc tìm đường ra khỏi một rừng các khả năng thì khái niệm tổng quan là một con đường đã được dọn dẹp sạch sẽ để giúp chúng ta không bị lạc lối. Bất kỳ tính năng nào không đóng góp trực tiếp cho mục tiêu chính của game đều là hướng đi chúng ta không cần phải tìm hiểu.

Đề xuất game

Đề xuất game là tài liệu bao gồm 2 trang mà chúng ta sẽ đưa tận tay cho người nghe khi ta trình bày về ý tưởng game trong các cuộc họp tìm nguồn tài chính cho game. Chỉ trong vòng vài trang ngắn ngủi này, chúng ta phải tóm tắt được game là cái gì, tại sao nó sẽ thành công và

tại sao nó kiếm được tiền. Tài liệu này gần giống như tài liệu khái niệm nhưng vẫn tốt hơn.

Tài liệu khái niệm

Đây là phiên bản chi tiết hơn của tài liệu đề xuất game – bao gồm khoảng 10-20 trang trao cho các thành viên đánh giá trong nhà xuất bản. Họ không có thời gian để xem chi tiết trong buổi giới thiệu ý tưởng game nhưng họ sẽ xem lại về sau để hiểu chi tiết hơn về game của chúng ta.

Tài liệu này nên được trình bày chuyên nghiệp và đóng gáy cẩn thận, có bìa đẹp, ấn tượng cũng như nên có nhiều hình ảnh khái niệm của game để gây ấn tượng tốt. Nó nên có đầy đủ các thành phần sau: khái niệm tổng quan, dòng game (genre), mô tả gameplay, các tính năng, thế giới game, cốt truyện, người chơi mục tiêu, nền tảng phần cứng, kế hoạch sơ khởi, kinh phí và ước lượng doanh số, phân tích về đối thủ cạnh tranh, giới thiệu về team, phân tích rủi ro và bản tóm lược mọi thứ.

Dòng game (Genre)

Cho biết game thuộc dòng chính nào (bắn súng, nhập vai, hành động, v.v..) và những yếu tố thuộc dòng game khác (nếu có)

Gameplay

Mô tả game thủ sẽ làm gì khi anh ta đang chơi game. Chúng ta cần nhấn mạnh những điểm thú vị mà game của mình sẽ mang lại cho dòng game.

Tính năng

Liệt kê các tính năng làm cho game của chúng ta trở nên đặc biệt. Ta có thể đưa vào đây mọi thứ từ phong cách mỹ thuật đặc biệt cho đến việc

dùng engine xin nhất. Hãy viết phần này giống như phần quảng cáo mà ta sẽ viết cho game thủ xem.

Thế giới game

Mô tả thế giới mà game thủ sẽ “đắm chìm” trong đó bao gồm cả các hình ảnh khái niệm (nếu có). Nếu game có cốt truyện, hãy làm rõ các điểm chính thú vị của thế giới game và giải thích ảnh hưởng của nó đến cốt truyện thế nào.

Cốt truyện

Nếu game của mình có cốt truyện, chúng ta hãy dành một trang để tóm tắt nội dung chính. Hãy giới thiệu nhân vật chính, nói rõ vấn đề của anh ta, mô tả nhân vật phản diện và giải thích làm sao vị anh hùng có thể tiêu diệt kẻ thù vào cuối truyện.

Game thủ mục tiêu

Giải thích chúng ta dự định làm game này dành cho ai và tại sao ta nghĩ game này sẽ hấp dẫn với họ.

Nền tảng phân cứng

Liệt kê danh sách các thiết bị mà game có thể chơi trên đó được. Ví dụ: PC, điện thoại Android, điện thoại iOS, Flash,...

Ước lượng kế hoạch, ngân sách và lợi tức

Chia nhỏ các giai đoạn chính của quá trình phát triển và mức độ nỗ lực ứng với mỗi giai đoạn để chứng tỏ chúng ta sẽ hoàn thành các kế hoạch được đưa ra trong tài liệu đề xuất game.

Nếu đang làm cho một nhà phát hành game, ta cũng sẽ được yêu cầu cung cấp ước lượng lãi lỗ (P&L – Profit & Loss) tại bước này. Đây là

ước lượng tổng chi phí từ lúc đầu đến lúc phát hành game ra thị trường cũng như tổng các đầu thu nhập có thể có được.

Ngược lại, nếu ta là một nhà phát triển game độc lập, sẽ khó tính ra được lãi lỗ do ta không thể biết được chi phí phát hành game thế nào. Thay vào đó, chúng ta chỉ cần đưa ra chi phí phát triển là đủ. Chúng ta cũng nên tính luôn chi phí cho giai đoạn tiền sản xuất.

Mục đích tối tượng của P&L là tính ra con số ROI (Return On Investment) – tỷ lệ tiền kiếm được so với tiền đầu tư. Con số này phải cho nhà đầu tư thấy được họ sẽ kiếm được nhiều tiền hơn khi bỏ tiền vào game của mình so với việc bỏ tiền vào ngân hàng kiếm lời trong vòng 2-3 năm. Mọi người đều làm để kiếm tiền, nếu chúng ta không thể thuyết phục được mọi người là lợi tức từ game của mình đủ để khóa lấp các mối nguy từ đầu tư thì chúng ta sẽ khó có được đầu tư.

Phân tích cạnh tranh

Lập danh sách các game có cạnh tranh trực tiếp với game của chúng ta và giải thích tại sao game của chúng ta tốt hơn. Nếu ta tin rằng game của mình tương tự như một trong những game đã từng rất thành công trong quá khứ, hãy giải thích sự giống nhau và dùng số liệu của những game này để chứng minh.

Đội ngũ phát triển

Tóm tắt những điểm mạnh của team và nhấn mạnh sự nổi trội của những thành viên chính trong đội ngũ (đặc biệt là kinh nghiệm làm game của họ). Các nhà đầu tư thường khá quan tâm đến đội ngũ phát triển và những thành công/thất bại của họ khi nhìn vào tài liệu đề xuất game. Mục tiêu của phần này là củng cố niềm tin của nhà đầu tư là họ có những người có thể hoàn thành tốt nhiệm vụ.

Phân tích các mối nguy hiểm

Phần này liệt kê ra những thứ có thể khiến mọi chuyện trở nên xấu đi và kế hoạch ứng phó với nó khi nó xảy ra. Một số mối nguy đe dọa đến project bao gồm:

- Khó khăn trong việc tuyển dụng
- Chậm trễ trong việc mua các thư viện, công cụ cần thiết
- Dựa vào các nguồn nhân sự bên ngoài cho những thành phần kỹ thuật quan trọng.
- Thay đổi về lượng người dùng đối với nền tảng phần cứng đã chọn.
- Sự phát triển của các công nghệ cạnh tranh.

Phần này cũng bao gồm ý kiến của bạn về những khu vực an toàn trong dự án. Nếu chúng ta đã có giải pháp sẵn cho những mối nguy được nêu ở trên (chẳng hạn: có đủ team – không cần tuyển thêm) – hãy nói rõ ở phần này.

Tóm lược

Nhấn mạnh lại lần nữa những điểm tốt đẹp của game và khả năng của đội ngũ chúng ta trong việc tạo ra một sản phẩm chất lượng, đúng thời gian và trong phạm vi ngân sách được cấp.

1.3.2. Giai đoạn Tiền sản xuất (Preproduction – Proof of Concept)

Giai đoạn tiền sản xuất là lúc chúng ta trang bị đủ đồ nghề để chiến đấu. Mục tiêu của chúng ta là hoàn tất thiết kế game, viết xong tài liệu phong cách mỹ thuật, lập kế hoạch sản xuất tổng thể, viết kế hoạch thi công, tạo bản prototype. Đây là cũng là giai đoạn chúng ta xây dựng một số bản thử nghiệm kỹ thuật (technical prototype) để minh họa tính khả thi của những công nghệ mới mà ta hy vọng sẽ tạo ra. Giai đoạn tiền sản

xuất chủ yếu để chứng minh đội ngũ của chúng ta có thể làm game và game đó xứng đáng để làm.

Nếu ta là một nhà phát triển độc lập, nhà phát hành (đầu tư) sẽ dùng thời gian này để kiểm tra xem ta có thể hình thành nên một mối quan hệ tốt với họ hay không. Nếu họ thấy chúng ta chuyên nghiệp, hợp lý và giao hàng đúng hạn, họ sẽ có khuynh hướng tiếp tục. Ngược lại, họ sẽ chấp nhận mất khoản đầu tư ban đầu và rời bỏ chúng ta.

Kết quả của giai đoạn này là tài liệu thiết kế game, kế hoạch thi công mỹ thuật, tài liệu thiết kế kỹ thuật và kế hoạch chung của dự án. Giai đoạn tiền sản xuất kết thúc bằng việc đưa ra một bản prototype – một mẫu phân mềm có thể chạy được chứng tỏ game chơi rất vui.

Tài liệu thiết kế game

Kết thúc giai đoạn tiền sản xuất, chúng ta phải có tài liệu thiết kế game mô tả tất tần tật chi tiết mọi thứ sẽ diễn ra trong game. Nội dung của tài liệu này sẽ là cơ sở để xây dựng kế hoạch thi công mỹ thuật cũng như kế hoạch kỹ thuật.

Trong suốt quá trình phát triển, tài liệu thiết kế phải luôn luôn là tài liệu được cập nhật nhất, thể hiện mọi thứ mọi người cần biết về trải nghiệm của game thủ trong game này. Tài liệu này cần mọi thông tin hoàn chỉnh về gameplay, giao diện người dùng, cốt truyện, nhân vật, quái vật, trí tuệ nhân tạo và mọi thứ khác – càng chi tiết càng tốt.

Một tài liệu cỡ vậy, nếu viết xuống giấy cũng phải dày cỡ cuốn danh bạ điện thoại – hầu như không thể nào theo dõi nổi, không ai muốn đọc nên chắc chắn sẽ nhanh chóng bị lạc hậu. Thay vì viết ra giấy, hãy đưa nó lên trang web bằng cách công cụ như blog hay Wiki.

Kế hoạch triển khai mỹ thuật

Giai đoạn tiền sản xuất sẽ là lúc bạn thiết lập “mặt mũi” game của bạn và quyết định mỹ thuật game sẽ được tạo ra như thế nào.

Phong cách mỹ thuật

Trong quá trình tiền sản xuất, designer, giám đốc mỹ thuật, họa sĩ khái niệm sẽ phối hợp với nhau để thiết lập phong cách mỹ thuật cho game. Họa sĩ khái niệm sẽ tạo ra các tài liệu tham khảo cho các họa sĩ khác tra cứu khi làm việc để đảm bảo cả team mỹ thuật có một cái nhìn đồng nhất. Thiết lập được tài liệu phong cách mỹ thuật sẽ giúp định hướng cho những họa sĩ mới vào dự án và đảm bảo kể quả cuối cùng có một phong cách thống nhất từ trên xuống. Hầu hết các sản phẩm mỹ thuật trong giai đoạn này có thể phác họa bằng bút chì nhưng đôi lúc để phục vụ quảng cáo chúng ta cũng có thể tút kỹ hơn các hình ảnh ở giai đoạn này.

Trong giai đoạn đầu của game, có một ý tưởng hay là ta nên tập hợp thành một thư viện tham khảo gồm có những hình ảnh có sẵn thể hiện được gu mỹ thuật mà ta muốn hướng tới. Những hình ảnh này có thể đến từ bất cứ đâu – tạp chí, sách hướng dẫn du lịch, quảng cáo phim, Internet – miễn sao các hình ảnh này chỉ phục vụ cho mục đích hướng dẫn/tham khảo chứ không xuất hiện trong game mà ta đang xây dựng.

Lộ trình sản xuất (Production Path)

Lộ trình sản xuất là quá trình chúng ta đi từ những khái niệm mỹ thuật cho đến thực tế, từ ý tưởng trong đầu ai đó cho đến hình ảnh thực tế xuất hiện trên màn hình của game thủ. Ví dụ: để tạo ra một nhân vật trong game hành động, bạn phải tìm cách hiệu quả nhất để đi từ đặc tả nhân vật của designer, đến giai đoạn vẽ phác thảo, đến đoạn dựng hình 3D, đến đoạn mặc “áo” cho nhân vật, đến đoạn tạo chuyển động cho nhân vật, đến đoạn áp dụng trí tuệ nhân tạo cho nhân vật, đến đoạn đưa nhân vật này vào game và xem “hắn” hoạt động ra sao. Tất cả công cụ ta chọn trong suốt quy trình này phải tương thích với nhau. Chúng phải “nói chuyện” được với nhau để kết quả công việc ở bước này có thể được mang vào bước tiếp theo để thay đổi, điều chỉnh và chuyển qua bước tiếp nữa.

Tài sản, ngân sách, công việc và kế hoạch

Kế hoạch sản xuất cũng có thể bao gồm bản nháp sơ khởi về danh mục các tài sản, các công việc của team, kinh phí cho trang thiết bị làm việc, chi phí nhân sự, Cũng giống như tài liệu thiết kế game, kế hoạch này cũng phải luôn được cập nhật trong suốt chu trình của dự án.

Tài liệu thiết kế kỹ thuật

Tài liệu thiết kế kỹ thuật vạch ra kế hoạch để trưởng nhóm kỹ thuật biến đổi bản thiết kế game trên giấy tờ thành phần mềm chạy trên máy tính. Nó thiết lập mặt kỹ thuật của quy trình sản xuất, “dàn trận” các nhiệm vụ cho mọi người có liên quan trong quá trình phát triển và ước lượng thời gian cần thiết cho những nhiệm vụ này. (Từ ước lượng công tính theo đơn vị người/tháng, ta sẽ biết được mình cần bao nhiêu người cho dự án và họ sẽ cùng làm việc với ta trong bao lâu. Thông tin này sẽ ảnh hưởng trực tiếp đến ngân sách của dự án).

Tài liệu kỹ thuật cũng sẽ mô tả những công cụ chính cần dùng để xây dựng game, cái nào đã có sẵn và cái nào cần phải mua hoặc tự viết lấy. Tài liệu này cũng liệt kê các phần mềm và phần cứng cần phải mua sắm cũng như những thay đổi / nâng cấp trong hạ tầng IT dành cho đội ngũ (dung lượng lưu trữ, năng lực backup, tốc độ mạng) để hỗ trợ cho việc phát triển.

Kế hoạch dự án

Bản kế hoạch dự án là “bản đồ đường đi” cung cấp cho mọi người cách thức chúng ta sẽ xây dựng game như thế nào. Nó bắt đầu với những danh sách công việc trong kế hoạch kỹ thuật, thiết lập các phụ thuộc, thêm vào đó những giờ phụ trội (overhead hours) và gom tất cả lại thành một kế hoạch thực tế. Kế hoạch dự án cuối cùng sẽ được chia ra thành nhiều tài liệu nhỏ để quản lý riêng biệt.

Kế hoạch nhân sự

Bản kế hoạch nhân sự đơn giản chỉ là một file Excel liệt kê mọi người trong dự án, khi nào họ bắt đầu và lương/thưởng của họ trong dự án.

Kế hoạch tài nguyên

Bản kế hoạch tài nguyên sẽ tính toán chi phí bên ngoài của dự án. Được xây dựng từ kế hoạch kỹ thuật, nó sẽ tính thời gian cần thiết để mua sắm phần cứng để phục vụ cho nhân lực nội bộ và cũng ước lượng thời khi nào các chi phí bên ngoài (lồng tiếng, nhạc, video, v.v..) sẽ cần dùng đến.

Tài liệu theo dõi dự án

Đây là nơi chúng ta theo dõi xem đội của mình có đúng tiến độ hay không. Một số producer dùng các phần mềm quản lý dự án nhưng các phần mềm này thường không đủ uyển chuyển để quản lý tất cả mọi khía cạnh của phát triển game. Producer thường đưa danh sách công việc vào một phần mềm để tạo ra Gantt Chart để làm rõ các phụ thuộc cũng như các đường găng (critical path), nhưng anh ta cũng có thể dùng vô số những kỹ thuật cá nhân khác để theo dõi dự án.

Ngân sách

Sau khi đưa các chi phí phụ trội và kế hoạch nhân sự, chúng ta kết hợp con số này với kế hoạch tài nguyên để tính ra được yêu cầu tiền mặt hằng tháng cũng như ngân sách của toàn bộ dự án cho game.

Lời lỗ

Ước tính lời lỗ ban đầu được thực hiện trong giai đoạn khái niệm. Khi quá trình phát triển tiến hành và các chi phí trở lên rõ ràng hơn, tài liệu tính toán lời lỗ cũng cần được cập nhật.

Kế hoạch phát triển

Nhiều nhà phát triển có xu hướng chống lại việc thiết lập một kế hoạch chi tiết và cam kết tuân thủ ngày phát hành game đã định trước, chúng ta có trách nhiệm phải thúc đẩy chính mình và đội ngũ của mình thực hiện điều này. Sau khi đã thiết lập ngày phát hành, một cỗ máy hoàn toàn khác sẽ được vận hành. Đội marketing sẽ đặt chỗ cho các quảng cáo trước hàng tháng tính từ ngày phát hành. Đội PR cũng sẽ thương thuyết với các tạp chí để viết các bài báo, đánh giá và giới thiệu. Đội bán hàng sẽ cam kết đưa game vào các cửa hàng. Thay đổi ngày phát hành sẽ là một đại họa đối với sự chuẩn bị kỹ lưỡng của những team này – kết quả sẽ là một doanh số thảm hại hơn so với ước tính ban đầu.

Xác định các mốc kế hoạch

Các mốc (milestone) là những thời điểm quan trọng trong quá trình phát triển được đánh dấu bằng việc hoàn thành một khối lượng công việc quan trọng nào đó – gọi là *deliverable* (kết quả công việc). Các kết quả công việc này cần phải được định nghĩa rõ ràng bằng những ngôn từ đại loại như “bảng phác thảo cho 15 nhân vật bao gồm cả mặt trước, mặt bên và mặt sau” hoặc “Vũ khí số 1 được mô hình xong, lợp bề mặt xong, có thể xài được trong game với âm thanh tạm, chưa có animation và hiệu ứng đặc biệt”.

Tránh mô tả kết quả theo kiểu không rõ ràng như “Thiết kế 25% hoàn thành”. Các kết quả tốt nhất nên theo kiểu “nhi phân” – hoặc hoàn thành toàn bộ hoặc không hoàn thành – không thể chấp nhận những kết quả nửa vời.

Bản nguyên mẫu (prototype)

Kết quả hữu hình của quá trình tiền sản xuất là bản prototype của game. Đây là một “mẫu” phần mềm có thể chạy được thể hiện lên màn hình ý tưởng cơ bản tạo nên sự khác biệt giữa game của chúng ta với “phần còn lại của thế giới”.

Bản demo mang tính cảm nhận (look&feel) này có thể là yếu tố có tầm ảnh hưởng lớn nhất đến việc dự án có được tiếp tục hay không. Các nhà

phát hành thường thích nhìn vào màn hình và “thấy” nó ngay lập tức. Nếu họ không thấy được “ý đồ” (hay định hướng) trong vòng một hoặc hai phút, họ sẽ ít khi đầu tư phần còn lại cho dự án.

Đây là một nhiệm vụ cam go, đặc biệt nếu dự án game của chúng ta cần dùng một engine mới hoặc công nghệ ta dùng chỉ có thể được hoàn chỉnh ở phần sau của quy trình phát triển. Trong trường hợp này, hầu hết các nhà phát triển sẽ **mô phỏng** lại để thấy được sản phẩm cuối cùng trông như thế nào bằng cách dựng sẵn (prerender) video một phần hoặc toàn bộ các thành phần được thể hiện real-time trong game.

Một cách tiếp cận khác là chuẩn bị một buổi demo độc lập để chứng minh các mảng kỹ thuật mà chúng ta chọn lựa có tính khả thi cao. Các buổi demo kỹ thuật nhỏ này có thể không có gì đáng chú ý ở góc độ mỹ thuật nhưng chúng chứng tỏ các mục tiêu của mình đặt ra là có thể đạt được. Một buổi demo kỹ thuật thường không có gì ngoài việc thể hiện một vài hình cầu được chiếu sáng theo nhiều cách, hay camera di chuyển trong một môi trường hình hộp không có chức năng gì cả hoặc là một lô hạt (particle) đụng lẫn nhau khi chúng được phát ra từ một nguồn vô hình nào đó. Ý định chính là cho thấy các thành phần kỹ thuật của chúng ta là chắc chắn, khả thi. Tuy nhiên, phải thẳng thắn mà nói rằng, nếu ta chọn prototype của mình theo hướng demo kỹ thuật riêng rẽ thì đây là cách demo khó nhất và có nhiều nguy cơ bị từ chối nhất – trừ khi team chúng ta là một team rất rất mạnh về công nghệ.

Bản prototype không chỉ cho nhà đầu tư thấy định hướng của ta là đúng đắn mà còn khẳng định lại quy trình sản xuất chúng ta đã đề ra là hữu hiệu và mọi người có thể đi từ ý tưởng đến thực tế với một cách thức hợp lý và hiệu quả.

1.3.3. Quá trình phát triển

Quá trình phát triển là một quãng thời gian dài đằng đặc!

Kế hoạch phát triển của chúng ta thường sẽ kéo dài từ 6 tháng đến 2 năm. Rất ít game có thể được thiết kế, code, kiểm tra và phát hành dưới

6 tháng (mặc dù với sự bùng nổ của các dòng game trên Internet – mạng xã hội và cả game trên di động như hiện nay). Tuy vậy, game được phát triển trong thời gian dài hơn 2 năm sẽ dễ rơi vào vòng xoáy trì hoãn, gặp phải vấn đề thay đổi nhân sự, các tính năng bị sao chép bởi game khác và sự lạc hậu trong kỹ thuật. Bất kỳ vấn đề nào cũng đều dẫn đến game phải được thiết kế lại, xây dựng lại và kết quả là bị trễ kế hoạch nghiêm trọng.

Tính “nguy hiểm” của việc phát triển là cảm giác có nhiều thời gian khi chúng ta bắt đầu. Do đã xây dựng một kế hoạch tốt nên ta sẽ dễ dàng nghĩ rằng mọi thứ có thể được thực hiện. Giai đoạn này của dự án cũng nguy hiểm giống như nghỉ hè vậy. Lúc đầu, chúng ta thấy nhiều tuần và tháng trải dài trước mắt mình, và thấy có khá nhiều thời gian để hoàn tất mọi thứ mình có trong đầu. Sau đó rất nhanh, khi thời hạn đến gần, chúng ta sẽ tự hỏi mình thời gian đã đi đâu mất ! và ta bắt đầu “điên cuồng” tìm cách nhồi nhét mọi thứ vào.

Mẹo để sống sót qua mỗi nguy này là chia các công việc lớn thành các nhiệm vụ nhỏ có thể quản lý vào theo dõi thường xuyên được. Chúng ta sẽ không thể biết mình có đang kịp tiến độ không nếu không theo dõi công việc. Tốt nhất ta nên theo dõi thường xuyên ít nhất là hằng tuần.

Một trong những kỹ thuật quản lý công việc hiệu quả là giao cho mỗi lập trình viên tự quản lý danh sách công việc của anh ta cùng với ước lượng thời gian. Các danh sách đơn lẻ này sẽ được tổng hợp thành một danh sách tổng để nhìn tổng quát thấy được thời gian cần thiết để hoàn thành toàn bộ dự án. Phương pháp này đặc biệt hữu dụng để biết được việc của người này có làm ảnh hưởng đến người khác không. Nếu có, người đó đang nằm trong đường găng của dự án và chúng ta phải theo sát danh sách công việc của anh ta để xem có thể giao bớt việc cho người khác hay không.

Phương pháp này cũng có ưu điểm là để cho lập trình viên hay họa sĩ chịu trách nhiệm với ước lượng của riêng anh ta thay vì áp đặt từ trên xuống. Điều này sẽ tăng sự dính kết của mọi người vào dự án và giúp mọi người giảm khả năng bị hù dọa.

Nếu chúng ta là một nhà phát triển bên ngoài làm việc cho một nhà phát hành, tiến độ của chúng ta sẽ được theo dõi bằng các mốc đã được thống nhất trong hợp đồng. Áp lực theo đúng tiến độ dự án khá rõ ràng – nếu chúng ta không theo kịp kế hoạch ta sẽ không được trả tiền.

Các mốc dự án được thiết lập ở đầu quá trình phát triển và thường có các buổi gặp hàng tháng để báo cáo tình trạng dự án trong đó tất cả producers ngồi lại với nhau để xem qua tình trạng chi tiết các dự án. Trong các buổi đánh giá này, các quản trị viên cấp cao sẽ kiểm tra xem các dự án có đúng tiến độ không và producer có làm việc để giảm thiểu các nguy cơ gây ảnh hưởng đến dự án trong tương lai không.

Giai đoạn Alpha

Định nghĩa giai đoạn Alpha có thể khác nhau tùy theo công ty nhưng thông thường đây là thời điểm mà game có thể chơi được từ đầu tới cuối. Có thể vẫn còn có nhiều điểm chưa hoàn thiện, một số hình ảnh chưa được ráp vào nhưng engine, hệ thống giao diện người dùng và những hệ thống chính khác đã xong.

Khi bước vào giai đoạn alpha, tiêu điểm chuyển từ xây dựng sang kết thúc, từ kiến tạo sang tinh chỉnh. Đây là thời điểm để nhìn kỹ lưỡng vào tập tính năng của game để quyết định có cần bỏ đi không để đảm bảo kế hoạch. Lúc này cũng là lúc có thêm nhiều tester tham gia dự án để “săn” bug. Đây cũng là lúc game được nhìn thấy và đánh giá bởi những người bên ngoài team phát triển.

Tin tốt về giai đoạn alpha là đây là điểm bắt đầu của sự kết thúc. Tin xấu là đi đến điểm kết thúc hiếm khi đơn giản.

Giai đoạn beta

Tại giai đoạn beta, tất cả tài nguyên đều được ráp vào game, mọi quá trình phát triển sẽ dừng lại, điều duy nhất còn xảy ra là sửa lỗi. Vẫn có thể cần tút lại một số hình ảnh, cần viết lại một số đoạn văn nhưng mục

tiêu chính của giai đoạn này là ổn định dự án và tiêu diệt càng nhiều bug càng tốt trước khi phát hành sản phẩm.

Nếu chúng ta đang làm một game console cần phải được phê duyệt bởi nhà sản xuất console, những tuần cuối của giai đoạn beta sẽ bao gồm việc gửi game đến công ty đó để các testers của họ kiểm định game thỏa mãn những quy định về chất lượng của họ. Nếu là một game trên nền PC, nó có thể được gửi đi cho những công ty test bên ngoài để test tính tương thích. Quá trình test tương thích sẽ cho bạn biết những phần cứng nào (kết hợp với những phần mềm nào) không tương thích với game của chúng ta.

Phần cuối của giai đoạn beta thường được gắn liền với thuật ngữ “crunch time”. Trong suốt những tuần này, mọi người ăn nằm tại văn phòng suốt ngày, ngủ trên mặt bàn, ăn uống tạm bợ, uống một lượng cà phê khổng lồ và trở thành người lạ với chính gia đình của họ. Đó là một thế giới chạng vạng kỳ lạ mà thứ duy nhất có nghĩa là hoàn thành game.

Nếu mọi thứ suông sẻ, chúng ta sẽ có được một đội ngũ đầy nhiệt huyết có niềm tin mãnh liệt rằng họ đang làm việc để đạt được điều gì đó đặc biệt và sẵn sàng hy sinh những khía cạnh khác của cuộc sống để nhìn thấy đứa con tinh thần của họ ra đời. Những người này làm việc chăm chỉ bởi vì họ **muốn** như vậy, bởi vì nó quan trọng đối với họ và nó là niềm vui. Động cơ của họ đến từ ham muốn bên trong thay vì áp lực từ bên ngoài. Nếu chúng ta đã từng làm việc với một tập thể để đạt được một mục tiêu đáng ca ngợi, ta sẽ có một cảm giác thật lâng lâng tuyệt vời.

Ngược lại, nếu mọi thứ không như ý, ta sẽ tạo ra những con người chăm chỉ chỉ vì sợ mất việc, họ không quan tâm đến game thế nào – chỉ cần hoàn thành nó, những người cảm thấy bị lợi dụng và vắt kiệt sức lao động.

Khi mọi thứ trở nên thật tồi tệ, “crunch time” sẽ trở thành “death march”, một khoảng thời gian cố sức trong tuyệt vọng kéo dài hơn một tháng. Hãy tránh tình huống này bằng mọi giá. Lợi ích của việc làm thêm giờ sẽ bị mất đi, thay vào đó là vô số lỗi lầm được tạo ra do sự

kiệt sức. Cả nhóm sẽ xuống tinh thần và tan rã. Làm như vậy chỉ dẫn đến việc game sẽ bị trì hoãn lâu hơn nữa thôi. Nếu muốn đập bàn và tuyên bố “chúng ta có thể đạt được mục tiêu nếu tất cả chúng ta làm thêm giờ liên tục trong 2 tháng!”, hãy dừng lại và suy nghĩ trước khi quá muộn.

Giai đoạn crunch-time hầu như khó tránh trong mọi dự án và khi nó đến, hãy chuẩn bị đối phó thật cẩn thận. Khi hết thời gian, mọi người sẽ càng nhạy cảm và nóng tính. Một trong những phần khó khăn của làm game tại thời điểm này là làm sao đưa ra được những quyết định đúng đắn khi mọi người đang ở trong không khí căng thẳng, quá ít thời gian để làm việc và để ngủ. Lúc này, hầu như không còn khái niệm quyết định “đúng” nữa. Chỉ còn những quyết định “cần thiết” (đôi khi chỉ là quyết định) để mọi người tiếp tục đi tới để hoàn tất game.

Cuối cùng, khi đến giai đoạn được đóng gói phiên bản cuối cùng để chuẩn bị phát hành, bạn nên có ít nhất hai người làm song song và kiểm tra chéo công việc lẫn nhau – hoặc sử dụng một quá trình đóng gói tự động đã được sử dụng đánh tin cậy nhiều lần trong suốt quá trình phát triển. Lúc này dựa trên một người đã mờ mắt vì thức đêm để nhớ tất cả chi tiết quan trọng khi đóng gói là một công thức để tạo nên thảm họa.

“Đóng băng” code (Code-Freeze)

Trong những ngày gần cuối của giai đoạn beta, chúng ta rất có thể bước vào giai đoạn “đóng băng code”, khi tất cả mọi thứ đã xong và mọi người đã sẵn sàng đóng gói phiên bản cuối cùng của game vào đĩa chủ (master-disc). Các đĩa này sẽ được gửi đi để kiểm tra lần cuối. Chỉ có những bug thuộc dạng kinh khủng (showstopper) được phát hiện ra lúc này mới dẫn đến việc sửa lại code.

RTM (Chuyển giao đến nhà sản xuất – Release to Manufacturer)

Game sẽ được chuyển giao đến nhà sản xuất để tạo ra hàng triệu bản sao đưa ra thị trường khi đĩa chủ đã được kiểm tra cẩn thận và được chấp nhận. Cuối cùng thì mọi người đã có thể ăn mừng!

Vá lỗi

Đối với các game PC, việc vá lỗi sau khi phát hành game là gần như không thể tránh khỏi. Nhiều ý kiến trên Internet cho rằng điều này là không cần nếu nhà phát triển kiểm tra chất lượng sản phẩm thật kỹ lưỡng trước khi phát hành. Tuy vậy, trong một thế giới có đến hàng ngàn các kiểu phối hợp phần cứng khác nhau, hầu như không thể kiểm tra game trong tất cả mọi trường hợp. Cũng vậy, nếu một game phức tạp thì hầu như rất rất khó để có thể hình dung hết tất cả mọi khả năng game thủ chơi game như thế nào. Nếu một game thủ nào đó phát hiện ra game của ta không thể chạy được trên một tổ hợp BIOS, card đồ họa, card âm thanh, màn hình, CPU, hệ điều hành, bàn phím, chuột.. cụ thể nào đó – nhà phát triển sẽ tìm cách truy lùng ra nguồn gốc của vấn đề. Nếu vấn đề đủ lớn, nhà phát triển sẽ phát hành ra các bản vá lỗi.

Nâng cấp

Bảng nâng cấp / bổ sung khác với bản vá lỗi. Bảng nâng cấp sẽ mang lại những nội dung mới để bổ sung vào game. Các công ty đưa ra các bản nâng cấp với nhiều lý do khác nhau. Thông dụng nhất là để kéo dài vòng đời của game gốc (nếu các bản add-on xuất hiện, các nhà bán lẻ sẽ phải giữ game của bạn trên kệ lâu hơn). Để làm bảng nâng cấp, chúng ta sẽ muốn giữ một phần của đội ngũ ban đầu làm việc đó đồng thời đưa một số thành viên khác sang các giai đoạn đầu của những dự án tiếp theo.

Các bảng nâng cấp được xem là các dự án nhỏ và cũng cần được xây dựng như một dự án thật với đầy đủ quy trình như kiểm lỗi, đặt ra các mốc kế hoạch và tất cả những yếu tố khác của quy trình quản lý phát triển phần mềm.

CHƯƠNG 2

NHẬP MÔN LẬP TRÌNH WINDOWS VÀ DIRECTX

2.1. LẬP TRÌNH WINDOWS BẰNG C++

Xây dựng một ứng dụng trên Windows có thể rất đơn giản hoặc rất phức tạp tùy thuộc vào loại ứng dụng bạn muốn xây dựng. Nếu chúng ta đã từng có nền tảng lập trình và có kinh nghiệm viết ứng dụng bằng C++, có lẽ cũng cần hiểu rằng chúng ta sẽ xây dựng được một ứng dụng với giao diện đồ họa (GUI) cực kỳ phức tạp trên Windows với menu, forms và hàng trăm các loại controls khác nhau cho những mục đích khác nhau.

Windows là một hệ điều hành đa nhiệm (multi-tasking) khá mạnh với cơ chế truyền thông điệp làm nền tảng. Nhiều người thường lầm tưởng Windows là hệ điều hành hướng đối tượng (message-driven) nhưng sự thật không phải vậy. Bên dưới Windows là hệ thống các thông điệp (message) chứ không phải các đối tượng (objects).

2.1.1. Hệ thống thông điệp của Windows (Windows Messaging)

Hệ điều hành – trong một ngữ cảnh hẹp – hoạt động tương tự như hệ thống thần kinh của con người (mặc dù nó không thể phức tạp và tinh xảo như não bộ người được!). Tuy nhiên, nếu chúng ta trừu tượng hóa hoạt động của não bộ (và bỏ qua yếu tố “linh hồn”) ta sẽ thấy các xung (impulse) chạy qua lại giữa các dây thần kinh để truyền tín hiệu từ các giác quan đến não và từ não đến các cơ bắp.

Lấy ví dụ, nếu ta lấy một ngón tay phải chạm vào tay trái, chuyện gì sẽ xảy ra? Chúng ta sẽ “cảm nhận” sự đụng chạm trên tay trái của mình. Tuy nhiên, sự thật không phải là cái tay chúng ta “cảm nhận” được cái chạm đó mà não của mình cảm thấy điều đó. Bộ não đã xử lý **tín hiệu** đến từ tay trái để chúng ta nhận ra nguồn gốc của sự va chạm.

Vậy thì điều này liên quan gì đến hệ thống thông điệp của Windows ? Ví dụ ở trên khá giống như cách làm việc của hệ thống thông điệp của Windows. Một sự kiện bên ngoài – chẳng hạn như người dùng click chuột, sẽ tạo ra một tín hiệu điện tử từ con chuột, truyền đến cổng USB

của máy tính rồi đi đến hệ thống bus dữ liệu của máy tính (hệ thống bus này tương tự như hệ thống dây thần kinh của con người). Từ đây, hệ điều hành sẽ “bắt” tín hiệu và tạo ra một **thông điệp** (message) rồi gửi thông điệp này đến cho ứng dụng đang chạy. Chương trình khi nhận được thông điệp sẽ “phản ứng” lại bằng cách hiện một hành động như hiện một hình ảnh, tắt cửa sổ hoặc chơi một đoạn nhạc, v.v.. tùy theo ứng dụng đó được thiết kế để làm gì.

2.1.2. Đa nhiệm

Windows là hệ điều hành đa nhiệm “đúng chuẩn” (pre-emptive multi-tasking) – nghĩa là chúng ta có thể chạy nhiều chương trình cùng lúc. Windows thực hiện điều này bằng cách thi hành từng chương trình đang mở một tí – trong một khoảng thời gian cực ngắn – gọi là time slicing - tính bằng phần ngàn giây rồi chuyển sang các chương trình đang chạy khác. Windows sẽ tạo ra một không gian ảo (có thể hình dung như một máy tính con nhỏ) cho mỗi chương trình trong bộ nhớ. Mỗi khi nhảy sang chương trình khác, trạng thái thực thi của chương trình hiện tại (bao gồm cả trạng thái bộ xử lý, giá trị các thanh ghi và toàn bộ dữ liệu của chương trình) sẽ được lưu lại để khi được chạy lại lần tới, toàn bộ trạng thái này sẽ được phục hồi đầy đủ, đảm bảo chương trình sẽ được chạy lại mà không bị mất mát dữ liệu hay bị sự cố gì.

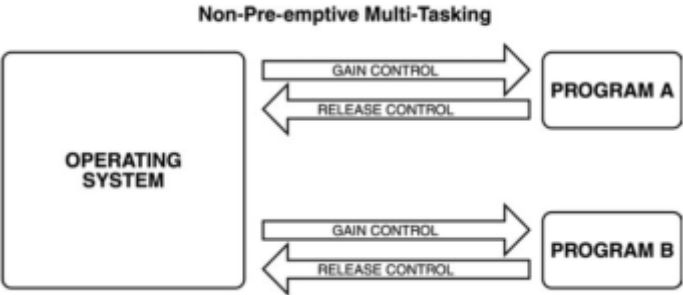
Ghi chú : Việc “nhảy” qua nhảy lại giữa các chương trình nghe có vẻ lãng phí năng lực xử lý của máy tính. Tuy nhiên, với các bộ vi xử lý ngày nay, chỉ cần 1 vài phần ngàn giây thì tẻ lắm bộ xử lý đã thực thi xong hàng trăm ngàn lệnh. Các bộ xử lý mới nhất có thể xử lý hàng triệu lệnh trong vài phần ngàn giây.

Có lẽ chúng ta nên dành chút thời gian ôn lại lịch sử một chút bằng cách nói về hệ điều hành đa nhiệm “chưa đúng chuẩn (hay còn gọi là **đa nhiệm hợp tác**)”. Các Windows đời 3.x về mặt bản chất chưa phải là hệ điều hành đúng nghĩa. Lúc đó, Windows chỉ là các chương trình “công nghệ cao” chạy trên nền tảng hệ điều hành MS-DOS nên Windows không thực sự toàn quyền điều khiển hoạt động máy tính. Thời đó,

chúng ta có thể dễ dàng viết một chương trình trên Windows 3.x để chiếm dụng toàn bộ thời gian chạy của CPU mà không trả lại CPU cho các chương trình khác chạy. Thậm chí, nếu cần thiết, chúng ta có thể dễ dàng tạo ra một chương trình làm treo toàn bộ hệ điều hành (!). Các chương trình thời đó để được cấp chứng nhận “Windows Logo” phải đảm bảo rằng nó phải giải phóng các tài nguyên của máy tính cho các chương trình khác xài chung. Windows 95 là phiên bản đầu tiên trong dòng Windows tạo nên cuộc cách mạng khi thực sự đạt được khả năng đa nhiệm “đúng chuẩn”.

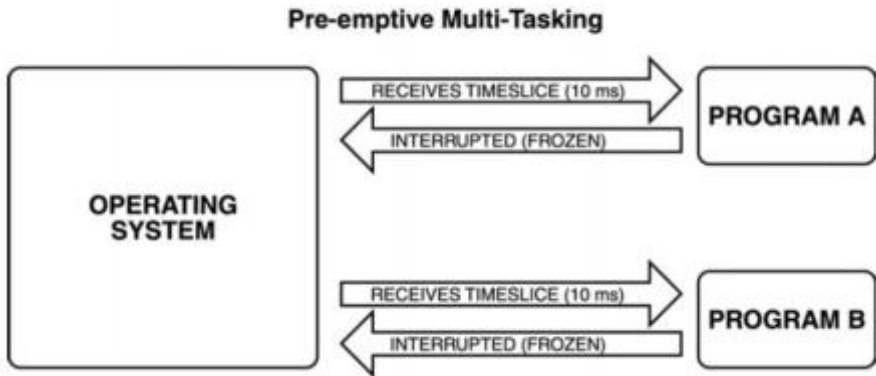
Để đạt được khả năng đa nhiệm “đúng chuẩn”, hệ điều hành phải có một lõi ở mức rất thấp ở hệ thống để quản lý toàn bộ hệ thống phần cứng máy tính và không cho phép bất cứ chương trình nào chạy trên hệ điều hành có quyền chiếm quyền kiểm soát máy tính. Với hệ điều hành đa nhiệm, nếu máy tính có sử dụng các bộ xử lý đa lõi hoặc nhiều bộ xử lý sẽ có thuận lợi lớn về mặt gia tăng hiệu năng cho ứng dụng – cụ thể là các game.

Hình dưới đây minh họa cách hoạt động của hệ điều hành đa nhiệm “chưa chuẩn”. Bạn lưu ý cách mỗi chương trình nhận toàn quyền kiểm soát CPU và phải trả lại điều khiển để hệ thống có thể hoạt động đúng. Các chương trình cũng phải lưu ý không được chiếm dụng CPU quá lâu nếu không sẽ làm cho toàn hệ thống bị đơ trong lúc CPU bị chiếm dụng. Nói tóm lại, việc chia sẻ CPU giữa các chương trình là “tự nguyện”.



Hình 2.1. Hệ điều hành không đa nhiệm

Trong hình minh họa tiếp theo về hệ thống đa nhiệm đúng chuẩn. Như bạn cũng thấy, sơ đồ nhìn cũng tương tự nhưng ở đây hệ điều hành điều khiển mọi thứ và không cần phải chờ các chương trình “chơi đẹp” trong việc chia sẻ CPU. Hệ điều hành chỉ cần tạm dừng chương



Hình 2.2. Hệ điều hành đa nhiệm

trình khi thời gian chạy được hệ điều hành cấp phát đã kết thúc rồi sau đó sẽ quay lại cấp phát thời gian chạy cho chương trình đó sau khi đã cho một vòng các chương trình khác thi hành.

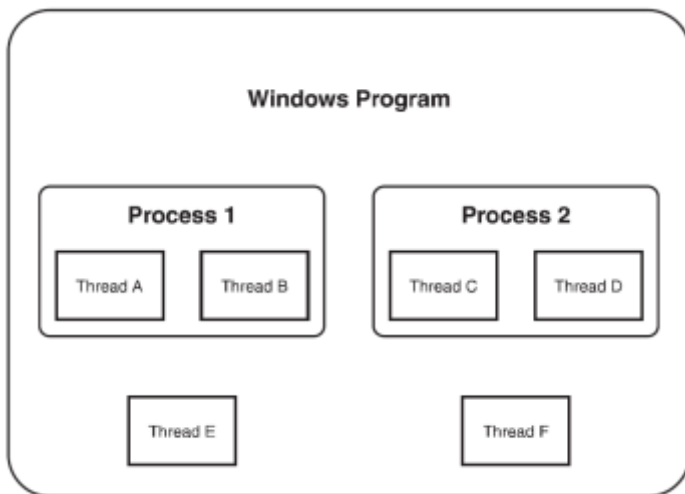
2.1.3. Đa tiểu trình (multi-threading)

Đa tiểu trình là quá trình phân chia một tiến trình – process - (một chương trình trên đĩa khi được thực thi sẽ được gọi là một tiến trình) thành nhiều phần độc lập cùng hoạt động với nhau để cùng đạt được một mục đích (hoặc đôi khi để đạt được những mục đích độc lập nhau một cách song song thay vì phải tuần tự). Cơ chế này khác so với cơ chế đa nhiệm ở cấp độ hệ điều hành. Đa tiểu trình giống như đa – đa nhiệm 😊 trong đó mỗi chương trình có những phần độc lập cùng chạy song song. Những “phần” cùng chạy song song này cũng tuân theo quy luật chia thời gian (time-slicing) do hệ điều hành quản lý. Chương trình Windows và tất cả thread của nó đều không có “cảm giác” mình bị hệ điều hành “chia cắt” hay phân phối thời gian thực thi mà ngược lại có

cảm giác có toàn quyền kiểm soát hệ thống và đang được chạy liên tục (do việc “nhảy” được diễn ra cực nhanh).

Một ví dụ về sự tiện lợi của cơ chế multi-thread trong game là game đánh cờ. Chương trình chính có thể tạo ra một thread riêng để tính trước các nước cho máy trong khi chờ người chơi suy nghĩ về nước đi kế tiếp. Mặc dù chúng ta vẫn có thể thực hiện được việc này bằng vòng lặp đợi gõ phím hoặc chuột như các chương trình cũ nhưng cách dùng thread sẽ có nhiều lợi thế hơn.

Cơ chế multi-thread cực kỳ “lợi hại” trong lập trình game. Các công việc phải thực hiện tuần tự trong vòng lặp game (game loop – khái niệm quan trọng trong khung chương trình game – sẽ được giới thiệu chi tiết ở các chương sau) có thể được thay thế bằng các thread chạy độc lập và các thread này sẽ liên lạc với chương trình chính để đồng bộ dữ liệu. Hiện nay các game engine đời mới đều hỗ trợ multi-threading một cách tự nhiên – nghĩa là đều hỗ trợ các máy có nhiều bộ xử lý. Điều này rất có ý nghĩa với gamer, mua máy càng nhiều bộ xử lý hoặc xử lý có càng nhiều lõi thì game sẽ càng chạy nhanh hơn. Cơ chế này lại càng hữu dụng đối với các hệ thống phần mềm server của các game mạng. Server game mạng cần phải xử lý rất nhiều nếu có nhiều gamer chơi game cùng lúc, do đó, nếu máy chủ càng có nhiều bộ xử lý thì server game sẽ phục vụ được càng nhiều gamer cùng lúc.



Hình 2.3. Cấu trúc một chương trình windows

2.1.4. Xử lý sự kiện

Lúc này bạn đọc có thể hỏi “làm sao Windows xử lý và quản lý nổi một lượng chương trình và thread chạy song hành lớn như thế được?” Windows quản lý việc này – đầu tiên – bằng cách yêu cầu các chương trình phải được xây dựng theo cơ chế sự kiện. Tiếp theo, Windows sử dụng các thông điệp ở mức toàn hệ thống (toàn cục) để liên lạc giữa các chương trình. Các thông điệp của Windows là các gói dữ liệu nhỏ do Windows gửi đến các chương trình đang chạy với 3 thông tin cơ bản (1) định danh cửa sổ (window handle) (2) định danh thể hiện (instance) của chương trình và (3) loại thông điệp – thông báo cho chương trình đó biết loại sự kiện gì đã diễn ra. Các sự kiện thông thường là thao tác người dùng (chuột, bàn phím) nhưng cũng có thể là các tín hiệu từ các cổng giao tiếp trên máy tính hoặc từ mạng.

Ghi chú

Window handle: là một số nguyên Windows dùng để định danh chính xác từng cửa sổ

Instance: một chương trình trên đĩa khi được nạp lên bộ nhớ để thi hành được gọi là một instance – thể hiện. Cùng một chương trình có thể được chạy cùng lúc nhiều lần nên có thể có nhiều instance cùng lúc. Ví dụ là chương trình MS Word – ta có thể mở cùng lúc nhiều cửa sổ Word giống hệt nhau cùng lúc.

Các chương trình Windows phải kiểm tra tất cả mọi thông điệp được hệ điều hành gửi đến nó (thông qua các trình xử lý sự kiện) và xác định xem có cần phải làm gì với thông điệp đó hay không. Những thông điệp không cần phải phản ứng sẽ được gửi trả lại cho Windows để được gửi đi đến những nơi khác. Hãy tưởng tượng các thông điệp như các con cá – khi bắt được cá quá nhỏ mà ta không thích thì ta sẽ thả lại xuống sông cho ai quan tâm thì bắt lên. Ngược lại, nếu cá to thì ta sẽ giữ lại và “xử” nó.

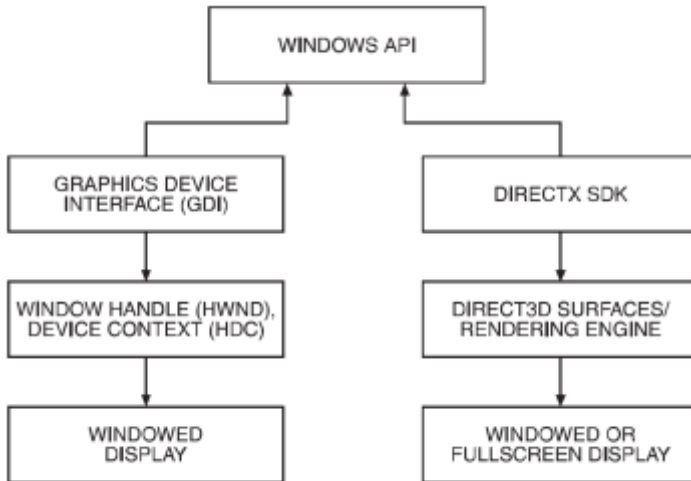
Sau này, khi đã “vọc” quen với lập trình trên Windows và đặc biệt là hệ thống thông điệp bạn đọc sẽ nhận ra rằng cơ chế này dành cho ứng dụng, không phải cho game. Tuy nhiên, vẫn có cách để lách vào hệ thống thông điệp này để đưa vào các đoạn mã nguồn thực hiện những ý đồ của game – sẽ được giới thiệu ở sau. Chúng ta sẽ tìm hiểu cách xây dựng khung chương trình Windows cũng như cách làm việc với hệ thống thông điệp của Windows ngay trong phần sau.

2.2. TỔNG QUAN VỀ DIRECTX

Chúng ta có thể nghe nhiều về DirectX thông qua chơi game hoặc đọc trên báo. Tuy nhiên có một sự thật lạ lùng là rất ít người thực sự biết DirectX là gì. Rất nhiều người hay nhầm lẫn DirectX là game engine, thư viện đồ họa hoặc là một bộ thư viện hỗ trợ làm game. Bản chất nhiệm vụ chính của DirectX là cung cấp một giao diện (interface) cho phép ứng dụng có thể truy cập trực tiếp đến các tính năng **cấp thấp** (low-level) của **nhiều** loại phần cứng – trong đó có card đồ họa – từ đó xây dựng lên một thư viện các hàm thống nhất và ổn định dành cho các game không dựa vào nền tảng Windows API hoặc GDI. Do cho phép

ứng dụng truy cập trực tiếp vào phần cứng nên DirectX nhanh hơn đáng kể so với GDI và đây là lý do khiến cho DirectX rất phù hợp cho game.

DirectX được tính hợp chặt với Windows và sử dụng rất nhiều thư viện nền tảng của Windows API nên nó sẽ không làm việc được trên hệ điều hành nào khác. Kiến trúc tổng quan của DirectX được minh họa trong hình dưới đây



Hình 2.4. Kiến trúc Windows API

Các thành phần chính của DirectX bao gồm

DirectX graphic: Đây là thành phần đồ họa của DirectX cho phép truy cập vào các card có tính năng tăng tốc đồ họa 3D cũng như việc vẽ cực nhanh các hình 2D thông qua thành phần DirectDraw (*DirectDraw đã được loại bỏ hoàn toàn từ DirectX 8 trở đi*). Thông qua giao tiếp Direct3D, các game 3D sẽ truy cập và sử dụng được các tính năng mới nhất trên card màn hình. DirectX 9 vẫn cung cấp khả năng tương thích ngược cho DirectDraw (nghĩa là các ứng dụng cũ dùng DirectDraw vẫn chạy được trên DirectX 9) nhưng lời khuyên là nếu dùng DirectX 9 trở lên, chúng ta không nên tiếp tục dùng DirectDraw mà chỉ nên dùng các

tính năng của Direct3D hỗ trợ cho cả 2D và 3D (chúng ta sẽ tìm hiểu cách dùng 2D trên Direct3D ở các chương sau)

Direct Sound: thành phần này bao gồm các giao tiếp để chơi các file âm thanh cũng như các loại file nhạc hay MIDI; ứng dụng chỉ cần sử dụng duy nhất một giao diện này là có thể hỗ trợ tất cả loại card âm thanh, định dạng file âm thanh trên PC; bao gồm cả việc phối nhạc, âm trên nhiều kênh ở chế độ realtime. Nói tóm lại, DirectSound sẽ “bao sân” toàn bộ nhu cầu về âm nhạc và âm thanh trên PC cho chúng ta !

Direct Input : thành phần này cho phép truy cập đến các thiết bị ngoại vi trên máy tính như bàn phím, chuột, joystick, gamepad và nhiều loại thiết bị ít thông dụng như tay lái xe (cho các game đua xe), cần lái máy bay (cho các game mô phỏng bay), v...v...

Direct Play: thành phần này cung cấp giao diện cho các trò chơi mạng hỗ trợ cho cơ chế lobby (lobby giống như cái sảnh ở khách sạn, nơi gamer tập trung lại, chat, trao đổi với nhau trước khi vào chơi game). DirectPlay được tối ưu cao độ và hiệu quả trong việc quản lý một lượng người lên đến 32 người trên một server. Mô hình này ổn cho đa phần game. Tuy nhiên, dễ thấy là ta không thể dùng DirectPlay để xây dựng các game MMO có số lượng người chơi cực lớn như hiện tại.

2.3. CẤU TRÚC CHƯƠNG TRÌNH WINDOWS BẰNG C++

Mọi chương trình Windows tối thiểu phải có một hàm tên là **WinMain**. Hầu hết các chương trình Windows cũng đều phải có một hàm callback gọi là **WinProc** để xử lý một số các sự kiện do hệ điều hành Window gửi đến. Nếu chúng ta viết một chương trình Windows đầy đủ (chẳng hạn như các chương trình thương mại như Word, Excel,...) chúng ta sẽ có một hàm **WinProc** cực kỳ lớn và phức tạp để quản lý nhiều loại trạng thái và sự kiện diễn ra đối với ứng dụng. Tuy nhiên, đối với một chương trình DirectX, thực ra ta không cần phải đụng đến việc xử lý sự kiện vì quan tâm của chúng ta dành cho việc tương tác với DirectX – mà nó vốn đã cung cấp các hàm của riêng nó để xử lý các sự kiện.

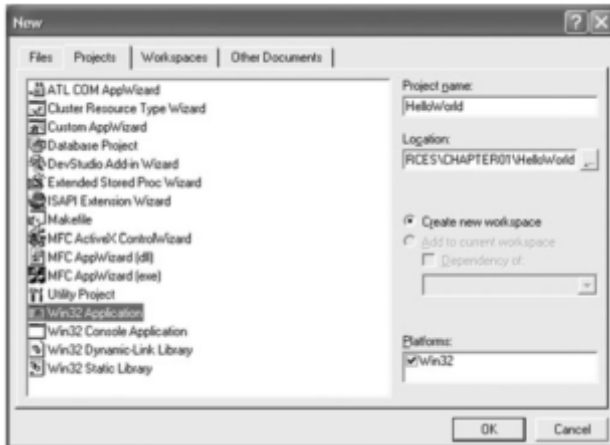
DirectX hoạt động theo cơ chế “kéo” (polled) – nghĩa là chúng ta đi lấy dữ liệu thay vì chờ nhận dữ liệu gửi đến từ đâu đó (như trong trường hợp WinProc). Chẳng hạn: khi làm việc với bàn phím hay chuột, chúng ta sẽ phải liên tục gọi hàm để biết trạng thái phím nào đã bị thay đổi.

2.3.1. Tạo một project Win32

Tất cả chương trình mẫu trong giáo trình này đều dùng chung một kiểu *project*, do đó một khi ta đã biết cách tạo project bằng Visual C++ thì ai cũng có thể dùng cùng cách này để tạo tất cả project cho các chương trình mẫu.

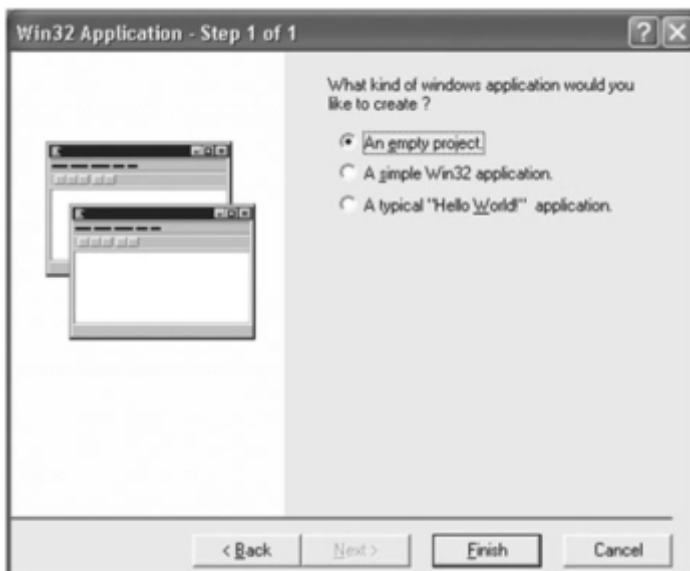
Vậy **Project** là gì? Bản chất **project** chỉ là một file trên đĩa dùng để lưu thông tin về tất cả các tập tin mã nguồn của một chương trình. Một số các chương trình ví dụ trong giáo trình này đều chỉ có một file mã nguồn duy nhất nhưng trên thực tế các chương trình game đều phải có nhiều file mã nguồn. Chẳng hạn, ta sẽ có một tập tin thư viện cho các hàm vẽ bằng Direct3D, một file thư viện cho âm thanh bằng DirectSound, một file thư viện code xử lý bàn phím bằng DirectInput và một file code chính của game. Project sẽ giúp quản lý được tất cả các file mã nguồn thông qua môi trường phát triển IDE của Visual Studio.

Loại project mà ta cần tạo ra thuộc loại **Win32 application** như minh họa ở hình dưới đây. Hãy chọn menu **File > New** để mở hộp thoại này.



Hình 2.5. Hộp thoại New Project

Sau khi đặt tên cho Project, nhấn OK. Visual Studio sẽ hỏi chúng ta loại project cần khởi tạo là gì. Do chúng ta muốn tìm hiểu cách xây dựng chương trình Windows từ đầu nên ta sẽ chọn “Empty project” – một project rỗng để ta tự thêm các tập tin mã nguồn của mình về sau.



Hình 2.6. Tạo project rỗng

Bây giờ chúng ta đã có một project rỗng. Tiếp theo là ta thêm một file code vào trong project này. Chọn menu **File > New** . Lần này chọn dòng **C++ source file**, đặt tên cho file ở text box bên phải rồi nhấn OK.

Như vậy chúng ta sẽ có một project với một file mã nguồn duy nhất. Vừa đủ để tạo chương trình HelloWorld nổi tiếng của chúng ta. Dưới đây là mã nguồn của một chương trình Windows hiển thị một hộp thoại với dòng chữ HelloWorld. Đơn giản đến kinh ngạc. Tuy nhiên, chương trình này đã bỏ qua các bước khó chịu như khởi tạo cửa sổ, menu, hàm WinProc..

```
#include <windows.h>

int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE
hPrevInstance,
LPSTR lpCmdLine, int nShowCmd)
{
    MessageBox(
    NULL,
    "Welcome!",
    "Hello, World",
    MB_OK | MB_ICONEXCLAMATION);

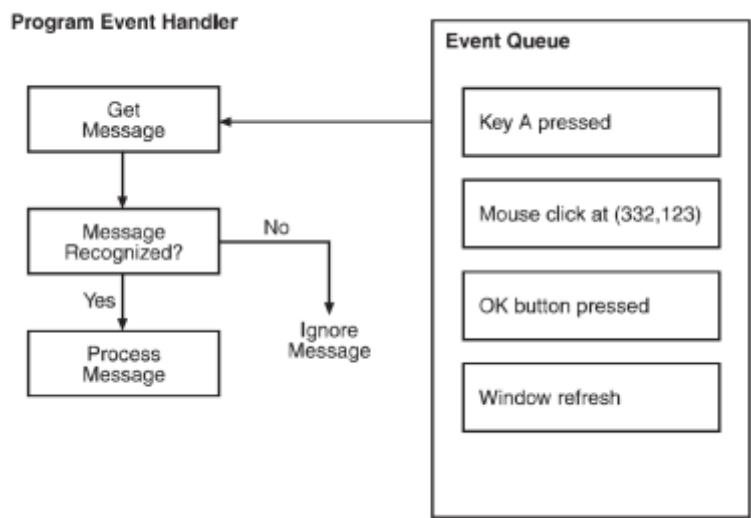
    return 0;
}
```

Bấm F5 để dịch và chạy đoạn chương trình này, chúng ta sẽ thấy một tập tin .exe được tạo ra và lưu trong thư mục tên là **Debug**

Điều cần để ý ở đoạn mã nguồn này là hàm **WinMain** đây là đầu vào của tất cả ứng dụng trên Windows (tương tự như hàm **main** ở ứng dụng C++ ở chế độ console).

Hàm WinMain chưa hề được thay đổi từ lúc Windows còn hoạt động ở chế độ 16-bit để đảm bảo tính tương thích ngược. Nhiệm vụ của WinMain là thiết lập chương trình, thiết lập vòng lặp thông điệp chính (message loop). Nhiệm vụ của vòng lặp này là để nhận các thông điệp

do hệ điều hành Windows gửi đến cho chương trình (như chúng ta tìm hiểu ở đầu chương). Khi nhận được thông điệp, vòng lặp này sẽ phân bổ lại thông điệp cho hàm **WinProc** , hàm này sẽ có các đoạn code xử lý các thông điệp nhận được (thông thường nhất là xử lý các thông điệp bàn phím, chuột). Hình dưới đây minh họa khái niệm này



Hình 2.7. Quy trình xử lý thông điệp

2.3.2. Các tham số của WinMain

Hàm WinMain có khai báo như sau

```
int WINAPI WinMain(
    HINSTANCE hInstance,
    HINSTANCE hPrevInstance,
    LPTSTR lpCmdLine,
    int nCmdShow);
```

Ý nghĩa của các tham số như sau:

HINSTANCE hInstance. Tham số này định danh *thể hiện* của chương trình được nạp và thi hành trên bộ nhớ (vì một chương trình trên đĩa có thể được thực thi nhiều lần, cùng lúc). Kiến trúc của hệ điều hành Windows cho phép các *thể hiện* dùng chung *mã chương trình* trong bộ nhớ để tiết kiệm bộ nhớ – trong khi dữ liệu của các thể hiện được lưu trữ riêng rẽ.

Tham số `hInstance` cho phép ta biết mình đang chạy ở *instance* nào.

HINSTANCE hPrevInstance : Tham số này cho biết định danh của thể hiện vừa được gọi *trước* của chương trình.

Một ứng dụng thường của *hPrevInstance* là để đảm bảo chỉ có một thể hiện duy nhất của chương trình. Nếu “chúng ta” là thể hiện đầu tiên thì `hPrevInstance` sẽ là null, ngược lại `hPrevInstance` sẽ khác null. Điều này khá hợp với game vì chúng ta thường sẽ không muốn người dùng mở cùng lúc nhiều cửa sổ của cùng một game.

LPSTR lpCmdLine : Tham số thứ 3 là chuỗi chứa các tham số dòng lệnh truyền cho chương trình. Tham số dòng lệnh thường dùng để thiết lập hoặc truyền thêm dữ liệu gì đó cho chương trình. Chẳng hạn: `notepad C:\tmp\test.txt` sẽ truyền tham số là tên tập tin cần mở.

int nCmdShow : tham số này gợi ý cho chương trình nên tạo ra cửa sổ thể nào khi thực thi.

Giá trị trả về của *WinMain* là một số nguyên thuộc kiểu **int WINAPI** . Giá trị trả về bằng 0 nghĩa là có lỗi và khác 0 nghĩa là mọi thứ bình thường.

2.3.3. Khung hàm WinMain đầy đủ

Ngay dưới đây là khung của một hàm *WinMain* trong một chương trình tiêu biểu. Chúng ta sẽ lần lượt đi qua từng thành phần của đoạn code này.

```
//đầu vào của chương trình Windows
int WINAPI WinMain(HINSTANCE hInstance,
```

```

        HINSTANCE hPrevInstance,
        LPSTR      lpCmdLine,
        int        nCmdShow)
{
    MSG msg; // khai báo biến chứa thông điệp

    // đăng ký lớp của sổ
    MyRegisterClass(hInstance);

    // khởi tạo ứng dụng
    if (!InitInstance (hInstance, nCmdShow))
        return FALSE;

    //khởi động bộ tạo số ngẫu nhiên
    srand(time(NULL));

    // vòng lặp thông điệp chính
    while (GetMessage(&msg, NULL, 0, 0))
    {
        TranslateMessage(&msg);
        DispatchMessage(&msg);
    }

    return msg.wParam;
}

```

Đây là khung đơn giản nhất có thể của một chương trình Windows đúng nghĩa. Ngay cả một chương trình tối thiểu để in ra chữ *HelloWorld* cũng phải đợi thông điệp của Window yêu cầu vẽ lại nội dung cửa sổ để vẽ dòng chữ này lên cửa sổ. Hơi bị rối rắm phải không nhỉ? Làm quen với hệ thống thông điệp của Windows sẽ mất một chút thời gian nếu ta đã quen với cách lập trình truyền thống “cần làm gì thì gọi hàm làm cái đó”. May mắn là chúng ta sẽ không phải dính đến cơ chế thông điệp của Windows quá nhiều vì chúng ta sẽ nhanh chóng làm việc với DirectX.

Một khi đã bắt đầu với Direct3D, chúng ta sẽ không cần quay trở lại WinMain nữa.

Dòng lệnh đầu tiên khai báo một biến để lưu một thông điệp do Windows gửi đến chương trình. Biến này sẽ được hàm **GetMessage** ở sau đổ dữ liệu vào

```
MSG msg;
```

Hai lời gọi hàm tiếp theo **MyRegisterClass** và **InitInstance** gọi đến hai hàm do chúng ta viết ra lần lượt dùng để khởi tạo lớp cửa sổ (ta sẽ nói chi tiết sau) và khởi tạo thể hiện của chương trình. Cả hai hàm đều nhận tham số là *hInstance* – định danh của thể hiện truyền từ WinMain. Hàm **InitInstance** sẽ tạo ra cửa sổ chương trình nếu *instance* của chúng là *instance* đầu tiên. Ngược lại nó sẽ thoát.

Tiếp theo là phần quan trọng của WinMain, vòng lặp chính dùng để xử lý tất cả thông điệp của chương trình

```
while (GetMessage(&msg, NULL, 0, 0))
{
    TranslateMessage(&msg);
    DispatchMessage(&msg);
}
```

Chúng ta dễ nhận thấy rằng vòng lặp sẽ chạy mãi mãi cho đến khi chúng ta nhận được một thông điệp và thông điệp này khiến cho hàm GetMessage trả về kết quả false (thường là thông điệp yêu cầu đóng cửa sổ ứng dụng). Hàm GetMessage có nhiệm vụ lấy ra một thông điệp trong số những thông điệp Windows gửi đến chương trình trong hàng đợi thông điệp của ứng dụng. Hàm GetMessage có cấu trúc như sau

```
BOOL GetMessage(
    LPMSG lpMsg,
    HWND hWnd,
    UINT wMsgFilterMin,
```

Ý nghĩa các thông số như sau:

LPMSG lpMsg : đây là con trỏ đến biến để lưu trữ dữ liệu của thông điệp mà chương trình sẽ nhận được từ Windows.

HWND hWnd: tham số này chứa định danh cửa sổ, cho biết sẽ lấy ra thông điệp của cửa sổ nào. Nếu truyền vào NULL, GetMessage sẽ trả ra toàn bộ thông điệp gửi đến instance của chương trình.

UINT wMsgFilterMin, UINT wMsgFilterMax: 2 tham số này giới hạn hàm GetMessage chỉ lấy ra thông điệp trong một phạm vi nhất định nào đó. Chúng ta sẽ không xử dụng 2 tham số này nên sẽ đặt cả 2 giá trị của chúng bằng 0.

Lời gọi tới hàm **GetMessage** là dòng code quan trọng nhất của chương trình Windows. Nếu không có dòng code này trong **WinMain**, ứng dụng của chúng ta sẽ tê liệt vì không thể phản ứng với bất kỳ thao tác nào của người dùng.

Hai dòng code bên trong vòng **while** dùng để xử lý thông điệp trả ra bởi hàm **GetMessage**. Hàm **TranslateMessage** dùng để chuyển đổi thông điệp chứa mã phím ảo (virtual-key) sang mã ký tự. Hàm **DispatchMessage** trả thông điệp trở lại hệ thống thông điệp. Kết hợp hai hàm này sẽ giúp chương trình lấy ra và chuyển thông điệp đến từ hệ điều hành Windows cho hàm xử lý thông điệp **WinProc**.

Nếu bạn đọc chưa hiểu lắm về vòng lặp thông điệp ở bước này thì đừng quá lo lắng. Vòng lặp thông điệp chỉ cần được viết ra một lần duy nhất và sẽ hầu như không cần sửa đổi về sau.

2.4. VÒNG LẶP CHÍNH CỦA GAME (GAME LOOP)

Khi game của ta có các đối tượng di chuyển trên màn hình (chẳng hạn như các nắm đấm, rùa, đạn bắn, cháy nổ, v...v...), ta sẽ cần mọi thứ liên tục chuyển động cho dù có hay không có thông điệp nào trong hàng đợi. Vòng lặp thông điệp ở trên hoàn toàn không phù hợp với game vì nó sẽ

dừng lại (và không làm gì cả) ngay tại hàm *GetMessage* khi không có thông điệp nào (khi người dùng không “đụng” gì đến chuột, bàn phím ...). Như vậy chúng ta phải có một vòng lặp chạy liên tục để giữ mọi thứ chuyển động nhưng đồng thời cũng vẫn phải nhận và xử lý thông điệp trong hàng đợi (nếu có).

Để làm điều này, ta sẽ sử dụng hàm *PeekMessage* thay vì *GetMessage*. *PeekMessage* kiểm tra xem có thông điệp trong hàng đợi hay không, nếu có thì lấy thông điệp ra y như *GetMessage*, nếu không có – thay vì đứng đợi như *GetMessage* – *PeekMessage* sẽ đơn giản là thoát ra để mọi thứ tiếp tục chạy.

Các tham số của *PeekMessage* như sau

```
BOOL PeekMessage (  
    LPMSG lpMsg,      // con trỏ đến nơi lưu trữ MSG  
    HWND hWnd,       // cửa sổ nhận thông điệp  
    UINT wMsgFilterMin, // thông điệp đầu tiên  
    UINT wMsgFilterMax, // thông điệp cuối cùng  
    UINT wRemoveMsg );
```

Hai tham số *wMsgFilterMin* và *wMsgFilterMax* không quan trọng với chúng ta nên chúng ta sẽ dùng giá trị 0 cho chúng. Tham số *wRemoveMsg* nhận vào một trong hai hằng số *PM_NOREMOVE* và *PM_REMOVE* tương ứng với việc gỡ thông điệp ra hàng đợi hay không khi đã nó đã đọc thông điệp từ hàng đợi vào *lpMsg*.

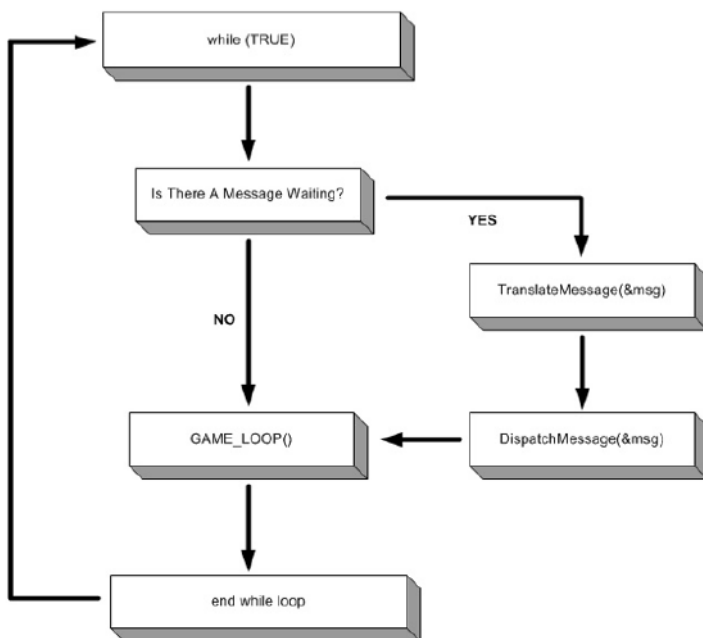
Đoạn code với *PeekMessage* sẽ trông như sau:

```
//khởi tạo game  
Game_Init();  
  
// vòng lặp thông điệp chính  
while (!done)  
{  
    if (PeekMessage(&msg, NULL, 0, 0, PM_REMOVE))  
    {  
        // xử lý thông điệp  
    }  
}
```

```
// kiểm tra xem có phải thông điệp thoát?  
if (msg.message == WM_QUIT)  
    done = 1;  
  
    //giải mã và truyền thông điệp cho WndProc  
    TranslateMessage(&msg);  
DispatchMessage(&msg);  
}  
  
//process game loop  
Game_Run();  
}
```

Bạn đọc hãy chú ý đến việc xử lý thông điệp WM_QUIT. Đây là nơi duy nhất để chương trình ngưng chạy. Thông điệp sẽ được gửi đến chương trình khi người dùng thoát chương trình một cách cưỡng bức (như bấm Alt-F4) hoặc chúng ta tự gửi thông điệp khi phát hiện người dùng bấm một phím nào đó như ESC.

Ý tưởng của PeekMessage có thể được tóm tắt qua hình sau:

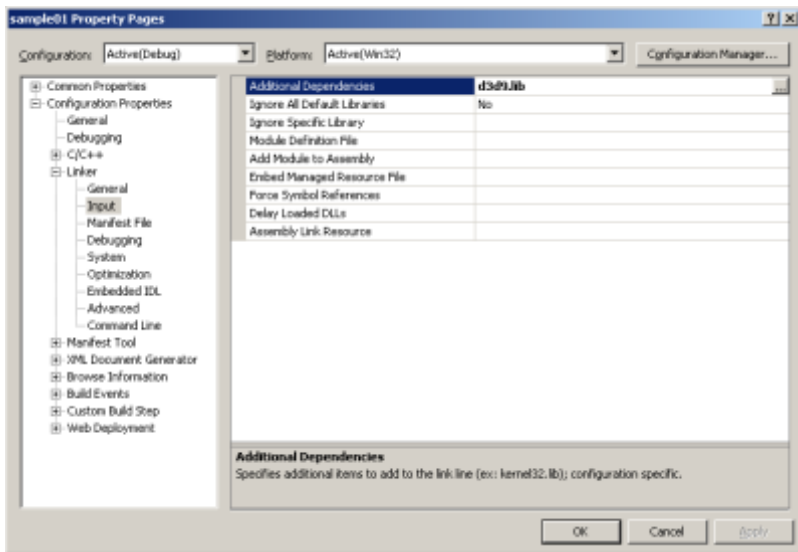


Hình 2.8. Vòng lặp Game dùng PeekMessage

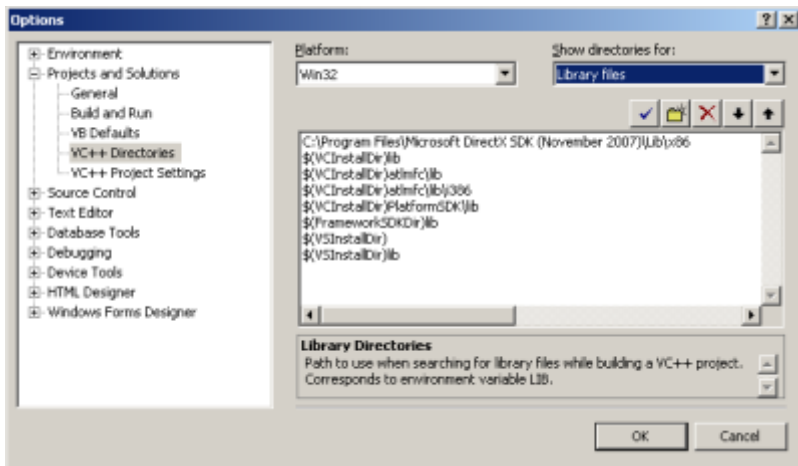
Cũng cần lưu ý là chúng ta gọi một hàm mới là *Game_Run*. Hàm này sẽ là trọng tâm của game về sau, nơi chúng ta xử lý gần như mọi thứ - cập nhật trạng thái thế giới, vẽ một khung hình, xử lý bàn phím, chuột, v...v...

2.5. KHỞI TẠO DIRECTX

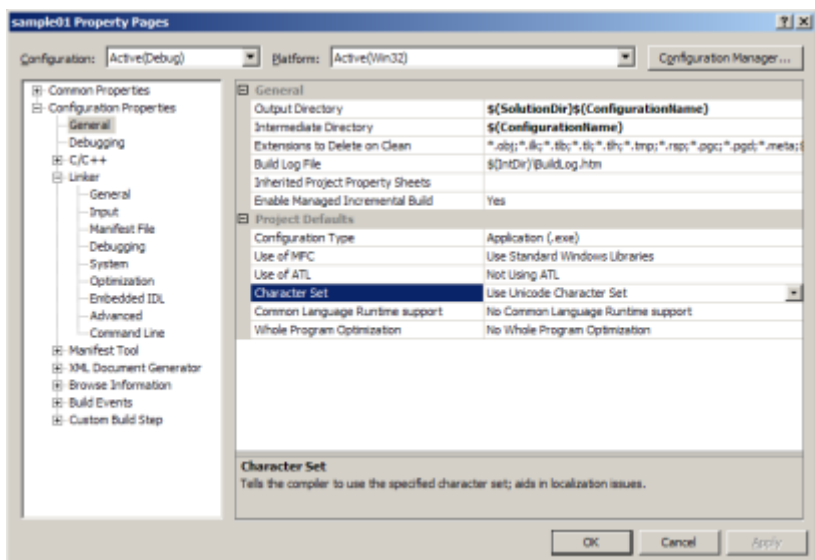
Để lập trình với Direct3D hay bất kỳ thành phần nào của DirectX, ta cần phải download và cài đặt DirectX SDK đồng thời khai báo các thư viện và file header của DirectX, cụ thể là file thư viện **d3d9.lib** và **d3d9.h**.



Hình 2.9. Liên kết tới thư viện d3d9.lib



Hình 2.10. Khai báo thư mục tìm kiếm .lib



Hình 2.11. Thiết lập Character Set.

2.5.1. Direct3D interface

Để viết chương trình có thể dùng Direct3D, ta cần phải tạo một biến để giữ một *interface* của Direct3D và một *interface* đến lớp thiết bị đồ họa. Kiểu biến của Direct3D interface là LPDIRECT3D9 và của thiết bị đồ họa là LPDIRECT3DDEVICE9. Ta có thể tạo các biến này như sau:

```
LPDIRECT3D9 d3d = NULL;
LPDIRECT3DDEVICE9 d3ddev = NULL;
```

Đối tượng LPDIRECT3D9 là biến “tổng” của toàn bộ thư viện Direct3D và có thể điều khiển khá nhiều thứ trong khi đó biến LPDIRECT3DDEVICE9 đại diện cho card đồ họa của máy tính. Các khai báo kiểu của hai đối tượng này đều nằm trong file header *d3d9.h*

2.5.2. Tạo đối tượng Direct3D và device

Đối tượng Direct3D được tạo bằng một lệnh đơn giản sau

```
d3d = Direct3DCreate9(D3D_SDK_VERSION);
```

Kế tiếp là tạo ra đối tượng device đại diện cho card màn hình từ đối tượng Direct3D vừa tạo.

```
d3d->CreateDevice(  
    D3DADAPTER_DEFAULT, // dùng card màn hình mặc định  
    D3DDEVTYPE_HAL,  
    // vẽ bằng phần cứng  
    //i.e. bằng card màn hình thay vì giả lập  
    hWnd,  
    D3DCREATE_SOFTWARE_VERTEXPROCESSING,  
    &d3dpp, // các tham số thể hiện của thiết bị  
    &d3ddev) // đối tượng dev được tạo ra
```

Trước khi gọi hàm này, ta phải thiết lập các tham số cho thiết bị thông qua một biến D3DPRESENT_PARAMETERS d3dpp và truyền con trỏ đến cho lời gọi hàm.

```
D3DPRESENT_PARAMETERS d3dpp;  
ZeroMemory(&d3dpp, sizeof(d3dpp));  
// xóa mọi thứ về 0 trước khi sử dụng
```

Có khá nhiều tùy chọn cho các tham số của d3dpp, chúng ta chỉ dùng tham số cần thiết trước mắt. Cụ thể là:

```
d3dpp.Windowed = TRUE; // thể hiện ở chế độ cửa sổ  
d3dpp.SwapEffect = D3DSWAPEFFECT_DISCARD;  
d3dpp.BackBufferFormat = D3DFMT_UNKNOWN;
```

Như vậy là đã đủ để khởi tạo một cửa sổ để có thể vẽ bằng Direct3D. Bây giờ chúng ta sẽ xây dựng khung cho hàm **Game_Run** để có thể vẽ một số thứ đơn giản

2.5.3. Game_Run

```
void Game_Run(HWND hwnd)  
{  
    //đảm bảo con trỏ đến Direct3D là hợp lệ  
    if (d3ddev == NULL)
```



```

        return;

        //Xoá backbuffer về màu đen
        d3ddev->Clear(0,          NULL,          D3DCLEAR_TARGET,
D3DCOLOR_XRGB(0,255,0), 1.0f, 0);

        //đánh dấu bắt đầu vẽ một frame
        if (d3ddev->BeginScene())
        {
            //đánh dấu kết thúc vẽ một frame
            d3ddev->EndScene();
        }

        //đào nội dung backbuffer lên front buffer
        d3ddev->Present(NULL, NULL, NULL, NULL);
    }

```

Hàm *Clear* tô toàn bộ *backbuffer* bằng một màu cho trước – trong ví dụ trên là màu xanh lá. Chúng ta cần phải gọi hàm *Clear* ứng với mỗi frame để xóa toàn bộ nội dung đã vẽ ở những frame trước. Nếu không xóa, hình ảnh của những frame trước sẽ còn lại, kết hợp với những hình ảnh vẽ ở frame này sẽ tạo ra một đồng hồ hỗn độn trên màn hình.

Hai hàm *BeginScene* và *EndScene* đánh dấu việc ta bắt đầu và kết thúc việc vẽ lên backbuffer.

Hàm *Present* sẽ “swap” nội dung backbuffer lên front-buffer để thể hiện frame lên màn hình.

2.5.4. Game_End

Trước khi đóng game, ta cần giải phóng các đối tượng *d3d* và *d3ddev* đã tạo. Khá đơn giản, chỉ cần gọi phương thức *Release* của chúng

```
void Game_End(HWND hwnd)
```

```
{
    MessageBox(hwnd, "Program is about to end",
"Game_End", MB_OK);

    if (d3ddev != NULL) d3ddev->Release();

    if (d3d != NULL) d3d->Release();
}
```

2.5.5. Chạy trong chế độ toàn màn hình

Game hiếm khi chạy trong chế độ cửa sổ. Để game chạy toàn màn hình, ta cần thay đổi một số tham số khi tạo cửa sổ và khi khởi tạo đối tượng LPDIRECT3DDEVICE như sau

```
//create a new window
hwnd = CreateWindow(
    APPTITLE,                //tên lớp cửa sổ
    APPTITLE,                //tiêu đề sửa sổ
    WS_EX_TOPMOST | WS_VISIBLE | WS_POPUP,    //window
    style
    CW_USEDEFAULT,           //toạ độ x
    CW_USEDEFAULT,           //toạ độ y
    SCREEN_WIDTH,            //chiều rộng
    SCREEN_HEIGHT,           //chiều cao
    NULL,                    //cửa sổ cha
    NULL,                    //menu
    hInstance,               //instance
    NULL);                   //các tham số cửa sổ

SOURCE d3dpp fullscreen
d3dpp.Windowed = FALSE;
d3dpp.SwapEffect = D3DSWAPEFFECT_DISCARD;
d3dpp.BackBufferFormat = D3DFMT_X8R8G8B8;
d3dpp.BackBufferCount = 1;
```

```
d3dpp.BackBufferWidth = SCREEN_WIDTH;  
d3dpp.BackBufferHeight = SCREEN_HEIGHT;  
d3dpp.hDeviceWindow = hwnd;
```

2.5.6. Kiểm tra phím để thoát game khi ở chế độ fullscreen

Khi chạy ở chế độ fullscreen, ta sẽ không thể dùng chuột để tắt cửa sổ game được. Để có thể thoát game một cách nhẹ nhàng, đoạn chương trình của chúng ta cần kiểm tra xem user có bấm phím ESC để thoát game không, nếu có, nó gửi thông điệp *WM_QUIT* đến chính nó để vòng lặp game kết thúc.

Để kiểm tra một phím có đang được bấm hay không, ta dùng một hàm có sẵn của Windows là *GetAsyncKeyState*. Đây chỉ là tạm thời. Sau khi tìm hiểu cách tương tác với DirectInput ta sẽ chuyển sang dùng DirectInput.

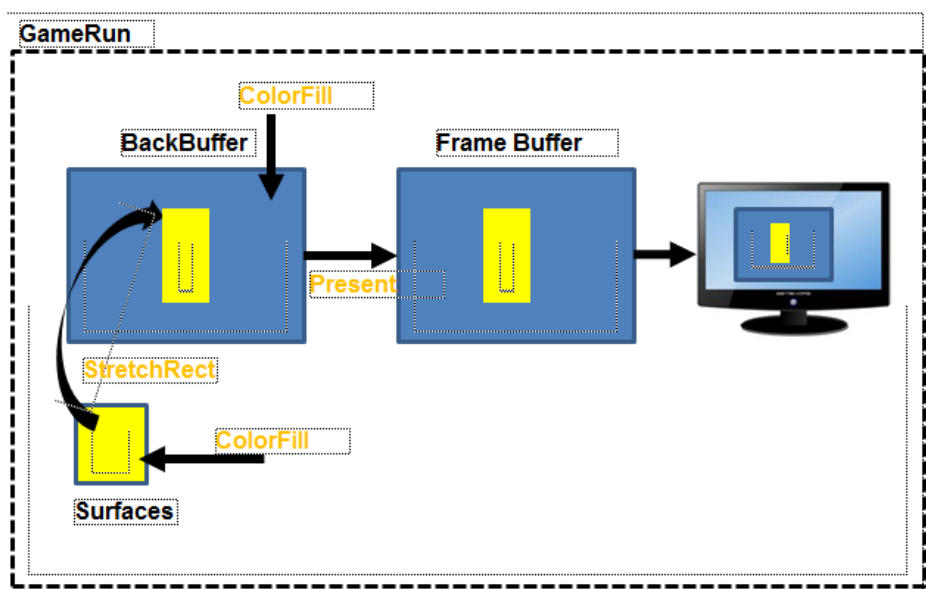
```
#define KEY_DOWN(vk_code)  
    ((GetAsyncKeyState(vk_code) & 0x8000) ? 1 : 0)  
#define KEY_UP(vk_code)  
    ((GetAsyncKeyState(vk_code) & 0x8000) ? 1 : 0)
```

2.6. Surface và Bitmaps

Direct3D sử dụng surface cho rất nhiều thứ. Màn hình sẽ hiển thị những những thứ mà card đồ họa gửi đến nó và card đồ họa sẽ lấy dữ liệu vùng nhớ đệm (frame buffer) và gửi tới màn hình từ pixel một.

Frame buffer nằm trong chip nhớ của card đồ họa và loại chip này có tốc độ khá nhanh. Có một thời gian những loại chip này rất mắc tiền so với RAM vì nó nhanh - nhanh hơn nhiều so với RAM. Tuy nhiên, càng về sau thì công nghệ sản xuất VRAM ngày càng phát triển, khoảng cách giữa RAM và VRAM ngày càng rút ngắn kể cả giá thành của chúng.

Frame buffer nằm ở bộ nhớ đồ họa, và biểu thị hình ảnh mà ta nhìn thấy trên màn hình. Do đó, để thay đổi hiển thị trên màn hình, cách đơn giản nhất là thay đổi trực tiếp dữ liệu trên frame buffer. Đây là cách hệ thống làm việc. Tuy nhiên, chúng ta sẽ không muốn vẽ trực tiếp lên framebuffer vì màn hình sẽ bị giật khi ta vẽ, xóa trong khi màn hình đang trong quá trình làm tươi (refresh). Thay vào đó, ta sẽ vẽ mọi thứ lên một vùng nhớ ngoài màn hình gọi là “double” hay “back” buffer, và chuyển nó lên màn hình rất nhanh. Cái đó người ta gọi là “double buffering”.



Hình 2.12. Minh họa khái niệm “double bufering”

2.6.1. Bề mặt chính

Ở phần trước, ta đã tạo back buffer bằng các tham số. Sau đó, dùng hàm Clear để tô màu xanh vào back buffer và sau đó dùng hàm Present để làm mới màn hình. Chúng ta cũng đã sử dụng double / back buffer vì double buffer ngày nay rất phổ biến trong game như côm hay nước mắm trong bữa ăn vậy! Frame buffer mà ta đề cập lúc đầu còn được gọi là front buffer, được back buffer sao chép lên trong mỗi vòng lặp. Cả

hai bộ nhớ đệm trên đều được tạo ra khi chúng ta cấu hình bằng hàm *CreateDevice*.

2.6.2. Bề mặt phụ ngoài (offscreen surface)

Còn một kiểu surface mà chúng ta gọi là bề mặt thứ cấp hoặc bề mặt ngoài màn hình. Kiểu bề mặt này thực chất là một mảng pixel giống như một bitmap. Ta có thể tạo bao nhiêu offscreen surface trong game cũng được miễn là đủ RAM hay VRAM để chứa tất cả. Thông thường thì sẽ có ngàn cái như vậy khi chạy game. Lý do là mọi thành phần đồ họa trong game đều lưu trữ trong surface. Những surface đó được sao chép lên màn hình thông qua thao tác được gọi là **bit-lock tranfer**.

2.6.3. Tạo surface

Hãy tạo một Direct3D surface bằng việc định nghĩa con trỏ. Đối tượng surface gọi là LPDIRECT3DSURFACE9, ta tạo biến như sau:

```
LPDIRECT3DSURFACE9 surface = NULL;
```

Một khi đã tạo được surface, ta có thể làm nhiều thứ với surface đó. Ta có thể vẽ hình lên nó, tô màu lên nó, và còn nhiều nữa. Nếu muốn xóa những thứ đang vẽ, ta có thể dùng hàm ColorFill

```
HRESULT ColorFill(  
    IDirect3DSurface9* pSurface,  
    CONST RECT *pRect,  
    D3DCOLOR color  
);
```

Ví dụ:

```
d3ddev->ColorFill(  
    surface,NULL, D3DCOLOR_XRGB(255, 0, 0));
```

2.6.4. Vẽ lên Surface

Tất nhiên đây là phần thú vị nhất. Ta có thể vẽ một surface lên surface khác. Chúng ta có hàm StretchRect. Hàm như sau:

```
HRESULT StretchRect(  
    IDirect3DSurface9 *pSourceSurface,  
    CONST RECT *pSourceRect,  
    IDirect3DSurface9 *pDestSurface,  
    CONST RECT *pDestRect,  
    D3DTEXTUREFILTERTYPE Filter  
);
```

Hàm này thường được gọi với các tham số đơn giản như sau:

```
D3ddev->stretchRect(surface,    NULL,    backbuffer,    NULL,  
D3DTEXF_NONE);
```

Hai surface đó phải cùng kích thước. Nếu surface nguồn nhỏ hơn surface đích thì nó sẽ được vẽ tại phía trên – bên trái của surface đích. Dĩ nhiên, vậy sẽ không thú vị, rect nguồn có thể nhỏ hơn rect đích và bạn có thể vẽ surface nguồn lên bất kì nơi nào trên surface đích. Đây là ví dụ:

```
rect.left = 100;  
rect.top = 90;  
rect.right = 200;  
rect.bottom = 180;  
d3ddev->StretchRect(surface,    NULL,    backbuffer,    &rect,  
D3DTEXF_NONE);
```

Đoạn code trên sao chép surface nguồn lên surface đích, theo hình chữ nhật (100, 90, 200, 180) tức 100 x 90 pixel. Kích thước của surface nguồn là NULL, nên surface nguồn sẽ được lấy toàn bộ.

Ta vừa sử dụng biến *backbuffer* mà chưa giải thích. Nó là một biến toàn cục để dễ dàng dùng. Điều này không bắt buộc, ta có thể tự tạo biến riêng của mình. Có thể lấy con trỏ đến buffer này bằng việc gọi hàm `GetBackBuffer`

```
HRESULT GetBackBuffer(  
    UINT iSwapChain,  
    UINT BackBuffer,  
    D3DBACKBUFFER_TYPE Type,
```

```
    IDirect3DSurface **ppBackBuffer  
);
```

Đây là ví dụ làm thế nào để lấy được backbuffer. Đầu tiên, tạo biến backbuffer. Sau đó, dùng hàm GetBackBuffer để lấy backbuffer.

```
LPDIRECT3DSURFACE9 backbuffer = NULL;  
d3ddev->GetBackBuffer(  
    0, 0, D3DBACKBUFFER_TYPE_MONO, &backbuffer);
```

Đừng nghĩ là Direct3D khó. Nó phụ thuộc vào cách nhìn của chúng ta. Có thể bạn hơi rối về những hàm chưa biết trong DirectX 9 SDK, hãy bắt đầu viết một game nho nhỏ, vẽ vài hình đa giác rồi ta cũng sẽ dần dần sẽ nắm được!

2.6.5. Ví dụ Create_Surface

Hãy làm một ví dụ đầy đủ hơn để ta có thể thấy cách kết hợp với nhau. Chúng ta có sẵn một chương trình “Create_Surface” để minh họa cách dùng ColorFill, StretchRect và GetBuffer, và quan trọng hơn, cho ta biết cách sử dụng surface. Kết quả của chương trình này giống như ở hình dưới đây. Nếu bạn đọc bản thảo tại sao hình này chỉ có một hình chữ nhật trên màn hình trong khi chương trình chạy có vẻ sẽ vẽ nhiều hình chữ nhật thì câu trả lời sẽ là : đó là vì khi chương trình chạy, chỉ có một hình chữ nhật trên màn hình tại một thời điểm, nhưng khi chạy do nó chương trình chạy quá nhanh và xuất hiện quá nhanh nên mắt ta có cảm giác là có nhiều hình xuất hiện cùng lúc.



Hình 2.13. Kết quả chương trình Create_Surface

Chương trình Create_Surface copy ngẫu nhiên một hình chữ nhật từ một surface bên dưới lên màn hình.

Tiếp tục và tạo một project tên “Create_Surface” và thêm các file vào project “winmain.cpp”. Bây giờ, như đã làm ở trước, vào Menu, chọn Setting, chọn Link, và thêm “d3d9.lib” ở mục Object/library.

Dưới đây là mã nguồn của toàn bộ chương trình, những dòng quan trọng trong code được đánh dấu in đậm để ta có thể nhìn ra sự khác biệt so với chương trình minh họa phần trước.

```
//các file header cần đưa vào
#include <d3d9.h>
#include <time.h>

//tiêu đề ứng dụng
#define APPTITLE "Create_Surface"

//các macro để đọc phím
#define KEY_DOWN(vk_code) ((GetAsyncKeyState(vk_code) & 0x8000) ? 1 : 0)
#define KEY_UP(vk_code) ((GetAsyncKeyState(vk_code) & 0x8000) ? 1 : 0)

//độ phân giải màn hình
```



```

#define SCREEN_WIDTH 640
#define SCREEN_HEIGHT 480

//các khai báo hàm
LRESULT WINAPI WinProc(HWND,UINT,WPARAM,LPARAM);
ATOM MyRegisterClass(HINSTANCE);
int Game_Init(HWND);
void Game_Run(HWND);
void Game_End(HWND);

//các đối tượng của Direct3D
LPDIRECT3D9 d3d = NULL;
LPDIRECT3DDEVICE9 d3ddev = NULL;

LPDIRECT3DSURFACE9 backbuffer = NULL;
LPDIRECT3DSURFACE9 surface = NULL;

//window event callback function
LRESULT WINAPI WinProc( HWND hWnd, UINT msg, WPARAM wParam,
LPARAM lParam )
{
    switch( msg )
    {
        case WM_DESTROY:
            Game_End(hWnd);
            PostQuitMessage(0);
            return 0;
    }

    return DefWindowProc( hWnd, msg, wParam, lParam );
}

//các hàm hỗ trợ để khởi động Window
ATOM MyRegisterClass(HINSTANCE hInstance)
{
    //create the window class structure
    WNDCLASSEX wc;
    wc.cbSize = sizeof(WNDCLASSEX);

```

```

//điền tham số hàm vào struct
wc.style           = CS_HREDRAW | CS_VREDRAW;
wc.lpfnWndProc     = (WNDPROC)WinProc;
wc.cbClsExtra      = 0;
wc.cbWndExtra      = 0;
wc.hInstance       = hInstance;
wc.hIcon           = NULL;
wc.hCursor         = LoadCursor(NULL, IDC_ARROW);
wc.hbrBackground   = (HBRUSH)GetStockObject(WHITE_BRUSH);
wc.lpszMenuName    = NULL;
wc.lpszClassName   = APPTITLE;
wc.hIconSm         = NULL;

//đăng ký lớp của sổ
return RegisterClassEx(&wc);
}

//đầu vào ứng dụng Window
int WINAPI WinMain(HINSTANCE hInstance,
                  HINSTANCE hPrevInstance,
                  LPSTR      lpCmdLine,
                  int         nCmdShow)
{
    MSG msg;

    // đăng ký lớp của sổ
    MyRegisterClass(hInstance);

    // khởi động ứng dụng
    // lưu ý - ta đã bỏ qua lời gọi InitApplication
    HWND hWnd;

    //tạo một cửa sổ
    hWnd = CreateWindow(
        APPTITLE,           //window class
        APPTITLE,           //title bar
        WS_EX_TOPMOST | WS_VISIBLE | WS_POPUP, //window style

```

```

        CW_USEDEFAULT,           //x position of window
        CW_USEDEFAULT,           //y position of window
        SCREEN_WIDTH,            //width of the window
        SCREEN_HEIGHT,           //height of the window
        NULL,                     //parent window
        NULL,                     //menu
        hInstance,               //application instance
        NULL);                   //window parameters

//kiểm tra lỗi nếu không tạo được cửa sổ
if (!hWnd)
    return FALSE;

//hiển thị cửa sổ
ShowWindow(hWnd, nCmdShow);
UpdateWindow(hWnd);

//khởi tạo game
if (!Game_Init(hWnd))
    return 0;

// vòng lặp thông điệp chính
int done = 0;
while (!done)
{
    if (PeekMessage(&msg, NULL, 0, 0, PM_REMOVE))
    {
        //kiểm tra điều kiện thoát
        if (msg.message == WM_QUIT)
            done = 1;

        //giải mã thông điệp và chuyển lại cho WinProc
        TranslateMessage(&msg);
        DispatchMessage(&msg);
    }
    else
        //xử lý game
        Game_Run(hWnd);
}

```

```

    }

    return msg.wParam;
}

int Game_Init(HWND hwnd)
{
    HRESULT result;

    //initialize Direct3D
    d3d = Direct3DCreate9(D3D_SDK_VERSION);
    if (d3d == NULL)
    {
        MessageBox(hwnd, "Error initializing Direct3D", "Error",
MB_OK);
        return 0;
    }

    //set Direct3D presentation parameters
    D3DPRESENT_PARAMETERS d3dpp;
    ZeroMemory(&d3dpp, sizeof(d3dpp));

    d3dpp.Windowed = FALSE;
    d3dpp.SwapEffect = D3DSWAPEFFECT_DISCARD;
    d3dpp.BackBufferFormat = D3DFMT_X8R8G8B8;
    d3dpp.BackBufferCount = 1;
    d3dpp.BackBufferWidth = SCREEN_WIDTH;
    d3dpp.BackBufferHeight = SCREEN_HEIGHT;
    d3dpp.hDeviceWindow = hwnd;

    //create Direct3D device
    d3d->CreateDevice(
        D3DADAPTER_DEFAULT,
        D3DDEVTYPE_HAL,
        hwnd,
        D3DCREATE_SOFTWARE_VERTEXPROCESSING,
        &d3dpp,
        &d3ddev);
}

```

```

    if (d3ddev == NULL)
    {
        MessageBox(hwnd, "Error creating Direct3D device",
"Error", MB_OK);
        return 0;
    }

    //set random number seed
    srand(time(NULL));

    //clear the backbuffer to black
    d3ddev->Clear(0, NULL, D3DCLEAR_TARGET,
D3DCOLOR_XRGB(0,0,0), 1.0f, 0);

    //create pointer to the back buffer
    d3ddev->GetBackBuffer(0, 0, D3DBACKBUFFER_TYPE_MONO,
&backbuffer);

    //create surface
    result = d3ddev->CreateOffscreenPlainSurface(
        100,                //width of the surface
        100,                //height of the surface
        D3DFMT_X8R8G8B8,    //surface format
        D3DPPOOL_DEFAULT,   //memory pool to use
        &surface,            //pointer to the surface
        NULL);              //reserved (always NULL)

    if (!result)
        return 1;

    //return okay
    return 1;
}

void Game_Run(HWND hwnd)
{
    RECT rect;
    int r,g,b;

```

```

//make sure the Direct3D device is valid
if (d3ddev == NULL)
    return;

//start rendering
if (d3ddev->BeginScene())
{
    //fill the surface with random color
    r = rand() % 255;
    g = rand() % 255;
    b = rand() % 255;
    d3ddev->ColorFill(surface, NULL, D3DCOLOR_XRGB(r,g,b));

    //copy the surface to the backbuffer
    rect.left = rand() % SCREEN_WIDTH/2;
    rect.right = rect.left + rand() % SCREEN_WIDTH/2;
    rect.top = rand() % SCREEN_HEIGHT;
    rect.bottom = rect.top + rand() % SCREEN_HEIGHT/2;
    d3ddev->StretchRect(surface, NULL, backbuffer, &rect,
D3DTEXF_NONE);

    //stop rendering
    d3ddev->EndScene();
}

//display the back buffer on the screen
d3ddev->Present(NULL, NULL, NULL, NULL);

//check for escape key (to exit program)
if (KEY_DOWN(VK_ESCAPE))
    PostMessage(hwnd, WM_DESTROY, 0, 0);
}

void Game_End(HWND hwnd)
{
    //free the surface
    surface->Release();
}

```

```

//release the Direct3D device
if (d3ddev != NULL)
    d3ddev->Release();

//release the Direct3D object
if (d3d != NULL)
    d3d->Release();
}

```

Chú ý: Để không tốn kém thời gian, chúng ta sẽ không lặp lại tất cả đoạn code Window từ thời điểm này mà chỉ thêm những đoạn code cần thiết để chứng minh cho nội dung của chủ đề đang thảo luận. Hãy dùng các đoạn code mẫu sẵn có nếu bạn muốn chạy chương trình.

2.6.6. Tải ảnh lên từ ổ cứng

Bước tiếp theo là tải ảnh bitmap lên vào surface từ ổ cứng và vẽ nó lên màn hình (thông qua backbuffer, tất nhiên rồi). Không may là Direct3D không có hàm nào để tải 1 file ảnh bitmap, do đó bạn phải tự viết hàm tải ảnh bitmap.

Direct3D thật sự không biết cách tải ảnh bitmap. May mắn thay, có một thư viện hỗ trợ là D3DX (viết tắt của Direct3D Extension) cung cấp rất nhiều làm tiện ích, include thư viện vào để tải 1 ảnh bitmap lên surface. Chỉ cần `#include <d3dx.h>` và `d3dx9.lib` vào trong phần *Settings* của project.

Hàm cần dùng là “D3DXLoadSurfaceFromFile, có cấu trúc:

```

HRESULT D3DXLoadSurfaceFromFile(
    LPDIRECT3DSURFACE9 pDestSurface,
    const PALETTEENTRY *pDestPalette,
    const RECT *pDestRect,
    LPCTSTR pSrcFile,
    const RECT *pSrcRect,
    DWORD Filter,

```

```
D3DCOLOR ColorKey,  
D3DXIMAGE_INFO *pSrcInfo  
) ;
```

Quan trọng: hàm này không chỉ có thể tải 1 ảnh bitmap chuẩn của Window, nó còn có thể tải nhiều định dạng khác. Bảng dưới đây cho thấy là danh sách các định dạng ảnh mà hàm này hỗ trợ:

Table 6.1 Graphics File Formats

Extension	Format
.bmp	Windows Bitmap (standard)
.dds	DirectDraw Surface (DirectX 7)
.dib	Windows Device Independent Bitmap
.jpg	Joint Photographic Experts Group (JPEG)
.png	Portable Network Graphics
.tga	Truevision Targa

Như thường lệ, hàm này sẽ có rất nhiều tham số có giá trị NULL. Hãy đọc ví dụ mẫu có sẵn để biết cách dùng hàm này.

2.6.7. Chương trình Load_Bitmap

Hãy viết một chương trình đơn giản để chứng minh cách nạp một ảnh bitmap vào surface và vẽ nó lên màn hình. Dĩ nhiên không cần phải viết lại tất cả code, chỉ cần thay đổi vài chỗ thay đổi so với chương trình Create_Surface, chúng ta sẽ xem qua một vài đoạn cần thay đổi mà thôi. Đừng quên mở Project Setting, chọn Link, điền vào “d3d9.lib” và “d3dx9.lib” vào mục Object/Library.

Đầu tiên ta thêm *#include <d3dx9.h>* vào đoạn code như sau:

```
#include <d3d9.h>  
#include <d3dx9.h>  
#include <time.h>
```



```
//application title
#define APPTITLE "Load_Bitmap"
```

Bây giờ, đi đến hàm `Game_Init` và thay đổi một chút để được như nội dung ở sau. Phần còn lại không thay đổi.

```
int Game_Init(HWND hwnd)
{
    HRESULT result;

    //initialize Direct3D
    d3d = Direct3DCreate9(D3D_SDK_VERSION);
    if (d3d == NULL)
    {
        MessageBox(hwnd, "Error initializing Direct3D", "Error",
MB_OK);
        return 0;
    }

    //set Direct3D presentation parameters
    D3DPRESENT_PARAMETERS d3dpp;
    ZeroMemory(&d3dpp, sizeof(d3dpp));

    d3dpp.Windowed = FALSE;
    d3dpp.SwapEffect = D3DSWAPEFFECT_DISCARD;
    d3dpp.BackBufferFormat = D3DFMT_X8R8G8B8;
    d3dpp.BackBufferCount = 1;
    d3dpp.BackBufferWidth = SCREEN_WIDTH;
    d3dpp.BackBufferHeight = SCREEN_HEIGHT;
    d3dpp.hDeviceWindow = hwnd;

    //create Direct3D device
    d3d->CreateDevice(
        D3DADAPTER_DEFAULT,
        D3DDEVTYPE_HAL,
        hwnd,
        D3DCREATE_SOFTWARE_VERTEXPROCESSING,
        &d3dpp,
        &d3ddev);
}
```

```

    if (d3ddev == NULL)
    {
        MessageBox(hwnd, "Error creating Direct3D device",
"Error", MB_OK);
        return 0;
    }

    //set random number seed
    srand(time(NULL));

    //clear the backbuffer to black
    d3ddev->Clear(0, NULL, D3DCLEAR_TARGET,
D3DCOLOR_XRGB(0,0,0), 1.0f, 0);

    //create surface
    result = d3ddev->CreateOffscreenPlainSurface(
        640, //width of the surface
        480, //height of the surface
        D3DFMT_X8R8G8B8, //surface format
        D3DPOOL_DEFAULT, //memory pool to use
        &surface, //pointer to the surface
        NULL); //reserved (always NULL)

    if (result != D3D_OK)
        return 1;

    //load surface from file into newly created surface
    result = D3DXLoadSurfaceFromFile(
        surface, //destination surface
        NULL, //destination palette
        NULL, //destination rectangle
        "legotron.bmp", //source filename
        NULL, //source rectangle
        D3DX_DEFAULT, //controls how image is filtered
        0, //for transparency (0 for none)
        NULL); //source image info (usually NULL)

    //make sure file was loaded okay
    if (result != D3D_OK)

```

```
        return 1;

    //return okay
    return 1;
}
```

Trong hàm `Game_Run` cũng có một ít thay đổi, kết quả như sau:

```
void Game_Run(HWND hwnd)
{
    //make sure the Direct3D device is valid
    if (d3ddev == NULL)
        return;

    //start rendering
    if (d3ddev->BeginScene())
    {
        //create pointer to the back buffer
        d3ddev->GetBackBuffer(0, 0, D3DBACKBUFFER_TYPE_MONO,
&backbuffer);

        //draw surface to the backbuffer
        d3ddev->StretchRect(surface, NULL, backbuffer, NULL,
D3DTEXF_NONE);

        //stop rendering
        d3ddev->EndScene();
    }

    //display the back buffer on the screen
    d3ddev->Present(NULL, NULL, NULL, NULL);

    //check for escape key (to exit program)
    if (KEY_DOWN(VK_ESCAPE))
        PostMessage(hwnd, WM_DESTROY, 0, 0);
}
```

Khi chạy chương trình ta sẽ thấy như hình dưới đây:



Hình 2.12. Chương trình Load_Bitmap tải 1 ảnh bitmap vào Direct3D surface và vẽ nó lên màn hình.

CHƯƠNG 3

TẠO CHUYỂN ĐỘNG

3.1. Cách vẽ một sprite chuyển động

Có 2 cách vẽ sprite với Direct3D. Cả 2 phương pháp đều yêu cầu ta giữ các thông tin: vị trí, kích thước, tốc độ, và các thuộc tính riêng khác. Cách đơn giản hơn trong hai cách là load một ảnh sprite vào trong surface và sau đó vẽ lên backbuffer sử dụng StretchRect như ta đã làm ở chương trước. Cách khó hơn – nhưng mạnh mẽ hơn – là sử dụng một đối tượng đặc biệt gọi là D3DXSprite để giữ những sprite trong Direct3D.

D3DXSprite sử dụng *texture* thay vì surface để giữ bức ảnh làm sprite. Do đó nó yêu cầu một cách tiếp cận hơi khó hơn những gì ta trình bày ở chương trước. Tuy nhiên, việc nạp một bức ảnh bitmap vào một texture thì không khó hơn nạp một bức ảnh vào surface. Chúng ta sẽ đi qua cách đơn giản để vẽ sprite trước, và rồi sẽ đến với D3DXSprite ở phần sau.

3.1.1. Project Anim_Sprite

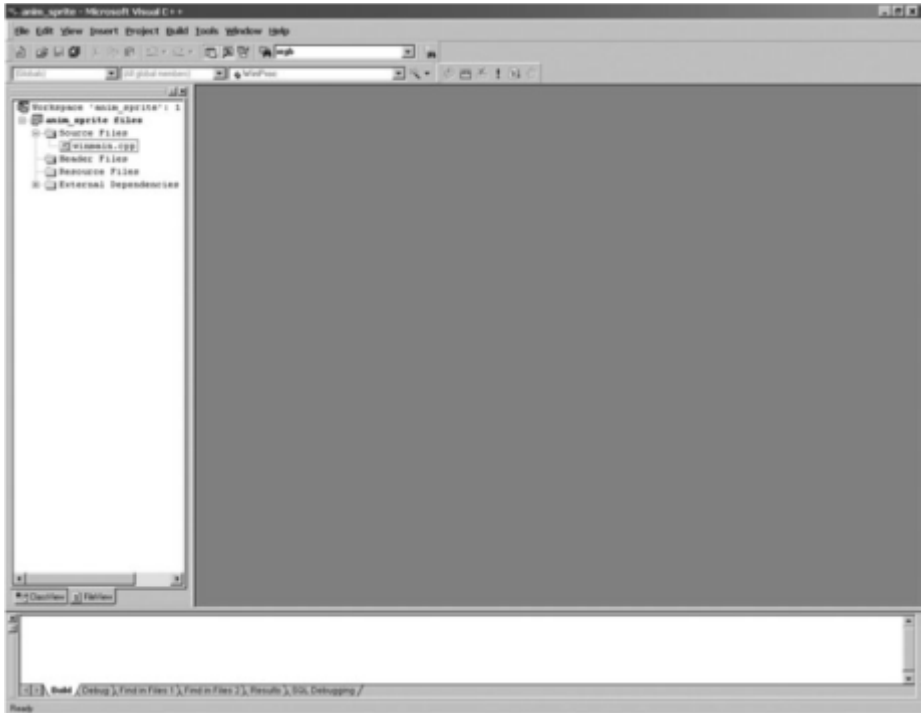
Trong chương trước, chúng ta đã thảo luận về việc tạo một framework cho game. Mục đích của framework là để làm cho việc tạo một chương trình game mới đơn giản hơn. Với framework, chúng ta không cần phải tạo lại toàn bộ DirectX 9 project từ đầu. Framework nên có source code của những hàm trợ giúp khởi tạo Direct3D, DirectInput, DirectSound, và nhiều hơn nữa,... bên cạnh các hàm load ảnh cho surface và texture. Một lý do nữa để tạo framework là để tận dụng code, hạn chế phải viết lại những đoạn code trùng lặp.

3.1.2. Cài đặt project

Hãy bắt đầu làm việc với framework bằng cách đặt code một cách logic, sắp xếp file source sẽ làm việc với nhau sao cho tạo ra được 1 game DirectX9. Đầu tiên, mở Visual C++. Tạo project mới bằng menu File và

chọn New. Project mới tên là “Anim_Sprite” và chọn “Standard Win32 Application” với tùy chọn “Empty Project” như bình thường.

Tiếp theo, thêm một file source “winmain.cpp” vào project. Hình dưới thể hiện trạng thái của project lúc này, nó giống như mọi project trước ta đã làm.

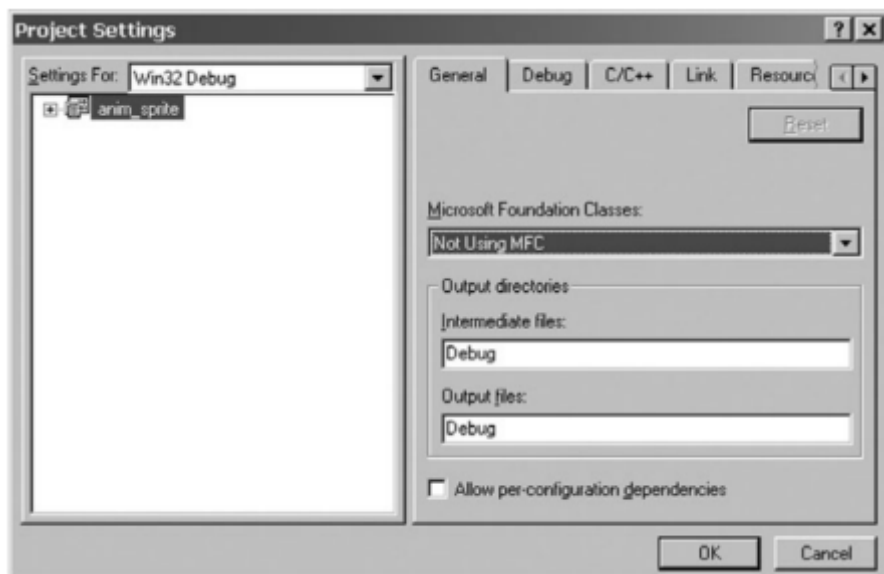


Hình 3.1. Minh họa cấu trúc project Anim_Sprite

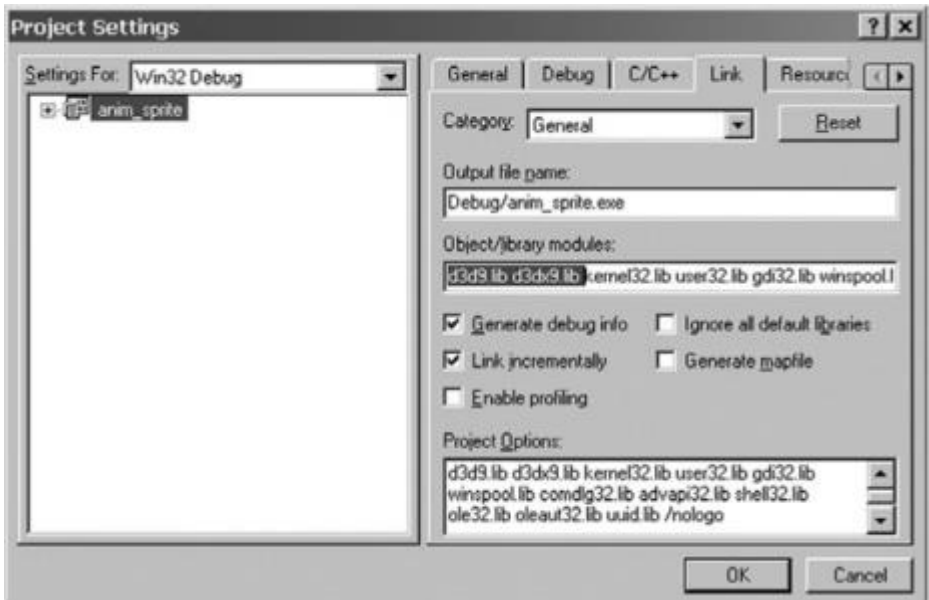
Ta sẽ đi qua từng bước thêm các thư viện DirectX vào project lại 1 lần nữa: Đầu tiên vào Project menu, chọn Setting, hộp thoại Project Setting hiện ra, hình 3.2.

Thêm các thư viện Direct3D vào theo list các file trong phần linker. Chọn tab Link và thêm “d3d9.lib” và “d3d9x.lib”.

Đó là tất cả những gì ta cần để Direct3D hỗ trợ chương trình của mình. Trong chương sau theo, chúng ta sẽ đưa bạn tìm hiểu DirectInput và DirectSound (với DirectMusic). Chúng ta sẽ học cách để hỗ trợ các thư viện này, và khi cần đến, chúng ta sẽ đi qua các DirectX component cần được thêm vào framework.



Hình 3.2. Hộp thoại Project Setting



Hình 3.3. Liên kết tới thư viện d3dx9.lib

3.1.3. File source code

Như vậy ta đã thêm file “winmain.cpp” vào project. Tiếp theo chúng ta sẽ thêm nhiều file source code khác vào project nữa. Source cho “winmain.cpp” lúc này sẽ chỉ cài đặt cửa sổ. Sẽ không có bất kỳ code DirectX nào bởi vì ta đang muốn tách riêng code cho Window, DirectX và game. Sau khi kết thúc, chúng ta sẽ thích kết quả. Thiết kế và lập trình một game là việc khó – với hàng trăm biến mà bạn phải kiểm soát trong đầu – nếu không có sự sắp xếp code một cách logic.

Bây giờ, đây là code cho “winmain.cpp”. Chú ý là nó giống hoàn toàn với code ta đã có trong các chương trước bởi vì các chức năng đã được viết thành hàm riêng.

```
// winmain.cpp - Windows framework source code file

#include <d3d9.h>
#include <d3dx9.h>
#include <time.h>
```

```

#include <stdio.h>
#include "dxgraphics.h"
#include "game.h"

//window event callback function
LRESULT WINAPI WinProc( HWND hWnd, UINT msg, WPARAM
wParam, LPARAM lParam )
{
    switch( msg )
    {
        case WM_DESTROY:
            //release the Direct3D device
            if (d3ddev != NULL)
                d3ddev->Release();

            //release the Direct3D object
            if (d3d != NULL)
                d3d->Release();

            //call the "front-end" shutdown function
            Game_End(hWnd);

            //tell Windows to kill this program
            PostQuitMessage(0);
            return 0;
    }

    return DefWindowProc( hWnd, msg, wParam, lParam );
}

//helper function to set up the window properties
ATOM MyRegisterClass(HINSTANCE hInstance)
{
    //create the window class structure
    WNDCLASSEX wc;
    wc.cbSize = sizeof(WNDCLASSEX);

    //fill the struct with info
    wc.style          = CS_HREDRAW | CS_VREDRAW;
    wc.lpfnWndProc    = (WNDPROC)WinProc;
    wc.cbClsExtra     = 0;
    wc.cbWndExtra     = 0;
    wc.hInstance      = hInstance;
    wc.hIcon          = NULL;
    wc.hCursor        = LoadCursor(NULL, IDC_ARROW);
    wc.hbrBackground

```

=

```

(HBRUSH)GetStockObject(BLACK_BRUSH);
    wc.lpszMenuName = NULL;
    wc.lpszClassName = APPTITLE;
    wc.hIconSm      = NULL;

    //set up the window with the class info
    return RegisterClassEx(&wc);
}

//entry point for a Windows program
int WINAPI WinMain(HINSTANCE hInstance,
                  HINSTANCE hPrevInstance,
                  LPSTR      lpCmdLine,
                  int         nCmdShow)
{
    MSG msg;
    HWND hWnd;

    // register the class
    MyRegisterClass(hInstance);

    //set up the screen in windowed or fullscreen mode?
    DWORD style;
    if (FULLSCREEN)
        style = WS_EX_TOPMOST | WS_VISIBLE | WS_POPUP;
    else
        style = WS_OVERLAPPED;

    //create a new window
    hWnd = CreateWindow(
        APPTITLE,           //window class
        APPTITLE,           //title bar
        style,              //window style
        CW_USEDEFAULT,      //x position of window
        CW_USEDEFAULT,      //y position of window
        SCREEN_WIDTH,       //width of the window
        SCREEN_HEIGHT,      //height of the window
        NULL,               //parent window
        NULL,               //menu
        hInstance,          //application instance
        NULL);              //window parameters

    //was there an error creating the window?
    if (!hWnd)
        return FALSE;
}

```

```

        //display the window
        ShowWindow(hWnd, nCmdShow);
        UpdateWindow(hWnd);

        if (!Init_Direct3D(hWnd, SCREEN_WIDTH, SCREEN_HEIGHT,
FULLSCREEN))
            return 0;

        //initialize the game
        if (!Game_Init(hWnd))
        {
            MessageBox(hWnd, "Error initializing the game",
"Error", MB_OK);
            return 0;
        }

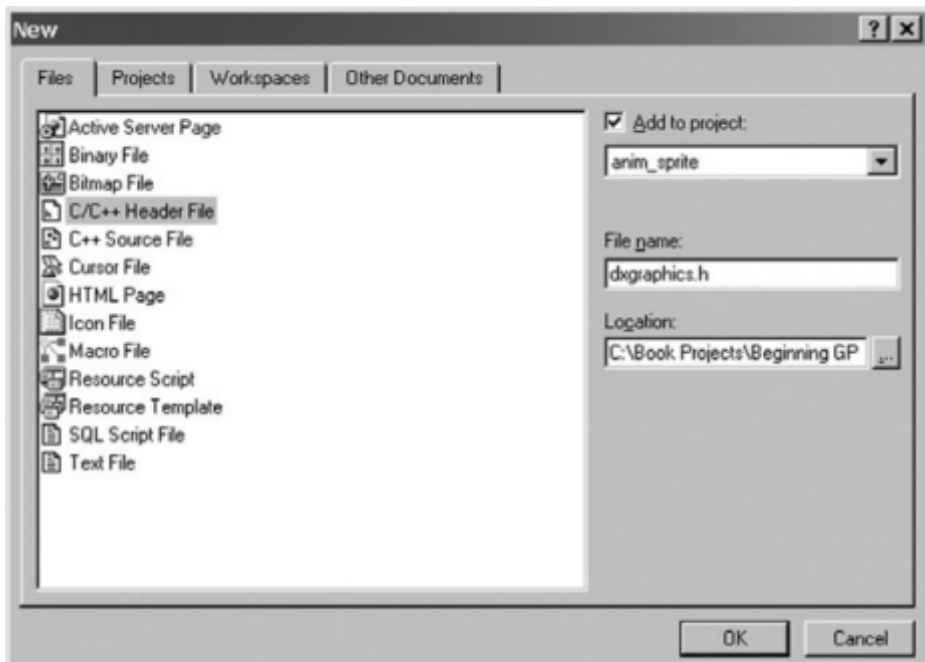
        // main message loop
        int done = 0;
        while (!done)
        {
            if (PeekMessage(&msg, NULL, 0, 0, PM_REMOVE))
            {
                //look for quit message
                if (msg.message == WM_QUIT)
                    done = 1;

                //decode and pass messages on to WndProc
                TranslateMessage(&msg);
                DispatchMessage(&msg);
            }
            else
                //process game loop (else prevents running
after window is closed)
                Game_Run(hWnd);
        }

        return msg.wParam;
    }

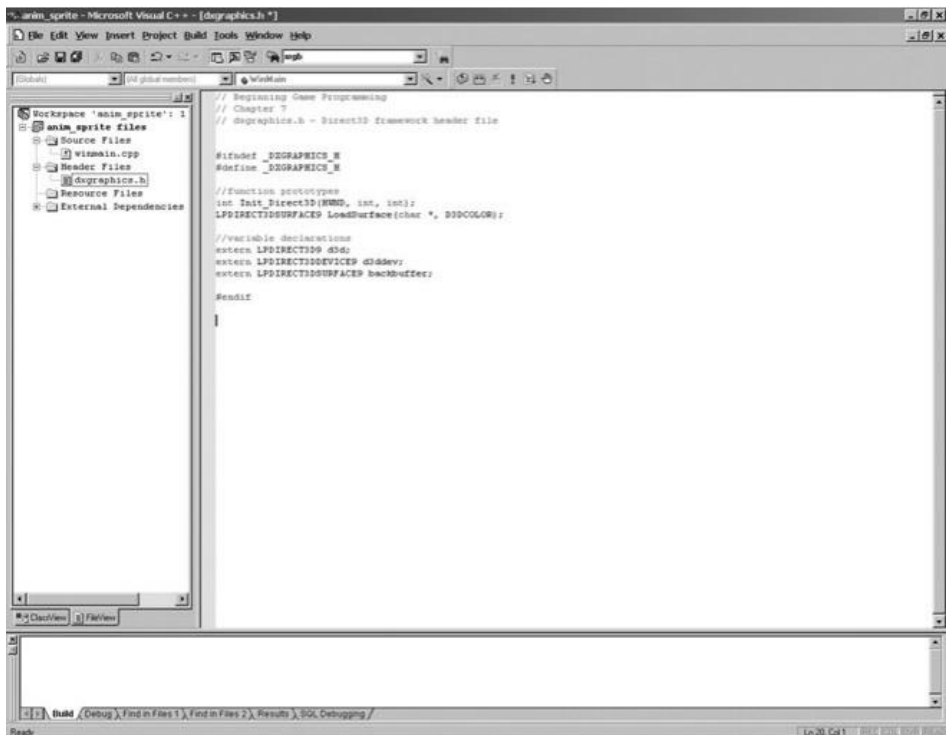
```

Bây giờ mở menu File, chọn New để mở hộp thoại, chọn C/C++ Header File từ list, đặt tên “dxgraphics.h”, hình 3.4.



Hình 3.4. Mở hộp thoại chọn C/C++ Header File

Đây là code trong “dxgraphics.h”. Sau khi ta vừa thêm file đó vào project sẽ giống như hình 3.5



Hình 3.5. Sau khi thêm tập tin dxgraphics.h

dxgraphics.h

```
// dxgraphics.h - Direct3D framework header file

#ifndef _DXGRAPHICS_H
#define _DXGRAPHICS_H

//function prototypes
int Init_Direct3D(HWND, int, int, int);
LPDIRECT3DSURFACE9 LoadSurface(char *, D3DCOLOR);

//variable declarations
extern LPDIRECT3D9 d3d;
extern LPDIRECT3DDEVICE9 d3ddev;
extern LPDIRECT3DSURFACE9 backbuffer;
```

```
#endif
```

Tương tự, thêm source file “dxgraphics.cpp” vào project. Project sẽ chứa các hàm đã khai báo trong file header vừa rồi.

Dxgraphics.cpp

```
// dxgraphics.cpp - Direct3D framework source code file

#include <d3d9.h>
#include <d3dx9.h>
#include "dxgraphics.h"

//variable declarations
LPDIRECT3D9 d3d = NULL;
LPDIRECT3DDEVICE9 d3ddev = NULL;
LPDIRECT3DSURFACE9 backbuffer = NULL;

int Init_Direct3D(HWND hwnd, int width, int height, int
fullscreen)
{
    //initialize Direct3D
    d3d = Direct3DCreate9(D3D_SDK_VERSION);
    if (d3d == NULL)
    {
        MessageBox(hwnd, "Error initializing Direct3D",
"Error", MB_OK);
        return 0;
    }

    //set Direct3D presentation parameters
    D3DPRESENT_PARAMETERS d3dpp;
    ZeroMemory(&d3dpp, sizeof(d3dpp));

    d3dpp.Windowed = (!fullscreen);
    d3dpp.SwapEffect = D3DSWAPEFFECT_COPY;
    d3dpp.BackBufferFormat = D3DFMT_X8R8G8B8;
    d3dpp.BackBufferCount = 1;
    d3dpp.BackBufferWidth = width;
    d3dpp.BackBufferHeight = height;
    d3dpp.hDeviceWindow = hwnd;
```

```

//create Direct3D device
d3d->CreateDevice(
    D3DADAPTER_DEFAULT,
    D3DDEVTYPE_HAL,
    hwnd,
    D3DCREATE_SOFTWARE_VERTEXPROCESSING,
    &d3dpp,
    &d3ddev);

if (d3ddev == NULL)
{
    MessageBox(hwnd, "Error creating Direct3D
device", "Error", MB_OK);
    return 0;
}

//clear the backbuffer to black
d3ddev->Clear(0, NULL, D3DCLEAR_TARGET,
D3DCOLOR_XRGB(0,0,0), 1.0f, 0);

//create pointer to the back buffer
d3ddev->GetBackBuffer(0, 0, D3DBACKBUFFER_TYPE_MONO,
&backbuffer);

return 1;
}

LPDIRECT3DSURFACE9 LoadSurface(char *filename, D3DCOLOR
transcolor)
{
    LPDIRECT3DSURFACE9 image = NULL;
    D3DXIMAGE_INFO info;
    HRESULT result;

    //get width and height from bitmap file
    result = D3DXGetImageInfoFromFile(filename, &info);
    if (result != D3D_OK)
        return NULL;

    //create surface
    result = d3ddev->CreateOffscreenPlainSurface(

```



```

        info.Width,           //width of the surface
        info.Height,         //height of the surface
        D3DFMT_X8R8G8B8,     //surface format
        D3DPOOL_DEFAULT,     //memory pool to use
        &image,              //pointer to the surface
        NULL);               //reserved (always NULL)

    if (result != D3D_OK)
        return NULL;

    //load surface from file into newly created surface
    result = D3DXLoadSurfaceFromFile(
        image,                //destination surface
        NULL,                 //destination palette
        NULL,                 //destination rectangle
        filename,             //source filename
        NULL,                 //source rectangle
        D3DX_DEFAULT,         //controls how image is
filtered
        transcolor,           //for transparency (0 for
none)
        NULL);               //source image info (usually
NULL)

    //make sure file was loaded okay
    if (result != D3D_OK)
        return NULL;

    return image;
}

```

Đó là tất cả những code cho Windows và DirectX như trước giờ ta đã tìm hiểu. Như chúng ta cũng thấy, vẫn còn một con đường dài phải đi, và chúng ta sẽ đi chi tiết hơn trong phần sau theo. Giờ hãy tập trung vào code của project Anim_Sprite.

Thêm một tập tin header nữa “game.h”. Đây là code cho “game.h”

game.h

```
// Anim_Sprite program header file

#ifndef _GAME_H
#define _GAME_H

#include <d3d9.h>
#include <time.h>
#include <stdio.h>
#include <stdlib.h>
#include "dxgraphics.h"

//application title
#define APPTITLE "Anim_Sprite"

//screen setup
#define FULLSCREEN 0          //1 = fullscreen, 0 = windowed
#define SCREEN_WIDTH 640
#define SCREEN_HEIGHT 480

//macros to read the keyboard asynchronously
#define KEY_DOWN(vk_code) ((GetAsyncKeyState(vk_code) & 0x8000) ? 1 : 0)
#define KEY_UP(vk_code) ((GetAsyncKeyState(vk_code) & 0x8000) ? 1 : 0)

//function prototypes
int Game_Init(HWND);
void Game_Run(HWND);
void Game_End(HWND);

//sprite structure
typedef struct {
    int x,y;
    int width,height;
    int movex,movey;
    int curframe,lastframe;
    int animdelay,animcount;
} SPRITE;

#endif
```

Tương tự thêm file source “game.cpp” vào project. Đây là code cho “game.cpp”

game.cpp

```
// Anim_Sprite program source code file

#include "game.h"

LPDIRECT3DSURFACE9 kitty_image[7];
SPRITE kitty;

//timing variable
long start = GetTickCount();

//initializes the game
int Game_Init(HWND hwnd)
{
    char s[20];
    int n;

    //set random number seed
    srand(time(NULL));

    //load the sprite animation
    for (n=0; n<6; n++)
    {
        sprintf(s,"cat%d.bmp",n+1);
        kitty_image[n] = LoadSurface(s, D3DCOLOR_XRGB(255,0,255));
        if (kitty_image[n] == NULL)
            return 0;
    }

    //initialize the sprite's properties
    kitty.x = 100;
    kitty.y = 150;
    kitty.width = 96;
    kitty.height = 96;
    kitty.curframe = 0;
    kitty.lastframe = 5;
    kitty.animdelay = 2;
    kitty.animcount = 0;
    kitty.movex = 8;
    kitty.movey = 0;

    //return okay
    return 1;
}
```

```

//the main game loop
void Game_Run(HWND hwnd)
{
    RECT rect;

    //make sure the Direct3D device is valid
    if (d3ddev == NULL)
        return;

    //after short delay, ready for next frame?
    //this keeps the game running at a steady frame rate
    if (GetTickCount() - start >= 30)
    {
        //reset timing
        start = GetTickCount();

        //move the sprite
        kitty.x += kitty.movex;
        kitty.y += kitty.movey;

        //"warp" the sprite at screen edges
        if (kitty.x > SCREEN_WIDTH - kitty.width)
            kitty.x = 0;
        if (kitty.x < 0)
            kitty.x = SCREEN_WIDTH - kitty.width;

        //has animation delay reached threshold?
        if (++kitty.animcount > kitty.animdelay)
        {
            //reset counter
            kitty.animcount = 0;

            //animate the sprite
            if (++kitty.curframe > kitty.lastframe)
                kitty.curframe = 0;
        }
    }

    //start rendering
    if (d3ddev->BeginScene())
    {
        //erase the entire background
        d3ddev->ColorFill(backbuffer, NULL,
D3DCOLOR_XRGB(0,0,0));

        //set the sprite's rect for drawing

```

```

        rect.left = kitty.x;
        rect.top = kitty.y;
        rect.right = kitty.x + kitty.width;
        rect.bottom = kitty.y + kitty.height;

        //draw the sprite
        d3ddev->StretchRect(kitty_image[kitty.curframe],
        NULL, backbuffer, &rect, D3DTEXF_NONE);

        //stop rendering
        d3ddev->EndScene();
    }

    //display the back buffer on the screen
    d3ddev->Present(NULL, NULL, NULL, NULL);

    //check for escape key (to exit program)
    if (KEY_DOWN(VK_ESCAPE))
        PostMessage(hwnd, WM_DESTROY, 0, 0);
}

//frees memory and cleans up before the game ends
void Game_End(HWND hwnd)
{
    int n;

    //free the surface
    for (n=0; n<6; n++)
        kitty_image[n]->Release();
}

```

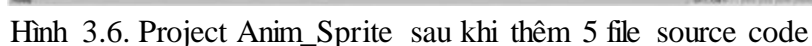
Cuối cùng, kết quả của việc add các file đã xong như hình 3.6

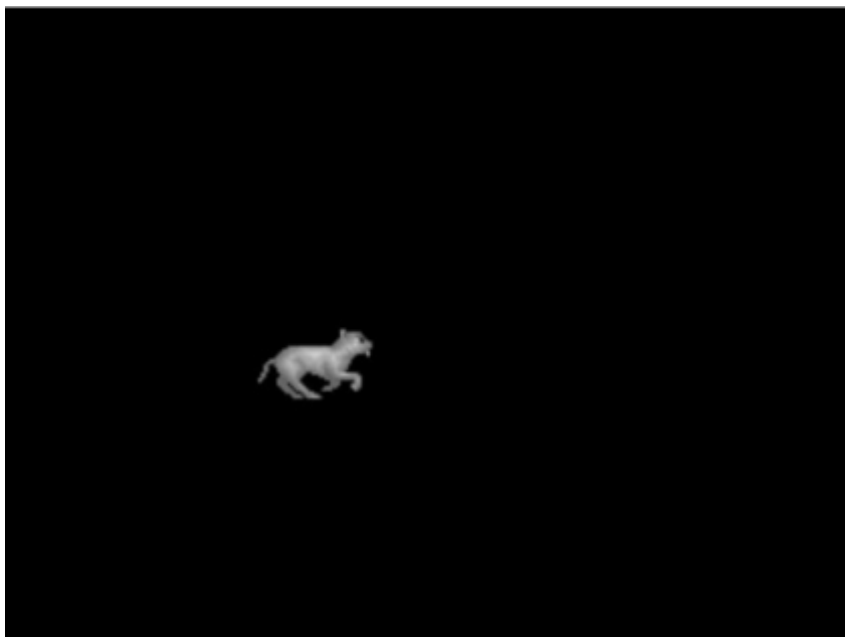
3.1.4. Đồ họa của Sprite

Dĩ nhiên là để có thể chạy chương trình này, ta cần phải có các tập tin hình ảnh mà ta dùng trong chương trình. Khi bạn chạy chương trình, sẽ giống như hình 3.7

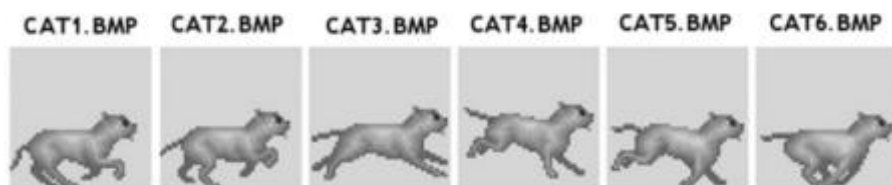
<http://www.arifeldman.com>. Ariman đưa ra SpriteLib để giúp những lập trình viên game bắt đầu mà không cần lo lắng về nội dung trong quá trình học. Có hàng trăm sprite (tĩnh và động) và hình nền (background) trong SpriteLib, and Ari vẫn đang tiếp tục thêm vào đó.

Có 6 khung hình của con mèo chuyển động trong hình 3.7. Bạn có thể copy file từ source code mẫu vào thư mục project trong ổ cứng của bạn để chạy chương trình.





Hình 3.7: Chương trình Anim_Sprite vẽ lên màn hình con mèo chuyển động



Hình 3.8: Sprite con mèo chuyển động với 6 khung hình

Sáu khung hình “cat.bmp” với kích thước 96x96, và có một nền màu hồng với giá trị RGB là (255, 0, 255). Nếu bạn tìm đến hàm

“Game_Init” đã đưa code ở trên, bạn sẽ để ý tới lời gọi “LoadSurface” có một giá trị màu cho tham số thứ 2.

```
//load the sprite animation
for (n=0; n<6; n++)
```

```

{
    sprintf(s, "cat%d.bmp", n+1);
    kitty_image[n] = LoadSurface(s,
D3DCOLOR_XRGB(255,0,255));
    if (kitty_image[n] == NULL)
        return 0;
}

```

Giá trị màu được thể hiện bởi D3DCOLOR_XRGB(255, 0, 255) là màu hồng. Nhưng tại sao hàm “LoadSurface” lại cần phải quan tâm về màu nền. Hãy để ý, project này không sử dụng chế độ trong suốt (transparent). Ta cần chỉ ra màu trong suốt để hàm “StretchRect” vẽ màu trong suốt thành màu đen (hàm StretchRect không hỗ trợ màu trong suốt thật sự). Thật tiện dụng, bởi vì ta có thể sử dụng bất kỳ màu gì mình muốn trong khi sửa sprite để bù nó từ background, và nó sẽ được vẽ thành màu đen khi load vào game.

Chúng ta muốn con mèo trông như thế nào khi được vẽ lên một nền không phải đen? Có một vài sửa chữa cần làm trong chương trình để thêm một hình nền. Ta có thêm file “background.bmp” vào trong thư mục project.

Đầu tiên, thêm dòng này ở gần trên cùng của “game.cpp” với một vài biến được khai báo.

```
LPDIRECT3DSURFACE9 back;
```

Tiếp theo, trong “Game_Init()”, thêm vài dòng để load ảnh nền lên một surface mới:

```
back = LoadSurface("background.bmp", NULL );
```

Tiếp theo, xuống “Game_Run()”, ẩn dòng “ColorFill” và thay bằng lời gọi “StretchRect”, như sau:


```
//d3ddev->ColorFill(backbuffer, NULL,  
D3DCOLOR_XRGB(0,0,0));  
d3ddev->StretchRect(back, NULL, backbuffer, NULL,  
D3DTEXF_NONE);
```

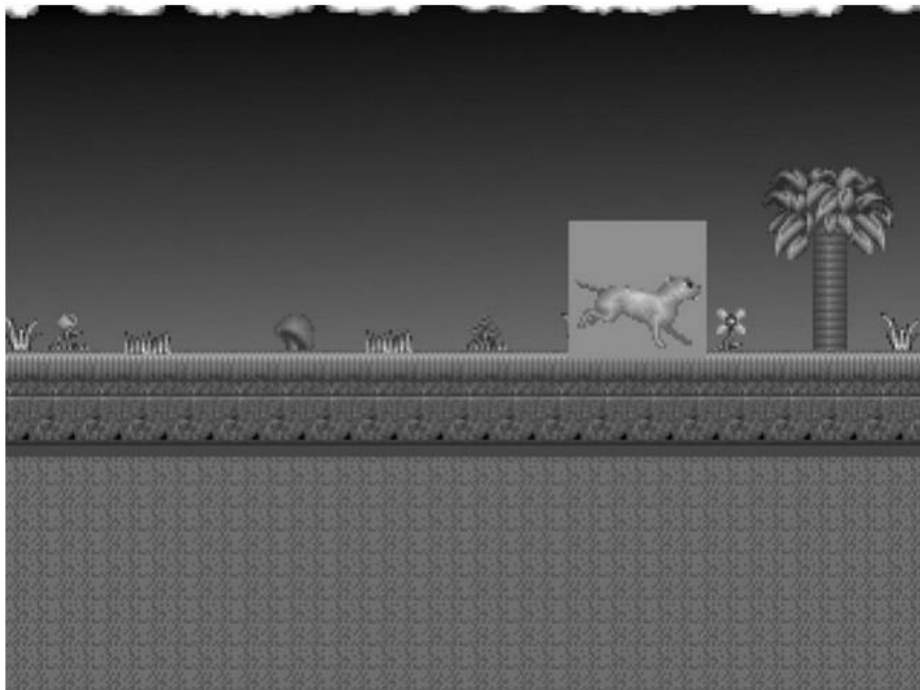
Cuối cùng, thêm 1 dòng ở “Game_End()” để giải phóng bộ nhớ đã sử dụng bởi backgroud surface.

```
Back->Release();
```

Bây giờ, chạy chương trình một lần nữa, hình nền sẽ được hiển thị giống như hình 3.9. Tại sao sau tất cả mọi việc ta làm, con mèo vẫn không được vẽ với màu trong suốt? Bởi vì chúng ta chỉ mới làm việc với surface thô, chuyển màu nền của sprite thành đen là cách tốt nhất chúng ta có thể làm lúc này này. Chúng ta sẽ đi qua sprite có màu trong suốt thật sự ở chương tiếp theo.

Một cách tự nhiên, ta có thể sử dụng màu đen cho màu nền trong suốt của sprite, nhưng nhớ rằng, hầu hết mọi người đều không dùng màu đen, họ sử dụng nhiều màu xen kẽ để dễ dàng hiển thị khi thay đổi ảnh gốc. Để xem một surface sẽ hiển thị như thế nào nếu không xác định màu trong suốt, ta có thể sửa lời gọi hàm “D3DXLoadSurfaceFromFile” trong “dxgraphics.cpp”.

Đề ý đến tham số thứ 2 từ cuối, “transcolor”. Nếu thay bằng 0, rồi chạy lại chương trình, Direct3D sẽ bỏ qua màu trong suốt trong bức ảnh và vẽ một cách tự nhiên.



Hình 3.9. Con mèo được vẽ không có màu trong suốt

```
result = D3DXLoadSurfaceFromFile(
    image,                //destination surface
    NULL,                 //destination palette
    NULL,                 //destination rectangle
    filename,             //source filename
    NULL,                 //source rectangle
    D3DX_DEFAULT,         //controls how image is filtered
    transcolor,           //for transparency (0 for none)
    NULL);                //source image info (usually NULL)
```

Concept Art (Hình khái niệm)

Ngày nay, hầu hết các sprite được vẽ từ những hình mẫu 3D (3D model). Hiếm khi có game được vẽ bằng tay thuần. Tại sao? Bởi vì 3D model có thể xoay, lắp ghép, và thay đổi tùy biến dễ dàng sau khi được tạo ra, trong khi hình vẽ 2D thì không được như vậy. Thật đơn giản khi thay lắp một hình mới cho 1 model, sau đó xuất ra những frame mới để cho game sử dụng. Ta sẽ không thảo luận toàn bộ quá trình tạo ra một concept art và đưa nó vào game như thế nào nhưng ta có một vài ví dụ.

Hình bên dưới là một bản phát thảo mà của một game RPG. Đây là bản phát thảo ban đầu của một nhân vật sẽ là một nữ cung thủ được vẽ bởi Jessica K. Fuerst.

Pixel artist hay 3D modelers (họa sĩ 2D hay họa sĩ dựng hình 3D) sử dụng những bản vẽ phát thảo để dựng nên những hình 2D và 3D cho game. Concept art là cực kỳ quan trọng bởi vì nó giúp bạn nghĩ xuyên suốt bản thiết kế của bạn và thật sự làm nhân vật như đang sống. Nếu bạn không phải là 1 họa sĩ tài ba hay không thể cố gắng để trả tiền cho những họa sĩ vẽ concept art cho game, ít nhất bạn cũng cố gắng vẽ bằng bút chì lên giấy những gì quan trọng nhất.



Hình 3.10. Phát thảo một nữ cung thủ cho một game RPG. Hình vẽ bởi Jessica K. Fuerst

Hình dưới đây là một bản màu của một nhân vật hư cấu nữ, vẽ bởi Eden Celeste, lấy cảm hứng từ một vài game RPG. Đôi khi lướt các trang bán hình họa online cũng là một cách để lấy cảm hứng cho game. Rất nhiều họa sĩ sẵn sàng để được thuê, hay bán các sản phẩm đã có của họ để dùng cho game.



Hình 3.11. Bản màu của một nhân vật hư cấu cho 1 game RPG, vẽ bởi Eden Celeste.

3.1.5. Diễn giải về sprite chuyển động

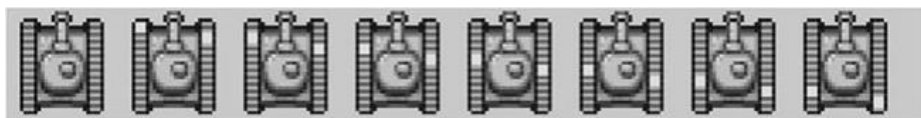
Bây giờ chúng ta sẽ bàn đến source code của chương trình hiển thị sprite chuyển động trên màn hình. Ta sẽ đi qua các khía cạnh quan trọng của chương trình để giúp bạn đọc hiểu.

Một sprite là một bức ảnh nhỏ được vẽ lên màn hình để thể hiện 1 đối tượng trong game. Sprite có thể được sử dụng cho đối tượng tĩnh như cây cối, đất đá, hoặc đối tượng động như hero, nhân vật nhập vai, ... Một điều cơ bản trong phát triển game: sprite là một phạm trù chỉ có trong game 2D. Chúng ta sẽ không tìm thấy sprite trong game 3D, trừ khi sprite được vẽ ở trên giao diện màn hình game 3D. Ví dụ, trong một game nhiều người chơi với chức năng chat, dòng tin nhắn xuất hiện trên màn hình. Hình dưới đây cho một ví dụ về một bitmap font (font ảnh).



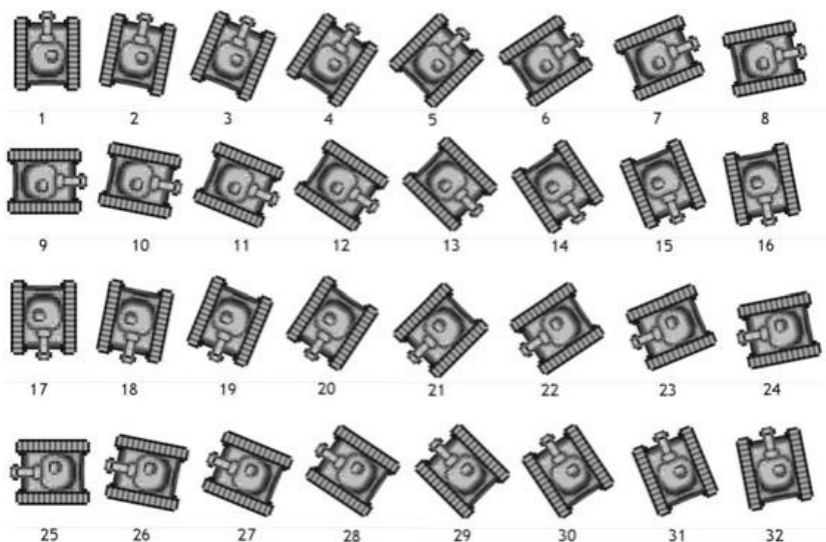
Hình 3.12. Bitmap font

Một sprite lưu một bức ảnh là 1 chuỗi các *tile*, mỗi tile sẽ hiển thị một khung hình của chuỗi chuyển động. Một chuyển động có thể trông không giống như đang di chuyển hơn là thay đổi hướng, như trong trường hợp của chiếc máy bay hay tàu bay trong game dạng shoot-em-up. Hình 3.13 hiển thị một chiếc xe tăng nhìn từ một phía nhưng hiển thị cho chuyển động di chuyển.



Hình 3.13. Một xe tăng với có dạng chuyển động, vẽ bởi Ari Feldman

Bây giờ cần làm gì nếu ta muốn xe tăng đó quay sang hướng khác? Như ta có thể đoán ra, số khung hình có thể tăng lên cấp số nhân nếu làm cho từng hướng đầy đủ những khung hình chuyển động như trên. Hình 3.14. thể hiện xe tăng không chuyển động được quay sang 32 hướng khác nhau cho quá trình quay mượt mà. Thật không may mắn, khi ta thêm từng bước chuyển động của xe tăng vào, 32 frame đó trở thành $32 * 8 = 256$ frame. Thật khó để lập trình một chiếc xe tăng với quá nhiều frame như vậy, và làm cách nào để bạn lưu nó trong ảnh bitmap. Một giải pháp tốt hơn là dùng ít frame ban đầu cho đến khi game hoàn tất và rồi có thể thêm nhiều frame chi tiết chuyển động về sau.



Hình 3.14. Một chuyển động quay gồm 32 khung hình của xe tăng.

MechCommander (MicroProse, FASA Studios) là một trong số những game động nhất được tạo, và được viết và nếu không phải vì AI quá tệ và chơi độ khó không thực tế của nó thì ta sẽ bình chọn nó là game hay nhất. Điều tuyệt vời về MechCommander là nó có những sprite 2D rất chi tiết. Mỗi chuyển động trong game là một sprite 2D lưu trong 1 chuỗi các file. Điều kì diệu của game là nó có đến 100.000 frame! Hãy tưởng tượng về lượng thời gian nó tốn cho model đầu tiên với 1 chương trình hoạt họa 3D (như 3DS Max), và rồi vẽ nó ra 100.000 hình chụp của rất nhiều góc quay, vị trí và rồi thay đổi kích thước lại và rồi cuối cùng gắn lại thành từng sprite.

Một loại sprite phổ biến là platformer game sprite, hình 3.15. Việc lập trình một game platform thì khó hơn lập trình một game shoot-em-up, nhưng kết quả thường tương xứng với những gì bỏ ra.



Hình 3.15. Một nhân vật chuyển động trong game platform

SPRITE Struct

Điểm chính của chương trình vẽ sprite tron suốt này là struct **SPRITE** được định nghĩa trong game.h:

```
//sprite structure
typedef struct {
    int x,y;
    int width,height;
    int movex,movey;
    int curframe,lastframe;
    int animdelay,animcount;
} SPRITE;
```

Các biến tự nhiên của struct là: x, y, width và height, còn lại là các biến movex và movey. Các biến này được sử dụng để update tọa độ x, y của sprite suốt quá trình update frame. Biến curframe và lastframe dùng để giữ giá trị frame hiện tại của chuyển động, curframe được update liên tục trong vòng lặp game, và khi nó đạt đến giá trị lastframe, nó sẽ được gán lại giá trị 0, cứ thế lặp liên tục. Biến animdelay và animcount dùng để làm việc với hai biến vừa rồi nhằm giới hạn thời gian hiển thị cho mỗi frame. Nếu frame được update liên tục ở mỗi vòng lặp game, chuyển động sẽ diễn ra quá nhanh. Ta sẽ không muốn làm cho vòng lặp game chậm lại chỉ để giữ cho chuyển động đó như ý muốn, do đó giải pháp là dùng việc update curframe một khoảng thời gian.

Sprite “kitty” được định nghĩa như sau:


```
LPDIRECT3DSURFACE9 kitty_image[7];  
SPRITE kitty;
```

Sprite được khởi tạo trong hàm `Game_Init` và được gán giá trị sau:

```
//initialize the sprite's properties  
kitty.x = 100;  
kitty.y = 150;  
kitty.width = 96;  
kitty.height = 96;  
kitty.curframe = 0;  
kitty.lastframe = 5;  
kitty.animdelay = 2;  
kitty.animcount = 0;  
kitty.movex = 8;  
kitty.movey = 0;
```

Vòng lặp game

Hàm `Game_Run` là vòng lặp game, do đó, hãy luôn nhớ rằng nó phải xử lý trong lần update màn hình và chỉ có vậy. Đừng bao giờ đặt vòng lặp `while` ở đây hoặc giống như thế (bởi vì điều khiển sẽ không trả về `WinMain`).

Có 2 phần cho hàm `Game_Run`. Phần đầu thực hiện di chuyển và chuyển động cho sprite. Phần 2 thực hiện vẽ sprite lên màn hình. Lý do chia ra thành 2 phần (1 cho logic, 1 cho làm mới màn hình) là bởi vì ta không muốn tốn nhiều thời gian xử lý giữa 2 lời gọi `BeginScene` và `EndScene`, do đó, hãy tối giản code ở đây, chỉ dùng để update đồ họa và đặt các xử lý khác ở phần còn lại.

Dòng chủ chốt mà ta cần quan tâm là đoạn di chuyển sprite, giữ sprite nằm trên màn hình và chuyển động sprite:

```
//move the sprite  
kitty.x += kitty.movex;  
kitty.y += kitty.movey;  
  
//"warp" the sprite at screen edges  
if (kitty.x > SCREEN_WIDTH - kitty.width)  
    kitty.x = 0;
```

```
if (kitty.x < 0)
    kitty.x = SCREEN_WIDTH - kitty.width;

//has animation delay reached threshold?
if (++kitty.animcount > kitty.animdelay)
{
    //reset counter
    kitty.animcount = 0;

    //animate the sprite
    if (++kitty.curframe > kitty.lastframe)
        kitty.curframe = 0;
}
```

Bạn đọc có thấy sự tiện dụng đoạn code di chuyển và chuyển động nhân vật khi ta tạo ra struct SPRITE? Code này là đủ để đưa vào một hàm riêng, truyền vào một biến SPRITE để update nhiều sprite trong một game.

3.2. Vẽ Sprite Trong Suốt

Đối tượng D3DXSprite thật sự là một điều tuyệt vời cho bất kì lập trình viên nào có kế hoạch viết game 2D sử dụng Direct3D. Một trong những lợi ích của việc thực hiện chúng là ta có đầy đủ những thể hiện của 3D theo ý khi sử dụng hàm 2D nhanh như dùng DirectDraw cũ. Bằng cách xử lí sprite như là một texture và rendering các sprite như một rectangle (bao gồm hai rectangle, vì đây là trường hợp với 3D rectangles), bạn có thể transform các sprite!

Transform nghĩa là ta có thể di chuyển sprite với bộ tăng tốc phần cứng 3D. Ta có thể vẽ sprite trong suốt bằng cách đặc tả màu alpha trong bitmap nguồn thể hiện cho pixel trong suốt. Màu đen (0, 0, 0) là màu thường được sử dụng cho độ trong suốt, nhưng đó không phải là một ý tưởng hay. Tại sao ? Đó là bởi vì nó khó chỉ ra pixel nào là trong suốt và pixel nào là màu tối. Tốt hơn hết là sử dụng màu hồng (255,0,255) bởi vì màu này hiếm khi được sử dụng trong đồ họa game và hiện rõ

ràng trong ảnh nguồn. Ta có thể phát hiện được điểm ảnh trong suốt trong một hình ảnh như vậy.

Rõ ràng, phương thức *D3DXSprite* là cách, nhưng chúng ta dự định đưa ra một phương thức đơn giản hơn bởi vì nó sẽ có ích trong trường hợp không sử dụng hình trong suốt, ví dụ như khi vẽ hình nền.

3.2.1. Tạo Đối Tượng Sprite Handler

Đối tượng *D3DXSprite* chỉ là người điều khiển sprite có chứa những hàm để vẽ sprites từ texture (với những thông tin về transformations). Đây là cách ta khai báo chúng:

```
LPD3DXSPRITE sprite_handler;
```

Bạn có thể khởi tạo đối tượng bằng cách gọi tới hàm

D3DXCreateSprite. Phần việc của nó là gắn sprite handler vào đối tượng *Direct3D* chính và device để chúng biết làm thế nào để vẽ sprite vào trong back buffer.

```
HRESULT WINAPI D3DXCreateSprite(  
    LPDIRECT3DDEVICE9 pDevice,  
    LPD3DXSPRITE *ppSprite  
);
```

Và đây là ví dụ về gọi hàm:

```
result = D3DXCreateSprite(d3ddev, &sprite_handler);
```

3.2.2. Bắt đầu Sprite Handler

Ta sẽ tải lên một tấm ảnh nhỏ, nhưng trong lúc chờ, chúng ta cần biết làm thế nào để sử dụng *D3DXSprite*. Khi gọi *BeginScene* từ *Direct3D* device chính, ta có thể bắt đầu vẽ sprites. Việc đầu tiên ta phải làm đó là khóa surface để sprites có thể vẽ. Ta làm điều này bằng cách gọi hàm *D3DXSprite.Begin* với định dạng:

```
HRESULT Begin( DWORD Flags );
```

Tham số flags được yêu cầu và thường là *D3DXSPRITE_ALPHABLEND*, nghĩa là sprite được vẽ hỗ trợ trong suốt. Đây là ví dụ:

```
sprite_handler->Begin(D3DXSPRITE_ALPHABLEND);
```

3.2.3. Vẽ Sprite

Vẽ sprite hơi phức tạp hơn so với việc thể hiện một hình chữ nhật nguồn và đích như với surface trong chương trước. Tuy nhiên, *D3DXSprite* chỉ sử dụng một hàm, *Draw*, cho tất cả những tự chọn về transformation, vì vậy, điều ta cần biết đó là làm cách nào để hàm này làm việc để có thể biểu diễn được độ trong suốt, tỷ lệ và độ xoay chỉ bằng thay thế những tham số. Đây là cách khai báo hàm *Draw*:

```
HRESULT Draw(  
    LPDIRECT3DTEXTURE9 pTexture,  
    CONST RECT *pSrcRect,  
    CONST D3DXVECTOR3 *pCenter,  
    CONST D3DXVECTOR3 * pPosition,  
    D3DCOLOR Color  
);
```

Tham số đầu tiên là quan trọng nhất, vì nó đặc tả texture được sử dụng cho hình ảnh nguồn của sprite. Tham số thứ hai cũng quan trọng không kém, vì ta có thể sử dụng nó để cắt “tile” khỏi ảnh nguồn và lưu trữ tất cả những frame animation của sprite trong một tập tin ảnh (chi tiết hơn trong những phần sau). Tham số thứ 3 đặc tả điểm tâm được dùng để xoay. Tham số thứ tư đặc tả cho vị trí của sprite, và nó thường là nơi để đặt giá trị x, y. Tham số cuối cùng mô tả cho màu thay thế sẽ được áp dụng khi vẽ sprite (không ảnh hưởng khi vẽ trong suốt).

D3DXVECTOR3 là một dữ liệu mới được phát sinh với *DirectX 9.0b*, bao gồm ba biến là: x, y và z.

```
typedef struct D3DXVECTOR3 {  
    FLOAT x;  
    FLOAT y;  
    FLOAT z;  
} D3DXVECTOR3;
```

x và y, ta chỉ cần dùng khi di chuyển sprite trên màn hình 2D. Ta sẽ có ví dụ về việc sử dụng Draw trong một chương trình ở sau.

3.2.4. Dùng Sprite Handler

Sau khi đã kết thúc việc vẽ sprite, nhưng trước khi gọi EndScene, ta phải gọi D3DXSprite.End để mở khóa surface cho các tiến trình khác có thể sử dụng. Đây là cú pháp:

```
HRESULT End(void);
```

Cách sử dụng thật sự dễ dàng bởi vì hàm này rất ngắn:

```
sprite_handler->End();
```

3.2.5. Tải Sprite Image

Một lần nữa cần lưu ý là D3DXSprite sử dụng texture thay cho surface để lưu trữ sprite image. Vì vậy, trong khi đối tượng LPDIRECT3DSURFACE9 được sử dụng cho sprites trong chương trước thì trong chương này, ta sẽ dùng đối tượng LPDIRECT3DTEXTURE9 thay thế. Nếu ta tạo ra game đi cảnh cuộn màn hình như là *Super Mario World* hay *R-Type* hay *Mars Matrix*, ta sẽ sử dụng surface để vẽ (và cuộn) hình nền, nhưng ta sẽ sử dụng texture cho những sprite thể hiện các đối tượng game/phi thuyền/kẻ thù. Ở đây thực sự không có một lợi ích nào cho hiệu suất game khi sử dụng surface thay cho texture, bởi vì video card (với chip 3D nâng cao) đắt tiền sẽ thể hiện sprite của bạn lên màn hình sử dụng hệ thống phần cứng kết nối texture nhanh hơn bất cứ những gì ta có thể làm với phần mềm. Chúng ta sẽ để Direct3D vẽ những sprite của chúng ta.

Việc đầu tiên phải là phải tạo D3DXSprite để chứa đối tượng texture mà bitmap image của sprite đã tải lên:

```
LPDIRECT3DTEXTURE9 texture = NULL;
```

Việc tiếp theo cần làm là lấy độ phân giải của tập tin ảnh (giả sử rằng ta đã có sprite bitmap) sử dụng hàm D3DXGetImageInfoFromFile:

D3DXIMAGE_INFO info;

```
result = D3DXGetImageInfoFromFile("image.bmp", &info);
```

Nếu tập tin đã tồn tại, thì ta sẽ có được Width và Height, những thứ cần thiết cho bước tiếp theo. Tiếp theo, ta tải ảnh của sprite từ tập tin ảnh vào trong texture bằng cách gọi hàm D3DXCreateTextureFromFile:

```
HRESULT WINAPI D3DXCreateTextureFromFileEx(  
    LPDIRECT3DDEVICE9 pDevice,  
    LPCTSTR pSrcFile,  
    UINT Width,  
    UINT Height,  
    UINT MipLevels,  
    DWORD Usage,  
    D3DFORMAT Format,  
    D3DPOOL Pool,  
    DWORD Filter,  
    DWORD MipFilter,  
    D3DCOLOR ColorKey,  
    D3DXIMAGE_INFO *pSrcInfo,  
    PALETTEENTRY *pPalette,  
    LPDIRECT3DTEXTURE9 *ppTexture  
);
```

Đừng lo lắng việc có quá nhiều tham số, vì hầu hết chúng được điền vào với những giá trị mặc định hoặc NULL. Điều duy nhất ta cần làm là viết một phương thức để đặt những thông tin này vào trong và trả về texture cho mình. Đây là hàm, được gọi là *LoadTexture*:

```

LPDIRECT3DTEXTURE9 LoadTexture(char *filename, D3DCOLOR
transcolor)
{
    // Con trỏ texture
    LPDIRECT3DTEXTURE9 texture = NULL;

    // struct để đọc thông tin tệp bitmap
    D3DXIMAGE_INFO info;

    // trả về giá trị windows thông thường
    HRESULT result;

    // lấy ra thông tin width và height của tệp bitmap
    result = D3DXGetImageInfoFromFile(filename, &info);
    if( result != D3D_OK)
        return NULL;

    // tạo texture mới bằng cách tải lên tệp bitmap
    result = D3DXCreateTextureFromFileEx(
        d3ddev,          //Đối tượng Direct3D
        filename,         //tên tệp bitmap
        info.Width,       //Width của tệp bitmap
        info.Height,      //Height của tệp bitmap
        1,               //kết nối level(1 nếu không có thay đổi)
        D3DPPOOL_DEFAULT, //kiểu của surface (thông thường)
        D3DFMT_UNKNOWN,   //định dạng surface (mặc định)
        D3DPPOOL_DEFAULT, //lớp bộ nhớ cho texture
        D3DX_DEFAULT,     //bộ lọc hình ảnh
        D3DX_DEFAULT,     //bộ lọc mip
        transcolor,       //màu chỉ ra trong suốt
        &info,            //thông tin tệp bitmap (từ tệp tải lên)
        NULL,             //đổ màu
        &texture);        //texture đích

    // Đảm bảo texture đã được tải lên thành công

```

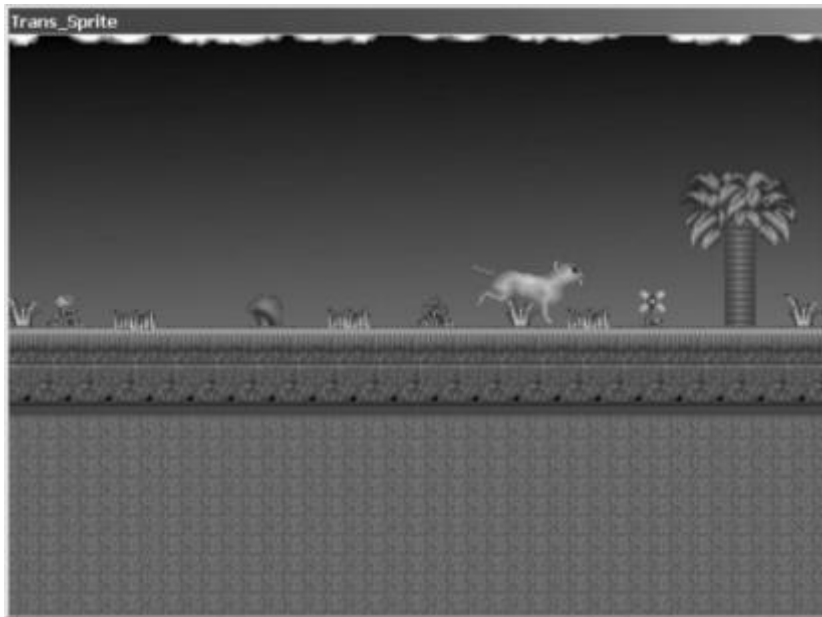
```
if(result != D3D_OK)
    return NULL;

return texture;
}
```

Vì kỹ thuật texture sẽ được mô tả kỹ hơn ở sau, nên chúng ta sẽ bỏ qua phần giải thích chi tiết hàm này ở đây. Chúng ta chỉ đảm bảo rằng sprites được tải lên. Hãy nhớ là ta nên phớt lờ đi những thứ ta không cần biết ngay và đi tiếp cho đến khi xong những thứ bạn cần hoàn thành trước. Ta chỉ quay lại những bước nâng cao khi có thời gian, mục đích rõ ràng để làm điều này.

3.2.6. Chương trình Trans_Sprite

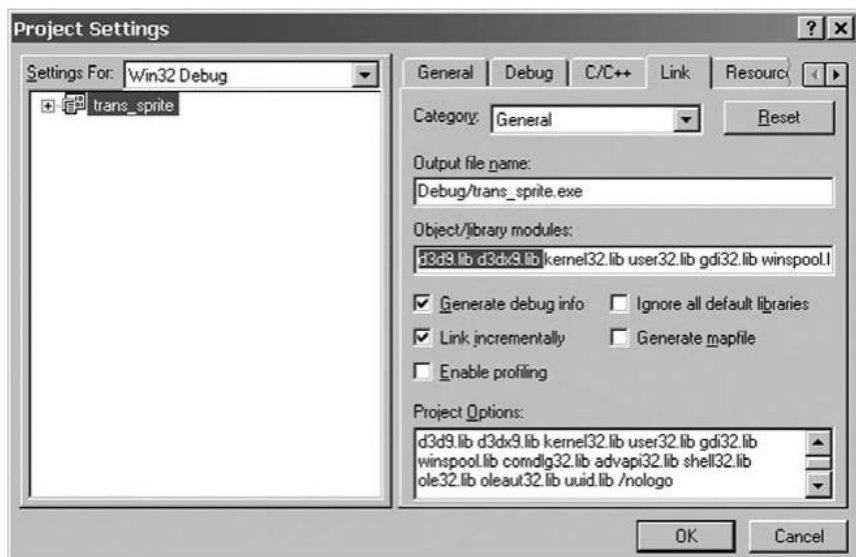
Bây giờ ta đã hiểu về cách D3DXSprite làm với Direct3D texture để vẽ sprite trong suốt (ít nhất là về nguyên lý), hãy viết một chương trình ngắn thể hiện làm cách nào để gắn chúng với nhau. Ta có thể tải project từ bộ mã nguồn ví dụ nếu muốn, hoặc có thể thay đổi Anim_Sprite project từ chương trước. Chúng ta sẽ khuyến khích bạn đọc tạo một project mới từ đầu. Đó là cách tốt nhất để học!. Hình minh họa 3.16 thể hiện chương trình Trans_Sprite.



Hình 3.16. Chương trình Trans_Sprite giải thích cách vẽ sprite trong suốt với Direct3D.

3.2.7. Tạo Project Trans_Sprite

Đầu tiên, mở Visual C++ và tạo một project Win32 Application với tên là Trans_Sprite. Tiếp theo, mở thực đơn Project và chọn Setting để mở ra Project Setting dialog. Click vào Link tab và thêm **d3d9.lib** và **d3dx9.lib** và trong Object/library mô-đun như minh họa 3.17.



Hình 3.17. Thêm hỗ trợ Direct3D vào trong project.

Tiếp theo, ta cần sao chép những tập tin như sau từ Anim_Sprite folder vào trong folder của project mới tạo.

- winmain.cpp
- dxgraphics.h
- dxgraphics.cpp

Ta có thể thêm cả game.h và game.cpp nếu muốn, nhưng khuyến khích chỉ tạo nó từ bắt đầu bởi vì hầu hết code ở đây sẽ thay đổi khi chuyển sang một project khác. Để thêm chúng, mở thực đơn File, chọn New để mở ra New dialog. Chọn C/C++ Header File cho tệp game.h, và chọn C++ Source File đối với tệp game.cpp.

game.h

Bây giờ ta sẽ tạo lại một project hỗ trợ game framework (thường chỉ bao gồm Window và Direct3D code, nhưng cũng sẽ bao gồm những thành phần DirectX), đây là lúc để viết những code thực tế cho chương trình.

Đây là code cho game.h

```
#ifndef _GAME_H
#endif _GAME_H

#include <d3d9.h>
#include <d3dx9.h>
#include <d3dx9math.h>
#include <time.h>
#include <stdio.h>
#include <stdlib.h>
#include "dxgraphics.h"

// tên của ứng dụng
#define APPTITLE "Trans_Sprite"

// Thiết lập màn hình
#define FULLSCREEN 0 // 0 = windowed, 1 = fullscreen
#define SCREEN_WIDTH 640
#define SCREEN_HEIGHT 480

// Lệnh để đọc keyboard
#define KEY_DOWN(vk_code) ((GetAsyncKeyState(vk_code) & 0x8000) ? 1 : 0)
#define KEY_UP(vk_code) ((GetAsyncKeyState(vk_code) * 0x8000) ? 1 : 0)

// prototypes hàm
int Game_Init(HWND);
void Game_Run(HWND);
void Game_End(HWND);

// structure sprite
typedef struct
{
```

```

    int x, y;
    int widht, height;
    int movex, movey;
    int curframe, lastframe;
    int animdelay, animcount;
} SPRITE;

#endif

```

game.cpp

```

#include "game.h"

LPDIRECT3DTEXTURE9 kitty_image[7];
SPRITE kitty;
LPDIRECT3DSURFACE9 back;
LPD3DXSPRITE sprite_handler;

HRESULT result;

// biến thời gian
long start = GetTickCount();

// Khởi tạo game
int Game_Init(HWND hwnd)
{
    char s[20];
    int n;

    // thiết lập số ngẫu nhiên
    srand(time(NULL));

    // tạo đối tượng sprite handler
    result = D3DXCreateSprie(d3ddev, &sprite_handler);
    if(result != D3D_OK)

```

```

        return 0;

// tải sprite animation
for (n=0; n<6; n++)
{
    // thiết lập tên tệp
    sprintf(s, "cat%d.bmp", n+1);

    // tải texture với màu "hồng" là màu trong suốt
    kitty_image[n] = LoadTexture(s,
D3DCOLOR_XRGB(255,0,255));
    if(kitty_image[n] == NULL)
        return 0;
}

// tải hình nền
back = LoadSurface("background.bmp", NULL);

// Khởi tạo thông tin sprite
kitty.x = 100;
kitty.y = 150;
kitty.width = 96;
kitty.height = 96;
kitty.curframe = 0;
kitty.lastframe = 5;
kitty.animdelay = 2;
kitty.animcount = 0;
kitty.movex = 8;
kitty.movey = 0;

// trả về okay
return 1;
}

void Game_Run(HWND hwnd)
{

```

```

// Đảm bảo Direct3D device đã có
if(d3ddev == NULL)
    return;

// Sau vài giây delay, sẵn sàng frame tiếp theo ?
// điều này giữ cho game chạy với một tốc độ ổn định
if(GetTickCount() - start >= 30)
{
    // Tái lập lại thời gian
    start = GetTickCount();

    // di chuyển sprite
    kitty.x += kitty.movex;
    kitty.y += kitty.movey;

    // Đóng gói sprite vào trong các khung màn hình
    if(kitty.x > SCREEN_WIDTH - kitty.width)
        kitty.x = 0;
    if(kitty.x < 0)
        kitty.x = SCREEN_WIDTH - kitty.width;

    // Có một animation delay
    if(++kitty.animcount > kitty.animdelay)
    {
        // Tái lập bộ đếm
        kitty.animcount = 0;

        // Animation sprite
        if(++kitty.curframe > kitty.lastframe)
            kitty.curframe = 0;
    }
}

// Bắt đầu render
if(d3ddev->BeginScene())
{

```

```

        // Xóa toàn bộ màn hình
        d3ddev->StretchRect(back, NULL, backbuffer, NULL,
D3DTEXF_NONE);

        // Bắt đầu Sprite Handler
        sprite_handler->Begin(D3DXSPRITE_ALPHABLEND);

        // Tạo vector để cập nhật vị trí của sprite
        D3DXVECTOR3 position((float)kitty.x,
(float)kitty.y, 0);

        // vẽ sprite
        sprite_handler->Draw(
            kitty_image[kitty.curframe],
            NULL,
            NULL,
            &position,
            D3DCOLOR_XRGB(255,255,255));

        // Dừng vẽ
        sprite_handler->End();

        // Dừng render
        d3ddev->EndScene();
    }

    // Thể hiện back buffer lên màn hình
    d3ddev->Present(NULL, NULL, NULL, NULL);

    // Kiểm tra phím escape (để kết thúc chương trình)
    if(KEY_DOWN(VK_ESCAPE))
        PostMessage(hwnd, WM_DESTROY, 0, 0);
}

// Giải phóng bộ nhớ và dọn dẹp trước khi kết thúc game
void Game_End(HWND hwnd)

```

```
{
    int n;
    for (n=0; n<6; n++)
        if(kitty_image[n] != NULL)
            kitty_image[n]->Release();

    if(back != NULL)
        back->Release();

    if(sprite_handler != NULL)
        sprite_handler->Release();
}
```

Thay đổi dxgraphics.h

Thêm những dòng dưới này vào trong dxgraphics.h với mô tả hàm như sau:

```
LPDIRECT3DTEXTURE9 LoadTexture(char *, D3DCOLOR);
```

Sau đó thêm những mô tả hàm như thế này:

```
// Mô tả hàm
Int Init_Direct3D(HWND, int, int, int);
LPDIRECT3DSURFACE9 LoadSurface(char *, D3DCOLOR);
LPDIRECT3DTEXTURE9 LoadTexture(char *, D3DCOLOR);
```

Thay đổi dxgraphics.cpp

Bây giờ ta phải định nghĩa hàm LoadTexture để chương trình có thể sử dụng nó, phải mở tập tin *dxgraphics.cpp* và thêm hàm thực sự vào trong tập tin đó.

```
LPDIRECT3DTEXTURE9 LoadTexture(char *filename, D3DCOLOR
transcolor)
```



```

{
    // Con trỏ texture
    LPDIRECT3DTEXTURE9 texture = NULL;

    // Cấu trúc để đọc thông tin tệp bitmap
    D3DXIMAGE_INFO info;

    // Trả về giá trị window thông thường
    HRESULT result;

    // Lấy ra thông tin width và height từ tệp bitmap
    result = D3DXGetImageInfoFromFile(filename, &info);
    if(result != D3D_OK)
        return NULL;

    // tạo texture mới bằng cách tải lên tệp bitmap
    result = D3DXCreateTextureFromFileEx(
        d3ddev,          //Đối tượng Direct3D
        filename,         //tên tệp bitmap
        info.Width,       //Width của tệp bitmap
        info.Height,      //Height của tệp bitmap
        1,               //kết nối level(1 nếu không có thay đổi)
        D3DPPOOL_DEFAULT, //kiểu của surface (thông thường)
        D3DFMT_UNKNOWN,   //định dạng surface (mặc định)
        D3DPPOOL_DEFAULT, //lớp bộ nhớ cho texture
        D3DX_DEFAULT,     //bộ lọc hình ảnh
        D3DX_DEFAULT,     //bộ lọc mip
        transcolor,       //màu chỉ ra trong suốt
        &info,            //thông tin tệp bitmap (từ tệp tải lên)
        NULL,             //đổ màu
        &texture);        //texture đích

    // Đảm bảo texture đã được tải lên thành công
    if(result != D3D_OK)
        return NULL;
}

```

```
    return texture;  
}
```

3.3. Vẽ Tiled Sprite

Cho đến lúc này, ta đã học được về cách tạo, đóng gói, vẽ, xoay và tỷ lệ chỉ bằng một hình bitmap đối với mỗi frame của animation (ít nhất, cho từng sprite được animate). Đây là một cách tốt để học về lập trình sprite, nhưng không phải là cách hiệu quả. Lúc này, game của ta sẽ có hàng trăm tập tin ảnh để tải, sẽ tốn thời gian dài.

Một cách để điều khiển sprite tốt hơn đó là lưu trữ những hình sprite trong một hình tile bitmap. Chúng ta đã được gợi ý về nó trong phần trước, khi ta nói về sprite xe tăng và nhân vật người trong hang, minh họa 3.18.



Hình 3.18. Nhân vật caveman có 8 frames chạy và 4 frames nhảy trong animation.

3.3.1. Hiểu về Tile

Mẹo để có thể hiểu tile là hiểu về một ảnh nguồn được tạo thành từ những dòng và cột tile. Những gì ta muốn là nắm được góc trên bên trái của từng tile nằm trong hình bitmap và sao chép chúng hình chữ nhật nguồn dựa trên width và height của sprite.

Đầu tiên, ta cần chỉ ra được left, hoặc x, vị trí của tile. Ta làm được thế bằng toán tử % (chia lấy dư). Chia lấy dư được phần dư. Ví dụ, frame hiện tại là 20, và chúng ta có năm cột trong bitmap, phép chia dư sẽ đưa ra vị trí ngang của tile (khi ta nhân nó với width của sprite). Tính ra được cạnh trên thì đơn giản bằng cách chia frame hiện tại cho số lượng cột, và nhân với kết quả của sprite height. Nếu có 5 cột, thì tile thứ 20 sẽ nằm ở hàng thứ 4, cột thứ 5. Đây là code minh họa:

```
left = (frame hiện tại % số lượng cột) * sprite width  
top = (frame hiện tại / số lượng cột) * sprite height
```

Code thực sự trong chương trình Tiled_Sprite giống như thế này (chú ý là sử dụng width và height trong tính toán left và top cũng giống như là tính toán với cạnh right và bottom của hình chữ nhật nguồn):

```
left = (frame hiện tại % số cột) * width;  
top = (frame hiện tại / số cột) * height;  
right = left + width;  
bottom = top + height;
```

3.3.2. Chương trình Tiled_Sprite

Đây là mã nguồn cho chương trình Tiled_Sprite giải thích cách để animate một sprite dựa trên một ảnh bitmap đơn. Đầu ra của chương trình **Minh họa 3.19**.



Hình 3.19. Chương trình Tiled_Sprite giải thích cách sử dụng hình tile bitmap cho chuyển động sprite.

```
#include "game.h"

LPDIRECT3DTEXTURE9 caveman_image;
SPRITE caveman;
LPDIRECT3DSURFACE9 back;
LPD3DXSPRITE sprite_handler;

HRESULT result;

// Biến thời gian
long start = GetTickCount();

// Khởi tạo game
int Game_Init(HWND hwnd)
{
    // Đặt lại tiến trình số ngẫu nhiên
    srand(time(NULL));
```

```

// Tạo đối tượng sprite handler
result = D3DXCreateSprite(d3ddev, &sprite_handler);
if(result != D3D_OK)
    return 0;

// Tải texture với màu hồng là màu trong suốt
caveman_image = LoadTexture("caveman.bmp",
D3DCOLOR_XRGB(255, 0, 255));
if(caveman_image == NULL)
    return 0;

// Tải hình nền
back = LoadSurface("background.bmp", NULL);

// Khởi tạo thông số cho sprite
caveman.x = 100;
caveman.y = 180;
caveman.width = 50;
caveman.height = 64;
caveman.curframe = 1;
caveman.lastframe = 11;
caveman.animdelay = 3;
caveman.animcount = 0;
caveman.movex = 5;
caveman.movey = 0;

// Trả về okay
return 1;
}

// Vòng lặp game chính
void Game_Run(HWND hwnd)
{
    // Đảm bảo Direct3D device đã có
    if(d3ddev == NULL)

```

```

return;

// Sau vài giây delay, sẵn sàng frame tiếp theo ?
// điều này giữ cho game chạy với một tốc độ ổn định
if(GetTickCount() - start >= 30)
{
    // Tái lập lại thời gian
    start = GetTickCount();

    // di chuyển sprite
    caveman.x += caveman.movex;
    caveman.y += caveman.movey;

    // Đóng gói sprite vào trong các khung màn hình
    if(caveman.x > SCREEN_WIDTH - caveman.width)
        caveman.x = 0;
    if(caveman.x < 0)
        caveman.x = SCREEN_WIDTH - caveman.width;

    // Có một animation delay
    if(++caveman.animcount > caveman.animdelay)
    {
        // Tái lập bộ đếm
        caveman.animcount = 0;

        // Animation sprite
        if(++caveman.curframe > caveman.lastframe)
            caveman.curframe = 0;
    }
}

// Bắt đầu render
if(d3ddev->BeginScene())
{
    // Xóa toàn bộ màn hình

```

```

        d3ddev->StretchRect(back, NULL, backbuffer, NULL,
D3DTEXF_NONE);

        // Bắt đầu Sprite Handler
        sprite_handler->Begin(D3DXSPRITE_ALPHABLEND);

        // Tạo vector để cập nhật vị trí của sprite
        D3DXVECTOR3 position((float)caveman.x,
(float)caveman.y, 0);

        // Thiết đặt kích thước cho từng tile nguồn
        RECT srcRect;
        int columns = 8;
        srcRect.left = (caveman.curframe % columns) *
caveman.width;
        srcRect.top = (caveman.curframe / columns) *
caveman.height;
        srcRect.right = srcRect.left + caveman.width;
        srcRect.bottom = srcRect.top + caveman.height;

        // vẽ sprite
        sprite_handler->Draw(
            caveman_image,
            &srcRect,
            NULL,
            &position,
            D3DCOLOR_XRGB(255,255,255));

        // Dừng vẽ
        sprite_handler->End();

        // Dừng render
        d3ddev->EndScene();
    }

    // Thẻ hiện back buffer lên màn hình

```

```
d3ddev->Present(NULL, NULL, NULL, NULL);

// Kiểm tra phím escape (để kết thúc chương trình)
if(KEY_DOWN(VK_ESCAPE))
    PostMessage(hwnd, WM_DESTROY, 0, 0);
}

// Giải phóng bộ nhớ và dọn dẹp trước khi kết thúc game
void Game_End(HWND hwnd)
{
    if(caveman_image != NULL)
        caveman_image->Release();

    if(back != NULL)
        back->Release();

    if(sprite_handler != NULL)
        sprite_handler->Release();
}
```


CHƯƠNG 4

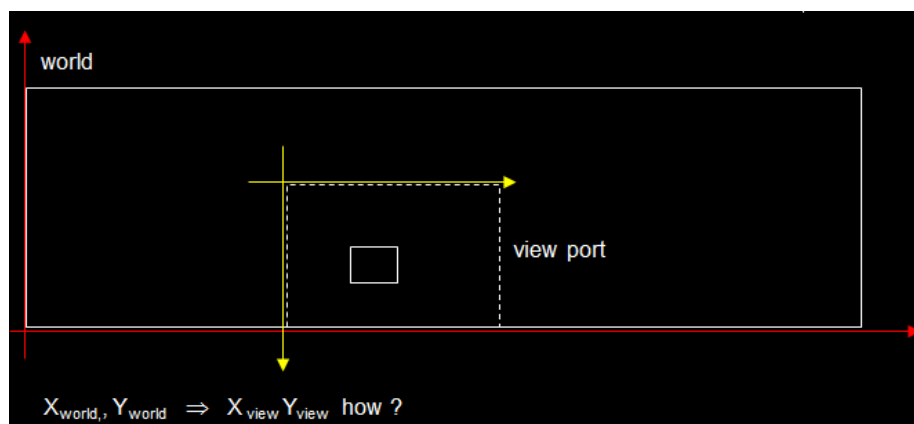
CÁC PHÉP BIẾN ĐỔI

Các đối tượng trong game có 2 loại tọa độ mà chúng ta cần lưu ý: tọa độ World (đây là tọa độ của thế giới game và nó là tọa độ thực) và tọa độ của View Port (View port là một khung nhìn trong game và nó cũng có một tọa độ trong thế giới game).

Nếu tọa độ View port nằm ở vị trí nào đó ngoài vùng của thế giới game thì chuyện gì sẽ xảy ra. Tất cả các đối tượng của game sẽ không được thể hiện lên màn hình. Chính vì vậy, chúng ta sẽ tìm cách sao để chuyển tọa độ World sang tọa độ View Port để có thể nhìn thấy các đối tượng trong game. Nhưng việc chuyển tọa độ World sang tọa độ View Port thì hết sức là khó khăn. Nên bài hôm nay chúng ta sẽ đi tìm một giải pháp mới, chúng ta sẽ tìm cách chuyển tọa độ View Port về tọa độ World.

Vấn đề trong game của chúng ta là giữa tọa độ thực và tọa độ trong game có sự khác biệt về hướng của trục Y. Tọa độ thực tế thì trục Y hướng lên còn trong game thì trục Y hướng xuống.

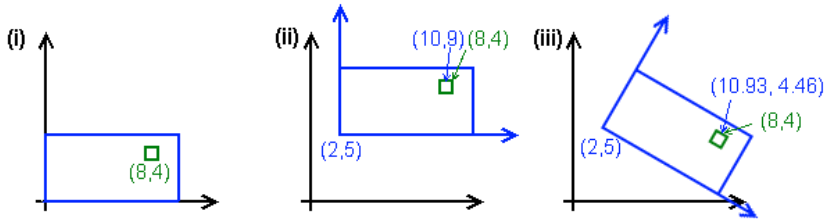
Bài này sẽ giải đáp được những thắc mắc trên. Chúng ta sẽ sử dụng một kỹ thuật để xử lý đó là Transform.



Hình 4.1. Tọa độ view port trong game và thế giới thực

4.1. Biến đổi tọa độ (Transform) là gì?

Biến đổi tọa độ là một hành động dùng để chuyển đổi các giá trị tọa độ của các đỉnh của một hình thành một tập hợp các giá trị tọa độ khác và hình gốc ban đầu có thể có kích thước khác, ở một vị trí khác và một góc xoay khác so với tọa độ gốc ban đầu.



Hình 4.2. Transform đối tượng

Vì sao chúng ta lại cần phải hiểu Transform. Vậy Transform sẽ giúp gì trong vấn đề làm game của ta.

Có khá nhiều kỹ thuật transform bằng ma trận như : Xoay (Rotate), Lật tọa độ(Flip) và Translate(Tịnh tiến). Tuy nhiên, trong chương này, chúng ta chỉ tìm hiểu Flip và Translate để phục vụ cho mục tiêu của giáo trình.

4.1.1. Flip

Flip là lật tọa độ. Trong không gian 2D thì ta có 2 chiều lật: lật tọa độ theo trục X và lật tọa độ theo trục Y. Khi bạn lật theo trục **X** thì tọa độ **X₀** của một Điểm vẫn giữ nguyên nhưng tọa độ **Y₀** thì bị đảo dấu. Còn lật theo trục Y thì ngược lại.

Để thực hiện phép lật thì các nhà toán học đã tính toán sẵn cho chúng ta. Bằng cách nhân tọa độ của điểm cần lật với một ma trận bổ sung. Tùy theo ý định của người sử dụng cần lật theo trục X hay Y mà ta sử dụng ma trận bổ sung thích hợp.

Để lật theo chiều X thì chúng ta sẽ lấy tọa độ điểm mà chúng ta cần lật tọa độ nhân với ma trận

$$\begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$$

chúng ta sẽ có một có một điểm mới có toạ độ $\mathbf{X_0}$ giữ nguyên và toạ độ $\mathbf{Y_0}$ thì mang giá trị trái dấu với toạ độ $\mathbf{Y_0}$ ban đầu.

Để lật theo chiều Y thì chúng ta sẽ lấy toạ độ điểm mà chúng ta cần lật toạ độ nhân với ma trận

$$\begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix}$$

chúng ta sẽ có một có một điểm mới có toạ độ $\mathbf{Y_0}$ giữ nguyên và toạ độ $\mathbf{X_0}$ thì mang giá trị trái dấu với toạ độ $\mathbf{X_0}$ ban đầu.

Sau đây ta sẽ hiểu rõ hơn qua hình dưới đây:

A					A'	
<div style="border: 1px solid black; padding: 5px; display: inline-block;"> <div style="display: flex; justify-content: space-between; padding: 0 10px;">510</div> </div>	×	<div style="border: 1px solid black; padding: 5px; display: inline-block;"> <div style="display: flex; justify-content: space-between; padding: 0 10px;">10</div> <div style="display: flex; justify-content: space-between; padding: 0 10px;">0-1</div> </div>	=	<div style="border: 1px solid black; padding: 5px; display: inline-block;"> <div style="display: flex; justify-content: space-between; padding: 0 10px;">5-10</div> </div>	Flip Y	
B					B'	
<div style="border: 1px solid black; padding: 5px; display: inline-block;"> <div style="display: flex; justify-content: space-between; padding: 0 10px;">510</div> </div>	×	<div style="border: 1px solid black; padding: 5px; display: inline-block;"> <div style="display: flex; justify-content: space-between; padding: 0 10px;">-10</div> <div style="display: flex; justify-content: space-between; padding: 0 10px;">01</div> </div>	=	<div style="border: 1px solid black; padding: 5px; display: inline-block;"> <div style="display: flex; justify-content: space-between; padding: 0 10px;">-510</div> </div>	Flip X	

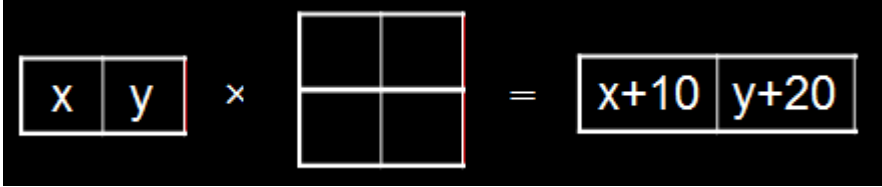
Hình 4.3. Lật toạ độ

4.1.2. Tịnh tiến

Tịnh tiến: chuyển động dọc theo các trục mà không làm thay đổi hướng.

Để làm được điều này, ta phải nhân hai ma trận. Nhân hai ma trận như thế nào để ta có được phép tịnh tiến thích hợp?

Phép tịnh tiến không thể nào được định nghĩa bằng việc nhân với ma trận cấp 2.


$$\begin{bmatrix} x & y \end{bmatrix} \times \begin{bmatrix} 10 & 20 \\ 0 & 0 \end{bmatrix} = \begin{bmatrix} x+10 & y+20 \end{bmatrix}$$

Bạn hãy thử điền giá trị vào ma trận cấp 2 trên để ra được kết quả là $x + 10$ và $y + 10$.

Tất nhiên là ta sẽ không tìm ra được một ma trận nào thích hợp. Cho nên người ta đưa ra một giải pháp là thêm vào các ô giả trong ma trận.

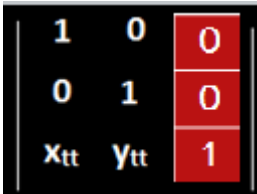
Cách thức thực hiện như sau:

- Tọa độ của điểm mà bạn đang cần tịnh tiến phải chuyển sang tọa độ 3D

VD: Điểm $A(x, y)$ là điểm cần tịnh tiến trong không gian 2D. Và để thực hiện được phép tịnh tiến thì Điểm A phải chuyển thành $A(x, y, 1)$.


$$\begin{bmatrix} x & y \end{bmatrix} \Rightarrow \begin{bmatrix} x & y & 1 \end{bmatrix}$$

- Tiến hành nhân với ma trận cấp 3 có cấu trúc như sau


$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ x_{tt} & y_{tt} & 1 \end{bmatrix}$$

x_{tt} : là khoảng cách chúng ta cần tịnh tiến so với vị trí gốc theo trục X

Y_{tt} : là khoảng cách chúng ta cần tịnh tiến so với vị trí gốc theo trục Y

Hình 4.4. Biến đổi cấu trúc ma trận

Các ma trận có thể nhân với nhau để tạo nên 1 loạt các biến đổi. Trong việc nhân 2 ma trận các bạn luôn để ý tới thứ tự của các ma trận, vì cùng 2 ma trận nhưng đặt với thứ tự khác nhau mà nhân lại thì nó sẽ cho ra kết quả là các ma trận khác nhau.

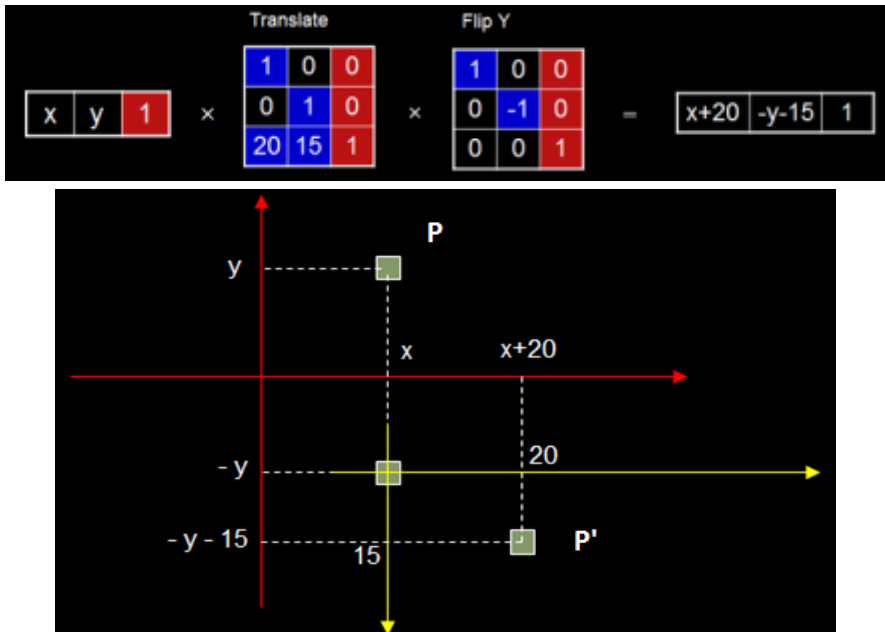
Chính vì việc xác định phải nhân ma trận theo thứ tự như thế nào để có được vị trí mà ta cần tịnh tiến. Thì các nhà toán học đã tính toán và cho ta một ma trận thích hợp cho việc chúng ta tịnh tiến.

Hình dưới đây mô tả lại phép biến đổi cả tịnh tiến và lật theo trục Y. Lưu ý ma trận Flip dưới đây là chúng ta đang dùng Flip Y. Để tịnh tiến $P(x, y) \rightarrow P'$. Thì trong đầu chúng ta sẽ suy nghĩ là đầu tiên chúng ta sẽ lật nó trước, sau đó tiến hành transform. Và chúng ta áp dụng cái thứ tự đó vào việc nhân ma trận thì chúng ta sẽ có thứ tự như sau:

Hình 4.5. Mô tả phép biến đổi tịnh tiến và lật theo trục Y

Và kết quả đã bị sai vì theo như cách chúng ta suy nghĩ thì $P'(x+20, -y-15, 1)$, nhưng đây lại cho ra kết quả là $P'(x+20, -y+15)$.

Chúng tôi là thứ tự nhân ma trận của chúng ta đã bị sai. Thứ tự nhân ma trận đúng phải là:



Hình 4.6. Nhân ma trận để thực hiện các biến đổi

Chính vì thấy được sự khó khăn trong việc nhân ma trận theo thứ tự nào thì cho phù hợp để chúng ta có thể Transform đúng, thì các nhà nghiên cứu đã tính sẵn cho chúng ta một ma trận trung gian dùng cho việc Transform bằng cách nhân Ma trận Translate với Ma trận Flip theo thứ tự sau:

1	0	0
0	1	0
20	15	1

×

1	0	0
0	-1	0
0	0	1

=

1	0	0
0	-1	0
20	-15	1

Chúng ta đã có được ma trận trung gian sau:

1	0	0
0	-1	0
20	-15	1

Gồm:

- | | |
|---|----|
| 1 | 0 |
| 0 | -1 |

 Dùng để Flip theo trục Y
- | | |
|----|-----|
| 20 | -15 |
|----|-----|

 Khoảng cách từ vị trí gốc (sau khi đã thực hiện Flip phía trên) đến vị trí mới theo toạ độ x và y. Nghĩa là toạ độ mới sẽ là $(x + 20, y - 15)$.

Giờ chúng ta chỉ việc nhân P với Ma trận trung gian này để có được kết quả mà chúng ta mong muốn.

Flip Y then Translate

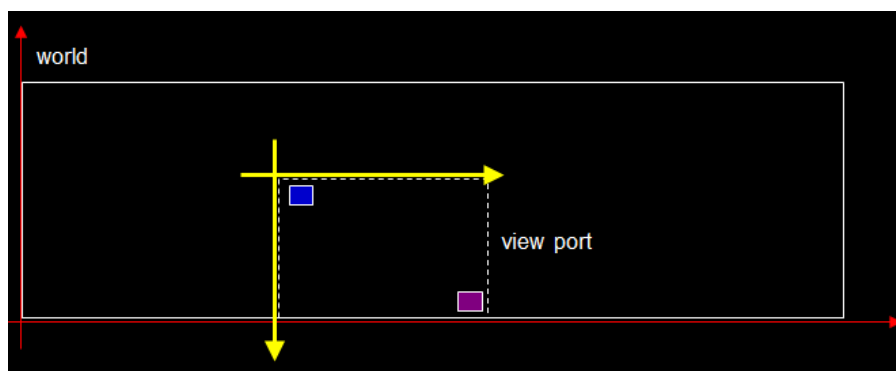
$$\begin{array}{|c|c|c|} \hline P & & \\ \hline x & y & 1 \\ \hline \end{array} \times \begin{array}{|c|c|c|} \hline 1 & 0 & 0 \\ \hline 0 & -1 & 0 \\ \hline 20 & -15 & 1 \\ \hline \end{array} = \begin{array}{|c|c|c|} \hline x+20 & -y-15 & 1 \\ \hline \end{array}$$

*Lưu ý: Đây là một ví dụ giúp bạn đọc hiểu được thứ tự mà chúng ta thực hiện nhân ma trận nó quan trọng như thế nào. Các bạn cũng có thể kết hợp nhiều ma trận khác tùy theo ý định của bạn muốn làm gì. Ở đây chúng tôi chỉ giới thiệu cho bạn một cách để Transform bằng cách sử dụng các ma trận ở trên. Còn tất cả các ma trận cần thiết cho việc làm Game thì các nhà phát triển Game đã hiện thực hoá thành các Phương thức cụ thể rồi.

4.2. Biến đổi từ hệ tọa độ thế giới (World) sang View Port trong Game

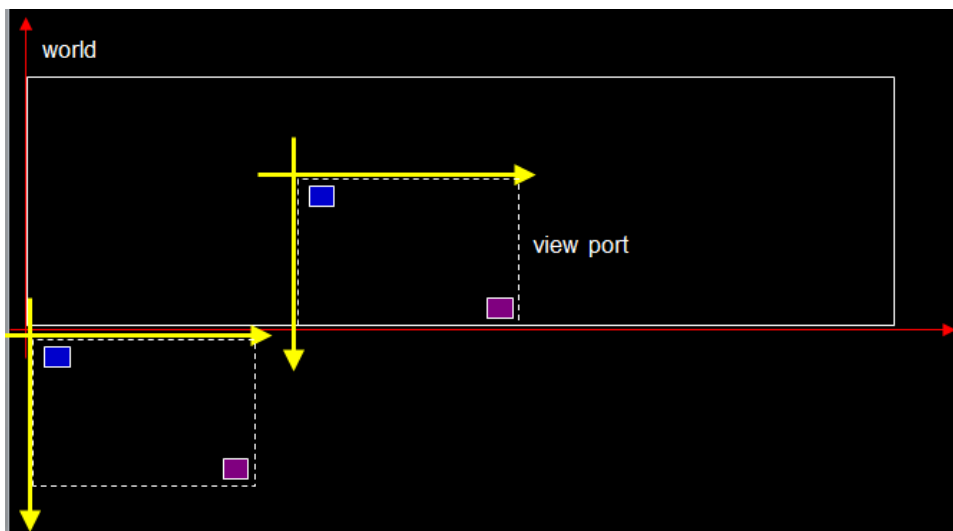
Trong game, cụ thể là game Mario. Ta có thể giới game Mario là một thế giới rộng lớn và có tọa độ theo hệ tọa độ Đề-Các (tọa độ này có hướng của trục X và Y giống với hướng của trục tọa độ trong không gian thật).

View Port là một khung nhìn, và nó chỉ thể hiện được một phần của thế giới game của chúng ta. Những đối tượng nào nằm trong View Port thì chúng ta mới có thể nhìn thấy chúng.



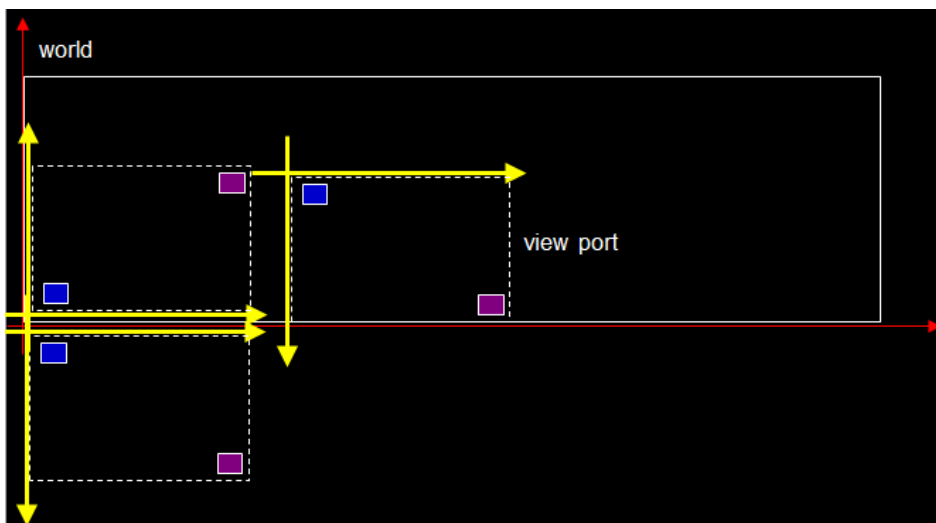
Hình 4.7. View port và World

Như ý định bạn đầu là chúng ta sẽ Transform View Port để tọa độ của View Port sẽ có trục Y hướng lên giống như tọa độ World.



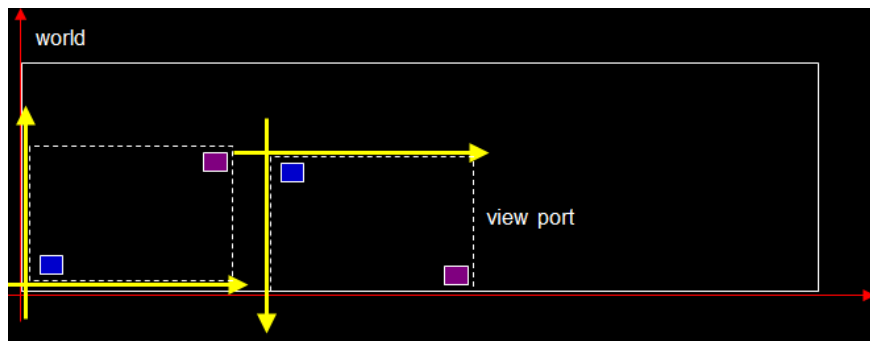
Hình 4.8. Minh họa tọa độ View Port

Đầu tiên chúng ta sẽ translate tọa độ gốc của View Port trùng với tọa độ gốc của Thế giới(World)



Hình 4.9. Minh họa tọa độ gốc của thế giới (World)

Sau đó ta tiến hành lật(Flip) View Port lên thì chúng ta sẽ có được kết quả là toạ độ của View Port trùng với toạ độ Thế giới(World) và trục Y của View Port đã hướng lên như Hình 7.



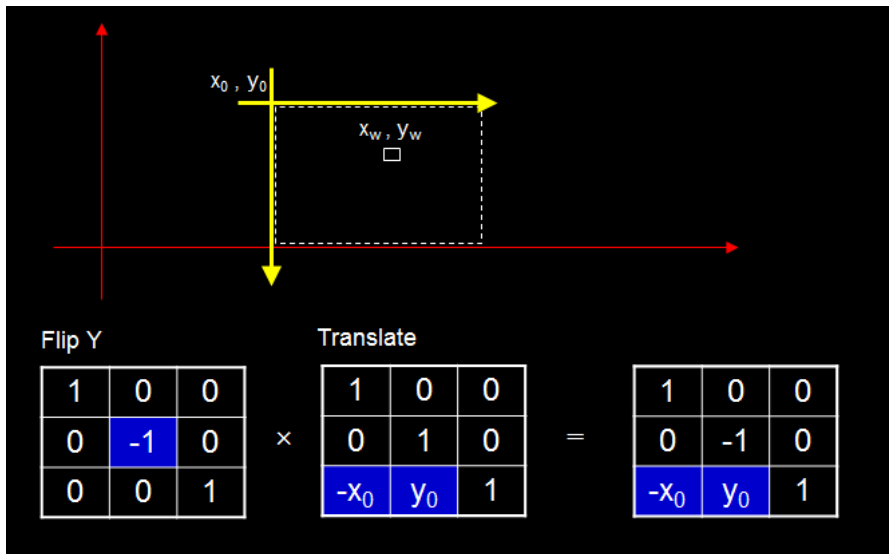
Hình 4.10. Kết quả transform từ world sang view port

Nhận xét: Hình bị lật ngược sau khi biến đổi. Chỉ có duy nhất một điểm không bị lật ngược chính là tâm của hình chữ nhật.

4.1.3. Xác định toạ độ của Object trong View Port

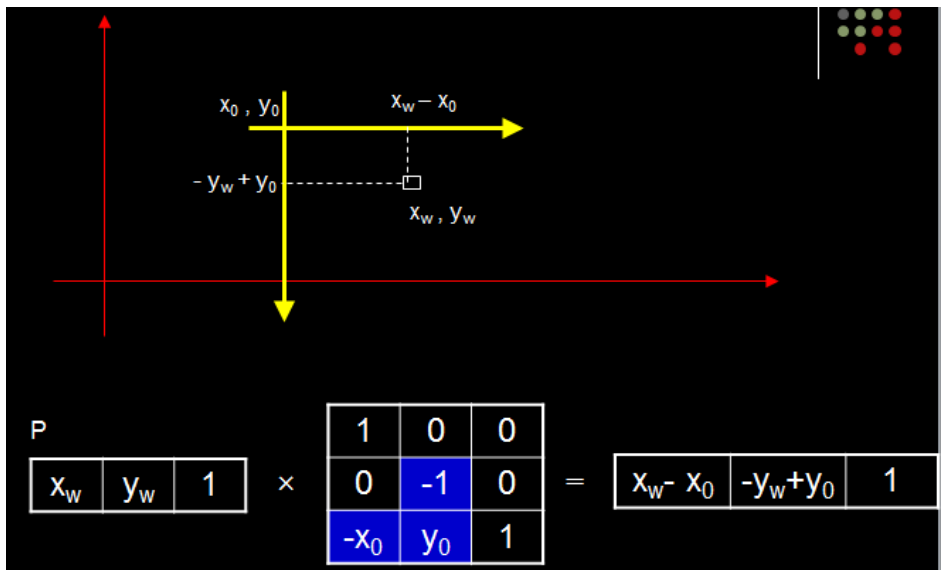
Đầu tiên ta xác định ma trận chuyển đổi như sau:

Lấy ma trận FlipY nhân với ma trận Translate ta được ma trận trung gian hình bên dưới



Hình 4.11. Minh họa phép biến đổi sử dụng ma trận trung gian

Sau đó ta lấy P nhân với Ma trận trung gian để tìm ra được vị trí của P trong View Port



Hình 4.12. Minh họa vị trí của P trong ViewPort

Có một điều quan trọng là trong DirectX thì ma trận mà các nhà phát triển game sử dụng đó là ma trận 4x4, bởi vì việc transform không chỉ để thực hiện trong môi trường Game 2D mà nó còn được thực hiện trong môi trường Game 3D. Giống như cách mà chúng ta thực hiện trong môi trường 2D. Để Transform thì tất cả ma trận cấp 2 đều chuyển về ma trận cấp 3 bằng cách thêm các ô giá trị giả cho ma trận. Tương tự thế trong môi trường 3D, họ cũng đã thực hiện như thế nên ta sẽ có được ma trận 4x4. Và họ áp dụng ma trận 4x4 này cho cả 2D và 3D.

Từ đầu bài tới giờ là chúng ta đang dẫn dắt mọi việc để làm sao có được ma trận cuối cùng như thế này.

1	0	0	0
0	-1	0	0
0	0	1	0
$-x_0$	y_0	0	1

4.1.4. Code biến đổi từ world sang view port

Toàn bộ “phép màu” cho phép ta vẽ một thế giới game rộng lớn bên trong một view port của chương nằm trong đoạn code “màu nhiệm” ngắn gọn dưới đây.

```
D3DXMATRIX mt;  
D3DXMatrixIdentity(&mt);  
mt._22 = -1.0f;  
mt._41 = -cameraX;  
mt._42 = 600;
```

```
D3DXVECTOR4 vp_pos;
D3DXVec3Transform(&vp_pos, &position, &mt);

D3DXVECTOR3 pos(vp_pos.x, vp_pos.y, 0);
D3DXVECTOR3 center(
    (float)p_texture->getWidth()/2, (float)p_texture-
>getHeight()/2, 0);

this->spriteHandler->Draw(
    p_texture->getTexture(),
    &srect,
    &center,
    &pos,
    D3DCOLOR_XRGB(255, 255, 255)
);
```

CHƯƠNG 5

LẬP TRÌNH CHUỘT VÀ BÀN PHÍM

5.1. Lập trình bàn phím

Bàn phím là thiết bị nhập thông thường cho tất cả các game, kể cả những game không sử dụng bàn phím, vì vậy nó cho phép game của ta có thể sử dụng bàn phím theo một hoặc nhiều cách. Nếu không có gì khác, ta nên cho phép người chơi thoát game hoặc ít nhất là hiện ra thực đơn của game bằng cách nhấn nút Escape (thông thường). Lập trình bàn phím sử dụng DirectInput không khó, nhưng ta cần khởi tạo DirectInput trước tiên.

Đối tượng DirectInput chính được gọi là IDirectInput8; ta có thể truy xuất trực tiếp hoặc sử dụng con trỏ kiểu LPDIRECTINPUT8. Tại sao lại là số 8 ? Bởi vì giống với DirectSound, DirectInput đã không thay đổi kể từ phiên bản cuối DirectX 8.1. Đó là phiên bản được mang theo trong bản nâng cấp 9.0b.

Thư viện DirectInput được đặt là *dinput8.lib*, vì vậy cần đảm bảo là đã thêm tập tin này vào trong tùy chọn linker trong *Project Setting* dialog giống như những thư viện khác. Nếu chưa hiểu, ta nên ôn lại chương trước để biết được cách thiết lập một project hỗ trợ DirectX và game framework đã xây dựng cho tới thời điểm này. Nếu có bất kì câu hỏi nào về thiết lập một project tại thời điểm này, hãy đọc lại toàn bộ những chương trước (phần tổng quan và hướng dẫn). Trong chương trình, ta sẽ phải thêm thành phần mới vào trong framework với DirectInput với 2 tập tin (*dxinput.h* và *dxinput.cpp*).

5.1.1. Đối tượng DirectInput và DirectInput Device

Ta đã quen với việc khởi tạo một thành phần DirectX; hãy học về làm cách nào để quét bàn phím về nhấn nút. Đầu tiên, ta phải định nghĩa được đối tượng DirectInput chính sử dụng trong toàn bộ chương trình song song với kiểu device của nó:

```
LPDIRECTINPUT8 dinput;  
LPDIRECTINPUTDEVICE8 dinputdev;
```

Sau khi định nghĩa những biến trên, ta có thể gọi `DirectInputCreate8` để khởi tạo `DirectInput`. Hàm này có định dạng như sau:

```
HRESULT WINAPI DirectInput8Create(  
    HINSTANCE hinst,  
    DWORD dwVersion,  
    REFIID riifltd,  
    LPVOID *ppvOut,  
    LPUNKNOWN punkOuter  
);
```

Hàm này chỉ tạo ra đối tượng `DirectInput` chính mà ta truyền vào. Tham số đầu tiên là thể hiện cho chương trình hiện tại. Một cách tiện lợi hơn để lấy được thể hiện hiện tại khi nó không có ngay (thường thì nó không chỉ thấy trong `WinMain`) đó là sử dụng hàm `GetModuleHandle`. Tham số thứ hai là phiên bản `DirectInput`, luôn luôn được truyền là `DIRECTINPUT_VERSION`, được định nghĩa trong `dxinput.h`. Tham số thứ ba là định danh về phiên bản `DirectInput` mà ta muốn sử dụng. Hiện tại, giá trị này là `IID_IDirectInput8`. Tham số thứ tư là con trỏ đến con trỏ đối tượng `DirectInput` chính (chú ý: con trỏ kép), và tham số thứ năm luôn luôn là `NULL`. Đây là ví dụ về gọi hàm:

```
HRESULT result =  
DirectInput8Create(  
    GetModuleHandle(NULL),  
    DIRECTINPUT_VERSION,  
    IID_IDirectInput8,  
    (void**) &dinput,  
    NULL);
```

Sau khi khởi tạo đối tượng, ta có thể dùng đối tượng để tạo thiết bị `DirectInput` bằng cách gọi `CreateDevice`:

```
HRESULT CreateDevice(  
    REFGUID rguid,  
    LPDIRECTINPUTDEVICE *pIpdirectInputDevice,  
    LPUNKNOWN punkOuter  
) ;
```

Tham số đầu tiên là giá trị đặc tả cho kiểu của đối tượng mà bạn muốn tạo (như là bàn phím hay chuột). Đây là giá trị có thể sử dụng cho tham số này:

```
GUID_SysKeyboard  
GUID_SysMouse
```

Tham số thứ hai là con trỏ đến đối tượng device nhận địa chỉ của người quản lý DirectInput Device. Tham số thứ ba luôn luôn bằng NULL. Đây là cách ta sẽ gọi hàm trên:

```
result = dinput->CreateDevice(GUID_SysKeyboard,  
&dikeyboard, NULL);
```

5.1.2. Khởi tạo đối tượng bàn phím

Ta đã có đối tượng DirectInput và đối tượng device cho bàn phím, đã có thể khởi tạo trình điều khiển bàn phím để chuẩn bị cho input. Bước tiếp theo là thiết lập định dạng dữ liệu cho bàn phím, chỉ dẫn cho DirectInput cách để truyền dữ liệu vào trong chương trình. Đây là một cách chính xác vì có hàng trăm thiết bị input khác nhau trên thị trường với nhiều tính năng khác nhau, vì vậy không thể đọc hết chúng được.

5.1.3. Thiết lập Định dạng Dữ liệu

SetDataFormat đặc tả cho định dạng dữ liệu được thiết lập.

```
HRESULT SetDataFormat (LPCDIDATAFORMAT lpdf);
```

Tham số duy nhất của hàm này đặc tả cho kiểu thiết bị. Đối với bàn phím, bạn sẽ truyền vào giá trị `c_dfDIKeyboard`. Hằng số như thế này cho chuột là `c_dfDIMouse`. Đây là ví dụ về gọi hàm:

```
HRESULT result =  
    dikeyboard-> SetDataFormat( &c_dfDIKeyboard);
```

Chú ý rằng ta không cần phải định nghĩa `c_dfDIKeyboard`, vì nó đã được định nghĩa trong *diinput.h*.

5.1.4. Thiết lập mức độ hợp tác

Bước tiếp theo là thiết lập mức độ hợp tác (*Cooperative Level*), chỉ ra được mức độ bàn phím DirectInput sẽ đưa vào trong chương trình theo độ ưu tiên. Để thiết lập độ hợp tác, bạn gọi hàm `SetCooperativeLevel`:

```
HRESULT SetCooperativeLevel(  
    HWND hwnd,  
    DWORD dwFlags  
);
```

Tham số đầu tiên là handle của cửa sổ. Tham số thứ hai khá thú vị vì nó xác định độ *ưu tiên* của bàn phím và chuột đối với chương trình của chúng ta. Giá trị chung thường được truyền khi làm việc với bàn phím là `DISCL_NONEXCLUSIVE` và `DISCL_FOREGROUND`. Đây là cách gọi hàm:

```
HRESULT result = dikeyboard->SetCooperativeLevel(hwnd,  
    DISCL_NONEXCLUSIVE | DISCL_FOREGROUND );
```

5.1.5. Giành kiểm soát thiết bị

Bước cuối cùng là khởi tạo bàn phím để giành bàn phím sử dụng hàm Acquire:

```
HRESULT Acquire(void);
```

Nếu hàm trả về giá trị hợp lệ (DI_OK) thì ta đã thành công trong việc giành lấy bàn phím để có thể bắt đầu kiểm tra nút được nhấn.

Một điểm quan trọng ta nên chỉ ra đó là ta phải trả lại keyboard trước khi kết thúc hoặc khi rời khỏi DirectInput và hoặc khi trình quản lý bàn phím không rõ trạng thái. Windows và DirectInput sẽ làm việc dọn dẹp sau cho ta nhưng nó còn phải dựa trên phiên bản Windows mà ta đang sử dụng. Tin hay không, khi vẫn còn những phiên bản Windows 98 và ME đang chạy, dù cho chúng đã quá hạn. Windows 2000 vẫn còn có một chút lợi ích, vì XP và 2003, nhưng ta không nên bỏ qua những cơ hội. Cách tốt nhất là trả thiết bị lại trước khi kết thúc game. Mỗi thiết bị DirectInput có đều có hàm Unacquire với định dạng như sau:

```
HRESULT Unacquire(VOID);
```

5.1.6. Đọc Phím nhấn

Ở đâu đó trong vòng lặp game, ta cần đặt bàn phím cho phép cập nhật giá trị của phím. Đặt giá trị của phím, tốt hơn là ta dùng một mảng phím chỉ ra được trạng thái của thiết bị bàn phím, giống như:

```
char keys[256];
```

Ta phải đặt ra mảng các phím để có thể đặt giá trị, sau đó gọi hàm GetDeviceState. Hàm này được gọi cho tất cả các thiết bị, không phân biệt kiểu:

```
HRESULT GetDeviceState(
```

```
DWORD cbData,  
LPVOID lpvData  
);
```

Tham số đầu tiên là kích thước của bộ đệm trạng thái thiết bị cần lắp dữ liệu. Tham số thứ hai là con trỏ cho dữ liệu. Trong trường hợp này là bàn phím, đây cách gọi hàm:

```
dikeyboard->GetDeviceState(sizeof(keys), (LPVOID)&keys);
```

Sau khi đã đặt bàn phím, ta có thể kiểm tra trong mảng giá trị các phím sao cho phù hợp với mã phím trong DirectInput

Đây là cách kiểm tra nhấn phím ESCAPE:

```
if(keys[DIK_ESCAPE] & 0x80)  
{  
    // phím ESCAPE đã được nhấn, làm ....  
}
```

5.2. Lập trình chuột

Ta đã viết chương trình thao tác trên bàn phím bằng DirectInput, tương tác với chuột cũng giống vậy - code của chúng rất giống nhau, và nó cùng chia sẻ đối tượng DirectInput và con trỏ thiết bị. Vì vậy, chúng ta sẽ cùng nhau nhanh qua phần này, và học các giao diện của chuột. Đầu tiên, định nghĩa thiết bị chuột:

```
LPDIRECTINPUTDEVICE8 dimouse;
```

Tiếp theo, tạo thiết bị chuột:

```
result =  
dinput->CreateDevice(GUID_SysMouse, &dimouse, NULL);
```

5.2.1. Khởi tạo chuột

Giả sử như ta đã tạo DirectInput và bây giờ muốn thêm vào quản lý chuột. Bước tiếp theo là đặt ra kiểu dữ liệu cho chuột, chỉ ra cách để DirectInput truyền dữ liệu vào trong chương trình của mình. Hàm này chính xác giống với hàm đã thực hiện trên bàn phím.

5.2.2. Thiết lập Định dạng Dữ liệu

SetDataFormat đặc tả cho định dạng dữ liệu được thiết lập.

```
HRESULT SetDataFormat (
    LPCDIDATAFORMAT lpdf
);
```

Tham số duy nhất của hàm này đặc tả cho kiểu thiết bị. Hằng số như thế này cho chuột là `c_dfDIMouse`. Đây là ví dụ về gọi hàm:

```
HRESULT result =
    dimouse->SetDataFormat(&c_dfDIMouse);
```

Chú ý thêm lần nữa rằng bạn không cần phải định nghĩa `c_dfDIMouse`, vì nó đã được định nghĩa trong `diinput.h`.

5.2.3. Thiết lập Mức độ Hợp tác

Bước tiếp theo là thiết lập mức độ hợp tác, chỉ ra được mức độ bàn phím DirectInput sẽ đưa vào trong chương trình theo độ ưu tiên. Để thiết lập độ hợp tác, bạn gọi hàm `SetCooperativeLevel`:

```
HRESULT SetCooperativeLevel(
    HWND hwnd,
    DWORD dwFlags
);
```

Tham số đầu tiên là quản lý cửa sổ. Tham số thứ hai là một điều khá thú vị, vì nó đặc tả cho độ ưu tiên của chương trình của ta về Bàn phím và Chuột. Giá trị chung thường được truyền khi làm việc với chuột / bàn phím là DISCL_NONEXCLUSIVE và DISCL_FOREGROUND. Đây là cách gọi hàm:

```
HRESULT result = dimouse->SetCooperativeLevel(hwnd,  
DISCL_NONEXCLUSIVE | DISCL_FOREGROUND);
```

5.2.4. Giành kiểm soát thiết bị

Bước cuối cùng là giành lấy thiết bị chuột sử dụng hàm Acquire. Nếu hàm trả về DI_OK thì ta đã giành thành công chuột và đã sẵn sàng để bắt đầu kiểm tra di chuyển và nhấn chuột.

Giống như với thiết bị bàn phím, ta cũng cần phải trả lại thiết bị chuột sau khi bạn đã hết sử dụng nó, hoặc khi bạn rời khỏi sử dụng DirectInput

```
HRESULT Unacquire(VOID);
```

5.2.5. Đọc chuột

Đâu đó trong vòng lặp game của ta cần phải đặt bộ cập nhật vị trí và trạng thái nút của chuột. Ta đặt điều này bằng cách sử dụng hàm GetDeviceState:

```
HRESULT GetDeviceState(  
    DWORD cbData,  
    LPVOID lpvData  
) ;
```


Tham số đầu tiên là kích thước của bộ đệm trạng thái thiết bị cần lấp dữ liệu. Tham số thứ hai là con trỏ cho dữ liệu. Đây là cấu trúc có sẵn để ta có thể gọi chuột:

```
DIMOUSESTATE mouse_state;
```

Đây là cách để lấp dữ liệu vào DIMOUSESTATE bằng cách gọi hàm `GetDeviceState`:

```
Dimouse->GetDeviceState( (sizeof(mouse_state),  
(LPVOID) &mouse_state);
```

Cấu trúc sẽ như thế này:

```
typedef struct DIMOUSESTATE {  
    LONG lX;  
    LONG lY;  
    LONG lZ;  
    BYTE rgbButton[4];  
} DIMOUSESTATE;
```

Đây là cấu trúc thay thế khi ta muốn sử dụng chuột phức tạp hơn với nhiều hơn là 4 nút, trong trường hợp này bạn sẽ có mảng với kích thước gấp đôi:

```
typedef struct DIMOUSESTATE2 {  
    LONG lX;  
    LONG lY;  
    LONG lZ;  
    BYTE rgbButton[8];  
} DIMOUSESTATE;
```

Sau khi đã đặt chuột, ta có thể kiểm tra `mouse_state` về vị trí `x` và `y` và nút nhấn. Ta có thể kiểm tra chuột về sự di chuyển, hay còn gọi là *mickeys*, sử dụng `lx` và `ly`. Mickeys là gì ? Mickeys biểu diễn cho sự di chuyển của chuột thay thế cho vị trí tuyệt đối, vì vậy, ta có thể giữ lại vị trí cũ nếu bạn muốn dùng nó cho vẽ con trỏ của mình. Mickeys là một cách tiện lợi để có thể điều khiển sự di chuyển của chuột bởi vì ta có tiếp tục di chuyển theo một hướng đơn và chuột sẽ báo cáo lại mức độ di chuyển, dù cho con trỏ đã đi đến cạnh của màn hình.

Như ta có thể xem cấu trúc, mảng `rgbButtons` nắm giữ kết quả của những nút đã nhấn. Nếu bạn muốn kiểm tra chỉ cho một nút đặc biệt nào nó (bắt đầu từ 0 cho button 1) đây là cách ta làm:

```
button_1 = obj.rgbButtons[0] & 0x80;
```

Một cách tiện lợi hơn đó là phát hiện nút được nhấn sử dụng định nghĩa:

```
#define BUTTON_DOWN(obj, button)  
    (obj.rgbButtons[button] & 0x80)
```

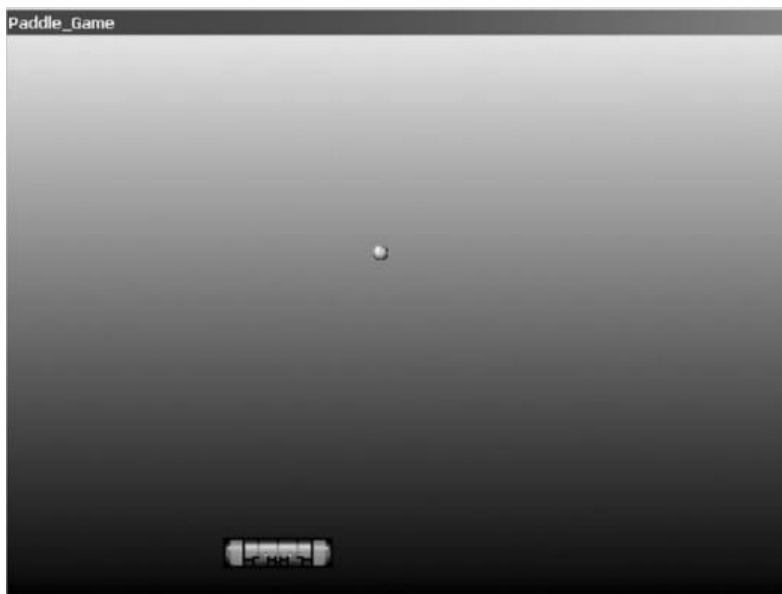
Bằng cách sử dụng định nghĩa, ta có thể kiểm tra nút được nhấn giống:

```
button_1 = BUTTON_DOWN(mouse_state, 0);
```

5.3. Ví dụ minh họa Paddle Game

Đây là sự tổng hợp của bàn phím và chuột. Ta đã sẵn sàng để tập luyện với chương trình đơn giản? Sau khi thêm `DirectInput` vào trong game framework, chúng ta sẽ xây dựng một game nhỏ gọi là Paddle Game giống với những game cơ bản khác là Breakout hay Arkanoid mà ta có thể thay đổi hay chỉnh sửa theo ý mình. Hình minh họa dưới đây thể hiện game Paddle đang chạy. Game này hỗ trợ cả bàn phím và chuột.

Sử dụng hàm Collision đơn giản, ta sẽ có thể thêm nó vào để quả bóng có thể đánh vào thanh bar.



Hình 5.1. Paddle Game là một game gần hoàn chỉnh minh họa cho việc sử dụng DirectInput để đọc bàn phím và chuột.

5.3.1. Framework code mới cho DirectInput

Khi làm việc với project Paddle Game, đây là thời gian tốt để cập nhật game framework để thêm DirectInput hỗ trợ nó.

winmain.cpp

Đáng tiếc là, sự thay đổi phải thực hiện trong WinMain. Sẽ thật sự là tốt nếu ta không cần phải mở winmain.cpp, và chúng được đóng gói trong game framework hoàn toàn và tách những công đoạn khởi tạo tách ra khỏi Game_Init, đây là một bước cần thiết.

Thêm `#include "dxinput.h"` vào trong mục include của winmain.cpp

Thêm phần khởi tạo DirectInput vào hàm WinMain trước vòng lặp while:

```
// khởi tạo DirectInput
if(!Init_DirectInput(hwnd))
{
    MessageBox(hwnd, "Error Initializing DirectInput",
"Error", MB_OK);
    return 0;
}
```

Tiếp theo, nhìn vào hàm WinProc và thêm những xử lý cho sự kiện WM_DESTROY vào:

```
// giải phóng đối tượng input
Kil_Keyboard();
Kill_Mouse();
If(dinput != NULL) dinput->Release();
```

Nghĩ thử xem, chúng ta đã cầu thả trong việc giải phóng DirectSound ở đây ! Tốt thôi, chúng ta đã quên, vậy hãy lấp vào khoảng trống đó bằng:

```
if (dsound != NULL) dsound->Release();
```

Chúng ta biết sự thay đổi này là khó khăn nhưng kết quả cuối cùng là một framework với mã được viết đầy đủ logic, hỗ trợ đầy đủ, độc lập và dễ dàng truy cập, sử dụng. Tóm lại: ta có thể chỉ cần tập trung vào gameplay thay cho Windows và DirectX.

dxinput.h

Thêm một tệp mới vào trong project của bạn gọi là dxinput.h. Đây là phần đầu tệp của DirectInput framework. Đây là code:

```
#ifndef _DXINPUT_H
#define _DXINPUT_H 1
```

```

#include <dinput.h>

//function prototypes
int Init_DirectInput(HWND);
int Init_Keyboard(HWND);
void Poll_Keyboard();
int Key_Down(int);
void Kill_Keyboard();
void Poll_Mouse();
int Init_Mouse(HWND);
int Mouse_Button(int);
int Mouse_X();
int Mouse_Y();
void Kill_Mouse();

//DirectInput objects, devices, and states
extern LPDIRECTINPUT8 dinput;
extern LPDIRECTINPUTDEVICE8 dimouse;
extern LPDIRECTINPUTDEVICE8 dikeyboard;
extern DIMOUSESTATE mouse_state;
#endif

```

dxinput.cpp

Thêm một tệp khác vào trong project – dxinput.cpp. Tệp này chứa toàn bộ mã nguồn cho quản lí bàn phím và chuột để giúp DirectInput hỗ trợ cho game.

```

#include "dxinput.h"

#define BUTTON_DOWN(obj, button) (obj.rgbButtons[button]
& 0x80)

LPDIRECTINPUT8 dinput;

```

```

LPDIRECTINPUTDEVICE8 dimouse;
LPDIRECTINPUTDEVICE8 dikeyboard;
DIMOUSESTATE mouse_state;

//keyboard input
char keys[256];

int Init_DirectInput(HWND hwnd)
{
    //initialize DirectInput object
    HRESULT result = DirectInput8Create(
        GetModuleHandle(NULL),
        DIRECTINPUT_VERSION,
        IID_IDirectInput8,
        (void**)&dinput,
        NULL);

    if (result != DI_OK)
        return 0;

    //initialize the mouse
    result = dinput->CreateDevice(GUID_SysMouse,
&dimouse, NULL);
    if (result != DI_OK)
        return 0;

    //initialize the keyboard
    result = dinput->CreateDevice(GUID_SysKeyboard,
&dikeyboard, NULL);
    if (result != DI_OK)
        return 0;

    //clean return
    return 1;
}

```

```

int Init_Mouse(HWND hwnd)
{
    //set the data format for mouse input
    HRESULT result = dimouse-
>SetDataFormat(&c_dfDIMouse);
    if (result != DI_OK)
        return 0;

    //set the cooperative level
    //this will also hide the mouse pointer
    result = dimouse->SetCooperativeLevel(hwnd,
DISCL_EXCLUSIVE | DISCL_FOREGROUND);
    if (result != DI_OK)
        return 0;

    //acquire the mouse
    result = dimouse->Acquire();
    if (result != DI_OK)
        return 0;

    //give the go-ahead
    return 1;
}

int Mouse_X()
{
    return mouse_state.lX;
}

int Mouse_Y()
{
    return mouse_state.lY;
}

int Mouse_Button(int button)

```

```

{
    return BUTTON_DOWN(mouse_state, button);
}

void Poll_Mouse()
{
    dimouse->GetDeviceState(sizeof(mouse_state),
(LPVOID) &mouse_state);
}

void Kill_Mouse()
{
    if (dimouse != NULL)
    {
        dimouse->Unacquire();
        dimouse->Release();
        dimouse = NULL;
    }
}

int Init_Keyboard(HWND hwnd)
{
    //set the data format for mouse input
    HRESULT          result          =          dikeyboard-
>SetDataFormat(&c_dfDIKeyboard);
    if (result != DI_OK)
        return 0;

    //set the cooperative level
    result          =          dikeyboard->SetCooperativeLevel(hwnd,
DISCL_NONEXCLUSIVE | DISCL_FOREGROUND);
    if (result != DI_OK)
        return 0;

    //acquire the mouse
    result = dikeyboard->Acquire();
}

```



```

        if (result != DI_OK)
            return 0;

        //give the go-ahead
        return 1;
    }

void Poll_Keyboard()
{
    dikeyboard->GetDeviceState(sizeof(keys),
(LPVOID)&keys);
}

int Key_Down(int key)
{
    return (keys[key] & 0x80);
}

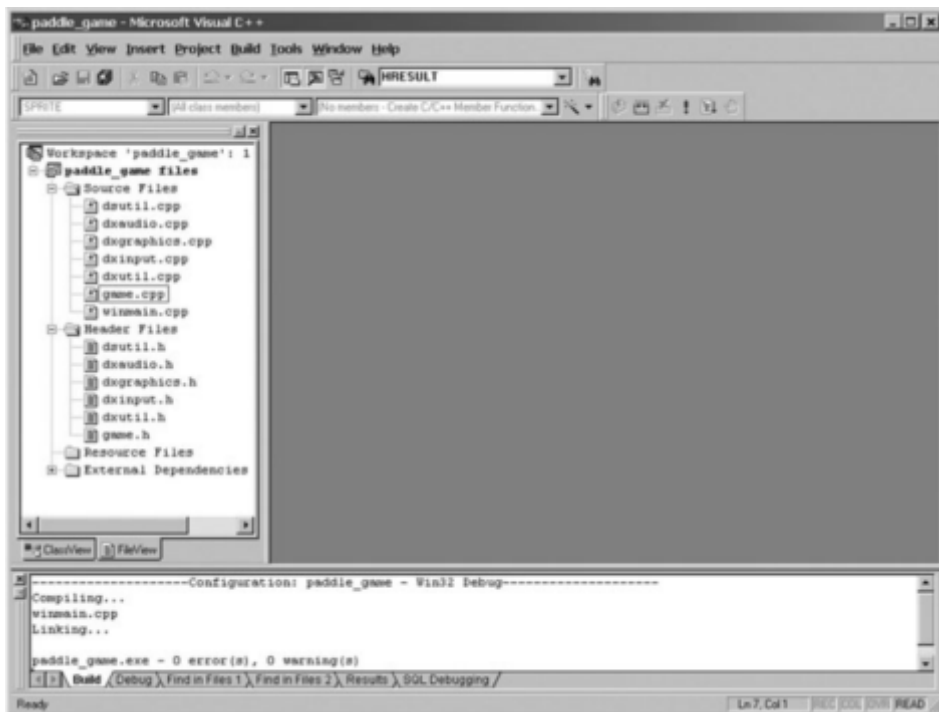
void Kill_Keyboard()
{
    if (dikeyboard != NULL)
    {
        dikeyboard->Unacquire();
        dikeyboard->Release();
        dikeyboard = NULL;
    }
}

```

5.3.2. Mã nguồn Paddle Game

Những thay đổi của chúng ta là thêm những thành phần cuối cùng của DirectX vào trong framework. Chúng ta thấy chúng thế nào? Bạn đọc sẽ thắc mắc là tại sao nó cần thiết – nó chỉ lãng phí giấy và thời gian, hay nó chính là điểm cần học ?

Dĩ nhiên đây là điểm cần học hỏi, hoặc ta sẽ không đặt mình vào vị trí lập trình game. Mã được tái sử dụng là một trong những chìa khóa để trở thành một lập trình viên chuyên nghiệp. Chúng ta đơn giản không cần viết lại code và chỉ mong muốn là làm những việc thực tế. Những mã nguồn ta đã tạo cung cấp một game framework tốt, giảm số lượng công việc cần làm khi phải làm Windows/DirectX game.



Hình 5.2. Paddle Game project.

game.h

Đây là phần đầu tệp cho game

```
#ifndef _GAME_H
#define _GAME_H 1

//windows/directx headers
#include <d3d9.h>
```

```

#include <d3dx9.h>
#include <dxerr9.h>
#include <dsound.h>
#include <dinput.h>
#include <windows.h>
#include <time.h>
#include <stdio.h>
#include <stdlib.h>

//framework-specific headers
#include "dxgraphics.h"
#include "dxaudio.h"
#include "dxinput.h"

//application title
#define APPTITLE "Paddle_Game"

//screen setup
#define FULLSCREEN 0           //0 = windowed, 1 = fullscreen
#define SCREEN_WIDTH 640
#define SCREEN_HEIGHT 480

//function prototypes
int Game_Init(HWND);
void Game_Run(HWND);
void Game_End(HWND);

//sprite structure
typedef struct {
    int x,y;
    int width,height;
    int movex,movey;
    int curframe,lastframe;
    int animdelay,animcount;
    int scalex, scaley;
    int rotation, rotaterate;

```

```
} SPRITE;
```

game.cpp

Đây là mã nguồn cho Paddle Game project. Tôi sẽ giải thích những phần chính mã ở cuối.

```
#include "game.h"

//background image
LPDIRECT3DSURFACE9 back;

//sprite handler
LPD3DXSPRITE sprite_handler;

//ball sprite
LPDIRECT3DTEXTURE9 ball_image;
SPRITE ball;

//paddle sprite
LPDIRECT3DTEXTURE9 paddle_image;
SPRITE paddle;

//the wave sound
CSound *sound_bounce;
CSound *sound_hit;

//misc
long start = GetTickCount();
HRESULT result;

//initializes the game
int Game_Init(HWND hwnd)
{
    //set random number seed
    srand(time(NULL));
```

```

        //initialize mouse
        if (!Init_Mouse(hwnd))
        {
            MessageBox(hwnd, "Error initializing the
mouse", "Error", MB_OK);
            return 0;
        }

        //initialize keyboard
        if (!Init_Keyboard(hwnd))
        {
            MessageBox(hwnd, "Error initializing the
keyboard", "Error", MB_OK);
            return 0;
        }

        //create sprite handler object
        result = D3DXCreateSprite(d3ddev, &sprite_handler);
        if (result != D3D_OK)
            return 0;

        //load the background image
        back = LoadSurface("background.bmp", NULL);
        if (back == NULL)
            return 0;

        //load the ball sprite
        ball_image = LoadTexture("ball.bmp",
D3DCOLOR_XRGB(255,0,255));
        if (ball_image == NULL)
            return 0;

        //set the ball's properties
        ball.x = 400;
        ball.y = 200;

```

```

    ball.width = 12;
    ball.height = 12;
    ball.movex = 8;
    ball.movey = -8;

    //load the paddle sprite
    paddle_image = LoadTexture("paddle.bmp",
D3DCOLOR_XRGB(255,0,255));
    if (paddle_image == NULL)
        return 0;

    //set paddle properties
    paddle.x = 300;
    paddle.y = SCREEN_HEIGHT - 50;
    paddle.width = 90;
    paddle.height = 26;

    //load bounce wave file
    sound_bounce = LoadSound("bounce.wav");
    if (sound_bounce == NULL)
        return 0;

    //load the hit wave file
    sound_hit = LoadSound("hit.wav");
    if (sound_hit == NULL)
        return 0;

    //return okay
    return 1;
}

int Collision(SPRITE spritel, SPRITE sprite2)
{
    RECT rect1;
    rect1.left = spritel.x+1;
    rect1.top = spritel.y+1;

```

```

    rect1.right = spritel.x + spritel.width-1;
    rect1.bottom = spritel.y + spritel.height-1;

    RECT rect2;
    rect2.left = sprite2.x+1;
    rect2.top = sprite2.y+1;
    rect2.right = sprite2.x + sprite2.width-1;
    rect2.bottom = sprite2.y + sprite2.height-1;

    RECT dest;
    return IntersectRect(&dest, &rect1, &rect2);
}

//the main game loop
void Game_Run(HWND hwnd)
{
    //ball position vector
    D3DXVECTOR3 position(0,0,0);

    //make sure the Direct3D device is valid
    if (d3ddev == NULL)
        return;

    //update mouse and keyboard
    Poll_Mouse();
    Poll_Keyboard();

    //after short delay, ready for next frame?
    //this keeps the game running at a steady frame
rate
    if (GetTickCount() - start >= 30)
    {
        //reset timing
        start = GetTickCount();

        //move the ball sprite

```

```

ball.x += ball.movex;
ball.y += ball.movey;

//bounce the ball at screen edges
if (ball.x > SCREEN_WIDTH - ball.width)
{
    ball.x -= ball.width;
    ball.movex *= -1;
    PlaySound(sound_bounce);
}
else if (ball.x < 0)
{
    ball.x += ball.width;
    ball.movex *= -1;
    PlaySound(sound_bounce);
}

if (ball.y > SCREEN_HEIGHT - ball.height)
{
    ball.y -= ball.height;
    ball.movey *= -1;
    PlaySound(sound_bounce);
}
else if (ball.y < 0)
{
    ball.y += ball.height;
    ball.movey *= -1;
    PlaySound(sound_bounce);
}

//move the paddle
paddle.x += Mouse_X();
if (paddle.x > SCREEN_WIDTH - paddle.width)
    paddle.x = SCREEN_WIDTH - paddle.width;
else if (paddle.x < 0)
    paddle.x = 0;

```



```

        //check for left arrow
        if (Key_Down(DIK_LEFT))
            paddle.x -= 5;

        //check for right arrow
        if (Key_Down(DIK_RIGHT))
            paddle.x += 5;

        //see if ball hit the paddle
        if (Collision(paddle, ball))
        {
            ball.y -= ball.movey;
            ball.movey *= -1;
            PlaySound(sound_hit);
        }
    }

    //start rendering
    if (d3ddev->BeginScene())
    {
        //erase the entire background
        d3ddev->StretchRect(back, NULL, backbuffer,
NULL, D3DTEXF_NONE);

        //start sprite handler
        sprite_handler->Begin(D3DXSPRITE_ALPHABLEND);

        //draw the ball
        position.x = (float)ball.x;
        position.y = (float)ball.y;
        sprite_handler->Draw(
            ball_image,
            NULL,

```

```

        NULL,
        &position,
        D3DCOLOR_XRGB(255,255,255));

//draw the paddle
position.x = (float)paddle.x;
position.y = (float)paddle.y;
sprite_handler->Draw(
    paddle_image,
    NULL,
    NULL,
    &position,
    D3DCOLOR_XRGB(255,255,255));

//stop drawing
sprite_handler->End();

//stop rendering
d3ddev->EndScene();
}

//display the back buffer on the screen
d3ddev->Present(NULL, NULL, NULL, NULL);

//check for mouse button (to exit program)
if (Mouse_Button(0))
    PostMessage(hwnd, WM_DESTROY, 0, 0);

//check for escape key (to exit program)
if (Key_Down(DIK_ESCAPE))
    PostMessage(hwnd, WM_DESTROY, 0, 0);
}

//frees memory and cleans up before the game ends
void Game_End(HWND hwnd)
{

```

```

    if (ball_image != NULL)
        ball_image->Release();

    if (paddle_image != NULL)
        paddle_image->Release();

    if (back != NULL)
        back->Release();

    if (sprite_handler != NULL)
        sprite_handler->Release();

    //if (sound_bounce != NULL)
    //    delete sound_bounce;

    //if (sound_hit != NULL)
    //    delete sound_hit;
}

```

5.3.3. Giải thích Paddle Game

Hầu hết code cho phần game không phức tạp và giống với những project trước đó. Một điểm khác biệt ở đây là hàm phát hiện va chạm, nó được sử dụng để phát hiện khi paddle và bóng va chạm với màn hình. Khi va chạm được phát hiện, quả bóng được chuyển hướng ngược lại. Windows cung cấp hàm rất tiện dụng để kiểm tra va chạm, gọi là `IntersectRect`. Hàm này có cú pháp:

```

BOOL IntersectRect(
    LPRECT lprcDst,
    CONST RECT *lprcDst1,
    CONST RECT *lprcDst2
);

```

Tham số đầu tiên thì không cần thiết, vì vậy ta chỉ truyền vào RECT giả cho nó. Tham số thứ 2 và 3 là cần thiết, yêu cầu là RECT, và thường thì nó dữ liệu từ 2 sprite trước khi gọi IntersectRect. Hàm này sử dụng hai hình chữ nhật để xác định, nếu có cắt nhau giữa chúng, dựa trên left, top, right, và bottom giá trị của chúng. Nếu ta thực sự quan tâm về cắt nhau giữa hai hình chữ nhật, thì ta có thể sử dụng RECT lấp dữ liệu với data vào trong tham số thứ nhất. Có một điều khiến ta lo lắng đó là khi sử dụng hàm này sẽ trả về giá trị chỉ ra: 0 là thất bại, 1 là thành công. Nếu nó trả về 1, thì ta biết là va chạm xảy ra.

Bạn có thể nghĩ về một hàm tốt hơn? Một ý tưởng đó là thêm vào nhóm các khối gạch vào trong game (sử dụng mảng RECT nắm giữ vị trí của chúng) và rồi sử dụng IntersectRect để xem nếu quả bóng có chạm với bất kì khối nào hay không. Ta có hủy khối gạch và cho quả bóng di chuyển ra xa, giống như khi chúng chạm vào paddle. Đây là một game truyền thống: paddle và bóng.

Chúng ta sẽ tìm hiểu kỹ thuật xử lý va chạm phức tạp hơn một chút ở chương 7.

CHƯƠNG 6

LẬP TRÌNH ÂM THANH

6.1. Tổng quan DirectSound

DirectSound là một thành phần của DirectX, quản lý tất cả hiệu ứng âm thanh trong game, và hỗ trợ trộn lẫn nhiều kênh. Nói một cách đơn giản, ta sẽ ra lệnh cho DirectSound chơi hiệu ứng âm thanh và nó sẽ thay ta quản lý những thông tin chi tiết của âm thanh(bao gồm thông tin âm thanh đang chạy).

Việc code âm thanh yêu cầu ta khởi tạo, load và play file wav bằng DirectSound thì ít hơn so với code bitmap và sprite mà chúng ta đã tìm hiểu trong các chương trước. Do đó, nó có vẻ còn thú vị hơn cả việc phát minh ra bánh xe! Rồi ta sẽ thấy làm cách nào để làm chủ được DirectSound và DirectMusic. Việc sử dụng các wrapper coi như chống lại bản năng của một lập trình viên. Chúng ta thích hiểu mọi thứ về code mà ta sử dụng hoặc thích tự mình viết code hơn sử dụng code của người khác viết. Tuy nhiên, vẫn có lúc dù thích hay không, chúng ta vẫn nên sử dụng những thứ có sẵn. Nói chung, DirectX bản thân nó đã là thư viện game được viết bởi người khác rồi. Tất nhiên, không vấn đề gì nếu ta viết hết bằng C, và khi viết cuốn sách này, cũng có lúc chúng ta sẽ sử dụng code C++ để tránh việc sử dụng lại code. Trong trường hợp này, ta sẽ sử dụng lớp “dxutil” nhưng sẽ không biết rõ nó hoạt động như thế nào. Ta có thể thắc mắc nó hoạt động như thế nào, có rất nhiều thứ ta chưa cần hiểu ngay được, nhưng dần dần, ta sẽ làm game mà không lo lắng về nó.

DirectSound và 1 wrapper bao gồm 2 file code, nằm trong DirectX 9.0b SDK. Những file đó là: dsutil.h và dsutil.cpp, và ta có thể tìm thấy ở \DX90SDK\Samples\C++\Common. Ta cần include 2 file đó vào project game của mình để có thể sử dụng DirectSound.

Chú ý: Không có gì sai khi sử dụng wrapper khi bạn không có thời gian hoặc quá phức tạp khi bạn tự viết các wrapper đó. Nếu muốn biết tất cả về DirectAudio, hãy đọc cuốn Beginning Game Audio Programing - Mason McCuskey. Cuốn sách này sẽ đi chi tiết các DirectSound interface và chỉ cho chúng ta làm cách nào để tạo một sound library cho dự án game của mình.

4 files cần thiết khi biên dịch chương trình trong chương này: dxutil.h, dxutil.cpp, dsuti.h, dsuti.cpp. Những file này có thể tìm thấy ở \DX90SDK\Samples\C++\Common. Nếu ko có những files này, chương trình sẽ không biên dịch được.

Có 3 class được định nghĩa trong file dsutil được quan tâm ở đây:

CSoundManager: quản lý phần cứng

CSound: dùng để tạo bộ nhớ đệm.

CWaveFile: dùng để load file wav vào CSound buffer.

6.1.1. Khởi tạo DirectSound

Việc đầu tiên phải làm để sử dụng DirectSound là tạo 1 thể hiện của lớp CSoundManager.

```
CSoundManager *dsound = new CSoudManager();
```

Tiếp theo, ta gọi hàm Initialize để khởi tạo trình quản lý DirectSound:

```
dsound->Initialize(window_handle, DSDCL_PRIORITY);
```

Tham số thứ nhất là handle của chương trình, tham số thứ 2 là mức độ phối hợp của DirectSound với các chương trình khác, có 3 lựa chọn:

- DSSCL_NORMAL: Chia sẻ phần cứng âm thanh với các chương trình khác.
- DSSCL_PRIORITY: Chiếm dữ phần cứng cao hơn so với chương trình khác (Khuyến dùng)
- DSSCL_WRITEPRIMATE: Có thể truy xuất, chỉnh sửa trực tiếp tới bộ nhớ âm thanh chính.

Thường thường thì ta nên chọn DSSCL_PRIORITY, để game chúng ta có thể dành riêng phần cứng âm thanh nhiều hơn các chương trình khác.

6.1.2. Tạo bộ nhớ đệm âm thanh

Sau khi khởi tạo trình quản lý DirectSound (thông qua CSoundManager), ta sẽ load hết tất cả hiệu ứng âm thanh cho game. Bạn sẽ cấp phát biến để truy xuất tới CSound như sau:

```
CSound *wave;
```

Đối tượng của CSound mà bạn vừa tạo sẽ bao quát luôn bộ nhớ đệm âm thanh gọi là: LPDIRECTSOUNDBUFFER8, và cảm ơn dsutil, chúng ta không phải code lại nó 😊

6.1.3. Nạp file định dạng wave

Các hiệu ứng âm thanh được tạo và trộn lẫn bởi DirectSound có thể được gọi là vùng nhớ đệm chính. Cũng giống như Direct3D, vùng đệm chính này chính sẽ được đưa ra thiết bị xuất. Nhưng đối với DirectSound, vùng nhớ đệm thứ 2 cho âm thanh này sẽ khác với hình ảnh và ta phải chạy hiệu ứng âm thanh đó bằng việc gọi hàm Play.

Việc tải file wave vào bộ nhớ đệm âm thanh thứ 2 này có thể thực hiện bằng việc gọi hàm sau Create:

```
HRESULT Create(  
    CSound** ppSound,  
    LPTSTR strWaveFileName,  
    DWORD dwCreationFlags = 0,  
    GUID guid3Dalgorithm = GUID_NULL,  
    DWORD dwNumBuffer = 1  
);
```

Tham số đầu tiên là đối tượng CSound mà bạn muốn load mới hiệu ứng âm thanh. Tham số thứ 2 là tên file wave. Các tham số còn lại có thể để mặc định. Đây là ví dụ:


```
result = dsound->Create(&wave, "snicker.wav");
```

6.1.4. Chạy hiệu ứng âm thanh

Ta có thể chạy hiệu ứng âm thanh bất cứ lúc nào mình muốn mà không phải lo về việc chồng lấn âm thanh vì DirectSound đã quản lý tất cả thông tin chi tiết cho chúng ta. Lớp CSound có hàm Play cho phép chạy được âm thanh

```
HRESULT Play(  
    DWORD dwPriority = 0,  
    DWORD dwFlags = 0,  
    LONG lVolumn = 0,  
    LONG lFrequency = -1,  
    LONG lPan = 0  
);
```

Tham số đầu tiên là mức độ ưu tiên của âm thanh. Ta có thể để mặc định là 0. Tham số thứ 2 là cờ quy định âm thanh có được lặp đi lặp lại hay không, tức là nó sẽ chơi lại âm thanh khi kết thúc. Nếu muốn lặp lại âm thanh, ta sẽ gán cờ DSBPLAY_LOOPING cho tham số này. 3 tham số còn lại ảnh hưởng tới độ lớn âm thanh, tần số âm thanh,... Ta có thể để mặc định hoặc thay đổi nếu có kinh nghiệm.

Đây là ví dụ làm cách nào để sử dụng những hàm này. Đầu tiên là hàm chơi nhạc thông thường. Chúng ta có thể điền vào các tham số mặc định.

```
wave->Play();
```

Đây là ví dụ lặp âm thanh.

```
wave->Play(0, DSBPLAY_LOOPING);
```

Để dừng việc chơi âm thanh, bạn gọi hàm Stop. Hàm này rất hữu dụng khi lập âm thanh.

```
HRESULT Stop();
```

Ví dụ:

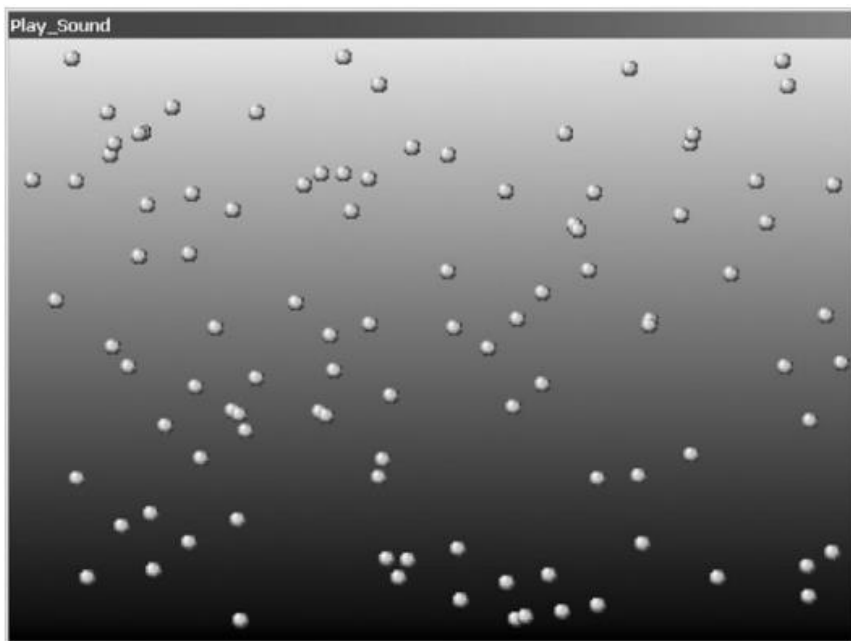
```
wave->Stop();
```

6.1.5. Kiểm tra DirectSound

Hãy viết một chương trình đơn giản về DirectSound để kiểm tra những cái ta vừa học trong chương này. Vì DirectSound là một thành phần mới, do đó, chúng ta phải thêm vào dự án một thứ được gọi là “framework” bằng việc tạo một file định nghĩa và file source. Chúng ta sẽ được trình bày làm cách nào để tạo project từ đầu, thêm vào những file cần thiết và chạy những hàm trong DirectSound mà chúng ta vừa học. Sau khi xong một project cơ bản, chúng ta sẽ làm 1 chương trình ví dụ hiện ngàn trái banh lên màn hình cùng với chạy âm thanh lặp và không lặp.

6.2. Xây dựng ứng dụng minh họa

Bây giờ, bạn đọc sẽ được hướng dẫn cách tạo một project từ đầu. Mặc dù ta có thể mở một project có sẵn và tiến hành chỉnh sửa nó, nhưng ta nên tạo một project mới vì ta có thể thực hành và trải qua nhiều bước để thực hiện.



Hình 6.1. Màn hình chương trình Play_Sound demo làm cách nào để sử dụng DirectSound.

Hãy bật VSC++, Mở File menu và chọn New, hiện lên cửa sổ New. Chắc chắn Projects tab được chọn. Chọn Win32 Application trong mục Project Type và gõ Play_Sound trong mục tên project. Click OK để tạo mới project. Thông thường, đừng để VC++ thêm vào bất kì file nào vào project.

6.2.1. Sao chép tái sử dụng mã nguồn

Tiếp theo, sao chép những file trong project trước vào thư mục mới trong project của bạn. Đây là những file ta cần:

- winmain.cpp
- dxgraphics.h
- dxgraphics.cpp
- game.h
- game.cpp

Những file `game.h` và `game.cpp` sẽ được thay thế bằng code mới, nhưng cũng chẳng ảnh hưởng nếu ta chép vào project mới này. Nó cũng chỉ đơn giản là việc tạo mới file trong cửa sổ New

6.2.2. Sao chép file `dsutil`

Bước tiếp khá là khó chịu nhưng thật sự cần thiết để có thể sử dụng class `dsutil`. Có 2 files mà ta phải copy vào thư mục project và phải add vào project:

- `dxutil.cpp`
- `dsutil.cpp`

File định nghĩa của 2 file source kia không cần phải thêm vào project, vì ta có thể thêm nó vào include path của VC++. Tuy nhiên, lời khuyên là ta cũng nên copy 2 file định nghĩa kia vào thư mục project luôn:

- `dxutil.h`
- `dsutil.h`

Ta có thể tìm những file đó ở đâu? Nếu đang sử dụng DirectX 9.0b, thì những file đó sẽ ở `\DX90SDK\Samples\C++\Common`. Hãy thêm đường dẫn vào list thư mục included trong VC++.

Có rất nhiều files định nghĩa mà `dxutil.cpp` cần, và bạn không muốn screw chúng lại?, dễ dàng include đường dẫn.

Trong VC++, mở Tools, chọn Options, cửa sổ Options hiện lên.

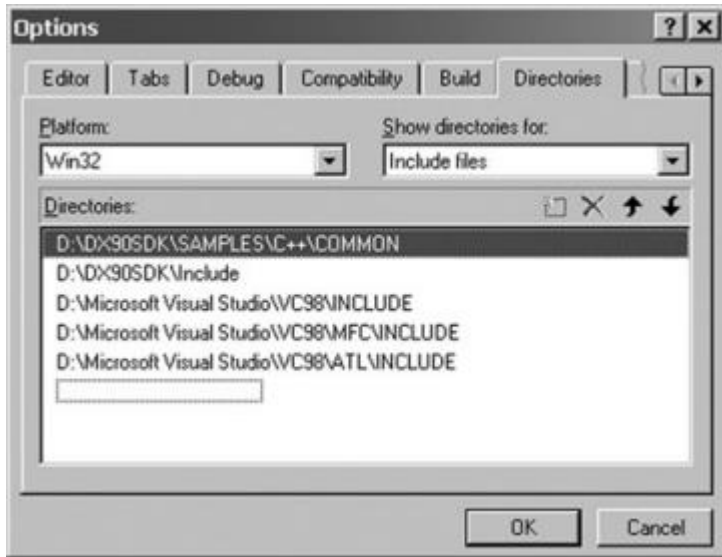
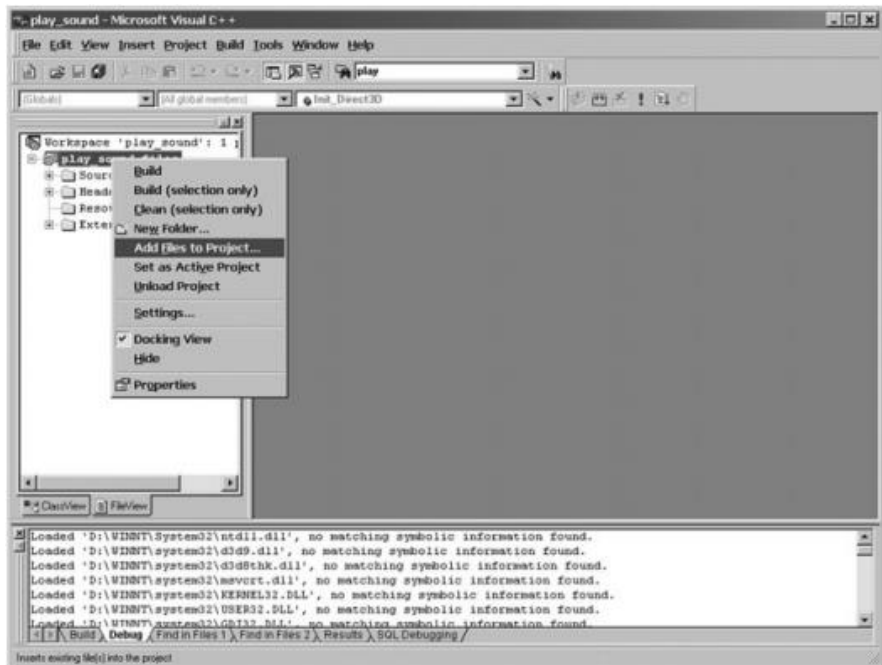


Figure 6.2. Cửa sổ Options để add đường dẫn Include file

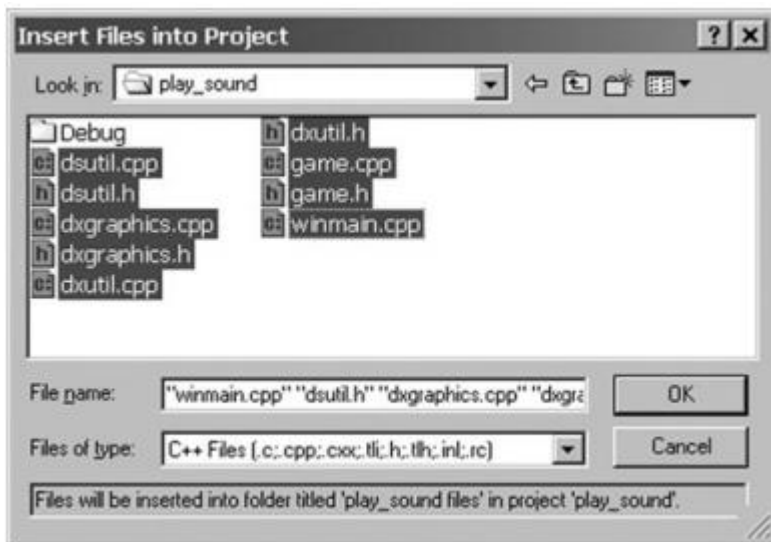
6.2.3. Thêm những file copy vào project

Sau khi copy tất cả những file trên vào trong thư mục project, ta có thể add nó vào project bằng cách click phải vào tên project chọn AddFiles to Project từ menu.



Hình 6.3. Thêm file vào project

Cửa sổ Insert Files into Project sẽ hiện ra, ta có thể add nhiều file bằng cách sử dụng Ctrl+Click vào trong project. Những file sau sẽ được thêm vào project:



- winmain.cpp
- dxgraphics.h
- dxgraphics.cpp
- game.h
- game.cpp
- dxutil.cpp
- dsutil.cpp
- dxutil.h
- dsutil.h

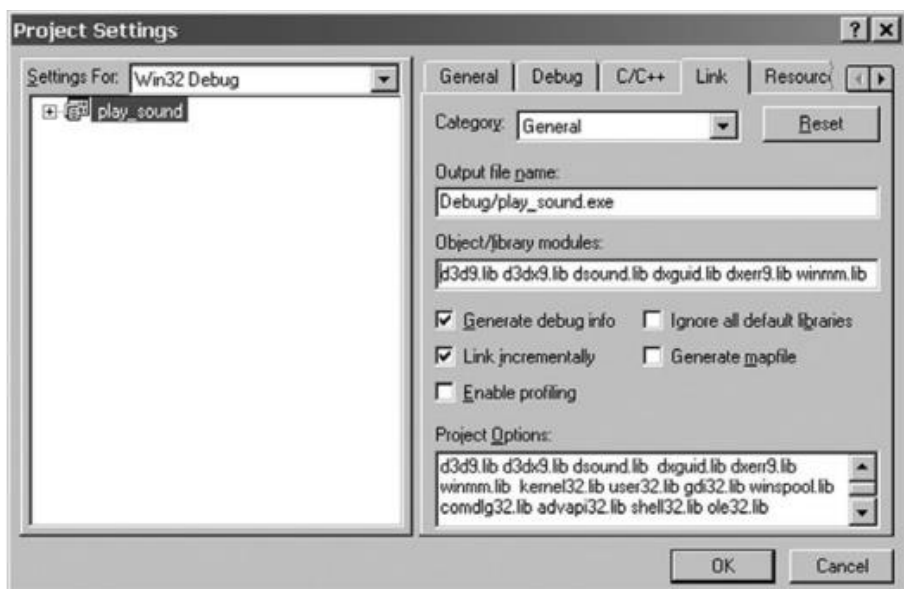
Ta có thể xác nhận lại bằng hình 6.4.

6.2.4. Thêm liên kết tới thư viện DirectX

Tiếp, cấu hình các thư viện DirectX cần thiết. Lúc này, những file hỗ trợ DirectSound cần được hỗ trợ trong code, vì thế có một số thư viện mà ta chưa thấy trước đó. Mở menu Open the Project, chọn Settings, hiện ra cửa sổ Project Setting.



Hình 6.5. Những file khung hệ thống sẽ được add vào project



Hình 6.6. Thêm liên kết thư viện DirectX vào danh sách thư viện trong cửa sổ Project Settings

Sau đây là danh sách tên file được thêm vào mục Object/library trong cửa sổ Settings

- d3d9.lib
- d3dx9.lib
- dsound.lib
- dxguid.lib
- dxerr9.lib
- winmm.lib

Đây là danh sách các file thư viện được thêm vào project

6.2.5. Tạo những file hỗ trợ cho DirectX Audio

Giờ, project Play_Sound đã sẵn sàng cho việc chạy DirectSound. Ta phải thêm code thay thế 2 files sau:

- dxaudio.h
- dxaudio.cpp

File khai báo sẽ bao gồm định nghĩa cho các hàm DirectSound mà bạn sẽ dùng để tải và chạy âm thanh trong game

Tạo dxaudio.h

Mở menu File chọn New, hộp thoại New mở ra. Chọn C/C++ Header File và gõ tên file .Click OK để thêm file vào project.

Đây là code của file dxaudio.h

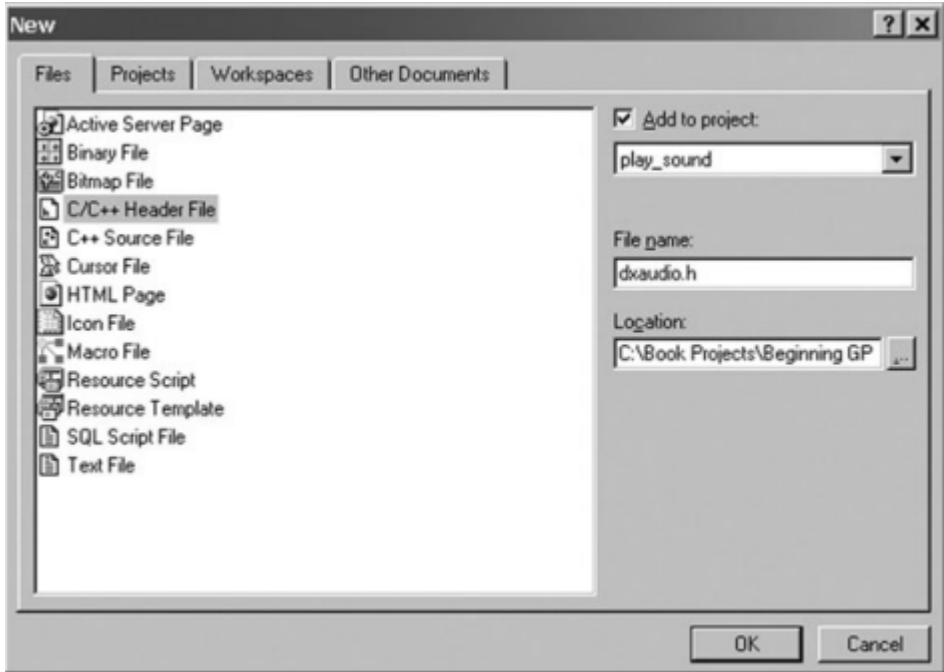
```
// dxaudio.h - DirectSound framework header file
#ifndef _DXAUDIO_H
#define _DXAUDIO_H

#include "dsutil.h"

//primary DirectSound object
extern CSoundManager *dsound;
```

```
//function prototypes
int Init_DirectSound(HWND);
CSound *LoadSound(char *);
void PlaySound(CSound *);
void LoopSound(CSound *);
void StopSound(CSound *);
#endif

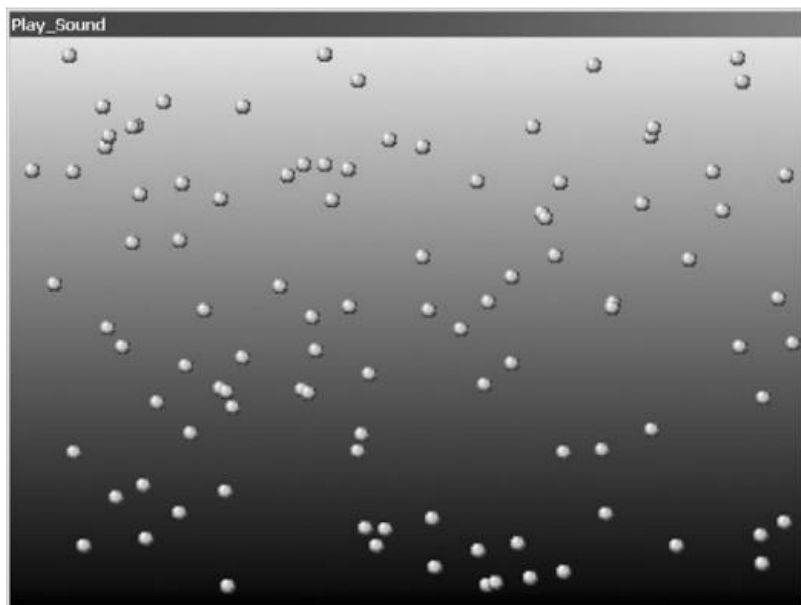
///
```



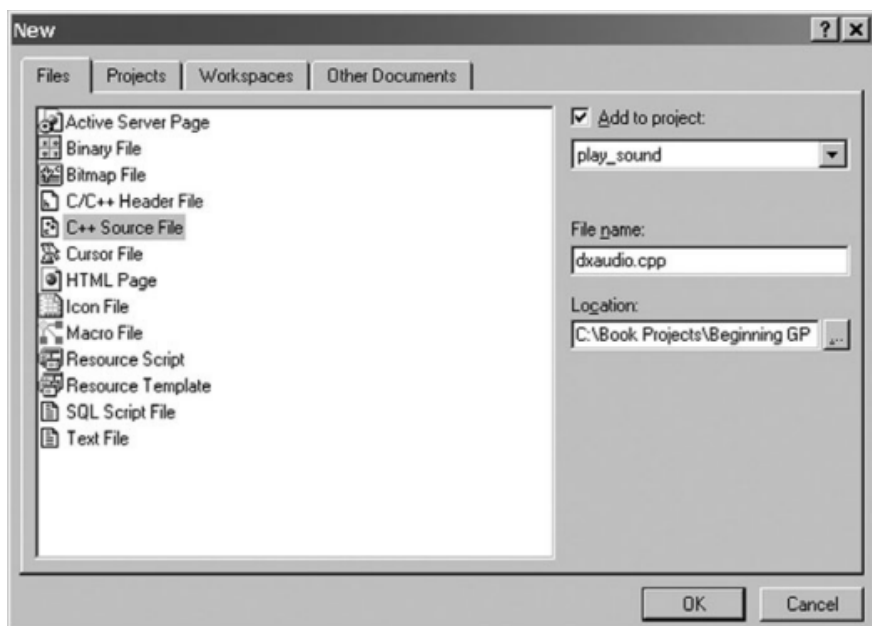
Hình 6.7. Thêm file dxaudio.h vào project

Tạo dxaudio.cpp

Mở menu File chọn New, hộp thoại New mở ra. Chọn C/C++ Header File và gõ tên file. Click OK để thêm file vào project.



Hình 6.8. Minh họa âm thanh file dxaudio.cpp



Hình 6.9. Thêm file dxaudio.cpp vào project

6.2.6. Tinh chỉnh hệ thống

Vấn đề tiếp theo của chúng ta là làm chủ thật tốt code. Ta có thể làm theo những bước sao để thêm DirectSound vào WinMain hoặc ta có thể gọi hàm Init_DirectSound từ hàm khởi tạo main. Tuy nhiên, lời khuyên là add nó vào Winmain.

Thêm khởi tạo DirectSound vào winmain.cpp

Mở winmain.cpp trong project. Kéo xuống đến đoạn bắt đầu vòng lặp while, giống thể này:

```
// main message loop
int done = 0;
while (!done)
```

Giống như trên, ta sẽ thấy phần khởi tạo Direct3D và phần khởi tạo game. Bạn có thể thêm phần khởi tạo DirectSound phía sau đó trước khi nó bắt đầu vòng lặp

```
// initialize DirectSound
if (!Init_DirectSound(hWnd))
{
    MessageBox(hWnd, "Error initialize DirectSound",
"Error", MB_OK);
}
```

Chú ý: Nếu không theo kịp hoàn toàn, hay một phần trong việc tạo project. Ta có thể tự cứu mình bằng cách chạy ví dụ có sẵn đi kèm.

Thêm file Game

Chúng ta cũng hơi dài dòng một chút, nhưng nếu ta theo dõi một cách cẩn thận thì ta sắp có một project chạy được rồi đó.

Hiện tại, file game.h vào game.cpp đang chứa source code của project trước, nó không làm gì với DirectSound cả. Do đó, để tiện, đây là những file, ta chỉ cần copy và thay thế chúng

game.h

Đây là code của file game.h. Chỉ cần xóa hết code đã có và thay thế bằng code mới dưới đây:

```
#ifndef _GAME_H
#define _GAME_H

//windows/directx headers
#include <d3d9.h>
#include <d3dx9.h>
#include <dxerr9.h>
#include <dsound.h>
#include <windows.h>
#include <time.h>
#include <stdio.h>
#include <stdlib.h>

//framework-specific headers
#include "dxgraphics.h"
#include "dxaudio.h"

//application title
#define APPTITLE "Play_Sound"

//screen setup
#define FULLSCREEN 0          //0 = windowed, 1 = fullscreen
#define SCREEN_WIDTH 640
#define SCREEN_HEIGHT 480

//macros to read the keyboard asynchronously
#define KEY_DOWN(vk_code) ((GetAsyncKeyState(vk_code) & 0x8000) ? 1 : 0)
#define KEY_UP(vk_code) ((GetAsyncKeyState(vk_code) & 0x8000) ? 1 : 0)
```

```
//function prototypes
int Game_Init(HWND);
void Game_Run(HWND);
void Game_End(HWND);

//sprite structure
typedef struct {
    int x,y;
    int width,height;
    int movex,movey;
    int curframe,lastframe;
    int animdelay,animcount;
    int scalex, scaley;
    int rotation, rotaterate;
} SPRITE;
#endif
```

game.cpp

Đây là code của file game.cpp. Chỉ cần xóa hết code đã có và thay thế bằng code mới dưới đây.

```
#include "game.h"

//number of balls on the screen
#define NUMBALLS 100

//misc variables
LPDIRECT3DTEXTURE9 ball_image;
SPRITE balls[NUMBALLS];
LPDIRECT3DSURFACE9 back;
LPD3DXSPRITE sprite_handler;
HRESULT result;
```

```

//timing variable
long start = GetTickCount();

//the wave sound
CSound *sound_bounce;
CSound *sound_electric;

//initializes the game
int Game_Init(HWND hwnd)
{
    int n;

    //set random number seed
    srand(time(NULL));

    //create sprite handler object
    result = D3DXCreateSprite(d3ddev, &sprite_handler);
    if (result != D3D_OK)
        return 0;

    //load the background image
    back = LoadSurface("background.bmp", NULL);
    if (back == NULL)
        return 0;

    //load the ball sprite
    ball_image = LoadTexture("ball.bmp",
D3DCOLOR_XRGB(255,0,255));
    if (ball_image == NULL)
        return 0;

    //set the balls' properties
    for (n=0; n<NUMBALLS; n++)
    {
        balls[n].x = rand() % SCREEN_WIDTH;
        balls[n].y = rand() % SCREEN_HEIGHT;
    }
}

```

```

        balls[n].width = 12;
        balls[n].height = 12;
        balls[n].movex = 1 + rand() % 6;
        balls[n].movey = rand() % 12 - 6;
    }

    //load bounce wave file
    sound_bounce = LoadSound("bounce.wav");
    if (sound_bounce == NULL)
        return 0;

    //load the electric wave file
    sound_electric = LoadSound("electric.wav");
    if (sound_electric == NULL)
        return 0;

    //return okay
    return 1;
}

//the main game loop
void Game_Run(HWND hwnd)
{
    D3DXVECTOR3 position(0,0,0);    //ball position vector
    int n;
    int playing = 0;

    //make sure the Direct3D device is valid
    if (d3ddev == NULL)
        return;

    //after short delay, ready for next frame?
    //this keeps the game running at a steady frame
rate
    if (GetTickCount() - start >= 30)

```



```

{
    //reset timing
    start = GetTickCount();

    //move the ball sprites
    for (int n=0; n<NUMBALLS; n++)
    {
        balls[n].x += balls[n].movex;
        balls[n].y += balls[n].movey;

        //bounce the ball at screen edges
        if (balls[n].x > SCREEN_WIDTH -
balls[n].width)
        {
            balls[n].x -= balls[n].width;
            balls[n].movex *= -1;
            PlaySound(sound_bounce);
        }
        else if (balls[n].x < 0)
        {
            balls[n].x += balls[n].width;
            balls[n].movex *= -1;
            PlaySound(sound_bounce);
        }

        if (balls[n].y > SCREEN_HEIGHT -
balls[n].height)
        {
            balls[n].y -= balls[n].height;
            balls[n].movey *= -1;
            PlaySound(sound_bounce);
        }
        else if (balls[n].y < 0)
        {
            balls[n].y += balls[n].height;
            balls[n].movey *= -1;

```

```

        PlaySound(sound_bounce);
    }
}

//start rendering
if (d3ddev->BeginScene())
{
    //erase the entire background
    d3ddev->StretchRect(back, NULL, backbuffer,
NULL, D3DTEXF_NONE);

    //start sprite handler
    sprite_handler->Begin(D3DXSPRITE_ALPHABLEND);

    //draw the balls
    for (n=0; n<NUMBALLS; n++)
    {
        position.x = (float)balls[n].x;
        position.y = (float)balls[n].y;
        sprite_handler->Draw(
            ball_image,
            NULL,
            NULL,
            &position,
            D3DCOLOR_XRGB(255,255,255));
    }

    //stop drawing
    sprite_handler->End();

    //stop rendering
    d3ddev->EndScene();
}

```

```

//display the back buffer on the screen
d3ddev->Present(NULL, NULL, NULL, NULL);

//check for escape key (to exit program)
if (KEY_DOWN(VK_ESCAPE))
    PostMessage(hwnd, WM_DESTROY, 0, 0);

//spacebar plays the electric sound
if (KEY_DOWN(VK_SPACE))
    LoopSound(sound_electric);

//enter key stops the electric sound
if (KEY_DOWN(VK_RETURN))
    StopSound(sound_electric);
}

//frees memory and cleans up before the game ends
void Game_End(HWND hwnd)
{
    if (ball_image != NULL)
        ball_image->Release();

    if (back != NULL)
        back->Release();

    if (sprite_handler != NULL)
        sprite_handler->Release();

    if (sound_bounce != NULL)
        delete sound_bounce;

    if (sound_electric != NULL)
        delete sound_electric;
}

```

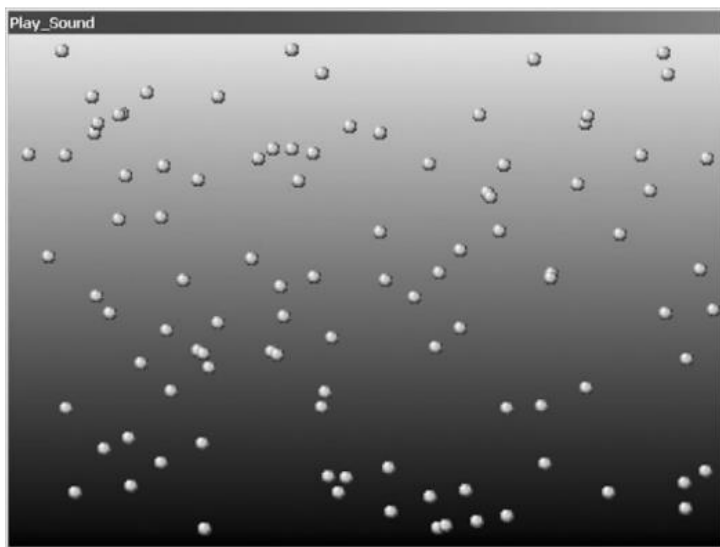
```
}
```

6.3. Chạy chương trình

Khi chạy chương trình, ta sẽ thiết lập chế độ toàn màn hình hay cửa sổ. Lời khuyên là nên chạy tất cả các ví dụ bằng chế độ toàn màn hình. Thay đổi chế độ trong file `game.h`:

```
#define FULLSCREEN 0 // 0 = windowed, 1 = fullscreen
```

Khi chạy chương trình, ta có thể chạy hoặc dừng việc chơi nhạc bằng cách nhấn phím “khoảng trắng” để bắt đầu chạy lặp nhạc, phím ENTER để dừng chơi nhạc.



Hình 6.10. Chương trình `Play_Sound` demo làm cách nào để sử dụng `DiectSound`

CHƯƠNG 7

ĐẠI CƯƠNG XỬ LÝ VA CHẠM

7.1. Giới thiệu

7.1.1. Tại sao cần xử lý va chạm?

Xử lý va chạm là một bài toán cực kỳ quan trọng trong phát triển game vì va chạm là một phần rất quan trọng trong việc thể hiện yếu tố vật lý trong thế giới game (bên cạnh yếu tố cơ bản như vận tốc, gia tốc, trọng lực). Đây là cơ sở tạo cho game thủ cảm giác thế giới game “gần gũi” với thế giới thật. Giải quyết bài toán này không tốt sẽ làm cho game trở nên *xa lạ, khó chấp nhận* đối với người chơi và hậu quả là game của chúng ta sẽ mau chóng bị chối bỏ.

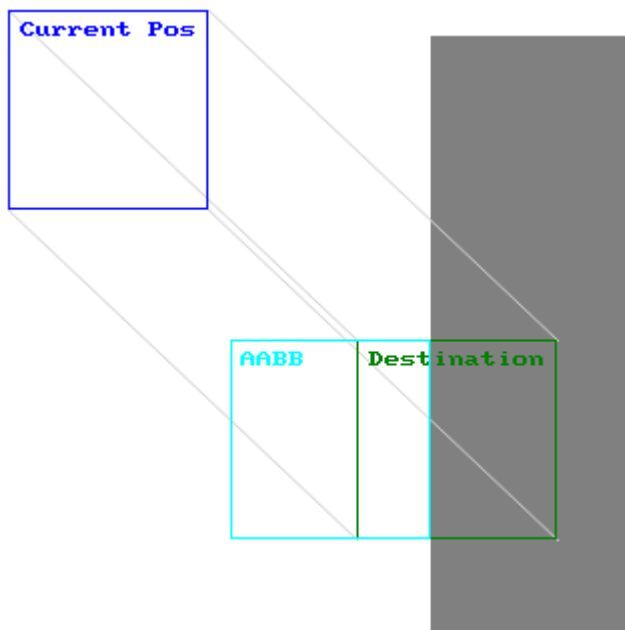
Hầu hết các lập trình viên mới chập chững vào nghề đều hiểu về xử lý va chạm khá một cách khá “ngây thơ” : đơn giản chỉ là kiểm tra hai đối tượng có giao nhau về mặt hình học hay không.

Lưu ý: Trước khi kiểm tra 2 vật có va chạm hay không, SV cũng cần lưu ý loại bỏ nhanh khả năng không có va chạm thông qua phân hoạch không gian.

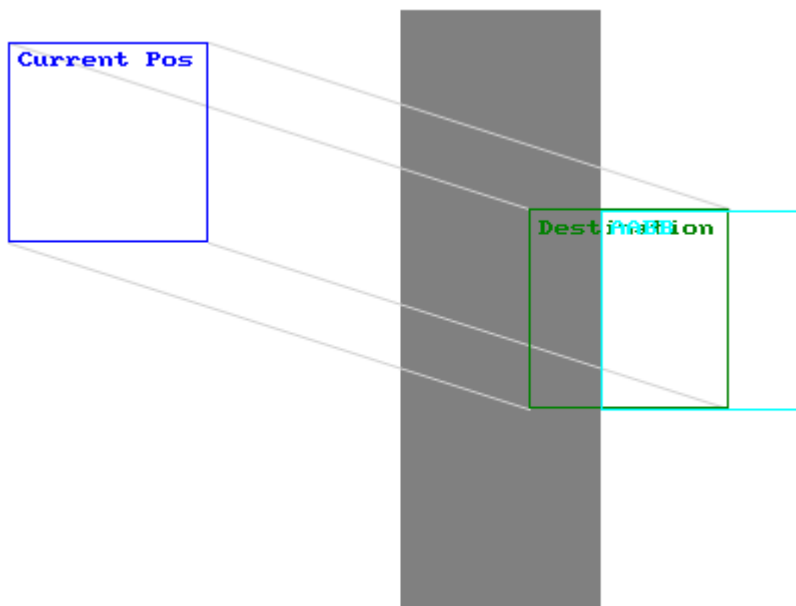
Ghi chú: Chúng tôi xem như bạn đọc đã hiểu thuật toán kiểm tra va chạm AABB cơ bản (kiểm tra hai hình chữ nhật có trực song song tọa độ có giao nhau hay không). Chương này cần chúng ta hiểu một ít kiến thức toán vector, nhưng sẽ không quá rắc rối. Chúng tôi cũng sẽ cung cấp ví dụ minh họa để bạn đọc tham khảo về sau.

7.1.2. Vấn đề đối với thuật toán “ngây thơ”

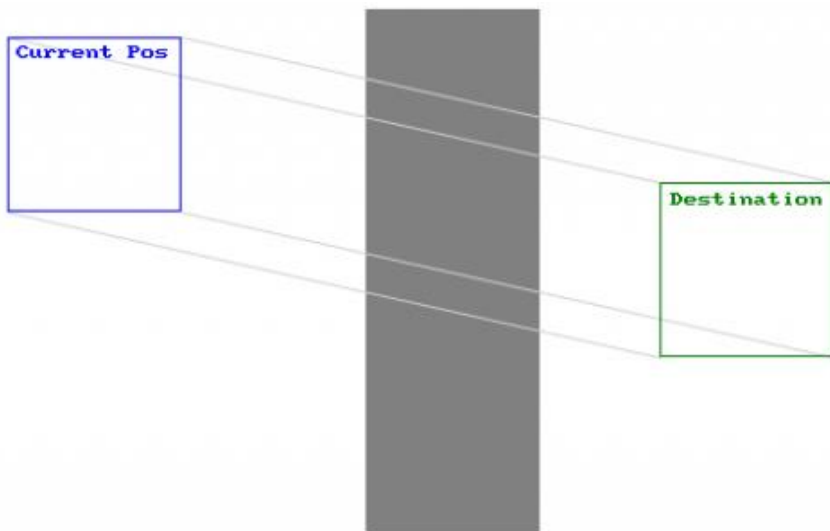
Thuật toán xử lý va chạm “ngây thơ” AABB gặp một số vấn đề cơ bản như 3 ví dụ dưới đây:



Ví dụ 1: Một va chạm AABB bình thường. Hộp màu xanh dương là vị trí của hộp tại điểm bắt đầu của frame. Hộp màu xanh lá cây là vị trí của hộp khi kết thúc frame. Hộp màu xanh nước biển là vị trí của hộp sau khi xét va chạm. Hộp màu xám là vật tĩnh (không dịch chuyển được) được dùng để kiểm tra va chạm. Và đây là kiểu va chạm thông thường. Hộp sẽ được dời tới điểm gần nhất mà không va chạm. Đây là kết quả được mong đợi của thuật toán va chạm AABB.



Ví dụ 2: Trường hợp va chạm khi vị trí đích nằm lệch qua phía bên kia của vật cản. Như ta thấy, vị trí sau khi tính toán va chạm nằm ở phía bên kia của vật cản, thay vì bị cản lại ở bên trái thì bỗng nhiên đối tượng bị kẹt lại ở phía bên phải. Theo logic thông thường, thì điều này là sai thực tế.



Ví dụ 3: Trường hợp va chạm khi vị trí đích không va chạm với vật cản. Thuật toán AABB sẽ không phát hiện ra va chạm và hộp sẽ tiếp tục di chuyển như không hề có va chạm.

Tại sao có thể xảy ra trường hợp như ví dụ 2 và 3 ở trên?

Đây là vấn đề thường gặp khi các đối tượng di chuyển *cực nhanh* hoặc do chương trình chạy *quá chậm*. Để tránh trường hợp trên, chúng ta cần có một chút dự đoán trước nơi vật sẽ đến giữa các frame. Khái niệm này gọi là “trượt” (swept)

7.2. Cài đặt thuật toán Swept AABB

Trong phần cài đặt thuật toán này, chúng ta sẽ quy ước rằng vị trí vật tại góc phía trên – bên trái của vật thể và chúng có chiều rộng cũng như chiều cao. Và bây giờ chúng ta sẽ nói về va chạm trượt (swept). Điều đầu tiên cần lưu ý là đến vận tốc.

Chú ý: Vận tốc của vật trong chương này là khoảng cách vật di chuyển trong 1 frame: có được bằng cách nhân vận tốc tính trên giây với khoảng thời gian giữa các frame (delta time)

Chúng ta sẽ định nghĩa hộp như sau:

```
// describes an axis-aligned rectangle with a velocity
struct Box
{
    // position of top-left corner
    float x, y;

    // dimensions
    float w, h;

    // velocity
    float vx, vy;
};
```

v_x , v_y là vận tốc; w và h là kích thước của vật.

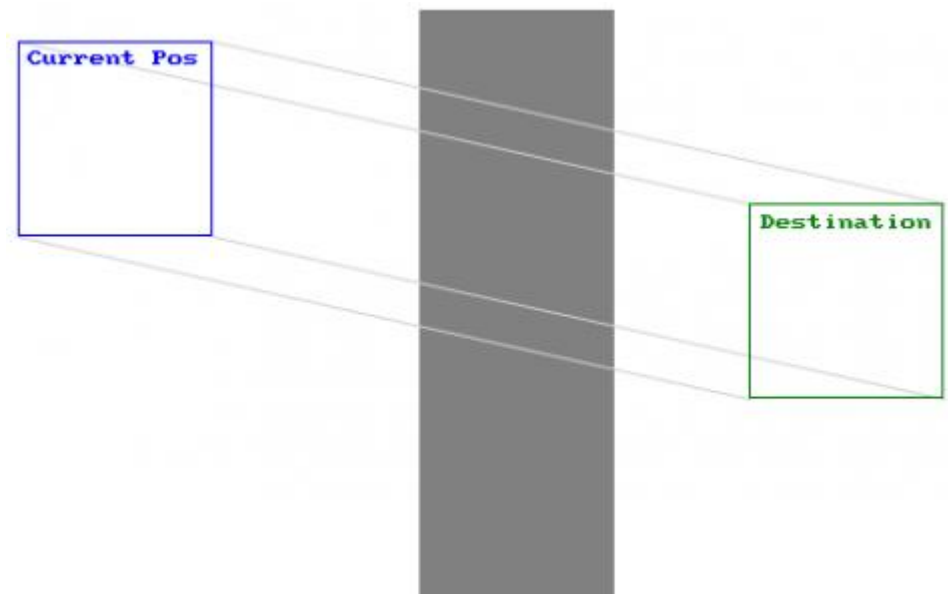
Hàm xét va chạm như sau:

```
float SweptAABB(Box b1, Box b2, float& normalx, float&
normaly)
```

Tham số đầu tiên là hộp di chuyển, tham số thứ 2 là vật cản. 2 tham số cuối cùng quy định vector pháp tuyến trên bề mặt tiếp xúc. Nó sẽ được dùng sau này khi chúng ta xử lý phản hồi va chạm.

Chú ý: vector pháp tuyến là hướng tiếp xúc trên bề mặt tiếp xúc của vật thể. Nó vuông góc với bề mặt tiếp xúc.

Giá trị trả về là 1 số thực có giá trị từ 0 đến 1 chỉ ra *thời điểm* khi nào va chạm xảy ra. Giá trị 0 chỉ điểm bắt đầu, giá trị 1 chỉ điểm kết thúc. Nếu nó trả về 1, chúng ta có thể kết luận không có va chạm xảy ra; nếu là 0.5 chẳng hạn, thì va chạm xảy ra ở điểm giữa đường đi. Chúng ta sẽ sử dụng nó sau trong việc phản hồi va chạm.



Bây giờ, để bắt đầu thuật toán, chúng ta cần tìm ra khoảng cách và thời gian nó va chạm được tính toán trên cả 2 trục x, y

```
float xInvEntry, yInvEntry;
float xInvExit, yInvExit;

// tìm khoảng cách giữa 2 vật thể ở cạnh gần và cạnh xa
if (b1.vx > 0.0f)
{
    xInvEntry = b2.x - (b1.x + b1.w);
    xInvExit = (b2.x + b2.w) - b1.x;
}
else
{
    xInvEntry = (b2.x + b2.w) - b1.x;
    xInvExit = b2.x - (b1.x + b1.w);
}

if (b1.vy > 0.0f)
{
    yInvEntry = b2.y - (b1.y + b1.h);
    yInvExit = (b2.y + b2.h) - b1.y;
```

```

    }
    else
    {
        yInvEntry = (b2.y + b2.h) - b1.y;
        yInvExit = b2.y - (b1.y + b1.h);
    }

```

$xInvEntry$ và $yInvEntry$ là khoảng cách gần nhất giữa 2 đối tượng. $xInvExit$ và $yInvExit$ là khoảng cách giữa 2 mặt xa nhất của vật thể. Bạn có thể hình dung như vật di chuyển được “bắn” xuyên qua vật cản, điểm bắt đầu va chạm là điểm viên đạn bắt đầu tiếp xúc, điểm kết thúc là vị trí nó bắt đầu rời khỏi vật cản ở mặt bên kia. Những giá trị đều được nghịch đảo thời gian đến khi nó va chạm đến vật cản. Chúng ta sẽ sử dụng những giá trị này để lấy tính toán vận tốc.

```

// find time of collision and time of leaving for
each axis (if statement is to prevent divide by zero)
float xEntry, yEntry;
float xExit, yExit;

if (b1.vx == 0.0f)
{
    xEntry = -std::numeric_limits<float>::infinity();
    xExit = std::numeric_limits<float>::infinity();
}
else
{
    xEntry = xInvEntry / b1.vx;
    xExit = xInvExit / b1.vx;
}

if (b1.vy == 0.0f)
{
    yEntry = -std::numeric_limits<float>::infinity();
    yExit = std::numeric_limits<float>::infinity();
}
else

```

```

{
    yEntry = yInvEntry / b1.vy;
    yExit = yInvExit / b1.vy;
}

```

Chúng ta sẽ chia `xEntry`, `yEntry`, `xExit` and `yExit` cho vận tốc của vật thể. Dĩ nhiên, nếu vận tốc trên 1 trục nào đó bằng 0 thì sẽ có lỗi chia cho 0. Chúng ta sẽ có các giá trị từ 0 đến 1 khi va chạm xảy ra ở mỗi trục. Tiếp theo, chúng ta sẽ tìm xem trục nào sẽ xảy ra va chạm đầu tiên

```

// find the earliest/latest times of collision
float entryTime = std::max(xEntry, yEntry);
float exitTime = std::min(xExit, yExit);

```

`entryTime` sẽ cho ta biết khi nào sẽ bắt đầu va chạm, `exitTime` sẽ cho ta biết khi nào kết thúc va chạm. Đây là những giá trị rất cần thiết, nhưng bây giờ, chúng ta cần tính toán xem va chạm có xảy ra hay không.

```

// if there was no collision
if (entryTime > exitTime || xEntry < 0.0f && yEntry < 0.0f || xEntry > 1.0f || yEntry > 1.0f)
{
    normalx = 0.0f;
    normaly = 0.0f;
    return 1.0f;
}

```

Câu lệnh `if` sẽ kiểm tra có va chạm hay không. Nếu thời gian va chạm không nằm từ 0 đến 1, thì nó sẽ không va chạm trong frame hiện tại. Hơn nữa, thời gian khi vật bắt đầu va chạm sẽ ko bao giờ xảy ra sau khi vật kết thúc va chạm, chúng ta cần kiểm tra, nếu sai, chúng ta có thể kết luận không có va chạm xảy ra. Trả về giá trị 1 cũng kết luận không có va chạm.

Nếu có va chạm, chúng ta sẽ tính vector pháp tuyến của bề mặt va chạm.

```

else // if there was a collision
{
    // calculate normal of collided surface

```

```

if (xEntry > yEntry)
{
    if (xInvEntry < 0.0f)
    {
        normalx = 1.0f;
        normaly = 0.0f;
    }
    else
    {
        normalx = -1.0f;
        normaly = 0.0f;
    }
}
else
{
    if (yInvEntry < 0.0f)
    {
        normalx = 0.0f;
        normaly = 1.0f;
    }
    else
    {
        normalx = 0.0f;
        normaly = -1.0f;
    }
}

// return the time of collision
return entryTime;
}

```

Vì vật thể chúng ta là hình hộp 4 mặt, do đó, vector pháp tuyến trên bề mặt va chạm chỉ có thể có 4 trường hợp (ứng với 4 mặt của đối tượng). Nó dễ dàng tính ra và cuối cùng trả về thời gian va bắt đầu va chạm.

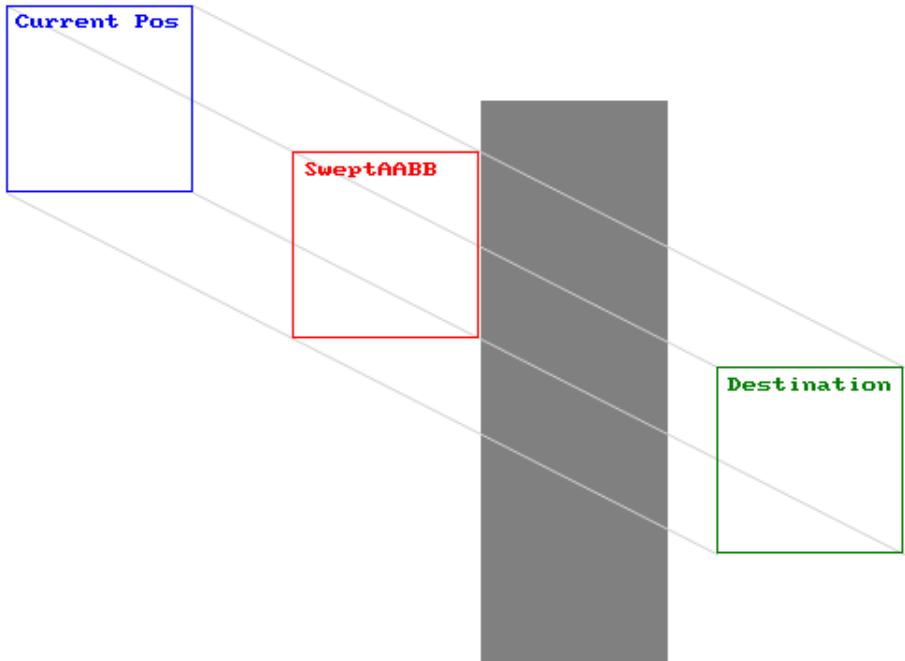
Đây là tất cả cho giả thuật AABB Swept nhưng nó cũng chưa đúng cho mọi trường hợp. Đó là vì chúng ta chưa cài đặt “giai đoạn rộng??”

(broadphase) (ở phần sau). Nhưng, bây giờ, bước tiếp theo, phản hồi va chạm.

7.3. Xử lý va chạm

Xử lý va chạm là cách mà chúng ta xác định lại vị trí của đối tượng sau khi phát hiện va chạm. Trước khi đi sâu vào những kiểu xử lý khác nhau, chúng ta cần xác định thời điểm xảy ra va chạm. Điều này khá dễ dàng khi chúng ta đã có hàm Swept AABB.

```
float normalx, normaly;  
float collisiontime = SweptAABB(box, block, out  
normalx, out normaly);  
box.x += box.vx * collisiontime;  
box.y += box.vy * collisiontime;  
  
float remainingtime = 1.0f - collisiontime;
```

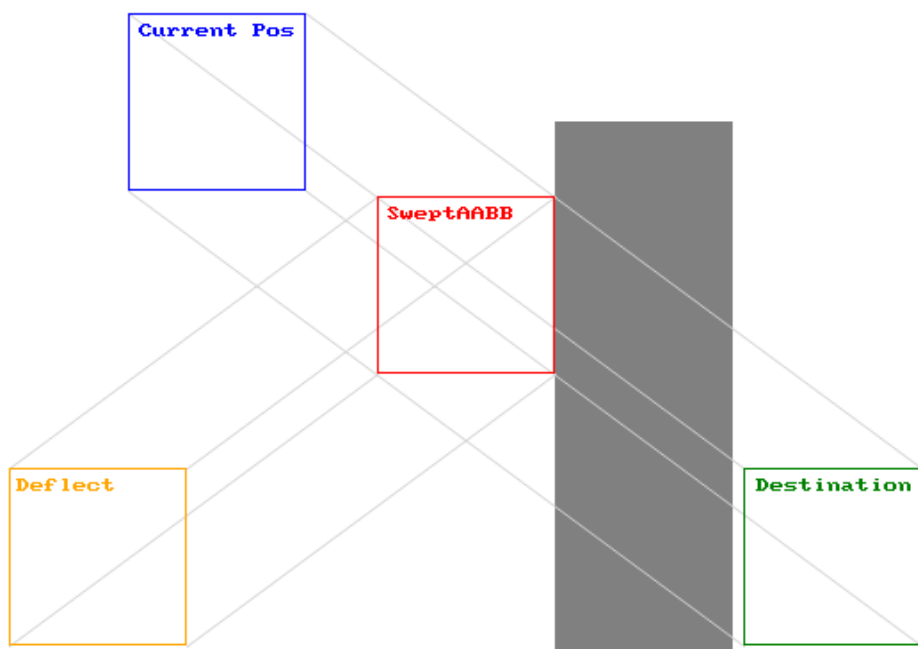


Chọn $1.0f - collisiontime$ sẽ trả về thời gian giữa từng frame (giá trị nằm giữa 0 và 1). Đó là thời gian đủ để biểu diễn chính xác va chạm và

thực hiện một số công việc. Nhưng nếu bạn cố gắng di chuyển chéo khối hộp vào trong đối tượng (“áp chặt vào tường”) và bạn phát hiện được rằng bạn không thể di chuyển. Khối hộp sẽ không thể di chuyển thêm nữa. Chính lúc này, cách xử lý va chạm có thể giải quyết.

7.3.1. Phản xạ

Đây là cách chung nhất mà những game như Pong thực hiện khi quả bóng va vào đối tượng.



Bạn cần chú ý khi đối tượng va chạm, đối tượng di chuyển vẫn còn sót lại một phần vận tốc di chuyển. Bạn chỉ cần sử dụng phần sót lại của vận tốc đó để di chuyển chúng theo hướng ngược lại, tạo ra hiệu ứng nảy.

```
// deflect
box.vx *= remainingtime;
box.vy *= remainingtime;
if (abs(normalx) > 0.0001f)
    box.vx = -box.vx;
```

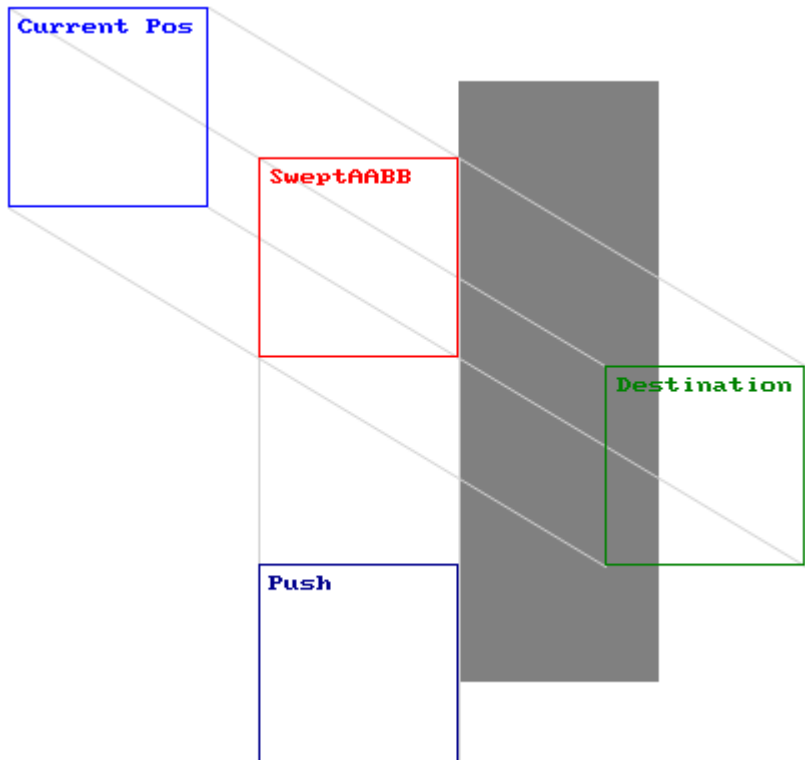


```
if (abs(normaly) > 0.0001f)
    box.vy = -box.vy;
```

Đầu tiên, chúng ta giảm vận tốc của chúng bằng khoảng thời gian va chạm. Sau đó, đảo ngược vận tốc của chúng theo hướng va chạm. Thật đơn giản.

7.3.2. Đẩy

Đẩy là kiểu khá truyền thống trong khái niệm ô-mat-tơ, trong trường hợp này, bạn sẽ di chuyển dọc theo tường, theo chiều dài, trượt lên bề mặt của tường.



```
// push
float magnitude = sqrt((box.vx * box.vx + box.vy *
box.vy)) * remainingtime;
float dotprod = box.vx * normaly + box.vy * normalx;
```

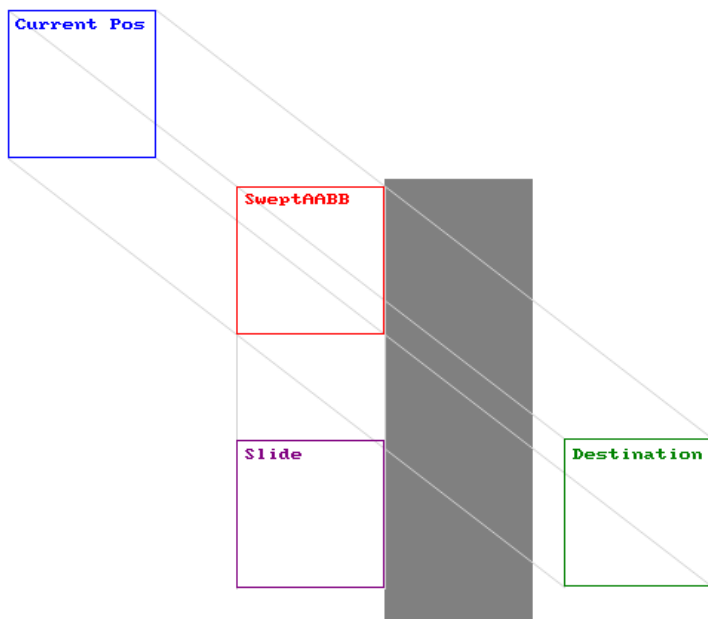
```
if (dotprod > 0.0f)
    dotprod = 1.0f;
else if (dotprod < 0.0f)
    dotprod = -1.0f;
NewBox.vx = dotprod * normaly * magnitude;
NewBox.vy = dotprod * normalx * magnitude;
```

Nó tái sử dụng lại phần vận tốc thừa và đẩy nó theo hướng song song với cạnh va chạm. Bước đầu tiên là tính toán được độ lớn vận tốc (theo Nguyên Lí Py-ta-go). Bước tiếp theo sử dụng tích vô hướng của vận tốc với véc tơ pháp tuyến của bề mặt va chạm. Sau đó, tối giản kết quả vô hướng đó (bởi vì chúng ta sẽ dùng nó để tính khoảng cách đạt được). Bước cuối cùng là nhân tích vô hướng đã được tối giản với độ lớn véc tơ pháp tuyến và độ lớn vận tốc thừa.

Thay cho việc tối giản hóa tích vô hướng, bạn có thể tối giản hóa vận tốc sau khi tính toán độ lớn.

7.3.3. Trượt

Vấn đề với kỹ thuật đẩy là đối tượng có thể bị đẩy nhanh hơn mong muốn. Một cách khác thực tế hơn – đó là trượt.



Nó sử dụng véc tơ chiếu để tìm ra vị trí tương đương trên cạnh. Đó là một cách tiếp cận đơn giản hơn so với phương pháp đẩy.

```
// slide
float dotprod = (box.vx * normaly + box.vy * normalx)
* remainingtime;
NewBox.vx = dotprod * normaly;
NewBox.vy = dotprod * normalx;
```

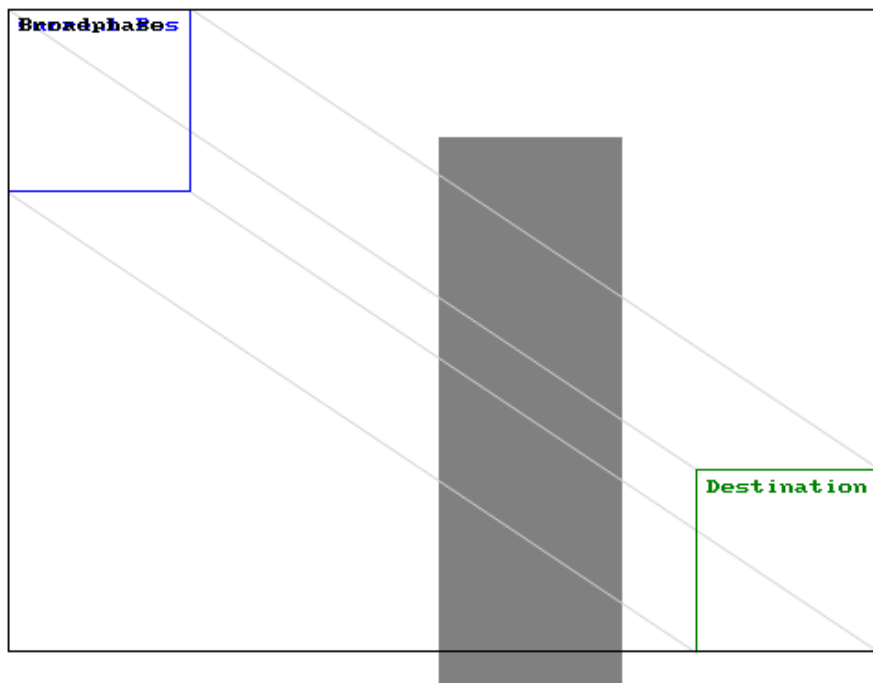
Điểm đầu tiên cần nhớ đó là đảo véc tơ pháp tuyến ngược (tráo x và y cho nhau). Chúng ta tính tích vô hướng, nhân nó với độ lớn và cuối cùng nhân với véc tơ pháp tuyến bị tráo. Và chúng ta có được vận tốc chiếu.

7.4. Các vấn đề khác

7.4.1. Kiểm tra diện rộng (Broad-Phasing)

Thuật toán swept AABB chạy rất nhanh, nhưng vì nếu có nhiều đối tượng chạy cùng lúc, hiệu suất sẽ bị giảm sút nhanh. Cách xử lý hiệu quả vấn đề này gọi là Broad-phasing. Với Broad-phasing, bạn có thể

làm chúng nhanh hơn, ít cần chính xác để có thể xác định được nếu không có va chạm. Một vài kĩ thuật cần để thực hiện nó là (kiểm tra theo hình tròn) nhưng, bởi vì đối tượng của chúng ta là hộp vuông góc với các trục tọa độ, nên khuyến cáo dùng là hình hộp.



Khung viền đen xung quanh chỉ ra vùng diện rộng của chúng ta. Với hình hộp, chúng ta có thể đơn giản chỉ cần kiểm tra AABB đơn giản để kiểm tra có va chạm hay không. Hãy nhìn vào bức ảnh, đảm bảo là những đối tượng không nằm trong vùng diện rộng thì chắc chắn sẽ không va chạm với đối tượng của chúng ta. Nhưng cũng không thể khẳng định những đối tượng nằm trong vùng diện rộng là có va chạm. Nếu các đối tượng có va chạm với vùng diện rộng, ta biết rằng ta sẽ phải dùng Swept AABB để kiểm tra kĩ hơn.

```
Box GetSweptBroadphaseBox(Box b)
{
    Box broadphasebox;
    broadphasebox.x = b.vx > 0 ? b.x : b.x + b.vx;
```

```

    broadphasebox.y = b.vy > 0 ? b.y : b.y + b.vy;
    broadphasebox.w = b.vx > 0 ? b.vx + b.w : b.w - b.vx;
    broadphasebox.h = b.vy > 0 ? b.vy + b.h : b.h - b.vy;

    return broadphasebox;
}

```

Bước đầu tiên là tính toán được vùng diện rộng. Thật tồi tệ, những gì cần làm là thêm vận tốc vào các cạnh (dựa trên hướng của vận tốc). Và bây giờ chúng ta phải viết một hàm kiểm tra AABB cực kì đơn giản

```

bool AABBCheck(Box b1, Box b2)
{
    return !(b1.x + b1.w < b2.x || b1.x > b2.x + b2.w ||
    b1.y + b1.h < b2.y || b1.y > b2.y + b2.h);
}

```

Đây là hàm trả về **true** nếu có va chạm xảy ra, và ngược lại.

Tiếp theo, chúng ta có thể đặt các thành phần vùng diện rộng của chúng ta:

```

// box is the moving box
// block is the static box

Box broadphasebox = GetSweptBroadphaseBox(box);
if (AABBCheck(broadphasebox, block))
{
    float normalx, normaly;
    float collisiontime = SweptAABB(box,
block, out normalx, out normaly);
    box.x += box.vx * collisiontime;
    box.y += box.vy * collisiontime;

    if (collisiontime < 1.0f)
    {
        // perform response here
    }
}

```

7.4.2. Giới hạn của thuật toán

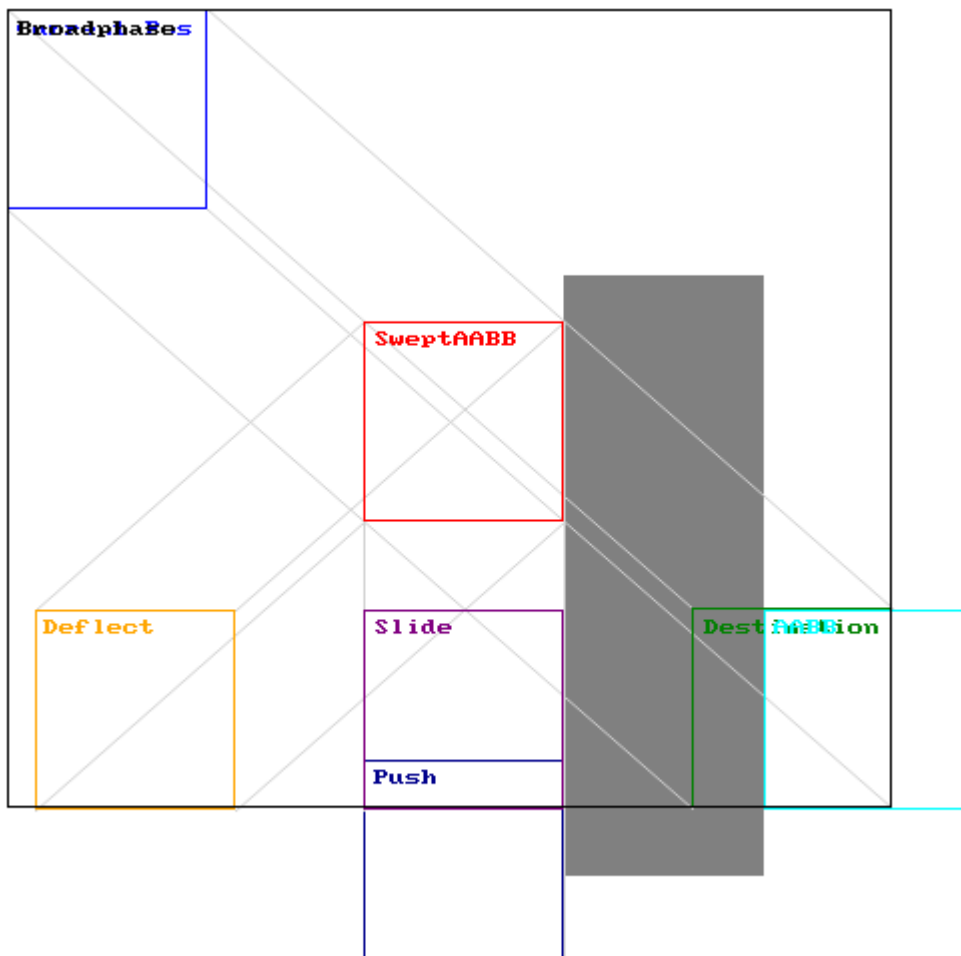
Thuật toán này có một số giới hạn mà người sử dụng cần phải nắm rõ:

- Không áp dụng được với những thay đổi về kích thước (nghĩa là có một hình hộp thay đổi kích thước trong một frame).
- Chỉ cho phép đối với những di chuyển theo đường thẳng. Nếu hình hộp di chuyển theo vòng cung (hình tròn), bạn sẽ không thể biết được đâu là phần mở rộng của vòng cung.
- Chỉ cho phép một hình hộp di chuyển (nghĩa là nếu có hai hình hộp di chuyển dọc theo nhau và va chạm).

Lưu ý: Để áp dụng đối với 2 đối tượng di chuyển, ta áp dụng định luật chuyển động tương đối của vật lý. Giữ nguyên một đối tượng xem như đứng yên và điều chỉnh lại vận tốc của đối tượng di chuyển.

Thuật toán này chỉ áp dụng được đối với các hình chữ nhật có cạnh song song trục tọa độ. Ta không thể xoay các hình chữ nhật. Điều này khá rõ ràng bởi vì tên của thuật toán này là AABB. Nếu ta đã hiểu rõ AABB thì ta có thể xem thêm các thuật toán dùng trong các trường hợp xử lý va chạm tổng quát và phức tạp hơn như SAT hay GJK

Thuật toán này có dùng cho 3D được không? Thật may mắn, nó dễ dàng để có thể chuyển qua 3D chỉ cần thêm trục Z vào là mọi thứ vẫn sẽ hoạt động tốt. Ta giữ nó ở 2D để dễ hiểu hơn mà thôi.



CHƯƠNG 8

BÀI TOÁN PHÂN HOẠCH KHÔNG GIAN

8.1. Tại sao cần phải phân hoạch không gian?

Đối với những game nhỏ, những game có ít thể giới nằm trong một màn hình máy tính thì thông thường trong vòng lặp game, chúng ta chỉ cần duyệt qua tất cả các đối tượng rồi gọi các xử lý (Update, Draw, trạng thái, kiểm tra va chạm ...).

Ví dụ: trong một map game có 100 đối tượng là các hình ảnh, tường, đất, đá,...thì việc duyệt từng phần tử để thực hiện các thao tác là khá dễ dàng, không tốn nhiều công xử lý.

```
for (int i=0;i<n;i++)//n = 100
{
    GameObject *o = currentMap->getObject(i);

    Renderer::getInstance()->render(o);
}
```

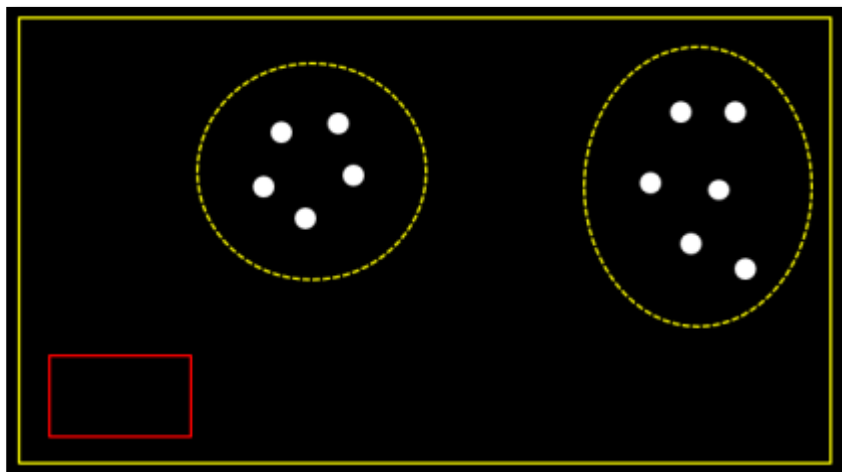
Và mọi thứ vẫn chạy ổn định, bình thường, không có chuyện gì xảy ra cả. Tuy nhiên, bạn thử tưởng tượng, khi chúng ta làm một game cực kỳ lớn, có cả ngàn đối tượng trong một map, thì vấn đề gì sẽ xảy ra khi chúng ta vẫn duyệt từ phần tử đầu tiên đến phần tử cuối cùng của map và tiến hành thực hiện các xử lý thì sẽ tốn rất nhiều chi phí, và cũng có thể có lỗi xảy ra, game chúng ta viết chạy không mượt, máy lại nóng, ...

Giả sử bạn đang xây dựng một game nho nhỏ, trong đó có tầm 4000 đối tượng di chuyển theo nhiều hướng khác nhau trong màn hình cũng như ở toàn bộ map và chúng va chạm hỗn loạn với nhau (giống như có rất nhiều quả bóng cùng di chuyển và chạm vào nhau vậy). Bạn sẽ giải quyết bài toán này như thế nào? Bạn có thể duyệt toàn bộ lần lượt 4000 đối tượng, vẽ và xét va chạm với 3999 đối tượng còn lại. Điều này thực sự không hiệu quả chút nào, tài nguyên của bạn bị hao tốn một cách lãng phí.

Vậy chúng ta sẽ giải quyết vấn đề này như thế nào? Phân hoạch không gian chính là một trong những phương pháp tối ưu nhất để giải quyết điều này.

Thực ra, thì phương pháp này được sử dụng rất phổ biến trong nhiều lĩnh vực khác nhau, cũng như trong cuộc sống hằng ngày. Có thể nói ý tưởng của phương pháp này là “chia để trị”, càng chia nhỏ ra chúng ta càng dễ quản lý. Ví dụ như ngày xưa lúc thực dân Pháp xâm lược nước ta và chia nước ta làm 3 miền(Bắc, Trung, Nam) cũng chỉ nhằm mục đích để cai trị, quản lý, thay vì quản lý toàn bộ cả nước, thì chúng quản lý từng miền. Công việc tổ chức và xử lý trở nên dễ dàng hơn và ít “tốn kém” công sức hơn.

Mục đích của việc phân hoạch không gian là để tiết kiệm chi phí cho các thao tác vẽ, update những đối tượng không có khả năng xuất hiện trên màn hình bằng việc chia thế giới trong game ra thành nhiều vùng không gian con, và chúng ta sẽ chỉ xử lý những đối tượng nằm trong vùng không gian con nào gần với viewport (vùng không gian mà người dùng để nhìn thấy thế giới game).



Hình 8.1. Minh họa việc phân hoạch không gian

8.2. Một số tiếp cận phân hoạch không gian trong game 2D

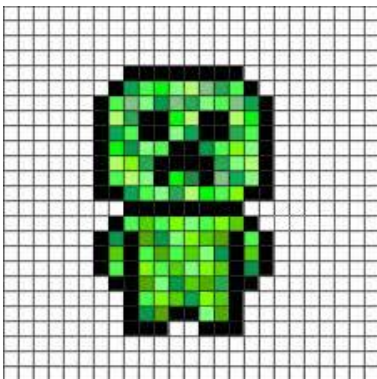
Có nhiều cách phân chia không gian trong game 2D, nhưng những cách tiêu biểu nhất là:

- Phân chia theo nhóm
- Phân chia bằng grid
- Cây nhị phân
- Cây tứ phân

Phân chia theo nhóm: nhóm các phần tử có một hay một số đặc điểm chung nào đó vào một hình chữ nhật(hình tròn , hình vuông hay một hình bất kỳ) cố định về kích thước và vị trí, ta sẽ dễ dàng quản lý những đối tượng con trong đó; biết chúng nằm ở đâu trong thế giới game, biết vị trí của chúng so với khung của viewport, có thể xác định được nhóm nào có thể va chạm với nhau.

Ưu điểm: dễ thực hiện.

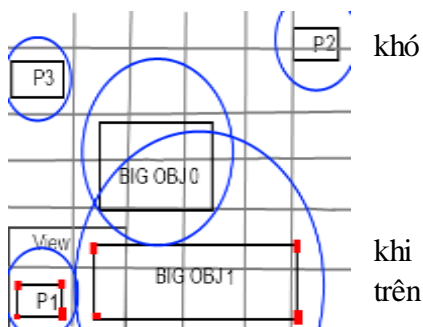
Nhược điểm: không tối ưu về phần xử lý các đối tượng nằm trong và ngoài viewport; khi viewport chạm đến một phần nhỏ hình chữ nhật chứa các đối tượng con, thì chúng ta cũng phải xử lý các đối tượng con trong đó mà có thể chúng không được hiển thị trên viewport, gây ra sự lãng phí tài nguyên.



Phân chia bằng grid: như ta đã biết, hệ thống lưới grid rất phổ biến trong phân chia không gian; không gian chính được chia thành nhiều không gian phụ là hình chữ nhật nhỏ hơn nó có cùng tính chất, kích thước mỗi ô trong grid cố định sẵn, khi viewport va chạm với mỗi ô trong grid, thì các đối tượng con trong mỗi ô đó được xử lý.

Ưu điểm: dễ thực hiện, phân chia không gian tương đối tốt.

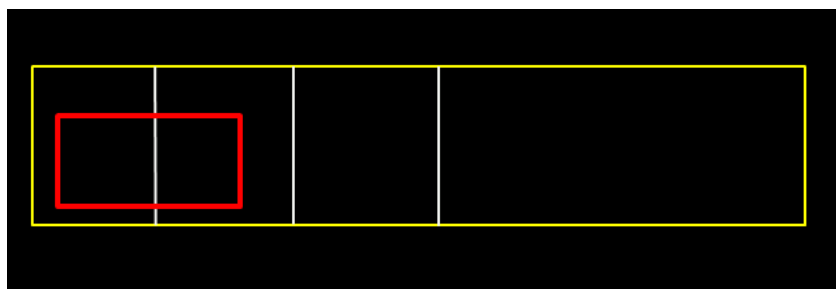
Nhược điểm: với không gian quá lớn, hoặc quá phức tạp, chúng ta sẽ rất xác định được kích thước mỗi ô trong grid, dẫn đến hai trường hợp là chúng ta xét quá kĩ đối tượng con trong mỗi ô (chia ô grid quá nhỏ), hoặc chúng ta sẽ lãng phí tài nguyên xét đến các đối tượng không thể hiện viewport (chia ô grid quá lớn).



Cây nhị phân: đối với không gian kích thước chiều ngang quá lớn, mà kích thước chiều dọc tương đối nhỏ, chúng ta có thể dùng kĩ thuật cây nhị phân để phân chia không gian. Kĩ thuật này chia không gian thành các ô theo chiều ngang, và mỗi ô không được lớn hơn viewport quá nhiều. Trong mỗi thời điểm viewport va chạm với không quá 2 ô.

Ưu điểm: dễ thực hiện, phân chia không gian tương đối tốt đối với không gian chiều ngang quá lớn.

Nhược điểm: vẫn chưa tối ưu về mặt tài nguyên khi xử lí, không áp dụng được với mọi thể giới game.



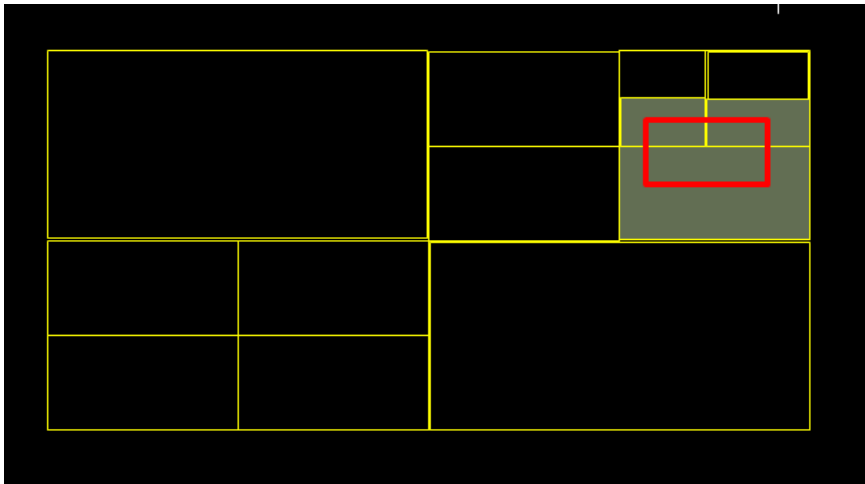
Hình 8.2. Minh họa phân chia không gian bằng cây nhị phân

Cây tứ phân: kĩ thuật phân chia không gian bằng cây tứ phân là một kĩ thuật tương đối tối ưu với không gian game. Không gian chính được

chia thành bốn không gian con nhỏ hơn được chứa trong không gian chính đó, không gian được chia ra đến khi đạt được độ nhỏ do người lập trình quy định(thông thường là lớn hơn viewport một chút) hoặc là không có đối tượng bên trong ô đó, mỗi ô gọi là một node của cây. Không gian chính phải là hình vuông, bốn không gian nhỏ hơn chứa trong nó cũng là hình vuông.

Ưu điểm: tương đối tối ưu về mặt xử lý.

Nhược điểm: khó thực hiện, triển khai nhiều bước phức tạp. Nhiều trường hợp xảy ra nên cần phải xét hết tất cả các trường hợp, tạm chấp nhận bỏ qua các trường hợp node va chạm với viewport nhưng mà va chạm rất ít, xem như không có va chạm.



Hình 8.3. Minh họa kỹ thuật phân chia không gian bằng cây tứ phân

8.3. Cài đặt kỹ thuật Quad tree

Ý tưởng chính:

Quá trình làm game của chúng ta sẽ chia làm 2 quá trình: quá trình tiền xử lý (hay chuẩn bị) và quá trình chạy game.

Quá trình tiền xử lý sẽ làm bên Map Editor sẽ được chia thành các công đoạn phân hoạch:

- Build cây
- Gắn đối tượng vào các node

- Lưu cây

Quá trình chạy game cũng chia thành 2 giai đoạn thực hiện việc phân hoạch:

- Load file có sẵn để tạo cây mới trong game Init
- Trong Game Loop, chúng ta sẽ lấy ra những node lá có va chạm với viewport để xử lý những đối tượng game object trong node đó.

Chúng ta sẽ bắt đầu từng công đoạn.

8.3.1. Build Tree

Công đoạn này sẽ mất khá nhiều thời gian xử lý nếu như kích thước map quá lớn và chứa nhiều đối tượng, vì thế chúng ta sẽ build quadtree trong giai đoạn tiền xử lý (Map Editor).

Trước tiên ta sẽ nói một chút về cấu trúc dữ liệu của quadtree.

```
class QNode
{
    QNode *tl, *tr, *bl, *br;
    int nodeID;
    List<CTreeObject> objects;
}
```

```
class CTreeObject
{
    int x1,y1,x2,y2;
    GameObject *target;
}
```

Quadtree chúng ta dùng trong game cũng có cấu trúc dạng cây với nhiều node, mỗi node có 4 node con. Vì thế chúng ta sẽ xây

dựng cấu trúc dữ liệu như trên, với QNode là kiểu dữ liệu của các node trong Quadtree. Trong QNode có chứa con trỏ của 4 node con của nó, một cái id để xác định node, và danh sách các đối tượng CTreeObject.

Đối tượng CTreeObject chứa các thông tin để xác định hình vuông bao quanh node và 1 con trỏ đến đối tượng GameObject.

Tại sao lại phải tạo đối tượng CTreeObject để tham chiếu tới đối tượng GameObject mà không dùng trực tiếp GameObject? Là vì để phục vụ cho thao tác xén hình sau này. :D

Tiếp theo, ta cần tạo node gốc bằng hình vuông bao thế giới game. Tại sao là hình vuông mà không phải là hình chữ nhật? Lý do đơn giản, nếu chúng ta dùng hình chữ nhật, thông thường thế giới game có dạng hình chữ nhật dài, nên càng chia nhiều node con, thì các hình chữ nhật con càng bị hẹp lại, gây khó khăn trong thao tác xử lý sau này.

Sau đó ta sẽ dùng đệ quy để build tree

```
void build(QNode *n)
{
    //kiểm tra xem có được phép chia node nữa không
    //và node đó có chứa đối tượng
    n->lt=new QNode(n->x0,n->y0,n->x1/2,n->y1/2);
    ...
    //duyet danh sách các object o ở trong n->objects
    clip(o,n->lt);
    clip(o,n->rt);
    ...
    delete n->objects;
    build(n->lt);
    ...
}
```

Đối số truyền vào đầu tiên của hàm build này chính là node gốc.

Đầu tiên, nó sẽ kiểm tra xem node truyền vào có được phép chia tiếp hay là không? Điều kiện để 1 node có thể được chia tiếp là

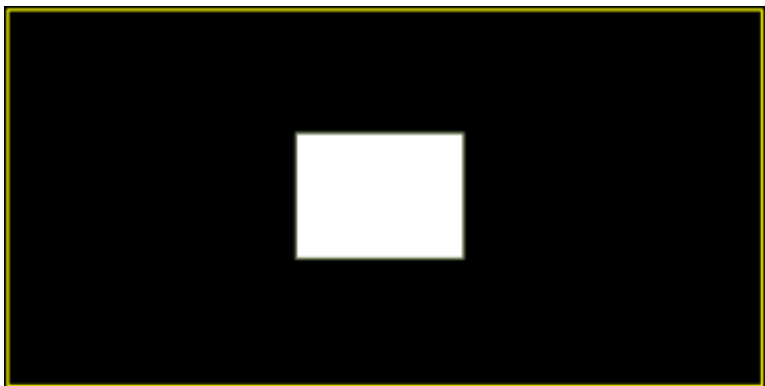
hình vuông bao node đó phải có kích thước lớn hơn hoặc bằng kích thước quy định sẵn (thường là lớn hơn viewport 1 tỷ) , thứ 2 là node đó phải có chứa đối tượng , đối tượng ở đây có là Ctreeobjec chứ không phải là GameObject. Và đây cũng chính là điều kiện dừng của hàm đệ quy này.

Nếu kiểm tra thỏa tất cả các điều kiện, ta sẽ tiến hành chia node truyền vào thành 4 node con, 4 node này được khởi tạo dựa vào hình chữ nhật và id của node cha.

Công việc tiếp theo sau khi đã có 4 node con đó là xén những đối tượng CTreeObject trong node truyền vào vào từng node con.Sau đó ta sẽ xóa các đối tượng trong node cha. Và gọi lại đệ quy cho 4 node con.

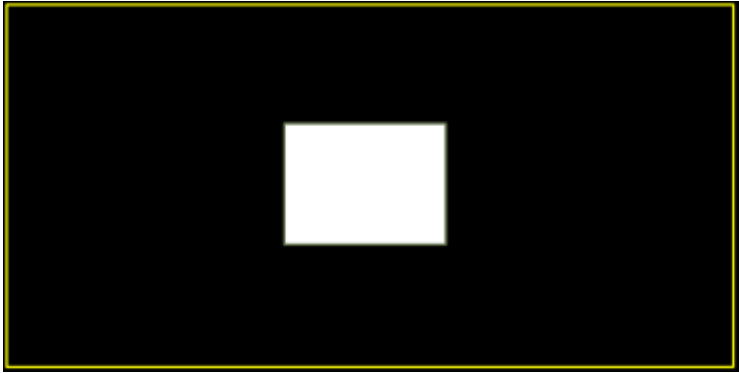
Thao tác xén này thực ra rất đơn giản, có nghĩ là chúng ta sẽ chia cái hình chữ nhật của đối tượng CTreeObject đó thành những phần nhỏ, mỗi phần nhỏ đó sẽ được add vào danh sách objects của node con chứa nó. Tất nhiên là những phần bị xén đó cũng cùng tham chiếu đến 1 GameObject duy nhất. Điều này có nghĩa là chúng ta sẽ có nhiều node chứa cùng 1 GameObject, chúng ta phải lưu ý điều này, vì nếu không xử lý chúng thì sẽ gây sai sót trong quá trình update.

Vậy tại sao chúng ta cần phải xén vào như thế? Mục đích của chúng ta là xác định xem cái hình chữ nhật được xén vào đó có đủ lớn để ta xem nó như một đối tượng không. Nếu như nó quá nhỏ, ta có thể bỏ qua nó.



Hình 8.4. Minh họa quá trình xén hình chữ nhật

Giả sử chúng ta có 1 đối tượng như hình trên. Sau khi thực hiện gọi hàm build, ta sẽ được



Hình 8.5. Minh họa sau khi thực hiện hàm build

Các điểm cần lưu ý khi build quadtree:

- Thứ nhất, chỉ có node lá mới được chứa đối tượng, những node không phải là node lá thì không được chứa đối tượng.
- Thứ hai, đối với những đối tượng động thì ta sẽ tạo cho nó 1 cái hình bao quanh vùng chuyển động đó và xử lý bình thường như đối tượng tĩnh. Còn với những đối tượng di chuyển xuyên suốt map (như mario) thì ta không cho nó vào quadtree mà xử lý riêng.
- Cuối cùng, nếu trong game có quá nhiều đối tượng di chuyển khắp nơi thì sao? Tốt nhất là ta đừng nên thiết kế quá nhiều những đối tượng như thế :D

8.3.2. Lưu Quadtree vào file

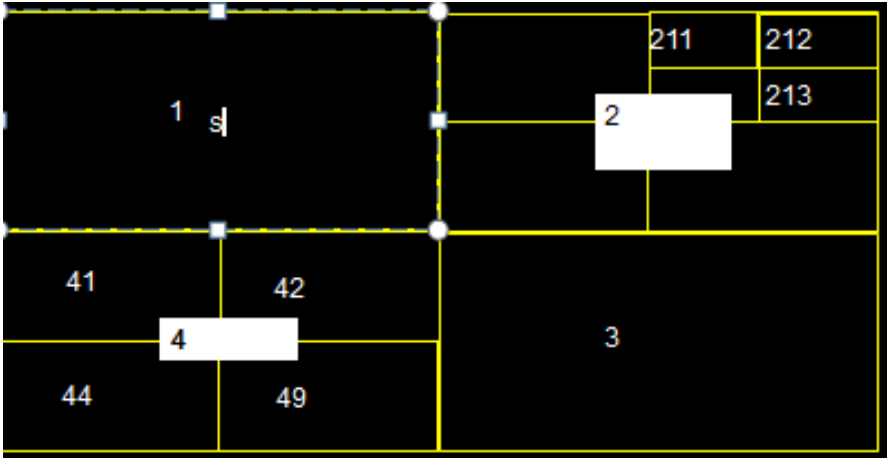
Một vấn đề lớn đó là sau khi đã build xong quadtree trong mapeditor rồi, thì làm sao để lưu lại để mà load vào game khi mà chúng toàn là con trỏ?

NodeID chính là giải pháp. NodeID chính là con số đại diện cho mỗi node. Vậy thì làm thế nào để xác định được node nào là con của node nào, hay là cha của node nào? Để làm việc này cũng khá dễ, chúng ta

có thể sử dụng bit hoặc tính toán thông thường qua các phép toán cộng trừ nhân chia v...v...

1 ví dụ đơn giản ta sử dụng phép nhân với số 10, giả sử node gốc của ta có id là 0 , ta sẽ có 4 node con có id lần lượt là 1,2,3,4 . node có id 2 có 4 node con, ta sẽ đặt id cho 4 node con đó là $20 \times 10 + 1 = 21$, 22 , 23, 24 .v.v rất đơn giản đúng không(cũng có thể bạn nhân cho 8).

Dưới đây là một ví dụ khác về cách đặt ID cho node.



Hình 8.6. Minh họa một phương pháp cài đặt ID cho node

Sau đó ta sẽ lưu quadtree vào file. Thông tin ghi ra file bao gồm NodeID , hình vuông bao quanh node và danh sách các đối tượng.

Có thể trông nó sẽ giống như thế này

```
0 ltx,lty      rbx,rby      id1,id2,id3,...(danh sách id của các
GameObject)
16 ltx,lty      rbx,rby
32 ltx,lty      rbx,rby
48 ltx,lty      rbx,rby      id4,id5,id6,...
...
```

Trong đó, cột đầu tiên là NodeID , 4 cột tiếp theo chứa thông tin về vị trí và kích thước của hình vuông bao của node . và những số còn lại chính là danh sách những đối tượng chứa trong node.

8.3.3. Load QuadTree vào game

Trong phần trước, dùng map editor tạo map chúng ta có hai file.

Một file lưu id và thuộc tính(vị trí, kích thước) của node và id của những object có trong node đó.

Một file lưu id của game object và thuộc tính(loại object, vị trí, kích thước, vùng di chuyển) của game object đó.

Trong game Init chúng ta xây dựng cây tứ phân như sau:

Có hai giai đoạn, giai đoạn một: tạo tất cả các node từ file; giai đoạn hai: liên kết các node thành một cây hoàn chỉnh.

Giai đoạn một:

Chúng ta sẽ tạo các node từ dữ liệu file load lên, và add các node đó vào 1 cấu trúc STL map với key là NodeID còn value là QNode* để tiện cho bước liên kết các node với nhau. Vd:

```
map<int, QNode*> mapQuadtree;
```

Và để thêm thuận tiện trong việc tạo mapQuadtree, ta cũng nên dùng cấu trúc map để chứa những đối tượng được tạo từ file GameObjet load lên.

```
map<int, GameObject*> mapGameObject;
```

Giai đoạn hai:

Chúng ta sẽ dựa vào NodeID để liên kết các node lại với nhau để tạo thành cây hoàn chỉnh. Đầu tiên, ta sẽ duyệt tất cả các node trong map đã tạo ở giai đoạn 1. Sau đó, ở mỗi node ta sẽ :

- Tính toán những NodeID của những node con của node hiện tại. Cách tính toán phụ thuộc vào cách mà ta chia node ở bước build quadtree trong quá trình tiền xử lý. Giả sử ta sử dụng cách tính id * 10 như ví dụ phía trên có nói. Ví dụ node hiện tại có id là 9 , ngay lập tức ta sẽ có được id của 4 node con lần lượt là 91,92,93,94.
- Nhờ vào cấu trúc map , ta có thể dễ dàng lấy ra những node có NodeID bất kỳ. Sau khi thực hiện bước tính toán trên, ta sẽ lấy ra được các node con của node hiện tại. Nếu như kết quả trả về

null (nghĩa là trong cấu trúc map của chúng ta không tồn tại giá trị key trùng với NodeID đó) thì có nghĩa đó là node lá. Nếu kết quả trả về khác null , thì ta trở chúng lại với nhau.

Đây là đoạn code ví dụ:

```
map<int , QNode*>::iterator it;
for (it = mapQuadtree.begin(); it!=mapQuadtree.end();it++)
{
    if(it->first == 0) // Node gốc
        quadtree->setRootNode(it->second);
    else //ko phải root
    {
        int p = it->second->getParentID(); //id của node cha
        int child = it->first%8; //
        QNode* pNode = mapQuadtree[p]; //con tro toi node cha
        switch(child)
        {
            case 1:
                pNode->_tl = it->second;
                break;
            case 2:
                pNode->_tr = it->second;
                break;
            case 3:
                pNode->_bl = it->second;
                break;
            case 4:
                pNode->_br = it->second;
                break;
        }
    }
}
```

}

Cách làm của mình có hơi khác một chút, nhưng vẫn đưa ra cùng một kết quả với cách mình đã nói trước đó. Nghĩa là thay vì ta tìm những node con của nó rồi trở vào chúng, thì ta cũng có thể tìm node cha của nó và cho nó trở vào. Hai cách này so về hiệu năng thì không khác nhau mấy.

8.3.4. Lấy danh sách những đối tượng trong màn hình

Sau khi đã có Quadtree, bước cuối cùng của quá trình chính là lấy ra danh sách những đối tượng GameObject có khả năng va chạm và nằm trong hay gần viewport. Sai lầm thường gặp của mọi người tới bước này là dùng cách duyệt tuần tự để kiểm tra xem viewport tiếp xúc với những node nào. Như vậy thì quả là hơi lãng phí công sức của chúng ta khi dựng nên một cấu trúc cây như thế. Vậy phải làm như thế nào cho đúng, cho tối ưu?

Chúng ta sẽ sử dụng cùng một phương pháp với việc xén object vào node.

Ở đây, ta sẽ tạo mới một đối tượng CTreeObject với hình chữ nhật bao chính bằng kích thước và vị trí của viewport.

Sau đó, ta thực hiện thao tác xén đối tượng đó vào từ node gốc, cho đến node lá. Và danh sách những đối tượng GameObject trong node lá đó chính là kết quả cuối cùng mà chúng ta cần lấy ra.

Ví dụ:

```
void CQuadtree::ListObjectInViewPort(CRect viewport, QNode* node)
{
    if(node == _rootNode) //Nếu đối số truyền vào là node gốc, thì clear list
        _listObjectsInViewPort.clear();
    //nếu đối số truyền vào là node lá thì đưa ra list
    if(!node->_tl)
    {
        if( node->getBound().intersectsRect(viewport) == true)
```

```

        if(node->getListGameObjects().size() > 0)
        {

            //Giải quyết việc trùng lặp các object và đưa
            //tất cả vào list

        }
    }
    else //Nếu chưa phải là node lá, thì tiếp tục gọi đệ quy
    {
        if(node->_tl->getBound().intersectsRect(viewport))
            ListObjectInViewPort(viewport,node->_tl);
        if(node->_tr->getBound().intersectsRect(viewport))
            ListObjectInViewPort(viewport,node->_tr);
        if(node->_bl->getBound().intersectsRect(viewport))
            ListObjectInViewPort(viewport,node->_bl);
        if(node->_br->getBound().intersectsRect(viewport))
            ListObjectInViewPort(viewport,node->_br);
    }
}

```

Danh sách này sau khi loại bỏ những đối tượng bị trùng lặp , do quá trình xén những object nằm đè lên nhiều node sẽ được dùng để Update , Draw .

Danh sách này cần phải cập nhật liên tục theo viewport.

8.4. Tóm tắt

Có nhiều kỹ thuật phân vùng không gian tùy theo kích thước, đặc điểm của không gian mà ta chọn các kỹ thuật phân chia phù hợp. Quadtree là một kỹ thuật quản lý vùng không gian tương đối tốt cho một không gian game không quá lớn. Trong quadtree ta cần lưu ý các khái niệm về : QNode, Tree, TreeObject , GameObject. Trong kỹ thuật quadtree này có hai giai đoạn chính là: tiền xử lý và chạy game. Giai đoạn tiền xử lý

dài, phải được xử lý kỹ trước khi chạy game, mọi quá trình build cây, gắn đối tượng vào node, lưu file đều được thực hiện ở giai đoạn này. Nhờ có giai đoạn tiền xử lý, giai đoạn chạy game sẽ trở nên dễ dàng, đơn giản hơn, chỉ cần load các file đã lưu lên và tạo cây thông qua đó. Một điều cuối cùng, kỹ thuật phân vùng này không phải là chính xác tuyệt đối, nó chỉ mang tính tương đối.

TÀI LIỆU THAM KHẢO

1. Charles Kelly, Programming 2D Games, 2012, CRC Press
2. Michael Dawson, Beginning C++ Through Game Programming Third Edition, 2011, Course Technology
3. Jeannie Novak, Game Development Essentials: An Introduction, 2012 Delmar, Cengage Learning
4. Jonathan S. Harbour, Beginning Game Programming, 2005 Thomson Course Technologies
5. Daniel Sánchez-Crespo Dalmau, Core Techniques and Algorithms in Game Programming, 2003 New Riders Publishing
6. Bob Bates, Game Design. 2nd edition, 2004 Thomson Course Technologies