TRƯỜNG ĐẠI HỌC BÁCH KHOA HÀ NỘI

VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

──────── \* ────────



ĐỒ ÁN

**TỐT NGHIỆP ĐẠI HỌC**

NGÀNH HỆ THỐNG THÔNG TIN

NGHIÊN CỨU VÀ ĐỀ XUẤT

GIẢI PHÁP CÔNG NGHỆ

XÂY DỰNG CỔNG GAME DI ĐỘNG

TRÊN NỀN TẢNG COCOS2D-X

Sinh viên thực hiện : **Nguyễn Tuấn Trí**

Lớp CNTT-TT 2.2 – K56

Giảng viên hướng dẫn: **ThS. Nguyễn Hồng Phương**

HÀ NỘI 05-2016

**PHIẾU GIAO NHIỆM VỤ ĐỒ ÁN TỐT NGHIỆP**

1. Thông tin về sinh viên

Họ và tên sinh viên: Nguyễn Tuấn Trí

Điện thoại liên lạc: 09937 09718 Email: [nguyen.tuan.tri@gmail.com](mailto:nguyen.tuan.tri@gmail.com)

Lớp: CNTT-TT 2.2 K56 Hệ đào tạo: Đại học chính quy

Đồ án tốt nghiệp được thực hiện tại: Bộ môn Hệ thống Thông tin, Viện CNTT & TT, Trường Đại học Bách Khoa Hà Nội

Thời gian làm đồ án tốt nghiệp: Từ ngày 26/01/2015 đến ngày 25/05/2015

2. Mục đích nội dung của ĐATN

* Nghiên cứu và đề xuất giải pháp công nghệ xây dựng cổng game di động, trên nền tảng cocos2d-x.
* Xây dựng các công cụ, các thiết kế, chuẩn bị cho dự án xây dựng một sản phẩm cổng game di động thực sự.

3. Các nhiệm vụ cụ thể của ĐATN

* Tìm hiểu về công nghệ làm game cocos2d-x và làm một game nhỏ để minh họa.
* Nghiên cứu, đề xuất giải pháp xây dựng cổng game trên nền tảng cocos2d-x.
* Xây dựng các công cụ, đưa ra các thiết kế cụ thể, để hiện thực hóa giải pháp.
* Xây dựng một ứng dụng cổng game di động nhỏ minh họa cho giải pháp.

4. Lời cam đoan

Tôi – *Nguyễn Tuấn Trí* – cam kết ĐATN là công trình nghiên cứu của bản thân tôi dưới sự hướng dẫn của *ThS. Nguyễn Hồng Phương*.

Các kết quả nêu trong ĐATN là trung thực, không phải là sao chép toàn văn của bất kỳ công trình nào khác.

*Hà Nội, ngày……tháng.…..năm……….*

Tác giả ĐATN

*Nguyễn Tuấn Trí*

5. Xác nhận của giảng viên hướng dẫn về mức độ hoàn thành của ĐATN, cho phép bảo vệ:

*Hà Nội, ngày……tháng.…..năm……….*

Giảng viên hướng dẫn

*ThS. Nguyễn Hồng Phương*

Tóm tắt nội dung đồ án tốt nghiệp

Lời cảm ơn

Mục lục

Danh mục các từ viết tắt

Danh mục các bảng

Danh mục các hình vẽ

Phần 1: Đặt vấn đề và định hướng giải pháp

(5 trang)

I-Lý do chọn đề tài

1-Xu hướng phát triển mạnh của game mobile

2-Nhu cầu xây dựng một cổng game mobile

3-Cocos2d-x được nhiều nhà phát triển game ở Việt Nam lựa chọn

II-Nhiệm vụ của đồ án

III-Công nghệ làm game cocos2d-x

Phần 2: Các kết quả đạt được

Trong phần này, ta sẽ trình bày chi tiết các kết quả của đồ án. Do đã trình bày sơ lược về công nghệ làm game cocos2d-x ở phần 1 rồi nên phần này ta sẽ trình bày về các nội dung sau:

* Phát triển một framework hỗ trợ việc kiểm thử đơn vị cho cocos2d-x (gọi tắt là framework test).
* Phát triển một sản phẩm game thực tế trên nền tảng cocos2d-x.
* Nghiên cứu và thử nghiệm công nghệ làm một cổng game di động dựa trên cocos2d-x

I-Phát triển framework test cho cocos2d-x

1-Tại sao cần phát triển một framework test?

Như chúng ta đã biết, kiểm thử là một công việc quan trọng để đảm bảo chất lượng sản phẩm. Unit test là một loại kiểm thử được thực hiện trên từng đơn vị mã nguồn riêng lẻ (module), ví dụ như các lớp, các hàm...Việc unit test cho một module được thực hiện bởi chính developer viết ra module đó. Unit test đem lại rất nhiều lợi ích. Lợi ích lớn nhất là giúp chúng ta phát hiện lỗi từ rất sớm. Mà càng phát hiện lỗi sớm, thì lại càng tiết kiệm chi phí.

Đáng tiếc là cocos2d-x chưa hỗ trợ cho ta một công cụ để thực hiện unit test một cách bài bản, chuyên nghiệp. Có lẽ cũng bởi vì hầu hết các đối tượng cần test trong game là các đối tượng liên quan đến đồ họa, khó mà unit test tự động được. Thành ra việc unit test, nếu có, đều được thực hiện thủ công.

Thực ra việc thực hiện unit test thủ công cũng không phải là vấn đề. Vấn đề là chúng ta thực hiện chúng một cách phi chính thức. Chẳng hạn như, vừa viết xong một lớp nào đấy, chúng ta sẽ kiểm tra luôn, bằng cách viết thêm một số đoạn code để test vào ngay mã nguồn chính, xem nó có “chạy được” hay không. Rồi sau đó nếu chạy được thì comment cái đoạn code để test vừa viết lại.

Cách làm này nhìn qua có vẻ nhanh, gọn, nhẹ, tiết kiệm thời gian. Nhưng bản chất là sự thiếu chuyên nghiệp, đánh giá thấp tầm quan trọng của unit test. Đem lại nhiều rắc rồi về lâu dài, bởi lẽ:

* Việc test một cách phi chính thức như vậy, chỉ đảm bảo cái module chúng ta vừa test “chạy được” là được. Như vậy, không có gì đảm bảo là module đó đã được cài đặt đúng, không có lỗi. Nhỡ trong trường hợp này chạy đúng, nhưng trong trường hợp khác chạy sai thì sao? Trên nền tảng này chạy tốt những trên nền tảng khác lỗi thì sao? Thành ra, nếu cứ lấy cái tiêu chuẩn tối thiểu là “chạy được” để áp dụng thì chẳng phải giống các sản phẩm “made in china” một thời được phương Tây đánh giá là: miễn ráp lại với nhau mà không bị bung ra là đạt chuẩn. Sử dụng tiêu chuẩn “chạy được”, chúng ta sẽ không thể phát hiện lỗi sớm, sản phẩm chúng ta cung cấp cho thị trường sẽ là một sản phẩm đầy bug 🡪 Thời gian fix bug kéo dài 🡪 Chi phí lớn + bòn rút thời gian mà đáng lẽ được dành cho việc phát triển các tính năng mới.
* Các code và resource cho mục đích develope và cho mục đích test sẽ bị lẫn lộn với nhau 🡪 Quản lý khó khăn. Nếu chúng ta muốn loại bỏ những phần liên quan đến test ra khỏi sản phẩm release, hay muốn chạy lại một đoạn code test nào đó, sẽ khá rắc rối và dễ xảy ra các sai sót, nhầm lẫn.
* Khi viết unit test một cách chính thức, thì những mã nguồn test đó sẽ chính là một loại tài liệu rất hữu ích. Các lập trình viên khác có thể nhìn vào và biết được cách sử dụng của từng module. Còn nếu viết một cách phi chính thức như trên thì sẽ chẳng có tài liệu hữu ích nào hết. Nếu muốn biết được cách sử dụng của module đó thì có 2 cách:
* Hoặc là ngồi đọc mã nguồn (trong nhiều trường hợp là không có một tài liệu hoặc comment gì cả).
* Hoặc là hỏi trực tiếp tác giả đã viết ra, để nhận được một sự giải thích tận tình, hoặc không.

Dựa trên quan điểm như vậy, cho nên công việc đầu tiên của đồ án này là phát triển một framework test để sử dụng trong quá trình phát triển các sản phẩm trên nền tảng cocos2d-x. Việc phát triển một framework như vậy không chỉ giúp thực hiện unit test một cách bài bản mà còn giúp cho việc kiểm thử các API, hoặc thử nghiệm và phát triển các ý tưởng. Đó đều là những công việc được thực hiện rất thường xuyên trong suốt quá trình phát triển sản phẩm.

2-Thiết kế framework test

2.1-Khái niệm cơ sở

Trước tiên, ta thấy rằng việc unit test cho các đối tượng liên quan đến đồ họa (mà chúng ta đang hướng đến) là test thủ công (manual unit test). Cho nên , sẽ không có sự khác biệt nhiều giữa unit test, hay kiểm thử API, hay các đoạn code để thử nghiệm, phát triển ý tưởng. Ta sẽ khái quát hóa chúng thành một khái niệm chung là: code for test (các đoạn code nhằm mục đích test), viết tắt là **ctest**. Phân biệt với các đoạn code liên quan đến phát triển sản phẩm: code for develope, viết tắt là **cdev**.

**Ctest** lại được chia làm 2 loại:

* **Console ctest**: Là loại ctest mà ta có thể quan sát kết quả của nó ở giao diện console, không cẩn đến giao diện đồ họa. Ví dụ: đoạn code để kiểm tra xem việc download một file có thực hiện thành công hay không.
* **Gui ctest**: Là loại ctest mà ta chỉ có thể quan sát kết quả của nó ở giao diện đồ họa. Ví dụ đoạn code để kiếm tra xem hàm làm nảy một quả bóng có thực hiện đúng hay không.

2.2-Ý tưởng cơ bản

Framework test sẽ hỗ trợ cả console ctest và gui ctest. Như đã định nghĩa ở trên, console ctest hay gui ctest thực chất chỉ là các đoạn mã. Mục đích của chúng ta là tách biệt các đoạn mã này ra khỏi mã phát triển sản phẩm, hay tách biệt ctest ra khỏi cdev, để giúp cho việc quản lý hiệu quả hơn.

Ý tưởng là:

* Mỗi một ctest đặt vào một file riêng biệt. Các file này nằm trong một thư mục riêng, độc lập hoàn toàn với thư mục chữa cdev.
* Cung cấp 2 chế độ chạy là: chế độ bình thường và chế độ test, được điều khiển bởi một biến, chẳng hạn: enableTest:
* enableTest = false là chế độ bình thường
* enableTest = true là chế độ test.

Khi ở chế độ test, chúng ta có thể lựa chọn và chạy các gui ctest hay console ctest đã được viết trước đó, quan sát kết quả của chúng.

2.3-Thiết kế chi tiết

2.3.1-Console ctest

Console ctest chẳng qua chỉ là các đoạn mã. Ta có thể đóng gói chúng trong một hàm. Mỗi console ctest đặt trong một file riêng biệt. Các file này sẽ đặt trong một thư mục chung chẳng hạn srctest/testfunc. Để tránh việc bị conflict tên hàm với các hàm có sẵn, ta có thể thêm vào trước nó một tiền tố chẳng hạn “f\_”. Ví dụ:

|  |  |  |
| --- | --- | --- |
| Tên ctest | Link file chứa ctest | Tên hàm tương ứng với ctest |
| example | srctest/testfunc/example.js | f\_example |

Khi cần chạy ctest example, framework sẽ load file srctest/testfunc/example.js vào và thực hiện việc gọi hàm f\_example.

2.3.1-Gui ctest

Gui ctest, thực chất là một đoạn mã có trách nhiệm chính là tạo ra một màn hình. Và cho phép ta thực hiện các thao tác test trên màn hình đó. Trong cocos2d-x, để tạo ra một màn hình, chúng ta sử dụng lớp cc.Scene. Vì vậy, ta sẽ tạo ra một lớp TestScene kế thừa từ cc.Scene, cug cấp cho nó thêm một số tính năng như:

* Tạo ra menu thao tác, nhờ đó ta có thể thực hiện nhiều thao tác khác nhau với đối tượng.
* Ghi log trên màn hình, nhờ đó, ta có thể quan sát được trạng thái bên trong của các đối tượng.
* Tạo các thông báo nhanh trên màn hình. Tính năng này cũng có phần hơi giống so với tính năng ghi log trên màn hình. Nhưng khác ở chỗ, các thông báo nhanh thì sẽ hiển thị và biến mất sau một thời gian nhất định. Còn ghi log trên màn hình thì quan sát một trạng thái của một đối tượng theo thời gian thực, và trong suốt thời gian thực hiện kiểm thử.



Hình trên thể hiện sơ đồ cài đặt của TestScene. Trong đó:

* TestScene: Lớp cơ sở để thực hiện các gui ctest, extend từ cc.Scene.
* CustomTestScene: Một gui ctest cụ thể, được extend từ TestScene.
* MenuTest: Một layer, có trách nhiệm tạo ra các menu thao tác cho TestScene.
* LogTest: Một layer, có trách nhiệm ghi log, theo dõi trạng thái của các đối tượng và tạo ra các thông báo nhanh trên màn hình.

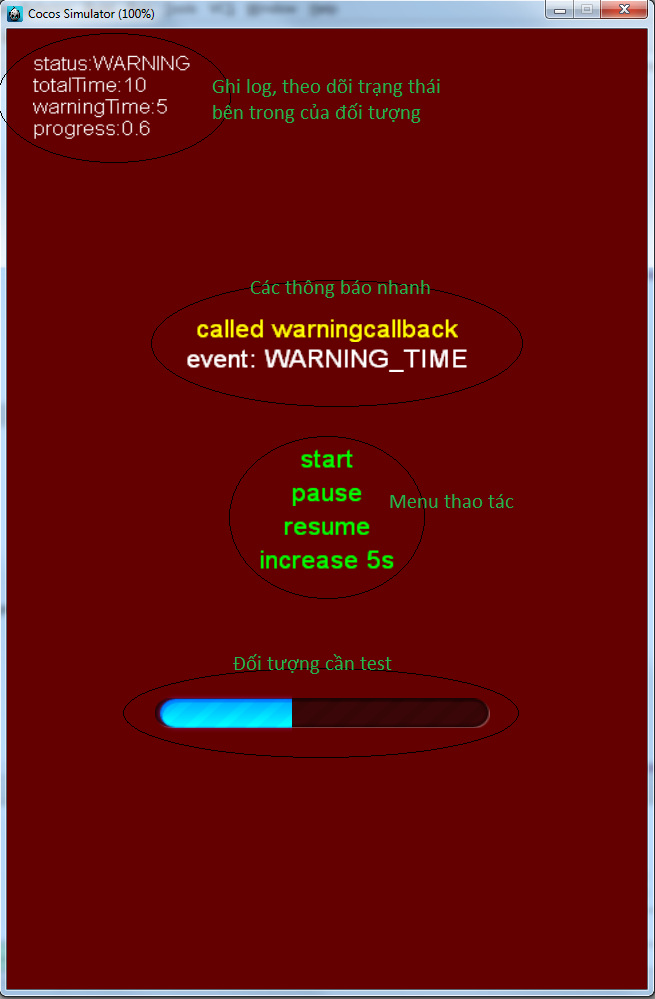
Các API cơ bản mà TestScene cung cấp:

|  |  |
| --- | --- |
| API | Mô tả |
| addCommand(name, callback) | Thêm một lệnh vào menu thao tác.   * name: Tên lệnh * callback: Hàm sẽ được thực hiện khi gọi lệnh |
| addWatch(name, callback) | Theo dõi trạng thái của một đối tượng   * name: Tên hiển thị cho trạng thái. * callback: Hàm trả về giá trị của trạng thái. |
| makeToast(text, setting) | Tạo ra một thông báo trên màn hình và mất đi sau một thời gian. Tương tự như tính năng của Toast trong android. Vì vậy, hàm này có tên là makeToast.   * text: Nội dung thông báo * setting: Một số thông số hiển thị, ví dụ setting.color: màu sắc, setting.size: kích cỡ... |

3-Ví dụ thực tế minh họa việc sử dụng tính năng gui ctest:

Sử dụng tính năng gui ctest của framework test không chỉ cho phép ta kiểm tra được sự hoạt động của các API, mà còn cho phép ta quan sát được các trạng thái bên trong của các đối tượng, nhờ đó phát hiện ra các sai sót tốt hơn.

Dưới đây là một ví dụ về sử dụng TestScene để thực hiện unit test cho lớp TimeBar:



Trong ví dụ này, ta thực hiện test một số hàm của lớp TimeBar:

* start
* pause
* resume
* increaseTime

Để kiểm thử tính đúng đắn của việc cài đặt các hàm trên, ngoài việc quan sát đối tượng bên ngoài, chúng ta còn quan sát cả trạng thái bên trong. Bên cạnh việc kiểm tra tính đúng đắn về mặt logic, chúng ta còn có thể kiểm tra được các yêu cầu liên quan đến hiển thị.

Một lợi ích rất đáng kể cho việc unit test theo kiểu này đó là: trên cơ sở các thao tác cơ bản được cung cấp ở menu thao tác như trên, chúng ta có thể thực hiện rất rất nhiều các test case khác nhau, ngoài một số testcase thông thường như Start🡪Pause🡪Resume, chúng ta còn có thể thực hiện rất nhiều testcase khác như:

* Thực hiện start liên tục 2 lần.
* Thực hiện pause liên tục 2 lần.
* Chưa start mà thực hiện pause luôn.
* Chưa start mà thực hiện resume luôn.
* Chưa pause nhưng vẫn thực hiện resume.
* Thực hiện increaseTime khi đang pause.
* increaseTime nhiều lần, vượt quá cả thời gian tối đa.
* Đã hết thời gian rồi, nhưng vẫn thực hiện increaseTime.
* Thực hiện increaseTime và trong quá trình thời gian đang tăng lên thì lại thực hiện pause, sau đó lại resume lại.
* Thực hiện pause, điều chỉnh đồng hồ của hệ thống chậm lại vài phút, rồi thực hiện resume.
* ....

Thực tế khi thực hiện unit test cho TimeBar lần đầu tiên, hầu hết các test case được liệt kê ở trên đều fail. Đây là một điều hết sức bình thường. Chỉ là không bình thường khi chúng ta luôn cho rằng mình đã thực hiện mọi thứ đúng đắn ngay từ đầu, vì vậy, không bao giờ thực hiện kiểm tra lại các công việc mình đã làm một cách bài bản, có hệ thống.

Với những bug được phát hiện, ngoài những lỗi về lập trình, chúng ta còn phát hiện ra cả những lỗi liên quan đến đặc tả và thiết kế. Có những lỗi chỉ liên quan đến TimeBar, nhưng có những lỗi lại ảnh hưởng đến toàn bộ hệ thống, ví dụ lỗi về tính thời gian, chúng ta đã tham khảo và tin tưởng hoàn toàn thông tin thời gian từ client, đó là một nguồn thông tin không thực sự đảm bảo, người chơi game có thể dễ dàng hack bằng cách điều chỉnh đồng hồ hệ thống.

Từ ví dụ trên, chúng ta có thể thấy, mặc dù framework test được thiết kế khá đơn giản, nhưng hiệu quả mang lại rất lớn. Tuy nhiên, cũng cần nói thêm rằng, việc có áp dụng được vào thực tế hay không còn phụ thuộc vào một yếu tố rất quan trọng khác là: việc thiết kế phần mềm có cho phép thực hiện unit test dễ dàng hay không. Nếu các lớp không tách biệt nhau, mà dính chặt với nhau thành một khối, sử dụng lớp này thì yêu cầu phải khởi tạo hàng đống đối tượng của các lớp khác, thì unit test sẽ rất khó mà thực hiện được. Có một số khái niệm và nguyên tắc thiết kế giúp chúng ta giải quyết điều đó. Chẳng hạn như: khái niệm decoupling, nguyên tắc dependency inversion...Tất nhiên, để hiểu và áp dụng được vào thực tế, cần có sự đầu tư nhất định về thời gian và công sức.

II-Phát triển một sản phẩm game thực tế trên nền tảng cocos2d-x

1-Mô tả game

(Giống tài liệu game design thôi)

2-Đặc tả feeling

3-Thiết kế

III-Nghiên cứu và đề xuất giải pháp xây dựng cổng game

Phần 3: Kết luận

(3 trang)

Tài liệu tham khảo

[1]