**SOFTWARE ARCHITECTURE**

**Game xếp hình for ZP**

**1-Giới thiệu**

***1.1-Mục đích tài liệu***

Đưa ra cái nhìn tổng quan nhất về kiến trúc của phần mềm, bao gồm các quyết định thiết kế (hoặc ý tưởng thiết kế) quan trọng và nổi bật nhất. Có thể kèm theo giải thích cho các quyết định, ý tưởng này.

***1.2-Phạm vi tài liệu***

Tài liệu sử dụng cho game xếp hình for ZP.

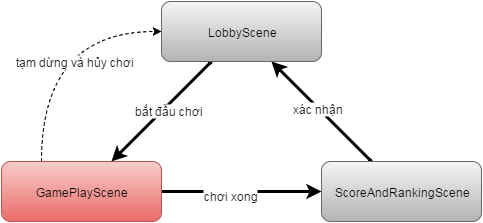
***1.3-Lịch sử phiên bản***

|  |  |  |  |
| --- | --- | --- | --- |
| **Tên phiên bản** | **Ngày thực hiện** | **Người thực hiện** | **Ghi chú** |
| 1.0 | 25/12/2015 | Nguyễn Tuấn Trí | Bản đầu tiên |
| 1.1 | 27/12/2015 | Nguyễn Tuấn Trí | -Bổ sung thêm các lớp BackGroundLayer, UILayer, EffectLayer. Tổ chức lại GamePlayScene theo các layer này.  -Bổ sung thêm lớp Item. |
| 1.2 | 28/12/2015 | Nguyễn Tuấn Trí | -Bổ sung mục 2.2, đưa ra quan niệm rõ ràng hơn về sự đóng gói. |
| 1.3 | 16/01/2016 | Nguyễn Tuấn Trí | -Đính chính lại về phạm vi sử dụng việc giao tiếp thông qua cơ chế event – listener.  -Đề xuất thêm cơ chế giao tiếp thông qua việc gọi hàm callback. |

**2-Kiến trúc phần mềm**

***2.1-Các scene***

Phần mềm có 3 scene: LobbyScene, GamePlayScene và ScoreRankingScene. Và ở mức tổng quát nhất thì hệ thống được phân rã thành 3 scene này. Việc di chuyển từ scene này đến scene khác được thực hiện qua hàm runScene của đối tượng cc.Director. Luồng di chuyển giữa các Scene được mô tả qua biểu đồ sau:



***2.2-Các thành phần trên giao diện là sự đóng gói của cả chức năng và giao diện***

**Sự đóng gói**

Một số thành phần mà ta thấy trên giao diện như điểm số, thanh fever, thanh thời gian, ... được ánh xạ thành các lớp như ScoreBarUI, FeverBarUI, TimeBarUI (hậu tố UI để ám chỉ rằng các lớp này đại diện cho các thành phần trên giao diện người dùng). Điều cần nhấn mạnh ở đây là: các lớp này không chỉ có chức năng hiển thị mà còn đảm nhận luôn cả phần quản lý các chức năng, nghiệp vụ logic liên quan đến phần hiển thị đó. Vì vậy, ta nói rằng **các lớp này (\*UI) là sự đóng gói của cả chức năng và giao diện**.

**Nguyên nhân**

* Thứ nhất là do sự liên quan chặt chẽ giữa logic và hiển thị của các thành phần này (tính cohesion). Cho nên việc tách riêng logic và hiển thị có thể gây khó khăn cho việc phát triển.
* Thứ hai là tính lỏng lẻo, có thể tách rời nhau của các chức năng (tính decoupling). Có nghĩa là các chức năng không cần thiết (và không nên) tập trung vào một chỗ vì thực ra chúng chẳng liên quan đến nhau nhiều lắm. Ví dụ:
* Chức năng quản lý thời gian và quản lý điểm số chẳng liên quan gì đến nhau.
* Chức năng quản lý thời gian và quản lý fever cũng vậy.
* Chức năng chơi game (được hiểu là tương tác và chơi trên GameBoard) và quản lý điểm số thì có liên quan đến nhau nhưng cũng không quá chặt chẽ. Người ta hoàn toàn có thể chơi game mà không cần tính điểm số. Có nghĩa là chỉ điểm số phụ thuộc vào chơi game chứ chơi game không phụ thuộc vào điểm số.

**Lợi ích**

* Logic được phân tách rõ ràng, dễ quản lý.
* Có thể tái sử dụng một số đối tượng cho các ứng dụng khác (ví dụ TimeBarUI).

**Một cách nhìn khác rõ ràng hơn về sự đóng gói này**

* Mỗi thành phần trên giao diện sẽ được khái niệm hóa, đóng gói cả phần hiển thị và phần chức năng thành một phần tử giao diện mới hoàn chỉnh. Giống như trước đây, chúng ta có Button, có Label, có Checkbox, ProgressBar, thì bây giờ chúng ta có TimeBar, có FeverBar, có ScoreBar...
* Các phần tử giao diện mới này được phân tích, thiết kế, phát triển và kiểm thử độc lập nhau.
* Và chúng có thể được tái sử dụng trong nhiều ngữ cảnh khác nhau, nhiều ứng dụng khác nhau.

***2.3-Các cơ chế giao tiếp của các thành phần trong hệ thống***

Việc giao tiếp giữa các thành phần của hệ thống (gọi là component) thông qua 3 cách:

* Gọi hàm: một component cung cấp các hàm (api) để một client nào đó gọi đến.
* Thay đổi trực tiếp thuộc tính của các đối tượng. Cách giao tiếp này nên hạn chế tối đa có thể. Một component không nên biết các thuộc tính của các component khác.
* Giao tiếp thông qua cơ chế **event-listener**: Một component phát sinh các sự kiện gọi là nguồn. Các component khác đóng vai trò là bộ nghe sẽ nhận được thông báo mỗi khi có sự kiện và sẽ gọi đến các hàm xử lý thích hợp.

Ví dụ: Khi người chơi ăn một cụm icon, sẽ tạo ra một sự kiện. Sự kiện này kéo theo yêu cầu cập nhật lại trạng thái của rất nhiều đối tượng khác, thanh điểm số cần cập nhật lại, thanh fever cần cập nhật lại, thậm chí thanh thời gian cũng cần cập nhật lại (nếu có item thời gian trong cụm icon ăn được), rồi trên màn hình phải xuất hiện hiệu ứng chúc mừng (pefect, good, cool, ...), rồi âm thanh, ... Tất cả các cập nhật này không nên để trong GameBoardUI, mà nên phân bổ ra cho các lớp liên quan (các bộ nghe). GameBoardUI chỉ có trách nhiệm thông báo rằng đã có sự kiện người chơi ăn một cụm icon.

Hai cơ chế đầu (gọi hàm, can thiệp trực tiếp vào thuộc tính) tạo ra sự gắn kết giữa các đối tượng, tạo ra các mối liên hệ ràng buộc chặt chẽ trong hệ thống. Nhưng đôi khi làm cho hệ thống trở nên quá cứng nhắc, khó thay đổi. Cách giao tiếp thứ 3 thông qua cơ chế event listener giúp làm cho các liên kết này trở nên lỏng lẻo và linh hoạt hơn, các thành phần sẽ độc lập với nhau, dễ bảo trì phát triển hơn. Vì vậy, bên cạnh các cơ chế giao tiếp thông thường như gọi hàm và thay đổi thuộc tính thì **cơ chế giao tiếp event-listener cũng sẽ là một cơ chế giao tiếp phổ biến trong hệ thống**.

/\*\* Cập nhật ở bản 1.3 – không lạm dụng cơ chế event listener\*/

Tuy nhiên, không nên quá lạm dụng cơ chế giao tiếp này.

Cơ chế giao tiếp event-listener trong game này nên được sử dụng trong trường hợp như sau: a muốn xây dựng một lớp A có khả năng reusable. Tại một thời điểm nào đó, lớp A có những thay đổi về trạng thái, sự thay đổi về trạng thái của lớp A kéo theo sự cập nhật trạng thái của nhiều lớp khác. Tuy nhiên, A cần tránh việc “biết” các đối tượng khác. Vì nếu A cần “biết” những đối tượng đó 🡪 nó phải gắn chặt với những đối tượng đó 🡪 có những đối tượng đó thì A mới làm việc được 🡪 A không reusable. Cho nên việc giao tiếp qua cơ chế event – listener sẽ phát huy tác dụng trong trường hợp này.

Nhưng nếu giả sử GameBoard phát sinh sự kiện ăn được item thời gian. Mà TimeBar lại phải bắt sự kiện này để xử lý thì TimeBar đã mất đi tính reusable vì nó phụ thuộc vào GameBoard. Đây gọi là lạm dụng. Trong trường hợp này, chỉ đơn giản là TimeBar cung cấp một API: increaseTime(t) là đủ. Một lớp bên ngoài sẽ bắt sự kiện ăn item thời gian do GameBoard phát sinh và gọi đến hàm increaseTime này.

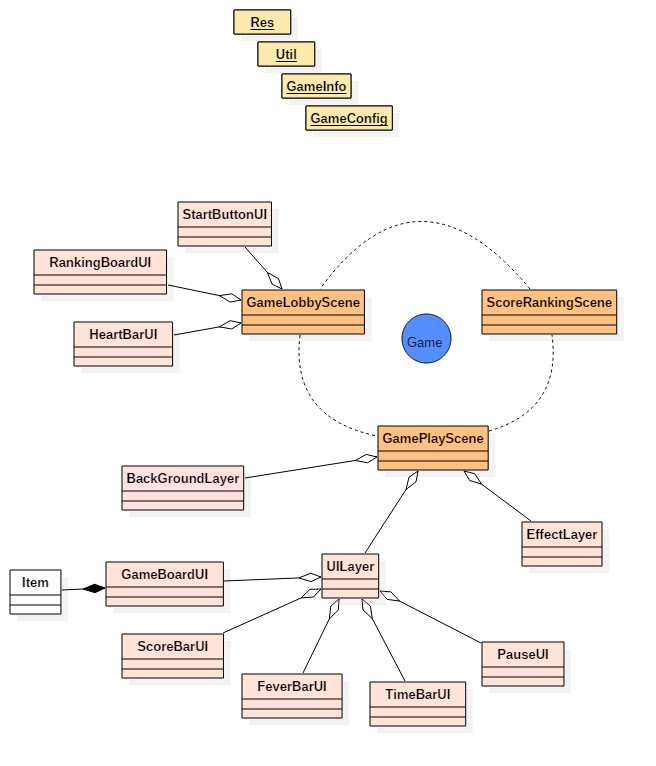
**/\*\*Cập nhật ở bản 1.3 – cơ chế giao tiếp thông qua callback \*/**

**Sẽ thiếu sót nếu sử dụng javascript mà không nhắc đến cơ chế giao tiếp thông qua callback. Đây là một cơ chế giao tiếp cực kỳ hiệu quả và tiện dụng.** Nếu không sử dụng thì sẽ rất phí. Thậm chí trong nhiều trường hợp, nên sử dụng cơ chế này thay thế cho event-listener, hoặc sử dụng kết hợp với event listener, để cung cấp đa dạng hơn các sự lựa chọn.

Để dễ hiểu, xin đưa ra một ví dụ. Chúng ta có lớp TimeBar. Khi một đối tượng TimeBar đếm ngược và hết thời gian, nó sẽ phát sinh sự kiện TIME\_OUT để các đối tượng khác biết. Nhưng ta cũng có thể cung cấp thêm một cơ chế callback bên cạnh cơ chế giao tiếp thông qua event-listener. Một hàm callback được truyền vào cho TimeBar (tại một thời điểm bất kỳ trước khi TIME\_OUT, nhưng thường là lúc khởi tạo TimeBar). Khi TIME\_OUT thì hàm này sẽ được gọi, đồng thời sự kiện cũng vẫn phát sinh. Như vậy, bây giờ chúng ta có thêm nhiều lựa chọn hơn. Một đối tượng có thể bắt sự kiện TIME\_OUT để xử lý. Hoặc đối tượng đó cũng có thể truyền vào callback cho TimeBar, callback sẽ được gọi khi TIME\_OUT. Mỗ

***2.4-Tổng hợp về các lớp trong hệ thống***

Dưới đây là biểu đồ tổng quan về các lớp của hệ thống, các lớp được liệt kê ở đây chỉ là sư phân tích ban đầu có thể chưa đầy đủ và chưa hoàn toàn hợp lý, nhưng ý tưởng thiết kế về cơ bản đã được thể hiện.



|  |  |  |
| --- | --- | --- |
| **Tên lớp** | **Mô tả** | **Các sự kiện mà nó phát sinh** |
| GameLobbyScene | Biểu diễn cho màn hình lobby | 0 |
| GamePlayScene | Biểu diễn cho màn hình bàn chơi (màn hình chính của game) | 0 |
| ScoreRankingScene | Biểu diễn cho màn hình thông báo kết quả khi người chơi chơi xong | 0 |
| StartButton | Biểu diễn cho nút bắt đầu của màn hình lobby, cần được đầu tư kỹ lưỡng về giao diện và hiệu ứng. | 0 |
| RankingBoardUI | Biểu diễn cho bảng xếp hạng ở màn hình lobby | 0 |
| HeartBarUI | Biểu diễn cho thanh chứa trái tim ở màn hình lobby | 0 |
| BackGroundLayer | Biểu diễn cho màn hình nền của GamePlayScene | 0 |
| UILayer | Là layer chính trong GamePlayScene, chứa tất cả các thành phần giao diện quan trọng của game. Và đồng thời có thể đóng vai trò như một controller, điều khiển sự phối hợp của các thành phần đó. | 0 |
| **GameBoardUI** | Biểu diễn cho bảng chứa icon ở màn hình bàn chơi, nơi người chơi tương tác với game. Đây là lớp có logic phức tạp, **cần được phân tích thiết kế kỹ lưỡng**. | EatItems |
| Item | Biểu diễn cho một item cấu thành nên gameboard. Quản lý cả hiển thị, hiệu ứng (rơi) và logic của item đó. | 0 |
| ScoreBarUI | Biểu diễn cho thanh điểm số ở màn hình bàn chơi. Quản lý luôn logic tính điểm. | 0 |
| FeverBarUI | Biểu diễn cho thanh fever ở màn hình bàn chơi. Quản lý luôn logic của về chuyển trạng thái fever. | StartFever  FinishFever |
| TimeBarUI | Biểu diễn cho thanh thời gian ở màn hình bàn chơi. Quản lý logic về thời gian. | WarningTime  Timeout |
| PauseUI | Biểu diễn cho màn hình tạm dừng khi người chơi nhấn nút tạm dừng. | PauseGame  ResumeGame |
| EffectLayer | Layer này là layer nằm trên cùng của GamePlayScene, có trách nhiệm quản lý các hiệu ứng trong khi chơi game. | 0 |
| GameInfo | Chứa các thông tin chung của game dùng cho tất cả các lớp như: tên người chơi, kỷ lục cũ của người chơi, danh sách các đối thủ, ... | 0 |
| GameConfig | Chứa các thông tin cấu hình cho game, ví dụ: thông tin về thời gian chơi của một ván chơi, bonus time, thời gian duy trì thanh fever, thời gian xuất hiện và tồn tại của các item ngẫu nhiên, điểm số khi ăn icon thường, điểm số khi ăn icon trong fever mode, điểm số khi ăn icon ở cú chót, thời gian phục hồi trái tim, .... | 0 |
| Res | Chứa url của các tài nguyên trong game như hình ảnh, âm thanh, hiệu ứng particle, ... | 0 |
| Util | Chứa các hàm tiện ích phục vụ cho hoạt động của các lớp khác. | 0 |