

Project áp dụng mô hình mvc gồm 3 thành phần cơ bản controller, model, view.

I.Phần Game:

* Message:
  + GameMessage: Đăng ký các name message để giao tiếp qua lại trong game, chú ý các tiền tố ở mỗi messeage
    - On: Dùng để gọi 1 action nào đó ở view (thường từ command) ví dụ như cập nhập tiền của user, chia bài, đánh bài
    - Send: View gọi 1 action để tương tác với sever, phần command sẽ lắng nghe sự kiện này để sử lý logic trước khi gọi lên server (vd như validate data)
    - Receive: server sẽ trả lại sự kiện, data, phần command sẽ lắng nghe sự kiện trả về này, sử lý logic rồi call sang view để thực thi
  + GameEvent: Đăng ký các event để dispatch event giữa các commponent sử dụng trong view
* Command: xử lý dữ liệu trước khi gửi và nhận data với server
  + SetupGameCommand: nơi đăng ký các event command, proxy, mediator
  + Chú ý luôn có 2 phương thức gửi (send) và nhận (receive) với mỗi event
  + Khi 1 command trong game có các xử lý riêng biệt không chung với các game khác thì override lại hoặc remove command ở core đi để add lại command khác (khi luồng game khác với các game khác)
* Model:
  + Proxy: là lớp trung gian dùng để giao tiếp data với các thành phần trong game, khi muốn lấy dữ liệu từ vo nào đều phải qua lớp proxy tương ứng, proxy thường có đầy đủ các phương thức bên VO
    - Cách đăng ký 1 proxy: this. facade.registerProxy(new TLMNProxy());
    - Cách hủy đăng ký 1 proxy: this.facade.removeProxy(TLMNProxy.proxyName);
    - Khi vào 1 game khác phần coregame đã tự hủy các command, proxy, mediator nên ko phải xử lý phần hủy cho từng game
  + VO: Là lớp data tương ứng với game, chú ý khi xử lý dữ liệu liên quan đến lớp data này nên tạo 1 function trong data để xử lý chứ ko viết bên ngoài
    - TableVO: các thuộc tính của bàn, với mỗi game sẽ có lớp TableVO tương ứng đc extent từ TableVO, khi thêm 1 params, nhớ reset data về default trong hàm reset
    - SeatVO: các thuộc tính của ghế, mỗi Game cũng sẽ có 1 SeatVO tương ứng
    - RulesVO: kiểu game vd như chơi 2 hay 4 người, max người trong bàn, mức cược, tiền tối thiểu, tối đa vào bàn
    - Logic: Xử lý logic trong game
* View :
  + Mediator: Là phần trung gian giữa lớp view và các thành phần khác
    - listNotificationInterests : Đăng ký name messeage để lắng nghe
    - handleNotification : khi có 1 sự kiện được gọi từ command, proxy hoặc mediator khác sẽ được vào function này -> call sang view để xử lý.
    - addHanlers: lắng nghe các sự kiên từ lớp view tương ứng
  + Ui: Mỗi Game đều có 1 Scene chính và các component trong game vd game TLMN có scene chính là GameTLMN
    - Scene chính (GameTLMN): chủ yếu để đăng ký các component, các event, các sự kiện click button để call ra ngoài mediator
    - Để đăng ký 1 component gọi: this.rootGame.add(this.mcPlayerCards);
    - Chú ý trong ui của coregame có 3 class Component và Composite, và CompositeNode để đăng ký các node, function, component
    - VD: khi server call startgame, command sẽ lắng nghe sự kiện này trong ReceiveStartGameCommand rồi call messeage ON\_START\_GAME sang view, phần mediator sẽ nắng nghe sự kiện này và call vào ui qua function onStartGame ở GameTLMN, khi gọi this.rootGame.startGame() tất cả các function trong component sẽ được gọi startgame, thứ tự call sẽ tùy thuộc vào lúc đăng kí component
* Network: có 2 loại soket và http
  + http: Đăng ký các request http HttpRequestProxy (network/http/model/proxy)
  + socket:
    - request: gửi dữ liệu lên server (network/sfs/controller/request)
    - reponse: nhận dữ liệu từ server (network/sfs/controller/reponse), trước khi gọi đến command phải xử lý dữ liệu trên server về dữ liệu game (vì game viêt chung mà server nhiều cái luống viết riêng nên phải convert lại)
    - môi game đều có class request và reponse để xử lý ác messeage riêng từng game, phần coregame để xử lý chung cho toàn bộ, chú ý phần ovrride để tránh mất dữ liệu trong core
    - SFSData: name data
    - SFSEvent: name Event

II. Asset cocos