21/11/2021

Nhóm 4

Lớp 20CTT1TN – N3 – TH CQ

Thiết kế phần mềm

Phần mềm Cờ tỷ phú – Đồ án cuối kì Thực hành PPLT OOP

Nội dung

[1 Sơ đồ UML các lớp dự kiến sử dụng 2](#_Toc88409156)

[2 Khái quát mô hình thiết kế của phần mềm 2](#_Toc88409157)

[3 Mô tả các lớp 2](#_Toc88409158)

[3.1 Lớp thuộc xử lý Giao diện đồ họa 2](#_Toc88409159)

[3.1.1 MainWindow 2](#_Toc88409160)

[3.2 Lớp thuộc xử lý logic 2](#_Toc88409161)

[3.2.1 Game 2](#_Toc88409162)

[3.2.2 Player 3](#_Toc88409163)

[3.2.3 Cell 3](#_Toc88409164)

[3.2.4 NormalLand 3](#_Toc88409165)

[3.2.5 Command 4](#_Toc88409166)

[3.2.6 MortageCommand/BuyCommand/GoCommand/PassCommand/SellCommand/RedeemCommand/BuildCommand 4](#_Toc88409167)

[3.2.7 CommandHandler 4](#_Toc88409168)

[4 Mô tả design pattern 5](#_Toc88409169)

# Sơ đồ UML các lớp dự kiến sử dụng

# Khái quát mô hình thiết kế của phần mềm

Cấu trúc thiết kế có thể được chia làm hai lớp: lớp xử lý Giao diện đồ họa (GUI) và lớp xử lý các lệnh logic. Hai lớp này có thể trao đổi thông tin qua lại với nhau. Lớp GUI nhận các đầu vào từ người sử dụng dưới dạng các lệnh và gửi yêu cầu xử lý các lệnh đó xuống lớp xử lý các lệnh logic thông qua Command Design Pattern hoặc gọi trực tiếp lệnh theo cơ chế Signal/Slot của Qt Framework. Lớp xử lý lệnh logic sau khi hoàn thành nhiệm vụ của mình gửi lại các thông tin lên lớp GUI để hiển thị cho người dùng quan sát thông qua việc gọi các lệnh thuộc lớp GUI phù hợp.

Sử dụng Qt Framework có thể tiếp nhận các tương tác của người dùng với các yếu tố đồ họa và kích hoạt các lệnh xử lý đúng đắn thông qua cơ chế Signal/Slot: ví dụ khi một nút được nhấn sẽ tạo một Signal và đối tượng, hàm hoặc lệnh quan tâm tới sự kiện này có thể thu thập Signal đó cùng các thông tin đi kèm để thực thi. Đây cũng là một cách triển khai Observer Design Patter.

Trò chơi trong phần mềm xoay quanh việc thực hiện các thao tác cơ bản đã được định nghĩa trước theo cơ chế của trò chơi, như: di chuyển/qua lượt/mua đất/xây nhà/bán nhà/cầm cố…. Command Design Pattern được sử dụng để đóng gói các lệnh này từ Client (có thể là lớp GUI khi có yêu cầu xử lý thao tác mong muốn của người chơi hoặc các đối tượng, hàm nào khác có thể khởi tạo và đóng gói một lệnh), lưu các lệnh đã đóng gói đó vào một đối tượng CommandHandler, và gọi việc thực thi gói lệnh mong muốn thông qua đối tượng CommandHadler đó từ bất kì đâu và bất kì lúc nào. Thông tin chi tiết về cách pattern được áp dụng trong thiết kế của phần mềm này được mô tả chi tiết hơn sau phần mô tả các lớp và đối tượng.

# Mô tả các lớp

## Lớp thuộc xử lý Giao diện đồ họa

### MainWindow

* Đóng vai trò là giao diện chính của trò chơi, chứa tất cả các đối tượng đồ họa và thông tin xuất hiện bên trong đó; làm cửa ngõ liên lạc giữa lớp GUI và lớp xử lý logic.

## Lớp thuộc xử lý logic

### Game

* Lưu trữ tất cả các thông tin về một ván trò chơi hiện tại; xử lý, thiết lập các quá trình bắt đầu, kết thúc và các cơ chế chung của trò chơi
* Thuộc tính (Tất cả các thuộc tính sau đều có phạm vi Private):
  + players <danh sách các đối tượng kiểu Player>: lưu trữ thông tin của tất cả người chơi trong ván chơi hiện tại
  + board <Board>: lưu trữ thông tin của tất cả các ô bàn cờ trong ván chơi hiện tại
  + currentPlayer <int>: chỉ số id của người chơi đang trong lượt của người đó
* Phương thức:
  + + initializeGame(): khởi tạo các thông tin cần thiết để bắt đầu ván chơi, bao gồm việc đọc thông tin từ file dữ liệu bên ngoài và gán vào các đối tượng tương ứng; liên lạc với lớp GUI để xử lý việc hiển thị
  + + initializePlayer(): khởi tạo các thông tin của người chơi; liên lạc với lớp GUI để xử lý việc hiển thị
  + + checkForEnd() <bool>: kiểm tra xem ván chơi đã đạt được điều kiện để kết thúc chưa
  + + Ranking(): sắp xếp các người chơi theo thứ tự chỉ định

### Player

* Lưu trữ tất cả thông tin của một người chơi nào đó; xử lý các lệnh, hành động liên quan tới người chơi mà có thể thay đổi thuộc tính của lớp này

Board

* Thể hiện như là đối tượng bàn cờ. Lưu trữ tất cả thông tin của các ô trên bàn cờ; thực hiện các lệnh liên quá tới phạm vi kích hoạt bởi các ô bàn cờ và truy xuất thông tin ô bàn cờ.

### Cell

* Lớp thuần ảo thể hiện đối tượng là các ô bàn cờ, gồm các interface để các loại ô bàn cờ khác nhau có thể thay đổi hành vi xử lý tùy thuộc vào tính của loại ô đó

### NormalLand

* Kế thừa từ lớp Cell, thể hiện là các ô bất động sản thông thường. Chứa thông tin về giá trị của ô đất đó; thực thi các lệnh làm thay đổi thuộc tính của lớp này (như mua, xây nhà,…)

### Command

* Lớp thuần ảo đóng vai trò làm interface để các loại lệnh khác nhau kế thừa lớp này có thể thực hiện các hoạt động khác biệt tùy thuộc vào loại lệnh của chúng, là Command trong Command design pattern

### MortageCommand/BuyCommand/GoCommand/PassCommand/SellCommand/RedeemCommand/BuildCommand

* Mỗi loại lớp trên đây đều kế thừa từ lớp Command, đóng vai trò như một gói lệnh chứa các thông tin cần thiết để gửi đi thực hiện tác vụ yêu cầu và là ConcreteCommand trong Command design pattern
* Thuộc tính: chứa thông tin đối tượng có thể thực thi các tác vụ được yêu cầu và các tham số cần thiết khác
* Phương thức:
  + Khởi tạo: được gọi vào đầu chương trình và lưu vào CommandHadler để có thể gọi lệnh thực thi thông qua chúng
  + execute(): xử lý lệnh mà lớp này đại diện thông qua những đối tượng (receiver) có thể thực thi phù hợp

### CommandHandler

* Đóng vai trò là Invoker trong Command design pattern. Lưu trữ danh sách các gói lệnh và các phương thức xử lý để chỉnh sửa gói lệnh cũng như khởi chạy gói lệnh đó.
* Thuộc tính:
  + – commandList <Command>: danh sách các gói lệnh được khởi tạo từ trước. Phần tử từ 0 tới 7 là các gói lệnh mặc định mà một người chơi có thể thực hiện được khởi tạo từ đầu ván chơi và gán vào đây.
* Phương thức:
  + + setCommand(Command\*, int): gán gói lệnh đã được tạo sẵn từ Client vào một vị trí trong commandList
  + Go(): truyền người chơi muốn thực hiện lệnh này vào trong gói lệnh và khởi chạy gói lệnh
  + BuildHouse(): truyền người chơi muốn thực hiện lệnh này và mã id của ô đất mà người chơi chọn vào gói lệnh và khởi chạy gói lệnh đó

# Mô tả design pattern

Command design patter được ứng dụng vào trong mẫu thiết kế của phần mềm này. Một cách tổng quan, Command design pattern gồm các thành chính sau:

* Client: bất kỳ đối tượng nào có thể tạo ra yêu cầu của mình thông qua việc khởi tạo một gói lệnh. Gói lệnh này chứa các thông tin cần thiết để gửi đến các reciever thực hiện như: đối tượng có thể thực thi yêu cầu, tham số thông tin thêm,… Có thể tưởng tượng đối tượng lớp này như là thực khác trong nhà hàng ra yêu cầu làm món mình muốn ăn.
* Command: một lớp thuần ảo làm interface cho các loại gói lệnh (command) khác nhau. Chỉ chứa duy nhất một phương thức execute() để Invoker có thể kích hoạt việc xử lý gói lệnh thông qua hàm này.
* Concrete command: các lớp kế thừa từ command, mỗi loại concrete command class tương ứng với một loại yêu cầu từ client nào đó. Gồm thông tin các receiver mà có thể xử lý gói lệnh này, các tham số thông tin cần thiết khác trong thuộc tính của lớp. Client có thể tạo lập một gói lệnh bằng cách gọi contructor của concrete command tương ứng. Hàm execute() sẽ gọi các phương thức các đối tượng receiver có thể xử lý yêu cầu. Các concrete command có thể được coi như là các lá đơn ghi yêu cầu món ăn mà các thực khách muốn được phục vụ.
* Invoker (sender): truyền đạt các gói lệnh tới các receiver thích hợp mà có thể xử lý yêu cầu. Nó còn có thể lưu trữ các gói lệnh được tạo sẵn, chỉnh sửa gói lệnh và các thao tác khác với gói lệnh. Invoker kích hoạt việc xử lý gói lệnh thông qua lệnh gọi hàm execute() của gói lệnh đó. Invoker giống như bồi bàn truyền đạt yêu cầu giữa vị khác và đầu bếp
* Receiver: là các đối tượng thật sự có thể xử lý được những gói lệnh. Đóng vai trò như đầu bếp chế biến các món ăn theo yêu cầu của thực khách trong các lá đơn mà bồi bàn đưa tới.

Một khi các yêu cầu xử lý trong mã nguồn được đóng gói thành các gói lệnh, chúng trở thành các đối tượng và ta có thể làm được nhiều thứ với chúng như lưu trữ, đặt thời điểm thực thi gói lệnh, undo/redo,… Trong phần mềm này, command design pattern giúp tách bạch giữa đối tượng muốn thực thi yêu cầu từ người sử dụng khỏi các đối tượng thật sự xử lý yêu cầu, đồng thời tăng tính mở rộng của phần mềm sau này khi cần thêm các thao tác, lệnh trong trò chơi hay thay đổi việc gán lệnh.

Trong thiết kế của phần mềm này, mỗi vai trò trong command design pattern được đảm nhiệm bởi các lớp như sau:

* Client: có thể từ bất cứ đối tượng nào trong mã nguồn gọi lệnh constructor của concrete command, chủ yếu là từ lớp MainWindow khi tạo các gói lệnh phù hợp với tương tác của người sử dụng. Khi bắt đầu một ván chơi, các lệnh cơ bản như đi/qua lượt/mua,… được khởi tạo trước và lưu vào danh sách các lệnh của đối tượng Invoker để có thể được gọi trong suốt ván chơi.
* Concrete command: là các lớp kế thừa từ command (GoCommand/PassCommand/…) chứa receiver có thể thực thi các lệnh xử lý yêu cầu. execute() gọi các receiver từ cả các lớp GUI hay lớp xử lý logic để vừa thực thi xử lý logic vừa hiện kết quả ra GUI.
* Ivoker: là lớp CommandHalder. Có các phương thức được định nghĩa sẵn để kích hoạt xử lý gói lệnh phù hợp. Các phương thức này dù vậy lại có thể chỉnh lại gói lệnh tùy vào tham số mà nó nhận được, đây là hạn chế hiện tại trong việc áp dụng mẫu comand design pattern vào đồ án này khi vai trò tạo lập các gói lệnh chỉ nên được client thực hiện.
* Receiver: bất cứ đối tượng nào mà được gọi thực thi phương thức trong hàm execute().