شطرنج شش‌ضلعی

پویا شمس کلاهی
پروژه پایان ترم اول سال ۱۴۰۲ در درس برنامه‌نویسی مقدماتی دانشکده ریاضی شریف.

مقدمه

در این پروژه قصد داریم تا نوعی از شطرنج در زمین شش ضلعی به نام شطرنج گلینسکی را در زبان جاوا پیاده سازی کنیم.

فهرست کلاس‌ها و پکیج‌ها

- ir/sharif/math/bp02_1/hex_chess
- util
  - ApplicationHolder
  - Coordinate
  - CopyAble (interface)
  - HexMat
  - IndexList
  - Pair
- Chess
  - ChessGame
  - ChessBoard
  - BoardCell
  - ChessPiece (all pieces extend this class)
    - Pawn
    - King
    - Queen
    - Knight
    - Rook
    - Bishop
  - MoveHelper
- config
  - HexConfig
- FileIO
  - LoadConfig
- listeners
  - ChessListener
- Main.java

توضیحات

ابتدا باید گفت که کتابخانه‌های گرافیکی در پکیج `ir/sharif/math/bp02_1/hex_chess` در اختیار ما قرار داده شده و

تغییر چندانی در آن داده نشده به جز چند فیچر کوچک که به آن‌ها اشاره خواهد شد.
در زیر توضیحات کلاس‌های مختلف آمده است.

util

ApplicationHolder

این کلاس شامل یک آبجکت `static` از نوع `Application` است و صرفا برای دسترسی بخش های مختلف کد به تنها `Application` موجود است.

Coordinate

این کلاس یک تایپ از مختصات را تعریف میکند که در آن هر خانه شش‌ضلعی با ترکیبی خطی از دو بردار به طور یکتا معین میشود. به همین دلیل تنها دو فیلد از نوع `int` دارد که این دو عدد را نشان میدهند. توضیح دقیقتر این روش مختصات‌دهی در این پیوند قابل مشاهده است.
با توجه به اینکه متخصات گلینسکی به شدت احمقانه و بی‌نظم است این کار میزان حالت بندی مورد نیاز را کاهش میدهد و خوانایی کد را به شکل نمایی افزایش میدهد.
همچنین دو تابع `static` به نام‌های toGlinski و fromGlinski در این کلاس برای تبدیل این روش مختصات‌دهی به گلینسکی و بالعکس وجود دارد. و بقیه متدها اعمال دیگری روی متخصات مانند جمع و منها، قرینه کردن، کپی کردن و بررسی برابری دو متخصات است.

CopyAble

این اینترفیس صرفا یک متد `copy` از نوع خود آبجکت تعریف میکند و هدف آن این است که انواع کلاس‌های جنریک مختلفی که تعریف شده‌اند قابلیت کپی کردن خود و اعضای داخل خود را داشته باشند. تقریبا تمامی کلاس‌های موجود در پروژه این اینترفیس را پیاده‌سازی میکنند.

IndexList

این کلاس صرفا یک `wraper` حول ArrayList است که اندیس‌های آن را به مقدار دلخواهی که با متغییر `offset` مشخص میشود به عقب شیفت میدهد تا امکان اندیس‌دهی با اعداد منفی فراهم شود. در آینده این فیچر برای راحت‌تر کار کردن با مختصات‌ها و تبدیل آن‌ها به اندیس در جدول استفاده میشود.

HexMat

این کلاس یک ماتریس شش‌ضلعی با طول متغییر (!) و محتوای جنریک (!) را تعریف میکند و در هستهٔ خود از یکی IndexList دو بعدی استفاده میکند تا ماتریس را عینا با متخصات‌های واقعی آن نشان دهد. متدهای کلاس شامل getter, setter ها و چند تابع کمکی برای بررسی معتبر بودن مختصات‌ها است.

Pair

این یک کلاس کمکی است که برای پاس دادن جفت‌های متغییر از نوع دلخواه (جنریک) استفاده میشود. و استثناً اینترفیس CopyAble را پیاده‌سازی نمیکند چون جفتی که به آن داده می‌شود لزوماً قابل کپی نیستند. شامل دو فیلد و getter, setter های آن‌هاست.

Chess

ChessPiece

مهم‌ترین کلاس در پکیج Chess است و تایپ یک مهره به صورت کلی را پیاده سازی می‌کند. فیلدهای آن شامل

- boolean is_white
- Coordinate pos
- String piece_name

می‌شوند. `is_white` برای نشان دادن رنگ مهره است و اگر سفید باشد مقدار آن true وگرنه false است. `pos` برای نشان دادن متخصات مهره در یک جدول شامل مهره است. و در نهایت `piece_name` یک رشته شامل نام مهره است که یکی از کاراکترهای یونیکد مهره‌های شطرنج است.
این کلاس یک متد `abstract` به نام get_valid_moves تعریف میکند که در تمامی فرزندان آن پیاده سازی می‌شود و کارش این است که با گرفتن یک HexMat به طول ۶ که نشان دهنده یک جدول است تمامی خانه‌هایی که این مهره در آن جدول می‌تواند به آن‌ها برود را در یک ArrayList از نوع مختصات خروجی دهد (توجه کنید که در این مرحله کیش بودن شاه بررسی نمیشود). کلاس چند متد دیگر هم برای بررسی کیش و مات و امکان حرکت کردن دارد و یک متد moveTo که کارهای مربوط به حرکت مهره به یک خانه دیگر را انجام می‌دهد.

BoardCell

نشان دهندهٔ یک خانه از جدول است و شامل سه فیلد زیر است:

- boolean highlighted
- Coordinate pos
- ChessPiece content

می‌شود. `highlighted` نشان دهنده این است که آیا مهره‌ای که انتخاب شده امکان حرکت به این خانه را دارد و در کلاس ChessBoard استفاده می‌شود. `pos` مختصات این خانه است و عملا بی‌خاصیت است و `content` یک مهره شطرنج است که نشان میدهد در این خانه چه مهره‌ای قرار دارد و اگر خالی باشد `null` می‌شود. متدهای آن صرفا getter, setter, copy هستند.

ChessBoard

نشان دهنده زمین بازی است و بیشتر لاجیک بازی در این قسمت اتفاق می‌افتد (می‌توان بحث کرد که آیا باید اینجا لاجیک را پیاده کرد یا نه. جواب این سوال این است که چون خانه ها ویژگی‌های این کلاس و آبجکت‌های آن هستند و کار کردن با آن‌ها از بیرون پیچیده، گیج‌کننده و سخت است، راحت‌تر است که این کارها را در همینجا انجام دهیم). فیلدهای آن در زیر آمده‌اند:

- HexMat<BoardCell> board
  نشان دهنده زمین است و در آن خانه‌ها ریخته شده‌اند.
- final int n = 6 (طول جدول)
- Coordinate selected
  اگر مهره‌ای انتخاب شود مقدار این متغییر برابر متخصات آن می‌شود وگرنه `null` می‌شود.
- ArrayList<Coordinate> cango
  مانند `selected` تمام خانه‌هایی که مهره انتخاب شده میتواند به آن‌ها برود در این لیست‌آرایه ریخته می‌شوند.
- boolean is_white (white's turn)
- ArrayList<ChessPiece> removed
  مهره‌های حذف شده را شامل می‌شود.

- Pair<Coordinate, Coordinate> last_move
متخصات آخرین حرکت را نشان می‌دهد و شامل خانه‌های مبدا و مقصد آن است. درصورتی که `null` باشند یعنی اولین حرکت بازی است.

متدهای زیر برای این کلاس تعریف شده‌اند که تقریبا تمام لاجیک بازی در آن‌هاست.

- set_default_board
خانه‌های جدول را مقدار دهی اولیه میکند (با توجه به چینش اولیه بازی)
- click
اگر خانه‌ای در جدول انتخاب شود این تابع صدا زده شده و با توجه به اینکه قبلا خانه‌ای انتخاب شده بود یا نه بازی را جلو می‌برد. اگر چیزی انتخاب نشده بود و خانه کلیک شده خالی نبود و نوبت بازیکن آن بود، متغییر `selected` برابر متخصات این خانه می‌شود و تمام خانه‌هایی که میتواند به آن‌ها حرکت کند در `cango` ریخته می‌شود و آن‌هایی که باعث کیش شدن شاه **نمی‌شوند** `highlight` می‌شوند (فیلد در BoardCell). وگرنه اگر `selected` موجود بود با توجه به اینکه خانه کلیک شده highlight شده یا نه مهره را به آن می‌برد یا `selected` را خالی می‌کند.
- draw
یک آبجکت از نوع `Application` می‌گیرد و وضعیت فعلی برد را در آن می‌کشد. هر خانه با توجه به این‌که انتخاب شده، مهره انتخاب شده میتواند به آن حرکت کند، کیش شده و یا خانهٔ حرکت قبلی است، رنگ می‌شود. همچنین اگر تنظیمات آن انجام شده باشد، در نوبت سیاه صفحه را می‌چرخاند تا بازی برای سیاه راحت‌تر باشد (!).
- write
مانند `toString` یک رشته قابل انکود/دیکود از برد را ارائه می‌دهد و برای سیو کردن وضعیت بازی در فایل استفاده می‌شود.
- load
یک آبجکت فایل دریافت می‌کند و سعی میکند وضعیت بازی را از روی آن بخواند. اگر هنگام خواندن به هرگونه اروری بخورد نشانه خراب بودن فایل است و باعث خراب شدن وضعیت برد می‌شود.
- copy, toString واضح هستند :\

ChessGame

نشان دهنده **خود** بازی است. شامل یک برد است که به کمک آن بازی را پیش می‌برد. همچنین انواع توابع برای ذخیره و بارگذاری (!) بازی به کمک فایل‌ها دارد. و از همه مهم‌تر یک لیست از وضعیت بازی تا به حال دارد که از آن برای عمل ***بسیار خلاقانه (!)*** `undo` استفاده می‌شود که صرفا برای اینکه دکمه آن موجود باشد تغییر کوچکی در کتابخانه گرافیکی به وجود آمده است و دکمه `undo` به منوی `file` اضافه شده.

MoveHelper

شامل انواع توابع برای کمک کردن به `ChessBoard` و `ChessGame` برای کارهای متخلفی از جمله بررسی کردن کیش و مات و پات و نوشتن و خواندن از فایل و رشته‌های متنی و غیره است و نام هر تابع نشان‌دهنده کاری است که می‌کند.

listeners

ChessListener

صرفا یک کلاس `wraper` حول `ChessGame` است و اعمال بازی را به آن انتقال می‌دهد.

config

HexConfig

شامل تنظیمات بازی به شکل تعدادی متغییر `static` است که در بخش‌های مختلف پروژه از آن‌ها برای تنظیم رفتار بازی استفاده می‌شود. تنظیمات فعلی شامل انواع رنگ‌های زمین بازی در محیط گرافیکی و نام بازیکن ها می‌شود و یک `boolean` که نشان دهندهٔ این است که آیا نوبت سیاه زمین بچرخد یا نه. این تنظیمات در فایل `config/config` ریخته و از آن خوانده می‌شوند و اگر موجود نباشد به تنظیمات پیشفرض برمی‌گردد.

FileIO

LoadConfig

شامل یک تابع `static` برای خواندن تنظیمات از فایل `config/config` است.

Main.java

بازی را با خواندن تنظیمات و ساختن یک `Application` و یک `ChessListener` شروع میکند.

خلاقیت‌های موجود

در چند قسمت از کد تغییراتی به وجود آمده است که با توجه به توضیحاتی که درباره آن‌ها داده شد میتوان این ویژگی‌ها را به عنوان کارهای خلاقانه‌تر جدا کرد:

- رنگ آمیزی جدول با توجه به حرکات موجود، حرکت قبلی و کیش بودن شاه.
- نام‌گذاری بازیکن ها.
- چرخاندن جدول در نوبت سیاه.
- تنظیمات به کمک فایل `config/config`.
- و از همه مهم‌تر، عمل `undo`!