رکب

فاز اول پروژه پایانی

ترم 4022

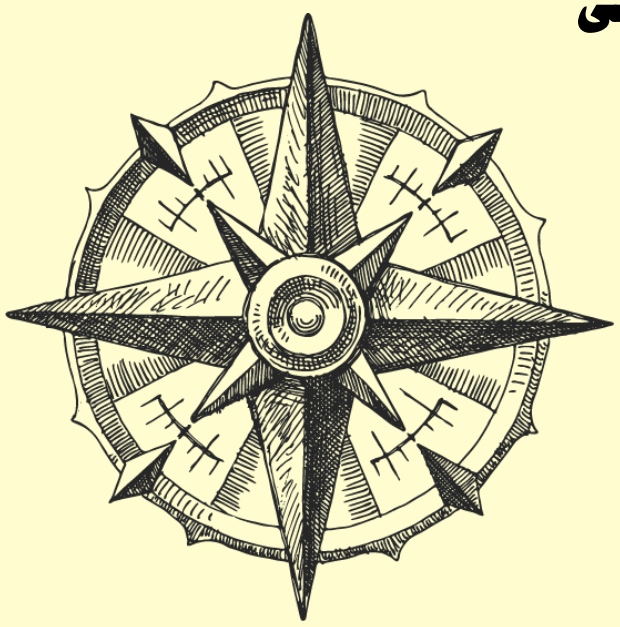
استاد :

مهدی سخایی نیا

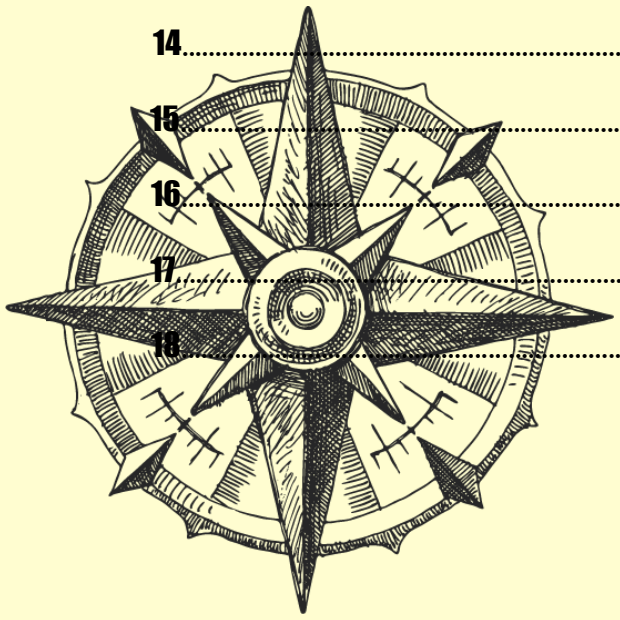
اعضا تیم :

مهرداد ورمزیار

محمد سینا ناظمی



فهرست :



مقدمه :

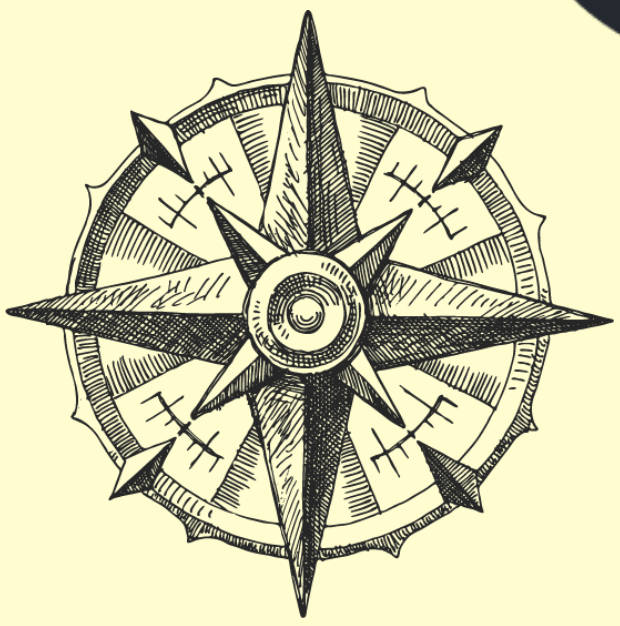
با سلام در این گزارش قصد دارم فرآیند کار تیم در پیاده سازی board game رکب را به طور مختصر و بخش به بخش توضیح بدم و در هر بخش از کد چالش ها و دستاورد ها و ..... را بیان کنم .

به طور کلی این فاز دو بخش حیاتی و دشوار داشت که شامل پیاده سازی کارت ها و پیاده سازی منطق بازی بود که جلوتر چالش های این بخش ها برسی می شود .

به طور کلی این فاز پروژه از این جهت که منطق بازی را شامل می شد می تواند به نوعی اسا

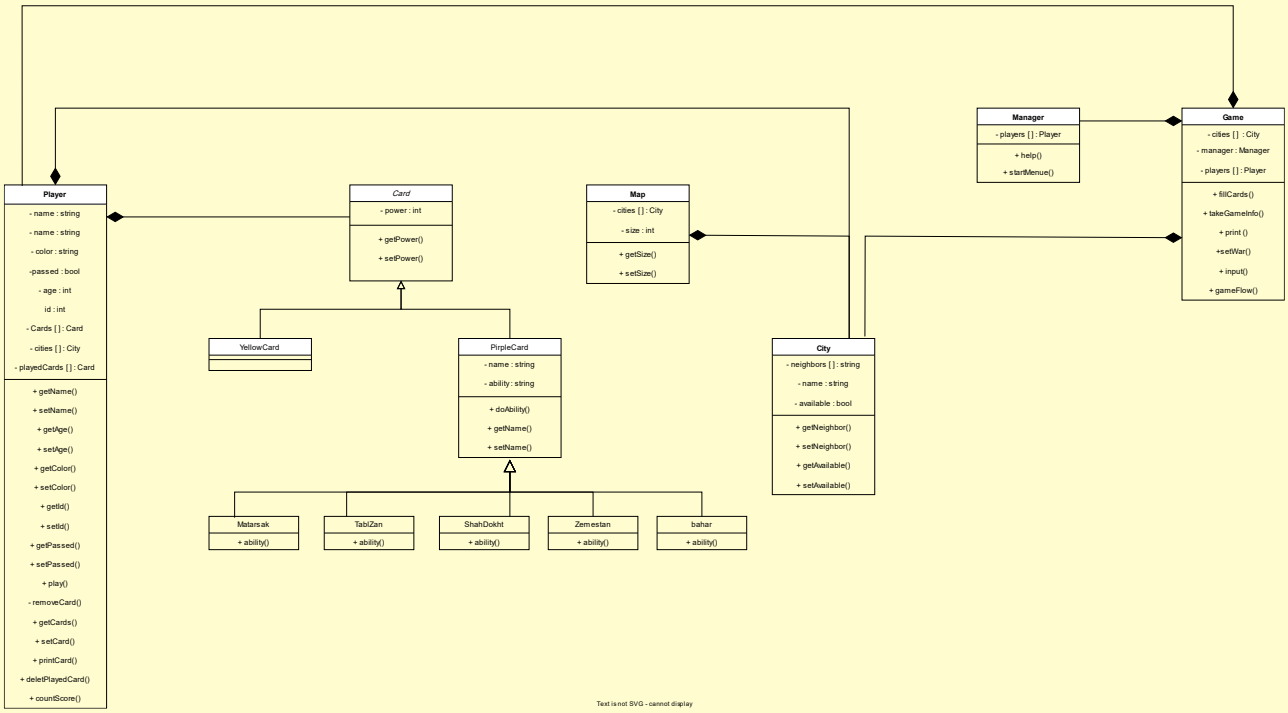
گیت :

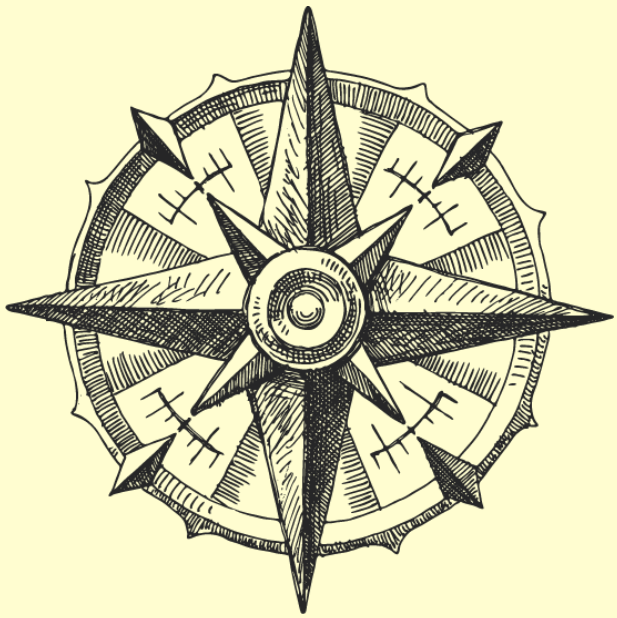
از طریق لینک زیر(لوگو github) می توانید فرآیند ساخت پروژه و class diagram پروژه و پروژه نهایی را مشاهده کنید .



Class diagram :

در طرح زیر می توانید طرح کلی اجرا پروژه و کلاس ها با ویژگی ها و رفتار هایشان را مشاهده کنید .





شرح کلی پروژه :

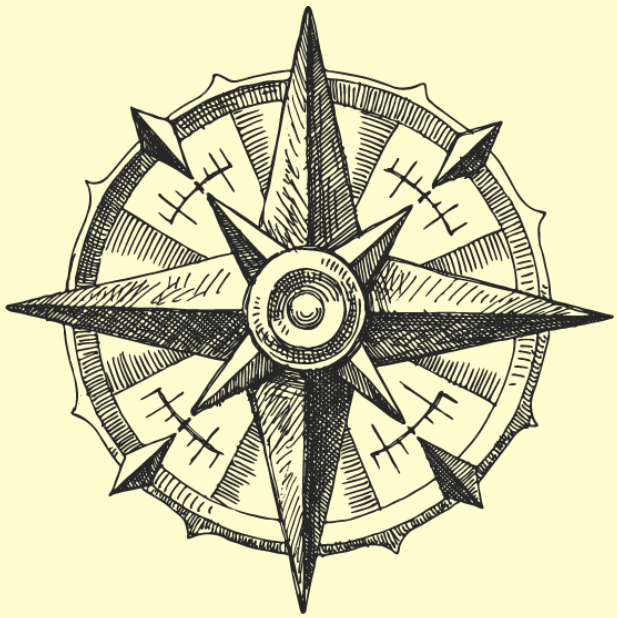
در این فاز از پروژه به طور کلی پروژه را در 13 کلاس پیاده سازی کردیم که کلاس کارت ها با توجه به روابط ارث بری پیاده سازی شدند و کلاسی که وظیفه اجرا منطق بازی و جریان آن را به عهده داشت را Game نام گذاری کردیم که سایر کلاس ها با آن رابطه composition یا aggregation دارند .

برای شرحی از وظایف به طور کلی من (مهرداد ورمزیار) پیاده سازی کلاس های Player , Map , City , Manager و بخش عمده ای از Game را بر عهده داشتم و هم تیمی من (سینا ناظمی) پیاده سازی کارت ها و بخشی از کلاس Game که با منطق کارت ها کار می کرد را پیاده سازی کرد .

هر کلاس در header file های جداگانه و با در نظر گرفتن declaration , interface پیاده سازی شده است و از طریق include شدن با هم ارتباط می گیرند .

از دیدگاه آموزشی این پروژه در بر گیرنده تمام مفاهیم شی گرا شامل ارث بری (inheritance) و چند ریختی (polymorphism) و کپسوله سازی (encapsulation) و کلاس ها (abstraction) بود و از تمام این مفاهیم در پیاده سازی استفاده شده است .

در ادامه کلاس به کلاس پروژه را بررسی می کنیم .



کلاس Player :

در این کلاس اطلاعات و داده های مربوط به یک بازیکن را ذخیره می کنیم ویژگی هایی از قبیل :

- کارت های بازیکن
- شهر هایی که مالک آن است
- نام و سن و رنگ و ...

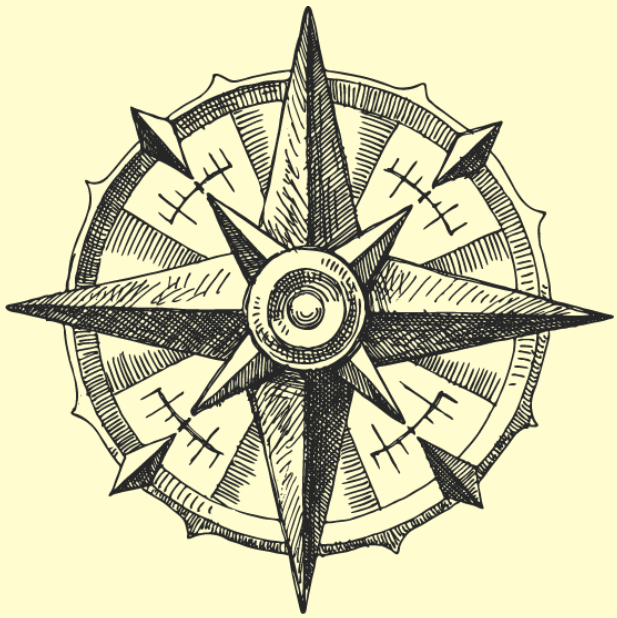
رفتار ها :

- بازی کردن
- نمایش شهر ها
- نمایش کارت ها
- حذف یا افزودن شهر یا کارت و ...

در پیاده سازی این کلاس این مورد در نظر گرفته شده که هر ویژگی یا رفتاری که وجود دارد تنها به خود بازیکن مربوط شوند و به طور بخش بخش شده متود ها اجرا شوند .

این کلاس از نظر روابط با کلاس های کارت و شهر رابطه aggregation دارد زیرا داده هایی از آن نوع دارد اما در صورت نبود کلاس Player آن ها نابود نمی شوند .

به طور کلی پیاده سازی این کلاس چالش بر انگیز نبود زیرا به طور عمده برای ذخیره مشخصات بود که پیرو الگورتم خاصی نمی باشد .



کلاس Card :

به طور کلی این کلاس و کلاس Game چالش بر انگیز ترین کلاس ها بودند از این جهت که کلاس از اصل ارث بری و چند ریختی و کلاس Game به دلیل روابط پیچیده بین سایر کلاس ها .

کلاس های فرزند : PurpleCard , YellowCard که خود کلاس PurpleCard دارای 5 کلاس فرزند است .

دلیل استفاده از ارث بری : ویژگی ها مشترک فرزندان

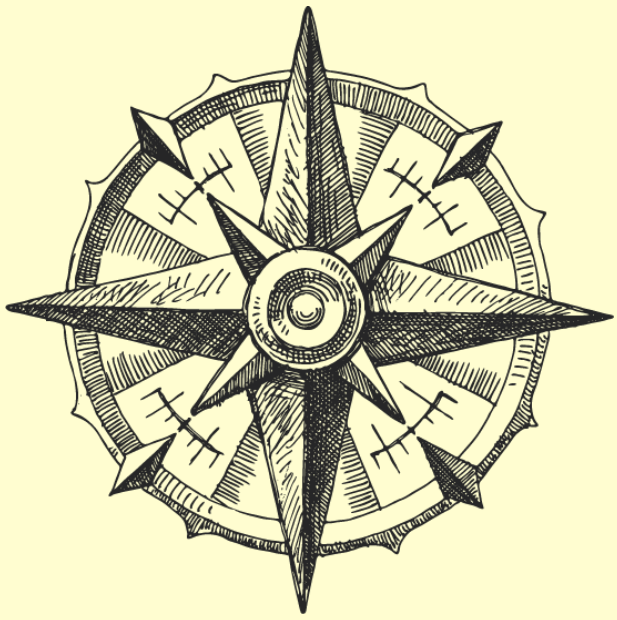
ویژگی ها :

- قدرت کارت
- نام کارت

رفتار ها :

- سازنده ها
- دریافت کننده و ست کننده ها

پیاده سازی این کلاس کمی جامع تر بود زیرا جزئیات بیشتر در کلاس های فرزند پیاده شده اند که در ادامه معرفی می شوند .



کلاس PurpleCard :

این کلاس خود دارای 5 کلاس فرزند است که شامل بهار و زمستان و شیردخت و مترسک و طبل زن است و از این جهت از ارث بری استفاده کرده ایم که پیاده سازی متود ability در هریک متفاوت صورت می گیرد پس ما یک virtual method ایجاد می کنیم و به صورت جداگانه برای هر کلاس فرزند با توجه به نوع توانایی ویژه آن کارت پیاده سازی را انجام می دهیم .

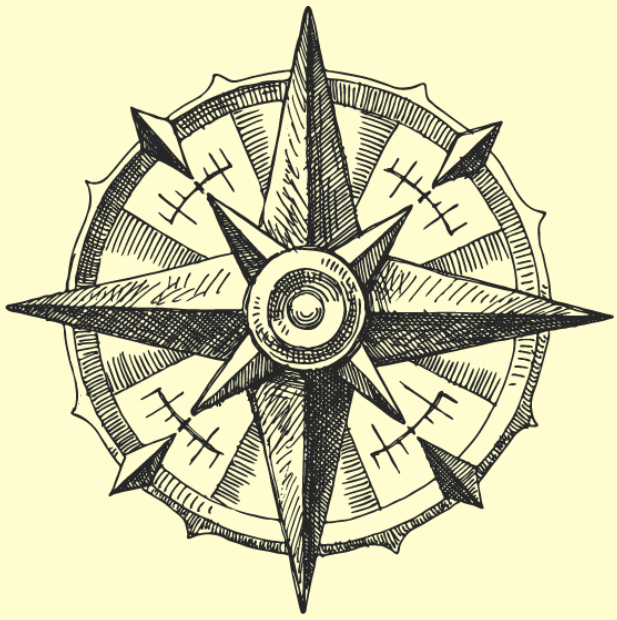
ویژگی ها کلاس :

- اولویت
- نام توانایی

رفتار ها :

- توانایی (2 تا با توجه به داده خروجی)
- دریافت ها و ست کردن ها
- سازنده ها

چالش مهمی که پیاده سازی این کلاس داشت چند ریختی بود و استفاده از یک تابع virtual و همچنین برای پاس دادن کارت های بازی شده توسط یک بازیکن از struct تحت عنوان playedCard استفاده شده است .



کلاس Bahar :

به طور کلی این کلاس و 4 کلاس فرزند دیگر جزئیات زیادی ندارند پس یکم بیشتر به پیاده سازی ها می پردازیم .

رفتار ها :

- اجرا توانایی
- سازنده

پیاده سازی توانایی این کلاس قرار است قویترین کارت بازی شده توسط بازیکن ها پیدا کند و قدرت آن را 3 واحد افزایش دهد که دارای یک پارامتر از نوع PlayedCard است و خروجی نیز از همین نوع است که ما در کلاس Game مقدار playedCards را برابر این تابع قرار می دهیم تا توانایی اعمال شود .

به طور کلی پیاده این کلاس چالش بر انگیز نبود .

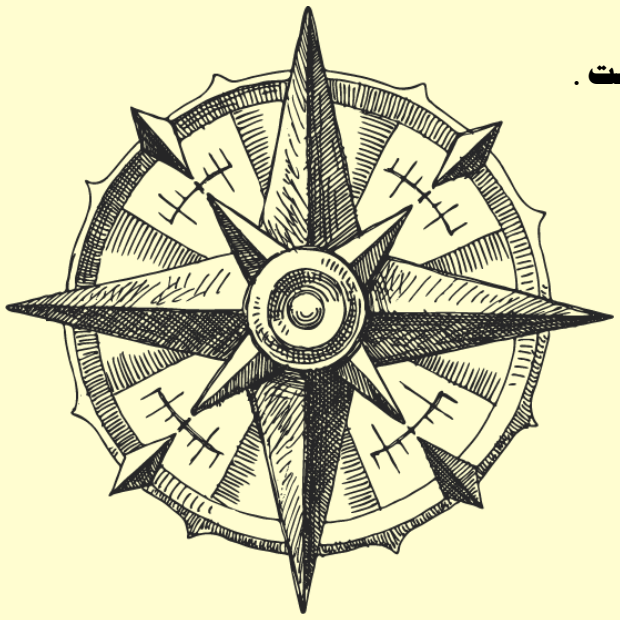
کلاس Zemestan :

این کلاس ساختار پیچیده ای ندارد و به طور کلی در ability امتیاز تمام کارت هارا 1 می کند .

رفتار ها :

- سازنده
- توانایی

به طور کلی ساده ترین کلاس این پروژه بود و هیچ چالشی نداشت .



کلاس TablZan :

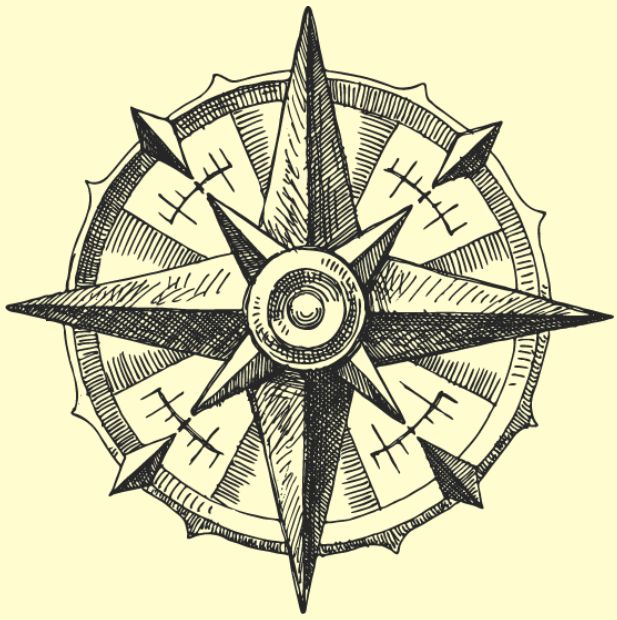
رفتار های کلاس :

- **سازنده**
- **توانایی**
- **بررسی زرد بودن**

نقطه تمایز اصلی این کلاس با دو کلاس قبل وجود تابعی برای بررسی زرد بودن کارتی است که قرار است توانایی رو بر روی آن پیاده سازی کند .

بدنه این تابع را می توانستیم در توانایی کارت پیاده سازی کنیم اما ترجیح دادیم برای افزایش ماژوال بودن کد آن را یک تابع جدا در نظر بگیریم و همچنین این تابع ایجاد شده تا خاصیت این کارت فقط بر روی کارت های زرد اعمال شود .

پیاده سازی توابع نیز پیچیدگی ای نداشت فقط امتیاز کارت های زرد دو برابر می شوند و همچنین اینکه این اتفاق برای کدام بازیکن بی افتد در کلاس گیم بررسی می شود .



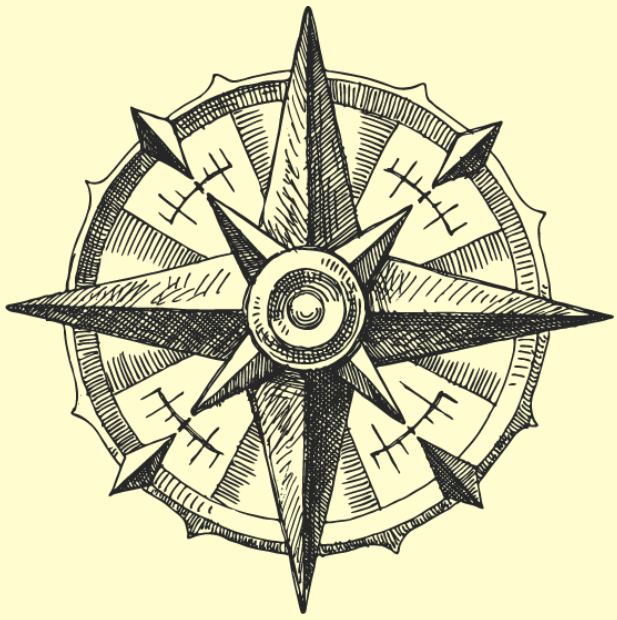
کلاس Matarsak :

رفتار های کلاس :

- **ورودی گرفتن از کاربر**
- **پیدا کردن کارت**
- **سازنده**
- **توانایی**

اگر بخوایم یه توضیح کلی بدیم بازیکن وقتی کارت مترسک را بازی می کند می تواند یکی از کارت های بازی شده خود را به دلخواه برگرداند پس ما یک تابع برای دریافت نام این کارت از کاربر و یک تابع برای پیدا کردن و بررسی وجود کارت و در نهایت تابع سربارگزاری شده (توانایی) را دارد که مسئولیت اجرا کلی توانایی مترسک را بر عهده دارد .

چالش و نکته جذاب این کلاس استفاده از دو تابع private دیگر علاوه بر توانایی بود که ماژوال بودن کد را افزایش می دهد .



کلاس ShahDokht :

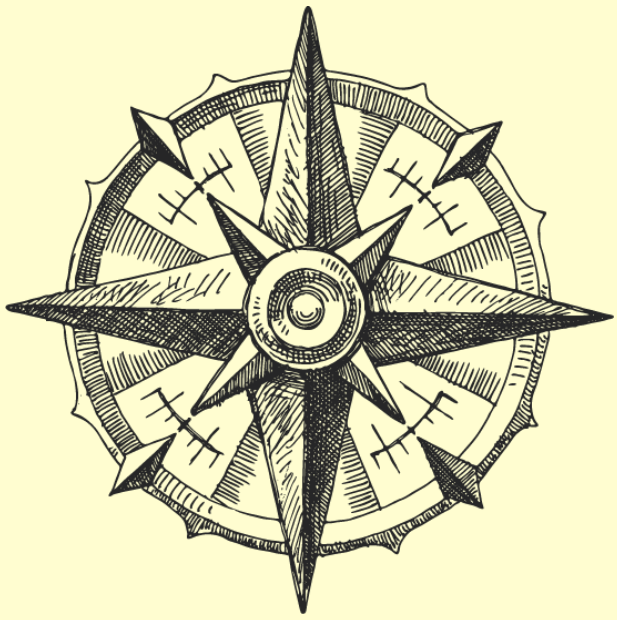
ویژگی ها :

- **سازنده**
- **توانایی**

توانایی در این تابع صرفا override شده و کاربردی ندارد و بر عکس سایر کارت های بنفش که قدرتی موقع ایجاد ندارند این کارت قدرت 10 را هنگام ایجاد دریافت می کند. (از کلاس Card ارث بری شده که برای سایر کارت های بنفش به جز این کارت در این فاز صفر است)

به طور کلی این کارت از هیچ کارتی تاثیر نمی پذیرد که این تاثیر ناپذیری به طور کلی در Game بررسی می شود .

این کلاس عملا چالشی نداشت و چند خط کد ساده برای پیاده سازی بود .



کلاس YellowCard :

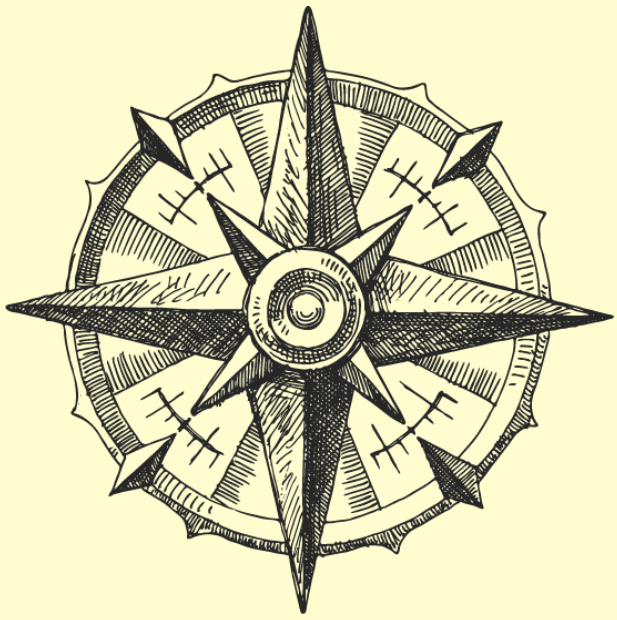
این کلاس هیچ داده ای جز یک سازنده ندارد .

رفتار ها :

- **سازنده**

تنها دلیل ایجاد این کلاس افزایش ماژوال بودن کد و قابل ارتقا و تغییر بودن کد بوده وگرنه ضرورتی بر ایجاد این کلاس وجود ندارد .

در واقع ما کارت های زرد را با این کلاس ایجاد می کنیم تا یک نقطه تمایز با کارت بنفش داشته باشیم .



کلاس Map :

ویژگس ها :

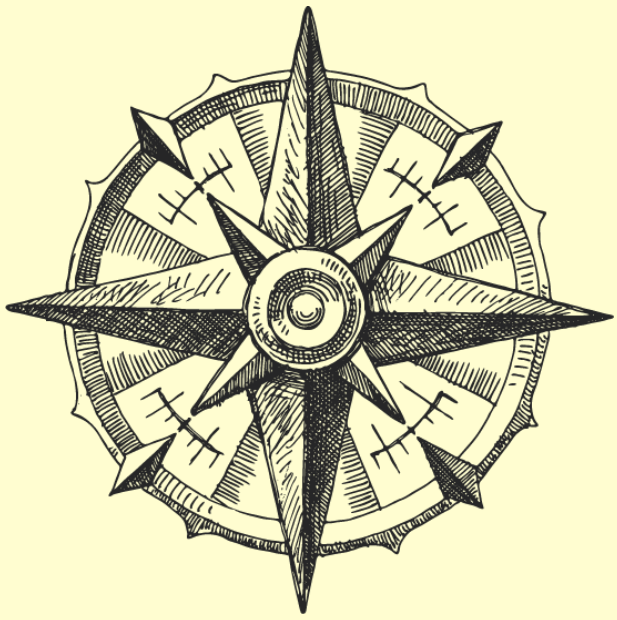
- **اندازه**
- **شهر ها**
- **ماتریس مجاورت شهر ها**

رفتار ها :

- **سازنده**
- **تشکیل دهنده و شکل دهنده نقشه**
- **گیرنده ها و ست کننده ها**

این کلاس نیز همانند YellowCard ضروری نبود اما برای افزایش ماژوال بودن و قابل تغییر بودن ایجاد شد که با تا توجه به اطلاعاتی که در اختیار دارم شهر ها ایجاد و همسایه ها آنها را مشخص و در نهایت به محض ایجاد به کمک سازنده این اطلاعات را در Game ایجاد می کند .

چالش کلی پیاده سازی map در پیاده سازی همسایگی ها بود که در نهایت از ماتریس مجاورت کمک گرفتیم .



کلاس City :

اول از هر چیزی دیدیم که با کلاس Map رابطه aggregation دارد و اگر مپ نباشد همچنان پابرجا است و فقط اطلاعات از دست می روند .

ویژگی ها :

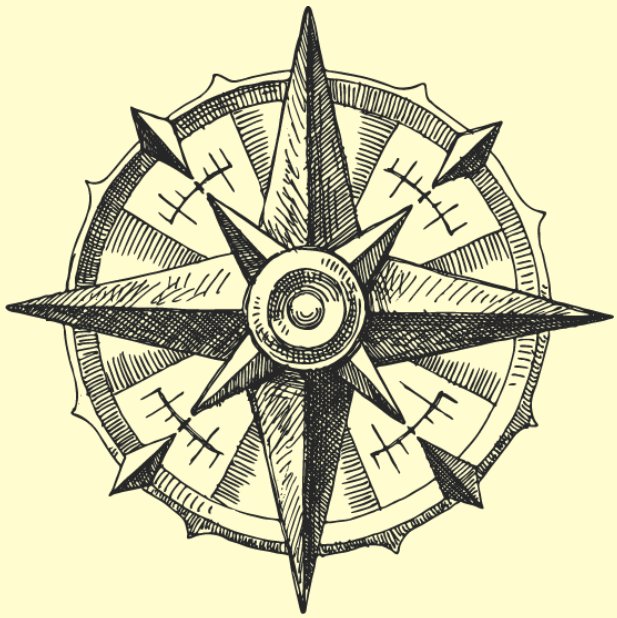
- شماره
- نام
- در دسترس بودن یا نبودن
- همسایه ها (آرایه از int بر حسب شماره شهر ها)

رفتار ها :

- ست کننده ها و گیرنده ها
- سازنده

این کلاس توضیح خاصی ندارد دقیقا یک شهر است با بخشی از اطلاعات شامل نام و همسایه و ... که در کلاس Map از آن برای ساخت یک مپ استفاده شد .

چالش این کلاس ذخیره همسایه ها بر حسب شماره برای راحت تر شدن جستجو بود .



کلاس Manager :

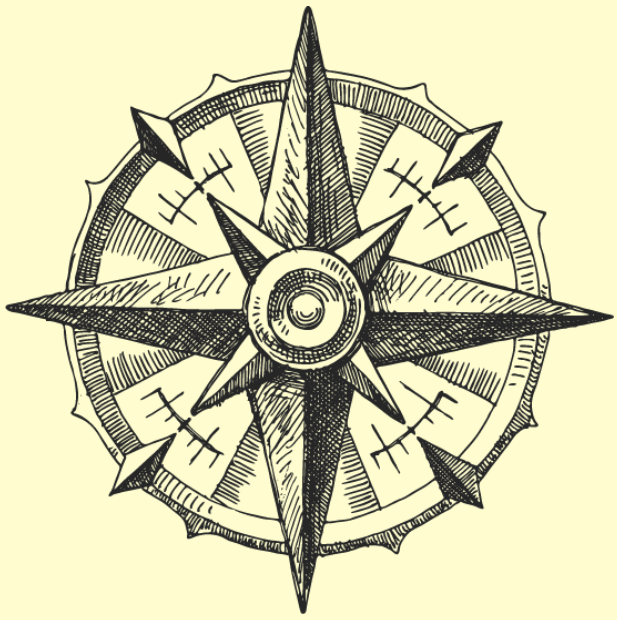
این کلاس برای مدریت کلی بازی ایجاد شده .

رفتار ها :

- **منوی بازی**
- **کمک و راهنمایی**
- **خروج**
- **پاک سازی اسکرین**

همانطور که از داده ها و رفتار ها واضح است برای امور مدریتی بازی است و اهمیت بالایی دارد چون کمک می کند چند نوع گیم را مدریت کنیم و برای سیو و خروج در ادامه اهمیت بالایی دارد .

چالش کلی ای وجود نداشت و اما نکته جدابی که ممکن است در فاز های بعد داشته باشه استفاده از فایل در این کلاس برای سیو و لود بازی است .



کلاس Game :

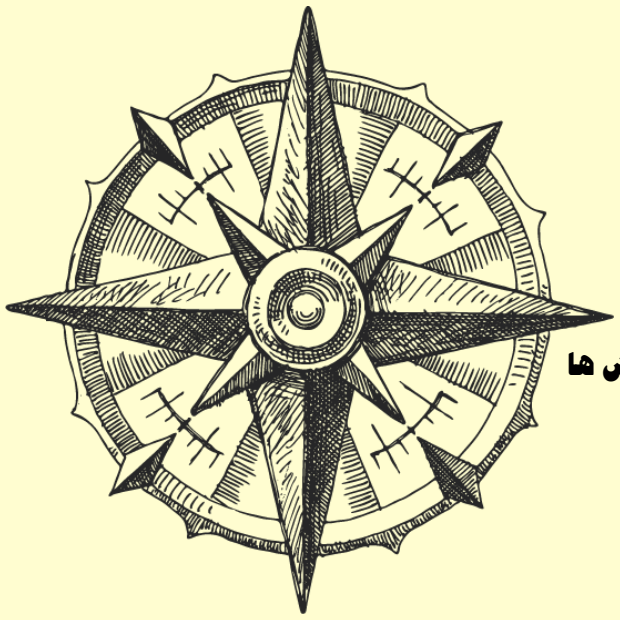
اساسی ترین کلاس که با لینک کردن کلاس ها با یکدیگر و ایجاد روابط بازی را شکل می دهد .

ویژگی ها :

- نقشه
- شهرها
- کارت ها
- بازیکن ها
- کارت های بازی شده
- مدیر
- نوبت
- شهر محل جنگ
- شهر محل صلح

رفتار ها :

- سازنده
- دریافت اطلاعات
- چاپ
- دست دادن
- ورودی
- تعیین محل جنگ
- پایات جنگ
- جریان بازی
- ایجاد کارت ها
- پیدا کننده برنده



این کلاس فعالیت مشخصی دارد پس سعی می کنم بشتر به چالش ها و سختی های پیاده سازی بپردازیم .

در این کلاس مشکلات فرآوان اند زیرا که بازی جزئیات فرآوان دارد که همه در اینجا برسی می شود .

از سوی دیگر برخی کارت های بنفش آنی اعمال نقش می کنند برخی در پایان جنگ که این امر باید مورد توجه باشد .

برای تعیین برد نهایی دائما نیاز به چک کردن بعد پایان هر جنگ داریم .

جمع بندی و منابع :

در این فاز به طور کلی برخی سرچ دم دستی در stackoverflow و geekforgheeks صورت گرفت و منبع خارجی جدی ای وجود نداشت .

در پیاده سازی فایل ها منظم پوشه به پوشه شدند و دیزاین شده اند .

به طور کلی بخش اساسی و زمینه کلی کار در این فاز ایجاد شده و پروژه به گونه ای اجرا شده که آماده فاز های بعدی باشد .