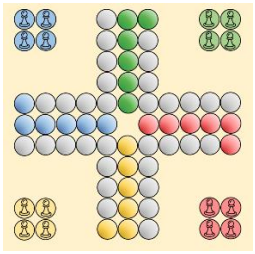
به نام خدا

پروژه پایانی برنامه‌نویسی پیشرفته



« بازی منچ »

فهرست مطالب

۱– مقـــــــــدمه

- پروژه‌ی جاری باید به صورت **انفرادی** پیاده‌سازی شود.
- برای پیاده‌سازی از نرم‌افزار **IntelliJ** و یا **Netbeans** استفاده شود.
- ارائه‌ی پروژه به صورت **حضوری** و در مکان و زمان مشخص شده در وبلاگ خواهد بود.
- در زمان تحویل پروژه، از تمرینات نیز (در صورت صلاحدید) سوال پرسیده می‌شود.
- مهلت تقریبی پروژه، یک هفته قبل از مهلت تایید نهایی نمرات است.
- نمره‌ی هر دانشجو، بسته به تسلط روی کد برنامه‌اش و کامل بودن برنامه‌اش محاسبه خواهد شد.
- ارائه با کامپیوتر شخصی دانشجو انجام می‌شود.

۲– صورت مسئله

پروژه‌ی پایانی این درس، بازی منچ است که دارای قابلیت‌های کلی زیر است. این بازی به صورت تمام گرافیکی و تمام شیءگرا نوشته می‌شود. نرم افزار موردنظر یک Java Application (یک برنامه عادی) است نه یک Applet یا ... .

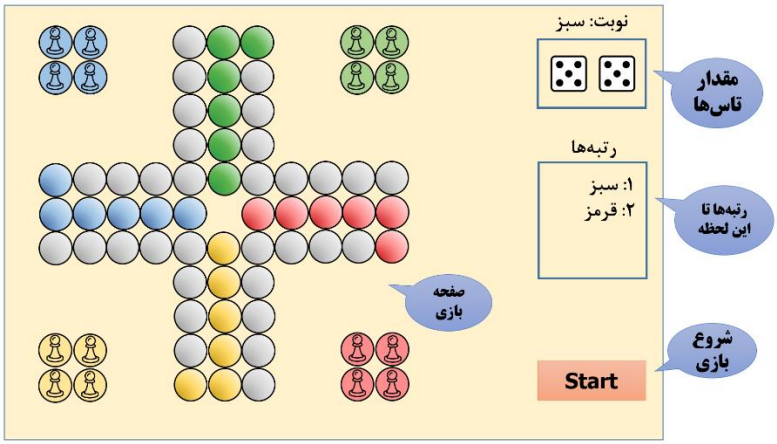
- منوی بازی
- صفحه اصلی بازی
- امکان ادامه بازی در زمانی دیگر
- به صورت چهار بازیکنه و هر بازیکن با طرز بازی متفاوت
- با دو تاس

۳– رابط کاربری نرم‌افزار

نرم‌افزار پس از اجرا باید دارای شکل تقریبی نشان داده شده در زیر باشد.

۳-۱- New Game

در صورتی که کاربر بر روی این گزینه کلیک کند، صفحه اصلی بازی ظاهر می‌شود. با کلیک کردن بر روی دکمه Start در صفحه اصلی بازی، بازی آغاز می‌شود. این صفحه به صورت تقریبی، باید شبیه به شکل زیر باشد.



۳-۲- Resume Game

در صورتی که بازی نصفه کاره‌ای وجود داشته باشد، این دکمه به کاربر نمایش داده می‌شود. در غیراینصورت باید آن را پنهان کنید. با کلیک بر روی این گزینه، بازی ناتمام قبلی از سر گرفته می‌شود.

۴– ذخیره‌سازی اطلاعات

تمامی اطلاعات مربوط به بازی در حال انجام در فایل ذخیره می‌شود. به صورتی که اگر کاربر در حین بازی، از برنامه خارج شود و دوباره وارد شود، امکان از سرگیری بازی از آخرین وضعیت، وجود داشته باشد.

۵– جزئیات پیاده‌سازی (منطق)

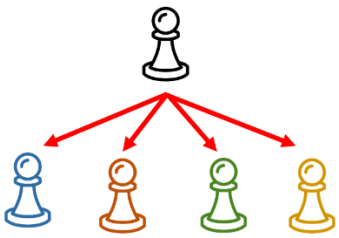
۵-۱- کلاس مهره (Piece)

مهره‌ها ابزار بازی بازیکن‌ها هستند. هر مهره دارای مشخصات زیر است:

- فیلد موقعیت: شماره خانه‌ای که مهره در آن قرار دارد را نگه می‌دارد.
- فیلد بازیکن: یک ارجاع به بازیکن مالکش نگه می‌دارد. در سازنده دریافت شده و قابل تغییر نیست.
- فیلد شماره: هر مهره شماره‌ای بین ۱ تا ۴ دارد.
- فیلد وضعیت: وضعیت مهره را مشخص می‌کند. (نوع شمارشی PieceStatus: IN_GAME, FINISHED_GAME, NO_STARTED_GAME)
  - خارج از بازی
  - در بازی
  - بازی را به پایان رسانده
- تابع paint(x,y): مختصات بالا-چپ خانه موردنظر را دریافت کرده و خود را در آن رسم می‌کند

۵-۲- کلاس بازیکن (Player)

در این بازی، ۴ نوع بازیکن وجود دارد که در شکل نشان داده شده‌اند. همه بازیکن‌ها از بازیکن عمومی به ارث می‌برند.



بازیکن پدر دارای فیلدها و توابع زیر است:

- فیلد وضعیت: وضعیت بازیکن را نگه می‌دارد (نوع شمارشی PlayerStatus: PLAYING, FINISHED).
- فیلد رنگ: رنگ در سازنده دریافت می‌شود و قابل تغییر نمی‌باشد (نوع شمارشی Color: BLUE, RED, GREEN, YELLOW).
- فیلد مهره‌ها (آرایه یک بعدی): هر بازیکن دارای چهار مهره (Piece) با شماره‌های ۱ تا ۴ است.

- فیلد صفحه بازی: هر بازیکن یک ارجاع به صفحه بازی در خود دارد.
- تابع dice: از شیء صفحه بازی دوبار درخواست انداختن تاس کرده و نتیجه را برمی‌گرداند.
- تابع move: این تابع ابتدا تابع تاس انداختن را فراخوانی کرده و بر اساس نتیجه‌ی آن، از صفحه بازی درخواست حرکت کردن به مقدار موردنیاز می‌کند و سپس مختصات مهره مربوطه را بروز می‌کند.
- تابع updatePeiceStatus(Piece piece, PieceStatus status): وضعیت جدید مهره را از صفحه بازی دریافت کرده و به مهره اعمال می‌کند.
- تابع paint: این تابع خود را بر روی صفحه بازی رسم می‌کند. برای رسم هر یک از مهره‌ها، مختصات مهره از صفحه‌ی بازی دریافت شده و سپس مهره در مختصات دریافتی، رسم می‌شود.

نکات:

- همه فیلدهای موردنیاز در بالا نامبرده نشده‌اند!
- کلاس بازیکن پدر قابل نمونه‌سازی نیست.
- بسته به منطق، انتزاعی بودن/نبودن توابع را تعیین کنید.
- شرط ورود به بازی، آوردن ۶ در یکی از تاس‌ها است.

۵-۲-۱- بازیکن سبز رنگ

هر بازیکن نتیجه‌ی دو تاس انداخته شده را به شکلی متفاوت محاسبه می‌کند. نتیجه تاس اول را dice1 و نتیجه تاس دوم را dice2 می‌نامیم.

بازیکن سبز رنگ نتیجه نهایی تاسش را به صورت زیر محاسبه می‌کند:

$$(dice1 + dice2) \% 6 + 1$$

منطق این بازیکن برای حرکت به صورت زیر است:

۱. در صورتی که مهره‌ای در خارج از بازی است، او را وارد بازی کن
۲. در غیر اینصورت، اولویت با حرکت دادن جلوترین مهره است

۵-۲-۲- بازیکن قرمز رنگ

بازیکن قرمز رنگ نتیجه نهایی تاسش را به صورت زیر محاسبه می‌کند:

$$(dice1 * dice2) \% 6 + 1$$

منطق این بازیکن برای حرکت به صورت زیر است:

۱. در صورتی که مهره‌ای از حریف قابل زدن است، آن مهره را بزن
۲. در صورتی که مهره‌ای در خارج از بازی است، او را وارد بازی کن
۳. در غیر اینصورت، اولویت با حرکت دادن جلوترین مهره است

۵-۲-۳- بازیکن آبی رنگ

بازیکن آبی رنگ نتیجه نهایی تاسش را به صورت زیر محاسبه می‌کند:

$$\big(max(dice1, dice2) - \min(dice1, dice2)\big) \,\%\, 6 + 1$$

منطق این بازیکن برای حرکت به صورت زیر است:

۱. در صورتی که مهره‌ای در خارج از بازی است، او را وارد بازی کن
۲. در صورتی که مهره‌ای از حریف قابل زدن است، آن مهره را بزن
۳. در غیر اینصورت، اولویت با حرکت دادن جلوترین مهره است

۵-۲-۴- بازیکن زرد رنگ

بازیکن زرد رنگ نتیجه نهایی تاسش را به صورت زیر محاسبه می‌کند:

$$\left(\frac{\max(dice1, dice2)}{\min(dice1, dice2)} * 7\right) \,\%\, 6 + 1$$

منطق این بازیکن برای حرکت به صورت زیر است:

۱. در صورتی که مهره‌ای درون بازی وجود دارد، اولویت با حرکت دادن تازه واردترین مهره است
۲. در غیراینصورت، مهره‌ای را وارد بازی کن

۵-۳- کلاس صفحه بازی (GameBoard)

صفحه بازی یک کلاس مجزا است که دارای فیلدها و توابع زیر است. در واقع بخش اعظم بازی توسط این کلاس انجام می‌گیرد.

- یک آرایه به طول چهار از چهار بازیکن
- یک آرایه یک بعدی: به اندازه‌ی ابعاد صفحه بازی از نوع Piece که تعیین می‌کند در هر خانه‌ای چه مهره‌ای قرار دارد.
- فیلد state: از نوع شمارشی State که دارای دو ثابت START و STOP است و وضعیت بازی را مشخص می‌سازد.
- فیلد رنگ پس زمینه: از نوع رنگ
- فیلد ثابت اندازه سلول (هر خانه برابر با ۵۰ پیکسل است: هنگام رسم دایره‌ها کاربرد دارد)
- تابع start: بازی آغاز می‌شود. اگر بازی قبلا در نیمه‌های خود متوقف شده است، ادامه می‌یابد. در غیر اینصورت از ابتدا شروع به کار می‌کند.
- تابع stop: بازی متوقف می‌شود.
- تابع move(Piece piece, int start, int steps): در صورتی که مهره مربوطه در خانه start قرار داشته باشد، او را به اندازه steps به جلو می‌برد. در صورتی که خانه‌ی مقصد با مهره دیگری در تداخل باشد، مهره مقصد از بازی خارج می‌شود (و وضعیت جدید مهره را به بازیکن مربوطه اطلاع می‌دهد). در صورتی که حرکت با موفقیت انجام شود true (وضعیت جدید مهره را به بازیکن اطلاع می‌دهد) و در غیراینصورت false برگردانده می‌شود. در صورت برگرداندن false، بازیکن می‌تواند تصمیم دیگری بگیرد.
- تابع moveIn(color): در صورتی که تعداد مهره‌های رنگ ورودی کمتر از ۴ عدد باشند (برای جلوگیری از تقلب)، او را وارد بازی کرده و مقدار true برمی‌گرداند. در غیراینصورت مقدار false برگردانده می‌شود.
- تابع pass: نوبت را به بازیکن بعدی می‌دهد.
- تابع isFilled(int dest): یک شماره خانه دریافت کرده و در صورتی که مهره‌ای از بازیکن‌های حریف در آن قرار داشته باشد باشد، true برمی‌گرداند.
- تابع paint: خود را رسم می‌کند.
- تابع getCellPosition(PieceState state, int position): شماره یک خانه را دریافت کرده و مختصات (x,y) آن را برمی‌گرداند. ورودی state مشخص می‌کند خانه مدنظر از چه نوعی است: خانه‌های داخل بازی، خارج از بازی و یا خانه‌های هدف
- تابع dice: یک عدد تصادفی بین ۱ تا ۶ تولید کرده و برمی‌گرداند.
- تابع load: اطلاعات ذخیره شده از بازی را بارگذاری می‌کند.
- تابع save: وضعیت جاری بازی را ذخیره می‌نماید.

نکات:

- بازی به صورت خودکار انجام می‌شود. هر چند ثانیه یک حرکت انجام می‌گیرد به شکلی که قابل مشاهده با چشم باشد.
- هیچ دو مهره‌ای حق قرار گرفتن در یک خانه را ندارد، حتی اگر از رنگ مشابه باشند.

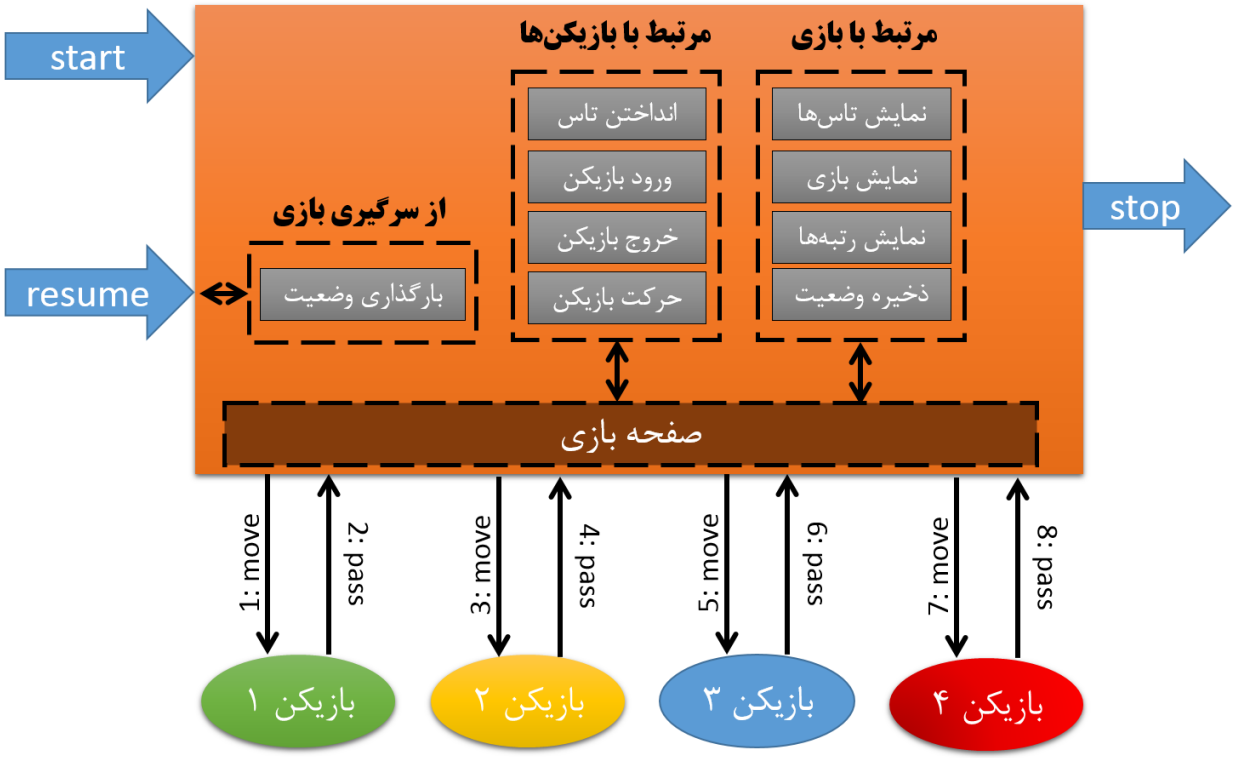
- در صورتی که تابع یا فیلد دیگری موردنیاز است، می‌توانید آن را اضافه کنید. به شرط اینکه اصول طراحی را زیر سوال نبرد!
- پس از هر حرکت، در صورتی که رنگ مشخصی همه بازیکن‌هایش را به خانه‌های پایانی رسانده بود، بازیکن مربوطه از دور بازی خارج شده و در فهرست رتبه‌ها ظاهر می‌شود.

نکات بسیار مهم:

- گرافیک بازی کاملا مستقل از منطق بازی است. همه کلاس‌هایی که در این سند توضیح داده شد، بخش منطق بازی را پوشش می‌دهد.
- در پروژه‌ی اصلی، تنها یک شیء از کلاس GameBoard ایجاد شده و همه امور توسط این کلاس مدیریت می‌شود.

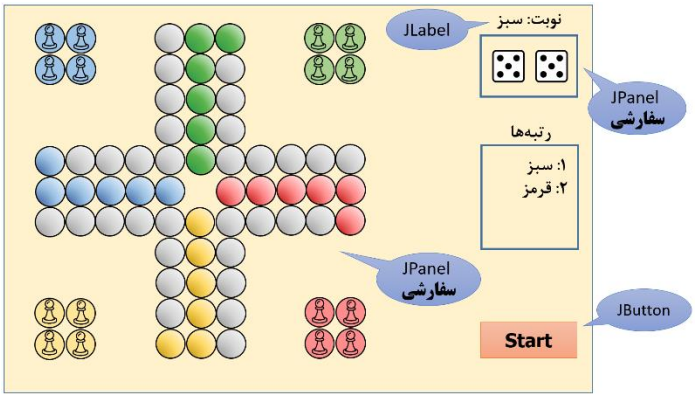
۵-۴- چرخه بازی

شکل زیر چرخه بازی را به تصویر کشیده است.



۶– جزئیات پیاده‌سازی (گرافیک)

برای طراحی گرافیکی بهینه، باید یک نگاشت مناسب بین هر یک از بخش‌های منطقی و گرافیکی یافت. همه پنجره‌های اصلی از نوع JFrame هستند و طراحی آنها ساده است. در ادامه به صفحه اصلی بازی می‌پردازیم که از پیچیدگی نسبی برخوردار است. همانطور که در شکل زیر مشاهده می‌کنید، پنجره اصلی بازی از بخش‌های زیر تشکیل شده است. در این شکل دو JPanel سفارشی دیده می‌شود.



JPanel سمت چپ، مهمترین نقش را دارد و صفحه بازی بر روی آن رسم خواهد شد. JPanel سمت راست (و بالا) برای نمایش آخرین تاس‌های انداخته شده است. بین JPanel سمت چپ و کلاس صفحه بازی (GameBoard) باید یک رابطه مستقیم برقرار باشد. هر دوی این کلاس‌ها باید یک ارجاع از هم در اختیار داشته باشند. هرگاه که کلاس GameBoard نیاز به رسم خود در صفحه داشت، باید به JPanel درخواست دهد تا او این کار را انجام دهد. خود JPanel با دوباره نویسی تابع paintComponent، کلاس GameBoard را رسم می‌کند.

اندازه JPanel سمت چپ به صورت دقیق قابل تعیین است. تعداد سطر و ستون های صفحه بازی مشخص است و همچنین تعداد پیکسل‌هایی که هر سلول اشغال می‌کند. در نتیجه عرض این کنترل برابر خواهد بود با "تعداد ستون ها * اندازه هر سلول" و همچنین تعداد سطرها برابر خواهد بود با "تعداد سطرها * اندازه هر سلول"

۷– تحویل پروژه

- تحویل پروژه به صورت حضوری صورت می‌گیرد.
- در پیاده‌سازی بخش‌هایی از پروژه که نحوه پیاده‌سازی آن‌ها صریحا بیان شده، **تنها** مجاز به استفاده از دستورات و مطالبی هستید که در کلاس آموزش داده شده است.
- برای بخش‌هایی که صحبتی در رابطه با نحوه پیاده‌سازی آن‌ها نشده، می‌توان نوآوری به خرج داده و از ایده‌های خود استفاده کنید.
- در صورتی که سوالی در مورد هر یک از بخش‌های پروژه دارید:
  ۱. در بخش نظرات وبلاگ بپرسید.
  ۲. فقط در صورتی که گزینه ۱ قابل اعمال نیست، از طریق ایمیل اقدام کنید.
- در صورت یافتن مشابهت در پروژه‌ی دو نفر، نمره‌ی پروژه به یکی از نفرات (به صورت تصادفی) داده شده و نمره **صفر** به گروه دیگر تعلق خواهد گرفت.

موفق باشید، بیگلری