



دانشکده مهندسی برق
دانشگاه صنعتی شریف

درس برنامه سازی شی گرا

نیمسال دوم ۱۴۰۲-۱۴۰۱
دکتر ونوقی وحدت، دکتر هاشمی

بارم

تمرين چهارم

۱. فقط ماسین حساب ساده مجاز است!

$0,25 + 0,75$ ۲۰۴۸ . ۲

$0,5 + 0,75$

مهلت تحويل:

جمعه ۲۰ خرداد ۱۴۰۲

توجه:

این تمرين در مجموع دارای ۲,۲۵ نمره می باشد؛ ۵/۱ نمره بارم اصلی تمرين چهارم، ۲۵٪ امتيازی تمرين سوم و ۵٪ دیگر می تواند امتيازی تمرين را جبران کند!

فقط ماشین حساب ساده مجاز است!

مهندس رادفرد که در تمرین قبلی دستورات گیت را پیاده سازی کرده، حال قصد دارد صفحه گیتهاب خود را ساخته و ارتقا بدهد. یکی از پژوههایی که برای تقویت صفحه به او پیشنهاد شده است، طراحی یک ماشین حساب است. اما از آنجایی که او از جاوا و شی گرایی سر در نمی آورد، شما را به او معرفی کرده ایم تا به او کمک کنید!

در این تمرین قرار است قسمت گرافیکی(front-end) یک ماشین حساب را پیاده سازی کنید.

فاز اول (اجباری)

ماشین حسابی که قرار است پیاده سازی کنید به این شکل است که ابتدا عملوند اول سپس عملگر و در آخر عملوند دوم را از کاربر گرفته و نتیجه را به او نمایش می دهد.

ساده ترین عملگرهایی که این ماشین حساب توانایی محاسبه آنها را دارد عبارتند از:

- Plus (+)
- Minus (-)
- Divide (/)
- Multiply (*)

قسمت خام logic ماشین حساب قرارداده شده است و برای این فاز کافیست قسمت گرافیکی برنامه را پیاده سازی کنید. به دلیل برخی اهداف آموزشی، دکمه های این بخش به صورت طراحی شده در اختیار شما قرار گرفته و باید از عکس های موجود در دایرکتوری `src/main/resources` کمک بگیرید و رابط کاربری گرافیکی بخش استاندارد ماشین حساب را بسازید.

انتظار می رود در آخر این بخش چنین شکلی تولید کنید:



که در آن دکمه ! به معنای clear می باشد، یعنی وقتی فشرده شود، می توانیم مجدداً عملوند اول را بنویسیم.

برای پیاده سازی به نکات زیر توجه کنید:

- ورودی ماشین حساب فقط از طریق کلیک کردن ماوس است.
- برای ساده سازی می توانید طوری پیاده سازی کنید که روند زیر در زمان اجرا طی شود:
 - a. عدد اول را وارد می کنید.
 - ii. عملگر را انتخاب می کنید.(در این زمان صفحه نمایش ماشین حساب خالی می شود یعنی عملوند اول و عملگر ذخیره می شوند)
 - iii. عدد دوم را وارد می کنید.
 - iv. دکمه = را انتخاب می کنید.(در این زمان عملوند دوم هم ذخیره می شود پس صفحه نمایش پاک می شود و نتیجه را نمایش می دهد)
 - v. حال می توانید با زدن دکمه clear به مرحله یک برگردید. یا اینکه نتیجه عملیات قبل را به عنوان عملوند اول عملیات جدید در نظر بگیرید و به مرحله دو برگردید.

مثال :

اگر دکمه های زیر به ترتیب از چپ به راست وارد شوند، باید نتایج زیر را مشاهده کنیم :

- $1,2,+2,=,+,1,= -> 15$
- $1,2,+2,=,!1,*4,= -> 4$

نکات قابل توجه:

- در صورت استفاده از بخش **logic** داده شده، احتمالا در صورت انجام محاسبه با اعداد صحیح، بخش اعشاری نیز دریافت کنید، مثلا در مورد اول مثال قبلی به جای 15 عدد 15.0 نمایش داده می شود. این اشکالات را رفع کنید.

- تضمین می شود که در هر مرحله بیش از 2 عملوند داده نمی شود، با این حال در صورتی که توالی عملگر و عملوند اشتباه وارد شود، نباید برنامه متوقف شود بلکه باید خطای مناسبی نمایش داده شود و ماشین حساب دوباره آماده دریافت عملوند گردد.

فاز دوم (اجباری)

از آنجایی که مهندس رادفرد بسیار فراموشکار است (!)؛

در این قسمت، قرار است امکان مشاهده تاریخچه محاسبات و عملیات ها را به ماشین حساب بیفزاییم. طراحی آن می تواند به صورت دکمه **History** (مانند ماشین حساب گوشی) و یا گزینه ای در سایدبار (**sidebar**) انجام گردد (در ادامه توضیح داده می شود) و محدودیتی بابت طراحی آن وجود ندارد. همچنین در صورت استفاده از دکمه، مکان آن روی صفحه اصلی ماشین حساب اهمیتی ندارد و کافی است با ابعاد طراحی قبلی سازگار باشد.

تاریخچه 10 محاسبه قبلی باید در صفحه (استیج) جدیدی و کاملا جدا از ماشین حساب اصلی به ترتیب (از قدیم به جدید) نمایش داده شود. جزئیات طراحی این صفحه به سلیقه شما می باشد. این صفحه مادامی که بسته (**Close**) نشود باز می ماند اما نیازی به آپدیت کردن آن به هنگام عملیات نیست. (پیاده سازی این مورد امتیازی خواهد بود و توضیح داده خواهد شد)

فاز سوم (امتیازی)

از آنجایی که مهندس رادفرد این ماشین حساب را برای کار های علمی سنگین می خواهد ؟

در این قسمت، مود علمی (**Scientific**) طراحی می گردد. در این بخش ابتکار عمل کاملا به دست شماتست و طراحی صفحه و دکمه های این بخش به شما داده نمی شود بلکه باید به کمک کلاس **Button** و به سلیقه خودتان صفحه و دکمه های مربوطه را طراحی کنید.

* نحوه انتقال به مود علمی و بازگشت از آن

انتقال از حالت استاندارد به حالت علمی می تواند به دو صورت زیر انجام گیرد: (انتخاب دلخواه است)

(1) به کمک طراحی دکمه **Standard** و دکمه **Scientific** برای بازگشت به حالت استاندارد

(2) به کمک ساید بار (همانند ماشین حساب ویندوز)

- دقتشود که صفحه مربوط به حالت علمی ماشین حساب در همان صفحه (استیج) مربوط به صفحه اصلی ماشین حساب باید باشد و صفحه (استیج) جدیدی ایجاد نمی شود.

* دکمه های اضافه شده در حالت علمی:

$x^2, x^y, \log, n!, \text{mod}, e, \pi, \sqrt{n}, \sin(\text{Other Trigonometrics}), |x|$

دکمه های بیشتری نیز میتواند در صورت تمایل شما اضافه شوند. دقت کنید بعضی از علمگر های این بخش تک عملوند می باشند!

موارد امتیازی بیشتر! (بارم بندی هر بخش هنوز مشخص نشده است و وزن یکسانی ندارند)

- طراحی صفحات تاریخچه و مود scientific به کمک fxml (پیشنهاد TA)

- هندل کردن محاسبات ممیزشناور (در مود علمی)

- هندل کردن محاسبات پرانتزدار به کمک دکمه های (و) (در مود علمی)

- اضافه کردن مموری و دکمه های مربوطه (در مود علمی)

- آپدیت شدن تاریخچه همزمان با انجام عملیات

- طراحی و استایل المان های داده نشده به کمک CSS (پیشنهاد TA)

- استفاده از سایدبار همانند ماشین حساب ویندوز برای تاریخچه و حالت علمی

- تعویض تم و گرافیک زیبا

- پیاده سازی بخش های مشابه در ماشین حساب ویندوز همانند ماشین حساب باینری یا تبدیل یکاهای و مبنایها

- هرگونه خلاقیت معقول و جذاب با صلاحیت TA

توضیحات:

توجه داشته باشید که کلاس هایی که شرح داده شده اند، لازم هستند اما لزوماً کافی نیستند، بنابراین اجازه دارید کلاس های جدیدی ایجاد کنید و یا کلاس های موجود (مخصوصاً logistic) را تغییر دهید.

و دوباره داستان!

آراد، برای زدن پروژه مبانی سال 97 استخدام شده بود. او برای این که بتواند سازماندهی شده تر فکر کند، تعدادی کلاس Abstract برای پروژه تعریف کرده بود؛ اما در صبح یک روز برفی، وی که به جلوی پایش نگاه نمی کرد، داخل چاهی افتاد؛ و مرد. حال، عموراً دفرد، دوست زمان جوانی او، از شما خواسته که کار ناتمام او را تمام کنید تا روح آراد بتواند بالاخره به آرامش برسد.

2048

در این سوال شما قرار است که بازی 2048 کلاسیک را به زبان Java، به طور کامل پیاده سازی کنید. در این بازی، شما یک جدول 4×4 دارید، که داخل این جدول، می تواند بلوک هایی قرار بگیرد که حاوی توان های طبیعی 2 است، یعنی 2، 4، 8، هدف بازی این است که با ترکیب بلوک هایی که عدد یکسانی دارند، بلوک های بزرگتری بسازید و به عدد 2048 برسید.

- بازی در نسخه ای که شما پیاده سازی خواهید کرد، با رسیدن به عدد 2048 تمام نمی شود، اما اگر دوست داشتید به طور اختیاری می توانید پیام تبریک نشان دهید!

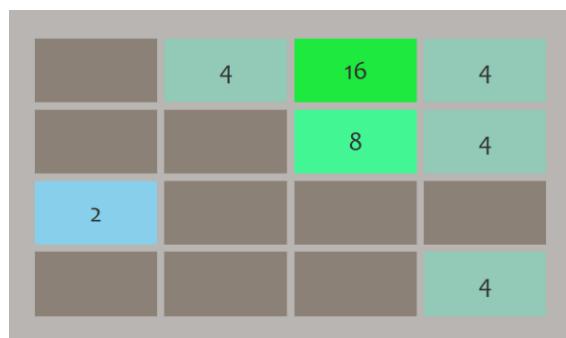
بازی در ابتدا با دقیقاً دو بلوک شروع می شود. با هر حرکت بازیکن باید یک بلوک جدید در صفحه پدیدار شود. بدیهی است که باید خانه ای که بلوک در آن به وجود می آید، خالی باشد (جلوتر مشخص شده است که برای این کار از چه تابعی استفاده خواهید کرد). بلوکی که به وجود می آید، می تواند عدد 2 یا 4 باشد؛ که باید به صورت تصادفی باشد. احتمال این که بلوک 2 ظاهر شود، باید بیشتر از 4 باشد. پیشنهاد می شود که به احتمال 75٪، هر بلوکی که ساخته می شود 2 باشد و در غیر این صورت، 4 باشد.

بازیکن، با فشردن دکمه های WASD، کنترل خواهد کرد که چه اتفاقی در بازی می افتد. با فشردن کلید W، باید طوری که انگار از بالا جاذبه اعمال شده است، بلوک ها به سمت بالا پرتاپ شوند. بلوک هایی که عدد یکسانی دارند، با هم ترکیب می شوند و یک بلوک با دو برابر عددی که قبلاً بودند را می سازند. به طور مشابه، این اتفاق برای فشردن A، برای پرتاپ شدن بلوک ها به سمت چپ، برای D به سمت راست، و برای S به پایین می افتد. برای انجام این کار می توانید از چنین کدی استفاده کنید:

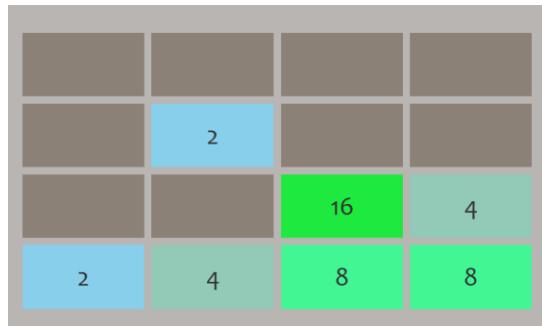
```
yourScene.setOnKeyPressed(e-> {
    if (e.getCode() == KeyCode.S)
        YourControllerClass.downPressed(); // do something here
    ...
})
```

بدیهی است که پس از پرتاپ شدن بلوک ها، باید بلوک جدید در جدول تشکیل شود، نه قبل. همچنین، گاهی اوقات یک حرکت بی اثر است (مثل راست و چپ رفتن در شکل 3). در حرکات بی اثر، نباید بلوک جدیدی ساخته شود.

مثالی از فشار دادن S در زیر برای شما قرار داده شده است:

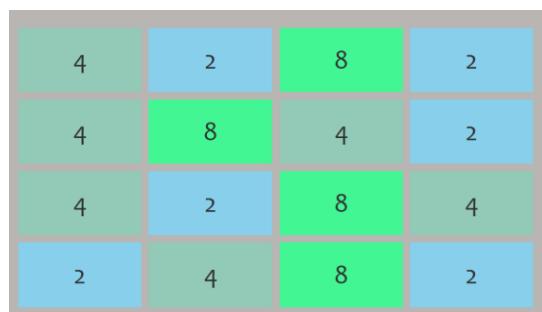


شکل 1- قبل از فشردن S



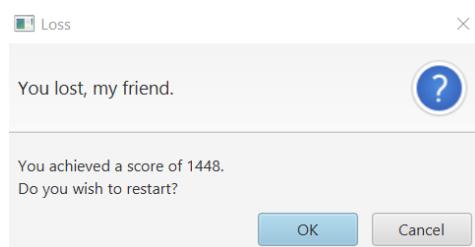
شکل 2- پس از فشردن 5 (پیک بلوك 2 هم به طور تصادفي ايجاد شده و دو تا 4 با هم تشکيل 8 داده اند)

بازى وقتی تمام می شود که تمام خانه ها پرشده باشند، و هیچ حرکت دیگری نتوان انجام داد. برای مثال، بازی در شکل زیر هنوز بازی تمام نشده است، چون می توان به بالا و پایین حرکت کرد و تعدادی بلوك را با یکدیگر ترکیب کرد تا جدول خالی تر شود.



شکل 3- بازی هنوز تمام نشده است!

پس از تمام شدن بازی، پنجره ای باز کنید که کاربر بتواند انتخاب کند دوباره بازی کند یا خیر.



شکل 4- پنجره باخت

محتواي پيامي که نشان می دهد لازم نیست دقیقا همین پیام باشد. با فشار دادن Cancel، بازی جدیدی ساخته نمی شود، و با فشردن OK یک بازی جدید از اول ساخته شود. توجه کنید که همهی این ها در همان پنجره اولیه باید رخ دهد. از طرفی، پنجره قبلی باید غیرفعال بماند تا وقتی این پنجره بسته نشده است.

- پیشنهاد می شود برای انجام این کار، از javafx.scene.controls.Alert استفاده کنید تا لازم نباشد زحمت اضافه برای یک پنجره بسیار ساده بکشید.

علاوه بر این، جایی در صفحه بازی، باید امتیاز بازیکن به صورت زنده نمایش داده شود.

- برای محاسبه امتیاز، هر بار که دو بلوك با هم ترکیب می شوند، عدد بلوك جدید با امتیاز قبلی جمع زده می شود. برای مثال اگر فقط دو 16 با هم ترکیب شوند و امتیاز فعلی 144 باشد، امتیاز جدید باید 166 باشد.

دو دکمه هم باید پیادهسازی کنید. دکمه اول Reset است که بازی جدیدی را از اول می‌سازد (مشابه زدن OK در پنجره‌ای که بعد باخت باز می‌شود). دکمه دوم، دکمه Debug است. با فشردن این دکمه، باید در Console، جدول بازی پرینت شود. شما مختار هستید که فقط توانهای دو از بلوک‌های باز را چاپ کنید، یا خود عدد هر بلوک را. انجام این کار، هنگام دیباگ کردن مشکلات احتمالی که به آن‌ها برخواهد خورد، بسیار کمک‌کننده است؛ چون می‌توانید بفهمید مشکل از منطق بازی است یا از گرافیکی که برای بازی پیادهسازی کرداید. می‌توانید هر چیز دیگری که نیاز دارید هم چاپ کنید.

Abstraction

تعدادی کلاس آبستره (abstract) برای شما قرار گرفته است. در ابتدا، فایل AbstractClasses.zip را دانلود کرده، و سپس آن‌ها را فolder src پروژه قرار دهید. اگر پروژه javafx ساخته‌اید، کلاس‌ها را در مسیر [YourProjectFolderHere]/src/main/java آن‌ها قاطی نشود.

کلاس‌هایی که برای شما قرار گرفته‌است، تعدادی کلاس مربوط به منطق بازی است؛ و هیچ ربطی به گرافیک ندارد (0 تا 100 گرافیک بازی به عهده خود شماست). هدف از این کلاس‌ها، دادن مسیر به کد منطق بازی شماست، و اگر از این کلاس‌ها استفاده نکنید و یا تغییری در آن‌ها ایجاد کنید، نمره کسر خواهد شد. دو کلاس آبستره داخل فایل زیپ وجود دارد، که بیشتر حجم کار شما در قسمت منطق بازی با آن‌ها خواهد بود. شما باید کلاس خودتان را بزنید و این کلاس‌ها را extend کنید؛ و توابعی که مشخص شده را پیادهسازی کنید. جلوتر به کاربرد هر کلاس اشاره خواهیم کرد.

Direction

این کلاس یک enum است که شامل 4 جهت اصلی است. این enum در حرکت بلوک‌ها به درد خواهد خورد.

GridPoint

این کلاس یک record جاوا است. رکوردهای جوا راهی برای کوتاه‌نویسی کلاس‌هایی هستند که فقط شامل چند فیلد هستند. هدف این کلاس این است که جای هر بلوک در جدول 4x4 با یک شیء بخصوص، مشخص شود تا کد تمیزتر شود.

داخل این کلاس، تعدادی متدهایی هست که باید خودتان پیادهسازی کنید. دو متدهای isIllegal و neighbors از قبل برای شما آماده هستند. متدهای isIllegal چک می‌کند که آیا این خانه در یک جدول 4x4 می‌تواند باشد یا خیر (اندیس‌ها از 0 تا 3 در نظر گرفته شده‌اند).تابع neighbors برای شما همسایه‌های هر خانه را می‌دهد. ابتدا هر 4 جهت اصلی را تست می‌کند، و سپس حالت‌های غیرمجاز را با استفاده از isIllegal فیلتر می‌کند. متدهای equals، hashCode، toString و left، right، down، up، toPoint، equals و left را باید خودتان پیادهسازی کنید. استفاده این متدها، باید منطقاً جلوتر و قتی بقیه تمرین را کامل می‌کنید مشخص شود.

Colorpicker

یک کلاس جانبی است که رنگ‌های هر بلوک را مشخص می‌کند. رنگ‌های مورد نظر خود را در color قرار دهید.

AbstractBlock

این شیء، نسخه آبستره خود بلوک است. هنگام حرکت بلوک‌ها، هر بلوک برای خودش منطقی را اجرا می‌کند که باید در تابع move پیادهسازی شود. هر بلوک‌جایی در جدول را به خود اختصاص داده است، که با یک GridPoint مشخص می‌شود. در نهایت، پس از حرکت هر بلوک، باید یک نقطه جدید به آن اختصاص داده شود که با changePoint انجام می‌شود.

شما در نهایت باید کلاس Block خودتان را پیادهسازی کنید، و AbstractBlock را extend کنید. اگر از IntelliJ استفاده می‌کنید، پس از اینکه کلاس AbstractBlock را extend کرده، به اروری خواهید خورد مشابه این ارور:

Class 'Block' must either be declared abstract or implement abstract method 'changePoint(GridPoint)' in 'AbstractBlock'

روی این ارور راست کلیک کنید و Implement methods را انتخاب کنید. روی Ok کلیک کنید و متدهای مشخص شده را پیادهسازی کنید. یقیناً قرار است متدهای دیگری هم داخل کلاس پیادهسازی کنید، بسته به نیازی که خودتان در حین انجام کار پیدا خواهید کرد.

- توابع انیمیشن هم داخل کلاس آبستره هستند و مربوط به قسمت امتیازی تمرین‌اند. اگر تصمیم بر نمره امتیازی انیمیشن ندارید، تابعی که در کلاس خودتان پیاده سازی می‌کنید را خالی بگذارید (ولی ساختار کلی کد را بر هم نزنید).

AbstractBlockgrid

این شیء، نسخه آبستره شیء جدولی است که باید پیاده‌سازی کنید. این جدول شامل 4×4 بلوک است. توابعی که باید پیاده‌سازی کنید عبارت‌اند از:

- 1 آیا بازی با توجه به منطق بازی تمام شده است؟ **:checkEnd**
- 2 آیا در نقطه مدنظر، بلوک وجود دارد یا خالی است؟ **:isBlockEmpty**
- 3 نقطه‌ای خالی به صورت تصادفی در جدول پیدا کنید. آیا ارتباطی با تابع بالا احساس می‌کنید؟ **:getRandomEmptyPoint**
- 4 آیا اگر بدانید که بازی قطعاً تمام شده است، می‌توانید از یک روش بازگشتنی استفاده کنید؟ **:addNewBlock**
- 5 با هر ورودی کاربر، باید منطقی (احتمالاً) متفاوت روحی جدول اعمال شود. این توابع برای اعمال حرکات روی بلوک‌های داخل جدول و حرکت کردنشان در نظر گرفته شده است. تابع **up** قبل از **up** برای شما پیاده‌سازی شده است. همچنین، می‌توانید روش متفاوت خودتان را پیاده‌سازی کنید.
- 6 **getter :getBlock** برای بلوک‌های داخل جدول. داشتن این تابع به شما کمک خواهد کرد که به جای کار با **فیلد** (که یک آرایه دو بعدی است) مستقیماً از طریق **GridPoint**‌ها با جدول ارتباط برقرار کنید.
- 7 **setter :setBlock** برای بلوک‌های داخل جدول.
- 8 چاپ پیام مربوط به دکمه **DebugMessage**.

نمره امتیازی!

- پخش کردن موزیک و افکت‌های صوتی: دو **Slider** باید قرار دهید که شدت صدای این دو را به طور مجزا از هم کنترل می‌کند. برای افکت‌های صوتی، باید حداقل دو صدای مختلف را داشته باشید:

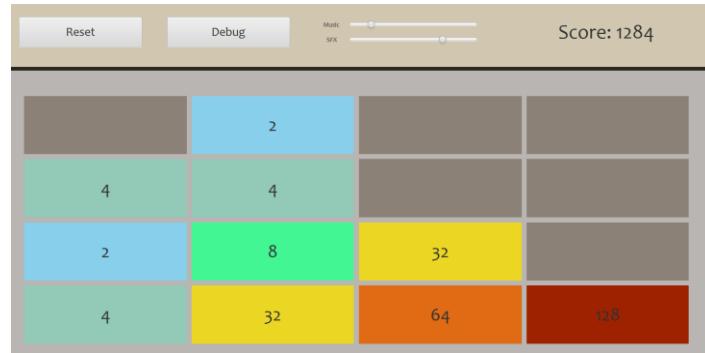
- 1 صدای حرکت بلوک‌ها هر وقت که کاربر حرکتی می‌کند
- 2 صدای ترکیب شدن دو بلوک

- انیمیشن صحیح حرکت بلوک‌ها: باید حداقل دو انیمیشن قرار دهید (نمره‌ی هر کدام جدا است):
 - 1 ساخته شدن بلوک جدید: وقتی که یک بلوک جدید ساخته می‌شود (نه ترکیب)، به جای اینکه از عالم نیستی، یهودی پدیدار شود، انیمیشنی قرار دهید که در طول زمانی مشخص، سایز بلوک جدید را از 0 تا 1.0 (سایز اصلی) تغییر دهد. استفاده از هر نوع **interpolation**، نمره امتیازی بیشتری دارد! ولی انیمیشن‌های اضافه (مثلا از 0 تا 1.2 برود و سپس از 1.0 تا 1.2 برگردد) نمره اضافه ندارد. پیشنهاد می‌شود از **javafx.animation.ScaleTransition** استفاده کنید.

- حرکت بلوک‌ها از خانه به خانه‌ی بعدی: به جای اینکه بلوک‌ها با هر حرکت کاربر، به خانه‌ای که منطق بازی حکم می‌کند تله‌پورت کند، انیمیشنی قرار دهید که طی زمانی مشخص، تمام بلوک‌ها به صورت همزمان به خانه‌ای که باید بروند، حرکت کنند. استفاده از هر نوع **interpolation**، نمره امتیازی بیشتری دارد! این حرکت باید خالی از هرگونه ایجاد باشد و طبیعتاً کاربر بتواند وسط انیمیشن حرکت جدیدی انجام دهد. پیشنهاد می‌شود برای این کار از **javafx.animation.TranslateTransition** استفاده کنید و برای مجاز کردن حرکت کاربر پیشنهاد می‌شود از چیزی مشابه قطعه کد زیر استفاده کنید:

```
yourTransition.setOnFinished(e-> {
    YourControllerClass.endTranslateAnimation(); // do what is needed here
});
```

- پیاده‌سازی گرافیک به صورت **Responsive**: طوری گرافیک را طراحی و پیاده‌سازی کنید که با تغییر اندازه پنجره، تمامی اجزای بازی (جدول، بلوک‌ها و انیمیشن‌هایشان، دکمه‌ها، امتیاز، ...) به تابعیت از تغییر اندازه پنجره تغییر کنند و:
 - 1 نیست و نابود نشوند، و همواره حضور داشته باشند.
 - 2 نسبت اندازه‌ها ثابت باشند.



شکل ۵ و ۶ بودن گرافیک responsive

بیشنهاد می‌شود از Container هایی مثل javafx.scene.layout.AnchorPane و javafx.scene.layout.GridPane یا استفاده کنید چون این Responsive design داخل آن‌ها از قبل تعییه شده‌است و شما کار زیادی لازم نیست بکنید. حساسیتی روی مسائل جزئی‌تر مثل اندازه font و یا قفل کردن اندازه پنجره (از جایی بیشتر کوچک نشود) نیست و اختیاری است.

- یک آیکون برای برنامه قرار دهید.



چند نکته و پند...

- برای این تمرین شما مجاز به استفاده هر نوع Dependency هستید؛ ولی نمره امتیازی ندارد.
- سعی کنید این تمرین را بدون استفاده از هرگونه تصویر پیاده‌سازی کنید.
- استفاده از فایل .css و .fxml. همراه با نرم‌افزارهای کمکی مثل SceneBuilder بسیار کمککننده است و کار شما را در نهایت بسیار راحت‌تر خواهد کرد.