Úvod

Původně jsem tuhle dokumentaci chtěl psát jako souhrn tříd a metod a popis ke všemu, co co přesně dělá a co jak funguje. Již po první velké úpravě v projektu jsem ale zjistil, že to je nereálné takhle dodržovat, tak jsem se rozhodl tuto dokumentaci vést spíše formou jakéhosi changelogu.

Jelikož používám IntelliJ, a ta automaticky maže obsah v adresáři bin, přesunul jsem tak složku data do rootu.

Pro spuštění z příkazové řádky stačí jako parametr zadat relativní adresu souboru (např. „data/lenna.pgm“).

Stručně funkčnost

Obrázek se načte přes parser třídy `NacitaniDat` a uloží se do pole intů. Následně `SpravceOkna` zařídí, aby se vykreslilo okno a v něm je vložena instance třídy `DrawingPanel`. `DrawingPanel` obrázek vykreslí pomocí `BufferedImage` a zařídí jeho bilineární interpolaci přes `AffineTransform`.

`DrawingPanel` si také nalezne `minimum`, `maximum` a `stoupání` v obrázku a zapomocí metody `drawArrow()` vykreslí správně ukazující šipky na daná místa s daným popiskem.

Dále se `DrawingPanel` stará o vykreslování vrstevnic a barevných ploch těchto vrstevnic. Vrstevnice jako čáry jsou vykreslovány po bodech, zatímco barevné plochy se vykreslí jako `BufferedImage`, který je opět zvětšen přes bilineární interpolaci za pomoci `AffineTransform`.

Třída `LegendaPanel` se stará o vykreslení přehledné legendy pro vizualizaci vrstevnic.

Jako vlastní vylepšení jsem přidal `ButtonPanel`, který zajišťuje funkční tlačítka v dolní části okna, kterými lze obrázek exportovat do požadovaného formátu, skrývat a odkrývat šipky a vrstevnice, zobrazovat grafy, nebo např. se lze jednoduše mezi obrázky přepínat.

Grafy se vykreslují pomocí knihovny `JFreeChart`. Třída `SpravceGrafu` a její metody zajišťují správná data a jejich vizualizaci do `ChartPanelu`.

Uživatel má možnost kolečkem myši obrázek přibližovat a oddalovat a poté levým stisknutím tlačítka se může intuitivně po obrázku posouvat.

Pro podrobnější informace o implementaci je nutné si pročíst zbytek dokumentace.

Volitelná rozšíření

- **Tisk** (2 body)
- **Export do PNG** (3 body)
- **ASCII Art** (3 body)
- **Zvětšení a posun** (4 body)
- **Export do SVG** (3 body)

Changelog

Verze 0.1.0 – 07. 03. 2021

Text

Dnes jsem začal pracovat na semestrální práci. Nejdivnější na tom asi je, že mi to ani nepřišlo, že dělám něco na seriózního, co se bude na semestrální práci hodit. Už dlouhou dobu mě zžíralo a zajímalo, co to vlastně v těch .pgm souborech je. Tak jsem si řekl, že si prostě udělám parser a vizualizuji tyto data.

Začal jsem hodně špatným parserem, co jen načetl data do listu Stringů a vždy jsem staticky určoval index, na kterém je rozlišení a index, od kterého pak začíná obrázek. Všechno tohle, se odehrává v metodě `nactiData()`. Dále jsem vytvořil metodu `zalozOkno()`, která prostě a jednoduše založí standardní JFrame okno, na které se pak přidává instance třídy `DrawingPanel`.

Ve třídě `DrawingPanel` je samozřejmě metoda `paint()`, ta si definuje škálování podle třetího cvičení. Pak se ve dvou smyčkách `y` a `x` od hodnot 0 do hodnot `sirka` a `vyska` (definované z `nactiData()`), a prochází se list Stringů a podle přislušné iterace se volí barva, kterou se bude kreslit. Pak už se jen vykresluje obdélníček z bodu [`x * scale` ; `y * scale`] a je široký a vysoký právě `scale` (`scale` je minimum ze (`scale_x`, `scale_y`)).

Najednou to vše funguje a vidím podle hory.pgm, že je to správně! Vidím dokonce i krásnou slečnu, lenna.pgm. Random.pgm a rbf.pgm fungují taktéž… Problém nastal u plzen.pgm a upg.pgm. Otevřel jsem si tak plzen.pgm v notepadu a zjistil jsem, že obsahuje více údajů na jednom řádku. Bylo tedy zapotřebí do hlavní třídy přidat metodu `rozkrajejRadky()`. Tato metoda vezme list dat a po řádcích ho projede a každý řádek si rozkrájí s oddělovacím symbolem mezera (opět to zatím není moc obecné, ale šlo mi opravdu jen o to vidět ty obrázky). Pak se tyto „rozkrájené" řádky načtou do nového listu a ten se vrací do metody `nactiData()`, která ho pak rovněž vrací jako svou návratovou hodnotu. Další problém s plzen.pgm je ten, že obsahuje hodnoty mnohem větší, než 255. Musel jsem proto vymyslet přeškálovávač barev, jelikož jsem nepřišel na žádnou rozumnou funkci, rozhodl jsem se to udělat velice jednoduše. List si seřadím, najdu největší a nejmenší hodnotu, a pokud je jejich rozdíl větší než 255 anebo je `maximum` větší než 255, vydělím `minimum` a `maximum` 2 a do počítadla si přičtu 1. Výslednou barvu jsem pak určoval jako (`hodnota[index]` – `minimum`) / `pocitadlo`.

Woala, najednou vidím i krásnou Plzeň. Jsem velice spokojen s výsledkem a po asi 5 hodinách práce jsem si šel přečíst zadání. Zjistil jsem tak, že jsem prakticky splnil část 1, základní funkční požadavky, až na škálování pomocí bilineární interpolace (používám prosté škálování ze cvičení 3).

Upg.pgm je stále tajemstvím, protože čínsky neumím (tento soubor mi ne jednou crashnul notepad s tím, jak je obsáhlý, ale vizualizovat mi ho zatím nejde).

Zatím práce není vůbec ideální, parser nestojí za nic, přeškálování barev je také při nejmenším zajímavé, ale jsem spokojen s tím, že vidím obrázky.

Changelog

Verze 0.1.0

Hodin strávených na této verzi: 3

třída `Mapa_SP2021`

- přidána metoda na načtení dat do listu Stringů, `nactiData()`
- přidána metoda na rozkrájení řádků dat, protože jeden řádek může obsahovat více dat, `rozkrajejRadky()`
- přidána metoda na založení okna, které si na sebe přidá instanci třídy `DrawingPanel`, `zalozOkno()`
- přidána metoda na založení chybového okna, neexistuje-li zadaný soubor pro načtení, `oknoError()`
- upravena metoda `main()`, aby využívala metody `nactiData()`, `zalozOkno()` a `oknoError()`

přidána třída `DrawingPanel`

- přidána metoda, co se stará o vykreslování obrázků z načtených dat, `paint()`
- přidána metoda na přeškálování barev, pokud jsou mimo range (0 ; 255), `preskalujBarvu()`

Verze 0.2.0 – 08. 03. 2021

Text

Dnes jsem se rozhodl trošku upravit můj prase-kód. Hlavně chci zobecnit parser, aby fungoval i na čínštinu a jakékoliv jiné znaky. Dále bych chtěl trošku rozsekat kód do více tříd a jít na to trošku objektověji. Čeká mě plný den refaktorizace a přemýšlení, paráda.

První změna je nová třída `NacitaniDat`, jedná se o návrhový typ jedniáček a obsahuje předchozí metody `nactiData()` a `rozkrajejRadky()`. Dále jsem pro načítání dat chtěl změnit ty `Stringy`, přijde mi lepší pracovat s polem čísel. Jelikož znám i velikost pro potřebná data (`vyska` * `sirka`), převedl jsem list `Stringů` na `intové` pole. Tuto funkčnost zajišťuje nová metoda `prevedNaCisla()`. Dále je tu také metoda pro načtení největší a nejmenší hodnoty v poli, `nactiMinMax()`. Tato metoda přiřadí nejmenší a největší hodnotu vyskytující se v poli do proměnných `minimum` a `maximum`. Zapomněl jsem zmínit změnu s načítáním rozměrů obrázku. Vlastně jsem to udělal tak, že se projede každý řádek od prvního toho ještě `Listu Stringů` a když se narazí na řádek, co obsahuje právě 2 položky, tak se uloží do proměnných `vyska` a `sirka`. Doufám, že to je dosti obecné a funkční pro hodně případů, to se uvidí.

Další změna je přidání další třídy! Opět se jedná o jedináčka a je to třídy, která se stará o okna, `SpravceOkna`. Ta si jen bere od hlavní třídy metody `zalozOkno()` a `oknoError()`.

Těmito změnami jsem se zbavil všech metod z hlavní třídy, kromě mainu, v kterém si ukládám referenci na `SpravceOkna` – `okenik` a na `NacitaniDat` – `nacitac`. Dost už ale OOP, pojďme zpět ke grafice.

V `DrawingPanelu` jsem strávil pěknou hodinku úprav, než se dostala do této finální podoby, kde je jen metoda `paint()` a metoda `preskalujBarvu()` (změněná). Nejprvě jsem změnil vykreslování, aby si neparsovali `Stringy` na inty samo, když už předávám data jako pole `intů`. Pak mě už konečně napadla skvělá funkce pro škálování barev do intervalu 0 až 255. Vlastně se může stát, že dostanu hodnoty od jakéhokoliv minima, do jakéhokoliv maxima. A já tento interval <min ; max> chci převést na <0 ; 255> a zachovávat poměrové rozdíly barevných odstínů. Vymyslel jsem tak rovnici:

$$novaBarva = \frac{(barva - minimum)}{(maximum - minimum)} \cdot 255$$

Takhle vlastně, když za barvu dosadím minimum z daného pole, dostanu 0. Když za barvu dosadím maximum, vykrátí se mi zlomek a dostanu 255. Když dosadím cokoliv mezi minimem a maximem dostanu správnou hodnotu v novém intervalu.

Škálování jsem stále nebyl schopen upravit na bilineární interpolaci a je zachováno škálování staré.

Po těchto všech úpravách jsem zkusil otevřít ono „čínské“ upg.pgm. A co se nestalo? Otevřelo se. Nyní jsem schopen otevřít jakékoliv (snad jakékoliv) .pgm a přeškálovávat obrázek podle velikosti okna. Začínám být spokojený se svou prací a už se ani nestydím (tolik) za svůj kód.

Changelog

Verze 0.2.0

Hodin strávených na této verzi: 3

upravena třída `Mapa_SP2021`

- odebrány (přemístěny) všechny metody, kromě `main()`

přidána třída `NacitaniDat`

- přesunuty metody z `Mapa_SP2021`, `nactiData()` a `rozkrajejRadky()`
- upraveny (lehce) výše zmíněné metody pro správnou funkčnost
- přidána metoda na převod listu Stringů na pole intů, `prevedNaCisla()`
- přidána metoda pro načtení nejmenšího a největšího čísla v poli, `nactiMinMax()`

přidána třída `SpravceOkna`

- přesunuty metody z `Mapa_SP2021`, `zalozOkno()` a `oknoError()`

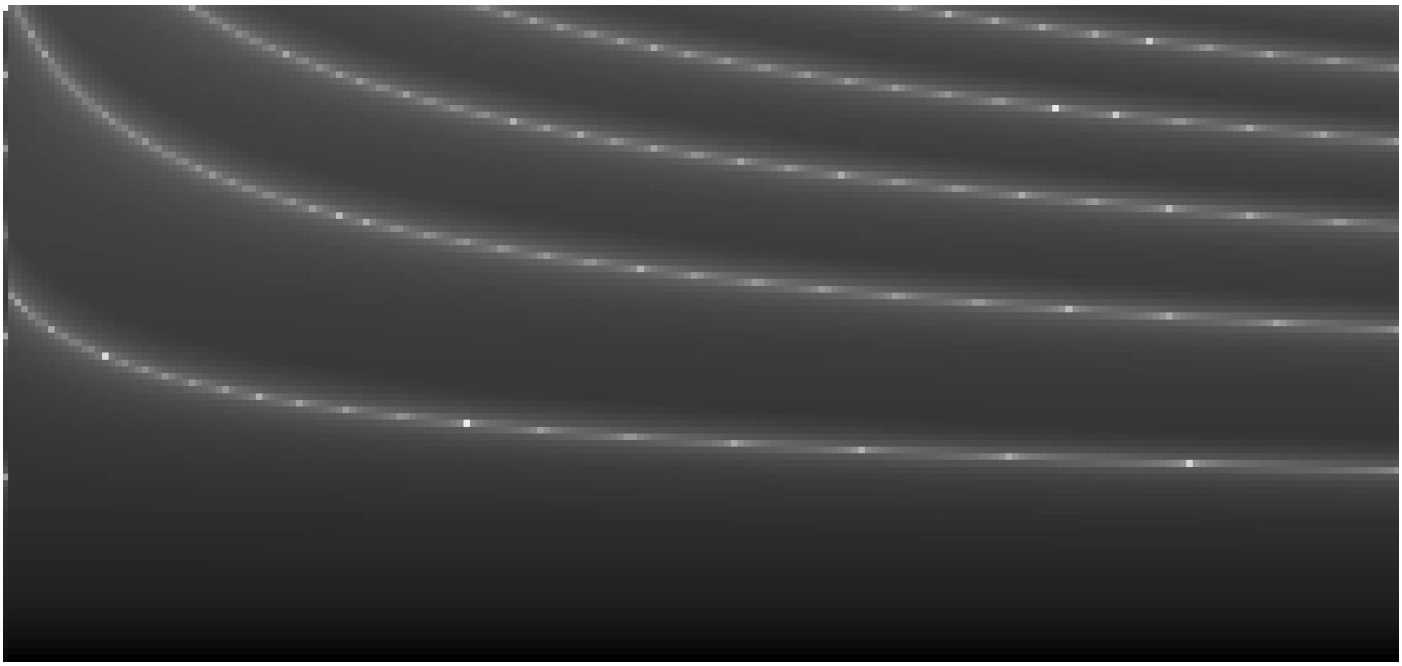
upravena třída `DrawingPanel`

- upravena metoda `paint()` pro správné vykreslování s polem `int`ů, už ne s listem `String`ů
- upravena metoda `preskalujBarvu()` pro správné a přesné přeškálovávání jakýchkoliv intervalů

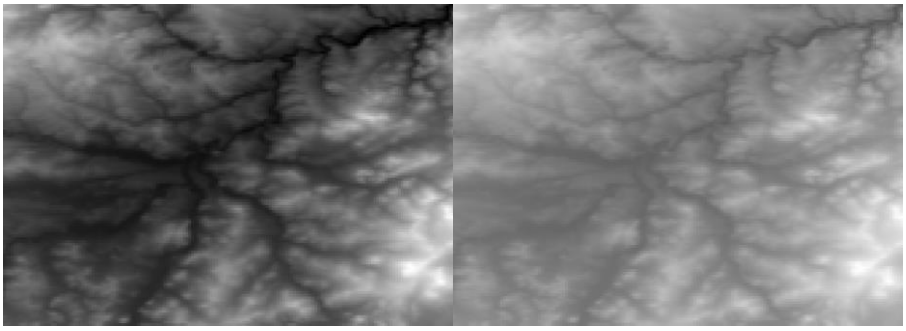
Verze 0.2.1 – 09. 03. 2021

Text

Dnes jsem si na internetu zkoušel najít více .pgm obrázků pro zajímavost a objevil jsem v tomto formátu jistý pattern. Vždy po řádku, který udává rozlišení je řádek udávající maximální hodnotu odstínu v obrázku. Došlo mi tedy, že moje metoda `nactiMinMax()` v `NacitaniDat` je zbytečná, protože mohu škálovat jen relativně vůči maximu, a nemusím `minimum` moc brát v potaz. Také mi došlo, že tohle je ta příčina, proč jsem v každém obrázku měl podivný první sloupeček, protože jsem celý obrázek měl posunutý o 1 pixel, tudíž první sloupeček pixelů měl být až ten poslední.



Dále jsem tedy změnil i metodu `preskalujBarvu()`, aby neškálovala interval < `min` ; `max` > na < 0 ; 255 >, ale aby škálovala vše jen relativně vůči `maximu` a `minimum` nebrala v potaz. Dosáhl jsem tak světlejší Plzně.



Vlastně jsem z původního vzorečku pro přeškálovanou barvu jen odebral `minimum`.

$$novaBarva = (barva/maximum) \cdot 255$$

Hlavně jsem touto změnou dosáhl o sortění pole méně. Ještě si nejsem jistý, která varianta se mi líbí více, jestli ta, která škáluje minimum na nulu a nebo ta, co ho škáluje relativně vůči maximu. Toto rozhodnutí nechám na budoucím Dominikovi.

Changelog

Verze 0.2.1

Hodin strávených na této verzi: asi 5 minut, takže to nebudu vůbec počítat

upravena třída `NacitaniDat`

- odebrána metoda `nactiMinMax()`
- lehká úprava metody `nactiData()` pro načtení `maxima`

upravena třída `DrawingPanel`

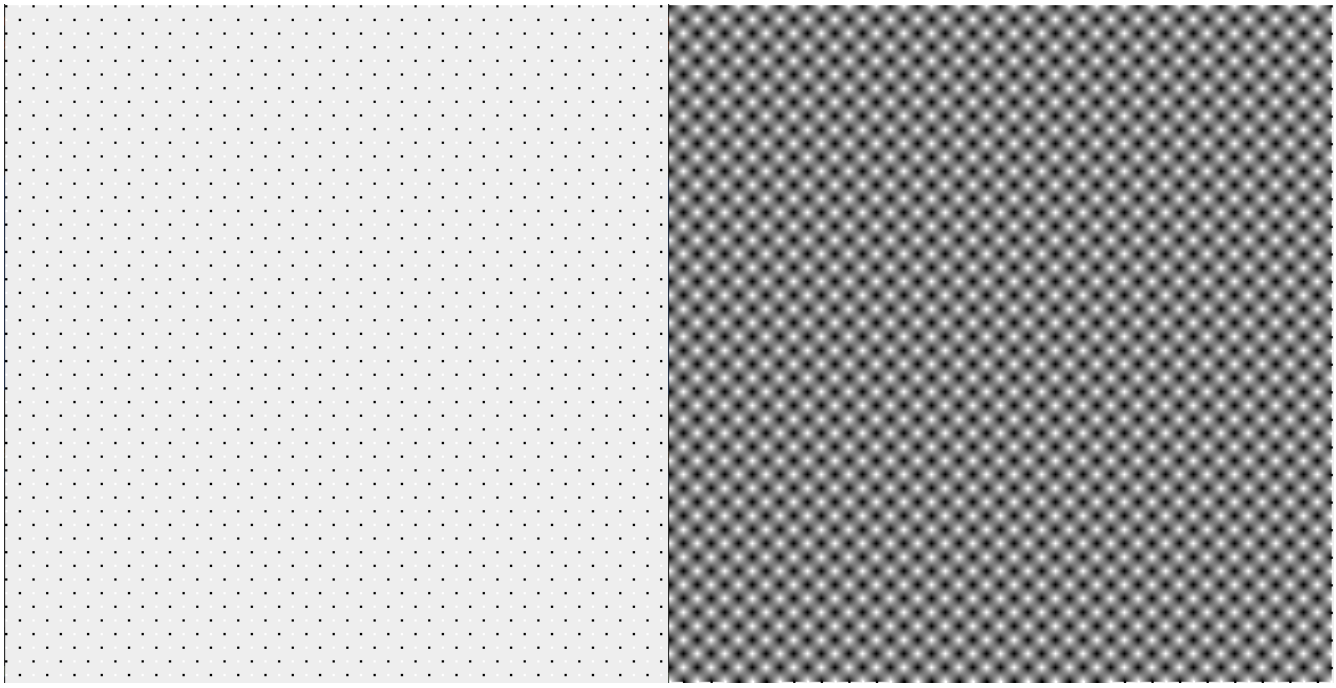
- upravena metoda `preskalujBarvu()`, změna vzorečku

Verze 0.3.0 – 21. 03. 2021

Text

Po dlouhých dvou týdnech přemýšlení a zkoušení jsem konečně implementoval bilineární interpolaci. To beru jako asi největší úspěch zatím, a proto jsem této změně věnoval i vlastní verzi. Pozměnil jsem obecně známý vzoreček tak, že rozdíl mezi dvěma body vždy beru, že je 1 a krok mezi nimi dělám 1 / `scale`.

$$hodnota = (y2 - y) \cdot \big((x2 - x) \cdot bod1 + (x - x1) \cdot bod2\big) + (y - y1) \cdot ((x2 - x) \cdot bod3 + (x - x1) \cdot bod4)$$



Z nějakého důvodu mi trvalo 2 týdny vymyslet, jak to implementovat na moje data. Hlavní problém asi byl s představou 1D pole jako 2D mapy. Na výše uvedeném obrázku je hezky vidět, jak z pár černých a bílých bodů je to nyní schopné zaplnit celý panel pro obrázek.

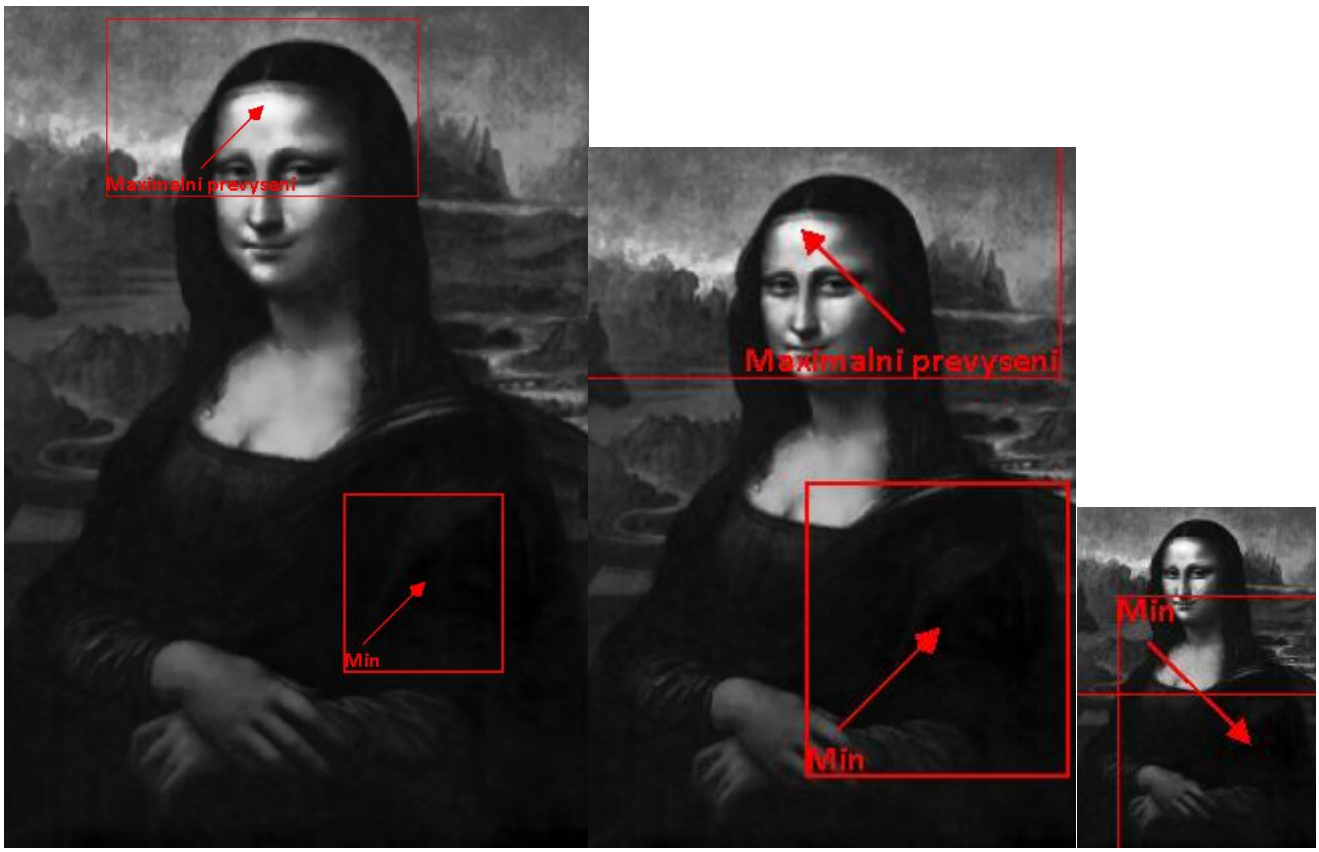
Dále jsem lehce pozměnil náčítání dat, aby ke přeškálování na < `min` ; 255 > došlo hned při načtení, ne až pak při kreslení.

Do vykreslování jsem vrátil zpět `nactiMinMax()`. Je to hlavně z důvodu šipek, ty mají ukazovat na minimální a maximální hodnotu.

Implementace šipek byla jednoduchá, použil jsem své řešení šipky z bonusové úložky 3. Přidal jsem k ní jen nápis, který se vykreslí podle směru šipky buď nad nebo pod ní. Aby šipky nevyšly mimo vykreslenou mapu, vymyslel jsem si hitboxy a obešel jsem vykreslování šipek různými směry, jak jsem původně chtěl (po kružnici), tak, že se vykreslují pouze do 4 směrů (jako střelec na šachovnici). Default směr je doprava nahoru, když ale zleva vyjde mimo vykreslenou mapu, šipka se překlopí doprava a bude ukazovat doleva nahoru. To samé když hitbox vyjde z mapy zespoda, tak se šipka překlopí nahoru a bude ukazovat doprava dolu. Finálně když dvě protilehlé strany hitboxu vyjdou z mapy, tak se šipka nemá kam překlopit, tak se prostě nevykreslí, aby nebyla mimo mapu. Velikost hitboxu samozřejmě závisí na délce řetězce, který je spojen s šipkou.

Zatím jsem neimplementoval šipku ukazující na maximální stoupání, ale jen pro minimální a maximální převýšení.

Nastal problém, když v obrázku bylo více minim a maxim, obrázek pak byl přehlcen šipkami, tak jsem implementoval booleanovské proměnné, které prostě kontrolují, aby tam šipka byla pro každou situaci jen jedna.



Dále jsem ještě upravil parser, aby načítal ještě obecněji, můj původní parser bral jako rozlišení první řádek, co měl řetězec o dvou „slovech“. A parsoval jen podle mezery, ne podle jakéhokoliv bílého znaku. Jak je na výše uvedeném obrázku vidět, našel jsem si na internetu více .pgm souborů a zkoušel, jestli je můj program přelouská. Nyní už tomu tak je, mona_lisa.pgm jsem vybral záměrně, protože má komentář dlouhý 2 slova a každý řádek byl odsazen mezerou a mezi hodnotami bylo více než 1 mezera. Takže tam nastalo hned více problémů: do resolution se mi snažil nacpat String, a to samé do hodnot dat.

Upravil jsem tak parser ze `.split(“ ”)` na `.trim().split(“\\s+”)`. Díky této změně a ještě změně podmínky pro načítání rozlišení jsem nyní schopný už snad načíst jakýkoliv .pgm!

Changelog

Verze 0.3.0

Hodin strávených na této verzi: asi věčnost, ale napíši si 10 hodin (myšlenkově to byla věčnost)

upravena třída `NacitaniDat`

- přesunuta metoda `preskalujBarvu()` z `DrawingPanel`
- upravena metoda `nactiData()`, úprava podmínky pro nacteni rozliseni
- upravena metoda `prevedNaCisla()`, úprava „krájení“ řádků

upravena třída `DrawingPanel`

- přidána metoda na vykreslení šipky `drawArrow()`
- přidána metoda na načtení `minima` a `maxima`, `nactiMinMax()`
- upravena metoda `paint()`, přidání bilineární interpolace, přidání šipek
- přesunuta metoda `preskalujBarvu()` do `NacitaniDat`

Verze 0.3.1 – 21. 03. 2021

Text

Finální úpravy parseru a finální úpravy šipek + přidání maximálního stoupání, tak bych shrnul tuto verzi.

Při prohlížení si všech možných .pgm souborů, co jsem nastahoval všude možně po internetu, jsem narazil na pár, pro které můj parser nefungoval. Nepočítal jsem totiž, že data mohou být čistě v jednom řádku, a tak jsem udělal finální úpravy parseru, který projíždí slovo od slova od začátku a hledá čísla. První tři čísla si načte jako rozlišení a maximum a se zbytkem už je to stejné jako dřív.

Za to v `DrawingPanelu` se toho dost změnilo. Původně jsem si hledal hodnoty `maxima` a `minima` pomocí metody `nactiMinMax()`, to už je opět minulostí. Hlavní problém byl, že já nejdřív načetl tyto hodnoty, ale poté jsem po vykreslení obrázku musel znovu procházet celá data obrázku a hledat shodu dané hodnoty pixelu s mými načtenými hodnotami `minima` a `maxima`... To mi přišlo poněkud zbytečné, a tak jsem to vymyslel trošku jinak. Místo hodnot `maxima` a `minima` si načtu jen jejich `x` a `y` index a pak nepotřebuji žádné smyčky a ify pro vykreslení šipek a prostě jim jen tyto indexy předám. Mimo jiné jsem se touhle změnou i zbavil těch pomocných booleanovských proměnných z minula, kdy jsem kontroloval aby byla šipka právě jedna.

První nápad, jak hledat maximální stoupání bylo, že nebudu projíždět první a poslední řádky a sloupečky a budu vždy okolo pixelu do takového znaku plus dělat rozdíly a hledat ten největší. To bylo fajn, ale k čemu porovnávat daný pixel s tím nad a vlevo, když už se s ním ty pixely předtím porovnaly? Tak tedy můžu porovnávat až na poslední sloupec a řádek vždy daný pixel s pixelem jen vpravo od něj a pod ním, vybrat největší rozdíl, a to je hodnota největšího stoupání. Opět si jen zapíšu jejich indexy `x` a `y` do proměnných a ty pak předám pro kreslení šipky.

K šipkám jsem ještě přidal pár vychytávek, protože se vyskytly případy, kdy mi dlouhý řetězec vedle šipky třeba vyjel mimo okno. To jsem vyřešil jednoduchým prodlužováním šipky – tudíž i odsouváním řetězce od kraje mapy. Sranda s hitboxy a překlopováním samozřejmě zůstaly.

Dolazování těchto drobností zabralo překvapivě poměrně dost času, ale myslím, že brzy budu moct tento projekt přesunout na verzi 1.0.0! (Což má být verze připravená k odevzdání).

Changelog

Verze 0.3.1

Hodin strávených na této verzi: 5 hodin

upravena třída `NacitaniDat`

- upravena metoda `nactiData()`, upraven a více zobecněn parser pro obecnější načítání dat

upravena třída `DrawingPanel`

- upravena a přejmenována metoda `nactiMinMax()`, nyní se tato metoda jmenuje `nactiMinMaxStoupani()` a nenačítá už hodnoty, ale indexy daných proměnných – `minima`, `maxima` a `stoupani`
- upravena metoda `drawArrow()`, přidáno pár podmínek a dalších ošetření pro vykreslování mimo mapu
- upravena metoda `paint()`, zjednodušeno vykreslování šipek, už není zapotřebí mít na to 2 fory a 3 ify, stačí prostě šikovné 3 řádky

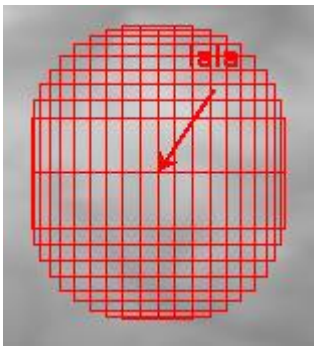
Verze 0.4.0 – 28. 03. 2021

Text

Bohužel a zároveň bohudík jsem na šestém cvičení zjistil, že existuje `BufferedImage` (bohužel protože mě mrzí ten čas a práce, co jsem věnoval vykreslování přes 2 smyčky po pixelu a hlavně ten čas strávený na vymýšlení bilineární interpolace).

Otestoval jsem rychlost svého vykreslování přes 2 for smyčky s vnořenými 2 dalšími smyčkami pro interpolaci (tj. celkem 4 for smyčky) oproti vykreslování přes `BufferedImage` a `BufferedImage` byl nepatrně rychlejší, předělal jsem tedy celé vykreslování do `BufferedImage`. Přidal jsem proto metodu `nactiObrazek()`, která načte originální obrázek z pole intů do proměnné `imgOrig`. Tento originální obrázek pak zpracovávám přímo v `paint()` přes `AffineTransform` a `AffineTransformOp`. Tuto implementaci bilineární interpolace jsem našel na internetu na adrese https://blog.idrsolutions.com/2019/01/image-scaling-options-in-java/. Tímto už zase funguje vykreslování a bilineární interpolace (a mnohem snadněji a lehce i rychleji).

Trápily mě šipky, nepřišly mi dostatečně správné s tím, jak se vykreslovali. Vlastně se vykreslovaly jen do 4 směrů (diagonálních). Ale co když by byl obrázek tenká nudlička? Šipka by se tam přeci stejně vejít mohla. Předělal jsem tak vykreslování šipek a každá šipka nyní ví na jaký bod ukazuje. Od tohoto bodu si pomyslně umí okolo sebe nakreslit kružnici s poloměrem délka šipky. Defaultní směr je stále doprava nahoru (diagonálně), ale když by se tam ta šipka takhle nevešla, tak se po té kružnici dokáže „klouzat“ do lepších pozic. Upravil jsem i hitboxy šipek, aby byly přizpůsobitelné snad každé situaci. X souřadnice se šikovně volí z té co je nejvíc vlevo ze 3 možných – `x1`, `x2` nebo půlka `Stringového` popisku co „přečuhuje“. Tak samo Y souřadnice, a tak samo šířka a výška hitboxu. Ve finále to vypadá nějak takto:



Každá šipka si takhle umí okolo sebe představit až 360 možných hitboxů a vybírá si z nich ten nejšikovnější. Reálně se samozřejmě nepočítá všech 360, ale hned jak se najde první vyhovující, šipka se nakreslí a další jí nezajímají. K výpočtu dalších možných dojde až v moment, kdy je to potřeba, tj. když by šipka nebo text měly být mimo okno. Když by náhodou žádný hitbox nevyhovoval, a šipka by se tedy měla vykreslit mimo mapu, nevykreslí se šipka vůbec.

Changelog

Verze 0.4.0

Hodin strávených na této verzi: 5 hodin

upravena třída `DrawingPanel`

- přidána metoda `nactiObrazek()`, metoda načte pole intů jako `BufferedImage` a uloží si ho do proměnné
- upravena metoda `nactiMinMaxStoupani()` tak, aby pracovala s `BufferedImage`
- upravena metoda `paint()` tak, aby vykreslovala `BufferedImage` a jeho interpolaci
- upravena metoda `drawArrow()`, změna vykreslování šipek, už nemají jen diagonální směr, ale všech 360° kolem dokola a vybírají si ten nejvhodnější směr

Verze 0.4.1 – 28. 03. 2021

Text

Delší dobu jsem uvažoval o vlastním vylepšení, protože mě nebavilo pořád přepisovat jaký soubor načítám, a tak jsem se rozhodl přidat třídu `ButtonPanel`, která se stará o panýlek v dolní části okna, který má tlačítka na přepínání obrázků. Naplnil jsem tento panel `JButtony`, které jsem si nějak přizpůsobil, aby se mi líbily vzhledově. Pak jsem jen přidal `ActionListener` a ten volá metodu `main()` ze třídy `Mapa_SP2021` s daným parametrem podle názvu tlačítka. Ještě předtím samozřejmě zavře původní okno, než otevře nové s novým obrázkem.
Toto vylepšení mi přijde jako fajn způsob, jak přepínat rychle a elegantně obrázky. Samozřejmě stále funguje klasické přepínání zavřením okna a otevřením nového přes jiný parametr z příkazové řádky.

Changelog

Verze 0.4.1

Hodin strávených na této verzi: 1 hodina

přidána třída `ButtonPanel`

- přidána metoda `nastavTlacitka()`, která nastavuje funkčnost a vzhled tlačítek, které přepínají obrázky

Verze 1.0.0 – 04. 04. 2021

Text

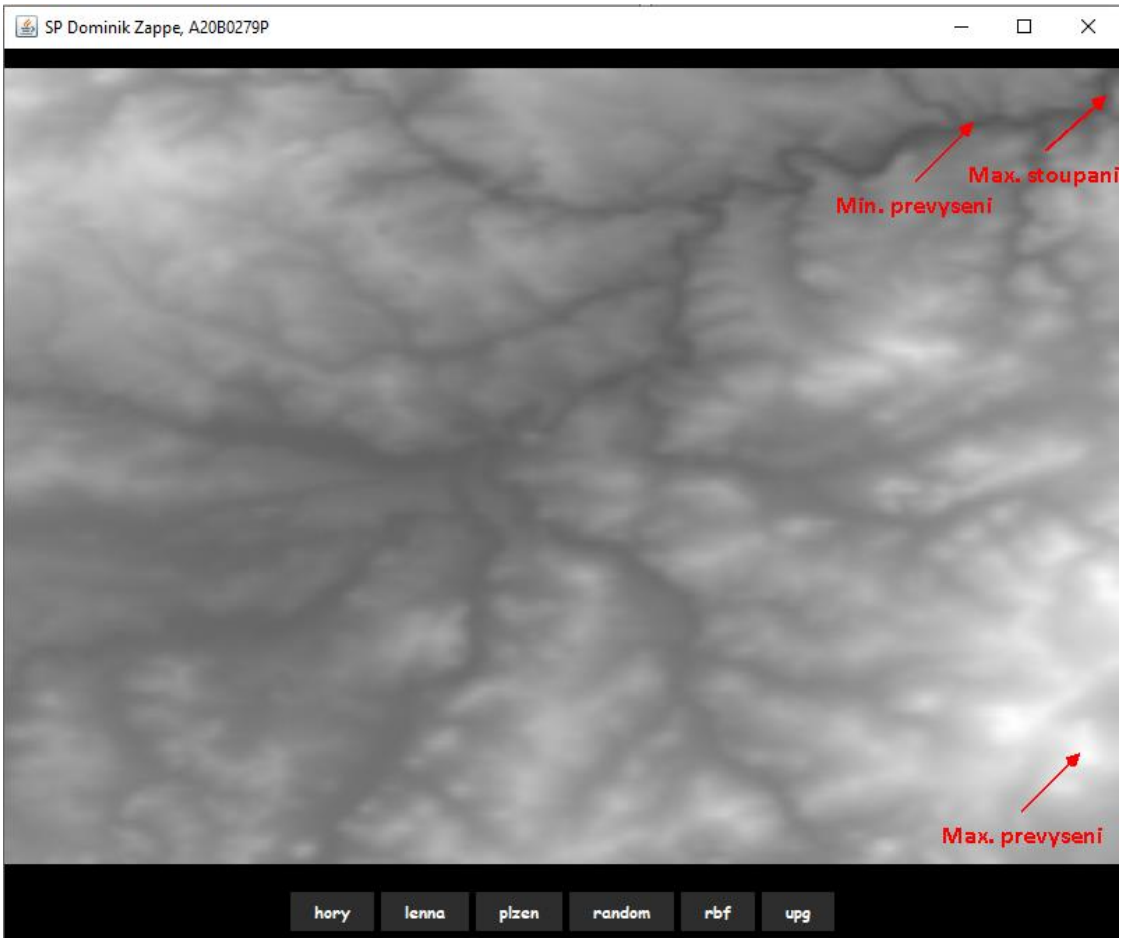
Finální úpravy a řešení malých bugů.

Přidal jsem do třídy `NacitaniDat` globální proměnnou `dataOriginal`, což je pole intů, které je plné originálních, nepřeškálovných, hodnot barev ze souboru.

Přidal jsem malou podmínku k šipkám, že když se nachází v úzkém horním, nebo dolním rohu, tak se nevykreslí vodorovně, páč pak jim lehounce vykukoval kousek hrotu mimo okno. Vykreslí se tedy pod mírným úhlem.

Upravil jsem dále vzhled tlačítek, aby vypadali lépe.

Tato verze je finální verzí pro první odevzdávání. Finální produkt vypadá takto:



Changelog

Verze 1.0.0

Hodin strávených na této verzi: 1

upravena třída `NacitaniDat`

- upravena metoda `prevedNaCisla()`, tato metoda nyní ukládá i nepřeškálované hodnoty dat ze souboru

upravena třída `DrawingPanel`

- dolazení bugů ohledně šipek

upravena třída `ButtonPanel`

- upraven vzhled tlačítek

Verze 1.1.0 – 08. 04. 2021

Text

Rozhodl jsem se udělat refaktorizaci kódu a roztrhat poměrně dlouhou metodu `drawArrow()` v `DrawingPanelu` na menší metody. Přidal jsem tak metodu `vyberSpravnySmerSipky()`, která se stará o hitboxy a správný směr šipky. Další přidaná metoda prostě vykresluje už danou šipku – `vykresliSipku()`.

Další refaktorizaci jsem udělal přidání třídy `Pozice`. `Pozice` je návrhový vzor přepravka a prostě jsem jen chtěl „scucnout“ k sobě neustálé opakování X, Y souřadnic. Neřešil jsem žádné zapouzdření, takže proměnné `x` a `y` jsou public a nepotřebuji k nim gettery, prostě k nim jde přes tečkovou konvenci. Refaktorizoval jsem tak celý kód a už tam není 8 globálních proměnných typu `double startX`, `double startY`, `double maximumX` atd. ale je jich jen 4 – `Pozice startXY`, `Pozice maximumXY`, `Pozice minimumXY` a `Pozice stoupaniXY`. Přijde mi to tak elegantnější.

Changelog

Verze 1.1.0

Hodin strávených na této verzi: 2

přidána třída `Pozice`

- přidán konstruktor, který nastavuje souřadnice X a Y na dané hodnoty

upravena třída `DrawingPanel`

- obecná refaktorizace souřadnic tak, aby byla využita třída `Pozice`
- roz-metodována metoda `drawArrow()` na další 2 pod-metody
- přidána metoda `vyberSpravnySmerSipky()`, která se stará o směr šipky, popř. o nekreslení šipky, nevejde-li se do mapy
- přidána metoda `vykresliSipku()`, která zařizuje samotné vykreslení šipky

Verze 1.2.0 – 18. 04. 2021

Text

Sice bych měl pracovat na vrstevnicích, ale export mi přišel atraktivnější, a tak jsem udělal volitelná rozšíření „Tisk“ a „Export do PNG“.

Do `ButtonPanelu` jsem přidal tlačítko `Export…`, které vytvoří instanci `JOptionPane` a dá na výběr `Tisk`, `PNG` nebo `Cancel` (do budoucna možná ještě přidám `SVG`).

Pokud se zvolí `Tisk`, metoda vytvoří `PrinterJob` a nastaví `DrawingPanel` na printable a zavolá nad ním `print()`. `Print()` v `DrawingPanelu` zařídí, aby bylo pozadí obrázku bíle a aby se obrázek naškáloval na celou A4, s malými okraji (padesátina `scalu` se odebere od `scale`). Rovněž se nastaví `startXY` na správnou pozici a vyhodí se dialogové okno, kde se dá přepnout na virtuální tiskárnu do PDF. To se pak zeptá na umístění a název od uživatele.

Pokud se zvolí `PNG`, metoda vytvoří další dvě dialogová okna, ty se požadují zadání požadované `výšky` a `šířky`. Dále se zajistí, aby se obrázek správně naškáloval na požadovanou výšku a šířku v `DrawingPanelu` pomocí volání metody `exportDoPNG()`. Výsledný PNG obrázek se uloží do stejného adresáře, v kterém je zdrojový obrázek ve formátu .pgm a toto .png bude mít stejný název jako obrázek původní.

Kvůli těmto změnám nastalo pár úprav v `DrawingPanelu`, jedná se hlavně o pár podmínek, které kontrolují booleanovské proměnné, které indikují, zdali se jedná o export, nebo ne (pro správný `scale` a `startXY` a pozadí).

Changelog

Verze 1.2.0

Hodin strávených na této verzi: 2

upravena třída `DrawingPanel`

- přidána metoda `exportDoPNG()`, ta se stará o export do formátu .png (volitelné rozšíření)
- přidána metoda `print()`, ta se stará o chod tisku (volitelné rozšíření)
- změny související s exportem – booleanovské proměnné a podmínky související s nimi

upravena třída `ButtonPanel`

- přidáno tlačítko `Export…`, které vyhazuje dialogové okno a dává možnost Tisku nebo Exportu do PNG

Verze 1.2.1 – 20. 04. 2021

Text

Hrál jsem si s dialogy a tlačítky a předělal jejich vzhled a `grid` v jakém jsou rozpoložení. Teď na `ButtonPanelu` jsou 2 další panely – horní a dolní lišta. Spodní lišta slouží jen pro přepínání mezi obrázky, horní lišta má důležitější tlačítka. Na horní lištu jsem přidal tlačítka pro skrytí šipek a vrstevnic.

Nový export přidán – ASCII art, takže další volitelné rozšíření – „ASCII Art“. Přidal jsem možnost do tlačítka `Export…` si vybrat `ASCII`. Tato volba zavolá v `DrawingPanelu` metodu `exportDoASCII()`, tato metoda projede pole dat a k hodnotám přiřazuje znaky. Rozhodl jsem se odlišovat hodnoty pixelů, když se liší o 10 a začínám znakem dvou mezer (vždy jsou použity znaky dva, protože znaky jsou obecně 2krát vyšší než širší, takže pro zachování poměru obrázku jsou vždy znaky dva) a končím @@. Takže hodnotám stupnice od 255 do 0 vypadá následovně:

```
..,,:::;;^^++**||(([[{{TTIIEEAAHH##RRBB88WW&&@@@@
```

Dále bylo potřeba ošetřit, když je obrázek moc velký a řádky jsou moc dlouhé, a tak vynechávám nějaké pixelu v případě, že je obrázek určité velikosti. Pro obrázky 500x500 a menší se projede každý pixel, když je šířka nebo výška obrázku mezi 500 a 1000 projede se každý druhý pixel. Obdobně pro 1000 až 2000 se projíždí každý čtvrtý pixel. A na závěr pro 4k pgm obrázky by se projížděl jen každý osmí pixel.

Exportovaný ASCII Art se uloží stejně jako png do stejného adresáře zdrojového .pgm souboru.

Dále byla přidána možnost klikat kdekoliv na obrázku a tím si zobrazit výšku místa, kam se kliklo. Byla by škoda nevyužít už funkční šipky, takže jsem jen přidal šipku na dané místo, kam se klikne, která jako popisek nese údaj o převýšení. Pro celou tuto funkčnost byl použit `MouseListener`.

Na závěr jsem přidělal pár dialogových oken, která hlásí chyby nebo zdaření různých akcí.

Changelog

Verze 1.2.1

Hodin strávených na této verzi: 2

upravena třída `DrawingPanel`

- přidána metoda `exportDoASCII()`, která se stará o export do .txt ASCII Artu. Přiřazuje hodnotám šedotónu různé znaky a stará se o správnou velikost výstupních řádků v souboru
- přidán `MouseListener`, který hlídá kam uživatel kliká a když se klikne do obrázku vykreslí šipku na daný bod, která jako popisek nese údaj o převýšení
- přidány booleanovské proměnné a podmínky k nim, které souvisí s novými tlačítky pro skrývání šipek a vrstevnic

upravena třída `ButtonPanel`

- přidána možnost exportu do ASCII Artu pod tlačítko `Export…`
- přidána dialogová okna, hlásící chyby, popř. zdary

Verze 1.3.0 – 21. 04. 2021

Text

Proběhlo mnoho úprav a velkých změn od poslední verze. Nyní je prakticky vše ze zadání splněno až na grafy (ty se mají brát až v pátek na cvičení). Přibyla třída `LegendaPanel` a v `DrawingPanelu` přibylo mnoho metod – `nakresliVrstevnici()`, `zvyrazniVrstevnici()`, `vytvorMozneHodnotyVrstevnic()`, `vytvorBarevnouPaletu()` a `nastavBarvyBarevnychPloch()`.

Začněme pěkně popořadě, vykreslení vrstevnice jako čáry. Šel jsem na to dle rady pana Doc. Kohouta a to tak, že projedu celý obrázek a porovnávám ho s určitou hodnotou. Když je menší, nastavím si v poli booleanů hodnotu tohoto pixelu na `false`, větší na `true`. Potom každý pixel porovnávám s jeho sousedem vpravo a pod ním a když se jejich logické hodnoty liší, nakreslí se mezi ně pixelík. Toto jsem chtěl nacpat do `BufferedImage` a udělat jen pro originální obrázek a vrstevnice přeškálovat pomocí interpolace, ale vypadalo to ne-hezky a tak jsem zvolil cestu, že projíždím už přeškálovaný obrázek pixel po pixelu (je to pomalejší, ale hezčí) a kreslím obdélníčky přímo.

Metoda `zvyrazniVrstevnici()` vlastně volá dříve popsanou `nakresliVrstevnici()` v momentě, kdy je to potřeba s odlišnou barvou. Tato funkčnost byla přidána k `MouseListeneru` v `DrawingPanelu`. Ještě byla přidána malá vychytávka – pravým tlačítkem jde zrušit označení vrstevnice.

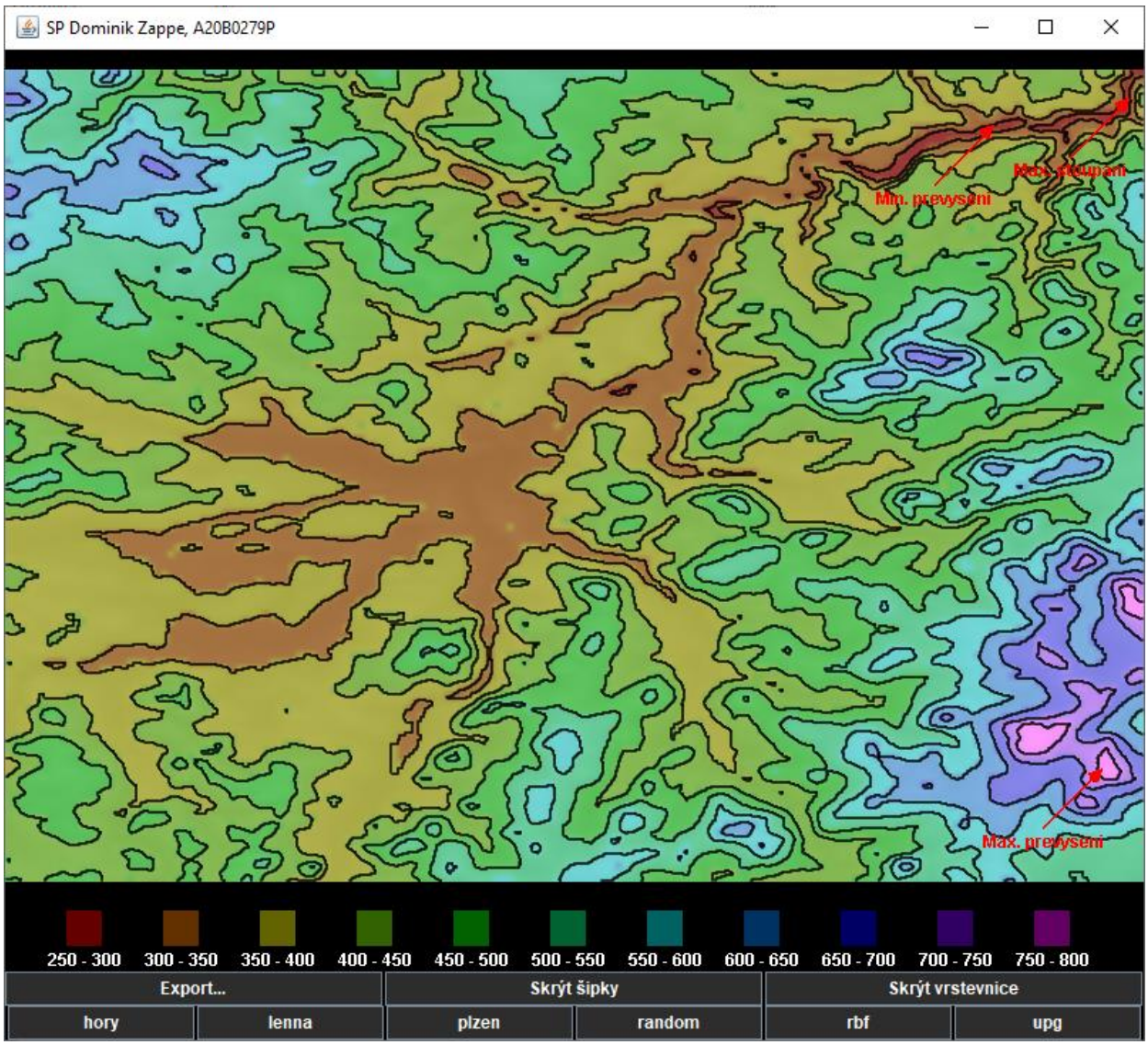
Metoda `vytvorMozneHodnotyVrstevnic()` jen naplní pole integerů hodnotami od 0 do 8850 (Mt. Everest) s krokem 50.

`vytvorBarevnouPaletu()` je velmi šikovná metoda, která na míru vytvoří danému obrázku barevnou škálu vrstevnicových ploch od červené do fialové. Vlastně když vím, že vrstevnic bude 19, budu potřebovat 20 barev. Těchto 20 barev se dá míchat mým způsobem (duhový gradient) přes 5 stěžejních stavů. Výchozí stav je vždy červená – 255, 0, 0 (RGB). První stav přidává zelenou, takže se z červené stává žlutá – 255, 255, 0. Druhý stav ubírá červenou, takže se ze žlutou stává zelená – 0, 255, 0. Třetí stav přidává modrou a tak se ze zelené stává tyrkysová – 0, 255, 255. Čtvrtý stav ubírá zelenou, takže se z tyrkysové stane modrá – 0, 0, 255. A poslední pátý stav přidá červenou, takže se z modré stane magenta – 255, 0, 255. Zpět k příkladu, kdy potřebuji 20 barev, vím tedy, že je 5 stavů základních. 20 děleno 5 mi dává 4. Takže krok jakým budu měnit jednotlivé stavy je 255 / 4. Obdobně to bude fungovat pro obrázek převýšení Mt. Everestu, gradient by byl jen jemnější a plynulejší, ale přechody by byly také krásně vidět. Myslím že tato metoda chytrého určování barev je povedená, každý obrázek má na míru barvičky duhy.

Finálně `nastavBarvyBarevnychPloch()` si založí `BufferedImage` a projedou se původní data obrázku. Šikovnou funkcí si vždy zaokrouhlím hodnoty na jejich nejbližší násobek 50 a pak jen přiřadím převýšením odpovídající barvu z vytvořené palety. Tento obrázek pak nechám vykreslovat přes původní obrázek a také se škáluje zapomocí interpolace.

Dále jsem jen využil už dané palety a vytvořil jsem třídu `LegendaPanel`, která se stará o přehlednou legendu pod obrázkem. Legenda opět zná počet barev a šikovně si rozvrhne prostředí tak, aby mezi jednotlivými barevnými vzorky byly stejné mezery. Pod barevným vzorkem je vždy ještě popisek hodnot převýšení, které reprezentuje.

Dále byly přidány tlačítka pro skrývání šipek a vrstevnic jako vlastní vylepšení vizualizace.



Changelog

Verze 1.3.0

Hodin strávených na této verzi: 10

upravena třída `DrawingPanel`

- přidána metoda `nakresliVrstevnici()`, která se stará o vykreslení jedné vrstevnice
- přidána metoda `zvyrazniVrstevnici()`, která zvýrazňuje nakliknutou vrstevnici
- přidána metoda `vytvorMozneHodnotyVrstevnic()`, která jen naplňuje pole čísel možnými hodnotami
- přidána metoda `vytvorBarevnouPaletu()`, která se šikovně stará o dobré rozpoložení barev vrstevnicových ploch
- přidána metoda `nastavBarvyBarevnychPloch()`, která obarvuje plochy mezi vrstevnicemi barvami z palety
- přidány booleanovské proměnné, které se starají o funkčnost skrývačích tlačítek

upravena třída `ButtonPanel`

- přidána tlačítka pro skrývání šipek a vrstevnic

přidána třída `LegendaPanel`

- třída reprezentuje panel legendy pod obrázkem

Verze 1.3.1 – 25. 04. 2021

Text

Bylo přidáno volitelné rozšíření „Zvětšení a posun“.

Je umožněn intuitivní pohyb ve vizualizaci. Kolečkem myši lze přiblížit / oddálit obrázek (oddálení je omezené na původní velikost, přiblížení není omezené). Tato funkčnost je zajištěna za pomoci `MouseWheelListenera` a jeho metody `getPreciseWheelRotation()`. Zbytek kódu byl uzpůsoben, aby správně s tímto rozšířením fungoval (hlavně šipky). Pohyb jako takový posouvání přiblíženého obrázku je umožněno za pomoci `MouseListenera` a jeho metod `mousePressed()` a `mouseReleased()` (+ gettery na X a Y souřadnice). Program si spočítá vektor posunu a o ten v obrázku posune. Posouvání je omezené, aby nešlo vyjít za hranice obrázku, tj. levý horní roh a pravý dolní roh. Zbytek kódu byl taktéž uzpůsoben tak, aby vše správně s rozšířením fungovalo.

Po veškerém výpočtu je v metodě `paint()` přiblížení a posun jednoduše udělán přes `g2.translate()` a `g2.scale()` metody.

Značně byla předělána metoda `nakresliVrstevnici()`. Nyní už neprojíždí celý obrázek a nekreslí vrstevnice všude, ale kreslí jen na viditelnou část, což prakticky znamená, že se program nezpomaluje s rostoucím přiblížením.

Changelog

Verze 1.3.1

Hodin strávených na této verzi: 3

upravena třída `DrawingPanel`

- přidána funkčnost přiblížení a oddálení obrázku (kolečkem myši)
- přidána funkčnost posunu v obrázku (taháním myší)
- upravena metoda `nakresliVrstevnici()` tak, aby správně fungovala s novými rozšířeními
- obecné úpravy v kódu, aby vše fungovalo dle očekávání s volitelným rozšířením

Verze 2.0.0 – 28. 04. 2021

Text

Přidání třídy `SpravceGrafu` a hrátky s knihovnou `JFreeChart`. `SpravceGrafu` je jedináček a obsahuje 2 metody – `vytvorGrafPrevyseni()` a `vytvorHistogramPrevyseni()`.

Pro graf převýšení je žádoucí, aby bylo vidět na první pohled minimum, maximum, kvartily, medián a průměr. Zvolil jsem tak boxplot pro tento případ. Defaultní nahrání listu do datasetu, ale nefungovalo dle očekávání a vznikaly mi tam tzv. *farouts* a divné trojúhelníčky a kolečka. To se mi nelíbilo, a tak datasetu data „předpočítám“. Metoda si nejdříve sama zjistí průměr, medián, kavartily a další potřebné hodnoty a pak se do datasetu vkládá `BoxAndWhiskerItem`. Dále jsem změnil barvu pozadí na bílou a zapnul pro přehlednost `GridLines`. Box mi přišel příliš široký, a jelikož průměr (kolečko) má vždy 50% šířky boxu, chtěl jsem nějak tuto šířku zmenšit. Docílil jsem toho přes `domainAxis` a její metody `setLowerMargin()` a `setUpperMargin()`. Pak už jsem jen změnil barvu grafu jako takového.

Pro histogram převýšení jsem chtěl použít `ChartFactory.createHistogram()`, což fungovalo hezky, ale v histogramu byly sloupečky příliš úzké a když jsem nevykresloval každý metr převýšení, ale např. jen každých 50 metrů (stejně jako vrstevnice), tak to vypadalo divně. Zvolil jsem proto místo histogramu `BarChart`. Opět hodnoty předpřipravím a upravím je do stejné podoby, jako jsou data vizualizována po vrstevnicích. Takže např 25 a 47 se dají do stejné kategorie, a to 0-50. Opět byla nastavena barva pozadí, `GridLines`, barva grafu a další potřebné věci pro hezčí vizualizaci. Sloupečky také mají popisky, ukazující přesnou četnost převýšení.

Changelog

Verze 2.0.0

Hodin strávených na této verzi: 6

upravena třída `SpravceOkna`

- přidána okna pro grafy

přidána třída `SpravceGrafu`

- přidána metoda `vytvorGrafPrevyseni()`, která vytvoří `ChartPanel` s `BoxPlotem`
- přidána metoda `vytvorHistogramPrevyseni()`, která vytvoří `ChartPanel` s `BarChartem`

upravena třída `ButtonPanel`

- přidána tlačítka pro zobrazení oken s grafy

Verze 2.1.0 – 29. 04. 2021

Text

Že vrstevnice vykresluji neefektivně, to jsem si myslel celou dobu. Dnes jsem si to opravdu ověřil, protože načtení nových souborů (s enormním množstvím vrstevnic (1000+)) trvalo klidně přes minutu. Byla potřebná změna, a tak došlo k dost zásadním. `nakresliVrstevnici()` už neexistuje a stejně tak `vytvorMozneHodnotyVrstevnic()`. Nyní všechny vrstevnice vykresluje jedna metoda naráz – `nakresliVrstevnice()`. Dříve jsem porovnával každou hodnotu pixelu s požadovanou hodnotou z možných hodnot vrstevnic (násobky 50) a tvořil na základě porovnání true / false mapy a podle nich pak vykresloval. Nyní metoda přečte všechny pixely jen jednou a na jejich základě si do pole přiřadí ke každému pixelu nějakou symbolickou hodnotu. Pro 0-50 si zařadí 0, pro 50-100 si zařadí 1, atd. Tudíž mám všechny „true / false mapy" v jednom poli a stačí mi porovnat vždy danou hodnotu s jejím sousedem napravo a pod ní a podle toho kreslit.

Jelikož se změnilo kreslení vrstevnic, muselo se změnit i jejich zvýraznění, `zvyrazniVrstevnici()` má nyní pomocnou metodu – `hodnotaZvyrazneneVrstevnice()`, která prakticky dělá, to co dříve zvyrazniVrstevnici(). Namísto volání `nakresliVrstevnici()` má nyní `zvyrazniVrstevnici()` právě starou implementaci `nakresliVrstevnici()`. Zkrátka jsem starou funkčnost metody přesunul do nové a starou implementaci vykreslování přesunul do zvýrazňování.

Dále byla upravena tlačítka, přidána nová přepínatelná tlačítka pro nové .pgm soubory.

Díky novým .pgm souborům byla upravena i legenda, když se nevejde na svůj panel v normální formě, tak se přemění na gradient zleva doprava s popiskem od jaké do jaké hodnoty gradient je.

S novou implementací vrstevnic jsem dosáhl rekordních časů. Dříve bez čar se mi načítali všechny obrázky pod 1 sekundu, ale s čarami klidně i 2 sekundy. Random2 jsem napočítal až do 68 sekund… Nyní se vše opět načítá pod sekundu i s čárami. Tuto verzi považuji za velký úspěch.

Changelog

Verze 2.1.0

Hodin strávených na této verzi: 4

upravena třída `DrawingPanel`

- přidána metoda `hodnotaZvyrazneneVrstevnice()`, která přepočítá správnou hodnotu vrstevnice, co se má zvýrazňovat
- upravena metoda `nakresliVrstevnici()`, metoda je nyní nespočetně krát rychlejší, nezpomaluje celou aplikaci a kreslí všechny vrstevnice na jeden průchod
- upravena metoda `zvyrazniVrstevnici()`, metoda nyní implementuje starou implementaci metody `nakresliVrstevnici()`, aby si zvýraznila požadovanou vrstevnici s pomocí `hodnotaZvyrazneneVrstevnice()`
- odebrána metoda `vytvorMozneHodnotyVrstevnic()`

upravena třída `ButtonPanel`

- přidána tlačítka pro nové .pgm soubory

upravena třída `LegendaPanel`

- upravena legenda, když je vrstevnicových ploch příliš a legenda by se na panel nevešla (udělá se místo toho gradient s jednoduchým popiskem od jaké hodnoty do jaké gradient znázorňuje)

Verze 2.2.0 – 10. 05. 2021

Text

Díky ukázce na cvičení 11 jsem dodělal poslední volitelné rozšíření a poslední export. O tento export se stará knihovna `JFreeSVG`, přidal jsem možnost si zvolit SVG Export pod tlačítko Export…. Stisknutím této možnosti vyskočí dialog, kde si může uživatel zvolit šířku a výšku (obdoba jako u png) a opět bylo ošetřeno, aby uživatel nemohl zadat řetězec, zápornou nebo, nulovou hodnotu. Jediný nedostatek, co pozoruji je, že jak vykresluji vrstevnice způsobem, že procházím celý obrázek pixel po pixelu, tak vykreslení .svg trvá poměrně dlouho, než se vše vykreslí. Šlo by to řešit tím, že bych od okna kreslil tak jako do teď, a do svg bych posílal BufferedImage, ztratila by se tím však vektorová grafika vrstevnic. Po domluvě se cvičícím nechávám tedy pomalejší načítání se zachováním vektorové grafiky.

Changelog

Verze 2.2.0

Hodin strávených na této verzi: 2

upravena třída `DrawingPanel`

- přidána metoda `exportDoSVG()`, která se stará o správné vykreslení do a zapsání do SVG

upravena třída `ButtonPanel`

- přidána možnost výběru exportu do formátu SVG

Závěr

I když práce není dokonalá a například zoom mohl být řešen elegantněji (aby se přibližovalo přímo na umístění kurzoru), nebo vrstevnice mohly být rychlejší, kdybych přišel na to, jak to implementovat přes `Path`, jsem s prací spokojen.

Nevybral jsem si nejvíce „vizualizační“ volitelná rozšíření – většina jsou exporty – ale fungují snad dobře, snažil jsem se ošetřit co nejvíce krajních případů, a tak by si práce měla poradit se snad čímkoliv.

Neočekávám, že bych něco mohl v soutěži vyhrát, ale i tak mě tvorba aplikace velice bavila po celý semestr.

Závěrem bych chtěl poděkovat panu Doc. Kohoutovi za skvělé a zábavné přednášky a cvičení a za veškerou pomoc, kterou nám poskytl. Stejně tak bych chtěl poděkovat panu Ing. Červenkovi za veškerou pomoc a rychlé odpovědi na discordu.