Zápočtový projekt pro Programování 2

Matěj Gajdoš

25. srpna 2023

1 Uživatelská dokumentace

Tento program slouží k dynamickému procedurálnímu generování šestiúhelníkové mapy. Mapa se dotváří postupně podle toho, jak s ní uživatel pohybuje, její rozměry se mohou stále zvětšovat až do limitů hardware počítače, na němž program běží.

Projekt je veřejně dostupný na mém GitHubu na odkazu `https://github.com/thrundusakunus/mapGen`.

Ke spuštění programu je potřeba Python3 a jeho balíčky `numpy` a `tkinter`. Na některých linuxových distribucích může být některý z těchto balíčků již nainstalován, případně k instalaci lze použít `pip install numpy` a `pip install tk`. Druhý příkaz na některých platformách dělá problémy, pak je potřeba stáhnout `tkinter` balíček v repozitáři. V takovém případě se instalace odvíjí od toho, jaký package-manager daná linuxová distribuce využívá.

Program se spouští příkazem `python main.py`. K jedinému ovládání slouží klávesy W, A, S, D k posouvání mapy nahorů, doleva, dolů a doprava (v tomto pořadí).

2 Projektová dokumentace

2.1 Úvod

Vize a realita projektu

Ne vše se nakonec vyvede tak, jak se to na samém začátku plánovalo. Hlavní myšlenka, tj. vytvářet za běhu programu náhodnou mapu, která se bude v případě potřeby rozšiřovat o další políčka tak, aby výsledek vypadal alespoň trochu věrohodně a ne jako náhodný šum, se vyvedla a výsledný program dělá to, co se od něj od začátku očekávalo. Nicméně některým částem programu bylo třeba věnovat více času, než jsem na začátku očekával, a to zase na úkor jiných částí, na které jsem se původně plánoval soustředit.

Myslel jsem si, že hlavní důraz budu v projektu klást na samotné metody realisticky vypadajícího terénu. Sice jsem vyzkoušel vícero způsobů, jak zarovnávat nadmořskou výšku a jak tvořit řeky, ale ve výsledném programu jsem pro oba tyto problémy nechal

vždy jen jedno řešení, které během experimentování vypadalo nejvíce slibně. Když měl projekt jenom hrubé rysy, plánoval jsem pro určování terénu více parametrů (teplota, vlhkost). Když jsem ale později přemýšlel, jak je zakomponovat, přišlo mi, že způsob jejich výpočtu by se víceméně shodoval s určováním nadmořské výšky (minimálně se jedná o stejný problém), takže bych vlastně nepsal žádný další zajímavý kód, jen bych přidával nové barvy pro políčka v závislosti na mezihře těchto parametrů, což mi přišlo zbytečné.

Naopak jsem při plánování projektu podcenil samotnou tvorbu mapy. Ukázalo se, že pohybovat mapou tak, aby se přesouvala viditelná políčka, vznikala správná nová políčka, ta se v grafu propojovala se všemi svými sousedy a aby to všechno běželo rozumně rychle a nedělalo se příliš zbytečné práce, není jednoduché. Většina kódu se věnuje právě tomuto úkolu. I v tomto případě vznikla spousta slepých nebo neperspektivních uliček, kterými jsem se ubíral, než jsem se nakonec spokojil s nynějším způsobem vytváření mapy. Dají se nalézt ve starších git commitech, ale zde je až na výjimky nebudu popisovat, neboť v nynějším programu již nemají místo a tento text je už tak dost dlouhý :).

Program z ptačí perspektivy

Popisu důležitých metod se budou věnovat samostatné sekce. Zde chci uvést základní přehled toho, co se v programu děje, ať je lépe vidět u jednotlivých komponent, k čemu jsou potřebné.

V programu jsou dvě nejvyšší metody. Jednou z nich je konstruktor grafické třídy **WindowHandler**, který spustí GUI, nechá vytvořit mapu a celou ji vykreslí. Také vytvoří vazby mezi zmáčknutím tlačítek a pohybem mapy, který je v režii druhé zmiňované důležité metody. Na vytvoření grafického okna a mapy v něm tedy stačí akorát vytvořit nový objekt třídy **WindowHandler**.

Metodou věnující se pohybu mapy je *WindowHandler.moveMap*. Ta posouvá políčka, která jsou vykreslená na grafickém plátně, a přitom si hlídá, jestli nějaké políčko:

- Se nemá nově vykreslit (bylo mimo plátno, ale mapa se posunula a nově tam je).
- Se nemá z plátna smazat (bylo na plátně, ale mapa se posunula a již tam není).
- Se nemá vytvořit (po posunu mapy by byl vidět konec mapy).

Nová políčka se tvoří po celých vrstvách pomocí těchto metod:

- *Map.generateNewLayers*
- *Map.generateLeftSide*
- *Map.generateUpSide*

- *Map.generateRightSide*
- *Map.generateDownSide*

K vytváření více realistického terénu slouží *Map.updateSandpiles*. Standardně se nejdřív vytvoří nové vrstvy políček, těm se pak zahladí terén, následně se rozhodne, ve kterých by měly pramenit nové řeky, a ty se pak dotvoří pomocí *Map.makeRivers*.

2.2 Přehled souborů a tříd

main.py

Slouží pouze jako hlavní program ke spuštění.

linkedListPkg.py

Obsahuje třídy **Node**, jejíž instance jsou uzly spojového seznamu **LinkedList**. Tento spojový seznam je obousměrný a udržuje si odkaz na první, poslední a prostřední prvek. Tato datová struktura je užitečná pro uchovávání odkazů na okrajová políčka mapy (`Map.boundary_tiles[key]`, kde `key` $\in$ {'left', 'up', 'right', 'down'}) – algoritmus vytváření nových vrstev polí potřebuje rychlé přidávání na počátek i konec posloupnosti (což spojový seznam umožňuje v $\mathcal{O}(1)$, zatímco klasický list o n prvcích přidává na svůj začátek v $\mathcal{O}(n)$) a také přístup k prostřednímu prvku. To je trochu nestandardní požadavek, se kterým by si například struktura `collections.deque` neporadila.

Objekt **Node** obsahuje odkaz na předchozí a následující uzel a proměnnou pro vlastní hodnotu v daném uzlu. **LinkedList** si udržuje odkaz na první, prostřední a poslední uzel a počítadlo `middle_counter` iniciované nulou. Metody **LinkedList** jsou *iterator* (pro snadné iterování skrze prvky od začátku do konce seznamu) a standardní *append* (přidání na konec), *prepend* (přidání na začátek), *popleft* (odstranění a navrácení počáteční hodnoty) a *popright* (odstranění a navrácení koncové hodnoty), vedle nich obsahuje třída ještě metodu *balance*. Ta se interně volá při každém volání jedné z předchozích čtyř metod a slouží k aktualizaci prostřední hodnoty.

Při každém *popleft* a *append* se `middle_counter` zvýší o jedna, při každém *popright* a *prepend* se `middle_counter` sníží o jedna. První dvě metody způsobují, že se nynější prostřední hodnota nachází blíž k začátku než ke konci seznamu (jedna počáteční hodnoty maže, druhá rozšiřuje konec). Naopak zbylé dvě metody zase prostřední hodnotu přibližují víc ke konci (ubíráním koncových hodnot a přidáváním hodnot na začátek). `middle_counter` zachycuje tento posun reálného prostředku seznamu – pokud je `middle_counter` roven dvěma, znamená to, že opravdový prostředek je v seznamu o jeden uzel dál než nynější prostřední uzel, a pokud je `middle_counter` roven minus dvěma, je opravdový prostředek v seznamu naopak o jeden uzel před nynějším prostředním uzlem. Metoda *balance* pak prostě zkontroluje,

zda není `middle_counter` $\in \{-2, 2\}$, a pokud ano, příslušně posune odkaz na prostřední uzel a nastaví `middle_counter` na nulu. Tím je zaručeno, že jakkoliv se bude seznam modifikovat, odkaz na prostřední hodnotu bude vždy co nejvíc uprostřed seznamu to půjde, přitom komplexita pro tuto konzistenci je stále $\mathcal{O}(1)$.

riverPkg.py

Obsahuje třídy **RiverVertex** a **RiverSegment**. Obě slouží k reprezentaci části řeky v jednom poli. **RiverVertex** je začátkem nebo koncem řeky a vykresluje se z prostředka pole na jednu z jeho hran, zatímco **RiverSegment** představuje ostatní části řeky, tudíž se vykresluje z jedné hrany pole doprostřed pole a odtud na jinou hranu. Celá řeka by se tedy dala chápat jako posloupnost instancí (`RiverVertex`, `RiverSegment`, `RiverSegment`, ..., `RiverSegment`, `RiverVertex`), což ale není v kódu potřebné.

mapPkg.py

Obsahuje třídy **Tile** a **Map**.

Třída **Tile** reprezentuje jednotlivá šestiúhelníková políčka mapy. Jejími hlavními proměnnými jsou:

- `Tile.x`, `Tile.y` ... Souřadnice středu tohoto pole na plátně aplikace.
- `Tile.gui_id` ... ID odkazující na vykreslený šestiúhelník na plátně odpovídající tomuto poli.
- `Tile.altitude` ... Číslo z intervalu $[-1, 1]$ reprezentující nadmořskou výšku tohoto pole.
- `Tile.rivers` ... Seznam řek (tj. objektů tříd **RiverSegment** a **RiverVertex**), které prochází tímto polem.
- `Tile.neighbours` ... Slovník s klíči { 'w', 'nw', 'ne', 'e', 'se', 'sw' } odpovídajícími světovým stranám. Hodnotou klíče je buď odkaz na jinou instanci třídy **Tile**, odpovídající sousedovi tohoto pole na dané světové straně, nebo `None`, pokud aktuální pole na této straně žádného souseda nemá.
- `Tile.gui_active` ... Boolean, `True` právě tehdy, když je toto pole momentálně vykreslené na plátně (tj. nachází se na obrazovce).

Třída **Tile** neobsahuje žádné komplikovanější metody, většina jich slouží k ušetření pár řádků kódu na jiných místech.

Třída **Map** reprezentuje celou mapu a obsahuje metody určené pro manipulaci s ní. Jejími proměnnými jsou `Map.centre_tile`, v níž se uchovává odkaz na prostřední

políčko mapy, a `Map.boundary_tiles`, což je slovník obsahující seznamy políček, které se nachází na hranicích mapy.

Většina metod této třídy má v tomto textu vlastní sekci. Zde zmiňme pouze *Map.tileIterator*, která slouží jako iterátor jednotlivých políček mapy. Políčka si udržují interní pravdivostní proměnnou `Tile.iterator_state`, která v průběhu metody označuje, která pole již byla navštívena. Při zavolání *Map.tileIterator* si metoda zapamatuje její nynější hodnotu (přečte ji z `Map.centre_tile`) a dá do zásobníku pro začátek `Map.centre_tile`, jemuž přepne `Tile.iterator_state`. V každém kroku se odebere políčko ze zásobníku, každý jeho existující soused se starým `Tile.iterator_state` se přidá na zásobník a přepne se mu `Tile.iterator_state`, následně se aktuální políčko navrátí pomocí `yield` a ve smyčce se přejde k dalšímu poli v zásobníku.

Metodu lze volat i s argumentem `True` – pak iterátor navrací pouze ta pole, která jsou momentálně graficky vykreslená (což se zjistí podle `Tile.gui_active`), což je užitečné pro optimálnější posun polí v *WindowHandler.moveMap*. V tu chvíli se ale ztrácí globální konzistentnost `Tile.iterator_state` napříč celou mapou – po jednom běhu *Map.tileIterator* jen skrze aktivní políčka se těmto aktivním políčkům přepne `Tile.iterator_state`, ale neaktivním zůstane jejich starý. To by byl ale problém pouze pro budoucí volání *Map.tileIterator* se vstupem `False`, k němuž už nedochází – pouze při prvotní tvorbě mapy se prochází celá mapa, ale pak při posouvání mapy se vždy již prochází jen přes graficky aktivní políčka. Jen je potřeba při aktivaci graficky neaktivních polí nebo tvorbě nových polí zajistit, že `Tile.iterator_state` se nastaví na nynější potřebnou hodnotu.

GUI.py

Má na starosti veškeré vykreslování skrze třídu **WindowHandler**, která má přístup k hlavnímu tkinter oknu aplikace, k vykreslovacímu plátnu v něm a k objektu třídy **Map**. Souřadnicový systém vykreslovacího plátna se řídí standardem kartézských souřadnic s počátkem v levém horním rohu okna, osou x směřující vodorovně doprava a osou y směřující svisle dolů.

Metody **WindowHandler** jsou z většiny buď jednoduché a přímočaré, nebo mají v tomto textu vlastní oddíl. Za zmínku možná stojí *WindowHandler.setColourOfTile*, která vstupnímu políčku přiděluje barvu na základě jeho nadmořské výšky. Volba jak to udělat je umělá, v kódu je nastavení několika barev pro určité nadmořské výšky, které při vykreslení vypadají dobře. `Tile.altitude` < 0 odpovídá moři.

2.3 Proces přidávání vrstev políček

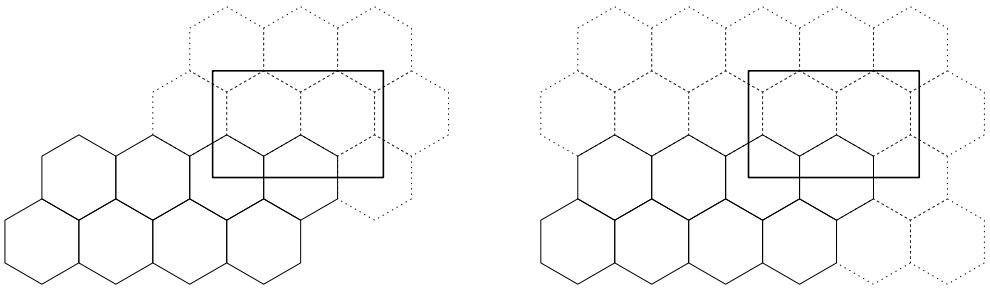
Tvorba mapy probíhá dynamicky, tj. teoreticky může být libovolně rozměrná a v případě potřeby se zvětšuje. Rozšiřuje se vždy celá jedna strana mapy, k

tomu slouží čtyři metody třídy **Map**: *Map.generateLeftSide*, *Map.generateUpSide*, *Map.generateRightSide* a *Map.generateDownSide*. Ačkoliv tedy mapa sestává z šestiúhelníkových polí, její celkový tvar je vždy přibližně obdelníkový.

V jedné sekci popíšeme *Map.generateLeftSide*. *Map.generateRightSide* probíhá díky svislé osové symetrii analogicky s prohozenou rolí západu a východu. *Map.generateUpSide* (a obdobně fungující *Map.generateDownSide*) budou popsány samostatně v další sekci, neboť pracují trochu jinak. Kdyby mapa sestávala ze čtvercových polí, bylo by možné tyto čtyři funkce naráz zakomponovat do funkce jedné, která by elegantně vytvořila požadovanou vrstvu podle vstupu svého argumentu strany. Šestiúhelníková mapa má ale trochu komplikovanější geometrii a redukce čtyř metod do jedné by spíš než ke zkrácení kódu vedla k jeho nepřehlednosti a spoustě podmínkových bloků, proto jsou algoritmy pro jednotlivé strany separované do vlastních metod.

Problém optimální tvorby nových políček

Proces přidávání políček často dělá více práce, než by bylo nutné. Ve starších commitech se vytvářela vždy jen ta pole, která byla zrovna potřebná pro vykreslení, jak znázorňuje obrázek (1a) (zatímco (1b) ukazuje nynější přístup vytváření po vrstvách).

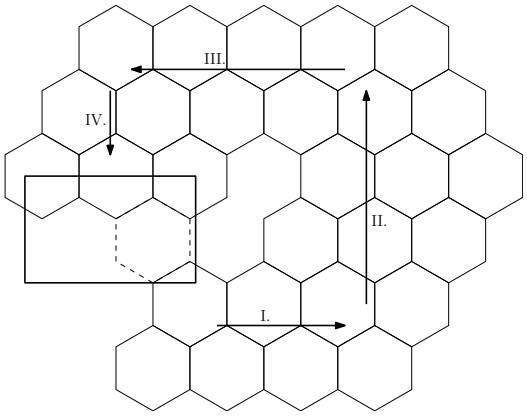


(a) Generování just-in-time. (b) Generování po vrstvách.

Obrázek 1: Dva různé přístupy ke generování nových polí. Obdelník reprezentuje aktuální pozici plátna, které se posunulo mimo vytvořenou mapu, a tedy je potřeba vytvořit nová pole (vyznačená tečkovaně).

Na obrázku (2) je znázorněn problém s tímto přístupem – dolní pole se vygenerovala na začátku, následně se plátno pohybovalo ve směru šipek I až IV. Čárkovaně vyznačené nově vytvořené pole by se mělo propojit jak s horními dvěma poli, tak s jedním dolním polem. Požadavek o vytvoření nového pole ale přišel od horních polí (ta byla v danou chvíli aktivní, zatímco dolní pole byla již nějakou dobu mimo obrazovku, a tedy neaktivní), a ta o blízkosti dolních polí nijak neví – geometricky jsou sice tato pole blízko u sebe, ale v použité grafové reprezentaci jsou od sebe velmi vzdálená přes všechna pole v cestě I - IV. Sice by šlo při vytvoření každého nového pole zkontrolovat celý již vzniklý graf, zda v něm již nejsou nějaká pole geometricky

blízko k novému vzniklému (skrze souřadnice `Tile.x` a `Tile.y`), ale to je ošklivé časově náročné řešení.



Obrázek 2: Problém s just-in-time gerenováním polí.

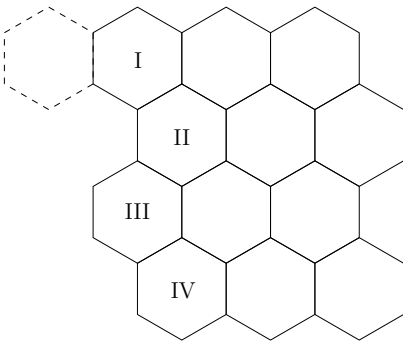
Problém by asi šlo řešit i jinak, například pomocí reprezentace mapy 2D polem namísto grafu. To je ale jednak trochu nepříjemné vzhledem k šestiúhelníkovému tvaru políček (2D pole by bylo přirozené pro čtvercová políčka; šestiúhelníkovou mřížku sice lze též umístit do 2D pole, ale je to taková nešikovná nepřirozená reprezentace), jednak by bylo potřeba pole různě rozšiřovat a přesouvat jeho prvky vlivem dynamické tvorby nových políček, takže bychom rychlejšího programu stejně nejspíš nedosáhli.

Tato řešení se nezdají být příliš dobrá, a proto jako výsledné, v programu použité řešení je při každém požadavku na vytvoření nového pole nagenerovat spolu s ním i celou vrstvu dalších polí a vytvořit potřebná propojení sousedů. Sice se tím vytváří i pole, která nejsou v daný okamžik pro vykreslení potřebná, ale vytvořená již zůstanou a při jejich pozdějším navštívení se naopak už žádná tvorba polí nemusí volat.

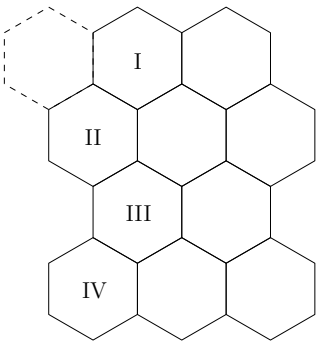
Tvorba levé vrstvy

Objekt **Map** si uchovává seznam odkazů na políčka tvořící levou vrstvu seřazená odshora dolů v proměnné `Map.boundary_tiles["left"]` obsahující strukturu **LinkedList**. Na obrázku (3) by tomuto seznamu odpovídala pole (I, II, III, IV).

Při tvorbě prvního, nejvrchnějšího políčka mohou nastat dvě situace, jak je znázorněno na obrázku (3), v závislosti na tom, zda svrchní políčko `Map.boundary_tiles["left"]` přesahuje doleva vzhledem k následujícímu hraničnímu poli (3a), nebo nepřesahuje (3b). V případě (3a) stačí nové pole propojit na východě s polem I, zatímco v případě (3b) je navíc nutné propojit nové pole na jihovýchodě s II. To, o kterou variantu se jedná, se v kódu pozná požadavkem na jihozápadního souseda pole I – pokud neexistuje, jedná se o situaci s přesahem, v opačném případě jde o situaci bez přesahu. Toto nové políčko se uloží do `new_boundary_tiles` nové instance **LinkedList**, která na konci metody *Map.generateLeftLayer* nahradí nynější `Map.boundary_tiles["left"]`.



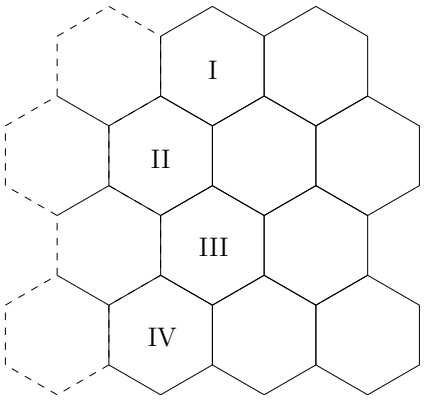
(a) Situace s přesahem.



(b) Situace bez přesahu.

Obrázek 3: Tvorba prvního pole nové levé vrstvy.

Následně program prochází přes zbylá levá hraniční políčka (v obrázcích II, III, IV) a každému z nich vytvoří na západě nového souseda. Toho zároveň propojí na jihovýchodě, pokud tam nějaké políčko existuje (tj. opět jde o levý přesah, v (3a) k tomuto propojení dojde pouze u II, zatímco v (3b) k němu dojde u I, které je startovací, a III). Na rozdíl od prvního políčka vždy navíc dochází k propojení se severovýchodním políčkem (to buď existovalo v `Map.boundary_tiles["left"]`, nebo bylo nově vytvořené, ale každopádně tam nějaké políčko je) a v některých případech se severozápadním políčkem (např. na obrázku (4) k tomu dojde pouze u III, protože předchozí políčko II přesahovalo doleva). Na jihozápadě k propojení v danou chvíli nedochází – sice např. na obrázku (4) má nové pole nalevo od III jihozápadního souseda, ale ten se vytvoří až při zpracovávání pole IV, tj. při vzniku políčka z pole III ještě není s čím na jihozápadě propojovat. Každé nové políčko se přidá do nově vznikajícího `new_boundary_tiles`.



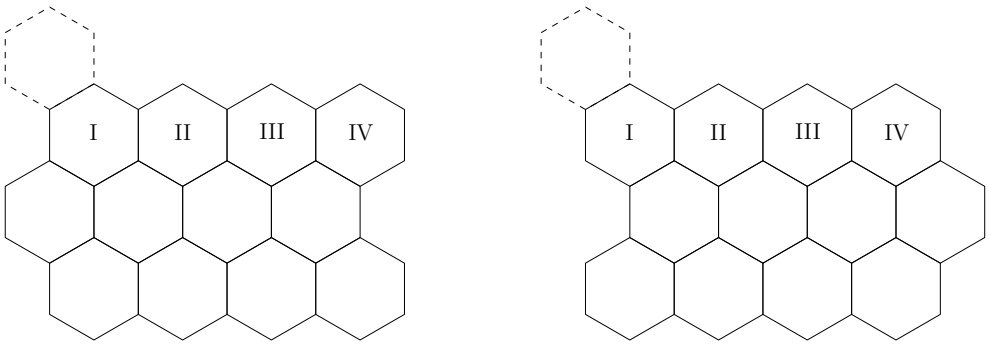
Obrázek 4: Tvorba celé nové levé vrstvy

Po navštívení všech políček `Map.boundary_tiles["left"]` se tato proměnná přealokuje na `new_boundary_tiles` jakožto na novou levou hraniční vrstvu mapy. Zároveň první prvek tohoto seznamu je součástí nejvrchnější vrstvy mapy, takže ho je

potřeba přidat na začátek `Map.boundary_tiles["up"]` pomocí *LinkedList.prepend* (proto je použit **LinkedList** namísto klasického Python seznamu). Obdobně poslední prvek `new_boundary_tiles` je novým prvním prvkem nespodnější vrstvy, proto se přidá na začátek `Map.boundary_tiles["down"]`.

Tvorba horní vrstvy

Tvorba horní vrstvy je podobná tvorbě levé vrstvy, liší se hlavně ve zpracování počátečního a koncového políčka `Map.boundary_tiles["up"]`. U prvního pole je potřeba rozhodnout, zda se má na jeho severozápadě vytvořit nové políčko. Opět dochází ke dvěma situacím v závislosti na levém přesahu prvního pole `Map.boundary_tiles["up"]`, viz obrázek (5). V případě (5a) se severozápadní políčko vytvoří, ale v případě (5b) ne, neboť toto nové políčko by moc přesahovalo doleva. Bez této regulace by se při každém zavolání *Map.generateUpSide* horní vrstva rozšířila doleva vzhledem k minulé, takže by s každým dalším voláním měla nová horní vrstva více a více políček. Zároveň by geometrie levé krajní vrstvy byla odlišná, než s jakou program počítá, což by mohlo vést ke špatnému propojování nových polí při volání *Map.generateLeftSide*.



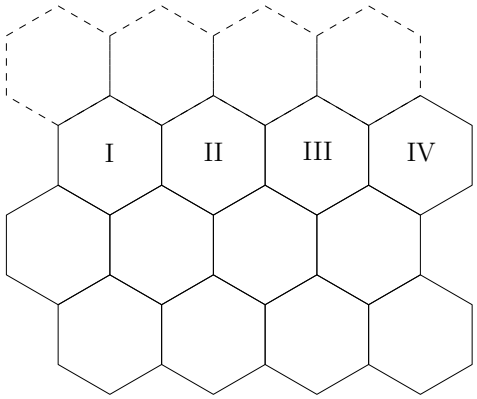
(a) Situace bez přesahu, pole se vytvoří. (b) Situace s přesahem, pole se nevytvoří.

Obrázek 5: Tvorba prvního pole nové horní vrstvy.

Po případném vytvoření severozápadního políčka k poli I a jeho uložení do nového **LinkedList** `new_boundary_tiles` se následně prochází přes další pole v `Map.boundary_tiles["up"]`. V každém kroku se vytvoří nové pole na severozápad od aktuálního zpracovávaného pole a propojí se na jihozápadě (tam vždy souseda má) a obvykle i na západě (tam mívá za souseda předchozí vytvořené políčko, vyjma prvního pole vytvořeného v situaci (5b)). Každé nové pole se vloží do `new_boundary_tiles`.

Nakonec se obdobně jako u prvního pole `Map.boundary_tiles["up"]` vyhodnotí, zda se má přidávat jedno nové pole na severovýchod k poslednímu hraničnímu poli. Na obrázku (6) by šlo o to, zda vytvořit pole na severovýchod od IV (na severozápadě již bylo pole vytvořeno při cyklení přes `Map.boundary_tiles["up"]`). V případě tohoto obrázku nové pole nevznikne, neboť by jinak přesahovalo moc doprava a narušovalo

strukturu pravé hraniční vrstvy mapy. Prakticky v programu dojde k podívání se na jihovýchodního souseda pole IV – protože tam žádný není (tj. je `None`), pole IV přesahuje doprava a nové severovýchodní políčko nevznikne.



Obrázek 6: Tvorba celé nové horní vrstvy.

Po vytvoření všech polí se přealokuje `Map.boundary_tiles["up"]` na nově vytvořený seznam `new_boundary_tiles` a na začátek `Map.boundary_tiles["left"]` a `Map.boundary_tiles["right"]` se přidá první, respektive poslední políčko ze seznamu `new_boundary_tiles`.

Map.generateDownSide pracuje podobně, akorát se prohodí role jihu a severu a přidávání polí do `Map.boundary_tiles["left"]` a `Map.boundary_tiles["right"]` se provádí na konci seznamu místo na začátku (dolní pole se vyhodnocují až na konec při tvorbě levých a pravých vrstev).

2.4 Zahlazování terénu

Hlavním faktorem rozhodujícím o vzhledu mapy je nadmořská výška políčka `Tile.altitude`, podle níž metoda *WindowHandler.setColourOfTile* určuje barvu políčka při vykreslování. Na tento parametr klademe několik podmínek:

1. Konzistence: Nadmořská výška se může měnit pouze v čase mezi vytvořením instance **Tile** a jejím prvním vykreslením na plátno, pak již musí zůstat fixní.

2. Hladkost: Nadmořská výška blízkých polí by se neměla skokově měnit, jinak by mapa nezachycovala rozumný terén.

3. Omezenost: Konkrétní hodnoty nadmořské výšky by měly ležet v předem daných mezích, aby metoda *WindowHandler.setColourOfTile* mohla rozumně kódovat nadmořskou výšku do barev.

Vyzkoušených metod bylo více, nakonec se dobře osvědčil postup implementovaný v metodě *Map.updateSandpiles*. Ta vezme jako vstup seznam políček, které začne

procházet a aktualizovat jejich nadmořské výšky. Pokud označíme nadmořskou výšku zpracovávaného políčka h a průměr nadmořských výšek jeho sousedů jako m, pak nová nadmořská výška zpracovávaného políčka se spočte jako $h' = \frac{h+\alpha m}{1+\alpha}$, kde $\alpha > 0$ je parametr (interně momentálně nastaven na 20). Protože h, $m \in [-1, 1]$, nezávisle na konkrétní volbě α platí $h' \in [-1, 1]$, čímž je zaručen požadavek 3 (pokud nadmořské výšky byly na začátku z $[-1, 1]$, pak i na konci budou z tohoto intervalu). Protože jde o jakousi variantu průměrování (hodnota nadmořské výšky se přiblíží okolnímu průměru, takto v každém poli, tj. hodnoty se začnou více shodovat), platí (alespoň okometricky bez nějaké formalizace) podmínka 2. K hodnotě h' se ještě přičte náhodná hodnota ξ z intervalu $[-\beta, \beta]$ ($\beta > 0$ je další parametr, momentálně nastavený na 0.01), pokud stále je $h' + \xi \in [-1, 1]$ – tím se do procesu uvede troška náhody.

Takto se vstupní pole *Map.updateSandpiles* projde celkem n-krát, v kódu $n = 5$. n je parametr, který nejvíce prodlužuje výpočet této metody, ale zároveň způsobuje, že terén má globálnější charakter, tj. místo menších ostrůvků vznikají kontinenty a oceány. To je dáno tím, že při více průchodech vstupního seznamu na sebe začnou mít nepřímo vliv i pole více vzdálená od sebe (např. nesousedící pole mající společného souseda se jedním výpočtem h' neovlivní, neboť nesousedí, ale při druhém průchodu už na sebe vliv mít můžou, neboť jejich nadmořské výšky ovlivnily jejich společného souseda a ten ve druhém průchodu zase ovlivní je).

Map.updateSandpiles pracuje s tím, že vstupní políčka již mají nějakou nadmořskou výšku, která se následně pouze zahladí. Prvotní nastavení nadmořské výšky se provádí metodou *Tile.setAltitude*, jež se volá již z konstruktoru třídy **Tile** a která s pravděpodobností $\frac{1}{2}$ nastaví `Tile.altitude` jako 1 a s pravděpodobností $\frac{1}{2}$ jako -1.

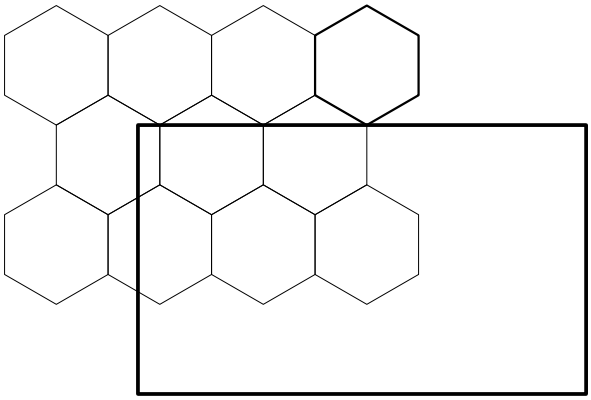
2.5 Počáteční tvorba mapy

Mapa musí již od začátku mít dostatek políček, aby měla co vykreslit po celém plátně. Konstruktor třídy **Map** pouze nastaví odkaz na prvotní centrální políčko `Map.centre_tile`, které zároveň uloží jako jedinou hodnotu ke všem klíčům hraničních políček `Map.boundary_tiles`. Většinu další práce provádí metoda *Map.generateGraph*.

Ta postupně rozšiřuje mapu do čtyř stran, dokud je to potřeba. Konkrétněji, zvolí se jedna ze stran 'left', 'up', 'right', 'down'. Následně se volá odpovídající metoda pro tvorbu vrstvy (*Map.generateLeftSide*, *Map.generateUpSide*, *Map.generateRightSide*, *Map.generateDownSide*) a to tak dlouho, dokud je odpovídající prostřední políčko dané hranice `Map.boundary_tiles[key]` viditelné. K tomu slouží metoda *WindowHandler.isTileOnScreen*, která vrací `True` právě tehdy, když je vstupní políčko viditelné na plátně (což se určí z jeho souřadnic).

Prostřední políčko se volí, protože má větší vypovídající hodnotu o tom, zda je opravdu potřeba mapu v daném směru rozšiřovat. Zvláště počáteční a koncová políčka v `Map.boundary_tiles` mohou být mimo plátno i přesto, že jejich hraniční strana

ještě potřebuje rozšířit, viz obrázek (7). Proto se v `Map.boundary_tiles` ukládají instance **LinkedList**, která umožňuje rychlý přístup k prostřední hodnotě.



Obrázek 7: Počáteční mapa po rozšíření doleva a nahoru. Vyznačené políčko v pravém horním rohu je sice součástí pravé hraniční vrstvy, kterou je ještě potřeba rozšířit, ale samo o sobě není momentálně na plátně vykreslované.

Takto se postupně rozšíří mapa do všech potřebných směrů. Následně se zavolá *Map.updateSandpiles* na všechna nová políčka, aby se zahladil terén nové mapy. Dále se znovu navštíví všechna políčka a u každého se pomocí metody *Tile.isRiverStart* rozhodne, zda by v tomto poli měla pramenit řeka. Políčka s pramenící řekou se ukládají do seznamu, který se nakonec předá jako argument metodě *Map.setRivers*, o níž pojednává následující oddíl.

Metoda *Map.generateGraph* se volá v konstruktoru třídy **WindowHandler**. V něm se dále projdou veškerá pole mapy pomocí *Map.tileIterator*, ta se nastaví na graficky aktivní (`Tile.gui_active = True`) a vykreslí se pomocí *WindowHandler.plotTile* (i v případě, že se políčko nachází částečně nebo úplně mimo plátno – to nedělá tkinteru problém a bude se jednat o nejvýše jednu dodatečnou vrstvu v každém směru, neboť tato pole způsobila, že se mapa již v daném směru nerozšiřovala při průběhu *Map.generateGraph*). Zároveň se u každého políčka program podívá do jeho `Tile.rivers` a případné řeky vykreslí pomocí *Map.plotRiver*. Tím je počáteční tvorba mapy u konce.

2.6 Tvorba řek

Metoda *Tile.isRiverStart* s pravděpodobností $k \cdot$ `Tile.altitude` rozhodne, že má v daném políčku pramenit řeka, kde $k = 0.1$ je vnitřní parametr. Řeky tudíž častěji pramení ve vyšších nadmořských výškách. Pokud je `Tile.altitude` ≤ 0, řeka v daném políčku jistě pramenit nebude.

S tvorbou řek se setkáváme jak při vzniku základní části mapy popsaném v minulé sekci, tak při vytváření nových polí vlivem posunu mapy, čemuž se podrobněji

věnuje sekce následující. V obou případech ale proces vypadá obecně takto (přičemž k prvnímu vykreslování dojde až po všech těchto krocích):

1. Vytvoří se nová pole.
2. Novým polím se upraví nadmořská výška pomocí *Map.updateSandpiles*.
3. Projde se přes nová pole a rozhodne se, ze kterých bude řeka pramenit. Pokud pro `tile` navrátí *Tile.isRiverStart* `True`, vytvoří se `river` instance třídy **RiverVertex** a dvojice (`tile`, `river`) se uloží do seznamu.
4. Vytvořený seznam dvojic se předá metodě *Map.makeRivers*.

Metoda *Map.makeRivers* slouží k vytvoření celých řek ze zmíněného seznamu dvojic políčko-pramen. Tento vstupní seznam funguje jako zásobník, který obecně obsahuje dvojice **Tile-RiverVertex** nebo **Tile-RiverSegment**. Cyklus postupně zpracovává zásobník, dokud není prázdný. Pro danou dvojici (`tile`, `river`) se nejdříve určí světové strany, na které by mohla řeka dále téct. Ty musí splňovat, že tam vůbec je nějaké pole, že toto pole ještě nebylo nikdy vykreslené (pro konzistenci, prakticky k tomu slouží proměnná `Tile.was_plotted`, která je v konstruktoru nastavena na `False` a při zavolání *WindowHandler.plotTile* na toto pole se přenastaví na `True`) a že nadmořská výška vedlejšího pole je nejvýše rovna nadmořské výšce nynějšího pole `tile` (aby řeky netekly nahoru).

Může se stát, že žádný vhodný směr neexistuje - v tu chvíli se `river` ani nevloží do seznamu řek `tile.rivers`, namísto toho se nastaví `tile.is_lake = True` a ze zásobníku se vezme další dvojice. Proměnná `Tile.is_lake` pak způsobí při volání *WindowHandler.setColourOfTile*, že se políčko vykreslí jako vodní, ikdyž má nadmořskou výšku vyšší než 0.

Pokud ale alespoň jeden vhodný směr existuje, `river` se přidá do seznamu řek `tile.rivers`. Zároveň se jeden z těchto směrů náhodně zvolí jako směr, kterým bude řeka dál proudit. Pokud je sousední políčko v daném směru mořem (tj. má nadmořskou výšku menší než 0), na zásobník se již nic nepřidává. Pokud sousední políčko není mořem, ale již v něm nějaká řeka teče (jeho seznam řek je neprázdný), na zásobník se nic nepřidá, pouze se v tomto poli vytvoří **RiverVertex** napojený na `river`, čímž graficky dojde k tomu, že se tato řeka bude vlévat do tamní již existující řeky. Nakonec, pokud pole není mořem a zároveň v něm ještě žádná řeka není, se vytvoří nový **RiverSegment**, jeho `RiverSegment.start_side` se nastaví tak, aby odpovídal napojení na aktuální `river` a spolu se svým políčkem se přidá do zásobníku.

Výsledkem je, že řeky tečou z kopce tak dlouho, dokud:

- Se nevlejí do jiných řek.
- Nedoústí do moře.

- Nedotečou do lokálního minima funkce nadmořské výšky, kde vytvoří jezero.
- Nedotečou na konec mapy nebo k poli, které již bylo někdy vykreslené.

2.7 Posouvání mapy

Při posouvání mapy by v nejjednodušším případě stačilo projít všechna pole mapy, každé z nich posunout na plátně pomocí tkinter metody *canvas.move* spolu s jeho řekami, a pokud by některé pole nemělo nějakého souseda, který by potřeboval vykreslit, tak ho vytvořit. To by ale s rostoucím počtem polí byla spousta zbytečné práce a program by byl velmi rychle nepoužitelný. Samotná posouvací část by měla asymptotickou složitost $\mathcal{O}(n)$, kde n je počet políček mapy. V programu je použito o něco méně primitivní řešení, které vede na komplexitu $\mathcal{O}(1)$ (což bohužel neplatí pro část s vytvářením nových potřebných polí – tomuto problému se více věnuje sekce o vytváření nových vrstev; přesto ale je užitečné mít alespoň rychlejší část s posouváním polí, byť vytváření nových polí následně může vyžadovat více a více času).

Každé políčko má interní proměnnou `Tile.gui_active`, která říká, zda je toto pole momentálně vykreslované. Zároveň na metodě *Map.tileIterator* lze vložením `True` do argumentu požadovat, aby iterovala jen přes pole, která mají `Tile.gui_active = True`. Při zavolání *WindowHandler.moveMap* pro posun mapy se prochází jen přes tato aktivní políčka. Těm se přenastavují souřadnice a jejich grafická vykreslení na plátně se posouvají. Pokud se některé z těchto aktivních políček nachází po aktualizaci souřadnic mimo plátno, zařadí se toto políčko do množiny políček k deaktivaci – ta se po posunu všech aktivních políček smažou z plátna a nastaví se na neaktivní. Dále, pokud má některé z aktivních políček nějakého neaktivního souseda, přepočtou se tomuto sousedovi souřadnice, a jestliže by se s novými souřadnicemi měl nacházet na plátně, zařadí se tento soused do množiny políček k aktivaci, která se po posunu všech aktivních políček nově vykreslí na plátno. Nakonec, pokud některé z aktivních políček nemá nějakého souseda, zaznamená se odpovídající strana, na které bude potřeba vytvořit nové vrstvy políček.

Po vykonání veškeré práce se zavolá metoda *Map.updateCentreTile*, která změní `Map.centre_tile` tak, aby proměnná odkazovala na políčko nejblíže středu plátna. Tím je zaručeno, že je `Map.centre_tile` vždy graficky aktivní, což je důležité pro *Map.tileIterator*, který začíná právě v tomto středovém políčku.

K vytváření nových vrstev slouží metoda *Map.generateNewLayers*, za kterou ale stejně většinu práce dělají již popsané metody pro vytváření vrstev na jednotlivých stranách mapy. Nejdůležitější na *Map.generateNewLayers* je, že nevytváří jen jednu vrstvu, nýbrž jich vytváří více naráz podle svého argumentu `chunk_size`. To má dva účely – jednak má díky tomu většina nových polí všech šest sousedů, což má význam pro *Map.updateSandpiles* (ve starších verzích programu při tvorbě jen jedné vrstvy vznikaly v důsledku nedostatku sousedů nepřirozené reliéfy připomínající

pruhy jednoho terénu), jednak se tím zvětšuje oblast ještě nevykreslených polí, v nichž mohou téct nové řeky. Interně je v kódu `WindowHandler.chunk_size = 10`.

Map.generateNewLayers tedy vytvoří daný počet vrstev, následně uhladí terén pomocí *Map.updateSandpiles*, projde všechna nová pole, rozhodne, kde mají začínat řeky, a ty dotvoří pomocí *Map.makeRivers*.

Asymptotická složitost přesouvací části *WindowHandler.moveMap* je $\mathcal{O}(1)$, protože počet políček, jež mohou být vykreslena na plátně, a tedy těch aktivních, přes které cyklus probíhá, je shora omezen nějakou pevnou konstantou (a u každého políčka se provede jen konstantně operací – vyhodnocení aktivace/deaktivace, výpočet souřadnic, podívání se na šest sousedů). Vytváření nových polí má bohužel lineární časovou složitost v počtu existujících polí mapy, protože čím víc polí se v mapě nachází, tím potenciálně delší vrstvy je potřeba vytvářet (počet kroků metody pro vytvoření vrstvy je přímo úměrný délce odpovídající `Map.boundary_tiles`). To vede k tomu, že když už je mapa velká, příliš požadavků na vytváření nových vrstev program zasekává. Nicméně lepší řešení než vytváření polí po celých vrstvách jsem nevymyslel (alternativy by byly horší) a při posunech mapy po již vytvořených políčkách je díky optimalizaci stejně rychlé nezávisle na tom, jak moc velká mapa je.