Katedra informatiky
Přírodovědecká fakulta
Univerzita Palackého v Olomouci

BAKALÁŘSKÁ PRÁCE

Procedurální generování lokací do hry typu
Dungeons & Dragons

2023

Jakub Novák

Vedoucí práce:
Mgr. Tomáš Urbanec

Studijní program: Informační technologie,
prezenční forma

Bibliografické údaje

Autor:	Jakub Novák
Název práce:	Procedurální generování lokací do hry typu Dungeons & Dragons
Typ práce:	bakalářská práce
Pracoviště:	Katedra informatiky, Přírodovědecká fakulta, Univerzita Palackého v Olomouci
Rok obhajoby:	2023
Studijní program:	Informační technologie, prezenční forma
Vedoucí práce:	Mgr. Tomáš Urbanec
Počet stran:	36
Přílohy:	elektronická data v úložišti katedry informatiky
Jazyk práce:	český

Bibliographic info

Author:	Jakub Novák
Title:	Procedural generation of locations for Dungeons & Dragons type game
Thesis type:	bachelor thesis
Department:	Department of Computer Science, Faculty of Science, Palacký University Olomouc
Year of defense:	2023
Study program:	Information Technologies, full-time form
Supervisor:	Mgr. Tomáš Urbanec
Page count:	36
Supplements:	electronic data in the storage of department of computer science
Thesis language:	Czech

Anotace

Bakalářská práce se zabývá metodami procedurální generace lokací do hry Dungeons & Dragons verze 3.5. V rámci práce jsou implementovány algoritmy založeny na metodě celulárního automatu a metodě binárního rozdělování prostoru do webové aplikace, která umožňuje uživateli vygenerovat jednoduchou mapu s pozicemi a stručným popisem podle zadaných parametrů.

Synopsis

Bachelor's thesis describes a few methods of procedural generation of locations for the game Dungeons & Dragons version 3.5. Within the thesis, algorithms based on cellular automaton and binary space partitioning are implemented into a web application, which allows the users to generate a simple map with positions and brief descriptions based on the provided parameters.

Klíčová slova: procedurální generace; webová aplikace; Python; Flask; Dungeons & Dragons

Keywords: procedural generation; web application; Python; Flask; Dungeons & Dragons

Děkuji svému vedoucímu práce Mgr. Tomáši Urbancovi za skvělé vedení a ohromnou pomoc při vytváření práce.

Odevzdáním tohoto textu jeho autor místopřísežně prohlašuje, že celou práci včetně příloh vypracoval samostatně a za použití pouze zdrojů citovaných v textu práce a uvedených v seznamu literatury.

Obsah

Seznam obrázků

Seznam zdrojových kódů

1 Úvod

V dnešní době se s počítačovými hrami setkáme na každém kroku díky jejich rostoucí popularitě. S rostoucí popularitou, ale rostou nároky na obsah her a s tím také nároky na lidi, kteří tento obsah vytvářejí. K vytvoření kvalitních a úspěšných her je v dnešní době zapotřebí čím dál tím více pracujících lidí ať už jsou to programátoři, návrháři, umělci nebo zvukový inženýři. Mnoho z nich přitom pracuje pravě na tvorbě herního obsahu. Potřeba rychle a levně vytvořit herní obsah dala možnost uplatnění procedurální generace. Tato technika umožňuje vytvářet dynamické a neustále se měnící herní světy, které poskytují hráčům nekonečnou rozmanitost a nové zážitky. Jedním ze zajímavých žánrů her, ve kterých je tato technika využívána, jsou role-playing hry. Mezi ty nejikoničtější zástupce tohoto žánru je stolní hra Dungeons & Dragons, která inspirovala celou řadu dalších her a je považována za počátek role-playing her. V rámci této práce se zaměříme na Dungeons & Dragons ve verzi 3.5, která je stále oblíbená mezi hráči a poskytuje možnost uplatnění procedurální generace k tvorbě obsahu.

Cílem této bakalářské práce je prostudovat a stručně pospat principy procedurálního generování obsahu do role-playing her typu Dungeons & Dragons. Následně vytvořit webovou aplikaci, která umožní generovaní map daných lokací spolu se stručným popisem výsledné mapy. Aplikace bude umožňovat nastavit základní parametry generované mapy jako je velikost, typ mapy, počet hráčů a jejich úroveň, maximální cena pokladu, ... Uživatel bude mít také možnost přidávat nepřátele a poklady, které budou poté zahrnuty ve vygenerovaných mapách.

2 Teoretická část

2.1 Procedurální generace obsahu

Procedurální generace obsahu je metoda, kdy je veškerý obsah nebo jeho část automaticky vytvářena počítačem pomocí algoritmů nebo přesněji *algoritmické vytvoření herního obsahu s omezeným nebo nepřímým vstupem uživatele* [1]. Tato metoda je dnes široce používaná v oblasti počítačové grafiky nebo animovaných filmů. V oblasti počítačových her jsou nejčastěji generovaným obsahem mapy, herní úrovně, textury, 3D objekty, postavy, pravidla apod. Dnešní hry využívají procedurální generaci pro různé účely. Pro malé části jako například generování zbraní ze hry *Borderlands* [2]. Při generování celých úrovní u her *Rogue* [3] a *Diablo* [4]. Nebo při generování celých galaxií ve hře *No Man's Sky* [5].

Použití procedurální generace pro generování herního obsahu může přinést mnoho výhod. Hlavní výhodou je snížení finančních nároků na vývoj her díky snížení počtu potřebných herních návrhářů. Dalšími výhodami mohou být zvýšení kreativity u návrhářů, kteří používají nástroje s procedurální generací a větší znovuhratelnost díky možné různorodosti vygenerovaného obsahu. Procedurální generovaní s sebou, ale přináší také mnoho nevýhod. Například při použití pro tvorbu map je složité hodnotit kvalitu vygenerované mapy ať už ze stránky hratelnosti, vzhledu nebo obtížnosti.

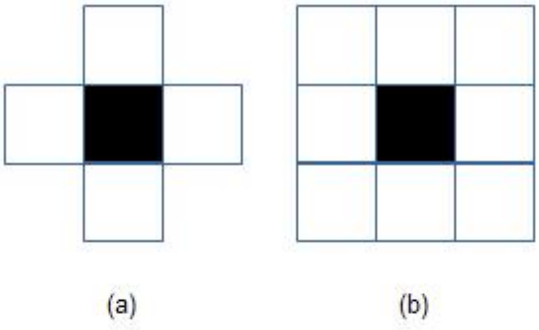
V rámci generování map a lokací do hry Dungeons & Dragons a podobných her se používá mnoho různých algoritmů. Každý z nich přináší rozdílné výsledky, které vyhovují odlišným potřebám. Používané algoritmy typicky obsahují tři prvky:

- Reprezentace mapy: způsob, kterým reprezentujeme strukturu výsledné mapy.
- Metodu pro konstrukci reprezentace mapy.
- Metodu pro vytvoření geometrie mapy (např. obrázek) z reprezentace mapy. [6]

Způsoby reprezentace mapy pomocí různých datových struktur a metody používané pro vytváření geometrie mapy se mohou lišit podle potřeb návrhářů. Proto se v následujících částech zaměřím pouze na metody, které provádějí konstrukci reprezentace mapy tj. procedurální generování mapy.

2.1.1 Celulární automat

Celulární automat (CA) je diskrétní výpočetní model, hojně studovaný v informatice, fyzice a bioinformatice jako model výpočtu, růstu a vývoje určitých jevů. Je všestranný a některé typy tohoto modelu jsou Turingovsky úplné [7]. Model se skládá z n-dimenzionální mřížky, množiny stavů a množiny pravidel vývoje. Nejčastěji používané mřížky jsou jednorozměrné (vektory) nebo dvourozměrné



Obrázek 1: a) Von Neumannovo okolí, b) Moorovo okolí

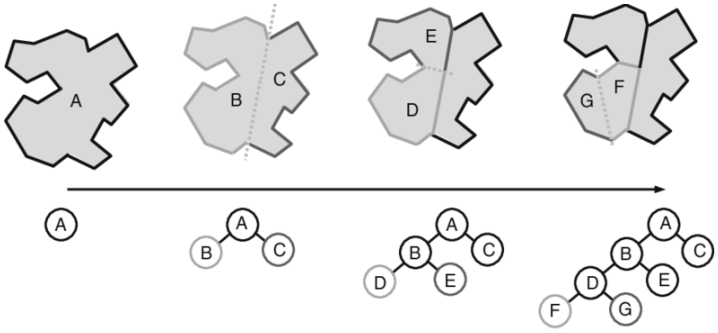
(matice). Ty se skládají z tzv. buněk, kdy každá buňka může nabývat nějakého stavu. V nejjednodušším případě buňky nabývají dvou stavů a to zapnutá nebo vypnutá. Z důvodu, že model představuje určitý růst a vývoj, bude docházet ke změně stavů buněk po čas t. Rozložení stavů buněk na začátku v čase t_0 se říká počáteční stav modelu. Počáteční stav každé buňky modelu můžeme vygenerovat pomocí náhody nebo jej definujeme. K definici můžeme použít pravděpodobnost, že buňka bude v počátečním stavu v určitém stavu, např. definujeme pravděpodobnost zapnuté buňky v počátečním stavu na 40%. Od tohoto okamžiku se model vyvíjí podle zadané sady pravidel. V každém čase t se každá buňka podle pravidel rozhodne o svém novém stavu na základě svého stavu a stavu sousedících buněk v čase $t-1$. Pro dvourozměrné mřížky jsou jako sousedící buňky nejčastěji používány buňky z *Moorova okolí* nebo *Von Neumannova okolí* (Obrázek 1). [8]

2.1.2 Rozdělování prostoru

Rozdělování prostoru je proces, který rozděluje prostor na dvě nebo více částí, tak aby každý bod rozdělovaného prostoru ležel přesně v jedné z těchto částí také nazývané buňka. Každá buňka je dále rozdělena rekurzivním způsobem za použitím stejného algoritmu. Takto rozdělované buňky vytvářejí určitou hierarchii. Nejpopulárnější metoda je binární rozdělování prostoru (binary space partitioning, BSP), která rozděluje prostor vždy jen do dvou podprostorů. Toto rozdělení nám umožňuje reprezentovat daný prostor jako binární strom nazývaný BSP strom (Obrázek 2). Rozdělování obrazu je nejvíce používáno v počítačové grafice díky rychlosti vyhledávání geometrických prvku v takto rozděleném prostoru. Přesněji je používáno při sledování paprsků a detekci kolizí. [9]

2.1.3 Metoda růstu založena na agentech

Tato metoda simuluje postupné rozšiřování a tvarování map pomocí entit nazývaných „agenti“. Agenti mají sadu pravidel, které definují jejich chování. Agenti jsou umístěni na náhodných nebo vybraných pozicích prázdné mapy. Poté postupně podle dané sady pravidel provádí různé operace na mapě jako třeba vykopávání chodeb a vytváření místností. Agent může být použit jeden nebo více. [10]



Obrázek 2: Reprezentace rozděleného prostoru pomocí BSP stromu

2.1.4 Evoluční algoritmy

Evoluční algoritmy (EA) jsou inspirovány přirozeným výběrem mezi živočišnými druhy tj. evoluční biologií. Principem EA je udržení si množiny možných řešení tzv. populace, kde se jedno řešení nazývá jedinec. Jedinci jsou hodnoceni pomocí fitness funkce, která vyhodnocuje, jak blízko je jedinec k optimálnímu řešení. Jedinci s nejlepšími výsledky fitness funkce mají možnost reprodukce, kdy pomocí operace křížení a mutace vytvářejí nové jedince tzv. potomky. Nejlepší řešení se tedy kombinují a vytvářejí tak nová a lepší řešení. Jedinci s nejhoršími výsledky fitness funkce jsou poté nahrazeny novými potomky a vzniká nová generace. Proces vzniku dalších generací se opakuje do té doby než je nalezeno řešení, které je dostatečně blízko k optimálnímu řešení. [11]

2.1.5 Generativní gramatika

Generativní gramatika je originálně metoda pro popis jazykových struktur a popis pravidel pro jejich tvorbu. Tyto struktury (věty) jsou modelovány podle konečné sady rekurzivních pravidel, která říkají jak vytvořit větší struktury z menších struktur. V kontextu generování map si můžeme představit větu popisující mapu, kterou chceme vytvořit. Věta se skládá z různých slov nebo frází popisujících např. topologii mapy nebo události, které hráč musí provést. Slova poté podle definované sady pravidel rozšiřujeme nebo měníme např. „místnost“ se muže náhodně změnit na „místnost s pokladem“ nebo „místnost s hádankou“. Slova, která rozšiřujeme a měníme se nazývají neterminály a slova, která dále neměníme se nazývají terminály. Změny provádíme do určitého případu např. dokud je ve větě neterminál „místnost“. V počáteční větě mohou být terminály i neterminály. Terminály se na počátku používají pro popis hranic mapy např. vstupní a výstupní místnost. [12]

2.2 Hry typu Dungeons & Dragons a prvky hry

Dungeons & Dragons (DnD) je fantasy role-playing hra (RPG), původně vytvořená Garym Gygaxem a Davem Arnesonem, vydána poprvé v roce 1974. V DnD

si hráči vytvářejí a následně hrají za fiktivní postavy, které se vydávají na dobrodružství ve smyšleném světě. Postavy hráčů tvoří družinu a společně pak bojují s nestvůrami, řeší záhady a problémy, získávají poklady a zkušenostní body, čímž se stávají silnějšími. Hra probíhá ve skupině více hráčů s jedním z nich zastupující roli *Pána Jeskyně* (v orig. *Dungeon Master*), který působí jako vypravěč a rozhodčí zodpovědný za pravidla, tvoří fiktivní prostředí hry, usměrňuje hráčské postavy a ztvárňuje cizí (nehráčské) postavy. Hráči se snaží splnit hlavní úkol dané kampaně s možností plnit také vedlejší úkoly. [13]

Některé hlavní prvky ze hry DnD jsou:

Dungeon – (v češtině možno Jeskyně, Žalář) je prostředí připomínající jeskyni nebo hrad. V textu budu pro totéž používat obecný termín mapa. Má alespoň jeden vstup a jeden výstup pro hráče.

Místnost – je prostor spojený s dalšími místnostmi pomocí dveří nebo chodeb. V místnosti se mohou vyskytovat nepřátelé, poklad, pasti atp.

Poklad – může být nějaká věc nebo zlaťáky.

Nepřátelé – většinou monstra v různém počtu. Mají své hodnocení výzvy určující průměrnou úroveň čtyř hráčů potřebnou k poražení.

Skupina hráčů – jeden a více hráčů, kteří vstupují do mapy. Každá hráčská postava má svou úroveň, podle které se poté počítá celková síla skupiny hráčů.

Celá hra nebo některé její části se hrají s použitím mapy, která je nejčastěji zobrazená jako mřížka čtverečků (n-úhelníků u jiných verzí hry). Po čtverečcích se hráči pohybují při soubojích a prozkoumávání pro lepší přehled nad hrou. Nejčastěji jsou mapy použity pro zobrazení hráčů uvnitř jeskyní, hradů, atp. Takové mapy jsou často rozděleny na více pater. Každá mapa by měla obsahovat vchod a východ, šachty a schodiště vedoucí do dalších pater a jednotlivé místnosti na mapě, které jsou očíslovány a jejich popis je připojen k mapě. Popis se nejčastěji skládá z nepřátel a pokladu v místnosti, ale může obsahovat i pasti, hádanky, atp.

3 Použité metody

Pro svou práci jsem si vybral dvě metody procedurálního generování a to celulární automat a rozdělování prostoru, které jsou nastíněny v sekcích 2.1.1 a 2.1.2. Obě metody tvoří jiný typ map. Mapy vytvořeny pomocí celulárního automatu více připomínají přirozené jeskyně na rozdíl od map tvořených rozdělováním prostoru, které spíše připomínají místnosti (např. místnost v hradu, zámku, atd.). Pro implementaci používám objektově orientované programování.

3.1 Algoritmus založený na CA

Při popisu CA jsme řekli, že se model skládá z matice, množiny stavů buněk a množiny pravidel vývoje. Pro použití při generování map chápeme každou část modelu trochu jinak. Matice nám představuje celou herní mapu, kdy každá buňka představuje herní políčko na mapě. Stavy buněk jsou dva a to kámen a podlaha. Kámen reprezentuje neprůchozí prostor a podlaha reprezentuje volné místo. Pravidlo vývoje je jedno a říká, že buňka, která má v Moorově okolí určitý počet kamenů se promění také v kámen.

Pro implementaci algoritmu založeném na CA jsem vytvořil třídu `CACave`. Třída má dvě hlavní metody `generate_map` a `make_svg`. Metoda `generate_map` je metoda pro konstrukci reprezentace mapy. Metoda `make_svg` je metoda, která vytváří geometrii mapy z její reprezentace. Mapa je reprezentována dvourozměrným polem v atributu *map*. Vytvoření nové instance třídy `CACave` bere mimo jiné tyto hlavní parametry ovlivňující vytvořenou mapu:

- Počet řádků.
- Počet sloupců.
- N iterací celulárního automatu.
- Minimální počet kamenů T kolem buňky, aby se proměnila v kámen.
- Pravděpodobnost F, že buňka v počátečním stavu bude podlaha.
- Odkaz na instanci třídy `CACave`, která reprezentuje vyšší patro.

Volání metody `generate_map` nejprve vytvoří dvourozměrné pole podle zadaného počtu řádků a sloupců. Hodnoty prvků v poli jsou při inicializaci počítány podle pravděpodobnosti F. Prvek bude podlaha s pravděpodobností F nebo kámen s pravděpodobností $1 - F$. Pole je uloženo do atributu `map`. Příklad inicializovaného pole je na obrázku 3a. Na vytvořené pole je poté aplikováno N iterací celulárního automatu. To zařizuje N volání metody `do_simulation_step` (algoritmus 1). Jedno volání metody `do_simulation_step` je jedna iterace CA. Metoda vytvoří nové pole o stejné velikost jako pole `map`. Hodnoty v novém poli

jsou počítány podle počtu kamenů kolem prvku na stejné pozici v poli `map`. Prvek bude kámen když má v Moorově okolí T a více kamenů jinak bude podlaha. Nové pole je poté uloženo do atributu `map`.

Tento algoritmus vytváří regiony buněk, které strukturou připomínají jeskyně v reálném světě jak můžete vidět na obrázku 3b po třech iteracích CA. Regionem rozumíme oblast sousedících buněk stejného typu.

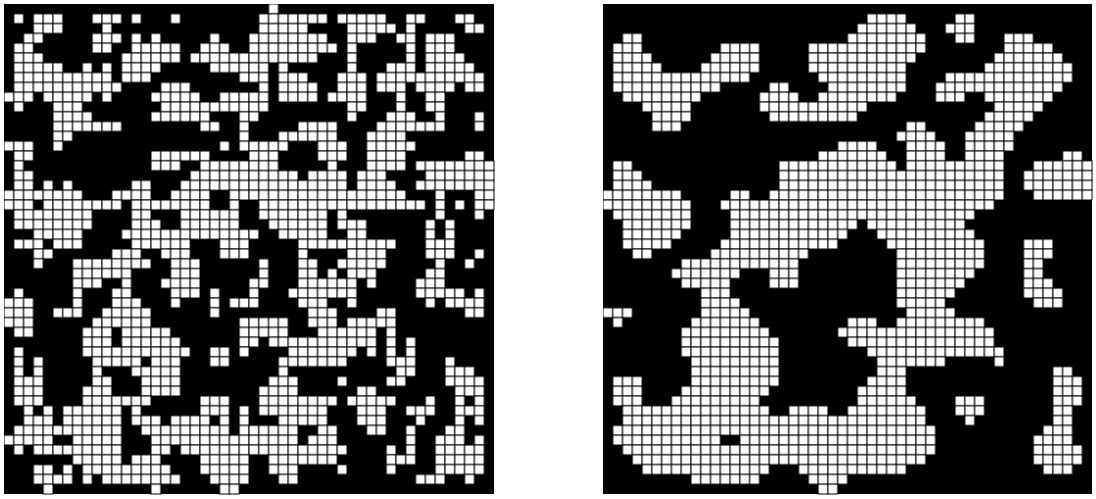
Metoda `do_simulation_step` používá následující metody a atributy:

- Metoda `count_bordering_rocks(row_index, col_index)`, která vrací počet kamenů v Moorově okolí prvku na souřadnicích (row_index, col_index).
- Atribut `rock_threshold` udávající nutný počet kamenů T kolem buňky, aby se proměnila v kámen.
- Atribut `rows` udávající počet řádků mapy.
- Atribut `cols` udávající počet sloupců mapy.

```
1  def do_simulation_step(self) -> None:
2        new_grid = np.zeros((self.rows, self.cols))
3
4        for row_index in range(self.rows):
5           for col_index in range(self.cols):
6              rock_count = self.count_bordering_rocks(row_index,
                  col_index)
7              if rock_count >= self.rock_threshold:
8                 new_grid[row_index][col_index] = ROCK
9              else:
10                new_grid[row_index][col_index] = FLOOR
11
12       self.map = new_grid
```

Zdrojový kód 1: Krok vývoje CA

Výsledné mapy jsou vhodné pro hraní DnD, ale zdaleka nejsou dokonalé. V mnoho případech trpí několika nedostatky a proto je potřeba vygenerovanou mapu dále zpracovat. Na mapě mohou vznikat malé shluky buněk uprostřed velkých regionů buněk jiného druhu např. uprostřed velkého regionu podlahy se vytvoří dvě políčka kamene. Tento problém lze vyřešit získáním všech regionů obou druhů prvků a zaplněním těch regionů, které mají velikost pod určitou hranicí. Nejprve jsou nalezeny veškeré regiony kamene a podlahy pomocí upraveného Flood fill algoritmu [14] v metodě `get_cells_region`, který jako parametry bere souřadnice začínajícího prvku. Metoda nejdříve inicializuje frontu a vloží do ní prvek z mapy na souřadnicích předaných parametry. Dokud není fronta prázdná jsou z ní odebírány prvky u kterých kontrolujeme zdali leží v hranicích mapy, že prvek ještě nebyl navštíven a že je stejného typu jako začínající prvek. Pokud



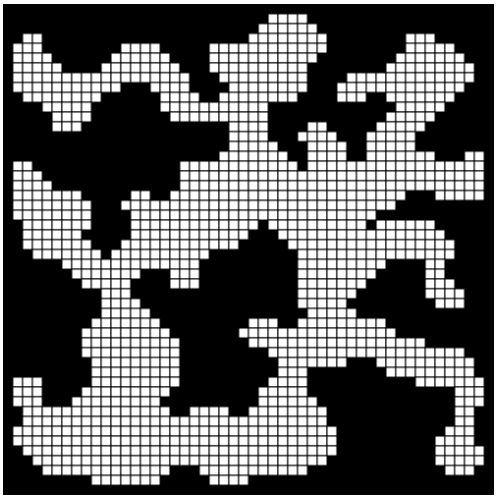
(a) Mapa při inicializaci pole (b) Mapa po třech iteracích algoritmu

Obrázek 3: Postupné vytváření mapy pomocí CA

toto všechno prvek splňuje jeho souřadnice jsou přidány do pole `cells`, prvek je označen jako navštívený a do fronty jsou přidány prvky z Von Neumannova okolí (obrázek 1). Metoda končí když je fronta prázdná a vrací pole `cells`, ve kterém jsou souřadnice všech prvků v regionu. Tímto způsobem lze získat veškeré regiony buněk na mapě. Regiony s velikostí pod určitou hodnotu jsou vyplněny prvky opačného typu.

Dalším nedostatkem je že algoritmem CA nijak neovlivníme propojení jednotlivých regionů prvků podlahy (propojení jeskyní na mapě). Mohou vznikat nedostupně části mapy, kde se hráč později při hraní nedostane. Spojování regionů je řešeno v metodě `connect_close_caves`, která jako parametr bere boolean `force_access_from_main`. Metoda prochází veškeré regiony podlahy a spojuje nejbližší regiony, které ještě nemají jiné spojení. Takové spojování nezaručí, že veškeré regiony jsou spojeny a lze je projít všechny. Proto je metoda volána znovu s parametrem `force_access_from_main = True`. Tento parametr říká, že chceme zaručit aby byla možnost se z každého regionu dostat do jednoho hlavního regionu. Hlavní region je určen jako největší region z regionů podlahy. Nyní metoda opět prochází regiony a spojuje ty nejbližší regiony, které mají přístup do hlavního regionu. Tím zaručíme, že všechny regiony jsou přístupné z hlavního regionu. Mapa po vyplnění nechtěných regionů a po spojení regionů s hlavním regionem je vidět na obrázku 4.

Posledním krokem při konstrukci reprezentace mapy je vybrání vstupu a východu pro hráče metodou `place_start_end`. Vstup do mapy je vybírán podle toho, zdali je generováno více pater mapy nebo jen jedno. Pokud generujeme více pater, tak je odkaz na instanci třídy `CACave`, která reprezentuje vyšší patro ukládán do atributu `upper_cave`. Pokud je `upper_cave` roven hodnotě None, tak je vstup mapy vybrán náhodně. Není-li roven hodnotě None tudíž, je generováno více pater, tak je pravděpodobnost 50%, že vstup bude vybrán náhodně a 50%,



Obrázek 4: Mapa po vyplnění nechtěných regionů a po spojení regionů

že vstup bude stejné nebo blízké políčko jako výstup z vrchního patra. Pokud je prvek na souřadnicích východu z vrchního patra typu podlaha, je určen jako vstup do nynějšího patra, a tak docílíme návaznosti mezi patry. Jinak je nalezen prvek typu podlaha, který je blízko souřadnic východu z vrchního patra a ten je vybrán jako vstup. Výstup je v každém případě vybrán náhodně, poté se spočítá vzdálenost vstupu od východu, pokud je vzdálenost moc malá tzn. vstup a výstup by byly hned u sebe, tak je výstup vybrán znovu.

3.2 Algoritmus založený na BSP

Při použití metody BSP pro generaci map rozumíme rozdělovaným prostorem celou mapu. Prostor mapy budeme reprezentovat pomocí BSP stromu. Kořen BSP stromu představuje celou mapu. Potomci kořene představují vzniklé podprostory. Listy stromu představují finální podprostory, do kterých umístíme místnosti.

Pro implementaci BSP stromu je použita třída `BSPTree`, která obsahuje mimo jiné dvě hlavní metody `create_map` a `make_svg`. Metoda `create_map` je metoda pro konstrukci reprezentace mapy a metoda `make_svg` je metoda, která vytváří geometrii mapy z její reprezentace. Vytvoření nové instance `BSPTree` bere mimo jiné tyto hlavní parametry:

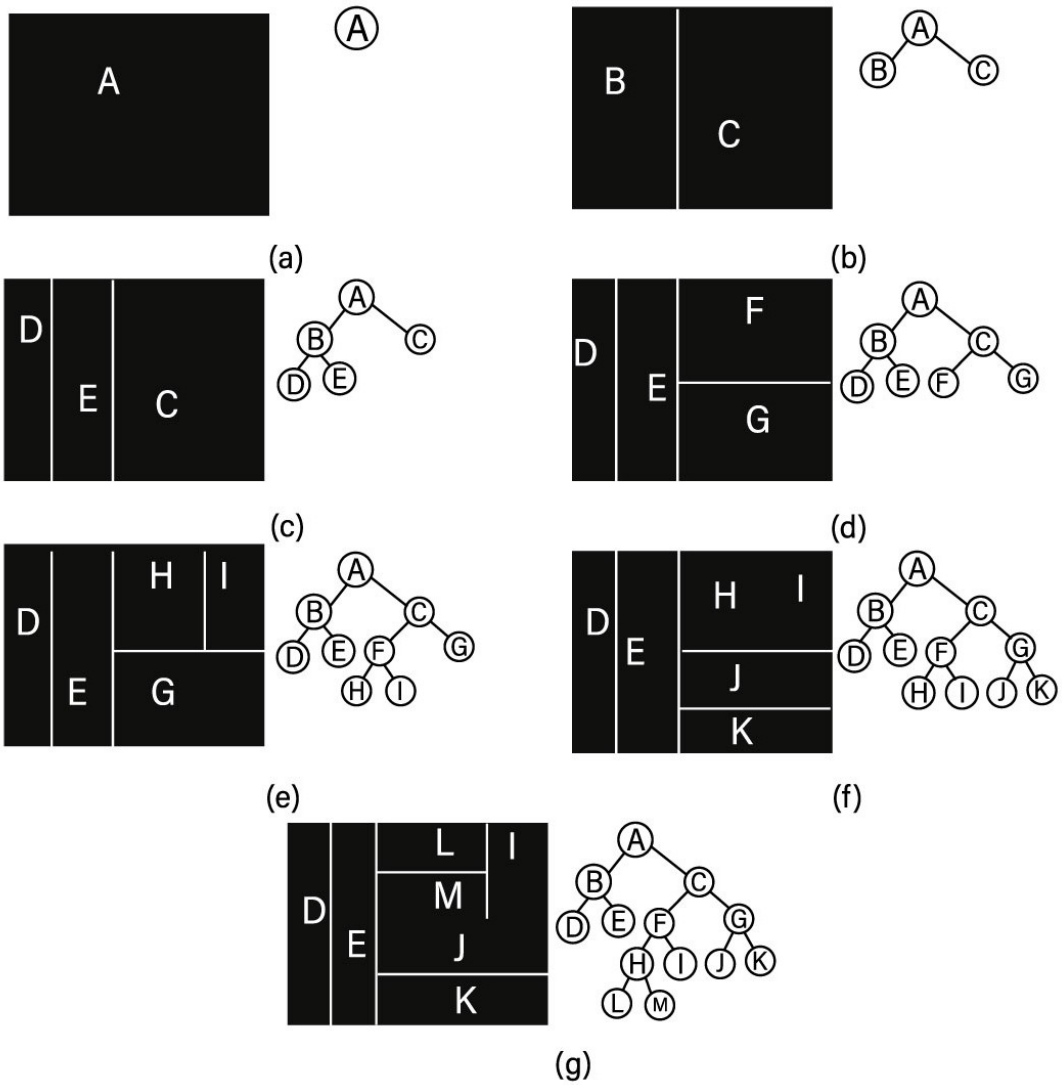
- Meze prostoru mapy reprezentovány obdélníkem třídy `Rectangle`.
- Odkaz na instanci třídy `BSPTree`, která reprezentuje vyšší patro.

Uzly stromu jsou implementovány ve třídě `BSPNode`. Ta v atributech obsahuje odkazy na další uzly (potomky), své meze (meze prostoru, který uzel představuje), meze místnosti, kterou uzel reprezentuje, a pole chodeb.

Metoda `create_map` nejprve na kořen stromu volá metodu `partition`, která rozděluje prostor uzlu do dvou podprostorů. Metoda `partition` (algoritmus 2) bere jako parametry minimální šířku rozdělovaného prostoru a minimální výšku rozdělovaného prostoru, které udávají minimální velikost finálního podprostoru, kde se budou nacházet místnosti. Metoda náhodně rozdělí prostor vertikálně nebo horizontálně na dva podprostory. Vytvořené podprostory jsou reprezentovány dvěma novými uzly, které budou potomky uzlu na který je metoda `partition` volána. Po rozdělení prostoru je metoda volána na nově vytvořené uzly (podprostory) a je tudíž rekurzivní. Rozdělování probíhá dokud je rozdělovaný prostor dostatečně velký tzn. dokud je větší než minimální šířka a výška zadaná parametry. Zároveň je při rozdělování prostoru kontrolován poměr stran vytvářených podprostorů abychom předešli extrémním případům. Při horizontálním rozdělení kontrolujeme poměr výšky k šířce podprostoru, aby nedošlo k příliš „nízkým a dlouhým“ podprostorům. V případě vertikálního rozdělení kontrolujeme zase poměr šířky k výšce podprostoru, aby nedocházelo k příliš „vysokým a úzkým“ podprostorům. Tvorba BSP stromu rozdělováním prostoru je znázorněna na obrázku 5

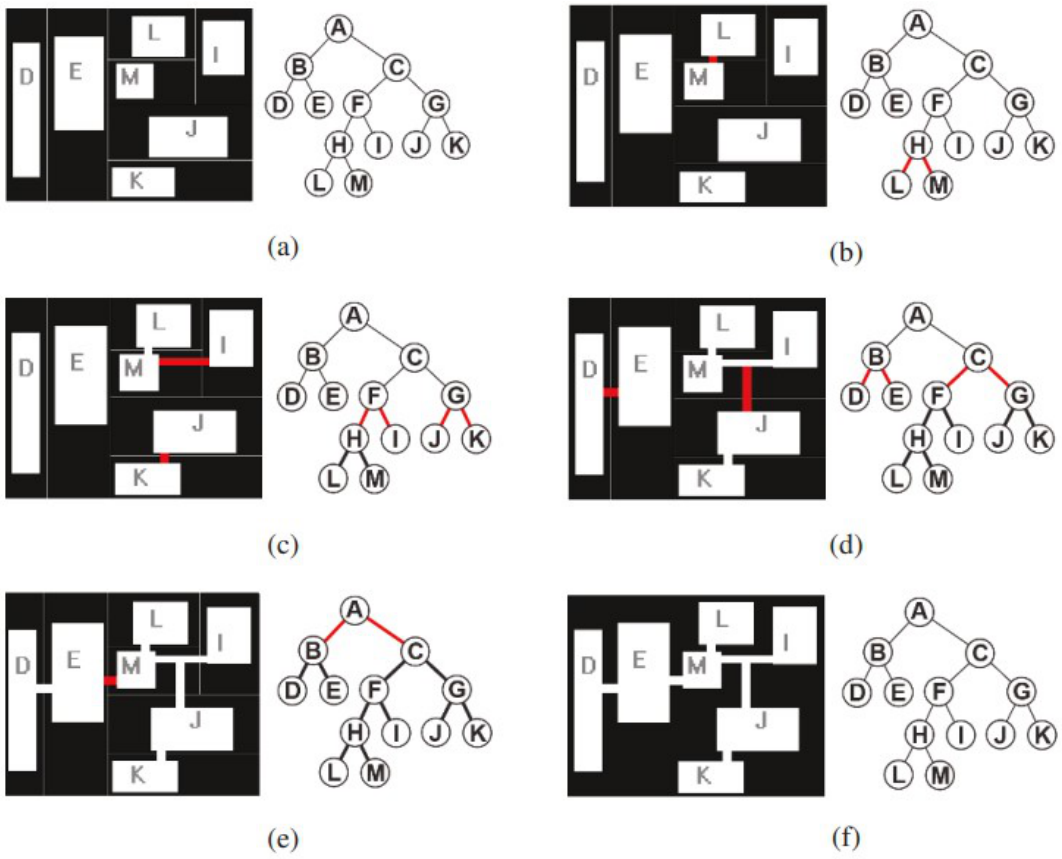
Dále vytvoříme v listech BSP stromu místnosti metodou `create_rooms`, která jako parametry bere minimální šířku a minimální výšku místnosti. Metoda prochází všechny listy BSP stromu a v podprostoru každého listu vytvoří místnost reprezentovanou objektem třídy `Rectangle`.

Pro spojení místností použijeme strukturu BSP stromu, která nám spojování zjednoduší a zaručí konektivitu všech místností. Chodby jsou tvořeny vzájemným propojením potomků stejného rodiče. Tvorba chodeb začíná u listů stromu, vždy jsou spojeny místnosti odpovídající potomkům stejného rodiče a pokračuje se ve stromě směrem nahoru ke kořenu. Spojování končí, když jsou spojeni potomci kořene. Spojování místností je znázorněno na obrázku 6.

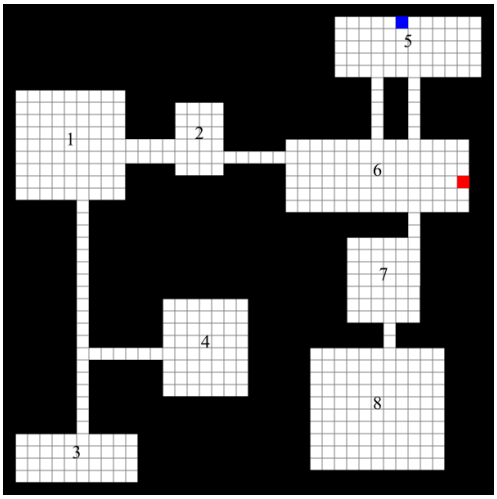


Obrázek 5: Rozdělování prostoru a s tím spojené vytváření BSP stromu ze sekce 3.2 knihy Procedural Content Generation in Games [7]

.



Obrázek 6: Spojování místností pomocí vlastností BSP stromu ze sekce 3.2 knihy Procedural Content Generation in Games [7].



Obrázek 7: Mapa vygenerovaná pomocí BSP algoritmu.

.

Dalším krokem je vybrání vstupu a východu z mapy. Při výběru vstupu jsou řešeny dvě možnosti. Pokud mapa nemá vrchní patro, tak je náhodně vybrán jeden směr, podle kterého je vybrána nejbližší místnost k okraji mapy daným směrem (při výběru *pravé* strany je vybrána místnost nejblíže k pravé hranici mapy). V takto vybrané místnosti je jako vstup určeno náhodné políčko na vybrané straně (v tomto příkladu by bylo na pravé straně). Tímto způsobem jsou vstupy vždy z vnější strany mapy a nestane se nám, že by vstup byl třeba uprostřed mapy. Pokud je mapa součástí více pater a je definováno vrchní patro tak bude vstup v místnosti, která je co nejblíže výstupu vrchního patra, dodávající pocit návaznosti jednotlivých pater. Při výběru východu jsou také řešeny dvě možnosti. Pokud mapa nemá vrchní patro a nebo je ze všech pater poslední (nemá navazující patro), tak je východ vybrán stejně jako vstup s tím rozdílem, že vybíraný směr nemůže být stejný jako směr vstupu, abychom nedostali vstup a východ v jedné místnosti. Pokud mapa má více pater a není poslední patro (má navazující patro), tak je vybrána náhodná místnost, která není místnost se vstupem, a v ní je udělán východ do dalšího patra. Příklad mapy vytvořené pomocí BSP je na obrázku 7

V rámci obou metod je po konstrukci reprezentace mapy potřeba vytvořit geometrii mapy metodou `make_svg`. Ta vykreslí mapu do SVG obrázku a uloží ji do složky **static**, ze které je poté obrázek načtený do webové stránky pro zobrazení uživateli.

```
1    def partition(self, min_width, min_height, cell_size):
2        if self.bounds.width <= 2*min_width or self.bounds.height <=
             2*min_height:
3            return
4
5        split_horizontal = random.choice([True, False])
6
7        if split_horizontal:
8            split_y = round(random.randint(self.bounds.top + min_height
                 , self.bounds.bottom - min_height) / cell_size) *
                 cell_size
9            rect_1 = Rectangle(self.bounds.left, self.bounds.top, self.
                 bounds.width, split_y - self.bounds.top)
10           rect_2 = Rectangle(self.bounds.left, split_y, self.bounds.
                 width, self.bounds.bottom - split_y)
11           if DISCARD_BY_RATIO:
12               if (rect_1.height / rect_1.width) < H_RATIO or (rect_2.
                     height / rect_2.width) < H_RATIO:
13                   self.partition(min_width, min_height, cell_size)
14                   return
15       else:
16           split_x = round(random.randint(self.bounds.left + min_width
                 , self.bounds.right - min_width) / cell_size) *
                 cell_size
17           rect_1 = Rectangle(self.bounds.left, self.bounds.top,
                 split_x - self.bounds.left, self.bounds.height)
18           rect_2 = Rectangle(split_x, self.bounds.top, self.bounds.
                 right - split_x, self.bounds.height)
19           if DISCARD_BY_RATIO:
20               if (rect_1.width / rect_1.height) < W_RATIO or (rect_2.
                     width / rect_2.height) < W_RATIO:
21                   self.partition(min_width, min_height, cell_size)
22                   return
23
24       self.left_child = BSPNode(rect_1)
25       self.right_child = BSPNode(rect_2)
26
27       self.left_child.partition(min_width, min_height, cell_size)
28       self.right_child.partition(min_width, min_height, cell_size)
```

Zdrojový kód 2: Implementace BSP algoritmu

4 Popis map

Popis je generován pomocí třídy `DescriptionGenerator`. Třída obsahuje metodu `generate_monster_description`, která vybírá nepřítele a jejich počet a metodu `generate_treasure_description`, která vybírá poklady a částku zlaťáků. Obě metody generují popis pro jednu místnost/region mapy, tudíž jsou procházeny veškeré místnosti/regiony na mapě a pro každou je vytvořen popis. Místnosti nebo regiony jsou před generací popisu rozděleny na několik skupin podle jejich velikosti:

- Velké – zde se objevují nepřátelé větší velikost a poklad.
- Malé – zde se objevuje pouze poklad.
- Ostatní – zde se objevují nepřátele menší velikosti nebo jsou prázdné.

Metoda `generate_monster_description` bere jako parametr seznam možných nepřátel, kteří se mohou v místnosti objevit. Seznam možných nepřátel je vybírán z databáze podle velikost místnosti a motivu vybraného uživatelem. Pokud je vybraný motiv „Random“, je náhodně vybrán některý z motivů v databázi. Ze seznamu možných nepřátel je poté vybrán jeden nepřítel a počet kolikrát se zde vyskytuje. Způsob vybírání je založen na pravidlech popsaných na stranách 48-49 v knize *Dungeon Master's Guide Core Rulebook* [15]. K vybírání je použita zjednodušená tabulka střetnutí z výše zmíněné knihy (tabulka 1), která je reprezentována dvourozměrným polem. Zvýrazněné hodnoty v prvním řádku udávají počet nepřátel a hodnoty v ostatních řádcích příslušnou úroveň nepřítele. Hodnota prvního sloupce v každém řádku udává celkovou sílu skupiny hráčů. Pokud je celková sílu skupiny hráčů například na druhé úrovni a úroveň náhodně vybraného nepřítele je jedna, podíváme se na druhý řádek, najdeme úroveň vybraného nepřítele ve třetím sloupci a počet nepřítel určíme podle prvního řádku třetího sloupce.

Metoda `generate_treasure_description` bere jako parametr seznam pokladů z databáze. Pokud uživatel nezadal celkovou hodnotu pokladu tak je pro každou místnost vybrána náhodná věc ze seznamu pokladů a k ní přidána suma zlaťáků podle průměrné úrovně hráčů. Pokud uživatel zadá celkovou hodnotu pokladu, jsou věci ze seznamu pokladů a sumy zlaťáků vybírány tak, že součet hodnoty pokladů a hdnoty zlaťáků ze všech místností dá dohromady celkovou hodnotu zvolenou uživatelem.

Veškeré popisy místností nebo regionů pro jednu mapu jsou spojeny do jedné proměnné, podle které je zobrazen popis map na stránkách webové aplikace (obrázek 8).

Cave: 1	The room looks empty.		
Cave: 2	**Treasure:**	Boots of Speed, Wondrous Item, 1 lb, 12000 gp	380 gp
Cave: 3	**Treasure:**	Berserking Sword, Cursed Item, 8 lb, 17500 gp	364 gp
Cave: 4	The room looks empty.		
Cave: 5	**Monsters:**	3× Human Warrior Skeleton, Medium, Undead, CR: 1	
Cave: 6	**Monsters:**	3× Hobgoblin, Medium, Humanoid, CR: 1	
	Treasure:	Chainmail, Armor, 40 lb, 150 gp	35 gp
Cave: 7	**Monsters:**	3× Ghoul, Medium, Undead, CR: 1	
Cave: 8	**Monsters:**	1× Allip, Medium, Undead, CR: 3	
	Treasure:	Book of Infinite Spells, Artifact, 3 lb, 50000 gp	385 gp

Obrázek 8: Popis vygenerovaný k mapě na obrázku 7

.

Celková úroveň skupiny	**Počet nepřátel**						
	1	**2**	**3**	**4**	**5**	**8**	**11**
1	1	0.5	0.3	0.25	0.16	0.125	0.125
2	2	1	0.5	0.5	0.3	0.25	0.16
3	3	2	1	0.5	0.5	0.3	0.3
4	4	2	1	1	0.5	0.5	0.3
5	5	3	2	1	1	0.5	0.5
6	6	4	3	2	1	1	0.5
7	7	5	4	3	2	1	0.5
8	8	6	5	4	3	2	1
9	9	7	6	5	4	3	2
10	10	8	7	6	5	4	3
11	11	9	8	7	6	5	4
12	12	10	9	8	7	6	5
13	13	11	10	9	8	7	6
14	14	12	11	10	9	8	7
15	15	13	12	11	10	9	8
16	16	14	13	12	11	10	9
17	17	15	14	13	12	11	10
18	18	16	15	14	13	12	11
19	19	17	16	15	14	13	12
20+	19+	18	17	16	15	14	13

Tabulka 1: Počet nepřátel na základě celkové síly skupiny hráčů

5 Programátorská dokumentace

5.1 Použité technologie

Pro vývoj aplikace jsem zvolil populární technologie, které jsou vhodné pro vývoj jednoduchých webových aplikací. Backend webové aplikace je napsán v jazyce Python pomocí frameworku Flask. Na frontend webové aplikace je použita knihovna Bootstrap. Pro práci s daty je použit databázový systém SQLite.

5.2 Python

Python je vysokoúrovňový programovací jazyk, který byl poprvé vydán v roce 1991. Python je dynamicky interpretovaný jazyk, což znamená, že se kód překládá až za běhu, čímž se případné chyby projeví teprve při spuštění programu. Je dynamicky typovaný a tudíž není potřeba explicitně deklarovat typy proměnných. Díky tomu, že je multiplatformní lze kód napsaný v Pythonu spustit na různých operačních systémech (včetně Windows, macOS a různých distribucích Linuxu). Obsahuje rozsáhlou standardní knihovnu, která nabízí velké množství modulů a funkcionalit. Nabízí jednoduchou a přehlednou syntaxi, která zlepšuje psaní a čtení kódu. [16]

5.2.1 Flask

Flask je populární mikro webový framework pro jazyk Python. Flask nevyžaduje žádné další nástroje ani další knihovny, ale podporuje rozšíření, která mohou přidávat další funkčnost, jako by byla implementována v samotném Flasku. Existují rozšíření pro objektově-relační mapovače, ověřování formulářů, zpracování nahrávání atd. Základem Flasku je URL směrování, které mapuje URL adresy na konkrétní funkce. Nejprve definujeme tzv. routy, což jsou pravidla, která určují, jak webová aplikace bude reagovat na různé URL adresy a HTTP metody. Routy jsou definovány pomocí dekorátorů nebo metod. Dalším důležitýn prvkem Flasku je použití šablon pomocí šablonovacího systému Jinja2. Ten umožňuje oddělení kódu od prezentace a snadnou tvorbu dynamických HTML stránek s proměnnými, cykly, podmínkami a dalšími šablonovacími prvky. [17]

5.2.2 Bootstrap

Bootstrap je framework používaný při tvorbě designu frontendové části webových stránek. Obsahuje velké množství předdefinovaných CSS tříd a Javascript funkcí pro tvorbu jednoduchého responzivního frontendu. Bootstrap dodržuje řadu moderních trendů ve webdesignu jako třeba responzivitu, mobile-first přístup, grid systém a flat design. [18]

5.3 Struktura aplikace

Jelikož je webová aplikace psaná v jazyce Python, soubory projektu ve složce **dungeon_generator** dodržují doporučenou strukturu pro balíček (package). Balíček je způsob, jak v Pythonu organizovat související soubory (moduly) do hierarchické struktury. Balíček projektu obsahuje:

- **___init___.py** – prázdný soubor, který slouží pro importování složky jako balíčku.
- **algorithms** – balíček, který obsahuje moduly pro generování map a generování popisu map. Soubor **bsp.py** obsahuje implementaci generování map pomocí algoritmu BSP. Soubor **ca.py** obsahuje implementaci generování map pomocí algoritmu CA. Soubor **desc_generator.py** obsahuje modul, který vytváří popis pro mapu.
- **static** – složka obsahující statické souboru projektu. Jsou zde obrázky výsledných map, CSS a Javascriptové souboru knihovny Bootstrap.
- **templates** – složka obsahující HTML šablony.
- **dependencies** – složka obsahující soubory potřebné k fungování některých použitých knihoven.
- **generator.py** – soubor obsahující funkci pro tvorbu map pomocí modulů ze složky **algorithms**.
- **routes.py** – soubor obsahující definice pravidel směrování pro pohybování se v aplikaci.
- **forms.py** – soubor, ve kterém jsou definice tříd reprezentující formuláře.
- **db.py** – soubor obsahující funkce pro práci s SQLite databází.
- **auth.py** – soubor obsahující definice pravidel směrování pro ověřování přihlášení.

6 Uživatelská příručka

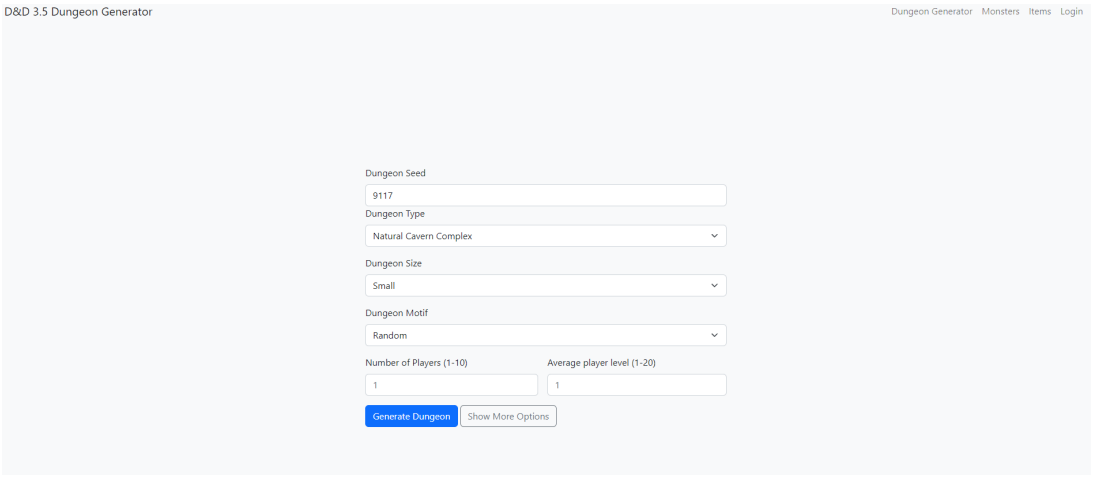
6.1 Generátor map

Hlavní stránkou aplikace je formulář na obrázku 9, kde si uživatel zvolí parametry pro vygenerovanou mapu. V poli *Dungeon Type* si může uživatel zvolit typ výsledné mapy, čímž volí použity algoritmus. Při zvolení *Natural Cavern Complex* se použije algoritmus CA a výsledná mapa bude připomínat přírodní jeskyni. Druhá možnost *Ruined/Occupied Structure* používá algoritmus BSP a výsledná mapa připomíná stavbu postavenou nějakými tvory. V poli *Dungeon Size* uživatel volí velikost výsledné mapy. To ovlivňuje velikost SVG obrázku na kterém bude mapa zobrazena. V rámci SVG se zobrazuje mapa jako mřížka buněk, kdy každá buňka má nějakou velikost. Proto výsledná velikost SVG bude vždy (počet řádků × velikost buňky) × (počet sloupců × velikost buňky). V dalším poli *Dungeon Motif* uživatel zvolí motiv mapy, podle kterého se zvolí motiv nepřátel/monster obývajících mapu. V posledních dvou polích *Number of Players* a *Average player level* uživatel napíše počet hráčů, kteří budou hrát na vygenerované mapě a jejich průměrnou úroveň postav. Podle těchto hodnot se určuje celková síla skupiny, která ovlivňuje hodnocení výzvy nepřátel (pro vyšší sílu skupiny se mohou objevit silnější nepřátele nebo větší počet slabších nepřátel). Tlačítko *Show More Options* zobrazuje uživateli další možnosti při nastavení parametrů mapy, které se mění podle zvoleného typu mapy. Zobrazené další možnosti reflektují podrobnější parametry pro jednotlivé algoritmy CA a BSP. Některé zobrazené možnosti jsou stejné pro oba typy:

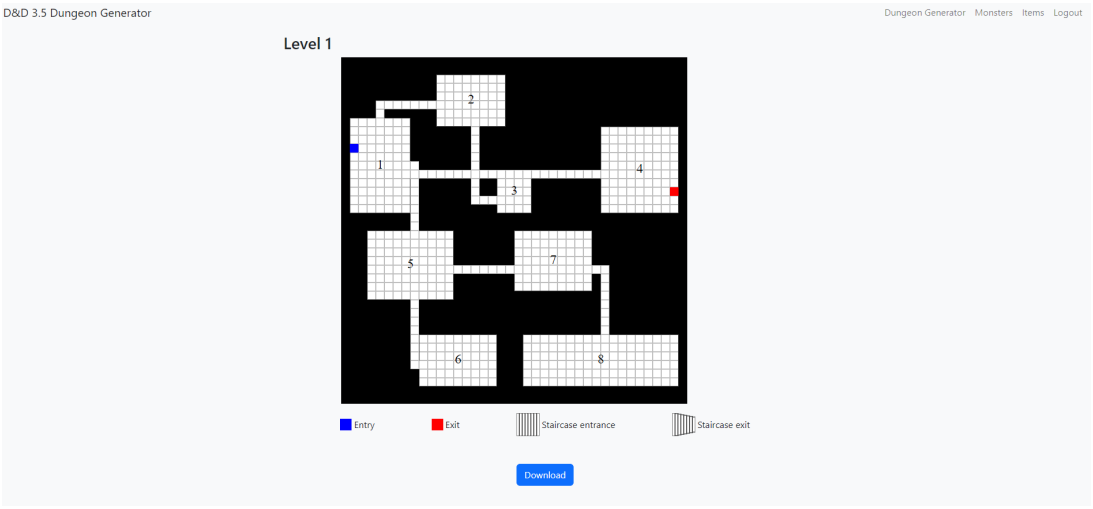
- *Number of Rows* – počet řádků mapy.
- *Number of Columns* – počet sloupců mapy.
- *Number of Levels/Floors* – počet pater vygenerované mapy.
- *Total Treasure Value* – celková cena pokladu na jednom patře mapy.
- *Cell size* – velikost buňky.

Unikátní možnosti při volbě *Natural Cavern Complex* jsou následující:

- *Floor Probability in T=0* – pravděpodobnost, že buňka v počátečním stavu bude podlaha. V rozmezí 0.1 až 0.9. Čísla bližší 0.1 generují mapy s malými nebo vůbec žádnými jeskyněmi. Naopak čísla blížící se 0.9 generují mapy s jednou velkou jeskyní přes celou mapu.
- *Rock Threshold* – minimální počet kamenů kolem buňky, aby se proměnila v kámen. V rozmezí 1 až 8. Menší čísla generují mapy s malými nebo žádnými jeskyněmi. Větší čísla generují velké rozsáhle jeskyně.
- *Number of Cellular Automata Iterations* – počet iterací CA.



Obrázek 9: Hlavní stránka webové aplikace s formulářem pro nastavení generace mapy.



Obrázek 10: Stránka s SVG obrázkem vygenerované mapy.

Unikátní možnosti při volbě *Ruined/Occupied Structure* jsou následující:

- *Minimal Room Width* – minimální šířka výsledných místností na mapě (počet v buňkách).
- *Minimal Room Height* – minimální šířka výsledných místností na mapě (počet v buňkách).

Vygenerování mapy se spouští odesláním formuláře tlačítkem *Generate Dungeon*, který uživatele přesměruje na stránku, kde se zobrazí SVG obrázek vygenerované mapy společně s jejím popisem. Pod obrázkem mapy je tlačítko *Download*, které po kliknutí stáhne do počítače mapu společně s jejím popisem. Stránka s SVG obrázkem vytvořené mapy je na obrázku 10.

6.2 Přidávání nepřátel a pokladů

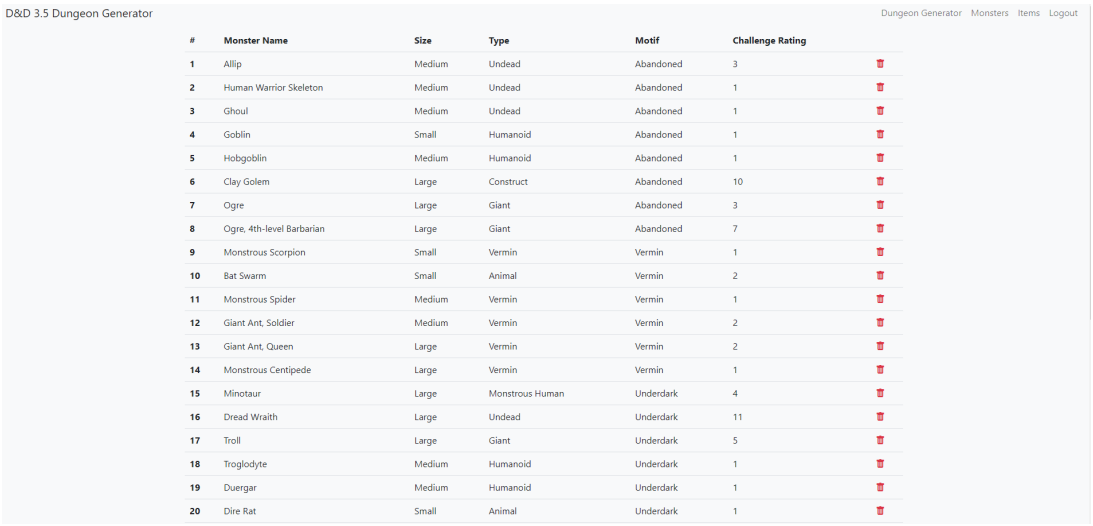
V navigaci webové aplikace jsou odkazy na sekce *Monsters* a *Items*, které po kliknutí zobrazují stránku se seznamem všech nepřátel nebo pokladů v databázi. Pokud je uživatel přihlášen, zobrazí se pod seznamem formulář umožňující přidání nového nepřítele nebo pokladu do databáze. Přidání probíhá posláním formuláře tlačítkem *Add Monster* a *Add Item*. Zároveň se u každé položky v seznamu zobrazí ikonka pro odstranění této položky. Ukázka seznamu nepřátel je na obrázku 11. Z důvodu, že se práce nezabývá řešením registračního systému je umožněno přihlášení pouze pro účet admina nebo jiný účet vytvořený vnitřně ve webové aplikaci. Přihlášení je umožněno v sekci *Login*, která zobrazuje jednoduchý formulář pro přihlášení. Pokud je uživatel přihlášen, je na stejném místě namísto *Login* zobrazen odkaz na *Logout*, který zajišťuje odhlášení uživatele.

Formulář pro přidání nepřátel (Obrázek 12) obsahuje:

- *Monster Name* – pojmenování nepřítele/monstra.
- *Monster Type* – typ nepřítele, výběr ze základních typů pro DnD 3.5.
- *Monster Size* – velikost nepřítele.
- *Monster Motif* – motiv nepřítele.
- *Challenge rating* – hodnocení výzvy nepřítele.

Formulář pro přidání pokladu obsahuje:

- *Item Name* – pojmenování pokladu.
- *Item Type* – typ pokladu, výběr ze základních typů pro DnD 3.5.
- *Item Weight* – váha pokladu.
- *Item Price* – cena pokladu.



Obrázek 11: Seznam všech nepřátel v databázi.

Obrázek 12: Formulář pro přidání nepřátel do databáze.

Závěr

Výsledkem práce je jednoduchá webová aplikace umožňující vytváření map lokací do role-playing hry Dungeons & Dragons verze 3.5. Webová aplikace vytváří mapy podle uživatelem zadaných parametrů na základě dvou metod procedurální generace. Vybrané metody jsou celulární automat a binární rozdělování prostoru. Díky vlastnostem procedurální generace je aplikace schopná generovat nespočet odlišných map.

Pro tvorbu základních typů map pro DnD jsou dvě použité metody dostačující, ale v rámci rozšíření by bylo rozumné implementovat jiné metody procedurální generace nebo nynější metody rozšířit o lepší možnost kontroly procesu generace. Dalším vhodným rozšířením by byl úplný registrační systém umožňující registraci a následné přihlášení uživatelů. Takto registrovaný uživatel by poté mohl mít vlastní databázi nepřátel a pokladů, které by mohl rozšiřovat, a ty by se následně zobrazili v generovaných popisech map.

Conclusions

The result is a simple web application that allows the creation of maps for the role-playing game Dungeons & Dragons version 3.5. The web application generates maps based on user-provided parameters using two methods of procedural generation: cellular automaton and binary space partitioning. Thanks to the properties of procedural generation, the application is capable of generating countless unique maps.

For creating basic types of maps for DnD, the two used methods are sufficient, but for future expansion, it would be reasonable to implement other methods of procedural generation or enhance the current methods to provide more control over the generation process. Another suitable extension would be a complete registration system that allows users to register and log in. A registered user could then have their own database of enemies and treasures, which they could expand, and these would be displayed in the generated map descriptions.

A Obsah elektronických dat

`text/`
: Adresář s textem práce ve formátu PDF, vytvořený s použitím závazného stylu KI PřF UP v Olomouci pro závěrečné práce, včetně všech příloh, a všechny soubory potřebné pro bezproblémové vytvoření PDF dokumentu textu, tj. zdrojový text textu a příloh, vložené obrázky, apod.

`README.txt`
: Textový soubor s informacemi o postupu spuštění software vytvořeného v rámci práce.

`src/`
: Adresáře a soubory vytvořeny v rámci práce obsahující vše potřebné pro bezproblémové spuštění Flask aplikace na „čisté“ instalaci operačního systému.

Seznam zkratek

BSP Binární rozdělování prostoru

CA Celulární automat

DnD Dungeons & Dragons

EA Evoluční algoritmus

Literatura

[1] Togelius, Julian; Noor, Shaker; Nelson, Mark J. Introduction. In. *Procedural Content Generation in Games*. Cambridge (Mass): Springer Cham, 2016, s. 1–15. ISBN 978-3-319-82643-1.

[2] Gearbox. *Borderlands*. Dostupný také z: ⟨https://borderlands.com/en-US/⟩.

[3] Wikipedia contributors. *Rogue (video game) — Wikipedia, The Free Encyclopedia*. Dostupný také z: ⟨https://en.wikipedia.org/w/index.php?title=Rogue_(video_game)&oldid=1154045795⟩.

[4] Blizzard. *Diablo*. Dostupný také z: ⟨https://www.blizzard.com/en-us/⟩.

[5] *No Man's Sky*. Dostupný také z: ⟨https://www.nomanssky.com/⟩.

[6] Linden, Roland van der; Lopes, Ricardo; Bidarra, Rafael. Procedural Generation of Dungeons. *IEEE Transactions on Computational Intelligence and AI in Games*. 2014, vol. 6, no. 1, s. 78–89. Dostupný také z: ⟨https://www.researchgate.net/publication/260800341_Procedural_Generation_of_Dungeons⟩.

[7] Noor, Shaker; Togelius, Julian; Nelson, Mark J. *Procedural Content Generation in Games*. First. Cambridge (Mass): Springer Cham, 2016. xvi, 237 s. Dostupný také z: ⟨http://dx.doi.org/https://doi.org/10.1007/978-3-319-42716-4⟩. ISBN 978-3-319-82643-1.

[8] Noor, Shaker; Togelius, Julian; Nelson, Mark J. Cellular automata. In. *Procedural Content Generation in Games*. Cambridge (Mass): Springer Cham, 2016, s. 42–44. ISBN 978-3-319-82643-1.

[9] Noor, Shaker; Togelius, Julian; Nelson, Mark J. Space partitioning for dungeon generation. In. *Procedural Content Generation in Games*. Cambridge (Mass): Springer Cham, 2016, s. 33–38. ISBN 978-3-319-82643-1.

[10] Noor, Shaker; Togelius, Julian; Nelson, Mark J. Agent-based dungeon growing. In. *Procedural Content Generation in Games*. Cambridge (Mass): Springer Cham, 2016, s. 38–41. ISBN 978-3-319-82643-1.

[11] Noor, Shaker; Togelius, Julian; Nelson, Mark J. Evolutionary search algorithms. In. *Procedural Content Generation in Games*. Cambridge (Mass): Springer Cham, 2016, s. 18–20. ISBN 978-3-319-82643-1.

[12] Noor, Shaker; Togelius, Julian; Nelson, Mark J. Grammar-based dungeon generation. In. *Procedural Content Generation in Games*. Cambridge (Mass): Springer Cham, 2016, s. 45–47. ISBN 978-3-319-82643-1.

[13] *Dungeons & Dragons*. 2022. Dostupný také z: ⟨https://cs.wikipedia.org/wiki/Dungeons_%26_Dragons⟩.

[14] InsideCode. *Flood fill algorithm - Inside code*. 2021. Dostupný také z: ⟨https://www.youtube.com/watch?v=VuiXOc81UDM&ab_channel=Insidecode⟩.

[15] Cook, Monte; Collins, Andy. *Dungeons & dragons: Dungeon master's guide, core rulebook II, V.3.5*. 2012.

[16] Čápka, David. *Úvod do Pythonu*. 2015. Dostupný také z: ⟨https://www.itnetwork.cz/python/zaklady/python-tutorial-uvod-do-pythonu-a-zakladni-matematicke-operace⟩.

[17] Čápka, David. *Úvod do frameworku Flask a webových aplikací v Pythonu*. 2018. Dostupný také z: ⟨https://www.itnetwork.cz/python/flask/uvod-do-frameworku-flask-a-webovych-aplikaci-v-pythonu⟩.

[18] Čápka, David. *Úvod do CSS frameworku Bootstrap*. 2017. Dostupný také z: ⟨https://www.itnetwork.cz/html-css/bootstrap/kurz/uvod-do-css-frameworku-bootstrap⟩.

[19] Viana, Breno M. F.; Santos, Selan R. dos. A Survey of Procedural Dungeon Generation. In. *2019 18th Brazilian Symposium on Computer Games and Digital Entertainment (SBGames)*. 2019, s. 29–38. Dostupný také z: ⟨https://www.researchgate.net/publication/338940959_A_Survey_of_Procedural_Dungeon_Generation⟩.