**Univerzita Hradec Králové**

**Fakulta informatiky a managementu**

BAKALÁŘSKÁ PRÁCE

**2017 Adam Ouhrabka**

**Univerzita Hradec Králové**

**Fakulta informatiky a managementu**

**Katedra Informatiky**

Strojové učení v počítačové grafice

Autor: Adam Ouhrabka

Aplikovaná informatika

Vedoucí práce: Ing. Bruno Ježek, PhD.

Hradec Králové XXX 2018

Prohlášení:

Prohlašuji, že jsem bakalářskou práci zpracoval samostatně a s použitím uvedené literatury.

V Hradci Králové dne

Anotace

Annotation

Obsah

[1 Úvod 1](#_Toc3812541)

[2 Strojové učení v obecných rysech 3](#_Toc3812542)

[2.1 Rozlišení diskriminativních a generativních modelů 4](#_Toc3812543)

[2.2 Metody strojového učení s omezením na neuronové sítě 5](#_Toc3812544)

[2.2.1 Hluboké dopředné neuronové sítě 5](#_Toc3812545)

[2.2.2 Učení neuronových sítí 8](#_Toc3812546)

[2.2.3 Konvoluční neuronové sítě 9](#_Toc3812547)

[2.2.4 Rekurentní neuronové sítě 11](#_Toc3812548)

[2.2.5 General adversarial networks 11](#_Toc3812549)

[3 Procedurální generování 13](#_Toc3812550)

[3.1 Klasifikace PCG algoritmů 14](#_Toc3812551)

[3.2 Tradiční metody procedurálního generování 14](#_Toc3812552)

[3.3 Motivace, účel a vize používání PCG 15](#_Toc3812553)

[4 Procedurální generování prostřednictvím strojového učení 17](#_Toc3812554)

[4.1 Současné experimenty PCGML 18](#_Toc3812555)

[5 Technologie pro implementaci algoritmů strojového učení 21](#_Toc3812556)

[5.1 Python 21](#_Toc3812557)

[5.2 NumPy 21](#_Toc3812558)

[5.3 TensorFlow 22](#_Toc3812559)

[5.4 Keras 23](#_Toc3812560)

[6 Demonstrace metod strojového učení na příkladech generování 24](#_Toc3812561)

[6.1 Příklad hanojských věží 24](#_Toc3812562)

[6.2 Užité technologie 24](#_Toc3812563)

[6.3 Definice cíle a východisek 24](#_Toc3812564)

[6.4 Popis užitého algoritmu 26](#_Toc3812565)

[6.4.1 Charakteristika vstupních dat 26](#_Toc3812566)

[6.4.2 Předzpracování dat 27](#_Toc3812567)

[6.4.3 Natrénování modelů 29](#_Toc3812568)

[6.5 Testování hypotetických výstupů algoritmu 30](#_Toc3812569)

[6.5.1 Výčet sledovaných vlastností 30](#_Toc3812570)

[6.5.2 Jednotkové testování 31](#_Toc3812571)

[6.5.3 Integrační testování 34](#_Toc3812572)

[6.6 Návrhy pro zlepšení algoritmu 34](#_Toc3812573)

[7 Závěr 36](#_Toc3812574)

Seznam zkratek a použitých symbolů

PCG Procedurální generování obsahu

PCGML Procedurální generování obsahu s pomocí strojového učení

GAN General adversarial neural networsk

CNN Konvoluční neuronové sítě

RNN Rekurentní neuronové sítě

EDPCG Procedurální generování obsahu na základě uživatelského chování

# Úvod

Strojové učení a počítačová grafika jsou oblasti, které v posledních dvou dekádách zaznamenávají v rámci technických oborů rapidní vývoj. Co se týká strojového učení, lze identifikovat tři příčiny tohoto progresu. Prvně je to invence sofistikovaných modelů a architektur. Za druhé produkce a existence obřích datových sad pro účely trénování a testování. A konečně existence softwarové a hardwarové podpory, která umožňuje relativně jednoduše implementovat komplexní modely   
a logiku (Abadi et al. nedatováno). Mezi obory strojového učení a počítačové grafiky existuje podstatný průnik. Jedná se například o techniky počítačového vidění   
a zpracování obrazu. Jádrem teoretické části textu je představení tohoto průniku   
a specifické podoblasti, jíž lze označit jako „Generování grafického obsahu s pomocí metod strojového učení“ - v literatuře označovanou zkráceně jako PCGML.

Motivací k sestavení tohoto textu byl zájem o počítačovou kreativitu v oblasti grafiky a snaha o poznání metod strojového učení, primárně neuronových sítí.

Úkoly a cíle bakalářské práce jsou definovány následovně: Za prvé bude v obecných rysech představena oblast strojového učení s explikací metod, které jsou využívány v praktické části. Výčet aplikací založených na těchto metodách (zejména na neuronových sítích) je pak striktně omezen na oblast počítačové grafiky.

Druhým cílem je vytvořit přehled postupů a metod, které jsou používány či diskutovány v souvislosti s procedurálním generováním grafického obsahu. Nejprve je tato oblast představena obecně a poté je text zaměřen na techniky procedurálního generování za pomocí strojového učení.

Třetím cílem je prozkoumat dostupné technologie a prostředí, jež jsou vhodné pro implementaci metod strojového učení a jsou využívány v praktické části bakalářské práce.

V rámci praktické části je pak představen návrh a implementace metody, která za využití jednoduchých architektur neuronových sítí řeší procedurální generování distribuci objektů v dvojrozměrné scéně na základě naučených dat. Tato metoda je otestována a vyplynulé výsledky jsou zhodnoceny v závěru, což je posledním zadaným cílem této práce.

# Strojové učení v obecných rysech

Z formálních definic strojového učení je vybráno tradiční znění Toma Mitchella, který definuje strojové učení následujícím způsobem. *„Stroj (počítačový program) se učí plnit třídu zadání T ze zkušenost E a s úspěšností P v tom případě, když se úspěšnost plnění zadaných úkolů T zlepšuje s využitím znalostí nabytých ze* zkušenosti E.“ (Mitchell 1997a). K základním praktickým úlohám, na něž jsou algoritmy strojového učení aplikovány patří klasifikace, predikce, expertní systémy, popřípadě obecná kontrola agenta konkrétního autonomního systému.

Obecně je cílem algoritmů strojového učení aproximace neznámé komplexní funkce (Goodfellow et al. 2016). Omezíme-li se na oblast učení s učitelem, dosahuje se této aproximace s menší nebo větší úspěšností za pomocí předkládání vstupních a výstupních vektorů hledané funkce. Celý aparát tohoto optimizačního procesu je často parametrizovatelný takzvanými hyperparametry, přičemž celková úspěšnost nalezení příslušné funkce je na těchto parametrech závislá (Jordan a Mitchell 2015).

Jednoduché funkce lze v zásadě aproximovat i jinými čistě programově-algoritmickými metodami, to je nicméně často neefektivní, popřípadě   
u komplexnějších funkcí vyloženě nereálné. Následuje výčet hlavních argumentů pro upřednostňování algoritmů strojového učení, namísto implementace jiného procedurálně či obecně programově laděného systému.

Mitchellem definovaná zkušenost *E* je v oblasti strojového učení reprezentována daty. Pro trénovací sadu je důležité to, jaké formy nabývá a v jakém pořadí vstupují jednotlivá data do algoritmu. Především je pak podstatné „*…jak dobře data reprezentují distribuci příkladů, přes které musí být výsledný systém evaluován úspěšností P.“* (Mitchell 1997b). V takto definované trénovací sadě, lze pak s pomocí vhodných algoritmů strojového učení hledat obecné vztahy a korelace, které by tradiční analýze zůstaly skryté. To znamená, že by se jen těžko hledala přiměřená algoritmizovatelná logika. Tak zní první argument. Vzhledem k záměru této práce lze dodat, že tyto skryté vztahy mohou být později užity ke generování nových vzorků při zachování obecné charakteristiky a distribuce trénovací sady.

Mezi další důvody pro preferenci algoritmů strojového učení patří dle Nilssona enormní velikost datové sady, která znemožňuje syntézu do tradičního programu. Dalším argumentem je dynamická proměnlivost dat, na kterou musí být systém schopný v ideálním případě reagovat. A konečně jsou některé úlohy definovatelné pouze pomocí předkládání četných příkladů (Nilsson 1998). Většinu z těchto jedinečných náležitostí lze demonstrovat na současném projektu Alterego, který mapuje neuromuskulární signály kolem úst na omezený slovník pojmů s úspěšností 92% (Kapur et al. 2018). Lze si jen stěží představit, že by tato funkce byla nalezena a prováděna programovou cestou.

## Rozlišení diskriminativních a generativních modelů

Klasicky se algoritmy strojového učení dají odlišit na „učení s učitelem“ a „učení bez učitele“. Nicméně vzhledem ke skutečnosti, že se v průběhu práce pracuje výlučně se  supervised algoritmy, bude užitečnější poukázat na jinou klasifikaci totiž na rozdíl diskriminativních a generativních modelů. Na tento rozdíl mezi nimi lze dobře poukázat v aplikaci na klasifikaci, kde jde obvykle o rozdělení datové distribuce do několika oblastí. Zatímco u diskriminativních modelů jde o namapování vstupů na konkrétní výstup, kterým může být konkrétní třída či skalární hodnota, generativní přístup modeluje pravděpodobnostní relace mezi proměnnými daného modelu (Jebara 1996). Pro N proměnných lze v těchto modelech nalézt úplnou sdruženou pravděpodobnost ve formě p(x1, ... ,xn). Existuje-li vyjádření takové distribuce, lze na základě této formule odvozovat další hodnoty proměnných pomocí bayesovských pravidel. Tomuto procesu se říká inference. Ve vztahu ke klasifikaci je to pak především pravděpodobnost p(y|x), kde y je třída objektu vzhledem k rozdělení x (Ng a Jordan nedatováno). Generativní modely existují většinou ve formě grafických modelů, mezi které patří primárně bayesovské sítě se směrovými relacemi mezi proměnnými a Markovovy modely, popřípadě Markovovy nahodilostní pole (Jebara 2004). Na příkladu klasifikace rastrových obrázků jednociferných čísel demonstruje Revow zásadní praktický důsledek při použití generativních modelů, totiž ten, že najdeme-li generativní model pro daný systém, našli jsme i model, který je schopen generovat nové vzorky (Michael Revow et al. 1996).

Diskriminativní modely se v jádru svého fungovaní snaží aproximovat ideální hranici, která žádoucím způsobem odděluje vícedimenzionální datovou distribuci. Příklad takového oddělení je znázorněn na Obr. 1. Mezi diskriminativní algoritmy patří typicky neuronové sítě, support vector machines, lineární regresní algoritmy a další (Jebara 1996). Primární užití diskriminativních algoritmů lze vidět v klasifikaci popřípadě regresi.



Obr. 1: Demonstrace typického vymezení hranice při klasifikování dat. Převzato z (Goodfellow et al. 2016).

## Metody strojového učení s omezením na neuronové sítě

V následujících kapitolách budou postupně představeny konkrétní metody strojového učení s omezením na formy neuronových sítí. Nejprve bude popsána standardní architekura umělé neuronové sítě a poté budou popsány tři další modely totiž General Adversarial Networks, Konvoluční neuronové sítě a Rekurentní neuronové sítě. Všechny tyto modely stojí na bázi standardních neuronových sítí a jsou předmětem aktuálního výzkumu a aplikací v posledních letech. Rovněž budou ukázány jejich aplikace v oboru počítačové grafiky.

### Hluboké dopředné neuronové sítě

Hluboké neuronové sítě jsou jednou z často používaných metod strojového učení. Je to aparát, který má charakter funkce a lze jej popsat jako *masivně paralelní procesor*. Mnohačetným předkládáním *m* vzorů do neuronové sítě je možné adaptovat funkční parametry jednotlivých neuronů tak, že vznikne obecně použitý model pro klasifikaci či predikci dat na základě nového vektoru vstupních hodnot. Tento proces adaptace je chápán jako učení neuronové sítě. Implementace tohoto učení stojí na minimalizaci chybové funkce , která agreguje rozdíl výstupních hodnot vyprodukovaných neuronovou sítí s datově předepsanými hodnotami pro konkrétní vzor (Vladimír Olej a Hájek Petr 2010). Přirozeně se během vývoje oboru strojového učení začalo pracovat i s jinými chybovými funkcemi.

Rov. 1: Příklad předpisuchybové funkce

Přestože existují různé varianty a topologie neuronových sítí, základní principy korelují (Tučková 2003).

Stavebním elementem neuronových sítí je neuron, jež si lze představit jako funkci na jejímž vstupu je vektor hodnot, které vyšly z předcházející vrstvy neuronové sítě (NS) a na jejím výstupu skalární hodnota, jež figuruje jako vstup pro další vrstvy popřípadě jako výstup neuronové sítě. Vstupní vektor je nejprve agregován do skalární hodnoty, která je následně vstupem do aktivační funkce . Ta transformuje vstupní hodnotu do výstupní hodnoty celého neuronu. Parametr přidává prahovou hodnotu neuronu (bias), která přispívá k aktivizaci neuronu.

Rov. 2: Agregace vstupnich hodnot neuronu

Konkrétních aktivačních funkcí existuje celá řada a mají zásadní charakter na fungování NS. Aktivační funkce mohou mít za důsledek lineární i nelineární charakter klasifikačních oblastí vyprodukovaných neuronovou sítí. Příkladem nelineární aktivační funkce jako je například sigmoidální funkce (Vladimír Olej a Hájek Petr 2010).

Rov. 3: Výstupní hodnota neuronu při užití funkce sigmoid

Aktuálním trendem v oblasti hlubokého učení je však v rámci vnitřních vrstev neuronových sítí jiná funkce, která se označuje jako „rectified linear unit“ zkráceně ReLU, kde (Goodfellow et al. 2016). Funkce neuronu v kontextu strojového učení napodobuje neuron bilogický. Dle Glorota je to však teprve užití aktivační funkce ReLU, která tuto podobnost ve velké míře dovršuje. Především diskutuje princip „řídkosti aktivace“, při kterém v rámci šíření signálu zůstane mnoho neuronových jednotek na svém výstupu nulových. Je to podstatný a pozitivní důsledek fungování NN právě při užití ReLU. (Glorot et al. 2011). Aplikace této aktivační funkce má rovněž za důsledek rychlejší trénování (Bhandare et al. 2016).



Obr. 2: Graf funkce ReLU. Převzato z (Goodfellow et al. 2016).

Podstatnou komponentou neuronové sítě je aktivační funkce poslední vrstvy. Zde záleží na tom, jaký charakter výstupu je žádoucí. Obecně se však užívá již prezentovaná funkce sigmoid a její odvozenina funkce softmax . Výstup neuronové sítě klasifikuje vstup do konkrétní proměnné. Je-li tato proměnná diskrétní, to znamená, že představuje například příslušnost vstupních dat do *n* tříd, měla by výstupní vrstva vyjadřovat vektor pravděpodobnosti příslušnosti objektu do těchto tříd. Pro tento konkrétní účel se využívá právě funkce softmax.

Rov. 4 formální vyjádření funkce softmax

Důsledkem užití této funkce je, že na výstupu neuronové sítě bude vektor *v1*, který bude vyjadřovat pravděpodobnosti pro jednotlivé třídy určené pořadím v rámci vektoru a díky „normalizační“ složce v čitateli bude složkový součet roven jedné.

### Učení neuronových sítí

Podobně jako existuje variabilita chybových funkcí, existují i různé optimalizační (učící) algoritmy. Obecně však jsou však na začátku učení iniciovány všechny váhy v systému jako náhodné nenulové hodnoty. K následné adaptaci vah ve standardních modelech hlubokého učení s učitelem se pak používá nějaká alternativa optimalizačního algoritmu, například Stochastic gradient descent ve spojení   
s algoritmem zpětného šíření chyby Backpropagation algorithm. Proces spočívá   
v minimalizaci chybové funkce ve směru gradientu tj. směrem k lokálnímu minimu. To se děje v každé iteraci za pomocí úpravy vah napojených na jednotlivé neurony   
v jednotlivých vrstvách sítě. Výsledným vztahem pro korekci všech vah v NS je rovnice, kde je keocifient posunu v rámci gradientu, je směr gradientu a je původní hodota příslušné váhy.

Rov. 5 korekce vah v BPA

Ve chvíli, kdy je v rámci jedné iterace průchodu dat neuronovou sítí spočítána hodnota chybové funkce výstupní vrstvy, je nutné zpětně pro všechny neurony skrytých vrstev vypočítat nakolik váhy, které z nich vedou přispívají ke konkrétní hodnotě celkové chyby výstupní vrstvy.

Rov. 6 výpočet chyby neuronu

Při této operaci

Adam - algoritmus

### Konvoluční neuronové sítě

Mezi nejúspěšnější architektury aplikované i v počítačové grafice posledních několika dekád patří mimo GAN také hluboké konvoluční neuronové sítě (CNN), které se v rámci soutěže ILSVRC[[1]](#footnote-1) umísťují na předních příčkách. (Goodfellow et al. 2016).

CNN jsou propojením dvou silných konceptů tj. dopředné neuronové sítě a principu konvoluce. Přestože aplikace z poslední doby užívají CNN i například pro rozpoznání řeči či práci s jazykem. Používají se jak pro obrazová, tak pro časová data. Formálně jsou definovány jako *„...sítě, které namísto maticového násobení užívají alespoň v jedné vrstvě konvoluci.“*. (Goodfellow et al. 2016)

Máme-li dvoudimeznionální pole dat aplikujeme konvoluční operací následovně:

Rov. 7 operace konvoluce pro pole I (**dim**= a jádro K (**dim**=

Z formálního vyjádření vyplývá, že se jedná o vzájemné pronásobení matice vzoru *I* a matice jádra *K*, přičemž výsledek tohoto pronásobení je agregován do skalární hodnoty a uložen do výsledné matice *S*. Základní motivací pro provedení této operace je identifikace specifických znaků na vstupních datech. Rovněž je díky využití konvoluce do určité míry zajištěno, že varianty dat jedné třídy budou do neuronové sítě vstupovat jako invariantní co do velikosti, občasných posunů a jiných distorzí (LeCun et al. 1998). Důvodem této skutečnosti je fakt, že se ve fázi učení naleznou taková jádra (tj. jejich váhy), která ve spojení s příslušnými vstupy vyselektují typické znaky, podle kterých bude možné jednoznačně určit třídu. Kromě těchto jader figurují ve výsledné klasifikaci ještě váhy klasických vrstev, které jsou napojeny na výstup konvoluční části modelu.



Obr. 3: Konvoluční architekture LaNet-5

Na obrázku výše je zobrazena architektura konvoluční sítě, která byla svými autory nazvána jako LeNet-5 (LeCun et al. 1998). Kromě klasických neuronových vrstev, vystupují v těchto architekturách ještě další jednotky v podobě konvolučních a „agregačních“ (subsampling) vrstev.

Vrstva, které se říká konvoluční, aplikuje operaci konvoluce na předchozí vrstvu modelu. Efektivita těchto architektur spočívá mimo jiné v tom, že se konvuluje hned několikrát. To znamená, že jednotlivá jádra vyústí v procesu učení do navzájem různých konfigurací, což má za důsledek „selekci“ odlišných znaků. Počet konvolučních jader figuruje jako hyperparametr. S řetězovitou aplikací konvoluce se tak uvnitř modelu navyšuje počet jaderných konfigurací a jejich konvolvovaných produktů (feature maps). Zároveň se snižuje velikost matice těchto produktů, což je přirozený důsledek konvoluce kde se výsledná velikost , kde = velikost vstupu, = velikost jádra, = odsazení, = velikost kroku.

V  posloupnosti vrstev CNN následuje obvykle za konvoluční vrstvou transformace, která na její výstup aplikuje aktivační funkci, čímž je dosaženo nelinearity výstupu. Třetí v tomto typickém CNN tripletu je takzvaná subsampling či pooling vrstva. Funkce této vrstvy se může lišit. Obecně jde ale o další redukci velikosti konvolučního výstupu na základě okolí. Hodnota výstupu tak může být například maximální numerická hodnota prvku uvnitř okolí či průměrná hodnota prvků. Účelem této operace je, aby se konvolvovaná reprezentace dat stala ještě více invariantní (Goodfellow et al. 2016). V některých případech je za tímto tripletem přítomna ještě tzv. dropout vrstva, která při šíření dopředného „signálu“ nahodile deaktivuje některé neurony. Dle Krizhevského zamezuje tato technika jevu takzvaného přeučení a obecně přispívá k přesnosti modelu (Krizhevsky et al. 2012).

Konvoluční sítě mají své aplikace především v počítačové grafice, konkrétně počítačovém vidění. Prezentovaný model LaNet-5 byl svými autory v roce 1998 použit ke strojovému čtení psaného textu. Chyba se v rámci testovací sady pohybovala v řádech desetin procent (LeCun et al. 1998).

Současné aplikace shrnuje ve svém článku Bhandare. Většina aplikací se týká dvourozměrných rastrových obrázků o třech barevných kanálech. CNN byly úspěšně použity pro rozpoznání tváře a klasifikaci objektů obecně. Z implementovaných řešení lze poukázat na klasifikátor ImageNET, jehož dataset obsahuje patnáct milionů obrázků patřících do 22 000 kategorií. Přesnost tohoto modelu byla 62.5%. (Krizhevsky et al. 2012).

Dalším odvětvím aplikace je takzvaný „scene labeling“, kde jde o identifikaci   
a zaměření objektů ve scéně. Průlomem v této oblasti byla v rámci hlubokého učení architektura CNN navržená Farabetem. Fully convolutional neural networks.

Vývoj CNN nadále spočívá v zacházení s hyperparametry a ladění architektur modelu. Na základě Bhandarova souhrného článku lze konstatovat, že existuje mnoho variant CNN, plnící cíle klasifikace, segmentace, ale i další. (Bhandare et al. 2016).

### Rekurentní neuronové sítě

When feedforward neural networks are extended to include feedback connections, they are called recurrent neural networks, presented in chapter (d183)

### General adversarial networks

Jsou specifické architektury neuronových sítí, které patří do rodiny generativních modelů a byly vynalezeny Ianem Goodfellowem v roce 2014. Sám Goodfellow obecně definuje generativní modely jako *„...modely, které se na základě tréninkové distribuce dat tvořených množinou pdata naučí odhad takové distribuce reprezentovat“*. V konkrétních aplikacích mohou takové modely buď plnit pouze onu reprezentativní funkci a nebo mohou na základě vnitřní struktury modelu generovat nové originální vzorky (Goodfellow et al. 2014). Základním principem fungováním je takzvaný princip „maximální pravděpodobnosti“ to znamená „*...nalezení takových parametrů pro model, aby maximalizovaly pravděpodobnost trénikových dat*“.

Goodfellow navrhl model skládající se ze dvou komponent. První z nich je diskriminátor, jehož funkcí je binární klasifikace generátorem nagenerovaných vzorků na 1 nebo 0. Druhou komponentou je generátor, jehož úkolem je vytvořit vzorky, které napodobují tréniková data. Generátor v každé iteraci učení ladí své váhy tak, aby transformoval prvotně iniciovaný šum na přijatelnější vzorky, které jsou diskriminátorem znovu klasifikovány a vzniklá chyba se projeví v příští iteraci generování. Obě komponenty tak stojí proti sobě a finální naučení redistribuce plyne ze vzniklé tenze mezi nimi (Goodfellow 2016).

Většina aplikovaných GAN architektur v sobě zakomponovává konvoluční mechaniku. Hovří se pak o tzv. DCGAN (Goodfellow 2016).

Tyto modely byly kromě generování nových vzorků použity například na obarvení obrázků v infračerveném spektru (Suarez et al. 2017). A obecně  transformaci obrázků do obrázku určité stylizace například transformace fotografie denní scény na scénu noční (Isola et al. 2016).

# Procedurální generování

V rámci počítačové grafiky existují v podstatě dva způsoby pro definici grafického obsahu modelované scény. Ať už uvažujeme dvojrozměrné, popřípadě trojrozměrné zobrazení scény, tak vlastnosti zobrazovaných objektů této scény musí být vždy předem zaneseny v příslušných datových strukturách, které pak figurují v rámci vizualizačního řetězce. Zachycení geometrie a umístění objektu v souřadnicích scény může být definováno buď staticky, to znamená „ručním“ modelováním objektů v příslušném softwaru anebo algoritmicky, kdy se vzhled a transformace objektů odvíjejí od sekvence příkazů daného algoritmu. Podobné rozdělení se týká například   
i textur a dalších elementů, které scénu spoluvytvářejí. Druhý ze zmíněných přístupu lze označit termínem procedurální generování obsahu (PCG).

Zkratka PCG bývá skloňována především v oblasti počítačových her, kde najdou tyto algoritmy široké uplatnění. Tyto algoritmy jsou však použitelné i v rámci budování parametrizovatelných virtuálních prostředí, simulací, vizualizací, popřípadě v oblasti dekorativního a technického designu. Nutno zmínit, že PCG se ve své definici neomezuje jen na generování grafického obsahu jako takového, nýbrž se tento pojem používá i pro algoritmické generování jakéhokoliv obsahu. Tím je například obsah textový, animovaný, popřípadě hudební. Yannakis zahrnuje do oblasti PCG i generování personalizovaného obsahu, které se uplatňuje například v rozhraních pro e-commerce a uživatelských rozhraních obecně (Yannakakis a Togelius 2011).

Univerzální vlastností většiny algoritmů pro PCG je, že ve svém těle pracují   
s faktorem nahodilosti, který je nezbytný pro zajištění diverzity obsahu. Základní metodou pro naplnění tohoto kritéria je například implementace pseudonáhodných čísel v podobě šumu (Hendrikx et al. 2013). Mezi další podstatné vlastnosti PCG algoritmů patří rychlost, spolehlivost, kontrolovatelnost a uvěřitelnost (Julian et al. 2016). Smith ve svém článku užívá v kontextu PCG výstižný pojem „řízená nahodilost“ (Smith 2015). Nahodilost má v PCG efekt spíše na dílčí části negenerovaného obsahu, než na samotný celek, který musí být uvěřitelný, popřípadě hratelný. Demonstrováno na příkladu: distribuce a geometrická variabilita jednotlivých listů může být náhodná do té míry, do kdy se objekt jeví stále jako objekt dané třídy, v tomto konkrétním případě - jako strom. Termínem „řízená“ je vystihnuto kritérium pro smysluplnost, ze kterého vyplývá další podstatná vlastnost PCG algoritmů, kterou je přítomnost určitých omezení, které v jádře algoritmu zajišťují, že obsah bude funkční a uvěřitelný.

## Klasifikace PCG algoritmů

Algoritmy PCG se dělí na parametrizovatelné a na ty které ke svému fungování žádnou parametrizaci nepotřebují. I minimální parametrizace jako například definice počtu iterací spadá do skupiny parametrizovatelných PCG algoritmů, které nabízejí designérovy kontrolu nad finální podobou generovaného obsahu (Carli et al. 2011).

Další dělení definující PCG algoritmy se týká toho, zdali má generování zpětnou vazbu na uživatelské interakce nebo takovou vazbu nemá, popřípadě zdali se obsah generuje při běhu aplikace (online) nebo slouží pouze jako podpůrný nástroj designu při vývoji (offline) (Togelius et al. 2010).

Jedná-li se o algoritmus, který přímo reaguje na interakci s uživatelem a v rámci běhu aplikace dotváří „personalizovaný obsah“, označuje ho Yannakakis zkratkou EDPCG, kterou lze přeložit jako „Generování obsahu na základě uživatelského chování“ (Yannakakis a Togelius 2011).

## Tradiční metody procedurálního generování

Tradiční PCG algoritmy jsou založeny na poznatcích z mnoha oblastí informatiky. Pro generování grafického obsahu se užívají evoluční algoritmy, Lindenmayerovi systémy, fraktály, a další specifickým způsobem aplikované postupy, které pro každý svůj běh negenerují jiný monotematický obsah. K tomu lze podotknout, že *„Všechny tyto metody mají společné to, že tyto algoritmy, parametry, omezení a cíle, které se podílí na vytváření obsahu jsou v podstatě ručně definované svými tvůrci.“* (Summerville et al. 2017).

Stěžejní proměnou PCG algoritmů je podoba reprezentace obecných genotypů, na základě kterých vzniká vždy originální entita (fenotyp). Takové genotypy mohou reprezentovat jak obecné geometrie objektů, tak například obecné vztahy mezi nimi. Tato biologická paralela vychází z často aplikované metody PCG, kterou jsou genetické algoritmy.

## Motivace, účel a vize používání PCG

Klíčovou součástí moderních grafických aplikací, popřípadě počítačových her jsou namodelované objekty plnící buď kosmetickou či interaktivní funkci. Nevyužije-li se v rámci produkce PCG metod je všechen obsah modelován ručně. Taková praxe výrazně zvyšuje cenu produktu a prodlužuje dobu jeho vývoje. Na druhé straně je možné potenciál PCG použít pouze tam, kde prvek prvek náhody dotváří funkci herního mechanismu nebo vytváří diverzitu, která nenarušuje účel obsahu (nebrání například v postupu či jinak nemate uživatele). Generovaný obsah tedy může být na základě tohoto kritéria rozdělen na funkční a čistě kosmetický (Togelius et al. 2010).

Vize a cíle současného bádání PCG v grafické oblasti definuje ve svém článku Julian Togelius. Jako první vize je nadneseně představena univerzální metoda na vytvoření komplexních virtuálních světů určitého žánru. I v případě, že by se jednalo pouze o statický grafický obsah takového světa dá se společně s Togeliusem konstatovat, že podobný algoritmus je nerealizovatelný i v rámci vzdálené budoucnosti. Komplexita takového úkolu je enormní. Problémů, které stojí v cestě takovému zadání je hned několik a tvoří primární úkoly v rámci výzkumu PCG (Togelius et al. 2013).

Generovaný obsah se potýká s originalitou, nedostatečnou diverzitou   
a kontextuální smysluplností. To vše jsou znaky kreativity v rámci ručního modelování. Klíčová otázka zní, zdali se v případě PCG dá kreativita algoritmizovat a v případě generování prostřednictvím strojového učení stochasticky modelovat i v rámci komplexních celků. Kritéria výpočetní kreativity, která významně korelují s požadavky na PCG definoval ve své práci Pereira. Hlavní znaky spatřuje v uchování modelu znalostí o dané třídě, schopnost redistribuce znalosti do nových spojení, schopnost kriticky zhodnotit produkt (validace), uvážení kontextu, do kterého je tvorba zasazena a užití konvergentních i divergentních postupů při kreaci (Pereira 2008). Všechny tyto podmínky, které Pereira postuluje pro svůj model („Creative general problem solver“) jsou realizovatelné a byly realizovány prostřednictvím generativních modelů strojového učení například za použití GAN architektur neuronových sítí v oblasti rastrových obrázků. Jako příklad může sloužit GAN model pro generování lidských tváří ve vysokém rozlišení (Karras et al. 2018).

Togelius ve svém článku nadále vybízí k syntéze metod, které byly vynalezeny v rámci generování jednotlivých objektů jako například terénu, budov či interiéru, do komplexního parametrizovatelného generátoru. Interakci mezi jednotlivými generátory podtrhuje ve svém článku i Hendrikx (Hendrikx et al. 2013). Výstupy jednotlivých generátorů by tedy měly být vzájemně kompatibilní a ve vztahu, který dodává generovanému celku přidanou hodnotu.

Tradiční metody PCG jsou často aplikovány ad hoc ke konkrétnímu použití. Znovupoužitelnost je realizovatelná pouze principielně a jen stěží se dají stejné algoritmy použít napříč různými aplikacemi. I to je nedostatek na který Togelius poukazuje (Togelius et al. 2013).

Uvážíme-li Togeliusův nárok na komplexní procedurální generátor, je nutné do něj zahrnout i dynamickou stránku scénu, tj. animační. Procedurální animování je další oblastí, v rámci které probíhá aktivní výzkum (Togelius et al. 2013).

Jako obecný trend v rámci PCG lze tedy primárně identifikovat redukci genericity generovaného obsahu a směřování k univerzálním, navzájem propojeným generátorům. K naplnění těchto cílů se výzkumníci v oblasti PCG často uchylují k metodám strojového učení, jež má v návaznosti na současné cíle PCG velký potenciál, neboť umožňuje nalezení modelu složitých datových distribucí. Takovým modelem není obvykle funkce. Vhodnější je najít generativní pravděpodobnostní datové modely, které v rámci svého učení algoritmicky hledají parametry pro , kde jsou vstupní data. Takovými parametry mohou být například proměnné normálního rozdělení. Pro generování nových dat založených na této distribuci lze pak uplatnit vztah , kde Tento konkrétní postup je zde popsán na základě Kingmem uvedené generativní architektury neuronových sítí známé jako variační autoenkodér (Kingma a Welling 2013).

# Procedurální generování prostřednictvím strojového učení

Tradiční PCG přistupuje k tvorbě obsahu s konkrétní referencí ve vztahu k cíli. Tato reference však není součástí samotného algoritmu, ale figuruje pouze jako inspirace toho, kdo algoritmus navrhuje. Znamená to, že primární část návrhu takového algoritmu tvoří hledání pravidel, abstrakcí a omezení, která pomohou aproximovat žádoucí obsah. Toto hledání figuruje i v rámci procedurálního generování prostřednictvím strojového učení, avšak s tím podstatným rozdílem, že toto hledání neprobíhá na straně vývojáře, nýbrž na straně výpočetního stroje. Ten s pomocí obecně laděných algoritmů strojového učení a vhodně zpracovaných vstupních dat vytvoří model, pomocí kterého je možné nagenerovat další variovatelný obsah. Tak ve zkratce definuje PCGML Summerville (Summerville et al. 2017).

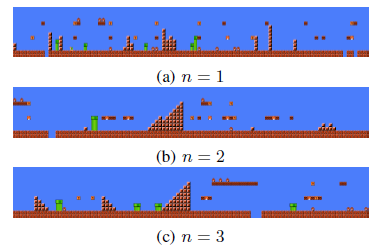
Definice strojového učení od Toma Mitchella uvedená v první kapitole   
je aplikovatelná rovněž pro PCGML, kdy zadáním *T* je generování konkrétní třídy obsahu na základě zkušenosti *E*, která je tvořena množinou ručně modelovaných obsahů (v případě učení s učitelem). Měření úspěšnost *P* může být vzhledem k žádoucí variabilitě výsledků značně komplikované a může v konečné fázi spočívat   
v subjektivním posouzení omezeného množství nagenerovaného obsahu. Příkladem implementace takové metriky jsou dotazníky vyplněné testery, na jejichž základě byly evaluovány hratelnosti procedurálně nagenerovaných levelů a tedy celého algoritmu (Roberts a Chen 2015). Pro posouzení většího vzorku dat je třeba implementace algoritmů, jejichž funkcí je evaluace generovaného obsahu na základě příslušných kritérií.

Ne všechny metody strojového učení, které se jinak výborně hodí pro klasifikaci či predikci, jsou vhodné pro generování nového obsahu. Odpovídá tomu i zmíněné rozdělení metod strojového učení na diskriminativní a generativní modely.   
Z generativních modelů strojového učení vyzdvihuje Summerville jako vhodné zejména, n-gramy, markovy modely, současné generativní modely hlubokého učení (GAN) a specifické architektury rekurentních neuronových sítí (Summerville et al. 2017).

## Současné experimenty PCGML

Následující kapitola prezentuje vybrané užité aplikace a metody v oblasti PCGML. Některé tyto příklady posloužily jako motivace k sepsání této bakalářské práce   
a vytvoření metody užité v její praktické části. V kontextu strojového učení se všechny tyto aplikace musely po svém způsobu vypořádat s problémem malé datové sady   
a s požadavkem na použitelnost nagenerovaných vzorků.

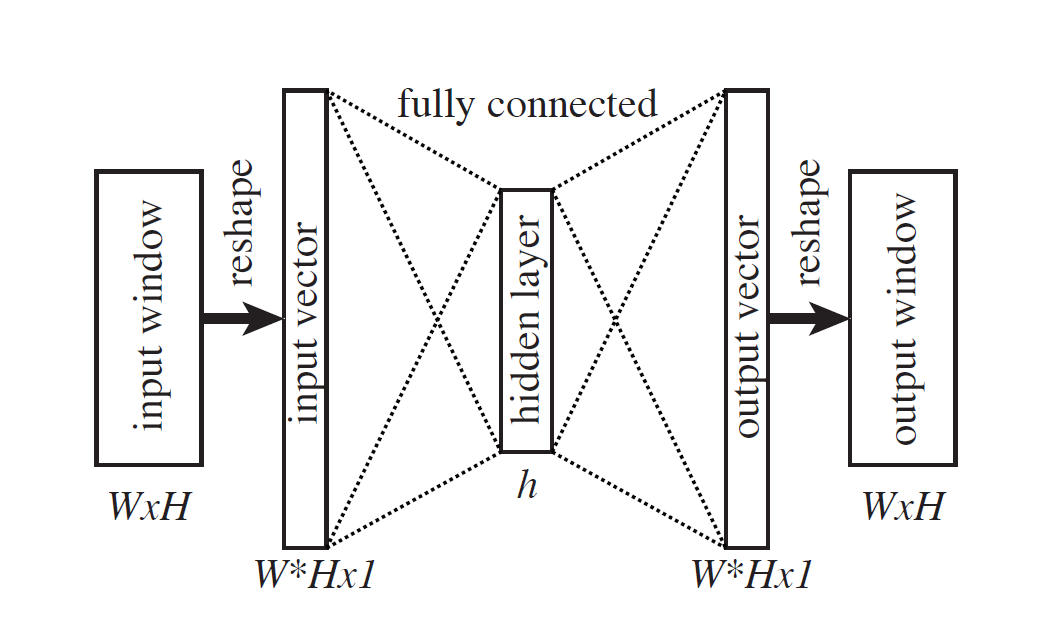
Silným nástrojem pro generování jednoduchého obsahu se ukázala být nejjednodušší varianta markovova řetězce ve spojení s n-gramy. Markovovův řetězec je graf, jehož vrcholy reprezentují určitý stavový prostor. Mezi vrcholy existují ohodnocené hrany, a ty vyjadřují pravděpodobnost přechodu mezi jednotlivými stavy (Gagnius 2017). N-gram je struktura, která vyjadřuje jaké bude následovat v případě, že pole délky obsahuje nějaké konkrétní hodnoty. Technika tohoto typu se dá dobře využít pro generování jednoduchých levelů viz. Obr. 4. V tomto experimentu byla vstupní data nejprve indexována a rozřezána na sloupce. Poté byl na základě n-gramů vytvořen pravděpodobnostní model, který iterativně predikoval další sloupec podmíněně k  posledním sloupcům (Summerville et al. 2017).



Obr. 4: Výsledné nagenerované levely pro jednotlivá n, vstupní data byla rozdělena na sloupce a za pomocí n-gramů transformována do Markovova řetězce. Převzato z (Summerville et al. 2017).

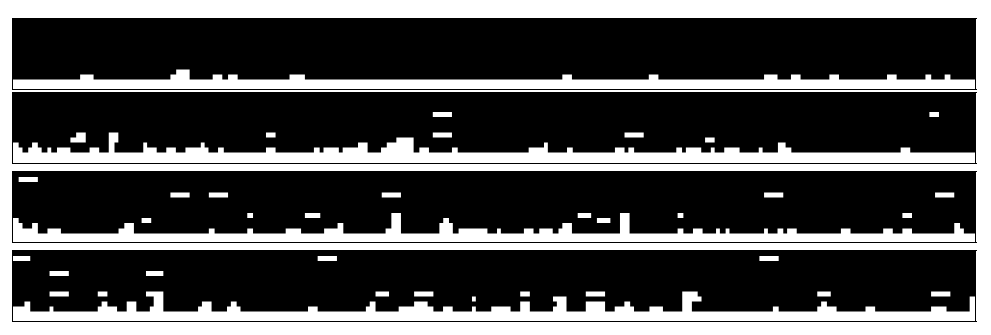
Komplexnější alternativou pro dosažení obdobného cíle je užití hlubokých neuronových sítí, a to konkrétně rekuretních architektur. V tomto přístupu se už generování neodehrává na úrovni sloupců, nýbrž na úrovni jednotlivých políček třinácti kategorií. Kromě těchto políček byla do algoritmu zahrnuta i informace   
o pohybu hráče uvnitř vstupních dat (levelů). Účelem tohoto kroku byla reálná hratelnost negenerovaných úrovní. Generování levelů probíhalo tedy iterativně   
v prostoru od pomyslného k zase zpátky dolů od   
k . To vše s paměťovou stopou na předtím vygenerovaná pole o délce zhruba 200 polí (Summerville a Mateas 2016).

Zakódovaný dataset použitý Summervillem byl východiskem i pro Jainův tým, který však namísto rekurentních neuronových sítí využil architektury klasického autoenkoderu (Jain et al. 2016). Jedná se o neuronovou síť, ve které se počet vstupních neuronů rovná počtu výstupních. Skryté vrstvy jsou nadefinovány tak, aby nejprve došlo ke komprimaci informace a její následné dekomprimaci. Toho je dosaženo postupným snižováním počtu neuronů v jednotlivých vrstvách na přijatelné minimum směrem do středu a jejich následnému zvyšování směrem od středu viz Obr. 5. Tyto sítě byly aplikovány především v oblasti zpracování obrazu (škálování obrazu, odšumění a další) (Goodfellow et al. 2016). Obecně tato síť umožňuje funkčně mapovat transformaci vstupů na výstupy o stejné dimenzi.



Obr. 5: Schéma autoenkóderu. Převzato z (Jain et al. 2016).

Aby Jain dosáhl požadované nahodilosti, zaznamenal distribuci hodnot vnitřní autoenkodérové vrstvy napříč trénovanými daty. Ve fázi generování poté reinterpretoval tuto vrstvu jako vstupní a tuto distribuci varioval za pomocí šumu (Jain et al. 2016). Binárně kódované levely jsou ukázáný na Obr. 6.



Obr. 6: Generování levelů při aplikace náhodného šumu s rozptylem 0,01 ; 0,1 ; 0,3 ; 0,6.   
Převzato z (Jain et al. 2016)

Jako čtvrtý příklad, který lze označit za výsledek v PCGML, je zde uveden Fisherův pokus o syntézu žánrových scén do modelu schopného tyto scény generovat v četných variacích. Konkrétně se jedná o scénu stolu obklopeném objekty. K učení algoritmu byly použito 130 manuálně vymodelovaných prostředí interiéru, ve kterých se dohromady znovupoužívalo 1723 kategorizovaných objektů. Variace objektů bylo dosaženo implementací takzvaných kontextuálních kategorií. Variování tak neprobíhá pouze na základě příslušnosti objektu k určité kategorii, ale je založené na pozičním kontextu objektu. Druhým modelem pro finální generování je model okurence, který je založen na bayesovských sítích, a který v podstatě řeší pravděpodobnost možné přítomnosti objektu vzhledem k již přítomným. V rámci modelu okurence byly rovněž implementována umělá omezení, za účelem zavedení informace o relaci „x je na y“ to znamená „y podpírá x“. Poslední složkou je pozicovací model, který řeší finální umístění a natočení objektu. Za tímto účelem jsou použity „Gaussian mixture models“ (Fisher et al. 2012).



Obr. 7 Výsledky generování a variování scény se stolem za pomocí Fisherova algoritmu.   
Převzato z (Fisher et al. 2012).

V souvislosti s PCGML lze rovněž zmínit experiment Giacomella a jeho týmu, který s využítím GAN vytvořil model pro generování herních levelů. V tomto případě však byla tréniková data o velikosti 1088 vzorků zakódována jako rastrové obrázky (128x128) půdorysů levelů ze hry DOOM. V rámci každého levelu existuje celkem šest obrázků, z nichž každý reprezentuje jinou informaci tj. například o topologii, výškovém profilu či rozmístění předmětů uvnitř levelu. Autoři vytvořili kromě standardního modelu i podmíněný model, který za pomoci sedmi parametrů nagenerované vzorky parametrizoval (Giacomello et al. 2018).

Podobným způsobem byly GAN architektury implementovány za účelem procedurálního generování terénu (Beckham a Pal 2017).

Na základě těchto experimentů lze konstatovat tři obecné rysy algoritmů pro generování logicky uspořádaných prostorů. Za prvé je třeba účelným způsobem předzpracovat vstupní data. Algoritmu musí být obvykle připraveny vhodně zakódované partikulární části žádoucího obsahu. Ve fázi generování jsou pak kromě nahodile logické redistribuce těchto částí, zachovány naučené relace, ve kterých tyto části figurovaly v rámci učení. V prvním příkladu generování herního levelu, byly těmito částmi jednotlivé sloupce a onou naučenou relací byla posloupnost sloupců za sebou. Dále je třeba vhodně variovat tyto relace sledujíce princip PCG „řízené nahodilosti“. Za třetí si lze všimnout, že generování probíhá obvykle sekvenčně, kdy stav aktuálního prostoru vystupuje jako vstup pro další krok algoritmu.

# Technologie pro implementaci algoritmů strojového učení

Většina používaných knihoven pro účely implementace algoritmů strojového učení je dnes navázána na programovací jazyk Python. Jmenovitě to jsou například knihovny PyTorch, Scikit-learn, Tensorflow a v této práci užívaný Keras. Vzhledem k jednostrannému zaměření této práce na neuronové sítě budou postupně ve stručnosti představeny ty technologie, se kterými se pracuje v praktické části a které jsou obecně základem pro aplikační práci s neuronovými sítěmi.

## Python

Python je opensource intepretovaný, multiplatformní jazyk s čistou syntaxí evokující pseudokód. Obecně je dnes tento jazyk používán především ve vědecké   
a technologické komunitě. Důvody pro jeho užívání vystihuje Oliphant: V první řadě je to obrovské množství jak nativních, tak dodatečných knihoven, které řeší komplexní úlohy vyplynulé z vědeckých potřeb. Jmenovitě se jedná například o vizualizační nástroje, knihovny pro statistiku, strojové učení a mnohé další. Za druhé lze jazyk užívat jak procedurálním, tak i objektovým způsobem. A konečně Python umožňuje živou interakci s běžícím kódem a má velice silnou odbornou komunitu uživatelů (Oliphant 2007).

## NumPy

NumPy je dle Oliphanta podstatným důvodem akademické oblíbenosti Pythonu (Oliphant 2007). Je to knihovna, která umožňuje za pomocí vyššího jazyku, kterým Python je, efektivně implementovat numerické operace (především na maticích   
a tensorech obecně). Za touto efektivitou stojí tři složky. V první řadě je to vektorizace veškerých výpočtů, za pomocí operací implementovaných v jazyce C. Za druhé efektivní správa proměnných v paměti a za třetí minimalizace počtu operací (Van Der Walt et al. 2011). V souvislosti se strojovým učením se NumPy využívá při předzpracování dat a je rovněž interní součástí většiny knihoven pro strojové učení.

Základní strukturou (objektem) této knihovny je NumPy pole (ndarray). Primární charakteristikou tohoto pole je, že každý jeho prvek může být nadále vícedimenzionální. Van der Walt konstatuje, že: *„...NumPy pole je pouze příhodný způsob popisu jednoho nebo více bloků paměti za účelem jednoduché manipulace s reprezentovanými čísly.“*. Každá ndarray struktura si drží ukazatel na první byte v poli, datový typ prvků, svůj tvar (shape), počet kroků pro skok na další element (strides)   
a příznaky, zdali je možné s polem manipulovat.

Důležitým parametrem je především onen počet kroků. Van der Walt tak hovoří o „krokovém paměťovém modelu“, který umožňuje intepretaci paměti na více způsobů bez nutnosti kopírovat data. Toho se využívá především při manipulaci s tvarem (shape) multidimenzionálního pole, kdy dojde pouze k přepočítání potřebných kroků. (Van Der Walt et al. 2011).

## TensorFlow

Tensorflow je knihovna pro podporu strojového učení vyvíjená společností Google. Primárně funguje v Pythonovském kontextu, ale existují její alternativy, popřípadě API pro jiné jazyky. Umožňuje poměrně komplikovanou distribuci výpočtů na externí výpočetní jednotky, popřípadě jejich paralelizaci či naopak synchronizaci. Obecně je tato knihovna uzpůsobena pro experimentální a dobře kontrolovatelný vývoj nejen algoritmů strojového učení. Myšlenka této knihovny je založena na toku operací uspořádaných v grafu. Tyto operace jsou výlučně prováděny na tensorech tj. vstupem i výstupem je tensor (v Pythonu se jedná o ndarray). Intepretováno jinak: Vrcholy tohoto grafu reprezentují matematické operace, které vlastní či mění stav výpočtu. Po hranách pak proudí jednotlivé tensory Kromě jiného obsahuje tensorflow nástroje pro diferenciaci libovolné chybové funkce. (Abadi et al. nedatováno).

Omezíme-li se na modely hlubokého učení, umožňuje Tensorflow v podstatě dva přístupy k implementaci. Odpovídá tomu i rozdělení oficiální dokumentace na „High level APIs“ a „Low Level APIs“. První „High level“ funkcionalita se překrývá se specifikací knihovny Keras a bude pojednána v následující kapitole. Je to právě ono „Low Level API“, které za pomocí nativních struktur umožňuje značnou kontrolu nad budovanými modely. V praktické části této práce se však s tímto API nepracuje, proto nebude nadále rozepisováno.

Za poslední zmínku z knihovny TensorFlow stojí nástroj TensorBoard, který slouží k vizualizaci jednotlivých fází učení modelů a mnohým dalším vizualizačním   
a optimalizačním potřebám vývojáře. Umí rovněž vizualizovat celý nadefinovaný komputační graf.

## Keras

Keras je již zmiňované „High level API“ primárně určené pro jednoduché budování modelů hlubokého učení. S jeho pomocí lze nadefinovat a parametrizovat většinu základních architektur (CNN, RNN, LSTM, ANN, GAN) a jejich variant pro praktické   
a aplikační potřeby.

Uživatelská přívětivost je podstatnou zásadou této knihovny, proto je možné nadefinovat funkční modely velice rychle. Keras je nadstavbou nad TensorFlow, popřípadě jinými alternativami knihoven strojového učení jako je například Theano. Pro trénování implementovaných modelů lze s výhodou využít grafického hardwaru, kde mohou výpočty na tensorech probíhat paralelně a celý proces je několikanásobně rychlejší.

Základní definovatelnou strukturou je v Kerasu Model. Ten slouží jako placeholder pro různorodé druhy vrstev. Jednotlivé vrstvy lze do modelu přidávat buď sekvenčně jednu za druhou nebo je v případě komplikovanějších modelů možné definovat souslednost vrstev do grafu. Veškeré vrstvy jsou plně parametrizovatelné a jejich funkce se odvíjí od jejich typu. Keras podporuje standardní plně propojené vrstvy, konvoluční triplety, rekurentní vrstvy a mnohé další. Keras rovněž obsahuje nativní nástroje pro datový preprocessing, škálu callbacků volatelných při trénování, nástroje pro načítání standardních datových sad, vizualizační nástroje a další utility. Pochopitelně jsou přítomné parametrizovatelné funkce pro trénování a evaluaci nadefinovaných modelů.

# Implementace metod strojového učení při generování

Konkrétně specifikovaný cíl této kapitoly a bakalářské práce vůbec je demonstrovat a evaluovat navrženou metodu pro generování scény v diskrétním dvojrozměrném prostoru. Veškeré implementace, které jsou zde demonstrovány byly implementovány v jazyce Python ve verzi 3.6. S daty bylo primárně pracována jako s csv soubory.

## Příklad hanojských věží

Jako postranní úloha při zpracování bakalářské práce vznikla implementace, která svým principem předjímá postup hlavní aplikace. Ve stručnosti je zde tedy prezentována.

## Definice cíle a východisek

Tato kapitola představuje předpoklady, východiska, a užité abstrakce, které byly v rámci prezentované metody aplikovány.

Na nejvyšší úrovni jsou vstupní data pro algoritmus definována jako scény . Scéna je diskrétním prostorem možných pozic , kde a . Na těchto pozicích se mohou vyskytovat objekty.

Rov. 8 Příklad diskrétního prostoru 32, kde každá pozice představuje jednu konkrétní třídu objektů

Dohromady vytvářejí tyto objekty určitou množinu . Tato množina vytváří smysluplnost tj. scénický význam ve dvou aspektech. Za prvé je pro podstatná samotná přítomnost nebo nepřítomnost objektů ve scéně. Například vyskytuje-li se ve scéně 5 postelí je její význam jiný než vyskytuje-li se v ní 5 krabic s dynamitem. Za druhé je význam scény v menší míře  spoluvytvářen i vzájemnými pozičními relacemi v diskrétním prostoru scény. Ukázáno na příkladu: Nachází-li se ve scéně 5 židlí a 5 stolů, je význam scény jiný, když se všech pět židlí nachází u stolu, než když tomu tak není. Prezentovaný algoritmus řeší druhý aspekt scénického významu, totiž umístění objektu na nejpravděpodobnější místo vzhledem k aktuálnímu kontextu prostoru. Přítomnost jednotlivých objektových tříd ve scéně je řízena uživatelsky.

Každý objekt přítomný ve scéně sebou nese vektor své pozice, rotace a velikosti. První abstrakcí prezentovaného modelu je, že z těchto tří vlastností objektu je pro zjednodušení úlohy uvažován pouze plošný dvoudimenzionální vektor . Vektory rotace a velikosti uvažovány nejsou. Kromě geometrických vlastností objektu, je každý objekt kategorizován danou třídou kterou představuje.

Z uvedeného již lze názorně představit obecný cíl. Kroky, které povedou k jeho naplnění jsou předmětem následující kapitoly. Obecný cíl je definován následovně: Za předpokladu výchozí scény , která je určena množinou objektů , umístit uživatelsky vybraný objekt na logickou pozici. Tím vzniká nová scéna . Tento proces bude iterativní, to znamená, že v jednom kroku generovacího algoritmu bude scéna doplněna právě o jeden objekt. Načež se tato nová scéna stane východiskem pro další krok generování.

Rov. 9: Proces generování scény , kde odpovídá výchozí scéně, vyjadřuje počet iterací algoritmu   
a představuje predikovaný objekt třídy ..

## Popis algoritmu a modelové situace

Celý algoritmus sestává ze tří hlavních fází. Za prvé je to fáze předzpracování dat. V druhé fázi se natrénují neuronové sítě a v poslední fázi je volána metoda generování. Modelovou situací je dogenerování uživatelsky volených objektů do prostoru libovolného pokoje ohraničeného zdmi. Aplikační užití tohoto principu se však neomezuje na tento konkrétní případ a lze ho pravděpodobně aplikovat i pro jiná zadání.

### Charakteristika vstupních dat

Pro všechny testované případy byla užita vstupní datová sada čítající 23 ručně modelovaných příkladů. Scénický význam je definovaný jako pokoj. Následující tabulka představuje použité objekty a logiku, se kterou byly předměty do scén umísťovány. Tato logika je podstatná pro následnou evaluaci nagenerovaných vzorků.

Tabulka 1: Názorně popisuje objekty přítomné ve vstupních datech. Objekt typu SS je takový objekt, který pouze vystupuje ve scénách a nehodláme ho predikovat.

|  |  |  |  |
| --- | --- | --- | --- |
| Id | Interpretace | Typ | Logika umístění |
| 0 | Prázdný prostor | SS | Reprezentuje prázdný **vnitřní** prostor místnosti. |
| 1 | Stěna | SS | Stěna je vždy umístěna po obvodu scény. |
| 2 | Stůl | O | Stůl je umístěn náhodně uvnitř prostoru scény včetně krajních oblastí. |
| 3 | Židle | O | Židle je objekt, na kterém sledujeme hned několik hypotéz a platí pro něj několik speciálních pravidel. Tato pravidla jsou rozvedena v kapitole 7. |
| 4 | Skříň | O | Skříň je umístěna vždy v blízkém kontaktu se stěnou. |
| 5 | Objekt ve volném prostoru | O | Objekt volného prostoru, je abstraktní objekt, který je v rámci scén vždy umístěn do prázdného prostoru. |
| 6 | Okno | O | Objekt umístěný ve zdi. Je dbáno na to aby se před ním nevyskytoval objekt třídy **4**. |
| 9 | Prostor za zdí | SS | Abstraktní objekt, který simuluje neznámý prázdný **vnější** prostor. Slouží jako protiklad k objektu **0** a pomáhá při predikci prostorů s nepravidelným tvarem. |

### Předzpracování dat

Tato fáze je operačně nejnáročnějším krokem, ve kterém dochází primárně k vektorizaci dat ze vstupních příkladů a vytvoření dvojic , které na konci vytvoří datovou sadu pro neuronové sítě.

Jako parametr pro tuto fázi figurují čtyři pole. První pole drží data ke všem vstupním příkladům. Druhé pole definuje jádra a jejich tvary, pomocí nichž jsou procházena vstupní data. Pole drží speciální symboly viz. Tabulka 1. Jsou to třídy, pro které nebudeme chtít predikovat jejich polohu - například podlaha či zeď. Čtvrté pole drží všechny třídy objektů v datech včetně .

1. **For** každé jádro v C **do:**

2. **For** každý příklad v D **do:**

3.V = x\_vektory\_po\_projití\_jádrem\_C()

5. **For** každý vektor v poli V **do:**

6. **For** každý objekt z O, který se nachází ve V **do:**

7.vytvor\_kombinace\_z\_pritomnych\_objektu\_stejne\_tridy()

8. pro\_tyto\_kombinace\_vytvor\_vektory\_x()

9. pro\_tyto\_kombinace\_vytvor\_vektory\_y()

Pseudokód výše vyjadřuje iterativní práci se vstupními daty. Každý příklad je pro každé jádro transformován („rozřezán“) na vektory velikosti příslušného jádra. Jádro se pohybuje po vstupní matici scény zleva doprava směrem dolů vždy o jeden krok, podobně jako konvoluční jádro. S operací konvoluce nemá však tato metoda nic společného - podobnost lze vidět pouze v pohybu po matici.

Ze vzniklých vektorů se už ustanovují dvojice Názorně je tato operace představena na následujícím příkladu, kde se generuje vstupní příklad pro třídu 4   
a velikost jádra (). Předpokládá se, že ve vstupních datech jsou jen tři třídy objektů, to znamená .

Jako vstup pro neuronové sítě se obecně hodí normalizované hodnoty. V rámci prezentovaného algoritmu spočívá tato normalizace v zakódování vstupního vektoru na tvar ,kde každý jedničkový příznak ve trojici označuje přítomnost objektu dané kategorie. Například šestá trojice z - označuje přítomnost objektu třídy čtyři na pozici . Pozice příznaku v rámci trojice se odvíjí od pořadí dané objektové třídy v poli . Čtyřka je v poli na druhé pozici. Počet prvků v tomto vektoru je dán vztahem .

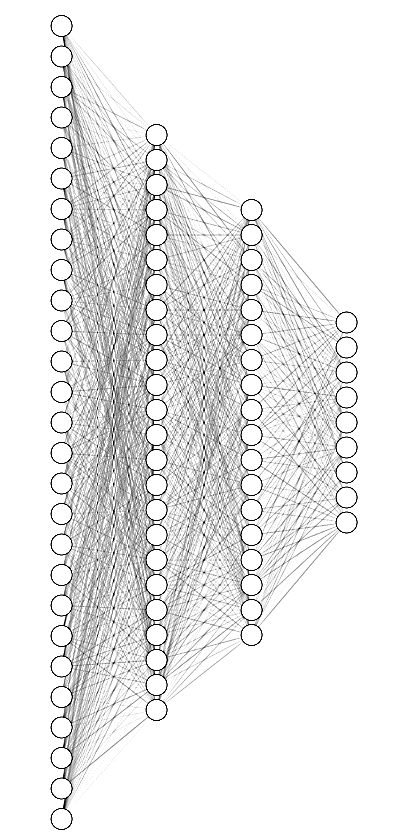
V tomto příkladu generujeme učící dvojice pro třídu 4. Proto je nutné vektor transformovat na , kde se na pozici objektu 4 vloží 0. Tím se vytváří vstupní vektor , který simuluje nepřítomnost objektu 4 na pozici a výstupní vektor , který na pozici tento objekt „vkládá“. Tato skutečnost je zvýrazněna ve vektorech modrou barvou. V tomto smyslu se iteruje přes všechny jaderné vektory a vytváří se dvojice .

Při takovém postupu je přirozeně vygenerováno mnoho redundantních dvojic, neboť vektory vyprodukované jádry se často opakují. Redundace dat pro neuronovou síť je obecně diskutovaným tématem. Článek Davida Medlera ji popisuje jako pozitivní jev (Medler a Dawson 1994). V případě prezentovaných experimentů byly redundance zachovány s hypotézou posílení často opakovaných vzorků. Nicméně v případě větší datové sady by bylo pravděpodobně žádoucí je odstranit.

Tímto byl popsán způsob generování učících dvojic pro jeden vektor jedné třídy jádra . Ve finálním příkladu se pracuje celkem se čtyřmi jádry a pěti predikovatelnými objekty viz Tabulka 1. Při tomto počtu vzniká na výstupu předzpracovací fáze celkem dvojic csv souborů, které agregují vektory a pro jednu konkrétní třídu a konkrétní jádro. Tato data budou v další fázi vstupem pro neuronových sítí. Obecný vztah pro výsledný počet potřebných sítí je tedy , to znamená počet jader krát počet unikátních objektových kategorií.

### Natrénování modelů

Pro natrénování všech modelů byla využita jedna obecná architektura dopředné neuronové sítě. Veškeré modely byly definovány ve frameworku Keras na bázi Tensorflow s podporou učení pomocí grafické karty. Experimentováno bylo s několika různými tvary a parametry. Výsledná topologie NN se v konečné fázi ustálila na dvou skrytých vrstvách a . Dimenze vstupní vrstvy pro vektor odpovídá vztahu a je tedy pro každé jádro jiná. Počet neuronů ve skrytých vrstvách se směrem k výstupní vrstvě snižuje do pyramidového tvaru s koeficientem   
0.9 P pro a 0.7 P pro . Výstupní vrstva má velikost jádra.



Obr. 8: Vizualizace neuronové sítě pro jádro , předpokládaje 3 objekty podobně jako v příkladu z kapitoly 6.4.2. Počet neuronů odpovídá L1 = 27, L2 =24, L3=18, L4=9. Vizualizováno pomocí nástroje dostupném na http://alexlenail.me/NN-SVG/index.html

Aktivační funkce za skrytými vrstvami je ReLU. Za předposlední vrstovu je za účelem pravděpodobnostního výstupu implementována funkce softmax. Jako chybová funkce byla zvolena categorical\_cross\_entropy implementovaná v Kerasu.

Tabulka 2.: Statistiky natrénovaných sítí TODO:

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  |  |  |
|  |  |  |

### Generování scén

Generování scén probíhá pomocí uspořádaného volání natrénovaných neuronových sítí. V každém kroku generování je do scény přidán jeden objekt. Ten je volen uživatelsky. Podobně jako ve fázi přípravy dat figuruje i v této fázi pole jader , které určuje, jakým způsobem se bude prostor procházet. Každý krok generování musí mít svoji výchozí scénu . V první iteraci je tato scéna zvolena uživatelsky a v dalších iteracích je výchozí scénou výstup předešlého generování viz. Rov. 9.

Postup generování shrnuje následující pseudokód:

**For** každé jádro v C **do:**

J = projdi\_výchozí\_prostor\_a\_vrať\_pole\_pro\_predikci()

**For** každé pole v J **do:**

N = nacti\_spravnou\_neuronovu\_sit()

P = N.predikuj(pole)

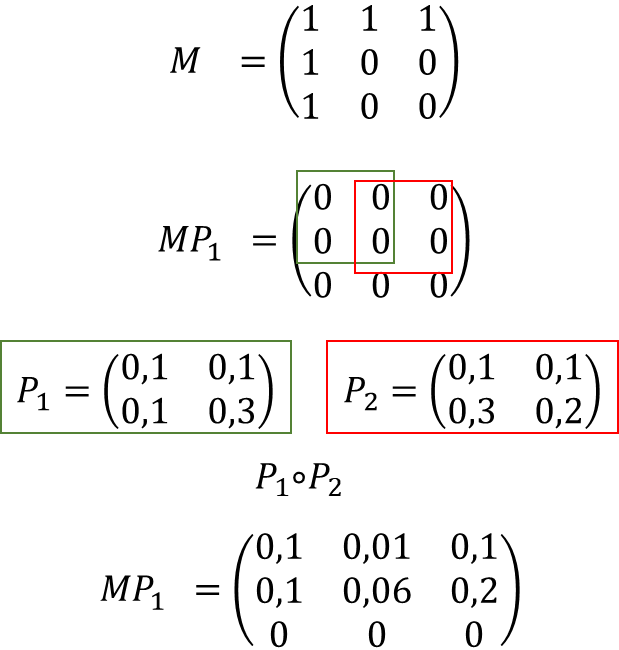
zanes\_predikci\_do\_soucinove\_matice(P)

K = vyber\_kandidata\_ze\_soucinove\_matice()

umisti\_do\_prostoru(K)

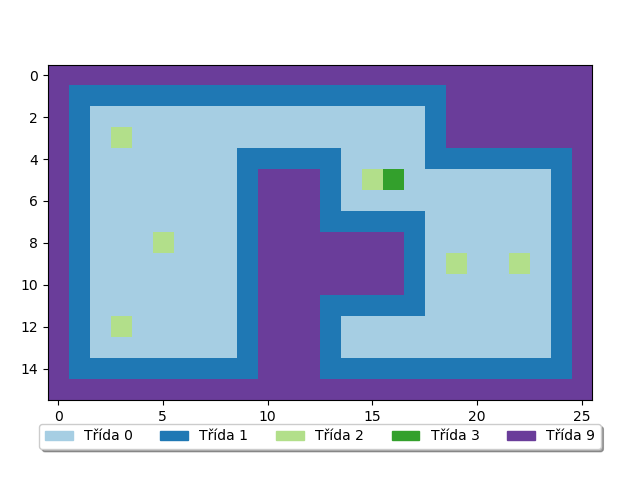
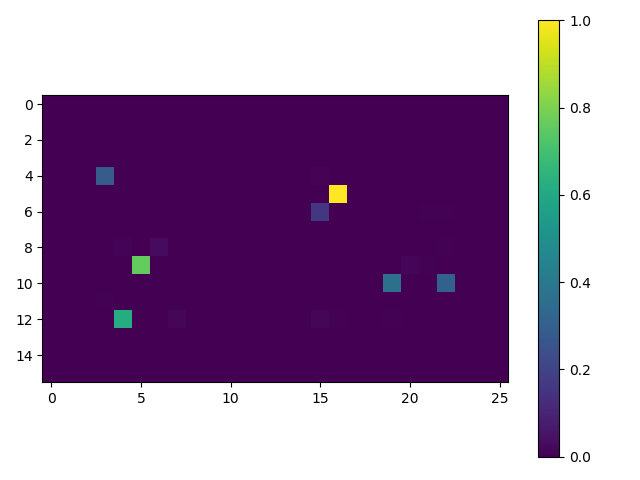
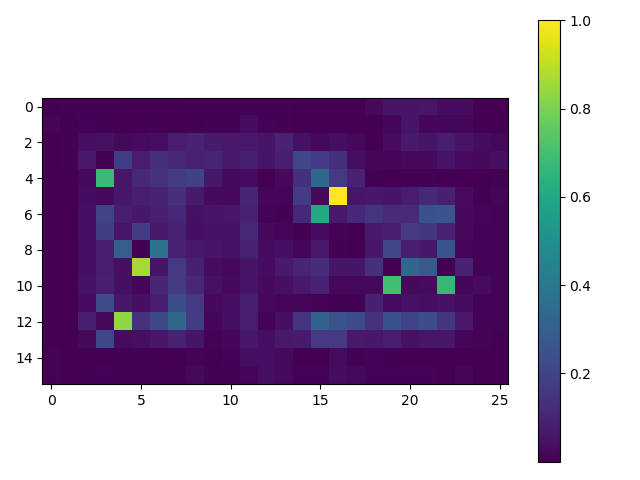
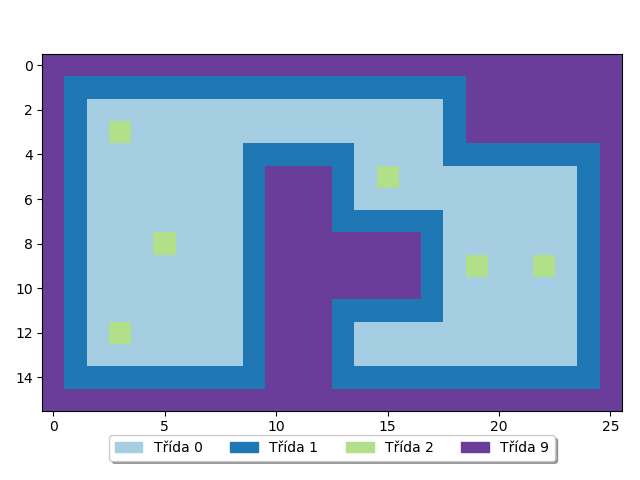
Jde tedy o postupné projití výchozí matice pomocí jader. Pro všechna tato jádra se musí načíst správná neuronová síť, která predikuje pozici objektu požadované třídy.

Výsledné predikce se agregují do dvou matic, které postupně vytvářejí matice pravděpodobnosti výskytu dané třídy v prostoru . Pro finální predikci se osvědčila matice, která je nadále označována jako součinová matice pravděpodobnost výskytu - . V rámci utváření této matice dochází k postupnému pronásobování prediktivních matic a vzniká tak finální obraz tom, kde by se objekt dané třídy měl nacházet. Nejedná se o násobení matic v klasickém smyslu, nýbrž ve smyslu Hadamardova produktu , to znamená násobení po prvcích. Proces vytváření názorně ukazuje následující obrázek.



Obr. 9: Demonstruje proces utváření . jsou postupné predikce na vstupním prostoru pro objekt třídy , které se pomocí Hadamardova produktu pronásobují do výsledné

Kromě se v průběhu generování utváří ještě součtová pravděpodobnostní matice . Děje se to podobným způsobem jako u součinové matice s tím rozdílem, že namísto Hadamardova produktu je aplikován součet.



Obr. 10: Postupně: Výchozí prostor , , , výsledná predikce objektu třídy 3.

Co se týče nejpravděpodobnějších pozic a ve větší míře korelují. Nicméně v průběhu testování produkovala méně chyb než a proto se výsledná predikce řídí součinovou maticí .

Výběr finální pozice predikovaného objektu lze určit různými způsoby. Nejjistější je vybrat nejpravděpodobnější místo na . V takovém případě by však v rámci stejných prostorů nedocházelo k žádoucí variabilitě obsahu. Proto lze aplikovat například náhodný vážený výběr z uspořádaných pravděpodobností či zvolit jinou techniku výběru. Záleží rovněž na aplikačním záměru, neboť někdy může být žádoucí vybrat právě maximum z

### Parametrizace algoritmu

Na závěr této kapitoly lze podotknout, že algoritmus je v mnoha částech svého fungování parametrizovatelný. Pro lepší výsledky se dá primárně experimentovat s tvary a počty jader, architekturou neuronové sítě, se způsobem výběru finálního kandidáta, popřípadě se způsobem agregace do matic pravděpodobnosti výskytu.

# Testování hypotetických výstupů algoritmu

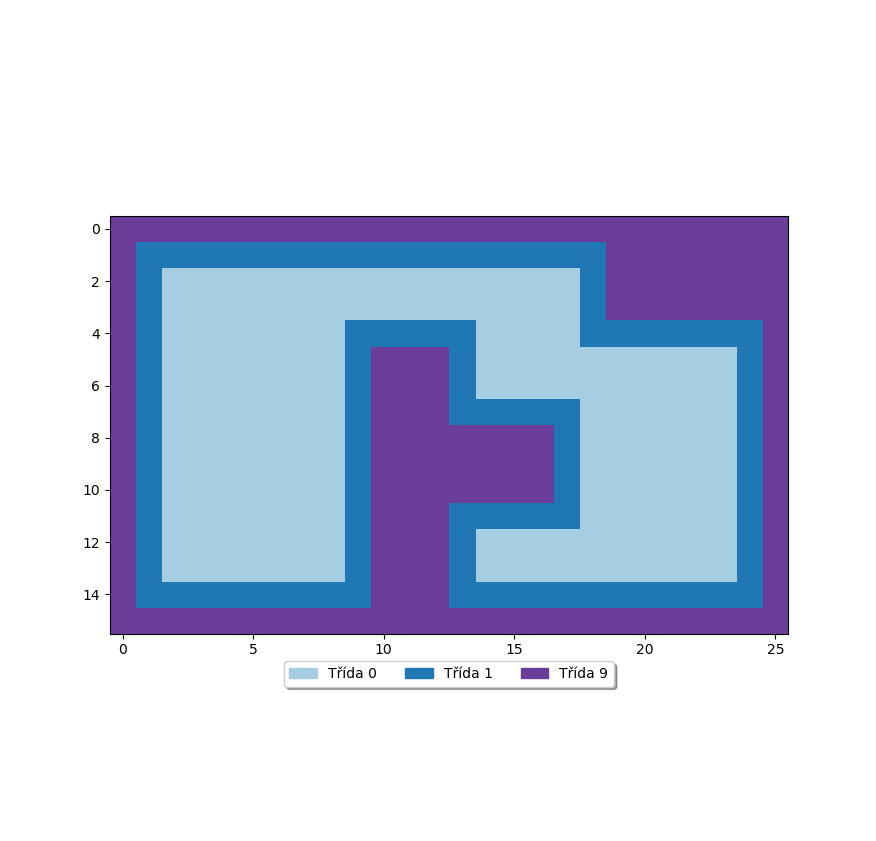
Postup měření úspěšnosti navrženého algoritmu je proveden následujícím způsobem. Uvnitř datové sady, která byla představena v předchozí kapitole, jsou dodržována určitá relační pravidla mezi objekty. V případě, že budou tyto vlastnosti nalezeny i v algoritmicky nagenerovaných datech bude potvrzeno, že se algoritmus naučil tuto logiku replikovat. Vzhledem k stochastické povaze inicializace neuronových sítí je pravděpodobné, že na konci každého procesu učení může algoritmus produkovat nepatrně odlišné výsledky. To samé se týká generovaných scén při zapnutí náhodného váženého výběru viz kapitola .

Následující příklad scény představuje prostor, ve kterém jsou zobrazeny a dodrženy všechny sledované vlastnosti. V dalších částech jsou pro vizualizace prostorů využity nástroje knihovny matplotlib[[2]](#footnote-2).

Rov. 10: Příklad vstupního prostoru.

## Jednotkové testování

V první fázi testování bylo ověřováno zda dokáže algoritmus replikovat konkrétní vlastnosti v jednoduchém prostoru, který není rušen kontextem ostatních objektů. Tento prostor je definovaný jako matice , po jejímž obvodu se nachází objekt třídy 1 to znamená zeď. Pro všechny predikované objekty byla vždy vybrána jejich nejpravděpodobnější pozice. (max matice pravděpodobnosti). Všechny výsledky testů jsou verifikovatelné s pomocí Pythonovských skriptů přiložených v příloze bakalářské práce.



Obr. 11: Vizualizace prostoru M1

### Hypotéza blízké relace na

V rámci testování se potvrdilo, že algoritmus replikuje těsnou relaci objektů tříd 2 a 3, kde sledujeme následující vlastnosti.

Ze všech 34 vnitřních pozic matice doplnil algoritmus v rozsahu čtyř iterací židle kolem stolů po vzoru .Algoritmus zafungoval dobře i v případech, že byl ve výchozím prostoru matice umístěn větší počet stolů. Při testování na deseti scénách, ve kterých byly náhodně umístěny tři stoly (přiměřeně od sebe) obkroužil algoritmus ve všech případech stoly židlemi, sledujíce vlastnosti a částečně i .

### Hypotéza speciální vlastnosti , a na

V tomto testu jsou sledovány vlastnosti vyobrazené v maticích , a . Jedná se o jednoduché pravidlo, které určuje, že v případě umístění stolu jednu pozici od stěny, nebude na tuto pozici přidána židle, která by tam standardně měla být dle pravidla.

Pro test vlastnosti byly stoly umístěny ob jednu pozici od obvodového zdiva. Celkem na  vzniklo padesát testovacích pozic. V rozsahu tří iterací neumístil algoritmus židle kolem stolu po vzoru v 6 případech. Při čtvrté iteraci došlo obvykle k tomu, že nejpravděpodobnější pozice byla predikována ke zdi po vzoru , což není nelogické, ale vzhledem k pravidlu je to chyba. V 11 případech byl výsledek v souladu s pravidlem i po čtvrté iteraci a algoritmus preferoval umístit čtvrtou židli raději do volného prostoru, a nikoliv ke stěně.

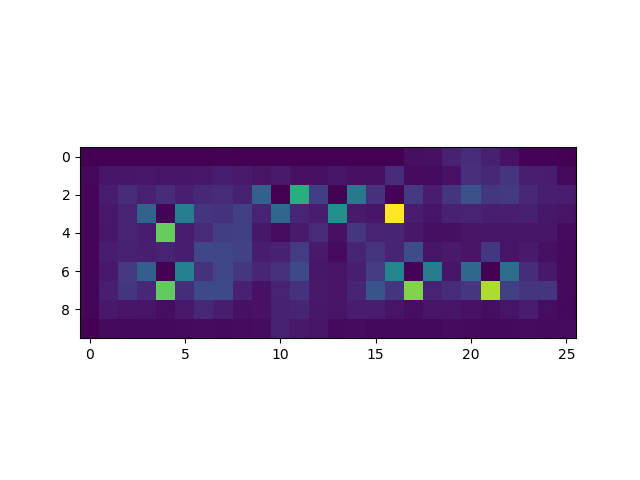
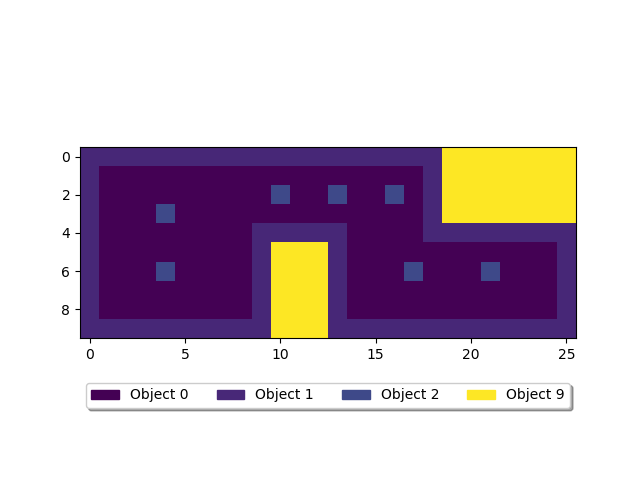
Tabulka 3.: Počet chyb ve všech 50 testovacích pozicích vzhledem   
k pravidlům , , po 2., 3. a 4. iteraci

|  |  |  |  |
| --- | --- | --- | --- |
| Případ / chybovost | Po 2. iteraci | Po 3. iteraci | Po 4. iteraci |
| U stěny (, ) | 0 | 3 | 30 |
| V rohu () | 0 | 3 | 3 |
| Úspěšnost (%) | 100% | 88% | 22% |

### Hypotéza vlastností a pro větší počet objektů

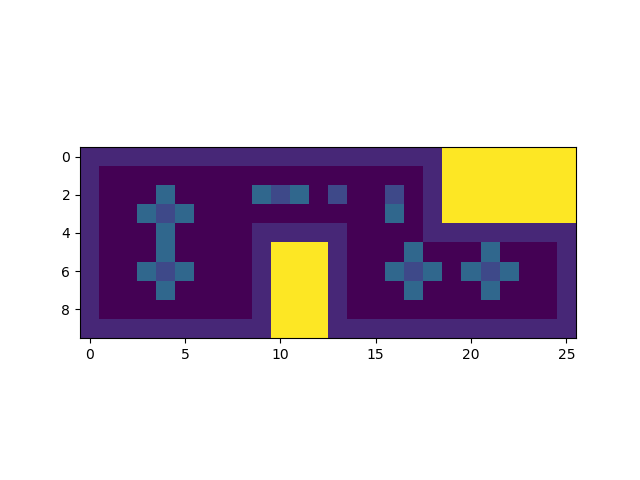
Pro účely tohoto testování byla zvolena demonstrativní situace viz. Obr. 10 založená na . Kromě již verifikované vlastnosti , jsou sledovány i vlastnosti a .

Celkem bylo spuštěno 30 iterací s cílem umístit objekt třídy 2 dle předpokladu a .V následujícím popisu bude ukázáno a komentováno chování algoritmu.



Obr. 12: Vlevo vizualizace výchozí situace (iterace 0). Vpravo pravděpodobnosti výskytu objektu kategorie 3 tj, židle

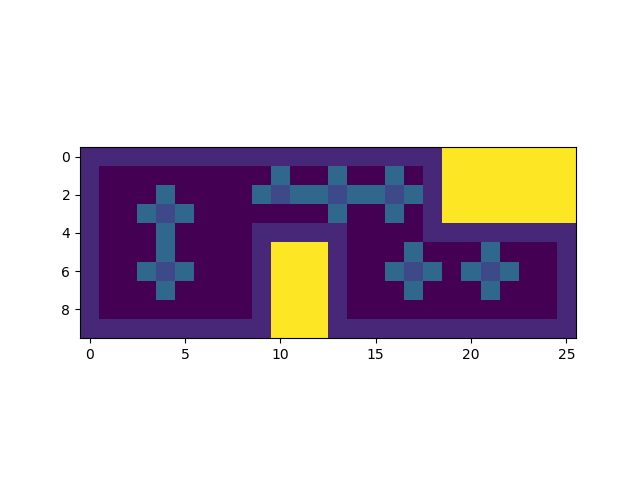
Až do 18. iterace se algoritmus držel pravidel a viz. Obr. 11. V rámci této výchozí situace lze konstatovat, že algoritmus víceméně upřednostňoval neproblematická místa vzhledem k pravidlům (oranžová oblast). Výjimku tvoří červeně vyznačené místo, které bylo doplněno už v 8. iteraci.





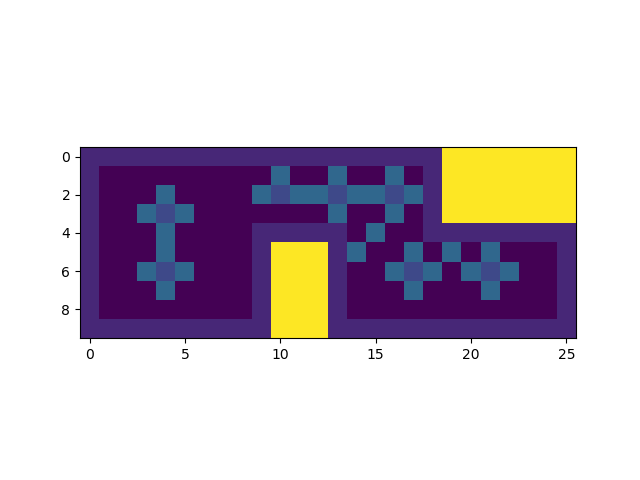
Obr. 13: Mezistav budování prostoru (iterace 16)

Od devatenácté iterace přestal algoritmus pravidlo dodržovat a objekty doplňoval pouze na základě pravidla viz. Obr. 12.



Obr. 14: Mezistav budování prostoru (iterace 26)

Konečně je zde představen stav po 30. iteraci (Obrázek 8). V červeně označeném místě bylo algoritmem sice dodrženo pravidlo [X2], nicméně vzhledem k tomu, že židle musely být někam umístěny, byly na úkor tohoto místa umístěny nelogicky do prostoru vyznačeném zelenou barvou.

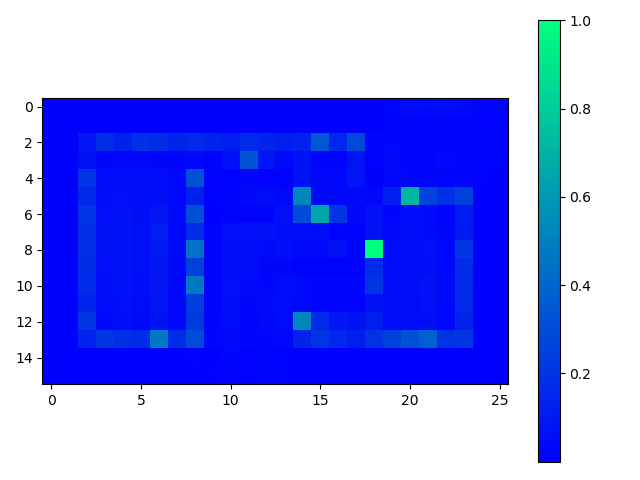


Obr. 15 ...

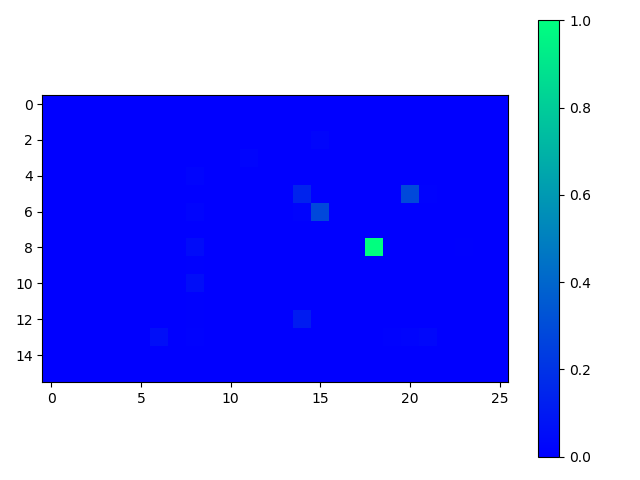
### Hypotéza vlastnosti a na

Tato hypotéza předpokládá, že se objekt třídy 4, to znamená skříň, bude stavět ke stěně. Testováno bylo 10 iterací, při kterých se vkládat objekt této třídy. Výsledek tohoto generování je vyobrazen na Obr. 16 . V rámci tohoto testování byl povolen vážený náhodný výběr ze seřazené posloupnosti predikcí.

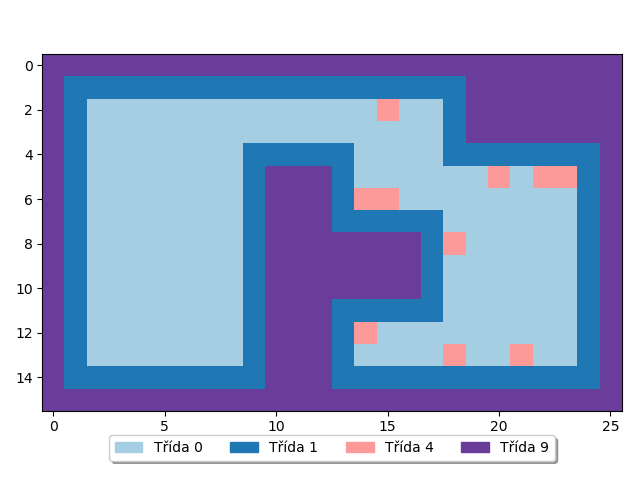
Následující obrázek (Obr. 14) ukazuje součtovou matici pravděpodobnosti výskytu třídy čtyři před první predikcí. Pravděpodobnosti jsou správně rozloženy kolem stěny . Predikce objektu se pak odvíjí od součinové matice viz. Obr. 15.



Obr. 16: Součtová matice pravděpodobnosti pro predikci výskytu objektu třídy 4 v první iteraci.



Obr. 17: Součinová matice pravděpodobnosti pro predikci výskytu objektu třídy 4 v první iteraci.



Obr. 18: Výsledná scéna po 10 iteracích.

V rámci tohoto testování se replikace vlastnosti potvrdila a napříč nagenerovanými scénami nedocházelo k odklonům od tohoto pravidla kromě občasného umístění skříně na roh. Tuto skutečnost lze vysvětlit nedostatečným zdůrazněním tohoto znaku ve vstupních datech.

### Hypotéza vlastnosti na

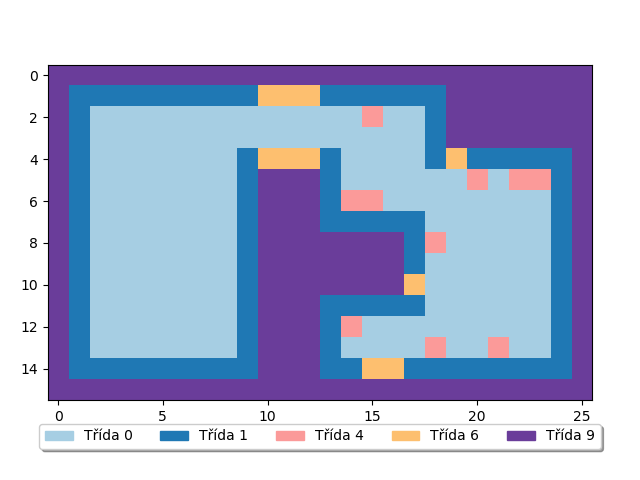
Hypotéza testuje vlastnost , která představuje skutečnost, že objekt třídy 5, to znamená objekt popsatelný jako „objekt volného prostoru”, by se uvnitř měl objevit na místě, kolem kterého existuje rozpoznatelný volný prostor. Řečeno jinak, existuje-li v shluk objektů, měl by být objekt třídy 5 predikován mimo tento shluk.

Lorem ipsum.

### Hypotéza vlastností a na

Vlastnosti a představují skutečnost, že objekt třídy 6, který zastupuje okno, by měl vždy nahradit objekt třídy jedna, tedy zeď. Současně je sledováno, že okno nebude umístěno za objektem, který by hypoteticky bránil průchodu světla. Primárně se jedná o objekt skříně (4).

Jako vstupní scéna pro tento test byl zvolen výstup předešlého testování vlastnosti . Algoritmus byl spuštěn desetkrát a v každém svém kroku měl do scény přiřadit jedno okno. Výsledek je znázorněn na Obr. 17. V rámci tohoto testu nebyl povolen vážený náhodný výběr.



Obr. 19 Příklad výsledné scény po 10 iteracích, ve kterých se přidávala objektová třída 6

## Integrační testování

Z oblasti tradičního softwarového vývoje je převzat pojem integračního testu. Ten je zde používán ve smyslu otestování, jak dobře jsou sledované vlastnosti replikovatelné v případě, že do generovaného prostoru vstupuje větší variabilita objektů. Zároveň jsou v této kapitole prezentovány tři náhodně vygenerované pokoje a jejich vizualizace v herním enginu Unity.

# Závěr

...bylo představil pozoruhodnou oblast, kde se metody strojového učení prakticky aplikují na generování obsahu.

Na základě poznaného aktuálního stavu a výsledků v oboru strojového učení generativních modelů lze konstatovat, že generování bude a je obecně doménou komplexnějších generativních architektur typu GAN v kombinaci s konvolučními   
a jinými architekturami.

V rámci praktické části byla implementována originální nadstavba, která umožnila použít klasické modely neuronových sítí diskriminativního charakteru pro úkol generování jednoduché scény pokoje. Výsledky tohoto generování byly otestovány a změřeny. Jádrem této implementace je účelná kombinace pravděpodobnostních výstupů četných neuronových sítí. Každá tato síť je namodelována tak, aby jejím výstupem byl pro tuto kombinaci hodnotný příspěvek. Principielně lze uvažovat o tom, že by se této myšlenky kombinace dalo využít, popřípadě ji rozšířit, i pro jiné úkoly. Omezení, které se s tímto pojí byla konstatována v průběhu práce.

Spojitost, konvoluce? Specifické žánry? Natrénování na různých prostorech. Našěptávač pozice.

# Seznam použité literatury

ABADI, Martın, Paul BARHAM, Jianmin CHEN, Zhifeng CHEN, Andy DAVIS, Jeffrey DEAN, Matthieu DEVIN, Sanjay GHEMAWAT, Geoffrey IRVING, Michael ISARD, Manjunath KUDLUR, Josh LEVENBERG, Rajat MONGA, Sherry MOORE, Derek G MURRAY, Benoit STEINER, Paul TUCKER, Vijay VASUDEVAN, Pete WARDEN, Martin WICKE, Yuan YU a Xiaoqiang ZHENG, nedatováno. TensorFlow: A system for large-scale machine learning. 21.

BECKHAM, Christopher a Christopher PAL, 2017. A step towards procedural terrain generation with GANs. *arXiv:1707.03383 [cs, stat]* [online]. [vid. 2019-03-15]. Dostupné z: http://arxiv.org/abs/1707.03383

BHANDARE, Ashwin, Maithili BHIDE, Pranav GOKHALE a Rohan CHANDAVARKAR, 2016. Applications of Convolutional Neural Networks. **7**, 10.

CARLI, D. M. D., F. BEVILACQUA, C. T. POZZER a M. C. DORNELLAS, 2011. A Survey of Procedural Content Generation Techniques Suitable to Game Development. In: *2011 Brazilian Symposium on Games and Digital Entertainment*: *2011 Brazilian Symposium on Games and Digital Entertainment* [online]. s. 26–35. Dostupné z: doi:10.1109/SBGAMES.2011.15

FISHER, Matthew, Daniel RITCHIE, Manolis SAVVA, Thomas FUNKHOUSER a Pat HANRAHAN, 2012. Example-based synthesis of 3D object arrangements. *ACM Transactions on Graphics* [online]. **31**(6), 1. ISSN 07300301. Dostupné z: doi:10.1145/2366145.2366154

GAGNIUS, Paul, 2017. *Markov Chains: From Theory to Implementation and Experimentation*. B.m.: Wiley-Blackwell. ISBN 1-119-38755-8.

GIACOMELLO, Edoardo, Pier Luca LANZI a Daniele LOIACONO, 2018. DOOM Level Generation using Generative Adversarial Networks. *arXiv:1804.09154 [cs, stat]* [online]. [vid. 2018-06-14]. Dostupné z: http://arxiv.org/abs/1804.09154

GLOROT, Xavier, Antoine BORDES a Yoshua BENGIO, 2011. Deep Sparse Rectiﬁer Neural Networks. 9.

GOODFELLOW, Ian, 2016. NIPS 2016 Tutorial: Generative Adversarial Networks. *arXiv:1701.00160 [cs]* [online]. [vid. 2018-06-14]. Dostupné z: http://arxiv.org/abs/1701.00160

GOODFELLOW, Ian, Yoshua BENIGO a Aaron COURVILLE, 2016. *Deep Learning* [online]. B.m.: MIT Press. Dostupné z: http://www.deeplearningbook.org

GOODFELLOW, Ian, Jean POUGET-ABADIE, Mehdi MIRZA, Bing XU, David WARDE-FARLEY, Sherjil OZAIR, Aaron COURVILLE a Yoshua BENGIO, 2014. Generative Adversarial Nets. In: Z. GHAHRAMANI, M. WELLING, C. CORTES, N. D. LAWRENCE a K. Q. WEINBERGER, ed. *Advances in Neural Information Processing Systems 27* [online]. B.m.: Curran Associates, Inc., s. 2672–2680 [vid. 2018-06-04]. Dostupné z: http://papers.nips.cc/paper/5423-generative-adversarial-nets.pdf

HENDRIKX, Mark, Sebastiaan MEIJER, Joeri VAN DER VELDEN a Alexandru IOSUP, 2013. Procedural Content Generation for Games: A Survey. *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM)*. **9**(1), 24. ISSN 1551-6857.

ISOLA, Phillip, Jun-Yan ZHU, Tinghui ZHOU a Alexei A. EFROS, 2016. Image-to-Image Translation with Conditional Adversarial Networks. *arXiv:1611.07004 [cs]* [online]. [vid. 2019-03-15]. Dostupné z: http://arxiv.org/abs/1611.07004

JAIN, Rishabh, Aaron ISAKSEN, Christoffer HOLMGA a Julian TOGELIUS, 2016. Autoencoders for Level Generation, Repair, and Recognition. *Proceedings of the ICCC Workshop on Computational Creativity and Games*. 9.

JEBARA, Tony, 1996. Discriminative, Generative and Imitative Learning. 212.

JEBARA, Tony, 2004. *Machine Learning Discriminative and Generative* [online]. 1. vyd. United States: Springer US. ISBN 978-1-4419-9011-2. Dostupné z: https://www.springer.com/gp/book/9781402076473

JORDAN, M. I. a T. M. MITCHELL, 2015. Machine learning: Trends, perspectives, and prospects. *Science* [online]. **349**(6245), 255–260. ISSN 0036-8075, 1095-9203. Dostupné z: doi:10.1126/science.aaa8415

JULIAN, Togelius, Noor SHAKER a Mark J. NELSON, 2016. Introduction. In: *Procedural Content Generation in Games: A Textbook and an Overview of Current Research*. B.m.: Springer. ISBN 978-3-319-42714-0.

KAPUR, Arnav, Shreyas KAPUR a Pattie MAES, 2018. AlterEgo: A Personalized Wearable Silent Speech Interface. In: *23rd International Conference on Intelligent User Interfaces* [online]. New York, NY, USA: ACM, s. 43–53 [vid. 2019-03-08]. IUI ’18. ISBN 978-1-4503-4945-1. Dostupné z: doi:10.1145/3172944.3172977

KARRAS, Tero, Samuli LAINE a Timo AILA, 2018. A Style-Based Generator Architecture for Generative Adversarial Networks. *arXiv:1812.04948 [cs, stat]* [online]. [vid. 2019-03-21]. Dostupné z: http://arxiv.org/abs/1812.04948

KINGMA, Diederik P. a Max WELLING, 2013. Auto-Encoding Variational Bayes. *arXiv:1312.6114 [cs, stat]* [online]. [vid. 2018-12-26]. Dostupné z: http://arxiv.org/abs/1312.6114

KRIZHEVSKY, Alex, Ilya SUTSKEVER a Geoffrey E HINTON, 2012. ImageNet Classification with Deep Convolutional Neural Networks. In: F. PEREIRA, C. J. C. BURGES, L. BOTTOU a K. Q. WEINBERGER, ed. *Advances in Neural Information Processing Systems 25* [online]. B.m.: Curran Associates, Inc., s. 1097–1105 [vid. 2019-03-09]. Dostupné z: http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf

LECUN, Yann, Léon BOTTOU, Yoshua BENGIO a Patrick HAFFNER, 1998. Gradient-Based Learning Applied to Document Recognition [online]. [vid. 2019-03-08]. Dostupné z: http://yann.lecun.com/exdb/publis/pdf/lecun-98.pdf

MEDLER, David A a Michael R W DAWSON, 1994. Using Redundancy to Improve the Performance of Artificial Neural Networks. *Proceedings of the Tenth Canadian Conference on Artificial Intelligence*. 8.

MICHAEL REVOW, CHRISTOPHER K.I WILLIAMS a GEOFFREY E. HINTON, 1996. Using Generative Models for Handwritten Digit Recognition. *IEEE Transactions on pattern analysis and machine intelligence*. **18**(6), 15.

MITCHELL, Tom M., 1997a. *Machine Learning*. New York: McGraw-Hill. McGraw-Hill series in computer science. ISBN 978-0-07-042807-2.

MITCHELL, Tom M., 1997b. *Machine Learning*. New York: McGraw-Hill. McGraw-Hill series in computer science. ISBN 978-0-07-042807-2.

NG, Andrew Y a Michael I JORDAN, nedatováno. On Discriminative vs. Generative Classifiers: A comparison of logistic regression and naive Bayes. 8.

OLIPHANT, Travis E., 2007. Python for Scientific Computing. *Computing in Science & Engineering* [online]. **9**(3), 10–20. ISSN 1521-9615. Dostupné z: doi:10.1109/MCSE.2007.58

PEREIRA, Francisco Camara, 2008. *Creativity and AI: A Conceptual Blending approach*. Portugal: University of Coimbra. ISBN 978-3-11-019856-0.

SMITH, Gillian, 2015. An Analog History of Procedural Content Generation. *FDG*. 6.

SUAREZ, Patricia L., Angel D. SAPPA a Boris X. VINTIMILLA, 2017. Infrared Image Colorization Based on a Triplet DCGAN Architecture. In: *2017 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*: *2017 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)* [online]. Honolulu, HI, USA: IEEE, s. 212–217 [vid. 2019-03-15]. ISBN 978-1-5386-0733-6. Dostupné z: doi:10.1109/CVPRW.2017.32

SUMMERVILLE, Adam a Michael MATEAS, 2016. Super Mario as a String: Platformer Level Generation Via LSTMs. *arXiv:1603.00930 [cs]* [online]. [vid. 2019-03-10]. Dostupné z: http://arxiv.org/abs/1603.00930

SUMMERVILLE, Adam, Sam SNODGRASS, Matthew GUZDIAL, Christoffer HOLMGÅRD, Amy K. HOOVER, Aaron ISAKSEN, Andy NEALEN a Julian TOGELIUS, 2017. Procedural Content Generation via Machine Learning (PCGML). *arXiv:1702.00539 [cs]* [online]. [vid. 2018-05-31]. Dostupné z: http://arxiv.org/abs/1702.00539

TOGELIUS, Julian, Alex J. CHAMPANDARD, Pier Luca LANZI, Michael MATEAS, Ana PAIVA, Mike PREUSS a Kenneth O. STANLEY, 2013. Procedural Content Generation: Goals, Challenges and Actionable Steps. In: Simon M. LUCAS, Michael MATEAS, Mike PREUSS, Pieter SPRONCK a Julian TOGELIUS, ed. *Artificial and Computational Intelligence in Games* [online]. Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl Follow-Ups, s. 61–75 [vid. 2018-05-31]. ISBN 978-3-939897-62-0. Dostupné z: doi:10.4230/DFU.Vol6.12191.61

TOGELIUS, Julian, Georgios N. YANNAKAKIS, Kenneth O. STANLEY a Cameron BROWNE, 2010. Search-Based Procedural Content Generation. In: Cecilia DI CHIO, Stefano CAGNONI, Carlos COTTA, Marc EBNER, Anikó EKÁRT, Anna I. ESPARCIA-ALCAZAR, Chi-Keong GOH, Juan J. MERELO, Ferrante NERI, Mike PREUß, Julian TOGELIUS a Georgios N. YANNAKAKIS, ed. *Applications of Evolutionary Computation* [online]. Berlin, Heidelberg: Springer Berlin Heidelberg, s. 141–150 [vid. 2018-05-31]. ISBN 978-3-642-12238-5. Dostupné z: doi:10.1007/978-3-642-12239-2\_15

TUČKOVÁ, Jana, 2003. *Úvod do teorie a aplikací umělých neuronových sítí*. B.m.: ČVUT. ISBN 80-01-02800-3.

VAN DER WALT, Stefan, S. Chris COLBERT a Gaël VAROQUAUX, 2011. The NumPy array: a structure for efficient numerical computation. *Computing in Science & Engineering* [online]. **13**(2), 22–30. ISSN 1521-9615. Dostupné z: doi:10.1109/MCSE.2011.37

VLADIMÍR OLEJ a HÁJEK PETR, 2010. *Úvod do umělé inteligence*. B.m.: Univerzita Pardubice Fakulta ekonomicko-správní. ISBN 978-80-7395-307-2.

YANNAKAKIS, G. N. a J. TOGELIUS, 2011. Experience-Driven Procedural Content Generation. *IEEE Transactions on Affective Computing* [online]. **2**(3), 147–161. ISSN 1949-3045. Dostupné z: doi:10.1109/T-AFFC.2011.6

1. Large Scale Visual Recognition Challenge - soutěž v oblasti rozpoznání objektu. [↑](#footnote-ref-1)
2. Python knihovna pro vizualizace. Dostupné z https://matplotlib.org/. [↑](#footnote-ref-2)