Katedra informatiky
Přírodovědecká fakulta
Univerzita Palackého v Olomouci

DIPLOMOVÁ PRÁCE

Procedurálne generovanie herného obsahu

2024

Bc. Marek Schindler

Vedoucí práce:
doc. RNDr. Jan Konečný, Ph.D.

Studijní program: Aplikovaná informatika,
Specializace: Vývoj software

Bibliografické údaje

Autor:	Bc. Marek Schindler
Název práce:	Procedurálne generovanie herného obsahu
Typ práce:	diplomová práce
Pracoviště:	Katedra informatiky, Přírodovědecká fakulta, Univerzita Palackého v Olomouci
Rok obhajoby:	2024
Studijní program:	Aplikovaná informatika, Specializace: Vývoj software
Vedoucí práce:	doc. RNDr. Jan Konečný, Ph.D.
Počet stran:	50
Přílohy:	elektronické data v úložisku katedry informatiky
Jazyk práce:	slovenský

Bibliographic info

Author:	Bc. Marek Schindler
Title:	Procedural content generation for games
Thesis type:	master thesis
Department:	Department of Computer Science, Faculty of Science, Palacký University Olomouc
Year of defense:	2024
Study program:	Applied Computer Science, Specialization: Software Development
Supervisor:	doc. RNDr. Jan Konečný, Ph.D.
Page count:	50
Supplements:	electronic data in the storage of department of computer science
Thesis language:	Slovak

Anotácia

Diplomová práca sa zaoberá procedurálnym generovaním 3D prostredia. Praktická časť zahŕňa návrh a implementáciu Levelu 37 z prostredí The Backrooms. Level 37 predstavuje rozsiahly komplex prepojených miestností pripomínajúcich zatopené kúpeľne zariadenie. Procedurálne generovanie je technika, ktorá umožňuje automatické vytváranie herných prostredí a obsahu na základe algoritmov. Návrh prostredia prebiehal v nástroji Unreal Engine. Na generovanie nekonečného bludiska levelu 37 je implementovaný Wave function collapse algoritmus, ktorý generuje prostredie z vopred definovaných miestností na základe pravidiel. Práca analyzuje fungovanie tohto algoritmu, jeho implementačné stratégie a možnosti využitia v 3D prostredí. Program tiež obsahuje editor, v ktorom môžu byť pravidlá generovania upravené užívateľom.

Synopsis

The diploma thesis focuses on the procedural generation of 3D environments. The practical part involves designing and implementing Level 37 from The Backrooms. Level 37 is a vast complex of interconnected rooms resembling a flooded bathhouse. Procedural generation is a technique that enables automatic creation of game environments and content through the use of algorithms. The design of the environment was carried out using the Unreal Engine tool. To generate the infinite maze of Level 37, the Wave function collapse algorithm is implemented, which constructs the environment from predefined rooms based on specific rules. The thesis examines the functioning of this algorithm, its implementation strategies, and its potential applications in 3D environments. The program also includes an editor that allows users to modify the generation rules.

Kľúčové slová: procedurálne generovanie; Wilsonov algoritmus; Wave function collapse; The Backrooms; Level 37; Unreal Engine

Keywords: procedural generation; Wilson's algorithm; Wave function collapse; The Backrooms; Level 37; Unreal Engine

Ďakujem vedúcemu diplomovej práce doc. RNDr. Janovi Konečnému, Ph.D. za jeho cenné poznatky, rady a pripomienky, ktorými mi bol nápomocný pri tvorbe tejto práce.

Čestne vyhlasujem, že som celú prácu vrátane príloh vypracoval/a samostatne a za použitia iba zdrojov spomínaných v texte práce a uvedených v zozname literatúry.

dátum odovzdania práce podpis autora

Obsah

Zoznam obrázkov

Úvod

Niektoré počítačové hry navodzujú dojem omnoho väčšieho priestoru, než aký je možné v skutočnosti vytvoriť alebo uložiť. Pri takýchto hrách je kľúčový proces automatického vytvárania obsahu prostredníctvom algoritmov, známy ako procedurálne generovanie. Procedurálne generovanie obsahu je výkonná technika, ktorá sa stáva trendom v oblasti vývoja počítačových hier. Má využitie pri vytváraní textúr, modelov a prostredí. Umožňuje efektívne vytvárať množstvá obsahu, tým šetrí čas vývojárom a podstatne znižuje náklady na vývoj. S technologickým pokrokom sa procedurálne generovanie stáva účinným nástrojom a významnou súčasťou budúcich videohier.

V prvej kapitole je predstavená problematika procedurálneho generovania, praktické a subjektívne dôvody prečo zvoliť procedurálne pre vytvorenie hry. V druhej kapitole sú rozobraté metódy generovania obsahu v hrách, ich popis a delenie. Kľúčovým je algoritmus Wave function collapse, ktorý je využitý ako jeden zo spôsobov generovania The Backrooms prostredia v praktickej časti práce. V tretej kapitole venovanej tomuto algoritmu sú opísané dôležité aspekty fungovania algoritmu vrátane jeho rôznych implementačných stratégií a konkrétne možnosti využitia v 3D prostredí. Štvrtá kapitola sa venuje optimalizácii procedurálneho generovania a procesu vykreslovania, s cieľom dosiahnuť plynulejší herný zážitok. Analyzuje techniky a prístupy, ktoré zvyšujú efektivitu generovania obsahu a jeho zobrazenia v reálnom čase. Piata kapitola obsahuje charakteristiku The Backrooms prostredí. Posledná šiesta kapitola rozoberá technológie, algoritmy a princípy použité počas vytvárania hry.

Praktická časť pozostáva z implementácie hry schopnej generovať nekonečné 3D bludisko inšpirované The Backrooms. The Backrooms tvorí sústavu prostredí, ktoré sa skladajú z mnohých levelov, kde každý level má svoju unikátnu charakteristiku a špecifickú atmosféru. Praktická časť je zameraná na Level 37 Poolrooms, ktorý tvorí rozsiahly komplex prepojených miestností a chodieb pripomínajúcich opustené zatopené kúpeľne zariadenie. V programe Unreal Engine boli navrhnuté miestnosti s danou tematikou. Miestnosti slúžia ako základné stavebné bloky. Prostredie hry je generované algoritmom Wave function collapse. Tento algoritmus generuje prostredie na základe vopred definovaných pravidiel. Pravidlá stanovujú usporiadanie miestností v procese generovania. Hráč prechádza prostredím z pohľadu prvej osoby.

1 Procedurálne generovanie v hrách

Procedurálne generovanie predstavuje počítačový software, ktorý dokáže automaticky vytvárať obsah pomocou algoritmov. V počítačových hrách sa používa napríklad na tvorbu levelov, máp, štruktúr, predmetov, pravidiel alebo charakterov. Obsah tvorený procedurálnym generovaním musí spĺňať dizajn a obmedzenia hry, do ktorej sa generuje. Kľúčovou požiadavkou generovaného obsahu je jeho hrateľnosť. Platí, že vygenerované herné levely musí byť možné dokončiť. Software dokáže produkovať obsah sám alebo s pomocou dizajnérov a hráčov. Napríklad v akčnej dobrodružnej hre The Legend of Zelda sa software používa na vytvorenie podzemných lokácií, pričom nepotrebuje žiadny ľudský vstup. Ďalej to môže byť grafický nástroj, pomocou ktorého môže užívateľ dizajnovať mapy pre strategické hry. Software simultálne vyhodnocuje nadizajnovanú mapu a navrhuje úpravy, aby mapa vyzerala lepšie a vyváženejšie. Nakoniec to môže byť systém, ktorý vytvára nové a vylepšené zbrane v strieľacej hre na základe tých, ktoré hráči radi používajú. Niekedy môže byť vytvorenie vhodného procedurálne generovaného obsahu náročné a výsledok môže byť horší v porovnaní s manuálne vytvoreným obsahom. [1, 2]

1.1 Dôvody prečo si vybrať procedurálne generovanie

Text podkapitoly je prevzatý z knihy Procedural Generation in Game Design [1]. Podkapitola sa venuje dôvodom, prečo si vybrať procedurálne generovanie k vytvoreniu hry. Existujú praktické, racionálne, ale aj subjektívne dôvody a motivácie vývojárov, prečo využívať procedurálne generovanie.

Praktické dôvody:

Šetrenie času: Pomocou procedurálneho generovania sa dá vyprodukovať obrovské množstvo obsahu rýchlejšie ako manuálnym procesom. Množstvo geograficky rozsiahlych hier sa spolieha na procedurálne generovanie ich masívnych svetov. Svet sa môže vygenerovať pomocou algoritmov a detaily doladia manuálne dizajnéri alebo sa môže vytvoriť manuálne a následne sa pomocou procedurálnych nástrojov dotvorí obsah, ako sú stromy, tráva a budovy. Niekedy môže proces vyváženia a ladenia procedurálneho systému zabrať viac času než ručné vytvorenie obsahu.

Rozšíriteľnosť: Generátory sú zvyčajne navrhnuté modulárne. Vždy, keď vývojár pridá novú vlastnosť, integruje sa do všetkých výstupov generátora. Každý manuálne navrhnutý detail neovplyvní len jednu položku, ale celú škálu položiek. Z toho vyplýva, že každý detail má nesmierny dopad na celú hru.

Opätovná prehrávateľnosť: Jeden generátor môže vyprodukovať množstvo odlišného obsahu. Túto techniku najčastejšie využívajú hry typu

rogue-like a hry inšpirované týmto žánrom. Algoritmus v hrách roguelike žánru vygeneruje vždy nový náhodný level po prejdení aktuálneho levelu. Je vysoko nepravdepodobné, že by sa hráč stretol s rovnakým levelom, v akom sa už nachádzal. Ak by sa takto vygenerovaný obsah mal vytvárať ručne, bolo by to veľmi nepraktické vzhľadom k potrebe vytvoriť množstvo prípadov.

Znovupoužiteľný kód: Generátory sa môžu opakovane používať medzi aplikáciami. Generátor púštnej krajiny z jednej hry môže mať viaceré premenné prispôsobené tak, aby sa v inej hre vytvorili ľadové hory. Medzi aplikáciami môžeme vymieňať modulárne elementy generátorov. Znovupoužiteľnosť kódu je veľmi dôležitá pri veľkých projektoch.

Uplatňovanie pravidiel: Generátory môžu byť postavené tak, aby dodržiavali vo všetkých výstupoch určité pravidlá, ako je prepojenie elementov alebo ich štandardy. Ak je kód správne navrhnutý a bezchybný je schopný aplikovať pravidlá dôkladnejšie ako ľudský dizajnér. V architektúre sa procedurálne techniky používajú na to, aby sa v nových projektoch dodržiavali štandardy a zákony fyziky. Návrhári hier môžu zahrnúť obmedzenia týkajúce sa vyváženosti obtiažnosti, konektivity, riešiteľnosti hádaniek a mnohé ďalšie dôležité vlastnosti.

Modelovanie reality: V generátoroch sa často používajú simulačné techniky napodobňujúce prirodzený rozvoj života alebo opotrebovanie terénu. Generátory inšpirované biologickými procesmi môžu byť mimoriadne efektívne pri produkcii komplikovaných detailov podobných realite.

Veľké rozsahy a zložité detaily: Generátory sa používajú pri vytváraní zložitých detailov, ako sú fraktály alebo v rozsiahlych hrách, kde treba vytvoriť celé galaxie. No Man's Sky je jednou z hier, ktorú je takmer nemožné vytvoriť ručne. Hra obsahuje kvantilión planét vytvorených procedurálnym generovaním.

Subjektívne dôvody:

Individuálny zážitok: Procedurálne generovanie nám umožňuje vytvárať pre každého hráča unikátny a nezabudnuteľný zážitok. Napríklad prostredníctvom vygenerovaného zvuku, prostredia alebo výziev v hre, s ktorými sa žiadny z hráčov ešte nestretol. Staticky vygenerovaným obsahom by sme efektu nedosiahli. Generovanie je dôležitý nástroj v interaktívnom prostredí, ako sú videohry.

Ovládanie hráčom: Hra Mushroom-11 využíva procedurálny obsah inovatívnym spôsobom. Necháva hráča kontrolovať roj buniek, ktoré sa regenerujú na základe pravidiel bunkových automatov. Vzniká nová forma hrateľnosti, kde hráč manipuluje s obsahom vyprodukovaným procedurálnym generovaním. Hráč si musí osvojiť správanie bunkových automatov, aby dokázal kontrolovať roj a poraziť výzvy hry.

Nepredvídateľnosť: Aký bude výstup generátora, nepozná ani jeho dizajnér. Z hľadiska zabezpečenia kvality môže byť nepredvídateľnosť odstrašujúca, ale na druhú stranu pre tvorcu vzrušujúca. Tvorca si tak môže užiť prácu rovnako ako koncový užívateľ výsledný produkt.

Živý systém: Procedurálnymi technikami môžeme vytvoriť živé efekty ako je oheň, meniace sa počasie, simulovať vývoj civilizácie a života. Ručne vytvorený obsah sa bude vždy javiť ako repetitívny v porovnaní s dynamickým, meniacim sa procedurálnym systémom. Techniky, ako je rozhodovanie na základe najbližšieho prvku, sú skvelé vo vytváraní zložitých a meniacich sa efektov vo veľkom rozsahu. Efekty sa nám javia ako skutočné, pretože sa riadia vzormi existujúcimi v reálnom svete.

Počítačová tvorivosť: Výstupy procedurálneho generovania môžu byť niekedy bizarné. Generátory vytvárajú veci, na ktoré by človek nikdy nepomyslel, od nezvyčajne štruktúrovaného prostredia hry až po nereálne animácie. Počítače neuvažujú o veciach rovnakým spôsobom, ako ľudia a niekedy prichádzajú s logickými extrémami, ktoré sú úplne mimo rámca našej predstavivosti.

1.2 Taxonómia

Vzhľadom na rôznorodosť problémov a metód tvorby obsahu, ktoré sú v súčasnosti k dispozícii, je dôležité mať štruktúru, ktorá zdôrazňuje rozdiely a podobnosti medzi prístupmi.

Prístupy k tvorbe obsahu

Procedurálne generovanie môžeme využiť na vygenerovanie obsahu online alebo offline. Online generovanie vytvára obsah prispôsobený hráčovi počas hrania hry. Offline generovanie je užitočné pre vytváranie komplexného obsahu, ako sú prostredia a mapy. Používa sa počas vývoja hry alebo pred začiatkom hernej relácie. Ďalším dôležitým využitím je tvorba a zdieľanie vygenerovaného obsahu. Hra Spore má editor bytostí, ktorý umožňuje hráčom upravovať a nahrávať bytosti alebo úrovne na server, z ktorého si ich môžu stiahnuť a použiť ostatní hráči. Online generovanie je použité v akčnej hre Warframe, ktorá má dynamické generovanie herných levelov.

Procedurálnym generovaním je možné vygenerovať herný obsah nevyhnutný pre dokončenie levelu a pomocný alebo voliteľný obsah, ktorý sa môže neskôr zahodiť. Hlavným rozlišovacím znakom medzi nevyhnutným a voliteľným obsahom je, že nevyhnutný obsah by mal byť vždy vygenerovaný správne. Podmienka nemusí platiť pre voliteľný obsah. Príkladom voliteľného obsahu je generovanie rôznych typov zbraní alebo zberateľských predmetov. Nevyhnutným obsahom môže byť štruktúra úrovní alebo predmetov potrebných na prechod do ďalšej úrovne. [2]

Procedurálne generovanie obsahu sa riadi dvoma spôsobmi. Jedným spôsobom je použitie náhodných, kladných čísel predstavujúcich štartovací bod pre generovací algoritmus, takzvané seed číslo. Druhý spôsob je využitie sady parametrov, ktoré upravujú generovaný obsah. Seed čísla sa používajú pri generovaní sveta v hre Minecraft (viď kap. 1.3). Pri použití rovnakého čísla sa znovu vygeneruje rovnaký svet.

Procedurálne generovanie má dva prístupy k tvorbe obsahu, generický a adaptívny. Generický obsah sa generuje bez ohľadu na správanie hráčov. Adaptívny obsah je personalizovaný na základe analýzy hráčovej interakcie s hrou. V hre Left 4 Dead sa používa adaptívne generovanie. Algoritmus dynamicky upravuje tempo hry v reálnom čase na základe analýzy správania hráčov. V tomto prípade sa adaptívne generovanie používa na úpravu náročnosti hry.

Deterministickým procedurálnym generovaním dokážeme vygenerovať rovnaký obsah, pri použití rovnakého východiskového bodu a parametrov v generátore. Vytvorenie rovnakého obsahu nie je možné stochastickým generovaním. Deterministicky sa generujú napríklad galaxie v hre Elite.

Konštruktívne generovanie vytvorí obsah bez možnosti jeho následnej modifikácie. Protikladom konštruktívneho generovania je metóda generovania a testovania, ktorá cyklicky generuje a výsledok testuje, kým nenájde vyhovujúce riešenie.

Automatické generovanie nepotrebuje dodatočné vstupy pri vytváraní obsahu. Obsah je regulovateľný upravovaním parametrov generovacieho algoritmu. Interaktívne generovanie zapája dizajnéra, alebo hráča do procesu navrhovania, v ktorom spolupracujú s algoritmom pri generovaní požadovaného obsahu. Tanagra je systém, v ktorom si dizajnér nakreslí časť 2D levelu a algoritmus mu vygeneruje chýbajúce časti tak, aby zostala zachovaná hrateľnosť. Ďalej interaktívne generovanie využíva procedurálny framework SketchaWorld na kreslenie náčrtov krajín a scenérií. Framework vytvorí z náčrtu 3D model, ktorý udržuje v konzistentnom stave počas dodatočných manuálnych úprav. [2]

1.3 Hry používajúce procedurálne generovanie

Jedným z dôvodov vývoja procedurálneho generovania v hrách, bola v minulosti obmedzená pamäť počítačov. Limitované hardvérové možnosti domácich počítačov na začiatku 80. rokov obmedzovali priestor dostupný na uloženie herného obsahu. Preto boli vývojári nútení hľadať metódy, ako efektívnejšie vytvoriť a uložiť obsah. [3]

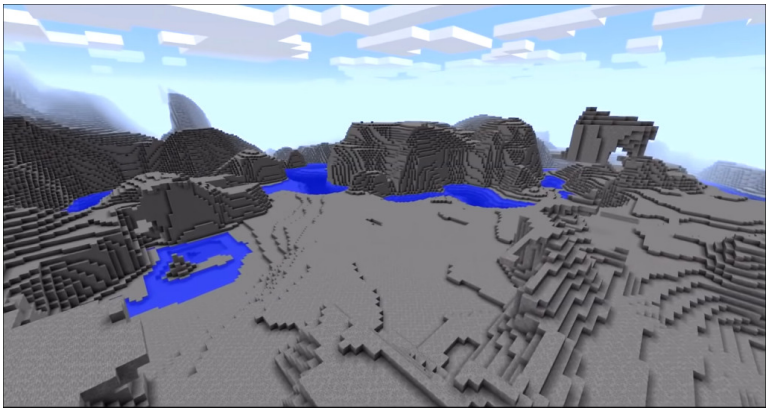
Diablo 1

Diablo je akčná roleplay hra z roku 1996. Odohráva sa v temnom fantasy svete, v ktorom má hráč za úlohu poraziť démona Diabla, ukrývajúceho sa v katakombách pod mestom Tristram. Prostredie hry je vytvorené na izometrickej dlaždicovej mape, ktorá vytvára efekt 3D sveta pomocou 2D spriteov. Hra je

rozdelená do štyroch etáp, katedrála, katakomby, jaskyne a peklo. Každá etapa obsahuje štyri úrovne a generovanie je prispôsobené ich tematike. Úrovne hry sú generované tak, aby zaplnili mriežku o veľkosti 40×40 políčok. Generovanie je rozdelené na dve fázy. V prvej fáze sa vytvára základná štruktúra úrovne, kde sa určujú prechodné miesta, veľkosť miestností a umiestnenie vstupov. V druhej fáze sa zo základnej štruktúry vygeneruje konečný vzhľad úrovne dosadením textúr. [4]

Minecraft

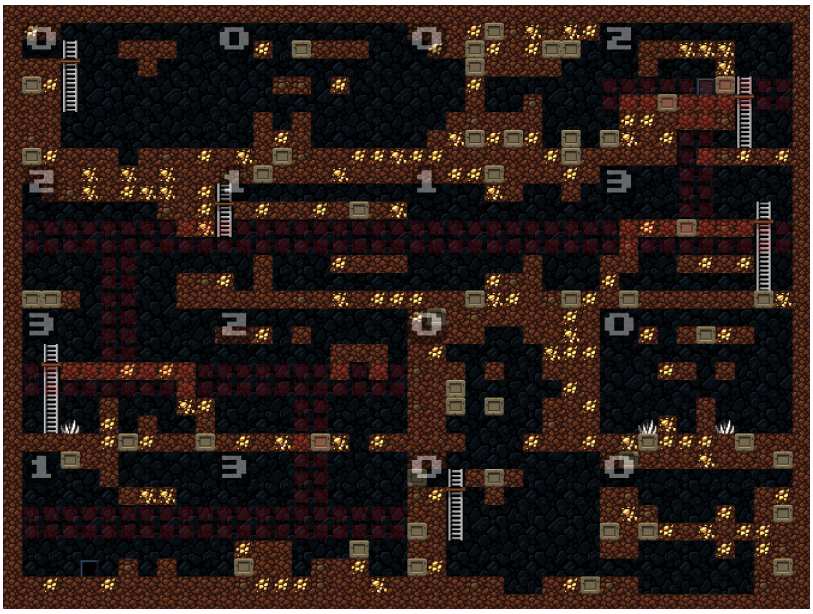
Minecraft je sandboxová hra, kde si hráči môžu vybudovať čokoľvek vo svete vytvorenom z blokov. Vzhľad vygenerovaného prostredia záleží na použitom seed čísle, ktoré sa používa pre výpočet všetkých blokov. Hra využíva algoritmy gradientového šumu, ako je Perlinov šum (viď kap. 2.1). Tým sa zabezpečí, že bloky a časti sveta sú v súlade so susednými blokmi, čo dodáva svetu kontinuitu a náhodnosť. Minecraft prostredie je zložené zo segmentov, ktoré sa postupne generujú. Segment má rozlohu $16 \times 16 \times 256$ blokov. V prvom kroku sa generuje topografická mapa terénu, zložená zo skál a vody (viď obr. 1). Potom sa terén postupne vyhladí a pridajú sa detaily, ktoré závisia od typu biomu. Generovanie biomu je založené na 6 parametroch, teplota, vlhkosť (množstvo vegetácie), kontinentalita, erózia, zvláštnosť a hĺbka. Okrem hĺbky sú parametre založené len na horizontálnych súradniciach. Výsledkom je kompletná topografia vytvorená z kameňa, vody a vzduchu. V ďalšom kroku sa do sveta pridajú bloky predstavujúce trávu, hlinu a piesok. Tieto bloky prepisujú už existujúce prostredie vytvorené z kameňa. Algoritmus berie do úvahy, že na púšti má prevahu piesok, v oceáne je viac štrku a podobne. Každý biom je zložený z vhodných kombinácií materiálov pre danú krajinu. Ako posledné sa vytvárajú jaskyne a rokliny. Priestory jaskýň sú generované algoritmom náhodnej chôdze. Do jaskýň sa pridajú ďalšie elementy ako meď, uhlie, zlato a ďalšie suroviny. Posledným krokom na dokončenia sveta je pridanie dekorácií, flóry a fauny. Umiestnenie objektov závisí na základe konkrétnych distribučných pravidiel. [5]



Obr. 1: Základná vygenerovaná topografia sveta. Prevzaté z [5].

Spelunky

Spelunky je 2D platformová indie hra rogue štýlu. Procedurálne generovanie používa na vytvorenie variácií herných levelov. Level sa skladá z 16 miestností umiestnených v mriežke o rozmeroch 4×4. Hra má štyri základné typy miestností, z ktorých generuje (viď obr. 2).



Obr. 2: Ukážka levelu z hry Spelunky pred vygenerovaním pascí, nepriateľov a zberateľských predmetov. Obsahuje 4 základné typy miestností. Miestnosť typu 0 je vedľajšia miestnosť. Miestnosť typu 1 má garantovaný prechod z ľavej a pravej strany. Miestnosť typu 2 má prechody z ľavej, pravej a spodnej strany. Ak je nad ňou ďalšia miestnosť typu 2, má garantovaný prechod aj zhora. Miestnosť typu 3 má prechody z ľavej, pravej a hornej strany. Obrázok vygenerovaný nástrojom [6].

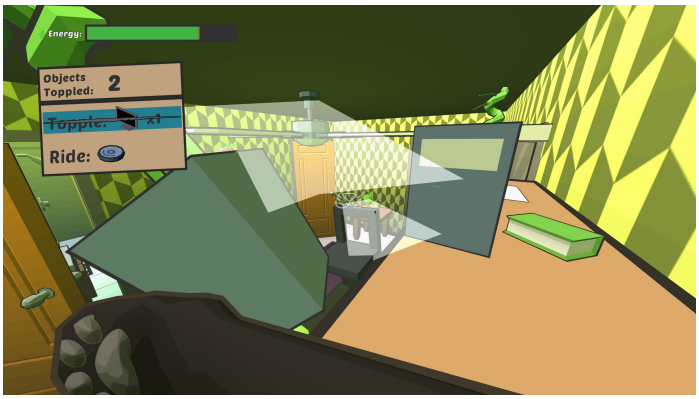
Algoritmus začína umiestnením miestnosti typu 1 alebo 2 do horného radu mriežky. Pri každom ďalšom umiestnení algoritmus použije miestnosť typu 1. Potom náhodne rozhodne o smere pokračovania, ak pôjde doľava alebo doprava, aktuálna miestnosť zostáva typu 1. Ak pôjde nadol, aktuálna miestnosť sa zmení na typ 2. Ak narazí na okraj mriežky, posunie sa nadol a zmení smer pohybu zľava doprava alebo opačne. Ak bola predchádzajúca miestnosť typu 2, aktuálna miestnosť musí byť buď typ 2 alebo typ 3. Ak algoritmus skončí v spodnom rade mriežky a vyberie smer nadol, umiestni miestnosť s východom. Nakoniec pridá náhodné miestnosti typu 0 do voľných miest v mriežke. Tieto miestnosti nemajú žiadne zaručené prechody a môžu byť uzavreté. Každý typ miestnosti má 8 až 16 šablón, ktoré si algoritmus náhodne vyberá (viď obr. 3). Označenia predstavujú bloky a ich pravdepodobnosti výskytu v hre. Bloky môžu byť statické, pravdepodobnostné alebo prekážkové. Umiestnenie zberateľských predmetov a nepriateľov závisí od počtu a rozmiestnenia blokov v ich okolí. [6]

```
1 1 0 0 0 0 0 0 0 0
4 0 L 5 0 0 0 0 0 0
1 1 P 0 0 0 0 0 0 0
1 1 L 0 0 0 0 0 0 0
1 L 1 5 0 0 0 0 0 0
1 1 0 0 0 0 0 0 0 0
1 1 0 0 0 0 0 0 0 0
1 1 1 1 1 1 1 1 1 1
```

Obr. 3: Vzor šablóny hry Spelunky. Šablóny s rozmermi 10×8 určujú rozloženie blokov v miestnosti. Prevzaté z [6].

Catlateral Damage

Catlateral Damage je oddychová 3D hra, v ktorej je hráč v roli mačky. Cieľom je zhodiť čo najviac predmetov z políc (viď obr. 4). Hra má niekoľko úrovní a úloh, ktoré sú procedurálne generované. Úrovne sa odohrávajú v interiéroch rôznych budov. Na začiatku je budova prázdna. Najprv sa vygenerujú prázdne miestnosti a následne ich zariadenie. Ďalej sa vygenerujú objekty, s ktorými môže hráč interagovať. Na základe vygenerovaných objektov sa určí úloha a stanoví časový limit na jej splnenie. Hra má pre každú úroveň uložený dátový súbor, ktorý špecifikuje veľkosť, nábytok, textúry a pravidlá, podľa ktorých sa bude generovať. Generátor pre rozdelenie budov používa algoritmus Squarified Treemaps. Algoritmus rozdeľuje plochu obdĺžnika na menšie obdĺžniky tak, aby najviac pripomínali štvorce. Nábytok obsahuje kolízny prvok, ktorý simuluje fyzikálne interakcie medzi objektmi a neviditeľného ohraničujúceho boxu, ktorý slúži ako bariéra zabraňujúca prekrývaniu objektov. Ďalej je k povrchu pripojený skript obsahujúci zoznam objektov, ktoré je naň možné umiestniť, spolu s informáciami o ich pravdepodobnosti a rotácii. [1]

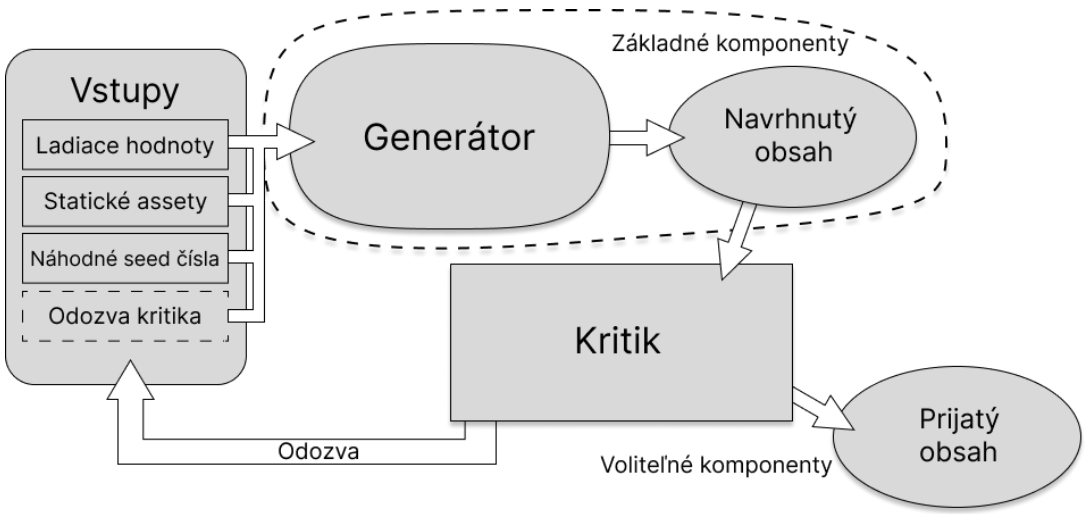


Obr. 4: Ukážka z hry Catlateral Damage. Prevzaté z [7]

2 Metódy

Aplikácia procedurálneho generovania obsahu v oblasti vývoja hier pomáha vývojárom uplatniť ich kreativitu, šetriť úložný priestor a minimalizovať potrebu pracovnej sily na vývoj. Existuje množstvo metód na generovanie obsahu pre hry. Tieto metódy sa delia na konštruktívne, vyhľadávacie, reštriktívne a metódy založené na strojovom učení. Konštruktívne metódy obvykle pracujú v krátkom konštantnom čase, pričom vygenerovaný výstup nie je hodnotený ani znovu generovaný. Metódy založené na vyhľadávaní sa sústredia na nájdenie obsahu, ktorý najlepšie vyhovuje určitým požiadavkám. Reštriktívne metódy aplikujú na generovaný obsah rôzne pravidlá, zatiaľ čo metódy strojového učenia využívajú modely na analýzu vzorového obsahu, a na jeho základe vytvárajú nový obsah. [2, 10]

Uvedené metódy vyžadujú pre svoju funkčnosť vstupné hodnoty, ako sú napríklad ladiace parametre a statické prostriedky (napr. 3D modely, segmenty príbehu, audio vzorky). Základný komponent každej generovacej metódy je generátor. Generátor môže využiť pri tvorbe obsahu viaceré metódy. Na kontrolu obsahu navrhnutého generátorom slúži komponent nazývaný kritik. Kritik generátora, ktorý vytvára úrovne hier, môže do návrhu generátora integrovať modely hráča vygenerované pomocou rôznych metód (viď obr. 5). Kritik môže výstup generátora prijať, zamietnuť, opraviť alebo mu poskytnúť odozvu. Zložitosť kritikov záleží od ich implementácie, od jednoduchého detektora kolízií až po heuristický optimalizátor, ktorý je schopný vrátiť vylepšenú verziu obsahu. Kritikom môže byť softvérový nástroj alebo človek. Ľudský kritik je schopný, na rozdiel od softvéru, subjektívne hodnotiť navrhnutý obsah. Softvérovým kritikom môžu byť neurónové siete alebo evolučné algoritmy, ktoré hodnotia vyváženosť, obtiažnosť alebo originalitu generovaného obsahu. [8]



Obr. 5: Schéma generovacieho procesu a jeho komponentov. Upravené z [8].

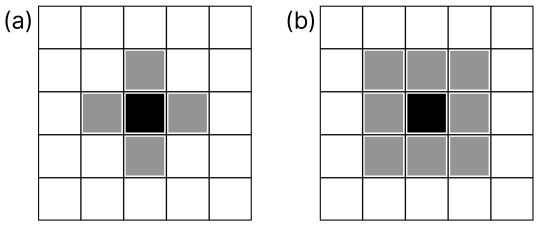
2.1 Konštruktívne metódy

Konštruktívne metódy náhodne spájajú a umiestňujú vopred vytvorené časti obsahu. Charakteristickým znakom konštruktívnych metód je, že vygenerovaný obsah nevyhodnocujú, a tým nedochádza k jeho opätovnému generovaniu. Vzhľadom na toto obmedzenie by mal algoritmus počas konštrukcie zaistiť, aby mal obsah požadované vlastnosti. Pri každom spustení vygenerujú len jeden výstup, na rozdiel od vyhľadávacích metód (viď kap. 2.2). Keďže generovaný obsah neposudzujú ani neopravujú, sú rýchle a preto s obľubou používané v hernom priemysle. Nevýhodou ich rýchlosti je však obmedzená flexibilita. Nie všetky algoritmy dokážu vždy vytvoriť použiteľný obsah, preto sa dodatočne používajú techniky na opravu vygenerovaného obsahu.

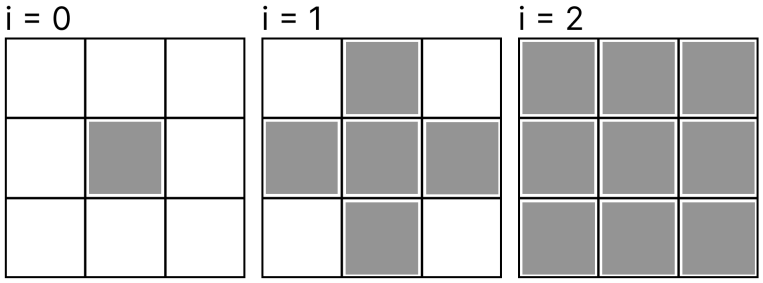
Konštruktívne metódy sa používajú pre špecifické účely, ako napríklad tvorba levelov pre konkrétnu hru, čo obmedzuje ich použitie na iné projekty. Metódy umožňujú generovanie herného obsahu v reálnom čase. Príkladom použitia môže byť generovanie terénu v hre Minecraft, kde sa terén vytvára dynamicky, zatiaľ čo hráči preskúmavajú svet. V konštruktívnom procedurálnom generovaní sa často uprednostňujú metódy, ktoré sú dobre ovládateľné a predvídateľné, ako napríklad metóda binárneho rozdeľovania, náhodná chôdza, Perlinov šum, Wave function collapse (viď kap. 3) a Bunkové automaty. [11]

Bunkový automat

Bunkový automat (angl. cellular automata) pozostáva z n-dimenzionálnej mriežky, ktorej bunky podľa stanovených pravidiel nadobúdajú jeden z konečného počtu stavov. Pravidlá definujú, ako má bunka zmeniť svoj stav na základe stavov priľahlých buniek vyplývajúcich z definície okolia. Automat pracuje v konečnom počte krokov, pričom v každom kroku, bunky aktualizujú svoj stav podľa pravidiel (viď obr 7). Okolie bunky je možné definovať dvoma základnými spôsobmi, a to von Neumannovým a Mooreovým okolím (viď obr. 6). Existujú rôzne rozšírenia týchto základných spôsobov, ktoré za susediace bunky považujú širšie okolie buniek. [38]



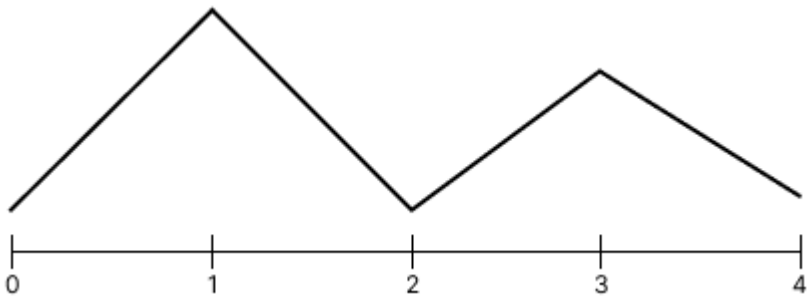
Obr. 6: Schéma zobrazuje okolie čiernej bunky na 2D mriežke (a) von Neumannové okolie, (b) Moorové okolie. Von Neumannové okolie pozostáva zo štyroch priľahlých buniek. V Moorovom okolí sú zahrnuté aj štyri najbližšie susedné bunky na hlavnej a vedľajšej diagonále. Upravené z [38].



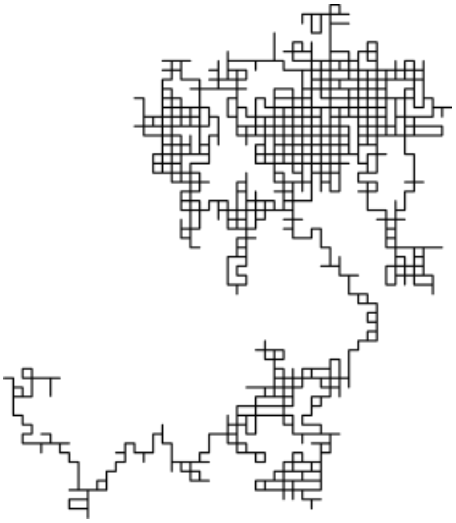
Obr. 7: Simulácia jednoduchého 2D bunkového automatu s mriežkou o veľkosti 3×3 v troch krokoch. Bunky môžu byť v jednom z dvoch stavov, aktívna a neaktívna. Aktívna bunka predstavuje v mriežke zašednuté políčko. Automat má definované jedno pravidlo, kde každá bunka, ktorá má aspoň jednu aktívnu susediacu bunku sa zmení na aktívnu. Bunky dodržiavajú pravidlá na základe von Neumannového okolia. Kroky sú označené indexom i, kde $i = 0$ pre počiatočný stav automatu. V počiatočnom stave máme v mriežke jednu aktívnu bunku. Nasledujúce dva kroky zobrazujú aktualizáciu buniek voči pravidlu. Upravené z [39].

Náhodná chôdza

Náhodná chôdza (angl. random walk) sa používa pri vytváraní podzemných systémov, trás alebo terénu. Algoritmus náhodne vyberie štartovací bod, z ktorého sa následne pokračuje o jeden krok do náhodne vybraného smeru (viď obr. 8 a obr. 9). Algoritmus skončí, keď vykoná požadovaný počet krokov.



Obr. 8: Schéma znázorňuje náhodnú chôdzu v 1D priestore. Osa zobrazuje počet vykonaných krokov a krivka spája hodnoty predstavujúce zvolený smer v danom kroku. Pri každom kroku sa posúva v priestore náhodne nahor alebo nadol. Náhodná chôdza sa v 1D priestore používa napríklad pri generovaní sekvencií kopcov a údolí pre platformové hry. Upravené z [1].



Obr. 9: Schéma zobrazuje 2D náhodnú chôdzu po 100 krokoch. 2D náhodná chôdza začína v ľubovoľnom bode, z ktorého sa vykoná krok do náhodne vybraného smeru. Prevzaté z [1].

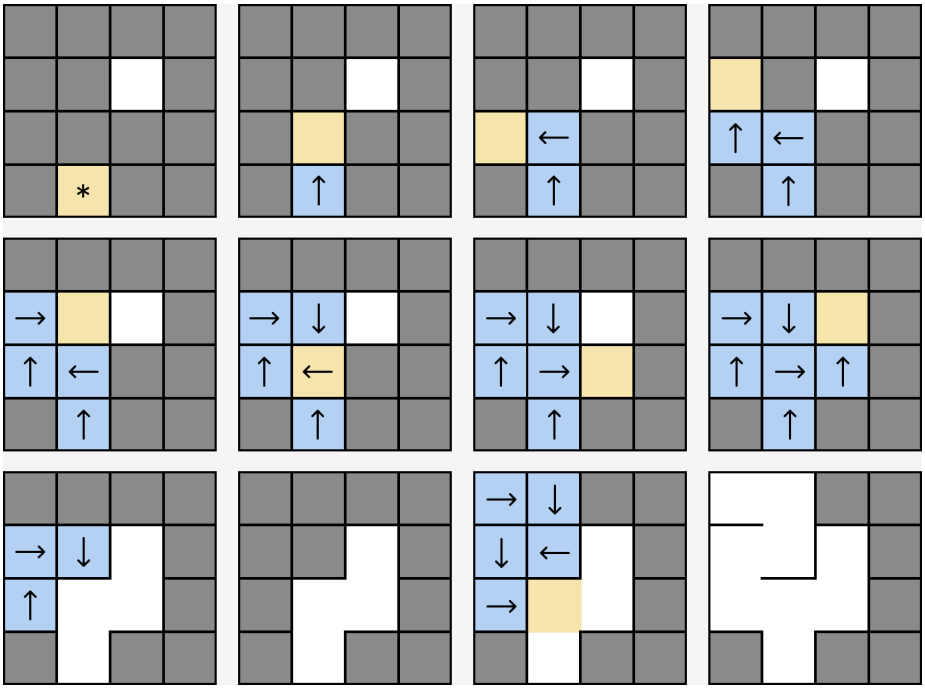
Náhodná chôdza nemusí vždy vygenerovať správny výsledok, aj ak sú definované pravidlá, pretože jej náhodnosť spôsobuje, že nemusia byť splnené. Nesprávne výsledky budú odstránené a algoritmus sa spustí znovu, v inom prípade sa implementuje backtracking, prípadne bude použitý zložitejší algoritmus, ako je napríklad Wilsonov alebo Kruskalov algoritmus. Nasledujúce algoritmy sú opísané pomocou pojmov z teórie grafov. Mriežku môžeme chápať ako špeciálny prípad grafu, kde jednotlivé bunky mriežky sú vrcholy, ktoré sú spojené hranami s ich priľahlými bunkami. [1]

Perlinov šum

Perlinov šum bol vytvorený pre procedurálne generovanie textúr, terénu a efektov. Generuje postupnosť čísel, kde rozdiely medzi susednými hodnotami sú malé a rovnomerné, na rozdiel od náhodnej postupnosti, kde čísla nemajú žiadny vzájomný vzťah. Perlinov šum očakáva na vstupe bod, a ako výstup vracia hodnotu z intervalu 0 až 1. Algoritmus najprv určí súradnice daného bodu v jednotkovom štvorci (kocke). Potom každému vrcholu štvorca priradí pseudonáhodný gradientný vektor. Gradientný vektor ukazuje smer najrýchlejšej zmeny hodnoty. Následne vypočíta vektory vzdialeností od daného bodu k vrcholom štvorca. Pre každý vrchol štvorca potom vypočíta skalárny súčin gradientného vektora a vektora vzdialeností. Na záver tieto skalárne súčiny interpoluje, čím získa vážený priemer medzi bodmi štvorca. Plynulosť prechodov medzi hodnotami závisí od použitej interpolačnej funkcie. Perlinov šum sa používa v 2D, interpretovaním hodnôt šumu ako výšky, napríklad pri vytváraní hornatého terénu. V 3D umožňuje vytváranie dynamických textúr, ako napríklad morských vĺn alebo oblakov. [9]

Wilsonov algoritmus

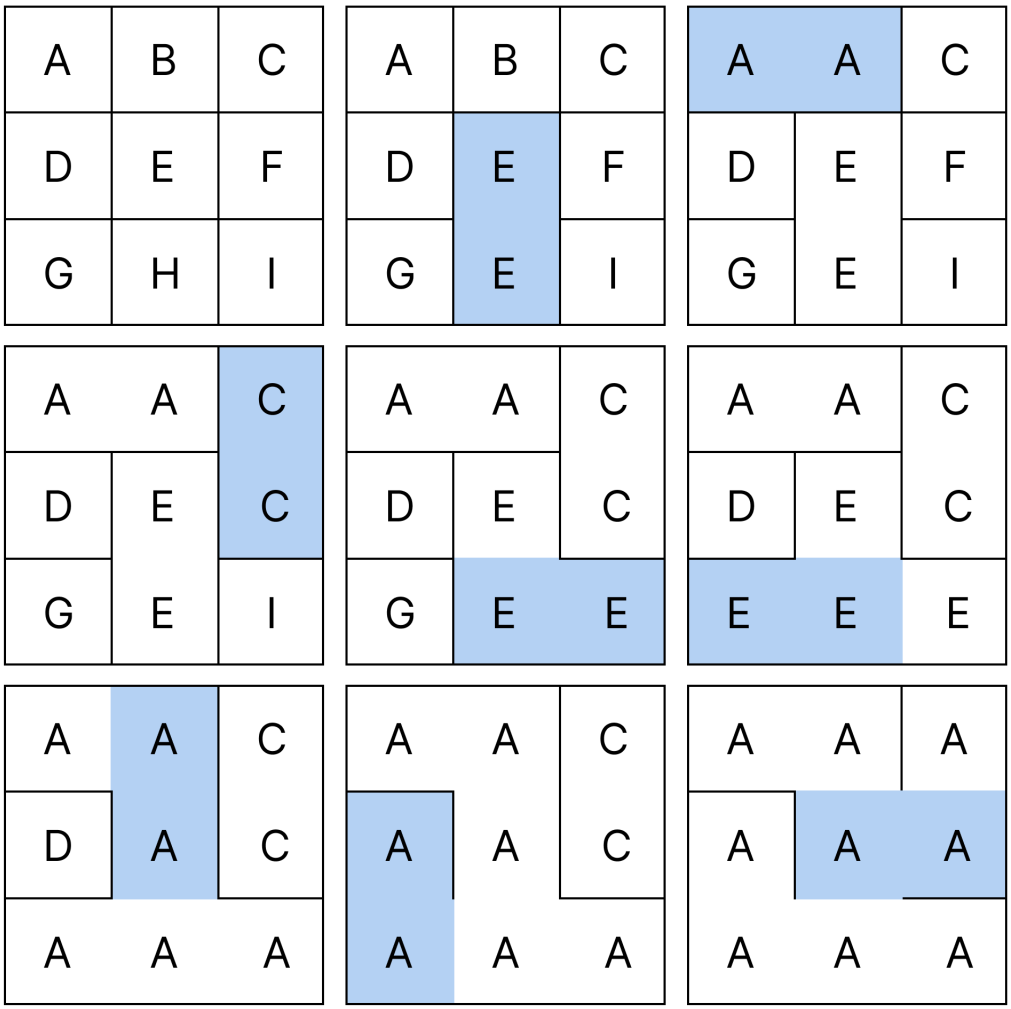
Wilsonov algoritmus sa používa pri tvorbe náhodných dokonalých bludísk (viď obr. 10). V dokonalom bludisku neexistuje cyklus medzi žiadnymi dvomi bunkami mriežky. Algoritmus hľadá uniformnú kostru grafu. Uniformná kostra je náhodne skonštruovaná z množiny všetkých kostier grafu. [40]



Obr. 10: Príklad Wilsonovho algoritmu na mriežke o rozmeroch 4×4. Na začiatku algoritmus vyberie náhodnú bunku a označí ju ako aktívnu (biela). Aktívne bunky sú súčasťou existujúceho bludiska. Ďalej vyberie doposiaľ neaktívnu bunku (šedá), označí ju ako počiatočnú bunku (*), z ktorej začne náhodnú chôdzu. Každá bunka, ktorú náhodná chôdza prešla (modrá) si pamätá posledný smer zvolený pri presune do ďalšej bunky. Keď algoritmus narazí na aktívnu bunku, vráti sa na počiatočnú bunku. Následne prechádza bunky podľa posledných zapamätaných smerov a každú označí ako aktívnu. Zvyšné neaktívne bunky, ktoré náhodná chôdza prešla budú zresetované. Algoritmus vykonáva náhodnú chôdzu, dokým nebudú všetky bunky označené ako aktívne. Cesty vytvorené náhodnou chôdzou predstavujú vo výslednom bludisku samostatné chodby. Prevzaté z [40].

Kruskalov algoritmus

Kruskalov algoritmus hľadá minimálnu kostru súvislého, hranovo ohodnoteného, neorientovaného grafu. Algoritmus konštruuje minimálnu kostru grafu postupným pridávaním hrán s najmenším ohodnotením tak, aby nevznikol cyklus. Algoritmus zastaví, keď vybrané hrany prepoja všetky vrcholy grafu. Upravením algoritmu, aby hrany vyberal náhodne, vznikne generátor dokonalých bludísk (viď obr. 11). [40]



Obr. 11: Príklad Kruskalovho algoritmu generujúceho bludiská na mriežke o rozmeroch 3×3. Každá bunka mriežky je označená písmenom, ktoré reprezentuje množinu, do ktorej patrí. Algoritmus náhodne vyberá hrany medzi dvomi bunkami odlišných množín. Bunky prepojené zvolenou hranou následne spojí spolu s ich množinami. Algoritmus skončí, keď všetky bunky mriežky budú patriť do jednej množiny. Hrany, ktoré neboli zvolené, budú vo výslednom bludisku predstavovať steny. Prevzaté z [40].

2.2 Vyhľadávacie metódy

Vyhľadávacie metódy v procedurálnom generovaní hľadajú možné riešenia (kandidátov), ktoré spĺňajú určité kvalitatívne kritéria. Pri vyhľadávaní sa používajú evolučné či optimalizačné algoritmy. Vyhľadávacím priestorom rozumieme súbor všetkých možných kandidátov, ktorých môže algoritmus počas generovania dosiahnuť, prostredníctvom zmien akejkoľvek inštancie v priestore. Aby bol generátor úspešný, tak by zmeny kandidátov nemali byť radikálne, vzhľadom na funkcionalitu alebo výzor generovaného obsahu. Zmeny by mali zachovávať základnú identitu a rozpoznateľné charakteristiky objektu, čím zabezpečujú kontinuitu a koherenciu generovaného obsahu. Napríklad, malá modifikácia v parametroch by mala spôsobiť, že objekt zmení svoju veľkosť alebo tvar v rozumnej miere. Nemala by však zmeniť objekt natoľko, že by z pôvodného predmetu (napr. dom), vznikol úplne odlišný objekt (napr. strom). [31]

Vyhľadávacie metódy sú špeciálnym prípadom metód generovania a testovania. Evaluačná funkcia hodnotí vlastnosti kandidátov, tak že priradí každému číselnú hodnotu, ktorá predstavuje vhodnosť skúmanej vlastnosti. Pri kombinácii viacerých evaluačných funkcií, kde každá hodnotí odlišnú vlastnosť, je výstupom vektor reálnych čísel. V prípade generovania herných úrovní by testovanou vlastnosťou mohla byť hrateľnosť. Ďalšou dôležitou vlastnosťou vyhľadávacích metód je spôsob, akým sa hľadajú noví kandidáti. Ten spočíva v modifikácii parametrov existujúcich kandidátov, pričom sa ponechávajú len tie zmeny, ktoré kandidátov zlepšujú. [32]

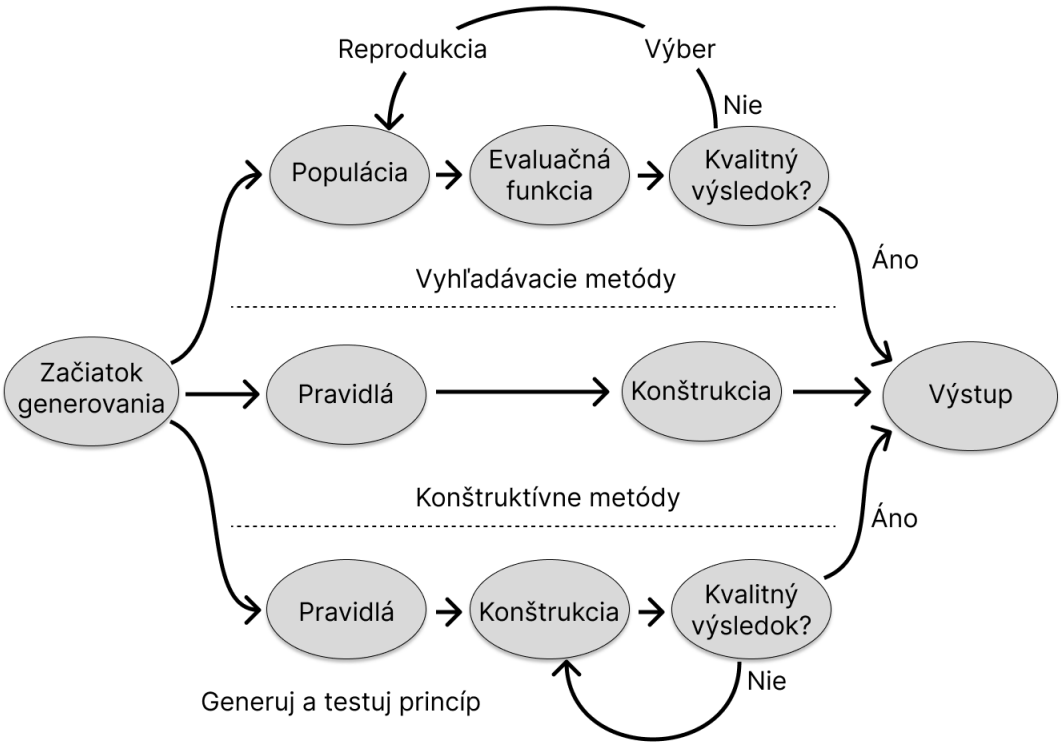
Evolučný algoritmus

Evolučný algoritmus je stochasticky vyhľadávací algoritmus voľne inšpirovaný Darwinovou evolúciou s prirodzeným výberom. Hlavnou myšlienkou je z každej generácie populácie zachovať kandidátov, ktorí sú na základe evaluačnej funkcie hodnotení ako najlepší. Algoritmus ostatných kandidátov z populácie eliminuje. Vybraní kandidáti sú reprodukovaní a ich vlastnosti sú adekvátne upravené. Generovanie môžeme rozdeliť na fázu výberu a reprodukcie (viď obr. 12). Evolučné algoritmy sú celkom spoľahlivým nástrojom na generovanie obsahu, ale v niektorých prípadoch môžu byť príliš zložité a neefektívne. V prípade, že je vyhľadávací priestor malý môžeme evolučný algoritmus nahradiť metódou exhausívneho vyhľadávania. Táto metóda systematicky prechádza všetky možné riešenia v danom vyhľadávacom priestore a hľadá optimálne riešenie. [2]

Náhodné vyhľadávanie

Metóda náhodného vyhľadávania náhodne generuje populáciu z počiatočnej vzorky. Populáciu vyhodnotí evaluačnou funkciou. Algoritmus ukladá najlepšieho kandidáta z danej populácie, ak nájde lepšieho, tak kandidáta aktualizuje. Algoritmus po niekoľkých iteráciách skončí. Je to rýchla a jednoduchá metóda, ale nie je zaručené, že nájde optimálneho kandidáta. Používa sa v prípadoch, keď

je dôležité zachovať rozmanitosť vygenerovaného obsahu. Náhodné vyhľadávanie umožňuje preskúmať iné oblasti vyhľadávacieho priestoru a nájsť tak globálne optimum. Umožňuje algoritmu uniknúť z lokálnych optím čím zabraňuje jeho stagnácii a zvyšuje šancu na nájdenie globálneho optima. Lokálne optimum je riešenie, ktoré je v porovnaní so svojimi susedmi optimálne, ale nie je optimálne v porovnaní s celým vyhľadávacím priestorom. Náhodná mutácia ďalej rozširuje možnosti náhodného vyhľadávania. Spočíva v náhodnej modifikácii parametrov existujúcich kandidátov, čím sa vytvárajú variácie s lepšou kvalitou. Cieľom je vyhnúť sa stagnácii a objavovať nové oblasti vyhľadávacieho priestoru. [2]



Obr. 12: Schéma generovacieho procesu vyhľadávacích a konštruktívnych metód. Upravené z [32].

2.3 Reštriktívne metódy

Reštriktívne metódy sú založené na pravidlách a využívajú deklaratívny prístup k tvorbe obsahu. Väčší dôraz kladú na definovanie cieľov a splnenie požiadaviek vo výslednom obsahu, ako na postup generovania. Metódy majú preddefinované pravidlá, ktoré musí generovaný obsah počas celého procesu konštrukcie dodržiavať. Vstup pozostáva zo súboru premenných alebo slotov. Podľa pravidiel definujúcich vzťahy medzi slotmi sa počas konštrukcie obsahu, dosádzajú ich príslušné hodnoty. Generovanie začína so všeobecnou definíciou obsahu a postupne ju konkretizuje prostredníctvom aplikácie obmedzení. [21]

Pravidlá

Metódy sa používajú napríklad pri generovaní interiérov, kde sémantické pravidlá určujú rozloženie nábytku a zariadenia. Numerické pravidlá môžu byť použité pri generovaní umiestnenia plošín a iných objektov v plošinových hrách. Na generovanie úloh a príbehov sa používajú rôzne gramatiky. Každý typ pravidiel umožňuje generovať iný druh obsahu, čím sa zvyšuje flexibilita a výkonnosť metód, založených na pravidlách pri riešení rozmanitých problémov generovania. Vygenerovanie herného obsahu spĺňajúceho určité pravidlá môžeme chápať, ako problém splnenia obmedzení (angl. constraint satisfaction problem). Správne definovanie všetkých pravidiel pre vytvorenie požadovaného obsahu je obtiažne. Ak výsledok generovania nespĺňa požadované očakávania, pridajú sa ďalšie pravidlá. Dôležité je však zachovať ich rozumný počet, aby generovanie nebolo príliž komplikované a výpočetne náročné. Komplexnosť metód sa mení v závislosti od rozsahu generovaného obsahu, jeho popisu a počtu pravidiel. Vhodné pridanie pravidiel dokáže zefektívniť proces, elimináciou nepotrebných častí generovacieho priestoru. Reštriktívne metódy zvyčajne generujú obsah presne podľa popisu, ale s nepredvídateľným časovým behom. [21, 26]

2.4 Metódy strojového učenia

Metódy strojového učenia generujú obsah pomocou modelov, ktoré majú generovanie nacvičené na existujúcom obsahu. V strojovom učení je obsah generovaný priamo z modelu, na rozdiel od vyhľadávacích metód, ktoré tiež môžu využívať modely strojového učenia (neurónové siete) na hodnotenie obsahu. Výsledkom modelu strojového učenia je samostatne interpretovaný obsah. Pri trénovaní modelov na generovanie obsahu sa používajú rôzne algoritmy, ako napríklad neurónové siete, pravdepodobnostné modely a rozhodovacie stromy. Modely môžu generovať obsah čiastočne alebo úplne. Ďalej ich delíme na autonómne, interaktívne alebo riadené. Pomocou strojového učenia môžu vyvojári vytvárať sady referenčného obsahu a použiť ho ako vzor pre generátor, ktorý je schopný vytvárať nový, štýlovo podobný obsah. Procedurálne generovanie pomocou strojového učenia odstraňuje komplikácie spojené s programovaním vlastných generovacích metód. [10]

Využitie

Metódy sa uplatňujú v autonómnom generovaní, spolugenerovaní a v kompresii dát. Vďaka ich schopnosti rozpoznať charakter obsahu, ich môžeme využiť na opravu, spätnú väzbu a analýzu nového obsahu. Pri manuálnom vytváraní obsahu slúžia metódy strojového učenia ako asistent vývojára a obsah automaticky dogenerujú. Použitím referenčného obsahu môžu algoritmy napríklad identifikovať nehrateľné oblasti herného sveta a ponúknuť návrhy na ich zlepšenie. Ďalej môžeme tieto metódy využiť pri kompresii dát, na identifikáciu opakujúcich sa vzorov v obsahu a následne uložiť iba ich odlišnosti. [10]

3 Wave function collapse

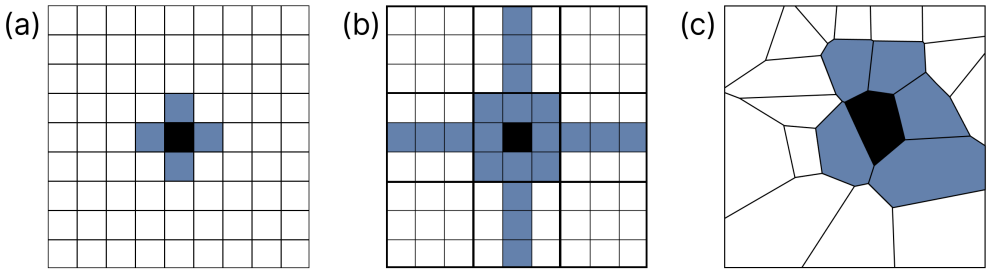
Wave function collapse (Kolaps vlnovej funkcie, WFC) je konštruktívno-reštriktívny algoritmus generujúci obsah na základe vopred definovaných pravidiel a vzorov dostupných zo vstupu. Vzorom môže byť ľubovoľný 2D alebo 3D objekt, podľa typu generovaného obsahu. Algoritmus je inšpirovaný princípom kvantovej superpozície, kde sa nepozorované častice nachádzajú v akomkoľvek možnom stave. Postupným zachytením častíc v určitom stave všetky ostatné stavy zanikajú a dochádza ku kolapsu. [27]

3.1 Základné charakteristiky

WFC pri generovaní obsahu používa mriežku (angl. grid). Bunky mriežky sú na začiatku prázdne a musia byť vyplnené jedným zo vstupných vzorov, nazývaných moduly. Podľa terminológie kvantovej mechaniky bunky kolabujú. Algoritmus končí po kolapse všetkých buniek. Dôležitým aspektom algoritmu je spôsob výberu kolapsujúcej bunky. Algoritmus vyberá na základe entropie buniek. Pri generovacom procese dochádza ku chybovým stavom, pri ktorých algoritmus nedokáže ďalej pokračovať. [27]

Štruktúra mriežky

Bunky v štruktúre mriežky, s ktorou WFC pracuje majú priradených susedov, ktorých algoritmus postupne kolabuje podľa stanovených pravidiel. Pravidlá určujú aké moduly môžu vedľa seba vo vygenerovanom obsahu existovať. Algoritmus dokáže pracovať s rôznymi 2D a 3D mriežkami, kde počet susedných buniek je určený štruktúrov mriežky (viď obr. 13). Pravidelné mriežky majú susedov určených podľa definície okolia (viď obr. 6). Napríklad, v hre Sudoku sa hracia mriežka skladá z 9×9 buniek, ktoré sú zoskupené vo štvorcoch s rozmermi 3×3. V Sudoku mriežke má každá bunka 20 susedov, teda zmena stavu bunky ovplyvňuje celý stĺpec, riadok a štvorec, v ktorom sa nachádza. Ďalej algoritmus dokáže pracovať aj s nepravidelnými mriežkami, ako sú napríklad Voronové diagramy, v ktorých má každá bunka variabilný počet susedov. [25]



Obr. 13: Susediace bunky v mriežke (a) pravidelná mriežka s von Neumanovým okolím, (b) Sudoku mriežka, (c) Voronov diagram. Modré bunky sú susediace k čiernej bunke. Upravené z [25]

Entropia

Entropia udáva informáciu koľko modulov môže byť do bunky dosadených bez porušenia definovaných pravidiel. Na začiatku generovania je hodnota entropie všetkých buniek nastavená počtom modulov obdržaných zo vstupu. Algoritmus vyberie vždy bunku s najmenšou entropiou obsahujúcu najmenší počet modulov, ktoré môže na jej miesto dosadiť. Moduly môžu mať nastavenú úroveň pravdepodobnosti, s akou sa budú vo výslednom obsahu vyskytovať. V takom prípade pravdepodobnosti modulov dodatočne ovplyvňujú entropiu bunky. Tá sa počíta podľa Shannonovej definície entropie (h), ako

$$h = -\sum p_i \log(p_i),$$

kde výsledná hodnota je súčet pravdepodobností dosaditeľných modulov (p_i) pre danú bunku. To znamená, že bunka s vyššou pravdepodobnosťou modulov má prednosť vo výbere pred bunkou obsahujúcou moduly s nižšou pravdepodobnosťou. Ak vo výpočte algoritmu existuje viac buniek s rovnakou minimálnou entropiou, vyberie sa náhodne. [27]

Chybové stavy

Zväčšenie rozmerov generovaného obsahu vedie k dlhšiemu času potrebnému na jeho vytvorenie, a k nárastu chýb počas generovacieho procesu. Po kolapse bunky algoritmus rekurzívne propaguje informácie o vybranom module k jej susedom. Susedné bunky si na základe tejto informácie aktualizujú entropiu, čím sa postupne redukuje entropia celej mriežky. Chyba nastane, keď bunka dosiahne nulovú entropiu. V tomto stave bunka nemôže obsahovať žiadny modul a algoritmus nemôže pokračovať v generovaní.

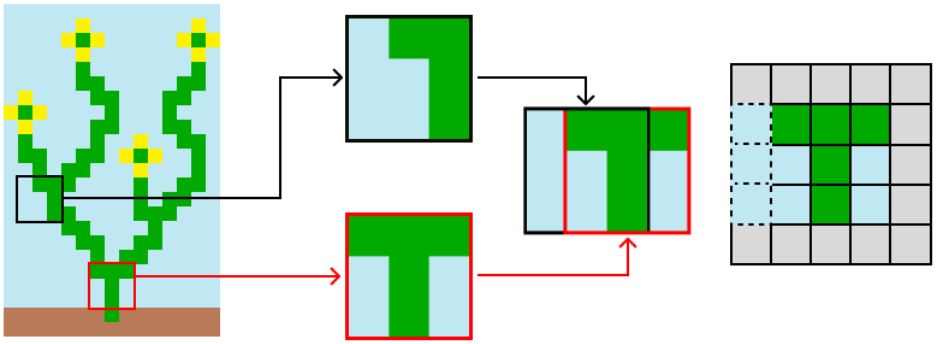
Pri hrách generujúcich konečné prostredie, je možné chybu odstrániť opätovným vygenerovaním. To však neplatí pri generovaní nekonečného prostredia, pretože jeho vygenerovaná časť už mohla byť preskúmaná hráčom. Chybný stav sa odstráni prostredníctvom backtrackingu. Backtracking je postup, pri ktorom sa algoritmus pri výskyte chyby vráti do posledného známeho bezchybného stavu. Pri každom backtrackingu zmenšíme entropiu aspoň jednej bunky. Takýmto postupom sa algoritmus zaručene dopracuje k výsledku. Doposiaľ vygenerovaný svet sa pomocou backtrackingu pri chybe čiastočne vymaže. Algoritmus pred každým kolapsom zaznamená stav mriežky a voľbu modulu kolabujúcej bunky, čím vytvára snapshoty. Pri chybe sa pomocou zaznamenaných snapshotov vráti do posledného správneho stavu a z bunky odstráni modul, ktorý viedol k chybe. Ďalej náhodne vyberie jednu zo zostávajúcich možností a pokúsi sa pokračovať. Ak už bunke nezostáva žiadna možnosť, algoritmus rekurzívne pokračuje ďalším snapshotom. Ak sa chyba odhalí v štádiu, z ktorého sa algoritmus musí vrátiť späť o značný počet krokov, môže byť tento postup rovnako neefektívny ako opätovné vygenerovanie celého prostredia. [24]

3.2 Modely

Algoritmus má dva implementačné modely, dlaždicový a prekrývací. Princíp algoritmu zostáva pri oboch rovnaký, avšak modely sa odlišujú vo vytvaraní vstupných modulov a pravidiel. V dlaždicovom modeli sa pravidlá a moduly vytvárajú manuálne. Prekrývací model rozdeľuje vzor na menšie časti, ktoré slúžia ako moduly. Pravidlá pre tieto moduly sú automaticky generované analýzou ich vzájomných vzťahov. Tieto modely majú nedostatky, ktoré rieši ďalší typ modelu, Markovov model. Zmienené implementačné stratégie sú pre jednoduchosť znázornené na 2D obsahu, no algoritmus je aplikovateľný aj pre tvorbu 3D obsahu. [23, 27]

Prekrývací model

Prekrývací model (angl. overlapping model, OWFC) na vstupe očakáva obrázok alebo bitmapu, z ktorej extrahuje všetky možné kombinácie modulov využitím posuvného okna. Rozmery modulov $N \times N$ závisia na rozmeroch posuvného okna. Čím väčšie bude zvolené N, tým viac bude výsledný obrázok pripomínať pôvodný. Dimenzie modulu určujú počet obsiahnutých pixelov. Každý modul môže mať až osem rôznych variácií, vrátane otočených a zrkadlových. Model po extrakcii modulov vykoná analýzu, z ktorej zistí ich vzájomné vzťahy prekrývaním (viď obr. 14). Počet modulov, ktoré sa môžu objaviť vedľa špecifického modulu je $(2(N-1)+1)^2$. Prekrývací model je vhodný pre tvorbu textúr, avšak nie je ideálny pre generovanie herných úrovní.



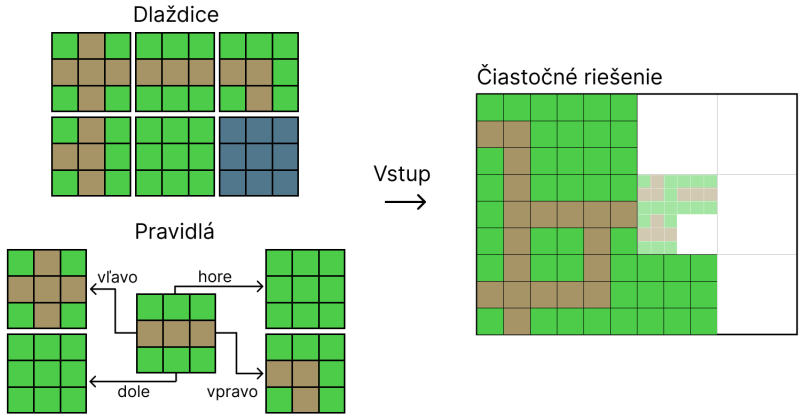
Obr. 14: Príklad prekrytia modulov extrahovaných zo vstupného vzoru. V prípade zhody prekrytej časti modulov sa vygeneruje pravidlo, ktoré určuje, že takto prekryté moduly môžu vedľa seba vo výstupe susediť. Upravené z [22].

Mriežka v počiatočnej fáze generovania pozostáva z pixelov bez farebnej hodnoty. Entropia každého pixelu je na začiatku rovná počtu farieb v zdrojovom obrázku. Model priraďuje pixelom farby postupným kolapsom oblasti o veľkosti modulu. Po každom kolapse sa informácia o vybranom module pre danú oblasť šíri do priľahlých pixelov, ktoré si zachovajú len tie farebné možnosti vyplývajúce zo správnych prekrytí modulov. Súčasne dochádza k aktualizácii ich entropie.

Iteratívny proces identifikácie modulov OWFC je výpočetne náročný. S rastúcimi rozmermi modulov sa zvyšuje počet možností ich prekrývania, čo znamená, že algoritmus musí pri každom volaní propagačnej funkcie kontrolovať viac obmedzení pri pixeloch. Výpočetný čas rastie kvadraticky vzhľadom k N. Pri použití zložitejších obrázkov s výraznými vzormi, algoritmus vytvára väčší súbor pravidiel, čím výsledok stráca na náhodnosti. [23]

Dlaždicový model

Dlaždicový model (angl. tiled model, TWFC) sa odlišuje od prekrývacieho modelu na vstupe, kde očakáva sadu modulov (dlaždíc) a pravidiel (viď obr. 15), ktoré určujú ich prepojenia a rotácie. Model nemusí vykonávať analýzu, pretože má pravidlá a moduly dostupné na vstupe. Moduly sú chápané rovnako ako pri prekrývacom modeli. Počet pravidiel je priamo úmerný počtu modulov. Zväčšením rozmerov modulov sa výpočetný čas nemení. Moduly a pravidlá sú užívateľsky modifikovateľné, čo umožňuje väčšiu kontrolu nad vygenerovaným obsahom. TWFC nemá priamy spôsob identifikácie frekvencie výskytu modulov, pretože nemá prístup k zdrojovému vzoru. Všetky potrebné informácie o moduloch získava zo súboru metadát, v ktorom sú uložené pravidlá, frekvencie a veľkosť modulov. Určenie frekvencie je v TWFC voliteľné.

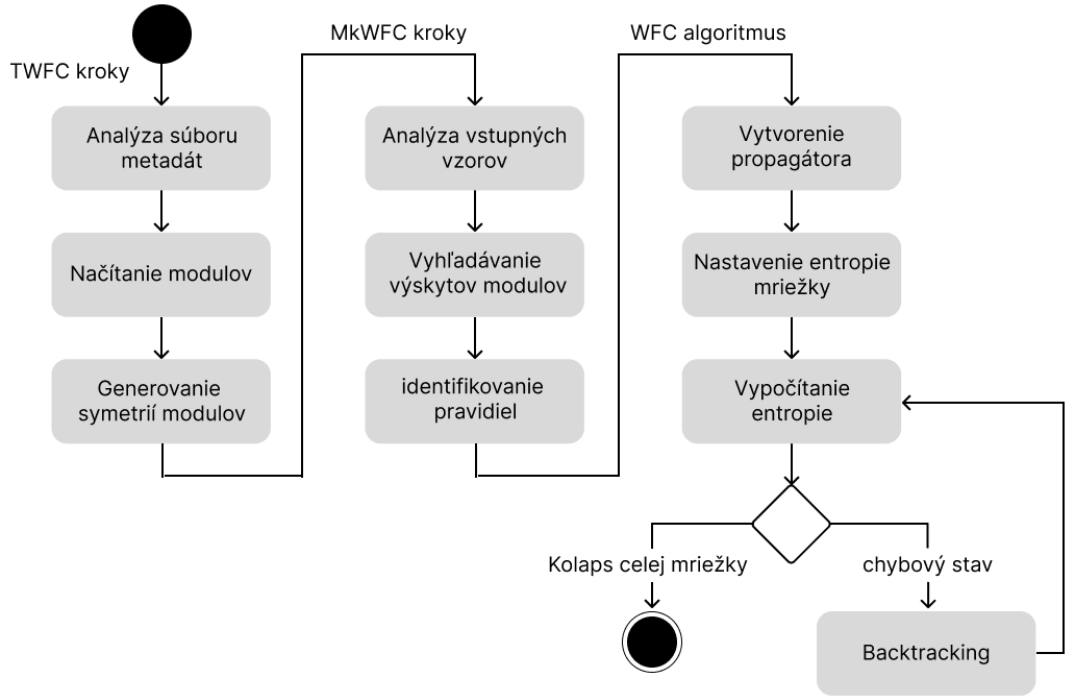


Obr. 15: Dlaždicový model pracujúci so vstupom, ktorý pozostáva zo súboru pravidiel a dlaždíc s rozmermi 3x3. Dlaždice sú podľa pravidiel postupne vkladané do mriežky. Čiastočné riešenie obsahujúce potencionálne možnosti, ktoré by mohli byť vložené do konkrétnej bunky po aktualizácii jej entropie a vylúčení dlaždíc nespĺňajúcich pravidlá.

Vytvorenie súboru metadát je zložité. Identifikácia správnych pravidiel a ich vplyv na generovaný výstup vyžaduje testovanie, ladenie pravidiel a vyvažovanie frekvencie používaných modulov. Odstránenie alebo pridanie pravidla môže významne ovplyvniť proces riešenia entropie. Zväčšenie počtu pravidiel alebo pridanie ďalších modulov, vyžaduje prekonfigurovanie už existujúcich pravidiel. Model nie je dobre škálovateľný. S rastúcim počtom pravidiel narastá aj komplexnosť generovacieho procesu a chybovosť pri definovaní pravidiel. [23, 27]

Markovov model

Markovov model (MkWFC) vďaka automatizácii procesu manuálneho vytvárania pravidiel poskytuje vyššiu rýchlosť ako OWFC a zároveň rieši problém škálovateľnosti TWFC. Moduly nevytvára pomocou posuvného okna ako OWFC, keďže všetky potrebné moduly má dostupné na vstupe v metadátach, podobne ako TWFC. MkWFC využíva informácie o moduloch zo vstupných vzorov, ktoré sú obsiahnuté v metadátach a odvodí z nich pravidlá. Zo získaných pravidiel vytvorí pre každý modul Markovo náhodné pole (angl. Markov random field) kompatibilných susedov a pravdepodobností ich výskytu (viď obr. 16). Na identifikáciu nových pravidiel model vyžaduje pridanie ďalších vstupných vzorov alebo zväčšenie ich veľkosti. Ak je potrebné vygenerovať výstup podľa iných pravidiel, stačí upraviť vzory. Cieľom implementácie MkWFC je skrátiť čas potrebný na vytvorenie metadát a súčasne zachovať výhody výkonnosti implementácie TWFC. Automatickou identifikáciou pravidiel sa skracuje čas vytvárania metadát, ako aj miera ľudskej chybovosti. [23]

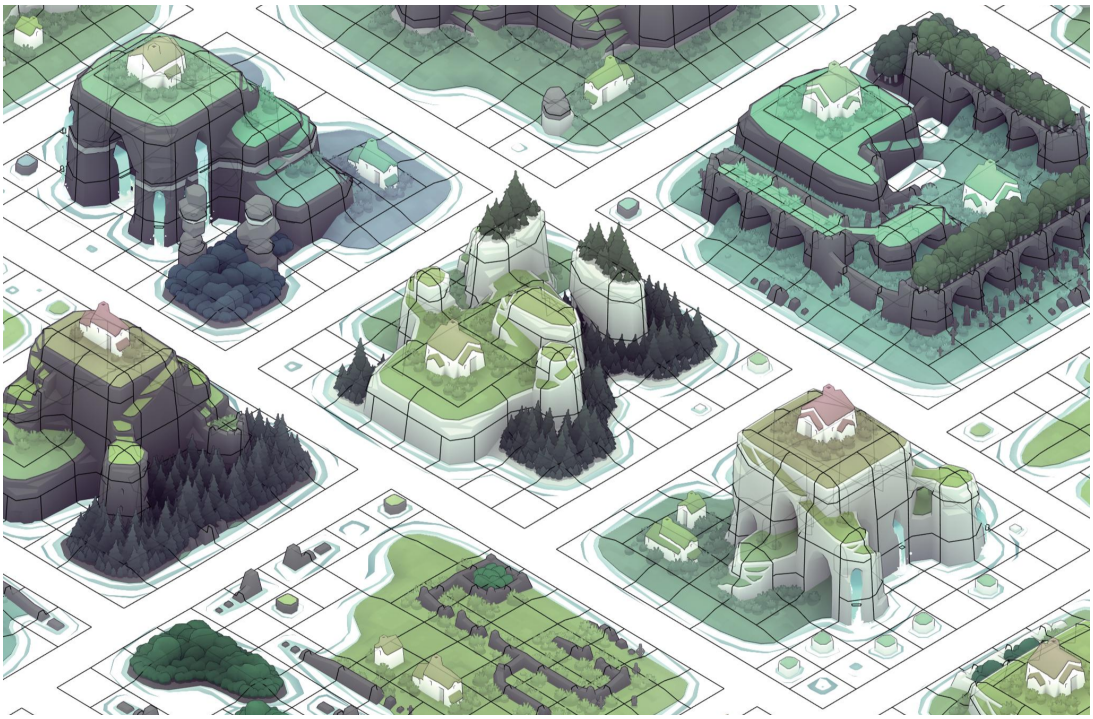


Obr. 16: Vývojový diagram MkWFC modelu. Po analýze metadát MkWFC vygeneruje všetky symetrie modulov a vykoná analýzu vstupných vzorov. Po analýze vzorov identifikuje susedské vzťahy medzi modulmi a vytvorí Markovove náhodné polia. Na základe vytvorených polí sa bude aktualizovať entropia po kolapse buniek. Upravené z [23].

3.3 Príklady použitia pri generovaní 3D prostredia

Táto podkapitola zahŕňa výbrané príklady použitia algoritmu Wave function collapse pri generovaní 3D prostredí v hrách. Prvým príkladom je strategická hra Bad North, kde hráč chráni svoj ostrov proti nájazdom Vikingov. Bad North pri každom novom štarte hry vygeneruje jedinečnú kampaň a ostrovy. Na generovanie ostrovov sa používajú predvytvorené dlaždice, ktoré sú upravené tak, aby zasahovali do viacerých voxelov mriežky (viď obr. 17). Porušením uniformity dlaždíc, vzhľad prostredia pôsobí prirodzenejšie. Vygenerované ostrovy musia byť vhodné na vylodenie a pohyb nepriateľských aj hráčových jednotiek. [37]

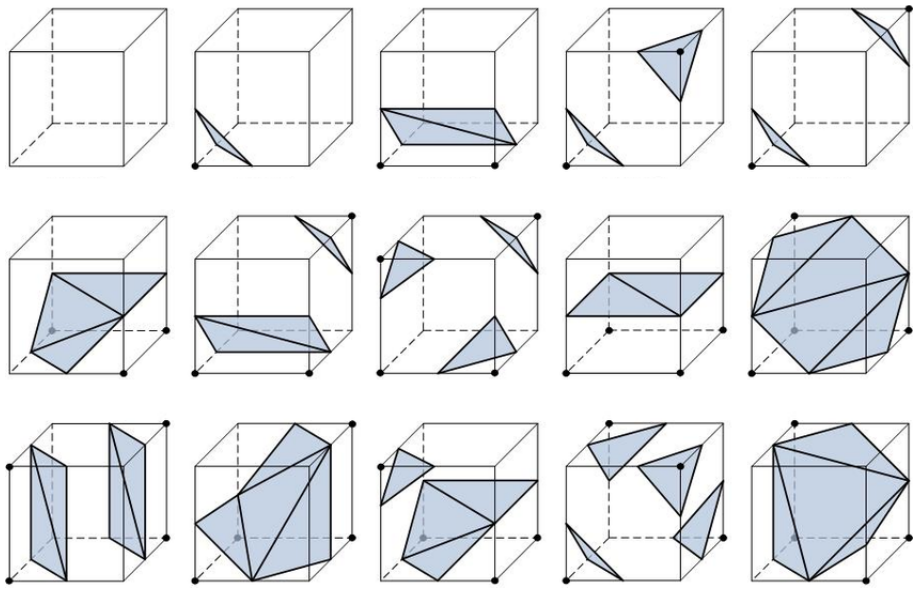
Ďalším príkladom je sandboxová stavebná hra Townscaper, v ktorej si hráč interaktívne vytvára mesto na nepravidelnej 3D mriežke klikaním na jej ľubovoľne voxely. Hra používa algoritmus Wave function collapse na identifikáciu vhodných stavebných prvkov pre zvolené miesto. Vzhľadom na nepravidelnosť mriežky hra používa aj algoritmus Marching Cubes (viď kap. 3.4) na vhodné umiestnenie vybraného prvku do prostredia. Na rozdiel od hry Bad North je tento proces optimalizovaný pre generovanie v reálnom čase. Zmeny vo voxeloch spúštajú propagáciu nových pravidiel do okolitého prostredia, ktoré sa im musí prispôsobiť. Tento efekt sa v hre prejavuje ako úprava okolných budov. Ak generovanie zlyhá v hre Bad North, celý proces musí prebehnúť od začiatku. Naopak, v hre Townscaper sa chyby v generovaní ignorujú, čo môže v hernom prostredí spôsobiť vizuálne defekty. [37]



Obr. 17: Schéma vygenerovaných ostrovov z hry Bad North. Prevzaté z [42]

3.4 Marching cubes

Algoritmus Marching cubes slúži na aproximáciu izopovrchov prostredníctvom triangulácie. Izopovrch je povrch definovaný implicitnou rovnicou v tvare $F(x, y, z) = f$, kde F je priestorová funkcia a f je konštanta. Prefix izo naznačuje, že funkcia F nadobúda rovnakú hodnotu f na celom povrchu, ktorý funkcia reprezentuje. Algoritmus pre ľubovoľný izopovrch funguje nasledovne, na začiatku rozdelí priestor okolo povrchu na pravidelnú voxelovú mriežku. Následne prostredníctvom implicitne definovanej rovnice, pre každý voxel mriežky vyhodnotí, či sa rohy voxelu nachádzajú vo vnútri, alebo mimo povrchu. Počet kombinácií hodnôt vrcholov voxelu je celkovo $2^8 = 256$. Odstránením symetrií je tento počet zredukovaný na 15 unikátnych kombinácií. Kombinácie slúžia ako index do vyhľadávacej tabuľky, ktorá pre každú kombináciu hodnôt vrcholov vráti príslušnú trojuholníkovú konfiguráciu (viď obr. 18), ktorú daný voxel nadobudne. Konfigurácie voxelov sa na záver spoja, čím vznikne výsledný aproximovaný povrch. Veľkosť voxelov ovplyvňuje kvalitu aproximácie povrchu a rýchlosť algoritmu. Menšie voxely zvyšujú presnosť detailov na úkor pomalšieho spracovania, zatiaľ čo väčšie voxely zlepšujú rýchlosť spracovania na úkor zníženej presnosti detailov. [41]



Obr. 18: Trojuholníkové konfigurácie podľa vrcholov voxelu nachadzajúcich sa vo vnútri izopovrchu (čierne body). Prevzaté z [41].

4 Optimalizácia

Kapitola je zameraná na prístupy a metódy, ktoré zlepšujú efektivitu procesu generovania a vykreslovania (angl. rendering) obsahu. Optimalizácia generovacieho procesu je dôležitým aspektom pri vývoji hier. Jej cieľom je zníženie výpočetných a pamäťových nárokov pre dosiahnutie lepšej reaktivity hry a snímkových frekvencií. Procedurálne generovanie prostredia môže rýchlo zaťažiť systémové zdroje, a preto je nevyhnutné algoritmy generujúce prostredie implementovať čo najefektívnejšie. Optimalizácia renderovacieho procesu závisí na jeho konkrétnom modeli, ktorý má každý herný engine implementovaný vlastným spôsobom. Model opisuje aké kroky má grafický systém vykonať, aby vykreslil hernú scénu na obrazovku. Kapitola sa venuje metódam, ktoré optimalizujú renderovanie ako vyraďovanie, úroveň detailov, dávkovanie, atlas textúr a rozdeľovanie priestoru. Zmienené metódy podporuje väčšina herných enginov, no občas pre konkrétne účely nie sú dostačujúce. V takom prípade si herné štúdia vytvárajú vlastné herné enginy, ktoré môžu prispôsobiť svojim špecifickým požiadavkám. Príkladom je štúdio Hello Games, ktoré si pre hru No Man's sky vytvorilo vlastný herný engine, špecializujúci sa na procedurálne generovanie. [35]

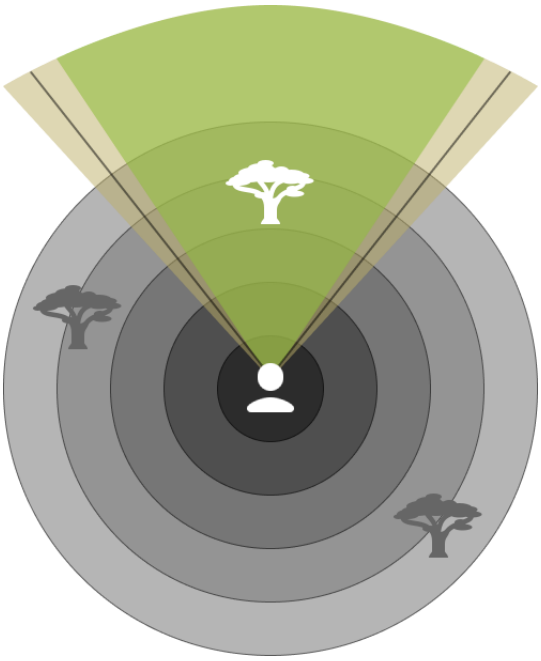
4.1 Vyraďovanie

Vyraďovanie (angl. culling) je metóda zvyšujúca efektivitu renderovania v herných prostrediach tým, že odstraňuje potrebu vykresľovať objekty, ktoré nie sú v aktuálnom zábere kamery. Metóda testuje viditeľnosť objektov v hernej scéne pomocou tienidiel (occluderov) a polohy hráčovej kamery. Occluder, ako napríklad budova alebo hora, blokuje alebo zatieňuje objekty umiestené za ním. Blokované objekty, ktoré sú mimo perspektívu hráčovej kamery sú z renderovacieho procesu vyradené, čo znižuje počet operácií potrebných na vykreslenie scény a zlepšuje výkon hry. Existujú rôzne typy vyraďovania, ako je frustum culling, occlusion culling a back-face culling. Frustum culling vyraďuje objekty mimo zorného poľa kamery. Occlusion culling vyraďuje objekty, ktoré sú zakryté inými objektmi. Back-face culling vyraďuje strany objektov otočené od kamery a teda nie sú viditeľné. Metóda vykonáva dodatočné operácie na identifikáciu objektov určených na vyradenie. V prípade prostredí s veľkým počtom objektov môžu dodatočné kroky na identifikáciu objektov znižovať rýchlosť renderovania. [28]

4.2 Úroveň detailov

Úroveň detailov (angl. level of details) prispôsobuje kvalitu textúr alebo objektov v závislosti od ich vzdialenosti od hráča alebo dôležitosti. Textúry môžu mať niekoľko úrovní detailov, pričom každá úroveň určuje kvalitu, v ktorej sa má textúra zobraziť (viď obr. 19). Znížením kvality sa zníži počet polygónov, potrebných na vykreslenie prostredia, čo vedie k úspore pamäti a k zníženiu výpočetného času procesoru grafickej karty. Metóda sa používa v kombinácii s metódami

ako geomorphing a blending, aby sa eliminoval nežiadúci popping efekt, ktorý vzniká, keď prechod medzi úrovňami nie je dostatočne plynulý. Geomorphing pridáva novú úroveň detailov medzi dve existujúce susedné úrovne ich aproximáciou. Blending počas prechodnej doby medzi úrovňami súčasne vykresľuje obidve úrovne a lineárne interpoluje ich alfa hodnoty. Tento proces postupne zoslabuje pôvodnú úroveň a zvýrazňuje novú, až kým pôvodná úroveň nezanikne. Popping efekt môže vnikať aj pri preťažení grafickej karty, ktorá nestíha dostatočne rýchlo renderovať detaily vzhľadom na zmeny v hráčovej pozícii. K implementácii metódy úrovne detailov sa využívajú dva hlavné prístupy, diskrétny a kontinuálny. V diskrétnom prístupe pre každý objekt existuje pevný počet preddefinovaných úrovní detailov, medzi ktorými systém prepína, keď sa objekt vzdiali alebo priblíži k hráčovi. Kontinuálny prístup poskytuje plynulý prechod medzi úrovňami detailov tým, že dynamicky upravuje počet polygónov objektu v reálnom čase. [29]



Obr. 19: Obrázok ukazuje použitie úrovní detailu a vyraďovania. Zóna označená zelenou farbou reprezentuje zorný uhol hráča. Vykreslené budú iba tie objekty, ktoré sa nachádzajú vnútri tohto zorného poľa. Úroveň detailov závisí na vzdialenosti od hráča, objekty v tmavších oblastiach bližšie k hráčovi budú vykreslené s vyšším stupňom detailov.

4.3 Dávkovanie

Dávkovanie (angl. batching) je proces, ktorý agreguje podobné objekty do skupín. Skupiny dokáže systém vykresliť pomocou menšieho počtu volaní vykreslovacej funkcie. Rozlišujeme dva typy dávkovania, statické a dynamické. Sta-

tické dávkovanie je vhodné pre objekty, ktoré nezmenia svoj stav počas celej hry (stromy, budovy, terén). Tieto objekty sú zoskupené do jednej skupiny a spracované ešte pred spustením hry. Dynamické dávkovanie je vhodné pre objekty, ktoré sa môžu počas hry meniť alebo pohybovať (postavy, interaktívne objekty). Pri tejto metóde sa objekty dynamicky zoskupujú v reálnom čase počas vykresľovania každého snímku. Táto metóda poskytuje väčšiu flexibilitu v meniacich sa herných scénach, avšak za cenu zvýšenej výpočtovej náročnosti, keďže systém musí neustále rozhodovať o najefektívnejšom spôsobe zoskupenia objektov. [33]

4.4 Atlas textúr

Atlas je textúra obsahujúca niekoľko menších textúr usporiadaných v mriežke pre rôzne herné objekty. Herný engine využíva atlas textúr tak, že vytvorí mapovaciu tabuľku, ktorá namapuje tieto podtextúry k jednotlivým objektom v hre. Ak sú objekty v scéne namapované na textúry v rovnakom atlase, potom ich dokáže simultánne vykresliť bez potreby načítavania novej textúry, samostatne pre každý objekt. Využitím atlasu môžeme účinne znížiť množstvo vykreslovacích volaní. Metóda sa využíva spolu s dávkovaním. [30]

4.5 Object pooling

Object pooling optimalizuje a redukuje alokácie pamäti vytvorením inventáru znovupoužiteľných objektov. Objekty sú udržované v pamäti namiesto stáleho ničenia a vytvárania nových inštancií. Inventár nepodlieha automatickej správe pamäte prostredníctvom garbage collectoru. Keď hra potrebuje nový objekt, vezme si ho z inventára existujúcich objektov. Po použití je objekt vrátený do inventára, čím sa stáva opäť dostupným na ďalšie použitie. Opakovaným využívaním existujúcich objektov sa znižujú režijné náklady, čo vedie k lepšiemu výkonu a plynulejšiemu priebehu hry. Inventár objektov je obzvlášť užitočný pre typy herných objektov, ako sú projektily, nepriatelia a častice používané v grafických efektoch. Počet potrebných objektov sa musí adekvátne odhadnúť, aby nedošlo k nadmernému využitiu pamäte na objekty, ktoré nebudú potrebné alebo, aby nebolo objektov príliš málo a predišlo sa tak potenciálnemu pretečeniu pamäte. [34]

4.6 Rozdeľovanie priestoru

Technika rozdeľovania priestoru (angl. spatial partitioning), slúži k efektívnejšiemu spracovaniu a správe objektov. Rozdeľuje priestor na menšie segmenty tak, aby každý objekt ležal práve v jednom segmente. Každý segment môže byť znovu rekurzívne rozdelený. Segmenty sú usporiadané v stromovej štruktúre. Štruktúra umožňuje rýchle vyhľadávanie geometrických informácií o ľubovoľnom bode v priestore, čo z nej robí neoddeliteľnú súčasť počítačovej grafiky. Technika umožňuje optimalizovať procesy detekcie, kolízie a vykresľovania tým, že sa vykonávajú len v segmentoch, ktoré sú relevantné pre danú operáciu. Napríklad, pri

hre s viacerými hráčmi by kontrola vzájomných interakcií medzi každým hráčom bola príliš náročná a neefektívna. Využitím techniky rozdeľovania priestoru sa priestor rozdelí na menšie segmenty. Namiesto kontroly interakcií medzi všetkými hráčmi sa môže zamerať iba na tých, ktorí sú vo vzájomnej blízkosti. Týmto spôsobom sa výrazne zníži počet kontrolovaných interakcií a zlepší výkonnosť hry. Pre rozčlenenie 2D priestorov sa využíva štruktúra zvaná quadtree, zatiaľ čo pre 3D priestory je typické použitie octree štruktúry. [2]

4.7 Plánovanie úloh

Hry zvyčajne využívajú jedno alebo dve procesorové vlákna. Prvé vlákno (herné) sa používa na spracovanie hernej logiky. Druhé vlákno (vykresľovacie) slúži na vytváranie vstupných a výstupných procesov. V dôsledku toho má procesor veľa nevyužitých a neoptimalizovaných vlákien. Nadmerné využívanie vlákien a zlé rozdelenie úloh môže znížiť výkonnosť hry kvôli prepínaniu kontextu. Proces rozdeľovania úloh je známy ako task farming. Pri preskúmavaní procedurálne generovaného herného sveta je dôležité, aby generovanie nového segmentu nenarušovalo plynulosť priebehu hry. Výpočtovo náročné úlohy môžu zaťažiť herné vlákno a spôsobiť výrazné zaseknutie, čo u hráča vytvorí dojem, že hra nereaguje. Paralelizácia a vhodné plánovanie úloh umožňujú hernému vláknu pokračovať vo vykonávaní hernej logiky bez výrazného spomalenia. Úlohy možno spracovať priamo v hernom vlákne, paralelne alebo prostredníctvom časovo rozloženého spracovania. Vykonávanie úloh priamo na hernom vlákne je vhodné pre operácie, ktoré vyžadujú priamu interakciu s herným enginom ako vytváranie, odstraňovanie a iné úpravy objektov. Paralelné spracovanie je vhodné pre výpočtovo náročné úlohy, ktoré nevyžadujú bezprostrednú interakciu s herným enginom, ako sú komplexné matematické výpočty. Časovo riadené spracovanie sa používa pri úlohách, ktoré nie je možné spracovať paralelne. Úlohy sú usporiadané v hernom vlákne vo fronte, odkiaľ sú postupne spracované v jednotlivých snímkoch. Každá úloha má pridelený časový limit, ktorý určuje, ako dlho ju môže herné vlákno spracovávať. Ak tento limit vyprší, spracovanie úlohy sa odloží na ďalší snímok a vlákno sa vráti k vykonávaniu hernej logiky. [36]

5 The Backrooms prostredie

The Backrooms je súhrnný názov pre prostredia, ktoré boli prvýkrát spomenuté na webovej stránke 4chan v roku 2019. Na stránke bola uverejnená ilustrácia (viď obr. 20), okolo ktorej sa strhla vlna konšpirácií. Obrázok predstavuje prvý level Backrooms, ktorý sa odohráva v kancelárskych priestoroch bez nábytku a ľudí. Na stenách kancelárií je žltá tapeta a podlaha je pokrytá hnedým kobercom. Prostredia predstavujú záhadné, nekonečné bludiská. Existuje množstvo interpretácií, niektoré opisujú bludiská ako paralelné dimenzie, iné ich prirovnávajú k reálnym miestam, ktoré sú prístupné pomocou tajných chodieb alebo skrytých dverí z reálneho sveta. V tejto kapitole bude bližšie opísaná verzia The Backrooms, konkrétne level 37, ktorý predstavuje priestory bludiska pripomínajúce opustené kúpeľné zariadenie. K Backrooms sa viažu rôzne fikcie a príbehy, ktoré hovoria o neobvyklých, nebezpečných bytostiach, pričom dostať sa z nich von sa podarí málokomu. [12]

The Backrooms boli použité ako zdroj inšpirácie pre rôzne hry, ktoré sa snažia napodobniť strašidelnú a znepokojujúcu atmosféru. Medzi hry inšpirované týmto prostredím patrí psychologický horor Lost in Vivo [13]. Príbeh hry sa odohráva v podzemných tuneloch, kde hlavná postava hľadá svojho psíka, ktorý do nich spadol. Ďalej to je hra Lethal Company [14], kooperatívna hororová hra o prežití, kde hráči zbierajú a predávajú šrot z opustených industriálnych komplexov.

Obr. 20: Ukážka prvého levelu The Backrooms. Prevzaté z [12]

5.1 Charakteristika levelov

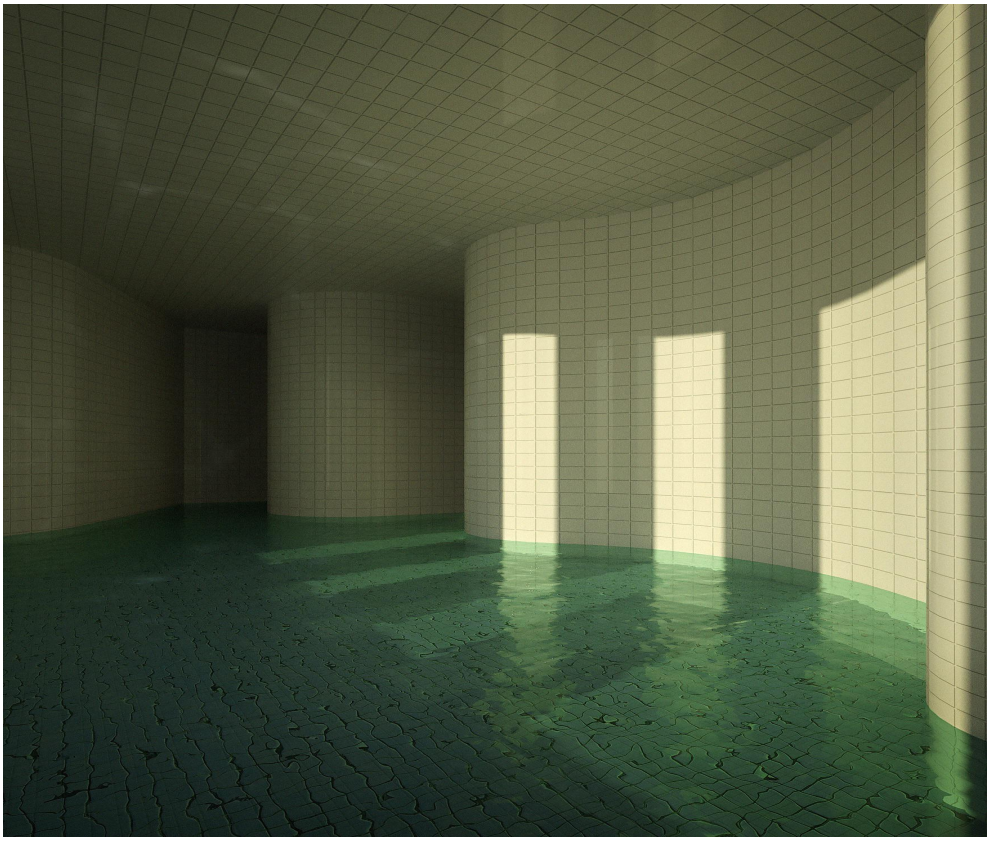
Backrooms sú rozdelené do viacerých typov levelov, ktoré sa líšia veľkosťou, štruktúrou a atmosférou. Levely sa rozdeľujú na klasické, enigmatické a podlevely. Klasické levely sú medzi sebou viac prepojené a najviac podobné reálnemu svetu. Levely sú postavené na tom, aby vyvolávali pocity strachu, osamelosti a prázdnoty. Sú pomerne bezpečné a ľahko sa v nich orientuje. Enigmatické levely sú záhadné, majú skrytý význam a môžu byť alegóriou na určitú spomienku alebo emóciu. Miestnosti sa v nich menia bez varovania. Súčasťou hlavného levelu môžu byť podlevely, ktoré majú s hlavným levelom spoločné vlastnosti. Podlevely môžu byť taktiež plné monštier, nástrah a iných nebezpečenstiev. Štruktúra levelov sa mení podľa predstáv a tvorivosti autorov. [16]

Nasledujúci popis prostredí levelov má za úlohu priblížiť rozmanitosť ich charakteristík. Backrooms sú populárne a neustále sa rozvíjajúce fiktívne svety. Popis zmienených levelov sa môže meniť. Jednotlivé levely majú svoje charakteristiky, ktorými sa väčšina autorov riadi pri ich vytváraní. Napríklad Level 5, nazývaný Terror Hotel, je opustený hotel s množstvom sál a apartmánov vybavených zastaraným nábytkom (viď obr. 20). Miestnosti sú poprepájané dlhými kľukatými chodbami. Ďalším je Level 6, nazývaný Lights Out [17], ktorý pozostáva z miestností a chodieb, ktoré majú betónové podlahy, železné steny s potrubiami. Nefungujú v ňom žiadne umelé zdroje svetla, a tak sa celý odohráva v šere. K levelu 6 existuje podobný podlevel 6.2 [18], ktorý je charakteristický slabo svietiacimi, náhodne rozmiestnenými neónovými svetlami. Level 17 [19] predstavuje nekonečné chodby a schodiská, ktoré pripomínajú vnútro opustenej nákladnej lode.

Obr. 21: Ukážka levelu 5. Prevzaté z [43].

5.2 Level 37

Level 37, známy ako Poolrooms, predstavuje rozsiahly komplex prepojených zaplavených miestností a chodieb. Priestory miestností pôsobia čisto, steny sú pokryté bielymi dlaždicami a podlaha je pod vodnou hladinou. Oblasti levelu sa líšia veľkosťou a dizajnom, od identických po otvorenejšie a abnormálne tvarované miestnosti a chodby. Miestnosti sú osvetlené z nepravidelných uhlov, čo spôsobuje, že niektoré časti sú úplne tmavé. Schody vedú do zaplavených šácht. Akustika v zaplavených miestnostiach nesie tichý zvuk vody, ktorý doplňuje ozvena krokov. Zvuk vody dodáva do levelu zvláštnu osamotenú atmosféru. Level je úplne bez známok života a výskytu bytostí, narozdiel od iných levelov (viď kap. 5.1). Level 37 je podľa [20] prepojený s ostatnými levelmi, a to vstupmi z levelov 7, 67 a 997. Východy z levelu vedú do levelov 43, 4 alebo do podlevelu 37.1, ktorý je temnejšou verziou levelu 37. [20]



Obr. 22: Ukážka levelu 37. Prevzaté z [20]

6 Dokumentácia

Kapitola sa venuje praktickej časti diplomovej práce, v ktorej bolo za úlohu vytvoriť hru generujúcu nekonečné 3D bludisko na štýl levelu 37 z Backrooms (viď kap. 5.2). V kapitole budú zhrnuté technológie a implementačné stratégie procedurálneho generovania, ktoré boli použité pri tvorbe hry.

6.1 Použité technológie

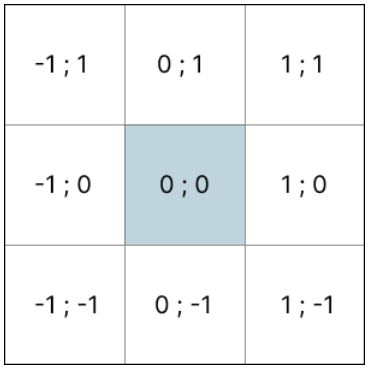
Na vývoj hry a modelovanie assetov bol použitý Unreal Engine 5. Unreal Engine vyvinutý spoločnosťou Epic Games je multiplatformový herný engine, ktorý disponuje komplexnou sadou nástrojov pre renderovanie 3D grafiky v reálnom čase. Používa sa nie len na vývoj hier, ale aj na tvorbu filmov, simulácií, vizualizáciu architektúry a automobilov. Na vývoj funkcionality bol v hernom engine použitý programovací jazyk C++ a vizuálny skriptovací jazyk Blueprint špecifický pre Unreal Engine. Blueprinty sú postavené nad jazykom C++ a slúžia k definovaniu objektovo orientovaných tried, bez nutnosti ich kódovania, prostredníctvom prepájania uzlov reprezentujúcich rôzne operácie a udalosti. Prostredie vytvorené v praktickej časti práce bolo implementované kombináciou programovania v jazyku C++ a Blueprintov. Kľúčové procedurálne aspekty, ako sú algoritmy pre generovanie prostredia a herné mechanizmy, ako ukladací systém boli implementované v jazyku C++. Blueprinty boli primárne využité na vytvorenie užívateľských rozhraní a interaktívnych prvkov hry. [44]

6.2 Nekonečne generované prostredie

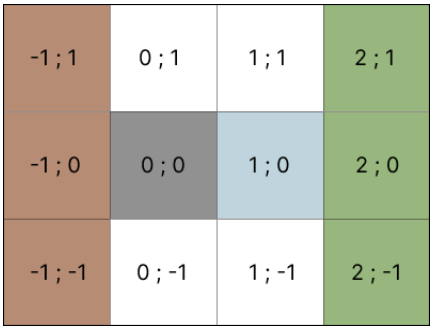
Procedurálne generovanie umožňuje tvorbu dvoch typov herných svetov. Prvý typ má pevne stanovené hranice. Druhý typ simuluje nekonečnosť tým, že kontinuálne generuje susediace segmenty prostredia, čím sa vytvára ilúzia neobmedzeného a súvislého sveta. Vzhľadom na obmedzenú kapacitu pamäte počítačov nie je možné neustále pridávať nový obsah bez súčasného odstraňovania starého obsahu. V hrách sa často používa technika rozdelenia herného sveta na menšie segmenty požadovanej veľkosti (chunky). (viď obr. 23).

Pri presune hráča do nového segmentu hra generuje nový obsah a súčasne odstraňuje starý, aby minimalizovala spotrebu pamäte. Segmenty sa pridávajú v smere pohybu hráča, zatiaľ čo segmenty na opačnej strane sa odstraňujú (viď obr. 24).

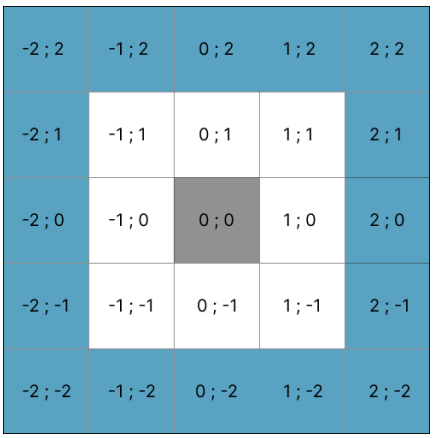
Ak sa hráč bude pohybovať tam a späť medzi dvoma segmentmi, budú sa neustále načítavať segmenty, ktoré sa práve odstránili. Riešením je zvýšiť počet segmentov, na ktoré je svet rozdelený, čím sa vytvorí buffer načítaných segmentov. Týmto spôsobom bude mať hráč väčšiu oblasť na pohyb bez potreby generovania nových segmentov (viď obr 25). To však vedie k zvýšenej spotrebe pamäte a procesorového času kvôli väčšiemu počtu segmentov, ktoré musia byť spracované.



Obr. 23: Ukážka rozdelenia sveta na segmenty označené súradnicami. Hráč je umiestnený na zvýraznenom segmente s označením 0;0. Prevzaté z [45].



Obr. 24: Ukážka aktualizácie herného prostredia pri prechode hráča zo segmentu s koordinátmi 0;0 do segmentu s koordinátmi 1;0. Nové segmenty, označené zelenou farbou, boli pridané do prostredia, zatiaľ čo staré segmenty, označené červenou farbou, boli z prostredia odstránené. Prevzaté z [45].

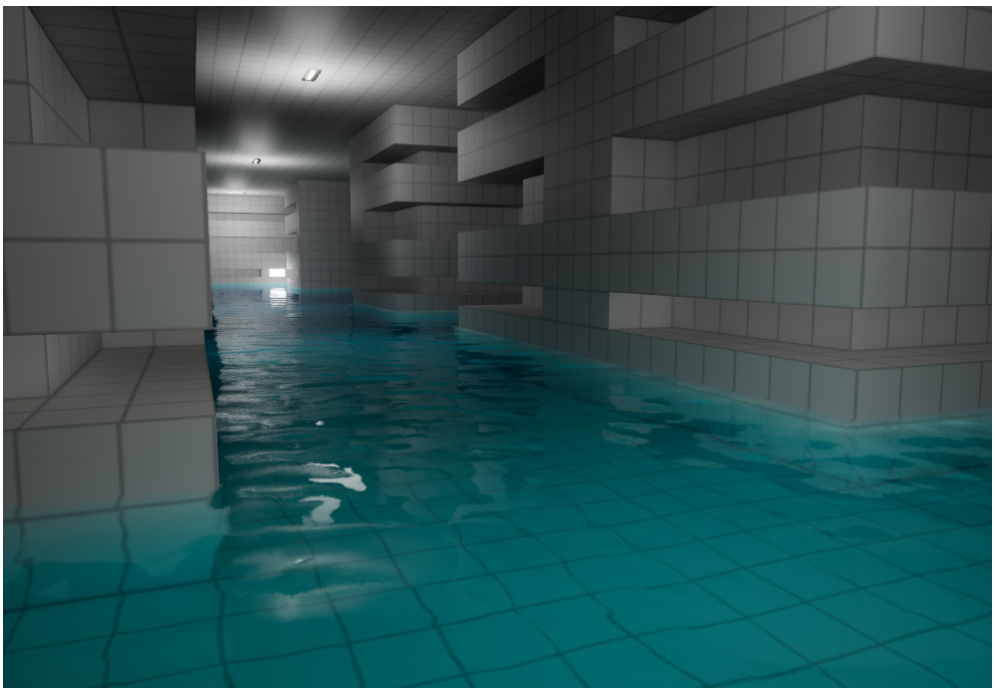


Obr. 25: Ukážka herného prostredia o veľkosti 25 segmentov. Modrou farbou sú označené hraničné segmenty. Hráč sa nachádza na šedom segmente so súradnicami 0;0. Pri vstupe do modrých segmentov sa vykoná aktualizácia prostredia. Prevzaté z [45].

Ak by sme umiestnili hranicu pre generovanie na okraj (viď obr. 24), mohlo by sa stať, že hráč uvidí prázdnu časť sveta ešte pred tým, než sa vygeneruje nová oblasť. Aby sme tomuto predišli, posunieme hranicu generovania dovnútra od okraja, čím vytvoríme rezervu, ktorá zabezpečí, že hráč neuvidí nevygenerovanú oblasť prostredia. V hrách sa často využívajú rôzne techniky a vizuálne efekty, ako je napríklad hmla, ktoré obmedzujú videnie hráča, aby skryli nevygenerované časti prostredia. [45]

6.3 Wave function collapse generátor

Jedným z implementovaných spôsobov a prvým pokusom o vygenerovanie prostredia bolo úplne náhodné generovanie. Generátor vytváral prostredie náhodným vyberaním modelu a jeho rotácie pre každé miesto na mriežke. Takéto generovanie by fungovalo relatívne dobre pokiaľ by sa na tvorbu sveta používali modely, ktoré by hráča nezablokovali v pohybe. Najvhodnejším na vytváranie prostredia bol algoritmus TWFC (viď kap. 3.2), ktorý generuje 3D prostredie z manuálne vytvorených miestností (viď obr. 27). Uzavreté miestnosti a terén, ktorý nie je veľmi členitý (viď obr. 26) boli jedným z dôvodov, prečo bol zvolený pre implementáciu. WFC umožňuje generovať vzory vo vysokej kvalite a je schopný pracovať s rôznymi typmi vstupov a generovať rôzne typy výstupov. Vďaka nastaveniu vstupných pravidiel a podmienok, ako aj implementovanému editoru, má aj samotný používateľ kontrolu nad generovaným obsahom.



Obr. 26: Prostredie z implementovanej hry

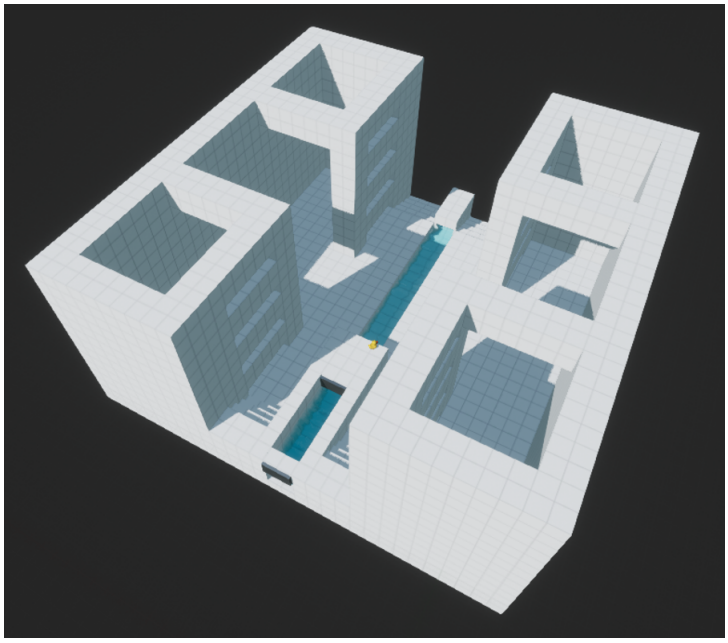
6.4 Implementačné detaily

Pre minimalizovanie záseku hráča pri generovaní prostredia sa výpočet entropie a backtracking vykonáva asynchrónne. Vytváranie a odstraňovanie miestností sa v Unreal Engine musí vykonávať na hlavnom hernom vlákne. Veľkosť miestností je optimalizovaná tak, aby neboli príliš veľké a prázdne. Na druhej strane, ak by boli príliš malé, hráč by ich rýchlo prechádzal, čím by sa zrýchlila potreba generovať nové časti prostredia. Grafickú optimalizáciu zabezpečuje herný engine (viď kap. 4). Ak by sme sa pozerali na prostredie zhora, dochádzalo by k zásekom kvôli renderovaniu značnej časti prostredia. Z hráčovej perspektívy sa však renderujú len okolité miestnosti, zatiaľ čo ostatné miestnosti sú z renderovania vyradené. V hre je predvolený rozsah prostredia pri generovaní nastavený na 7×7 segmentov, pre optimálnu rýchlosť načítavania sveta bez zásekov. Hranica generovania je nastavená na polovicu rozmeru mriežky, 3×3 horizontálne a vertikálne po prejdení dvoch poschodí. WFC generátor obsahuje pomocné funkcie na sledovanie smeru pohybu hráča a aktualizáciu jeho súradníc. Generátor kontroluje či hráč prešiel polovicu vygenerovanej časti prostredia. V menu je možnosť nastaviť rozsah generovaného prostredia v rozmedzí od 7×7 do 15×15 segmentov. Čím väčší je zvolený rozsah, tým pomalšie sa svet generuje. Algoritmus musí riešiť entropiu väčšieho počtu segmentov a väčší počet chybových stavov. V smere osi z sa vždy generuje 5 segmentov nad sebou (5 poschodí). Pri nastaveniach rozsahu $15 \times 15 \times 5$ sa naraz vygeneruje 1125 miestností. Algoritmus niekedy vytvára dlhé chodby alebo iné formácie miestností, ktoré odhaľujú nevygenerované oblasti sveta. Preto si používateľ môže v grafických nastaveniach hry zapnúť hmlu. Hmla nevygenerované oblasti zakrýva a zároveň funguje ako occluder, ktorý skrýva vzdialené miestnosti a zabraňuje ich vykresľovaniu, čo zlepšuje výkon hry (viď kap. 4.1).

6.5 Miestnosti

Jednotlivé miestnosti predstavujú segmenty (viď kap. 6.2). Pri spustení hry sa zvolené nastavenie analyzuje a vytvoria sa pravidlá pre generátor. Bunky mriežky aktualizujú svoju entropiu na základe von Neumanového okolia (viď kap. 2.1). Miestnosti nemajú definovanú dodatočnú pravdepodobnosť s akou sa majú generovať. Hra štandardne používa prednastavené konfigurácie miestností na generovanie prostredia. Ak sú pravidlá správne nastavené, algoritmus dokáže vždy vyriešiť chybný stav pomocou backtrackingu (viď kap. 3). Ak backtracking zlyhá, algoritmus sa niekoľkokrát pokúsi znovu vygenerovať celý svet. Ak aj tieto pokusy zlyhajú, hra vráti hráča do hlavného menu. Backtracking môže znovu vygenerovať značnú časť prostredia, vrátane aktuálnej pozície hráča. Generátor dodržiava nielen lokálne pravidlá susedností, ale aj globálne pravidlo. Pravidlo zaručuje vygenerovanie priechodných miestností v mieste spawnu hráča, čo zabráni jeho zaseknutiu. Prepojenie poschodí nie je zaručené, pretože chýba globálne pravidlo zabezpečujúce existenciu schodov. V prípade generovania bludiska to však nepredstavuje problém. Rovnako neexistuje pravidlo, ktoré by kontrolovalo vý-

skyt hráča v cyklickej formácii miestností. Ak sa hráč dostane do miestností bez východiska (môže nastať v dôsledku backtrackingu alebo vytvorenia cyklu), stačí resetovať úroveň cez pause menu alebo tlačidlom na reštart levelu. Niektoré miestnosti majú dva dizajny, z ktorých sa náhodne jeden vyberie pri jej vygenerovaní.

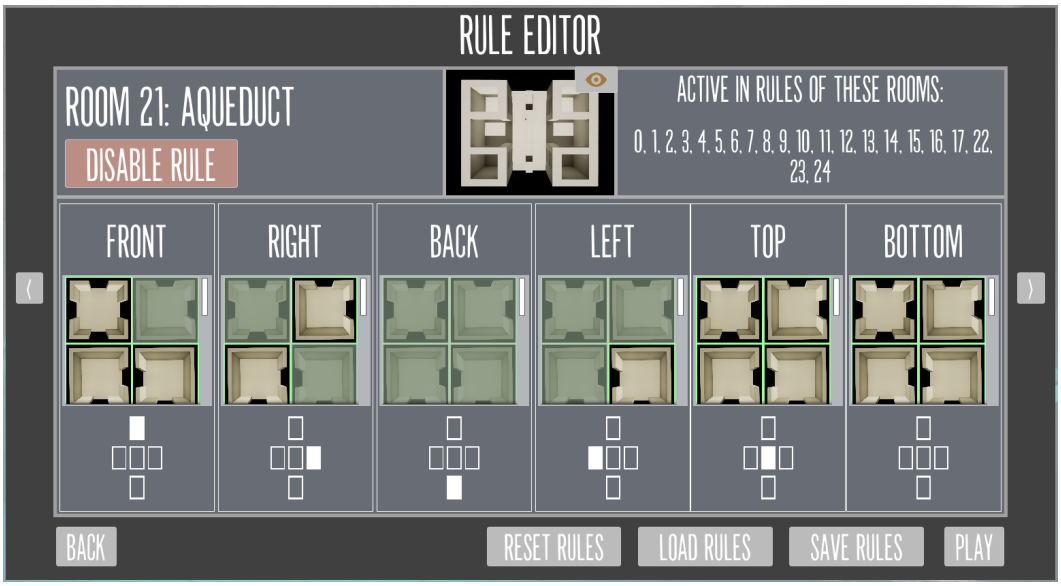


Obr. 27: Ukážka miestnosti použitej v generovaní prostredia.

Editor

Hra obsahuje editor (viď obr. 28), ktorý užívateľovi umožňuje vybrať miestnosti, z ktorých sa bude generovať. Ďalej umožňuje nastavovať zvoleným miestnostiam pravidlá susedností.

Editor sa otvorí v hlavnom menu po výbere rozsahu prostredia. Editor na začiatku obsahuje prednastavené pravidlá, ktoré zaisťujú správne prepojenie miestností. Hráč má možnosť pravidlá algoritmu meniť, prípadne nastaviť, ktoré miestnosti sa budú generovať. Hráč môže manuálne zmeniť napríklad pravidlo pre miestnosť 1, ktorým určí aký typ miestnosti k nej bude napojený. Na toto pravidlo editor automaticky nastaví proti pravidlo, ktoré zaistí, že vybrané miestnosti budú mať miestnosť 1 zvolenú z opačnej strany. To znamená, že ak je manuálne zvolené pravidlo, ktorým určíme, že miestnosť 1 bude mať vpravo miestnosť 4, proti pravidlo umožní, aby sa v miestnosti 4 generoval priechod na ľavej strane do miestnosti 1. V editore sa neustále zosúlaďujú jednotlivé pravidlá pre susediace miestnosti a hráč vidí, ktoré miestnosti má v danom pravidle zapnuté alebo vypnuté (viď obr. 28). Editor disponuje ďalšími funkciami ako uloženie vytvorených pravidiel a ich následné načítanie alebo ich reset. V editore sa dajú uložiť až 4 rozličné nastavenia pravidiel.



Obr. 28: Ukážka editora TWFC algoritmu. Na obrázku je vidieť nastavenie jednej z preddefinovaných miestností. Miestnosti sa v editore prepínajú pomocou šípiek. ROOM 0 je názov miestnosti, tlačidlo Disable rule slúži pre vypnutie. Vypnuté miestnosti nebudú v generovaní použité. Vpravo sú vypísané miestnosti, v ktorých je ROOM 0 aktívna a zahrnutá v pravidlách. Spodná časť sa skladá zo 6 selektorov, kde každý predstavuje jednu stranu miestnosti. Zvýraznené pôdorysy miestností sú aktívne a algoritmus ich vygeneruje vedľa vybranej miestnosti. Vyblednuté pôdorysy nie sú do pravidiel zahrnuté. Editor bráni vyradeniu všetkých miestností z generovania a zo selektora. Pôdorysy miestností zobrazujú len základnú štruktúru a umiestnenia prechodov. Pre lepšiu vizualizáciu editor disponuje 3D náhľadom miestností, ktoré je možné otáčať.

Záver

V oblasti vývoja počítačových hier zohráva procedurálne generovanie kľúčovú úlohu pri vytváraní rozsiahlych a detailných herných svetov, ktoré presahujú možnosti tradičných metód tvorby obsahu. Táto technika umožňuje automatické generovanie textúr, modelov a prostredí. Tejto problematike bola venovaná úvodná teoretická časť práce. Ďalej boli popísané jednotlivé metódy generovania a algoritmy, ktoré sa využívajú pri tvorbe obsahu. Tretia kapitola bola venovaná WFC algoritmu, ktorý bol kľúčovým pri generovaní prostredia v praktickej časti práce. Inšpiráciou pri navrhovaní obsahu bol koncept The Backrooms, ktoré predstavujú nekonečný labyrint miestností a chodieb.

V praktickej časti práce bolo využité procedurálne generovanie 3D prostredia pomocou Unreal Engine. V Unreal Engine boli vytvorené základné stavebné bloky, miestnosti a chodby, ktoré používal algoritmus WFC pri generovaní na základe preddefinovaných pravidiel. Vytvorenie komplexných pravidiel je dôležité pre zachovanie hrateľnosti hry, to však môže byť niekedy zložité. Navrhli sme automatický systém pravidiel, globálne obmedzenie a obmedzenie vzdialenosti pre zabezpečenie kontrolovateľného vytvárania prostredia. Globálne pravidlo bolo použité pri spawne hráča, aby sa predišlo jeho uviaznutiu v textúrach.

Na záver tejto práce by som rád poukázal na niektoré aspekty, ktoré by mohli byť v budúcnosti vylepšené. Jedným z hlavných zlepšení mohla byť implementácia viacvrstvového generovania obsahu. Herné prvky na rôznych vrstvách sa môžu prekrývať. Zvyčajne nepodliehajú pravidlám o susednostiach, ale vyžadujú medzivrstvové obmedzenia. Ak by bola herná scéna zložená z viacerých vrstiev herných prvkov s rôznymi významami, tak by bolo potrebné generovanie po vrstvách. Napríklad, ak by jedna vrstva predstavovala prostredie a druhá nepriateľov alebo interaktívne predmety. Dosiahnuť takéto vrstvenie pomocou pôvodného algoritmu WFC by bolo náročné. V našom prípade bola generovaná jedna vrstva, ktorá sa dá zvyčajne aplikovať len na jednoduché typy herných scén. V Unreal Engine môže byť správne nastavenie svetiel komplikované a v hre by mohlo byť lepšie zvolené. Napríklad, svetlá často prestávajú svietiť z určitých uhlov alebo vzdialeností, čo môže narušiť atmosféru a plynulosť osvetlenia v hre.

Návrat z chybných stavov vo WFC by sa dal zefektívniť využitím sofistikovanejších techník namiesto backtrackingu, ako je backjumping (spätné skokové prehľadávanie) alebo modifying in blocks (blokové úpravy). Pomocou backjumping techniky by počet krokov, o ktoré sa vraciame, rástol kvadraticky. Pri opakovaných konfliktoch by sme sa vracali o stále väčší počet krokov. Technika modifying in blocks by rozdelila veľkú generujúcu sa oblasť na niekoľko menších prekrývajúcich sa blokov, ktoré by sa generovali jeden po druhom. V prípade konfliktu by sa reštartoval iba daný blok, nie celá oblasť. Tento prístup by zabezpečil priechodnosť prostredia vďaka prekrývaniu blokov, čím by sa eliminovala nutnosť úplného reštartu a zlepšila efektivita generovania.

Conclusions

In the field of game development, procedural generation plays a crucial role in creating detailed game worlds that exceed the capabilities of traditional content creation methods. This technique allows for the automatic generation of textures, models, and environments. The introductory theoretical part of this thesis was dedicated to this topic. Furthermore, various generation methods and algorithms used in content creation were described. The third chapter focused on the WFC algorithm, which was key in generating the environment in the practical part of the thesis. The concept of The Backrooms, an endless labyrinth of rooms and corridors, served as the inspiration for content design.

In the practical part of this work, procedural generation of a 3D environment was implemented using Unreal Engine. Basic building blocks, rooms, and corridors were created in Unreal Engine and used by the WFC algorithm for generation based on predefined rules. Creating complex rules is crucial for maintaining the playability of the game, but this can sometimes be challenging. We proposed an automatic rule system, global constraints, and distance constraints to ensure controllable environment creation. A global rule was applied during the player's spawn to prevent him from getting stuck in textures.

In conclusion, I would like to highlight some aspects that could be improved in the future. One of the main enhancements could be the implementation of multi-layer content generation. Game elements on different layers can overlap. They usually do not adhere to adjacency rules but require inter-layer constraints. If the game scene were composed of multiple layers of game elements with different significances, it would necessitate layered generation. For instance, one layer could represent the environment, while another could represent enemies or interactive objects. Achieving such layering with the original WFC algorithm would be challenging. In our case, only a single layer was generated, which can typically be applied only to simpler types of game scenes. Additionally, correctly setting up lights in Unreal Engine can be complicated and could be improved in the game. For example, lights often stop illuminating from certain angles or distances, which can disrupt the atmosphere and smoothness of the game's lighting.

The efficiency of resolving erroneous states in the WFC algorithm could be significantly improved by using more sophisticated techniques, such as backjumping or modifying in blocks. With the backjumping technique, the number of steps we backtrack would grow quadratically. In the case of repeated conflicts, we would backtrack an increasingly larger number of steps. The modifying in blocks technique would divide the large generating area into several smaller overlapping blocks that would be generated one by one. If a conflict occurred, only the specific block would be restarted, not the entire area. This approach would ensure the passability of the environment due to the overlapping blocks, thereby eliminating the need for a complete restart and improving the efficiency of the generation process.

A Obsah elektronických dat

bin/
: Program hry TheBackrooms_37.exe. Zložka obsahuje všetky potrebné súbory pre bezproblémový beh hry.

doc/
: Text práce vo formáte PDF, vytvorený s použitím závazného štýlu KI PRF UP v Olomouci pre záverečné práce, vrátane všetkých príloh, a všetkých súborov potrebných pre bezproblémové vygenerovanie PDF dokumentu textu (v ZIP archive), tj. zdrojový text textu, vložené obrázky, apod.

src/
: Kompletné zdrojové kódy hry TheBackrooms_37.exe so všetkými potrebnými (príp. prevzatými) zdrojovými textami, knižnicami a dalšími súbormi potrebnými pre bezproblémové vytvorenie spustitelných verzií hry.

readme.txt
: Inštrukcie pre spustenie hry TheBackrooms_37.exe, vrátane všetkých požiadavkov pre jej bezproblémové fungovanie.

demo/
: Ukážka správneho fungovania hry.

Literatúra

[1] SHORT, Tanya; ADAMS, Tarn. Procedural Generation in Game Design [online]. 2017-06-12. [cit. 2023-01-21]. ISBN 978-1-4987-9919-5. Dostupné z: https://www.taylorfrancis.com/books/edit/10.1201/9781315156378/procedural-generation-game-design-tanya-short-tarn-adams

[2] SHAKER, Noon; TOGELIUS, Julian; NELSON, Mark. Procedural Content Generation in Games [online]. 2016-10-18. [cit. 2023-01-21]. ISBN 978-3-319-42714-0. Dostupné z: https://link.springer.com/book/10.1007/978-3-319-42716-4

[3] FREIKNECHT, Jonas; EFFELSBERG, Wolfgang. A Survey on the Procedural Generation of Virtual Worlds [online]. 2017-10-30. [cit. 2023-03-04]. Dostupné z: https://doi.org/10.3390/mti1040027

[4] BORIS. Dungeon Generation in Diablo 1 [online]. 2019-07-14. [cit. 2024-07-20]. Dostupné z: https://www.boristhebrave.com/2019/07/14/dungeon-generation-in-diablo-1/

[5] STANTON, Lee. How Minecraft Generates Worlds [online]. 2023-08-01. [cit. 2024-07-20]. Dostupné z: https://www.alphr.com/how-minecraft-generates-worlds/

[6] KAZEMI, Darius. Spelunky Generator Lessons [online]. [cit. 2024-07-19]. Dostupné z: https://tinysubversions.com/spelunkyGen/

[7] MANEKOWARE. Catlateral Damage [online]. 2015-5-27. [cit. 2024-07-20].

[8] COMPTON, Kate et al. Generative Methods. Atlas of Digital Architecture, 145–174 [online]. 2022. [cit. 2023-02-13]. Dostupné z: https://www.academia.edu/69211262

[9] BIAGIOLI, Adrian. Understanding Perlin Noise [online]. 2014-08-09. [cit. 2024-07-21]. Dostupné z: https://adrianb.io/2014/08/09/perlinnoise.html

[10] SUMMERVILLE, Adam et al. Procedural Content Generation via Machine Learning (PCGML) [online]. 2018-06-12. [cit. 2023-02-05]. Dostupné z: https://ieeexplore.ieee.org/document/8382283.

[11] KHALIFA, Ahmed; TOGELIUS, Julian. Marahel: A Language for Constructive Level Generation. AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment, 13(2), 84-91 [online]. 2021-06-25. [cit. 2024-04-21]. Dostupné z: https://doi.org/10.1609/aiide.v13i2.12970

[12] ROGERS, REECE. How to 'No-Clip' Reality and Arrive in the Backrooms [online]. 2022-05-11. [cit. 2023-01-15]. Dostupné z: https://www.wired.com/story/what-are-the-backrooms/

[13] KIRA. Lost in Vivo [online]. 2018-11-05. [cit. 2023-01-15].

[14] ZEEKERSS. Lethal Company [online]. 2023-10-23. [cit. 2024-05-05].

[15] DEVEYERR. Level 0: "The Lobby"[online]. 2022-12-01. [cit. 2023-01-15]. Dostupné z: https://backrooms.fandom.com/wiki/Level_0

[16] WWW.BACKROOMS-WIKI.WIKIDOT.COM. How To Write A Level [online]. 2022-08-09. [cit. 2023-01-15]. Dostupné z: http://backrooms-wiki.wikidot.com/how-to-write-a-level

[17] WWW.BACKROOMS-WIKI.WIKIDOT.COM. Level 6 [online]. 2023-01-13. [cit. 2023-01-15]. Dostupné z: http://backrooms-wiki.wikidot.com/level-6

[18] WWW.BACKROOMS-WIKI.WIKIDOT.COM. Level 6.2 [online]. 2023-11-30. [cit. 2023-03-18]. Dostupné z: https://backrooms-wiki.wikidot.com/level-6-2

[19] WWW.BACKROOMS-WIKI.WIKIDOT.COM. Level 17 [online]. 2024-01-4. [cit. 2023-03-17]. Dostupné z: https://backrooms-wiki.wikidot.com/level-17

[20] WWW.BACKROOMS-WIKI.WIKIDOT.COM. Level 37: "Sublimity"[online]. 2022-04-10. [cit. 2023-01-15]. Dostupné z: http://backrooms-wiki.wikidot.com/level-37

[21] RABIN, Steven et al. Game AI Pro 2: Collected Wisdom of Game AI Professionals [online]. 2015-03-12. [cit. 2024-02-22]. ISBN 9780429160790. Dostupné z: https://doi.org/10.1201/b18373

[22] GUMIN, Maxim. mxgmn/WaveFunctionCollapse [online]. 2022-07-21. [cit. 2024-02-22]. Dostupné z: https://github.com/mxgmn/WaveFunctionCollapse

[23] ARABNIA, R. Hamid et al. Advances in Artificial Intelligence and Applied Cognitive Computing [online]. 2021-10-14. [cit. 2024-02-24]. ISBN 978-3-030-70296-0. Dostupné z: https://doi.org/10.1007/978-3-030-70296-0

[24] PANDEY, Vishal et al. Wave Function Collapse Visualisation [online]. 2022-07-08. [cit. 2024-02-25]. Dostupné z: https://vixra.org/pdf/2207.0062v1.pdf

[25] HWANHEE, Kim et al. Automatic Generation of Game Content using a Graph-based Wave Function Collapse Algorithm [online]. 2019-09-26. [cit. 2024-02-28]. ISBN: 978-1-7281-1884-0. Dostupné z: https://doi.org/10.1109/CIG.2019.8848019

[26] DARUI, Cheng et al. Automatic Generation of Game Levels Based on Controllable Wave Function Collapse Algorithm [online]. 2021-01-05. [cit. 2024-03-3]. ISBN: 978-3-030-65735-2. Dostupné z: https://doi.org/10.1007/978-3-030-65736-9_3

[27] NIE, Yuan et al. Extend Wave Function Collapse to Large-Scale Content Generation [online]. 2023-8-14. [cit. 2024-03-08]. Dostupné z: https://doi.org/10.48550/arXiv.2308.07307

[28] EUN-SEOK, Lee; BYEONG-SEOK, Shin. Vertex Chunk-Based Object Culling Method for Real-Time Rendering in Metaverse [online]. 2023-06-09. [cit. 2024-03-06]. Dostupné z: https://doi.org/10.3390/electronics12122601

[29] LIANG, Yuzhi et al. Automatic Mesh and Shader Level of Detail [online]. 2022-07-06. [cit. 2024-03-06]. Dostupné z: https://doi.org/10.1109/TVCG.2022.3188775

[30] PURNOMO, Budirijanto et al. Seamless texture atlases [online]. 2004-07-08. [cit. 2024-03-06]. Dostupné z: https://doi.org/10.1145/1057432.1057441

[31] LUCAS, M. Simon et al. Artificial and Computational Intelligence in Games [online]. 2013-11-18. [cit. 2024-03-07]. Dostupné z: https://doi.org/10.4230/DFU.Vol6.12191

[32] TOGELIUS, Julian et al. Search-Based Procedural Content Generation: A Taxonomy and Survey. IEEE Transactions on Computational Intelligence and AI in Games, vol. 3, no. 3, pp. 172-186 [online]. 2011-04-11. [cit. 2024-07-10]. Dostupné z: https://doi.org/10.1109/TCIAIG.2011.2148116

[33] DICKINSON, Chris. Unity 2017 Game Optimization: Optimize all aspects of Unity performance, Second Edition [online]. 2017-11-22. [cit. 2024-03-08]. Dostupné z: https://www.packtpub.com/en-us/product/unity-2017-game-optimization-9781788392365

[34] NYSTROM, Robert. Game Programming Patterns [online]. 2014-11-02. [cit. 2024-03-08]. Dostupné z: https://gameprogrammingpatterns.com/

[35] MORGAN, Thomas. Tech Analysis: No Man's Sky [online]. 2016-8-11. [cit. 2024-03-08]. Dostupné z: https://www.eurogamer.net/digitalfoundry-2016-no-mans-sky-tech-analysis

[36] MAULANA, Firman. Real-time Landscape Generation in Games Using Parallel Procedural Content [online]. 2020-11-17. [cit. 2024-03-09]. ISBN: 978-1-7281-8283-4. Dostupné z: https://doi.org/10.1109/CENIM51130.2020.9297946

[37] THOMPSON, Tommy. How Townscaper Works: A Story Four Games in the Making [online]. 2022-04-06. [cit. 2024-03-17]. Dostupné z: https://www.gamedeveloper.com/game-platforms/how-townscaper-works-a-story-four-games-in-the-making

[38] LICHTENEGGER, Klaus; SCHAPPACHER, Wilhelm. Phase Transition in a Stochastic Forest Fire Model and Effects of the Definition of Neighbourhood [online]. 2009-03-23. [cit. 2024-03-29]. Dostupné z: https://doi.org/10.1142/S0129183109014321

[39] SHIFFMAN, Daniel. Cellular Automata [online]. [cit. 2024-03-29]. Dostupné z: https://natureofcode.com/cellular-automata/

[40] BUCK, Jamis. Mazes for Programmers: Code Your Own Twisty Little Passages [online]. 2015-08-18. [cit. 2024-04-7]. ISBN: 978-1-6805-0055-4 Dostupné z: https://pragprog.com/titles/jbmaze/mazes-for-programmers/

[41] LONG, Zhongjie; NAGAMUNE, Kouki. A Marching Cubes Algorithm: Application for Three-dimensional Surface Reconstruction Based on Endoscope and Optical Fiber [online]. 2015-04-01. [cit. 2024-04-7]. Dostupné z: https://inria.hal.science/hal-01205823

[42] PLAUSIBLE CONCEPT; STÅLBERG, Oskar et al. Bad North [online]. 2018. [cit. 2024-04-12]. Dostupné z: https://www.badnorth.com/

[43] FANCY GAMES. Escape the Backrooms [online]. 2022-07-22. [cit. 2024-04-16].

[44] EPIC GAMES. Unreal Engine 5.4 Documentation [online]. [cit. 2024-02-26]. Dostupné z: https://dev.epicgames.com/documentation/en-us/unreal-engine/unreal-engine-5-4-documentation

[45] ENGBERG, Jackie. Infinite World using Procedural Content Generation [online]. 2016-04-25. [cit. 2024-02-26]. Dostupné z: https://www.jacksendary.dk/Papers/Infinite%20Worlds%20using%20Procedural%20Content%20Generation.pdf