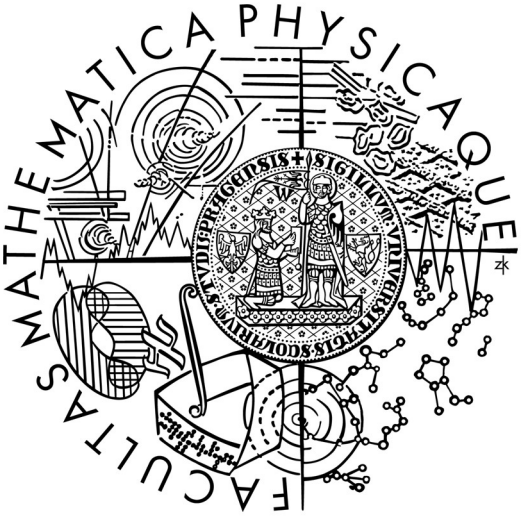
Univerzita Karlova v Praze

Matematicko-fyzikální fakulta

BAKALÁŘSKÁ PRÁCE



Štefan Pacinda

Implementace RPG hry Dračí Doupě

Středisko informatické sítě a laboratoří

Vedoucí bakalářské práce: Mgr. Lenka Forstová

Studijní program: Informatika, Programování,

2008

Děkuji paní Mgr.Lence Forstové za její podporu a velkou trpělivost při vedení této práce.

Prohlašuji, že jsem svou bakalářskou práci napsal samostatně a výhradně s použitím citovaných pramenů. Souhlasím se zapůjčováním práce.

V Praze dne 5.8.2008 Štefan Pacinda

OBSAH

Název práce: Implementace RPG hry Dračí doupě

Autor: Štefan Pacinda

Katedra (ústav): Středisko informatické sítě a laboratoří

Vedoucí bakalářské práce: Mgr. Lenka Forstová

E-mail vedoucího: forstova@sisal.mff.cuni.cz

Abstrakt:

V rámci ročníkového projektu byla vytvořena hra na motivy Dračího doupěte. Cílem bakalářské práce je popsat a zformalizovat herní rámec hry implementované jako ročníkový projekt, na základě této formalizace poté navrhnout, vytvořit a detailně popsat postup, kterým bude možné automaticky generovat rámec hry pouze ze zadaných parametrů. Součástí práce je formální popis vytvořených algoritmů a rozbor technických podrobností implementace vytvořeného systému.
Kromě samostatně pracujícího systému generování herního prostředí je vyvinuta uživatelsky orientovaná aplikace poskytující grafické rozhraní samočinnému generátoru. Kromě toho tato aplikace nabízí nástroje pro manuální úpravy již vygenerovaného herního rámce.

Klíčová slova: Dračí doupě, Generování prostředí počítačové hry

Title: Implementation of the Dungeons & Dragons Roleplaying Game

Author: Štefan Pacinda

Department: Network and Labs Management Center

Supervisor: Mgr. Lenka Forstová

Supervisor's e-mail address: forstova@sisal.mff.cuni.cz

Abstract:

In the scope of the recent Academic Year Project a computer game, loosely inspired by the Czech game "Dračí Doupě", was created. "Dračí Doupě" is a role-playing game and its main principles, such as draw-oriented game flow or development of the main character's abilities, were adopted into our game.
The aim of this Bachelor Thesis is to formalize the game scope of our implementation. After formalizing, we will focus on designing a representation of the game framework. The main objective is, however, creating a tool capable of generating the game framework automatically, after receiving a set of required parameters.
Attention shall be paid to the formal description of the approach in which the game data will be generated. Namely, we will detail all algorithms developed within the project and discuss their characteristics. We shall also define objects, and the relationships between them, to model the world of the game.
A graphically-oriented application was developed on top of the automatically working game engine. This application allows us to easily present the results of generating the game data. Moreover, the additional functionality of manual editing is integrated within the application.
Keywords: Dungeons and Dragons, Computer Game Framework Generating

Úvod

Pohled do historie dračího doupěte

Hra Dračí doupě se stala populární v devadesátých letech dvacátého století.
Dračí doupě je původní česká fantasy hra na hrdiny, kterou v roce 1990 vytvořilo nakladatelství ALTAR. Hra je inspirována americkou hrou Dungeons & Dragons.

V roce 2004 vyšla nová hra, určená spíše pokročilejším hráčům, zvaná Dračí doupě Plus.

Základní principy hry

Dračí doupě patří mezi klasické (nepočítačové) hry na hrdiny (role-playing games). Jeden z účastníků, tzv. Pán jeskyně (PJ), před hrou připraví úvodní příběh, mapy apod. Ostatní hráči se poté tohoto příběhu zúčastní – každý z hráčů představuje jednu postavu fantasy světa (válečníka, kouzelníka apod.), celá družina pak dohromady putuje světem vytvořeným PJ s cílem splnit zadané úkoly, získat peníze a slávu. Obecně neexistují limity, co hráči mohou a nemohou, o následcích jejich činů rozhoduje PJ, některé běžné herní situace (boj, sesílání kouzel, využívání zvláštních schopností apod.) se řeší na základě číselných tabulek uvedených v pravidlech a hodů kostkami (používají se klasické šestistěnné a o něco netradičnější desetistěnné).

pozn: text „Základní principy hry“ pochází z wikipedia.cz

Bakalářská práce a ročníkový projekt

Hra Dračí doupě je velice spletitá a volně definovaná. Její pravidla pouze ukazují směr, jejich výklad však může být nejasný a definuje ho s konečnou platností až Pán jeskyně. Mnoho částí hry je z pohledu informatiky zajímavých, je ovšem potřeba je popsat formálněji, než jak činí pravidla Dračího doupěte.
Pro potřeby bakalářské práce a ročníkového projektu jsme se inspirovali Dračím doupětem jako typem hry. Přejali jsme některé jeho rysy, jako tahový styl hry či diskrétní herní prostor Tyto rysy jsou společné snad všem fantasy hrám na hrdiny.
Nesnažíme se ovšem vytvořit implementaci Dračího doupěte, jako počítačové verze nepočítačové hry.

V rámci ročníkového projektu byla vytvořena hra na motivy Dračího doupěte. Cílem bakalářské práce je popsat a zformalizovat herní rámec hry implementované jako ročníkový projekt. Na základě této formalizace bude navržen, vytvořen a popsán

algoritmus generování nového herního rámce. Generování bude možné provádět zcela automaticky pouze ze zadaných parametrů.
Kromě toho bude k dispozici uživatelsky orientovaná aplikace s grafickým výstupem poskytující nástroje pro manuální úpravy již vygenerovaného herního rámce.
Během prací na formalizaci herního rámce hry implementované jako ročníkový projekt jsme mírně změnili formát dat, který obě aplikace sdílejí.

Oddíl první

Očima uživatele

V kapitolách tohoto úvodního oddílu představíme počítačovou implementaci hry a herní svět pro potřeby uživatele.
Seznámíme se s pravidly, která jsme převzali a přetvořili.
Zmapujeme rozsah herního rámce, souvislosti a interakce mezi jeho částmi. Zaměříme se na aktivity, které může hráč jakožto hrdina provádět. Vysvětlíme jejich dopad a podmínky jejich vykonání.
Dále se budeme zabývat generátorem herního prostředí. Ukážeme si jeho fungování a jeho použití z pohledu prakticky orientovaného čtenáře, který nemá potřebné znalosti v oblasti vývoje software ani se neorientuje v terminologii a principech teoretické informatiky popřípadě teorie grafů. Funkcionalitu systému nicméně představíme v celé šíři.
Odhlédneme pouze od implementace a implementačních detailů. Do hloubky budeme v popisu postupovat až v oddílech dvě a tři.

Kapitola 1

Popis počítačové implementace hry

Tato úvodní kapitola se zaměří na popis herního rámce, který jsme vytvořili pro počítačovou implementaci hry. Přiblížíme charakteristické vlastnosti hry, důležité situace a možné akce, které mohou být prováděny. Rozebereme všechny typy entit, které se v tomto namodelovaném světě nacházejí.

1.1. Jak se hra hraje

Tato část popíše cíl hry a aktivity, které je možné vykonávat. Pozornost bude věnována tahovému stylu hry a jeho dopadu na dynamiku hry.

Herní prostředí

Herním prostředím je v této hře grafická aplikace používající klasické rozhraní oken a ovládací prvků windows. Většina událostí a obsluha hry probíhá v hlavním okně.

1.1.1.Cíl

Implementovaná podmnožina světa Dračího doupěte je vymezena na dobrodružství v jeskyni.
Cílem je porazit hlavního protivníka v daném labyrintu. K tomu je nezbytně nutné přejít úspěšně přes překážky nastražené v celém bludišti. Hlavní protivník se totiž nachází až na nejméně dostupném místě celého labyrintu.

V průběhu hry ovšem hrdina narazí i na vedlejší protivníky (alespoň co do významu v rámci celého dobrodružství). Je jen na něm, zda s nimi bude bojovat, nebo se jim pokusí vyhnout.

Bezpochyby je velmi užitečné cestou zlepšovat kvality a schopnosti hrdiny sbíráním předmětů a jejich používáním.

1.1.2.Tahový styl hry

Hlavním rysem samotné předlohy je tahový ráz. Ve všech důležitých situacích se děj odehrává po časových úsecích, takzvaných tazích.

Tah

Jeden tah je základní a dále nedělitelnou jednotkou času. Během jednoho tahu je možné provést jednu jedinou akci. Tempo hry tedy není konstatní, ale udává ho hráč rychlostí odehrání svých vlastních tahů . Tahový princip se samozřejmě uplatňuje i na nestvůry. Nestvůry popíšeme dále.

Kolo

Kolo je časový úsek hry sestávající se z několika tahů. Tahy jsou v jednom kole prováděny těsně za sebou až do vyčerpání posledního. Všechny tahy jsou pochopitelně prováděny právě jednou postavou zároveň, tedy hrdinou či právě jednou z nestvůr.

Jednotlivá kola jsou blokující. Hráč, či nestvůra, která je na tahu provede všechny své tahy bezprostředně za sebou. Ostatní čekají nečinně, než se na ně dostane řada. Strany se střídají v provádění svých kol pravidelně dokola.Celou hru zahájí svým kolem hrdina.

Počet tahů na kolo

Hrdina disponuje stejným počtem tahů v každém kole. Počet tahů se zvyšuje s přibývajícími úrovněmi. Na začátku hry má hrdina k dispozici 5 tahů na kolo.
Nepřátelé mají všichni fixní počet tahů na kolo, který odpovídá jejich úrovni.

1.1.3.Možné akce

Jak již bylo zmíněno v popisu tahového způsobu hry, je tah nejmenší časovou jednotkou a během jeho trvání lze vykonat právě jednu ze základních akcí. Níže v této části popíšeme jednotlivé akce, které lze provádět. Některé akce souvisí s předměty, které budou blíže popsány až v oddíle 1.2, pro potřeby této části není potřeba jejich detailnější popis.

Pohyb

Pohyb po labyrintu probíhá v diskrétních krocích. Jeden krok je změna pozice ve směru osy x nebo y o jedno pole. Za jeden tah je možné vykonat právě jeden krok.
Podmínkou k vykonání kroku je to, aby cílové pole mělo vlastnost „volný prostor“ (postava by neměla vstupovat do zdi) a aby pole, mezi kterými pohyb probíhá, nebyly odděleny zamčenými dveřmi.

Souboj

Pokud se při provádění svého tahu na sousedním poli nachází nestvůra (respektive hrdina) je možné, aby postava na tahu (hrdina či nestvůra) vyvolala souboj. Jeho průběh podrobněji rozebírá část 4.

Naslouchání

Naslouchání je velice praktická činnost. Lze ji provádět pouze v bezprostřední blízkosti dveří, tedy na poli, které patří do dané místnosti a které dveře spojují s druhou místností. Jejím výsledkem je soupis zvuků, které je slyšet ze sousední místnosti. Tato akce trvá jeden tah.

Sbírání a odkládání předmětů

Předměty je možné v labyrintu nalézt volně, nebo po zabití některých nepřátel v jejich ostatcích. Předměty ležící v labyrintu je možné sebrat, pokud jsou v dosahu jednoho pole ve smyslu pravoúhlých souřadnic (podobně jako při pohybu).
Pokud z nějakého důvodu hrdina nechce dále nést nějaký předmět v inventáři, je možné jej odložit v labyrintu. Tento předmět potom zůstane na aktuální pozici hrdiny v labyrintu.

Odemykání dveří

Každé dvě sousední místnosti v labyrintu jsou odděleny dveřmi. Některé z těchto dveří mohou být zamčené a tudíž neprůchodné. K jejich odemčení je zapotřebí klíč. Klíče nejsou univerzální, ale ke každým zamčeným dveřím existuje právě jeden klíč, který je odmyká.
Odemčení dveří trvá jeden tah.

Užití lektvaru

Jediným typem předmětu, který lze zužitkovat, je lektvar. Použití lektvaru vyléčí udaný počet životů s okamžitým efektem. Každý lektvar lze užít pouze jednou.
Užití lektvaru trvá jeden tah.

Přezbrojení

Výměna zbraně, nebo ochranné pomůcky ovlivní hrdinovy statistiky, jmenovitě útok, respektive obranu. Měnit výstroj lze libovolně dlouho a celé přezbrojení trvá pouze jeden tah (započítá se jen výsledný efekt). K přezbrojení lze využít pouze položek z inventáře.

Spánek

Pokud je hrdina zraněn, může se zotavit nejen použitím lektvarů, ale i spánkem. To ovšem není tak efektivní a může to být nebezpečné. Během spánku může na hrdinu zaútočit libovolná nestvůra. To způsobí hrdinovo probuzení. Spánek trvá libovolně dlouho, ale může uzdravit jen zlomek životů.

Žádná akce

Pokud je v zájmu hrdiny vyčkávat, je možné svůj tah obětovat a v jeho průběhu nedělat nic. To může být někdy taktické. Stejnou volbu má i protivník.

1.1.4. RPG charakter hry

Kromě tahového paradigmatu je nedílnou součástí Dračího doupěte vývoj hrdiny. Tento vývoj probíhá na základě získávání zkušeností za vítězné souboje s nepřáteli. Po dosažení určitého hraničního množství zkušeností se zvýší úroveň hrdiny a zlepší se jeho vlastnosti (počet životů, útok, obrana, síla a obratnost nebo počet tahů na kolo)

1.1.5. Platnost hry

Aktuální stav hry je možné uložit a později znovu načíst. Veškerý kontext a herní informace zůstanou zachovány a posléze tedy mohou být obnoveny.
Data jsou ukládána ve formátu XML.

1.2. Rozsah světa

(co je implementováno a s jakými vlastnostmi)

Rozsahem světa je zde míněna množina objektů v implementovaném herním rámci, se kterými je možno se ve hře setkat. Patří sem vzájemné interakce mezi objekty a také jejich vlastnosti.

1.2.1 Předměty

Jako předmět označujeme tři, respektive čtyři skupiny objektů - zbraně, ochranné pomůcky, lektvary a případně klíče. S prvními třemi kategoriemi je nakládáno v rámci labyrintu shodně, rozdíly jsou patrné až při použití hrdinou. Klíče jsou rozdílné v tom, že je jejich použití nemá žádný efekt na uživatele - odemykají dveře.

Zbraně

Zbraň slouží k boji, přesněji k útoku. Její použití zvyšuje útočné číslo hrdiny. Najednou lze použít nejvýše jednu zbraň. Tato zbraň musí být k dispozici v inventáři. Právě nepoužívané zbraně ovšem nemají žádný efekt (kromě toho, že obsazují volné položky inventáře).
Charakteristickými vlastnostmi zbraně jsou útočné čístlo a útočnost. Útočnost i útočné číslo modelují sílu útoku. Šanci na to, že útok překoná obranu, ovlivňuje pouze útočné číslo.
Příklad: Těžký kyj má malé útočné číslo, ale velkou útočnost. To modeluje vlastnost kyje. Zasáhnout cíl je těžké, ale případný zásah je drtivý.

Ochranné pomůcky

Ochranné pomůcky slouží k obraně. Zvyšují obranné číslo a tím šanci na úspěšné vykrytí útoku protivníka, nebo alespoň snížení jeho dopadu. Ochrannou pomůcku lze použít pouze jednu. Ostatní v inventáři odložené ochranné pomůcky nemají žádný efekt.

Lektvary

Lektvary lze použít k zpětnému obnovení ztracených životů. Lektvar vyléčí udaný počet životů a poté ho již nelze znovu použít.

Klíče

Protože jsou klíče speciálním případem předmětu, budou rozebrány zvlášť později.

Inventář

Hrdina s sebou může přenášet předměty, které vlastní, nebo našel v labyrintu. K tomu slouží inventář. Inventář je v podstatě seznamem předmětů.
V inventáři mohou být všechny typy předmětů avšak omezený počet kusů.
V rámci kapacity inventáře lze nést libovolné množství zbraní, ochranných pomůcek, lektvarů či klíčů a jejich kombinací.
Počet či složení předmětů nemá ve hře žádný postranní efekt.

1.2.2 Dveře a klíče

Mezi některými dvěmi místnostmi v labyrintu se nachází dveře. Dveře se mohou nacházet pouze mezi sousedními dvěmi místnostmi. Některé dveře mohou být navíc zamčené. K odemčení konkrétních dveří musí mít hrdina v inventáři odpovídající klíč. Ke každým zamčeným dveřím se v labyrintu nachází klíč, který je odemyká.
Po použití (odemčení dveří) klíč zmizí a nedá se znovu použít.
V každé místnosti se nacházejí nejméně jedny dveře, nejvýše však troje.
Dveře jsou překážkou v průchodu mezi dvěma poli. Lze přes ně projít pouze pokud jsou odemčené.

1.2.3 Nestvůry

Protivníky nazýváme souhrnně nestvůrami. Nestvůry jsou nepřátelské postavy pohybující se v labyrintu. Jejich cílem je znemožnit, nebo alespoň znesnadnit, hrdinovi cestu za splněním cíle dobrodružství (viz 1.1.1.).

Chování

Aktivita nestvůry závisí na tom, jak byla naprogramováno její rozhodování. Základní nestvůra má preference činností v pořadí - bojovat, pohybovat se, nedělat nic. To znamená, že pokud je to možné, pohybuje se po labyrintu (v rámci komponenty, do které patří) a pokud je v dosahu hrdina, pokusí se na něj zaútočit.
Podmínky pro uskutečnění jednotlivých akcí jsou stejné pro hrdinu jako pro nestvůru, proto platí totéž, jako v části 1.1.3 - Možné akce. Nestvůra ovšem může konat jen akce: 'pohyb', 'souboj' a 'žádná akce'.

Typy nestvůr

Nestvůry se vyskytují v různých variantách a různé síle. Typ nestvůry záleží na oblasti, kde se tato vyskytuje (v hrobkách jsou kostlivci, v bažinách trollové), její síla závisí zase na obtížnosti labyrintu. Síla nestvůry (úroveň) ovlivňuje její charakteristické vlastnosti jako útok, obranu či počet životů. Úroveň nestvůry podmiňuje také např. počet zkušeností, které její poražení hrdinovi přinese.

Hlavní nestvůra

Hlavní nestvůra je definovaná stejně jako ostatní nestvůry. Rozdílem je to, že se vyskytuje vždy v právě jednom exempláři. Její poražení znamená úspěšné dokončení hry. Tato nestvůra se nachází nejhlouběji v labyrintu (ve smyslu dostupnosti).

1.2.4 Místnost, pole

Místnost je základní stavební jednotka labyrintu. Celý labyrint je beze zbytku rozdělen na místnosti. Všechny místnosti jsou obdélníkovými oblastmi. Každá místnost je obroubena zdí, kromě míst, kde ji ke zbytku labyrintu připojují nějaké dveře. Pokud hovoříme o místnosti, máme na mysli celou obdélníkovou oblast včetně okrajové vrstvy zdí.
Po vstupu do místnosti se dostane hráči jejího popisu.
Pole je nejmenší jednotka v labyrintu. Každý objekt ve hře zabírá právě jedno pole(hrdina, nestvůry, předměty). Na jednom poli se může vyskytovat libovolné množství předmětů. Na počátku hry může nastat i situace, kdy je na stejném poli více

nestvůr. Poté, co nestvůry společné pole opustí, nemohou již znovu vstoupit na totéž pole zároveň. Vstoupit na pole lze pouze tehdy, pokud na něm nestojí již jiná postava (platí vzájemně pro nestvůry i hrdinu). Přítomnost předmětů se při pohybu postav nezohledňuje.

1.2.5 Typy místností

Typ místnosti lze rozlišit podle dvou hledisek. Zaprvé je možné klasifikovat místnosti podle rozložení zdí v jejich vnitřní části. V takovém případě můžeme hovořit o místnostech základních či místnostech s hustým bludištěm. K jejich podrobnému popisu se dostaneme později, například v části 3.2.1.
Druhým hlediskem je typ místnosti ve smyslu jejího vztahu k místnotem ostatním. Místnosti totiž tvoří dohromady větší, tematicky stejnorodé celky – komponenty. Jak tedy vypadá zmíněná tematická stejnorodost? Každá komponenta obsahuje pouze vybrané typy místností. Typ místnosti potom implikuje množinu objektů, které se k této místnosti váží.

1.2.6 Komponenta

Komponenta je část labyrintu, která je tématicky stejnorodá. Do komponenty patří vždy několik místností, které tvoří souvislou oblast.
Komponenty jsou sérií na sebe navazujících místností, které se dále nevětví (z jedné místnosti vede cesta nejvýše do jedné další).
Zvláštní případ je základní komponenta, do které ústí vstup do labyrintu. Tato komponenta má totiž rozvětvenou strukturu. Tuto komponentu někdy nazýváme centrální komponentou.

Jednotlivé komponenty jsou vždy dveřmi spojené se základní komponentou. Dveře mezi první místností z komponenty a styčnou místností základní komponenty jsou vždy zamčené. Klíč k nim je uložen v poslední místnosti jiné komponenty. V základní komponentě je přímo umístěn klíč k prvním zamčeným dveřím, protože jinak by byl labyrint neprůchodný.

Příklad:

Příkladem budiž ukázka z databáze:

* Typ komponenty „Pohřebiště“ definuje jako přípustnou množinu místností „Svatyně“, „Upíří hrobka“ a „Kobka“.
* Pro ilustraci například v místnosti typu „Upíří hrobka“ se vyskytují nestvůry „Upír“ a „Přízrak“.
* V této místnosti se mohou vyskytovat předměty „Dýka nemrtvých“ a „Lektvar krvežíznivé síly“
* Textem, který doprovází celou komponentu je :

„Vstoupili jste na znesvěcenou půdu. Každým krokem vás začíná svazovat tíživý pocit, že vás sleduje několik párů očí. Všude kolem se mísí čpivá vůně staletí starého kadidla s příměsí pachu čerstvě překopané zeminy....“

* Po vstupu do místnosti „Upíří hrobka“ hráči situaci nastíní text:

„Tato místnost se od všech okolních značne liší. Váš pohled uchvátil obrovský katafalk z černého mramoru. Něco vás však nenechává klidným. Masivní žulový příklop hrbky vážící jistě stovky uncí je ledabyle odsunutý na stranu...“

1.2.7 Vstup do labyrintu

Vstup do labyrintu je pole, na kterém dobrodružství začíná. Lze si ho představit jako například schody vedoucí z povrchu do podzemního labyrintu. Tento vstup vede do základní části labyrintu (centrální komponenty). Z této oblasti je na začátku přístupná pouze jedna komponenta bludiště.

1.2.8 Fyzikální (prostorová) definice mapy

Princip nazvaný „zeď-volno“ vystihuje charakteristiku labyrintu. Labyrint je jeskyně a její přirozenou výchozí vlastností je, že se pole nedá vstoupit (není vyhloubené).
Jednotlivá pole mohou být tedy ve smyslu tohoto principu dvojího typu

a) volné místno

Všechna pole, na které může teoreticky hráč vstoupit mají tuto vlastnost.
Teoreticky proto, že vstoupení na pole vyžaduje kromě toho, aby cílové pole mělo vlastnost „volné místo“, aby pohyb začínal na sousedním poli. Takové pole ovšem musí být přístupné ze vstupu do labyrintu. Některá volná pole tedy mohou být nepřístupná.

b) zeď

Pole, která nejsou volná, jsou nepřístupná nezávisle na tom, jaké je jejich okolí. Tyto pole nazveme „zeď“.

Nezbytná omezení:

Některá pole se zvláštním významem musí být typu „volné místo“. Takovými poli jsou vstup do labyrintu nebo obě pole, která spojují dveře. Naopak některá pole musí mít vlastnost „zeď“. Zde jde především o pole na hranici místnosti (aby místnost byla ohraničená) s vyjímkou dveří.

1.3. Hrdina

(vlastnosti)

Akce, které může hrdina provádět, jsme popsali v předchozí části. Zde popíšeme jeho vlastnosti.

1.3.1. Vlastnosti

Vzhledem k akcím, které hrdina v labyrintu může v naší hře provádět jsme zahrnuli jen některé vlastnosti z Dračího doupěte.

Síla

Síla přímo ovlivňuje hrdinův útok a počet životů. Zdokonaluje se postupem na vyšší úrovně.

Obratnost

Obratnost má vliv na obranné číslo. Také se zdokonaluje postupem na vyšší úrovně.

Úroveň

Úroveň přímo odpovídá zkušenostem a schopnostem hrdiny. Tato vlastnost je měřítkem pokroku ve vývoji hrdiny.

Počet tahů na kolo

Počet tahů na kolo odpovídá počtu základních akcí (viz 1.1.3), které může hrdina za jedno kolo vykonat. Po přechodu na další úroveň se zvyšuje.

1.3.2. Statistiky

Počet životů

Maximální počet životů se spočítá podle vlastností a úrovně hrdiny. Aktuální počet životů nemůže překročit maximální množství.

Obranné číslo

Základní obranné číslo je závislé na pouze vlastnostech hrdiny. Základní obranné číslo se spočte jako součet úrovně a obratnosti. Aktuální obranné číslo je zvýšeno použitím ochranné pomůcky. Obrannému číslu je přímo úměrná šance vykrýt nepřátelský útok.

Útočné číslo

Základní útočné číslo je také závislé pouze na vlastnostech hrdiny. Základní útočné číslo se spočte jako součet úrovně a síly. Aktuální útočné číslo je zvýšeno použitím zbraně. Útočnému číslu je přímo úměrná hrdinova šance prorazit obranu nepřítele při útoku.

Počet zkušeností

Vlastnost jemněji popisující zdatnost hrdiny, než úroveň. Zkušenosti se sbírají poražením nepřátel a nasbírání určitého počtu umožní přechod na vyšší úroveň. Zkušenosti se po přechodu na další úroveň nevynulují, ale sbírají se dál.

1.3.3. Přechod na další úroveň

Po dosažení požadovaného počtu zkušeností se hrdinovi zvýší úroveň. To má za následek zlepšení některých vlastností. Konkrétně se mění počet životů, základní obranné a útočné číslo, dále pak síla a obratnost.
(To ovšem neodpovídá pravidlům Dračího doupěte. Některé vlastnosti, které se mění s přechodem na další úroveň jsou obtížně formalizovatelné a proto jsme od jejich implementace upustili.).

1.4. Souboj

(vlastnosti)

Souboj jako postavou prováděná akce zde v podstatě znamená pouze útok. Obrannou fázi nelze provádět bez předchozího útoku. Obrana tedy není akcí, kterou lze vykonat samostatně.

K útoku je potřeba, aby se útočník a obránce nacházeli na sousedních polích (nad sebou či vedle sebe).

1.4.1 Útok a obrana

Jeden útok trvá jeden tah. Po útočné fázi provede obránce obrannou fázi a vypočítá se výsledek souboje.
Výsledkem souboje může být smrt obránce, zranění obránce za nenulový počet životů, či úspěšná obrana a tedy žádná ztráta životů obránce. Útočník se nemůže sám při útoku poranit, jakkoliv byla obrana úspěšná.
Jestliže je protivník zabit, získá hrdina zkušenosti a může si přivlastnit protivníkovy předměty.
V případě zabití hrdiny hra končí prohrou.
Obrana nespotřebovává tahy. Během čekání na tahy ostatních je možné se bránit neomezenému počtu útoků. To je smysluplné, protože útoky soupeřů by v reálném čase trvaly nějakou dobu. Tuto dobu lze využít k obraně. Obránce tedy nemá k dispozici potenciálně nekonečný čas, nebo více času než ostatní, jak by se mohlo zdát.

1.4.2 Výpočet následků souboje

Nejprve se provede výpočet aktuálního útočného čísla a útočnosti útočníka. Sečte se jeho základní útočné číslo a útočné číslo zbraně, kterou používá. Analogicky se postupuje s útočností s tím rozdílem, že základní útočnost je nula.
Poté se vypočte obranné číslo obránce sečtením základního obranného čísla a obranného čísla případně použité ochranné pomůcky.

Dále po vzoru originální hry budeme simulovat hod šestistěnou kostkou (popsáno níže).

K celkovému útočnému a obrannému číslu se přičte hod kostkou (celkově se provedou 2 různé hody) a získáme celkový útok a obranu. Od útoku odečteme obranu a získáme počet životů, za které je obránce zraněn po provedení útoku.
Počet životů obránce je potom snížen o vypočtené zranění. Záporné zranění se ignoruje (dobrá obrana nemůže vyléčit životy).

1.4.3 Hod kostkou

Výsledkem hodu kostkou v Dračím doupěti se počítá tak, že pokud hráč hodí číslo menší, než maximální hodnota na kostce, toto číslo se započítá, jinak se k maximální hodnotě připočte ještě výsledek dalšího hodu (se stejným pravidlem pro další opakování hodu).
To znamená, že výsledek hodu není shora omezený, ovšem jeho limita je samozřejmě nula. Další zajímavou vlastností je, že výsledkem nemohou být násobky šesti.
Počet stěn kostky je shodný s maximální hodnotou na kostce. Ostatní stěny znamenají postupně nižší hodnoty. Každá hodnota se vyskytuje právě jednou.

Kapitola 2

Generátor herního prostředí

V této kapitole se zaměříme na prezentační a uživatelskou vrstvu generátoru. V kostce popíšeme použitou strategii generování a způsob, kterým můžeme sami generovat nový labyrint.

Na základě fungujícího samočinného generátoru bylo vytvořeno uživatelské prostředí k automatické i ruční tvorbě labyrintů. Pro tvorbu nového labyrintu je možné zvolit jeden ze dvou režimů - automatický nebo manuální.

2.1 Parametry

Prvním krokem, který předchází obě možnosti generování, je nastavení parametrů. Tyto parametry jsou, jak pro ruční, tak pro automatický režim, společné.

Rozměry bludiště

Šířka a výška celkové plochy labyrintu. Minimální velikost je 20 pro oba rozměry, maximální pak 100. Tato omezení mají čistě praktický význam - technicky lze generovat jak menší, tak i výrazně větší labyrinty.

Počet větví

Počet oddělených komponent v bludišti, které mají vzniknout. Není ovšem zaručené, že se generátoru podaří tomuto parametru vyhovět. Pro danou velikost labyrintu je možné v něm vytvořit pouze omezené množství místností a tedy i komponent.

Databáze objektů

Databáze objektů je součást poskytnutá spolu s generátorem. Z této databáze se načtou a do labyrintu vloží předměty a nestvůry. Modifikace databáze není v rámci poskytnutého řešení možná. Na vlastní zodpovědnost je ovšem možné databázi ručně editovat.

Úroveň labyrintu

Podle nastavení tohoto parametru generátor vybírá typy předmětů, které mohou být použity v labyrintu. Generátor také upraví charakteristické vlastnosti nestvůr podle tohoto parametru.

Minimální a maximální poměr dělení místností, minimální hrana místnosti

Níže popíšeme, jak je možné vytvořit nové místnosti pomocí dělení již existujících místností. Dělení místností závisí právě na následujících parametrech.

Zahnízdění generátoru

Tento parametr nastaví zahnízdění generátoru, který je použit pro všechna pseudonáhodná rozhodnutí učiněná generátorem.Tento parametr tedy určuje náhodnost generování.
Po nastavení stejných parametrů docílíme díky tomuto parametru vygenerování vždy stejného bludiště. Pro každou kombinaci velikosti, počtu větví, poměru dělení a minimální velikosti hrany místnosti máme k dispozici řádově 32 tisíc možných labyrintů, což je uspokojivý počet.

2.2 Automatický režim

Pokud se rozhodneme generovat labyrint automaticky, postupujeme následovně.
Nejprve načteme databázi. Poté nastavíme parametry generování. Dále stiskneme tlačítko generovat automaticky a práce je hotová. Výsledek si můžeme prohlédnout stiskem tlačítka zobrazit/skrýt mapu. Pro uložení stiskneme tlačítko uložit a po zadání jména cílového souboru je labyrint exportován do formátu XML a generování je úspěšně hotové.

2.3 Manuální režim

Tato možnost je z pohledu grafického uživatelského prostředí mnohem zajímavější. Nabízí totiž možnost generování téměř libovolně přetvářet.
Idea je následující: automatický generátor generuje labyrint ve třech fázích. V mauálním režimu se provede vždy jedna fáze (v jednom případě dokonce polovina) a uživatel poté může sám provádět stejné typy kroků, které byly prováděny generátorem. Tento přístup umožní, až na detaily, vygenerovat labyrint podle vlastní vůle. Zároveň zůstanou zaručeny všechny vlastnosti, které má labyrint mít. Dodržování většiny z nich je kontrolováno generátorem, který nedovolí jejich porušení.
Poté, co již uživatel upravil vše, jak chtěl, může spustit další fázi a dát se do upravování jejích výsledků.
Vzhledem k tomu, že generování jedné fáze záleží na výsledcích fáze předchozí, není možné se k již uzavřeným fázím vracet.

Rozeberme nyní podrobněji, jakou funkcionalitu nabízí grafické rozhraní generátoru v jednotlivých fázích.

2.3.1 Fáze první

Zde je cílem dělit místnosti tak, aby vznikl požadovaný počet větví. Počet větví je předtím nastaven jako parametr.
Uživatel vstoupí do procesu v momentě, kdy algoritmus dokončí generování požadovaného počtu větví, respektive místností pro ně.
Uživatel potom může pokračovat v této činnosti. Může zvolit libovolnou z místností a pokud má dostatečně velké rozměry, může ji podélně či příčně rozdělit. Poměr dělení je možné buď nechat zvolit náhodně, nebo jej lze nastavit přímo. Poměr dělení lze nastavit pro každou místnost zvlášť. Nastavené poměry se automaticky opraví tak, aby odpovídaly parametrům maximální a minimální poměr dělení.

Je také možné přidat místnost náhodně. To znamená nechat generátor náhodně vybrat jednu místnost a tu rozdělit v nějakém poměru a směru. Poměr lze opět nastavit, či nechat vybrat náhodně.
Pro přejití do fáze dvě je potřeba ještě vygenerovat dveře mezi místnostmi. Místnosti, které budou spojeny, určí generátor (není možné je ručně zvolit, protože by mohlo snadno dojít ke ztrátě vlastností, které od labyrintu požadujeme). K tomu opět stačí stisknout tlačítko.

2.3.2 Fáze druhá

V této fázi se vyplňuje obsah místností. Jak lze očekávat, je na výběr místnost základní, či místnost s hustým bludištěm.
Do některých místností bylo bludiště samo předem nagenerováno, jiné zůstaly prázdné. Můžeme zvolit kteroukoliv místnost a toto nastavení změnit. Přechod od bludiště k základní místnosti je triviální. Při opačné změně se samočinně vygeneruje husté bludiště uvnitř zvolené místnosti. To lze znovu a znovu přegenerovat tím, že zvolíme jiné zahnízdění generátoru.
K dispozici je ovšem ještě jemnější úprava - ruční editace polí. K tomuto účelu se otevře nové okno zobrazující stávající situaci v editované místnosti. Nyní můžeme znovu generovat automaticky bludište podle různého nastavení generátoru náhodných čísel nebo místnost jednoduše vyprázdnit. V každé situaci můžeme měnit ručně stav polí uvnitř místnoti z pole typu „zeď“ na pole typu „volné místo“ a zpět.
Protože vyžadujeme průchodnost bludiště mezi všemi dvojcemi dveří v místnosti, provádíme ruční editaci pouze na kopii místnosti. Před konečným aplikováním změn do labyrintu je spuštěno hledání cesty grafu pro všechny dvojice dveří příslušných k právě editované místnosti. Pouze pokud všechna spojení mezi dveřmi existují, povolí generátor změny provést. V opačném případě se může uživatel vrátit k editaci či od všech změn upustit.
Pro větší pohodlí je možné spustit hledání cest i během editace. Každá nalezená cesta se vyznačí přímo ve vizualizaci editované místnosti. Takto lze snadno odhalit, které spojení bylo přerušeno a včas chybu opravit. V rozsáhlejších místnostech není snadné po chvíli editování objevit pole, které brání v průchodnosti ke dveřím.

2.3.3 Fáze třetí

V této závěrečné fázi vkládáme do labyrintu předměty a nestvůry. Kromě toho se v této fázi řeší otázka klíčů a zamykání dveří.
Tato fáze jako jediná umožňuje přetvořit skutečně vše, co bylo předgenerováno.
Před prvním zásahem užívatele rozmístí generátor klíče, zamkne dveře, umístí předměty a nestvůry.
Zamknutí dveří a rozmístění klíčů k nim je na začátku smysluplné a správné vzhledem k vytyčené taktice. Protože ovšem lze vytvořit mnoho jiných smysluplných kombinací zamčených podsoučástí grafu, byla nechána uživateli plná kontrola nad zamykáním a odemykáním všech dveří a umístěním klíčů. Je ovšem na uživateli, zda výchozí labyrint bude průchodný do plné hloubky, či ne. Pro zamčení vybraných dveří je nezbytné je zvolit. Každé dveře jsou zobrazeny jako dvě pole, která jsou dveřmi oddělena. Po zvolení dveří lze v případě odemčených dveří zvolit „zamknout“ a poté kliknutím do bludiště umístit klíč na platnou pozici. Pokus umístit klíč do zdi se ignoruje. V případě zamčených dveří se zvýrazní klíč, který k nim náleží. Takové dveře lze naopak odemknout, po čemž příslušný klíč je odebrán z labyrintu.

Po kliknutí na klíč se naopak zvýrazní příslušné dveře. Po odebrání klíče se dveře odemknou.
Předměty a nestvůry se editují jednotným stylem. Uživatel vybere místnost, kterou chce upravovat a zvolí tlačítko editovat. Zobrazí se podobné okno, jako v případě editace bludiště. Tentokrát se ovšem mohou přidávat či odebírat na jednotlivých polích předměty respektive nestvůry. Zde je ovšem dbáno na dodržení tematické homogenity, takže přidat lze jen objekty, které se mohou nacházet v dané komponentě. Množina objektů, které mohou být přítomny v příslušné komponentě je určena databází.
V tomto okně je také možné měnit textovou legendu jednotlivých komponent.

Generovat znovu

Generování třetí fáze lze provést znovu v plném rozsahu po stisku tlačítka Generuj znovu.

2.4 Uložení labyrintu

Ukládání labyrintu je společné oběma metodám generování.
V případě automatického generování je možné labyrint uložit ihned po stiknutí tlačítka ke generování. V případě manuálního generování se uložení labyrintu zpřístupní teprve po provedení poslední fáze generování.
Po stisku tlačítka Uložit bludiště a zvolení cílového souboru se provede export do XML formátu.

Oddíl druhý

Formální pohled, algoritmy

V kapitolách tohoto oddílu se budeme zabývat popisem modelu světa popsaného v prvním oddíle. Budeme hovořit o reprezentaci jednotlivých objektů a modelování vztahů mezi nimi. Tím vymezíme způsob, kterým se dá trvale uložit herní rámec.
Nastíníme zde vlastnosti, které budeme požadovat od smysluplného labyrintu, integritní omezení či vlastnosti, které činí labyrint zajímavějším.
Dále v této části přiblížíme jiný pohled na labyrint a jeho části. K popisu některých kýžených vlastností použijeme nástroje z teorie grafů.
Poté, co dokončíme sumarizování všech požadavků se pustíme do popisu problematiky generování herního světa.
Rozložíme problém na tři základní oblasti a podáme jejich podrobný popis. Uvedeme některé algoritmy včetně slovního popisu i zápisu v pseudokódu.
Popíšeme zde možnosti parametrizování procesu generování, roli databáze.
Cílem této kapitoly je podat ucelený náhled na problematiku parametrizovaného generování herního rámce v abstraktní rovině bez technického popisu implementačních detailů. K tomu přistoupíme až v posledním oddíle.

Kapitola 3

Zavedení termínů a popis modelu

3.1. Zavedení termínů a uvedení do souvislostí

Doposud jsme představili implementovanou verzi hry z pohledu uživatele - z vnějšku. V této kapitole se zaměříme na formálnější popis některých již představených vlastností.
Popíšeme zde vnitřní reprezentaci celého herního prostředí. To nám přiblíží, jak je po technické stránce řešena implementace hry. Hlavním cílem formalizace je ovšem připravit prostředky pro následující kapitoly, ve kterých budeme rozebírat požadované vlastnosti labyrintu a generátor, který bude tyto požadavky naplňovat.

3.1.1 Mapa a její součásti

Jednotlivé konstrukty budeme popisovat shora dolů, to znamená komplexní objekty nejprve a jejich dílčí části potom.

Labyrint

Labyrint je základní jednotka, o jejíž sestavení se snažíme. Obsahuje definice typů objektů a seznam jejich instancí (předměty, nestvůry).
Dále se zde nachází struktura držící informace o mapě - seznam místností a jejich struktury, dveří spojujících jednotlivé místnosti.

Mapa, rastrová reprezentace

Pokud používáme termín mapa, znamená to rastrovou reprezentaci labyrintu ve smyslu principu „zeď - volný prostor“ (viz kapitola 2).
Rastrovou reprezentací máme na mysli dvourozměrné pole obsahující informaci vztahující se k poli labyrintu na odpovídajících souřadnicích (zde informace značí vlastnosti pole zeď/volno). V podstatě jde o jednostranně zaměřený pohled na labyrint.
Rastrová reprezentace bude znamenat v celé práci totéž.
Mapa nebere ohled na objekty nacházející se v labyrintu.

Místnost

Mapa je tvořena místnostmi. Místnost je obdélníková oblast mapy. Místnosti jsou vzájemně disjunktní. Mapa je beze zbytku rozdělitelná na místnosti (každé pole leží v nějaké místnosti).

Pole

Místnost je tvořena poli - nejmenšími „adresovatelnými“ jednotkami mapy. Pole je konečná jemnost granularity. Proto je mapa diskrétním objektem.

3.1.2 Topologické vlastnosti labyrintu (mapy)

Mapa je tvořena místnostmi, které jsou navzájem spojené dveřmi.

Uvažme graf G=(V,E), kde V, množina vrcholů, odpovídá množině místností mapy M. Množina hran E obsahuje hranu mezi místnostmi m1 a m2 právě tehdy, když jsou tyto dvě místnosti spojeny dveřmi.
Takový graf nazveme grafem mapy M.

Všechny labyrinty, které byly vytvořeny generátorem mapy popsaným v další kapitole vykazují následující vlastnosti:

Vlastnosti cílového labyrintu ve smyslu grafu G

Graf G nebude obsahovat kružnici. Tato vlastnost je nezbytná k naplnění požadavku, aby bylo nutné „projít celý labyrint“ k dosažení cíle. Nyní je totiž náš graf stromem. Toho s výhodou využijeme k tvorbě oddělených komponent.

Graf sestává ze dvou typů komponent. Tyto komponenty odpovídají komponentám 2.2.7.

Větev

První z nich je takzvaná větev.
Větev je podgraf grafu G. Množinu jeho vrcholů tvoří vrcholy grafu G tranzitivně spojených s právě jedním listem (v grafu G). Stupeň vrcholu všech vrcholů v grafu G je nejvýše dvě. Množinu hran tvoří všechny hrany z G, jejichž oba přilehlé vrcholy patří do větve.

Centrální komponenta

Centrální komponenta je zbytek stromu po ořezání větví.
Vrcholy spojené hranou s větvemi mají nutně stupeň alespoň 3 (jinak by byly pokračováním větve). Ve skutečnosti mají stupeň právě tři - vlastnost plynoucí ze způsobu tvorby místností.

Vstup do labyrintu, cíl, finální větev

Vstup do labytintu je pole, na kterém se na začátku hry objeví hrdina. Je vždy v nějaké místnosti, která se nachází v centrální komponentě grafu labyrintu.
Cíl je pole, kde bude později umístěna hlavní nestvůra. Je umístěno v listu. Větev příslušná k tomuto listu se nazývá finální větev.

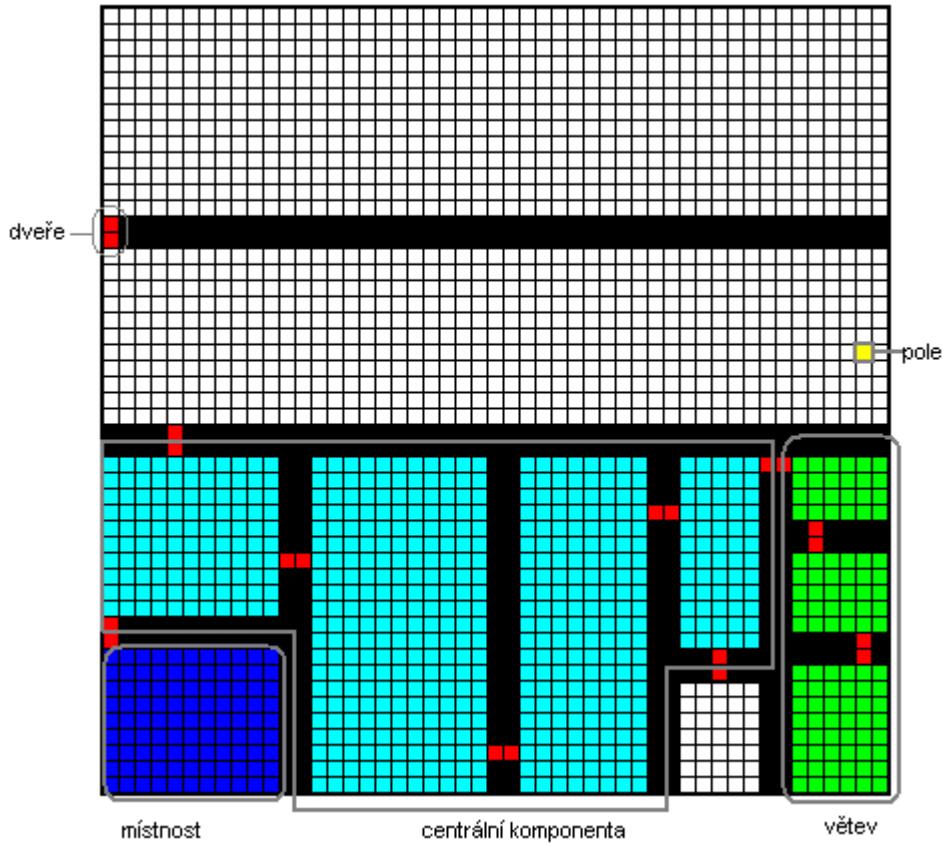
Dveře

Na dveře můžeme nahlížet jako na relaci mezi dvěma poli. Tato pole musí ležet vedle sebe, nebo nad sebou. Dále ještě každé z těchto polí leží v jiné místnosti.

Princip dveří

Dveře jsou překážka mezi dvěma sousedními poli. To znamená, že se nenachází na pozici pole a mají jen jeden rozměr (toto je pouze neformální pozorování). Dřívější

implementace modelovala dveře jako pole, na které jde vstoupit pouze s daným klíčem v inventáři. To ovšem plně neodpovídá podstatě dveří (takové dveře brání i jejich minutí ve směru roviny dveří v těsné blízkosti)



OBRÁZEK 1: Ilustrace termínů části 3.1: místnost, dveře, centrální komponenta, větev a pole.

3.2 Popis statického modelu

3.2.1 Vrstevnatá reprezentace

Po uvážení souvislostí jednotlivých částí herního světa jsme se rozhodli zavést třívrstevnou reprezentaci herního prostředí.
Každá vrstva reprezentace přesně odpovídá fázi generování.
Rozvržení do vrstev je provedeno tak, že žádná vrstva nepotřebuje informace z vyšší vrstvy. Jinými slovy, entity a jejich vztahy modelované danou vrstvou nejsou závislé na žádných informacích z vyšších vrstev.

Vrstva 1: Místnosti

Vrstva místností je základní vrstvou. Jejím úkolem je držet informace o rozdělení plochy labyrintu na jednotlivé místnosti. Stará se o to, aby místnosti byly disjunktní a byly správně propojené. V neposlední řadě se zde nacházejí informace o dveřích.
Tuto třídu lze také dotazovat na rastrovanou podobu mapy se zobrazovanými referencemi do tabulky místností pro jednotlivá pole.

Vrstva 2: Obsah místností

Druhá vrstva uchovává data týkající se vnitřního uspořádání jednotlivých místností. Teprve zde se do hry dostává obsah jednotlivých polí, tj. vlastnost „zeď-volno“.
V tomto duchu jsou rozlišovány dva typy místností - místnosti základní a místnosti s bludištěm.

Základní místnost

Všechna pole jsou prázdná (vlastnost „volno“) s vyjímkou polí po obvodu. Pole patřící do relace dveře jsou také prázdná.

Místnost s bludištěm

Místnost s bludištěm má stejně vyplněná pole na obvodu jako základní místnost. Rozdíl je v tom, že některá vnitřní pole jsou vyplněna. Pro každou dvojici dveří ovšem vždy existuje cesta, která ctí pravoúhlé souřadnice a spojuje tyto dveře.

Tato vrstva fakticky drží informace o klíčích a zamčenosti jednotlivých dveří. V oddíle jedna byla problematika klíčů popsána v rámci třetí fáze.
Z této třídy je možné získat bitmapový raster obsahu „zeď-volno“ pro každou místnost v labyrintu.
Na této úrovni se také rozhoduje o přiřazení jednotlivých místností do komponent.

Vrstva 3: Vrstva objektů

V nejvyšší vrstvě jsou umístěny informace o jednotlivých objektech (předmětech a nestvůrách). K tomu je použit model „třída-odkaz“ popsaný později.
Je zde také rozšířen kontext místnosti a komponenty, ke kterým byly přidány další informace.
Dále se zde rozliší finální a centální komponenta od ostatních a je zde vyznačeno pole „vstup do labyrintu“.

3.2.2 Princip třída-odkaz

Pro reprezentaci předmětů je zavedena jakási paralela katalog - výrobek. Je naší snahou umožnit vznik v principu neomezeně velkých labyrintů, ve kterých se ovšem vyskytuje poměrně malá množina typů předmětů či nestvůr. Pro efektivní reprezentaci (hlavně na disku) se snažíme vyhnout zbytečnému opakování informací.
Zavedeme tedy tabulku typů objektů, ve které jsou zachyceny společné vlastnosti pro všechny instance objektů této třídy. Potom při vlastním výskytu objektu, ku příkladu instance předmětu v inventáři či nestvůry v labyrintu, uložíme do struktury, ke které se objekt váže, pouze odkaz a specifické vlastnosti (jako polohu).

Místnosti a komponenty

Princip „třída-odkaz“ je také použit u odkazů na příslušnost k místnosti a komponentě například u nestvůr či u při ukládání rasterizované podoby polí. Každé pole má referenci do místnosti, ve které se nachází.

3.2.3 Hierarchie předmětů, doména identifikátorů

Předměty se dají rozdělit na tři kategorie. Čtvrtou by tvořily klíče, ale ty stojí, jak bylo zmíněno, stranou.
Kategorie předmětů jsou zbraně, ochranné pomůcky a lektvary. Protože nakládáme s těmito předměty jednotným způsobem, sdílí spolu i doménu identifikátorů použitou v systému „třída-odkaz“.

3.2.4 Statická část reprezentace hrdiny a nestvůr

Abychom dosáhli transparentního návrhu ovládání akcí, které hrdina či nestvůry provádějí, oddělili jsme vlastnosti těchto objektů od jejich činností. To v praxi znamená, že manipulace s hrdinou či nestvůrou probíhá mimo tyto třídy.
Tím dosáhneme oddělení modelu prostředí od modelu postav. Postavy nemusí „rozumět“ tomu, proč není možné udělat krok z pole A na pole B. Tuto funkcionalitu zaručí „run-time model“ rozebraný v následující části.

Hrdina a jeho vlastnosti

Kromě RPG vlatností z části 1.3 je potřeba držet informace o aktuální pozici, o zbývajícím počtu tahů a o aktuálním stavu iniciativy (tj. zda je hrdina je na tahu nebo čeká na tahy ostatních).
U nestvůr je situace podobná.

3.3 Popis dynamického modelu

Dynamická část modelu hry má na starosti akce a reakce prostředí.

Akce

Akcí je rekogniskace stavu prostředí a vyhodnocení přípustných činností a jejich následné provádění.

Reakce

Reakce simuluje odezvu prostředí a také fakticky provádí veškerou manipulaci s prostředím.

Na začátku každého tahu kterékoliv postavy se vyhodnotí konfigurace prostředí a určí, které akce jsou přípustné. Poté model nabídne seznam možných akcí postavě a podle její volby model tuto akci simuluje.
V případě tahu hrdiny vybere hráč, kterou akci chce provést, v případě nestvůry se zavolá její metoda simulující inteligenci a ta akci vybere.

Implementovaný dynamický model je co možná nejjednodušší. Implementace složitého modelu by mohla být námětem na samostatnou a velmi rosáhlou práci.

Struktura projektu ovšem nevylučuje jeho následnou reimplementaci.

Pravidla

V této vrstvě návrhu v podstatě je zaručeno dodržování všech pravidel hry. Je totiž nemožné, aby objekty sami o sobě prováděly neplatné akce - objekty nemají žádnou možnost přímo měnit nastavení prostředí.

Protože ovšem tato část prostředí je jednou pro vždy zvolená a tedy neměnná, není její studium předmětem této práce. Není zde totiž prostor pro jakoukoliv smysluplnou činnost generátoru.

Kapitola 4

Smysluplný labyrint

(praktická a nutná omezení)

V této kapitole rozebereme žádoucí vlastnosti labyrintu ať již z pohledu hratelnosti, tak i z pohledu praktičnosti. Dále se zaměříme na omezení kladená na labyrint, jejichž dodržení je nezbytně nutné.

4.1 Graf nesmí obsahovat kružnici

Jednoduchá možnost jak mít kontrolu nad místy, kudy hrdina musí putovat. Jednoznačnost volby cesty platí jen na úrovni místností - uvnitř místností není a priori zaručeno nic.
Tato vlastnost je poměrně nešťastná - její ověření je výpočetně velmi složité. Musíme tedy dopředu zaručit, že postup, kterým se labyrint tvoří, tuto vlastnost zaručí.

4.2 Minimální počet větví

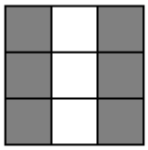
Prvním omezením je minimální počet větví rovný třem. Tak je zaručena existence centrální komponenty.
Komponenta je graf, který je cestou začínající v listu grafu G. Graf G je grafem labyrintu, jak byl popsán v části 3.1.2. Tato cesta je maximální délky, to znamená, že buď zahrne celý graf, nebo skončí nepřidáním vrcholu stupně vyššího než tři.
Protože graf G neobsahuje kružnici, graf, v němž lze najít právě jednu větev, by byl cesta. Cesta má ovšem dva vrcholy stupně jedna a tedy by byly větve dvě, které se ovšem zcela překrývají.
Ze stejného důvodu není možné smysluplně požadovat dvě větve.

4.3 Umístění dveří

Žádné pole v relaci kterýchkoliv dveří se nesmí nacházet v rohu místnosti, protože by nebylo možné na toto vstoupit a tedy průchodnost labyrintu by byla ztracena. Vstoupit na toto pole by nebylo možné proto, že na místech, ze kterých se na něj dá vstoupit je pole s vlastností zeď. Není tedy odkud vstoupit.

4.4 Minimální délka hrany místnosti

Minimální délka hrany místnosti je stanovena na pět polí. Opravdu minimální délka hrany by byla tři, protože místnost musí mít dveře, které nejsou v rohu. Taková místnost by ovšem měla vnitřní prostor pouze jedno pole a vypadala by jako chodba. (Vzpomeňme, že okrajová vrstva zdí je součástí místnosti.)
Minimální velikost 5 dávala nejlepší výsledky (empiricky). Toto omezení je tedy praktické.



Místnost velikosti tři vypadá jako chodba

OBRÁZEK 2: Místnost s délkou hrany 3.

4.5 Přítomnost a poloha klíče

Je zřejmé, že ke každým zamčeným dveřím by se měl v labyrintu nacházet klíč. Strategie rozmístění klíčů uvnitř zamčených komponent to dokonce nutně vyžaduje.
Zajímavější je, jak klíče distribuovat, aby mapa byla průchozí. Zamčené jsou vždy dveře na rozhraní centrální komponenty a větví.
Podmínka první - pokud nejsou všechny dveře v mapě odemčené, musí být klíč k některé větvi v centrální komponentě. Nejpozději po použití tohoto klíče musí být v dosahu hrdiny další klíč, nebo musí být všechny dveře otevřeny.
Podmínka druhá - dveře k finální komponentě musí být odemknutelné až po odemčení ostatních dveří.

4.6 Integrita struktury místnosti

Uvnitř jednotlivých místnotí nesmí být struktura „zeď-volno“ zcela libovolná.
Graf našeho labyrintu je spojitý a neobsahuje kružnici. Z toho plyne, že mezi kterýmikoliv dvěma místnostmi vede právě jedna cesta (na úrovni mísntostí).
Abychom zaručili průchodnost na úrovni polí, tedy reálnou průchodnost z pohledu hrdiny, musí být všechny místnosti, které mají v grafu stupeň vyšší než dvě, průchodné pro postavy od dveří ke dveřím.

4.7 Poloha předmětů

Pro úplnost ještě zopakujeme podmínku polohy objektů v labyrintu - pole, na které poloha odkazuje, musí mít vlastnost „volno“, aby s daným objektem byla možná nějaká interakce a také proto, že to odpovídá realitě.

Kapitola 5

Generátor

5.1 Motivace

Zvolená reprezentace herního prostředí je poměrně komplexní. Její podoba ve formátu XML je navíc velice obsáhlá. To platí zvláště v případě zachycení vrstvy 'zeď - volné místo'. V případě rozsáhlých (rozsáhlejších) labyrintů navíc není možné si udržet představu o celkové podobě labyrintu bez jeho vizualizace.

Automatický generátor

Kromě výše zmíněných veskrze praktických a zřejmě do jisté míry překonatelných obtíží, musíme přihlédnout k tomu, že v případě hry je více než žádoucí přítomnost prvku náhody. Toho ovšem lze dosáhnouti pouze automatickým nebo alespoň poloautomatickým režimem generování herního prostředí.

K tomuto účelu byl navržen 'engine', který samočinně vytvoří plnohodnotné herní prostředí jen ze zadaných parametrů na vstupu.
Tento generátor je deterministický v následujícím smyslu: Zadání stejných parametrů vede vždy ke tvorbě identického labyrintu.

5.2 Automatický generátor - cíle

5.2.1 Naivní generování

První otázkou je: „Proč negenerujeme celý labyrint naivně?“

Co by mohlo znamenat, generovat „naivně“.

* Labyrint náhodných polí s jedinou podmínkou - průchodností od startovního pole k cílovému.
* Umístění náhodně zvolených předmětů a nestvůr na náhodné pozice.
* Zamknutí dveří a umístění klíčů, které je odmykají stejným způsobem.

Je neoddiskutovatelné, že takový přístup by byl velice snadno implementovatelný. Jeho přímočarost by byla pravděpodobně eliminovala velké množství chyb, které by hrozily při aplikaci složitějšího scénáře.

5.2.2 Požadavky na generátor

Jakýkoliv naivní přístup založený pouze na náhodném umisťování objektů ovšem nezaručuje žádné další kvality. Co se týká vlastností labyrintu, nezaručuje v podstatě kvality žádné. Které vlastnosti tedy nemůžeme postrádat?

Nutnost projít celým labyrintem

Pokud bychom generovali labyrint zcela náhodně, může dojít k situaci, že cesta ze startu do cíle bude extrémně krátká a hrdina bude muset navštívit jen zlomek labyrintu. Dokonce ani umístěním vstupu do labyrintu (startu) a cíle co možná nejdále od sebe nezaručíme v principu o mnoho lepší výsledky. Velká část labyrintu zůstane většinou stejně opomenuta.

Škálovatelnost obtížnosti jednotlivých fází hry

Hra by měla obsahovat nějaký dynamický vývoj. K tomu je potřeba postupně zvyšovat náročnost úkolů nebo kvality protivníků. Aby se ovšem jednalo o postupné zvyšování, je nutné zaručit dopředu pořadí dosažení jednotlivých úseků labyrintu. To samozřejmě náhodně průchodný labyrint neumožňuje.

Tématicky sourodé, souvislé a navzájem oddělené komponenty

Základem Dračího doupěte jsou komponenty, které jsou tematicky jednolité. Takové komponenty sdílí textové legendy a obsahují stejné typy nestvůr a předmětů. Důležité je, že jsou ohraničené, co do dostupnosti a přístupnosti. Jinými slovy se do nich dá vstoupit, či z nich vystoupit, jen na určitých místech.
Ze stejných důvodů jako není možné škálovat obtížnost, není ani možné vytvořit takovéto sublabyrinty náhodným generováním.

5.2.3 Generování labyrintu

V minulých kapitolách jsme již představili podstatu hry, způsob, kterým byla namodelována pro implementaci, a konečně i reprezentaci tohoto modelu.
Dále jsme se také pokusili přiblížit, jaké vlastnosti musí nezbytně splňovat rozumný labyrint.
Tyto informace jsou tedy specifikací pro automatický generátor, který máme za cíl vytvořit.

Vše potřebné již bylo vyřčeno, můžeme se tedy pustit do popisu vnitřní logiky našeho řešení, našeho generátoru. Jeho popis je rozdělen na tři logicky soudržné celky.

Poznámka

Dále v této kapitole považujeme všechny již popsané vlastnosti labyrintu z kapitoly 3 a 4 za platné. Pokud například zmíníme místnost, máme na mysli místnost se všemi vlastnostmi, které jsme již popsali.

5.3 Fáze první - místnosti

První fáze začíná takříkajíc z ničeho. Jedinými vstupními daty jsou parametry labyrintu, který má být vytvořen.
Na konci první fáze chceme mít rozdělenou plochu labyrintu na místnosti a připravené všechny možné pozice, na které je možné umístit dveře. Dveře se umístí pouze mezi spojené místnosti.

5.3.1 Parametry

Rozměry bludiště

Celková velikost plochy pokrytá labyrintem. Minimální velikost je stanovena 20 x 20. Menší labyrinty nemají přílišný smysl. Technicky lze generovat labyrint o minimální velikosti jedné místnosti. Maximální velikost bludiště není v tomto momentě omezena. Omezení je nezbytné pouze z prezentačních důvodů jako je maximální rozlišení monitoru.

Počet větví (resp. počet místností)

Zásadní parametr ovlivňující počet iterací v této fázi generování bludiště. Algoritmus může dostat za parametr jak cílový počet větví, které se mají v labyrintu nacházet, tak počet místností, které mají být vytvořeny. Pokud daná konfigurace (např. rozměry bludiště) nedovolí plné uspokojení tohoto parametru, skončí generování v momentě, kdy není možné další místnost resp. větev do labyrintu přidat.

Úroveň labyrintu, databáze objektů

Tyto parametry jsou sice na začátku zadány, v této fázi ovšem nemají žádný význam.

Minimální a maximální poměr dělení místností, minimální hrana místnosti

Minimální a maximální poměr dělení místností jsou parametry přímo použité v algoritmu popsaném níže.

Zahnízdění generátoru

Seed integer pro generátor náhodných čísel. Díky tomu, že je možné nastavit generátor ručně, je možné dosáhnout determinismu v generování. O determinismu zde uvažujeme tak, že ze stejných zadaných parametrů vznikne vždy stejný labyrint.

5.3.2 Algoritmus

K popisu algoritmu použijeme pseudokód. Pseudokód je velice volnou paralelou jazyka C. Pomocné rutiny vrací návratovou hodnotu a přijímají parametry. Řídící cyklus while pracuje také jako v C. Proměnné jsou netypované, respektive typ je zřejmý z kontextu.

```
Generuj_místnosti(parametry)
Zbyva := počet zamýšlených místností
M - množina místností
H - množina hran mezi místnostmi
While (je_místnost_k_dělení(M) && Zbyva > 0)
{       m = Vyber_místnost();
        rovina_dělení = vyber_rovinu_dělení();
        pomer_deleni = zvol_pomer_deleni();
        {mistnost1, mistnost2} = rozděl_místnost(m, rovina_dělení,
pomer_deleni);
        M = M U {mistnost1, mistnost2} / m;
        H = přepoj_hrany(mistnost1, mistnost2, H);
}
```

5.3.3 Pomocné rutiny

Pro popis pomocných rutin používáme také pseudokód.

bool je_místnost_k_dělení(Množina místností)

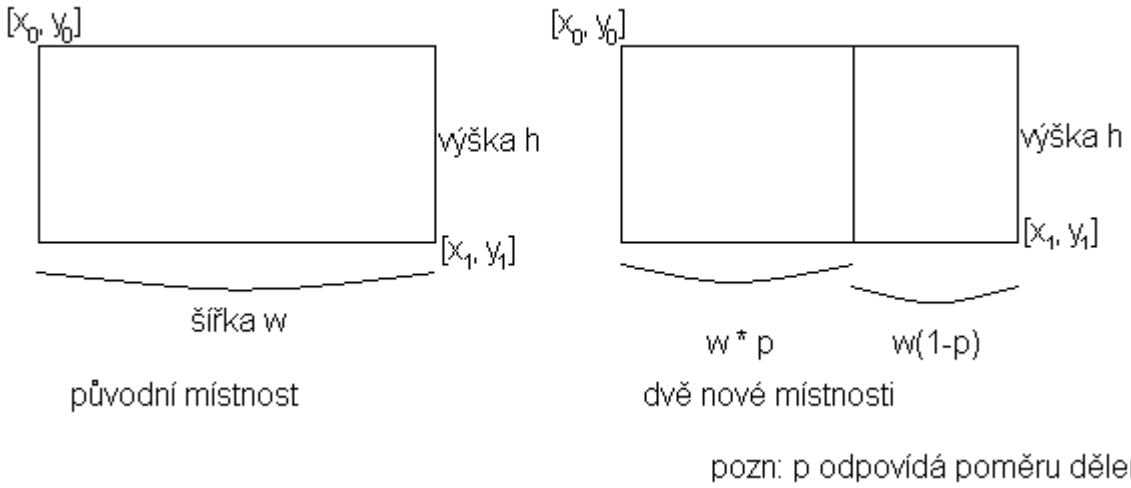
Zjistí, zda se v množině nachází dostatečně velká mísnost, která by se dala rozdělit. Testovaná podmínka je délka alespoň jedné hrany rovná dvojnásobku minimální délky hrany a druhá alespoň rovná minimální délce.

mistnost Vyber_místnost(Množina místností)

Náhodně vybere místnost z množiny místností a otestuje, zda ji lze dělit.

{místnost1,místnost2} Rozděl_místnost_vertikálně(místnost, poměr_dělení)

Pokud místnost lze vertikálně rozdělit, vytvoří se dvě nové místnosti. Jejich výška je stejná jako výška původní místnosti a jejich šířky jsou rovny původní šířce rozdělené v daném poměru. Místnosti jsou umístěny tak, aby se nepřekrývali a dohromady zaplnily plochu, na které se rozkládala původní místnost.
Vrací tyto dvě nové místnosti.



OBRÁZEK 3: Průběh (vertikálního) dělení místnosti.

{místnost1,místnost2} Rozděl_místnost_horizontálně(místnost, poměr_dělení)

Horizontální varianta procedury Rozděl_místnost_vertikálně.

{mistnost1, mistnost2} Rozděl_místnost(místnost, rovina_dělení, poměr_dělení)

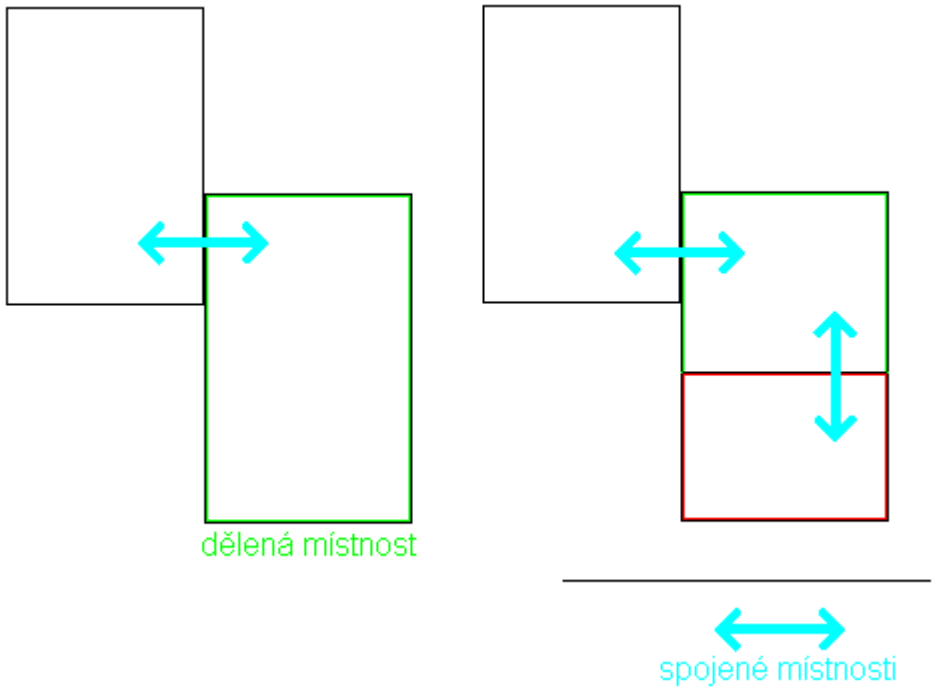
Metoda obalující dvě předchozí metody. Po zavolání vhodné z nich se do seznamu hran mezi místnotmi přidá hrana mezi místností1 a místností2.
Tím je zaručena souvislost labyrintu, který konstruujeme.

int Zvol_poměr_dělení()

Tato rutina vrátí náhodně vybrané číslo mezi parametry určeným minimálním a maximálním poměrem dělení.

void Přepoj hrany(mistnist1, mistnist2, seznam_hran)

Tato metoda se stará o integritu labyrintu. Místnost, kterou jsme dělili (označena jako m) byla spojena s některými dalšími místnostmi z množiny M. Bohužel místnost má jisté topologické vlastnosti a dvě místnosti mohou být spolu spojeny pouze v případě, že se jejich hrany dotýkají a tento dotyk má délku alespoň tři. Tři proto, aby dveře neležely v rohu jedné z místností.
S podmínkou minimální velikosti místnosti 5 je vždy zaručeno, že pokud byla původní místnost spojena s jinou, bude možné toto spojení udržet alespoň s jedním z produktů dělení.
Vlastní přepojení hran probíhá tak, že se každá hrana testuje na to, která z nových místností sousedí s cílovou místností hrany - pomocí metody Je_sousedni_mistnost - a potom se přepojí k té nové místnosti, se kterou fyzicky sousedí. Pokud sousedí s oběma, rozhodne se náhodně.



OBRÁZEK 4: Připojitelná a nepřipojitelná místnost vzniklá dělením původní místnosti.

bool Je_sousední_místnost(místnost1, místnost2)

Celkově jsou čtyři situace, jak spolu mohou dvě místnosti sousedit a každá má dva typy překryvu. Extensivně se otestují všechny kombinace a pokud některé z variant sousedství tyto dvě místnosti vyhovují, metoda vrátí true, jinak false.
Vyber rovinu dělení(místnost) -> {horizontálně, vertikálně }
Při výběru zda bude místnost rozdělena horizontálně nebo vertikálně bude rozhoduto náhodně. Pokud je možná jen jedna varianta, vybere se ta.
Před voláním této funkce se ověří, že je možné dělení provést a je zvolena místnost, na které je dělení možné provést. Proto je pro tuto funkci při takovémto volání dobře definován výsledek.

5.3.4 Některé aspekty algoritmu z pohledu teorie grafů

Dělení místností

Dělení místností je vlastně jakousi formou podrozdělování vrcholů. Při dělení místnosti m, které v grafu místností G odpovídá vrchol v. Vrchol v je nahrazen dvěma novými vrcholy a místo všech hran, které vedly do vrcholu v, jsou po rozdělení v grafu hrany vedoucí z původních vrcholů do jednoho z nástupců vrcholu v. Rozhodnutí, do kterého ze dvou nových vrcholů hrana povede, je provedeno na základě topologických vlastností grafu tak, aby graf zůstal stále rovinný.

Labyrint si je sice možné představit jako souvislý rovinný graf. Pokud ovšem podrozdělujeme nějaký vrchol, není možné pouze spojit oba nové vrcholy se všemi, se kterými byl spojen původní vrchol, protože by byla porušena rovinnost.

5.3.5 Složitost

Složitost tohoto algoritmu je lineární, co do počtu místností. Implementace algoritmu umožňuje bez problémů generovat grafy o rozměrech několika tisíc polí a stovek místností - tedy větší, než je myslitelně použitelné v reálné hře. Pro praktické použití ve grafické formě generátoru jsme omezili generování na 100x100 polí.

5.4 Fáze druhá - obsah místností

Na začátku druhé fáze máme rozdělenou plochu labyrintu na místnosti a připravené všechny možné pozice, kam mezi spojenými místnostmi umístit dveře.
Na konci druhé fáze chceme mít místnosti ohraničené a případně vyplněné zdmi. Dále chceme rozdělit místnosti do komponent, zamknout dveře na rozhraní centrální komponenty a jednotlivých větví.
Po celou dobu musí navíc labyrint zůstat průchodný z místnosti do místnosti, pokud jsou tyto propojeny dveřmi - u této vlastnosti vyžadujeme tranzitivitu.

5.4.1 Dva typy místností

Pro potřeby této fáze zavedeme kategorizaci místností podle obsahu.

Základní místnost

Základní místnost je místnost, jejíž všechna pole, vyjma některých hraničních, mají vlastnost „volný prostor“. Tato místnost je pouze obroubena na svrchní vrstvě rámcem „zdí“ o šířce jedna. V této vrstvě mají pole, která patří do relace dveří, také vlastnost „volný prostor“.

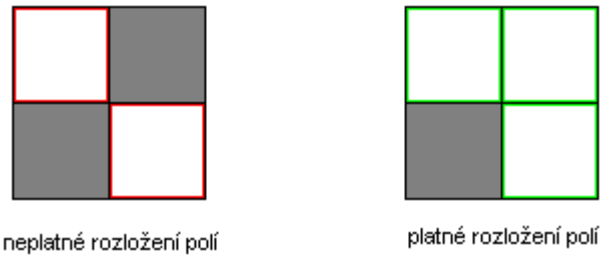
Místnost s bludištěm

Místnost s bludištěm je místnost, která je vyplněna hustým labyrintem. Tento labyrint je souvislý a průchodný z kteréhokoliv pole náležícího ke dveřím na kterékoliv jiné pole náležící jiným dveřím v jedné místnosti (viz 4.6). Tento labyrint je vytvořen náhodně na základě níže popsaného algoritmu.

5.4.2 Algoritmus pro tvorbu hustého bludiště

Vstup: zahnízdění generátoru

Všechna pole jsou na začátku označena za nenavštívená.
Princip algoritmu spočívá v tom, že se do hloubky prochází oblast m x n. V průběhu procházení algoritmus označuje pole, na která vstoupil, jako volná. Na pole, na která nelze vstoupit, se postaví zeď. Na pole nelze vstoupit, pokud by se tímto krokem označilo nově jako volné takové pole, které má ve své bezprostřední blízkosti jiné volné pole. To odpovídá charakteristické vlastnosti labyrintu, že cesta je vždy široká jedno pole a nesousedí spolu volná pole přes roh bez toho, aby mezi nimi vedla ještě i pravoúhlá cesta.



OBRÁZEK 5: Platné a neplatné rozložení polí z pohledu algoritmu pro tvorbu hustého bludiště.

Algoritmus pokračuje rekurzivně do hloubky, dokud zamýšlené kroky vedou na platné pozice (tj. na pole s výše popsanými vlastnostmi ležící v platném rozsahu). Pokud z aktuálně zpracovávaného pole lze vstoupit na další pole, kromě toho, které algoritmus náhodně vybral pro rekurzi, jsou tato pole zařazena do fronty. Poté, co není možné udělat žádný krok, se vyjme další pole z fronty a pokračuje se z něj.
Algoritmus dokončí celou rekurzi (vyprázdní frontu). Poté se nahradí všechna nenavštívená pole polem s vlastností zeď. Nenavštívená mohou zůstat pole obklopená ze všech stran vlastností zeď. Jde vždy o oblast velikosti jednoho pole, takže její přímé označení za zeď nezmění průchodnost bludiště.
Konečným produktem algoritmu je pole m x n obsahující booleanovské hodnoty. True na pozici x,y znamená, že pole místnosti na pozici x,y má vlastnost „volno“, pro false totéž, ovšem s vlastností „zeď“.

5.4.3 Pomocné rutiny

Prozkoumej

Prozkoumá všechna čtyři sousední pole a vrátí případné směry, kam je možné vkročit. Pokud krok v některém směru není možný, postaví tam zeď (označí pole číslem '2').
Krok konkrétním směrem není možné učinit pokud:
a) pole v daném směru je volné
b) následující pole v daném směru je volné
c) pole nad či pod polem v daném směru je volné
d) pole v daném směru není dosud nenavštívené (v podstatě je zeď)
Jinak je krok možný.

Proveď_krok(k)

Změň proměnnou aktuální_pole podle směru kroku k. Vyplň raster na pozici aktuální_pole číslem '1'.

5.4.4 Schématický zápis algoritmu v pseudokódu

Raster je pole čísel z množiny {0, 1, 2} velikosti n x m. '0' znamená dosud nezpracováno, '1' volné pole, '2' zeď.
Místnost je pole booleanských hodnot velikosti m x n. Po ukončení běhu algoritmu je to jeho výsledek.

```
Vyplň_bludiště(zahnízdění_generátoru){
        inicializuj generátor nahodných čísel podle parametru;
        vytvoř raster m x n a vyplň ho '0' ;
        fronta.přidej(startovní_pole);
        fronta.přidej(prozkoumej(startovní_pole));
        While (fronta není prázdná){
                aktuální_pole := fronta.pop_front;
                možné_kroky := prozkoumej(aktuální_pole);
                While(možné_kroky nejsou prázdné){
                        if (možné_kroky > 1) fronta.přidej(aktuální_pole);
                        k := možné_kroky.náhodně_vyber_krok ;
                        proveď_krok(k);
                        možné_kroky := prozkoumej(aktuální_pole);}
        }
        místnost[x,y]:= true, pokud je v raster[x,y] rovný '1', false jinak.
}
```

5.4.5 Průchod ke dveřím a vlastnost nejbližšího pole

Algoritmus vyplní celou plochu hustým bludištěm. Toto bludiště má tu výhodnou vlastnost, že každé volné pole sousedí s polem se zdí a naopak. Dále je z charakteru algoritmu zřejmé, že systém volných polí tvoří souvislý systém ve smyslu pohybu ctícího pravoúhlé souřadnice.
Není vždy zaručeno, že všechna pole patřící k nějakým dveřím jsou přímo spojena s labyrintem. Máme-li dodržet vlastnost průchodnosti od dveří ke dveřím, musíme je spojit s nejbližším volným polem. (Z tranzitivity dostaneme potom průchodnost mezi každými dvěma dveřmi)
Za tímto účelem provedeme na výsledném rastru ještě poslední úpravu - proražení dveří.

Proražení dveří

Pro každé pole příslušné nějakým dveřím v aktuální místnosti najdeme obyčejným průchodem do šířky nejbližší volné pole. Všechna pole cesty nalezené tímto průchodem označíme jako volná.

5.4.6 Možnost parametrizovat

Tento algoritmus je možné zobecnit tak, že místo zkoušení kroku délky jedna by zkoušel krok obecné délky n či kratší v koncových případech. To by udělalo labyrint na pohled řidší, respektive by v něm nebylo tolik „zatáček“. Po otestování byl ovšem

zvolen model co nejhustšího labyrintu. Je smysluplnější vytvořit maximálně spletité bludiště vzhledem k tomu, že prodlouží herní čas.

5.4.7 Složitost

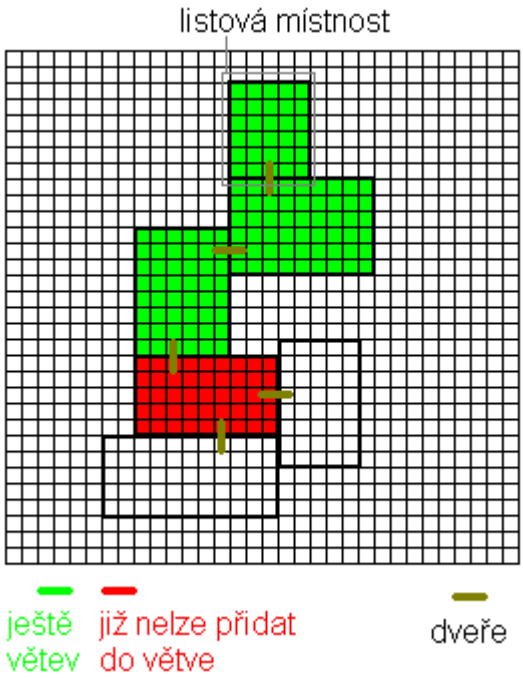
Algoritmus prozkoumává celý prostor sice rekurzivně, ale pole, která již jednou zpracoval, již znovu nezařadí do fronty. Jeho složitost je tedy lineární vzhledem k počtu polí ve vyplňovaném prostoru.

5.4.8 Užití algoritmu

Nyní zbývá ještě říci, jak se tento algoritmus použije při generování labyrintu. Některé místnosti mají zůstat prázdné a jiné vyplněné hustým bludištěm. Proto generátor náhodně zvolí místnosti, které mají být vyplněny hustým bludištěm a toto bludiště do nich nageneruje. Ostatní místnosti vyplní ve stylu základní místnosti.
Každá místnost je reprezentovaná samostatnou třídou, která drží informace o její struktuře polí. Toto pole se inicializuje výstupem algoritmu pro tvorbu hustého bludiště.

5.4.9 Rozlišení komponent

Kromě obsahu místností se ve druhé fázi generování identifikují všechny větve a určí se, která z nich bude finální.
K tomuto účelu se nejprve vyhledají místnosti, které mají pouze jedny dveře. Tyto místnosti jsou listy v grafu G a tedy budou nejhlubšími místnostmi větví. Poté se iterativně prozkoumávají sousední místnosti každého listu. Budované větve se rozšíří o tyto místnosti právě tehdy, když tyto obsahují pouze dvoje dveře a jsou tedy pokračováním grafové cesty popsané dříve.
Zbytek místností, tedy místnosti nepatřící do žádné větve, tvoří souvislou komponentu, kterou nazveme centrální komponentou. Ta odpovídá dříve popisované centrální části labyrintu.



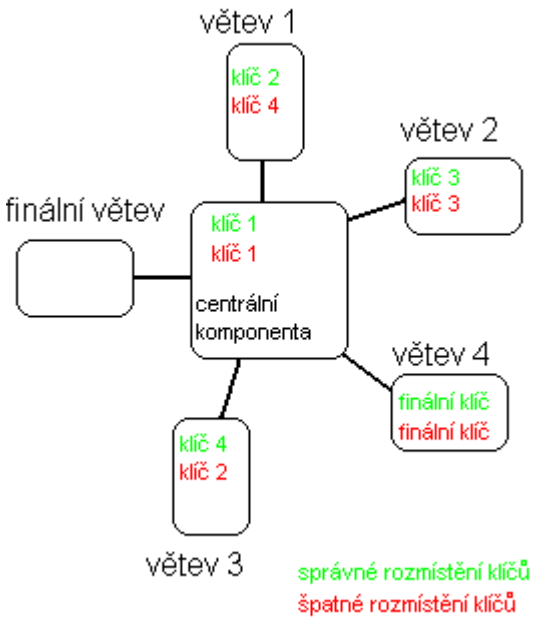
OBRÁZEK 6: Větev a první nepřidatelná místnost

5.4.10 Klíče a zamčení dveří

Abychom dostáli závazku nutnosti navštívit všechny listy grafu labyrintu k dokončení dobrodružství, musíme tyto speciální místnosti zpřístupňovat postupně. Proto všechny dveře mezi poslední místností každé větve a následující místností označíme za zamčené. Nyní musíme umístit klíče do labyrintu.
Nejprve musíme rozhodnout, která větev bude finální. Pokud se budeme dále odvolávat na finální větev, jde vždy o tuto konkrétní zde zvolenou větev.

Rozmístění klíčů

Klíče k zamčeným dveřím se budou umísťovat do listových místností (místností, které jsou listy v grafu labyrintu), aby bylo nutné k jejich získání projít všechny místnosti příslušné větve.
Do finální větve se nebude umisťovat žádný klíč a do zbylých větví se umístí právě jeden. Klíče se samozřejmě umisťují na platné pozice na volných polích.
Klíče začneme umisťovat od konce, tedy první se umístí klíč od finální větve. K tomu se zvolí libovolná nefinální větev. Dále se umístí klíč od větve, do které jsme umístili klíč. Takto se postupuje až do té doby, kdy nezbývá větev, do které ještě nebyl umístěn klíč. V tom okamžiku nám ovšem zbývá umístit pouze jediný klíč - máme totiž n klíčů od n větví a do jedné(do finální) se klíč neumisťuje. To je ovšem v pořádku, protože tento klíč umístíme na nějaké platné volné pole v centrální komponentě.
Celý postup tedy zaručuje požadované vlastnosti. Pokud bychom klíče neumístili tímto způsobem (existuje samozřejmě více permutací správného umístění klíčů), labyrint by byl sice vždy průchodný ve smyslu odemčení komponent, některé větve by ovšem nebyly vůbec přístupné a byly by tedy generovány zcela zbytečně.



OBRÁZEK 7: Špatné a srávné rozestavení klíčů v komponentách.

5.4.11 Vstup do labyrintu

V neposlední řadě se ve druhé fázi generování zvolí vstup do labyrintu, tedy místo, kde hrdina začne své dobrodružství. Vstup se musí samozřejmě umístit na volné pole. Dále toto pole musí ležet uvnitř centrální komponenty, aby mohl hrdina nalézt první klíč a tedy začít prozkoumávat labyrint.

5.5 Fáze třetí - umístění objektů

Na začátku této fáze je hotova práce s vlastním labyrintem. Teď jsou na pořadu objekty, které nemají stálou pozici, mohou být nějakým způsobem použity či vykazují nějaké interaktivní vlastnosti.
Dále opatříme jednotlivé místnosti a také komponenty textovým doprovodem.
Všechny předměty, nestvůry či textové komentáře jsou vkládány do labyrintu synchronizovaně tak, aby ve výsledku vytvořily tematicky sourodé celky. K tomu se využije organizace dat v databázi, která obsahuje kromě všech předmětů také prefabrikáty těchto celků.
Pro všechny reprezentace a následné uložení objektů se použije princip „třída-odkaz“ popsaný dříve.
Na konci této fáze je labyrint připraven k použití.

5.5.1 Obtížnost labyrintu

Obtížnost labyrintu ovlivňuje charakteristické vlastnosti předmětů a nestvůr. Objekty jsou v databázi uloženy v neparametrizované podobě - pokud je potřeba vytvořit nestvůru dané úrovně, její vlastnosti se přenásobí koeficientem odpovídajícím dané úrovni a teprve tato instance se uloží do vytvářeného labyrintu.

Nyní popíšeme vlastní procedury vyplňování všech komponent a jejich místností. Místnosti jsou vyplňovány odděleně.

5.5.2 Vyplňení větví

Tento případ se vztahuje obecně na vyplnění všech komponent (v případě finální větve popíšeme odlišnosti později).

Předpřipravené polotovary větví z databáze obsahují název komponenty, její popis, který je společný všem jejím místnostem, dále pak speciální legenda pro jednotlivé místnosti.
Dále jsou zde vyjmenovány všechny typy předmětů a také všechny typy nestvůr, které se mohou v komponentě vyskytovat. Ve skutečnosti jsou zde pouze odkazy do tabulky nestvůr a předmětů.

Položky z těchto tabulek obsahují již všechny relevantní informace pro jednotlivé objekty.

V případě tabulek nestvůr a předmětů nás při generování labyrintu zajímají pouze atributy s číselnými hodnotami. Jmenovitě položka id je identifikátor ke spojení tabulky nestvůr a odkazu, který do ní vede z položky definující komponentu.
Zbylé číselné charaktakteristiky nemají žádný vliv na tvorbu labyrintu.

Před adoptováním položky z databáze je potřeba některé číselné charakteristiky vynásobit koeficientem úrovně labyrintu. Dále již nakládáme s těmito položkami jako s daty, kterým nerozumíme.

Podrobný výklad struktury databáze bude podán v oddíle tři, konkrétně v části 7.4.

Popišme nyní jak celý proces probíhá. Nejprve určíme, kterou větev budeme vyplňovat. Z databáze se náhodně vybere prefabrikát obecné větve, podle kterého se nastaví popis a jméno.
Z fáze druhé máme pro tuto větev seznam místností, které do ní patří.
Každou místnost zpracujeme stejně.
Vybereme pro ni textovou legendu. Dále pro každou třídu nestvůr či předmětů vylosujeme číslo z intervalu 0..100 a porovnáme ho s pravděpodobností výskytu dané třídy uvedeného v instanci větve. Pokud je naše číslo menší, tato třída se bude vyskytovat. Nyní vylosujeme číslo z rozsahu 1..maxCount (maximální počet výskytů v rámci místnosti) a to bude počet instancí dané třídy v aktuálně zpracovávané místnosti. Poté vybereme pro každý výskyt platné pole v rozsahu místnosti. Pro potřeby grafického generátoru jsme umožnili existenci stavu, ve kterém se na stejném místě zároveň vyskytuje více nestvůr. Pro předměty je taková vlastnost přirozená. V runtime hře se z tohoto nastavení vychází, ale prostředí nedovolí žádnou akci, která by k němu vedla (smí ho ovšem zachovat).

Dřívější přístup

Dříve jsme počet nestvůr neomezili explicitně. Vycházeli jsme pouze z pravděpodobnosti výskytu, kterou jsme používali tak, že dokud byla náhoda uspokojena, byly přidávány další instance. To ovšem vedlo ke dvěma extrémům. Buď se v komponentě nevyskytovaly (téměř) žádné nestvůry, nebo naopak neúměrně velké množství.

5.5.3 Finální větev

Finální větev se od ostatních komponent liší v tom, že se zde kromě obyčejných nestvůr vyskytuje ještě hlavní nestvůra. Prefabrikát této komponenty se technicky liší pouze jednou přidanou položkou - odkazem na hlavní nestvůru. Tento odkaz může směřovat na kteroukoliv nestvůru, ovšem je smysluplné vytvořit zvláštní třídu nestvůr pro tento účel. (Přiložená databáze tak činí).
Kromě formálních omezení a podobností jsou v databázi v rámci těchto komponent obsaženy předměty v hojnější míře jako odměna za zdolání nástrah labyrintu.

Oddíl třetí

Implementace a použitá technologie

V tomto závěrečném oddíle se zaměříme na podrobnosti spojené s vývojem vlastní aplikace. Minulé dva oddíly byly psány v abstraktní rovině. Odhlíželo se od praktických témat spojených s implementací. Tématem tohoto oddílu bude popis použitých technologií a zamyšlení nad jejich výhodami či nevýhodami.

Dále se v tomto oddíle vrátíme na začátek práce. Projdeme znovu již jednou zmíněná témata a doplníme je o technické podrobnosti, které jsme vynechali. Nebudeme se zabývat tématy, která nesouvisí s implementací.
K nastíněnému modelu herního rámce představíme významné třídy se svými hlavními vlastnostmi a metodami. Ukážeme paralelu třívrstvého generování popsaného v oddíle dvě a způsobu, kterým je generátor skutečně naprogramován. Zaměříme se na strukturu hlavních tříd a jejich interakce.

Nebudeme se zde naopak vůbec zabývat žádnými podrobnostmi implementace vlastní hry, protože to bylo již součástí ročníkového projektu. Do rámce této práce spadá pouze namodelování a popsání světa a jeho následné parametrizované generování.

Kapitola 6

Použitá technologie

6.1 Programovací jazyk

Celý projekt je napsán v programovacím jazyce C#. Hlavní výhodou tohoto jazyka je bohatá nabídka funkcionality. Hlavní nevýhodou je přinejmenším omezená přenositelnost kódu na platformy, které nejsou založené na systému Windows.
Jazyk C# je samozřejmě součástí platformy .NET. Tato platforma nabízí komfortní práci s formuláři a ovládacími prvky.
Ročníkový projekt, na který tato práce navazuje byl v počátečních stádiích programován v C++. Vzhledem k tomu, že program vykazoval poměrně značnou interaktivitu, stalo se po čase neúnosným obsluhovat uživatelské rozhraní více méně ručně pomocí knihovny SDL. Kód spojený s pouhou obsluhou uživatelského rozhraní se stal neúnosně komplikovaným a rozsáhlým a jako takový byl zdrojem mnoha chyb.
Byla tedy zvolena alternativní cesta v podobě platformy .NET, která představuje moderní pojetí zpracování uživatelského rozhraní. V době, kdy byla zahájena práce na bakalářské práci byl k dispozici .NET verze 2.0. V průběhu implementace se objevila již stabilní a poměrně úspěšně nasazovaná verze 3.0, která dokonce řešila některé problémy, které v použité verzi nebyly postižitelné. Pro kontinuitu prací jsme ovšem zůstali u použití verze 2.0.

6.2 Použité konstrukty

Při vývoji generátoru a jeho grafického rozhraní bylo použito mnoho moderních a efektivních nástrojů poskytovaných platformou .NET.
Generátor byl vytvořen jako grafická aplikace pro windows používající winforms a ovládací prvky windows. K obsluze akcí uživatele bylo samozřejmě použito systému událostí a jejich handlerů. Události ovšem byly použity i pro řízení chodu generátoru.
O bezpečnost a stabilitu aplikace se stará hojné použití vyjímek a jejich odchytávání. Aplikace je poměrně stabilní i při velmi nestandardním použití.
Pro manipulaci se soubory byly většinou použity streamy.

Použité komponenty

Během celé implementace nebylo zapotřebí využít žádné knihovny, která by poskytovala funkcionalitu nepodporovanou v jazyce C# platformy .NET 2.0. Jedinou výjimkou by byla serializace, která v prostředí .NET 2.0 neposkytuje vše, co by bylo zapotřebí. Přesto byly tyto komplikace vyřešeny v rámci verze 2.0. Tomuto tématu se budeme věnovat níže.

6.3 Trvalá reprezentace dat (na disku)

Potom, co jsme v předchozím oddíle popsali, jak labyrint generovat, stojí za zmínku také to, jak výsledný produkt trvale reprezentovat, tedy, jak ho uložit do souboru a jaký datový formát k tomu použít.

Použitý framework .NET v kombinaci s jazykem C# nabízí všestranně podporovaný formát XML. Další použitelnou možností je vlastnost [Serializable], umožňující přenechat fyzickou manipulaci s daty systému a pouze určit, jaký je jejich formát. Takto lze data exportovat nejen do XML, ale i například do binárních souborů. Tato vlastnost je mnohem obecnější a silnější, ale zbylé scénáře použití jsou v tuto chvíli irelevantní.

XML nebo binární soubory

Pro to, abychom zvolili formu, ve které data uložíme, musíme porovnat výhody a nevýhody zvažovaných reprezentací stejně jako jejich důležitost.

6.3.1 XML

Výhodou formátu XML je jeho čitelnost. Je nespornou výhodou moci data, která mají čitelný charakter, zobrazit bez nutnosti použít nějaké speciální programové vybavení. XML dokument má čitelnou nejen datovou část, ale pokud je vhodně navržen, je snadné se v něm vyznat, protože názvy jednotlivých entiti či atributů jsou samovysvětlující. Není zdaleka praktické kontrolovat výsledky generování tím, že budeme zkoušet každý nový labyrint předat hře a pomocí hraní ověřovat jeho správnost.
Kromě toho, že můžeme data sami snadno zobrazit například v každém alespoň trochu moderním webovém prohlížeči či prostě v textovém editoru, tento formát poskytuje díky své přesné a jednotné definici skvělý základ pro dobrou přenositelnost. Pokud popíšeme strukturu XML souboru, například pomocí formátu DTD nebo XSD, můžeme s velkou šancí použít data z generátoru na libovolné architektuře nebo přinejmenším operačním systému.
Bohužel i formát XML má nedostatky. Tím hlavním je poměr velikost - obsah, respektive velmi malá hustota dat. Je možné používat co možná nejkratší názvy elementů a atributů, to ovšem potírá výše zmíněné výhody.
Přesto však vzhledem k velikosti dnešních disků a aktuálním trendům v přístupu k uživatelským datům věřím, že mapy velikosti řádově několika málo set kilobytů nebudou nikterak omezující. Případně by se dalo uvažovat o použití komprimace, která na tyto data funguje velice efektivně.
XML formát jsme tedy zvolili za diskovou reprezentaci jak výsledného labyrintu, tak i databáze.

6.3.2 Binární soubory

V jednoduchosti by se dalo napsat, že vlastnosti binární reprezentace jsou při nasazení na náš problém doplňkem vlastností XML. Soubory jsou malé a kompaktní, nelze z nich ovšem příliš vyčíst a jejich zpracování je pro třetí stranu přinejmenším velice nesnadné, tedy přenositelnosti tím nevycházíme vstříc.
Je zde ovšem jedno použití - vlastní data generátoru, jako je nastavení či jiné persistentí informace platné i po skončení běhu programu.

6.3.3 Serializace

Princip serializace je velice praktický a může ušetřit mnoho práce, chyb v syntaxi cílového souboru a konečne také řádků kódu.
Bohužel .NET 2.0 neposkytuje takovou funkcionalitu, aby bylo možno serializaci efektivně použít pro ukládání labyrintu. Pro potřeby práce s databází ovšem není v jeho použití problém.
V čem tedy tkví nesnáze? Serializovat lze celé třídy, ovšem pouze jejich public vlastnosti. Verze .NETu 3.0 řeší serializaci odlišně a lze se tomuto vyhnout. Ovšem v poslední fázi projektu jsme raději vytvořili výstupní formát ručně, respektive méně automaticky pomocí standardních rutin pro vytváření XML elementů a atributů, které jsme naplnili daty manuálně.

6.4 GUI

Grafické uživatelské rozhraní využívalo, až na jednu výjimku, standardní ovládací prvky pro formuláře jako button, radio button, rich text boxy a podobně. Aplikace je přirozeně řízena událostmi. K jednotlivým událostem různých ovládacích prvků byly napsány příslušné obslužné metody.
Naprogramování takového kódu bylo pracné, ale naprosto standardní.
Hlavním cílem grafické verze generátoru je přehledná prezentace průběhu generování. Za tímto účelem bylo potřeba vizualizovat některé situace a vlastnosti labyrintu.

6.4.1 Prezentační okno

Právě zobrazení aktuálního stavu labyrintu je onou výše zmíněnou výjimkou. Protože bylo potřeba vykreslovat aktuální konfiguraci polí v labyrintu v rámci formuláře, museli jsme přijít s nějakým řešením. Oblasti, na které zobrazujeme stav labyrintu, nazveme prezentační okno. Celkem se v GUI vyskytují tři různá prezentační okna.
Další funkcionalitou, kterou je potřeba zaručit, je obsluha klikání myší do tohoto prezentačního okna, abychom z něj vytěžili maximální uživatelský komfort.
Tento problém není rutinním použitím komponent, popíšeme ho tedy detailněji.

6.4.2 Pomalé vykreslování

Prvním pokusem bylo kreslení na formulář. Základní nevýhodou je, že po automatickém překreslení formuláře zmizí vše, co jsme vykreslili. Takovéto automatické překreslení je poměrně časté, pokud formulář obsahuje více ovládacích prvků, které jím manipulují. Je možné překrýt metodu obsluhující událost Paint a provézt celé kreslení vždy, když se kreslí i formulář. To je ovšem značně neefektivní, protože se toto překreslení nedá optimalizovat. Formulář se tedy musí vykreslit celý.
Dalším kamenem úrazu se ukázal být fakt, že jsme kreslili jedno pole po druhém přímo na formulář. Takové kreslení je hrozivě pomalé. Při maximálním velikosti labyrintu sto krát sto polí je potřeba deset tisíc krát provést operaci kreslení. Tato operace má vedlejší náklady typu zamčení plochy, na kterou se kreslí. Veškerá inicializace před vlastním kreslením a operace po něm jsou ve skutečnosti potřeba jen jednou.
Takovýto přístup byl neúnosně pomalý.

6.4.3 Rychlé vykreslování

Zde shrneme další dvě iterace pokroku najednou. Ukázalo se, že čelíme dvěma problémům - ztrátě vizualizace po vykreslení a neefektivním kreslení.
Řešení problému rychlosti kreslení bylo přímočaré. Operace s formulářem jsou pomalé, je nutné je provádět jen v nezbytných situacích. Je vhodné celou scénu, kterou chceme mít nakonec vykreslenou, připravit pouze v paměti. K tomu bylo s výhodou použito třídy Bitmap. Statickou metodou třídy Graphics jsme získali grafický handler bitmapy a provedli dále kreslení zcela stejně, jako bychom kreslili s použitím grafického handleru formuláře. Takovéto vykreslení probíhá z pohledu člověka v nulovém čase, protože je to jen velice malá operace s pamětí.
Takto jsme nyní získali Bitmap s požadovaným obsahem. Místo kreslení na plochu formuláře jsme na cílové místo vložili komponentu pictureBox. Pokaždé, když se scéna změnila, vytvořili jsme novou bitmapu, kterou jsme potom předali jako zdrojový obrázek komponentě pictureBox. O vykreslování komponent se za nás stará .NET automaticky, takže poté co zavoláme jednou metodu Refresh pro aplikaci nového obrázku se při každém překreslení formuláře překreslí automaticky a především optimalizovaně i náš obrázek.

6.4.4 Obsluha událostí

Jak jsme si předsevzali, je nezbytné obsloužit událost kliknutí na prezentační okno. Tato událost je zachycena jako kliknutí na pictureBox. Odtud dostaneme absolutní souřadnice kliknutí. Tato informace nám nepomůže. Potřebujeme zjistit, na které pole labyrintu uživatel kliknul. Situaci stěžuje ještě fakt, že pole jsou ve dvou ze tří prezentačních oken velká podle toho, jak je velký labyrint. Po vypočítání pole, na které bylo kliknuto, se zavolá příslušná metoda, která obsluhuje kliknutí na pole labyrintu.

V podstatě jsme na základě objektu typu pictureBox vytvořili novou komponentu a obsluhujeme ji také na základě událostí.

6.5 Trocha softwarového inženýrství

Pro úplnost závěrem této kapitoly zatřídíme náš projekt z pohledu softwarového inženýrství

Pipes and Filters

Postup, kterým jsme navrhli a implementovali engine generátoru odpovídá design pattern „Pipes and Filters“. Principem tohoto přístupu je sekvenční zpracování. Vstupem i výstupem každé operace jsou data v konzistentní a ucelené formě. Každá etapa má smysluplný výstup a lze ji provést nezávisle na ostatních iteracích.
To odpovídá rozvržení generování na fáze. Každá fáze je svébytná a samostatná. Provedení jedné fáze nevyžaduje nutně provedení následujících fází. Pokud by to bylo nutné, dal by se náš generátor rozložit na tři samostatné aplikace a jejich výstup respektive vstup spojit po vzoru unixových aplikací, které jsou asi nejznámější ukázkou tohoto design patternu v praxi.

Kapitola 7

Popis objektového a datového modelu

Cílem této kapitoly je přiblížit strukturu tříd, které implementují doposud pouze formálně popsané algoritmy. Tato kapitola by měla dát základní představu o fungování aplikace tomu, kdo by v jejím vývoji chtěl pokračovat. K plnému porozumění návrhu je nutné přečíst i dokumentaci, která je přílohou této práce. V druhé části kapitoly potom nastíníme, v jakém formátu a rozsahu ukládáme cílová data na disk.

7.1 Třídy generátoru

Jak bylo již několikrát zmíněno, generátor je navržen ve třech úrovních. Toto pojetí bylo zachováno i při vlastní implementaci.
Pro udržování právě generovaných dat byla pro každou vrstvu vytvořena jedna generace hlavní třídy, která reprezentuje labyrint. V každé fázi byly přidány ty vlastnosti, které spolu tvoří kompaktní a úplný celek a nevyžadují žádné interakce s později přidanými vlastnostmi či daty. Tak byly konečně i jednotlivé fáze navrženy.
Popišme tedy nyní jednotlivé vrstvy. Zopakujme, že termín fáze generování a vrstva labyrintu si vzájemně odpovídají a společně popisují jedno konkrétní stádium generování.
Poznámka: všechny komentáře v kódu jsou napsány v anglickém jazyce. Ve stejném duchu jsou pojmenovány i jednotlivé metody a třídy. Pokud by měl totiž kód být podkladem k nějakým dalším pracím, je třeba se s ním seznámit. Pro čtenáře, kteří nemluví česky je téměř nemožné porozumět kódu, protože je pro ně v podstatě neokomentovaný. Logiku fungování česky pojmenovaných tříd a metod lze také jen obtížně pojmout. Máme za to, že je přinejmenším dobrým zvykem používat při vlastním programování pouze anglický jazyk. Ten by konec konců měl být,alespoň na úrovní čtení a porozumění, základní výbavou každého moderního informatika.

7.1.1 Vrstva první - vrstva oblastí (pro místnosti)

Název hlavní třídy: Areas

Tato vrstva má za úkol poskytnout mechanismus na dělení místností, umístění dveří a nagenerování potřebného počtu místností/komponent.

Místnost - Region

V tuto chvíli je místnost nazývána oblastí. Reprezentuje ji třída Region. Tato třída uchovává informace o poloze a velikosti místnosti, seznam místností, se kterými je spojena a seznam dveří.
Dveře v seznamu jsou vytvořeny podle seznamu místností, se kterými je daná místnost spojena.
Dělení místnosti probíhá fakticky přímo na nejnižší úrovni. Třída poskytuje metody pro zjištění, zda je ještě dělitelná (tj. má dostatečnou velikost) a metody pro podélné a

svislé rozdělení, které vrací druhý produkt dělení. Původní místnost se naopak přeformuje na druhý produkt dělení.
Pro tvorbu dveří místnost umí zkontrolovat, zda sousedí s jinou, parametrem zadanou místností.

Dveře - Door

Tato třída obsahuje informace o dvou polích náležících daným dveřím a příznak zamčenosti.

Společná hrana - Border

Tato třída po inicializování dvěma sousedními místnostmi vypočítá všechny možné polohy dveří mezi těmito místnostmi.

Vrstva oblastí - Areas

Toto je první generace hlavní třídy držící informace o celé fázi generování. Její instance je po skončení první fáze předána do druhé fáze.
Jsou zde umístěny seznamy místností, dveří, spojených dvojic místností, a některé pomocné informace.
Tato třída je rozhraním, které zpracovává všechny požadavky na generování.
Třída nabízí možnost přidat oblast, přímo ručně rozdělit požadovanou oblast, generovat dveře, kontrolovat počet vygenerovaných komponent. Dále pro potřeby GUI vrací raster o jemnosti jedno pole pro vlastnost „místnost-zeď“ a také raster obsahující odkazy do místností pro jednotlivá pole.
Metody pro dělení oblastí volají dílčí metody místností. Princip a postup při dělení se drží popsané strategie v algoritmické části.

7.1.2 Vrstva druhá - vyplněné místnosti

Název hlavní třídy: FilledAreas

Tato vrstva má na starosti vyplňování místností hustým bludištěm. Dále potom vytyčení jednotlivých větví, zamčení dveří a umístění klíčů.

Místnost - Room

Tato třída je potomkem třídy Region. Datovou část rozšiřuje o raster reprezentující rozložení zdí v rámci místnosti, dále pak obsahuje textovou legendu naplněnou ve fázi tři. Ve třídě je ještě příznak určující, zda je místnost vyplněna hustým bludištěm, nebo se jedná o místnost základní.
Základními metodami jsou BasicFill a MazeFill. BasicFill vyplní krajní vrstvu polí zdí. MazeFill volá na začátek BasicFill a potom vyplní zbytek místnosti hustým bludištěm, jak bylo popsáno v teoretické části této práce.
Metoda AddDoors prorazí pole s dveřmi a zajistí průchodnost mezi všemi dvojicemi dveří, jak bylo popsáno v oddíle druhém.
Pro potřeby ruční editace v GUI je zavedena metoda SetWallRaster, která jednoduše přepíše celé rozložení zdí podle předaného parametru.
Pro prezentaci v GUI slouží metoda GetWallRaster, která vrací raster polí s vlastností „zeď-volno“

Klíč - Key

Instance této třídy reprezentuje klíč ke konkrétním dveřím. Obsahuje informace o poloze dveří (dvě pole, která dveře spojují) a o vlastní poloze klíče. Klíč má také svůj identifikátor.

Vrstva vyplněných místností - Filledreas

Hlavní třída druhé fáze je potomkem třídy Areas. Tu rozšiřuje o seznam větví, centrální komponentu, vstup, seznam klíčů a konečně seznam místností.
Větve a centrální komponenta jsou zde pouze seznamy místností, které je budou tvořit. Vstup je pole v centrální komponentě.
Třída se inicializuje parametrem typu Region, který dostane z první fáze generování. Popišme novou funkcionalitu:
Metoda AddEnter umístí vstup do labyrintu na platné volné pole centrální komponenty.
LockDoors identifikuje dveře na rozhraní větví a centrální komponenty a zamkne je.
Metoda DistributeKeys implementuje algoritmus umísťující klíče z oddílu dvě pro dveře, které byly zamčeny metodou LockDoors.
FillRegions vyplní všechny místnosti zdmi. Náhodně se rozhodne, které budou obsahovat husté bludiště a které budou základními místnostmi.
Opět se zde vyskytuje mnoho metod pro přístup k různým informacím prezentovaným v GUI.

7.1.3 Vrstva třetí

Název hlavní třídy: Map

V této vrstvě se do labyrintu umístí předměty a nestvůry. Do větví s místností se doplní textové komentáře a popisy.

Instance předmětů - ItemInstance

Třída ItemInstance reprezentuje instanci předmětu ve smyslu „třída-odkaz“. Takto jsou reprezentovány všechny předměty kromě klíčů. V této třídě jsou uložena data, která jsou pro daný předmět specifická, tedy pozice, odkaz do tabulky předmětů a identifikátor místnosti, kde se předmět nalézá. Zbytek informací o daném předmětu je přístupný přes spojení tabulek typů předmětů. Takové spojení je možné, protože všechny předměty kromě klíčů sdílí stejnou doménu identifikátorů.

Instance nestvůr - CreatureInstance

Třída CreatureInstance je paralelou třídy ItemInstance.

Tabulka předmětů a nestvůr

Úplné tabulky předmětů a nestvůr jsou součástí databáze. V průběhu třetí fáze generování jsou do labyrintu přidávány instance předmětů a nestvůr. Protože velikost úplných tabulek není nijak omezena, není praktické do výstupu generování zahrnovat tyto tabulky celé. Proto se postupně tvoří tabulka minimální velikosti obsahující všechny třídy objektů, které jsou adresovány instancemi.
Po přidání libovolné instance nějaké třídy objektu se zkontroluje, zda tabulka tříd již obsahuje položku, na kterou se daná instance obsahuje. Pokud ne, tato třída se do tabulky vloží.

Větev - Branch

Tato třída modeluje větev labyrintu. Zahrnuje textový popis větve společný pro všechny místnosti, které do ní patří. Dále obsahuje vlastní název, vlastní identifikátor a konečně seznam identifikátorů místností, které do dané větve patří.
Třída Branch se používá pro reprezentaci všech tří typů komponent - klasické větve, finální komponenty a také centrální komponenty.

Vrstva předmětů a nestvůr - Map

Třída Map je potomkem třídy FilledAreas, kterou rozšiřuje o seznam větví, o zvlášť určenou finální komponentu, a o zvlášť reprezentovanou centrální komponentu. Dále je zde instance nestvůry pro finální nestvůru, seznam instancí nestvůr pro řadové nestvůry a tabulka typů nestvůr. Pro potřebu uložení předmětů je zde seznam instancí předmětů a také tabulka typů předmětů.
Tabulky nestvůr a předmětů jsou klasickým seznamem jednotlivých položek. Třída také zmiňuje obtížnost labyrintu.
Třída se inicializuje metodou init s parametrem typu FilledAreas předaným z předchozí fáze generování.
Metoda FillBranch vyplňuje všechny místnosti zadané větve objekty podle prefabrikátu komponenty z databáze. Tato metoda implementuje přístup z části 5.4.2.
Metody GenerateBranches, GenerateCenterPart a GenerateFinalBranch specifickým a případně následným voláním FillBranch nagenerují nestvůry a předměty do všech komponent podle jejich typu.
Tato třída obsahuje metodu serialize, která ji exportuje do formátu XML. Tuto metodu ovšem z praktických důvodů nakonec v rámci grafického generátoru nepoužíváme.
V této třídě je ještě celá řada pomocných metod pro potřeby GUI a také pro potřeby generování.

7.2 Wrapper class generator

Cíl celého generování je jen jediný - vytvořit labyrint. Tříd a metod, které jsme za tímto účelem vytvořili, je ovšem velmi mnoho. K úspěšnému vygenerování nového labyrintu je potřeba dodržet závazné pořadí volání některých metod. Několikanásobné volání některých metod může vést k nekonzistencím.
Je tedy zřejmé, že celý engine by měl generování sám řídit a koordinovat.
Proto byla vytvořena třída generator, která obaluje všechny tři generace tříd a jejich součásti, které byly popsány v 7.1.
Poskytovaný interface umožňuje generovat první, druhou a třetí fázi, načíst a uložit nastavení generátoru a konečně uložit vytvořený labyrint jako soubor XML.
Generátor obsahuje také seznam chybových hlášek a seznam varování. Pokud se nějaká akce nezdaří, anuluje se dopad této akce a do seznamu chybových hlášek se zapíše příslušné hlášení.

7.3 Databáze

Na databázi jsme se doposud odvolávali v poměrně abstraktní rovině. Dotknuli jsme se její reprezentace ve formátu XML a také jsme popsali data, která obsahuje. Tedy první možností je nahlížet na databázi jako na zdroj strukturovaných dat.
Při implementaci je ovšem nezbytné s takovým zdrojem dat nějak zacházet, nějak k datům přistupovat. Tady narážíme na druhý možný pohled na databázi, tedy jako na

třídu, která poskytuje rozhraní, přes které se k datům přistupuje. Navrhli jsme třídu, která se inicializuje ze souboru ve formátu XML. Třída se jmenuje jednoduše database. Po inicializaci je možné třídu přes její interface dotazovat na třídy předmětů a nestvůr, či data číst přímo ze seznamů. Jsou zde drženy seznamy prefabrikátů větví a místností, potom seznamy zbraní, ochranných pomůcek či lektvarů. K předmětům je zde i generický seznam položek typu item. Item popíšeme níže.
Protože je databázi potřeba načítat ze souboru, využili jsme možnosti serializace, které jsou v tomto případě zcela postačující. Databáze se během generování nemění a není ji tedy potřeba ukládat.

Item

Tato třída je společným předkem tříd pro všechny předměty. Tvrdili jsme, že se všemi předměty zacházíme jednotně. Proto se s předměty manipuluje přes tuto společnou předkovskou třídu a přes mechanismus virtuálních metod.

7.4 Popis formátu databáze

Závěrem této kapitoly popíšeme strukturu datových souborů pro databázi a pro uložení mapy. Zmíníme způsob, kterým ukládáme hotový labyrint do souboru.

7.4.1 XML struktura databáze

Připomeňme si ještě jednou princip ukládání informací o jednotlivých objektech nazvaný „odkaz-třída“. Při umístění objektu do labyrintu se uloží pouze jeho poloha a odkaz do tabulky tříd objektů. Všechny tabulky objektů jsou v plném znění uloženy právě v databázi. Z databáze se do cílové reprezentace labyrintu zkopírují jen položky tabulky odpovídající objektům přítomným v labyrintu.

Nejvyšší úroveň databázového souboru obsahuje tabulky předmětů a nestvůr, prefabrikáty komponent a prefabrikáty finální komponenty.

Strukturu dat představíme na příkladech zástupců jednotlivých tříd.

7.4.2 Předměty

Pro realizaci kategorií předmětů byla v rámci enginu generátoru zavedena hierarchie s abstraktním předkem item. Od ní byly odvozeny třídy defence, potion a weapon.
Pro všechny tři kategorie byla použita jediná doména identifikátorů. Výhoda společného adresového prostoru tkví v tom, že se dá s předměty zacházet univerzálně nejen díky polymorfismu, který je k dispozici v samotném programu, ale i při práci se samostatným datovým souborem. To je nezbytné pro předání dat z generátoru do hry. Generátor a aplikace používající vygenerovaný labyrint jsou dva rozdílné programy komunikující pouze přes jednotné souborové rozhraní.
Struktura hierarchie i organizace identifikátorů je společná aplikaci generátoru i fyzické reprezentaci v XML formátu.

Následují příklady položek předmětů.

```
<Item xsi:type="Weapon">
        <level>1</level>
        <id>1</id>
        <attack>4</attack>
        <bonus>1</bonus>
        <name>kratky mec</name>
        <text>Nijak zajimavy kratky mec s mirne ztupenou cepeli1.</text>
</Item>

<Item xsi:type="Defence">
        <level>1</level>
        <id>5</id>
        <defence>1</defence>
        <name>stit1</name>
        <text>Lehce nastiply plochy kulaty stit.</text>
</Item>

<Item xsi:type="Potion">
        <level>1</level>
        <id>10</id>
        <heals>5</heals>
        <name>lektvar rudeho krize</name>
        <text>Lahvicka rude tekutiny s hranicarskym napisem L.R.K.</text>
</Item>
```

7.4.3 Nestvůry

V případě nestvůr nebyla zavedena žádná hierarchie. Na jedné jediné úrovni stojí všechny typy nestvůr. Dokonce ani finální nestvůra není nijak formálně odlišena od ostatních. Tohoto odlišení dosáhneme až instanciací tříd nestvůr.
Situace s tabulkou tříd a jejím uložením do výsledné reprezentace generátoru je stejná, jako v případě předmětů.

Uveďme příklad jedné položky reprezentující nestvůru.

```
<Creature>
        <id>6</id>
        <name>Černokněžník</name>
        <noise>Listování v knihách a občasné potažení z dýmky.</noise>
        <level>10</level>
        <gold>2000</gold>
        <exp>1000</exp>
        <attack>6</attack>
        <defence>5</defence>
        <text>Stařec s bílými vlasy a pěstěnými leč řídkými vousy. Přes jeho
očividný věk z něj vyzařuje až nadpozemská vitalita.</text>
</Creature>
```

7.4.4 Prefabrikáty komponent

Prefabrikáty komponent obsahují název komponenty, textový popis komponenty, seznam přípustných typů předmětů a nestvůr, které se mohou v dané komponentě nacházet a také seznam popisů různých místností, které jsou součástí dané komponenty. Položky nestvůr a předmětů obsahují element tripple. Tripple obsahuje za prvé maximální počet instancí daného objektu v jedné místnosti, za druhé pravděpodobnost, že se objekt vůbec vyskytne a za třetí odkaz do tabulky tříd objektů (předmětů či nestvůr).

Příklad jedné položky prefabrikátu komponenty:
(pro potřeby této ukázky jsme pro každou kategorii vybrali jen jednu položku)

```
<BranchClass>
        <name>Stoka</name>
        <Description>Vstoupil jsi do zelene stoky...</Description>
        <Creatures>
                <tripple>
                        <maxCount>2</maxCount>
                        <probability>30</probability>
                        <id>1</id>
                </tripple>
        </Creatures>
        <Items>
                <tripple>
                        <maxCount>2</maxCount>
                        <probability>30</probability>
                        <id>5</id>
                </tripple>
        </Items>
        <RoomComments>
                <string>Tato místnost připomíná </string>
                <string>Stěny místnosti jsou porostlé </string>
        </RoomComments>
</BranchClass>
```

7.4.5 Prefabrikáty finální komponenty

Obsah prefabrikátu finální komponenty se syntakticky liší pouze přidáním elementu BossCreature pro finální nestvůru.
Pro smysluplně navržený labyrint by ovšem měl polotovar finální komponenty obsahovat silné nestvůry a cenné předměty.

7.5 Popis formátu souboru s labyrintem

Protože je způsob reprezentace databáze a hotového labyrintu v mnoha ohledech stejný, popíšeme reprezentaci formátu souboru s labyrintem jen velmi schematicky.

7.5.1 XML struktura labyrintu

Root element obsahuje jako atribut definici velikosti a úrovně obtížnosti labyrintu.

Dále jsou jako jednoduchý element uloženy souřadnice vstupu.
Zvláštní element door je vyčleněn pro dveře. Ten obsahuje seznam všech dveří jako dvojic souřadnic polí, která se k daným dveřím váží.

7.5.2 Předměty a nestvůry

Pro předměty a nestvůry je vždy vytvořen zvláštní element pro třídu objektu a zvláštní element pro seznam instancí. Elementy pro třídy objektů mají zcela stejný formát jako v databázi (viz 7.4.2, 7.4.3). Instance objektů jsou prosté elementy, které obsahují pozici a odkaze do tabulky tříd.
Jediná nestvůra, která je uložena mimo tuto strukturu je finální nestvůra. (viz níže)

Příklad třídy předmětu:
<defence id="6" name="Kulatý štít" defence="1" text="Lehce nastiply plochy kulaty stit.2" level="2" />

Příklad instance předmětu:
<item x="45" y="75" class="2" />

7.5.3 Komponenty (větve)

Komponenty jsou v podstatě pouze seznamem místností, které do nich patří. Spolu se seznamem místností je uložen ještě textový popis komponenty a její jméno.
Finální komponenta je uložena samostatně. Oproti standardním komponentám obsahuje ještě položku finální nestvůry s jejími souřadnicemi a odkazem do tabulky nestvůr.
Příklad komponenty:

```
<branch name="pohrebiste" text="Vstoupil jsi ...">
        <room id="21" />
        …
        <room id="26" />
</branch>
```

Příklad finální komponenty:

```
<final_branch name="draci sluje" text="Vsude kolem se vali polamane kusy vyzbroje. V rohu mistnosti je obrovita kupa vsemoznych pokladu. A v mistnosti je jeste neco....">
        <room id="42" />
        …
        <room id="46" />
        <boss class="5" x="36" y="27" room="42" />
</final_branch>
```

7.5.4 Místnosti

Místnosti jsou označeny identifikátorem a je udána jejich velikost a poloha. Informace je držena i o doprovodném textu.

Příklad:

```
<room id="0" sizeX="9" sizeY="8" posX="10" posY="20">
        <info text="Tato místnost Vás na první pohled ničím nezaujala..." />
</room>
```

7.5.5 Bludiště

Poslední věc, kterou jsme nezmínili, je vlastní rozložení zdí a volného prostoru v labyrintu.
Na pozice polí mapy s vlastností zeď nemůže hrdina ani nestvůra vstoupit ani zde nemohou být umístěny žádné předměty. Tato pole nemohou být ani v relaci dveře. Není tedy potřeba je explicitně zaznamenávat. Pokud je z nějakého důvodu potřebujeme, můžeme je dopočítat ze znalosti „volných“ polí.
Do XML souboru tedy ukládáme pouze informace o volných polích, čímž razantně snížíme velikost výsledného souboru. U netriviálně velkých bludišť automaticky vytvořených naším generátorem se ve standardních případech jedná o desítky procent.
Volná pole jsou organizována po řádcích shora dolů. V rámci řádku pak zleva doprava.

7.6 XML Writer

Již v části 6.3.3 jsme zmiňovali nedostatky serializace v .NETu verze 2.0. Proto jsme přistoupili k pracnějšímu způsobu tvorby výsledného XML souboru. Toto řešení však na druhou stranu dovoluje vytvořit XML dokument zcela libovolně a mít přesnou kontrolu nad exportovanými daty. Lze tedy ještě dodatečně kontrolovat validitu dat, která hodláme uložit. Lze také ukládat i data z private položek a ukládaná data před zápisem přeformátovat.
XML_Writer je třída navržená čistě pro export produktu třetí fáze generování do XML. Jediná metoda Write postupně v paměti vytvoří XML dokument a ten potom exportuje do cílového souboru. Pro vytvoření tohoto XML dokumentu je použito standardních tříd jazyka C#, jako XmlElement či XmlDocument.

7.7 Namespace

Protože má tato část sloužit potencionálními pokračovateli projektu k základní orientaci, je nutné pro úplnost zmínit, že jsme navržené třídy rozdělili do několika namespace. Třídy vážící se ke každé jednotlivé fázi generování jsou umístěny ve zvláštním namespace (Generate_Areas_part01, Generate_Rooms_part02, Generate_part03). Namespace blud drží třídu generující husté bludiště ve fázi dvě. Generate_database obsahuje databázi a třídy s ní spojené. Namespace Generator obsahuje třídu generátor (viz 7.2) a třídu držící parametry generování (ta je triviální a tudíž jsme ji nepopisovali). Konečně Xml_Writer je namespace se třídou pro export do XML.

Závěr

Téma, které tato bakalářská práce zpracovává, je velice široké a rozsáhlé. V této práci jsme představili poměrně obecnou a snadno rozšiřovatelnou reprezentaci světa Dračího doupěte, která by mohla být základem pro další projekty.

Vývoj aplikace přinesl autorovi cenné zkušenosti pramenící z používání celé řady technických nástrojů. Velkou hodnotu mají zkušenosti nabité při tvorbě textu práce.

Značným přínosem byla možnost poznat průběh prací na rozsáhlém projektu, při kterých bylo nutno koordinovat návrh, design, programování, předkládání průběžných výsledků a jejich konzultace s vedoucí projektu. Bylo nezbytné učitit několik kritických rozhodnutí.

Implementaci považujeme za úspěšnou a všechny cíle, které jsme si předsevzali, za naplněné.

Použitá literatura:

Martin Klíma *Pravidla Dračího doupěte*, nakladatelství ALTAR, 1990