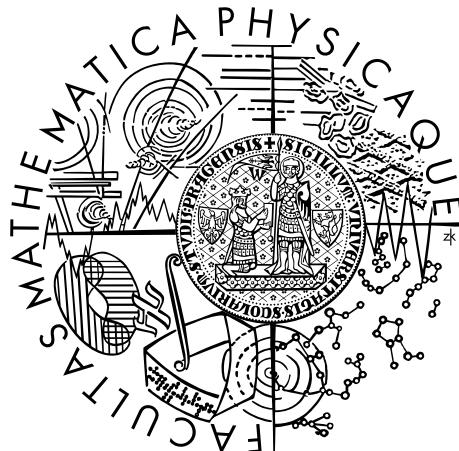


Univerzita Karlova v Praze
Matematicko-fyzikální fakulta

BAKALÁŘSKÁ PRÁCE



Petr Geiger

Hra Dungeon Master pro platformu .NET

Katedra distribuovaných a spolehlivých systémů

Vedoucí bakalářské práce: Mgr. Ježek Pavel Ph.D.

Studijní program: Informatika

Studijní obor: Programovaní a softwarové systémy

Praha 2016

Prohlašuji, že jsem tuto bakalářskou práci vypracoval(a) samostatně a výhradně s použitím citovaných pramenů, literatury a dalších odborných zdrojů.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona v platném znění, zejména skutečnost, že Univerzita Karlova v Praze má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle §60 odst. 1 autorského zákona.

V dne

Podpis autora

Název práce: Hra Dungeon Master pro platformu .NET

Autor: Petr Geiger

Katedra: Katedra distribuovaných a spolehlivých systémů

Vedoucí bakalářské práce: Mgr. Ježek Pavel Ph.D., Katedra distribuovaných a spolehlivých systémů

Abstrakt: Cílem bakalářské práce je reimplementovat hru Dungeon Master. V současné době existuje již několik klonů této známé hry. Nicméně oproti nim se tato práce zaměřuje především na následující aspekty. Hra je naprogramována v jazyce C# s využitím platformy .NET. Dále celý engine je navrhnutý směrem k udržitelnosti a rozšířitelnosti. Tzn. s využitím tohoto enginu je možné naprogramovat i jinou hru založenou na podobných základech. Ale především je jednoduché přidávat do enginu nové funkce. Engine je také připravený na rozdílné vstupní formáty herních úrovní. Dále je také kompletně oddělena zobrazovací vrstva. Vzhledem k povaze projektu by engine měl sloužit jako ukázkový příklad použitelný pro vzdělávání.

Klíčová slova: RPG Dungeon Master Architektura softwaru vzdělávání

Title: Dungeon Master Game for the .NET Platform

Author: Petr Geiger

Department: Department of Distributed and Dependable Systems

Supervisor: Mgr. Ježek Pavel Ph.D., Department of Distributed and Dependable Systems

Abstract: The aim of this thesis is to reimplement the game Dungeon Master. There are currently several clones of this wellknown game. However, compared to them this work focuses on the following aspects. The game is programmed in C# using .NET platform. Furthermore, the entire engine is designed towards sustainability and scalability. That means that by using this engine it is possible to program slightly different game based on the same basics. Especially, it is easy to add new features to the engine. The engine is also prepared for different input formats of levels. Also the rendering layer of the game engine is completely separate. Due to nature of the project the engine should serve as prime example used for education.

Keywords: RPG Dungeon Master Software architecture education

Děkuji svému vedoucímu práce Mgr. Pavlu Ježkovi Ph. D. jak za pomoc s výběrem práce, tak za cenné rady při její tvorbě.

Obsah

1	Úvod	3
2	Analýza	5
2.1	Vstupní data pro herní úrovně	6
2.2	Data s vlastnostmi objektů	10
2.2.1	Akce a kombi	10
2.2.2	Předměty	11
2.2.3	Kouzla a symboly	11
2.2.4	Nepřátelské entity	12
2.3	Parsování vstupních dat	12
2.3.1	Herní úrovně - DUNGEON.DAT	13
2.3.2	Vlastnosti objektů - GRAPHICS.DAT	14
2.4	Použití frameworku pro práci s grafickou kartou	14
2.5	Reprezentace dungeonu jako celku	14
2.6	Reprezentace levelů	14
2.7	Reprezentace dlaždic	16
2.8	Přepínače	16
2.9	Popis funkce a složení	16
2.9.1	Reprezentace přepínač	17
2.10	Builder map	18
2.11	Inicializace a sestavení herních map a objektů	18
2.12	Renderování a interakce	19
2.13	Herní entity	20
2.13.1	Úvod	20
2.13.2	Vlastnosti a schopnosti entit	20
2.13.3	Relace mezi entitami	21
2.13.4	Tělo a inventáře	21
2.14	Předměty	21
2.15	Akce	22
2.16	Kouzla	22
3	Vývojová dokumentace	23
3.1	Jádro enginu	23
3.1.1	Renderovaní	23
3.1.2	Inicializace hráče	24
3.2	Rozšiřitelnost dlaždic	24
3.2.1	Popis dlaždic	24
3.2.2	Inicializace dlaždic	24
3.2.3	Implementace dlaždic	25
3.2.4	Strany dlaždic	26
3.3	Renderery	27
3.4	Přepínače	28
3.4.1	Úvod	28
3.4.2	Implementace přepínačů se senzory	28
3.4.3	Implementace obecných přepínačů	30

3.5	Herní entity	31
3.5.1	Implementace vlastností entit	31
3.5.2	Implementace schopností entit	32
3.5.3	Tělo a inventáře entity	32
3.5.4	Rozmístění entity na dlaždici	33
3.5.5	Relace s dalšími entitami	34
3.5.6	Implementace entit	34
3.6	Předměty	36
3.6.1	Implementace předmětů	36
3.7	Akce	36
3.8	Kouzla	36
3.9	Builder map	37
3.9.1	Vlastní implementace builderu	37
3.9.2	Implementace builderu Dungeon Masteru	37
3.9.3	Rozšíření builderů Dungeon Masteru	38
4	Uživatelská dokumentace	39
4.1	Mechaniky ve hře	39
4.2	Cíl hry	39
4.3	Ovládání	39
4.3.1	Pohyb	39
4.3.2	Aktivace přepínačů a sbírání předmětů	39
4.3.3	Souboj	40
4.3.4	Konzole	40
Závěr	41	
Seznam použité literatury	42	
Přílohy	43	

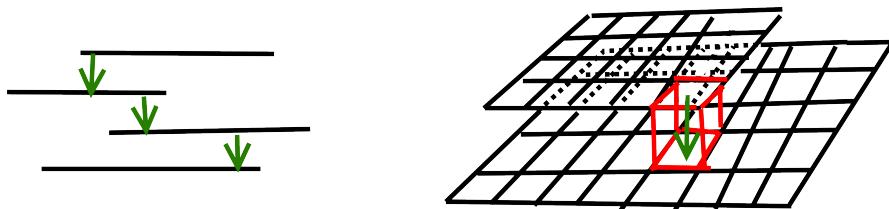
1. Úvod



Obrázek 1.1: Screenshot originální hry Dungeon Master

Dungeon Master je počítačová hra žánru RPG(role playing game) vyvinuta firmou Faster Then Light v roce 1987. Byla to první real-time hra tohoto typu s pseudo 3D pohledem a ovládáním pomocí myši. Hráč má k dispozici skupinu až čtyř hrdinů, s kterými prochází podzemní bludiště a bojuje s nepřáteli(viz obr. 1.1). Tito hrdinové se ve hře nazývají šampioni a mohou se zdokonalovat v různých dovednostech.

Bludiště se skládá z několika úrovní uspořádaných vertikálně pod sebou. Jednotlivé úrovně pak nemusí být stejně velké a mohou být od sebe různě horizontálně odsazené. Každá úroveň je tvořena obdélníkovou plochou s pravidelnou mřížkou(viz obr. 1.2). Pole vymezena mřížkou nazýváme dlaždice a je jich ve hře několik typů, které definují jejich vzhled a funkci. Některé dlaždice lze aktivovat tzv. přepínače. Takové typy dlaždic jsou například dveře nebo jáma(viz obr. 1.1), které lze takto otvírat či zavírat. Přepínače mohou být buď nášlapné na podlaze nebo aktivovatelné pomocí myši na zdech. Mezi úrovněmi lze sestupovat pomocí dlaždic typu schody. Dále je možné se teleportovat mezi dlaždicemi, a to i v různých úrovních, pomocí dlaždice typu teleport.

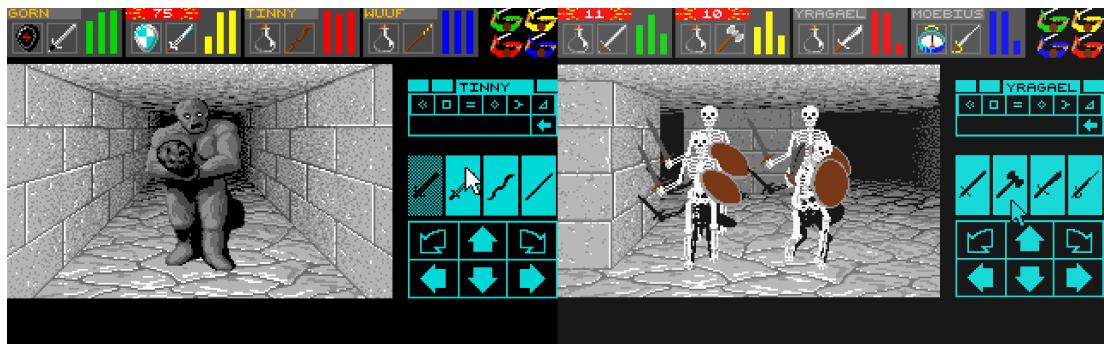


Obrázek 1.2: Ilustrace uspořádání herních úrovní.

Hráč je na začátku postaven do nejvýše položené úrovně, kde si vybere svoji skupinu šampionů. Pohyb skupiny mezi dlaždicemi je zcela diskrétní, to znamená, že se nelze se skupinou zastavit mezi dlaždicemi, ale pohyb je vždy dokončen až na vedlejší dlaždici. Se skupinou je tedy vždy asociována pouze jedna dlaždice. Na dlaždicích pak mohou být různé předměty, které je možné sbírat. Šampioni

mohou s předměty provádět různé akce. Například se zbraněmi lze bojovat nebo lektvary je možné pít a zlepšit si tak dočasně vlastnosti. Kromě těchto akcí může ještě šampion vyvolávat kouzla. Nicméně k tomu potřebuje dostatečnou úroveň odpovídající dovednosti. Tímto způsobem je možné vytvářet lektvary nebo vyvolávat útočná či obraná kouzla.

Ve hře je celá řada nepřátelských entit. Liší se vzhledem, útokem a pohybem. Pohyb některých entit probíhá ještě po hustší mřížce než jsou dlaždice. Každá entita má definovaný prostor dlaždice, který zaujímá. Je to buď prostor celé dlaždice, polovina či čtvrtina. (viz obr. 1.3) Pohyb entit je potom opět diskrétní jako v případě hráčovi skupiny, ale tentokrát mezi definovanými částmi dlaždic.



Obrázek 1.3: Ilustrace prostorů zaujímaných entitami.

Cílem této práce je vytvořit engine pro hru Dungeon Master, tak aby splňovala následující požadavky:

- Engine bude naprogramován v jazyce C#.
- Engine bude schopný sestavit herní úrovně podle vstupních dat použitých v originální hře. Nicméně bude také poskytovat možnost, dodělat si podporu pro jiné formáty.
- Engine bude obsahovat podporu pro funkce a mechaniky vyskytující se ve hře Dungeon Master.
- Engine bude obsahovat oddělenou zobrazovací vrstvu, tak aby mohl být její výstup jednoduše změněn.
- Bude kladen důraz na dobrý objektový návrh, tak aby bylo možné do enginu dodávat jednoduše nové funkce.
- Projekt bude cílený pro vzdělávaní, tak aby si studenti mohli vyzkoušet do enginu doprogramovat další funkce.

2. Analýza

Tato kapitola nejprve blíže popisuje formát a mechaniky originální hry a dále pojednává o postupech a řešeních použitych při implementaci nového enginu. Jsou zde popsány jak slepé a nevhodné návrhy, tak návrhy, které se ukázaly jako nevhodnější řešení.

Pro splnění práce bylo v prvé řadě zapotřebí získat vstupní data originální hry Dungeon Master. Nejdůležitější data jsou obsažená ve dvou souborech: DUNGEON.DAT a GRAPHICS.DAT. První z nich obsahuje definice herních úrovní a seznamy použitých předmětů, přepínačů a nepřátelských entit. Druhý z nich obsahuje konkrétní vlastnosti objektů a entit použitých v prvním souboru. Soubor GRAPHICS.DAT pak také obsahuje definicí akcí, vlastnosti kouzel, textury, atd. Později se ukázalo, že pouze data hry nebudou pro implementaci většiny funkcí dostačující - jedná se například o přesný způsob získávání dovedností šampionů, vyvolávání kouzel nebo provádění akcí. Dalším zdrojem jsou dekomplilované zdrojové kódy[9] originální hry v jazyce K&R C. Tyto zdrojové kódy jsou často náročné na porozumění, nicméně potřebné informace pro dokončení této práce se z nich podařilo získat.

Pro detailnější popis dat Dungeon Masteru si bude dobré nejprve ujasnit vlastnosti a schopnosti šampionů. Každý šampion má následující vlastnosti:

- zdraví - určuje kolik útoku šampion vydrží než zemře,
- výdrž - určuje kolik akcí je schopen šampion vykonat než se unaví,
- mana - reprezentuje magickou energii pro vyvolávání kouzel,
- zatížení - určuje maximální hmotnost, kterou je šampion schopen unést,
- síla - hodnota je používána pro výpočet zranění, síly hodu předmětů a maximální výše zatížení,
- obratnost - zvyšuje pravděpodobnost zásahu nepřítele a pomáhá se vyhnout nepřátelským útokům,
- moudrost - je to vlastnost důležitá pro kouzelníky a kněze, určuje rychlosť obnovy man,
- vitalita - určuje rychlosť obnovy zdraví a výdrže,
- odolnost proti magii - snižuje účinnost magických útoků,
- odolnost proti ohni - snižuje účinnost ohnivých útoků,
- hodnota jídla a vody - pomocí těchto hodnot je obnovována výdrž a zdraví
- štěstí - zvyšuje či snižuje pravděpodobnost provedení akce.

Každý šampion má ve hře čtyři základní dovednosti a to bojovník, ninja, kněz a kouzelník. Kromě základních dovedností má šampion ještě šestnáct skrytých dovedností, které nejsou hráči zobrazeny. Každá z těchto skrytých dovedností náleží

k nějaké ze základních dovedností. Jak moc daný šampion disponuje danou dovedností určuje úroveň dovednosti. Tato úroveň lze navyšovat pomocí zkušeností, které lze získat souboji nebo prováděním akcí. Každá akce pak může navyšovat zkušenosti pro jinou dovednost. Pokud jsou v některé ze skrytých dovedností získány zkušenosti, dostane odpovídající zkušenosti i její základní dovednost. Po získání dostatečného počtu zkušeností v nějaké dovednosti, získá šampion v této dovednosti novou úroveň. Nová úroveň potom navýší určité vlastnosti šampiona. Přesné hodnoty potřebných zkušeností a navyšovaných vlastností pro odpovídající dovednosti jsou pevně stanovené ve zdrojových kódech hry.

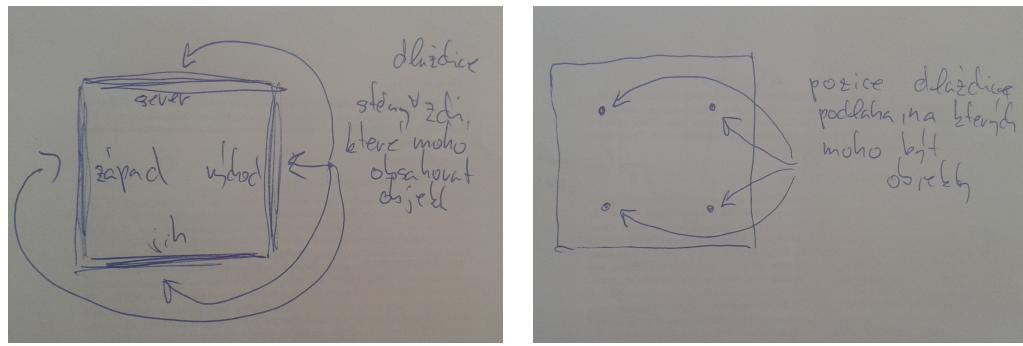
2.1 Vstupní data pro herní úrovně

Originální hra má herní úrovně definované v binárním souboru DUNGEON.DAT. Formát tohoto souboru nebyl tvůrci nikdy zveřejněn, nicméně kolem hry se vytvořila početná komunita, která k němu dokumentaci[6] vytvořila. Tuto dokumentaci jsme použili k porozumění obsahu souboru. Následuje stručný popis zmíněného souboru, pro kompletní dokumentaci navštivte přímo stránky dokumentace.

Soubor DUNGEON.DAT obsahuje jednotlivé herních úrovně a objekty v nich obsažené. Nejprve definuje základní vlastnosti o každé herní úrovni. První z nich jsou rozměry dané úrovně, tj. počet dlaždic na šířku a výšku. Dále definuje obtížnost úrovně, která je použita pro výpočet získaných zkušeností nebo zdraví nepřátelských entit. Každá úroveň má také určené použité podmnožiny dekorací na zdech, dveřích a přepínačích. Dekorace jsou v originální hře identifikovány čísla, která jsou naopak zabudovaná v kódu hry. Některé dekorace tak mohou mít ještě speciální význam, který je identifikován pouze podle zmíněného číselného identifikátoru. Jedná se například o dekorace s výklenky, které mohou být umístěny na zdech. V tomto případě originální herní engine rozpozná, že lze na dané místo vkládat předměty.

Pomocí přepínačů lze dlaždicím zaslat zprávu, na kterou může cílová dlaždice reagovat. Zpráva obsahuje akci a datovou položku. Akce dlaždici říká, zda se má aktivovat či deaktivovat. Co daná datová položka znamená si určuje každý typ cílové dlaždice sám. Originální hra pak pracuje s následujícími typy dlaždic:

- Zed' - zajišťuje zobrazování stěn a nelze na ni vstoupit. Pro každou stěnu(sever, východ, jih, západ - viz obr. 2.1a) dlaždice zdi je určeno, zda může mít tzv. náhodnou dekoraci. Pokud ji může mít, engine podle náhodného generátoru určí jaká to bude (případně žádná). Každá strana může obsahovat přepínač, který si může do zdi uložit předměty. Pokud má některá strana zdi dekoraci výklenku, jsou v něm zobrazeny předměty uložené ve zdi. Na stěnách zdi ještě mohou být popisky, které lze zobrazovat nebo skrývat pomocí zaslání zprávy. Aktivační zpráva popisek zviditelní a deaktivace skryje. Datová položka zprávy zde určuje, pro kterou ze stěn zdi je zpráva určena.
- Podlaha - Po těchto dlaždicích se hráč běžně pohybuje se svou skupinou. Obdobně jako u stran zdí může definovat, zda zobrazuje na podlaze náhodnou dekoraci. Podlaha dále může mít nášlapný přepínač. Dlaždice obsahuje



(a) Dlaždice zed' - stěny.

(b) Dlaždice podlahy - pozice.

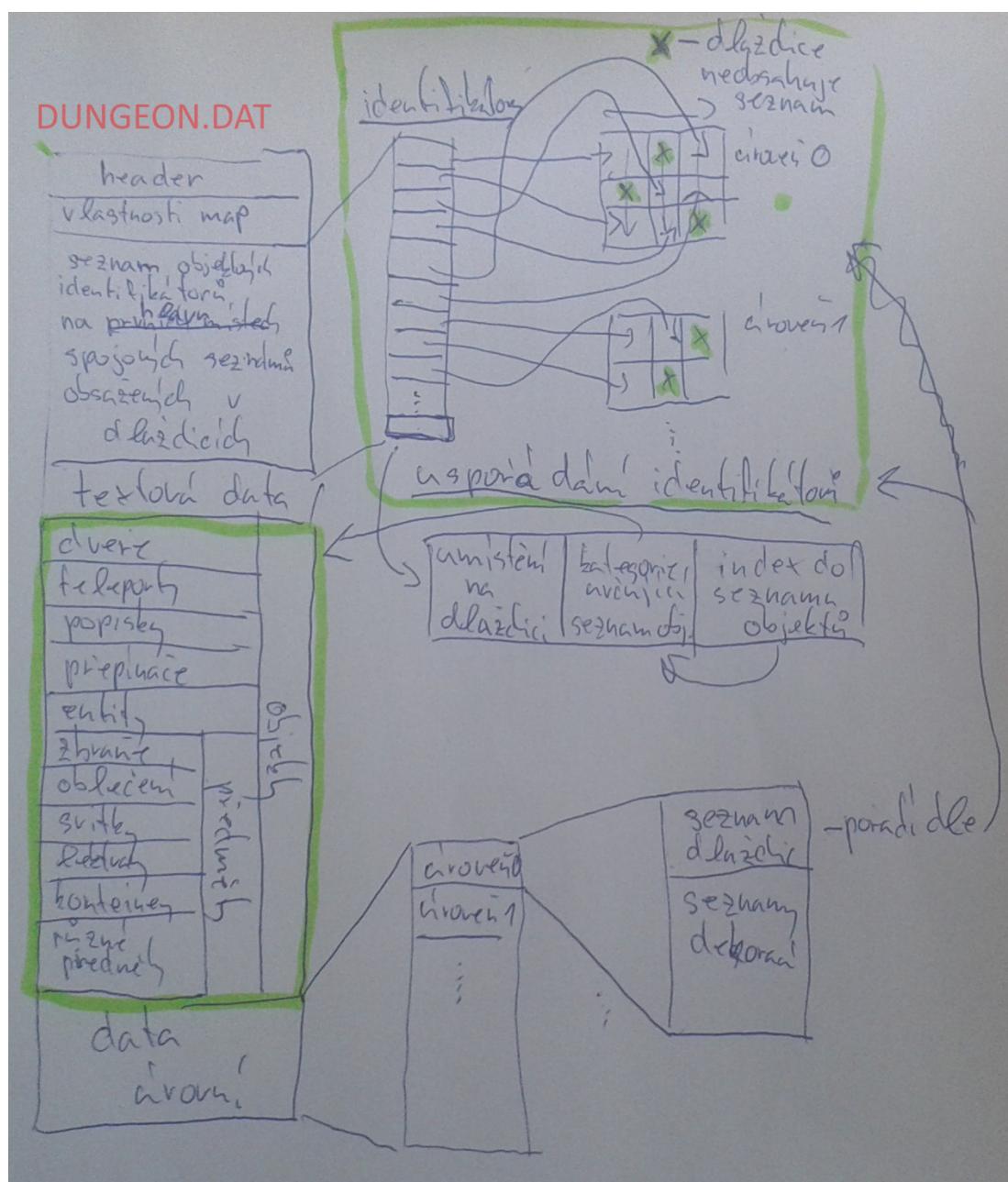
Obrázek 2.1: Možné pozice pro uložení objektů na dlaždici.

čtyři pozice, na které lze pokládat předměty. Tyto pozice vzniknou rozdělením plochy dlaždice na čtvrtiny (viz obr. 2.1b).

- Jáma - Jáma může být buď otevřená nebo zavřená. Nicméně lze na ni vždy vstoupit a pokud je otevřená hráč spadne s svojí skupinou o úroveň níže. Pád může být i přes několik úrovní. Živé objekty pak po dopadu obdrží zranění. Přijmutím aktivační zprávy se jáma otevře a deaktivací se zavře.
- Schody - Po této dlaždici se může hráč dostat o úroveň výše resp. níže.
- Dveře - Na těchto dlaždicích je vždy objekt dveře. Na dlaždici lze vstoupit pouze v případě, že jsou dveře otevřeny. Otevření je možné provést aktivační zprávou či pomocí tlačítka - pokud ho dveře obsahují. Některé dveře lze také rozbít útokem, to si ale definuje již samotný objekt dveří.
- Teleport - Tyto dlaždice obsahují objekt teleport, který určuje, které objekty dokáže přenést. Mohou to být předměty, hráčova skupina nebo nepřátelské entity. Po vstupu resp. vložení objektu na dlaždici, je objekt teleportován, pokud je odpovídajícího typu. Příjem aktivační resp. deaktivacní zprávy aktivuje resp. deaktivuje teleport.
- Iluze zdi - Tato dlaždice může být buď iluze zdi a nebo otevíratelná zed'. V obou případech je vizuálně neodlišitelná od dlaždice typu zdi. V prvním případě je možné na dlaždici vstoupit vkročením do zdi. V druhém případě lze zed' odstranit pomocí zaslání odpovídající zprávy na dlaždici. Aktivační zpráva otevře zed', deaktivací zavře.

V binárním souboru jsou často data uložena po slovech procesoru, které v tomto případě mají velikost dvou bajtů. V další části souboru jsou uložena data s použitými texty ve hře. Tyto texty používají speciální kódování, kdy jednotlivé znaky jsou uloženy do slov (těch procesorových) a každé toto slovo obsahuje tři znaky. Všechny texty jsou pak uloženy za sebou v jednom bloku a ke konkrétním textům je přistupováno přes offsety počtu bajtů od začátku textových dat. Přesný popis může být opět nalezený v dokumentaci [6].

Kromě sbíratelných předmětů obsahují data ještě další objekty. Řadí se sem dveře, teleport, nepřátelské entity, textové popisky a senzory (přepínače). Tyto objekty si lze představit jako instance typů objektů, jejichž vlastnosti jsou definované v souboru GRAPHICS.DAT a vymezují pouze vlastnosti, které mohou



Obrázek 2.2: Ilustrace formátu souboru DUNGEON.DAT

být různé pro každou instanci. Za tímto účelem má každý předmět uložený index typu předmětu v dané kategorii.

Sbíratelné objekty se dělí do následujících kategorií:

- zbraně - lze je vložit šampionům do ruky a pak s nimi mohou bojovat,
- oblečení - lze jím obléknout šampiony, a tak zvýšit jejich odolnost,
- svitky - obsahují text, který může hráči usnadnit hru,
- lektvary - dělí se na lektvary modifikující šampionovi schopnosti a na lektvary provádějící nějaké akce - např.: výbuch. Každá instance má vlastnost určující sílu efektu.
- kontejnery - mohou obsahovat další předměty,
- různé - v této kategorii je například jídlo a pak různé předměty, s kterými nelze provádět žádné akce.

Objekt dveře může být pouze na dlaždici typu dveře a obsahuje informace, zda-li jsou dveře rozbitelné a jestli mají tlačítko, kterým je lze otevřít. Teleporty mohou být pouze na dlaždici typu teleport a definují cílovou dlaždici a kategorii objektů, které teleportují. Textové popisky mohou být pouze na dlaždici typu zeď a obsahují odkaz na konkrétní text. Nepřátelské entity jsou definovány po skupinách a určují typ příšer ve skupině, jejich počet a využití prostoru na dlaždici. Posledními objekty jsou senzory, které vytvářejí základní herní mechaniky bludiště. Senzory mají daný číselný typ, pomocí kterého hra určí jakým způsobem je možné senzor aktivovat. Po aktivaci senzor buď může provést lokální akci nebo odeslat zprávu s akcí na vzdálenou dlaždici. Přepínače se pak typicky skládají z několika senzorů. Více o přepínačích a senzorech bude zmíněno v následujících sekcích.

Ke každému objektu popsanému v předchozích dvou odstavcích existuje unikátní identifikátor (viz obr. 2.2). Tento identifikátor se skládá z pozice, která lze buď interpretovat jako světová strana nebo umístění na dlaždici - jedná-li se o podlahu. Dále obsahuje kategorie, v které je objekt uložen. Pro každou kategorii existuje v datovém souboru seznam instancí objektů dané kategorie. Poslední složkou identifikátoru je index do seznamu v dané kategorii. Tyto identifikátory se dají chápat jako reference na konkrétní instance objektů.

Každá dlaždice specifikuje, zda může obsahovat spojový seznam složený z objektů zmíněných v předchozích odstavcích. Objekty se na sebe potom odkazují pomocí identifikátorů. Další částí datového souboru je pak seznam prvních identifikátoru na dlaždicích. Tento seznam obsahuje první identifikátory pro dlaždice ve všech úrovních. Je seřazen od nejnižší úrovně po nejvyšší od první dlaždice po poslední - bráno v každé úrovni po sloupcích.

Samotné úrovně jsou pak vždy definovány následujícími seznamy v tomto pořadí:

- seznam dlaždic - seřazený po sloupcích dané úrovně,
- seznam dekorací příšer,
- seznam dekorací zdí,

- seznam dekorací podlah.

Počty jednotlivých dekorací a dlaždic jsou popsány ve vlastnostech úrovní.

2.2 Data s vlastnostmi objektů

Soubor GRAPHICS.DAT obsahuje textury dekorací, vlastnosti předmětů a objektů, definici akcí a kouzel. Formát tohoto souboru nebyl tvůrci uveřejněn, avšak komunitě se opět podařilo data vyextrahovat i z tohoto souboru. K některým částem také existuje dokumentace[8], nicméně už není tak přehledná a ucelená jako v případě dokumentace pro soubor DUNGEON.DAT. Zároveň jsou vyextrahovaná data zveřejněna na webu v HTML formátu. Z toho důvodů jsme se rozhodli pro využití již extrahovaných dat v HTML formátu. Dále následuje podrobnější popis využitého obsahu souboru.

2.2.1 Akce a komba

S předměty je možné provádět akce. Množina akcí pro daný předmět se nazývá kombo a obsahuje až tři akce. Ve hře je k dispozici čtyřicet čtyř akcí a stejný počet komb. Kompletní seznam jednotlivých akcí a komb viz[2]. Zde popíšeme alespoň jejich vlastnosti.

Vlastnosti akcí jsou :

- název
- zkušenosti - počet zkušeností získaných po provedení akce,
- dovednost - identifikátor dovednosti, která získá zkušenosti,
- obrana - modifikátor obrany při používání akce,
- výdrž - modifikátor výdrže nutné pro provedení akce,
- pravděpodobnost provedení akce
- zranění - modifikátor pro výpočet konečného zranění nepřítele,
- únava - doba po kterou nelze provádět žádné akce.

Vlastnosti pro každou akci komba jsou:

- index akce ze seznamu akcí,
- úroveň dovednosti - minimální úroveň dovednosti pro úspěšné provedení akce.

2.2.2 Předměty

Ke každému typu předmětu existují v tomto souboru popisovače vlastnosti[3]. Pro každý předmět jsou definovány následující vlastnosti:

- globální identifikátor - unikátní identifikátor, který je použit k identifikaci konkrétního typu předmětu například v přepínačích,
- index útočného kombu,
- lokace - místo, na kterou část těla nebo, do kterého inventáře šampiona lze předmět vložit,
- index v kategorii - index typu předmětu v dané kategorii.

Globální identifikátor lze z instance předmětu získat pomocí jeho kategorie a indexu typu předmětu, který má každá instance předmětu. Každý předmět má také definovanou hmotnost. Zbraně a oblečení mají navíc definované následující vlastnosti[4].

Zbraně:

- poškození - hodnota zranění aplikována při útoku na nepřítele,
- kinetická energie - hodnota udává, jak daleko lze zbraň hodit,
- střelné poškození - hodnota dodatečného poškození u střelných zbraní.

Oblečení:

- síla brnění - obrana, kterou oblečení poskytuje,
- odolnost proti útoky ostrými předměty.

2.2.3 Kouzla a symboly

Další částí dat uložené v souboru GRAPHICS.DAT jsou vlastnosti kouzel. Pojďme si nejprve přiblížit, jak se ve hře kouzla přesně vyvolávají. Každé kouzlo je složeno z několika symbolů, přičemž první z nich je speciální a jde o tzv. „power symbol“. Power symbol určuje, jak bude celkové kouzlo silné. Další symboly už určují konkrétní kouzlo. Každé vyvolání symbolu stojí šampiona manu. Po stanovení všech symbolů může šampion vyvolat samotné kouzlo. Vyvolání kouzla může selhat, pokud nemá šampion dostatečnou úroveň dovednosti vyžadované kouzlem. Datový soubor tedy obsahuje pro každý symbol při odpovídajícím power symbolu množství potřebné many pro jeho vyvolání. Dále obsahuje následující vlastnosti kouzel[5]:

- modifikátor obtížnosti,
- dobu - po které šampion nemůže vyvolávat další kouzla,
- úroveň dovednosti - nutné pro vyvolání kouzla.

2.2.4 Nepřátelské entity

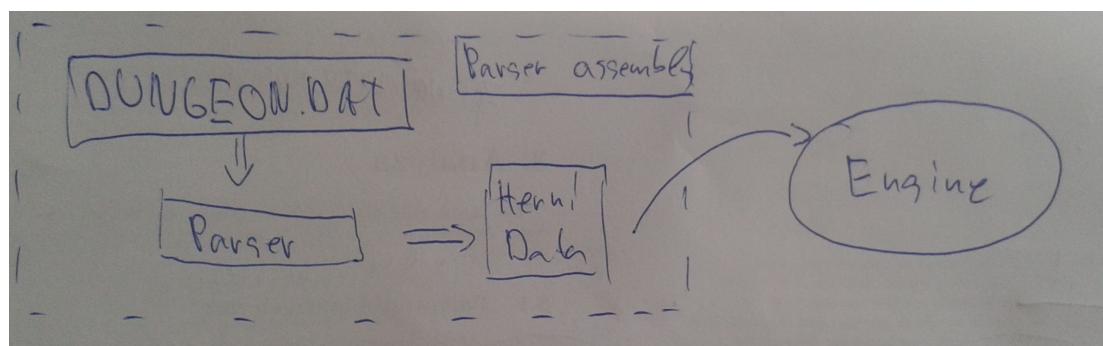
Poslední částí obsaženou v souboru GRAPHICS.DAT, kterou jsme využili, jsou vlastnosti nepřátelských entit. Následující výčet obsahuje vlastnosti, které implementuje nový engine.

- doba mezi pohybem
- doba mezi útoky
- síla brnění
- zdraví
- síla útoku
- otrava - určuje, zda útok může způsobit otravu
- obrana
- dohled
- vzdálenost, na kterou detekuje nepřitele bez vidění
- odolnost proti ohni
- odolnost proti jedu
- pravděpodobnosti zranění určitých částí těla

Celý seznam vlastností je možné nalézt v dokumentaci[7].

2.3 Parsování vstupních dat

V této fázi práce ještě nebylo jasné, zda nalezená dokumentace[6] bude dosatečná pro splnění práce. Parser je proto oddělen od zbytku enginu do zvláštní assembly, která nereferencuje žádnou knihovnu pro zobrazování grafiky. Cílem této assembly je tedy pouze převést struktury obsažené ve vstupních datech do odpovídajících datových tříd v jazyce C#. Tento balík dat se potom předá enginu, který z nich nich vytvoří herní úrovně. Tuto zjednodušenou představu znázorňuje obrázek 2.3.



Obrázek 2.3: Ilustrace vztahu parseru k enginu.

2.3.1 Herní úrovně - DUNGEON.DAT

Parsování herních úrovní je rozděleno do dvou fází. V první fázi probíhá samotné čtení dat objektů(viz 2.1) ze souboru DUNGEON.DAT a transformování je na odpovídající objekty v jazyce C#. V druhé fázi jsou pak objekty z první fáze upraveny, aby měli přehlednější strukturu.

První fáze

V této fázi je nejprve vytvořen datový objekt, který bude shromažďovat všechny ostatní data. Tento objekt obsahuje:

- metadata z hlavičky souboru - to jsou zejména velikosti všech seznamů s objekty(viz 2.1) a počet map,
- seznamy objektů - to jsou seznamy objektů jazyka C# vzniklé rozparsováním odpovídajících dat objektů v souboru,
- seznam herních úrovní - to jsou objekty reprezentující data herních úrovní, které obsahují jejich vlastnosti(viz 2.1) a seznam jejich dlaždic, dekoraci, atd.

Dále parser rozparsuje všechny data vstupního souboru dle dokumentace[6] a vytvoří z nich adekvátní objekty, které uloží do zmíněného datového objektu.

Druhá fáze

Po skončení první fáze je struktura vzniklého datového objektu podobná struktuře souboru DUNGEON.DAT. Tím máme namysli, že například spojové seznamy jsou reprezentovány pomocí objektových identifikátorů(viz 2.1), což není příliš intuitivní a pro práci s těmito daty by engine musel rozumět jejich formátu. Navíc spojové seznamy obsahují objekty různých typů. Proto pro zjednodušení následného čtení těchto dat enginem tato fáze inicializuje v datových objektech pro dlaždice následující položky:

- seznam sbíratelných předmětů,
- seznam senzorů,
- seznam popisků,
- seznam nepřátelských entit,
- položku dveří pro dlaždici typu dveře,
- položku teleport pro dlaždici typu teleport,
- a pro dlaždici typu zeď a podlaha stanoví identifikátory náhodných dekorací.

Na počátku vývoje projekt neobsahoval druhou fázi, její vytvoření vedlo k zjednodušení kódu v enginu.

2.3.2 Vlastnosti objektů - GRAPHICS.DAT

Pro sestavení herních úrovní potřebuje engine také vlastnosti společné pro všechny konkrétní typy objektů ze souboru DUNGEON.DAT. Jak bylo zmíněno v sekci 2.2 tyto vlastnosti jsou v souboru GRAPHICS.DAT. Nicméně data z tohoto souboru již existují vyextrahovaná v HTML formátu. Pro zjednodušení práce s HTML soubory byla použita knihovna HTML Agility Pack[10]. Pro každý typ vlastností je pak vytvořen datový objekt. Tyto seznamy jsou pak uloženy do výsledného datového objektu zmíněného v předchozí sekci 2.3.1. V této části jsou získána data vlastností všech objektů (viz 2.1) až na kouzla. Vlastnosti kouzel jsou rozparsovány v enginu přímo do objektů v něm použitých. Proto by případné vytváření datových souborů v této vrstvě bylo zbytečné. Celý proces zobrazuje diagram na obrázku 2.4.

2.4 Použití frameworku pro práci s grafickou kartou

Celý projekt není nikterak extrémně vázaný na konečně vybranou platformu. Samozřejmě pro použití jiné platformy by bylo třeba provést portaci. Ale jelikož použitý rendering je pouze ve fázi proof of concept, neobsahuje žádné pokročilejší akce s grafickým hardwarem. A je tedy závislý pouze na několika strukturách. Troufám si tedy říct, že případná portace by s dostatkem času nebyl až takový problém.

I z předchozího důvodu jsem se nakonec rozhodl přistoupit k výběru frameworku, s kterým už některé zkušenosti mám. Jedná se tedy o XNA Framework od firmy Microsoft. Nicméně protože tento framework již není firmou nadále podporován a vyvíjen, použil jsem konečně jeho klon MonoGame ([1]). Velkou výhodou je, že framework je multiplatformní. Jeho tvůrci tvrdí, že podporuje platformy iOS, Android, MacOS, Linux, Windows, OUYA, PS4, PSVita, Xbox One.

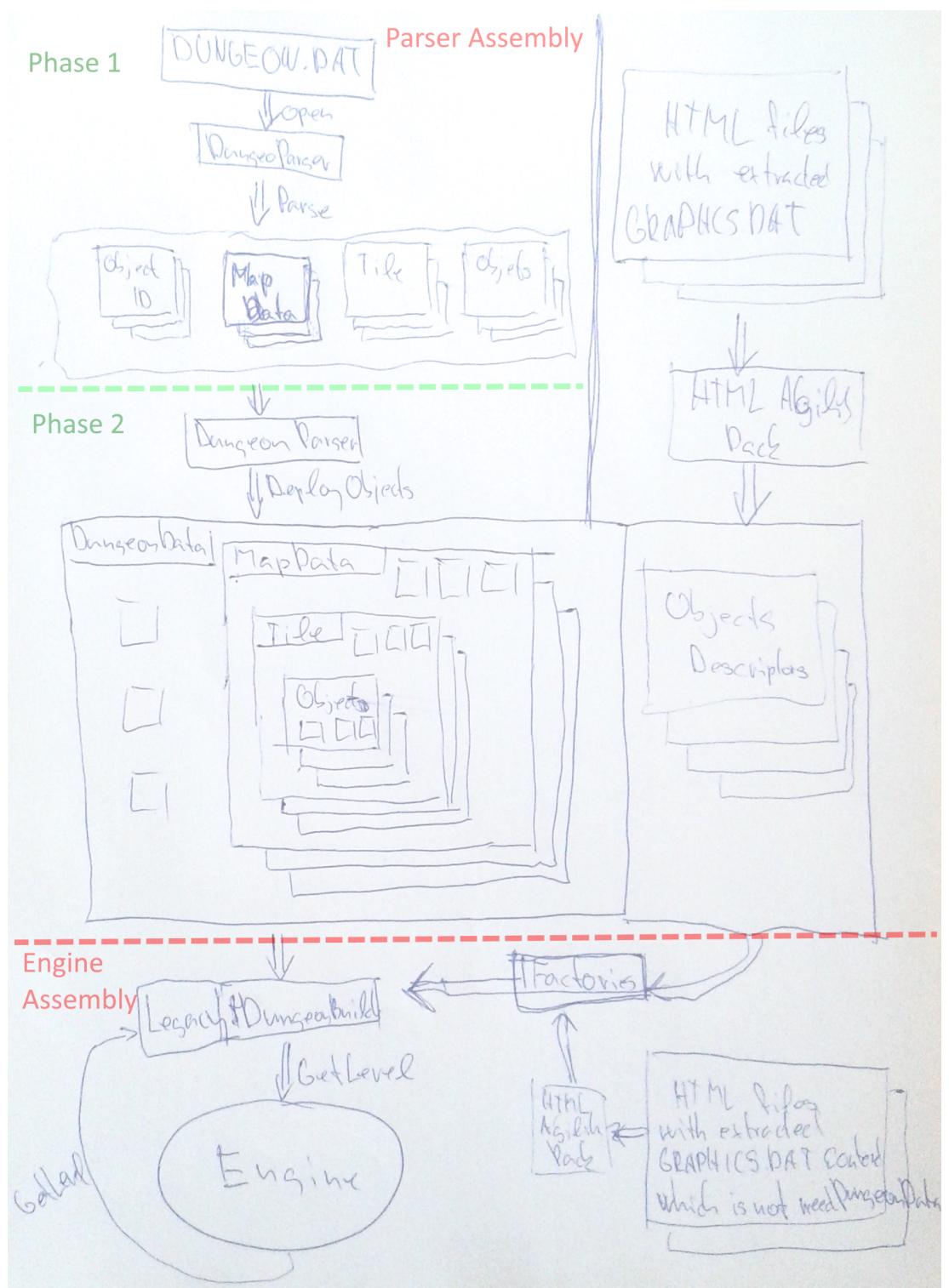
2.5 Reprezentace dungeonu jako celku

Dungeon je logicky rozdělen do jednotlivých levelů, o tyto levele se pak stará samotné jádro enginu. Jádro enginu se ve výsledku stará pouze o načítání levelů a vykreslování správné oblasti dle pozice hráče. Pro načítání levelů se používají buildery, které jsou zároveň bodem rozšiřitelnosti. Naprogramováním různých builderů je dosaženo toho, že engine dokáže načítat mapy v různých formátech.

Při každé změně pozice hráče se vyhledají viditelné dlaždice. Pokud je některá z těchto dlaždic označena jako spojnice levelů, engine propojí tyto dlaždice s odpovídajícími dlaždicemi v cílovém levelu. V originální hře jsou takové dlaždice například schody, teleport nebo jámy. Nicméně je opět na programátorovi, jaké dlaždice tak označí. Viditelné dlaždice jsou potom aktualizované a renderované.

2.6 Reprezentace levelů

Samotné struktury levelů obsahují pouze seznam dlaždic na dané úrovni dungeonu, identifikátor levelu a seznam živých entit. Engine vždy pracuje se třemi



Obrázek 2.4: Průběh parsování herních dat.

posledně načtenými leveley.

2.7 Reprezentace dlaždic

V originální hře jsou dlaždice vždy uloženy v nějakém obdélníkovém poli. To znamená, že dlaždice, které nejsou využity pro chodby, jsou vždy vyplněny zdmi. Za účelem udělat engine co nejdynamičtější, nerozhodl jsem se ukládat dlaždice do obdélníkového pole, ale reprezentovat vazby mezi nimi jako obousměrný orientovaný graf. Jednak pro hodně řídké mapy to může ušetřit nějaké místo. Nicméně důležitějším důvodem bylo, že při takovéto reprezentaci lze jednoduše získat sousedy pouze ze znalosti konkrétní dlaždice. Kromě toho je takto celý engine obecnější. Například by takto bylo jednodušší rozšířit celý engine tak, aby nemusely být jednotlivé dlaždice na mřížce nebo aby mohly mít dlaždice více sousedů. Byla by to sice velký zásah do celého enginu, ale mnoho částí by se takto dalo znova použít nebo jen drobně upravit.

Předchozí rozhodnutí má následující důsledek. Například již nejsou potřeba dlaždice typu zed. Což je dost výrazná změna oproti originální hře, kde dekorace, přepínače, předměty atp. jsou uložené právě v dlaždicích typu zed. Namísto originálního přístupu jsem se rozhodl, že zdi budou definovány v samotných dlaždicích typu podlaha. Toto rozhodnutí vedlo také k některým problémům. Například pro vytváření podlahy bylo nutné čist i další dlaždice kolem té právě vytvářené. Zejména je pak nutné posílat zprávy původně cílené zdí na odpovídající dlaždice podlahy. Ve výsledku si myslím, že tato reprezentace je vhodnější například kvůli možnému rozšíření popsaném v předchozím odstavci. Zdá se být také intuitivnější a tedy srozumitelnější reprezentací.

Zdi jsou tedy součástí dlaždic. Nyní přichází otázka, jak tyto zdi tedy reprezentovat. Buď přímo v samotných dlaždicích nebo zed udělat jako zvláštní objekt, který je součástí dlaždic. Ze začátku byl v enginu použit první zmínovaný způsob. Ukazovalo se ale, že tato reprezentace není příliš vhodná, protože potom samotná dlaždice řeší věci, které k ní přímo nenáleží. Nakonec tato reprezentace byla změněna a v enginu bylo použita druha zmiňovaná možnost. Vedlo k ní zejména předělání oddělené grafické vrstvy od té logické. Výhoda tohoto přístupu byla kromě samotného oddělení kódu i možnost znova použití kódu za pomocí dědičnosti. Například zed s nějakým přepínačem může dědit z normální prázdné zdi. Nicméně ukázala se i nevýhoda tohoto přístupu. A to že komunikace směrem ze zdi k dlaždici je trochu těžkopádná. Buď by bylo zapotřebí předat zdi referenci na jejich rodičovské dlaždice nebo se museli použít události. Pokud to bylo v některých případech třeba, přiklonil jsem se k použití událostí.

2.8 Přepínače

2.9 Popis funkce a složení

Základní mechaniky map dungeon Masteru tvoří tzv. přepínače. Jsou to objekty, které lze nějakým způsobem aktivovat. V originální hře je to možné kliknutím myši na dekoraci přepínače, přesunem na dlaždici s přepínačem nebo aktivací jiným přepínačem. Ve skutečnosti se každý přepínač může skládat z několika

senzorů, kde poslední z nich určuje dekoraci přepínače. Každý senzor přepínače obdrží aktivaci a případně provede svoji akci. Každý sensor může provést lokální nebo vzdálenou akci. Pro lokální akce je to zárotovaní sekvencí senzorů přepínače nebo přidání zkušeností hráči. Pro vzdálené akce je to odeslání zprávy, která se skládá ze samotné akce, tj. deaktivace, aktivace, přehození stavu. A dále obsahuje směr, který může být interpretován jako strana dlaždice typu zdi, která má být aktivována. Nebo pro speciální přepínače může určovat jinou akci. Kromě toho obsahuje další vlastnosti jako typ, zda je opakovatelný, zda je použitelný pouze jednou, jakou má dekoraci, doba, po které se provede akce.

Výše popsaný systém přepínačů přináší hned několik problémů: Jakým způsobem efektivně a přehledně rozparsovat data senzorů. Z technické dokumentace není zcela zřejmé, jak se mají jaké vlastnosti senzorů použít. Jak vůbec reprezentovat přepínače. Jakým způsobem provádět vzdálené akce.

2.9.1 Reprezentace přepínač

První reprezentace

Jako první způsob jsem se rozhodl vždy pro sekvenci senzorů originální hry vytvořit objekt, který měl stejné nebo o něco obecnější vlastnosti dané sekvence. Ze začátku se totiž zdálo, že jen výjimečně jsou sekvence senzorů větší či složitější. Proto jsem se rozhodl vytvořit v novém enginu objekty, které mají obecnější vlastnosti pro několik sekvencí. A určitá sekvence senzorů si poté inicializuje objekt podle svých vlastností. Vlastnosti senzorů jsem určoval podle textového popisu v dokumentaci, která nebyla úplně dostačující. I proto čím více přepínačů jsem naimplementoval, tím více ukazovalo problémů. Nakonec jsem dospěl k závěru, že tento způsob reprezentace je nemožný.

Druha reprezentace

Jiná varianta se stále spoléhala na typ objektů z předchozího případu. Parsování sekvence senzorů jsem se rozhodl udělat pomocí konečného automatu. Tedy tak, že pro každý objekt existovaly předdefinované sekvence senzorů. A tak se pro každou sekvenci senzorů našel správně reprezentující objekt. Později se ale ukázalo, že i malá změna v sekvenci senzorů může generovat podobný objekt. Takže by bylo třeba spoustu předdefinovaných šablon a jednoduše by mohl nastat případ, že pro některou sekvenci neexistuje žádný objekt.

Třetí reprezentace

Nezbylo teď jiné možnosti než se přiblížit více k implementaci podobné originální hře. Za tímto účelem jsem se začal poohlížet po po zdrojových kódech originálu. Originální zdrojový kód pravděpodobně nebyl a nebude nikdy publikovaný. Naštěstí se ale našel člověk, který celou hru dekompileval, takže po zkompilování tehdejším komplilátorem výsledná binárka odpovídala té originální (viz.[?]). Kód to je sice těžko čitelný, ale obsahuje přesný popis, jak které senzory mají fungovat. Vytvořil jsem tedy objekt přepínač, který má v sobě sekvenci senzorů, tak jako tomu je v originální hře. S využitím zdrojových kód jsem vytvořil odpovídající objektově orientovaný kód v jazyce c#. Je tedy například možné dovytvořit další

nové senzory. V bylo toto rozšiřovaní jen značně omezené, jednotlivé type senzorů se odlišovali číselným identifikátorem. Jelikož jsem ale nechtěl případné rozšířiteli enginu limitovat tímto způsobem tvorby herních mechanik. Je samozřejmě možný, vytvořit jiný přepínač, který se bude chovat tak, jak si programátor zvolí. Takže případné nové přepínače nemusí vůbec senzory používat. To jak funguje senzor uvnitř je tedy čistě na programátorovi.

Reprezentace zprávy přepínače

Jak již jsem psal, celý systém přepínačů je závislý na posílání zpráv. V originální hře je cíl odeslané zprávy vázán na souřadnice dlaždice, na kterou se má zpráva odeslat. Jelikož jsem se v enginu nechtěl na plno vázat na tyto pevné souřadnice, rozhodl jsem se to i zde udělat trošku jinak. Například jsem zmiňoval, že je principiálně možné, aby dlaždice měla více sousedů, u kterých by pak souřadnice nešli přesně nastavit. Z tohoto důvodu jsem se rozhodl přepínačům při jejich inicializaci předat referenci na jejich cílovou dlaždici. To řeší problémy, které jsem popsal v předchozích větách, nicméně přináší pár dalších (viz. inicializace). Jak jsem již zmiňoval, v originální hře jsou zprávy opět reprezentovány pevnou datovou strukturou. V této části jsem chtěl dát také programátorovi větší svobodu. A to tak, aby bylo možné posílat vlastní zprávy alespoň vlastně vytvořeným dlaždicím. Toho je nakonec možné dosáhnout tak, že vlastní dlaždice bude podělena z generické dlaždice, která bere jako typový parameter typ dané zprávy. Zpráva pak musí alespoň dědit ze základní zprávy, která odpovídá zprávě v původní hře.

2.10 Builder map

Builder map je objekt, který je schopen dodávat levele samotnému enginu. A stojí na něm, aby vytvořil potřebné mapy z dostupných dat. Vytvoření builderu je samozřejmě opět na programátorovi. Builder by se měl také start o případně kešování, již načtených levelů. Tato práce obsahuje pouze builder schopny rozparsovat mapy originální hry.

Jelikož jsem si původně myslел, že některé zde nebude vůbec nutné parsovovat, rozhodl jsem se v prvotních fázích procházet data map pomocí algoritmu depth first search. Nakonec se ale ukázalo, že i některé nepřístupně části map jsou používané. Například pomocí teleportu, nebo vazbami mezi přepínači. Proto jsem později zvolil techniku, že projdu všechny dlaždice od shora dolů. A podle nich vytvořím odpovídající objekty v novém enginu.

2.11 Inicializace a sestavení herních map a objektů

V celém projektu jsem se snažil držet paradigmatu defenzivní programování. Tedy jde o to nechávat veřejně pouze takové položky, u kterých je to nezbytně nutné. Na to navazuje problém s inicializací takových tříd. jelikož například přepínače potřebují referenci na cílovou dlaždici, ale zároveň jsou obsahem dlaždice, je třeba oddělit fázi inicializace dlaždic od inicializace přepínačů. Potom ale nejde

přepínače předat do dlaždic, aniž by to narušovalo cíl držet se defenzivního programování. Kromě toho mají konstruktory takových tříd mnoho parametrů. Nejprve jsem inicializaci řešil právě těmito konstruktory, ale z předchozích důvodu jsem přišel s tzv. inicializátory.

Inicializátor je datová třída obsahující vlastnosti, které by normálně, byly parametry konstruktoru inicializovaného objektu. Místo těchto parametrů je do konstruktoru předán inicializátor. Inicializátor má také události vyvolané při inicializaci a při dokončení inicializace. Třída beroucí inicializátor jako parametr konstruktoru se zaregistrouje na tyto události. Tím pádem není nutná žádná přebytečná položka ve třídě pro inicializátor. Události jsou pak volané skrze metodu v inicializátoru. Inicializátorem inicializovaná třída si z něj nakopíruje parametry a odhlásí událost. Od té chvíle už není možné třídu modifikovat. Data inicializátoru se tedy mohou postupně naplňovat a po jejich naplnění se inicializace provede zavoláním jejich metody. S využitím a asynchronních metod, je navíc možné vyvolat inicializaci prvku (např. přepínačů, které potřebují reference na dlaždice) již při vytváření dlaždic. Což vede k přehlednějšímu kódovi a celkově k inicializaci cyklických struktur, bez porušení našeho požadavku defenzivního programování.

S využitím dědičnosti inicializátorů je možné nasimulovat volání rutin konstruktorů, tak jak je v c# běžné. Na každé úrovní dědičnosti se využijí některé vlastnosti inicializátoru. Nechť inicializátor B je potomek inicializátoru A v hierarchii dědičnosti. A nechť C a D jsou třídy, které chceme inicializovat a zároveň D je potomkem C. A také platí, že konstruktor třídy D bere inicializátor třídy B a konstruktor třídy C bere inicializátor A. Potom inicializátor B můžeme použít pro oba konstruktory tříd C i D. Přičemž na každé úrovni dědičnosti inicializátoru může být zvláštní inicializaci oznamující událost. Pokud tedy inicializátor volá inicializační události v e správném pořadí, může to nasimulovat pořadí inicializace běžné u konstruktorů. Při inicializaci dlaždic je právě tento způsob používán.

2.12 Renderování a interakce

Interakcí je myšleno reakce kliku myší na herní objekt. Z počátku si renderovaní řešil každý objekt sám v sobě. Pozicování těchto objektů bylo z pravidla absolutní. Naopak interakce byla zase řešena pro všechny objekty stejně pomocí bounding boxu. Obojí se později ukázalo jako špatný přístup.

Namísto toho vznikla nová vrstva, oddělená od samotných objektů zajišťující jak renderování tak interakci. Každý objekt který to vyžaduje má tak vlastní renderer-interactor na míru. Jak takový objekt vypadá uvnitř je zpravidla na programátorovi. Nicméně obvyklý přístup je takový, že renderer má referenci na konkrétní renderovaný objekt. Podle jeho zpravidla readonly vlastnosti potom určuje chování vykreslování nebo interakce. Pokud se jedná o renderery pro statické objekty (např. dlaždice), tak se zpravidla pozicování určuje relativně vůči rodiči. Také interakční či renderovací funkce dostane jako parametr dosavadní transformační matici.

Celá výhoda tkví v tom, že za takovýchto podmínek je možné renderovací-interakční vrstvu libovolně přepracovat. Oproti originální hře, je právě v tomto projektu odlišně provedené renderování. Nicméně nic nebraní tomu napsati s vlastní vrstvu, tak aby vpadala jako originál.

2.13 Herní entity

2.13.1 Úvod

V originální hře existují pouze dva typy entit. Jsou to nepřátelské příšery a hráčovi šampioni. Každé tyto entity se mohou lišit v závislosti na jejich vlastnostech. Tedy ve hře existuje několik druhů šampionů a několik druhů příšer. V originální hře jsou tyto dva světy naprostoto oddělené. Já jsem se nicméně rozhodl udělat celou situaci o něco obecnější.

Entitou v tomto enginu může být téměř cokoliv, s čím je potřeba nějak interagovat ve smyslu bojů. Tedy entita je cokoliv co má nějaké vlastnosti, které jsou ovlivnitelné například útokem. Entity si dále definují tzv. uspořádaní na dlaždici. Takže například dveře jsou entita, která má vlastnosti jako zdraví, odolnost atp. Tento příklad je ukázkovou věcí, která je v originální hře řešena staticky. S využitím entit v těchto případech je možné dosáhnout větší obecnosti a tím pádem je se možné vyhnout zbytečně specializovanému a podobnému kódu.

Kromě entit jsou v enginu ještě tzv. živé entity. Což jsou entity s tím rozdílem, že kromě vlastností mají také schopnosti. Dále mají definovanou pozici dle uspořádaní, relaci vůči ostatním živým entitám(zda jsou přátelští či nepřátelští) a konečně části těla a inventáře.

Z hlediska enginu jsou teď oproti originální hře šampioni i příšery na rovnomenné úrovni.

2.13.2 Vlastnosti a schopnosti entit

Cílem nového enginu je opět poskytnout možnost rozšíření vlastností a schopností. Za následujících podmínek je teď možné na tomto enginu postavit hru, která si sama definuje, jaké vlastnosti a schopnosti, které entity budou mít. Dále je možné nadefinovat akce, které tyto vlastnosti a schopnosti budou využívat. Tim je dosaženo opět další dynamičnosti enginu.

Vlastnosti

V původní hře jsou vlastnosti opět pevně definované. Jsou to například vlastnosti jako zdraví, mana, výdrž, síla atp. Šampioni a příšery mají většinu vlastností odlišných, tím pádem i způsob jejich použití je odlišný. Jak jsme si ale řekli v předchozí sekci, tak příšery i šampioni v novém enginu jsou nyní na stejně úrovni. Zároveň některé entity některé vlastnosti mít nemusí a tím pádem jsou u nich tyto vlastnosti v základní hodnotě. Z toho důvodu jsem se rozhodl nechat seznam vlastnosti na každé entitě zvlášť zcela dynamicky. Pokud daná entita některou vlastnost nemá, vrátí svoji základní hodnotu. Potom při konání některé akce ovlivňující vlastnosti dané entity se akce sama rozhodne, které vlastnosti nějakým způsobem použije. Například pokud provádíme akci útok magickou ohnivou koulí a entita nemá žádné vlastnosti odolnost proti magii či odolnost proti ohni. Jsou tyto vlastnosti v základním stavu, čili nijak neovlivně damage způsobené touto akcí.

Schopnosti

Schopnosti jsou stejně jako vlastnosti v originální hře pevně definovány. I v tomto bodě se nový engine snaží o větší dynamičnost. Opět nevšechny entity musí mít všechny schopnosti. Pokud určitou schopnost entita nemá, vrátí se její hodnota s leolem nula. Podle schopností se určuje, zda je entita danou akci schopna provést. Tedy každá akce má definovaný minimální levely pro sadu schopností. Pokud pak levely vyhovují, může být daná akce provedena.

V originální hře jsou dva typy schopností. Jsou to základní schopnosti a skryté schopnosti. Pokud některá akce vylepšuje schopnost skrytou rozdělí se získané zkušenosti mezi obě schopnosti. Tento koncept schopností není v enginu vyžadován. Je možné si udělat jednoúrovňové schopnosti nebo klidně několika úrovňové schopnosti. Vše záleží na tom jaké chování programátor daným schopnostem naprogramuje.

2.13.3 Relace mezi entitami

Další funkcí kterou oproti originálu engine umožňuje je definovat pro entity jejich nepřátelé. Každá živá entita má token identifikující skupinu. Entity v této skupině jsou mezi sebou přátelské. Každá entita si může nadefinovat svoje nepřátelské tokeny. Podle vzoru originálu jsou v této konkrétní hře pouze dva relační tokeny. Jeden pro šampiony a druhý pro příšery. Nicméně celý tento systém je navržen právě pro stanovení obecnějších vztahů mezi entitami. To může být aplikováno v libovolném rozšíření hry.

2.13.4 Tělo a inventáře

Každá živá entita má definované její části těla. Oproti originálu, kde se řeší pouze části těla šampionů, zde je možné definovat tělo pro každou entitu. Je tak možné pracovat s částmi těla entit, které nemají humanoidní formu. Některé části těla se dají použít jako úložiště věcí. Předměty potom mají nadefinováno s jakým typem úložiště jsou kompatibilní. Tato reprezentace dává dobrý smysl. Například pokud máme lidskou část těla - nohy. A medvědí část těla - nohy. Tak lidské kalhoty by měly jít nasadit pouze na lidské nohy, nikoliv medvědí. Naopak medvědí chrániče na nohy půjdou dát pouze na nohy medvědí. Takto pokročilejší věci v originální hře nejsou použity, předměty tam mohou sbírat pouze šampioni. Nicméně pro případné rozšířitele je principiálně možné navrhnut hru, v které se tyto techniky budou používat.

Kromě částí těla jsou zde ještě definovány další externí úložiště, které slouží jako hlavní úložný prostor. O velikosti a typu těchto úložišť opět rozhoduje tvůrce konkrétních entit. Nicméně stejně tak jako v originální hře, i tato instance má implementované úložiště pouze pro šampiony. Mezi taktové úložiště patří například batoh, kapsa, bedna, toulec atp.

2.14 Předměty

Tato sekce pojednává o předmětech, které se objevují v mapách hry Dungeon Master. Myšleno předměty, které jdou sebrat ze země, vložit do ruky hráče či

šampiona, vložit do inventáře či úložiště kdekoliv v dungeonu(např. do výklenků). Takovéto předměty mohou zejména definovat akce, které s nimi lze provádět. Předměty v originální hře se dělí do několika skupin tj. zbraně, zbroje, lektvary, truhly a další. Každý předmět může mít několik akcí a je na uživateli enginu, jakou akci danému předmětu přiřadí.

V originální hře má každý předmět unikátní číselný identifikátor, který používá pro jednoznačnou identifikaci typu předmětu. Nikoliv typu jako typ proměnné, ale jako například jeden konkrétní typ zbraně. Tyto identifikátory jsou používány především u různých druhů klíčů, ale není to podmínkou. Identifikátory jsou obecně používané sensory, například tak, že konkrétní senzor se aktivuje pouze pokud má hráč v ruce předmět s určitým identifikátorem. Není tedy pravda, že každá instance, například klíče, má rozdílný identifikátor. Naopak, všechny instance klíče stejného druhu, mají stejný identifikátor.

Z počátku jsem pro stanovení identifikátoru zvolil stejnou strategii jako tvůrci originální hry, ale celou dobu se mi to zdálo jako dosti dřevní řešení. Tento problém nakonec vyřešil problém jiný a to jakým způsobem přesně reprezentovat předměty. Tak aby neměli zbytečně položky uložené několikrát a aby se dali snadno vytvářet. Ze začátku měli všechny předměty uložené všechny jejich vlastnosti u sebe. Bylo to tak i s položkami, které byly vždy pro instanci jednoho typu stejné. Tak jsem se tyto položky rozhodl delegovat do zvláštních tříd, odpovídající jednotlivým identifikátorům. Reference těchto tříd tedy není slouží jako identifikátory. Samotné instance předmětů potom obsahují právě reference na tyto třídy.

Navíc třídy popsané v předchozím odstavci slouží jako továrny na předměty jejich typu jakožto identifikátoru. Se znalostmi továrny lze buď vytvořit předmět se základními hodnotami jejich vlastností. A nebo se znalostmi konkrétního typu(myšleno jazykového) továrny, lze předáním inicializátoru vytvořit nový předmět s hodnotami dle inicializátoru. Každá továrna má také následující vlastnosti: hmotnost, jméno, akce, které lze s předměty provádět, a definice míst v inventáři , kam lze předmět uložit. Specializované typy, myšleno jako zbraně, lektvary, atp. pak mají některé další vlastnosti.

2.15 Akce

TODO

2.16 Kouzla

TODO

3. Vývojová dokumentace

Následující sekce této kapitoly jsou určeny především pro lidi, kteří chtějí porozumět více celému enginu. Především však dobře poslouží lidem, kteří by tento engine chtěli použít pro svoji hru nebo by chtěli jen rozšířit některé části tohoto enginu Dungeon Masteru.

3.1 Jádro enginu

Jádro enginu tvoří třída `DungeonBase`, jak již bylo řečeno, stará se zejména renderování, aktualizování herních objektů, inicializaci hráče, načítání a propojování herních map. Pokud má čtenář zájem o úpravu některých z těchto věcí, je tu správně.

3.1.1 Renderování

Tato sekce slouží pro čtenáře, kteří mají zájem o reimplementaci následujících věcí:

- Způsob výběru dlaždic, které se mají používat pro aktualizaci a rendering.
- Pořadí v jakém se jednotlivé dlaždice renderují.
- Použití osvětlení a celkově nastavení grafického zařízení.
- Dosah viditelnosti
- atp.

Výběr a vykreslování použitých dlaždic

V základní verzi vždy existuje kolekce právě používaných dlaždic. Tato kolekce se znova naplní vždy, když hráč změní pozici. dlaždice se zde vyhledávají algoritmem breath first search.

Vlastní algoritmus výběru dlaždic je možný přidat poděděním třídy `DungeonBase` a overridnutím metody `UpdateVisibleTiles`. Metoda uloží do proměnné `currentVisibleTiles` vybraný seznam dlaždic. Tato proměnná je pak využívána v metodě `Draw`, kde se ale renderují v opačném pořadí kvůli průhlednosti.

Nastavení grafického zařízení

Nastavení osvětlení a celkově efektu, textury a batcheru pro vykreslovaní minimapy, se provádí v metodě `InitializeGraphics`. Jejím overridnutím je možné provést modifikace. Pokud nechcete upravovat všechny popsané věci, je doporučené zavolat nejprve funkci rodiče která data zinicializuje na základní hodnotu. Samotná minimapa se vykresluje funkcí `DrawMiniMap`, která se volá klasicky ve funkci `Draw`. Overridnutím této funkce je tedy možné například minimapu úplně odstranit, tak jako to je v originální hře. V této sekci je ještě možná dobré zmínit

proměnou **FogHorizont**, která je v základu používaná jednak jako maximální vzdálenost, po kterou jsou dlaždice hledány, a za druhé slouží v efektu jako hodnota FogHorizontu.

3.1.2 Inicializace hráče

O abstrakci různých způsobů načítání levelů se stará interface **IDungeonBuilder**. Instanci implementace tohoto rozhraní je nutné předa Dungeonu v konstruktoru. V základu se nové mapy načítají v momentu, kdy některá ze zobrazených dlaždic je dlaždice navazující na další level. Takové dlaždice jsou právě dlaždice implementující rozhraní **ILevelConnector**. Poslední tři levele získané skrze **IDungeonBuilder** jsou uložené v kolekci **ActiveLevels**. Pokud je nutné změnit strategii načítání levelů, je třeba overridovat metody **SetupLevelConnectors** popř. **ConnectLevels**. Pro změnu strategie ukládání a mazání aktivních levelů, je třeba podělit třídu **LevelCollection**. Kterou je pak nutno nastavit do vlastnosti **ActiveLevels**.

Následující sekce popisují jaké objekty lze jakým způsobem rozšířit či upravit. O tom jak takové nové objekty potom používat pojednává pozdější sekce.

3.2 Rozšiřitelnost dlaždic

3.2.1 Popis dlaždic

Nejobecnější strukturu dlaždice definuje interface **ITile**. Dlaždice tedy musí obsahovat pozici vzhledem k mřížce dlaždic a potom level, ve kterém se nachází. Tyto vlastnosti používají příšery nebo hráč k určení své pozice. Dále obsahuje vlastnosti sloužící k rozhodovaní pohybu entit, předmětů a orientaci příšer. Další částí je již zmiňovaný **LayoutManager**, který slouží k rozdělení prostoru mezi entitami na dlaždici. Podle něj se příšera či hráč rozhoduje, zda může na danou dlaždici vstoupit a obsadit, tak část jejího prostoru. Má také definované soudy, podle kterých se zase entity rozhodují při pohybu mezi dlaždicemi. Další část API slouží k modifikaci stavu dlaždice. Stav jde buď přímo nastavit pomocí volání odpovídajících funkcí nebo předáním zprávy. Poslední část obsahuje metody a události pro vstup a odchod včí z dlaždice. O volání těchto funkcí se musí statrat samy objekty, které chtějí vstoupit či odejít. Takové objekty jsou poté skrze dlaždici aktualizovány, pokud implementují rozhraní **IUpdateable**. To však již musí zařizovat jednotlivé implementace dlaždic, o niž bude řeč v další sekci.

3.2.2 Inicializace dlaždic

Jak již bylo zmíněno v analýze, k inicializaci dlaždic jsou použity tzv. inicializátory. Inicializátor obsahuje vlastnosti, které by normálně byly předány jako parametry v konstruktoru. Ovšem v moment předávaní inicializátoru do konstruktoru, inicializátor ještě nemusí být plný. Namísto toho má události **Initializing** resp. **Initialized**, které jsou vyvolány při resp. po inicializaci. Na tuto událost je třeba zaregistrovat inicializační funkci, která zkopíruje data z inicializátoru do samotného objektu. Pro každou úroveň hierarchie dědičnosti jsou určeny zvláštní vlastnosti a zvláštní inicializační události. Rodičovské události inicializátoru jsou

vždy po zdědění rodičovského inicializátory zakryty novými inicializačními událostmi pro danou proveď dědičnosti. Tento způsob je použit pouze pro inicializaci dlaždic. Pro jiné objekty, může inicializátor sloužit pouze jako objekt udržující parametry. Parametry, které jsou hned v konstruktoru inicializované. To vše záleží na konvice kterou si programátor zvolí.

3.2.3 Implementace dlaždic

O částečnou implementaci rozhraní **ITile** se stará třída **Tile**. Definuje základní layout manager, nicméně je ho popřípadě možné overridovat vlastní implementaci. Zejména se pak ale stará o inicializaci pozic na mřížce, levelu a sousedů skrze **TileInicializator**. Dále abstraktně deklaruje vlastnost **SubItems**, na jejichž objektech, které implementují **IUpdateable** volá metodu **Update** skrze metodu **Update** na dlaždici. Abstraktně deklaruje také vlastnosti **Sides**, kde těmto stranám přeposílá případné zprávy, které přišly na tuto dlaždici skrze metodu **AcceptMessageBase**. Dále implementuje také funkce pro vstup/odchod objektů tím, že vyvolá odpovídající události na dlaždici. Pokud tedy chcete volání těchto událostí zachovat, je při případném overridování metod nutné zavolat implementaci rodiče. Všechny předchozí popsané funkce je možné overridovat v potomkově a tak přizpůsobit prováděně akce.

Přímý generický potomek **Tile<TMesssage>** poskytuje již zmiňovanou možnost přijímat v potomcích vlastní zprávy. Poděděním tohoto typu, specifikovaným typu zprávy v typovém parametru a následnou implementací metody **AcceptMessage** lze definovat rutinu při příjmu zprávy. Tato třída overriduje a zároveň uzavírá metodu rodič **AcceptMessageBase**, která deleguje zprávy typu **TMesssage** do metody **AcceptMessage**. Naopak základní implementace metody **AcceptMessage** je zavolání právě rodičovské implementace **AcceptMessageBase**. Takže při případném overridování této funkce stojí za zamýšlení, zda-li je třeba volat implementaci rodiče či nikoliv. Všechny dlaždice, které dědí ze třídy popsané v tomto odstavci vytváří potomky dva. Jednoho pro případně rozšířitele, která ponechává typový parametr. A druhá, která definuje typový parametr na obecný typ **Message**. Ještě stojí z zmínku, že každá zpráva musí dědit právě ze třídy **Message**.

Dlaždice podlahy

Za zmínku ještě stojí přímý potomek třídy **FloorTile<TMesssage>** a to **FloorTile<TMesssage>**. Tato třída se stará jak o interakce, tak rendering zdi a podlahy. Stará se tady například o možnost pokládaní předmětů na zem, či u zdí do případných výklenků. Dále se stará o zobrazování a aktivování případných přepínačů. Z toho důvodu, pokud se chystáte vytvářet nějakou novou dlaždici, je dobré se zamyslet, jestli nevyužít již tuto implementaci. Nicméně nic nebrání tomu podělit přímo z **Tile<TMesssage>**, pokud by tato implementace z nějakého důvodu nevyhovovala. Při drobných úpravách je zase možné třídě podstrčit pozměněné-poděděné implementace jejich stran, ať už zdí nebo podlahy. Třída deleguje veškeré akce vstupu/odchodu do samotné podlahy typu **FloorSide**. Stejně tak implementace kolekce **SubItems** se deleguje na podlahu.

Další dlaždice

Další dlaždice již převážně využívají dědičností právě již zmíněnou předchozí podlahu. Výjimkou jsou například schody. Schody jsou příkladem dlaždice, který implementuje rozhraní **ILevelConnector**, které požaduje implementaci následujících vlastností

- cílový level
- pozice cílové dlaždice v cílovém levelu

Jak už bylo zmíněno engine potom při načtení levelu nastaví poslední položku tohoto interface **NextLevelEnter** na odpovídající dlaždici. A je už na samotné dlaždici, aby při změně této vlastnosti udělala potřebné akce. U schodů je to například nastavení sousedů vedoucích do dalšího levelu. Právě zde se například hodí overridování vlastnosti **Neighbors** vlastní třídou starající se o sousedy. Tato možnost je použita ještě u jámy, kvůli propadu do nižšího levelu. Rozhraní spojující leveley je pak ještě u teleportu, kde při vstupu na teleportační dlaždici a splnění požadavku teleportu se tento objekt teleportuje na jinou dlaždici. Za zmínku ještě stojí existence rozhraní **IHasEntity**, které obsahuje vlastnost typu entity. Implementuje ho například dlaždice typu dveře, která vrací jako entity dveře. Tímto způsobem lze pak například útokem dveře rozbít, protože jsou definované jako entita s vlastností zdraví, odolnost atp. Nikde jinde v enginu toto využito není, tím více prostoru mají případní rozšířit.

3.2.4 Strany dlaždic

Jak již bylo naznačeno v předchozích sekčích, dlaždice mohou mít strany. To mohou být ať už zdi, tak i podlaha nebo strop. Každou tuto stranu reprezentuje samostatný objekt, který má samostatný renderer-interactor. Poděděním z těchto hotových tříd si lze tedy ušetřit nějakou práci. Není to však nutnost, je možné si pro své dlaždice implementovat vlastní strany implementací rozhraní **ITileSide**. Toto rozhraní vyžaduje pouze položku pro renderer a z kompatibilních důvodů pro starou verzi požaduje, aby byla schopna přijímat zprávy **Message**.

Jednotlivé implementace

Třída **TileSide** slouží jako rodič všech zdí implementovaných ve hře. Ne-reaguje nijak na interakci hráče a její renderer pouze renderuje zdi texturou na správné místo, popř. dekoraci.

Jejím přímým potomkem je třída **ActuatorTileRenderer**, ta navíc obsahuje přepínač a její renderer se navíc stará o jeho vykreslování a interakci.

Dalším potomkem je třída **TextTileSide**, jejíž renderer navíc zobrazuje na zdi text, v závislosti na tom, zda-li je viditelný. Viditelnost textu lze změnit zasláním zprávy s odpovídajícími informacemi této straně.

Předposlední a nejsložitější stranou je třída **FloorTileSide**. Tato strana obsahuje čtyři úložiště na předměty, kam je může uživatel pokládat. Je to jedno z míst, kde je potřeba komunikovat s rodičovskou dlaždicí, k tomu slouží událost **SubItemsChanged**, která se vyvolá vždy, když byl nějaký předmět odebrán nebo přidán. Pomocí enumerátoru tohoto objektu je potom možné vyenumerovat obsažené věci. Předměty lze pak na podlahu přidávat dvěma způsoby, jednak přímo

z ruky hráče pomocí renderer-interactoru do úložného prostoru na podlaze. Toto přidání opět pomocí události informuje přes podlahu rodičovskou dlaždici. Další způsob je zavolání metody **OnObjectEnter** resp. **OnObjectLeft**, které také vyvolají notifikační událost o změně a přidají objekt do nějakého prostoru. Tento způsob je použit například u teleportu, kde se vědci přidávají do dané dlaždice voláním metody **OnObjectEnter**. Tato metoda pak volá odpovídající metodu podlahy. Poslední stranou je třída **ActuatorFloorTileSide**, která pouze navíc přidává nášlapný přepínač.

3.3 Renderery

Jelikož každá dlaždice a každé strany dlaždic mají renderery, je jím čas věnovat tuto sekci. Všechny tyto objekty a mnoho dalších k tomuto účelu implementují rozhraní **IRenderable**, které pouze vyžaduje položku typu **IRenderer**. Stěžejní funkce tohoto rozhraní jsou funkce **Render** a **Interact**.

Nejdůležitější z parametrů těchto funkcí je dosavadní transformace. Tj. obsahuje složeninu transformací složenou z jednotlivých transformací na cestě z kořene stromu reprezentující závislosti renderer na sobě. Takže například kořen strumu bude renderer dlaždice, jeho syn bude strana dlaždice, její syn bude výklenek ve zdi a její syn bude předmět ve výklenku(list). Při takovéto reprezentaci se pak musí všechny renderované objekty posunout či jinak transformovat vůči jejich rodiči. Tzn. každý renderer si musí zvolit pozicovací konvenci, kterou pak musí závislé renderery dodržovat. Tento způsob renderování je požíván u statických objektu jako jsou dlaždice, zdi, výklenky a podobně. Pro pohybující objekty je používaná absolutní pozice a renderery těchto objektů transformují objekty přímo na jejich pozici. Volba mezi těmito dvěma případy je na programátora.

Renderery jsou vždy dělány na míru objektu, který mají renderovat. Tzn. často se renderer v konstruktoru inicializuje instancí s konkretním typem. Třída této instance pak má zpravidla readonly vlastnosti, podle kterých renderer určuje, co má vykreslovat. Jelikož implementovaná grafická vrstva je pouze ve formě proof of concept, je co nejjednodušší a neobsahuje například animace. Nicméně ze zde popsané povahy renderer je jasné, že toho může být dosaženo vytvořením událostí na renderovaných objektech, které si renderer zaregistrouje. Dovedu si představit, že by šla s takovýmto návrhem velmi jednoduše udělat grafická vrstva, která bude velmi pěkná, bude mít kvalitní 3D modely animace. Zabralo by to jen spoustu času a byla by k tomu potřeba horda grafiků.

Při podědění nějakého rendereru je třeba zjistit dosavadní transformaci, která je normálně vypočítána v rodiči. K tomuto záměru slouží funkce **GetCurrentTransformation**, která bere jako parametr dosavadní transformaci. Poděděním jež existujícího rendereru lze ušetřit mnoho práce a opakujícího se kódu. Proto je takový přístup v této implementaci často použit. Funkce popsané na začátku sekce jsou samozřejmě virtuální a jdou overridevat, obsahují taky jeden parametr typu **object** pro případné předávaní dodatečných dat mezi renderery. Tohoto parametru není nikde v této implementaci využito.

Jelikož renderery zásadně mění vzhled a pozici všech věcí, je nutné tuto vrstvu propojit i s interakcemi. Interakční funkce má skrze parametr typu **ILeader** přístup k položce interactor, která je typu **object**. Daný renderer musí vědět, pro jaký typ interactoru je a na ten si ho musí vhodně přetykovat. V této implementaci je

použit paprsek (**Ray**). Celý tento koncept interactoru by šel sice udělat genericky, ale prolínal by se celou strukturou rendererů a z toho důvodu jsem se rozhodl v tomto místě ustoupit. A udělat situaci jednodušší na úkor kontroly za překladu.

3.4 Přepínače

3.4.1 Úvod

Jak již bylo řečeno, přepínače se skládají z jednotlivých senzorů. Senzory pak mohou provádět následující akce:

- změna stavu dlaždice
- proházení senzorů přepínače
- přidání zkušeností hráči

Senzory mohou být aktivovány:

- Kliknutím na dekoraci senzoru, pokud je senzor na zdi a je to poslední senzor přepínače.
- Zprávou od vyslanou jiným senzorem.

Tento typ přepínačů odpovídá přepínačům v původní hře.

Dále engine povoluje nové přepínače, který nemusí používat senzory. Je tedy čistě na programátorovi, jakým způsobem se bude přepínač aktivovat.

3.4.2 Implementace přepínačů se senzory

Za přepínačem stojí navenek interface **IActuatorX**. Toto rozhraní pouze vyžaduje API pro příjem zpráv typu **Message** pro dodržení kompatibility s původní hrou. Jelikož ostatní aktivace záleží na typu přepínače. Protože například nášlapný přepínač se aktivuje jiným způsobem než přepínač na zdi.

Implementace senzorů

Senzory provádějí samotné akce přepínačů popsané v úvodu. Odpovídající akce jsou provedeny pouze pokud byl daný senzor aktivován. Po kliknutí na přepínač, dojde postupně u všech senzorů o pokus jejich aktivace. Každý typ senzoru má jinou podmínu aktivace. Senzory se dělí na senzory použité na podlaze, na zdi a na speciální tzv. logické senzory.

Hlavní třídou reprezentující přepínač je třída **SensorX**. Tato třída obsahuje některé společné funkce, které může využít každý potomek. Dále také definuje vlastnosti, které lze rozdělit do tří skupin. Tyto vlastnosti jsou inicializovány inicializátorem, který slouží pouze jako datový nosič. Jde o inicializátor typu **SensorInitializerX**. Pro použití v potomcích se můžou použít poděděné verze tohoto inicializátoru. Následuje seznam vlastností.

Obecné vlastnosti:

- Opoždění akce - čas v milisekundách, po kterém se provede případná akce

- Informace, zda-li je efekt určen lokálně pro rodičovskou dlaždici, či pro nějakou jinou.
- Flag opakovatelnost - každý senzor ji používá trochu jinak, ale většinou pokud je true, tak se obrátí cílová akce
- Na jedno použití - senzor se po první aktivaci zablokuje, a není dále používán.
- Flag určující, zda po aktivaci dojde k přehrání zvuku. (v tomto enginu není použit)

Vlastnosti pro lokální akci:

- Flag určující, zda-li se má seznam senzorů při aktivaci zarotovat.
- Flag určující, zda-li se mají hráči po aktivaci přidat zkušenosti.

Vlastnosti pro vzdálenou akci:

- Efekt - Určuje jaký efekt má odeslaná zpráva po aktivaci. (Akce může být obrácená pomocí flagu popsaného v obecných vlastnostech) Hodnoty efektu jsou následující:
 - Aktivace
 - Deaktivace
 - Přepnutí stavu
 - Drž - Tento stav nelze odeslat ve zprávě a je na senzoru, aby ho interpretoval pomocí aktivace či deaktivace.
- Směr použitý v odesланé zprávě. Směr může být interpretován jako číslo. (Lze získat pomocí **MapDirection.Index**)
- Reference na cílovou dlaždici.

Kromě předchozích vlastností obsahuje tato třída funkce pro vykonání akcí senzorů. Funkce **TriggerEffect** odešle zprávu podle vlastností senzoru na danou pozici, pokud je cíl vzdálený. Jinak provede akci pro lokální efekt. Lokální efekt lze také vykonat přímo zavoláním funkce **TriggerLocalEffect**. Tyto funkce lze použít pouze v potomcích.

Dále se senzory dělí na senzory použité na zemi resp. na zdi. Tomu odpovídají třídy **FloorSensor** a **WallSensor**. Obě tyto třídy mají funkci **TryTrigger**, která se stará o provedení případného efektu. Zda-li se efekt provede stanoví funkce **TryInteract**. Pro každou variantu mají funkce odlišné parametry. Pokud tedy chcete vytvořit vlastní senzor, implementujte funkci **TryInteract**. Tato funkce bude volána z funkce **TryTrigger** a bude tedy dělat obecnou funkcionalitu odeslání zprávy a určení výsledného efektu zprávy. Pokud chcete změnit i tuto funkcionalitu, poskytněte vlastní implementaci i pro tuto funkci.

Kromě předchozích senzorů ještě existují tzv. logické senzory. První z nich je **LogicGateSensor**, který funguje jako logické hradlo. Celkem má k dispozici osm bitů. Čtyři nich jsou nastaveny na nějaké hodnoty a ostatní na nuly.

S druhou čtvericí lze manipulovat pomocí zpráv. Index směru zprávy určuje kolikátý bit se má ovlivnit. A tento bit je pak ovlivněn akcí zprávy. Pokud se pak stane, že jsou obě čtveřice stejné, je vyvolaný efekt senzoru. Dalším senzorem je **CounterSensor**, který má dané číslo. Aktivační zprávy toto číslo poté navyšují a deaktivace zprávy ho snižují. Pokud je číslo nula, je vyvolaný efekt.

Implementace přepínačů používající senzory

Rodiče všech přepínačů kompatibilních originální hře tvoří třída **Actuator**. Tato třída poskytuje pouze společnou rutinu pro zarotování senzory přepínače. Lze vyvolat pouze z potomků. Zda-li se mají senzory zarotovat určuje dle vlastnosti **Rotate**, která se poté nastaví na false. Tato funkce se volá z potomků pokud proběhl pokus o aktivaci nějakého senzoru.

Prvním potomkem předchozí třídy je třída **FloorActuator**, která může obsahovat pouze senzory určené na podlahu tj. **FloorSensor**. Pokus o její aktivaci se provádí funkcí **Trigger**. Jako první parametr se jí předává objekt, který vstupuje/odchází z dlaždice. Potom seznam všech objektů na dlaždici a naposled informace zda objekt vstupuje či vystupuje. Pro přepínač je vytvořena speciální podlaha **ActuatorFloorSide**, která se stará o volání zmíněné funkce.

Dalším potomkem je třída **WallActuator**, která může obdobně obsahovat pouze senzory určené na zeď tj. **FloorSensor**. Pokus o její aktivaci se provádí opět funkcí **Trigger**, která bere jediný parametr typu **ILeader**. Pro přepínač již není vytvořena žádná speciální strana dlaždice, ale je použita strana přijímající obecné přepínače **ActuatorWallTileSide**. Funkce **Trigger** je pak volána skrze renderer dané strany.

Poslední speciální implementací je třída **LogicActuator**, která může obsahovat již zmíněné logické senzory. O komunikaci s jejími senzory se stará rodič, pomocí zasílání zpráv. Pro tento přepínač je vytvořena speciální dlaždice, která není odnikud přístupná. Tato dlaždice je reprezentovaná třídou **LogicTile**.

3.4.3 Implementace obecných přepínačů

Tato sekce pojednává přepínačích, které ke své funkci nepoužívají senzory. Jejich vznik byl podnícen snahou, dát programátorovi větší svobodu při tvorbě přepínačů, nicméně zároveň řeší následující problém. Kromě standardních dekorací v se v dungeonu vyskytují dekorace, které provádějí nějakou funkci po kliknutí na ně. K těmto dekoracím však v originální hře nenáleží žádné senzory. Jedná se o tzv. náhodné dekorace, kdy každá zeď má definováno, zda může obsahovat náhodné dekorace. Při generování mapy se pak s určitou pravděpodobností dekorace na zeď dá či nikoliv. Z toho důvodu tento engine neobsahuje stejně dekorace na stejných místech, protože není znám náhodný generátor k tomu použitý. Nicméně zpět k problému akcí po kliknutí na dekoraci. Studii dekomplikovaného kódu originální hry (viz.[?]) se ukázalo, že některé akce jsou fixovány na konkrétní typy dekorací. Takže například v dekorace s výklenky mají napevno implementovaný kód umožňující vkládaní předmětů do výklenku. Dalším příkladem jsou fontánky, které naplňují lahve s vodou. Posledním příkladem speciální dekorace je Vi Altair výklenek, který dokáže po vložení kostí oživit šampiona. Jelikož tyto dekorace mohou být i součástí senzorů, bylo zapotřebí udělat kolem samotných dekorací

wrapery, které zajišťují případné akce. Tyto wrapery jsou tedy implementací přepínačů tj. rozhraní **IActuatorX**, kdy dekorace bez akce neprovádějí žádnou akci a dekorace s nějakou akcí danou akci provádějí. O samotné vyvolání interakce se opět musí postarat renderery. Takže zmiňované senzory jsou vlastně přepínače v přepínačích.

Obecné přepínače tedy nemají nikterak napevno zvolený formát, je čistě na programátora, jak implementuje uvnitř jejich funkci a jakým způsobem bude s přepínači prováděna interakce. Existující implementace tedy jsou **DecorationItem**, která neprovádí žádnou interakci, ale pouze rendering o který se stará **DecorationRenderer**. Dále jde o třídu **Alcove**, která reprezentuje výklenek a má odpovídající **AlcoveRenderer**. A poslední z nich je **ViAltairAlcove**, který dědí ze třídy **Alcove** a stará se o již zmiňovanou reinkarnaci šampionů.

3.5 Herní entity

Za deklarací neživých entit stojí rozhraní **IEntity**. Definuje pouze funkci pro získání vlastnosti dané entity. Za živými entitami stojí naopak rozhraní **ILiveEntity**, která navíc disponuje funkcí pro získání schopností. Dále deklaruje:

- tělo entity
- způsob rozmístění entity na dlaždici
- relace s dalšími entitami

3.5.1 Implementace vlastností entit

Vlastnost deklaruje rozhraní **IProperty**. Nejprve obsahuje vlastnost **Value**, která stanovuje nynější hodnotu dané vlastnosti. Tato hodnota je jako jediná možná měnit z venčí. V jejím getteru a setteru se typicky provádějí kontroly okrajových hodnot a případné reakce na ně. Další vlastností je **BaseValue**, což je nejvyšší hodnota, které může vlastnost nabýt, pokud není nějak modifikovaná. Maximální hodnotu včetně modifikací určuje vlastnost **MaxValue**. Poslední vlastností je **AdditionalValues**, která shromažďuje právě dané modifikace. Je to sekvence typu **IEntityPropertyEffect**, která definuje hodnotu a typ hodnoty. Poslední vlastností je již zmíněný typ vlastnosti. Jednotlivé typy vlastností jsou definovány jako reference na třídy implementující rozhraní **IPropertyFactory**. Entita na základě tohoto typu potom vrátí příslušné vlastnosti. Typem může být jakákoli třída implementující toto rozhraní, nicméně typická taková třída nemá žádné položky. Proto existuje generická třída **PropertyFactory**, která bere jako typový parametr právě typ alespoň **IProperty**. Tato třída se řídí vzorem singleton, jejíž jediná instance lze získat přes statickou položkou **Instance**. Takto lze pak generovat typy pomocí samotné třídy reprezentující vlastnosti, takže není nutné deklarovat stále nové třídy, které nic nedělají.

Za zmínku ještě stojí částečná implementace rozhraní **IProperty** a to třída **Property**. Tato třída definuje maximální hodnotu jako součet základních hodnoty a sumu modifikujících hodnot. Dále Při nastavení konkrétní hodnoty oře-

zává hodnotu do povoleného intervalu to je mezi nulou a maximální hodnotou. Také definuje událost vyvolanou při změně hodnoty. A naposledy definuje kolekci modifikujících hodnot jako `HashSet<T>`. Všechny vlastnosti jsou deklarovány abstraktně nebo virtuálně. Tato třída lze použít k usnadnění implementace vlastností.

Některé vlastnosti mohou požadovat přístup k jiným vlastnostem či přístup k samotné rodičovské entitě. V takovém případě je čistě na programátora, jak takového cíle dosáhne. Typicky se v takovém případě předá v konstruktoru potřebná vlastnost nebo se vytvoří událost, která danou závislost deleguje vně.

Seznam předdefinovaných vlastností je možné najít ve jemném prostoru `DungeonMasterEngine.DungeonContent.Entity.Properties`. Jak již bylo zmíněno, může nastat, že daná entita vlastnost neobsahuje. V takovém případě je dobré kvůli dodržení konzistentnosti vracet předdefinovanou instanci třídy `EmptyProperty`. Nicméně rozhodně to není povinností při všech použití enginu.

3.5.2 Implementace schopností entit

Schopnosti deklaruje rozhraní `ISkill`. První vlastností tohoto rozhraní je `SkillLevel`, která udává úroveň, na které je daná schopnost. Do této hodnoty jsou započítány i úrovně získané například kouzelnými předměty. Hodnotu bez těchto extra úrovní určuje vlastnost `BaseSkillLevel`. Dále obsahuje vlastnosti `Experience` a `TemporaryExperience` určující dosavadní hodnotu zkušeností. Dále obsahuje vlastnost `BaseSkill`, což je reference na základní(viz. analýza) schopnost. Pokud je daná schopnost již základní, je hodnota `null`. Přidat schopnosti zkušenosti je možné zavoláním funkce `AddExperience`. Konkrétní algoritmus přepočítávání zkušeností na levele záleží vždy na konkrétní implementaci. Stejně jako u vlastností i zde existuje vlastnost `Type`, tentokrát ale typu `ISkillFactory`. Stejně jako u vlastností, je možné tyto reprezentanty vytvářet pomocí generické třídy `SkillFactory`.

Třída `SkillBase` implementuje získávání zkušeností schopností jako v originální hře. Všechny schopnosti využívající tuto třídu jsou ve jemném prostoru `DungeonMasterEngine.DungeonContent.Entity.Skill`. Stejně jako u vlastností i zde entita nemusí mít dotazovanou schopnost. V takovém případě vrací instanci třídy `EmptySkill`.

3.5.3 Tělo a inventáře entity

Následující API umožňuje vytvořit různé druhy těl pro různé entity. Každá část těla může sloužit jako úložiště. Dále pak existují externí úložiště. API je natrhnuté tak, aby dovolovalo definovat různé druhy úložišť a těl. V základu obshuji definice pouze pro lidské tělo.

Inventář

Inventář je definován rozhraním `IInventory`. Tak především každý inventář má definované jakého je typu. Typ inventáře musí implementovat rozhraní `IStorageType`. Toto rozhraní definuje velikost inventáře. Dále definuje samotné úložiště jako `ReadOnlyList`. A v poslední řadě obsahuje funkce pro přidávaní a odebírání předmětů z úložného prostoru.

Třída **Inventory** implementuje všechny funkce inventáře. Je jí tedy možno přímo použít nebo rozšířit.

Část těla

Část těla je definovaná rozhraním **IBodyPart**, které je potomkem rozhraní **IInventory**. Navíc definuje TODO

Obecnou implementací rozhraní je tříd **BodyPart**. Typ konkrétního úložiště předaný v konstruktoru určuje typ části těla.

Tělo entity

Tělo entity je definováno rozhraním **IBody**. Obsahuje seznam částí těla, seznam všech úložišť včetně částí těla a funkce pro vyhledání úložiště či části těla dle typu.

Engine obsahuje pouze implementaci pro lidské tělo tj. třida **HumanBody**

3.5.4 Rozmístění entity na dlaždici

Definice prostorů a cest na dlaždici

Rozhraní **IGroupLayout** definuje obecně možné rozmístění na dlaždici. Nejprve obsahuje všechny možné prostory, které lze využít. Jednotlivé prostory reprezentují rozhraní **ISpace**. Dale API musí poskytovat funkce pro nalezení cesty z libovolného prostoru ,definovaného layoutem, na libovolnou sousední dlaždici nebo k libovolné ze stran dlaždice. Jednotlivé články cesty jsou reprezentovány rozhraním **ISpaceRouteElement**. Poslední funkce musí umět vytvořit z prostoru a dlaždice článek cesty, tedy **ISpaceRouteElement**.

Rozhraní **ISpace** musí definovat, jakým stranám dlaždice je prostor přilehlý. Dále musí definovat prostor, který využívá na dlaždici, pomocí obdélníku. Jak již bylo řečeno obdélník musí zabírat prostor z 1000x1000 pole. Přičemž souřadnice rostou shora dolů a zleva doprava. Nahoře je pak sever, vpravo východ, dole jih a vlevo západ. Kromě toho také musí definovat sousední prostory, k čemuž využívá již známé generické rozhraní **INeighborable**.

Rozhraní **ISpaceRouteElement** se potom skládá pouze s prostoru, dlaždice a absolutně pozicované lokace, na které má stát daná entita. Pro výpočet pozice se může použít pozice dlaždice a samozřejmě daný prostor. Při definici vlastního layoutu je možné využít předem připravený hledač nejkratších cest pro prostory **GroupLayoutSearcher**. Pro reprezentaci sousedů prostorů je tu zase tříd **FourthSpaceNeighbors**.

Celé toto rozhraní je readonly struktura, která pouze definuje prostory na dlaždici a způsob pohybu mezi nimi. Z toho důvodu dává dobrý smysl instance tříd reprezentující toto rozhraní implementovat jako singletony. Engine definuje tři layouty a to pro čtyři, dvě a jednu entitu. Ostatní možné

Řízení obsazeného prostoru na dlaždici

K předchozímu mechanismu je ještě třeba další část, která bude zaznamenávat samotné využité pozice na dlaždici. K tomuto účelu existuje třida **LayoutManager**. Lze pomocí ní získat seznam entity na dlaždici a seznamy

využitých prostorů dlaždic. Dále poskytuje API pro přidání entity na daný prostor na dlaždici, odebrání prostoru a získání entit, které využívají alespoň část nějakého prostoru. Z toho vyplývá, že entity s různým layoutem mohou být na stejně dlaždici, pokud je na ní dostatek prostoru pro oba prostory definované layoutama.

3.5.5 Relace s dalšími entitami

Každá živá entita musí definovat vlastnost typu **IRelationManager**. Toto rozhraní musí definovat relační token typu **RelationToken** pro danou entitu. Dále definuje funkce, která pro daný token vrátí, zda entity odpovídající danému tokenu je nepřátelská. Jednoduchou implementací tohoto rozhraní je třída **DefaultRelationManager**, která má si při svém vzniku definuje své neměnné nepřátelé. Nicméně pokud daná implementace nevyhovuje je čistě na programátora, aby si vytvořil implementaci vlastní. K dispozici je ještě generátor unikátních tokenů a to statická třída **RelationTokenFactory**.

3.5.6 Implementace entit

Jak již bylo zmiňováno, jsou v tomto enginu obsažené dvě implementace živých entit a jedna neživá.

Šampion

Šampion je první živá entita a reprezentuje jej třída **Champion**. Šampion neobsahuje žádnou umělou inteligenci, je ovládán hráčem skrze třídu **Theron**, která reprezentuje hráčovu skupinu šampionů. Inicializace vlastností a schopností probíhá přes datový initializer definovaný rozhraním **IChampionInitializer**. Resp. jeho předáním do konstruktoru, dále je nutné specifikovat relační token a seznam nepřátel. Posunout daného šampiona je možné přiřazením do vlastnosti **Location**.

Zde je dobré zmínit generickou třídu **Animator**, která bere dva typové parametry. První z nich je typ posouvaného objektu, který musí implementovat alespoň rozhraní **IMovable**. Toto rozhraní definuje vlastnosti, pomocí kterých lze měnit pozici daného objektu. Dalším parametrem je typ prostorů, mezi kterými se objekt pohybuje. Ten musí implementovat alespoň rozhraní **IStopable**, které definuje pozici, na kterém se má objekt postavit. Animátor poskytuje API pro plynulý posun objektů mezi prostory.

U šampiona je tento animátor použit automaticky při změně lokace.

Příšera

Další a poslední živou entitou ve hře je příšera, kterou reprezentuje třída **Creature**. Tato třída reprezentuje všechny příšery ve hře. Vlastnosti jednotlivých typů příšer jsou ve třídách typu **CreatureFactory**. Každá konkrétní instance příšery má referenci na tuto třídu a její chování je ovlivněno vlastnostmi v ní obsažené.

Tato třída definuje pro příšery jednoduchou umělou inteligenci. Za zmínu stojí, že ke zjednodušení implementace jsou zde opět využity asynchronní funkce.

Ty jsou především využity pro vytváření zpoždění akcí pomocí **Task.Delay**, bez nutnosti složitého počítání času a vracení se zpět do funkce po jeho uplynutí. Asynchronní funkce jsou však prováděny v jednom vlákně a to vždy ve funkci **Update**. V každém volání funkce je vyprázdněna celá fronta asynchronních funkcí, proto je třeba dát pozor na její přehlcení, které může vést až k neresponzivnosti aplikace. Následuje seznam funkcí a jejich popis použitých pro simulování inteligence příšer. Všechny tyto funkce je možné overridovat.

- **Live** - V této funkci je nekonečný cyklus, který volá obslužné rutiny podle stavu příšery. Jsou to:
 - Lov - příšera spatřila nepřítele a pronásleduje ho
 - Cesta domů - příšera pronásledovala nepřítele, kterého následně ztratila, proto jde domů tj. na místo svého vzniku
 - Hlídkování - příšera hlídka v okolí oblasti svého vzniku
- **FindEnemies** - Pomocí prohledávání do šírky se pokusí najít entity s nepřátelským tokenem. Maximální hledanou oblast je určena dle vlastnosti příšery. Pokud je nepřítel nalezen nastaví proměnnou **hauntingPath**.
- **MoveToSpace** - Přesune příšeru mezi prostory pomocí animátoru a nastaví správně použité prostory pomocí **LayoutManageru**.
- **MoveThroughSpaces** - Přesune příšeru přes prostory dlaždice až na cílovou dlaždu pomocí funkce **MoveToSpace**. Při každém pohybu hledá nepřátele pomocí funkce **FindEnemies**, pokud je tak nastaveno parametrem funkce. Pokud nějaké najde, přeruší pohyb.
- **MoveToNeighbourTile** - Posune příšeru na určenou dlaždu přes prostory dlaždice pomocí funkce **MoveThroughSpaces**. Pokud je cesta nepřístupná vrátí **false**.
- **GoHome** - Příšera jde domů podle cesty uložené v proměnné **homeRoute** a poté ji nastav na **null**. Mezi dlaždicemi se posunuje pomocí funkce **MoveToNeighbourTile**.
- **FindNextWatchLocation** - Pomocí prohledávání do šírky z místa objevení příšery naleze dlaždu, na které dlouho příšera nebyla a vrátí k ní cestu.
- **WatchAround** - Pomocí funkce **FindNextWatchLocation** naleze cestu na další místo k hlídkování. Poté jde na dané místo pomocí funkce **MoveToNeighbourTile**.
- **Fight** - provede útok na nepřítele.
- **PrepareForFight** - Dostane se co nejblíže k dlaždi, na který je nepřítel a zahájí útok pomocí funkce **Fight**.
- **GetPathHome** - Pokusí se nalézt cestu domů. Pokud ji naleze nastaví **homeRoute**.
- **EstablishNewBase** - Nastaví výchozí dlaždu na nynější.

- **Hount** - Příšera pronásleduje nepřítele na poslední místo, kde ho spatřila. Pokud je nepřítel na sousední dlaždici připraví se k útoku pomocí funkce **PrepareForFight**. Pokud nepřítele ztratí, pokusí se najít cestu domů pomocí **GetPathHome**. Pokud cestu nenaleze založí si domov tam, kde je pomocí funkce **EstablishNewBase**.

K prohledávání do šířky se používá třída **BreathFirstSearch** nebo některý její potomek.

3.6 Předměty

Předměty reprezentuje rozhraní **IGrabableItem**. Jak již bylo zmíněno v analýze, každý předmět musí mít referenci na svoji továrnu minimálně typu **IGrabableItemFactoryBase**. Dále musí definovat vlastnost určující dlaždici, na které se nachází. Přiřazení do této vlastnosti by mělo vyvolat přidání daného předmětu na danou dlaždici. Existují i případy, kdy toto není žádoucí a proto musí ještě definovat funkci, která pouze danou vlastnost nastaví. V poslední řadě musí definovat renderer.

Rozhraní továrny na obecné předměty vyžaduje vlastnosti jako název, hmotnost, seznam možných akcí s předmětem a také definuje místa, kam lze předmět uložit.

3.6.1 Implementace předmětů

Při tvorbě vlastního předmětu je třeba vytvořit implementaci zmíněného rozhraní **IGrabableItem**. Tato implementace by kromě vyžadovaných položek měla obsahovat položky, které se můžou měnit pro každou instanci daného typu předmětu. Dále je třeba vytvořit samotnou továrnu na tyto předměty implementací rozhraní **IGrabableItemFactory**. Ta by měla obsahovat všechny společné vlastnosti pro daný typ předmětu. Z toho důvodu je dobré do vlastností přidat kromě rozhraním požadované reference na továrnu i přesně typovanou referenci, skrze kterou bude možné k vlastnostem přistupovat. Podobnou věc je dobré udělat i v továrně pro funkci vytvářející daný objekt. Inspirace lze nalézt v již implementovaných třídách, které jsou v jemném prostoru **DungeonMasterEngine.DungeonContent.GrabableItem**.

3.7 Akce

TODO

3.8 Kouzla

TODO

3.9 Builder map

Nyní když máme rozebrané všechny části enginu, které je potenciálně možné rozšířit, je čas popsat, jak se vše sestaví dohromady v herní úroveň.

3.9.1 Vlastní implementace builderu

Jak již bylo zmíněno builder musí implementovat rozhraní **IDungeonBuilder**. Toto rozhraní je potom dotazované pro načtení levelu jádrem enginu. Je úkolem builderu rozhodnout jak bude jednat v případě, že je podán několikrát dotaz na stejnou mapu. Typicky je se tedy nutné rozhodnout jestli načtené mapy kešovat či nikoliv. Existuje částečná implementace tohoto rozhraní **BuilderBase**, která má funkci pro vyplnění sousedů dlaždic do inicializátorů. Nicméně nic jiného neposkytuje, takže je zcela v pořádku implementovat přímo dané rozhraní.

Správně implementovaný builder by měl načítat mapy zhruba tímto způsobem.

- Vytvoření dlaždic a naplnění jejich inicializátorů. To znamená vytvořit strany dlaždic a přepínače či předměty v nich obsažené.
- Nastavení sousedů dlaždic do inicializátorů.
- Vytvoření samotného levelu **DungeonLevel**.
- Dokončení inicializace inicializátoru a zavolení inicializaci dlaždic skrze inicializátory.
- Případné kešování levelu pro případný opakováný požadavek.

Na tyto body se samozřejmě navazuje spousta další věcí, které je třeba vyřešit. Je to například vytvoření továren pro akce a předměty. Továrny pro akce jsou pak použity pro továrny pro předměty a ty zase pro vytvoření samotných předmětů. Dále je potřeba vyřešit nastavení rendererů. Pokud má builder podporovat různé zobrazovací vrstvy, je potřeba nějakým způsobem poskytnout možnost předání jiných rendererů builderu. Je tedy například možné vytvořit rozhraní, které bude vracet dané renderery používané v builderu. Odlišnou implementaci rozhraní se potom budou požívat odlišné renderery.

3.9.2 Implementace builderu Dungeon Masteru

Builder pro samotnu hru Dungeon Master řeší třída **LegacyMapBuilder**. Tento builder využívá parseru binární dat originální hry a to **DungeonParser**. Při vytvoření instance builderu předchozím parserem rozparsuje všechna data, která pak jsou skrze něj k dispozici jako třída **DungeonData**. Jak bylo zmiňováno v analýze, parser kromě samotných dat načítá i ostatní data jako například vlastnosti předmětů atp. Vše je k dispozici ve zmíněné třídě. Po rosparsování dat, jsou na základě těchto dat vytvořeny továrny pro předměty, akce a příšery. To vše stále v konstruktoru.

Samotný builder map potom využívá ještě další buildery. Jsou to třídy pro vytváření dlaždic, přepínačů, předmětů a příšer. Při požadavku na určitou mapu pak následuje postup zmíněný v předchozí sekci.

3.9.3 Rozšíření builderů Dungeon Masteru

TODO

4. Uživatelská dokumentace

Tento projekt si neklade za cíl udělat zcela kompletní a dobré hratelnou reimplementaci Dungeon Master. Naopak se soustředí na dobrý návrh enginu jako takového. Z toho důvodu není herní zážitek nikterak oslnující, nicméně jako demonstrace funkčnosti enginu poslouží dobře. Hru je možné spustit souborem `DungeonMasterEngine.exe`.

4.1 Mechaniky ve hře

Hráč reprezentuje vůdce skupiny bojovníků známých jako šampióni. Může ovládat jejich pohyb, předávat jim předměty, donutit je k boji, ke kouzlení, ke konzumaci jedlých předmětů nebo lektvarů. Každý bojovník má sadu vlastností a schopností, v kterých je schopný se zdokonalovat získáváním zkušeností. Zkušenosti lze získat bojem na prázdro, bojem proti příšerám, kouzlením, nebo jen použitím přepínačů. V dungeonu jsou přepínače na zdích, které lze aktivovat kliknutím myši. Dále jsou zde přepínače na podlaze, které lze aktivovat vstoupením na podlahu nebo hozením předmětu na daný spínač. Předměty lze pokládat na zem, do výklenků a nebo je uložit na tělo bojovníka či do jeho batohu. Přepínače pak mohou aktivovat nebo deaktivovat určité objekty ve hře jako jsou například dveře, teleporty, jámy, otevírací zdi atp. Některé dveře je také možné rozbít útkem. Některé teleporty jsou pouze na konkrétní typy objektů. Do jam lze spadnout, ale většinou se z nich lze právě teleportem dostat ven.

4.2 Cíl hry

Jak již bylo zmíněno, tak hra není úplně kompletní, proto ji není možné zcela dohrát. Z toho důvodu by se za cíl hry dalo považovat dostání se do posledního implementovaného levelu hry.

4.3 Ovládání

4.3.1 Pohyb

Pohybovat skupinou lze pomocí kláves w,s,a,d tj. dopředu, dozadu, doleva, doprava. Dále se je možné rozhlížet pomocí šipek.

4.3.2 Aktivace přepínačů a sbírání předmětů

Přepínače lze aktivovat namířením ukazatele myši na daný objekt a kliknutím nebo stiskem klávesy **ENTER**. Pokud se pod kurzorem vyskytuje předmět nebo přepínač, který může vložit nějaký předmět hráčovi do ruky, tak se tak provede pokud je hráčova ruka prázdná. Pokud hráč již v ruce něco má, provede se opačná akce tj. předmět se položí na dané místo nebo je pohlcen přepínačem. Ostatní interakce probíhá přes konzoli. Konzole se aktivuje stiskem klávesy **TAB** a deaktivuje opětovným stisknutím tytéž klávesy. Příkaz **hand** zobrazí popis předmětu

v hráčově ruce. Přidáním parametru **take** se provede uložení předmětu do dotažovaného inventáře daného bojovníka. Přidáním parametru **put** se naopak vloží dotazovaný objekt z inventáře do ruky.

4.3.3 Souboj

Pro souboj je nejprve nutné vložit zbraň do akční ruky. Což je možné provést příkazy z minulé sekce. Samotný souboj probíha potom skrze příkaz **fight**, kdy je interaktivně vybrán bojovník a způsob útoku.

4.3.4 Konzole

Pro více informací o příkazech je možné napsat příkaz **help** do konzole.

Závěr

V rámci této práce byl naimplementovaný engine pro Hru Dungeon Master. Engine neobsahuje přesně všechny funkce, jaké byly v originální hře. Nicméně drtivou většinu z nich se podařilo uskutečnit. Pro implementaci byl použit jazyk C#, platforma .NET a framework MonoGame. Většina částí enginu je jednoduše rozšířitelná či modifikovatelná. To vede na dobrou udržitelnost celého systému. Primárně je engine určený pro hru Dungeon Master a dokáže si herní úrovně načíst z originální binárních dat. Nicméně načítání map je v oddělené vrstvě. Proto je možné tuto vrstvu upravit kvůli případným rozšířením nebo ji je možné nově naimplementovat i pro jiné vstupní formáty. Takže je možné tento engine využít i pro implementaci jiných her. Engine má také oddělenou renderovací vrstvu. Z toho opět vyplývá, že renderovací vrstva lze jednoduše rozširovat.

Seznam použité literatury

- [1] J. A. L. de Farias. MonoGame. 2009. URL <http://www.monogame.net/>.
- [2] C. Fontanel. Technical Documentation - Dungeon Master and Chaos Strikes Back Actions and Combos. 2005. URL <http://dmweb.free.fr/?q=node/690>.
- [3] C. Fontanel. Technical Documentation - Dungeon Master and Chaos Strikes Back Items properties. 2005. URL <http://dmweb.free.fr/?q=node/886>.
- [4] C. Fontanel. Dungeon Master Items. 2005. URL <http://dmweb.free.fr/?q=node/259>.
- [5] C. Fontanel. Dungeon Master and Chaos Strikes Back Spells. 2005. URL <http://dmweb.free.fr/?q=node/195>.
- [6] C. Fontanel. Technical Documentation - File Formats - Dungeon File (DUNGEON.DAT). 2005. URL <http://dmweb.free.fr/?q=node/217>.
- [7] C. Fontanel. Technical Documentation - Dungeon Master and Chaos Strikes Back Creature Details. 2008. URL <http://dmweb.free.fr/?q=node/1363>.
- [8] C. Fontanel. Technical Documentation - Graphics.dat Item 559. 2008. URL <http://dmweb.free.fr/?q=node/1395>.
- [9] C. Fontanel. Back to the source: ReDMCSB. 2014. URL <http://www.dungeon-master.com/forum/viewtopic.php?f=25&t=29805>.
- [10] S. Mourier. Html Agility Pack. 2012. URL <https://htmlagilitypack.codeplex.com/>.

Přílohy