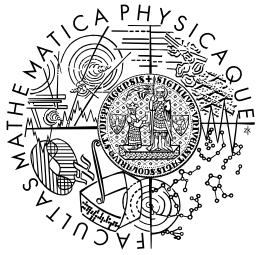


MATEMATICKO-FYZIKÁLNÍ
FAKULTA
Univerzita Karlova

BAKALÁŘSKÁ PRÁCE

Petr Geiger

Hra Dungeon Master pro platformu .NET

Katedra distribuovaných a spolehlivých systémů

Vedoucí bakalářské práce:	Mgr. Pavel Ježek, 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 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. Pavel Ježek, 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 dále uvedené aspekty. Hra je naprogramována v jazyce C# s využitím platformy .NET. Dále celý engine je navržený směrem k udržitelnosti a rozšiřitelnosti, tzn. s využitím tohoto enginu je možné vyrobit a vyvinout 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 engine může sloužit jako ukázkový příklad použitelný při výuce programování.

Klíčová slova: RPG, engine, 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. Pavel Ježek, Ph.D., Department of Distributed and Dependable Systems

Abstract: The goal of this thesis is to reimplement the Dungeon Master game. Currently there exist several clones of this wellknown game. However, compared to them this thesis focuses on aspect stated below. The game is implemented in the C# language using .NET platform. Furthermore, the entire engine is designed towards sustainability and scalability – i. e. that by using this engine it is possible to design slightly different game based on the same principles. 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 can serve as a representative example of a complex program in programming courses.

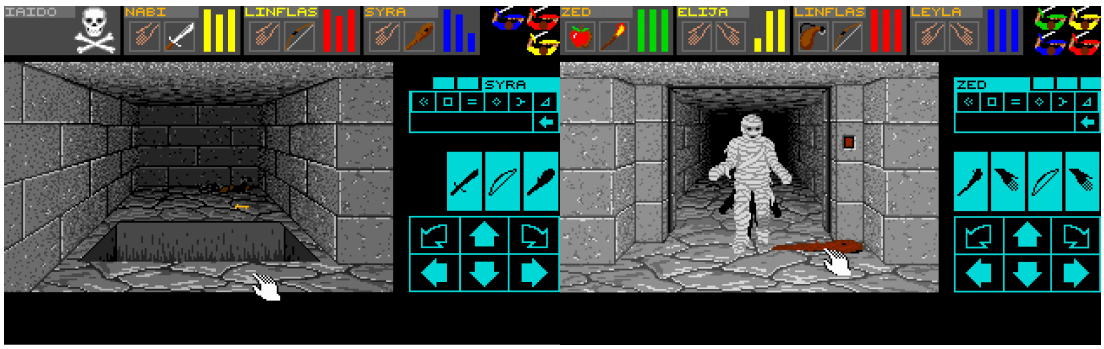
Keywords: RPG, engine, Dungeon Master, software architecture, education

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

Obsah

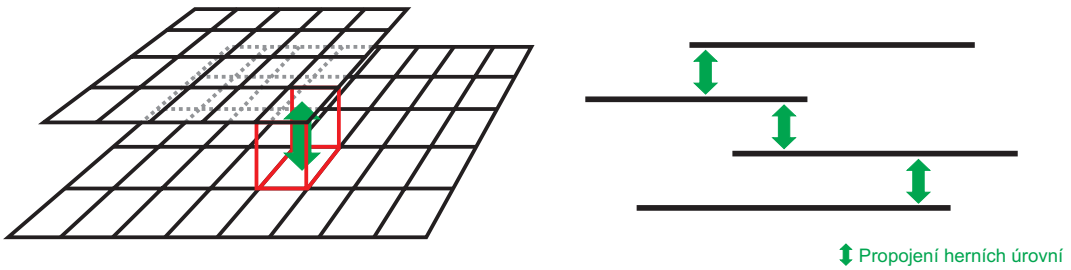
1. Úvod

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 mají různé dovednosti, ve kterých se mohou zdokonalovat.



Obrázek 1.1: Screenshot originální hry Dungeon Master

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. přepínači. 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.

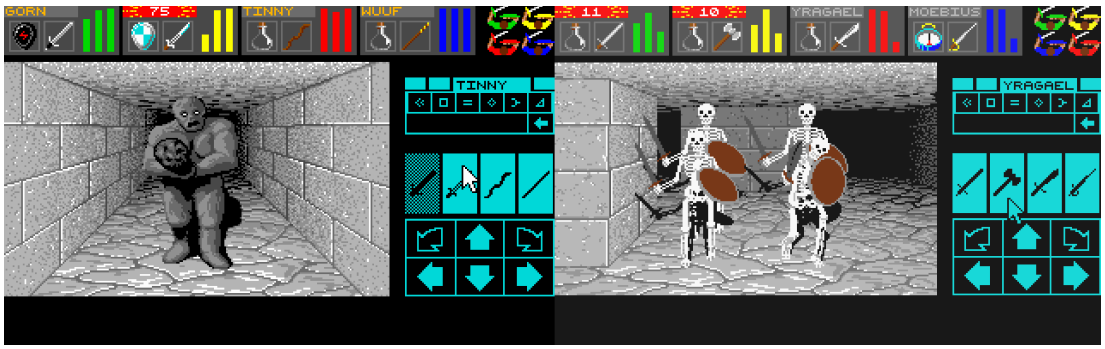


Obrázek 1.2: Ilustrace uspořádání herních úrovní.

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. Kouzla pak mohou být útočná, obraná nebo jimi lze například vytvářet lektvary.

Ve hře je celá řada nepřátelský 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: Nepřátelská entita vlevo zaujímá celý prostor dlaždice. Entity vpravo oproti tomu využívají pouze čtvrtiny dlaždic.

1.1 Reimplementace Dungeon Masteru

Hra Dungeon Master má potenciálně mnoho míst, která by šla jednoduše rozšířit, kdyby měl její engine vhodný objektový návrh. V takovém případě by bylo například možné jednoduše přidat dlaždice, předměty, kouzla, akce, šampiony nebo nepřátelské entity s novou funkcionalitou. Z toho důvodu jsme se rozhodli provést reimplementaci hry Dungeon Master, v které bude právě možné její části takto rozšiřovat. Výsledkem reimplementace má být engine vhodný pro výuku jazyka C#. Z tohoto důvodu je nutné, aby byl jazyk C# použit pro implementaci enginu.

Jazyk C# je objektově orientovaný, a proto je třeba, aby výsledný engine měl dobrý objektový návrh, který bude sloužit jako vhodný materiál k jeho výuce. Díky tomu by mělo být možné vytvářet úlohy pro studenty, v kterých by si mohli doimplementovat další funkce enginu nad rámec této práce a tak si vyzkoušet objektově orientované programování v praxi. Proto by také mělo být možné pro engine jednoduše dodělat komponentu, která by prováděla odlišný výstup enginu, díky kterému by mohly být úkoly kontrolované strojově například formou CodExu.[1]

Aby byl výsledný engine dostatečně komplexní je třeba zachovat co nejvíce herních mechanik hry Dungeon Master. Pro využití enginu způsobem popsaným v

[1]CodEx [1] je systém pro automatické vyhodnocování zdrojových kódů vzniklý na MFF UK. V CodExu lze vytvořit úlohy skládající se ze specifikace a sady testovacích vstupních dat. Studenti pak můžou pro tyto úlohy vypracovat řešení a odevzdat jeho zdrojové kódy. Tyto zdrojové kódy jsou následně zkompilované a je provedena kontrola jejích výstupů.

předchozím odstavci je pak nutné, aby tyto mechaniky šlo rozšiřovat. Důsledkem toho musí engine obsahovat komponentu pro převedení vstupních dat z originální hry na odpovídající herní úrovně. Kromě toho je třeba poskytnout mechanismus, kterým bude možné dodávat vlastní implementace těchto komponent. V poslední řadě je nutné oddělit výstup enginu do zvláštní vrstvy, aby ji bylo možné taktéž jednoduše změnit poskytnutím nové implementace.

1.2 Cíle

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

C1 Engine bude naprogramovaný v jazyce C#.

C2 Engine bude obsahovat podporu pro funkce a mechaniky vyskytující se ve hře Dungeon Master.

C3 Bude kladen důraz na dobrý objektový návrh tak, aby byl engine co nejlépe rozšiřitelný a bylo tak možné do enginu dodávat jednoduše nové funkce.

C4 Engine bude schopný sestavit herní úrovně podle vstupních dat použitých v originální hře. Nicméně bude také poskytovat možnost dodělat si podporu pro jiné formáty.

C5 Engine bude obsahovat oddělenou zobrazovací vrstvu tak, aby mohl být její výstup jednoduše změněn.

C6 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 použitých při implementaci nového enginu. Jsou zde popsány jak slepé a nevhodné návrhy, tak návrhy, které se ukázaly jako nejvhodnější 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 proto dekompilované zdrojové kódy [10] originální hry v jazyce C s archaickou konvencí Kernighan & Ritchie.[1] Tyto zdrojové kódy jsou často špatně srozumitelné, nicméně potřebné informace pro dokončení této práce se z nich podařilo získat.

[1] Brian Kernighan a Dennis Ritchie jsou autoři první verze jazyka C, jehož neformální specifikaci popsali v první edici knihy *The C Programming Language* [11]. Podle iniciálů autorů této knihy je tato první verze jazyka známa jako K&R C. Mezi největší specifika K&R C patří:

A. deklarace funkcí nespecifikuje parametry a není prováděna jejich kontrola,

B. funkce nemají návratový typ `void`,

C. při definici funkce jsou typy parametrů specifikovány na separátních řádcích za hlavičkou funkce,

D. lokální proměnné funkcí jsou deklarovány na začátku bloku funkce.

Ukázka kódu:

```
//declaration
typedef int VOID; // viz bod B
VOID write_sorted(); // viz bod A

//definition
VOID write_sorted(first, second) // viz bod C
int first;
int second;
{
        int min; // viz bod D
        int max;

        printf("We are about to find out sorted sequence\n");

        if(first < second){
                min = first;
                max = second;
        } else {
                min = second;
                max = first;
        }
        printf("Sorted values are (%d, %d)\n", min, max);
}
```

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

- *Zdraví* (Health) – určuje kolik útoku šampion vydrží, než zemře,
- *Výdrž* (Stamina) – určuje kolik akcí je schopen šampion vykonat, než se unaví,
- *Mana* (Mana) – reprezentuje magickou energii pro vyvolávání kouzel,
- *Zatížení* (Load) – určuje maximální hmotnost, kterou je šampion schopen unést,
- *Síla* (Strength) – hodnota je používána pro výpočet zranění, síly hodu předmětů a maximální výše zatížení,
- *Obratnost* (Agility) – zvyšuje pravděpodobnost zásahu nepřítele a pomáhá se vyhnout nepřátelským útokům,
- *Moudrost* (Wisdom) – je vlastnost důležitá pro kouzelníky a kněze, určuje rychlost obnovy many,
- *Vitalita* (Vitality) – určuje rychlost obnovy zdraví a výdrže,
- *Odolnost proti magii* (Magic Defense) – snižuje účinnost magických útoků,
- *Odolnost proti ohni* (Fire Defense) – snižuje účinnost ohnivých útoků,
- *Jídlo a Voda* (Food and Water) – pomocí těchto hodnot je obnovována výdrž a zdraví,
- *Štěstí* (Luck) – zvyšuje či snižuje pravděpodobnost provedení akce.

Každý šampion má ve hře čtyři základní dovednosti: 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 je daný šampion zkušený v dané dovednosti, určuje úroveň dovednosti. Tuto 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.

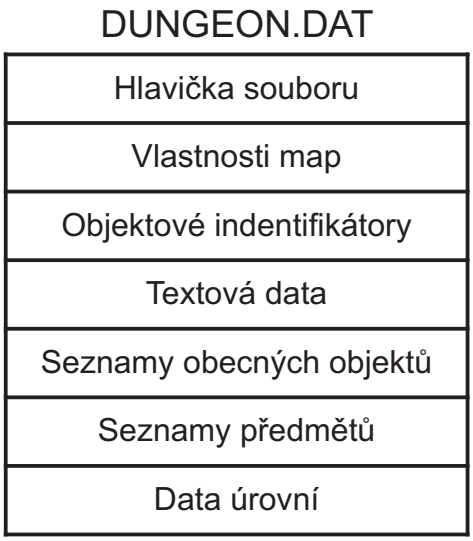
2.1 Data s herními úrovněmi – `DUNGEON.DAT`

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 [7] 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. Formát souboru lze rozdělit do následujících bloků (viz obr. 2.1):

- Hlavička souboru – obsahuje velikost jednotlivých bloků souboru nebo velikost seznamů objektů.
- Vlastnosti herních úrovní – mezi tyto vlastnosti patří 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 čísly, která jsou napevno zabudovaná v kódu hry. Některé dekorace tak mohou mít navíc speciální význam, který je enginem identifikován pouze podle zmíněného číselného identifikátoru. Jedná se například o dekorace s výklenky, které mohou být umístěné na zdech. V tomto případě originální herní engine rozpozná, že lze na dané místo vkládat předměty.
- Textová data – obsahuje všechny texty ve hře.
- Seznamy obecných objektů – mezi tyto objekty patří, dveře, teleporty, popisky na zdech a přepínače.
- Seznamy předmětů – jsou to předměty, které se dají v herních úrovních sbírat a používat.
- Data úrovní – obsahují data pro každou dlaždici v každé úrovni.

Jednotlivé části souboru budou popsány detailněji v následujících sekcích.



Obrázek 2.1: Ilustrace formátu souboru **DUNGEON.DAT**.

2.1.1 Dlaždice (Tiles)

Dlaždice se dělí na několik typů a pomocí přepínačů jim lze zaslat zprávu, na kterou mohou 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:

- *Zeď* (Wall) – zajišťuje zobrazování stěn a nelze na ni vstoupit. Pro každou stěnu (sever, východ, jih, západ – viz obr. 2.2a) 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 *zdi* může obsahovat přepínač, který si může do *zdi* uložit předměty. Pokud má některá strana 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 deaktivační skryje. Datová položka zprávy zde určuje, pro kterou ze stěn *zdi* je zpráva určena.

- *Podlaha* (Floor) – Po těchto dlaždicích se hráč běžně pohybuje se svoji skupinou. Obdobně jako u stran *zdí* může definovat, zda zobrazuje na podlaze náhodnou dekoraci. Podlaha dále může mít nášlapný přepínač. Dlaždice obsahuje čtyři pozice, na které lze pokládat předměty. Tyto pozice vzniknou rozdělením plochy dlaždice na čtvrtiny (viz obr. 2.2b).

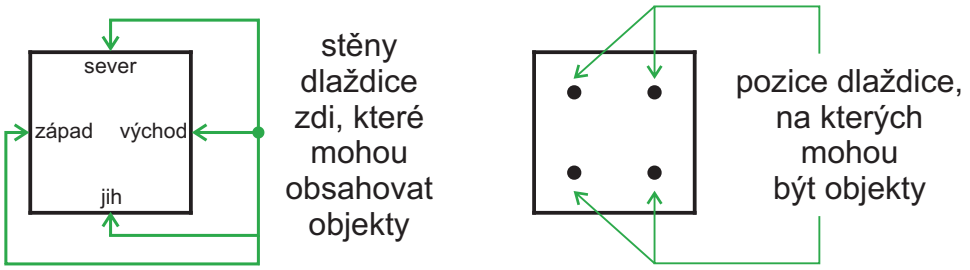
- *Jáma* (Pit) – *Jáma* může být buď otevřená nebo zavřená. Nicméně lze na ni vždy vstoupit, a pokud je otevřená, hráč spadne se svoji skupinou o úroveň níže. Pád může být i přes několik úrovní. Živé objekty pak po dopadu obdrží zranění. Přijmutím aktivační zprávy se jáma otevře a deaktivační se zavře.

- *Schody* (Stairs) – Po této dlaždici se může hráč dostat o úroveň výše resp. níže.

- *Dveře* (Door) – 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 samotné dveře obsahují. Některé dveře lze také rozbít útokem, to si ale definuje již samotný objekt dveří.

- *Teleport* (Teleporter) – 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. deaktivační zprávy aktivuje resp. deaktivuje teleport.

- *Iluze zdi* (Trick wall) – Tato dlaždice může být buď iluze zdi a nebo otevíratelná zeď. V obou případech je vizuálně neodlišitelná od dlaždice typu *zeď*. V prvním případě je možné na dlaždici vstoupit vkročením do zdi. V druhém případě lze zeď odstranit pomocí zaslání odpovídající zprávy na dlaždici. Aktivační zpráva otevře zeď, deaktivační zavře.



(a) Dlaždice zeď – stěny. (b) Dlaždice podlaha – pozice.

Obrázek 2.2: Možné pozice pro uložení objektů na dlaždici.

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

- seznam dlaždic – seřazeny po sloupcích dané úrovně,
- seznam dekorací příšer,
- seznam dekorací zdí,
- seznam dekorací podlah.

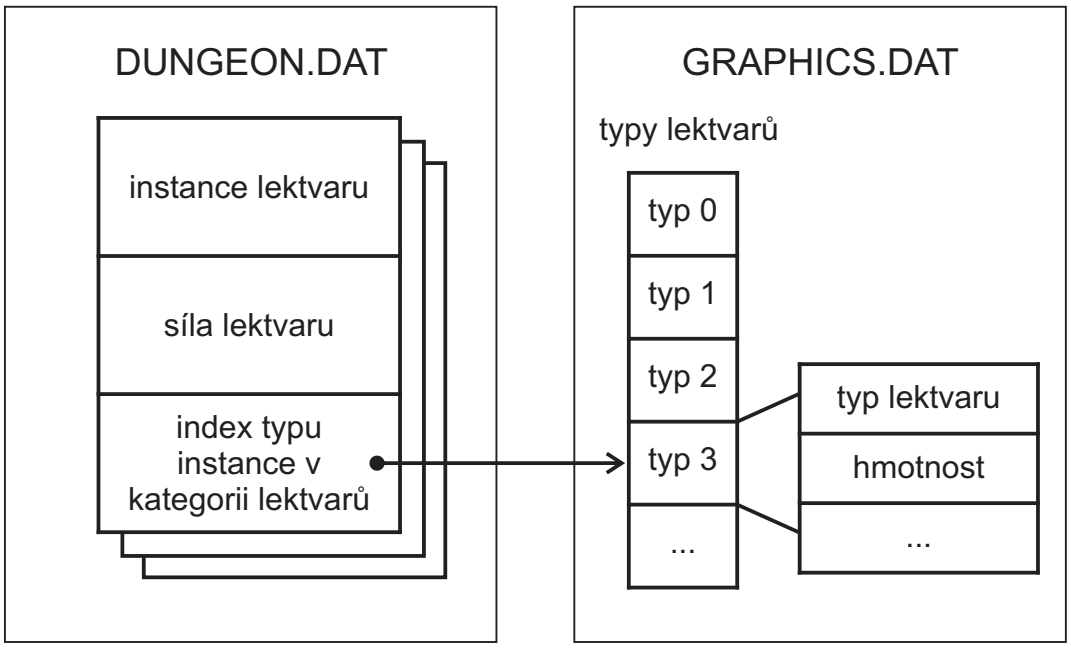
Počty jednotlivých dekorací a dlaždic jsou popsány ve vlastnostech úrovní.

2.1.2 Textová data

V binárním souboru jsou často data uložena po slovech procesoru, které v tomto případě mají velikost dvou bajtů. V této 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 je opět k nalezení v komunitní dokumentaci [7].

2.1.3 Objekty

Tyto objekty uložené v souboru `DUNGEON.DAT` tvoří jednotlivé instance. Odpovídající typy k těmto instancím jsou potom spolu s jejich neměnnými vlastnostmi uloženy v souboru `GRAPHICS.DAT`, jehož obsah si rozebereme v následující sekci 2.2. Tyto typy se pak ještě dělí do kategorií popsaných dále v této sekci. Pro určení konkrétního typu má potom každá instance uložen index typu v dané kategorii (viz obr. 2.3). Objekty reprezentující instance obsahují pouze vlastnosti, které jsou rozdílné pro každou konkrétní instanci.



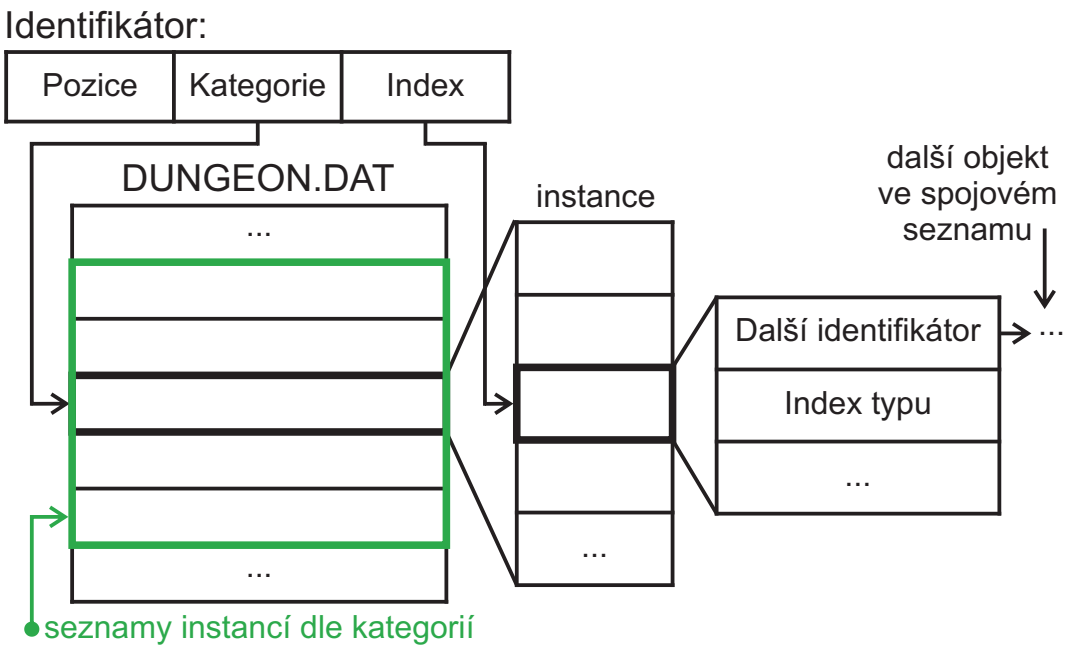
Obrázek 2.3: Ilustrace vztahu instancí a typů.

Identifikátory

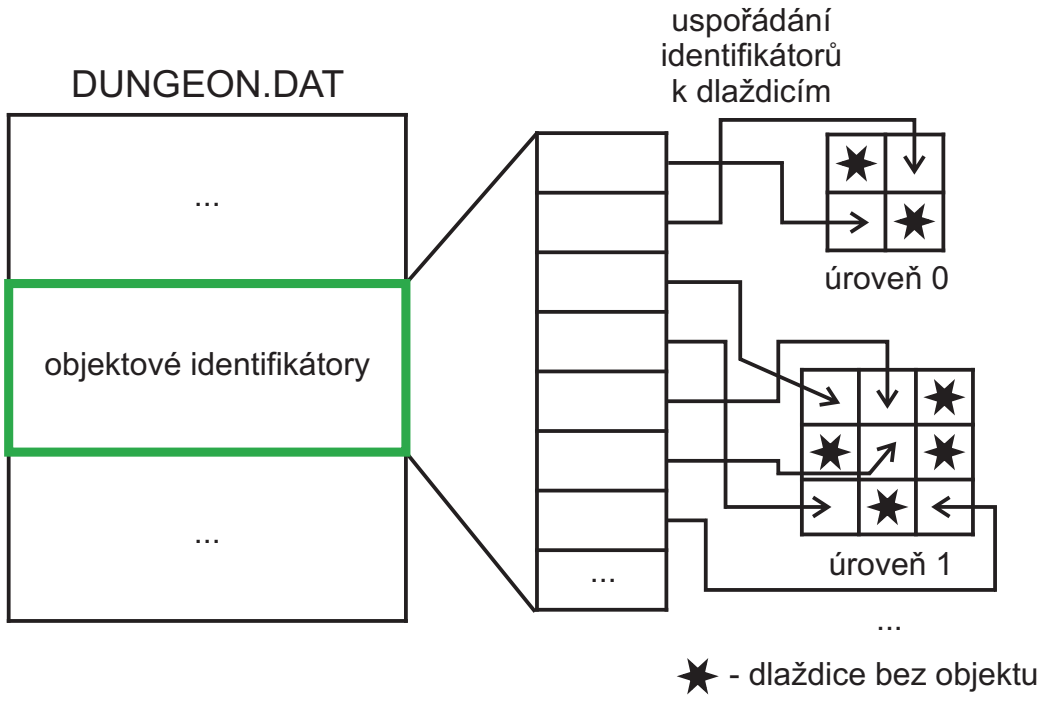
Ke každé instanci objektu existuje unikátní identifikátor, který se skládá z následujících částí:

- Pozice – lze buď interpretovat jako světovou stranu, nebo umístění na dlaždici, jedná-li se o podlahu.
- Kategorie typu objektu – pro každou kategorii existuje v souboru `DUNGEON.DAT` seznam jejich instancí.
- Index – určuje index objektu v seznamu instancí jeho kategorie.

Tyto identifikátory se dají chápat jako reference na konkrétní instance objektů (viz obr. 2.4), pomocí nichž jsou vytvořeny spojové seznamy. Každá dlaždice pak specifikuje, zda může mít spojový seznam. Seznam objektových identifikátorů obsahuje první identifikátory spojových seznamů na dlaždicích, které je mají. Seznam je uspořádaný pro všechny dlaždice od první do poslední úrovně, vždy po sloupcích odshora dolů (viz obr. 2.5).



Obrázek 2.4: Ilustrace obsahu identifikátoru a jeho využití ve spojových seznamech.



Obrázek 2.5: Ilustrace uspořádání prvních objektových identifikátorů dlaždic.

Předměty

Předměty jsou objekty, které lze sbírat, používat nebo ukládat k šampionům. Dělí se do následujících kategorií:

- *Zbraně* (weapons) – lze je vložit šampionům do ruky a ti pak s nimi mohou bojovat.
- *Oblečení* (clothes/armor)– lze jím obléknout šampiony, a tak zvýšit jejich odolnost.
- *Svitky* (scrolls) – obsahují text, který může hráči usnadnit hru.
- *Lektvary* (potions) – dělí se na lektvary modifikující šampionovi schopnosti a na lektvary provádějící nějaké akce – např.: výbuch. Každá instance má vlastnost určující sílu efektu.
- *Kontejnery* (containers) – mohou obsahovat další předměty.
- *Různé* (miscellaneous)– v této kategorii je například jídlo nebo různé předměty, s kterými nelze provádět žádné akce.

Ostatní objekty

Tyto objekty slouží jako pomocné objekty pro dlaždice a dělí se do následujících kategorií:

- *Dveře* (door) – můžou být pouze na dlaždici typu *dveře* a obsahují informace, zda-li jsou dveře rozbitelné a jestli mají tlačítko, kterým je lze otevřít.
- *Teleport* (teleporter) – může být pouze na dlaždici typu *teleport* a definuje cílovou dlaždici a kategorii objektů, které je schopen teleportovat.
- *Textové popisky* (texts) – mohou být pouze na dlaždici typu *zeď* a obsahují odkaz na konkrétní text do textový dat.
- *Skupina nepřátelské entity* (creatures) – definuje skupinu nepřátelských entity na dlaždici, jejich počet, rozmístění na dlaždici a předměty, které lze získat jejich zabitím.
- *Senzory* (sensors/actuators) – vytvářejí základní mechaniky herních úrovní. 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 sekci 2.7.

2.2 Data s vlastnostmi objektů – `GRAPHICS.DAT`

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. K některým částem také existuje komunitní dokumentace [9], nicméně není ucelená jako v případě dokumentace souboru `DUNGEON.DAT`. Zároveň jsou vyextrahovaná data zveřejněna na webu v HTML formátu ([3], [5], [4], [6], [8]). 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 44 akcí a stejný počet komb. Kompletní seznam jednotlivých akcí a komb lze nalézt v komunitní dokumentaci [3]. Zde si popišme alespoň jejich vlastnosti.

Vlastnosti akcí jsou:

- *název* (name) – textový popis akce ve hře,
- *zkušenosti* (experience gain) – počet zkušeností získaných po provedení akce,
- *dovednost* (skill) – identifikátor dovednosti, která získá zkušenosti,
- *obrana* (defense modifier) – modifikátor obrany při používání akce,
- *výdrž* (stamina) – modifikátor výdrže nutné pro provedení akce,
- *pravděpodobnost provedení akce* (hit probability),
- *zranění* (damage) – modifikátor pro výpočet konečného zranění nepřítele,
- *únava* (fatigue) – doba, po kterou nelze provádět žádné akce.

Vlastnosti pro každou akci komba jsou:

- index akce ze seznamu akcí (action index),
- úroveň dovednosti (skill level) – minimální úroveň dovednosti pro úspěšné provedení akce.

2.2.2 Předměty

Ke každému typu předmětu existují v tomto souboru popisovače vlastností [4]. Pro každý předmět jsou definovány následující vlastnosti:

- *globální identifikátor* (global item index) – unikátní identifikátor, který je použit k identifikaci konkrétního typu předmětu například v přepínačích,
- *index útočného komba* (attack combo),
- *lokace* (carry location) – místo, na kterou část těla, nebo do kterého inventáře šampiona lze předmět vložit,
- *index v kategorii* (index in its category) – 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, jejichž hodnoty lze nalézt v komunitní dokumentaci [5].

Zbraně:

- *poškození* (damage) – hodnota zranění aplikovaná při útoku na nepřítele,
- *kinetická energie* (energy) – hodnota udává, jak daleko lze zbraň hodit,
- *střelné poškození* (shoot damage) – hodnota dodatečného poškození u střelných zbraní.

Oblečení:

- *síla brnění* (armor strength) – obrana, kterou oblečení poskytuje,
- *odolnost proti útokům ostrými předměty* (sharp resistance).

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 [6]:

- *Obtížnost* (Difficulty) – modifikátor obtížnosti vyvolání kouzla,
- *Doba trvání* (Duration) – po tuto dobu nemůže šampion vyvolávat další kouzla,
- *Úroveň dovednosti* (Skill level) – postačující úroveň určité dovednosti pro úspěšné 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.

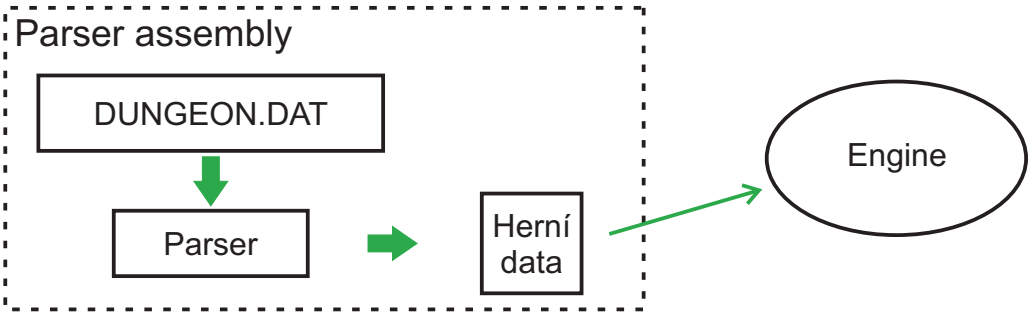
- *Doba pohybu* (Movement duration) – doba, po které se entita rozhodne změnit pozici.
- *Doba útoku* (Attack duration) – doba, po které může entita provést útok.
- *Síla brnění* (Armor) – určuje odolnost proti útokům.
- *Zdraví* (Base health) – hodnota použitá pro výpočet zdraví nových nepřátelských entit.
- *Síla útoku* (Attack power) – hodnota pro výpočet útoku.
- *Otrava* (Poison) – určuje, jak velkou otravu způsobí útok.

- *Obrana* (Defense) – určuje obtížnost pro šampiona, se kterou je entita zraněna.
- *Detekce* (Detection range) – určuje vzdálenost, na kterou detekuje nepřítele.
- *Odolnost proti ohni* (Fire resistance) – odolnost proti kouzlům způsobující ohnivý útok.
- *Odolnost proti jedu* (Poison resistance) – odolnost proti kouzlům způsobující jedový útok.
- *Pravděpodobnosti zranění částí těla* (Wounds) – pravděpodobnost pro každou část těla nepřítele, s kterou ji entita zraní.

Celý seznam vlastností je možné nalézt v komunitní dokumentaci [8].

2.3 Parsování vstupních dat

V době implementace této části práce nebylo jasné, zda nalezená dokumentace [7] bude dostatečná 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 něj nich vytvoří herní úrovně. Toto schéma znázorňuje následující obrázek 2.6.



Obrázek 2.6: Ilustrace vztahu parseru vůči enginu.

2.3.1 Herní úrovně – `DUNGEON.DAT`

Parsování herních úrovní je rozděleno do dvou fází. V první fázi probíhá transformace dat objektů (viz sekce 2.1) ze souboru `DUNGEON.DAT` na odpovídající objekty v jazyce C#. V druhé fázi jsou pak objekty z první fáze upraveny, aby měly 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 sekce 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 sekce 2.1) a seznam jejich dlaždic, dekorací, atd.

Dále parser rozparsuje všechna data vstupního souboru dle komunitní dokumentace [7] a vytvoří z nich adekvátní objekty, které uloží do zmíněného datového objektu. Ilustrace této fáze je na obrázku 2.7.

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 sekce 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 práce enginu s těmito daty jsou v datových objektech inicializovány ještě následující položky:

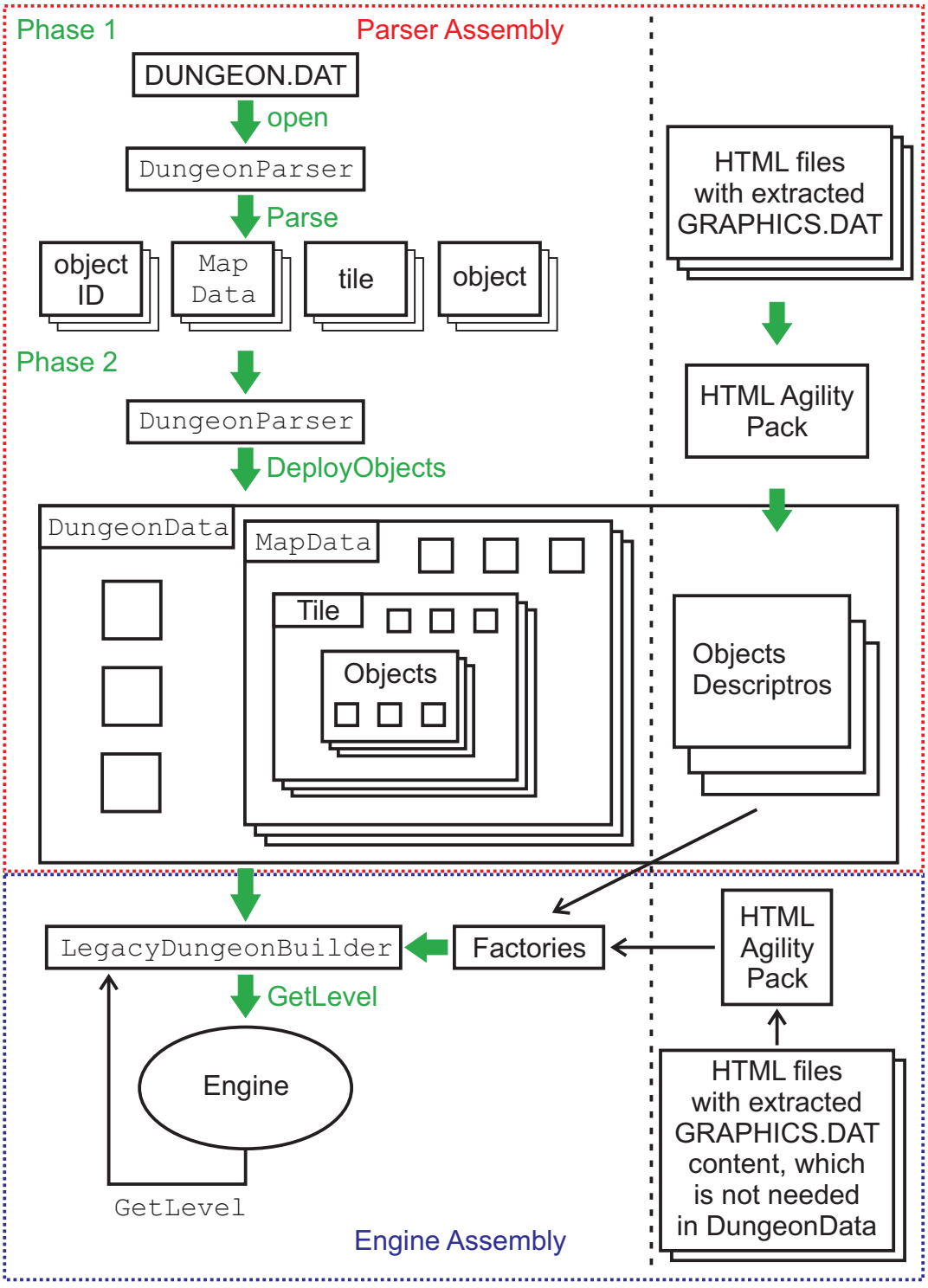
- seznam sbíratelných předmětů,
- seznam senzorů,
- seznam popisků,
- seznam nepřátelských entit,
- položka dveří pro dlaždici typu dveře,
- položka teleport pro dlaždici typu teleport,
- a pro dlaždici typu zeď a podlaha stanoví identifikátory náhodných dekorací.

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 originální hře v souboru `GRAPHICS.DAT`. Nicméně tato data již existují vyextrahovaná v HTML formátu, a proto jsme je použili namísto původních dat. Ke zjednodušení parsování dat z HTML souborů byla použita knihovna HTML Agility Pack [12]. Pro každé typy vlastností jsou pak vytvořeny odpovídající datové objekty, které jsou uloženy do seznamů ve výsledném datovém objektu zmíněného v předchozí sekci 2.3.1. V této části jsou získána data vlastností všech objektů (viz sekce 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 2.7.



Obrázek 2.7: Průběh parsování herních dat.

2.4 Výběr frameworku pro práci s grafickým výstupem

Po úspěšném rozparsování herní dat se bylo třeba rozhodnout, kterou knihovnou použít pro zobrazení grafického výstupu enginu. Aby se bylo možné soustředit hlavně na samotný návrh enginu (viz cíl práce **C3**), použili jsme framework, se kterým už jsme měli nějaké zkušenosti. Jedná se o XNA Framework od firmy Microsoft. Nicméně tento framework již není firmou nadále podporován a vyvíjen, proto jsme nakonec využili klon MonoGame [2]. Výhodou této volby je také to, že framework je multiplatformní. Jeho tvůrci tvrdí, že podporuje platformy iOS, Android, MacOS, Linux, Windows, OUYA, PS4, PSVita, Xbox One.

2.5 Reprezentace jádra enginu

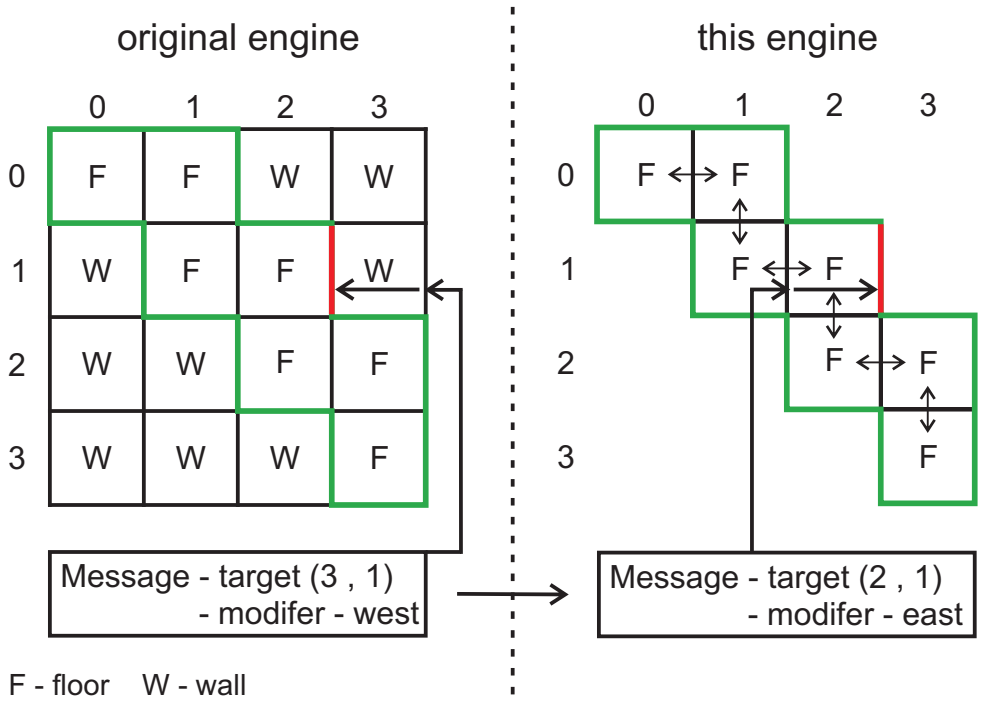
O samotnou herní smyčku se stará framework MonoGame, který na jádru enginu volá aktualizační a vykreslovací obsluhy. Jádro enginu obsahuje následující tři důležité komponenty:

- Hráč – tato komponenta zajišťuje interakci s uživatelem. Skrze ni lze ovládat hráčovu skupinu šampionů, sbírat předměty nebo bojovat.
- Builder herních úrovní – tato komponenta zajišťuje sestavování herních úrovní.
- Objekt obsahující factories – pro každý typ popsaný v sekci 2.2 existuje odpovídající factory obsahující jeho vlastnosti. Factories mohou vytvářet objekty, jejichž typ reprezentují. Toto bude konkrétně specifikováno později (viz sekce 2.15), nicméně důležité je, že skrze jádro enginu lze k těmto factories přistupovat.

Podrobnosti o jádře enginu jsou ve vývojové dokumentaci (viz sekce 2.5).

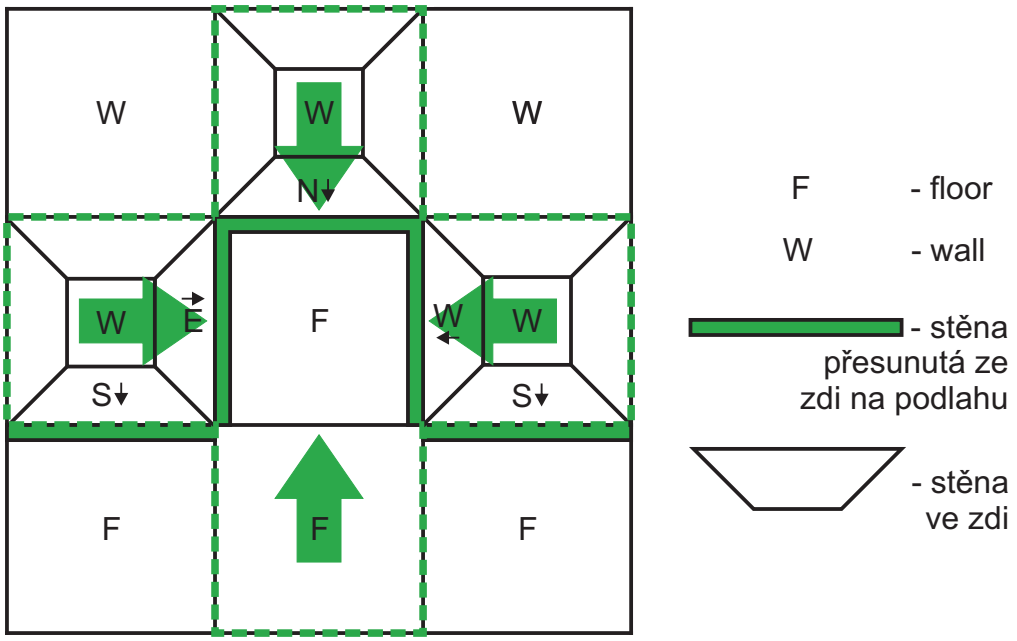
2.6 Reprezentace dlaždic

V originální hře jsou dlaždice vždy uloženy v obdélníkovém poli (viz sekce 2.1 a obr 2.8 vlevo). To znamená, že dlaždice, které nejsou využity pro chodby, jsou vždy vyplněny zdmi. Jelikož jedním z cílů této práce je udělat engine co nejrozšiřitelnější (viz cíl práce **C3**), rozhodli jsme se tento problém vyřešit o něco obecněji (viz obr. 2.8 vpravo). Každá dlaždice je vrchol grafu, a pokud jsou dvě dlaždice sousední, jsou mezi nimi hrany oběma směry, to dohromady tvoří obousměrně orientovaný graf. Nicméně, pro dlaždice stále platí, že mají nanejvýš čtyři sousedy a jsou uspořádané do mřížky. Tato reprezentace pro hodně řídké mapy může ušetřit paměť. Avšak důležitější je, že při takové reprezentaci lze jednoduše získat sousedy pouze ze znalosti konkrétní dlaždice. V této podobě by bylo navíc jednoduší engine upravit tak, aby dlaždice nemusely být na mřížce a aby mohly mít více sousedů.



Obrázek 2.8: Uložení dlaždic v originálním vs. tomto enginu.

Předchozí rozhodnutí také vede k tomu, že již nejsou potřeba dlaždice typu zeď (viz obr. 2.8), jelikož její stěny mohou být definovány ve dlaždicích typu podlaha (viz obr. 2.9). Při vytváření podlahy je pak nutné číst i její sousední dlaždice. Pokud je soused daným směrem zeď, vytvoří se na této straně stěna a soused není nastaven. V opačném případě je soused nastaven na odpovídající dlaždici. Zprávy původně cílené zdím je také třeba posílat na odpovídající dlaždice podlahy (viz obr. 2.8). Tato reprezentace zdí opět vede k vyšší flexibilitě, jelikož případné místo využité původními dlaždicemi typu zdi lze využít jiným způsobem.



Obrázek 2.9: Ilustrace přesunu stěn do dlaždice *podlahy*

Pokud jsou tedy stěny součástí dlaždic, je třeba vyřešit jejich reprezentaci. Buď je možné, aby veškeré funkce stěn obsahovala přímo dlaždice, a nebo je možné tyto části delegovat do zvláštních objektů. Ze začátku byl v enginu použit první zmiňovaný způsob. Nicméně tato reprezentace není příliš vhodná, protože potom samotná dlaždice řeší funkce, které k ní přímo nenáleží. Nakonec tato reprezentace byla změněna a v enginu byla použita druhá zmiňovaná možnost. Vedlo k ní zejména oddělení zobrazovací vrstvy od logické. Výhoda tohoto přístupu byla kromě samotného oddělení kódu i možnost znovu použití kódu za pomocí dědičnosti. Například třída reprezentující stěnu, která obsahuje přepínač, může dědit ze třídy reprezentující prázdné stěny. Avšak ukázala se i nevýhoda tohoto přístupu, a to, že komunikace směrem ze stěny k dlaždici je trochu těžkopádná. Buď by bylo zapotřebí předat zdi referenci na jejich rodičovské dlaždice, nebo by se musely použít události. Pokud to bylo v některých případech nutné, přiklonili jsme se k použití událostí.

2.7 Přepínače

Přepínače tvoří základní mechaniky v herních úrovních Dungeon Masteru, mohou být buď na stěnách, nebo na podlaze. Na stěnách je lze aktivovat kliknutím myši a na podlaze potom přesunutím hráče na dlaždici s přepínačem. Aktivace ještě může být podmíněna dalšími faktory: obsah daného typu předmětu v ruce nebo směr, kterým je hráč otočen.

Ve skutečnosti se každý přepínač skládá z řady senzorů a platí, že:

- poslední senzor určuje dekoraci přepínače,
- při pokusu o aktivaci přepínače dojde postupně k pokusu aktivace každého senzoru,

- každý senzor může definovat akci, kterou provede, pokud byl aktivován.

Senzor má následující vlastnost:

- Typ (type) – číselné označení, dle kterého originální herní engine určí způsob aktivace.
- Akce (action type) – může být buď lokální, nebo vzdálená. Pro lokální akce je to zarotování sekvencí senzorů přepínače nebo přidání zkušeností hráči. Pro vzdálené akce je to odeslání zprávy (viz sekce 2.1) na danou cílovou dlaždici.
- Data (data) – pro každý typ senzoru se může obsah lišit.
- Zpoždění (action delay) – doba, po které se provede akce. Jedna jednotka tohoto času je rovna 1/6 sekundy.
- Opačný efekt (revert effect) – liší se dle typu senzoru, nicméně většinou otáčí podmínku aktivace pro daný typ senzoru.
- Opakovatelnost (once only) – určuje, zda lze senzor aktivovat neomezeně či pouze jednou.
- Dekorace (decoration) – číselný identifikátor dekorace, jehož hodnota mínus jedna slouží jako index do právě používaných dekorací, které jsou definovány pro každou herní úroveň. Dekorace s číslem nula potom označuje, že nemá být použita žádná dekorace.

Výše popsaný systém přepínačů přináší hned několik problémů:

P1 Z technické dokumentace [7] není vždy jasně zřejmé, jaká je aktivační podmínka senzoru pro určitý typ.

P2 Jakým způsobem efektivně a přehledně rozparsovat data senzorů, když mají tolik nezávislých vlastností.

P3 Jak reprezentovat přepínače v novém enginu.

P4 Jakým způsobem provádět vzdálené akce.

2.7.1 Příklady

Pro ujasnění celého konceptu přepínačů si pojďme ukázat několik příkladů. Textový popis aktivační podmínky senzorů pro jednotlivé jejich typy může být nalezen v komunitní dokumentaci [7]. Přesná funkce senzorů pak může být nalezna v dekompilovaných zdrojových kódech [10] v následujících funkcích, které jsou v souboru `SENSOR.C`:

- `F275_aszz_SENSOR_IsTriggeredByClickOnWall` – chování pro senzory na zdech,
- `F276_qzzz_SENSOR_ProcessThingAdditionOrRemoval` – chování pro senzory na podlaze.

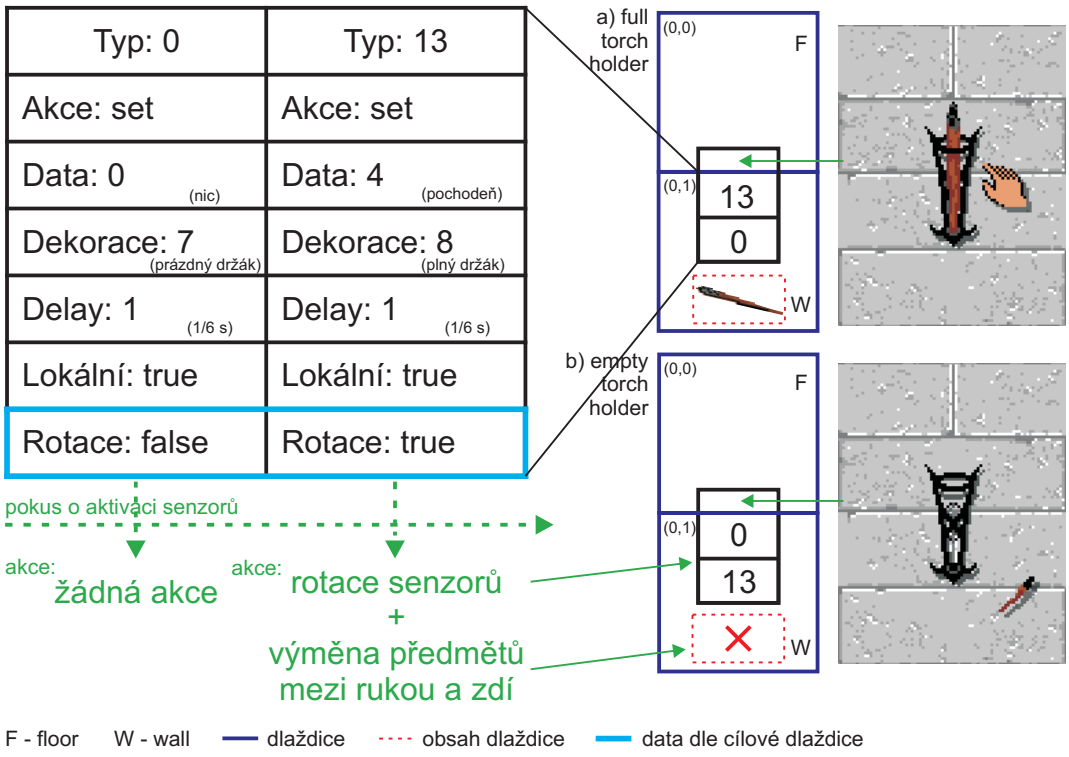
Držák na pochodně

Držák na pochodně (viz obr. 2.10) se skládá ze dvou senzorů, jehož data a umístění je znázorněno na obrázku 2.10. Dlaždice na pozici (0, 0) je typu *podlaha*, takže na ní může být hráč a může kliknout na přepínač na stěně dlaždici *zdi* na pozici (1, 0).

Senzor na první pozici je typu 0, což znamená, že tento senzor nelze aktivovat. Senzor může pouze zobrazovat svoji dekoraci, pokud je na posledním místě sekvence, ostatní jeho vlastnosti nejsou využity. Dekorace má hodnotu 7, což znamená, že bude použita dekorace stěn pro danou herní úroveň s indexem 6, která v tomto příkladu reprezentuje dekoraci prázdného držáku na pochodně.

Senzor na druhé pozici je typu 13, což znamená, že tento senzor lze aktivovat buď pokud má hráč prázdnou ruku a ve zdi je uložen předmět, jehož globální identifikátor je roven položce data, nebo pokud má hráč v ruce předmět s globálním identifikátorem předmětu rovným položce data a zeď neobsahuje žádný předmět. Položka data s číslem 4 reprezentuje globální identifikátor předmětu pochodně. Položku akce tento senzor nevyužívá. Dekorace s číslem 8 potom reprezentuje plný držák na pochodně. Zpoždění má v tomto případě hodnotu 1, což znamená, že se akce provede 1/6 sekundy po aktivaci senzoru. Akce senzoru je lokální, což znamená, že po aktivaci senzoru může být provedena rotace senzorů v sekvenci přepínače nebo přidání zkušenosti hráči. V tomto případě se provádí rotace senzorů. Veškeré akce přepínače v tomto příkladě dělá tento druhý senzor.

Při aktivaci přepínače se tedy vezme pochodeň uložená ve zdi a vloží se hráči do ruky. Následně se provede rotace senzorů, čímž se zobrazí dekorace prázdného držáku na pochodně. Při opětovné aktivaci přepínače se vloží pochodeň z ruky hráče zpět do zdi. A následuje opět rotace senzorů, čímž se zobrazí opět dekorace plného držáku na pochodně.



Obrázek 2.10: Ilustrace přepínače pro držák pochodní.

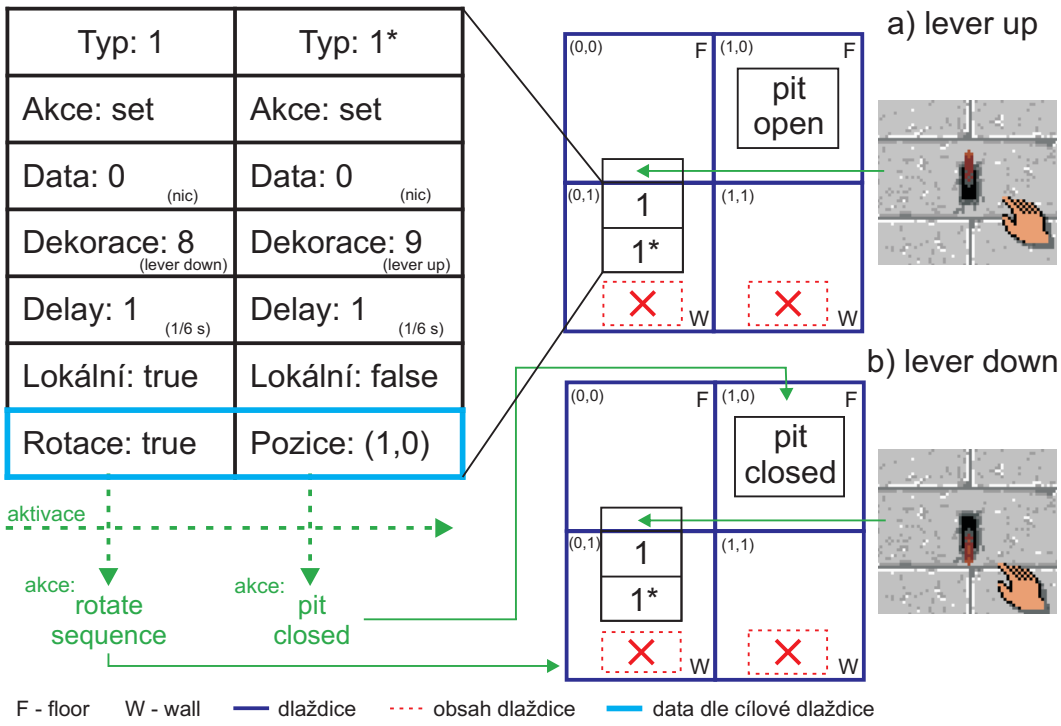
Páka

Páku tvoří dva senzory stejného typu 1, jejichž data a umístění je znázorněno na obrázku 2.11. Dlaždice na pozici (0, 0) je typu *podlaha*, takže na ní může hráč vstoupit a kliknout na přepínač na stěně dlaždice *zdi* na pozici (0, 1). Na pozici (1, 0) je potom otevřená dlaždice typu *jáma*, na kterou nelze vstoupit, aniž by do ní hráč spadl. Senzor typu 1 lze potom aktivovat nezávisle na tom co má hráč v ruce. Položku data tento senzor nepoužívá.

První senzor má dekoraci číslo 8, která v tomto příkladě reprezentuje páku nahoru. Zpoždění má hodnotu 1, což znamená, že se akce senzoru provede po 1/6 sekundy. Akce je pro tento senzor lokální, což znamená, že se nevyužije položka akce, ale provede se rotace senzorů, jelikož je nastavena položka rotace.

Druhý senzor má dekoraci 9, která reprezentuje v tomto příkladě dekoraci s pákou nahoře. Akce je nastavena na nelokální, takže se provede vzdálená akce, jejíž pozici určuje položka pozice tj. (1, 0). Položka akce pak reprezentuje přehození stavu, které se provede opět 1/6 sekundy od aktivaci.

Po aktivaci přepínače tedy nejprve dojde k rotaci senzorů způsobené prvním senzorem – takže se zobrazí dekorace páky dole. Se zpožděním 1/6 sekundy pak druhý senzor odešle *jámě* na pozici (1, 0) zprávu, aby přehodila její stav – což znamená, že se jáma zavře. Opětovná aktivace přepínače provede opět rotaci senzorů, takže se zobrazí znovu dekorace páky nahoře. Druhý senzor následně odešle *jámě* zprávu o změně stavu, takže se *jáma* opět otevře.

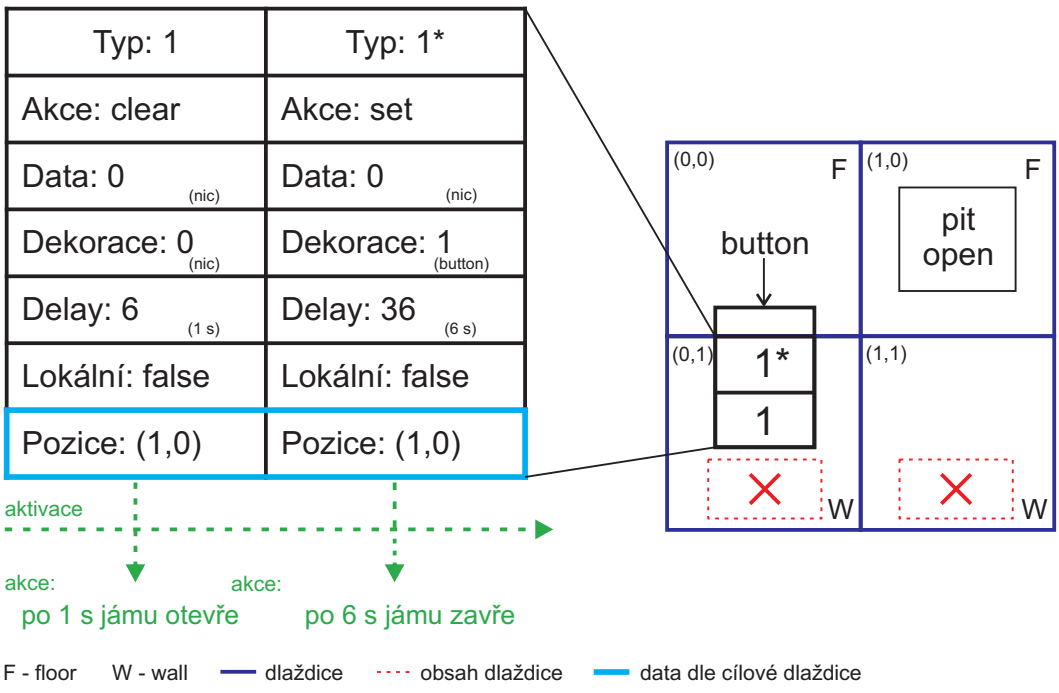


Obrázek 2.11: Ilustrace přepínače reprezentující páku.

Tlačítko

Toto tlačítko se skládá opět ze dvou senzorů typu 1 a rozmístění je analogické jako v předchozím příkladu (viz obr. 2.12). Oba senzory potom mají vzdálenou akci na dlaždici (1, 0) – první senzor vyšle deaktivační zprávu a druhý aktivační (viz obr. 2.12). První senzor nemá nastavenou dekoraci a druhý senzor má dekoraci s číslem 1, která reprezentuje tlačítko. Akce prvního senzoru se provede se zpožděním jedné sekundy a akce druhého se zpožděním šesti sekund.

Po aktivaci přepínače je tedy skrze první senzor odeslána deaktivační zpráva na dlaždici typu *jáma* se zpožděním jedné sekundy. Po obdržení této zprávy se jáma zavře a hráč na ni tak může vstoupit. Druhý senzor poté po šesti sekundách provede aktivaci dlaždice typu *jáma*, která povede na její opětovné otevření.



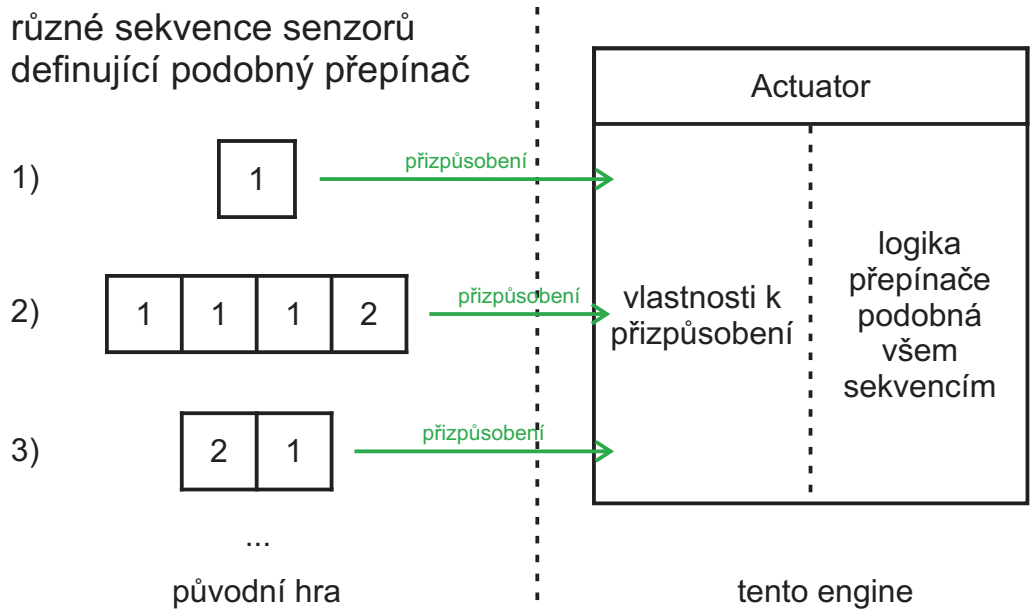
Obrázek 2.12: Ilustrace přepínače reprezentující dočasné tlačítko.

2.7.2 Reprezentace přepínačů

První tři body problémů (viz **P1**, **P2**, **P3**) řeší tato sekce.

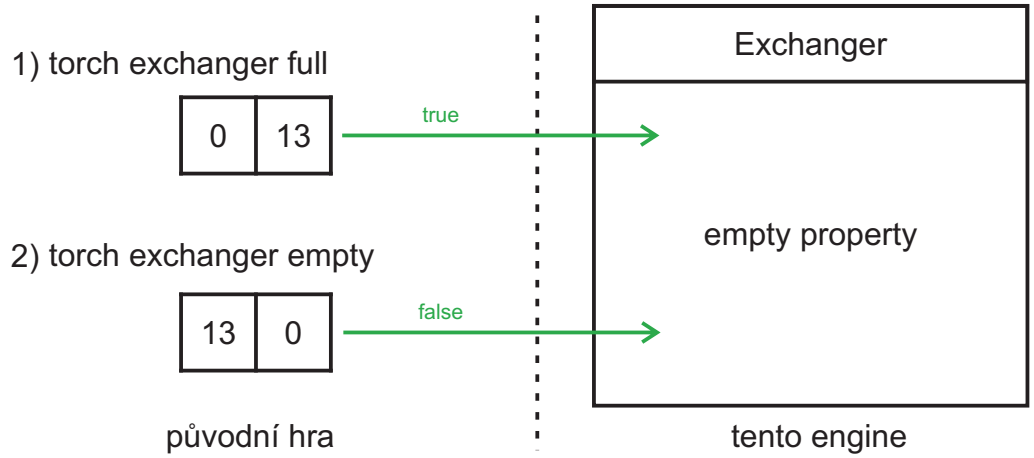
První reprezentace

Jako první způsob jsme se rozhodli vždy pro několik sekvencí senzorů vytvořit objekt, který měl stejné nebo o něco obecnější vlastnosti než dané sekvence. Z počátku se totiž zdálo, že jen výjimečně jsou sekvence senzorů větší či složitější. Proto jsme se rozhodli vytvořit v novém enginu objekty, které mají obecnější vlastnosti, a tak mohou být použité pro několik podobných sekvencí. Dle určité sekvence senzorů se poté objektu inicializují jeho vlastnosti. Celou situaci ilustruje obr. 2.13.



Obrázek 2.13: Ilustrace transformace sekvence senzorů na objekt přepínač.

Konkrétním příkladem může být objekt přepínače reprezentující držák na pochodně (viz obr. 2.14). Kdy první sekvence reprezentuje držák s pochodní a druhá držák bez ní – při vytváření objektu držáku se pak specifikuje, zda má či nemá pochodeň.



Obrázek 2.14: Ilustrace transformace sekvence senzorů na objekt přepínač.

Tento způsob sice řeší problém samotné reprezentace (viz bod **P3**), ale později se ukázalo, že velice nevhodně. Například kód převádějící sekvence na přepínače je složitý a nepřehledný. Bylo také složité vytvořit přepínač podle technické dokumentace [7] tak, aby fungoval správně. Popis v dokumentaci [7] také není dostatečně přesný pro vytvoření tohoto úkolu. I proto čím více přepínačů jsme tímto způsobem naimplmentovali, tím více se ukazovalo problémů. Nakonec jsme dospěli k závěru, že tento způsob reprezentace je nemožný.

Druhá reprezentace

Tato reprezentace se snažila vyřešit problém nepřehlednosti kódu(viz bod **P2**) převádějícího sekvence na objekty přepínačů. Proto je i zde použit typ objektu přepínač z první reprezentace v sekci 2.7.2. Parsování sekvence senzorů jsme se rozhodli udělat pomocí konečného automatu. Tedy tak, že pro každý objekt přepínače existovaly předdefinované sekvence senzorů. Konečný automat potom pomocí vstupní sekvence identifikoval výsledný objekt přepínače. Při inicializaci objektu přepínače zde pak byla sekvence jasně definovaná, což řešilo problém s přehledností (viz bod **P2**). Později se však ukázalo, že i malá změna v sekvenci senzorů může generovat podobný objekt. Takže by bylo třeba spoustu předdefinovaných šablon, které by se lišily ve drobnostech. Tím se ukázalo, že tento přístup není dostatečně obecný a nelze jej tedy použít.

Třetí reprezentace – výsledná

Tato reprezentace se oproti předchozím snažila přiblížit co nejvíce k systému použitém v enginu originální hry. K dosažení tohoto cíle jsme se rozhodli použít dekompilované zdrojové kódy [10] hry. Tím je vyřešen problém nedostatečné dokumentace ve věci přepínačů(viz bod **P1**).

Vytvořili jsme tedy objekt přepínač, který má v sobě sekvenci senzorů tak, jako tomu je v originální hře. Tím je vyřešen i problém nepřehlednosti převodního kódu(viz bod **P2**), jelikož jsou tyto reprezentace téměř identické. S využitím zdrojových kódů jsme vytvořili odpovídající objektově orientovaný kód v jazyce C#. Tato reprezentace tak zajišťuje maximální korektnost implementace a přitom poskytuje možnost rozšíření – což je jedním z cílů této práce (viz bod **C3**). Je tedy možné vytvořit nové senzory, které budou mít vlastní aktivační podmínky a ty následně použít v přepínačích. V originálním enginu je toto rozšiřovaní značně omezené, jednotlivé typy senzorů se tam odlišují jednobajtovým číselným identifikátorem. Ale jelikož jsme nechtěli rozšiřování enginu limitovat pouze na tento způsob vytváření senzorů, je možné si vytvořit přepínač, který bude fungovat na úplně jiné bázi. Takže případné nové přepínače nemusí vůbec senzory používat.

2.7.3 Reprezentace zprávy přepínače

Jak již bylo řečeno (viz sekce 2.1), celý systém mechanik herních úrovní je závislý na posílání zpráv. V originální hře je cíl odeslané zprávy vázán na její souřadnice dlaždice. Jelikož jsme se v enginu nechtěli vázat na tyto pevné souřadnice, rozhodli jsem se namísto nich cílovou dlaždici identifikovat její referencí. To by případně ulehčilo práci při transformaci enginu tak, aby dlaždice mohly mít více sousedů nebo aby nemusely být na pevné mřížce (viz sekce 2.6). Při takovéto reprezentaci však nastává problém s inicializací dlaždic a přepínačů (viz sekce 2.13).

V originální hře lze mezi dlaždicemi posílat pouze jeden typ zprávy. Jelikož cílem této práce je udělat engine co nejrozšiřitelnější (viz bod **C3**), engine poskytuje za určitých podmínek používat i jiné typy zpráv. Těmito podmínkami jsou:

- Vlastní zprávy lze zasílat pouze vlastně vytvořeným dlaždicím, které na daný typ zprávy musí být připraveny.

- Všechny dlaždice musí umět přijímat originální typ zprávy – nicméně nemusí na ně reagovat.
- Třída reprezentující vlastní zprávu musí být potomkem originální zprávy v hiearchii dědičnosti jazyka C#.

2.8 Reprezentace entit

V originální hře existují pouze dva typy živých objektů, tj. nepřátelské entity a šampioni. Za účelem udělat engine co nejrozšiřitelnější (viz cíl práce **C3**) jsme se rozhodli rozdělit entity na živé a neživé. Entitou je v tomto enginu každý objekt, s jehož vlastnostmi může interagovat nějaká akce (viz sekce 2.2.1). Například akce útok může entitu zranit nebo poškodit (pokud je neživá). Příkladem použité neživé entity jsou dveře, které je možné rozbít útokem. Živé entity disponují následujícími atributy:

- vlastnosti – jsou to vlastnosti jako zdraví, odolnost, atd. (viz sekce 2),
- dovednosti – to ve výsledku znamená, že jsou schopny používat akce,
- relaci vůči ostatním živým entitám – určuje, zda jsou vůči daným entitám přátelské či nepřátelské,
- části těla a inventáře,
- poloha na dlaždici – definuje, jaké části dlaždic mohou zaujímat.

Každý z těchto bodů lze rozšířit, což může být užitečné v případě použití enginu pro implementaci jiné hry než Dungeon Master.

2.8.1 Vlastnosti

V původní hře jsou vlastnosti šampionů (viz sekce 2) a nepřátelských entit (viz sekce 2.2.4) pevně definované. Několik vlastností nepřátelských entit je odlišných od tě šampionových, nicméně ve většině se shodují. S těmito odlišnostmi musí počítat akce, které modifikují vlastnosti entit. Pokud entita nějakou vlastností nedisponuje, je vrácena vlastnost se základní hodnotou. Například pokud provádíme akci útok magickou ohnivou koulí a entita nemá žádnou z vlastností odolnost proti magii či odolnost proti ohni, výsledný útok akce nebude nijak modifikován.

2.8.2 Dovednosti

V originální hře mají šampioni pevně definované dovednosti (viz sekce 2), jako je tomu v případě vlastností. I v tomto bodě se nový engine snaží o větší dynamičnost. Opět platí, že každá entita nemusí mít všechny dovednosti. Pokud je po entitě vyžadována dovednost, kterou entita nedisponuje, je její úroveň nulová. V originální hře jsou dva typy dovedností: základní a skryté. Pokud některá akce vylepšuje skrytou dovednost, rozdělí se získané zkušenosti mezi obě dovednosti (viz sekce 2). Tento koncept schopností není v enginu vyžadován a záleží potom na

konkrétní implementaci dovednosti. Ta také specifikuje množství zkušeností nutné pro získání nových úrovní. Implementace dovedností pro hru Dungeon Master odpovídá té ve zdrojových kódech hry [10], konkrétně v následujících funkcích v souboru `CHAMPION.C`:

- `F304_apzz_CHAMPION_AddSkillExperience` – přidání zkušeností dané dovednosti.
- `F303_AA09_CHAMPION_GetSkillLevel` – získání úrovně dané dovednosti.

2.8.3 Relace mezi entitami

Další funkcí, kterou oproti originálu engine umožňuje, je definovat pro entity jejich nepřátele. 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 hře pouze dva relační tokeny – první pro šampiony a druhý pro nepřátelské entity. Nicméně celý tento systém je navržen právě pro stanovení obecnějších vztahů mezi entitami a to může být aplikováno v libovolném rozšíření hry.

2.8.4 Tělo a inventáře

Každá živá entita má definované 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 předmětů, ty potom mají nadefinováno s jakým typem úložiště jsou kompatibilní. Tato reprezentace principiálně umožňuje podporu následujícího příkladu. Pokud máme lidské a medvědí nohy, tak lidské kalhoty by měly jít nasadit pouze na lidské nohy, nikoliv na medvědí. Naopak medvědí chrániče na nohy půjdou nasadit pouze na nohy medvědí. Takto pokročilejší mechanismy v originální hře nejsou použity, předměty tam mohou sbírat pouze šampioni. Nicméně tyto mechanismy jsou k dispozici při použití tohoto enginu k implementaci jiné hry.

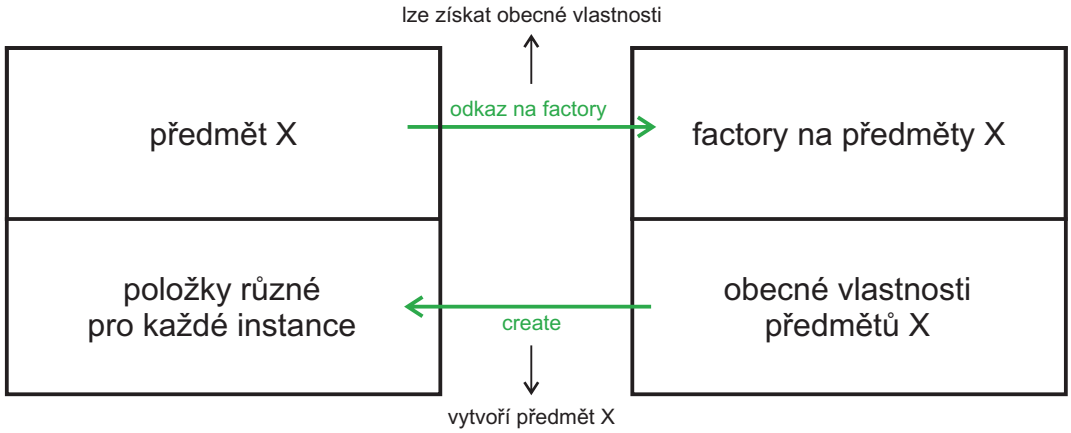
Předměty lze pak dále ukládat do dalších externích úložných prostorů. Mezi taktové úložiště patří například batoh, kapsa, truhla, toulec atd. O velikosti a typu těchto úložišť opět rozhoduje tvůrce konkretních entit. Avšak stejně jako v originální hře, i tato instance má implementované úložiště pouze pro šampiony.

2.9 Reprezentace předmětů

V herních úrovních jsou různě rozmístěné předměty, které lze sbírat a ukládat si je do inventářů šampionů. Každý takový předmět náleží do nějaké kategorie (zbraň, lektvar, atd. – viz sekce 2.1.3). Všechny tyto kategorie obsahují několik typů předmětů. Pro každý typ předmětu pak existuje jednoznačný globální identifikátor (viz sekce 2.2.2). Identifikátory jsou použité například v přepínačích pro identifikaci typu předmětu, dle kterého se pak rozhodnou, zda se mají aktivovat či nikoliv. Platí tedy, že všechny instance daného typu předmětu mají stejný identifikátor.

Z počátku jsme pro stanovení identifikátorů zvolili stejnou strategii jako tvůrci originální hry, tedy každá instance měla daný číselný identifikátor. Nicméně takto zvolený způsob reprezentace identifikátorů by byl nepřehledný pro případné rozšiřitele, jelikož by nemuselo být jasné, které identifikátory už jsou obsazené a které nikoliv. Dále v počátcích každá instance předmětu měla všechny jeho vlastnosti a to i ty, které byly společné pro všechny instance daného typu (viz sekce 2.2.2).

Pro splnění cíle **C3** této práce bylo zapotřebí reprezentaci předmětů vylepšit. Řešením problému bylo delegování společných vlastností předmětů do zvláštních tříd (viz obr. 2.15), které odpovídají jednotlivým identifikátorům. Tyto třídy navíc slouží jako factories na předměty daného typu. Reference na konkrétní instanci třídy pak slouží jako identifikátor. To ale znamená, že reference na tyto factories jsou potřeba pro vytváření přepínačů. Z toho důvodu je součástí jádra enginu objekt obsahující všechny factories splňující tento vzor (viz sekce 2.5).



Obrázek 2.15: Ilustrace vztahu objektů a jejich factories.

Každá factory na předměty má také následující atributy:

- hmotnost,
- jméno,
- kombo – definuje akce, které lze s předměty provádět,
- definice míst v inventáři, kam lze předmět uložit.

2.10 Reprezentace komb a akcí

Každý typ akce definuje svoji factory, která je uložena v jádře enginu (viz sekce 2.5) – jako je tomu v případě předmětů. Tyto factories obsahují vlastnosti o akci (viz sekce 2.2.1), které také určují požadavky pro provedení dané akce. Pomocí factory lze pak vytvořit samotný objekt reprezentující konkrétní akci, kterou lze následně vyvolat určitým směrem (sever, východ, jih, západ). Jak podmínky vytvoření akce, tak její provedení je naimplementováno dle dekompilovaných zdrojových kódů [10] originální hry. Každá factory na předměty má potom specifikovaný seznam typů akcí, které lze s předměty provádět.

2.11 Reprezentace kouzel

V jádře enginu jsou uloženy všechny symboly, včetně power symbolů (viz sekce 2.2.3). Tyto symboly mají definovány, kolik je třeba many pro jejich vyvolání při daném power levelu. Každé kouzlo potom definuje svoji factory, která je rovněž uložena v jádře enginu (viz sekce 2.5) – stejně jako je tomu v případě předmětů a akcí. Objekt reprezentující kouzlo lze pak opět vytvořit pomocí jeho factory. Pro vyvolání kouzla je potom třeba určit směr a entitu, která kouzlo vyvolává. Vyvolávání a provádění kouzel je opět naimplementováno dle dekompilovaných zdrojových kódů [10] originální hry. Modifikace vlastnosti entity při vyvolávání symbolů kouzla pak zajišťuje speciální manager.

2.12 Builder herních úrovní

Aby bylo možné sestavovat herní úrovně z různých vstupních formátů (viz cíl práce **C4**), bylo nutné tuto část vyčlenit do samostatného objektu, který je pak předán jádru enginu (viz sekce 2.5). Tento objekt budeme dále nazývat builder. Jádro enginu po builderu vyžaduje vytvoření herních úrovní, přičemž mu předá sadu factories – která je rovněž uložena v jádře. Builder by se měl také starat o kešování načtených úrovní, jelikož je engine po něm může požadovat několikrát. Objekt builderu je předán jádru enginu při jeho vytváření, tímto způsobem je možné jádru předat vlastní implementaci builderu.

Tato práce obsahuje pouze jeden builder, který je schopný sestavit herní úrovně z datového objektu popsaném v sekci 2.3. V sekci 2.6 se došlo k závěru, že dlaždice zdí nebudou v tomto enginu existovat. Z toho důvodu jsme se v prvních fázích projektu rozhodli při vytváření herní úrovně procházet pouze dlaždice, které jsou od pozice hráče přístupné. Toho bylo docíleno použitím algoritmu prohledávání do hloubky. Nicméně později se ukázalo, že některé nepřístupné části herní úrovně jsou používané teleporty nebo přepínači. Z toho důvodu se od tohoto způsobu procházení dat dlaždic upustilo a nyní se procházejí postupně všechny dlaždice.

Pro transformování dat na objekty srozumitelné enginu se používají ještě další pomocné subbuildery. Samotné sestavování herní úrovně potom probíhá následovně. V cyklu jsou vytvářeny všechny dlaždice – přitom se za pomocí factories z jádra enginu provádí vytváření veškerých objektů, které dlaždice obsahuje.

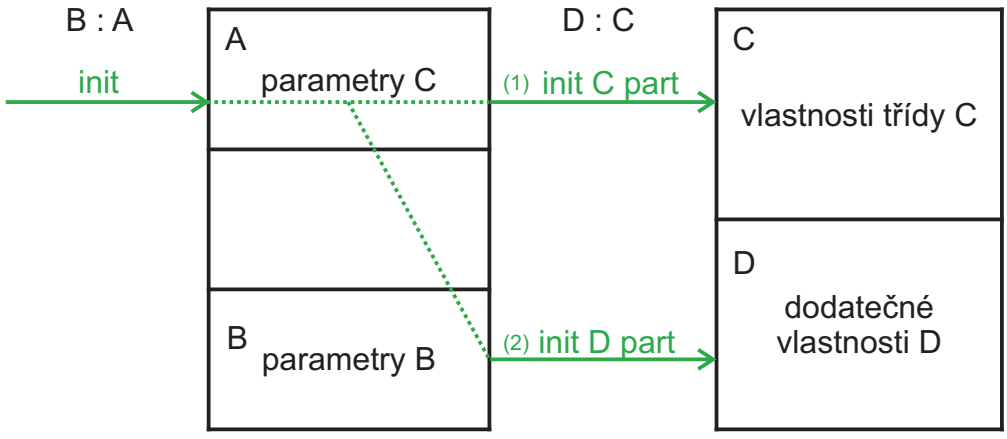
2.13 Inicializace objektů

Tento projet si klade za cíl (viz cíl práce **C3**) vytvořit co nejlepší objektový návrh. Jedním z faktorů, které by měl takový návrh zahrnovat je, že veřejné budou jen nezbytně nutné položky tříd. Na to navazuje problém s inicializací takových tříd. Například přepínače potřebují referenci na cílovou dlaždici, ale zároveň jsou obsahem dlaždic, je tedy třeba oddělit fázi inicializace dlaždic od inicializace přepínačů. Nicméně potom nelze přepínače předat do dlaždic v konstruktoru a tím pádem je pro tuto akci třeba vytvořit veřejnou funkci nebo vlastnost, která tak provede. Tato funkce pak narušuje náš cíl, protože tuto funkci může kdokoliv zavolat později, kdy už to není validní. Z toho důvodu jsme přišli s konceptem

tzv. inicializátorů.

Inicializátor je datová třída obsahující vlastnosti, které by běžně byly parametry konstruktoru inicializovaného objektu. Místo těchto parametrů je do konstruktoru předán inicializátor, který má také události vyvolané při inicializaci a při dokončení inicializace. Třída beroucí inicializátor jako parametr konstruktoru se zaregistruje na tyto události a 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 metody na inicializátoru. S využitím asynchronních metod je navíc možné vyvolat inicializaci prvku (např. přepínačů, které potřebují reference na dlaždice) již při vytváření dlaždic. Což vede k přehlednějšímu kódu a celkově k inicializaci cyklických struktur bez nutnosti přebytečných veřejných položek.

S využitím dědičnosti inicializátorů je možné nasimulovat volání rutin konstruktorů tak, jak je v C# běžné. Na každé úrovni dědičnosti se využijí některé vlastnosti inicializátoru. Nechť inicializátor B je potomek inicializátoru A v hiearchii dědičnosti. Nechť C a D jsou třídy, které chceme inicializovat a zároveň D je potomkem C. Také platí, že konstruktor třídy D vyžaduje inicializátor třídy B a konstruktor třídy C vyžaduje 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 ve správném pořadí, může to nasimulovat pořadí inicializace běžné u konstruktorů. Při inicializaci dlaždic je právě tento způsob používán. Celou situaci znázorňuje obrázek 2.16.



Obrázek 2.16: Ilustrace použití inicializátoru.

2.14 Renderování a interakce

Dalším cílem(viz cíl práce **C5**) tohoto projektu je oddělit v enginu zobrazovací vrstvu tak, aby ji bylo možné jednoduše nahradit za jinou. Se zobrazovací vrstvou nicméně úzce souvisí, jakým způsobem bude prováděna interakce s objekty, proto je v této vrstvě zahrnuta i ona. Každý objekt, který to vyžaduje, má vytvořený vlastní renderer-interactor na míru. Jak renderer 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. Zobrazovací vrstva v tomto enginu nemá stejný grafický výstup jako originální hra, nicméně nic nebrání tomu napsat tuto vrstvu tak, aby jí odpovídala.

3. Vývojová dokumentace

Celá práce je rozdělena do dvou projektů Visual Studia, pro jejichž otevření je zapotřebí mít nainstalované MonoGame SDK 3.4. První z nich je `DungeonMasterParser`, který zajišťuje rozparsování herních dat (viz sekce 2.3). Celý tento projekt se skládá z datových tříd odpovídajících formátu souboru `DUNGEON.DAT` resp. `GRAPHICS.DAT` (viz sekce 2.1 resp. 2.2), které pak sestaví dohromady třída `DungeonParser` za vzniku objektu třídy `DungeonData`. Při vytváření instance třídy `DungeonData` se provede načtení dat odpovídající souboru `DUNGEON.DAT`. Kvůli tomu tento projekt referencuje knihovnu HTML Agility Pack [12].

Druhý projekt `DungeonMasterEngine` obsahuje už samotný engine, kterému se bude hlavně věnovat tato kapitola. Projekt s enginem pak referencuje projekt `DungeonMasterParser` a framework MonoGame [2]. Data týkající se herních úrovní jsou ve složce `/Data` a data jako textury, fonty nebo modely jsou ve složce `/Content`, jak je tomu u MonoGame zvykem. Oba projekty se pak snaží dodržovat následující konvence:

- Každá třída nebo rozhraní jsou ve zvláštním souboru korespondujícím s jejich názvy.
- Skupiny tříd, které patří logicky k sobě, se separují do zvláštních složek.
- Jednotlivé složky odpovídají jmenným prostorům.
- Názvy rozhraní začínají písmenem `I`.
- Třídy rozšiřující abstraktní třídy mají část jejich názvu.
- K abstraktním třídám většinou existuje odpovídající rozhraní se stejnými položkami.

Následující sekce této kapitoly popisuje třídy reprezentující jednotlivé části enginu a vztahy mezi nimi.

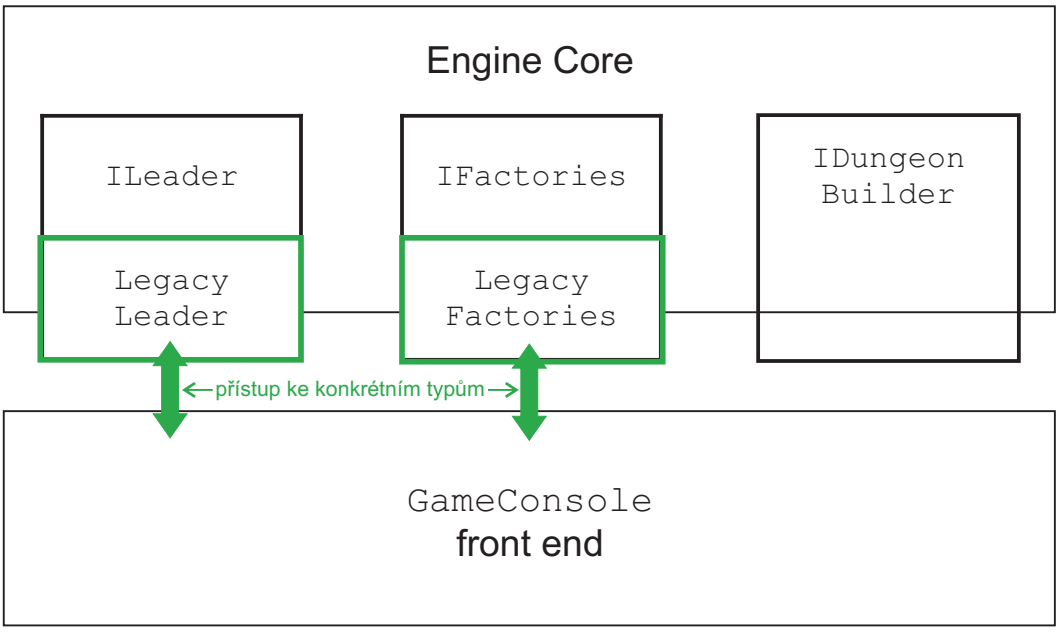
3.1 Jádro enginu – `DungeonBase`

Jádro enginu reprezentuje třída `DungeonBase` a skládá ze tří důležitých komponent, které do něj lze předat při jeho vytváření. Za těmito komponentami stojí následující rozhraní:

- `ILeader` – reprezentuje hráče a jeho skupinu, kterou lze skrze něj ovládat.
- `IFactories` – obsahuje factories pro tvorbu předmětů, nepřátelských entit, kouzel, akcí a definuje úrovně osvětlení.
- `IDungeonBuilder` – vytváří pro engine herní úrovně.

Pro reimplementaci hry Dungeon Master jsou použity následující třídy **`LegacyLeader`**, **`LegacyFactories`**, **`LegacyMapBuilder`**. Konkrétní typy implementací rozhraní **`ILeader`** a **`IDungeonBase`** lze předat jádru jako typové parametry. Případný front end potom může skrze instanci jádra pracovat přímo s konkrétními typy (viz obr. 3.1). Tento projekt neobsahuje žádné uživatelské rozhraní, namísto něho je pro udávání některých příkazů použita konzole **`GameConsole`**.

Jádro lze pak nalézt ve jmenném prostoru:
`DungeonMasterEngine.DungeonContent.Tiles`.



Obrázek 3.1: Ilustrace komunikace frondendu s jádrem enginu.

3.1.1 Asynchronní funkce

Samotné jádro enginu je vytvořeno a aktualizováno ze třídy **`Engine`**, která je potomkem třídy **`Game`** zajišťující herní smyčku. V řadě částí enginu se využívají asynchronní funkce ke zjednodušení implementace. Důležité je poznamenat, že tyto funkce předpokládají, že budou vykonávané v jednom vlákně – jinak může velmi jednoduše dojít k race condition. Z tohoto důvodu je nutné přepnout chování asynchronních funkcí tak, aby se prováděly v jednom vlákně. To lze udělat nastavením vhodného synchronizačního kontextu v hlavním vláknu skrze funkci **`SynchronizationContext.SetSynchronizationContext`**. Tato funkce má jako parametr instanci třídy **`SynchronizationContext`**. Správnou implementaci synchronizačního kontextu – tak, aby se asynchronní funkce prováděly v jednom vlákně – zajišťuje třída **`GameSynchronizationContext`**. Tento synchronizační kontext je nastaven v konstruktoru třídy **`Engine`** za předání fronty, do které budou vkládány asynchronní požadavky. Tato fronta je poté v každém cyklu herní smyčky ve funkci **`Engine.Update`** vyprázdněna a každá její funkce je provedena. Asynchronní funkce implementované v enginu se vždy provádějí ve vlákně herní smyčky. Nicméně při awaitování nějaké asynchronní funkce z jiných zdrojů, například **`Task.Delay`**, je možné, že se tato funkce provádí v jiném vlákně. Z tohoto

důvodu je nutné při vkládání a odebírání funkcí z fronty, provádět synchronizaci pomocí zámků. Jelikož v každém volání této funkce je vyprázdněna celá fronta asynchronních funkcí, je třeba dávat pozor na její přehlcení, které může v konečném důsledku vést až k neresponzivnosti aplikace.

Následující příklad ukazuje použití asynchronní funkce, která odesílá zprávu dlaždici. Odeslání zprávy je přitom provedeno až po uplynutí doby `TimeDelay`.

```
async void SendMessageAsync(Message message)
{
   await Task.Delay(TimeDelay);
   // zpráva je odeslána až po uplynutí doby TimeDelay
   TargetTile.AcceptMessageBase(message);
}
```

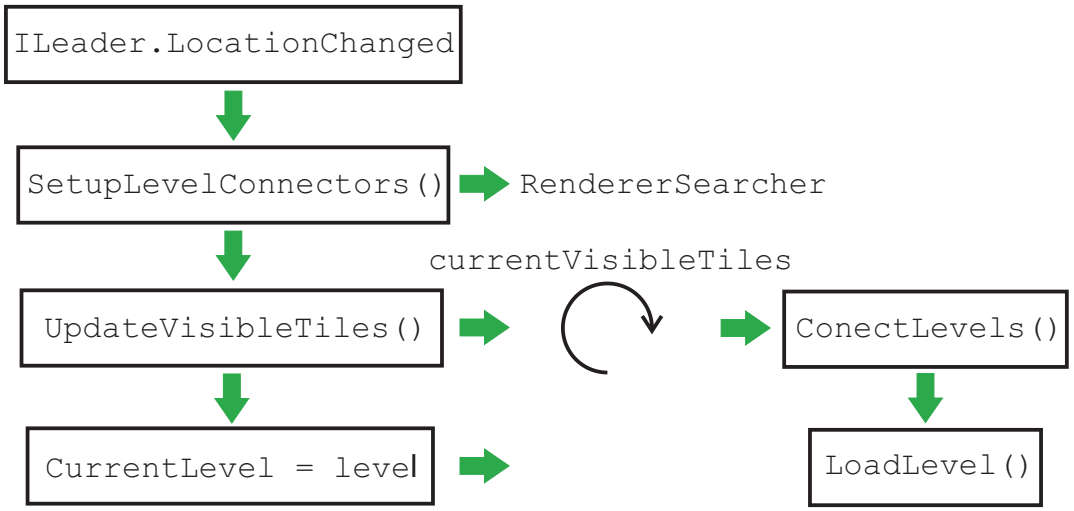
Další příklady využití asynchronních funkcí v enginu bude v následujících sekcích:

1. sekce (viz sekce 3.10) – inicializace cyklických struktur,
2. sekce (viz sekce 3.4.6) – umělá inteligence nepřátelský entit,
3. sekce (viz sekce 3.12) – čekání na vstup do konzole.

3.1.2 Správa herních úrovní a dlaždic

Při inicializaci jádra je na objektu hráče zaregistrována událost `ILeader.LocationChanged`, která je vyvolána po přesunu hráče na novou dlaždici. Zaregistrování resp. odregistrování události se provede po přiřazení hodnoty do vlastnosti `Leader`. V obsluze této události se pak provede aktualizace viditelných dlaždic pomocí funkce `UpdateVisibleTiles` (viz obr. 3.2). K nalezení viditelných dlaždic tato funkce používá třídu `RendererSearcher`, která je potomkem třídy `BreathFirstSearch`. Nalezené dlaždice se uloží do protected proměnné `currentVisibleTiles`. V dalším kroku je zavolána funkce `SetupLevelConnectors`, jenž projde všechny viditelné dlaždice vedoucího do jiných úrovní. Pro každou tuto dlaždici dojde pak voláním funkce `ConnectLevels` k propojení dané dlaždice s další úrovní. Poslední akcí je přiřazení úrovně, v které se hráč vyskytuje, do proměnné `CurrentLevel`.

Funkce `ConnectLevels` načte danou úroveň pomocí funkce `LoadLevel`, která tak učiní skrze builder. Načtená úroveň je pak uložena do kolekce `ActiveLevels`, ve které jsou vždy tři poslední úrovně. Dlaždice, které propojují herní úrovně, by měly implementovat rozhraní `ILevelConnector`. Funkce `ConnectLevels` pak nastaví cílovou dlaždici do vlastnosti `ILevelConnector.NextLevelEnter`. Dlaždice implementující toto rozhraní by měla zajistit její propojení s dlaždicí `ILevelConnector.NextLevelEnter`. Všechny funkce zmíněné v této sekci jsou virtuální, a proto je možné jejich chování upravit přetížením.



Obrázek 3.2: Ilustrace funkce jádra.

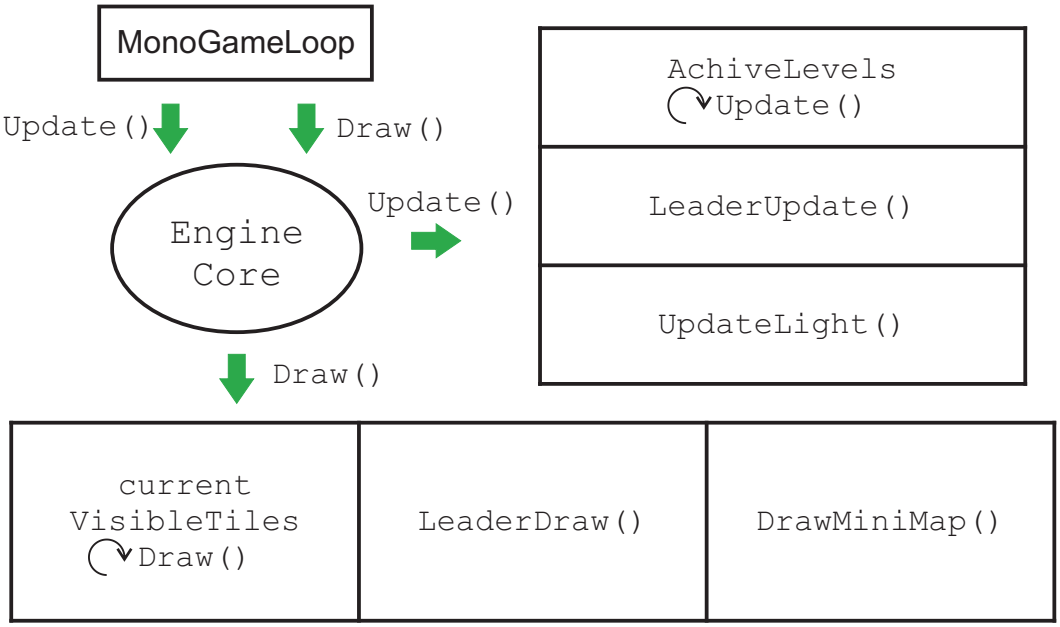
3.1.3 Aktualizace a rendering

MonoGame zajišťuje herní smyčku, skrze kterou je pak třeba na enginu volat funkce **`Update`** a **`Draw`**.

V každém zavolání funkce **`Update`** se provede aktualizace všech herních úrovní v kolekci **`ActiveLevels`**. Herní úroveň reprezentuje typ **`DungeonLevel`**, který má v sobě uložené dlaždice, obtížnost a objekty vyžadující aktualizaci – které se u něj zaregistrovaly (viz sekce 3.1). Funkce **`DungeonLevel.Update`** pak zavolá na všech zaregistrovaných objektech funkci **`IUpdate.Update`**. Všechny tyto objekty musí implementovat rozhraní **`IUpdate`** a musí se samy starat o odregistrování z kolekce, když už aktualizaci nevyžadují. Dále je volána funkce **`ILeader.Update`** na hráči. Při každém provedení aktualizace se ještě zavolá virtuální funkce **`UpdateLight`**. Proces reprezentuje obrázek 3.3.

Funkce **`UpdateLight`** přiřadí do proměné **`Light`** hodnotu určující úroveň světla, která je v setteru vlastnosti přepočítána na vzdálenost, po kterou se vykreslují dlaždice. Getter pak vrací již samotnou vzdálenost, nikoliv hodnotu světla. Hodnota světla je 0 až 5, kde 0 reprezentuje největší osvětlení. Takto je to nastaveno z důvodu kompatibility se zdrojovými kódy originální hry [10]. Funkce projde všechny šampiony v kolekci **`ILeader.PartyGroup`** a nalezne všechny objekty implementující rozhraní **`ILightSource`**. Na základě hodnot **`ILightSource.LightPower`** vypočte výslednou hodnotu osvětlení. Tento výpočet lze změnit přetížením funkce.

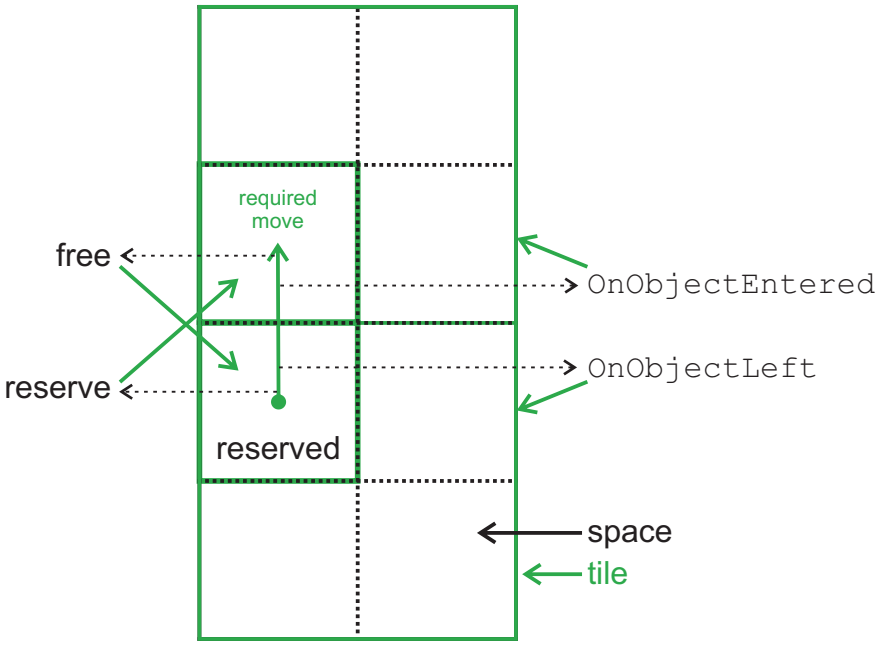
V každém zavolání funkce **`Draw`** se potom provede vykreslení všech dlaždic v kolekci **`currentVisibleTiles`** skrze jejich renderer (viz sekce 2.14). Dále jsou zavolány funkce **`ILeader.Draw`** pro vykreslení objektů ve správě hráče a virtuální funkce **`DrawMiniMap`**, která provede vykreslení minimapy.



Obrázek 3.3: Ilustrace aktualizace a vykreslování z pohledu jádra.

3.2 Dlaždice – `ITile`

Nejobecnější strukturu dlaždice definuje rozhraní **`ITile`**. Každá dlaždice musí mít definované sousedy, to zajišťuje položka **`Neighbors`** typu **`TileNeighbors`**. Tento typ implementuje rozhraní **`INeightbors`**, které vyžaduje pouze enumerátor dvojic se směrem a dlaždicí. Směr reprezentuje struktura **`MapDirection`**, kde se jednotlivé směry dají získat skrze její statické členy. Třída **`TileNeighbor`** potom implementuje ještě pomocné funkce a vlastnosti pro jednoduší práci se sousedy. Pomocí této vlastnosti se objekty mohou pohybovat mezi dlaždicemi (viz obr. 3.4). Po vstupu objektu na dlaždici je třeba zavolat na předchozí dlaždici funkci **`OnObjectLeft`** a na nové dlaždici funkci **`OnObjectEntered`**. Díky tomu může dlaždice reagovat na vstup resp. odchod objektů – což je použité například pro přepínače na podlaze. Další vlastností dlaždice je **`LayoutManager`**, který musí používat entity při pohybu po dlaždicích a mezi nimi. **`LayoutManager`** je třída poskytující správu prostoru na dlaždici, tak aby každý prostor zaujímala pouze jedna entita (viz sekce 3.4.4). Před pohybem entity je tedy ještě třeba zarezervovat (reserve) skrze **`LayoutManager`** cílový prostor na dlaždici a po dokončení pohybu je naopak třeba uvolnit prostor předchozí (free). Poslední položkou potřebnou pro pohyb mezi dlaždicemi je vlastnost **`IsAccessible`**, která určuje, zda lze na dlaždici vstoupit.



Obrázek 3.4: Ilustrace pohybu mezi prostory a dlaždicemi.

Pro implementaci funkce přepínačů je třeba zajistit komunikaci mezi dlaždicemi (viz sekce 2.7.3). Za tímto účelem obsahuje dlaždice vlastnosti pro změnu jejího stavu. Buď lze obsah dlaždice aktivovat přímo pomocí funkce `ActivateTileContent` resp. `DeactivateTileContent` nebo pomocí zaslání zpráv, k jejichž příjetí slouží funkce `AcceptMessageBase`. Konkrétní stav je obsahuje pak vlastnost `ContentActivated`.

V poslední řadě rozhraní reprezentující dlaždice obsahuje prvky:

- `Position` – absolutní pozice od počátku souřadnic,
- `GridPosition` – souřadnice dlaždice na mřížce,
- `Drawables` – do této kolekce se mohou zaregistrovat objekty implementující rozhraní `IRenderable`, které se nechtějí registrovat u dlaždice pomocí funkcí pro vstup a výstup a zároveň se potřebují renderovat,
- `ObjectEntered` resp. `ObjectLeft` – události vyvolané při vstupu resp. odchodu objektu,
- `Level` – reference na level, ve kterém se dlaždice nachází,
- `IsInitialized` – určuje, zda je dlaždice plně inicializovaná (viz sekce 2.13),
- `Initialized` – tato událost je dlaždicí vyvolána pouze jednou a to v momentě, kdy jsou všechny její části inicializované. Tuto událost využívají živé entity k určení doby jejich obživnutí.

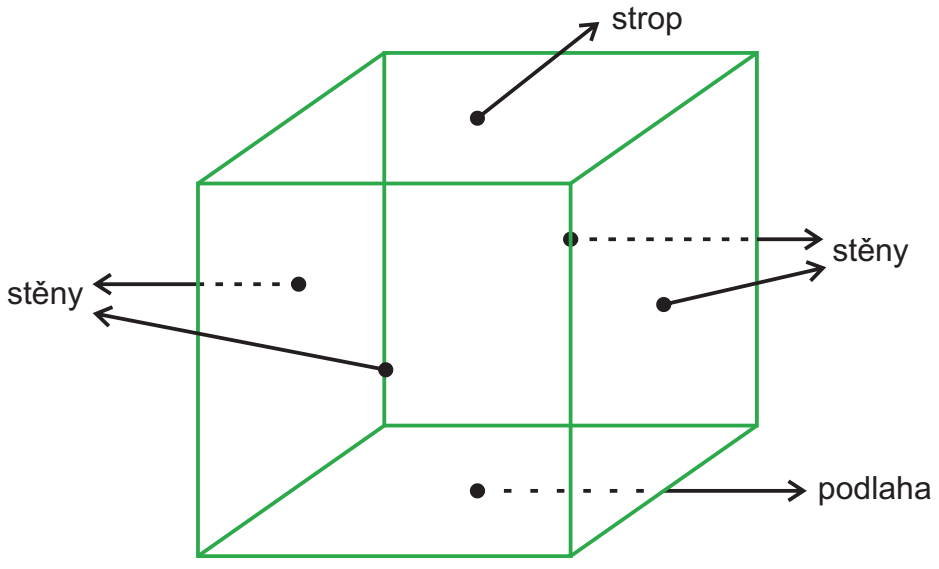
Všechny třídy s dlaždicemi jsou ve jmenném prostoru:
`DungeonMasterEngine.DungeonContent.Tiles`.

3.2.1 Inicializace dlaždic

Jak již bylo zmíněno v analýze (viz sekce 2.13), k inicializaci dlaždic jsou použity tzv. inicializátory. Inicializátor obsahuje vlastností, které by normálně byly předány jako parametry v konstruktoru. Ovšem v moment předávaní inicializátoru do konstruktoru, inicializátor ještě nemusí mít naplněné všechny hodnoty. 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 inicializátoru zakryty novými inicializačními událostmi pro danou úroveň dědičnosti. Tento způsob je použit pouze pro inicializaci dlaždic. Pro jiné objekty, může inicializátor sloužit pouze jako objekt udržující parametry, které jsou hned v konstruktoru inicializované.

3.2.2 Implementace dlaždic

Částečnou implementaci rozhraní `ITile` zajišťuje abstraktní třída `Tile`, která definuje virtuálně `LayoutManager`. Dále abstraktně deklaruje kolekci `SubItems`, která obsahuje všechny objekty na dlaždici. Objekty, které implementují rozhraní `IRenderable`, jsou potom skrze tuto kolekci vykreslovány. Dlaždice také má tzv. strany, a jsou to buď stěny, strop nebo podlaha (viz obr. 3.5). Pro tyto strany pak abstraktně deklaruje kolekci `Sides`, jejímž stěnám pak přeposílá případné zprávy, které přišly na tuto dlaždici skrze metodu `AcceptMessageBase`. Dále implementuje také funkce `OnObjectEntered` resp. `OnObjectLeft` tak, že vyvolá jejich odpovídající událost. Proto je pří případném přetížení těchto funkcí v potomkovi nutné zavolat implementaci rodiče. Všechny předchozí popsané funkce je možné přetěžovat v potomkovi, a tak přizpůsobit prováděné akce. V poslední řadě se tento inicializátor stará o inicializaci pozic na mřížce, úrovně a sousedů skrze `TileInicializator`.



Obrázek 3.5: 3D ilustrace dlaždice a jejích stran.

Přímý generický potomek předchozí třídy `Tile⟨TMessage⟩`, kde `TMessage` je alespoň `Message`, poskytuje možnost přijímat v potomcích zprávy vlastního typu (viz sekce **P3**). V případných potomcích teto třídy lze naimplementovat funkci `AcceptMessage`, která přijímá zprávy typu `TMessage`. Tato třída přetěžuje a zároveň uzavírá metodu `AcceptMessageBase` tak, že deleguje zprávy typu `TMessage` do metody `AcceptMessage`. Naopak základní implementace metody `AcceptMessage` je zavolání právě rodičovské implementace `AcceptMessageBase`. Při případném přetěžování této funkce je třeba rozhodnout, zda je nutné volat implementaci rodiče. Všechny dlaždice, které dědí ze třídy `Tile⟨TMessage⟩`, vytváří dva potomky. První z nich je pro případné rozšiřitele, a proto ponechává typový parametr, druhý pak specifikuje typový parametr na obecný typ `Message`.

Dlaždice podlaha – `FloorTile⟨TMessage⟩`

Tato dlaždice implementuje funkci všech jejích stran (viz obr. 3.5). Na stranách, kde nemá dlaždice sousedy jsou přidány stěny, které mají podporu pro přepínače (viz sekce 2.1) a dekorace (viz sekce 2.7). Dále je přidána strana na místo stropu a podlahy. Strana s podlahou pak může obsahovat přepínač a je na ni možné odkládat či z ni odebírat předměty. Dlaždice také deleguje veškeré vstupy resp. odchody objektů do samotné strany podlahy typu `FloorSide` (viz sekce 3.2.3) a to kvůli podpoře odkládání předmětů na zem.

Další implementace dlaždic

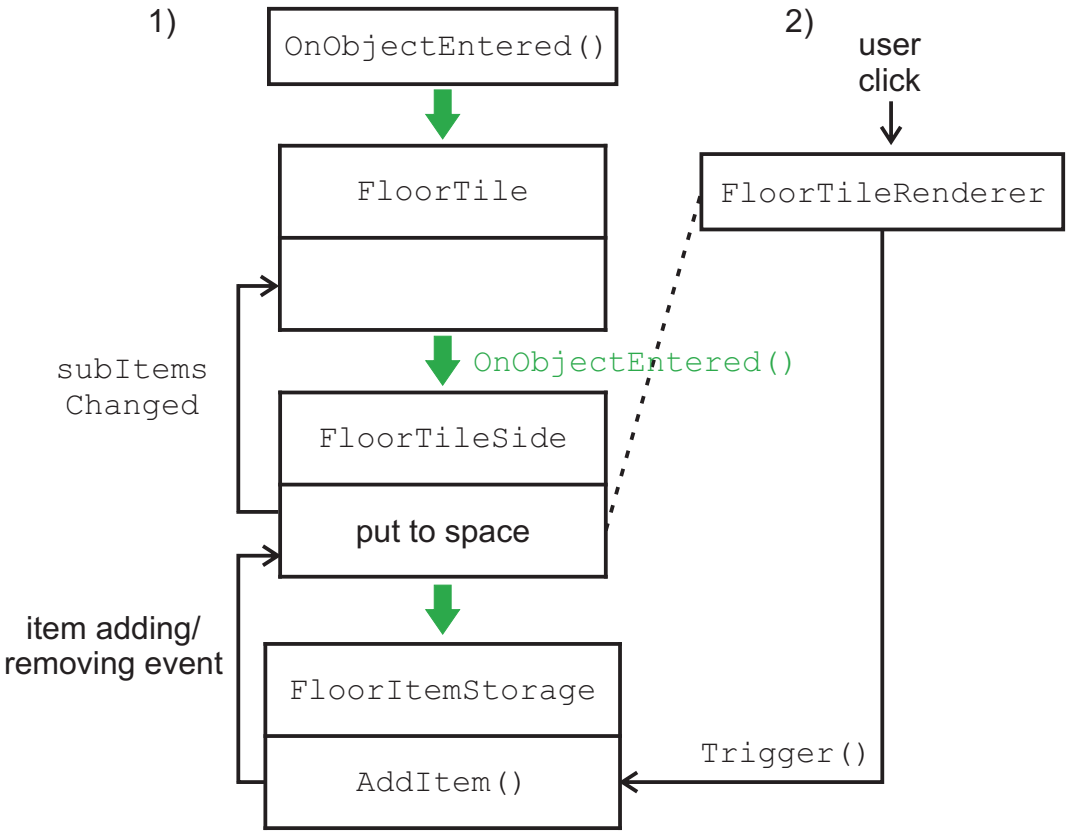
Další dlaždice převážně využívají dlaždici podlahu pomocí dědičností. Výjimkou jsou například schody, které jsou zároveň příkladem dlaždice implementující rozhraní `ILevelConnector`. Engine při načtení levelu nastaví vlastnost `ILevelConnector.NextLevelEnter` na odpovídající dlaždici (viz sekce 3.1.2). Při změně předchozí vlastnosti provede dlaždice nastavení sousedů vedoucích do dalšího levelu a používá k tomu vlastní implementaci sousedů přetížením vlastnosti `Neighbors`. Tato možnost je použita ještě u jámy, kvůli propadu do nižšího levelu. Rozhraní spojující levely je ještě použito u teleportu, kde při vstupu správného objektu na teleportační dlaždici dojde k teleportaci objektu na dlaždici `ILevelConnector.NextLevelEnter`. Další dlaždicí jsou dveře, které opět dědí z podlahy a navíc implementují rozhraní `IHasEntity`. Toto rozhraní obsahuje vlastnost typu `Entity`, která reprezentuje neživou entitu na dlaždici. S touto entitou můžou pak pracovat akce, kterými lze dveře rozbít. Poslední dlaždici reprezentuje třída `LogicTile`, na kterou nelze vstoupit a slouží pouze jako podpora pro přepínače (viz sekce 3.3).

3.2.3 Strany dlaždic – `ITileSide`

Jak již bylo naznačeno (viz obr. 3.5), dlaždice mohou mít strany, které jsou buď stěny, podlaha nebo strop. Každou tuto stranu reprezentuje samostatný objekt, který má svůj renderer-interactor (viz sekce 2.14). Toto rozhraní vyžaduje pouze položku pro renderer a funkci pro přijímání základních zpráv `Message`. Tato funkce je zde z kompatibilních důvodu s originální hrou, kdy texty na zdech mohou být těmito zprávami skryty či zobrazeny.

Následuje seznam jednotlivých implementací:

- **`TileSide`** – reprezentuje pouze holou stěnu bez žádných funkcí, která obsahuje pouze vlastnost typu **`MapDirection`** určující její pozici.
- **`ActuatorTileRenderer`** – přímý potomkem předchozí strany, který navíc obsahuje přepínač a renderer, jenž se navíc stará o jeho vykreslování a interakci.
- **`TextTileSide`** – opět přímý potomek první strany, jejíž renderer navíc zobrazuje na zdi text. Viditelnost textu lze změnit zasláním zprávy této straně.
- **`FloorTileSide`** – tato strana obsahuje čtyři úložiště, na které může hráč pokládat předměty. 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 získat obsažené objekty. Předměty lze pak na podlahu přidávat dvěma způsoby, přičemž oba pomocí události informují rodičovskou dlaždici o změně jejího obsahu. První možností je položení předmětu přímo z ruky hráče pomocí renderer-interactoru. Druhý způsob je zavolání metody **`OnObjectEntered`** resp. **`OnObjectLeft`** objevující se například při teleportaci nebo hodu předmětu. Celou situaci zobrazuje obrázek 3.6.
- **`ActuatorFloorTileSide`** – přímý potomek předchozí strany podlahy, který navíc přidává na podlahu nášlapný přepínač.



Obrázek 3.6: Ilustrace komunikace podlahy s její dlaždicí při přidávání předmětu.

3.3 Přepínače – `IActuatorX`

Za přepínači stojí rozhraní **`IActuatorX`**, které vyžaduje pouze API pro příjem zpráv typu **`Message`** – pro dodržení kompatibility s původní hrou. Toto rozhraní pak implementuje třída **`ActuatorX`**, která reprezentuje přepínače používající pro jejich funkci senzory z originální hry (viz sekce 2.7). Kromě toho dále existují implementace přepínačů, které jsou provedeny jiným způsobem.

Třídy související s přepínači a senzory lze nalézt ve jmenném prostoru: **`DungeonMasterEngine.DungeonContent.Actuators`**.

3.3.1 Implementace obecných přepínačů

Tato sekce pojednává o přepínačích, které ke své funkci nepoužívají senzory. V tomto enginu jsou použity pro dekorace, které provádějí nějakou funkci. K těmto dekoracím v originální hře nenáleží žádné senzory. Studií dekompilovaného kódu originální hry [10] se ukázalo, že některé akce jsou fixovány na konkrétní typy dekorací. Například pro dekorace s výklenky je naimplementována možnost vkládaní předmětů. Jelikož tyto dekorace mohou být i součástí senzorů, bylo zapotřebí, aby objekty reprezentující dekorace implementovaly rozhraní **`IActuatorX`**, a tak bylo možné naimplementovat funkce pro jejich interakci.

U těchto přepínačů lze zvolit jejich vnitřní formát. Existující implementace jsou:

- **`DecorationItem`** – stará se pouze o rendering dekorace, který zajišťuje třída **`DecorationRenderer`**.
- **`Alcove`** – reprezentuje dekoraci výklenku, o jehož rendering a interakci se stará třída **`AlcoveRenderer`**.
- **`ViAltairAlcove`** – potomek třídy **`Alcove`**, který se navíc stará o reinkarnaci šampionů.

3.3.2 Implementace přepínačů se senzory

Tyto přepínače se skládají z řady senzorů, které mohou provádět následující akce:

- změna stavu vzdálené dlaždice,
- zarotování sekvencí senzorů přepínače,
- přidání zkušeností hráči.

Při pokusu o aktivaci přepínače dojde postupně k aktivaci všech jeho senzorů. Senzory pak mohou být aktivovány následujícími způsoby:

- kliknutím myši na dekoraci senzoru, pokud je senzor na zdi,
- vstupem resp. odchodem hráčovi skupiny na dlaždici resp. z dlaždice,
- zprávou vyslanou jiným senzorem.

Více viz sekce 2.7.

Implementace senzorů

Každý typ senzoru má jinou podmínku aktivace. Senzory se dělí na senzory použité na podlaze, zdi a na speciální tzv. logické senzory.

Hlavní třídou reprezentující senzor je abstraktní třída **`SensorX`**. Tato třída obsahuje společné funkce, které jsou využívány jejími potomky. Dále také definuje vlastnosti, které lze rozdělit do tří skupin. Tyto vlastnosti jsou inicializovány inicializátorem (viz sekce 2.13), který slouží pouze jako datový nosič. Jde o inicializátor, který reprezentuje třída **`SensorInitializerX`**. Pro inicializaci potomků jsou používány poděděné implementace tohoto inicializátoru.

Vlastnosti senzorů třídy **`SensorX`**:

- Obecné vlastnosti:
  - **`Delay`** – opoždění akce v milisekundách, po kterém se provede akce senzoru.
  - **`LocalEffect`** – informace, zda-li je akce určena lokálně pro rodičovskou dlaždici nebo pro vzdálenou.
  - **`OnceOnly`** – informace, zda-li je senzor možné aktivovat pouze jednou.
  - **`RevertEffect`** – u většiny přepínačů obrátí efekt výsledné akce, nicméně přesný význam si stanoví vždy konkrétní senzor.
  - **`Audiable`** – informace určující, zda po aktivaci dojde k přehrání zvuku. (v tomto enginu není použit)
  - **`GraphicsBase`** – reprezentuje dekoraci a zároveň přepínač obecného typu implementující rozhraní **`IActuatorX`**. Pokus o aktivaci tohoto přepínače se provádí, pouze pokud je zobrazena dekorace (viz obr. 3.7).
- Vlastnosti pro lokální akci:
  - **`Rotate`** – informace určující, zda-li se má seznam senzorů při aktivaci zarotovat.
  - **`ExperienceGain`** – informace určující, zda-li se mají hráči po aktivaci přidat zkušenosti.
- Vlastnosti pro vzdálenou akci:
  - **`Effect`** – určuje, jaká zpráva je odeslaná na cílovou dlaždici. (Akce zprávy může být obrácená, pokud je nastavena vlastnost **`RevertEffect`**). 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.
  - **`Specifier`** – směr použitý v odeslané zprávě, který může být interpretován jako číslo (lze získat pomocí **`MapDirection.Index`**).
  - **`TargetTile`** – reference na cílovou dlaždici.

Kromě předchozích vlastností obsahuje tato třída funkce pro vykonání akcí senzorů. Pokud je efekt senzoru vzdálený, funkce `TriggerEffect` odešle zprávu na danou cílovou dlaždici s daným zpožděním `Delay`. Zpoždění je prováděno pomocí asynchronních funkcí (viz sekce 3.1.1). Jinak provede akci pro lokální efekt, který lze také vykonat přímo zavoláním funkce `TriggerLocalEffect`. Tyto funkce lze použít pouze v potomcích.

Následující třídy jsou přímými potomky třídy `SensorX`:

1. `FloorSensor` – tyto senzory lze vložit pouze do přepínačů, které mohou být na podlaze.
2. `WallSensor` – tyto senzory lze vložit pouze do přepínačů, které mohou být na stěně.
3. `LogicGateSensor` resp. `CounterSensor` – pro tyto senzory existuje speciální dlaždice `LogicGateTile`

Senzory z bodu 1 a 2 mají funkci `TryTrigger`, která se pokusí senzor aktivovat a pokud uspěje, stará se o provedení efektu. Zda-li se senzor aktivuje stanoví abstraktní funkce `TryInteract`, kterou implementují potomci. Pro každou variantu senzoru mají funkce odlišné parametry. Pokud je třeba modifikovat způsob provedení výsledného efektu, je možné funkce `TryTrigger` přetížit.

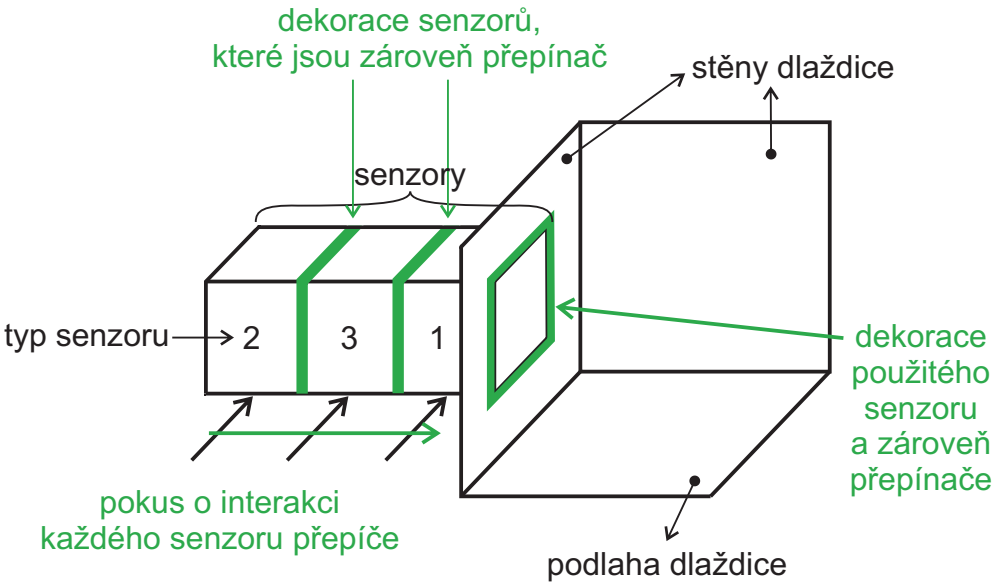
Kromě předchozích senzorů ještě existují tzv. logické senzory (viz 1). První z nich `LogicGateSensor` funguje jako logické hradlo, které má k dispozici osm bitů. Čtyři z nich jsou nastaveny při designu na hodnoty a ostatní na nuly. S druhou čtveřicí lze pak manipulovat pomocí zpráv. Index směru zprávy určuje, kolikátý bit se má ovlivnit. Tento bit je pak ovlivněn akcí zprávy. Pokud jsou obě čtveřice stejné, je vyvolán efekt senzoru. Dalším senzorem je `CounterSensor`, který má dané číslo. Aktivační zprávy toto číslo poté navyšují a deaktivační zprávy ho snižují. Pokud je číslo nula, je vyvolán efekt.

Implementace přepínačů používající senzory

Rodiči těchto přepínačů je třída `Actuator`. Tato třída poskytuje funkci pro zarotování senzory přepínače, kterou lze vyvolat pouze z potomků. Zda-li se mají senzory zarotovat se určuje nastavením vlastnosti `Rotate`, která se po jeho provedení nastaví zpět na false. Samotné zarotování se pak provede zavoláním funkce `ProcessRotationEffect`.

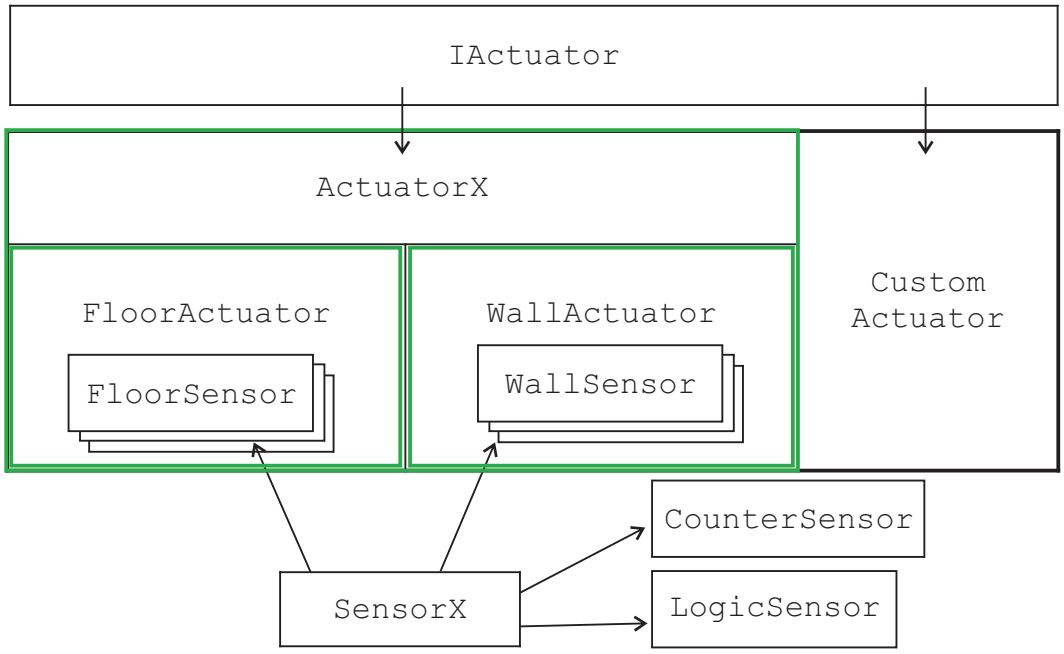
Prvním potomkem předchozí třídy je třída `FloorActuator`, která může obsahovat pouze senzory určené na podlahu tj. `FloorSensor`. Pokus o její aktivaci se provádí funkcí `Trigger`. Jako první parametr se jí předává objekt, který vstupuje resp. odchází z dlaždice, dále seznam všech objektů na dlaždici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

Dalším potomkem je třída `WallActuator`, která může obdobně obsahovat pouze senzory určené na zeď tj. `WallSensor`. Pokus o její aktivaci se provádí opět funkcí `Trigger`, která má jediný parametr typu `ILeader`. Pro přepínač je určena strana `ActuatorWallTileSide`. Funkce `Trigger` je pak volána skrze renderer dané strany. Ta se pak pokusí postupně aktivovat všechny senzory a navíc se pokusí aktivovat přepínač zobrazené dekorace (viz obr. 3.7).



Obrázek 3.7: Ilustrace struktury přepínače se senzory.

Poslední speciální implementací je třída **`LogicActuator`**, která může obsahovat logické senzory. Komunikace se senzory probíhá pomocí zasílání zpráv. Pro tento přepínač je vytvořena speciální dlaždice, na kterou nelze hráčem vstoupit. Tato dlaždice je reprezentována třídou **`LogicTile`**.



Obrázek 3.8: Ilustrace hierarchie přepínačů a senzorů.

3.4 Herní entity

Za deklarací neživých entit stojí rozhraní `IEntity`, které definuje funkci `GetProperty` pro získání vlastnosti dané entity. Za živými entitami stojí naopak rozhraní `ILiveEntity`, které navíc disponuje funkcí `GetSkill` pro získání schopností. Dále také deklaruje:

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

3.4.1 Implementace vlastností entit

Vlastnost deklaruje rozhraní `IProperty`, které obsahuje následující atributy:

- `Value` – stanovuje aktuální hodnotu dané vlastnosti. V jejím getteru a setteru se typicky provádějí kontroly okrajových hodnot a případné reakce na ně.
- `BaseValue` – maximální hodnota, které může vlastnost nabýt, pokud není nějak modifikovaná.
- `MaxValue` – určuje maximální hodnotu včetně modifikací.
- `AdditionalValues` – je to kolekce typů `IEntityPropertyEffect` obsahující modifikace dané vlastnosti.
- `Type` – je to vlastnost typu `IPropertyFactory`, která reprezentuje typ vlastnosti. Entita na základě tohoto typu potom vrací odpovídající vlastnost. Pro usnadnění tvorby unikátních typů existuje generická třída `PropertyFactory`, která má jako typový parametr typ alespoň `IProperty`. Tato třída se řídí vzorem singleton, jejíž jedinou instanci lze získat přes statickou položkou `Instance`. Takto lze pak generovat typy pomocí třídy reprezentující vlastnost.

Pro usnadnění práce existuje implementace rozhraní `IProperty` a to třída `Property`. Tato třída definuje maximální hodnotu jako součet základních hodnot a sumu modifikujících hodnot. Dále při nastavení konkrétní hodnoty ořezává hodnotu do povoleného intervalu, který je mezi nulou a maximální hodnotou. Také definuje událost vyvolanou při změně hodnoty. Všechny vlastnosti jsou deklarovány abstraktně nebo virtuálně.

Některé vlastnosti mohou požadovat přístup k jiným vlastnostem či přístup k rodičovské entitě. V takovém případě je na programátorovi, 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 jmenném prostoru `DungeonMasterEngine.DungeonContent.Entity.Properties`. Jak již bylo zmíněno (viz sekce 2.8.1), může nastat, že daná entita vlastností nedisponuje. V takovém případě je vrácena instance třídy `EmptyProperty`.

3.4.2 Implementace dovedností entit

Dovednosti deklaruje rozhraní `ISkill`, které obsahuje následující atributy:

- `SkillLevel` – udává úroveň, na které je daná dovednost včetně úrovní získaných kouzelnými předměty.
- `BaseSkillLevel` – udává úroveň bez úrovní získaných kouzelnými předměty.
- `Experience` – určuje celkovou dosavadní hodnotu zkušeností.
- `TemporaryExperience` – určuje dočasné zkušenosti, které můžou být jak přidávány tak odebírány.
- `BaseSkill` – je reference na základní dovednost (viz sekce 2). Pokud je daná schopnost už základní, je hodnota `null`.
- `AddExperience` – umožňuje přidat zkušenosti.
- `Type` – je to vlastnost typu `ISkillFactory`, která reprezentuje typ dovednosti. Entita na základě tohoto typu potom vrací odpovídající dovednost. Stejně jako u vlastností, je možné tyto reprezentanty vytvářet pomocí generické třídy `SkillFactory`.

Konkrétní algoritmus přepočítávání zkušeností na úrovně dovedností záleží vždy na konkrétní implementaci.

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

3.4.3 Tělo – `IBody`

Následující API umožňuje pro entity vytvořit různé druhy těl. Tělo se skládá z částí, které mohou sloužit jako úložiště. Dále pak existují externí úložiště tzv. inventáře. API je navržené tak, aby dovolovalo definovat různé druhy úložišť a těl. Engine obsahuje pouze implementaci pro lidské tělo tj. třída `HumanBody`. Tělo entity je definováno rozhraním `IBody` a obsahuje následující atributy:

- `BodyParts` – obsahuje seznam částí těla,
- `Storages` – obsahuje seznam všech úložišť včetně částí těla,
- `GetStorage` – funkce pro vyhledání úložiště včetně částí těla,
- `GetBodyPart` – funkce pro vyhledání části těla.

Externí úložiště – `IInventory`

Každý inventář má definovanou vlastnost **`Type`**, což je typ úložiště, který reprezentuje instance třídy implementující rozhraní **`IStorageType`**. Dále pak definuje readonly kolekci **`Storage`** pro ukládání předmětů. A v poslední řadě obsahuje funkce pro přidávaní a odebírání předmětů z úložného prostoru tj. **`TakeItemFrom`**, **`AddItemTo`**, **`AddItem`**, **`AddRange`**. Třída **`Inventory`** implementuje všechny funkce inventáře a je jí tedy možno přímo použít nebo rozšířit.

Část těla – `IBodyPart`

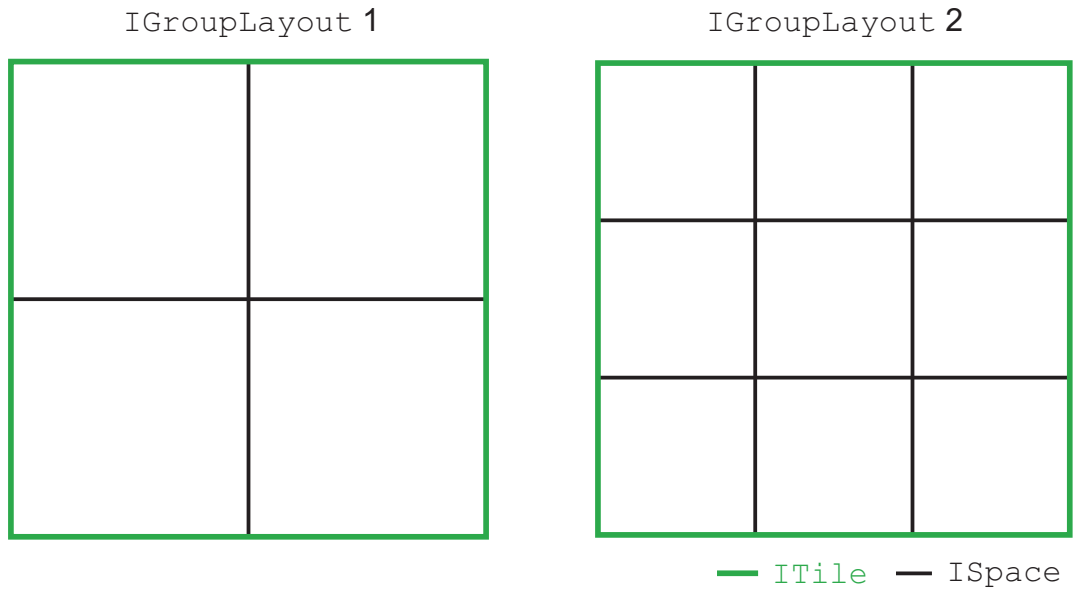
Část těla je definovaná rozhraním **`IBodyPart`**, které je potomkem rozhraní **`IInventory`**. Toto rozhraní navíc definuje následující vlastnosti:

- **`IsWounded`** – definuje, zda je část těla zraněna,
- **`InjuryMultipler`** – definuje pravděpodobnost zranění části těla.

Obecnou implementací rozhraní je třída **`BodyPart`**.

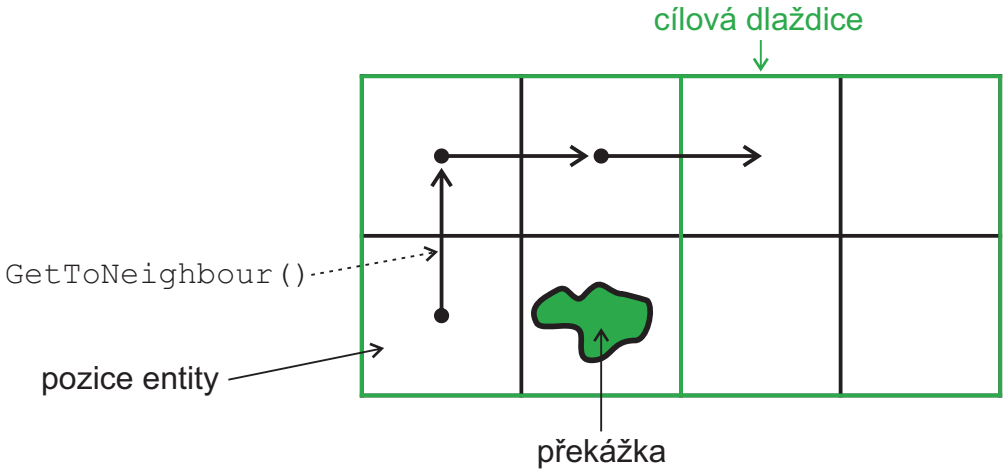
3.4.4 Rozmístění entity na dlaždici

Rozhraní **`IGroupLayout`** definuje obecně možné rozmístění na dlaždici. Jeho vlastnost **`AllSpaces`** obsahuje všechny možné prostory, které daná entita může využít. Tyto prostory rozdělují prostor dlaždice na mřížku (viz obr. 3.9), která nemusí být rovnoměrná. Jednotlivě prostory jsou reprezentovány rozhraním **`ISpace`**. Dale API vyžaduje funkci **`GetToNeighbour`**, která nalezne cestu skrze prostory na cílovou sousední dlaždici (viz obr. 3.10). Funkce **`GetToSide`** vrací cestu 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`**.

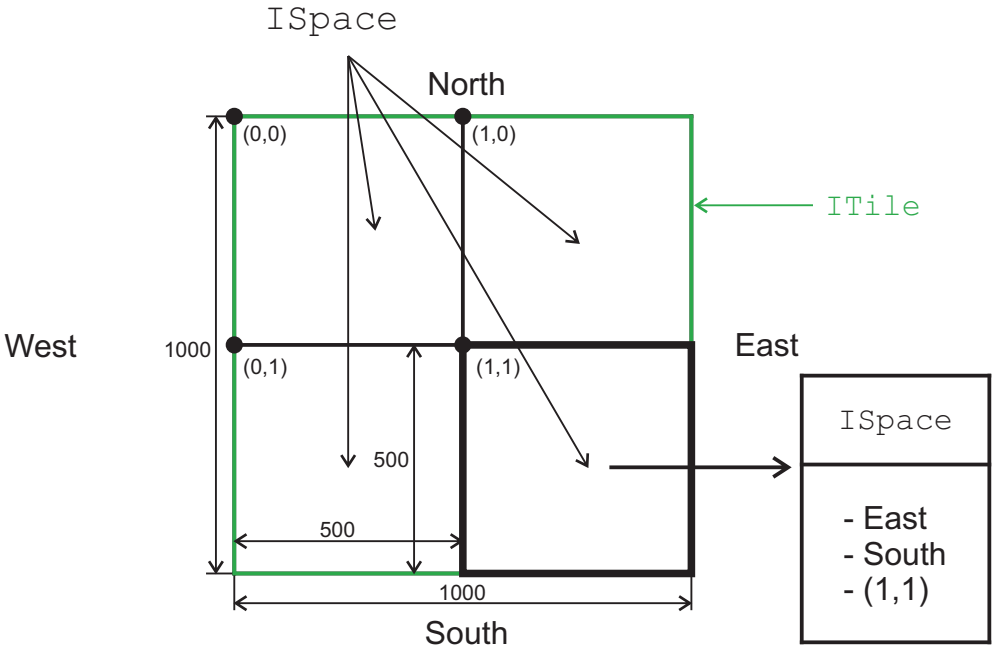


Obrázek 3.9: Ilustrace rozdělení dlaždice na prostory.

Rozhraní **ISpace** musí definovat, k jakým stranám dlaždice je prostor přilehlý. Dále musí pomocí obdélníku definovat prostor, který na dlaždici využívá. Celý prostor dlaždice je pak definovaný na pole o velikosti 1000x1000. 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 generické rozhraní **INeighborable**. Ilustrace viz obrázek (viz obr. 3.11)



Obrázek 3.10: Ilustrace nalezení cesty na sousední dlaždici.



Obrázek 3.11: Ilustrace vlastností prostorů.

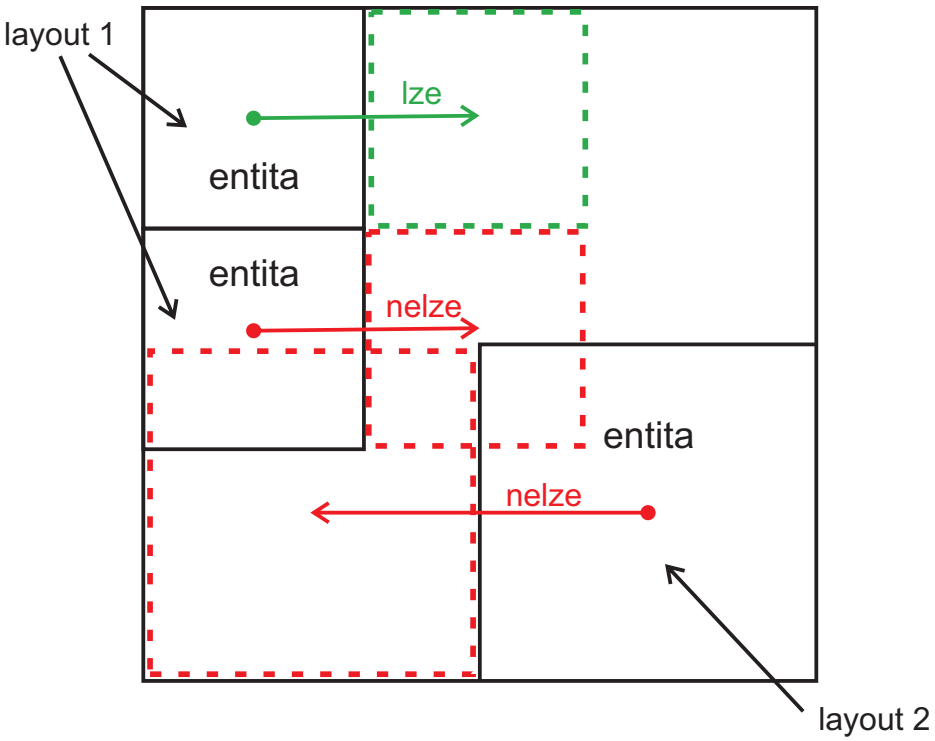
Rozhraní **ISpaceRouteElement** se potom skládá pouze z prostoru, dlaždice a absolutní pozice, na které má stát daná entita. Pro výpočet této pozice se může použít pozice dlaždice a prostor. Při při implementaci tohoto roz-

hraní je možné využít předem připravený hledač nejkratších cest pro prostory `GroupLayoutSearcher`. Pro reprezentaci sousedů prostorů mřížky existuje třída `FourthSpaceNeighbors`.

Celé toto rozhraní je immutable objekt, který pouze definuje prostory na dlaždici a způsob pohybu mezi nimi. Z toho důvodu instance tříd implementující toto rozhraní mohou být singeltony. Engine definuje layout pro prostor reprezentující celou dlaždici a pro prostory jako čtvrtiny dlaždice.

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 třída `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. Entity s různými rozmístěními mohou být na stejné dlaždici, pokud je na ní dostatek prostoru pro obě entity (viz obr. 3.12).



Obrázek 3.12: Ilustrace vlastností prostorů.

3.4.5 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 funkci, která pro daný token vrátí, zda entita odpovídající danému tokenu je nepřátelská. Jednoduchou implementací tohoto rozhraní je třída `DefaultRelationManager`, která při svém vzniku definuje nepřátele. Nicméně,

pokud daná implementace nevyhovuje, je čistě na programátorovi, aby si vytvořil implementaci vlastní. K dispozici je ještě generátor unikátních tokenů a to statická třída `RelationTokenFactory`.

3.4.6 Implementace entit

Engine obsahuje implementaci dvou entit: šampiony a nepřátelské entity. Třídy související s entitami jsou ve jmenném prostoru:
`DungeonMasterEngine.DungeonContent.Entity`.

Šampion

Šampion je první živá entita a reprezentuje jej třída `Champion`. Šampion neobsahuje žádnou umělou inteligenci, je ovládán hráčem skrze třídu `Leader`, která reprezentuje hráčovu skupinu šampionů. Inicializace vlastností a schopností probíhá přes datový inicializér 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 má dva typové parametry. První z nich je typ posouvaného objektu, který musí implementovat alespoň rozhraní `IMovable`. Toto rozhraní definuje vlastnosti, pomocí kterých lze měnit pozice daného objektu. Dalším parametrem je typ prostorů, mezi kterými se objekt pohybuje. Ten musí implementovat alespoň rozhraní `IStopable`, které definuje pozici, na které se má objekt postavit. Animátor poskytuje API pro plynulý posun objektů mezi prostory.

U šampiona je tento animátor použit automaticky při změně lokace.

Nepřátelské entity

Všechny nepřátelské entity ve hře reprezentuje třída `Creature`. Vlastnosti jednotlivých typů nepřátelský entit jsou ve třídách typu `CreatureFactory`. Každá konkrétní instance nepřátelské entity má referenci na tuto třídu a její chování je ovlivněné vlastnostmi v ní obsažené.

Tato třída definuje pro nepřátelské entity jednoduchou umělou inteligenci. K zjednodušení implementace jsou zde využity asynchronní funkce (viz sekce 3.1.1). Ty jsou především využity pro vytváření zpoždění akcí pomocí `Task.Delay`, bez nutnosti počítání času a vracení se zpět do funkce po jeho uplynutí.

Následující příklad ukazuje použití asynchronních funkce pro životní cyklus nepřátelské entity. Každá z funkcí `Hount`, `GoHome`, `WatchAround` provádí určitou obslužnou rutinu, při které může dojít k její změně. V tom případě se z funkce vyskočí a další iterace cyklu zvolí správnou rutinu dle stavu.

```
async void LiveAsync()
{
        while (Activated)
        {
                if (hounting)
                        await Hount();
                else if (gettingHome)
```

```
                        await GoHome();
                else
                        await WatchAround();

                await Task.Delay(100);
        }
}
```

Složitějším příkladem může být funkce pro provádění pohybu mezi prostory. Funkce zjistí, zda je na cílové dlaždici nepřítel a zda lze na cílový prostor dlaždice vstoupit. Pokud je na dlaždici nepřítel nebo je cílový prostor obsazený, funkce vrátí neúspěch. Jinak je proveden posun mezi prostory dlaždic pomocí animátoru. V tomto příkladě je navíc funkce **`animator.MoveToAsync`** implementována pomocí future a promise, která se splní po dokončení pohybu. Aktualizace samotného pohybu je potom prováděna synchronně v metodě **`animator.Update`**. Po dokončení pohybu je nastavena future a tak dojde k pokračování kódu funkce.

```
async Task<bool> MoveToSpace(ISpaceRouteElement destination)
{
   bool EnemyAtTile = destination.Tile.LayoutManager.Entities
      .Any(x => RelationManager
             .IsEnemy(x.RelationManager.RelationToken));

   if (!EnemyAtTile && destination.Tile.LayoutManager
             .TryGetSpace(this, destination.Space))
   {
      //free previous location
      location?.Tile.LayoutManager
         .FreeSpace(this, location.Space);
      await animator
             .MoveToAsync(this, destination, setLocation: true);
      return true;
   } else {
      return false;
   }
}
```

Následuje seznam funkcí a jejich popis použitých pro simulování inteligence. Všechny tyto funkce je možné přetěžovat.

- **`Live`** – V této funkci je nekonečný cyklus, který volá obslužné rutiny podle stavu entity. Jsou to:
  - Lov – entita spatřila nepřítele a pronásleduje ho,
  - Cesta domů – entita pronásledovala nepřítele, kterého následně ztratila, proto jde domů tj. na místo svého vzniku,
  - Hlídkování – entita hlídkuje v okolí oblasti svého vzniku.
- **`FindEnemies`** – Pomocí prohledávání do šířky se pokusí najít entity s nepřátelským tokenem. Maximální hledaná oblast je určena dle vlastností nepřátelské entity. Pokud je nepřítel nalezen, nastaví proměnou **`hountingPath`**.

- `MoveToSpace` – Přesune entitu mezi prostory pomocí animátoru a nastaví správně použité prostory pomocí `LayoutManageru`.
- `MoveThroughSpaces` – Přesune entitu přes prostory dlaždice až na cílovou dlaždici pomocí funkce `MoveToSpace`. Při každém pohybu hledá nepřátele pomocí funkce `FindEnemies`, pokud je tak nastaveno parametrem funkce. Pokud nějaké najde, přeruší pohyb.
- `MoveToNeighbourTile` – Posune entitu na určenou dlaždici přes prostory dlaždice pomocí funkce `MoveThroughSpaces`. Pokud je cesta nepřístupná vrátí `false`.
- `GoHome` – Entita jde domů podle cesty uložené v proměnné `homeRoute` a poté ji nastaví na `null`. Mezi dlaždicemi se posunuje pomocí funkce `MoveToNeighbourTile`.
- `FindNextWatchLocation` – Pomocí prohledávání do šířky z místa objevení entity nalezne dlaždici, na které dlouho entita nebyla a vrátí k ní cestu.
- `WatchAround` – Pomocí funkce `FindNextWatchLocation` nalezne 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 dlaždici, na které je nepřítel a zahájí útok pomocí funkce `Fight`.
- `GetPathHome` – Pokusí se nalézt cestu domů. Pokud ji nalezne nastaví `homeRoute`.
- `EstablishNewBase` – Nastaví výchozí dlaždici na nynější.
- `Hount` – Entita 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 nenalezne, založí si domov tam, kde je pomocí funkce `EstablishNewBase`.

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

3.5 Předměty

Předměty reprezentuje rozhraní `IGrabableItem`. Jak již bylo zmíněno v analýze (viz sekce 2.15), každý předmět musí mít referenci na svoji factory minimálně typu `IGrabableItemFactoryBase`. Dále musí definovat vlastnost `Location` určující prostor dlaždice, na kterém se nachází. Přiřazení do této vlastnosti by mělo vyvolat přidání daného předmětu na danou dlaždici. Existují i případy, kdy toto není žádoucí, a proto předměty musí ještě definovat funkci `SetLocation`, která pouze danou vlastnost nastaví. V poslední řadě musí definovat renderer.

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

3.5.1 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 factory 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 factory i přesně typovanou referenci, skrze kterou bude možné k vlastnostem přistupovat. Inspirace lze nalézt v již implementovaných třídách, které jsou v jmenném prostoru `DungeonMasterEngine.DungeonContent.GrabableItem`.

3.6 Akce

Akce definuje rozhraní `IAction`, které má následující položky:

- `Factory` – reference na factory pro daný typ akcí.
- `Apply` – aplikuje akci, přičemž může použít směr, který je předaný jako parametr. Ostatní položky je třeba předat při inicializaci akce. Co akce provede záleží na konkrétní implementaci. Implementované akce boje využívají směr k určení dlaždice, kam je útok aplikovaný.

Existuje ještě generický potomek, který může mít jako typový parametr potomka třídy `IActionFactory`. Toto rozhraní vyžaduje jedinou funkci `CreateAction` pro tvorbu akcí typu `IAction`. Každá factory je pak uložena ve třídě implementující `IFactories` v jádře enginu (viz sekce 3.1) tak, aby mohl tyto akce kdokoliv používat.

Třída `AttackBase` obsahuje některé funkce, které používají jak útoky nepřátelský entit, tak útoky šampionů. Třída `CreatureAttack` je jejím prvním potomkem a implementuje útoky nepřátelský entit. Třída `HumanAttack` je jejím druhým potomkem a je rodičem pro všechny útoky šampionů, které lze nalézt ve jmenném prostoru `DungeonMasterEngine.DungeonContent.Actions`.

Implementace akcí lze nalézt ve jmenném prostoru:
`DungeonMasterEngine.DungeonContent.Actions`.

3.7 Kouzla

Kouzlo definuje rozhraní `ISpell`, které vyžaduje implementovat jedinou funkci `Run`. Tato funkce má dva parametry a to vyvolávající entitu a směr vyvolání kouzla. Pro každé kouzlo pak existuje factory, kterou definuje rozhraní `ISpellFactory` s následujícími položkami:

- `CastingSequence` – sekvence symbolů typu `ISpellSymbol`, které vyvolají kouzlo.
- `Name` – název kouzla.
- `Difficulty` – modifikátor obtížnosti při vyvolávání kouzla.
- `Duration` – doba, po kterou šampion nemůže znovu vyvolávat kouzla.
- `Skill` – dovednost typu `ISkillFactory` nutná pro vyvolání kouzla.
- `SkillLevel` – úroveň předchozí dovednosti nutná pro vyvolání kouzla.
- `CastSpell` – funkce, která vytvoří kouzlo typu `ISpell`.

Za jednotlivými symboly potom stojí rozhraní `ISpellSymbol`, které definuje název symbolu a cenu many pro každý power level za použití symbolu. Pro samotné vyvolávání kouzel je pak použita třída `SpellCastingManager`, která implementuje rozhraní `ISpellCastingManager`.

Implementace kouzel lze nalézt ve jmenném prostoru:
`DungeonMasterEngine.DungeonContent.Magic`.

3.8 Projektily – `Projectile`

Objekty létající vzduchem – aniž by interagovaly s dlaždicemi, oproti tomu jak to je v případě entit – reprezentuje třída `Projectile`. Tato třída je použita například pro kouzla nebo při hodech předmětů. Projektily se samy zaregistrují do kolekce `DungeonLevel.Updateables`, odkud jsou následně aktualizovány. Po nárazu může projektil způsobit výbuch a zranění entitám, které reprezentuje třída `Impact`. Vypuštění projektilu je provedeno zavoláním asynchronní funkce `Projectile.Run` (viz sekce 3.1.1), která provádí celý pohyb a zajišťuje interakci projektilu.

Projektily jsou ve jmenném prostoru:
`DungeonMasterEngine.DungeonContent.Projectiles`.

3.9 Renderery

Jak bylo zmíněno v analýze, engine odděluje zobrazovací vrstvu od zbytku enginu (viz sekce 2.14). Všechny objekty, které potřebují mít grafický výstup, by měly implementovat rozhraní `IRenderable`, které vyžaduje položku typu `IRenderer`. Stěžejní funkce tohoto rozhraní jsou funkce `Render` a `Interact`.

Nejdůležitější z parametrů těchto funkcí je dosavadní transformace. Ta obsahuje složeninu transformací složenou z jednotlivých transformací na cestě z kořene stromu reprezentující závislosti rendererů na sobě. Takže například kořen stromu 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 používán u statických objektů, jako jsou dlaždice, stěny, výklenky a podobně. Pro pohybující

se objekty je používána absolutní pozice a renderery těchto objektů transformují objekty přímo na jejich pozici. Volba mezi těmito dvěma způsoby pozicování stojí vždy za zamyšlení.

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 rendererů je jasné, že toho může být dosaženo vytvořením událostí na renderovaných objektech, které si renderer zaregistruje. Dovedeme si představit, že by šla s takovým návrhem velmi jednoduše udělat grafická vrstva, která bude velmi pěkná, bude mít kvalitní 3D modely a animace. Zabralo by to jen spoustu času a byla by k tomu potřeba spousta grafiků.

Při rozšíření 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. Rozšířením již existujícího rendereru si 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 přetěžovat, obsahují také 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 objektů, je nutné tuto vrstvu propojit i s interakcí. 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 přetypovat. V této implementaci je použit paprsek (`Ray`). Celý tento koncept interactoru by šel udělat genericky, ale prolínal by se celou strukturou rendererů a z toho důvodu jsme se rozhodli v tomto místě ustoupit a udělat situaci jednoduší na úkor kontroly za překladu.

3.10 Builder herních úrovní

Nyní, když máme rozebrané všechny části enginu, které je potenciálně možné rozšířit, můžeme si vysvětlit, jak se vše sestaví dohromady v herní úrovně.

3.10.1 Vlastní implementace builderu

Jak již bylo zmíněno, builder musí implementovat rozhraní `IDungeonBuilder` (viz sekce 3.1). Toto rozhraní je potom dotazované jádrem enginu s požadavkem o načtení určité herní úrovně. Je úkolem builderu rozhodnout, jak bude jednat v případě, že je podán několikrát dotaz na stejnou mapu. Typicky je tedy nutné rozhodnout se, jestli načtené mapy kešovat či nikoliv.

Správně implementovaný builder by měl načítat mapy zhruba tímto postupem.

1. Vytvoření dlaždic a naplnění jejich inicializátorů. To znamená vytvořit strany dlaždic a přepínače či předměty v nich obsažené.

2. Nastavení sousedů dlaždic do inicializátorů.

3. Vytvoření samotné herní úrovně `DungeonLevel`.

4. Dokončení inicializace dlaždic skrze inicializátory.

5. Případné kešování herní úrovně pro případný opakovaný požadavek.

3.10.2 Builder Dungeon Masteru – `LegacyMapBuilder`

Sestavení herní úrovně pro hru Dungeon Master zajišťuje třída `LegacyMapBuilder` z již připravených dat v podobě třídy `DungeonData`. Tato třída používá ještě ke své funkci další buildery, které mohou mít opět další – celá jejich struktura viz obr. 3.13. Každý tento builder má běžně referenci na `LegacyMapBuilder`, aby měl přístup k celému kontextu. Vlastnosti kontextu lze rozdělit na dvě skupiny:

- data, která se načítají pro každou herní úroveň znovu,
- data, která jsou inicializovaná pouze při vytváření builderu.

Do první skupiny patří následující důležité vlastnosti:

- `CurrentLevelIndex` – určuje pořadí právě sestavované herní úrovně.
- `CurrentMap` – obsahuje data právě sestavované herní úrovně.
- `TilePositions` – kolekce, v které lze za pomocí pozice získat dlaždici, která ji náleží. Do této kolekce by se měly přidávat dlaždice s jejich pozicí při jejich vytváření. Kolekce je pak předána do výsledné herní úrovně reprezentované třídou `DungeonLevel`.
- `TileInitializers` – tato kolekce obsahuje inicializátory dlaždic, které by se měly do kolekce přidávat při vytváření odpovídajích dlaždic. Po vytvoření všech dlaždic je skrze tyto inicilizéry dokončena inicilizace dlaždic zavoláním funkce `InitializerBase.Initialize`.

Dále jsou pak v této skupině seznamy s texturami pro dveře, dekorace zdí, dekorace podlah a šampiony. Tyto seznamy mají prvky seřazeny tak, aby odpovídaly správným texturám při použití indexů dekorací specifikovaných v datech. Dále obsahuje ještě texturu pro dveře, zeď, tlačítko dveří a teleport.

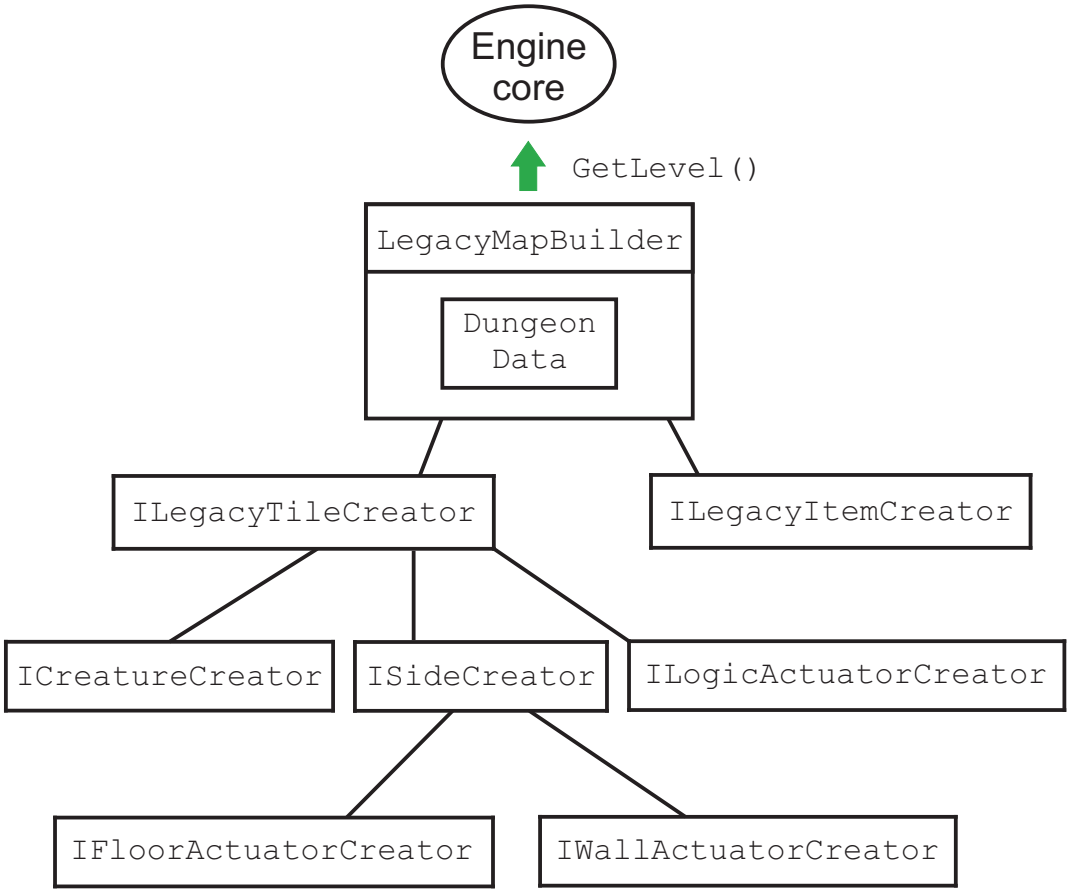
V druhé část jsou pak obsaženy následující vlastnosti:

- `Data` – veškerá data herních úrovní získané z parsovací vrstvy (viz sekce 2.3) v objektu typu `DungeonData`.
- `Factories` – objekt obsahující všechny factories (viz sekce 3.1).
- `ItemCreator` – objekt typu `IItemCreator` zajišťující vytváření sbíratelných předmětů z odpovídajících dat. Konkrétní implementaci je pak možné předat do konstruktoru při vytváření builderu.
- `TileCreator` – objekt `ITileCreator` zajišťující tvorbu dlaždic z odpovídajících dat. Konkrétní implementaci je pak možné předat do konstruktoru při vytváření builderu.

- **`CreatureToken`** a **`ChampionToken`** – tokeny šampionů a nepřátelských entit (viz. relace mezi entitami v sekci 3.4.5)

Funkce **`GetLevel`** potom provede postup zmíněný v předchozí sekci (viz 3.10.1). Pro načtení dalších vlastních textur před sestavováním herní úrovně je možné přetížit funkce **`InitializeMapTextures`**. Pro získání factory dle globálního identifikátoru (viz sekce 2.2.2) existuje funkce **`GetItemFactory`**, která lze také případně přetížit. Pro získání již vytvořeného objektu dlaždice lze pak použít virtuální asynchronní funkci **`GetTargetTile`**. Funkci je možné zavolat ještě když objekt pro dlaždici na dané pozici neexistuje. V tom případě se čeká na promise, která se naplní po vytvoření všech dlaždic. Funkce také řeší přesměrování pozice na dlaždici typu *zeď* na odpovídající dlaždici typu *podlaha* (viz sekce 2.6).

Buildery jsou ve jmenném prostoru **`DungeonMasterEngine.Builders`**.



Obrázek 3.13: Ilustrace struktury builderu.

Builder dlaždic – `ILegacyTileCreator`

Rozhraní **`ILegacyTileCreator`** má následující položky:

- **`MiniMap`** – obsahuje texturu herní úrovně s dlaždicemi, které byly vytvořeny.
- **`GetTile`** – tato funkce by měla z odpovídajících dat předaných v konstruktoru vytvořit dlaždici a její inicializátor přidat do kolekce

`LegacyMapBuilder.TileInitializers`. Dale by také měla zaznamenat dlaždici do minimapy.

Implementací tohoto rozhraní je třída `LegacyTileCreator`, která dlaždice třídí pomocí návrhového vzoru visitor. Pro rozšíření této třídy je možné přetížit funkci `GetTile`. Tato třída pak ještě obsahuje buildery pro tvorby stěn dlaždice, nepřátel a logických přepínačů. Konkrétní implementace těchto tříd lze vložit do konstruktoru.

Builder stran dlaždic – `ISideCreator`

Rozhraní `ISideCreator` obsahuje funkce `SetupSideAsync` a `SetupSideAwaitableAsync`, které vytvoří stěny dlaždic a vloží je do předaného inicializátoru. Obě funkce dělají to samé, akorát druhou z nich je možné awaitovat. Konkrétní implementaci tohoto rozhraní tvoří třída `SideCreator`, která obsahuje buildery pro vytvoření přepínačů na zdech a podlaze. Implementace těchto buildrů lze opět specifikovat v konstruktoru. Rozšířením této třídy lze pak přetížit funkce `GetCeelingSide`, `GetWallSide` a `GetFloorSide`.

Builder přepínačů na zdi – `IWallActuatorCreator`

Rozhraní `IWallActuatorCreator` vyžaduje funkci `ParseActuatorX`, která ze zadané sekvence dat senzorů a předmětů ve zdi vytvoří přepínač. Pro toto rozhraní existuje implementace `WallActuatorCreator`, u které lze při rozšíření přetížit následující funkce:

- `ParseSensor` – funkce dle typu senzoru v jeho datech, vytvoří senzor pro tento engine.
- `CreateWallDecoration` – funkce vytvoří dekoraci odpovídající přepínači (viz sekce 3.3.1).

Builder přepínačů na podlaze – `IFloorActuatorCreator`

Rozhraní `IFloorActuatorCreator` definuje funkci `GetFloorActuator`, která ze zadané sekvence dat senzorů vytvoří přepínač. Třída `FloorActuatorCreator` potom implementuje toto rozhraní a při jejím rozšíření lze přetížit funkci `CreateSensor`. Tato funkce vytvoří z dat o senzoru senzor pro tento engine.

Builder nepřátelský entit – `ICreatureCreator`

Toto rozhraní definuje jedinou funkci, a to `AddCreatureToTile`, která by měla rozparsovat data o skupině nepřátel a vytvořit z nich odpovídající entity. Nepřátelské entity se potom zaregistrují na událost indikující dokončení inicializace všech dlaždic a obživnou na specifikované dlaždici. Třída `CreatureCreator` implementuje toto rozhraní, kdy její funkce `AddCreatureToTile` lze přetížit.

Builder logických přepínačů – `ILogicActuatorCreator`

Toto rozhraní implementuje třída **`LogicActuatorCreator`** a obsahuje funkci **`SetLogicActuator`**, která vytvoří logické senzory a nastaví je do předaného inicializátoru. Přetížením funkce **`LogicActuatorCreator.ParseLogicSensor`** lze dodat podporu pro další typy senzorů.

Builder předmětů – `ILegacyItemCreator`

Tímto rozhraním lze za pomocí jeho funkce **`CreateItem`** vytvářet předměty na základě odpovídajících dat. Toto rozhraní implementuje třída **`LegacyItemCreator`**, která vytváří předměty pomocí návrhového vzoru visitor. Přetížením její funkce **`CreateItem`** je možné přidat podporu pro další předměty.

3.11 Implemetace hráče – `ILeader`

Hráče reprezentuje rozhraní **`ILeader`**, které má následující položky:

- **`LocationChanged`** – událost vyvolaná při změně dlaždice hráče. Jádro je na tuto událost zaregistrované a při jejím vyvolání provede aktualizaci viditelných dlaždic a pokus o načtení případných dalších úrovní (viz sekce 3.1).
- **`PartyGroup`** – hráčova skupina obecných entit. Jádro tuto vlastnost používá k výpočtu osvětlení dle předmětů, které entita obsahuje.
- **`Hand`** – ruka hráče, do které se vloží předmět po kliknutí na něj nebo jinými akcemi.
- **`Interactor`** – objekt, který je používán k interakci s objekty. Používají ho renderery, které musí vědět jakého typu je. Na tento typ ho pak musí přetypovat.
- **`Layout`** – určuje layout hráče na dlaždici.
- **`Leader`** – šampion, který je označený jako vůdce. Pomocí jeho statistik se pak určuje síla hodu předmětem.
- **`View`** – matice zobrazení.
- **`Projection`** – matice projekce.
- **`Enabled`** – určuje, zda má hráč reagovat na vstup z periferií – při zobrazení konzole je například zablokován.
- **`AddChampoinToGroup`** – funkce pro přidání šampiona do skupiny.
- **`Draw`** – funkce, v které lze vykreslovat objekty.
- **`Update`** – funkce, která je volána v každém cyklu herní smyčky.

Implementaci hráče pro hru Dungeon Master tvoří třída **`LegacyLeader`** ve jmenném prostoru **`DungeonMasterEngine.Player`**. Jelikož v jádře se specifikuje typovým parametrem přímo typ hráč, je možné přistupovat z front endu i k položkám, které nejsou v rozhraní **`ILeader`**.

3.12 Herní konzole – `GameConsole`

Tato práce obsahuje vlastní implementaci konzole pro zadávání složitějších úkonů hráči. Konzole pak může vykonávat příkazy, které jsou reprezentovány rozhraním `IInterpreter`. Od konzole lze vytvořit pouze jedna instance a to skrze statickou funkci `InitializeConsole`, které lze předat sadu interpretrů příkazů.

3.12.1 Interpreter – `IInterpreter`

Interpreter slouží ke komunikaci s hráčem skrze textový vstup a výstup. Na základě odpovědí od hráče provede příkaz, který reprezentuje.

Rozhraní `IInterpreter` má typový parametr určující kontext a obsahuje následující položky:

- `Input` – `TextReader` obsahující vstupní stream typu `KeyboardStream` z klávesnice.
- `Output` – `TextWriter` obsahující výstupní stream typu `ScreenStream` do konzole.
- `ConsoleContext` – jakýkoliv kontext, jehož typ lze zvolit jako typový parametr.
- `Parameters` – pole textových parametrů, s kterým byl interpreter vytvořen.
- `CanRunOnBackground` – vlastnost určující, zda lze konzole minimalizovat v průběhu provádění tohoto interpretru.
- `Run` – asynchronní funkce obsahující algoritmus interpretru.

Ke každému interpretru pak existuje factory, kterou definuje rozhraní `ICommandFactory` s následujícími položkami:

- `CommandToken` – token, ke kterému náleží tato factory.
- `HelpText` – text reprezentující pomocnou zprávu, kterou můžou použít další příkazy.
- `ParameterParser` – tokenizer parametrů definovaný rozhraním `IParameterParser`, které obsahuje jedinou funkci `ParseParameters`, která rozdělí jednotlivé parametry do pole. Tvůrce interpretru by měl tento tokenizer použít pro získání pole parametrů, které následně předá interpretru.
- `GetNewInterpreter` – vytvoří novou instanci interpretru.

Je zodpovědností tvůrce interpretru, aby mu nastavil vlastnosti `Input`, `Output`, `ConsoleContext`, `Parameters` a vyvolal funkci `Run`. Tato funkce je asynchronní, takže je možné čekat na její dokončení bez blokování vlákna. Pro čtení vstupu je pak nutné používat asynchronní funkci. Ke čtení `KeyboarStreamu` se potom používá `KeyboardStreamReader`, která dokáže číst vstup čistě asynchronně bez nutnosti použití dalších vláken.

Jednotlivé příkazy lze najít ve jmenném prostoru:
`DungeonMasterEngine.GameConsoleContent`.

3.12.2 Implementace konzole

Konzole má pak interpreter typu **BaseInterpreter** implmentující rozhraní **IInterpreter**. Tomuto interpreteru jsou při jeho vytvoření předány všechny factories interpretrů, které podporuje. Ve funkci **Run** potom obsahuje nekonečnou smyčku, jež čte vstup, dle kterého se pokusí najít factory odpovídajícího interpretru. Pomocí factory pak vytvoří daný interpreter, nastaví mu všechny vlastnosti zmiňované v předchozí sekci a předá mu kontrolu zavoláním funkce **Run**. Kromě tohoto způsobu ještě umožňuje vyvolat daný interpreter, který je mu předán do funkce **RunCommand**.

Následující kód pak ukazuje smyčku **BaseInterpreteru** jako další příklad užití asynchronních funkcí v enginu.

```
async Task Run()
{
   Output.WriteLine("Welcome!");

   while (Running)
   {
      RunningCommand = await GetInterpreter();

      if (RunningCommand != null)
      {
         RunningCommand.Input = Input;
         RunningCommand.Output = Output;
         RunningCommand.ConsoleContext = ConsoleContext;

         await RunningCommand.Run();
         RunningCommand = null;
         Output.WriteLine();
      }
      else
      {
         Output.WriteLine("Unrecognized command!");
      }
   }

   Output.WriteLine("Have a nice day.");
}
```

Jako kontext potom interpreter konzole používá **IConsoleContext**, který definuje kolekci použitých interpretrů a referenci na jádro enginu.

3.13 Neimplementované funkce enginu

Tato sekce obsahuje převážně seznamy neimplementovaných funkcí enginu pro hru Dungeon Master. Pokud je určitá množina funkcí jasně stanovena, jsou u ní uvedeny i implementované funkce.

3.13.1 Akce

Seznam všech akcí může být nalezen v komunitní dokumentaci [7] nebo v dekompilovaných zdrojových kódech [10] v souboru `DEFS.H` na řádku 1467.

- Třída `SwingAttackFactory` implementuje následující akce: BASH, HACK, BERZERK, KICK, SWING a CHOP.
- Třída `MeleeAttackFactory` implementuje následující akce: DISRUPT, JAB, PARRY, STAB, STAB, STUN, THRUST, MELEE, SLASH, CLEAVE a PUNCH. V této třídě chybí podpora pro útok proti nemateriálním nepřátelským entitám. Také chybí reakce na obtížnost herních úrovní, kterou lze získat skrze vlastnost `DungeonMap.Difficult`. Implementace výběru nepřátelského cíle se také drobně liší od originálu.
- Třída `ThrowActionFactory` implementuje akci THROW.
- Následující akce enginu nejsou implementovány: BLOCK, BLOW_HORN, FLIP, WAR_CRY, CLIMB_DOWN, FREEZE_LIFE, HIT, FIREBALL, DISPELL, CONFUSE, LIGHTNING, INVOKE, SHOOT, SPELLSHIELD, FIRESHIELD, FLUXCAGE, HEAL, CALM, LIGHT, WINDOW, SPIT, BRANDISH a FUSE. Pro případné doimplementované akce je nutné zaregistrovat jejich factories ve funkci `LegacyFactories.GetFightActionsFactories`.

3.13.2 Přepínače

Seznam všech typů přepínačů může být nalezen v komunitní technické dokumentaci [7] nebo v dekompilovaných zdrojových kódech [10] v souboru `DEFS.H` na řádku 1467.

- Implementované senzory:
  - Senzory na zdech – 1, 2, 3, 4, 5, 6, 11, 12, 13, 16, 17 a 127.
  - Senzory na podlaze – 1, 2, 3, 4, 6, 7 a 8.
- Neimplementované senzory:
  - Senzory na zdech:
    - 7, 8, 9, 10, 14 a 15 – senzory střílející rakety.
    - 18 – senzor pro konec hry.

    Případné dodělané senzory na zeď je nutné přidat do funkce `WallActuatorCreator.ParseSensor`.
  - Senzory na podlaze:
    - 5 – senzor, který může být pouze na schodech – do verze 2.1 originální hry je nefunkční,
    - 6 – generátor nepřátelských entit,
    - 9 – version checker.

    Případné dodělané senzory na podlahu je nutné přidat do funkce `FloorActuatorCreator.CreateSensor`.

3.13.3 Nepřátelské entity

- Chybí implementace rozhraní `IGroupLayout` pro entity obsazující polovinu prostoru dlaždice.
- Nepřátelské entity se neotáčejí a vidí všemi směry. Nicméně při pohybu by měly vidět pouze před sebe a při změně pohybu by se měly otáčet.
- Chybí speciální implementace pro některé nepřátelské entity. Například nepřátelská entita `Black Flame` by se neměla pohybovat a ohnivé útoky by ji měly posilovat. Dalším příkladem je entita `Giggler`, která v originální hře může šampionům krást věci z rukou. Kompletní seznam těchto speciálních funkcí není znám, nicméně lze je nalézt různě roztroušené ve zdrojových kódech [10].
- Chybí implementace útoků na dálku pro určité nepřátelské entity. Kód vykonávající útoky je ve funkci `F207_xxxx_GROUP_IsCreatureAttacking` v dekompilovaných zdrojových kódech hry [10]. Implementace útoku na blízko je ve třídě `CreatureAttack`.
- V originální hře lze nepřátelské entity některými útoky vyděsit tak, že před hráčem utíkají – tato funkce opět není implementována.
- Pozůstatky po nepřátelských entitách definované v souboru `GRAPHICS.DAT` nejsou implementované. Nějaké informace je možné nalézt v dokumentaci nepřátelský entit [8] nebo v dokumentaci pro část souboru `GRAPHICS.DAT` [9] viz „Creature droppings definitions".
- Útok nepřátelských entity by měl mít vyšší účinnost, když skupina odpočívá (viz třída `CreatureAttack`).

Pro nové implementace nepřátelský entity je třeba vytvořit jejich factories a zaregistrovat je ve funkci `LegacyFactories.InitCreatureFactories`

3.13.4 Předměty

Některé předměty mohou modifikovat nositelovy vlastnosti. Implementace této funkce u předmětů chybí. Jak který předmět modifikuje vlastnosti svého nositele lze buď nalézt různě roztroušené ve dekompilovaných zdrojových kódech [10], nebo v komunitní dokumentaci pro předměty [5]. Nicméně u vlastností je k tomuto účelu připravena kolekce `IProperty.AdditionalValues`, do které by se tyto modifikátory zaregistrovaly při oblečení předmětu.

Některé předměty by také mělo jít prohodit skrze dveře, jejich seznam je pevně stanoven v kódu hry [10] ve funkci `F217_xxxx_PROJECTILE_HasImpactOccured`

3.13.5 Kouzla

Následuje seznam implementovaných kouzel:

- Třída `ExplosionProjectileSpellFactory` implementuje následující vzdálená útočná kouzla: fire ball, weaken nonmaterial beings, poison bolt,

poison cloud, lightning bolt. Některá z těchto kouzel mohou vyvolávat i nepřátelé, a tak tyto kouzla mohou zranit jak šampiony tak nepřátelské entity. V originální hře jsou pro výpočet výsledného zranění používány následující dvě funkce:

  - `F190_zzzz_GROUP_GetDamageCreatureOutcome` – provádí výpočet zranění pro nepřátelské entity,
  - `F321_AA29_CHAMPION_AddPendingDamageAndWounds_GetDamage` – provádí výpočet zranění pro šampiony, navíc oproti předchozí funkci určuje, zda útok zranil nějakou část těla šampiona.

  Zde nastává problém s tím, že jsme pro šampiony i nepřátele použili stejnou abstrakci entit. Proto, aby implementace zranění byla stejná jako v originální hře, by útočná kouzla musela zjišťovat konkrétní typ entity. To by ale vedlo na nepřehledný a hlavně špatně rozšiřitelný kód. Z toho důvodu jsme se rozhodli v tomto případě použít pro výpočet zranění pouze funkci určenou pro nepřátelské entity. Nicméně lepším řešením by bylo detailně porozumět kódu obou funkcí a na jejich základě vytvořit jednotnou variantu, která by rozdíly mezi entitami řešila pomocí jejich vlastností.

- Třída `PotionSpellFactory` implementuje následující kouzla vytvářející lektvary: poison potion, dexterity potion, strength potion, wisdom potion, vitality potion, cure poison potion, stamina potion, shield potion, mana potion a health potion.

- Kouzlo torch je implementované třídou `MagicTorchSpellFactory`.

- Kouzlo open door je implementované třídou `OpenDoorSpellFactory`.

Následuje seznam neimplementovaných kouzel: fire shield, shield, darkness, light, magic footprints, see through walls, invisibility, zokathra spell,

Kouzla ovlivňující vlastnosti jako magická pochodeň, magický štít, štít proti ohni nebo štít ovlivňují každého šampiona zvlášť, oproti originálu, kde jsou to vlastnosti skupiny.

Factories doimplementovaných kouzel je nutné přidat ve funkci `LegacySpellCreator.GetSpell`.

3.13.6 Další neimplementované nebo pozměněné funkce

- Šampiony nelze ve skupině rozmisťovat – funkcionalita lze doimplementovat do třídy `LegacyLeader`.

- Aktualizace vlastností šampionů v originální hře je ovlivněna kritériem, závislým na způsobu počítání času v originální hře. Význam tohoto kritéria se nepodařilo rozluštit a je místo něj použita náhodná hodnota v rozsahu hodnot, které kritérium nabývá v původní hře viz `Champion.F331_auzz_CHAMPION_ApplyTimeEffects_COPYPROTECTIONF`.

- Nejsou známy hodnoty pro převod „light power"na "light amount"použité pro výpočet osvětlení. Hodnoty jsou zvoleny tak, aby přibližně odpovídaly hodnotám při hraní originální hry, viz funkce `DungeonBase.UpdateLight` resp. `F337_akzz_INVENTORY_SetDungeonViewPalette`.

Závěr

V rámci této práce byl naimplementovaný engine pro hru Dungeon Master. Engine obsahuje podporu pro všechny funkce z originální hry. Nicméně všechny konkrétní funkce nejsou naimplementované, což dává možnost navázat na práci případným následovníkům. Pro implementaci byl použit jazyk C#, platforma .NET a framework MonoGame. Většina částí enginu je jednoduše rozšiřitelná či modifikovatelná, což vede k dobré udržitelnost celého systému. Primárně je engine určený pro hru Dungeon Master a dokáže si herní úrovně načíst z originální binárních dat. Avšak načítání herních úrovní je v oddělené vrstvě, a proto je možné tuto vrstvu upravit kvůli případným rozšířením nebo ji je možné nově naimplementovat i pro jiné vstupní formáty. Z toho důvodu lze tento engine využít i pro implementaci jiných her na podobných principech. Engine má také oddělenou renderovací vrstvu, což umožňuje jednoduše doimplementovat hře lepší „look and feel". Tento projekt může sloužit pro vzdělávání jazyka C# a objektově orientovaného programování. Studenti si tak můžou vyzkoušet do enginu dodělat nové komponenty a tím si vyzkoušet objektově orientované programování v praxi.

Možný budoucí vývoj

Na tuto práci je potenciálně možné navázat v následujících bodech:

- Dodělání všech funkcí Dungeon Masteru tak, aby bylo možné hru dohrát až do konce hry (viz sekce 3.13).
- Dodělání zobrazovací vrstvy tak, aby odpovídala vzhledu původní hry Dungeon Master.
- Dodělání lepší 3D zobrazovací vrstvy.
- Udělat podporu pro textový výstup zobrazovací vrstvy pro možnost automatizovaného kontrolování například systémem CodEx [1].
- Vytvořit editor pro hru Dungeon Master.
- Vytvořit jinou hru na tomto enginu.
- Upravit engine tak, aby nebyl vázaný na stejně velké dlaždice uspořádané do mřížky a umožňoval tak flexibilnější tvorbu herních úrovní.

Seznam použité literatury

[1] CodEx – The Code Examiner. URL http://codex.ms.mff.cuni.cz/project/.

[2] J. A. L. de Farias. MonoGame. 2009. URL http://www.monogame.net/.

[3] C. Fontanel. Technical Documentation – Dungeon Master and Chaos Strikes Back Actions and Combos. 2005. URL http://dmweb.free.fr/?q=node/690.

[4] C. Fontanel. Technical Documentation – Dungeon Master and Chaos Strikes Back Items properties. 2005. URL http://dmweb.free.fr/?q=node/886.

[5] C. Fontanel. Dungeon Master Items. 2005. URL http://dmweb.free.fr/?q=node/259.

[6] C. Fontanel. Dungeon Master and Chaos Strikes Back Spells. 2005. URL http://dmweb.free.fr/?q=node/195.

[7] C. Fontanel. Technical Documentation – File Formats – Dungeon File (DUNGEON.DAT). 2005. URL http://dmweb.free.fr/?q=node/217.

[8] C. Fontanel. Technical Documentation – Dungeon Master and Chaos Strikes Back Creature Details. 2008. URL http://dmweb.free.fr/?q=node/1363.

[9] C. Fontanel. Technical Documentation – Graphics.dat Item 559. 2008. URL http://dmweb.free.fr/?q=node/1395.

[10] C. Fontanel. Back to the source: ReDMCSB. 2014. URL http://www.dungeon-master.com/forum/viewtopic.php?f=25&t=29805.

[11] B. Kernighan and D. Ritchie. *The C Programming Language*. 1978.

[12] S. Mourier. Html Agility Pack. 2012. URL https://htmlagilitypack.codeplex.com/.

Příloha A – Uživatelská dokumentace

Pro spuštění reimplementace Dungeon Masteru je zapotřebí alespoň:

- Windows x86
- DirectX 9.0c runtime
- .NET 4.6

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

3.14 Mechaniky ve hře

Hráč reprezentuje vůdce skupiny šampionů. Může ovládat jejich pohyb, předávat jim předměty, donutit je k boji, kouzlení, konzumaci jedlých předmětů nebo lektvarů. Každý šampion má sadu vlastností a dovedností, v kterých je schopný se zdokonalovat získáváním zkušeností. Zkušenosti lze získat bojem na prázdno, bojem proti nepřátelům, kouzlením nebo jen použitím některých přepínačů. V herních úrovních jsou přepínače na zdech, které lze aktivovat stiskem klávesy **`ENTER`**. 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 šampiona či do jeho batohu. Přepínače pak mohou aktivovat nebo deaktivovat určité objekty ve hře, jako jsou například dveře, teleporty, jámy, otevírací zdi, atd. Některé dveře je také možné rozbít útokem. Některé teleporty jsou pouze na konkrétní typy objektů. Do jam lze spadnout, ale lze se z nich většinou teleportem dostat ven. V chodbách je tma, a proto je nutné používat pochodně – což lze udělat vložením do jedné z ruk šampiona – nebo je nutné vyvolat magické pochodně.

3.15 Cíl hry

Hra není úplně kompletní, proto ji není možné zcela dohrát. Z toho důvodu by se za cíl hry dalo považovat dostaní se do poslední herní úrovně, kde nebudou neimplementované funkce bránit dalšímu postupu.

3.16 Tutorial

(a) obraz s šampionem (b) předmět jablko

Obrázek 3.14

Po spuštění hry se objevíme v první úrovní hry, kde si je nutné vybrat sadu šampionů. Mezi dlaždicemi se můžeme pohybovat pomocí kláves W, S, A, D tj. dopředu, dozadu, doleva, doprava. Dále je možné se rozhlížet pomocí šipek na klávesnici. Teď když víme, jak se pohybovat jděme chodbou dokud nenarazíme na nápis „HALL OF CHAMPION". Poté jděme hned vpravo směrem na pozici (7, 9), dokud nenarazíme na obraz na obrázku 3.14a. Obrazy v sobě mají uložené šampiony, stisknutím klávesy **ENTER** při namířeném kurzoru na obraz, se zobrazí konzole, která nám vypíše statistiky šampiona a nabídne, zda ho chceme buď reinkarnovat nebo osvobodit – v prvním případě lze specifikovat navíc jméno šampiona. V obou případech ale dojde k přidání šampiona do skupiny. Po přidání šampiona do skupiny se vpravo zobrazí jeho základní statistiky, tj. zdraví, výdrž a mana. Konzoli lze zobrazit či skrýt stiskem klávesy **TAB**. Pojděme dále na pozici (9, 13), kde je další šampion, kterého přidejme do skupiny. Hráč může mít až čtyři šampiony, nicméně dva nám budou pro ukázku postačovat. První šampion je dobrý bojovník a druhý dobrý čaroděj, takže si s nimi budeme moci ukázat jak souboj tak kouzla.

Vraťme se zpět k původnímu obrazu, vlevo od něj je brána a před ní na podlaze přepínač. Tento přepínač se aktivuje, pokud na něj vkročí hráč s alespoň jedním šampionem. Projděme tedy bránou, dokud na zemi neuvidíme jablko (viz obr. 3.14b). V herních úrovních jsou všude roztroušeny potraviny, které jsou nutné k přežití skupiny. Ukázáním kurzorem na jablko a stisknutím klávesy **ENTER** se jablko dostane do hráčovy ruky. S předměty v hráčově ruce lze provádět akce. Zobrazme konzoli a napišme do ní příkaz **hand**, tím se zobrazí popis věci v ruce. Přidáním parametru **use** lze vybrat šampiona, který jablko sní a sníží se tak jeho hlad. Jděme dále, dokud vpravo neuvidíme schody vedoucí o úroveň níže. Na zemi lze nalézt lahev na vodu a svitek. Šampioni potřebují kromě jídla i pití – do lahve lze nalít voda z fontánek, které jsou různě rozmístěné po herních úrovních. Vezměme si tedy tuto lahev do ruky a pomocí následujícího příkazu ji vložme do batohu druhého šampiona.

```
> hand take
Select index:
```

```
0 HALK
1 TIGGY
> 1
Select index:
0 HeadStorageType
1 NeckStorageType
2 TorsoStorageType
3 ActionHandStorageType
4 ReadyHandStorageType
5 LegsStorageType
6 FeetsStorageType
7 BackPackStorageType
8 PouchStorageType
9 SmallQuiverStorageType
10 BigQuiverStorageType
> 7
```

Text svitku lze přečíst příkazem **hand**, pokud ho máme v hráčově ruce. Tento svitek říká, že kouzelnou pochodeň můžeme vyvolat pomocí symbolu ful. Sekvence symbolů pro všechna kouzla mohou být nalezena v dokumentaci [6].

```
> hand
Scrool "INVOKE FUL FOR A MAGIC TORCH"
```

Pro vyzkoušení kouzel pojděme po schodech dolů, kde je naprostá tma. Kouzlit lze pomocí příkazu **spell** a to buď v interaktivním módu, nebo pouze pomocí parametrů. Následující příklad ukazuje interaktivní mód, kdy je nejprve proveden dotaz na šampiona, který bude kouzlo vyvolávat a dále už lze odříkávat jednotlivé symboly zakončené příkazem **cast**. První symbol je tzv. power symbol, který určuje sílu kouzla. Každé vyvolání symbolu pak šampiona stojí manu. Každé kouzlo také specifikuje úrovně dovedností, které jsou nutné, aby bylo vyvolání úspěšné. Pokud má šampion nižší úroveň, tak stále může kouzlo vyvolat ale s mnohem menší pravděpodobností. Úspěšný i neúspěšný pokus o vyvolání kouzla šampionovi rovněž přidá zkušenosti – avšak neúspěšný pokus daleko méně.

```
> spell
Select index:
0 HALK
1 TIGGY
> 1
Specify one of the following symbols: lo, um, on, ee,
pal, mon (sorted by difficulty ascending).
Write symbol or cast it by "cast"
> lo
ManaProperty: 27 of 35 ; -8
Write symbol or cast it by "cast"
> ful
ManaProperty: 22 of 35 ; -5
Write symbol or cast it by "cast"
> cast
```

```
MagicalLightProperty: 409 of 2147483647 ; 409
Spell successfully casted!
```

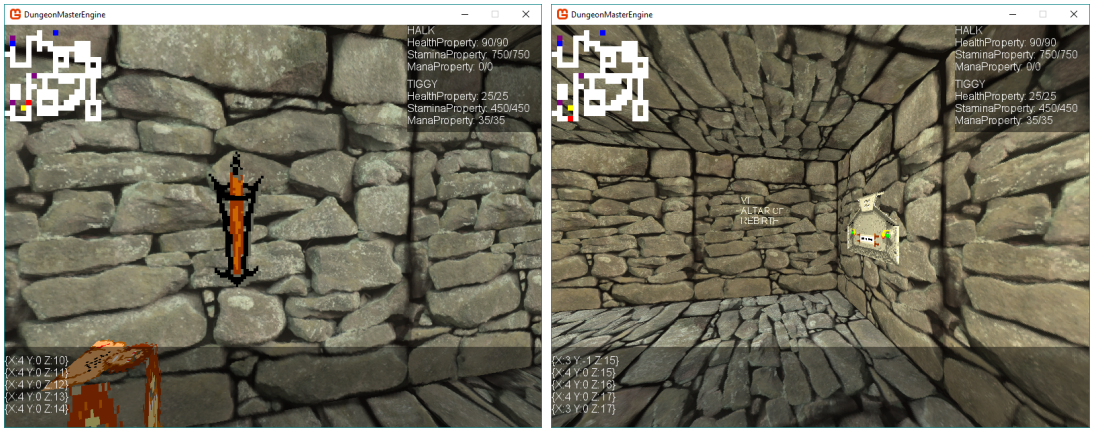
Toto kouzle je slabé, proto můžeme jeho efekt navýšit opětovným vyvoláním. Nyní tak ale proveďme druhou variantou pomocí parametrů, kde první specifikuje index šampiona, druhý power symbol a ostatní už jsou běžné symboly.

```
>spell 1 lo ful
ManaProperty: 14 of 35 ; -8
ManaProperty: 9 of 35 ; -5
MagicalLightProperty: 818 of 2147483647 ; 409
Spell successfully casted!
```

Jak vidíme ve výstupu, na další vyvolání kouzla už nám nebude stačit mana. Rychlé obnovení zdraví, výdrže a many lze provádět pomocí odpočinku. K němu lze využít příkaz **champion** s parametrem **sleep** – probuzení lze provést příkazem **wake**, viz následující ukázka.

```
> champion sleep
Write 'wake' to wake the group.
FoodProperty: 1541 of 2048 ; -2
WaterProperty: 1598 of 2048 ; -1
StaminaProperty: 750 of 750 ; 0
StaminaProperty: 443 of 450 ; -7
ManaProperty: 3 of 35 ; 1
...
FoodProperty: 1497 of 2048 ; -2
WaterProperty: 1483 of 2048 ; -1
StaminaProperty: 450 of 450 ; 0
FoodProperty: 1465 of 2048 ; -2
WaterProperty: 1560 of 2048 ; -1
StaminaProperty: 750 of 750 ; 0
wake
Party had just woken up.
```

Odpočinek zvyšuje u šampionů rychleji hlad a žízeň, proto je třeba doplňovat pití a jídlo.



(a) držák na pochodeň (b) VI Altar Rebirth

Obrázek 3.15

Světlo je stále slabé, pojďme tedy zpět po schodech na pozici (4, 14), kde je na zdi pochodeň (viz obr. 3.15a). Stiskem klávesy **ENTER** a kurzorem namířeným na držák pochodně se pochodeň přemístí hráči do ruky. Tím, že se pochodeň vloží do jedné z ruk některého šampiona, se rozsvítí. Intenzita světla, které pochodeň produkuje postupně klesá, až úplně zhasne. Vložme tedy pochodeň například prvnímu šampionovi do druhé ruky následujícím příkazem.

```
> hand take
Select index:
0 HALK
1 TIGGY
> 0
Select index:
0 HeadStorageType
1 NeckStorageType
2 TorsoStorageType
3 ActionHandStorageType
4 ReadyHandStorageType
5 LegsStorageType
6 FeetsStorageType
7 BackPackStorageType
8 PouchStorageType
9 SmallQuiverStorageType
10 BigQuiverStorageType
> 4
```

Ještě než sejdeme opět dolů, podívejme se na výklenek (viz obr. 3.15b) na pozici (4, 17). Pokud nějaký šampion zemře, zbudou po něm na zemi kosti. Do takto vypadajících výklenků lze pak tyto kosti vložit, čímž dojde k oživení šampiona. Dále v chodbě jsou pak dveře, které lze otevřít červeným tlačítkem na jejich okraji.

(a) mumie útočící na hráče (b) dveře na klíč

Obrázek 3.16

Pojďme ale zpět o úroveň níž a pak vpravo až ke dveřím na pozici (6, 14), za kterými se skrývá mumie (viz obr. 3.16a). Po otevření dveří tlačítkem na nás zaútočí. Když bude mumie na dlaždici před hráčem lze na ní aplikovat útok na blízko. Útok se provede příkazem **fight**, který je rovněž k dispozici v interaktivní či parametrické formě. Nejprve je třeba vybrat šampiona, s kterým chceme útok provést a následně akci, kterou lze provést se zbraní šampiona. Používá se vždy zbraň v akční ruce. Po provedení akce se zbraní se šampionovi sníží výdrž, pokud je výdrž na příliš malé úrovni, mají pak útoky menší poškození. Po provedení akce zároveň šampion nemůže nějakou dobu útočit.

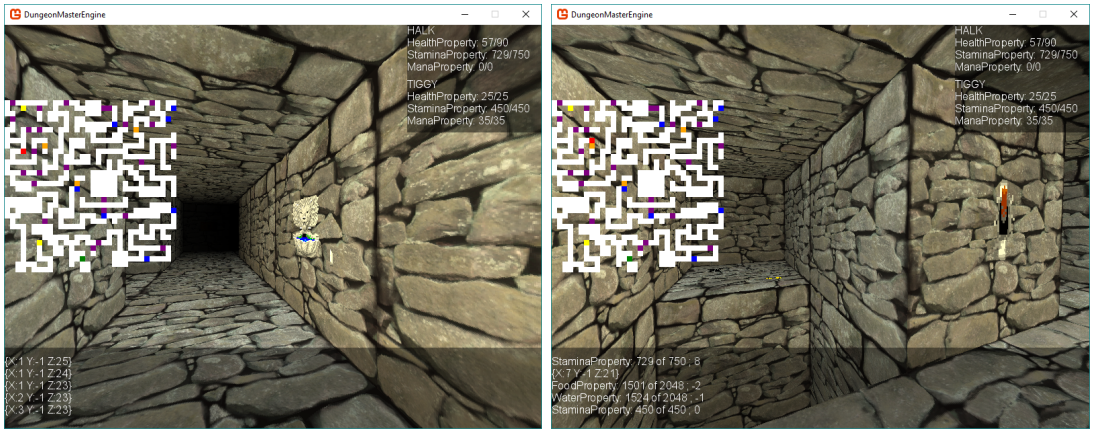
```
> fight
Select index:
0 HALK
1 TIGGY
> 0
Select index:
0 Throw
1 Bash
> 1
HealthProperty: 0 of 20 ; -20
StaminaProperty: 746 of 750 ; -4
StaminaProperty: 736 of 750 ; -10

Action available.
```

V tomto případě jsme měli štěstí, jelikož šampion zabil mumii na jeden útok. Pokud by bylo útoků potřeba provést více, existuje také zmíněná parametrická varianta příkazu, viz následující ukázka útoku na jinou nepřátelskou entitu. První parametr opět specifikuje index šampiona, další pak index akce.

```
fight 0 1
HealthProperty: 37 of 56 ; -11
StaminaProperty: 731 of 750 ; -6
StaminaProperty: 729 of 750 ; -2
```

```
Action available.
```

Nyní projděme dveře na pozici (1, 14), až dojdeme ke dveřím (viz obr. 3.16b), které nemají tlačítko. Vedle nich lze na zemi nalézt klíč, který lze vložit do zdířky vedle na stěně. Tím jsou dveře otevřeny. Na pozici (3, 22) je potom fontánka (viz obr. 3.17a), v které si lze doplnit vodu do lahve nebo prázdné baňky. Další klíč je na pozici (1, 23) a lze s ním otevřít dveře na pozici (4, 19), za kterými jsou dveře další. Pro jejich otevření musíme získat klíč na pozici (7, 23). Cestu k tomuto místu blokují dveře, které lze otevřít pákou na pozici (4, 25) a jáma (viz obr. 3.17b), kterou lze otevřít pákou na pozici (6, 21).



(a) fontánka s vodou (b) jáma zavíratelná pákou

Obrázek 3.17

Tímto jsme prošli základní mechaniky hry, další postup už je na hráči.

3.17 Seznam příkazů a jejich použití

- `hand - použití: hand [put|putsub|take|takesub]`
  - `bez parametrů: zobrazí textový popis předmětu v ruce`
  - `put: vloží předmět z šampionova inventáře do hráčovi ruky`
  - `putsub: vloží předmět z truhly v šampionově inventáři do hráčovy ruky`
  - `take: vezme předmět z hráčovy ruky a vloží jej to šampionova inventáře`
  - `takesub: vezme předmět z hráčovy ruky a vloží jej do truhly v šampionově inventáři`
- `champion - použití: champion list|sleep`
  - `list: zobrazí seznam šampionů a pro vybraného šampiona zobrazí jeho dovednosti`

    - sleep: přivede skupinu šampionů k odpočinku -- probuzení se provede příkazem wake

- spell - použití: spell [championIndex symbolSequence]
    - bez parametrů: interaktivní výběr šampiona a vyvolání kouzla
    - s parametry: šampion daný indexem championIndex vyvolá kouzlo definované sekvencí symbolů symoblSequence
- fight - použití: fight [champoinIndex] [actionIndex]
    - bez parametru: interaktivní výběr šampiona a akce
    - s parametry: šampion s indexem championIndex provede akci s indexem actionIndex
- help - použití: help [commandToken]
    - bez parametrů: zobrazí všechny příkazy
    - commandToken: zobrazí pomocnou zprávu k příkazu commandToken

Příloha B – Struktura přiloženého CD

Zdrojové kódy jsou určeny pro Visual Studio 2015 s doinstalovaným MonoGame SDK 3.4.

- `/SourceCodes` – obsahuje solution s oběma následujícími projekty.
  - `/DungeonMaster` – projekt s parserem vstupních dat z originální hry Dungeon Master.
  - `/DungeonMasterEngine` – projekt s reimplementovaným enginem hry Dungeon Master.
- `/Framework` – obsahuje použitý instalátor MonoGame SDK 3.4.
- `/DungeonMaster` – obsahuje všechny soubory reimplementace hry Dungeon Master.
  - `/DungeonMasterEngine.exe` – binárka spouštějící hru.
- `/prace.pdf` – text této práce.
- `/README.txt` – informace o struktuře CD.