Jabber dungeon

Michal Staruch
Ondřej Hlavatý
Petr Mánek

1 Zadání

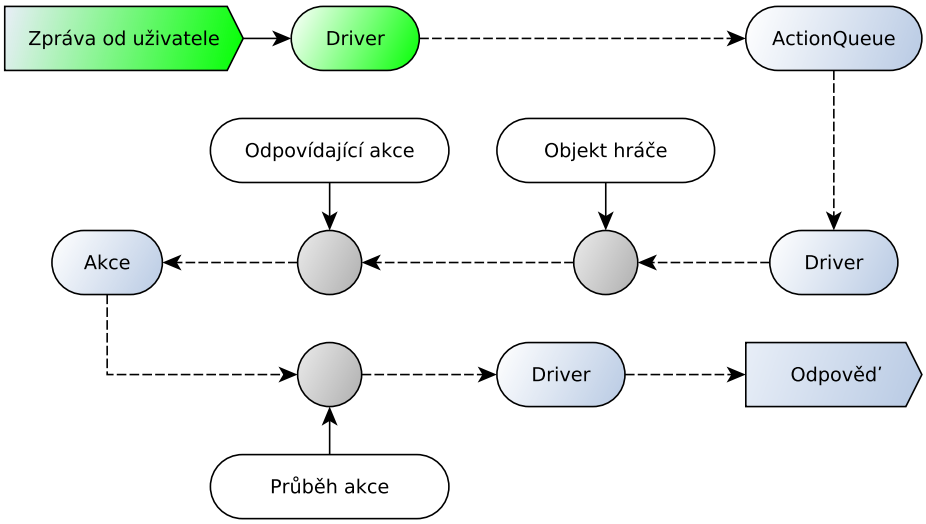
Projekt byl zadán jako zápočtový program na MFF UK. Cílem bylo vytvořit herní systém textové adventury, kterou bude možno hrát přes komunikační síť Jabber. Samotná adventura by měla být rozumně ovladatelná textovými příkazy a komunikovat s uživatelem přirozeným jazykem - v angličtině. Herní svět je společný pro všechny hráče, kteří se v něm pohybují nezávisle.

Jako jazyk implementace byl zvolen `C++`. Součet schopností programovat v týmu byl pro tento jazyk největší, a zároveň je vhodný pro objektový návrh.

2 Základní návrh

Srdcem celé hry jsou objekty *GameManager* a *ActionQueue*. Většinu informací přenáší objekty *ActionDescriptor* (AD). Komunikace s uživatelem probíhá skrz *Driver*.

Nejprve, jak vypadá životní cyklus *ActionDescriptoru*.



Obrázek 1: Životní cyklus *AD*

Každý běžící *Driver* má své vlastní vlákno (na obrázku zeleně), odlišné od hlavního herního vlákna (modře). V okamžiku přijetí zprávy vytvoří driver objekt AD nebo jeho po-

tomka. Uloží si do něj všechny potřebné informace pro další zpracování (odesílatele, text zprávy atp.) a zařadí jej do fronty v ActionQueue. Ta pak ve svém vlákně přiděluje čas na zpracování akce postupně a zajišťuje tím atomicitu operací „zadarmo“. Když na tuto akci dojde řada, tak se teprve zjišťuje Driver co vlastně má provést. Jako první sváže akci s herním objektem uživatele - v tomto vlákně již má přístup k databázi. Poté z věcí v okolí sežene seznam proveditelných akcí a zjistí, která odpovídá vstupnímu textu. Jakmile je akce svázána, předá se řízení dané akci. Ta provede změny v herním světě a vloží do AD informace potřebné k sestavení odpovědi.

Stručně pár dalších důležitých tříd v návrhu:

GameManager (GM) zprostředkovává samotný přístup k herním objektům a světu. Izoluje herní mechanismy od úložiště informací, zpřístupňuje některé zkratky.

ObjectPointer (OP) Během načítání herních objektů se může stát, že GM odstraní z paměti jiný načtený objekt z důvodu úspory paměti. OP umožňuje při přístupu objekt nechat znovu načíst.

Driver Implementovány jsou aktuálně *TextDriver*, obsluhující zprávy textového charakteru, a jeho potomci *ConsoleDriver* a *JabberDriver*. Jeden umožňuje ovládat postavu ze standartního vstupu a zprávy vypisovat na standartní výstup, druhý se přihlašuje jako jabber klient s adresou `dungeon@eideo.cz`a komunikuje pomocí XMPP.

Action Její potomci reprezentují proveditelnou akci. Obsahuje zejména metodu `commit`, která spustí provedení akce.

IObject Každý objekt ve hře implementuje toto rozhraní. Obsahuje metody pro zjišťování informací o objektu, persistenci vlastností a udržování relací.

ThorsHammer Speciální herní objekt. Kdo jej vlastní, má přístup k administrátorským funkcím hry.

3 Reprezentace herního světa

Každý objekt ve hře má svůj unikátní textový identifikátor a je instancí třídy implementující rozhraní IObject. Téměř každý objekt také implementuje rozhraní *IDescriptable*, které poskytuje objektu název a rozsáhlejší popis.

Objekty mezi sebou uchovávají relace, což jsou orientované hrany s daným typem. Mezi dvěma objekty může existovat pouze jedna relace od každého typu a směru.

3.1 Persistence

Jednotlivé vlastnosti objektů i relace jsou uchovávány v databázi. Relace jsou uložitelné jednoduše, na vlastnosti objektů má každý potomek IObjectu metodu `registerProperties`. V ní ohlásí, jaké vlastnosti má. Tyto data využije objekt *Archiver* a serializuje data do binárního streamu. Ten se pak může do databáze uložit. Stejnou metodou se pak data do

objektu dají načíst zpět – obrátí se směr Archiveru, a ten místo čtení hodnoty nastaví. Neví přitom nic o tom, jaké hodnoty ukládá, stačí mu k tomu typ.

To ale není jediné využití této metody. Dá se ji podstrčit libovolný objekt implementující *IPropertyStorage*, například administrátorský nástroj *PropertyEditor*, který je využije k pohodlné úpravě všech vlastností objektu přes libovolné herní rozhraní.

3.2 Uchovávání v paměti

Herní svět může být snadno docela veliký, proto jej nikdy nechceme načítat do paměti celý. Všechny herní objekty se proto načítají na vyžádání v době potřeby, kde to jde tak se používá pouze id objektu, předávané pomocí OP. Pro uchovávání již načtených objektů používáme Splay strom, protože umožňuje rychle přistupovat k objektům, které se používají hodně.

V současné chvíli to není implementováno, ale je možné tento strom upravit tak, aby objekty v určité hloubce (tedy dlouho nepoužité) zahazoval, a tím uvolnil paměť.

Z databáze načítá objekty *ObjectLoader*, ke čtení vlastností se využívá Archiveru. *DatabaseHandler* uzavírá práci s databází, používáme SQLite.

4 Herní objekty a akce

Jak už bylo řečeno, všechny objekty dědí IObject a zpravidla také IDescriptable. Základní vlastností herního objektu je schopnost zjistit akce dostupné na tomto objektu. Tak je dosaženo maximální flexibility světa. Není třeba mít na jednom místě seznam všeho co lze kdykoli provést, a v různém kontextu se mohou stejné zprávy vyhodnotit různě.

Při zpracovávání akce se v jednu chvíli zjistí akce na všech objektech v okolí uživatele, a postupně se na nich spouští metoda `match`. Jakmile se první akce k příkazu přihlásí, je provedena a na ostatní se již nedostane.

Dalšími obecnými typy objektů jsou:

Alive, Human Reprezentují živé objekty a uživatele. Každý živý objekt má své umístění a vlastnosti týkající se boje (tzn. životy, útočné a obranné číslo.) Dostupné akce na těchto objektech jsou ty které se týkají pouze daného uživatele. Tj. zjišťující aktuální stav, různé nápovědy, přejmenování atp.

Creature Reprezentují všechny objekty, na které se dá útočit, tedy všechna monstra a živočichy. Akce na tento objekt je zejména attack.

Item Cokoli, co jde uchopit. Objekt samotný nemá akce, ale má vlastnosti `size` a `weight`, udávající velikost a váhu objektu.

Wearable Cokoli, co jde použít jako část oděvu. Tento objekt používají zbraně a různé typy brnění. Umožňuje objektům měnit bojové vlastnosti postav.

Inventory Potomek Wearable, přidává navíc akci drop a umožňuje v sobě ukládat předměty.

Location Reprezentuje lokaci, ve které se ostatní objekty nacházejí. Tyto lokace jsou buď místnosti, které jsou s ostatními místnostmi spojeny dveřmi, nebo truhly, které obsahují předměty. Akce na lokaci jsou zejména průzkumné.

Door Jedno nebo obousměrný teleportační tunel mezi dvěma lokacemi. Akcí na tomto objektu je průchod skrz.

Crafter Speciální objekt, který je používán pro tvorbu věcí. Udržuje v sobě objekty *Recipe*, které říkají, jaké věci umí *Crafter* vyrobit. Ve hře se vyskytuje například jako kovadlina.

Resource Potomek třídy *Item*, reprezentuje věci, které se seskupují, tedy například peníze, nebo materiály pro *Crafter*.

4.1 Parametrizované akce

Velmi častým požadavkem je, aby akce měla nějaký parametr. Například když v místnosti leží tři lektvary, je potřeba určit který z nich se má akcí vypít. Tento problém důstojně řeší *MultiTargetAction* (MTA). Objekt vytvoří instanci MTA, a do seznamu cílů přidá sám sebe nebo i jiné objekty, na které lze akci zacílit. Při přidávání do seznamu se podle identifikátoru zjistí, že stejný druh akce již v seznamu je, a seznamy cílů se spojí v jeden. Akce si potom při spuštění vybere ze seznamu ten objekt, který nejlépe odpovídá zadanému textu.

Zároveň, pokud je v místnosti pouze jeden lektvar, pak spuštění akce vypije právě ten jeden lektvar i bez bližšího určení.

Při výběru objektů se porovnává editační vzdálenost (Damerau-Levenshteinova) slov z názvu objektu od slov ze zadaného textu. Při počítání vzdálenosti je změna znaku pocítána jako větší úprava, než přidání či odebrání. Počítají se pouze dostatečně podobná slova ve správnm pořadí. Proto přidání neurčujících slov (členů, zájmen) neovlivní volbu, a naopak nespecifikování částí názvu (bližší popisy) nemusí nutně znamenat chybu ve výběru. Pokud se nedokáže mechanismus rozhodnout, který objekt vybrat, dá uživateli na výběr.

5 Další (nejen) herní mechanismy

5.1 Skupinový popis

Aby hraní působilo přirozeněji, popisovatelné objekty se umí sdružit podle typu a vypsat jednotný popis za všechny. Funkcionalitu zajišťuje třída *ObjectGroup*. Výpis pak může vypadat třeba takto:

```
<dungeon> You see Piece of wood and Stone. There lies Red potion, Green
potion and Blue potion.
```

5.2 Dialogy

Zpočátku vypadal hra striktně podle režimu příkaz – odpověď, a hra připomínala spíše příkazový řádek[1]. Toto schéma rozbíjí mechanismus dialogu:

```
<user> rename myself
```

[1] Z čehož vznikly aliasy pro akce, např. `cd` pro průchod dveřmi :)

```
<dungeon> Well then, what do you want your new name to be?
<user> New name
<dungeon> OK. You shall now be called New name. I'm just curious, why did
you change your name?
<user> Because I wanted to show, how dialogs work.
<dungeon> Interesting. Now, back to the dungeon!
```

5.3 Boj

Systém boje vznikl krátce po vzniku dialogů a hojně jich využívá. Každá instance třídy *Creature* přidá akci `attack`, která inicializuje boj. Po zavolání akce `attack` se pustí obslužná metoda, která vždy provede požadovanou akci a v dialogu se zeptá uživatele, co dělat dále. Pokud hráč dosáhne 0 životů, zemře, pokud dosáhne 0 životů *Creature*, dostane se do stavu `Dying`. V tomto stavu je přidána akce kill, která dorazí příšeru.

5.4 Smrt

Smrt nastane, pokud životy hráče nebo nepřátel dosáhnou nuly. Metoda `changeHp`, která se stará o změnu počtu životů automaticky zavolá obslužnou proceduru `die`. V případě třídy *Human* procedura změní stav hráče na mrtvého, zakáže mu většinu akcí kromě některých výjimek a znemožní mu hrát po stanovený interval. Po uplynutí toho intervalu se může hráč akcí `respawn` oživit, při oživení je přemístěn do poslední místnostni s flagem `respawnable`, kterou hráč navštívil. V případě třídy *Creature* je na zem puštěna odměna za zabití (pokud nějaká je) a instanci se naplánuje automatický respawn, který je kontrolován (a popřípadě proveden) při zjišťování akcí.

5.5 Pasti

Pasti jsou velmi důležitá součást naší hry. Umožňují nám měnit svět v reakci na události a mít svět dynamičtější. Jednotlivé pasti jsou potomky třídy *Trap*. Ta definuje 2 metody:

- metodu `trigger`, která je zavolána pokaždé, když se vyvolá akce, na kterou je past napojena. V této metodě se rozhodne, jestli se past má pustit a něco udělat, nebo nechat svět plynout dále.
- metodu `exceptionTrigger`, ta reaguje na výjimky *TrapException*. Tato výjimka je vyhozena v metodě `trigger` a její smysl je měnit probíhající akce. Tato metoda může akci zrušit, nebo ji změnit (například past *AttackTrap*, která zahájí boj při vstupu do místnosti.)

V současné době máme ve hře několik pastí, například *HealingTrap* (past, která léčí jednoho uživatele, nebo všechny v místnosti), *AttackTrap* (nechá něco zaútočit na uživatele), *DoorLock* (past, která se chová jako zámek na dveřích), nebo *ItemRespawner* (past, která umožňuje věcem se znovu objevit po určitém intervalu.)

6 Programová část

V této kapitole bychom chtěli vypíchnout některé části kódu, které se nám líbí, jsou zajímavé, nebo se k nim váže nějaká příhoda.

6.1 Logování

Podstatnou částí systému, na kterou jsme patřičně hrdí, je logovací mechanismus. Jeho srdcem je třída *Logger*, se kterou aplikace komunikuje během hry v singletonovém režimu použitím několika šikovných maker.

```
LOGS("Loader", Fatal) << "Error saving object "
                      << id << ", error code " << err << LOGF;

LOG("Loader") << ... // default is Severity::Info

LOGH("Database cleanup") // creates headline for the next section
```

Díky přetížení operátoru `<<` můžeme logovat téměř jakýkoliv druh obsahu. Protože každý záznam uvádí kromě zprávy i důležitost, chyby nezapadnou mezi běžnými záznamy.

Všechny záznamy z logu jsou přesměrovány skrz buffery do výstupů – podtříd `std::ostream`. Tento objektový přístup je velmi praktický, protože nám umožňuje postavit výstupní soubory na stejnou úroveň jako standardní výstup aplikace. Navíc můžeme pro každý výstup nastavit minimální důležitost, od které události zaznamenává. V současném stavu se log ukládá takto:

- na `stdout` proudí zprávy s důležitostí >= `Severity::Info`
- do souboru `YYYY-MM-DD-stdout.log` se kopíruje `stdout`
- do souboru `YYYY-MM-DD-verbose.log` proudí úplně všechny zprávy
- do souboru `YYYY-MM-DD-warnings.log` proudí zprávy s důležitostí >= `Severity::Warning`

(soubory jsou opatřeny datem zapnutí aplikace a zapisuje se do nich režimem append)
V případě potřeby můžeme kdykoliv toto nastavení změnit i za běhu aplikace. Typickým příkladem této operace je parametr `--verbose`, se kterým se na obrazovku vypisují úplně všechny záznamy. Této změny dosáhneme jedním řádkem kódu:

```
Logger::getInstance().setMinSeverity(cout, Logger::Severity::Verbose);
```

Logování je samozřejmě thread-safe, takže se nemůže stát, že by některá vlákna aplikace (například *JabberDriver*) mohly začít zapisovat uprostřed zapisování jiného záznamu.

6.2 Výběr náhodné zprávy

Pro ozvláštnění komunikace je dobré nemít všude stejné statické texty, ale výpis trochu měnit.

```
<user> something
<dungeon> One does not simply ask me to "something".
<user> something
<dungeon> I wouldn't do that if I were you. Besides, I can't do that.
<user> something
<dungeon> The day may come when I understand "something" but it's not
this day.
```

Pro tento účel jsme si napsali utilitku *RandomString*, která se používá takto:

```
return RandomString::get()
        << "What do you mean \"" + input + "\"?" << endr
        << "One does not simply ask me to \"" + input + "\"." << endr
        ...
        << "This is not the thing you are trying to do." << endr;
```

Třída využívá přetížení operátoru `<<` a přetypovaní na string.

6.3 InstanceOf

Potřeba použití nečeho takového signalizuje chybu v návrhu, ale pokud chceme mít obecný mechanismus práce s objekty, je občas potřeba o obecném `IObject*` zjistit něco více. Proto se hodí tento operátor. Použití:

```
Alive* A;
A->instanceOf(Backpack); // true / false
```

Celá věc funguje trochu magicky, ale hlavně rychle. Pomocí preprocesorových maker se v každém objektu vytvoří virtuální inline funkce `className` a `isInstanceOf`. ClassName vrací konstantní řetězec obsahující název třídy – ten se používá například pro serializaci do databáze. Pro použití s instanceOf se ale počítá jen s ukazatelem na tento řetězec. Pomocí dalších maker se ukázka přepíše takto (s využitím konstanty):

```
A->isInstanceOf(Backpack::BackpackClassName); // true / false
```

A protože je funkce inline, kompilátor ukázku přepíše například takto:

```
Alive::AliveClassName == Backpack::BackpackClassName
    || IDescriptable::IDescriptableClassName == Backpack::BackpackClassName
    || IObject::IObjectClassName == Backpack::BackpackClassName;
```

Celý operátor tedy pouze projde hiearchii volané třídy a porovná ukazatele na tento konstantní řetězec.

6.4 Dialogy

O tom jak vypadají dialogy v praxi bylo již psáno v části **??**. Jak se ale dialogy používají? Příkladem je celý kód akce na přejmenování uživatele:

```
*ad << "Well then, what do you want your new name to be?";
ad->waitForReply([] (ActionDescriptor *ad, string reply) {
    ((Human*) ad->getAlive())->setUsername(reply)
            ->save();
    *ad << "OK. You shall now be called "
        << ad->getAlive()->getName() << ". "
        << "I'm just curious, why did you change your name?";
});
ad->waitForReply([] (ActionDescriptor *ad, string reply) {
    *ad << "Interesting. Now, back to the dungeon!";
});
```

ActionDescriptor umožňuje zařadit do své vnitřní fronty lambda funkci, která bude reagovat na odpověď uživatele. Pokud AD nemá frontu prázdnou, musí si jej Driver někde zapamatovat a vrátit se k němu při přijetí další zprávy.

Všechny lambda funkce nemusí být vytvořeny hned, dají se vytvářet postupně. Tento zápis je však přehledný, a z kódu je hned vidět jak dialog probíhá. Zároveň je ale možno udělat dialog, který není lineární, nebo třeba není předem známý - například boj.

Dialog ale může rozbít atomicitu operací, jelikož při čekání na odpověď od uživatele mohou probíhat jiné akce, je tedy potřeba po získání odpovědi zkontrolovat stav herního světa, aby nedošlo k ilegální úpravě světa.

6.5 Cloner

Původní myšlenkou bylo to, že všechny objekty budeme vytvářet ručně a během hry. Postupem času se však ukázalo, že toto není nejlepší nápad a tak jsme hledali způsob, jak objekty vytvářet rychleji.

Nakonec jsme přišli s třídou *Cloner*, která využívá vlastnosti objektů a pomocí *IPropertyStorage* je schopen je přečíst. Při jejich čtení si je ukládá do své paměti. Po přečtení využije další metodu naší perzistence a tou je `createObject`, kterou vytvoří čistou kopii objektu. Poté z uloženého seznamu nahraje všechny vlastnosti nového objektu, změní mu `id` a vrátí tento objekt. Tuto kopii nazýváme `shallowCopy`.

Cloner umí vytvořit ještě `deepCopy`, která navíc zkopíruje všechny relace objektu.

Tuto třídu hojně využíváme při vytváření světa ve třídě *WorldCreator*, kde nejprve vytvoříme vzory všech objektů a poté už je jenom rozkopírujeme, kde je potřeba.

7 Ukázky

Rádi bychom ukázali nějaké naprosto základní ukázky komunikace s programem.

7.1 Průzkum světa

```
<user> hello
<dungeon> Hi!
<user> where am i
<dungeon> You are in Base Camp. Nothing much to be found here, but get
familiar with basic commands - try to type 'help'. There is a wooden
door. You recognize Salmelu and CiTrus. You have a Leather backpack
with you.
<user> go to that wooden door
<dungeon> You are in Equip room. Looks like noone has been there for
a long time. There is a wooden door. You see Green potion, Blue potion
and Red potion.
<user> drink the blue potion
<dungeon> You've drunk Blue potion. You've healed 100 hitpoints
<user> pick the red potion
<dungeon> You've picked up Red potion.
<user> explore
<dungeon> You are in Equip room. Looks like noone has been there for
a long time. There is a wooden door. You see Green potion lying on the
ground. You have a Leather backpack with you. There is Red potion.
<user> drop the red potion
<dungeon> You've dropped Red potion.
```

7.2 Systém boje

```
<user> explore
<dungeon> You are in Landcastle. A nice stone castle, having a magical aura
surrounding it. You feel the aura and feel much better here. If you are confused
```

what to do, try typing 'help'. Around you are Bridge and Climb. You smell the presence of Aearsis and CiTrus.

<user> go to bridge

<dungeon> You walk over the bridge. Seems like you are in Faygrass. A vast grass field. You see Rat and Rat wandering around. You see Small spider wandering around. Around you are Muddy walkway, Bridge and Stairs.

<user> attack spider

<dungeon> You have dealt 7 damage to Small spider by your Wooden axe. Small spider caused you 10 damage by its poisoned bite. Your current hitpoints have changed to 290. Type 'attack' (a) to attack the enemy, 'check' (c) to check its status or 'run' (r) to run from the fight.

Small spider: [*******---] You: [**********]

<user> attack

<dungeon> Your Wooden axe have caused 7 damage to Small spider. Small spider caused you 10 damage by its poisoned bite. Your current hitpoints have changed to 280. Type 'attack' (a) to attack the enemy, 'check' (c) to check its status or 'run' (r) to run from the fight.

Small spider: [****------] You: [**********]

<user> a

<dungeon> Your Wooden axe have caused 6 damage to Small spider. Small spider is mortally wounded. Use 'kill ...' to finish it.

<user> kill spider

<dungeon> Small spider has dropped nothing. You have killed Small spider.

7.3 Resources

<dungeon> You are in Shadowcliff. Dark dangerous cliffs. You shouldn't leave the path you are currently on. You see Rat wandering around. There lie corpses of Small spider and Small spider. You see Pathway and Climb. You see 58 sand and 78 sand lying there. There lies 41 white powder and 31 white powder.

<user> pick 58 sand

<dungeon> You've picked up 58 sand.

<user> pick 78 sand

<dungeon> You've picked up 78 sand.

<user> pick 41 white powder

<dungeon> You've picked up 41 white powder.

<user> pick 31 white powder

<dungeon> You've picked up 31 white powder.

<user> list items

<dungeon> As your backpack, you're using Leather backpack. There are 136 sand and 72 white powder. Wooden axe is your main weapon.

7.4 Výroba věcí

<user> craft table

<dungeon> Small red potion: 12 white powder and 20 sand. Tiny red potion: 4 white powder and 8 sand.
<user> create small red potion
<dungeon> You have tried to craft Small red potion but you succeeded just barely. You have acquired a Tiny red potion and put it in your backpack.

7.5 Past

<user> explore chest
<dungeon> You have cut yourself accidentaly with an icicle. You have received 37 damage. There is also some Small red potion.
<user> pick potion
<dungeon> You've picked up Small red potion.

7.6 String matching

<dungeon> Seems like you are in Shadowcliff. Dark dangerous cliffs. You shouldn't leave the path you are currently on. You see Rat wandering around. There lie corpses of Small spider and Small spider. There are Pathway and Climb. There lies 75 sand. There are 31 wood and 30 wood.
<user> pick wood
<dungeon> Sorry, did you mean 31 wood or 30 wood?
<user> 31
<dungeon> You've picked up 31 wood.

7.7 Sdružující akce

<user> use the wooden thing
<dungeon> Sorry, did you mean wooden door, Wooden axe or Wooden club?
<user> axe
<dungeon> You have successfully equipped Wooden axe.
<user> use the wooden thing
<dungeon> Sorry, did you mean wooden door or Wooden club?
<user> door
<dungeon> You've gone through that door...