VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

PORT HRY DIGGER NA SOUČASNÉ OPERAČNÍ SYSTÉMY

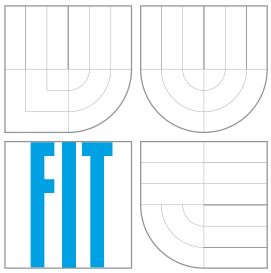
BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

MARTIN PRES

BRNO 2014

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

PORT HRY DIGGER NA SOUČASNÉ OPERAČNÍ SYSTÉMY

PORT OF DIGGER GAME TO THE MODERN OPERATING SYSTEMS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

MARTIN PRES

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. JAN PEČIVA, Ph.D.

BRNO 2014

Abstrakt

Práce se zabývá vývojem portu hry Digger podle vzoru existující hry Digger Remastered. Vývoj probíhá od analýzy chování referenční hry přes návrh aplikace po její implementaci v jazyce C++ a nasazení. Teoretická část práce se věnuje výběru vhodných knihoven frameworku Qt a návrhu herních mechanik. Praktická část se pak podrobně věnuje implementaci jednotlivých navržených entit a způsobu distribuce hotové aplikace.

Abstract

The bachelor's thesis concerns development of the port of Digger game. The port of game is modeled after the Digger Remastered game. Development starts from behaviour analysis of existing game, followed by application proposal, implementation in C++ and distribution. Teoretical part of this thesis concerns selection of appropriate libraries from the Qt framework and proposals of game mechanics. Practical part concerns implementation of proposed entities and distribution of final product.

Klíčová slova

Digger, hra, Qt, Graphics View Framework, port, C++.

Keywords

Digger, game, Qt, Graphics View Framework, port, C++.

Citace

Martin Pres: Port hry Digger na současné operační
systémy, bakalářská práce, Brno, FIT VUT v Brně, 2014

Port hry Digger na současné operační systémy

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Jana Pečivy, Ph.D.

.....
Martin Pres
20. května 2014

Poděkování

Děkuji tímto panu Ing. Janu Pečivovi, Ph.D. za cenné rady a připomínky při vypracování této bakalářské práce.

Obsah

Kapitola 1

Úvod

V dnešní době existují hry, které je prakticky nemožné si zahrát, protože byly vydány pro platformu, na jejíž poslední kusy se práší někde v podkroví, v muzeu či leží na skládce. Veliké množství starých her, které jsou nekompatibilní s moderními PC, lze hrát na tzv. emulátorech. Ty vytvářejí virtuální prostředí původní platformy a hru je možno, často bez jakéhokoliv problému, spustit. Ne pro všechny platformy však existují emulátory.

Další možností, jak hrát hru, která již nejde spustit, je sehnat její port pro konkrétní platformu. Herní porty mohou poměrně přesně kopírovat předlohu včetně grafiky, herních mechanik, ozvučení apod. Některé porty do těchto her přidávají i něco navíc a některé jen zkopírují nápad. Port ale také nemusí vůbec existovat pro platformu, kterou používáte.

V této práci bude popsán proces tvorby portu hry Digger, v knihovně Qt, pro operační systémy Ubuntu, Linux Mint a MS Windows. Hra ponese název *CuteDigger*. V následující kapitole je stručně uvedena historie hry Digger včetně jejích existujících a zajímavých portů a remaků. Je zde také analýza knihoven Qt použitelných pro tvorbu 2D her a detailnější popis fungování hry.

V kapitole Návrh detailněji popisuji herní objekty, herní mapu a rozšíření, která se do portu hry implementují. Je zde obsažen i návrh jádra aplikace, které bude řídit její chod a řídit chování herních objektů. V kapitole Implementace je pak popsáno, jak jsou navržené struktury do portu hry zabudovány.

V kapitole Nasazení je pak popsána tvorba *.deb* a *.rpm* balíčků pro linuxové distribuce a instalátoru pro MS Windows.

Kapitola 2

Analýza

2.1 Hra Digger a její porty

Hra digger byla vydána v roce 1983 kanadskou společností Windmill Software pro platformu IBM PC. Byla distribuována na 5,25" disketách s ochranou proti kopírování. Hra byla napsána v jazyce C a kompilována pomocí překladače Lattice C v1.x. Nízkoúrovňovější části kódu byly napsány v assembleru.[14][12][13]

Prostředí hry je zasazeno do podzemního bludiště, kde je úkolem hráče sesbírat všechny zelené drahokamy v dané úrovni nebo zabít všechna monstra. Hráč je při své cestě monstry ohrožován a při přímém kontaktu s nimi přijde o život. Hráč není v pohybu omezen pouze na výchozí tunely, ale může vykopat své vlastní. To mu umožňuje vyhýbat se monstrům, která jsou při pohybu ve své základní formě odkázána pouze na tunely. Ve své druhé formě však umějí kopat, stejně jako hráč. Hráč navíc disponuje schopností vystřelit jednu střelu, která se po čase opět nabije. Tím se může hráč ochránit před bezprostředním nebezpečím. Kromě drahokamů se v úrovni vyskytují i pytlíky se zlatem se kterými je možno pohnout a které mohou propadnout vertikálním tunelem. Při pádu mohou usmrtit jak hráče, tak monstra a pokud padají dostatečně dlouho, vypadne z nich zlato, které může hráč sebrat. Hráč může také přejít do bonusového módu, kde kontakt s monstry způsobí smrt monstra, zatímco hráč zůstane živý. V tomto módu také monstra od hráče utíkají. V základní variantě hry je k dispozici osm sad úrovní které se cyklicky mění. S každou další úrovní se zvyšuje obtížnost hry. Ve hře hrála hudba na pozadí, která se změnila při hráčově smrti a při vstupu do bonusového módu. Hra používala polyfonní zvukové efekty, které byly v roce 1983 ještě poměrně neobvyklé a dá se říci pokročilé.[17]

Protože hra ke svému chodu vyžadovala CGI kartu, není ji možné hrát na moderních PC sestavách. Z toho důvodu vzniklo mnoho remaků této hry pro modernější PC sestavy a systémy.

V roce 1998 vytvořil Andrew Jenner novou verzi hry pomocí reverzního inženýrství a nazval ji Digger Remastered. Je dostupná pro platformu MS DOS a v této práci z ní budu vycházet. Napsána je v jazyce C s využitím knihovny SDL. [15] Andrew Jenner také provozuje webové stránky www.digger.org, kde je možné hrát také další porty hry, které vycházejí z Digger Remastered (např. Java verzi hry od Marka Futrega nebo další porty na různé operační systémy).

V roce 2009 vydala ruská společnost Creat Studios remake s názvem Digger HD pro konzoli PlayStation 3. Tato verze hry využívá koncept 2.5D a rozšiřuje hru například o nové typy monster, výbušniny a nová herní prostředí. Zajímavostí je, že spolu s remakem je dodáván i port původní hry pro tuto konzoli.[11]

2.2 Analýza hry

Počítačová hra Digger je hra probíhající v reálném čase (průběh hry není rozdělen na tahy). Ve hře se objevují čtyři typy objektů, které se mohou pohybovat a reagovat s ostatními objekty. Jsou jimi:

- hráč
- monstra
- pytle se zlatem
- střely

Kromě nich může hráč interagovat se dvěma dalšími objekty, které nejsou pohyblivé. Hráč po interakci s nimi získá body, případně spustí další akci, která s objekty samotnými nesouvisí. Jedná se o:

- drahokamy
- třešničky

Hráč se ve hře ocitne na předem určeném místě (*player spawner*), ihned po jejím spuštění a může se volně pohybovat ve čtyřech směrech (nahoru, dolů, doleva a doprava) po celé herní mapě. Na počátku hry má hráč tři životy, vynulované skóre a nabitou střelu. Jeho cílem je na dané mapě posbírat všechny drahokamy nebo zabít všechna monstra. Drahokamy posbírá pouhým pohybem přes místo, kde se drahokam nachází. Monstra může zabít třemi způsoby:

- vystřelenou střelou – počet střel není omezen, nicméně nabití každé střely chvíli trvá
- padajícím pytlem se zlatem (tímto způsobem se mohou monstra nechtěně zabíjet navzájem)
- kontaktem s monstrem v bonusovém módu

Na herní mapě jsou od začátku hry přítomny také všechny objekty mimo monster. Mapa může mít čtvercový nebo obdélníkový tvar. Na mapě se nacházejí vykopané tunely, ty však omezují v pohybu pouze monstra (jen v jejich základní formě). Hráč svým pohybem může vytvořit nové tunely.

Monstra se na herní mapě objevují postupně na předem daném místě (*monster spawner*). První monstrum se ve hře objevuje několik sekund po startu hry a ve stejném intervalu se objevují další. Ve hře se současně vyskytuje maximálně pět monster. Další se objeví, jakmile bude některé monstrum zabito. Jakmile se na mapě nachází všechna monstra, objeví se na místě spawneru třešnička. Pokud ji hráč sebere, přičte se mu jistý bodový zisk a dostane se do bonusového módu. Monstra se pohybují ve čtyřech směrech, jako hráč a mají dvě formy:

- *Nobbin* – základní forma, pohybuje se pouze v tunelech, při kontaktu s pytlem zlata jej posune.
- *Hobbin* – pohyb není omezen, dokáže vytvořit nové tunely, při kontaktu zničí drahokamy, pytle se zlatem a rozsypané zlato.

Monstra se přemění do Hobbin formy po jistě době strávené ve hře a po chvílí se změní zpět do Nobbin formy. S každou další úrovní se snižuje doba pro přechod do Hobbin formy a zvyšuje se doba, kdy monstrum v této formě setrvává.

Dalším pohyblivým objektem na herní mapě je zlato. Zpočátku je zlato přítomno ve formě pytlů, které může hráč a monstra posouvat. Je-li pytel posunut k vertikálnímu tunelu nebo pokud je pod ním podkopána země, pytel začne padat (pokud je podkopán tak chvíli trvá než spadne, ale pokud je posunut do tunelu, padá ihned). Pokud pytel padá, dokáže zabít hráče i monstra. Pokud při pádu urazí určitou vzdálenost, rozbije se při nárazu o zem a vysype se z něj zlato. Zlato může hráč posbírat a přičíst si tak bodový zisk.

Posledním pohyblivým objektem je střela. Ta vznikne při výstřelu u hráče a cestuje tunelem ve směru, kterým se hráč díval. Zaniká při kontaktu se zdí, s jiným pohyblivým objektem nebo pokud vyprší její životnost. Při kontaktu s monstrem v kterékoliv formě je zabije a připočte hráči bodový zisk. Hráč může vystřelit jednu střelu a pak musí čekat několik sekund, než se mu nabije střela nová. Nabíjení střely jasně signalizuje změna animace hráčova avataru.

Velmi důležitým nepohyblivým objektem je drahokam. Není možno jej jakkoli posunout ze svého místa. Je možné ho pouze sebrat. Drahokamy při sesbírání přičtou hráči bodový zisk a zmizí z mapy. Kromě hráče může drahokamy sbírat také Hobbin, ten je však zničí a hráči se nepřipočtou žádné body. Hobbin sebere drahokam jen pokud mu stojí v cestě, nedělá to účelně.

Informace o stavu hry shromažďuje a zobrazuje HUD. To je přítomno v horní části herní obrazovky a obsahuje informace o:

- aktuálním skóre
- počtu zbývajících životů

2.3 Knihovna Qt a hry

Her v C++, využívající knihovnu Qt, vzniklo poměrně nemalé množství, viz [2]. Jejich vývoj je relativně snadný díky *Graphics View Frameworku*, který Qt obsahuje od verze 4.2, kde nahradil framework `QCanvas`. Tento framework v sobě sdružuje třídy pro organizaci scény, grafických objektů a pro vykreslení grafických primitiv. Qt poskytuje také moduly pro přehrávání zvuků, hudby a videí, zpracování vstupů z klávesnice a myši, nástroje pro tvorbu herních smyček a pro práci s daty.

2.3.1 Graphics View Framework

Jednou ze základnch tříd je třída `QGraphicScene`. Poskytuje místo pro organizaci velkého množství 2D objektů. Tato třída slouží jako kontejner pro prvky typu `QGraphicsItem` a je používána společně s `QGraphicsView`, jelikož se sama o sobě nedá zobrazit. Scéna má vlastní souřadnicový systém a její silnou stránkou je schopnost efektivně vrátit umístění konkrétního objektu i v případě, že je jich ve scéně velké množství. Dokáže také sdělit, které objekty jsou viditelné v zobrazované oblasti scény.

`QGraphicsItem` je základní bázová třída pro všechny objekty ve scéně. Třídy, které z ní dědí mohou definovat tvar objektu, detekci kolizí, reakce na události a vlastní vykreslování objektu. Každý objekt scény má vlastní lokální souřadnicový systém a zná i svou vlastní pozici uvnitř scény. Nad objekty je možno provádět základní operace, jako posunutí, rotace, zkosení, apod.

Objekty ve scéně mohou být také potomky třídy `QGraphicsWidget`. Ta rozšiřuje možnosti `QGraphicsItem` o funkce, které mají normální widgety v knihovně Qt, ale není abstraktní třídou, jako `QGraphicsItem`. Je použitelná zejména v případě, že po objektech požadujeme např. umístění v layoutech, či pokročilejší zpracování focusu na objekty. Pro objekty, jenž by měly zobrazovat herní entity, jako postavy a nepřátele, tato třída není příliš vhodná, ale např. pro prvky *HUD* (Head Up Display - informace o stavu hry) nebo herní menu význam má.

O vlastní vykreslování objektů ve scéně se stará třída `QPainter`. Zvládne vykreslovat nejen 2D grafická primitiva, ale i složitější útvary, text a bitmapové obrázky. Používá se převážně v metodě `paintEvent()`, která je volána vždy, když je požadováno překreslení objektu. [16]

Třída `QGraphicsView` slouží jako pohled do scény vytvořené třídou `QGraphicScene`. Dokáže zobrazit celou scénu nebo jen její výřez. Ve výchozím nastavení ji zobrazuje ve scrollovatelném prostředí. Pohled má vlastní souřadnicový systém nezávislý na scéně, kterou zobrazuje. Souřadnice je však možné vzájemně přepočítat. Pohled do scény je také možné transformovat pomocí 2D transformačních matic.

V Qt je možné renderovat grafické objekty pomocí knihovny OpenGL. Slouží k tomu třída `QGLWidget`, která vytvoří okno pro vykreslované objekty. V této třídě si lze zvolit, jestli se bude pro kreslení používat `QPainter` nebo standardní OpenGL příkazy. OpenGL lze také použít pro akceleraci pohledu do scény. To se provádí tak, že se do `QGraphicsView` vloží jako výřez (*viewport*) objekt tříty `QGLWidget`. Rendering pak bude probíhat za podpory OpenGL. [1]

2.3.2 Zvuky a video

Pro přehrávání zvuku v Qt je k dispozici několik tříd v modulu `QtMultimedia`. Modul samotný zvládá i nahrávání zvuku a také nahrávání a přehrávání videa. V oblasti práce se zvukem tento modul nahradil knihovnu `Phonon`, která již není součástí Qt od verze 5.

Pro zvukové efekty, které by měly mít co nejmenší odezvu při přehrávání, lze použít třídu `QSoundEffects`. Tato třída dokáže přehrávat zvuky ve formátu *wav* a obsahuje zejména metody pro ovládání hlasitosti a opakování efektu.

Pro přehrávání hudby a videa je vhodná třída `QMediaPlayer`. Obsahuje, kromě funkcí pro kontrolu přehrávání a hlasitosti, také možnost pracovat s playlisty. Zvládne zpracovávat mnoho formátů souborů, jako *wav*, *mp3*, *mp4*, apod. [5]

2.3.3 Vstupy a výstupy

Obsluha klávesnice a myši probíhá přes *systém událostí* (*event system*). Ten pracuje tak, že vyskytne-li se nějaká událost (např. stisknutí klávesy nebo pohyb myši), vytvoří Qt instanci příslušné podtřídy třídy `QEvent` a předá ji konkrétnímu objektu, nad kterým událost vznikla. Ten ji buď akceptuje nebo ignoruje a událost pak putuje dál v hierarchii Qt objektů, dokud ji některý z nich nepřijme. Objekty na událost také mohou reagovat, aniž by ji přijali.

Událostí je několik typů. Kromě událostí spojených s klávesnicí a myší existují události spojené s okny a widgety (např. změna velikosti, zavření, žádost o překreslení). Většina událostí má svou vlastní třídu (např. `QMouseEvent` pro události myši). [4]

2.3.4 Herní data a nastavení

Většina her nějakým způsobem umožňuje ukládat svůj stav, ať už jde o nastavení nebo hráčův postup hrou. Qt poskytuje několik možností, jak ukládat externí herní data. Je zde možnost ukládat je v XML a JSON strukturách, v SQL databázích a nebo v *ini* souborech. [3]

Je zde k dispozici i *Resource System* používaný pro ukládání dat, která jsou součástí aplikace a bývají zakompilována v jejím binárním kódu (např. obrázky menu, sprity, styly). [6]

2.3.5 Herní smyčky a herní čas

Základem většiny her je nekonečná smyčka uvnitř které se vyskytují a jsou obsluhovány herní události. Smyčka obvykle začíná při startu hry a končí spolu s herní instancí. Lze je vytvořit jednoduše pomocí nekonečného cyklu. Qt však umožňuje použít i jiné metody implementace herních smyček. Jednou z nich je použití Qt časovačů z třídy `QTimer`, což jsou jednoduché časovače se schopností cyklicky opakovat odpočet. Při vypršení časového limitu vyvolá objekt časovače signál a spustí odpočet od začátku. Na signál lze reagovat *slot* funkcí obsahující kód herní smyčky. Tímto způsobem lze regulovat *FPS* (Frames Per Second) nastavením vhodné hodnoty časového limitu. Časovač lze také nastavit tak, aby po vypršení limitu nezačínal odpočet znova, ale vyčkal na podnět. [8]

K modelování herního času může posloužit třída `QTime` poskytující funkce pro práci s kontinuálně tekoucím časem. Objekt této třídy obsahuje reálný čas ve 24-hodinovém formátu s možností dělení na hodiny, minuty, sekundy a milisekundy (přesnost také závisí na přesnosti měření času hostujícího operačního systému). Tuto třídu je také možné využít jako stopky. Narozdíl od třídy `QDateTime` neuchovává informaci o časovém pásmu. Počáteční čas lze zadat explicitně nebo jej načíst ze systémových hodin.[7]

Kapitola 3

Návrh

Tento port bude implementovat všechny herní mechanismy popsané v kapitole 5 pomocí knihovny Qt. Oproti původní hře a portu pro MS DOS zde bude několik výrazných rozdílů.

V původní hře byla velikost mapy omezena na herní okno. V tomto portu bude možné mít mapu přesahující velikost okna a při pohybu hráče bude prostředí plynule scrollovat. Bude zde možnost hrát vytvořené mapy uložené v textových souborech a nebo bude možné mapu vygenerovat procedurálně.

V původní hře mohlo být na herní ploše přítomno maximálně pět monster. V tomto portu však nebude počet přítomných monster omezen. Souvisí to s možností mít velkou herní mapu, kde by malý počet monster výrazně snížil obtížnost hry. Monstra se postupně objeví všechna a hráč bude jejich počet snižovat.

Herní HUD bude také pozměněno. Bude uchovávat navíc tyto informace:

- je-li střela připravena
- zdali je hráč v bonusovém módu
- aktuální hodnota FPS

3.1 Jádro aplikace

Jádro aplikace bude vycházet z návrhového vzoru *Mediator*. Jeho funkcí bude vytvářet instance jednotlivých herních objektů, zajišťovat interakce mezi nimi a udržovat herní smyčku a stavy hry.

Jádro po startu aplikace načte uložená nastavení a data, pokud budou k dispozici. Vytvoří instance objektů, které jsou k dispozici po celou dobu běhu aplikace a zobrazí hlavní menu. Před samotnou hrou bude předcházet další menu, kde hráč vybere startovní mapu, případně nechá vygenerovat novou.

Před startem samotné hry aplikace inicializuje potřebné časovače, vytvoří objekt herní mapy, hráče a objekty pytlů se zlatem, které se na mapě objevují už na začátku hry a nikoliv postupně. Poté vytvoří scénu, umístí do ní objekty a vytvoří pohled do scény. Dále napojí sloty herních objektů na signály příslušných časovačů.

Se startem hry se aktivuje hlavní smyčka tvořená časovačem a funkcí (slot) vyvolanou při signalizaci vypršení časovače. Funkce provede při vypršení časovače předepsaný krok každého herního objektu podle stavu tohoto objektu a podle interakcí s jinými objekty.

Rovněž provede překreslení scény. Nastavením různých časových limitů pro vypršení hlavního časovače lze docílit pevného nastavení FPS. Pro hodnoty kolem 60 FPS je možné nastavit limit přibližně na 16ms.

V průběhu hry se na herní mapě postupně objevují monstra. O vytvoření instance třídy monster se postará samostatná metoda jádra napojená na časovač odpočítávající čas mezi příchody monster. Po každém vypršení časovače bude vytvořeno monstrum a umístěno do scény. Po určitém počtu vytvořených monster (bude záviset na nastavení jednotlivých map) bude časovač zastaven a nová monstra se již nebudou objevovat.

Po ukončení hry (výhra, prohra, dobrovolný odchod) budou herní objekty, scéna a její pohled zdestruovány, skóre získané v průběhu hry bude uloženo a jádro zobrazí hlavní menu.

3.2 Objekty hry

Herní objekty budou reprezentovány podtřídou třídy `QgraphicsItem` aby bylo možno umístit je do herní scény. Tyto objekty typicky obsahují rozhraní pro:

- získání a změnu pozice v rámci scény
- kontrolu a nastavení různých interních stavů objektu (např. hráč je živý nebo mrtvý, pytel zlata padá nebo stojí, apod.)
- získání hranic objektu, kde je již detekována kolize s jiným objektem

Objektům je také potřeba přiřadit nějakou vizuální podobu. Z toho důvodu bude vytvořena podtřída třídy `QgraphicsItem`, která bude obsahovat kód pro vykreslení objektu a jeho animací na obrazovku. Budeme jí říkat `Sprite`. Z této podtřídy pak budou dědit ostatní třídy kontrétních objektů. Vizuální podoba bude řešena pomocí tzv. *spritů* (bitmapové obrázky). Návrh třídy `Sprite` je detailněji popsán na straně 12.

Jelikož herní mapa bude moci plynule scrollovat, budou objekty uchovávat navíc pozici svého spritu. Tato pozice se bude počítat z pozice vlastního objektu a pozice hráče. Sprit se bude vykreslovat jen v případě, že se bude nacházet uvnitř viditelného výřezu mapy. Sám objekt pak bude rozhodovat, jestli jeho sprit bude zobrazen a kde bude umístěn.

3.2.1 Hráč

Objekt hráče ovládán uživatelem aplikace pomocí klávesnice. Samotný objekt nebude zpracovávat stisky klávesnice a podle nich měnit svou polohu. O to se postará jádro aplikace, které pak bude pomocí rozhraní měnit pozici objektu. Jak se hráč pohybuje, mění jádro hodnoty dlaždic na mapě podle toho, po které dlaždici přejel. Podle typu dlaždice se budou provádět i různé akce popsané v tabulce 3.1.

Může také nastat kolize s jiným pohyblivým objektem. V takovém případě jádro aplikace kontroluje s jakým objektem kolize nastala a podle toho vyvolá patřičné reakce. Přehled kolizí a reakcí na ně je znázorněn v tabulce 3.2.

3.2.2 Monstra

Objekty monster budou popisovat své chování v metodě, kterou bude volat jádro aplikace při každém timeoutu hlavního časovače. V této metodě provedou monstra jeden krok svého pohybu o předepsaný počet pixelů a updatují svůj sprit podle pozice hráče a pozice ve viditelném výřezu. Samotný pohyb se bude odvíjet od pozice hráče. Monstra budou vědět,

Typ dlaždice	Reakce hry
zemina	změna na tunel
drahokam	přičtení bodového zisku hráči změna na tunel
třešnička	přičtení bodového zisku změna na monster spawner hráč přejde do bonusového módu

Tabulka 3.1: Popis akcí při změně hodnoty dlaždice herní mapy.

Typ kolize	Dodatečná podmínka	Reakce hry
kolize s monstrem	hráč není v bonusovém módu	hráč zemře, sníží se mu počet životů, všechna monstra zmizí, hráč se objeví na výchozí pozici
	hráč se nachází v bonusovém módu	monstrum bude zabito, hráči se přičte bodový zisk
kolize s pytlem zlata	pytel zlata padá	hráč zemře, sníží se mu počet životů, všechna monstra zmizí, hráč se objeví na výchozí pozici
	hráč se nachází na stejné Y souřadnici	pytel zlata se posune

Tabulka 3.2: Popis akcí při kolizi monstra s jiným objektem.

kde se na herní mapě nachází hráč a podle toho rozhodnou, kterým směrem se vydají. Jádro aplikace pak zkontroluje, jestli nedošlo ke kolizi s hráčem nebo jiným objektem a podle toho provede určitou akci. Možné scénáře kolizí a reakce na ně popisuje tabulka 3.3. Kolize monster s hráčem popisuje tabulka 3.2.

Typ kolize	Dodatečná podmínka	Reakce hry
kolize se střelou	žádná	monstrum bude zabito, hráči se přičte bodový zisk
kolize s pytlem zlata	pytel zlata padá	monstrum se pohybuje s pytlem dolů po dopadu monstrum zemře.
	monstrum je typu Nobbin pytel zlata leží na stejné hodnotě Y	monstrum posune pytel zlata
	monstrum je typu Hobbin pytel zlata leží na stejné hodnotě Y	monstrum zničí pytel zlata

Tabulka 3.3: Popis akcí při kolizi monstra s jiným objektem.

Monstra budou moci změnit hodnotu dlaždice herní mapy jen v jednom případě a to, budou-li v Hobbin formě. Pokud přejdou přes jakoukoli dlaždici (kromě spawnerů a třešničky), jádro aplikace ji automaticky změní na tunel.

Do Hobbin formy přechází monstrum po náhodné době, která bude dána vzorcem:

$$timeout = (rand() \mod (max\_mut\_time - min\_mut\_time) + max\_mut\_time) \quad (3.1)$$

kde *min_mut_time* a *max_mut_time* je minimální a maximální čas po kterém se monstrum promění. Výsledek je v sekundách. Stejný vzorec bude aplikován při výběru času, po kterém se monstrum promění zpět.

3.2.3 Zlato

Zlato bude objekt, který se nachází ve dvou stavech. V prvním má vizuální podobu pytle se zlatem, reaguje na okolí a může se po něm pohybovat. Ve druhém stavu je ve formě rozsypaných zlatých nugetů, či mincí. Jediná možná interakce pak bude kolize s hráčem, což způsobí zmizení objektu a přičtení bodového zisku na hráčovo konto.

V prvním stavu bude chování objektu, podobně jako u monster, popsáno metodou jeho třídy, kterou zavolá jádro aplikace při každém timeoutu hlavního časovače. Pytel zlata zkontroluje hodnotu dlaždice pod sebou a v případě, že pod ní bude prázdné místo, začne padat, dokud nenarazí na konec tunelu. Podle toho, jak velkou vzdálenost urazí, se z něj po dopadu buď vysypou zlaťáky nebo zůstane nezměněn. Dále se bude kontrolovat kolize s hráčem, monstry nebo jinými objekty své třídy v průběhu pádu. Oproti kolizím a akcím, popsaným v tabulkách 3.2 a 3.3, které se zlatem souvisely, bude mít objekt navíc kolize a akce popsané tabulkou 3.4.

Typ kolize	**Dodatečná podmínka**	**Reakce hry**
kolize s pytlem zlata	zlato padá	oba pytle se rozbijí
	zlato je na stejné Y souřadnici zlato se pohybuje horizontálně	pytel se posune ve stejném směru

Tabulka 3.4: Popis akcí při kolizi pytle zlata s jiným pytlem.

3.2.4 Střela

Střela je objekt, který vznikne pouze, když hráč vystřelí. Bude se pohybovat určitou rychlostí směrem, do kterého byl hráč natočen ve chvíli výstřelu. Zanikne po nárazu do jiného objektu (mimo pytle se zlatem, kterým projde), po nárazu do zdi tunelu nebo po určitém čase (při výstřelu se spustí časovač, který ji při timeoutu zlikviduje). V případě, že střela narazí do monstra, monstrum zemře.

3.2.5 Animace spritů

Každý pohyblivý objekt má své animace. Animace tvoří většinou 3 snímky uložené společně v jednom souboru, v tzv. *spritesheetu*, které se cyklicky opakují. Výjimkou jsou speciální případy, kdy animace proběhne jen jednou (např. rozbití pytle se zlatem, smrt monstra nebo hráče). V tomto portu bude využit spritesheet ze hry Digger Remastered.

O animace a zobrazení herních objektů se postará třída `Sprite`, jenž bude podtřídou třídy `QGraphicsItem` a bázovou třídou pro třídy herních objektů. Základem bude obrázkový soubor obsahující všechny snímky všech herních objektů (spritesheet), který bude třídě předán jako instatnce třídy `QPixmap` při inicializaci. Tato třída, podle zadané pozice prvního snímku ve spritesheetu, načte dalších N snímků a bude mezi nimi přepínat a vykreslovat je v zadaném intervalu. Třída `Sprite` bude napojena na speciální časovač určující rychlost všech animací.

Vykreslování snímku bude provádět `QPainter` vždy, když přijde signál vypršení časovače. Po jeho vykreslení bude tento nahrazen snímkem následujícím.

3.2.6 Ozvučení hry

Jednotlivé objekty budou při různých akcích vydávat konkrétní zvuky. Pro ovládání zvukových efektů a hudby bude k dispozici třída, která nahraje všechny potřebné zvukové soubory a pomocí svých metod je bude přehrávat. Třída se bude jmenovat `Audio`.

Zvuky se budou dělit do dvou kategorií:

- zvukové efekty
- hudba na pozadí

O přehrávání obou typů zvuků se postará přehrávač médií v Qt typu `QAudioPlayer`. V třídě budou k dispozici dva tyto přehrávače – jeden pro efekty, druhý pro hudbu (ten však bude `static` neboť nebude možné aby hrálo více melodií najednou). Při konstrukci objektu se vytvoří pro melodie i pro efekty samostatné *playlisty* (typu `QPlaylist`), které se předají přehrávači. Hudba a efekty budou od sebe odděleny, tedy nebude možné přehrávat hudbu přehrávačem pro efekty a naopak. O samotný výběr skladby se postarají metody, které budou mít za parametr textový klíč identifikující melodii v playlistu. Podle tohoto klíče nastaví v playlistu příslušnou melodii a dají pokyn přehrávači, aby ji začal přehrávat.

Hudba na pozadí se bude cyklicky přehrávat po celou dobu hraní. Melodie se změní v případě, že hráč zemře, vstoupí do bonusového módu, bonusový mód skončí nebo vyhraje úroveň.

Zvukové efekty se budou přehrávat, pokud nějaký objekt provede určitou akci (např. hráč sebere drahokam nebo vystřelí). Každý herní objekt bude mít svou instanci třídy `Audio` aby si mohl, dle libosti, přehrávat příslušné efekty tak, jak potřebuje. Při každé akci, která vyvolá i zvuk, bude zavolána metoda této instance s klíčem identifikujícím zvukovou stopu, jenž se postará o přehrání příslušného efektu.

Základní balíček zvuků bude ve formátu `WAV` celý převzat z existující hry Digger Remastered. Jedná se o polyfonní melodie a efekty, které budou nahrány v průběhu hry programem *audacity*.

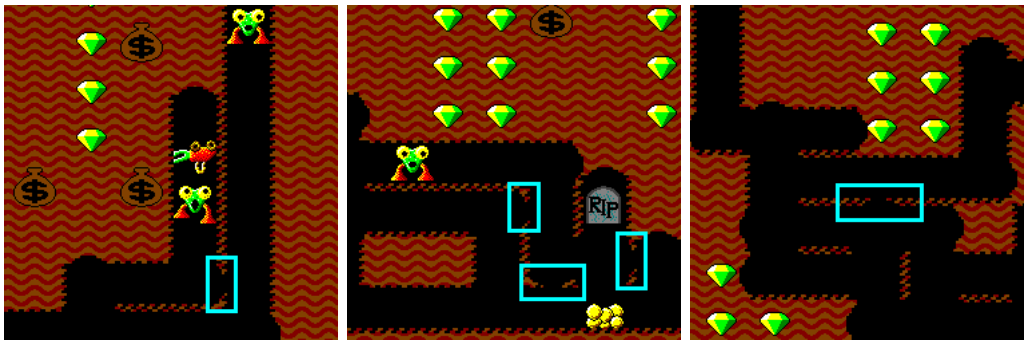
3.3 Herní plocha

Plocha, kde se odehrává děj hry, je reprezentována herní mapou, po které se budou pohybovat herní objekty. Objekty jsou na mapě reprezentovány svými sprity (viz str. 10). Tato mapa se načte při startu hry ze souboru, nebo se vygeneruje generátorem map. Mapa bude organizována do tzv. *dlaždic*. Data mapy de uloží ve 2D matici hodnot typu `int`. Hodnoty buněk budou reprezentovat hodnoty dlaždic, které se ve scéně vykreslí jako souvislý povrch. Ve hře se budou nacházet tyto typy dlaždic:

- Chodba – reprezentována prázdným místem přes které může přejít hráč i monstrum Nobbin.
- Zemina – při průchodu hráče nebo Hobbina se dlaždice změní na tunel.
- Drahokam – dlaždice obsahující jeden drahokam na popředí a zeminu na pozadí dlaždice.
- Zlato – dlaždice se zeminou na pozadí na které se objeví objekt pytle se zlatem.
- Player spawner – místo, kde se objevuje hráč. Dlaždice má formu tunelu, lze přes ni projít.

- Monster spawner – místo, kde se objevují monstra. Daždice má rovněž formu tunelu a lze přes ni projít.
- Třešnička – objeví se na místě monster spawneru, jakmile budou vytvořena všechna monstra.

Tento způsob organizace mapy řeší drobnou nepříjemnost v původní hře, kdy monstra prošla stěnou, kterou by teoreticky neměla vůbec projít (viz obrázek 3.1).



Obrázek 3.1: Azurovou barvou jsou zvýrazněny oblasti, kterými monstra mohou projít i když to není na první pohled zřejmé.

Herní mapa bude po svém vytvoření přístupná všem herním objektům, které se po ní mohou pohybovat. Bude tedy vhodné modelovat ji podle návrhového vzoru *singleton*. [12]

3.3.1 Formát souboru s mapou

Data a nastavení herní mapy budou uložena ve formátu `text/plain` v externím textovém souboru s příponou `.lev` uloženém v adresáři `~/.cutedigger/maps`. V případě generovaných map se data a nastavení nikam neukládají, ale zůstanou načtena v paměti.

Hra bude obsahovat sadu základních map, které se zakompilují do výsledného binárního souboru. Pokud bude adresář pro uložení souboru s mapami prázdný, základní mapy se tam naklopírují.

Soubor s mapou obsahuje na prvních řádcích parametry mapy a až poté následuje matice s hodnotami dlaždic. Parametry mapy nemusí být uvedeny v přesném pořadí a některé z nich mohou chybět. Na každém řádku je vždy jeden parametr ve tvaru `název:hodnota`. Pokud chybí povinný parametr, skončí načítání mapy chybou. Povinné parametry jsou:

- `x:integer` – šířka mapy v dlaždicích.
- `y:integer` – výška mapy v dlaždicích.
- `monsters:integer` – počet monster na mapě.

Volitelné parametry:

- `name:string` – název mapy.
- `dirt_texture:integer` – vzor pro dlaždice zeminy. Hra obsahuje osm vzorů zeminy pro dlaždice, parametr může tedy nabývat hodnot $N \in \langle 0, 7 \rangle$. V případě, že je zadáno číslo $N > 8$, použije se jako hodnota číslo $N \mod 8$.

- `next_map:string` – jméno souboru mapy, která se načte po úspěšném dohrání této úrovně. Tímto způsobem je řešeno řetězení úrovní. Pokud není zadán žádný soubor, hra po dohrání úrovně končí.

Vlastní hodnoty dlaždic se nacházejí za parametry a uvádí je řetězec `map_data` zakončený znakem konce řádku. Mají formu matice, kde jsou hodnoty odděleny jednou mezerou. Počet řádků matice odpovídá počtu řádků zadaným parametrem `y` a počet sloupců odpovídá hodnotě zadané parametrem `x`. Pokud sloupce nebo řádky přebývají, jsou ignorovány. Pokud řádek nebo sloupec chybí, skončí načítání mapy chybou.

```
x:10
y:10
name:Level 1
monsters:3
dirt_texture:5
next_map:another_map.lev
map_data:
1 1 1 1 1 1 1 1 1 1
1 3 0 0 0 0 0 0 5 1
1 1 1 1 1 1 1 1 1 1
1 0 0 0 0 1 8 8 8 1
1 1 7 1 0 1 8 8 8 1
1 0 1 1 0 1 1 1 1 1
1 0 1 1 0 0 0 0 0 1
1 0 1 1 1 8 8 1 1 1
1 1 1 1 0 8 8 1 1 1
1 1 1 1 1 1 1 1 1 1
```

Obrázek 3.2: Ukázka souboru jednoduché herní mapy s rozměry 10×10 dlaždic. Na mapě se postupně objeví nejvýše tři monstra.

3.3.2 Pseudo-nekonečná herní mapa a scrolling

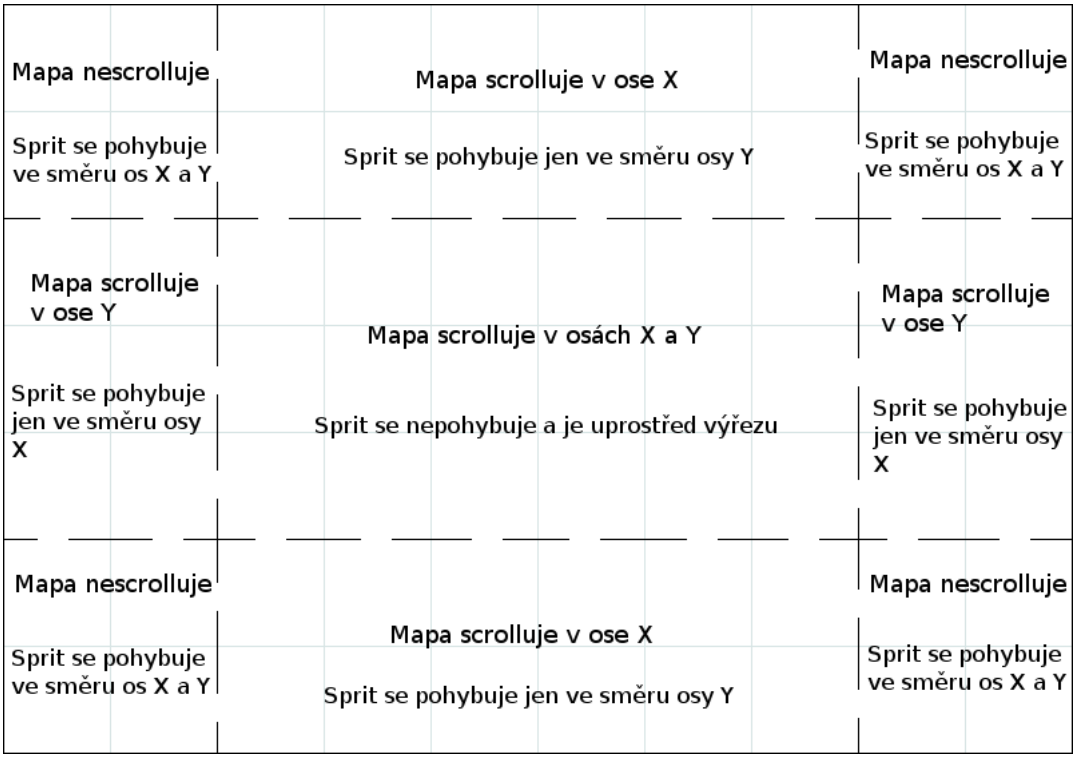
Vlastní dlaždice mají výšku i šířku 40 pixelů. Při velikosti herního okna 800×600 pixelů je v něm možné zobrazit najednou 20×13 dlaždic.[1] Mapy však mohou mít větší velikost než je možné v jednom okně najednou zobrazit. Proto je nutné implementovat plynulý scrolling prostředí v případě, že by hráč vyjel ze zobrazené části mapy.

Při scrollingu se jeví, že se výřez mapy s hráčem uprostřed plynule posunuje po herní mapě. Ve skutečnosti to bude fungovat tak, že uprostřed výřezu bude zobrazen sprit hráče, který se nebude hýbat. Kolem něj se pak bude pohybovat mapa v opačném směru, než je požadovaný směr hráče. V tomto případě je nutno ošetřit krajní oblasti herní mapy, kde se bude scrollovat buď jen horizontálně nebo jen vertikálně nebo vůbec. Jednotlivé oblasti s typem scrollování jsou znázorněny na obrázku 3.3. Velikosti těchto oblastí jsou dány velikostí herního okna a velikostí mapy.

Při scrollingu uprostřed mapy se viditelné dlaždice odvíjejí od hráčovi reálné pozice na mapě. Sleduje se X a Y souřadnice dlaždice, na které se nachází a relativní pozice jeho středového bodu vůči levému hornímu rohu této dlaždice. Podle velikosti herního okna se vypočítá, kolik dlaždic a pixelů od hráče ve všech směrech se má vykreslovat. Výpočet se provádí podle vzorců:

$$width\_in\_tiles \doteq \frac{window\_width}{tile\_width} \tag{3.2}$$

[1] S odečtením výšky HUD, která je rovna výšce dvou dlaždic.



Obrázek 3.3: Znázornění oblastí na mapě, kde scrolling probíhá odlišným způsobem.

$$height\_in\_tiles \doteq \frac{window\_height - HUD\_height}{tile\_height} \tag{3.3}$$

$$x\_margin = \frac{width\_in\_tiles}{2} \tag{3.4}$$

$$y\_margin = \frac{height\_in\_tiles}{2} \tag{3.5}$$

Výsledné hodnoty jsou v dlaždicích. Pro hodnoty okraje v pixelech stačí vynásobit x_margin a y_margin rozměry dlaždice.

Odečtením a přičtením těchto hodnot od hodnot pozice dlaždice, na které je hráč, se získá minimální a maximální X a Y souřadnice ohraničující výřez mapy. Tento výřez však ještě nebude vykreslen. Pokud by totiž hráč přešel z jedné dlaždice do druhé, výřez by poskočil.

Je tedy nutno počítat také s relativní vzdáleností hráče od levého horního rohu dlaždice, ve které stojí (znázorněno na obrázku 3.5). O tuto vzdálenost (počítána v pixelech) se okolí posune v opačném směru, než se hráč pohybuje. Poté se může výřez vykreslit.

Při scrollování musí také herní objekty, vyjma hráče, upravit pozici svého spritu, aby se také plynule posouval spolu s prostředím. Sprit se posune ve stejném směru, jako prostředí a o stejnou vzdálenost.

3.3.3 Generátor map

Generátor map bude pracovat v několika krocích v tomto pořadí:

- generování shluků drahokamů (obrázek 3.6)

- generování tunelů (obrázek 3.7)
- umístění spawneru pro hráče a monstra (obrázek 3.8)
- umístění pytlů se zlatem (obrázek 3.9)

Na vstupu obdrží generátor požadované rozměry generované mapy. Ihned, po alokování matice pro data, vyplní všechny buňky hodnotou pro zeminu a začne umisťovat shluky drahokamů na náhodné pozice. Shluk bude mít tvar obdélníku, jehož strany budou mít náhodnou velikost od jednoho drahokamu do určité maximální hodnoty (aby se nestalo, že shluky zaberou příliš velkou plochu mapy). Před umístěním shluku je ověřeno, že se neprotíná s již existujícím. Pokud je v určeném místě již jiný shluk, vybere se nová náhodná pozice a velikost. Jinak se umístí na zadané místo a nastaví se hodnoty příslušných buněk v matici. Celkový počet shluků na herní mapě se určí pomocí vzorce:

$$gem\_chunk\_count = \frac{map\_width \times map\_height}{128} + 1 \tag{3.6}$$

Výsledek se zaokrouhlí na celé číslo.

V druhém kroku se do mapy umístí tunely, přičemž mohou překrýt již existující drahokamy. Tunely se budou generovat upraveným algoritmem *Drunkard Walk* (viz [9]). Tento algoritmus vybere náhodně jeden ze čtyř směrů od aktuálního bodu, kterým se vydá. Modifikace spočívá v tom, že se do zvoleného směru neposune o jednu buňku, ale o náhodně zvolený počet buněk v rozmezí $\langle 3, 10 \rangle$. Cestou modifikuje hodnoty jednotlivých buněk. V případě, že by se mělo cestovat zpět, vybere směr znovu. Pokud narazí na buňku, kde už jednou byl, algoritmus končí.

Počet tunelů bude určen podle vzorce:

$$tunnel\_count = \frac{map\_width \times map\_height}{383} \tag{3.7}$$

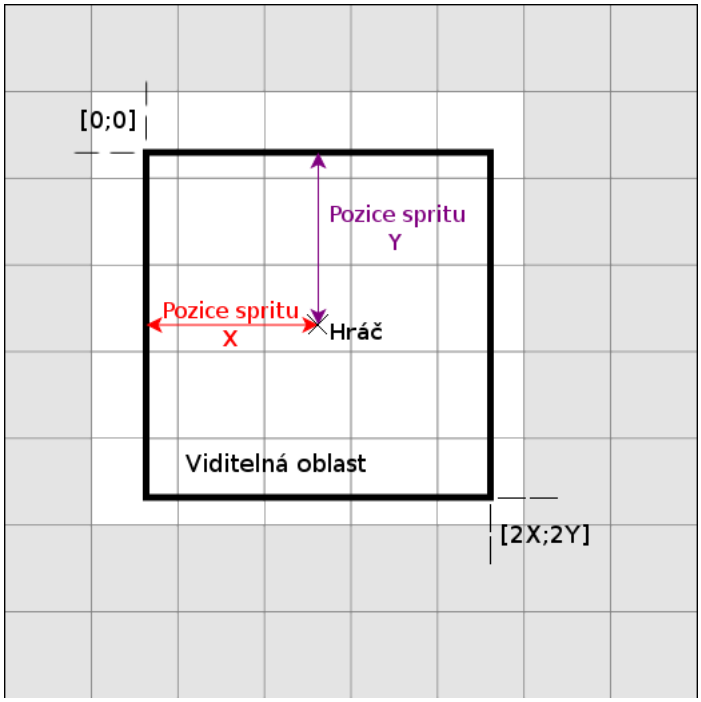
Výsledek se zaokrouhlí na celé číslo. Generování první chodby bude probíhat tak dlouho, dokud algoritmus nenarazí na svou vlastní chodbu. V případě dalších chodeb skončí, jakmile narazí na jakoukoliv chodbu. V průběhu vytváření chodeb se postupně vytvoří jejich seznam, který bude použit při umisťování spawneru hráče a monster.

V následujícím kroku se projdou seznamy chodeb. Zjistí se, které na sebe navazují a umístí se do nich spawner hráče a monster tak, aby bylo zajištěno, že monstra budou mít přístup k hráči. Pokud by se tak nestalo, mohl by si hráč dávat při kopání cestiček pozor, aby se neprokopal k monstrům a vyhrál by tak velmi jednoduše.

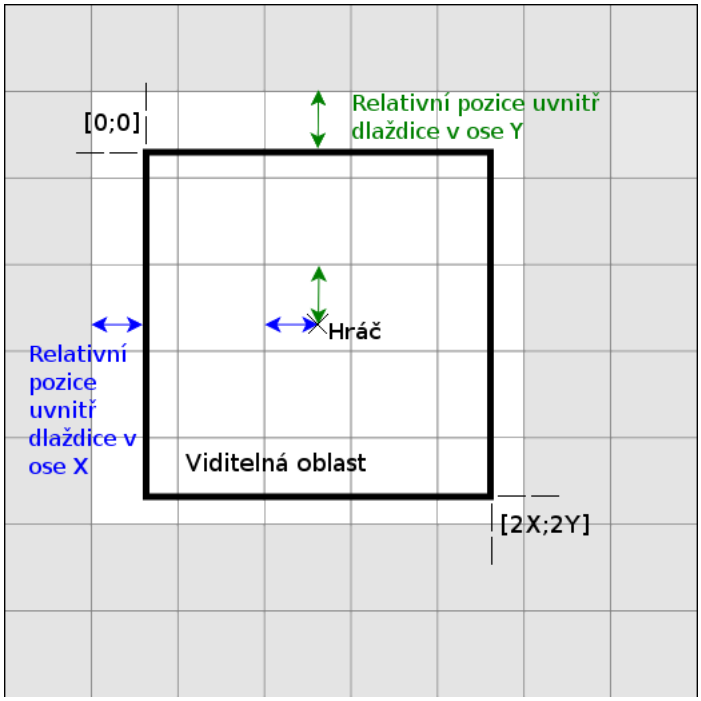
V posledním kroku se jen zvolí několik náhodných míst, kam se umístí pytle se zlatem. Zlato se bude umisťovat pouze tam, kde je zemina. Počet pytlů bude dán vzorcem:

$$gold\_count = \frac{map\_width \times map\_height}{64} \tag{3.8}$$

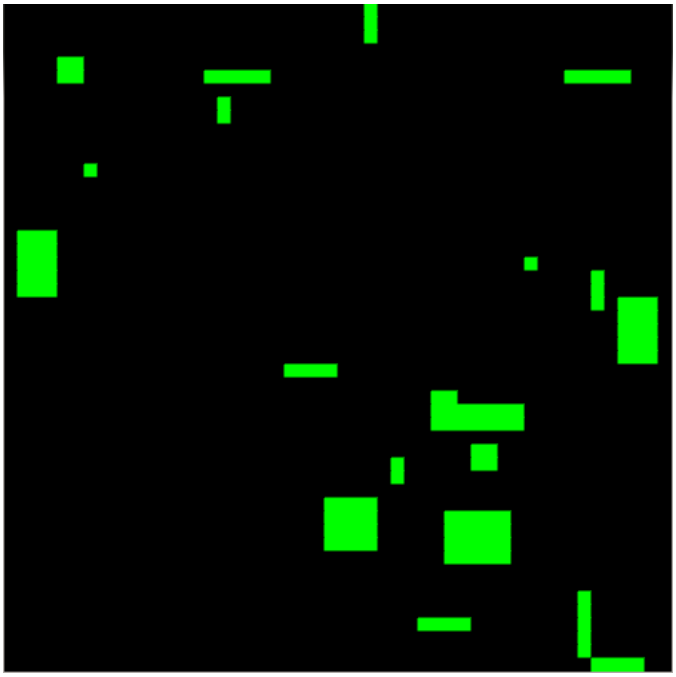
Výsledek se opět zaokrouhlí na celé číslo. Jakmile je umístěno i zlato, je matice mapy vygenerována a může být předána instanci třídy reprezentující mapu.



Obrázek 3.4: Znázornění výřezu mapy. Bílou barvou jsou znázorněny vykreslené dlaždice. Výsledná viditelná oblast je však ještě ořezána samotným oknem, takže dlaždice v krajních oblastech nemusí být vidět celé.



Obrázek 3.5: Posun viditelné oblasti podle relativní pozice hráče uvnitř dlaždice.



Obrázek 3.6: Mapa s vygenerovanými shluky drahokamů (zelená barva). Černá barva reprezentuje zeminu.

Obrázek 3.7: Mapa s vygenerovanými chodbami (bílá barva).

Obrázek 3.8: Mapa s vygenerovaným spawnerem hráče (modrá barva) a spawnerem příšer (červená barva).

Obrázek 3.9: Hotová mapa, na které jsou umístěny i pytle se zlatem (žlutá barva).

Kapitola 4

Implementace

4.1 Třída `GameSession`

Třída implementuje základní jádro aplikace, které vytváří instance ostatních tříd a stará se o interakce mezi nimi. Je potomkem třídy `OMainWindow`. Konstruktor nahraje nastavení uložené v souboru `settings.conf`, vytvoří instanci třídy `Audio` a `Hud`. Tyto dvě instance budou k dispozici v průběhu všech her, proto se vytvoří už zde. Vytvoří se zde také instance třídy `QPixmap` do které se nahraje spritesheet s herní grafikou. Nastavení je uloženo ve formátu `ini` a o parsování se stará třída `QSettings`. Do konstruktoru se jí zadá cesta k souboru s nastavením a formát obsahu (`QSettings::IniFormat`). Ihned po spuštění aplikace se volá metoda `GameSession::showMainMenu()`. Tato volá metodu `mainMenu()` třídy `Menu`, která obsahuje pouze kód GUI mimo vlastní hru a metody pro načtení seznamu uložených map a nejvyšších skóre. Sloty pro zobrazování celých menu jsou následující:

- `GameSession::showMainMenu()` zobrazující hlavní menu hry
- `GameSession::showNewGameMenu()` zobrazující menu pro start nové hry
- `GameSession::showOptionsMenu()` zobrazující menu pro nastavení aplikace a hry

Sloty pro zobrazování jednotlivých položek nastavení hry jsou:

- `GameSession::setGameplay()` pro menu nastavení prostředí hry
- `GameSession::setVideo()` pro menu nastavení zobrazení
- `GameSession::setSound()` pro menu nastavení hlasitosti zvuku a hudby
- `GameSession::setControls()` pro menu nastavení ovládání hry

Pro vlastní hru je nejdůležitější slot `GameSession::showNewGameMenu()`, který zobrazuje menu pro výběr startovní mapy, dává možnost vygenerovat novou mapu a umožňuje výběr jména hráče.

Po klepnutí na tlačítko pro start hry se spustí slot `GameSession::prepareGame()`. Tento slot uloží nastavení hry do souboru `settings.conf` a nastaví do HUD jméno hráče. Dále se zavolá slot `GameSession::startGame()`, který provede inicializaci všech herních objektů potřebných pro start hry.

Nejprve se inicializuje herní mapa. Do instance třídy `Map` se předá ukazatel na spritesheet. Pokud uživatel zvolil vygenerování nové mapy a zadal rozměry, spustí se metoda `MapGen::generate()`. Ta do instance objektu třídy `Mapgen` vygeneruje herní mapu,

která je poté vrácena přes parametr metodou `MapGen::returnMatrix(int ***m)`. Následně se vytvoří instance třídy `Map`, jíž je matice s daty herní mapy předána metodou `Map::loadMapFromMatrix(QSize size, int** matrix)`. Pokud uživatel generovat mapu nechtěl a vybral jednu z dostupného seznamu, načtou se její data do objektu mapy metodou `Map::loadMap(QString filename)`, kde její název udává parametr `filename` zobrazeného seznamu map.

Dále se vytvoří objekt hráče, což je instance třídy `Player`. Při inicializaci se objektu hráče předá spritesheet s herní grafikou a ukazatel na objekt mapy. Poté se hráč umístí metodou `Player::spawn(int row, int col)` na příslušnou dlaždici, která je určená jako výchozí bod. Nakonec se z dat herní mapy zjistí, kolik pytlů zlata se má do ní umístit, vytvoří se instance třídy `Gold` a umístí se na příslušné dlaždice mapy. Všechny objekty zlata jsou navíc uchovávány v kontejneru `std::vector`, aby byly později lehce k dispozici.

Nyní se začne budovat herní scéna a pohled do ní. Scéna je tvořena instancí třídy `QGraphicsScene`. Pomocí metody `QGraphicsScene::addItem(QGraphicsItem* item)` se do scény přidá objekt hráče, pytle se zlatem a objekt mapy. Dále je vytvořen pohled do scény instancí třídy `QGraphicsView` a metodou `QGraphicsView::setScene(QGraphicsScene)` je mu předána scéna, do které bude mířit. V pohledu jsou zakázány scrollbary a je mu nastaven mód úplného překreslování. To znamená, že pohled bude pokaždé překreslen úplně a ne jen v určité části.

Instance třídy `QWidget` vytvoří plochu pro zobrazení herní scény. Je do ní vložen layout složený z instance `QGraphicsView` a `Hud`. Nastaví se jí rozměry a nastaví se hlavnímu oknu jako *centrální widget*. V tuto chvíli může být okno zobrazeno.

Nyní se spustí základní časovač `base_timer`, časovač pro animace `animation_timer` a časovač pro umisťování monster `monster_spawn_timer`. Základní časovač má nastavenou hodnotu pro odpočet 16 ms. Tato bude udržovat FPS hry na maximální hodnotě 60 snímků ($\pm$ 3 snímky). To pro plynulý běh hry zcela postačuje. Vyšší hodnoty hru příliš urychlují. Časovač pro animace je nastaven na 100 ms, takže plynulost animací neovlivní ani nižší FPS.

Po spuštění časovačů se napojí jejich `timeout()` signály na sloty dalších objektů. Timeout hlavního časovače je napojen na slot `doGame()`, který obstarává hlavní herní smyčku a na slot `Hud::countFrames()`, který zajišťuje počítání FPS v průběhu hry. Timeout časovače animací se napojí na slot `Player::nextFrame()`, který mění snímky animace hráče. Na instance monster se časovač animací napojí až při jejich vzniku. Časovač pro umisťování monster je napojen na slot `GameSession::createMonster()`, v němž se do hry umisťují monstra a napojuje časovač na jejich slot `nextFrame()`.

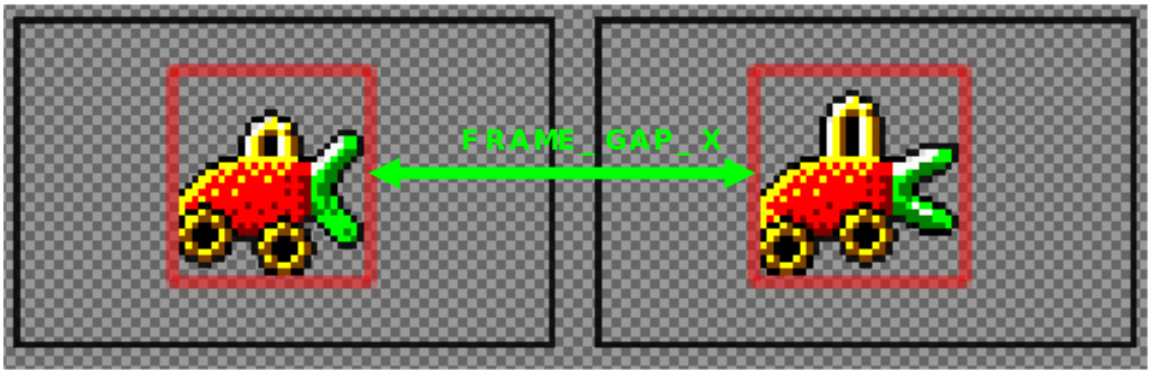
4.2 Třídy `Sprite` a `GameObject`

`Sprite` je třída, která dědí z `QGraphicsObject`. Její instance je možné vkládat do herní scény a manipulovat s nimi. Navíc implementuje i zobrazení spritů, animací a ozvučení herního objektu. Udržuje informaci o pozici objektu v rámci scény a také o pozici spritu v rámci viditelného výřezu. Tyto dvě pozice mohou být stejné, nebo se lišit v závislosti na tom, kde se objekt nachází na herní mapě (viz obr. 3.3). Obě pozice jsou typu `QPoint`.

Konstruktor nejprve nastaví pozici spritu na souřadnice (0,0) a šířku a výšku samotného objektu na velikost podle maker `TILE_SIZE_X` a `TILE_SIZE_Y`. Hodnoty maker odpovídají rozměrům dlaždice v pixelech a jsou definovány v souboru `config.h`.

Instanci se musí předat ukazatel na instanci `QPixmap`, v níž je uložen spritesheet. Předá se jako parametr metody `Sprite::initSprite(Qpixmap *)`, která musí být zavolána po

vytvoření každého herního objektu. Metoda dále nastaví základní velikost spritu podle maker `FRAME_SIZE_X` a `FRAME_SIZE_Y` (definována v `config.h`). Protože snímky animací nejsou ve spritesheetu hned u sebe, ale je mezi nimi odstup, nastaví tato metoda vzdálenost o kterou se posune načítání dalšího snímku. Je dána součtem maker `FRAME_SIZE_X` (šířka snímku) a `FRAME_X_GAP` (reálná vzdálenost mezi snímky, viz obr. 4.1). O tuto vzdálenost bude `QPainter` přeskakovat při načítání snímků animace ze spritesheetu.



Obrázek 4.1: Znázornění reálné vzdálenosti mezi snímky, která se musí přeskočit od konce jednoho snímku po začátek druhého.

Vykreslování snímku ze spritesheetu provádí `QPainter` svou metodou `drawPixmap(int x, int y, QPixmap* pixmap, int sx, int sy, int sw, int sh)`. Hodnoty parametrů `sx` a `sy` udávají souřadnice v obrázku, od nichž se bude vykreslovat výřez. Parametry `sw` a `sh` udávají výšku a šířku vykresleného výřezu. V tomto případě jsou nastaveny na výšku a šířku snímku. Změnu snímků provádí slot `Sprite::nextFrame()`, který se spouští při timeoutu časovače animací. Způsobí posunutí parametru `sx` v metodě `drawPixmap`, takže snímek se načte ze správného místa.

`GameObject` je podtřídou třídy `Sprite` a je zároveň bázovou třídou pro podtřídy `Monster`, `Projectile` a `Gold`. Vznikla zejména kvůli jednotné práci s herními objekty, jejichž chování je závislé na pohybu hráče po herní ploše. U objektů je důležité vědět, jestli se nacházejí uvnitř viditelné oblasti a podle jejich pozice a podle pozice hráče zobrazovat jejich sprit. Aktualizaci spritu provádí metoda `GameObject::updateSprite()`. Pozice hráče se získává z instance třídy `Map` metodou `QPoint Map::getPlayerLoc()`. Velikost výřezu se také získá z instance herní mapy uvnitř metody `GameObject::init()`. Pokud je objekt mimo viditelnou oblast, vykreslování jeho spritu se zastaví, dokud nebude opět uvnitř.

4.2.1 Třída `Player`

Třída `Player` je podtřídou třídy `Sprite` a implementuje rozhraní pro ovládání objektu hráče. Třída se také stará o to, aby byl hráčův sprit vždy ve správné pozici podle toho, jak se pohybuje po herní mapě. Veškeré potřebné informace získává při použití metody `Player::initPlayer(Map*)`. Uvnitř ní se hráči nastaví herní mapa, která byla předána parametrem, jako aktivní mapa, se kterou bude interagovat. Kromě toho se z mapy zjistí, jak veliké mají být okraje výřezu. Podle této informace a podle pozice objektu hráče na herní mapě upravují metody `Player::updateSpriteX()` a `Player::updateSpriteY()` pozici spritu ve viditelném výřezu.

4.2.2 Třída `Monster`

Monstra jsou instance třídy `Monster`, která je podtřídou třídy `GameObject`. Obsahuje metodu `Monster::behavior()`, která provede jeden krok chování monster, pokud je zavolána. Monstrum si volí směr svého pohybu podle její pozice vůči hráči tak, aby její další směr byl, pokud možno, směrem k hráči. Pokud nelze jít tím směrem, který zaručí nejlepší zmenšení vzdálenosti mezi monstrem a hráčem, nastaví se směr s druhým nejlepším výsledkem, atd. Jako poslední možnost monstrum použije směr opačný, než jakým se doposud pohybovalo. V případě, že hráč je v bonusovém módu, obrátí se pořadí výběru směrů. Monstrum se pak snaží jít směrem, který zaručí, že bude od hráče co možná nejdále. Výběr směru ovlivňuje, v prvních pěti úrovních, také náhodný faktor. V první úrovni existuje 25% šance, že monstrum zamění nejprioritnější směr za druhý nejméně prioritní. S každou další úrovní se tato šance sníží o 5%. Od šesté úrovně v pořadí náhoda směr neovlivňuje a monstra jej mění jen, pokud musí.

Po vytvoření je monstrum ve formě Nobbin. V konstruktoru se spustí časovač, který odpočítává dobu, po jejímž uplynutí je monstrum na určitý čas přeměněno do formy Hobbin. Tento je dán vzorcem 3.1. Po přeměně zůstává způsob výběr směru pohybu stejný, mění se však interakce s okolím, kde monstrum neomezují dlaždice se zeminou. Monstrum pak jde nejkratším možným směrem ke hráči. Monstru se rovněž změní vizuální podoba nastavením začátku animace na příslušný snímek ve spritesheetu.

4.2.3 Třída `Gold`

Další podtřídou třídy `GameObject` je třída `Gold` modelující pytle se zlatem. Chování její instance je popsáno metodou `Gold::behavior()`, která kontroluje stav dlaždic pod sebou a podle toho mění stav objektu. Změna stavu vždy způsobí změnu animace.

Interakci s ostatními objekty řídí metoda `GameSession::doGold()`. Detekuje se kolize jak s hráčem, tak s monstry a s ostatními pytli zlata. V případě monster a hráče se detekuje i ze které strany kolize nastala a dochází k posunutí pytle zlata v ose X ve směru pohybu kolidujícího objektu. V případě kolize zespoda se nad příslušným objektem zavolá metoda `kill()`, která nad objektem nastaví příznak smrti a zakáže vykreslování jeho spritu.

4.2.4 Třída `Projectile`

Tato třída modeluje vystřelený projektil, který se objeví pouze, pokud hráč stiskne klávesu pro výstřel a má nabito. Veškerý konstruování, destruování a pohyb projektilu řeší třída `GameSession`, která zjistí metodou `Player::getDirection()` směr, kterým se hráč dívá a v tomto směru mění pozici objektu metodou `GameSession::doProjectile()`. Po výstřelu se spustí časovač, jehož vypršení zavolá slot `GameSession::projectileEnd()`, který projektil zdestruuje.

4.3 Třída `Map`

Herní plocha (mapa) je modelována třídou `Map`, která v sobě zapouzdřuje hodnoty dlaždic a nastavení mapy. Všechny herní objekty dostanou při inicializaci ukazatel na její instanci. Není však modelována jako čistý singleton.

Instance třídy je vytvořena při startu hry v metodě jádra `GameSession::startGame()`. Pokud je mapa načítána ze souboru, přečte se její velikost z příslušných řádků, naalokuje se matice potřebné velikosti a podle dat ve zdrojovém souboru se nataví hodnoty dlaždic.

Pokud je mapa generována procedurálně, je její velikost načtena z prvků `QLineEdit` z GUI. Instanci třídy se předá metodou `Map::initMap(QPixmap *)` spritesheet, který obsahuje i sprity dlaždic.

Předtím, než začne vykreslování mapy, se nastaví příznaky podle toho, jestli a kde velikost mapy přesahuje velikost okna. Pro mapy, které jsou širší nebo delší, je upraven scrolling, takže probíhá pouze v jedné z os nebo vůbec. Nastavení příznaků probíhá při alokaci paměti, kde jsou již rozměry mapy a okna známy.

Poté je nutné ve všech směrech nastavit vzdálenost od hráče, po kterou budou vykreslovány dlaždice a ostatní objekty. To provede metoda `Map::setScrollingMargins(QSize)`, která nastaví vzdálenosti v osách x a y. Vzdálenosti jsou počítány v dlaždicích i v pixelech.

Před vykreslením v metodě `Map::paintEvent()` se vypočítá minimální a maximální řádek a sloupec, který bude z matice vykreslen. Při vykreslování se tedy v cyklu neprochází celá matice herní mapy, ale pouze část ohraničená těmito hodnotami. Výpočet probíhá uvnitř metody `Map::setVisibleArea()`, která je volána pro každé překreslení. Výpočet hranic totiž závisí na pozici hráče. Bere se zde v úvahu také pozice hráče uvnitř dlaždice a o tuto pozici je pak vykreslení dlaždic (a také ostatních objektů) posunuto. Docílí se tak efektu plynulého scrollingu.

Vykreslování probíhá ve dvou průchodech. Nejdříve se vykreslí dlaždice, které obsahují zeminu (včetně dlaždic s drahokamy a pytli zlata, které mají zeminu na pozadí) a až při druhém průchodu se vykreslí tunely a ostatní dlaždice. To proto, že tunely mají nepravidelné stěny a jsou o trochu větší než normální velikost dlaždice. Kdyby bylo vše vykresleno v jediném průchodu, byly by pravé a spodní strany tunelů hladké, protože by je překreslily sousedící dlaždice se zeminou.

4.3.1 Třída `MapGen`

Instance třídy `MapGen` slouží ke generování herních map. Generátoru jsou předány pouze požadované rozměry vygenerované mapy. Je zde metoda pro nastavení *seedu*, ta ale není nikde použita. Mapa je generována metodou `MapGen::generate()` a vrácena přes parametr v metodě `MapGen::returnMatrix(int ***)`.

Generátor map vygeneruje, podle zadaných rozměrů, matici a všechny hodnoty nastaví na hodnotu `TILE_DIRT` (viz soubor `config.h`). Tím se celá mapa vyplní zeminou. V dalším kroku jsou do mapy umístěny drahokamy metodou `MapGen::putGems()`. Poté jsou do mapy umísťovány tunely metodou `MapGen::putTunnels()`.

Při generování tunelů se do mapy umístí určitý počet bodů, které reprezentují vrtáky. Jsou to obyčejné instance třídy `QPoint`, které se pohybují po matici a měnit její hodnotu dlaždic na `TILE_EMPTY`. Pokaždé je aktivní pouze jeden vrták a jakmile skončí s tvorbou tunelu, přijde na řadu další, dokud nebyly použity všechny.

Předtím, než se vrták pohne, zvolí náhodně délku tunelu v intervalu $\langle 3, 10 \rangle$ dlaždic a směr. Pohybuje se pouze nahoru, dolů, doleva a doprava. Cestou modifikuje hodnoty dlaždic a každou dlaždici, kterou projde, uloží do seznamu, který reprezentuje tunel. Jakmile dorazí do cíle, opakuje stejné chování. Činnost vrtáku končí ve chvíli, kdy narazí na vykopaný tunel. Poté zanese nový tunel do seznamu tunelů, který je později využit pro výběr pozice umístění hráčů a monster. Pokud prorazí do tunelu, který předtím nevykopal on sám, nevkládá do seznamu tunelů novou položku, ale připojí seznam dlaždic k tomu tunelu, ve kterém se objevil. První vrták končí jen v případě, že narazí do svého vlastního tunelu.

Po vygenerování tunelů se do nich vloží metodou `MapGen::putSpawners()` spawnery pro hráče a monstra. Ze seznamu chodeb se náhodně vybere jedna a z ní se vybere jedna

dlaždice, do které se umístí spawner hráče. Do stejné chodby se na jinou dlaždici umístí i spawner pro monstra.

Nakonec se do mapy na pole, kde nejsou drahokamy ani tunely, umístí pytle se zlatem metodou `MapGen::putGold()`. Zde se pouze volí náhodné místo a pokud je na něm zemina, umístí se tam zlato.

4.4 Třída `Audio`

O práci se zvukem se starají instance třídy `Audio`. Každý herní objekt má svoji vlastní instanci, která je volána, když je potřeba. Jádro aplikace má také svoji vlastní, která slouží zejména pro změnu hudby na pozadí.

Uvnitř se nachází statické instance dvou tříd – `QMediaPlayer` pro přehrávání hudby a `QMediaPlaylist` uchovávající hudbu. Jedna instance obou tříd stačí, protože více melodií najednou nemá smysl přehrávat. Dále každá instance obsahuje `QMediaPlayer` pro přehrávání zvukových efektů, kterých může být přehráváno v jednu chvíli více.

Zvukové soubory jsou uloženy ve formátu `wav` v externích souborech. Hudba je nahrána v konstruktoru rovnou do statické instance `QMediaPlaylist`. Zvukové efekty jsou nejprve postupně nahrány do seznamu typu `QList <QMediaContent>` a poté jsou předávány jednotlivým playlistům pro efekty v každé nové instanci třídy `Audio`.

Zvuky jsou v playlistech uloženy pod indexem. Pro jednodušší výběr přehrávaného zvuku uchovává třída seznam `QMap <QString, int>`, kde první část dvojice je název zvuku nebo hudby a druhá část je index do playlistu. Tímto způsobem je možné v metodách `Audio::playSound(Qstring)` a `Audio::playMusic(Qstring)` vybírat zvuk pro přehrání podle názvu.

Při testování se ukázalo, že `QMediaPlayer` není pro zvukové efekty nejvhodnější i v případě, že má nastaven příznak `QMediaPlayer::LowLatency`. Vhodnější volbou by byla třída `QSoundEffect`, která však odmítala na linuxu přehrát jakýkoli zvuk. Proto jsem se rozhodl, že `QMediaPlayer` pro práci se zvukem zachovám.

Kapitola 5

Nasazení

5.1 Tvorba .deb a .rpm balíčků

Pro tvorbu obou typů balíčků je využit nástroj `CPack`, který je distribuován s programem `CMake`. Aby mohl `CPack` generovat balíčky, je nutno nastavit v souboru `CMakeLists.txt` proměnnou `CPACK_GENERATOR` hodnotami `"DEB"` nebo `"RPM"` podle toho, který balík je požadován. Dále se musí nastavit několik proměnných pro metadata balíčků. [10]

Následující volby jsou společné pro oba typy balíčků:

- `CPACK_PACKAGE_NAME` – Název balíčku.
- `CPACK_PACKAGE_VERSION` – Verze balíčku.
- `CPACK_PACKAGE_DESCRIPTION` – Dlouhý popis.
- `CPACK_PACKAGE_DESCRIPTION_SUMMARY` – Zkrácený popis.
- `CPACK_PACKAGE_CONTACT` – Kontakt na osobu, která balíček udržuje. Je nutno uvést platný email.
- `CPACK_PACKAGE_VENDOR` – Vydavatel nebo prodejce.
- `CPACK_PACKAGE_FILE_NAME` – Název vygenerovaného balíčku. Zde je nastaveno spojení proměnných `CMAKE_PROJECT_NAME` a `CPACK_PACKAGE_VERSION`.

Pro `RPM` balíčky jsou specifikovány následující volby, které nemají svou obdobu mezi společnými volbami:

- `CPACK_RPM_PACKAGE_RELEASE` – Číslo vydání balíčku. Pokud není uvedeno, použije se hodnota `"1"`.
- `CPACK_RPM_PACKAGE_REQUIRES` – Závislosti balíčku. Tato volba je velmi důležitá.
- `CPACK_RPM_PACKAGE_LICENSE` – Licence aplikace. Jako licence byla zvolena `GNU GPL`.
- `CPACK_RPM_PACKAGE_GROUP` – Skupina, pod kterou se obsah balíčku řadí. Zde je použita skupina `Amusements/Games`.

`RPM` balíčky mají navíc tyto volby:

- `CPACK_DEBIAN_PACKAGE_DEPENDS` – Závislosti balíčku. Velmi důležité uvádět.

- `CPACK_DEBIAN_PACKAGE_SECTION` – Zařazení obsahu balíčku. Zde nastaveno `games`.

Jakmile jsou všechny proměnné v souboru `CMakeLists.txt` nastaveny, je programem `CMake` vygenerován, kromě souboru `Makefile` také konfigurační soubor `CPackConfig.cmake`. Ten obsahuje nastavení pro program *CPack*, který, pokud je spuštěn s tímto konfiguračním souborem, vygeneruje příslušné balíčky.

Balíčky na přiloženém CD a na portálu sourceforge byly vytvořeny pomocí statického slinkování pomocí programů *qmake*, *dh_make* a *dpkg*. V tomto případě bylo nutno mít na počítači statickou verzi knihoven Qt, které se slinkovaly do výsledného binárního souboru. Ten byl potom použit při sestavení balíčku.

5.2 Tvorba instalátoru pro Microsoft Windows

Spustitelný soubor hry vznikl kompilací v operačním systému Windows 7, 64-bit. verze. Verzi 32-bit. nemám v tuto chvíli k dispozici. Instalační balíček byl vytvořen programem *InnoSetup* za pomoci nadstavby *ISTool*.

V instalačním balíčku jsou, spolu se spustitelným souborem také následující *dll* soubory, potřebné pro chod aplikace:

- `icudt52.dll`
- `icudt52.dll`
- `icuin52.dll`
- `icuuc52.dll`
- `libgcc_s_dw2-1.dll`
- `libstdc++-6.dll`
- `libwinpthread-1.dll`
- `Qt5Cored.dll`
- `Qt5Guid.dll`
- `Qt5Multimediad.dll`
- `Qt5Networkd.dll`
- `Qt5Widgetsd.dll`
- `qwindowsd.dll`

Bez nich se nepodařilo aplikaci vůbec spustit i přes to, že v počítači, kde překlad probíhal, je nainstalována nejnovější verze Qt. Po nainstalování zabere aplikace přibližně 450 MB. Nastavením příznaku komprese souborového systému NTFS na největších souborech se ale podařilo nároky na volné místo snížit přibližně na 250 MB. To je ovšem, na takto jednoduchou hru, poměrně hodně.

5.3 Zdrojové kódy

Zdrojové kódy aplikace jsou k dispozici na portálu sourceforge a na přiloženém CD. Pomocí programů *CMake* a *qmake* lze vygenerovat Makefile, kterým se posléze aplikace přeloží. Je také možno využít příkaz `make install`.

Kapitola 6

Závěr

Cílem práce bylo vytvořit funkční port počítačové hry Digger v knihovně Qt a doplnit ho o další smysluplná rozšíření. Většina kódu hry byla napsána od začátku. Převzata byla pouze inteligence monster a spritesheet. Oboje bylo pro potřeby projektu mírně poupraveno. Implementována byla hra jednoho hráče, ve hře se nacházejí drahokamy, monstra a pytle se zlatem. Prostředí herní mapy se mění v závislosti na pohybu hráče (kopání tunelů).

Z rozšíření byl implementován scrolling herního prostředí, který je aktivní ve chvíli, kdy velikost mapy přesahuje velikost herního okna. Také je zde možnost vygenerovat celou mapu procedurálně.

V případě pokračování projektu by bylo vhodné implementovat swapování herní mapy v případě, že přesáhne určitou velikost. Nyní je celá uložena v paměti počítače a v případě, že obsahuje několik desítek tisíc dlaždic alokuje aplikace velké množství paměti. Další vhodná rozšíření jsou například kooperativní lokální či síťová hra více hráčů a další herní módy.

Aplikace je umístěna na portálu sourceforge[1] spolu se zdrojovými soubory. Jsou zde k dispozici *.deb* balíčky pro linuxové distribuce založené na distribuci debian. Byly testovány na systémech Ubuntu 14.04 a Linux Mint 16. Pro ostatní distribuce jsou připraveny konfigurační soubory pro programy *CMake* a *qmake*, kterými je možno vytvořit Makefile. S pomocí programu *CMake* a *CPack* je možné vytvořit *.deb* nebo *.rpm* balíček.

[1] https://sourceforge.net/projects/digdigtreasure/

Literatura

[1] Graphics View Framework [online]. http://qt-project.org/doc/qt-5/graphicsview.html, 2013 [cit. 2014-04-24].

[2] Qt based games [online]. https://qt-project.org/wiki/Qt_Based_Games, 2013 [cit. 2014-04-24].

[3] Data Storage: Saving and Loading Data [online]. http://qt-project.org/doc/qt-5/topics-data-storage.html, 2013 [cit. 2014-04-25].

[4] The Event System [online]. http://qt-project.org/doc/qt-5/eventsandfilters.html, 2013 [cit. 2014-04-25].

[5] Qt Multimedia [online]. http://qt-project.org/doc/qt-5/qtmultimedia-index.html, 2013 [cit. 2014-04-25].

[6] The Qt Resource System [online]. http://qt-project.org/doc/qt-5/resources.html, 2013 [cit. 2014-04-25].

[7] QTime Class [online]. http://qt-project.org/doc/qt-5/QTime.html, 2013 [cit. 2014-04-25].

[8] QTimer Class [online]. http://qt-project.org/doc/qt-5/QTimer.html, 2013 [cit. 2014-04-25].

[9] Drunkard Walk [online]. http://pcg.wikidot.com/pcg-algorithm:drunkard-walk, 2013 [cit. 2014-04-29].

[10] CMake:CPackPackageGenerators [online]. http://www.cmake.org/Wiki/CMake:CPackPackageGenerators, 2013 [cit. 2014-05-17].

[11] Creat Studios, I.: Digger HD [online]. http://www.creatstudios.com/games/digger-hd.php, 2014-04-21 [cit. 2014-04-21].

[12] DeLoura, M. A.: *Game Programming Gems*. 10 Downer Avenue, Higham, Massachusetts 02043: Jenifer Niles, Charles River Media, 2000, iSBN 1-58450-049-2.

[13] Foley, J. D.; van Dam, A.; Feiner, S. K.; aj.: *Computer Graphics: Principles And Practice*. The System Programming Series, Addison-Wessley, druhé vydání, 2003, iSBN 0201848406.

[14] Jenner, A.: Digger - Back and Digitally Remastered [online]. http://www.digger.org/, 2012-12-29 [cit. 2014-04-21].

[15] Jenner, A.: Download [online]. http://www.digger.org/download.html, 2012-12-29 [cit. 2014-04-21].

[16] Parmar, K.: Sprite animation from sprite sheet in Qt [online]. http://kunalmaemo.blogspot.cz/2010/07/sprite-animation-from-sprite-sheet-in.html, 2010 [cit. 2014-05-12].

[17] Wikipedia: Digger (video game) — Wikipedia, The Free Encyclopedia [online]. http://en.wikipedia.org/w/index.php?title=Digger_(video_game)&oldid=584500509, 2013 [cit. 2014-04-21].

Příloha A

Obsah CD

- `cutedigger.tar.gz` – archiv se zdrojovými soubory
- `xpresm00.pdf` – text bakalářské práce
- `xpresm00.zip` – zdrojové soubory textu bakalářské práce
- `cutedigger_1.0-1_amd64.deb` – instalační *deb* balíček pro 64-bit. systémy
- `cutedigger_1.0-1_i386.deb` – instalační *deb* balíček pro 32-bit. systémy
- `cutedigger-setup.exe` – instalátor pro MS Windows 7, 64-bit.
- `cutedigger.mp4` – demonstrační video.

Příloha B

Manuál

B.1 Překlad

- `qmake` nebo `cmake`
- `make`
- `make install`

B.2 Ovládání aplikace

- Tlačítka `W, S, A, D` – pohyb.
- Mezerník – výstřel.
- Esc – konec hry.

B.3 Uložená data a nastavení

Aplikace ukládá herní data a nastavení do adresáře `~/.cutedigger`. Při odinstalování pomocí balíčkovacího systému nebo windows uninstaller není smazán.