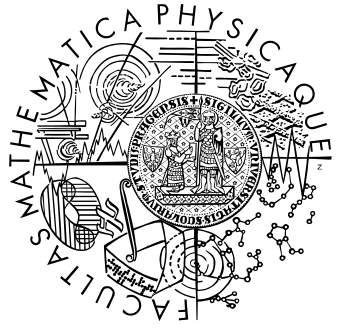
Univerzita Karlova v Praze

Matematicko-fyzikální fakulta

BAKALÁŘSKÁ PRÁCE



Karel Fišer

Bombič 2

Katedra aplikované matematiky

Vedoucí bakalářské práce: RNDr. Bernard Lidický

Studijní program: Informatika, programování (IP)

2010

Na tomto místě bych chtěl poděkovat všem, kteří se podíleli na testování hry, především mé přítelkyni Michale Brabcové, která se zasloužila v roli primárního laického testera a Bc. Františku Princovi, jenž mi navíc pomáhal s portem pro Windows. Děkuji svému vedoucímu RNDr. Lidickému za skvělé rady a postřehy a také za nápad vytvořit hru jako je Bombič. Panu Ing. Zdeňku Böswartovi, který navrhl grafiku pro první verzi Bombiče a dal svolení k její recyklaci.

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

V Praze, dne Karel Fišer

Obsah

Název práce: Bombič 2

Autor: Karel Fišer

Katedra: Katedra aplikované matematiky

Vedoucí bakalářské práce: RNDr. Bernard Lidický

E-mail vedoucího: bernard@kam.mff.cuni.cz

Abstrakt: Cílem práce bylo analyzovat nedostatky první verze hry Bombič a navrhnout hru a editor map odstraňující tyto nedostatky. Návrh poté realizovat, vytvořit programátorskou a uživatelskou dokumentaci. Výsledkem práce je programátorské dílo, které sestává ze dvou programů. V textu této práce jsou popsané zjištěné nedostatky první verze hry, koncepci uložení dat a architekturu obou programů. V kapitole 6. se nachází uživatelská dokumentace. Obsahuje popis ovládání hry i editoru map.

Klíčová slova: Bombič, plošinová hra, arkáda, souboj

Title: Bombic 2

Author: Karel Fišer

Department: Department of Applied Mathematics

Supervisor: RNDr. Bernard Lidický

Supervisor's E-mail address: bernard@kam.mff.cuni.cz

Abstract: The aim was to analyze the shortcomings of the first version of the game Bombic and to desing game and map editor removes these deficiencies. This concept then implement and create a programming and user documentation. As a result of the work originated programming piece, which consists of two programs. In the chapters of this work I describe in details the shortcomings of the first version, design of data storage and architecture of both programs. In chapter 6. I bring the user documentation. Read here how to install the programs, how to set up and play the game. I'm describing how to work with the map editor and how to create a new map.

Keywords: Bombic, side-scrolling, arcade game, deathmatch

1. Úvod

Dyna Blaster [8] je legendární počítačová hra. Hráč prochází koly, pokládáním bomb zabíjí rafinované protivníky, sbírá bonusy, aby zachránil unesenou princeznu. Kromě hlavního příběhu lze hrát i souboj na jednom PC pro dva až čtyři hráče (na klávesnici mohou hrát pouze dva, ostatní na nějakém herním zařízení typu joystick či gamepad). Hra se dočkala mnoha přepracování, z nichž bych uvedl Atomic Bomberman [2], X-Blast [16] a Bombič [3].

Bombič je počítačová akční real-time hra plná příšer, bomb a bonusů, doprovázená zajímavým příběhem. Bombič se také jmenuje postava hráče, která v *kooperativním* módu prochází koly hry a bojuje bombami proti rozličným příšerám. V módu *souboj*[1] potom několik Bombičů bojuje proti sobě na život a na smrt.

Prostředí těchto kooperativních kol a soubojů je dáno mapou, ve které jsou umístěny objekty, které mezi sebou interagují. Pro tvorbu a editaci těchto map jsem vyvinul uživatelsky přívětivý editor map, ve kterém lze jednoduše mapu připravit.

RNDr. Bernard Lidický ve spolupráci s Ing. Zdeňkem Böswartem tuto hru navrhl a také implementoval první verzi[2], která se ujala v mezinárodním měřítku a v Polsku ji byl dokonce vytvořen fanklub [10].

Ačkoli byla hra Bombič 1 zdařilá, měla několik nedostatků, které bylo cílem odstranit. Protože některé nedostatky vycházely již z původního návrhu, celá hra byla navržena a naimplementována od začátku. Vzhledem k tomu, že grafické zpracování a příběh hry byly velice povedené, se souhlasem vedoucího práce RNDr. Bernarda Lidického a autora grafiky Ing. Zdeňka Böswarta jsem grafiku (obrázky) a příběh (text, rozložení kol) převzal bez větších zásahů.

1 překlad anglického *deathmatch* - souboj na smrt

2 První verzí nebo také Bombič 1 budu nazývat původní hru Bombič jak je prezentována v [4]

2. Analýza nedostatků

Po dlouhém ale zábavném testování Bombiče 1 jsem již měl představu, jaké vlastnosti je potřeba změnit a jaké naopak zachovat. Cílem bylo naprogramovat hru tvárnější, více provázat jednotlivá kola a umožnit uživateli cítit se více svobodně.

2.1 Formát dat

První nedostatek jsem objevil, ani jsem nemusel hru spustit. Je totiž uveden na seznamu, co je třeba opravit [4]. V první verzi byla data uložena ve špatně specifikovaném binárním formátu, což dělá problémy s přenositelností a neumožňuje jednoduše data modifikovat.

Data uložená na disku jsou logicky rozdělena do dvou skupin. Do první jsou řazeny obrázky, které mají nějaký svůj obecně uznávaný formát a do programu se načítají automaticky pomocí externí knihovny. Do druhé skupiny patří definice všech map, objektů v mapách a dalších prostředků navržených výhradně pro potřeby hry Bombič.

Pro druhou skupinu jsem tedy vybíral, jak budou data uložena na disku, abych napravil výše zmíněné nedostatky. Z hlediska přenositelnosti by bylo vhodné použít nějaký textový formát. Nestrukturovaný text ale nepřináší jednoduchou strojovou čitelnost a tím znepříjemňuje modifikování dat. Pro reprezentaci takových dat jsem tedy vybral obecný jazyk XML[3], který má navíc mnoho dobrých vlastností. Z programátorského hlediska je výhodný díky hotovým knihovnám na manipulaci s XML. Modifikující uživatel snad ocení čitelnost takových dat. Nezanedbatelná nevýhoda XML je určitě režie, která je vynaložena na onu čitelnost. Data hry ale nejsou nijak obsáhlá, proto jsem tuto nevýhodu přijal.

2.2 Provázanost kol

V první verzi hry nebyla kola nijak provázána. Abstraktně byla spojována příběhem, který se snažila prezentovat. Také obtížnost kol se postupně zvyšovala. Přišlo mi frustrující, že i proti nejobtížnějším protivníkům stál Bombič na začátku kola úplně bezbranný a hráč trávil hodně času sbíráním bonusů, aby alespoň trochu vylepšil svůj arzenál. Bonusů potom muselo být v kole tolik, že to hráče po chvíli demotivovalo bonusy sbírat.

Rozdělil jsem proto nově bonusy logicky na dvě skupiny. První skupina bonusů (plamen, bomba, život, rychlost) se mezi koly přenáší. Čím víc hráč takových bonusů posbírá, tím má větší herní sílu. Těchto bonusů je v kolech obecně méně, protože si je uživatel přenáší do dalšího kola.

Ostatní bonusy se do dalšího kola nepřenášejí, může jich být proto v kole více nebo jen určité druhy.

3 XML - Extensible Markup Language

To zachovává jedinečnost kola, co se týče použitých bonusů, a zároveň zvyšuje vybavenost hráče se zvyšující se obtížností kola. Navíc je uživatel motivován přenosné bonusy sbírat, aby mu v dalších kolech nechyběly.

2.3 Ukládání a načítání rozehrané hry

Aby uživatel nemusel hru hrát „na jeden zátah“, je dobré určit mechanismus, kterým se uživatel po spuštění programu dostane do kola, ve kterém naposledy skončil.

Řešení pomocí kódů kola bylo v první verzi dostačující a nebylo vůbec špatné. V druhé verzi je ale třeba si u rozehrané hry zapamatovat jak je hráč vybaven, protože má určité vlastnosti dané tím, kolik posbíral přenositelných bonusů.

Je třeba v uložené hře zachytit aktuální stav. Kdybychom umožnili hru ukládat v libovolném okamžiku, nevyhnuli bychom se ukládání celé mapy, což je pro náš účel nevhodné. Postačí, když umožníme uložit hru ve stavu posledního úspěšně dokončeného kola. Jedno kolo musí hráč tedy zvládnout najednou. Pokud je hráč zabit, kolo je restartováno a hráč vybaven tak, jak přišel z posledního dokončeného kola. Při načtení takové hry je také hráč vybaven tak, jak přišel z minulého kola, je zde tedy konzistence mezi ukládáním, načítáním a hraním.

Jako formát dat opět poslouží XML, hra je uložena jako soubor na disk a je v ní zakódováno v jakém kole hráči skončili, kolik jich případně hrálo a jaké vlastnosti si do tohoto kola každý přinesl. Po načtení rozehrané hry se hráči objevují v daném kole tak, jak do něj přišli před uložením.

2.4 Tvárnost map

Představte si, že jedno kolo hrajete celý den, vždy vás zabije stejná příšera, opět se narodíte, sbíráte bonusy, zabíjíte příšery, které jsou na stejných pozicích a opět vás tatáž příšera na tomtéž místě zabije.

Tvárností zde chci naznačit, že stejné kolo se stejnou mapou se stejnými bonusy a příšerami nemusí být vždy úplně stejné. Do Bombiče 2 je zakomponována náhoda při vytváření mapy. V definici mapy lze přesně určit, který objekt bude na které pozici (kromě bonusů samozřejmě, ty jsou skryté, náhodně rozmístěné pod bednami a objeví se až když bedna bouchne). Navíc lze u dynamických objektů (beden a příšer) specifikovat, kolik se jich má rozmístit náhodně. V takovém kole, pokaždé když vás zabijí se narodíte do nové, trochu jiné mapy. Navíc se může stát, že projít mapou s nějakým rozložením objektů bude jednodušší než s jiným, což pomůže hráči, který mnohokrát neuspěje.

2.5 Přesnější výběr bonusů

Jak v kooperativním módu, tak v souboji by bylo dobré umožnit přesně definovat, který bonus se v mapě vyskytuje kolikrát.

Kooperativní mód má bonusy zakódované přímo v definici mapy. Ke každému druhu bonusu, který se má v mapě objevit je dán počet, kolik takových bonusů se má v mapě náhodně vygenerovat.

Bonusy pro souboj si určí sám uživatel pomocí podmenu *Bonusy*, ve kterém opět specifikuje, kolik kterých bonusů má v mapě být.

2.6 Nastavení

Hodně uživatelů si stěžovalo, že si v první verzi nemohou zvolit herní klávesy. V komentářích na polském webu [5], kde byl Bombič 1 vystaven jsem se dokonce dočetl (pokud jsem polštině porozuměl správně), že nějaký uživatel neměl na klávesnici pravý *Ctrl* (čímž se pokládají bomby), takže si bohužel nemohl hru vůbec zahrát. V druhé verzi je proto implementována podpora výběru kláves.

2.7 Rozdělení obrazovky

Při hře více uživatelů na jednom stroji nastává u větších map problém, že se hráči do společného pohledu na mapu nevejdou.

Implementoval jsem rozdělení okna do více pohledů na mapu. Automaticky, když se nějaký hráč do pohledu nevejde, se vytvoří pro každého hráče samostatný pohled. Když se hráči opět sejdou, pohledy se spojí v jediný.

Při častém rozcházení a scházení hráčů se pohledy spojují a rozdělují, což může uživateli vadit. Proto lze v menu *nastavení* vybrat, jestli má být obrazovka rozdělena pořád, nebo jen když je to nutné.

2.8 Ruční odpalování

Ruční odpalování je časově omezený bonus, který dává hráči možnost odpálit položenou bombu kdy chce. V první verzi měl tento bonus často sebevražedné následky, protože stlačením pokládací klávesy hráč bombu položil a uvolněním odpálil. Když tedy hráč zapomněl, nebo si neuvědomil, že bonus právě má, okamžitě sám sebe zabil.

Ke čtyřem pohybovým klávesám a jedné pokládací přidávám ještě odpalovací klávesu, kterou hráč odpálí první položenou bombu, pokud má bonus ruční odpalování. Přidáním odpalovací klávesy jsem vyřešil problém sebevražedného ručního odpalování a navíc jsem přidal možnost položit více bomb a odpálit je postupně.

3. Datové soubory

Jak jsem se již zmínil v podkapitole 2.1, datové jsou soubory rozděleny do dvou skupin – grafické (binární) a XML soubory. V XML souborech jsou herní (programová) nastavení, jazykové mutace, definice map, pozadí a objektů.

Data jsou rozdělena do souborů, které jsou načítány, až když jsou opravdu potřeba. Datové (binární) soubory – obrázky – jsou referencovány z XML souborů pomocí jména souboru (bez cesty). XML soubory jsou referencovány pomocí názvu (druhu) objektu, který představují. Takový soubor má stejný název jako reprezentovaný objekt a jednotnou příponu .xml. Aby nemusely být všechny soubory v jednom adresáři, jsou soubory podle jména vyhledávány v několika adresářových strukturách. První prohledávaný podstrom začíná v podadresáři *.bombic* domovského adresáře uživatele, což umožňuje uživatelskou modifikaci hry, která je instalována v systému globálně a uživatel nemůže její soubory měnit. Další podstrom je dán adresářem, do kterého je hra instalována. Je tedy možné uchovávat datové soubory ve stromové struktuře logicky separované.

3.1 Grafické prvky

Grafickým prvkem si označme entitu, se kterou je v programu pracováno při určité míře abstrakce jako s nedělitelným článkem a uživateli je znázorňována jednoduchým obrázkem či obrázky, které mohou připomínat pohyb nebo například pohled na prvek z více stran. Grafickým prvkem může být např. objekt mapy nebo panel s vlastnostmi hráče.

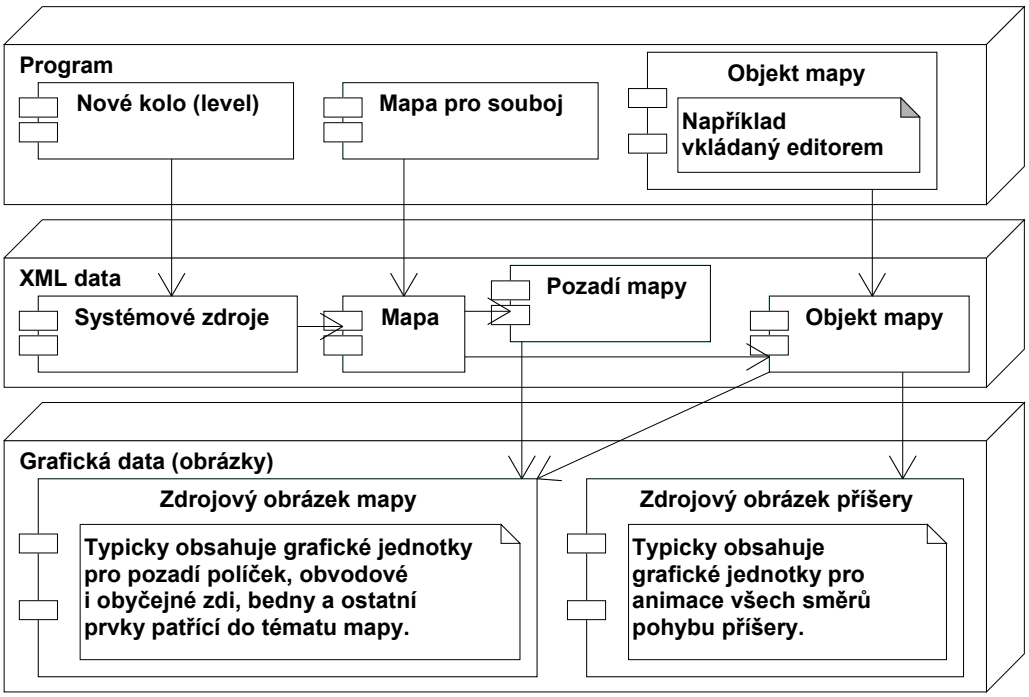
Grafická jednotka je potom jeden nedělitelný obrázek, který se na obrazovku nakreslí vždy celý a najednou. Grafický prvek rozhoduje, jaká grafická jednotka se v daném okamžiku na obrazovku nakreslí.

Zdrojový obrázek je soubor na disku obsahující grafická data. Ve zdrojovém obrázku může být najednou umístěna grafika pro více prvků. Je to hlavně kvůli tomu, že objekty se načítají při konstrukci mapy najednou. Poslední načtený zdrojový obrázek se uchovává a nemusí se tedy tolikrát hledat soubor, načítat z disku atd.

Veškeré grafické prvky jsou definovány pomocí XML souborů. U XML elementu (označme si jej E) je definován (většinou v atributu *src*) název souboru se zdrojovým obrázkem, ze kterého se potom grafické jednotky vyřezávají. Odkud má být grafická jednotka vyříznuta se definuje v podelementech elementu E. Takový podelement má vetšinou atributy *x,y,* což udává souřadnice ve zdrojovém obrázku. Velikost grafické jednotky je dána podstatou načítaného objektu nebo je specifikována atributy *width* a *height*.

Jak znázorňuje obrázek 3.1, referencovány jsou z programu přímo jen XML soubory (a to jen několik). Z XML definic se získají informace potřebné k načtení zobrazovaných obrázků. V prvním případu program načítá mapu nového kola. Z herního XML souboru *levels.xml* zjistí, jaká mapa se má nahrát k danému kolu, načte XML soubor mapy, v něm zjistí referenci na pozadí mapy a všechny objekty. V XML souboru definice pozadí (respektive objektu mapy) je již dáno, ze kterého

zdrojového obrázku se mají které grafické jednotky načíst. Před načítáním mapy pro souboj uživatel přímo vybere soubor definice mapy z disku, ze kterého se má mapa nahrát. Z definice mapy se pak opět zjistí reference na pozadí a objekty a postup je stejný jako v předešlém případě. V editoru map se obdobně načítá například i objekt mapy, který je buď referencován z načítací sady (specifikované v konfiguračním XML souboru pro editor), nebo přímo vybrán uživatelem z disku.



Obrázek 3.1: Schéma uložení grafických prvků

3.2 Jazykové mutace

Editor map je připraven na lokalizaci použitím mechanismu, který je více obvyklý a zabudovaný v knihovně Qt. Samotný překlad jsem zatím ale neprováděl, proto se o jazykových mutacích editoru dále zmiňovat nebudu.

Hra Bombič má několik (zatím anglickou a českou) jazykových mutací. Samotný překlad textů jsem převzal z první verze, mechanismus jsem ovšem navrhl nový. Oproti standardním mechanismům, kde je základní mutace součástí zdrojových kódů (a tedy i binárky) programu jsem zvolil cestu *klíčových hesel*, pomocí kterých se v programu označuje, co se má vypsat. Ona klíčová hesla potom slouží pro vyhledání překladu. Z druhého pohledu jsou texty uložené v externím strukturovaném XML souboru, je jasné jaký text má jakou funkci a pro překlad stačí soubor s texty okopírovat, přeložit a nový překlad registrovat v konfiguračním souboru, aby byl dostupný v menu.

K tomuto řešení jsem se uchýlil hlavně kvůli tomu, že texty, které se zobrazují před kolem (případně za ním) nemohou být součástí zdrojových kódů, protože definice jaká kola jsou vlastně dostupná je také v XML souboru. Takto jsem vyřešil tuto zvláštnost, protože klíčové heslo vedoucí k vyhledání překladu může být

součástí definice kol hry. Aby byly texty přímo součástí definice kol hry jsem zavrhl, protože jsem chtěl lokalizaci pro každý jazyk umístit do jednoho zvláštního souboru.

3.3 Mapy a objekty

Definice mapy je XML soubor, ve kterém je popsáno, jaký má mapa název, rozměry, styl pozadí a jaké jsou v ní umístěny, popřípadě náhodně generovány, objekty. Pozadí mapy a objekty jsou v definici referencovány pouze názvy, které po přidání přípony .xml dávají jméno souboru, ve kterém je objekt či pozadí definováno. Tento soubor, jak je již popsáno v úvodu této kapitoly 3. je pak vyhledán a definice zpracována.

Kořenový element *map* obsahuje povinné atributy *name* (název mapy), *width, height* (rozměry mapy v políčkách) a *background* (název pozadí). Definice vyžaduje jediný povinný podelement *players*, který definuje umístění hráčů při kooperativním módu s povinnými čtyřmi podelementy *players0-3*, které definují umístění hráčů při souboji.

Ostatní podelementy jsou nepovinné a definují rozličné objekty. Obecně je to vždy element s názvem typu objektu v plurálu (např. w*alls, creatures*) s povinným atributem *name* (název objektu) a nepovinným *random_generated* (počet takových objektů, které se mají náhodně vygenerovat). Náhodně generovat lze ovšem jen příšery, bonusy a bedny zabírající pouze jedno políčko. Pro každé pevné umístění objektu do mapy dále takový element obsahuje jeden podelement s názvem typu objektu v singuláru (tedy *wall, creature)*, který v atributech *x, y* definuje políčko, na které se má objekt vložit (u rozlehlejších objektů se na takové políčko vloží horní levý roh objektu). Pevně do mapy nelze vkládat bonusy.

Definice pozadí mapy je XML soubor s kořenovým elementem *background*, který obsahuje povinné atributy *name* (název pozadí) a *src* (soubor se zdrojovým obrázkem). Definice pozadí specifikuje obvodové zdi (podelementy *topleft, topright, bottomleft, bottomright* specifikující zdi umístěné v rozích mapy a *top, bottom, left, right* specifikující zdi opakující se po stranách mapy). Každý element obvodové zdi má povinně atributy *x, y* (souřadnice grafické jednotky ve zdrojovém obrázku) a nepovinně atributy *width, height* (rozměry v políčkách, které objekt zabírá v mapě, defaultně 1) a parametr *toplapping* (počet políček, které objekt přesahuje nahoru – užitečné u efektu vysokých objektů, defaultně 0). Dále může mít element obvodové zdi podelement *animation* s atributem *rate* (počet obrázků animace za sekundu) a podelementy *animation_item,* které definují pomocí atributů *x, y* další grafické jednotky použité v animaci objektu. Kromě obvodových zdí ještě definice pozadí specifikuje obrázek na pozadí políček, a to elementem *clean_bg* obyčejné pozadí a elementem *burned_bg* spálené pozadí. Tyto elementy mají pouze atributy *x, y* (souřadnice grafické jednotky ve zdrojovém obrázku).

V definicích všech objektů se objevují podobné konstrukce. Kořenový element odpovídá singuláru typu objektu, obsahuje atributy specifikující jméno objektu, zdrojový obrázek, popřípadě doplňující parametry objektu (rychlost, inteligence příšery). Podelementy specifikují grafické jednotky objektu a často mohou obsahovat podelement *animation,* který tvoří z jednoduchého obrázku animaci.

Jako příklad je zde mapa použitá v prvním kole hry, uložená v souboru *map_forest_1.xml*.

```
<map name="map_forest_1" height="12" width="16" background="bg_forest">
	<players x="1" y="2">
		<player0 x="1" y="2" />	<player1 x="14" y="10" />
		<player2 x="1" y="10" />	<player3 x="14" y="2" />
	</players>
	<walls name="wall_stone">
		<wall x="2" y="3" />	<wall x="2" y="5" />
		<wall x="2" y="7" />	<wall x="2" y="9" />
		<wall x="4" y="3" />	<wall x="4" y="5" />
		<wall x="4" y="7" />	<wall x="4" y="9" />
		<wall x="6" y="3" />	<wall x="6" y="5" />
		<wall x="6" y="7" />	<wall x="6" y="9" />
		<wall x="8" y="5" />	<wall x="8" y="7" />
		<wall x="10" y="3" />	<wall x="10" y="9" />
		<wall x="11" y="5" />	<wall x="11" y="7" />
		<wall x="12" y="3" />	<wall x="12" y="9" />
		<wall x="13" y="5" />	<wall x="13" y="8" />
	</walls>
	<boxes name="box_barrel" random_generated="38">
		<box x="3" y="2" />	<box x="3" y="3" />
		<box x="3" y="4" />	<box x="1" y="4" />
		<box x="2" y="4" />
	</boxes>
	<bonuses name="bonus_flame" random_generated="1" />
	<bonuses name="bonus_speed" random_generated="1" />
	<creatures name="creature_rat" random_generated="4" />
</map>
```

Obrázek 3.2: Pohled na mapu map_forest_1

4. Implementace hry

4.1 Jazyk a knihovny

Samotná hra je programována v jazyce C++ s použitím STL[4]. Jako manuál k C++ a STL jsem používal web cpp reference [6]. Pro práci s grafikou a přístup k periferiím jsem použil SDL[5] na popud knihy o programování her pro Linux [9]. Menu hry je programováno pomocí knihovny AGAR [1], která je navržena pro jednoduchou integraci a následuje myšlenku stavění GUI[6] okolo aplikace, nikoli obráceně. K načítání dat je použita knihovna TinyXML [14], což je jednoduchý XML parser, který může být do programů jednoduše integrován. Knihovny AGAR a TinyXML jsem si vybral zejména proto, že jsou jednoduché a mohou být distribuovány společně se hrou v podobě zdrojových kódů.

4.2 Propojení částí programu

Hra se skládá ze dvou částí, mezi kterými uživatel volně přechází. Začíná v menu, které uživateli umožňuje provést potřebné nastavení, vybrat si, co chce hrát. Po spuštění hry se dostává do druhé části programu – hraní hry.

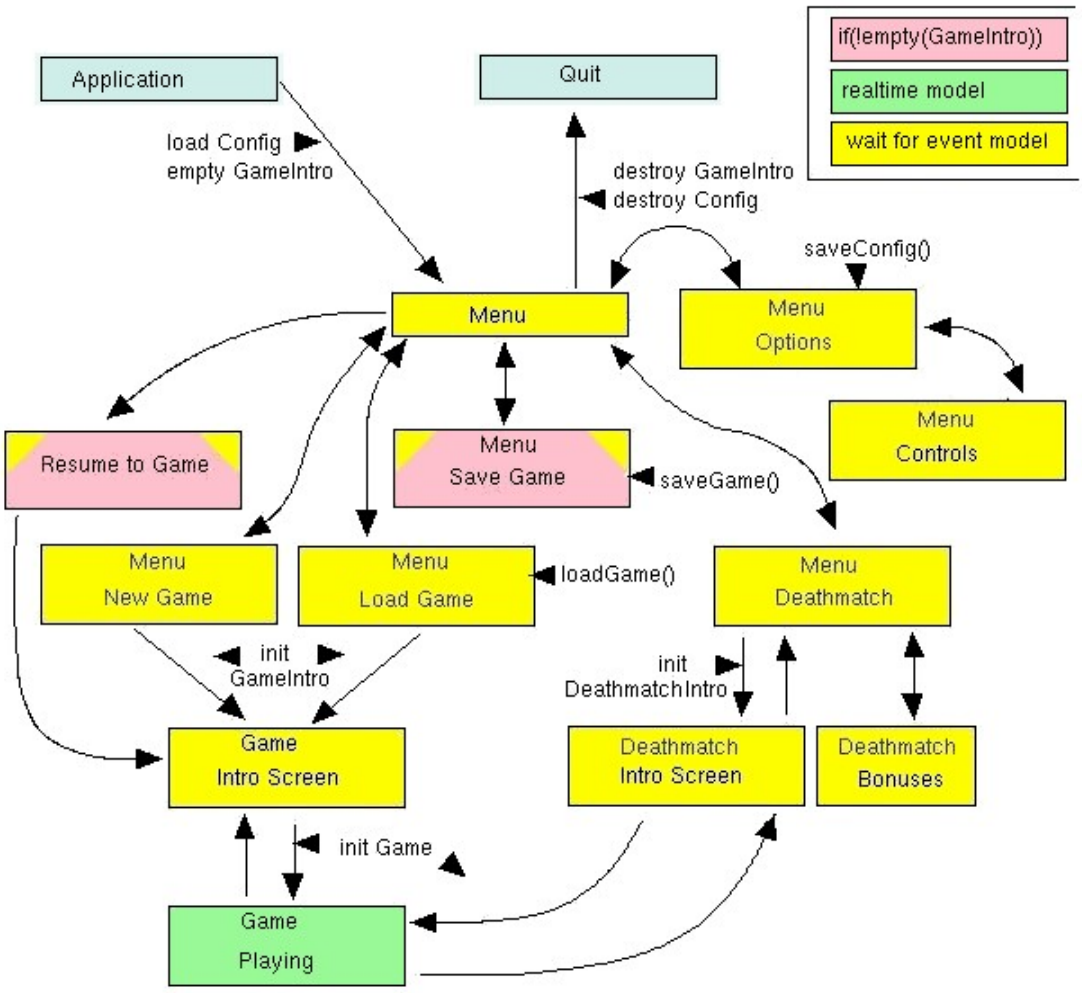
Samotná hra může být spuštěna v kooperativním módu, nebo jako souboj. Hraní je pak velice podobné, proto je samotná hra implementována v jediné třídě *Game*, která tyto módy rozlišuje. Jedna instance *Game* reprezentuje jednu konkrétní hru od vytvoření po úspěšné nebo neúspěšné ukončení. Stará se o její spuštění, správné časování a ukončení. O typu ukončení poskytuje informace. O to, jaká se bude aktuálně hrát rozhoduje intro, které jednotlivé hry spojuje. Zde už se přístup módů markantně liší, proto máme také dvě rozdílné třídy *GameIntro* a *DeathmatchIntro*. Instance od *GameIntro* je vytvořena hned při startu programu a veřejně používána, protože s ní potřebuje pracovat menu na různých místech v kódu. Instanci *DeathmatchIntro* si vytváří lokálně pouze menu, ze kterého se hra v módu souboj spouští, vždy, když přecházíme do herní části programu.

V obrázku 4.1 je znázorněno, jak jsou mezi sebou provázána jednotlivá menu a z kterých menu jsou nastavována a zobrazována intra.

4 STL – C++ Standard Template Library – standardní C++ šablonovaná knihovna

5 SDL - Simple DirectMedia Layer – jednoduchá ale silná multiplatformní knihovna pro nízkoúrovňový přístup k audiu, klávesnici, myši a video paměti [13].

6 GUI – Graphical User Interface – grafické uživatelské rozhraní



Obrázek 4.1: Přehled propojení částí hry

4.3 Menu

Menu je systém oken, ze kterého je vždy vidět jen horní. Tato okna jsou držena v zásobníku. Na spodku zásobníku je hlavní menu, ze kterého se můžeme dostat do podmenu, vrátit se k rozehrané hře v kooperativním módu nebo ukončit program. Jednotlivá podmenu mají každé svou jedinečnou úlohu, proto jsou implementována jako jednotliví potomci třídy *MenuBase*. Společný předek jim umožňuje používat metody pro vytváření položek menu a navenek mají všechny menu jednotné rozhraní pro řízení. Jednotlivá menu si sama vytváří položky a provádějí akce podle uživatelovy aktivity.

V obrázku 4.1 je mimo jiné znázorněno, jak jsou jednotlivá menu provázána a jakou mají stěžejní funkci. Růžově označená menu se zobrazují pouze, když je právě rozehrána nějaká kooperativní hra.

4.4 Hra a její základ

Jedna instance *Game* představuje jednu nedělitelnou hru od narození po úmrtí. *Game* podporuje oba módy hraní, rozdíl je v podstatě jen v podmínce pro ukončení hry.

1. Kooperativní mód končí negativně, pokud jsou všichni hráči mrtví. Pozitivně končí, zůstane-li naživu alespoň jeden hráč ale mrtvé jsou všechny příšery.
2. Souboj končí negativně, jsou-li všichni hráči mrtví. Úspěšně (body se započítávají), pokud zůstane naživu právě jeden hráč.
3. Hru lze explicitně přerušit (klávesa *Esc*). V takovém případě se úspěšnost ukončení posuzuje opět podle 1. nebo 2.

Game drží aktuální rozmístění objektů ve mapě, objekty vykresluje a hýbe jimi. Vede řízení programu v čase, kdy uživatel opravdu hraje. Výhradně zde se používá aktivní přístup, na rozdíl od pasivního čekání na události v menu a intru.

V obou módech se často opakují hry s tou samou konfigurací (mapa, počet hráčů, bonusy atd.). V kooperativním módu je to opakované neúspěšné hraní jednoho kola (levelu), v souboji potom utkání na více vítězství. Aby se pokaždé nemusela načítat mapa se všemi objekty z disku (což je časově náročné), předpřipraví se startovní struktura bokem a při opakovaném vytváření hry se použije pouze ta. Připravit si rovnou hru ve startovním stavu a pak ji pouze okopírovat nelze, protože je třeba pokaždé náhodně umístit generované objekty. Ona předpřipravená struktura je instance třídy *GameBase*, která již má umístěné pevně pozicované objekty (opravdu v objektové formě) a seznam objektů (opět v objektové formě), které jsou potřeba pokaždé náhodně vygenerovat. Zároveň drží informaci o tom, kam generované objekty umístit nelze.

Při generování konkrétní hry se objekty ze základu okopírují, v základu tedy zůstávají v původním stavu. To dává záruku, že například hráči po opakovaném hraní jednoho kola zůstávají vybaveni přenosnými bonusy tak, jak přišli z kola předcházejícího. Při ukládání hry se ukládá aktuální stav hráčů v základu hry.

Zde bych udělal malou odbočku ke grafickým prvkům. Každý objekt je musí mít přístupné, aby je mohl kreslit na obrazovku. Protože je objektů v mapě hodně shodných a ještě jsou uloženy v základu hry, bylo třeba vyvinout mechanismus sdílení grafických jednotek. Konkrétněji, objekt A je vizualizován stejnou animací jako objekt B. Jejich animace mají ale jiný posun, celé animace tedy sdílet nemohou. Implementoval jsem tedy počítání referencí pro elementární grafickou jednotku. Třída *Surface* vlastní jednu instanci *SDL_Surface* (která může zabírat hodně paměti) a počítá, kolik referencí v programu existuje. Když už není tato grafická jednotka z programu referencována, automaticky ji dealokuje.

4.5 Intro do hry či souboje

Intro je část programu, které tvoří rozhraní mezi menu a samotným hraním. Nejdříve je intro nastaveno z menu (pomocí parametrů, které jsou v menu známé – např. výše dosaženého kola, počet hráčů, či nastavení bonusů pro souboj), potom je mu prostřednictvím metody *show_screen()* přenecháno okno. Intro rozhodne, která

mapa se bude hrát a dokud není sekvence kol dohrána (podle kol epizody, nebo podle skóre souboje), nebo uživatel explicitně smyčku nepřeruší, jsou opakovány následující úkony:

1. zobrazení úvodní obrazovky (zobrazuje-li se)
2. připravení základu hry *GameBase,* pokud se nepoužije předcházející
3. vytvoření konkrétní hry ze základu
4. hraní (*GameIntro* přenechá obrazovku instanci *Game* metodou *play()*)
5. zobrazení obrazovky po hře (zobrazuje-li se)
6. rozhodnutí, která mapa se bude hrát dál (jestli vůbec nějaká)

GameIntro je intro ke kooperativní hře. Podle výše kola intro zjistí (ze souboru *levels.xml*), jaký obrázek, text a mapa odpovídá danému kolu. Vždy před hrou je zobrazena motivující obrazovka s obrázkem a textem příběhu. Zatímco uživatel čte (nebo nečte) motivační text, je připraven základ pro hru *GameBase*. Ze základu je pak vygenerována hra, která je po potvrzení uživatelem spuštěna a uživatel hraje. Podle toho, jak hra skončila, intro buď zobrazí úmrtní obrazovku nebo obrazovku s textem po úspěšném kole (je-li nějaká k dispozici). Pokud bylo kolo zahráno neúspěšně, opět ze stejného základu vygeneruje hru. V opačném případě zvýši dosažené kolo a opakuje celou sekvenci od začátku. Z úvodní obrazovky, nebo po hře lze z intra odejít (klávesa *Esc*). V takovém případě se řízení vrátí menu. Protože je v programu jediná instance *GameIntro,* která si navíc pamatuje základ rozehrané hry, je možné se do hry z menu vždy vrátit, dokud nebude rozehrána jiná kooperativní hra.

DeathmatchIntro je intro k souboji. Úvodní obrazovku nemá, před hrou jen vytvoří základ hry, který odpovídá parametrům, které uživatel zvolí v menu (mapa, bonusy, s nebo bez příšer atd.) a vygeneruje hru, které rovnou předá řízení. Po ukončení hry zobrazí skóre hráčů (obrazovka po hře) a ze stejného základu opět vygeneruje další hru. To opakuje tak dlouho, dokud není dosaženo konečné skóre nebo uživatel explicitně neodejde z obrazovky po hře (opět klávesou *Esc*). Vzhledem k tomu, že je instance *DeathmatchIntro* vytvářena z menu vždy nová, nelze se vrátit k rozehranému souboji (k poslednímu skóre), z podstaty intra by to ale šlo. Usoudil jsem, že to ale není nutné, v menu zůstávají poslední nastavené hodnoty, takže lze jednoduše začít stejný souboj od začátku.

4.6 Objekty mapy

Z pohledu definice mapy jsou objekty pevně umístěné nebo generované. Totéž rozlišuje i třída *GameBase*, která definici mapy zpracovává. Třída *Game* se sama o náhodné rozmístění objektů stará, takže při přípravě mapy o generovaných objektech ví, ale již jsou to pro ní obyčejné objekty, které jen rozmisťuje. Dále při hraní již není tato informace potřeba, proto implementace samotného objektu mapy nijak nerozlišuje jestli je objekt generovaný či umístěný.

Další rozlišení, které už implementace rozlišuje, nám dává mobilita objektů.

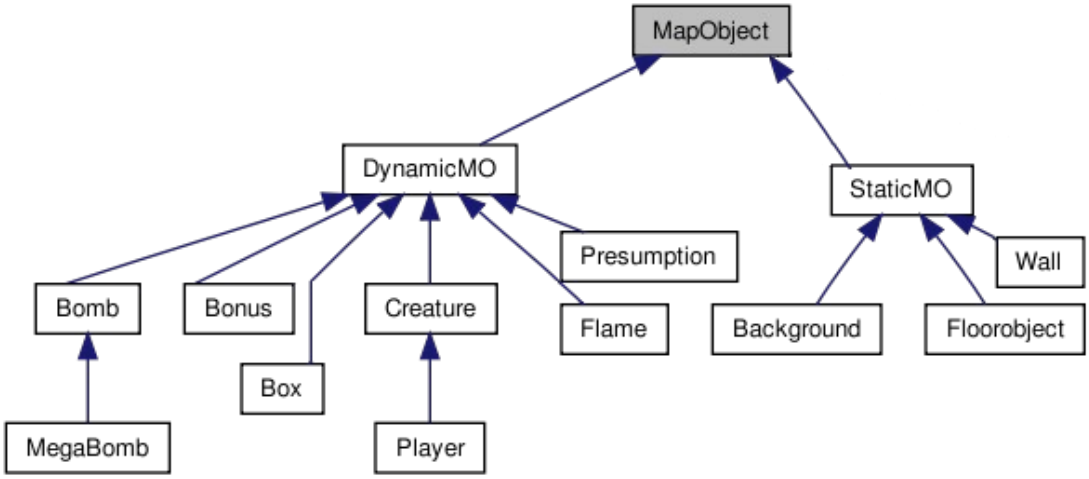
- Statické objekty – objekt je od vytvoření do ukončení hry pořád na stejném políčku mapy, během hry se tedy nehýbe, neobjevuje ani

nemizí. Aby i statické objekty mohly vykazovat známky dynamiky, mohou být animovány nebo se v rámci posouvání animace jinak vnitřně měnit (pozadí políček například změní svůj vzhled když na nich bouchne bedna). Tyto objekty nezatěžují herní smyčku pohybu, snižují velikost seznamu objektů, se kterými se dynamicky pracuje.

- Dynamické objekty – objekt je možno kdykoli do mapy přidat či ho z mapy odstranit, může se pohybovat mezi políčky. Stejně jako statické objekty může být nezávisle na pohybu animován. Pokud objekt není z podstaty statický, měl by být dynamický.

Na základě tohoto rozdělení důležitého pro urychlení implementace herní smyčky jsem vytvořil strukturu dědičnosti C++ tříd, kde jsou jako potomci jednotlivé objekty mapy, které reprezentují objekty rozlišované uživatelem podle typických rysů.

- *MapObject* – abstraktní předek všech objektů mapy. Pamatuje si pouze, na kterém políčku aktuálně leží. Navenek poskytuje rozhraní pro řazení objektů na políčku, animování a vykreslení. Toto rozhraní je ale potřeba v potomcích implementovat.
  - *StaticMO* – abstraktní předek pro statické objekty. Do rozhraní nepřidává žádnou funkcionalitu.
    - *Background* – pozadí políčka mapy
    - *Floorobject* – objekt na zemi (na pozadí mapy)
    - *Wall* – zeď (překážka, která neshoří)
  - *DynamicMO* – abstraktní předek pro dynamické objekty. Do rozhraní přidává metodu *move()* pro pohyb objektu. Pro potomky implementuje metodu *setFieldInMap* pro případné posunutí objektu v mapě na nové políčko.
    - *Box* – bedna (překážka, která shoří)
    - *Bonus* – objekt, který může být skrytý pod bednou, když jej hráč sebere, dostane nějakou z aplikací bonusu (což už není objekt mapy). Viz 4.8.
    - *Flame* – plamen, velmi dynamický objekt, který je typicky vytvářen bombou po výbuchu. Pro většinu dynamických objektů má smrtící účinky.
    - *Presumption* – předpověď plamene. Aby příšery (potažmo uživatel) věděly, kde v blízké době vznikne plamen. Viz 4.9
    - *Bomb* – objekt, který typicky vkládá do mapy hráč a při výbuchu tvoří plamen
      - *MegaBomb* – speciální případ bomby
    - *Creature* – příšera, proti které Bombič bojuje. Viz 4.10
      - *Player* – hráč je postava Bombiče. Viz 4.10



Obrázek 4.2: Hierarchie dědičnosti objektů mapy

4.7 Detekce kolizí

U mnoha her a podobných aplikací je řešení kolizí velký problém. Ve hře Bombič jsou v kolizi objekty, které sdílí políčko mapy. Zjistí-li tedy bedna (třída *Box*), že na jejím políčku je plamen, sama shoří. Toto řešení kolizí je jednoduché a rychlé. Přístup do mapy na určité políčko je v konstantním čase, výběr relevantního objektu lineárně k počtu objektů na jednom políčku (kterých jsou typicky řádově jednotky).

Při implementaci pohybu příšer jsem ale narazil na problém. Příšera nemůže chodit až ke kraji políčka, je-li na dalším políčku nějaký blokující objekt (zeď či bedna). Pokud je tedy příšera za polovinou políčka, musí se kontrolovat i následující políčko. Podobný problém nastává i ve směru kolmém na pohyb příšery při situaci jako na obrázku 4.3. Sousední políčka příšery jsou volná, příšera je v dolní části políčka a na diagonálním políčku je blokující objekt. Příšera se nemůže pohybovat přímo podle červené šipky, protože by došla na cílové políčko v dolní polovině ovlivněné blokujícím objektem. Posun příšery se provede obloukem podle zelené šipky tzv. centrováním (metoda *AI::centerCoordinates()*).

Obrázek 4.3: Situace

Obrázek 4.5: Špatné řešení

Obrázek 4.4: Dobré řešení

4.8 Bonusy

Bonus je objekt mapy (třída *Bonus*), který je na začátku hry vždy skrytý pod bednou (třída *Box*). Když bedna bouchne (shoří), bonus se objeví. Je vizualizován

tak, aby uživatel viděl jaký bonus se v tomto objektu mapy skrývá. Mimo to, že může bonus shořet v plameni, může být také sebrán hráčem. Pokud totiž hráč vstoupí na políčko, na kterém je bonus, bonus se aplikuje na hráče a z mapy zmizí také.

Tím se dostáváme k druhému smyslu bonusu. Aplikace bonusu (třída *BonusApplication*) je předek jednotlivých aplikací na konkrétního hráče. Tato třída může zasahovat do vlastností hráče, do vlastností ostatních hráčů nebo může působit na hru. *BonusApplication* již není objektem mapy.

Jedním z nejzábavnějších bonusů je nemoc, jehož aplikace *BonusIllness* je předek různých škodících bonusů, které modifikováním různých vlastností hráče dělají souboj velmi zábavným a dramatickým. Zejména při použití bonusu, jehož aplikace *BonusOthersIllness* aplikuje nemoc na všechny hráče kromě toho, který bonus sebral.

Přenositelnost bonusu, jak o ní bylo mluveno v 2.2 není vlastnost bonusu, ale hráče, jehož vlastnosti jsou měněny. Například počet bomb, které může hráč najednou položit je vlastnost hráče, kterou aplikace bonusu *BonusBomb* modifikuje tak, že jednorázově zvýší počet bomb. Při přechodu do následujícího kola se tato vlastnost hráče přenese, tedy počet bomb, které může položit se zachovává.

4.9 Předpovídání výbuchu

Aby příšery mohly reagovat na položené, nebo dokonce pohybující se bomby, je do mapy vkládána informace o tom, kde a za jak dlouho se objeví na daném políčku plamen. Kdyby každá příšera, kterou tato informace zajímá měla zjišťovat podle všech aktuálních bomb v mapě kam jejich plamen dosáhne, bylo by to příliš neefektivní.

Každá bomba tedy do doby svého výbuchu sofistikovaným způsobem předpovídá, na kterém políčku a za jak dlouho exploduje. U nepohyblivých bomb je cílové políčko jasné, dobu exploze ale ovlivňují okolní položené bomby. U pohybujících se bomb je náročnější určit, na jakém políčku bouchnou, zejména pohybuje-li se jich více, což je běžné, má-li hráč bonus posílání bomb. U bomb, které jsou odpalovány ručně nelze předem zjistit kdy explodují, protože tento okamžik určuje hráč. Pokud se navíc bomba pohybuje, nelze ani určit cílové políčko výbuchu. Pro účely předpovědi se počítá s okamžitou explozí ručně odpalované bomby a tedy aktuálním políčkem. Vylepšit by se tento případ dal označením více cílových políček dopředu, aby příšera věděla, na která políčka případně může bomba dojet a měla čas se ukrýt.

Ze znalosti cílového políčka výbuchu se už jednoduše určí, která všechna políčka výbuch zasáhne. Na tato políčka bomba vloží speciální předpovědní objekt (třída *Presumption*), podle kterého už všechny příšery poznají, že na políčku bude v blízké době smrtící plamen.

Když už tuto znalost máme, není problém ji vizualizovat i pro uživatele. Může si nastavit v menu, jestli předpovědi exploze chce vizualizovat či ne. Na výpočtu předpovědí ale toto nastavení nic nemění.

4.10 Příšery a hráč

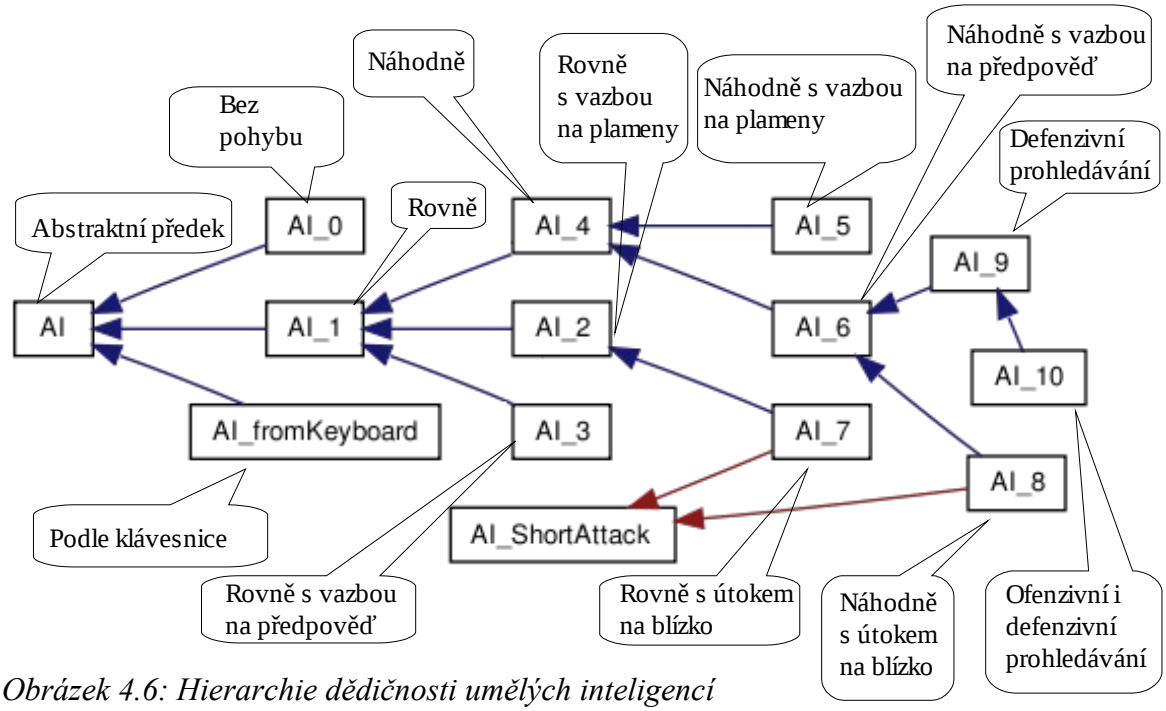
Příšera (třída *Creature*) je dynamický objekt hry. Pohybuje se, je smrtelná a zároveň pro hráče smrtící. Její pohyb není přímo zabudován ve třídě *Creature*. Je delegován do umělé inteligence (třída *AI*), která je parametrizována vlastností příšery – inteligencí.

Třída *AI* je abstraktní předek tříd, které implementují strategie pro pohyb příšer. Navenek je rozhraní umělé inteligence velice strohé (metoda *move()*), což minimálně omezuje implementaci umělé inteligence, která může v rámci této metody hýbat s příšerou jak chce.

Za nejzákeřnější inteligenci považuji *AI_8*, která na základě předpovědí plamene dává pozor, aby do plamene nevstoupila a za pomoci předka *AI_ShortAttack* útočí na blízko na hráče vyšší rychlostí, než je její obvyklá.

Nejzdařilejší a nejhůře zabitelná příšera disponuje inteligenci *AI_10*, která využívá ofenzivně i defenzivně prohledávání mapy do šířky. Pokud je v nebezpečí (podle předpovědí výbuchu), pokusí se nalézt nejbližší bezpečné políčko. Pokud v nebezpečí není, použije prohledávání do šířky pro nalezení nejbližšího hráče, na kterého by zaútočila.

Hráč je v hierarchii dědičnosti speciálním případem příšery. Mají totiž mnoho aspektů společných, jsou smrtelní, pohybují se stejným způsobem, dokonce i systém umělých inteligencí slouží u hráče k implementaci jeho pohybové strategie. Třída *AI_fromKeyboard* je speciální (ne)umělá inteligence pro hráče ovládaného uživatelem. Je řízena vstupem z klávesnice. Pokud uživatel nic nedělá, hráčem nehýbe. Řeší, aby uživatel svou interakcí chodil pouze tudy, kudy může. Chodí s hráčem čtyřmi směry, pokládá bomby, kope do nich, odpaluje je. Toto implementační řešení podporuje možnost jednoduše přidat opravdu umělou inteligenci pro hráče, což by však bylo nad rámec této práce.



Obrázek 4.6: Hierarchie dědičnosti umělých inteligencí

5. Implementace editoru map

5.1 Požadavky

Protože by měl editor sloužit jak pro vývojáře, tak i pro uživatele hry, měl by být intuitivní, uživatelsky přívětivý a dostatečně silný, aby uměl vytvářet nové a prohlížet či měnit existující mapy pro hru v obou módech.

Jako vývojář hry jsem určil definici mapy tak aby ji bylo možné upravovat v textovém formátu XML co nejvíce intuitivně. Sám jsem si to na několika desítkách map vyzkoušel a většinu map, které jsou nyní součástí hry, jsem napsal ručně. Je to ale komplikované a zejména u složitějších map musí mít člověk velkou představivost. Konstrukce některých map mi trvala dlouho a musel jsem pomocí hry zkoušet, zda je mapa správně. V editoru map bych tedy chtěl, aby uživatel viděl mapu tak, jak se bude hrát.

Je třeba zobrazovat (a nechat modifikovat) všechny důležité vlastnosti mapy. Rozměr mapy, její pozadí, všechny umístěné i generované objekty, obvodové zdi, umístění startu hráčů, informace o tom, kde lze generovat bedny nebo příšery.

Mezi požadavky jsem nepřidal možnost tvorby či modifikace definic objektů mapy či pozadí, protože ty není potřeba modifikovat v takové míře jako mapy a je to mnohem jednodušší. Vývojář si je může tvořit ručně. Do budoucna by ale mohlo být zajímavé takové rozšíření editoru vytvořit, jestli se ukáže, že je to opravdu potřeba.

5.2 Jazyk a knihovny

Jako programovací jazyk pro editor map jsem si zvolil C++, protože je v něm možné psát i aplikace více orientované na GUI a protože jsem se v jeho používání během programování hry zdokonalil.

Editor map je typický GUI program, s jedinou zvláštností. Tou je vizualizace mapy a objektů v ní. Pro GUI vzhledem k přenositelnosti, využitelnosti a kompaktnosti jsem vybral knihovnu Qt. Po přečtení dokumentu *Co je nového ve verzi Qt 4.2* [15], jsem zjistil, že Qt implementuje nový framework pro zobrazování velkého množství grafických prvků a interaktivní práci s nimi. To bylo hlavním důvodem, proč jsem se nakonec pro Qt opravdu rozhodl. Jako příručku jsem používal referenční dokumentaci pro verzi *Qt 4.3* [12]. Pod touto verzí jsem také editor vyvíjel, je ale kompatibilní i s aktuálně poslední verzí 4.6.

Parsování XML souborů i načítání grafických souborů knihovna Qt také implementuje, pro XML jsem použil rozhraní DOM[7] implementované v modulu QtXml.

7 DOM (akronym anglického Document Object Model – objektový model dokumentu) je objektově orientovaná reprezentace XML nebo HTML dokumentu. DOM je API umožňující přístup či modifikaci obsahu, struktury, nebo stylu dokumentu, či jeho částí [7].

5.3 Hlavní části programu

Editor map ovládá pouze jedno okno a může editovat v jednom okamžiku pouze jednu mapu. Pokud chceme editovat mapu jinou, je třeba aktuální mapu zavřít (myšleno interně, uživatel pouze otevře jinou mapu a stará se zavře automaticky). Z tohoto důvodu máme ve hře několik singletonů[8].

- *MainWindow* obstarává jediné okno editoru
- *ResourceHandler* – třída na načítání (a ukládání) vnějších zdrojů
- *MapObjectPalette* – paleta objektů drží prototypy všech objektů mapy načtených do editoru. Ty, které se dají vkládat do mapy, nabízí uživateli k výběru. Po vybrání se takový objekt stává pracovním objektem.
- *MapView* – primárně pohled na mapu. Tato třída ale zastřešuje celou práci s mapou a vizualizuje aktuální stav mapy.

5.4 Vnější zdroje

Vzhledem ke zkušenosti ze hry, kdy se mi převod XML dokumentu dostal hluboko do programu, jsem se rozhodl, že více oddělím načítání objektů definovaných pomocí XML souborů, načítání jejich obrázků atd. od ostatního funkčního kódu. Za tímto účelem jsem vyvinul třídu *ResourceHandler*, který se stará sám o načítání veškerých objektů z vnějších zdrojů nebo alespoň poskytuje rozhraní a načítání deleguje na specializované třídy.

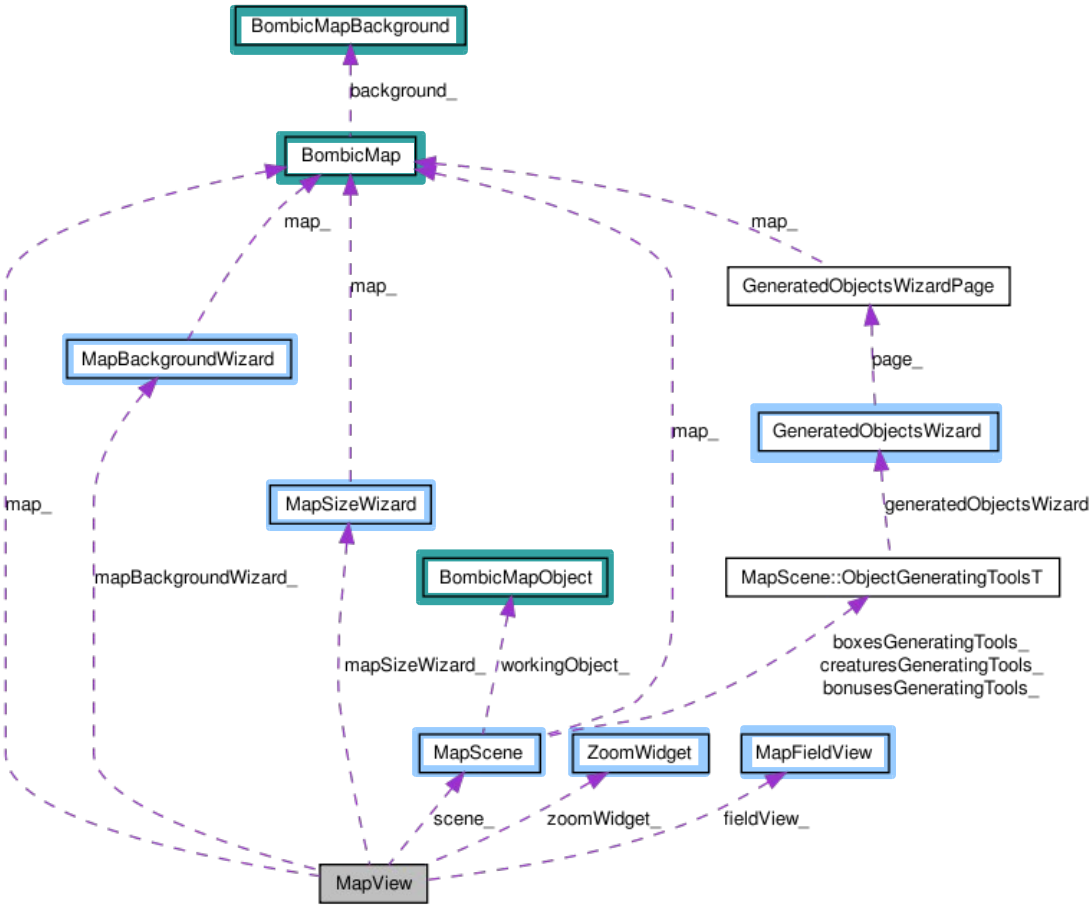
Jednou z takových specializovaných tříd je *MapResourceHandler*, který se stará o načítání definice mapy, o její konstrukci a o ukládání rozpracované mapy zpět do souboru. Struktura definice mapy je tedy výhradně řešena v této třídě.

Dalším specializovaným handlerem je *MapObjectResourceHandler*, třída, která je předkem načítacích handlerů všech objektů mapy. Tyto handlery, specializované na jednotlivé typy objektů registruje a podle XML dokumentu dokáže rozhodnout, který z nich (jestli nějaký) by jej mohl dokázat zpracovat. Struktura definice jednotlivých typů objektů mapy je tedy řešena výhradně ve specializovaných handlerech podle daného typu. Je tedy implementačně jednoduché upravit definici stávajícího nebo přidat definici nového typu objektu (viz 5.7).

5.5 Prvky GUI

Centrálním prvkem grafického rozhraní je instance *MapView* (potomek *QGraphicsView*), ve kterém se zobrazuje scéna mapy, pokud je nějaká otevřena. Ostatní prvky jsou buď dokovatelné (součástí okna, uživatel si je může z okna vytrhnout nebo uspořádat v okně podle sebe), nebo dialogové (uživateli se kvůli určité akci zobrazí a po ukončení akce zase zmizí). Obrázek 5.1 znázorňuje, které prvky *MapView* využívá k prezentaci a práci s mapou.

8 Singleton je třída, která tvoří v programu jedinou instanci.



Obrázek 5.1: Provázanost MapView a podřazených prvků GUI

Významným dokovatelným prvkem je paleta objektů *MapObjectPalette*, ve které jsou přehledně rozděleny objekty podle typu. Uživatel si zde může vybrat objekt, který následně vloží do mapy. Takový objekt jsem nazval *pracovní objekt.* Jaký objekt je aktuálně pracovní (jestli vůbec je) vizualizuje dokovatelný prvek *working object label* (třída *QLabel*).

Na jednom políčku mapy může být více objektů. Ve scéně se tyto objekty zobrazí přes sebe (i když jsem implementoval funkci, která objekty na políčku posune tak, aby nebyly úplně v zákrytu) a pracovat se dá pouze s jedním vrchním objektem. Z tohoto důvodu jsem implementoval další dokovatelný prvek *MapFieldView*, který zobrazuje o vybraném políčku mapy, zda na něj lze generovat objekty, a vizualizuje přehledně všechny objekty napevno umístěné na tomto políčku. Uživatel tedy pomocí detailního pohledu na políčko může pracovat (odstranit, přesunout) s kterýmkoli objektem na políčku a zakázat či povolit generování objektů na tomto políčku.

Aby měl uživatel celkový přehled i nad velkými mapami, implementoval jsem *ZoomWidget*, který dokáže přibližovat či oddalovat pohled na mapu. Zejména oddálení pomůže uživateli při vyvíjení map, které se nevejdou celé na obrazovku. Ve škálovaném pohledu se dá s mapou pracovat stejně jako při originálním zvětšení, reakce scény jsou ale kvůli škálování trochu pomalejší.

Výše uvedené prvky GUI jsou naskládané do jednoho okna editoru (viz obrázek 6.14).

Mezi dialogové prvky patří různé hlášení chyb, výběr souboru pro načtení či uložení a průvodci složitějšími úpravami s mapou.

Změna rozměru mapy (respektive změna pozadí) je z implementačního hlediska složitá operace. Pokud mapu zmenšujeme (respektive nastavujeme pozadí, jehož obvodové zdi zabírají více místa) musíme dbát na to, aby uživatel případnou ztrátou objektů v inkriminované oblasti nepřišel o rozdělanou práci (někdy by to uživatel mohl chtít, ale kdyby se spletl a mapu zmenšil více než chce, bylo by to špatné). Z tohoto důvodu jsou zde průvodci (viz obrázek 6.18). Uživatel si zde nastaví nové rozměry mapy (respektive nové pozadí). Po potvrzení je vytvořena nová mapa s nastavenými parametry a objekty ze staré mapy jsou do nové překopírovány. Pokud dojde k chybě, uživatel se o tom dozví a o starou mapu nepřijde. Pokud projde vše hladce, mapy se pouze vymění.

Do mapy se náhodně generují některé typy objektů (viz podkapitolu 2.4). V menu lze zapnout vizualizace těchto vygenerovaných objektů. Nastavit, které objekty se mají generovat a v jakém množství, lze skrze průvodce generovaných objektů (viz obrázek 6.18). V menu pod položkou např. *generované bonusy* se skrývá průvodce nastavením generování bonusů. Jsou zde zobrazeny všechny načtené objekty typu *Bonus* a u každého objektu, kolik se jich v mapě má vygenerovat. Tento údaj lze změnit a potvrzením akce se počty generovaných objektů upraví.

5.6 Mapa a její vizualizace

Mapa je reprezentována třídou *BombicMap*, vlastní pozadí mapy (třída *BombicMapBackground*) a umístěné objekty (potomci *BombicMapObject*). Pro každé políčko navíc drží informace o generování objektů (viz 5.8). Obsahuje všechny informace potřebné ke korektní definici mapy pro hru. Informace jsou v ní uloženy tak, aby umožňovala efektivní editaci. Objekty mapy lze přidávat či odstraňovat. Pomocí *BombicMap* se dá zjistit, jestli je možné na konkrétní políčko vložit konkrétní objekt.

Jako rozhraní mezi uživatelem a mapou slouží třída *MapScene* (potomek *QGraphicsScene*). Ta provádí vizualizaci a práci s mapou. Pro přesouvání objektů je zde použit mechanismus *Drag&drop*[9], kdy uživatel táhne objekt na vrchu políčka a umístí jej na jiné políčko. Tato akce byla rozšířena i pro detail políčka, odkud mohou být objekty taženy do scény (kvůli lepšímu výběru objektu z políčka).

Objekty mapy jsou vizualizovány tak, jak jsou prezentovány ve hře. U animovaných objektů se zobrazuje pouze první obrázek animace. U generovaných objektů jsem řešil, jestli zachovat jejich vzhled tak, jak je zná uživatel ze hry, nebo je nějak odlišit od pevně umístěných objektů. Vzhledem k tomu, že umístěné objekty jsou jasně vidět v detailu políčka a generované objekty se dají úplně skrýt, jsem se uchýlil k první variantě zachování nativního vzhledu objektu.

9 Drag and drop – česky táhni a upusť je intuitivní akce myší, kdy uživatel chce přesunout (nebo okopírovat) objekt z místa A na místo B. V místě A tedy objekt myší chytne (stiskem tlačítka), objekt táhne a v místě B jej upustí (uvolněním tlačítka).

5.7 Objekty mapy

Podobně jako u hry je pro objekty mapy vytvořena hierarchie dědičnosti. Zde se ale objekty mapy liší zejména v tom, jestli mohou být uživatelem přemístěny, odstraněny, s jakými objekty mohou být na stejném políčku a jak podporují na políčku generování objektů. Hierarchie je tedy plochá, abstraktní předek *BombicMapObject* určuje společné rozhraní všech objektů a jeho přímí potomci (pro každý typ objektu jeden) toto rozhraní implementují.

Pro přidání nového typu objektu je třeba provést následující:

1. Podědit od *BombicMapObject* – vytvořit vlastní typ objektu (používaný v mapě).
2. Podědit od *MapObjectResourceHandler* – vytvořit handler pro načítání definice našeho nového objektu.
3. V implementaci *MapObjectResourceHandler* přidat hlavičkový soubor nového handleru a nový handler na vyznačeném místě registrovat.
4. Přidat nový typ objektu do výčtového typu *BombicMapObject::Type*
5. Přidat novou stránku do palety objektů (nezapomenout přidat větev pro nový typ do switch v *MapObjectPalette::addObject()*).

5.8 Generování objektů

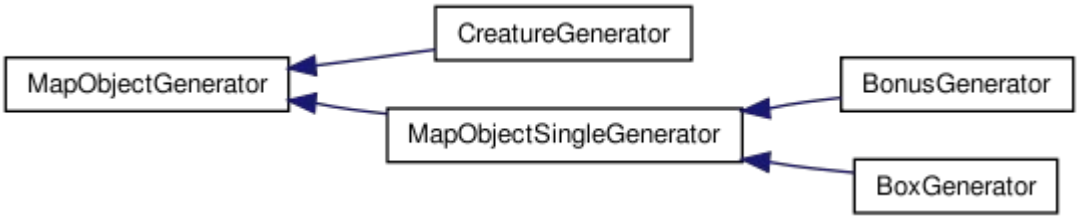
Aby byla uživatelova představa o mapě ucelená, bylo třeba simulovat generování objektů tak, jak se to dělá při vytváření mapy ve hře. Současně se musí dbát na zakázaná políčka z definice mapy, na objekty, které nedovolují na stejném políčku vygenerovat určité objekty a hlavně na rychlost umisťování objektů během editace mapy. Kdyby se totiž při každé změně mapy generované objekty všechny znovu vygenerovaly, příliš by to aplikaci brzdilo a ani by to nebylo hezké, protože by se mapa příliš měnila.

Navrhl jsem tedy řešení pomocí generátorů objektů (třída *MapObjectGenerator*). Každé políčko mapy má pro každý typ objektu, který lze generovat, generátor, který již sám řeší, jestli může či nemůže generovat objekty.

Generátor má několik aspektů, podle kterých rozhoduje, zda v něm lze generovat další objekt či nikoli.

- zakázaný/povolený – z definice mapy - dáno explicitně uživatelem
- blokovaný/odblokovaný – z vlastností objektů – dáno ostatními objekty
- plný/prázdný – z vnitřního stavu - Generátor obecně může generovat najednou více objektů. Jeho specializace *MapObjectSingleGenerator* tuto vlastnost omezuje a dovoluje generovat pouze jeden objekt. Pokud je tedy již jeden objekt generován, další objekty generovat nemůže.

Generátory, které mohou generovat další objekty a objekty, které je ještě potřeba vygenerovat spravuje scéna mapy *MapScene*, která se při každé modifikaci mapy pokusí potřebné objekty vygenerovat. Zároveň od generátoru přijímá objekty, které generoval a nyní nemůže (například kvůli zakázání uživatelem). Scéna také vizualizuje generování (informaci jestli je či není možné generovat a aktuálně generované objekty).



Obrázek 5.2: Hierarchie dědičnosti generátorů objektů

5.9 Načítací sady

Všechny načtené objekty mapy registruje automaticky *ResourceHandler v* paletě objektů *MapObjectPalette*. Chce-li uživatel přidat do mapy nový nenačtený objekt, musí jeho definiční soubor najít na disku a nechat objekt načíst a registrovat v paletě. Z palety už si jej může jednoduše vybrat a vkládat do mapy. Práce se soubory definic ale není příliš uživatelsky přívětivá a pokud navíc uživatel neví, kde na disku soubory hledat, mohlo by mu to dělat potíže.

Načítací sada je množina objektů, které k sobě logicky patří (například všechny příšery, všechny objekty vhodné pro téma les atd.). Součástí načítací sady může být pozadí mapy, ke kterému se sada vztahuje.

Začne-li tedy uživatel vytvářet mapu s pozadím les, automaticky se mu načtou všechny prvky sady vhodné pro téma les. Pokud navíc chce vkládat do této mapy příšery a nechce je hledat mezi soubory na disku, může si přes menu načíst sadu obsahující všechny příšery. Ty potom může z palety do mapy jednotlivě vložit nebo je může nechat náhodně vygenerovat průvodcem generování příšer.

6. Uživatelská dokumentace

6.1 Instalace

Nejdříve je třeba nainstalovat potřebné knihovny. Hra potřebuje nainstalované knihovny SDL, SDL_ttf, SDL_image (s podporou jpg formátu) a knihovnu AGAR. Editor vyžaduje nainstalované Qt SDK.

Pokud nemáte SDL instalované, pokračujte prosím podle [13]. Nezapomeňte nainstalovat kromě samotné SDL knihovny také přídavné knihovny SDL_ttf a SDL_image.

Knihovna agar není příliš rozšířena, distribuce Bombiče 2 tedy obsahuje knihovnu agar ve verzi 1.3.3 s menšími změnami. Pro instalaci prosím změňte pracovní adresář na *game/libs/agar* například takto:

```
$ cd game/libs/agar
```

Zde je třeba knihovnu rozbalit a konfigurovat. Pro jednoduchost jsem zde připravil konfigurační skript. Stačí jej tedy spustit:

```
$ ./configure
```

Nyní už jen zkompilovat a instalovat knihovnu do systému:

```
$ make
```

```
$ sudo make install
```

Pokud nemůžete získat práva superuživatele nebo nechcete instalovat knihovnu do systému, můžete použít parametr *--prefix* u příkazu *make install* pro nainstalování knihovny kamkoli jinam. Pokud ale spustitelný *agar-config* nebude v automatické cestě *$PATH*, je třeba uvést cestu v *game/Makefile*.

Knihovnu Qt (pokud ji již nemáte instalovanou) prosím nainstalujte z balíčku nebo podle [11].

Konfiguraci instalace je možno provést v hlavním *Makefile* modifikací několika proměnných na začátku souboru.

Nyní je možno zkompilovat a instalovat projekt. Pokud chcete zkompilovat pouze hru, spusťte jednoduše:

```
$ make
```

Pro hru i editor je třeba spustit:

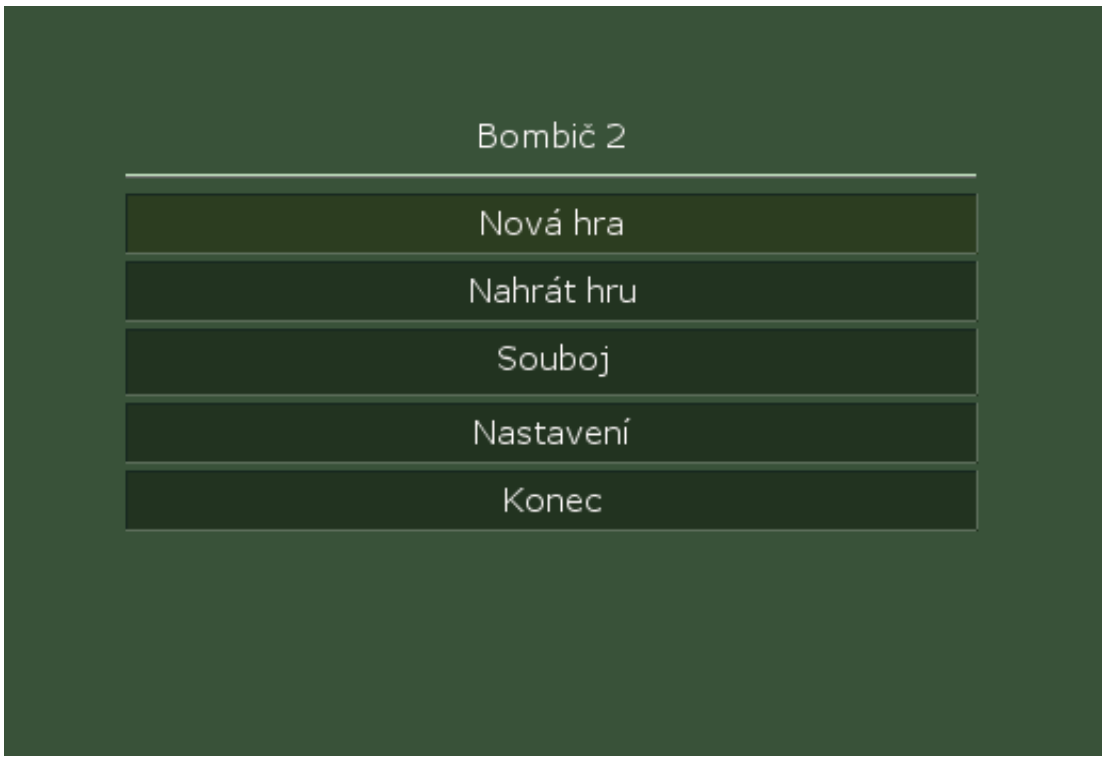
```
$ make all
```

Instalovat lze celý projekt pomocí příkazu:

```
$ make install
```

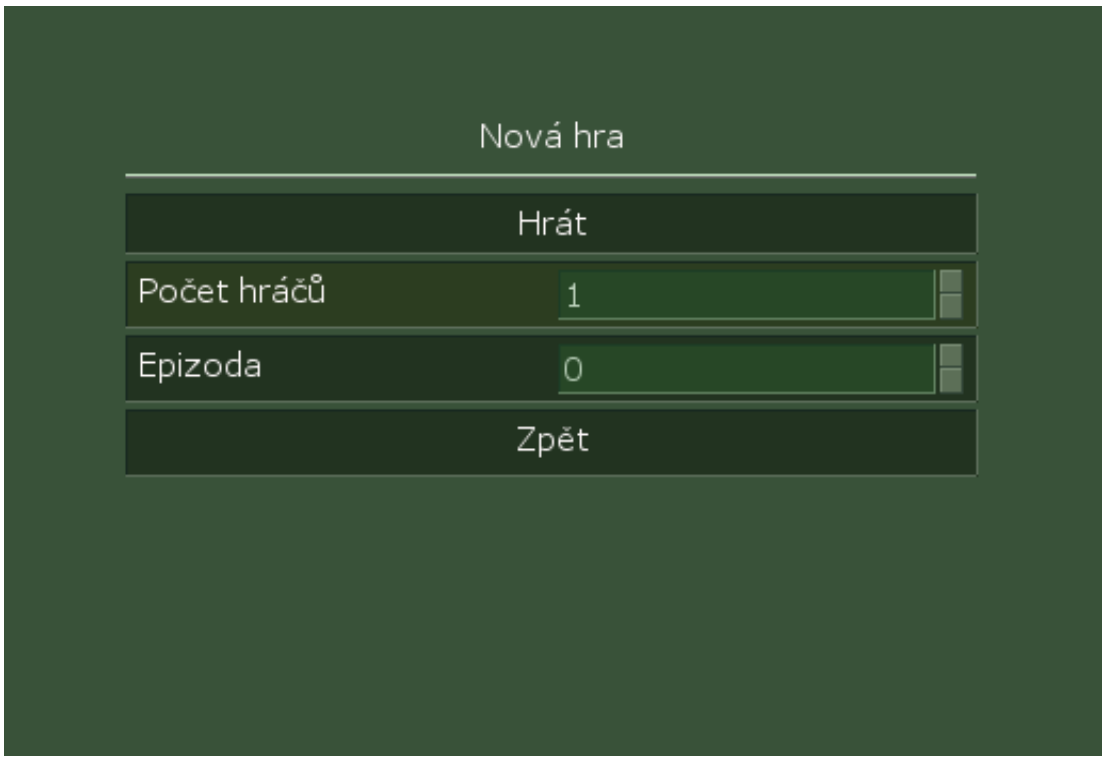
6.2 Menu hry

Kooperativní hru (jeden až čtyři hráči) lze spustit z hlavního menu pomocí první položky *Nová hra*.



Obrázek 6.1: Nová hra – hlavní menu

Zvolíte si požadovaný počet hráčů a opět první položkou hrát se dostanete na úvodní obrazovku prvního kola.



Obrázek 6.2: Nová hra - spuštění

Obrázek 6.3: Nová hra – úvodní obrazovka

Z úvodní obrazovky se klávesou *Esc* dostanete zpět do menu, klávesami *Enter, Ctrl* nebo *Space* pokračujete do hry.

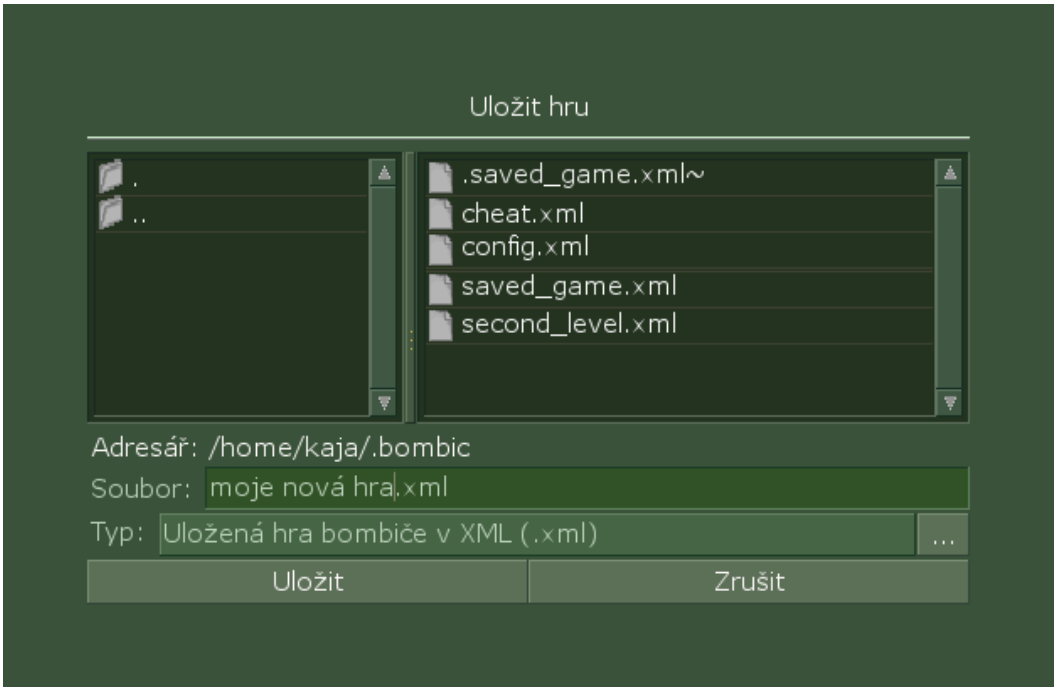
Obrázek 6.4: Nová hra – hraní prvního kola

Po nezdařilém ukončení kola se vám objeví úmrtní obrazovka.

Obrázek 6.5: Nová hra – úmrtní obrazovka

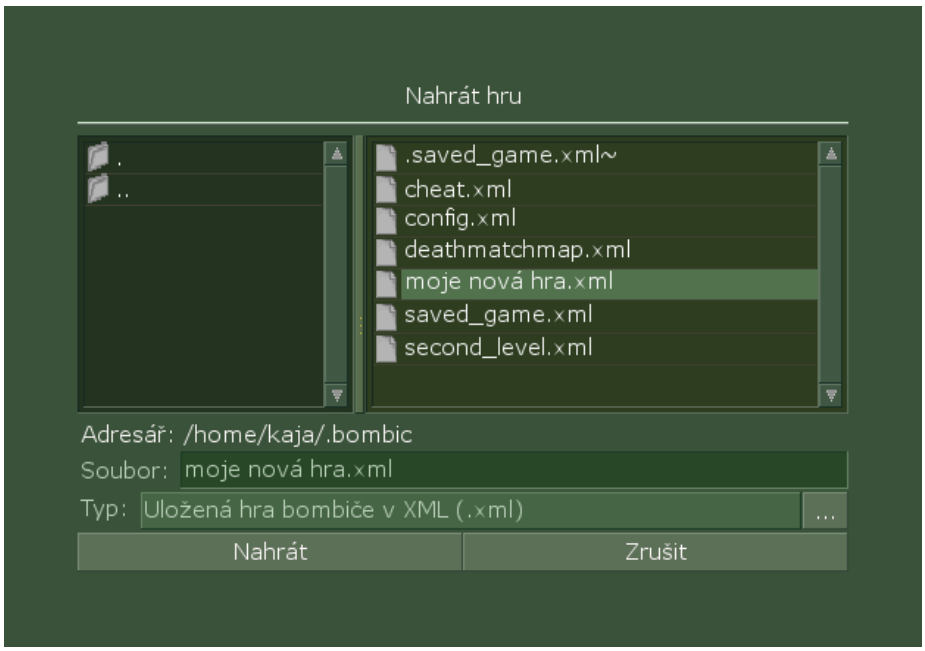
Stiskem *Esc* se dostáváte do menu, klávesami *Enter, Ctrl* nebo *Space* opět na úvodní obrazovku kola a můžete jej hrát znovu. Po úspěšném ukončení kola se vám automaticky ukáže úvodní obrazovka kola dalšího.

Po návratu do menu se můžete zpět do rozehrané hry vrátit nebo hru uložit. Položka *Uložit hru* zobrazí souborový manažer, ve kterém si vyberete soubor, do kterého se má hra uložit. Po uložení hry můžete pokračovat ve hře nebo program ukončit.



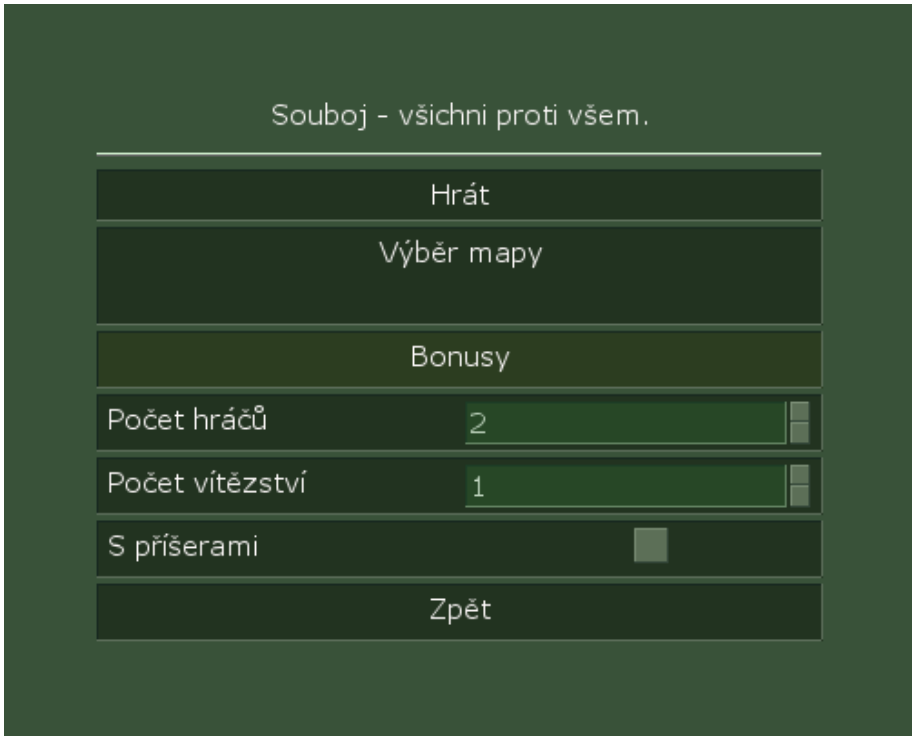
Obrázek 6.6: Uložení hry

Pro návrat do hry opět spusťte program a v hlavním menu vyberte položku *Nahrát hru*. Zobrazí se souborový manažer, ve kterém si vyberete soubor s uloženou hrou. Po potvrzení již můžete hrát.



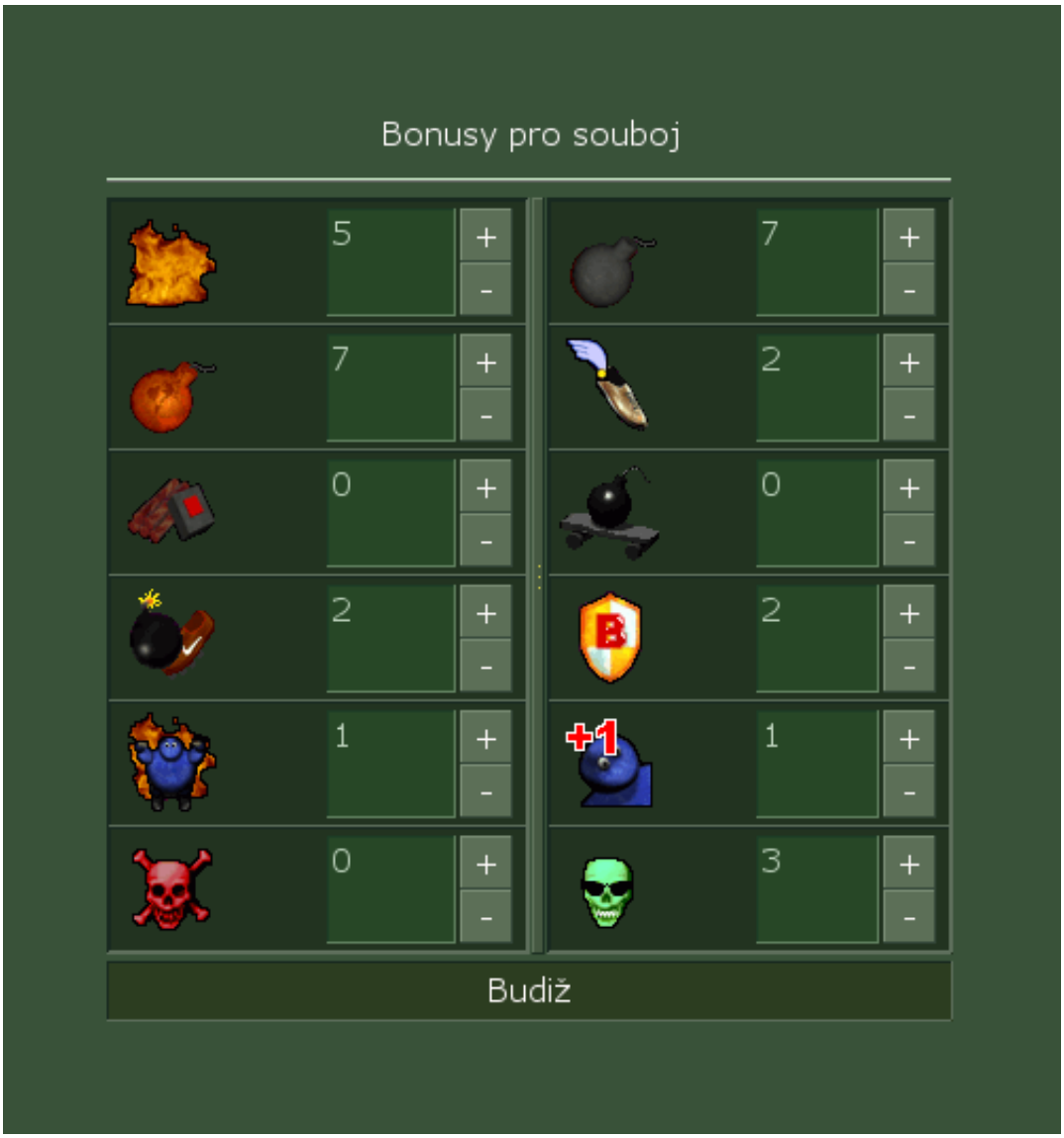
Obrázek 6.7: Nahrání hry

Kromě kooperativního módu, kde hráč (či více hráčů společně) bojuje proti příšerám, hra poskytuje také souboj, kde se utkají hráči na život a na smrt proti sobě. Pro spuštění tohoto módu vyberte v hlavním menu položku *Souboj* a v podmenu nastavte mapu, počet hráčů, na kolik vítězství chcete utkání hrát a mají-li se do hry umístit i příšery.



si tedy

Obrázek 6.8: Spuštění souboje

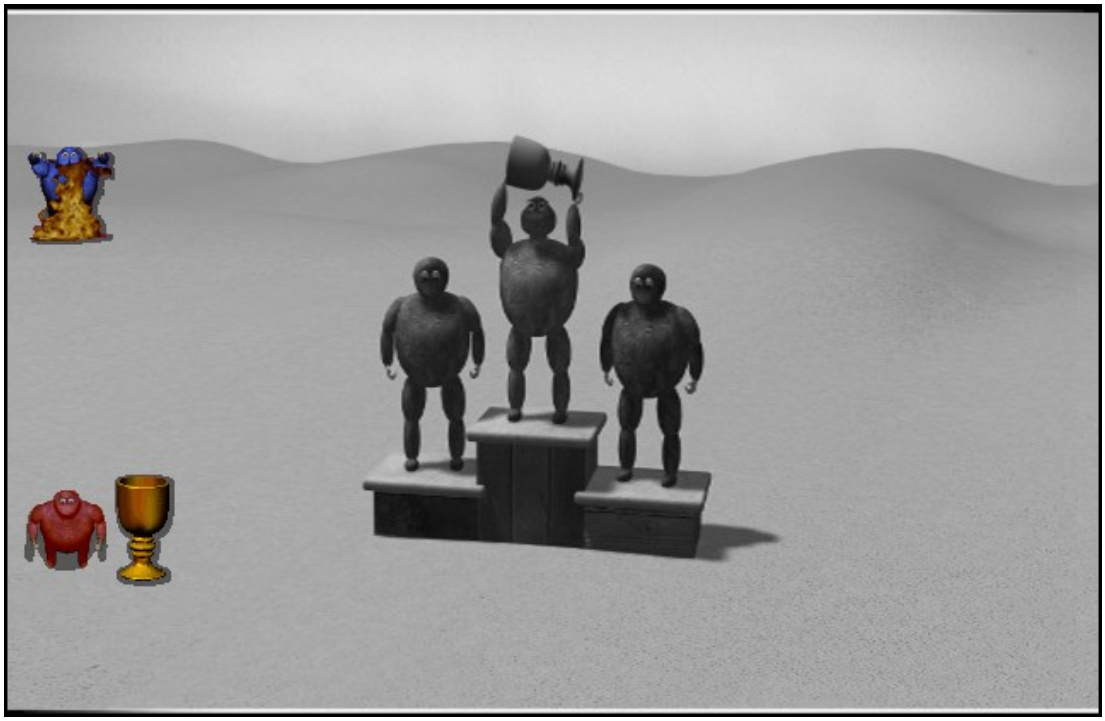


Obrázek 6.9: Nastavení bonusů pro souboj

Nyní se položkou *Hrát* dostáváte přímo do souboje. Souboj hrají proti sobě hráči, dokud jsou alespoň dva živí.

Obrázek 6.10: Hraní souboje

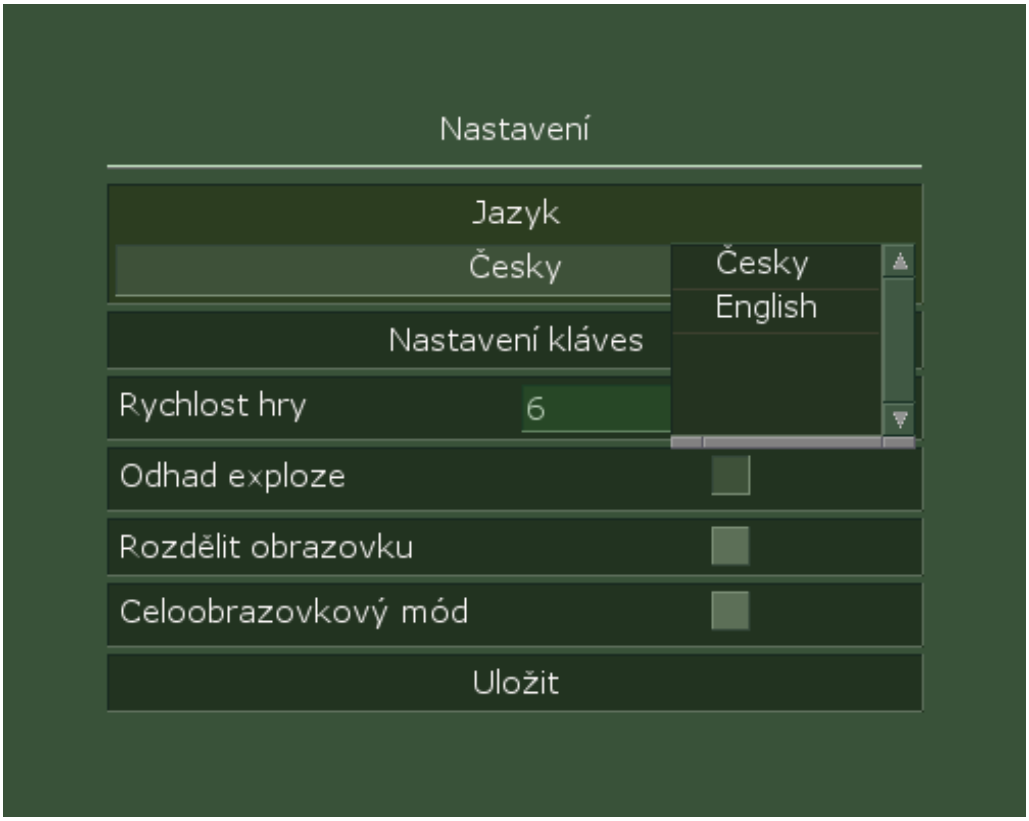
Po ukončení jedné hry se započítá bod tomu, kdo přežil, a zobrazí se aktuální skóre. Stejně jako u kooperativní hry lze turnaj kdykoli ukončit klávesou *Esc*, po zobrazení výsledků se další hra spustí po stisknutí jedné z kláves *Enter, Ctrl* nebo *Space.*



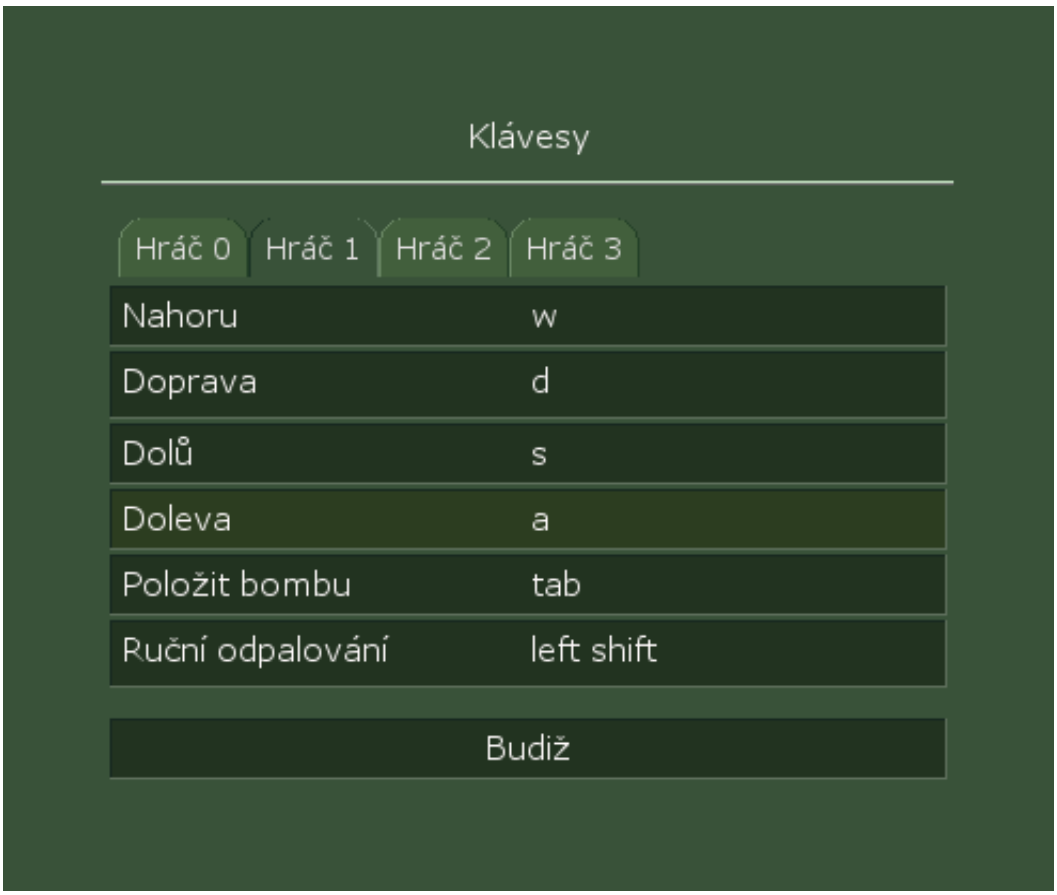
Obrázek 6.11: Skóre souboje

V podmenu *Nastavení* lze konfigurovat jazyk hry, různé její aspekty a klávesy, kterými uživatelé ovládají hráče.

- Jazyk – zvolte si preferovaný jazyk, zatím na výběr a*ngličtina, čeština.*
- Nastavení kláves – pro jednotlivé hráče nastavte šest ovládacích kláves, čtyři pro směr pohybu, klávesa pokládající bombu a klávesa pro ruční odpalování. Klávesy nemohou být přiřazeny více funkcím (druhé přiřazení zruší první).
- Rychlost hry – relativní rychlost hry určuje, jak rychle hra probíhá. Poměr například rychlosti příšer, hráče, doby výbuchu bomby, doby působení bonusu se nemění. Vyšší číslo zvyšuje rychlost hry (vše bude relativně urychleno).
- Odhad exploze – chcete-li vidět, kam plamen bomby dosáhne ještě než exploduje, nechte zapnutý odhad exploze. Při vypnutí této vlastnosti zůstane odhad skrytý.
- Rozdělit obrazovku – při hře více hráčů nastává problém, že se nemusí najednou vejít na obrazovku. Automaticky se vždy obrazovka rozdělí tak, že každý hráč vidí svůj kousek mapy. Při přiblížení hráčů se opět obrazovka spojí. Pokud chcete nechat vždy rozdělenou obrazovku, zapněte tuto vlastnost. Pro rozdělování a spojování obrazovky během hry tuto vlastnost vypněte.
- Celoobrazovkový mód – tzv. fullscreen. Zapnutí roztáhne okno na celou obrazovku.



Obrázek 6.12: Nastavení hry



Obrázek 6.13: Nastavení kláves

6.3 Hra

Hráč je ovládán šesti klávesami nastavenými v menu. Může chodit nahoru, dolů, doleva či doprava. Může pokládat bomby (má však omezeno, kolik jich může položit najednou) a pokud má časově omezený bonus ruční odpalování, jeho položené bomby neexplodují, dokud je ručně (klávesou pro odpalování bomb) neodpálí. Je možné položit několik bomb a pokud se neodpálí navzájem, lze je odpálit ručně postupně.

V mapě jsou objekty, které hráče blokují (nemůže přes ně jít) a které hráči ubírají životy (pokud je hráč s takovým objektem na jednom políčku, jeho počet životů se sníží, pokud se sníží na nulu, hráč umírá). Zde uvedu přehled objektů mapy, které jsou pro hráče nějakým způsobem zajímavé:

- Plamen – objekt, který se o mapy dostává po výbuchu bomby. Může zapálit či usmrtit některé objekty.
- Zeď – hráč ani příšera přes ni nemůže přejít. Zeď nehoří, nedá se tedy žádným způsobem odstranit.
- Bedna – hráč ani příšera přes ni nemůže přejít. Bedna hoří, dá se tedy odstranit tak, že v jejím okolí exploduje bomba.

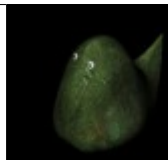
- Bonus – hráč jej může sebrat. Bonus ovšem hoří, exploduje-li tedy poblíž bomba, bonus zmizí.
- Příšera – smrtelná nestvůra, pro hráče je ale smrtící. Musí se jim tedy vyhýbat a bojovat proti nim.

Cíl hry záleží na jejím módu. Při kooperativní hře se hráči primárně snaží zabít svými bombami všechny příšery. Když v mapě nezůstane žádná příšera, hráč se automaticky dostává do dalšího kola. V souboji jde o to, zabít ostatní hráče a zůstat naživu. Kdo zůstane jako jediný naživu, získává zlatý pohár. Když nějaký hráč dosáhne nastaveného počtu pohárů (vítězství), vyhrává celé utkání.

Hráč může vylepšit své schopnosti různými bonusy, které jsou schovány pod bednami. Po výbuchu bedna shoří a bonus se objeví.

	Plamen – zvýší dosah bomb. Je přenosný mezi koly.		Bomba – zvýší počet bomb, které může hráč najednou položit. Je přenositelný.
	Megabomba – přidá jednorázově hráči do výbavy dvě bomby s velkým dosahem.		Superbota – zvýší hráčovu rychlost. Je přenositelná mezi koly. Čím více jich hráč sebere tím méně se projevuje.
	Posílání bomb – pokud hráč není v pohybu, bomba není položena ale poslána směrem, kterým hráč stojí. Bonus platí do konce kola.		Kopání do bomby – hráč může bomby rozpohybovat či měnit jejich směr kopáním. Bonus platí do konce kola.
	Ruční odpalování – položené bomby explodují, až když je hráč odpálí. Časově omezeno.		Štít – hráč je po dobu držení nesmrtelný. Bonus je časově omezený.
	Život – jednorázově přidá hráči život. Přenositelný mezi koly.		Ohnivý muž – z hráče místo pokládání bomb vystřelují plameny a je nesmrtelný.
	Nemoc – negativní bonus aktivuje nějakou špatnou vlastnost. Časově omezen.		Nemoc pro ostatní – všem hráčům, kromě toho, který ji sebral, dá nějakou nemoc.

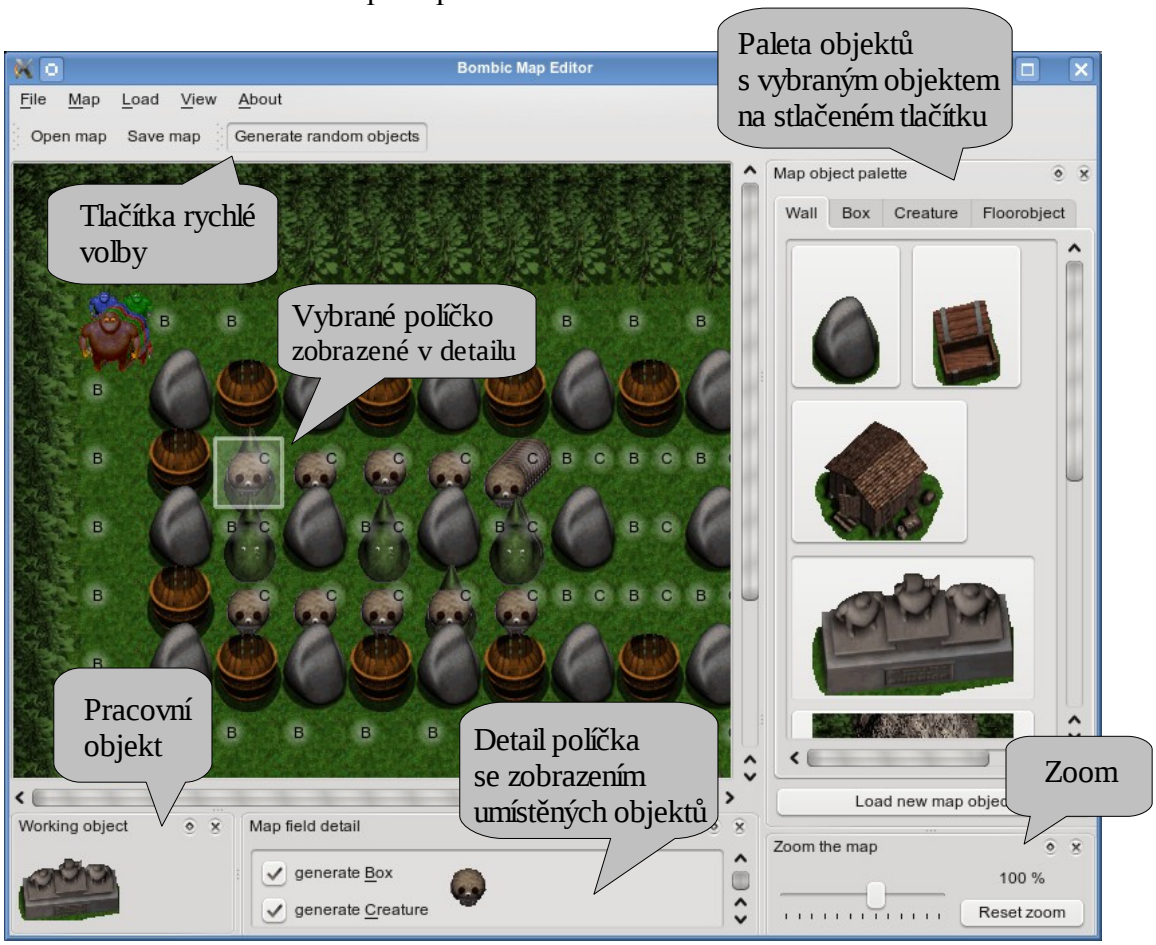
Největším hráčovým nepřítelem jsou jistě příšery. Ať jich je hodně, jsou těžko zranitelné, chytře se vyhýbají nebo zákeřně útočí, vždy dá hráči práci, aby se jim ubránil.

	Prototyp hloupé příšery. Její pohyb je předvídatelný. Má jeden život.		Hodně pomalá ale odolná příšera. Má dva životy.
	Není nejrychlejší, ale dává pozor na plameny a létá nepředvídatelně. Má jeden život.		Mutace předchozí hloupé příšery je již chytřejší a má dva životy.
	Velmi chytrá a rychlá příšera. Má sice jeden život, ale zabít ji nemusí být jednoduché.		Spíš než příšera by se dalo mluvit o pasti. Je to přímočaře valící se koule se dvěma životy.
	Vypadá jako ta přímočará hloupá past, ale vyhýbá se plamenům a jakmile je v blízkosti hráč, zákeřně zaútočí. Dva životy.		Tento duch vždy najde nejkratší cestu z nebezpečí. Má sice jen jeden život, ale zabít ho vyžaduje dobrou taktiku.
	Nejen že v nebezpečí ovládá taktiku nejbližšího úkrytu, ona také cíleně útočí na hráče a to vždy nejkratší možnou cestou. Naštěstí pro hráče má pouze jeden život a není příliš rychlá.		

Nyní vám mohu jen popřát hodně štěstí a zábavy při pokořování těchto nepřátel.

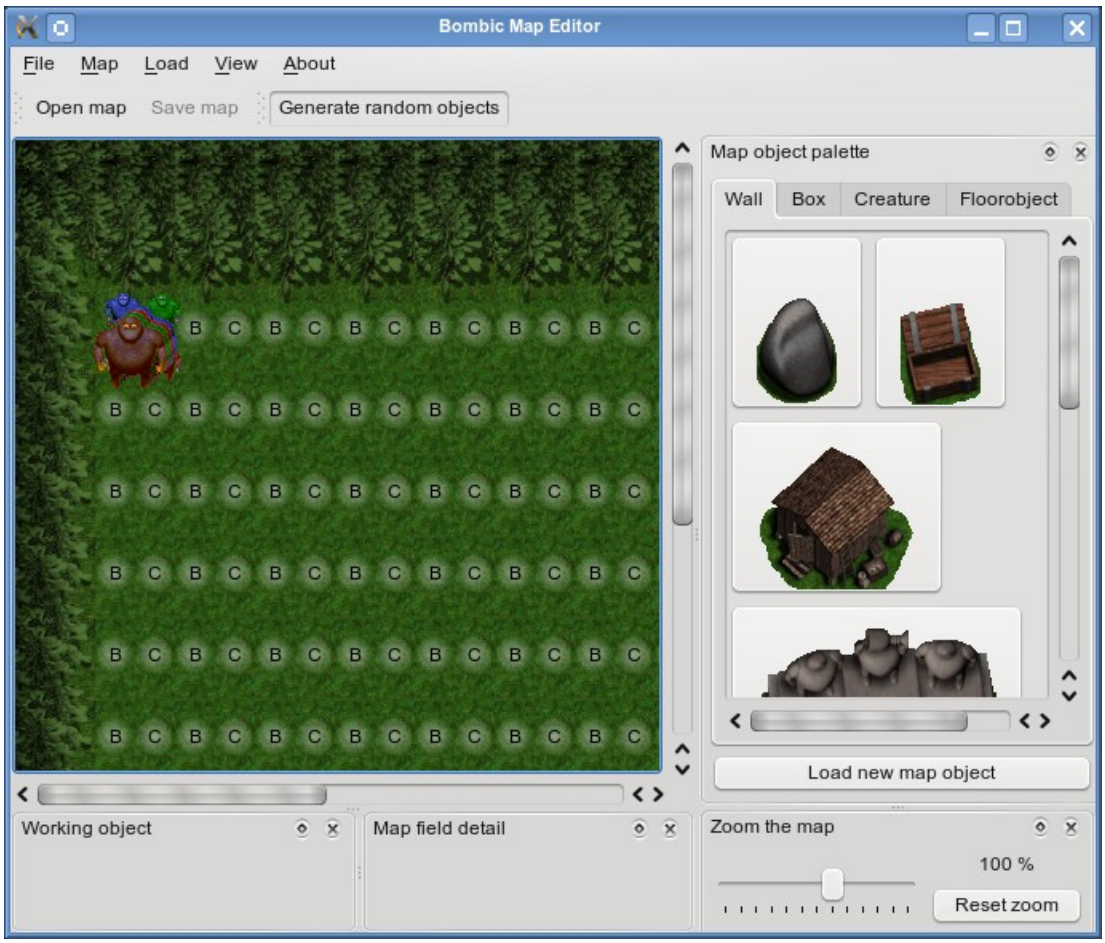
6.4 Editor map

Po spuštění editoru se vám zobrazí okno, ve kterém jsou vidět menu, tlačítka rychlé volby, nejspíše prázdná paleta objektů se záložkami podle jejich typů, zobrazení pracovního objektu, detail vybraného políčka a zoomovací nástroj. Většina těchto prvků jde přesunout, vytrhnout z okna nebo úplně skrýt. Po načtení mapy a objektů může okno editoru vypadat podobně jako na obrázku 6.14. S jednotlivými funkcemi vás ale seznámím postupně od začátku.



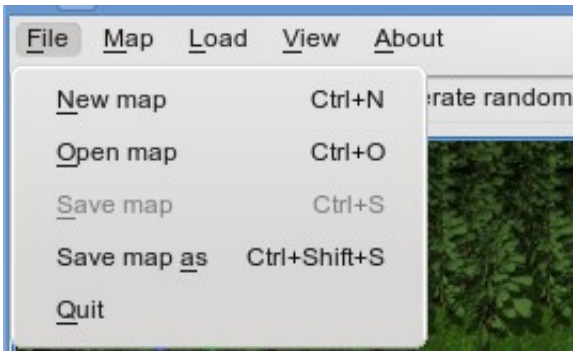
Obrázek 6.14: GUI editoru map

Začněme s vytvářením nové mapy. V menu tedy zvolíme položku *File > New map* a již se nám zobrazí prázdná mapa s obvodovými zdmi a umístěnými hráči tak, jako na obrázku 6.15. Obvodové zdi jsou součástí pozadí, nedají se ani odstranit ani přesunout, pouze se dají překrýt obyčejnými zdmi. Postavy hráčů určují políčka, kde ve hře mají hráči začínat. Velké postavy znázorňují start jednotlivých hráčů v souboji, čtyři malé postavičky pak společný start v kooperativním módu. Těchto pět objektů musí být v každé mapě. Nelze je tedy odstranit, pouze přesunout jinam. S načtením pozadí se podle sady *forest* načetly i všechny objekty, které s tímto pozadím souvisí. V paletě objektů si tedy již můžeme vybrat objekt, který následně do mapy vložíme. Než to ale uděláme, projdeme si jednotlivá menu.



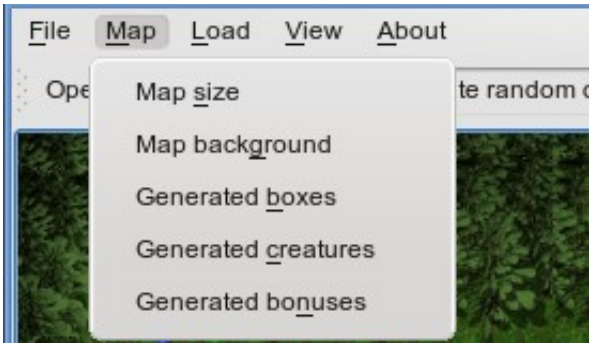
Obrázek 6.15: Nová mapa

Menu Soubor (*File*) obsahuje položky pracující s mapou jako se souborem. Můžete zde otevřít mapu (*Open map*) z disku nebo začít editovat novou mapu (*New map*). Pokud jste v mapě provedli nějakou změnu, můžete ji zde uložit (*Save map*) nebo zvolit nový soubor pro uložení (*Save map as*). V tomto menu můžete také program ukončit (*Quit*), což je stejné jako zavřít okno editoru. Některé položky mají své zástupce mezi tlačítky rychlé volby. Tato tlačítka mají stejný význam jako shodně pojmenovaná položka menu.



Obrázek 6.16: Menu File

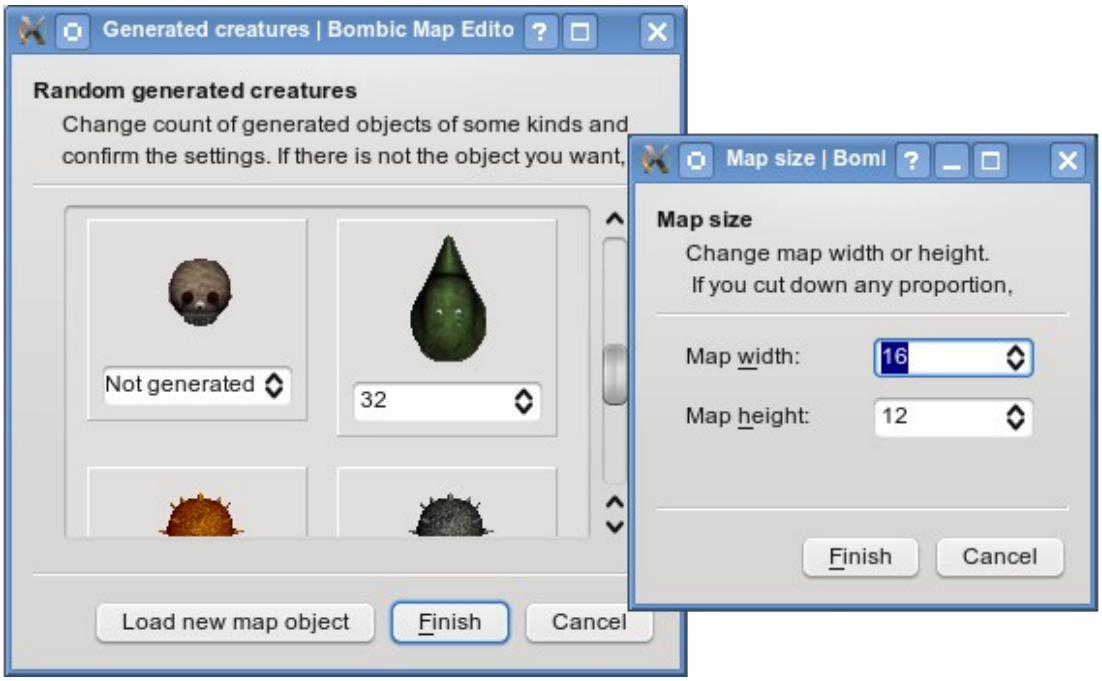
V menu Mapa (*Map*) lze pracovat s parametry mapy. Můžete zde pomocí přehledného průvodce změnit rozměry mapy, její pozadí a nastavit generované bedny, příšery či bonusy. Pokud vyvíjíte mapu výhradně pro použití v souboji, nemusíte žádné bonusy nastavovat. V souboji nemají bonusy nastavené v mapě žádný efekt, nastavují se ve hře před spuštěním souboje.



Obrázek 6.17: Menu Map

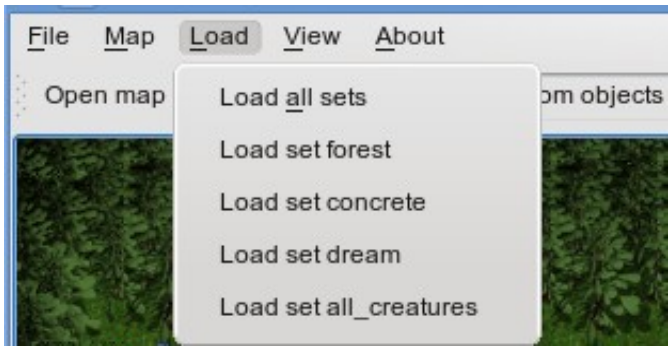
Při nastavování generování (např. příšer) se v průvodci zobrazí všechny aktuálně načtené příšery a u každé je uvedeno, kolik se jich má v mapě náhodně vygenerovat. Tento údaj můžete změnit a po potvrzení (tlačítko *Finish*) se počty generovaných objektů v mapě nastaví.

Při zmenšování rozměrů mapy je třeba, aby nebyly v odstraňované oblasti mapy umístěny žádné objekty. Pokud tomu tak není, první takový nalezený objekt vyvolá chybu. Z chybového hlášení lze poznat, na kterém políčku takový objekt leží. Objekt ručně odstraňte nebo přesuňte a změnu rozměrů mapy opakujte znovu.



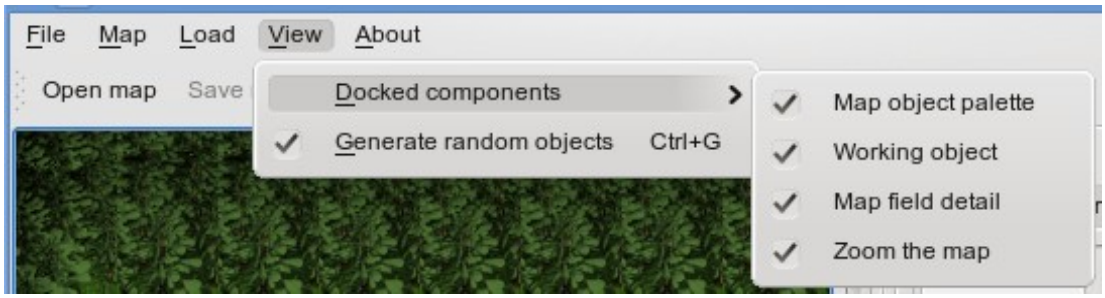
Obrázek 6.18: Průvodci nastavením generování příšer a rozměrů mapy

Menu Načítání (*Load*) skrývá seznam načítacích sad, které obsahují objekty nějak příbuzné (např. tématicky nebo podle typu). Výběrem položky tyto objekty načtete do palety objektů. Také si můžete načíst všechny sady (*Load all sets*). Objekty, které nejsou v sadách ani mezi automaticky načítanými objekty (např. některé bonusy pro souboj, vaše vlastní objekty, o kterých editor neví) můžete načíst pomocí tlačítka *Load new map object* v paletě objektů. Zobrazí se vám souborový manažer, pomocí kterého naleznete na disku definiční soubor pro objekt, který chcete načíst. Aby vše pracovalo správně a aby soubor mohla najít i hra, musí být soubor umístěn v adresáři, který je editorem i hrou prohledáván. Soubory vašich proprietárních objektů prosím umístěte do adresáře *.bombic* ve vašem domovském adresáři nebo do adresáře *common* (typicky instalovaný v systému nebo někde poblíž spustitelných souborů hry a editoru).



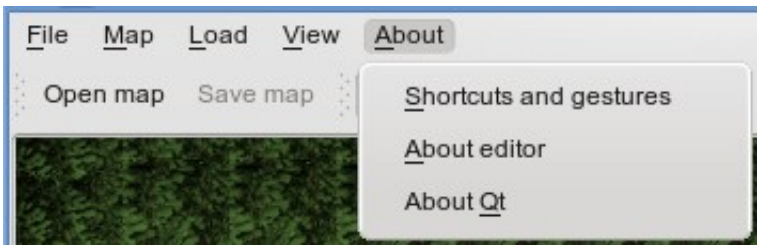
Obrázek 6.19: Menu Load

V menu Zobrazit (*View*) se nalézá podmenu s dokovatelnými nástroji. Můžete si zde nastavit, jestli nástroj chcete zobrazit nebo skrýt. Další položka se týká generovaných objektů. V mapě mohou být nastaveny generované objekty, které se před hrou vždy náhodně rozmístí. Editor map toto jednorázové rozestavování simuluje v průběhu editace mapy. Generované objekty jsou tedy vidět v mapě stejně jako umístěné. Touto položkou *Generate random objects* (popřípadě stejnojmenným tlačítkem rychlé volby nebo její klávesovou zkratkou *Ctrl+G*) můžete generování náhodných objektů dočasně vypnout.



Obrázek 6.20: Menu View

Poslední menu (*About*) poskytuje informace o použité knihovně Qt, o programu samotném a v položce *Shortcuts and gestures* shrnuje ovládání programu.

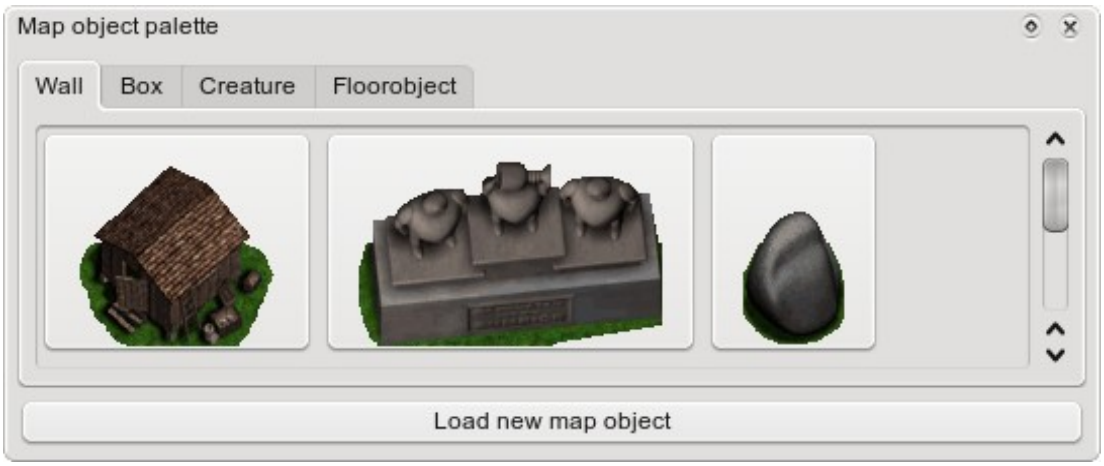


Obrázek 6.21: Menu About

Některé položky menu mají svou klávesovou zkratku. Stisknutím této zkratky vyvoláte stejnou akci, jako vybráním patřičné položky v menu.

Akce	Anglický popisek	Klávesová zkratka
Otevře novou prázdnou mapu	New map	Ctrl + N
Otevře mapu z disku	Open map	Ctrl + O
Uloží mapu na disk	Save map	Ctrl + S
Uloží mapu do jiného souboru	Save map as	Ctrl + Shift + S
Přepne generování objektů	Generate random objects	Ctrl + G

Akce dosažitelné z menu jsme si již představili, zbývá už jen nejdůležitější – umisťování objektů do mapy. Objekty jsou uloženy na disku v souborech. Tyto soubory jsou buď automaticky vyhledány a načteny nebo je pomocí výše uvedeného postupu vyhledáte a načtete ručně přes tlačítko *Load new map object* umístěné pod paletou objektů. Všechny načtené objekty, které lze vložit do mapy jsou rozděleny podle typu do záložek palety. Každý objekt zde má své tlačítko, které může být stisknuté či uvolněné. Objektu, který má stisknuté tlačítko, se říká pracovní objekt. Pokud vybereme jiný objekt jako pracovní, tlačítko posledního pracovního objektu se uvolní. Takto tedy myší přepínáme, jaký objekt chceme mít právě jako pracovní. Pokud nechceme mít žádný jako pracovní, musíme tlačítko pracovního objektu uvolnit kliknutím myši.



Obrázek 6.22: Paleta objektů

Pokud máme nějaký pracovní objekt vybraný, můžeme jej vložit do mapy. Pohybem myši přes scénu mapy se nám zobrazuje pracovní objekt svým obrázkem, pokud lze do mapy na danou pozici vložit a červeným obdélníkem, pokud na danou pozici vložit nelze. Kliknutím levého tlačítka myši objekt do mapy vložíme (pokud to lze). Pracovní objekt zůstává vybraný, můžeme tedy takto navkládat objektů kolik chceme. Odstranění vloženého objektu provedeme dvojklikem. Ve scéně se odstraní vždy objekt na vrchu, dvojklik na konkrétní objekt v detailu políčka odstraní tento objekt.

Pro přesunutí objektu je zde použita metoda *Drag&Drop*. Myší objekt uchopíme (najedeme nad objekt, stiskneme a držíme levé tlačítko). Ve scéně se opět uchopí vrchní objekt, v detailu políčka objekt pod kurzorem myši. Takto uchopený objekt zůstává na své původní pozici, dokud přesun nedokončíme. Hýbáním myší po scéně se nám označuje modře oblast, kam je možno objekt přesunout a červeně oblast, kam to nelze. Když objekt pustíme (uvolníme tlačítko myši) v oblasti, kam lze umístit, objekt se přesune, jinak zůstane na svém původním místě.

V detailu políčka je vidět přesně, jaké objekty na políčku jsou, a lze také s konkrétními objekty pracovat. Když umístíme na políčko více objektů jako na obrázku 6.23, sice v mapě je vidět, že je jich zde více, informace ale není přesná. Když si políčko označíme, zobrazí se jeho detail. Tam jsou umístěné objekty vidět přesně a může s nimi být přesně pracováno. Také v detailu políčka můžeme explicitně zakázat generování. Implicitně může být generování objektů blokováno objekty umístěnými (i generovanými) na políčku.

Obrázek 6.23: Detail políčka

7. Závěr

7.1 Srovnání s podobnými projekty

Hra, která nejspíše první prezentovala postavičku hráče trousícího bomby a bojujícího jimi proti příšerám, byla Dyna Blaster vyvinutá firmou Hudson Soft. Na svou dobu měla perfektní grafiku, zajímavý příběh, kterým mohl procházet jeden hráč. Poskytovala také souboj, ale s běžně vybaveným počítačem (tedy bez herních zařízení) jej mohli hrát proti sobě pouze dva hráči. Všechny mapy měly zdi rozestavěné v pravidelné mřížce, v jednom kole pouze jeden typ zdí, beden a pozadí. Mapy měly dostatečnou tvárnost, takže uživatele nenudilo ani opakované hraní. V každém kole byl pouze jeden bonus. Nebyla zde možnost vybrat si mapu pro souboj.

- − Pravidelnost rozmístění zdí
- − Málo bonusů
- − Jediná mapa pro souboj
- − Malá podpora pro multiplayer
- \+ Tvárnost map (podobně jako Bombič2)
- \+ Více témat pro kola

Obrázek 7.1: Screenshot ze hry Dyna Blaster

Dalším veteránem, který navazoval na Dynu byl X-Blast. Jeho tvůrci se zaměřili pouze na soubojový mód. Portovaný na Windows i unixové systémy se rozšířil po celém světě. Mnoho studentů světových univerzit přispělo k rozvoji této hry [17]. Vzniklo několik editorů map a mnoho grafických témat. Poskytuje jen souboj přes síť.

- − Absence příběhu
- − Nelze hrát pouze na jednom stroji (nutné připojit se přes síť)
- − Žádné příšery
- + Množství map, bonusů, grafických témat
- + Možnost hrát přes síť
- + Nastavitelnost chování prostředí mapy – např. různé podmínky pro explozi bomb (běžné, bomba bouchne při kontaktu s hráčem, bomba bouchne při kontaktu s jinou bombou).

Obrázek 7.2: Screenshot ze hry X-Blast

Poslední hra, kterou bych chtěl srovnat s Bombičem 2, je opět od firmy Hudson Soft. Poskytuje pouze souboj hráčů. Nyní ale až osm hráčů na jednom stroji. Je možné hrát přes síť nebo proti hráčům ovládaným umělou inteligencí. Není portovatelný pro unixové systémy. Mapy jsou statické (náhoda pouze v rozmístění bonusů) a není zde podpora pro větší mapy. Mapa se tedy musí vejít na jednu obrazovku.

- − Absence příběhu
- − Žádné příšery
- − Jen malé mapy
- \+ Množství map, bonusů, grafických témat
- \+ Možnost hrát přes síť i na jednom PC
- \+ Zajímavé objekty mapy (např. teleportující díry, trampolíny, směrovače pro pohyblivé bomby)
- \+ Vtipné (před)úmrtní animace

Obrázek 7.3: Screenshot ze hry Atomic Bomberman

7.2 Budoucnost projektu

V první řadě by bylo dobré dokončit port pro platformu win32. Sice se mi podařilo hru zprovoznit, ale v implementaci výběru souborů jsou platformě závislé chyby (především v práci s cestou), které už jsem nestihl odstranit.

Dále by se do budoucna mohly odstranit nedostatky jako zvuky, absence umělé inteligence pro hráče, více rozvinout animace objektů. Na tato vylepšení bylo myšleno při návrhu architektury, neměl by tedy být příliš velký problém je implementovat.

Větší úspěch by měla hra jistě s možností hrát přes síť, a to jak v kooperativním módu, tak souboje. Pro přidání podpory síťové hry ale bude potřeba změnit podstatným způsobem architekturu aplikace, proto bych toto vylepšení odsunul do vzdálenější budoucnosti.

Jako nedostatky bych také uvedl menu bez obrázků, což nijak nebrání kvalitní hře, uživatele by ale více taková funkce navnadila. V menu také chybí logo hry. Menu tedy patří mezi části, které se příliš nepovedly.

Naopak cíle zmíněné v kapitole 2. se implementovat podařilo a zejména tvárnost map, uživatelský výběr bonusů a rozdělení obrazovky při hře ve více hráčích odlišuje hru od ostatních výše zmíněných.

Myslím si, že editor map je v této formě použitelný, přidat by se dalo ovládání klávesnicí, což může zvýšit pohodlí uživatele při rozsáhlejších úpravách mapy. Jako velké rozšíření bych zmínil tvorbu a modifikaci samotných objektů mapy, která zatím není v programu řešena vůbec.

Literatura

[1] Agar - http://libagar.org/

[2] Atomic Bomberman - http://atomicbomberman.blog.cz/

[3] Bombič na Hippo games - http://hippo.nipax.cz/download.cz.php?id=58

[4] Bombič na Sourceforge.net - http://bombic.sourceforge.net/

[5] Bombič na polském webu - http://www.victorygames.pl/Bombic-pl

[6] C++ Language reference - http://www.cppreference.com/

[7] DOM na wikipedii - http://cs.wikipedia.org/wiki/Document_Object_Model

[8] Dyna Blaster - http://reocities.com/Eureka/4979/

[9] Hall John: Programming Linux Games, Linux Journal Press, 2001

[10] Polský fanklub Bombiče - http://bombic.gry-online.pl/

[11] Qt products - http://qt.nokia.com/products

[12] Qt Reference Documentation - http://doc.trolltech.com/4.3/

[13] SDL Library - http://libsdl.org/

[14] TinyXML - http://www.grinninglizard.com/tinyxml/

[15] What's New in Qt 4.2 - http://doc.trolltech.com/4.2/qt4-2-intro.html

[16] X-Blast - http://xblast.sourceforge.net/

[17] X-Blast Center - http://freexbresse.free.fr/xblast/links.php

Dodatek

Obsah přiloženého CD:

Na CD přikládám elektronické přílohy k této práci. Zejména zdrojové kódy hry a editoru map, elektronickou formu tohoto dokumentu a vygenerovanou referenční dokumentaci (ta může být kdykoli vygenerována pomocí programu *doxygen,* tato byla vygenerována verzí 1.5.6 bez jediného warningu).

- bakalarska-prace_bombic2_karel-fiser_2010.pdf
  – tento text v elektronické podobě
- documentation – dokumentace programů
  - diagrams – grafické diagramy použité v tomto textu
    - classes – diagramy tříd upravené pro použití v textu
    - collision – diagramy kolize objektů
  - game_documentation – referenční dokumentace ke hře
  - map_editor_documentation – referenční dokumentace ke editoru map
- experimental – experimentální produkty (hra pro platformu win32)
- libraries – knihovny SDL a Qt potřebné k přeložení a chodu projektu
- release – poslední vydaná verze, obsahuje zdrojové kódy pro hru i editor map připravené pro kompilaci a instalaci
- screenshots – obrázky ze hry a editoru map
  - editor
    - menus
    - wizards
  - game
    - deathmatch
    - levels
    - menu