Dark Oberon

Programátorská dokumentácia

Obsah

1 Prenositeľnosť

Marián Černý, Martin Košalko, Peter Knut

1.1 Cieľ projektu

Cieľom projektu bolo vytvoriť hru, ktorá bude prenositeľná na bežne používané platformy: Windows a Linux.

1.2 Použité prostriedky

1.2.1 Grafika

Pre zabezpečenie prenositeľnosti sme sa rozhodli grafický výstup riešiť pomocou knižnice OpenGL, ktorá poskytuje rozhranie pre prácu s grafickou kartou. Prenositeľnosť aplikácie z hľadiska vstupu a práce s oknom rieši knižnica GLFW (An OpenGL Framework), ktorá podporuje množstvo platforiem. Knižnica okrem funkcií potrebných pre prácu s oknom ponúka naviac jednotné rozhranie pre prácu s vláknami, zámkami a podmienenými premennými.

1.2.2 Zvuk

Pre prácu so zvukom sme sa rozhodli použiť knižnicu FMOD. Je to síce komerčná knižnica (pre nekomerčné projekty zadarmo), ale zato veľmi kvalitná. Jej nevýhodou je, že je dostupná v binárnej forme a iba pre niektoré platformy. Z tých pre náš projekt zaujímavých sú tieto: Windows 95+, Linux (IA-32) a Apple Macintosh.

1.3 Podpora platforiem

Projekt sme vyvíjali a testovali na týchto platformách: Windows 2000, Windows XP a FreeBSD – všetko IA-32 (32 bitová architektúra). Na platforme Linux sme projekt priamo nevyvíjali, iba občas testovali. Podobnosť s FreeBSD je ale natoľko veľká, že sa nikdy nevyskytli závažnejšie problémy s prenositeľnosťou. Použité prekladače boli: MS Visual Studio 6.0, MS Visual Studio 7.0 a GNU Compiler Collection (GCC) vo verziách 2.95, 3.2 a 3.4.

Okrem platforiem, na ktorých sme projekt vyvíjali, sme testovali tieto platformy: ostatné verzie Windows 95, Windows 98, Windows Millenium, Windows XP AMD-64 (so spustiteľným súborom v 32 bitovej forme), Linux (IA-32).

Projekt by mal byť ďalej preložiteľný na ostatných platformách, ktoré sú podporované knižnicou GLFW. Na Unixových platformách SGI IRIX, SUN Solaris, QNX, NetBSD, OpenBSD, HP-UX a IBM-AIX by nemali byť potrebné žiadne úpravy, prípadne iba minimálne. Podobné to bude aj s platformou MAC OS X, ktorú podporuje dokonca aj knižnica FMOD (knižnica pre prácu so zvukom). Na platforme MS-DOS budú potrebné úpravy funkcií pre prácu s adresármi. Inak by preklad mal byť bezproblémový, keďže aj pre túto platformu je dostupný prekladač GCC. GLFW podporuje ešte platformu Amiga OS, pre ktorú ale neexistuje verzia prekladača GCC, takže tu môžu nastať s prekladom problémy.

Preklad projektu by mal byť možný aj na architektúrach procesorov iných ako 32 bitové little-endian procesory. Problém bude pri sieťovej hre medzi rôznymi architektúrami procesorov, fungovať bude iba s rovnakými architektúrami. K tomu, aby fungovala sieťová hra aj medzi rôznymi architektúrami, je potrebné upraviť iba dve funkcie: funkciu

`TEVENT::Linearise()`, ktorá pripravuje správu na poslanie po sieti a **`TNET_EVENT::Delinearise()`**, ktorá z prijatej správy zo siete správne naplní štruktúru **`TEVENT`**. Nemali sme však k dispozícii iné architektúry pri vývoji a preto tieto funkcie neboli ani implementované prenositeľne. Nebol to ani cieľ nášho projektu.

Sieťová hra na architektúre IA-32 funguje bez problémov aj na rôznych platformách. Testované boli kombinácie Windows, Linux a FreeBSD. Taktiež AMD-64 v 32 bitovom (so spustiteľným súborom v 32 bitovej frome) režime funguje.

2 Správy – jadro hry

Martin Košalko

2.1 Úvod

Projekt Dark Oberon nie je len jediná strategická hra, ale výpočtový stroj pre strategické hry bežiace v reálnom čase, čomu sú prispôsobené aj hlavné programové štruktúry. Keďže je program šablónou pre hry, o jednotkách, prostredí a ani mapách konkrétnej inštancie hry nie je možné predpokladať takmer nič. Jednotky je však možné podľa ich vlastností zaradiť do dopredu pripravených skupín (bojové jednotky, pracanti, zdroje, budovy a továrne). Každá z týchto skupín môže vykonávať rôzne špecifické činnosti (ťažiť materiály, vyrábať jednotky). Niektoré akcie však zostávajú spoločné (útočenie), pričom ale môžu byť vykonávané s menšími rozdielmi (budovy môžu síce strieľať, no nesmú sa pohybovať). Nedá sa však predpokladať, ako rýchlo budú dané akcie prebiehať. Aby samotný výpočtový stroj zaťažoval systém len minimálne vzhľadom na aktivitu jednotiek a nechával tak viac priestoru na náročnejšie operácie ako napríklad vykresľovanie scény, používa prioritnú frontu. Riadenie činností jednotiek je teda riešené systémom správ (events) spracovávaných už spomínanou prioritnou frontou. Navyše sa táto štruktúra veľmi dobre hodí pri návrhu sieťovej komunikácie zabezpečujúcej synchronizáciu jednotiek. Na vzdialený počítač sa zasielajú práve tie správy (events), ktoré sú zaraďované do fronty na lokálnom počítači. Problém synchronizácie činností jednotiek cez sieťové rozhranie sa tým transformoval (zjednodušil) na synchronizáciu lokálnej a vzdialenej fronty správ.

2.2 Typy správ

S pohľadu počítača a fronty správ je možné jednotky hráčov rozdeliť na lokálne a vzdialené. Lokálne jednotky sú jednotky hráčov bežiacich na lokálnom počítači (v princípe to môže byť jeden reálny – ľudský hráč a niekoľko počítačových hráčov) a vzdialené sú jednotky hráčov pripojených cez sieťové rozhranie. Idea systému správ je taká, že o tom, akú akciu vykoná jednotka sa rozhoduje vždy na lokálnom počítači, pričom výsledok rozhodovania sa zasiela aj do front vzdialených počítačov, kde sú tieto „výsledky“ bez všetkého vykonané. Z tohto pohľadu je možné správy kategorizovať ako rozhodovacie (plánovacie) – vkladajú sa len do fronty lokálneho počítača (lokálnej fronty) a akčné, ktoré sa plánujú aj v lokálnej a aj vo vzdialených frontách. Je teda automaticky zabezpečené, že na všetkých počítačoch bude jednotka vykonávať tú istú akciu.

Každá jednotka si v svojej premennej „state“ udržiava svoj aktuálny stav – aktuálne vykonávanú akciu. Na stav jednotky sú naviazané ďalšie rozhodovacie procesy a jej audiovizuálne prejavy – textúry a zvuky. V programe sú akcie, ktoré sa dajú popísať postupnou zmenou stavov jednotky (chodenie = státie, pohyb, otočenie, pohyb..., otočenie, státie), no niekedy je nutné, aby jednotka navonok vykonávala jednu akciu, ale vnútorne musí byť členená na menšie podakcie (strieľanie = nabitie náboja, mierenie, vystrelenie náboja, čakanie, nabíjanie). Z tohto pohľadu sú správy delené na „events“ – správy meniace stav jednotky a „requests“ – žiadosti o vykonanie nejakej akcie. V zdrojových kódoch sú čísla správ definované systematicky: US_XXX – event (US = Unit State), RQ_XXX – request.

2.3 Popis správy – trieda TEVENT

Pre správy bol vytvorený špeciálny objekt: **`TEVENT`**, ktorý je potomkom triedy **`TOOL_ELEMENT`**. Ako už napovedá sám názov predka, všetky aktuálne nepoužité inštancie

objektov **TEVENT** sú umiestnené v zásobníku – „bazéne správ“. Zásobníkom sa síce zvyšujú pamäťové nároky programu, no pri každom vkladaní do fronty správ nie je nutné alokovať pamäť na novú správu a pri každom výbere z fronty dealokovať pamäť, čo je zbytočné a hlavne časovo náročné. V návrhu sme uprednostnili časové nároky pred pamäťovými – inštancia správy sa miesto alokácie a dealokácie vyberie a vloží do zásobníka, čo sú primitívne operácie s pointrami.

Trieda **TEVENT** obsahuje:

- **adresné položky** – jednoznačná identifikácia konkrétnej jednotky,
  - **player_id** – jednoznačný identifikátor hráča (index do poľa hráčov),
  - **unit_id** – jednoznačný identifikátor jednotky v rámci hráča. Keďže jednotky nie sú v poli, ale v niekoľkých spojových zoznamoch (pretože ich počet je veľmi premenlivý) je nutné, aby prevod medzi číslom jednotky a pointrom na jednotku bol rýchly (prevod sa pochopiteľne vykonáva s každou správou vybranou z fronty). Na prevod je využitá hashovacia tabuľka optimalizovaná pre prípad, že identifikátory jednotiek sa postupne zvyšujú. Vychádza sa z predpokladu, že staršie jednotky (s menším ID) budú zničené skôr ako jednotky s vyšším ID. Každá správa musí mať teda konkrétneho adresáta – jednotku.
- **manipulačné položky** – určujú do ktorej fronty správ a na aké miesto má byť (je) správa zaradená,
  - **priority** – určuje, či sa má správa zaradiť do prioritnej fronty. Existencia prioritnej fronty je dôsledkom toho, že v niektorých prípadoch je nutné, aby dva po sebe idúce stavy jednotky neboli prerušiteľné iným stavom,
  - **time_stamp** – časová značka, kedy má byť správa vybraná z fronty (podľa toho je do nej samozrejme vkladaná). Čas sa počíta od spustenia hry a všetky počítače ho majú synchronizovaný. Položka zabezpečuje, aby sa akcie tej istej jednotky vykonali v správnom poradí a podľa možností v správnom okamihu. Z dôvodu nenulového času prenosu správ cez sieťové rozhranie nie je možné vo všetkých prípadoch zabezpečiť rovnaké poradie vykonávania akcií dvoch jednotiek bežiacich na dvoch rôznych počítačoch,
  - **queue_left, queue_right** – väzba správy vo fronte (organizovanej ako obojsmerný spojový zoznam).
- **dátové položky** – informácie pre spracovanie správy jednotkou.
  - **event** – typ správy (US_XXX, RQ_XXX). Podľa typu správy prebieha jej spracovanie,
  - **last_event, request_id** – pomocné položky, ktoré slúžia na upresnenie spracovania správy jednotkou,
  - **simple1 – simple, int1, int2** – parametre správy. Prenášajú sa v nich všetky potrebné parametre na vykonanie akcie (pozície jednotky na mape, cieľ pohybu, cieľ útočenia....).

2.4 Generátory správ

Správy sú do fronty zaraďované v princípe zo štyroch zdrojov:

- **správy generované reálnym hráčom** – reálny hráč ovláda svoje jednotky myšou a klávesnicou a tým generuje správy predávané jednotkám ako informácie o tom, že majú začať vykonávať nejakú akciu (generujú sa teda „štartovacie“ správy),
- **správy generované počítačovým hráčom** – každý počítačový hráč rozhoduje o svojich jednotkách z pohľadu správ totožne s reálnym hráčom. Rozdiel je len v tom, že nepoužíva klávesnicu a myš, ale neurónové siete,
- **správy generované vzdialeným hráčom** – správy zaraďované do fronty správ prostredníctvom sieťového rozhrania. Ako už bolo vysvetlené, správy prichádzajúce zo siete sú „akčné“ a vykonávajú sa bez ohľadu na lokálne podmienky,
- **správy generované jednotkou ako reakcia na prijatú správu** – typicky dostane jednotka od hráča len „štartovaciu“ správu (napríklad presuň sa na pozíciu [X,Y,Z]). Samotné vykonanie tohto príkazu je rozložené do elementárnych akcií, ktoré si už jednotka plánuje sama. Správy, ktoré si jednotka sama generuje sú aj „akčné“ (zasielajú sa súčasne aj do sieťového rozhrania) a aj „plánovacie“ (testuje sa v nich dostupnosť pozícií, existencia iných jednotiek ... a následne plánuje „akčná“ správa).

Z uvedeného je jasné, že je nutné zabezpečiť, aby bola jednotka v danom okamihu „ovládaná“ len jedným generátorom správ a nedostávala tak protichodné správy. Program to rieši tak, že jednotka môže mať v každom okamihu vo fronte správ len jednu správu typu US_XXX (správa meniaca stav jednotky). Znamená to, že ak má jednotka vo fronte správ naplánovaný napríklad výpočet ďalšej pozície pohybu (teda sa momentálne pohybuje medzi dvoma pozíciami – mapelmi) a užívateľ chce začať s danou jednotkou úplne inú akciu (napríklad útok), nová akcia sa naplánuje až v okamihu skončenia predchádzajúcej elementárnej akcie – správe, ktorá je vo fronte, sa zmenia len dátové položky (miesto vo fronte a časová značka ostanú nezmenené). Existujú však aj výnimky. Príkladom je deštrukcia jednotky, ktorá sa musí vykonať okamžite bez ohľadu na správy meniace stav vo fronte jednotky (ktoré budú pochopiteľne ignorované).

Stavy jednotky teda možno z pohľadu prerušiteľnosti rozdeliť na prerušiteľné (regenerácia jednotky), neprerušiteľné (pohyb jednotky medzi dvoma mapelmi) a prioritné (deštrukcia jednotky).

2.5 Spracovanie správ

Celé spracovanie správ beží v samostatnom vlákne. Hlavná funkcia spracovania správ je funkcia `ProcessFunction()`, ktorá v nekonečnom cykle vyberá z fronty správ. V danom okamihu sú vyberané všetky správy, ktorých časová značka je menšia ako aktuálny čas (prípadne žiadne, ak vo fronte nie sú). Správa je potom predaná na spracovanie príslušnej jednotke.

Ako už v úvode do spracovania správ bolo spomenuté, o jednotkách konkrétnej inštancie hry nie je možné predpokladať takmer nič. Avšak podľa akcií, ktoré vykonávajú, ich je možné zaradiť do vopred definovaných skupín (útočné jednotky, pracanti, továrne...). Presne takto sú typy jednotiek reprezentované aj vnútorne. Existuje hierarchia typov jednotiek, ktorá je podrobne popísaná v dokumentácii programových štruktúr, odzrkadľujúca existenciu skupín s rozdielnymi vlastnosťami. Niektoré vlastnosti jednotiek sú spoločné (jednotky majú rozhľad), niektoré sú špeciálne pre daný typ jednotky (ťaženie materiálov) a niektoré sú podobné (strieľanie budov a bojových jednotiek). Táto skutočnosť sa dobre implementuje systémom virtuálnych funkcií typov jednotiek.

2.5.1 Funkcia ProcessEvent

Funkcia **`ProcessEvent(..)`** je virtuálna funkcia typu jednotky, ktorá spracováva správy vyberané z fronty správ. Z pohľadu spracovania správ sa funkcia vnútorne člení na dve časti:

- **spracovanie akčných správ** (ktoré sa bez akýchkoľvek testov vykonajú) – tieto prichádzajú jednak ako reakcie na plánovacie správy jednotiek a jednak zo sieťového rozhrania. Táto časť má dôležitú podčasť – synchronizáciu, kde sa synchronizujú hlavné vlastnosti jednotky – poloha na mape a natočenie jednotky,
- **spracovanie plánovacích** (rozhodovacích) **správ** (testujú sa všetky podmienky a plánuje sa nasledujúci stav jednotky).

Plánovacie správy lokálnej jednotky sa spracovávajú vždy len na lokálnom počítači. Niektoré akčné správy majú pochopiteľne „plánovaciu dohru“ (po pohybe sa musí jednotke naplánovať testovanie nasledujúcej pozície), no tá sa tiež samozrejme vykonáva len pre lokálne jednotky na lokálnom počítači.

Spracovanie akčných správ má dôležitú podčasť – synchronizáciu. Tá zabezpečuje, že jednotky na všetkých počítačoch majú rovnakú pozíciu na mape a rovnaké natočenie. Z dôvodu nenulového času prenosu správy sieťovým rozhraním a neexistencie centrálnej sieťovej autority (servra) sa môže stať, že jednotka zo vzdialeného počítača a lokálna jednotka budú chcieť obsadiť tú istú pozíciu na mape. Program to rieši tak, že existujú dvoje súradnice jednotiek – skutočné a lokálne. Skutočné sú na všetkých počítačoch vždy rovnaké a lokálne sú vždy nastavené tak, aby na lokálnom počítači nedochádzalo ku kolíziám jednotiek. Na nejaký (s vysokou pravdepodobnosťou malý) čas sa môže stať, že hráč vidí jednotku na nesprávnom mieste, no s ďalšou synchronizačnou správou sa situácia s vysokou pravdepodobnosťou napraví a súradnice sa zosynchronizujú správne. Všetky akčné cykly jednotiek (hlavne strieľanie, ktoré vyžaduje interakciu dvoch jednotiek) sú prispôsobené existencii lokálnych a skutočných súradníc jednotky tak, aby lokálny hráč nespozoroval problém.

2.6 Popis akčných cyklov jednotky

Vo všetkých diagramoch, ktoré budú nasledovať, je začiatočný bod – zásah hráča do hry – označený symbolom počítačovej myši a koncový bod – výsledná správa, prípadne stav – podfarbená sivo.

2.6.1 Chodenie

Chodenie je ako celok ukážkové, pretože sa v ňom v ideálnom prípade striedajú akčné a plánovacie správy. Navyše v prípade, že nastane akýkoľvek problém, sú využité aj žiadosti (requesty). Ideálny prípad chodenia nastáva, keď jednotka počas presunu medzi jednotlivými mapelmi nenarazí na žiadnu inú jednotku a ani na pre ňu nepriechodný terén. Vtedy sa postupne striedajú stavy **`US_NEXT_STEP`** a **`US_MOVE`** (pripadne **`US_ROTATING`**, **`US_LANDING`**, **`US_UNLANDING`**). Stav **`US_NEXT_STEP`** je plánovací a testovací. Ako je vidieť z diagramu, testuje sa v ňom dostupnosť a obsadenosť nasledujúcej pozície, správne natočenie jednotky a ešte niekoľko ďalších (pre samotný algoritmus nedôležitých) vecí.

V prípade, že nasledujúca pozícia cesty jednotky je obsadená inou jednotkou, jednotka čaká, či sa neuvoľní cesta a až po desiatich (definované makrom) pokusoch spočíta inú cestu (cyklus **`US_TRY_TO_MOVE`**).

Zaujímavý je tiež spôsob hľadania cesty. Hľadanie cesty pre jednotku je vo všeobecných podmienkach časovo náročné a beží preto v inom (predpripravenom) vlákne. Jednotke sa teda pred začiatkom hľadania cesty správou **US_WAIT_FOR_PATH** oznámi, že má čakať na výsledok výpočtu (každý výpočet ma špeciálne jednoznačné číslo). Jednotka v tomto stave (**US_WAIT_FOR_PATH** samozrejme mení stav jednotky) zotrvá, až kým jej nepríde správa **RQ_PATH_FINDING** (zo správnym identifikátorom) z vlákna, ktoré hľadalo cestu. Táto správa naštartuje ďalší pohyb jednotky.

Všetky akčné správy chodenia – **US_MOVE**, **US_RIGHT(LEFT)_ROTATING**, **US_LANDING**, **US_UNLANDING** sa posielajú aj cez sieťové rozhranie na vzdialené počítače, kde sú vykonávané.

2.6.2 Útočenie

Strieľanie spolu s ťažením materiálov je pravdepodobne najzložitejší cyklus, pretože v ňom musia vzájomne interagovať jednotky dvoch rôznych hráčov, čo kladie zvýšené nároky na sieťové rozhranie. Do cyklu vstupujú tri objekty: útočiaca jednotka, náboj, brániaca sa jednotka.

Úloha útočiacej jednotky je dostať sa „na dostreľ“ k cieľu a vystreliť. V ideálnom prípade sa teda budú striedať testovací stav **US_NEXT_ATTACK** a akčné stavy (zasielané aj cez sieťové rozhranie) **US_ATTACKING**, **RQ_FIRE_OFF** (náboj vyletí z hlavne) a **RQ_FEEDING** (nabíjanie). V menej ideálnom prípade sú prerušované hľadaním cesty a presunom za cieľom.

Náboj vstupuje do cyklu v okamihu jeho vystrelenia, čo je podchytené správou **RQ_FIRE_OFF**. V tomto okamihu sa náboju naplánujú všetku okamihy, kedy mení segment (**RQ_CHANGE_SEGMENT**) a hlavne okamih a miesto dopadu (**RQ_IMPACT**), čím sa životný cyklus náboja končí. Všetky správy, ktoré sú určené náboju, sú v schéme šrafované.

V okamihu dopadu náboja sa na každom lokálnom počítači odoberie lokálnym jednotkám, ktoré boli dopadom zasiahnuté, časť života (spôsob odobratia je presne popísaný v dokumentácii k automatickému strieľaniu) a na vzdialené počítače sa od každej takejto jednotky zašle synchronizačná správa **RQ_SYNC_LIFE**. V prípade, že má jednotka nulový život (čo sa kontroluje len na lokálnom počítači), nasleduje „deštrukcia“ jednotky – postupnosť stavov **US_DYING** (umieranie), **US_ZOMBIE** (rozkladanie) a **US_DELETE** (zrušenie), ktoré sú zasielané aj na vzdialené počítače.

2.6.3 Ťaženie materiálov

Podobne ako strieľanie, kladie ťaženie materiálov zvýšené nároky na sieťové rozhranie, pretože vzájomne interagujú jednotky dvoch rôznych hráčov – schémového hráča a reálneho hráča, ktorý chce ťažiť. Do cyklu vstupujú dva objekty: pracant a zdroj. V schéme sú akcie a stavy pracanta popísané nad vodorovnou čiarou a zdroja pod čiarou.

Hlavnú líniu pracanta tvorí postupnosť správ **US_NEXT_MINE**, **RQ_CAN_MINE**, **US_MINING**. Stav **US_NEXT_MINE** je testovací a plánovací. Testuje sa v ňom existencia zdroja (či ho náhodou niekto nezničil), či je pracant na správnej pozícii...). Ak niečo nie je v poriadku, pracant sa pokúsi nájsť vo svojom okolí iný zdroj a začať ťažiť z neho. Ak je všetko OK, pracant zašle zdroju žiadosť o vyťaženie jednej jednotky materiálu – správa **RQ_CAN_MINE** (?) a sám čaká na odpoveď.

Ak má zdroj voľnú jednotku, odpovedá pracantovi správou **RQ_CAN_MINE** (Y) (s príslušným identifikátorom otázky), inak odpovedá **RQ_CAN_MINE** (N). Ak sa zmení množstvo materiálu v zdroji, synchronizuje sa správou **RQ_SYNC_MAT_AMOUNT**.

Pracant po obdržaní odpovede reaguje buď hľadaním nového zdroja, alebo ďalším cyklom ťaženia. V prípade, že pracant vyťažil maximálne množstvo, ktoré je schopný odniesť, nájde najbližšiu budovu, ktorá akceptuje daný materiál a odnesie ho.

Po vyložení materiálu (pomocou cyklu správ **US_UNLOADING** a **US_NEXT_UNLOADING**) sa pracant automaticky pokúsi ťažiť zo zdroja, z ktorého ťažil naposledy – má ho uložený v premennej source.

2.6.4 Stavanie a opravovanie

Pri stavaní a opravovaní musia spolu interagovať dve jednotky toho istého hráča a informácie o oboch sa musia posielať aj cez sieťové rozhranie. Všetky stavy popísané v diagrame v rámčeku s prerušovaným okrajom sú synchronizačné správy stavanej jednotky a aj pracanta, a sú zasielané cez sieťové rozhranie (na lokálnom počítači sa potrebné akcie vykonávajú v stavoch pracanta). Stavanie a opravovanie sú algoritmicky úplne rovnaké a líšia sa len v detailoch (stav pracanta, ktorý má však rovnaké textúry a zvuky).

Oba stavy začínajú tým, že sa pracant musí dostať k opravovanej jednotke. Ak sa to nepodarí, opravovanie skončí a pracant ostane stáť. Potom už len prebieha cyklus plánovacích (**US_NEXT_REPAIRING**, **US_NEXT_CONSTRUCTING**) a akčných (**US_REPAIRING**, **US_CONSTRUCTING**) správ, ktorými sa postupne pridáva život a „progress“ (len v prípade stavania) opravovanej jednotke. V prípade úspešného dokončenia sa stavanej jednotke zašle správa **US_STAY**.

Všetky akčné správy – **US_CONSTRUCTING**, **US_REPAIRING**, **US_STAY** sa zasielajú aj na vzdialené počítače.

2.6.5 Vyrábanie nových jednotiek

Vyrábanie (cvičenie) nových jednotiek sa deje v továrňach. Továreň má možnosť atakovať cudzie jednotky (v prípade, že tak bola definovaná) a to aj v prípade, že momentálne vyrába nové jednotky. Vyrábanie jednotiek sa teda musí diať „na pozadí“ – nesmie teda meniť stav budovy. To je hlavný dôvod, prečo sa používa správa **RQ_PRODUCING** a nie **US_PRODUCING**. S každou prijatou správou **RQ_PRODUCING** sa testuje, či má hráč dostatok materiálov, jedla a energie, a podľa toho sa vyrábanej jednotke zvýši progress, alebo nie. V každom prípade sa však do fronty správ naplánuje ďalšia správa **RQ_PRODUCING**. Ak je vidieť z diagramu, ostatným hráčom sa zasielajú synchronizačné správy **RQ_SYNC_PROGRESS**.

Zaujímavý je okamih, keď továreň jednotku vyrobí, no jednotka nemôže z budovy vyjsť (napríklad pretože sú pozície obsadené). Vtedy si budova plánuje správy **RQ_TRY_TO_LEAVE** až do vtedy, kým jednotka neopustí budovu.

Po opustení budovy sa jednotke na lokálnom počítači zašle **US_NEXT_STEP** (aby korektne ostala stáť), no na vzdialených počítačoch sa jednotka len v danom okamihu vytvorí (správa **RQ_CREATE_UNIT**).

2.6.6 Regenerácia zdrojov a obnovovanie jednotiek

Regenerácia zdrojov a ozdravovanie jednotiek majú úplne rovnaký princíp, no aplikujú sa na iné typy jednotiek a každé zvyšuje inú vlastnosť. Regenerácia sa aplikuje na obnoviteľné zdroje a zvyšuje aktuálnu kapacitu zdroja, zatiaľ čo obnovovanie sa aplikuje na pohyblivé jednotky (bojové jednotky a pracanti) a zvyšuje život jednotky.

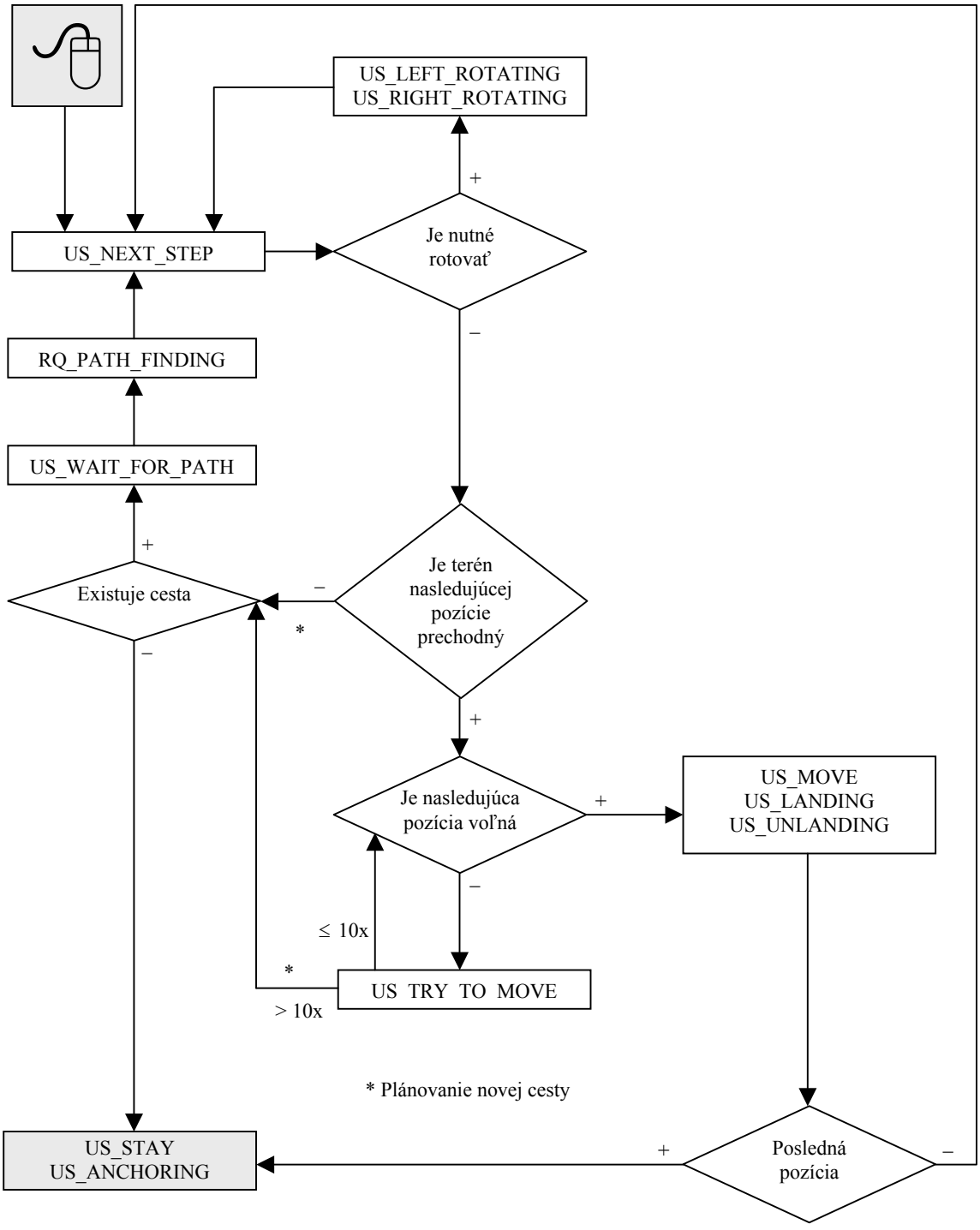
Čo sa týka správ, obnovovanie využíva **US_HEALING** a regenerácia **US_REGENERATING**. Ani jedna z týchto správ sa neposiela cez sieťové rozhranie na vzdialené počítače, kam sa zasielajú synchronizačné správy **RQ_SYNC_LIFE** a **RQ_SYNC_MAT_AMOUNT**. Pre vzdialených hráčov je totiž dôležitý iba dôsledok regenerácie a obnovovania, teda to, že sa zvýšila kapacita zdroja, prípadne život jednotky.

Obnovovanie jednotiek sa vykonáva len v prípade, že jednotka stojí, alebo je pristatá a zároveň má vlastnosť **RAC_HEAL_WHEN_STAY**, prípadne **RAC_HEAL_WHEN_ANCHOR**, ktoré sa zadávajú v konfiguračnom súbore. Časy, keď jednotka stála, sa sčítavajú (plánovanie správ **US_HEALING** je tomu prispôsobené).

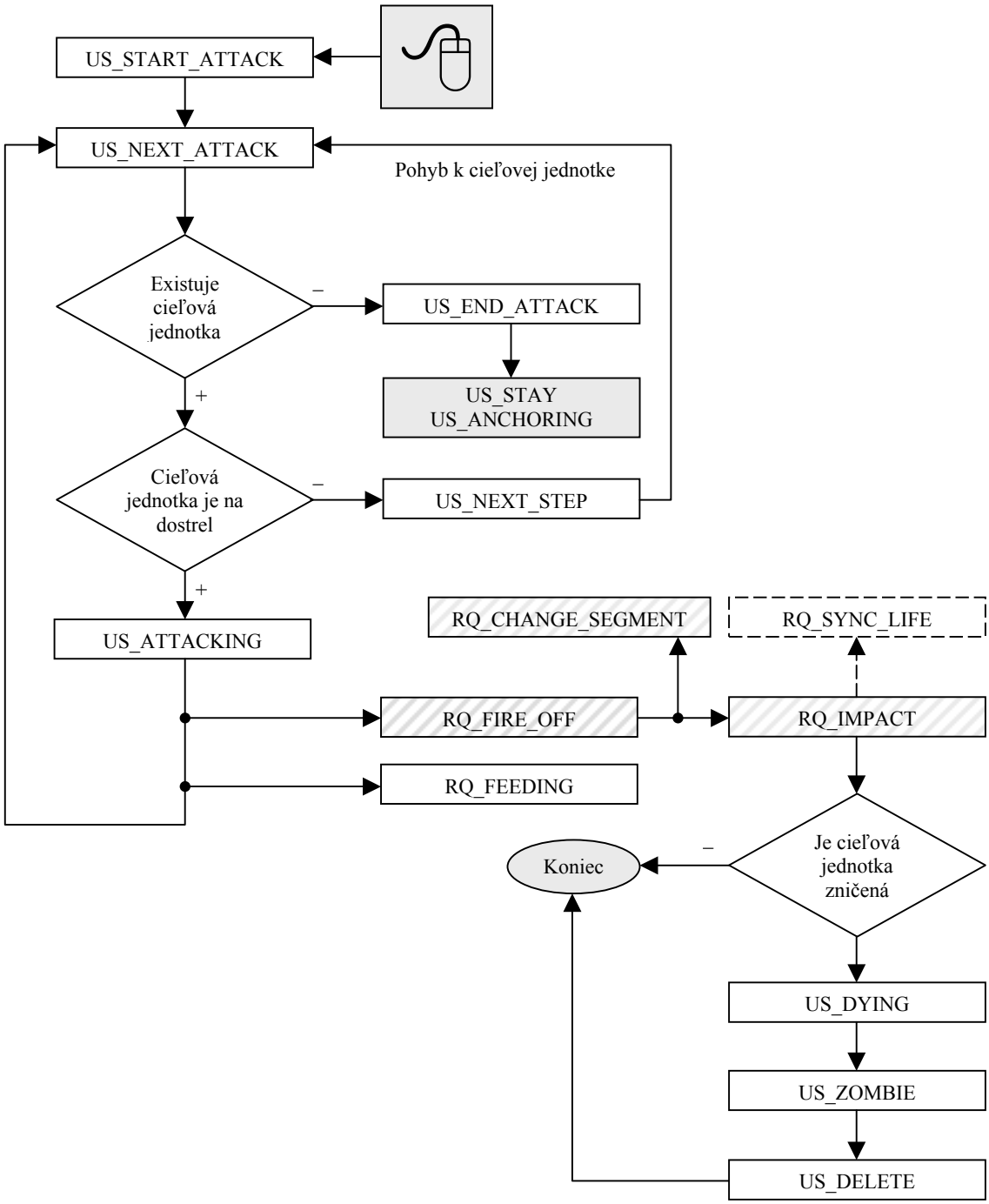
Regenerácia zdrojov prebieha len v prípade, že je zdroj obnoviteľný. Prvá jednotka materiálu (po tom, čo bol zdroj úplne vyťažený) sa pridáva v inom čase, ako všetky ostatné jednotky materiálu. Regenerácia prebieha vždy, keď zdroj nemá plnú kapacitu.

2.7 Schémy akčných cyklov jednotiek

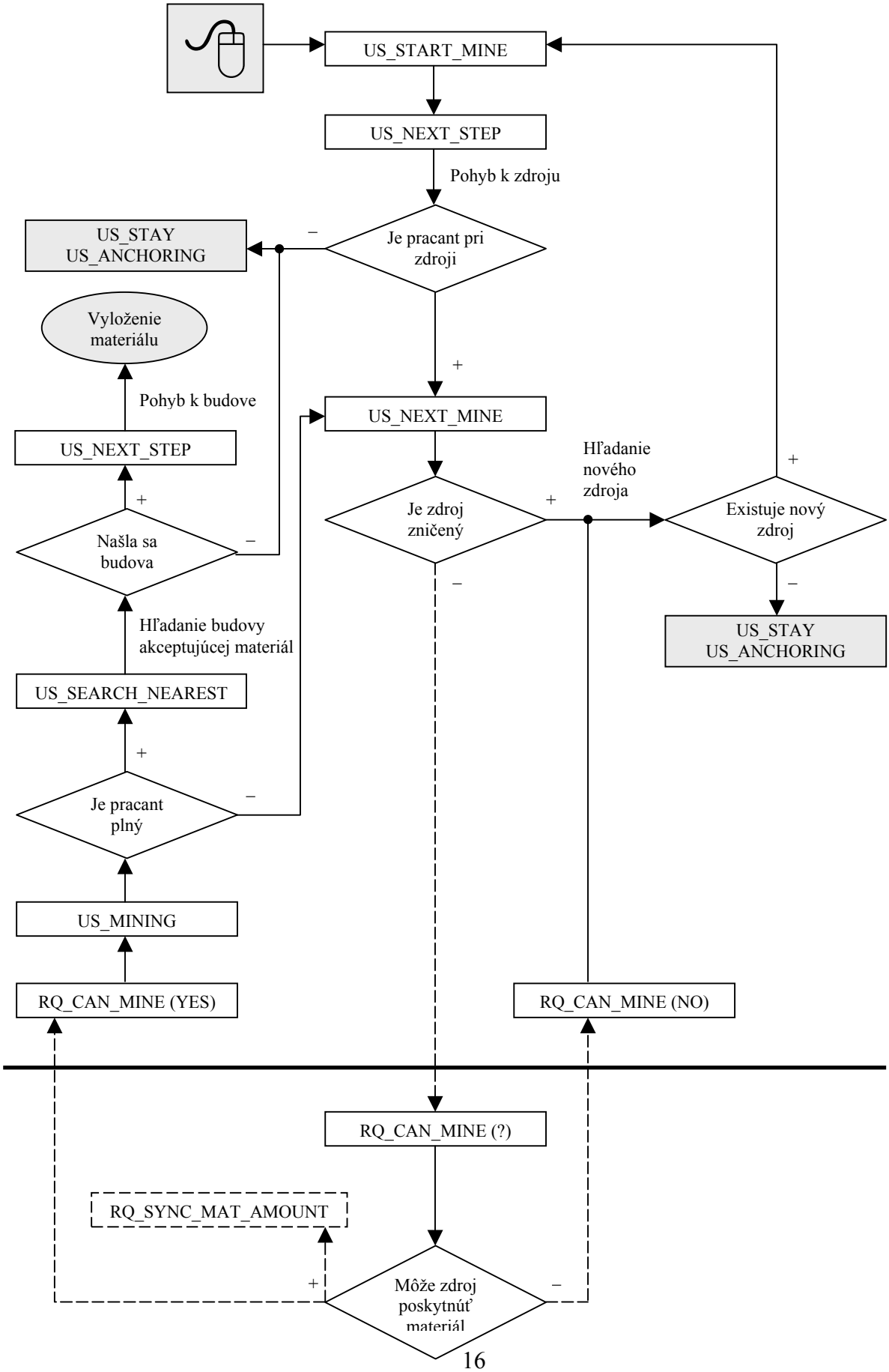
2.7.1 Chodenie



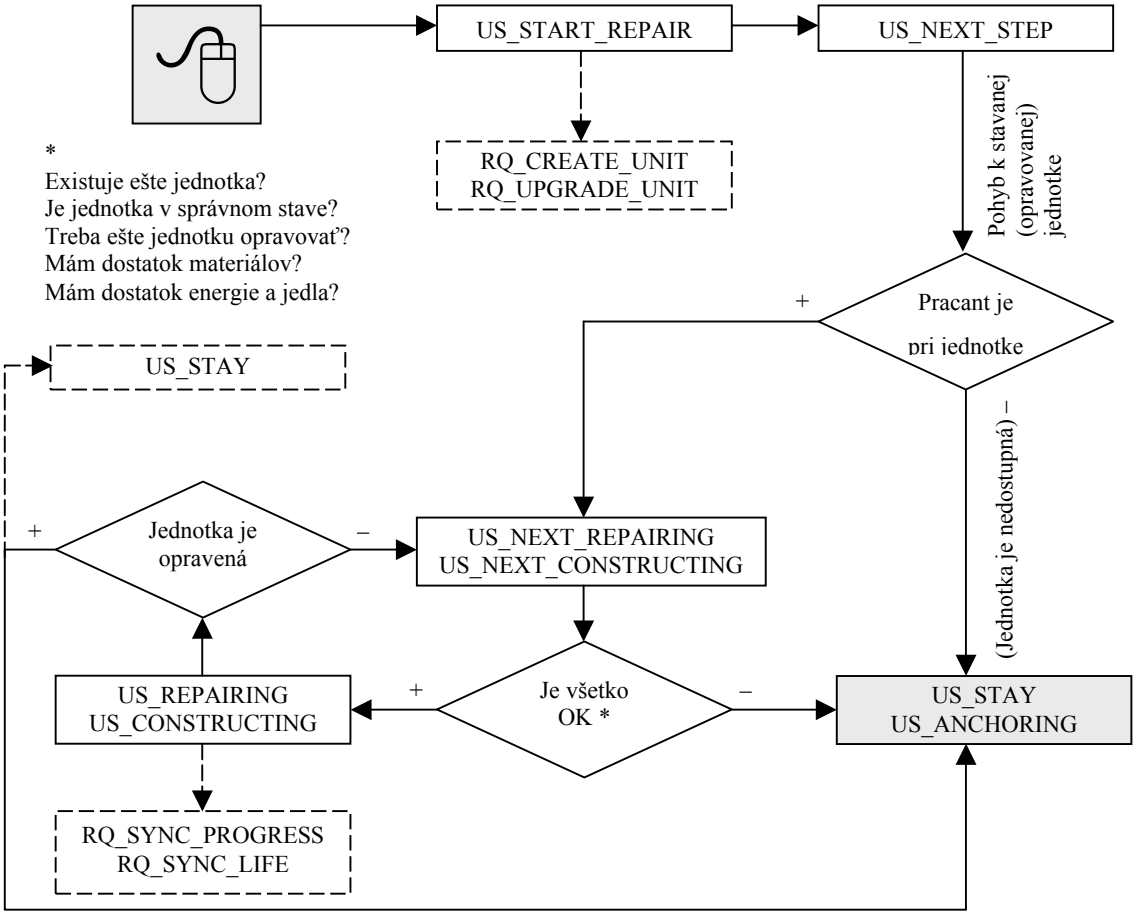
2.7.2 Útočenie



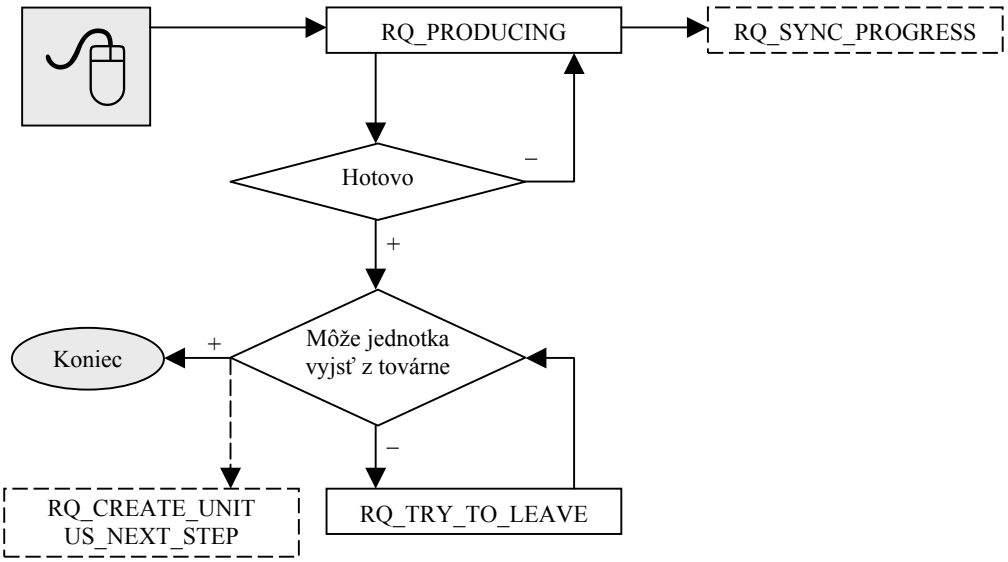
2.7.3 Ťaženie materiálov



2.7.4 Opravovanie a stavanie



2.7.5 Vyrábanie novej jednotky



3 Štruktúry

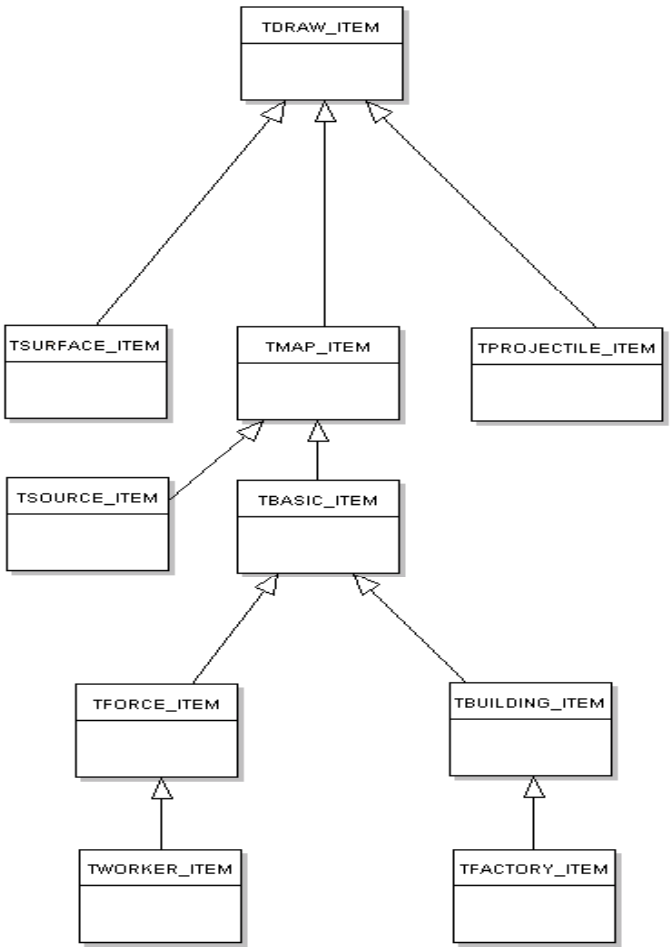
Jiří Krejsa, Peter Knut

3.1 Úvod

V tejto časti dokumentácie budú popísané základné dátové štruktúry a vzťahy medzi nimi.

3.1.1 Štruktúry jednotiek

Z hľadiska jednotiek sú kľúčové dva typy štruktúry. Prvá z nich popisuje vlastnosti druhu jednotky, ktoré sa označujú slovom **ITEM**, a druhá udržuje informácie o konkrétnych inštanciách týchto druhov. Tieto inštancie sú označované slovom **UNIT**. Samozrejme existujú rôzne typy druhov jednotiek, pričom spolu vytvárajú stromovú hierarchiu. Z vyššie uvedeného plynie, že existujú dve stromové hierarchie tried.



Obrázok 1: Hierarchia tried typov druhov jednotiek.

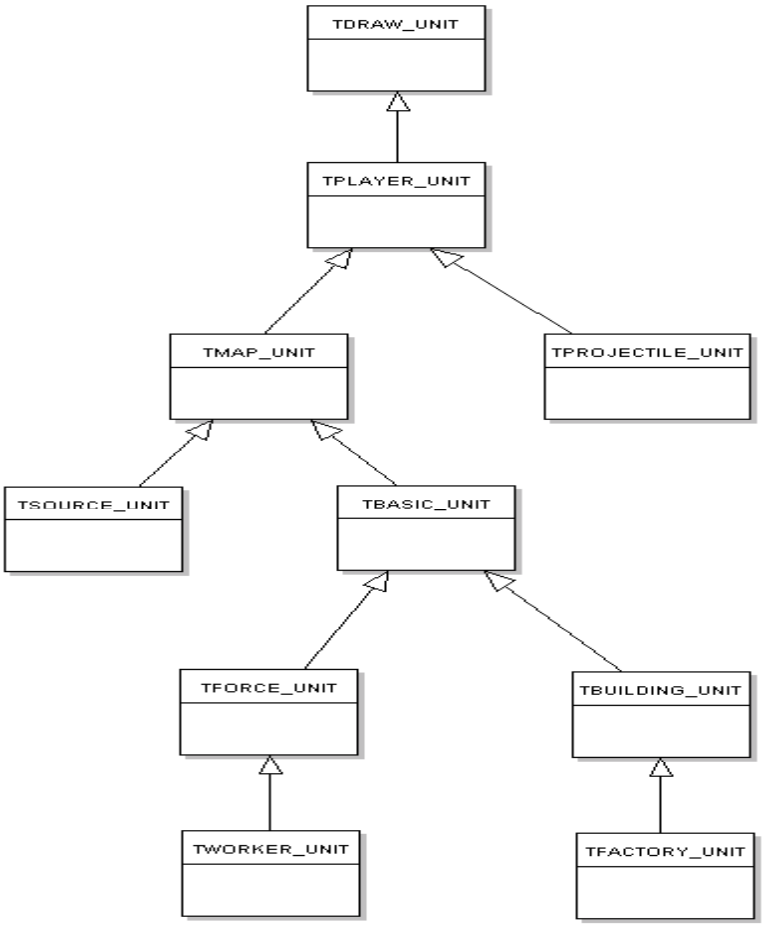
Pozrime sa teraz stručne na význam jednotlivých tried (stromovú štruktúru znázorňuje obrázok č.1):

- **`TDRAW_ITEM`** – trieda je základným kameňom hierarchie, obsahuje informácie potrebné pre vykresľovanie jednotiek,
- **`TPROJECTILE_ITEM`** – trieda uchováva vlastnosti náboja,
- **`TSURFACE_ITEM`** – trieda sa používa pre objekty, ktoré sú súčasťou mapy. Uchováva lokálne zmeny povrchu,
- **`TMAP_ITEM`** – trieda je spoločným predkom všetkých objektov, ktoré môžu stáť na mape,
- **`TSOURCE_ITEM`** – trieda uchováva vlastnosti zdrojov,
- **`TBASIC_ITEM`** – ide o spoločného predka budov a jednotiek. S potomkami tejto triedy môže hráč manipulovať,
- **`TFORCE_ITEM`** – trieda uchováva všetky potrebné informácie o pohyblivých jednotkách. Všetko, s čím môže hráč na mape pohybovať, je inštanciou tejto triedy alebo jej potomkom,

- **TWORKER_ITEM** – rozšírenie pohyblivých jednotiek o umožnenie ťažby materiálov, stavanie budov a opravovanie jednotiek,
- **TBUILDING_ITEM** – uchováva informácie potrebné pre budovy,
- **TFACTORY_ITEM** – rozšírenie budovy o umožnenie výroby pohyblivých jednotiek.

Význam tried je úplne zhodný s významom typov druhov jednotiek. Spomeňme teda v krátkosti len triedu, ktorá je v hierarchii naviac.

- **TPLAYER_UNIT** – naviac obsahuje informácie o vlastnení jednotky hráčom.



Obrázok 2: Hierarchia tried inštancií druhov jednotiek.

3.1.2 Štruktúra mapy

V hre sú používané dva druhy mapy:

- globálna, ktorá obsahuje úplnú informáciu o mape hry,
- lokálna, ktorá sa líši pre každého hráča.

3.1.2.1 Lokálna mapa

V lokálnej mape sú udržiavané tri druhy informácie, ktoré majú spoločnú vlastnosť. Týkajú sa mapy a každý hráč môže mať tieto informácie odlišné od ostatných hráčov. Mapa je

reprezentovaná trojrozmerým poľom štruktúr **TLOC_MAP_FIELD**. V každom políčku lokálnej mapy sú tri dôležité atribúty:

- **state** – vyjadruje viditeľnosť poľa v globálnej mape pre majiteľa lokálnej mapy. Sú rozlišované tri druhy hodnôt:
  - **WLK_UNKNOWN_AREA** (=255) – neznáma oblasť, hráčovi nie sú známe žiadne informácie o políčku,
  - **WLK_WARFOG** (=0) – vyjadruje neznámosť aktuálneho stavu políčka,
  - **ostatné kladné čísla** (>0 a < 255) – znamenajú plnú znalosť aktuálnych informácií o stavu políčka, teda, že je v dohľade aspoň jednej jednotky hráča. Hodnota čísla vyjadruje počet jednotiek hráča, ktoré na dané políčko vidia.
- **terrain_id** – druh terénu. Opäť sú rozlišované tri základné typy:
  - **WLK_UNKNOWN_AREA** (=255) – hráčovi je terén neznámy, táto situácia nastáva len pokiaľ je to zároveň pre hráča neznáma oblasť,
  - **nezáporné čísla menšie ako sto** – podľa hráčových informácií je na políčku s týmto terrain_id terén uvedeného typu, pričom hodnoty typov sa zhodujú s hodnotami používanými v globálnej mape,
  - **nezáporné čísla od sto do dvesto** – podľa hráčových informácií je na políčku s touto hodnotou terén typu „aktuálna hodnota – 100“, na ktorom stojí budova.
- **player_id** – vyjadruje id hráča, ktorý na poli je podľa hráčových informácií. Rozlišuje dva druhy hodnôt:
  - **WLK_EMPTY_FIELD** (=255) – na políčku nestojí žiaden hráč, teda políčko je prázdne,
  - **iná hodnota** – vyjadruje ID hráča, toto ID je rovné indexu v globálnom poli hráčov.

3.1.2.2 Globálna mapa

Obsahuje informácie odpovedajúce skutočnému stavu hry. Na každom počítači pripojenom do hry (pri sieťovej hre) je jej kopia. Tieto kópie sú synchronizované pomocou správ, ktoré si jednotky vymieňajú.

Globálna mapa (**TMAP**) obsahuje predovšetkým pole troch segmentov, pole troch zoznamov jednotiek pre každý segment a štruktúru hospodáriacu s warfogom. Každý segment (**TMAP_SEGMENT**) ďalej obsahuje polia fragmentov, objektov a vrstiev patriacich do tohto segmentu, a dvojrozmerné pole s povrchom segmentu. Povrch segmentu (**TMAP_SURFACE**) zahŕňa pre jedno políčko mapy tri položky:

- identifikátor terénu,
- jednotku stojacu na políčku,
- aktivitu pre každého hráča. Táto aktivita sa využíva pre rozhodovanie počítačového hráča.

Zoznamy jednotiek pre každý segment slúžia pre zoraďovanie a vykresľovanie jednotiek.

3.1.3 Štruktúry hráčov

Existujú dva typy hráčov: obyčajný (ľudský) hráč reprezentovaný triedou **`TPLAYER`** a počítačový hráč reprezentovaný triedou **`TCOMPUTER_PLAYER`**. Každý hráč obsahuje informácie o svojom stave, zoznam jednotiek, ktoré mu patria a nástroje pre hľadanie cesty. Počítačový hráč má naviac informácie a nástroje pre rozhodovanie pomocou neurónových sietí.

4 ALGORITMUS CHODENIA

Valéria Šventová

4.1 Úvod

Chodenie všetkých typov jednotiek zabezpečuje volanie funkcie **`PathFinder(..)`**. Táto funkcia dostane prostredníctvom parametrov zadané informácie o jednotke, ktorá sa má pohnúť (pointer na danú jednotku), informácie o teréne (pointer na lokálnu mapu) a tiež súradnice cieľa. Na výstupe vráti nájdenú cestu a prípadne nový cieľ. Nový cieľ je políčko mapy, ktoré je najbližšie pôvodnému cieľu cesty a je vždy jednotkou dosiahnuteľné (jednotka doňho môže dôjsť). Ako príklad slúži poslanie jednotky do nedostupného cieľa (napríklad obkoleseného skalami, prípadne terénom, ktorý jednotka nemôže nijak obísť, aby sa dostala k cieľu.).

O tom, akým spôsobom je možné panáčika donútiť k pohybu pojednáva užívateľský manuál k hre Dark Oberon (užívateľská dokumentácia).

V projekte sú rozlíšené dva typy, resp. spôsoby chodenia:

- chodenie jednotlivca,
- chodenie skupín.

V oboch prípadoch sa používa modifikovaný A*algorimtus, popísaný v nasledujúcom odstavci.

4.2 A* algoritmus – podstata fungovania

A * algoritmus slúži k nájdeniu najkratšej cesty na mape. Jeho výkonnosť je vysoká a patrí medzi najlepšie vyhľadávacie algoritmy.

Vstupné informácie sú:

- znalosť veľkosti mapy (je nutné, aby mapa bola rozdelená na políčka),
- priestupnosť všetkých jej políčok (políčka rozdelíme na priestupné a nepriestupné),
- súradnice počiatku a cieľa cesty.

4.2.1 Určenie cesty

Pre určenie cesty je potrebné ohodnotiť jednotlivé políčka. Ohodnotenie sa určuje zo vzťahu:

`F = G + H (+D)`

kde:

F – koeficient vzdialenosti

G – vzdialenosť práve ohodnocovaného políčka od štartového políčka

H – vzdialenosť práve ohodnocovaného políčka od cieľového políčka

D – obtiažnosť políčka

Dôvod takéhoto ohodnotenia spočíva v tom, že kratšia cesta sa považuje za lepšiu.

POZNÁMKA: Základný a najjednoduchší A* algoritmus pracuje na mape, ktorej všetky políčka majú rovnakú obtiažnosť, preto uvádzame D vo vyššie uvedenom vzorci len v zátvorkách.

4.2.2 Využívané štruktúry

A* algoritmus využíva 2 základné množiny:

- **Open set** – organizovaná ako halda, obsahuje všetky uzly (políčka mapy), ktoré už boli algoritmom ohodnotené, ale neboli ešte vybraté do konečnej cesty
- **Close set** -- organizovaná ako zotriedený zoznam, obsahuje všetky uzly, ktoré už boli ohodnotené a boli vybraté do cieľovej cesty. Každý prvok tejto množiny bol vybratý z Open set ako prvok s minimálnym ohodnotením.

4.2.3 Práca A* algoritmu

A* algoritmus pracuje nasledovne:

Pridá štartové políčko do Open set (hodnota G je 0). Prevádza sa cyklus cez všetky políčka Open množiny a to až do okamihu, kým nie je prázdna. Pokiaľ algoritmus skončí týmto spôsobom, cesta do pôvodného cieľa nie je nájdená. V spomínanom cykle sa vykonáva:

- Vyber minimálnu položku z Open (v prípade haldy je táto položka v koreni).
- Pridaj vybratú položku do množiny Close
- Pokiaľ je táto položka cieľovým políčkom, algoritmus končí, cesta bola nájdená. V opačnom prípade dôjde k ohodnoteniu všetkých susedných políčok vybraného políčka a k ich pridaniu do množiny Open (resp. k zaradeniu do haldy)

4.3 Modifikácie algoritmu chodenia

V projekte Dark Oberon je A* algoritmus použitý s niekoľkými drobnými zmenami a to najmä za účelom zvýšenia rýchlosti či predchádzaniu možného zacyklenia.

1. V hre je použitý systém 3 horizontálnych segmentov, v ktorých sa dané jednotky môžu pohybovať (pokiaľ majú takúto schopnosť nadefinovanú v konfiguračnom súbore). Bolo žiadúce, aby jednotka mohla prechádzať z jedného segmentu do druhého, ak je takto zostavená cesta pre ňu výhodnejšia. Z tohto dôvodu sa u každého políčka vkladaného do Close množiny neohodnocujú len susedia vo význame 2D (teda tie, ktoré majú od daného políčka súradnice x, y +1 resp. –1), ale susedia vo význame 3D (tretiu súradnicu definuje segment).
2. Ohodnocovanie políčok je komplikovanejšie ako to v základnej verzii A* algoritmu. Dôvodov je niekoľko: použitie warfogu (políčka doposiaľ neobjavené dostávajú špeciálne ohodnotenie), rôzne typy jednotiek (každý typ jednotky má nadefinovaný vlastný rozsah terénu, ktorý je pre jednotku prípustný. Políčka s terénom mimo tento rozsah sú pre jednotku neprístupné, naviac jednotlivé typy jednotiek sa na danom povrchu môžu pohybovať inak rýchlo), atď. Políčka sú v mape rozdelené na:
    a. ***Prípustné pohybové*** – jedná sa o políčka, ktoré nie sú mimo mapu, jednotka na tieto políčka smie vstúpiť, políčko nie je obsadené nepriateľskou jednotkou (v projekte označené ako moveable),
    b. ***Prípustné pristávacie*** - políčka, ktoré nie sú mimo mapu, nie sú obsadené nepriateľskou jednotkou, a daná jednotka ich má v zozname políčok, na ktorých môže pristáť (v projekte označené ako landable),
    c. ***Neprípustné*** - pre jednotku nepovolené.

Nazvime políčko, ktorého susedov sa snažíme ohodnotiť, centrálnym políčkom. Potom je ohodnotenie susedného políčka dané súčtom:

```
C + D + H
```

kde

C – reálna vzdialenosť centrálneho políčka od štartu
D – diagonálna vzdialenosť: závisí na polohe políčka vzhľadom k centrálnemu políčku.
Označme: a – obtiažnosť terénu, s – rýchlosť jednotky v segmente. Potom `D=a*odmocnina(2)/s` pokiaľ sa ohodnocované políčko nachádza v diagonálnom smere od centrálneho políčka, `D=a/s` pokiaľ sa políčko nachádza od centrálneho políčka v priamom smere, `D=2*a/s` pokiaľ sa políčko nachádza vo vertikálnom smere od centrálneho políčka.
H – heuristický odhad vzdialenosti daného políčka od cieľa

POZNÁMKA: pri políčkach, kde sa dá pristáť, sa diagonálna vzdialenosť násobí naviac penalizačnou konštantou, ktorá vyjadruje obtiažnosť pristávacieho manévru. Použiť pristávacie políčko je možné len na posledný krok cesty.

3. Cieľové políčka: V A* algoritme bol cieľ vždy reprezentovaný len jedným políčkom. V projekte Dark Oberon je cieľová oblasť rôzne modifikovaná v závislosti na tom, či užívateľ za cieľ označil inú jednotku, alebo len políčko na mape, na ktorom žiadna jednotka nestojí. V druhom prípade je cieľové políčko len jedno. V prvom prípade cieľová oblasť obsahuje políčka pod jednotkou, na ktorú užívateľ klepol a naviac všetky políčka okolo tejto jednotky do takej vzdialenosti, aby jednotka, pre ktorú sa cesta hľadá, bola v susedstve s jednotkou, na ktorú užívateľ klepol. Jednotka je teda v cieli v prípade, ak svojím ľavým dolným rohom zasahuje do cieľovej oblasti. (Obrázok 3).

Obrázok 3: Cieľová množina pre pohybujúcu sa jednotku rozmerov 3x3

4. Cieľ: v projekte Dark Oberon sa pri hľadaní cesty zapamätáva políčko, ktoré je najbližšie cieľu zo všetkých políčok množiny close. Toto políčko sa použije ako náhradný cieľ v prípade, že pôvodný cieľ je pre pohybujúcu sa jednotku nedosiahnuteľný.
5. Na rozdiel od klasického A* algoritmu sa používa okrem množín Open a Close aj množina Path, ktorá obsahuje všetky políčka, ktoré sa budú vyskytovať vo výslednej ceste, pokiaľ bude nájdenie tejto cesty úspešné.

4.4 Chodenie jednotiek

Ako už bolo spomenuté v odseku 1, dochádza k rozlíšeniu chodenia jednej jednotky a chodenia skupiny jednotiek.

4.4.1 Chodenie jednotlivca

Algoritmus na nájdenie cesty pre jednu jednotku je modifikáciou A* algoritmu. Pokiaľ jednotka má spočítanú nejakú cestu z predchádzajúceho volania funkcie **`PathFinder`**, táto cesta je zničená a je nahradená novou.

4.4.2 Chodenie skupiny jednotiek

Pri hľadaní cesty pre viac označených jednotiek, ktoré tvoria formáciu, sa **`PathFinder`** nevolá pre každú jednotku tejto formácie zvlášť. Celá formácia sa rozdelí na jednu alebo viacero skupín (toto rozdelenie zabezpečuje funkcia **`GetGroup`**). Rozdeľovanie do skupín prebieha tak, že sa vezme vždy prvá jednotka zo zoznamu jednotiek a k nej sa nájdu všetky jednotky rovnakého typu, ktoré od tejto jednotky nemajú príliš veľkú vzdialenosť. Zároveň sa tieto jednotky vyberú zo zoznamu jednotiek, takže pri hľadaní ďalšej skupiny sa s nimi už nepočíta. V jednej takejto skupine sa nájde tzv. leader, t.j. jednotka, pre ktorú existuje cesta do cieľa. Pre každú jednotku skupiny sa následne spočíta jej posunutie vzhľadom k leadrovi a cesta tejto jednotky je rovná ceste leadra plus príslušné posunutie. Pokiaľ by niektorej jednotke zo skupiny pri pokuse o pohyb podľa „posunutej cesty“ stála v ceste prekážka, jednotka si od tejto prekážky do cieľa spočíta cestu už nezávisle na ostatných jednotkách skupiny a to volaním funkcie **`PathFinder`**.

Dôvodom takéhoto rozdeľovania do skupín bola snaha zvýšiť rýchlosť pri počítaní cesty pre skupinu.

4.5 Návratové hodnoty funkcie PathFinder

Ako už bolo popísané na začiatku tejto dokumentácie, funkcia **`PathFinder`** vracia pointer na nájdenú cestu (path), a reálny cieľ, t.j. políčko najbližšie pôvodnému cieľu cesty, ktoré je ale pre jednotku dostupné. Nájdená cesta i reálny cieľ sú parametrami predávanými odkazom. Funkcia **`PathFinder`** vracia typ boolean, ako návratovú hodnotu. **`FALSE`** vráti v prípade, že:

- Vstupné parametre nie sú prípustné (pointer na pohybujúcu sa jednotku je **`NULL`** a pod.)
- Cieľové a štartové políčko je zhodné
- Cestu pre danú jednotku nebolo možné zostaviť

Pokiaľ nenastáva ani jedna zo spomínaných situácií, funkcia **`PathFinder`** vráti **`TRUE`**.

4.6 Chodenie a imlementácia do vlákien

Celý proces chodenia bol za účelom zvýšenia rýchlosti výpočtu cesty implementovaný do vlákien. V hlavnom vlákne beží hra, pre výpočet cesty sa vezme z bazénu vlákien čakajúce vlákno, ktoré dostane za úlohu nájsť cestu pre danú jednotku. Podrobnejší popis celkového

fungovania vlákien je možné nájsť v sekcii o funkcii **ProcessEvent**, prípadne v sekcii o bazéne vlákien.

4.7 Implementačné detaily funkcie PathFinder

Dáta a metódy pre modifikovaný A* algoritmus zapúzdruje trieda **TA_STAR_ALG**. Udržiava pointer na tzv. star_mapu, do ktorej sa zaznamenávajú hodnoty políčok v procese hľadania cesty a ktorá je implementovaná ako pointer na triedu **TA_STAR_MAP**. Táto trieda už predstavuje samotnú mapu v 3D (šírka, výška, segmenty). Mapa sa pre danú jednotku alokuje za nasledujúcich podmienok:

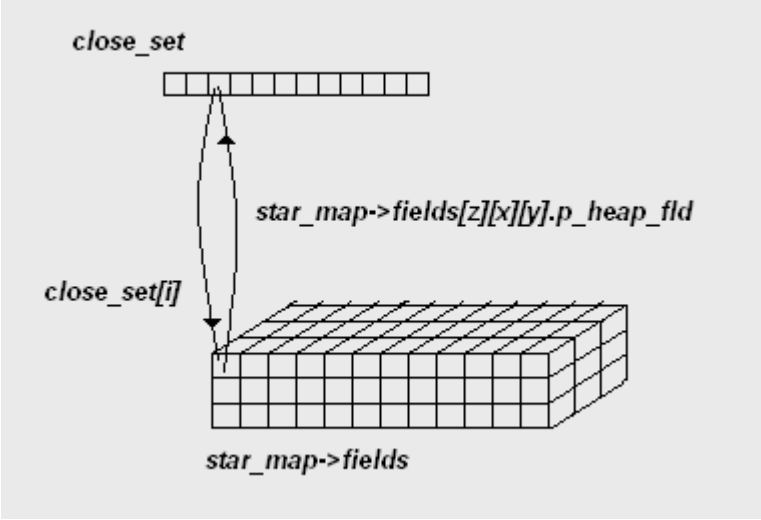
- Dané vlákno prvý krát vykonáva funkciu **PathFinder**, t.j. nemá naalokované políčka star mapy
- Aktuálna mapa nemá odpovedajúce rozmery - v tomto prípade sa táto mapa uvoľní a je naalokovaná mapa nová so žiadanými rozmermi. Táto situácia nastáva, pokiaľ vlákno po prvý krát vykonáva funkciu **PathFinder** v novej hre a už sa pred tým v tomto spustení programu hrala iná hra s inak veľkou mapou

ZHRNUTIE: Z už spomínaného vyplýva, že sa jedná o lenivú implementáciu a naalokované mapy sa prenášajú medzi rôznymi spusteniami hry (pozor, nie aplikácie). Pokiaľ je už naalokovaná mapa prípustná, dôjde iba k jej vyčisteniu (funkcia **ResetMap**).

Trieda **TA_STAR_ALG** si okrem star mapy tiež udržiava informácie o Open a Close množinách (open set, close set, path množina). Množina Open je ogranizovaná ako halda v poli, minimum vyberá funkcia **ExtractMinOpenSet**, vkladanie do Open set je úlohou **InsertToOpenSet**.

Množina Close je poľom s rozmermi *[počet segmetov*širka mapy* výška mapy +1]*, alokuje sa dynamicky v konštruktore triedy **TA_STAR_ALG**. Vyplnené políčko tohto poľa ukazuje pointrom na nejaké políčko v star mape. Zároveň dané políčko star mapy si v premennej **p_heap_fld** udržiava adresu daného políčka v štruktúre, do ktorej bolo políčko zaradené (Open alebo Close set).

Najlepšie sa táto situácia obrazne popíše obrázkom 4 (je zachytená situácia, pri ktorej políčko mapy ukazuje na políčko množiny Close, môže však rovnako ukazovať na políčko množiny Open):

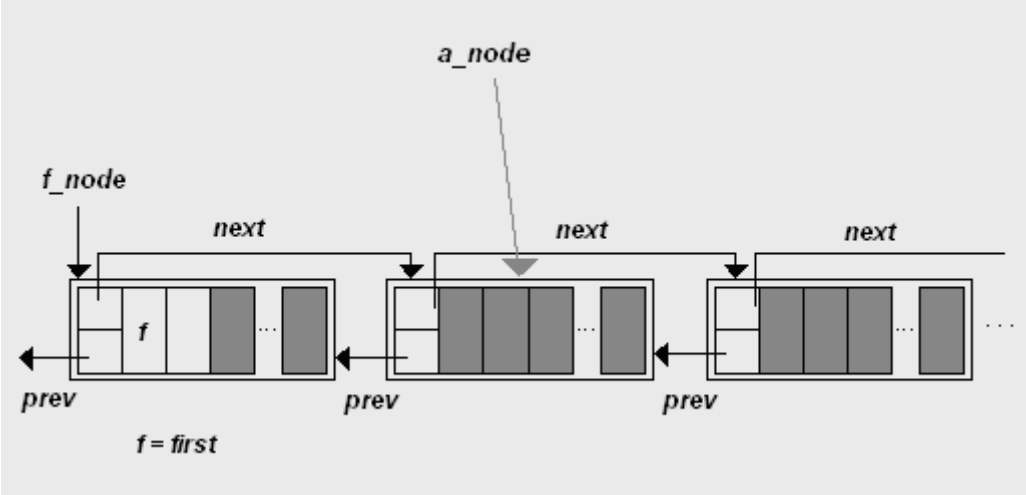


Obrázok 4: Prepojenie star mapy a množiny Close

Takáto štruktúra bola zvolená najmä z dôvodu potreby rýchleho zostavenia výslednej cesty.

Výsledná cesta je zostavená metódou **`CreatePathList`** triedy **`TA_STAR_ALG`**. Táto funkcia dostáva ako parametre cieľ a štart cesty a vráti ako návratovú hodnoty pointer na prvé políčko cesty, prípadne **`NULL`**, pokiaľ cestu nie je možné zostaviť. Na začiatku najprv dôjde k vytvoreniu inštancie triedy **`TPATH_LIST`**, ktorá zapúzdruje metódy a premenné týkajúce sa výslednej cesty.

Štruktúru uchovávajúcu výslednú cestu popisuje obrázok č.5.:



Obrázok 5: PathList

POZNÁMKA: K vzniku inštancie štruktúry **`TPATH_NODE`** dôjde po zaplnení už existujúcich inštancií, t.j. po zaplnení všetkých prvkov poľa **`path_pos`** (na obrázku znázornené tmavo sivou farbou). Novovzniknutá inštancia je zaradená na začiatok zoznamu. Nakoľko zostavovanie cesty prebieha od cieľa smerom k štartu, je pozícia štartového a cieľového políčka, popísaná na obrázku, logická. Rovnako je z procesu vzniku tohto zoznamu zrejmé, že pole **`path_pos`** prvého prvku zoznamu môže zostať čiastočne nevyplnené. Preto z dôvodu rýchlosti bola na uchovanie prvého zaplneného políčka využitá samostatná premenná **`first`**.

Dôvodom vzniku takejto štruktúry bol fakt, že cesta pre danú jednotku si vo veľkej väčšine nevyžiada vznik viac ako jednej inštancie štruktúry **`TPATH_NODE`** (tá obsiahne až 256 krokov cesty), čo je úsporné z hľadiska alokácie. Používanie tejto štruktúry má naviac dobré výsledky z hľadiska rýchlosti.

5 Útok a obrana

Jiří Krejsa

5.1 Úvod

Napriek tomu, že je nutné niektoré časti kódu súvisiace s útočnou fázou hry napísať odlišne pre rôzne typy jednotiek, je možné nájsť spoločnú množinu vlastností a chovaní. Pred ponorením sa do detailov implementácie, pozrime sa na samotnú myšlienku.

Prvým úkonom, ktorý musí útočiaca jednotka previesť, je overenie možnosti zaútočiť, teda napríklad skontrolovať, či je vybraný cieľ v dosahu dostrelu a pod. Po overení dosiahnuteľnosti cieľa nastáva fáza výstrelu, ktorá je reprezentovaná napríklad pohybom ramena katapultu či švihnutím meča. Po skončení výstrelu logicky prichádza vypustenie strely. Tu sa nám udalosti delia na dve dejové línie. Prvou z nich sú činnosti, ktoré nasledujú z hľadiska útočníka, ktorý začína nabíjať. Druhým pohľadom sú akcie, ktoré prevádza samotná strela. Projektil sa okamžikom uvoľnenia stáva nezávislým na svojom pôvodcovi, teda útočník sa stáva iba obyčajnou jednotkou, ktorá môže byť zasiahnutá. Prvou činnosťou, ktorú musí strela previesť, je dosiahnutie dopadovej pozície. Po prekonaní potrebnej vzdialenosti nastáva samotný dopad, ktorý zahŕňa detonáciu, ktorej následkom je zasiahnutie okolia dopadu. Pre všetky objekty v zasiahnutej oblasti prichádza okamžik vyhodnotenia účinku strely. Teda celková sila zásahu sa určí ako sila zbrane, ktorá je z časti náhodne modifikovaná, delená počtom políčok v oblasti dopadu a kvalitou brnenia jednotky, a upravená súčinom schopnosti sa ukryť a náhody na strane obrancu.

5.2 Implementácia

Zaistenie požadovanej funkčnosti je rozdelené do dvoch logických celkov. Prvou oblasťou je prechádzanie jednotlivých fáz útoku a druhou je samotná implementácia. Prvá oblasť je riadená z metódy **`ProcessEvent`**, z ktorej sú volané metódy zaisťujúce samotnú činnosť. Teraz si rozoberieme predovšetkým druhú oblasť, teda hlbšie detaily realizácie. Väčšina zdrojových kódov je umiestnená v súboroch **`dofight.cpp`** a **`dofight.cpp`**.

Každý objekt umiestnený na mape, teda inštancia triedy **`TMAP_UNIT`** (typicky od nej oddedených potomkov), má inštanciu triedy **`TARMAMENT`**, ktorá obsahuje samostatnú útočnú a obrannú časť. Každá inštancia tejto triedy obsahuje obrannú časť, teda triedu **`TDEFENSE`**.

Útočná časť výzbroja, ktorá je reprezentovaná triedou **`TGUN`**, je nepovinná, čím je vyjadrovaná schopnosť či neschopnosť samostatného útočenia na iné jednotky. Jediným účelom je uskladnenie informácie o kvalite brnenia jednotky a o schopnosti vyhnúť sa následkom zásahu.

5.2.1 Test zasiahnuteľnosti

Nosnou časťou útoku je počiatočný test, či je možné na daný cieľ okamžite zaútočiť. Pretože väčšina výpočtov nutná k prevedeniu útoku sa musí previesť už pri overení dostupnosti cieľa, je výsledkom testu nielen hodnota udávajúca, či sa zaútočiť dá, ale tiež informácia, čo je nutné prípadne previesť pre dosiahnutie dostupnosti cieľa. Zároveň sa tiež určí skutočný cieľ strely, ktorý sa od plánovaného líši započítaním nepresnosti zbrane. Všetky tieto informácie sú vrátené v štruktúre **`TATTACK_INFO`** metódou **`IsPossibleAttack`**. Rozoberme si teraz podrobnejšie, čo sa prevádza v metóde testujúcej možnosti útoku. Najprv sa prevádza niekoľko jednoduchých testov - či je cieľ v zasiahnuteľnom segmente, či nemusím najprv ukončiť zakotvenie a či som schopný zaútočiť na tento typ súpera, pričom rozlišujeme tri druhy objektov – budovy, zdroje a ostatné. Ďalšou podmienkou na otestovanie je, či je cieľ

v dostrele, čo platí, pokiaľ je aspoň jedným svojím políčkom v dostrele. Vzdialenosť je tu testovaná pomocou trojrozmernej Pythagorovej vety, kde tretí rozmer predstavujú segmenty.

Špeciálnym prípadom sú jednotky s dostrelom práve jedna, kedy sa všetkých osem okolitých políčok považuje za nachádzajúcich sa v dostrele. Pokiaľ cieľ vyhovie všetkým okolitým podmienkam, je spočítaný najprv ideálny bod zásahu a na jeho základe ešte skutočné miesto dosahu, ktoré je modifikované podľa nepresnosti zbrane.

5.2.2 Útočný cyklus útočníka

V tomto odstavci predstavíme priebeh útoku z pohľadu útočníka a správ, ktoré zasiela či na ktoré reaguje. Tento cyklus je v zásade podobný ostatným akčným cyklom ako je ťaženie, chodenie či stavanie. Správou, ktorá začína útočenie, je **`US_START_ATTACK`**, ktorá korektne ukončí prevádzané akcie, zapamätá si cieľ a zašle správu **`US_NEXT_ATTACK`**. Tá otestuje, či zapamätaný cieľ nie je mŕtvy, umierajúci alebo či prípadne neunikol z dohľadu. Pokiaľ áno, zašle správu **`US_END_ATTACK`**, ktorá sa postará o ukončenie útočenia, a pokiaľ nie, prevedie test zasiahnuteľnosti. Podľa jeho výsledkov a typu útočiacej jednotky (pohyblivá alebo nepohyblivá) buďto začne útočiť, pokúsi sa zaujať lepšiu pozíciu alebo útok ukončí.

Ak dôjde k útoku, jednotka prejde do stavu **`US_ATTACKING`**, počas ktorého sa pripraví inštancia projektilu (**`TPROJECTILE_UNIT`**) a začne sa útočný manéver, čo napríklad znamená pohyb ramena katapultu či švihnutie mečom. Zároveň sa zašle požiadavka na vypustenie projektilu v správnu dobu. Všetky tieto akcie sa prevedú v metóde **`FireOn`**. Aj napriek tomu, že vystrelenie a nabíjanie sú dve rôzne akcie, zostáva jednotka po prevedení oboch v stave **`US_ATTACKING`**. Po uplynutí času vyhradeného na vystrelenie a nabitie prechádza jednotka späť do stavu **`US_NEXT_ATTACK`**.

Ako bolo zmienené v predchádzajúcom odstavci, počas trvania stavu **`US_ATTACKING`** by mala jednotka spracovať požiadavku (**`RQ_FIRE_OFF`**) na vypustenie strely. Počas spracovania je náboj zobrazený a zašlú sa prípadné požiadavky na zmenu segmentu a požiadavky s časom dopadu. Všetky tieto požiadavky sú smerované na inštanciu náboja.

5.2.3 Vyhodnotenie dopadu projektilu

Najprv si stručne preberme súslednosť činností, ktorá je viazaná k projektilu. Po uvoľnení projektilu do priestoru sa začne pohybovať smerom k miestu dopadu. Pokiaľ dráha strely prechádza viacerými segmentami, je ich zmena signalizovaná požiadavkou **`RQ_CHANGE_SEGMENT`**. Okamžik dopadu je signalizovaný požiadavkou **`RQ_IMPACT`**. Reakciou na ňu je vyvolanie vlastnej metódy **`Impact`**, ktorá sa postará o vyriešenie dopadu a uvoľnenie inštancie projektilu z pamäte.

Vo vyššie zmienenej metóde je priechod okolia dopadovej pozície v rozsahu výbuchu, tzv. dopadovej oblasti. Pokiaľ je na niektorom z políčok jednotka, spočíta sa intenzita zásahu, ktorý utrpela a to podľa nasledujúcich vzťahov:

$$DN = PrC * DPr$$

kde:

PrC – náhodné číslo z intervalu [0,4 ; 0,6]

DPr – schopnosť obrany druhu zasiahnutej jednotky

Potom DN je obranné číslo vyjadrujúce šancu na zníženie poškodenia

```
w = (PMin + (PMax – Pmin)*PoC)/CHM
```

kde:

Pmin – minimálna sila útoku zbrane útočníka

Pmax – maximálna sila útoku zbrane útočníka

PoC – náhodné číslo z intervalu [0 ; 1]

CHM – počet políčok v dopadovej oblasti

A výsledok w udáva mieru zranenia podľa kvality rany útočníka

```
W = (w / DA) * (1 – DN)
```

kde:

DA – kvalita brnenia zasiahnutého

DN – obranné číslo spočítané zo vzťahu 1

w – výsledok predchádzajúceho vzťahu

Výsledkom tohto vzťahu je W – skutočné zranenie zasiahnutého

5.2.4 Riešenie útoku bez projektilu

Doposiaľ sme počas popisu hovorili len o útoku na diaľku, teda takom, pri ktorom zranenie spôsobuje projektil vystrelený útočníkom. Nehovorili sme o útoku tvárou v tvár, teda napríklad o útoku mečom alebo päsťou, prípadne obuškom. Zdanlivé ignorovanie ale malo jednoduchý dôvod - i tento nestrelecký útok je vyriešený úplne rovnakým mechanizmom ako bol popísaný vyššie. Jediným rozdielom je to, že „náboje vystrelené“ mečom nie sú viditeľné, pretože nemajú žiadnu textúru a tiež čas ich letu je nulový. Alebo - že čas ich dopadu je zhodný s časom vypustenia projektilu útočníkom.

Hlavnou výhodou zvolenia spomínaného riešenia je absolútna zhodnosť zaobchádzania s útočnou činnosťou, teda i ušetrenie nutnosti vytvárať ďalší kód, ktorý sa až na umiestnenie metód bude plne zhodovať v kľúčových častiach s útočením na diaľku.

5.2.5 Automatická obrana

Samovoľné bránenie vlastných jednotiek voči nepriateľovi úzko súvisí s ich nastavenou agresivitou. Ako bolo spomenuté v užívateľskej časti dokumentácie (konkrétne v užívateľskom manuáli), jednotkám sa dá nastaviť agresivita ich chovania a to v rozsahu štyroch stupňov – stay, guard, offensive guard, agressive guard. Základná myšlienka, že obranca pri nájdení narušiteľa svojho priestoru naň zaútočí, je obrátená, teda narušiteľ hlási obrancovi vniknutie do jeho priestoru. Táto metóda je dosiahnutá pomocou udržiavania pomocných zoznamov pre každé políčko mapy. Zoznamy sú dvoch typov:

- Prvý vyjadruje dohľad jednotiek,
- Druhý vyjadruje dostreľ jednotiek.

Pri zmene polohy teda jednotka zašle správu všetkým jednotkám, ktoré spĺňajú všetky z nasledujúcich podmienok:

- Majú nastavený agresívny mód (agressive guard) a narušiteľ sa ocitol v ich dohľade alebo majú nastavený obranný (offensive guard), prípadne opatrný mód (guard) a v oboch týchto módoch je narušiteľ v ich dostrele,
- Narušiteľ nie je jednotka hyperhráča (hráča vlastniaceho objekty mapy ako prasatá alebo stromy, ktoré nie sú zdrojom materiálu a pod.),
- Obranca nemá za cieľ inú jednotku.

K implementácii zoznamov je použitá trieda **`TMAP_POOLED_LIST`**, ktorá je potomkom triedy **`TPOOLED_LIST`**. Ide o jednoduchý jednosmerný spojový zoznam, ktorý svoje prvky nealokuje, ale využíva dopredu naalokovaných z „bazéne“. Výber jednotiek, ktoré začnú útočiť, prevádza metóda **`AttackEnemy`**, ktorá odlišuje triedu **`TMAP_POOLED_LIST`** od jej predka. Ako bolo spomenuté vyššie, o jej volanie sa nestará obranca, ale narušiteľ.

6 Bazén vlákien

Jiří Krejsa

6.1 Implementácia bazénu vlákien

6.1.1 Použitie bazénu vlákien

Bazén vlákien je využívaný na dvoch miestach. Tým prvým je vyhľadávanie ciest pre jednotky a tým druhým je meranie vzdialenosti medzi jednotkou a zdrojom, jednotkou a budovou a pod. Pozrime sa teraz na samotnú implementáciu podrobnejšie.

6.1.2 Implementačné detaily

Rozhranie pre použitie bazénu vlákien je v súbore **dothreadpool.h**, a pretože je bazén naprogramovaný ako šablóna, je to tiež umiestnenie definícií funkcií. Všetko potrebné je zapúzdrené v triede **TTHREAD_POOL**. Najprv sa stručne pozrime na použitie bazénu. Základnou myšlienkou je vytvorenie rozhrania, ktoré zaistí spoľahlivý prístup k viacerým vláknam bez toho, aby bol užívateľ (v zmysle programátor, ktorý chce šablónu použiť) nútený akokoľvek riešiť problémy s obsadenosťou vlákien a pod. Preto je myšlienka vyberania vlákien posunutá do úzadia a je nahradená ideou kladenia požiadaviek na bazén vlákien. Každá z týchto požiadaviek je pridaná do vstupnej fronty, na ktorej je vytvorená podmienková premenná. Vložením do fronty sa prebudí jedno z pripravených vlákien, ktoré požiadavku odoberie a vyrieši. Vypočítaný výsledok je uložený do výstupnej fronty riešení, odkiaľ sú tieto riešenia po poradí odoberané. Pred samotným ponorením sa do detailov implementácie, spomeňme ešte jednu myšlienku. Pretože k výpočtu sú často nutné pomocné dáta, na ktorých musí byť medzi vláknami zaručené vzájomné vylúčenie, sú tieto dáta dávané k dispozícii samostatne každému jednému vláknu. S trvalým pridaním do triedy, ktorá zapúzdruje pomocné dáta, bola tiež rozšírená možnosť kladenia požiadaviek. Ku každej vznesenej požiadavke sa pridá informácia o metóde, ktorá sa má využiť k vypočítaniu výsledkov. Dá sa zvoliť ktorákoľvek členská metóda pomocnej triedy, ktorá typom odpovedá.

Rozoberme teraz podrobne použité riešenie. Hlavička šablóny je nasledujúca:

```
template <class I, class O, class A> class TTHREAD_POOL {…}
```

Význam troch argumentov je jednoduchý. Prvý parameter I označuje typ, ktorý obsahuje informácie nutné pre prevedenie výpočtu. Platí jednoduché pravidlo, že jedna inštancia odpovedá jednej požiadavke. Druhý parameter O je výstupnou hodnotou, ktorá je vkladaná do fronty odpovedí. Nakoniec tretí parameter A je typ, ktorý zapúzdruje pomocné dáta. Každé vlákno bude pracovať nad vlastnou inštanciou tohto typu.

6.1.2.1 Vytvorenie bazénu

Jediným spôsobom, ako vytvoriť inštanciu bazénu vlákien je použitie jeho statickej metódy **CreateNewThreadPool**.

```
static TTHREAD_POOL<I, O, A>* CreateNewThreadPool(
      int thread_count,
      unsigned int req_queue_size = THP_QUEUE_SIZE,
      bool use_res_queue = false,
      unsigned int res_queue_size = THP_QUEUE_SIZE)
```

Význam prvého parametru nie je potrebné príliš rozoberať. Určuje počet vlákien v bazéne. Druhým parametrom predávame informáciu o očakávanej veľkosti fronty požiadaviek. Pokiaľ neuvedieme hodnotu, bude očakávaná fronta do veľkosti 100 požiadaviek. Údaj je využívaný pre predprípravu potrebného množstva inštancií triedy obaľujúcej požiadavky. Tretí parameter hovorí, či sa bude využívať výstupná fronta odpovedí. Pokiaľ má hodnotu **`false`**, fronta sa nepoužije a výsledky funkcií sú zahadzované. Štvrtý a posledný parameter je veľkosť výstupnej fronty a má význam len vtedy, ak je používaná fronta odpovedí. Metóda vracia pointer na vytvorený bazén vlákien v prípade úspechu a hodnotu **`NULL`**, pokiaľ došlo k chybe.

Samotný proces vytvorenia sa dá popísať veľmi stručne. Vytvorí sa vstupná, a pokiaľ je požadované, tak aj výstupná fronta. Následne sa pripraví podmienková premenná a je vytvorené pole inštancií triedy **`TTHREAD`**, ktorá obsahuje inštanciu pomocných dát. V samotnom závere sú pripravené vlákna. Každé vlákno je spustené v metóde **`FunctionStarter`**, ktorá zabezpečuje vyberanie požiadaviek, spustenie výpočtu a vloženie odpovedi.

6.1.2.2 Pridanie požiadaviek

Ako už bolo zmienené, s bazénom sa pracuje pridávaním požiadaviek. Pre vznesenie požiadavky sa použije rozhranie nasledujúcej metódy:

```
unsigned int AddRequest(I *request,O* (A::*processor)(I*))
```

Pozrime sa na význam jednotlivých parametrov. Prvý parameter je inštancia typu, ktorý obsahuje všetky informácie potrebné k začatiu výpočtu. Druhý parameter je zaujímavejší, je ním pointer na členskú metódu triedy, ktorá obsahuje pomocné dáta výpočtu a ktorej inštanciu si drží každé vlákno. Z hlavičky je vidieť, že dostáva jediný parameter a to pointer na vstupný typ a jej návratovou hodnotou je pointer na výstupný typ.

6.1.2.3 Výber odpovedí

Pokiaľ je využívaná výstupná fronta s odpoveďami, je potrebné výsledky vyberať. K tomu slúži metóda **`O* TakeOutResponse()`**. Metóda nemá žiadne parametre, teda zostáva len podotknúť, že výsledky sú vracané v takom poradí, v akom boli vrátené počítajúcimi metódami.

6.1.2.4 Zničenie bazénu

Všetky úkony potrebné k zničeniu bazénu sú prevedené v deštruktore, ktorý je na rozdiel od konštruktoru verejný. Najprv sú všetky vlákna, a to i tie, ktoré prevádzajú výpočet, ukončené. Následne je uvoľnená podmienková premenná, vstupná a výstupná fronta.

7 Počítačový hráč

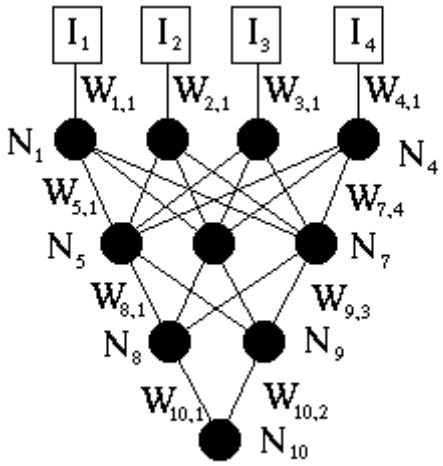
Michal Král

7.1 Úvod

Implementácia chovania počítačového hráča bola veľmi sťažená veľkou všeobecnosťou hry. Na rozdiel od väčšiny strategických hier, v ktorých sú jednotlivé budovy a jednotky pevne dané, v hre DarkOberon si môže užívateľ nadefinovať rôznorodé typy. Pre aspoň čiastočné vysporiadanie sa s týmto problémom sú v niektorých miestach použité neurónové siete namiesto čisto algoritmického riešenia. Zo všeobecného pohľadu funguje rozhodovanie počítačového hráča na základe rozdelenia mapy na menšie časti, ktoré sa všetky pravidelne ohodnocujú (ich dôležitosť, rozhodnutie čo a kde robiť) a s použitím jednotiek blízkych danej ploche sa vykoná akcia.

7.2 Neurónové siete

Väčšina ohodnocovania sa deje za pomoci neurónových sietí. Pre jednoduchšiu implementáciu bol zvolený len jeden typ sietí a pre jednotlivé prípady sa používajú len rôzne počty neurónov v jednotlivých vrstvách. Všetky siete použité v projekte DarkOberon sú perceptronové, majú len jednu vstupnú a dve skryté vrstvy, a len jeden neurón vo výstupnej vrstve. Prepojenie medzi jednotlivými vrstvami je urobené štýlom každý s každým (každý neurón zo vstupnej vrstvy je prepojený s každým neurónom prvej ukrytej úrovne atď.) ako ukazuje obrázok č. 6.



Obrázok 6: Neurónová sieť

Ako aktivačná funkcia je použitá sigmoida, presnejšie jej varianta s nasledujúcim vzorcom

`1/(1+exp{-4*Σ[vstup neuronu]})`

a počty neurónov v jednotlivých vrstvách sú volené skôr tak, aby sa sieť naučila príklady, než podľa nejakého pravidla. Pre učenie sietí bol zvolený algoritmus Back-Propagation, ale v rámci hry v tejto fáze žiadne učenie neprebieha, bol však použitý pri vytváraní sietí pre hru a je implementovaný. Pre tento algoritmus bola vo finálnej verzii použitá varianta s pevným koeficientom skoku (tzv. η) s hodnotou 3. Boli skúšané ako varianty s dynamickou hodnotou, tak i s inými hodnotami, ale táto sa ukázala ako najlepšia a najjednoduchšia. Všetky siete sú uložené v adresári **`Nets`** a ich mená sú pevne dané kódom programu, ich forma uloženia je ale čitateľná. Súbor je členený do riadkov a v nich sú jednotlivé údaje oddelené medzerou. Čísla

označujúce počet neurónov sú zapísané bez desatinnej časti (tvar %i), čísla s desatinným rozvojom sú uložené vo formáte %e, teda s čiarkou ako desatinným oddeľovačom a bez použitia exponentu. Na prvom riadku sú v tomto poradí:

- Počet neurónov v prvej vrstve (vstupné neuróny),
- Počet neurónov v druhej vrstve,
- Počet neurónov v štvrtej vrstve (výstupný neurón),
- Prahová hodnota (spoločná pre všetky neuróny v sieti),
- η (koeficient skoku v algoritme Back –Propagation.

Ďalšie riadky odpovedajú každý jednému neurónu od prvého vstupného, cez druhý vstupný až po posledný výstupný neurón. Na riadku sú potom hodnoty váh vstupov do neurónu, napr. pre sieť s topológiou 4-7-5-1 bude 15. riadok odpovedať 3. neurónu tretieho radu a bude tu 7 hodnôt odpovedajúcich váham medzi 1. neurónom druhého radu až 7. neurónom druhého a týmto 3. neurónom tretieho radu.

Ukážka takéhoto súboru pre sieť z obrázku č.6:

```
 4 3 2 1 -4.316021e+008 3.000000e+000
-9.180000e-002
-6.600000e-003
-3.320000e-002
 0.000000e+000
-6.620000e-002  4.480000e-002 -4.400000e-003 -2.840000e-002
 9.240000e-002 -7.200000e-003  4.100000e-002 -7.100000e-002
-4.380000e-002  6.540000e-002  9.220000e-002 -1.800000e-003
 9.900000e-002  8.840000e-002  6.540000e-002
-1.280000e-002 -2.180000e-002  2.080000e-002
 8.040000e-002 -6.940000e-002
```

Využitie sietí bude vysvetlené neskôr v kontexte celého procesu rozhodovania počítačového hráča.

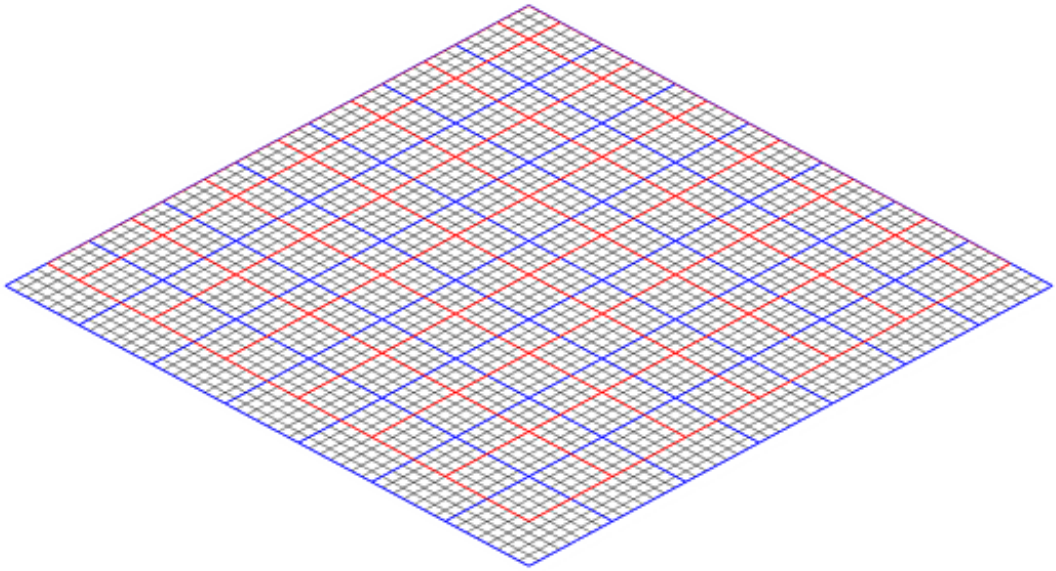
7.3 Ohodnocovanie rás

Ešte pred vlastným začiatkom hry, pri načítaní rás, prebieha pomocou neurónových sietí ohodnotenie jednotlivých jednotiek. Každej jednotke sa najprv ohodnotí sila jej zbrane a potom celková jej užitočnosť. Zvlášť sa ohodnocujú jednotky a zvlášť budovy, ale pre zistenie sily sa používa jedna neurónová sieť. Ako prvá sa použije neurónová sieť zo súboru **network_for_armaments**. Ako vstupy sa jej dajú takmer všetky vlastnosti zbrane jednotky. Keď je ohodnotená zbraň, ohodnocuje sa celá jednotka. Tu sa už používajú dve rôzne neurónové siete – jedna pre pohyblivé jednotky (**network_for_force**) a druhá pre budovy (network_for_building). Pre jednotky sú vstupnými hodnotami maximálna rýchlosť, dohľad, energia, jedlo, maximálny život, príznaky, či sa jednotka má schopnosť stavať, opravovať a ťažiť, sila zbrane a podobne. Pre budovy sú to napríklad ohodnotenie najlepšej jednotky, ktorú môže budova stavať, počet prijímaných materiálov, dohľad, energia, jedlo

a sila zbrane. V oboch prípadoch by mala byť sieť schopná rozlíšiť jednotky líšiace sa v dôležitých parametroch.

7.4 Ohodnocovanie plôch a ohodnocovací cyklus

Na úplnom začiatku vznikla idea získavania globálneho stavu hry z hľadiska počítačového hráča pomocou neurónovej siete majúcej ako vstupy políčka mapy. Zároveň bolo jasné, že sieť majúca toľko vstupov sa bude len veľmi ťažko učiť. Konečná podoba vyzerá tak, že je mapa rozdelená na niekoľko oblastí (presnejšie 98), na nich sa spočítajú charakteristiky pomeru síl, z nich sa spočíta pomocou neurónovej siete aktuálny stav na ploche, zo všetkých týchto stavov a globálnych štatistík (rovnaké ako tie na ploche, ale pre celú mapu dohromady) sa spočíta globálny stav a z globálneho stavu a opäť lokálnych štatistík sa pre každú plochu spočíta jednak dôležitosť danej plochy a jednak potreba vystavať útočnú budovu (napr. vežu), ekonomickú budovu (prijímajúcu suroviny) a tiež potreba vytvoriť pracovné jednotky (vedia ťažiť) na ploche (4 rôzne neurónové siete) a podľa týchto piatich čísel je na ploche vyvolaná nejaká akcia (bližšie popísané v odseku o vlastných pohyboch jednotiek).



Obrázok 7: Ukážka rozloženia plôch premapu 50x50. Čiernou sú zobrazené políčka mapy, modrou plochy prvej vrstvy a červenou plochy druhej vrstvy

Plochy sú vytvorené tak, že sa mapa rozdelí na 7x7 obdĺžnikových plôch veľkosti (šírka mapy/7) x (výška mapy/7). Pokiaľ túto sieť položíme tak, aby jej počiatok bol v počiatku mapy, dostaneme prvú vrstvu plôch. Pokiaľ počiatok tejto siete bude splývať so súradnicami (1/2*šírka plochy) x (1/2*výška plochy), dostaneme druhú vrstvu plôch. Stred plochy prvej vrstvy tak vždy vychádza na rovnaké miesto ako roh štyroch plôch druhej vrstvy a naopak. Každé políčko mapy tak patrí vždy jednej ploche z prvej a jednej ploche z druhej vrstvy, okrem úzkeho pruhu na okraji mapy, ktorý nie je pokrytý druhou vrstvou, aby bol počet plôch v oboch vrstvách rovnaký.

Rozdelenie na 7x7 vychádza z toho, že v prípade „8x8“ je príliš mnoho vstupov pre neurónovú sieť pre globálny stav a pre „6x6“ sú plochy zbytočne veľké (pre typické mapy s rozmermi okolo 200x200). Preto bol zvolený popísaný kompromis.

Neurónová sieť pre zistenie stavu plochy je uložená v súbore **network_for_state** a ako vstupy má lokálne štatistiky, ktoré sú:

- súčet síl zbraní mojich jednotiek v nultom segmente,
- súčet síl zbraní mojich jednotiek v prvom segmente,
- súčet síl zbraní mojich jednotiek v druhom segmente,
- súčet síl zbraní všetkých cudzích jednotiek v nultom segmente,
- súčet síl zbraní všetkých cudzích jednotiek v prvom segmente,
- súčet síl zbraní všetkých cudzích jednotiek v druhom segmente,
- počet mojich útočných jednotiek,
- počet mojich práve stavaných útočných jednotiek (nie je úplne presné, z každej budovy sa počíta vždy len prvá stavaná jednotka),
- počet mojich pracovných jednotiek,
- počet mojich práve stavaných jednotiek,
- počet mojich budov,
- počet mojich práve stavaných budov,
- počet cudzích útočných jednotiek,
- počet cudzích pracovných jednotiek,
- počet cudzích budov,
- moja aktivita,
- cudzia aktivita,
- množstvo suroviny 1,
- množstvo suroviny 2,
- množstvo suroviny 3,
- množstvo suroviny 4.

Výsledok tejto siete je číslo medzi 0 a 0.5, kde 0 znamená moju úplnú prevahu a 0.25 – vyrovnané sily a 0.5 úplnú prevahu súpera, resp. súperov.

Ďalšou použitou sieťou je sieť pre zistenie globálneho stavu hry, tá je uložená v súbore **network_for_global**. Táto sieť má ako vstupy jednak všetkých 98 plôch, potom rovnaké štatistiky, aké boli spomínané, ale v globálnej verzii a nakoniec ešte množstvo jedla, energie (ako dodanej, tak spotrebovanej) a aktuálne množstvo jednotlivých surovín. Dohromady teda 98 +21 + 4 + 4 = 127 vstupov. Výstupom tejto siete je opäť číslo medzi 0 a 0.5, tentokrát však značiace, do akej fáze už hra dospela z pohľadu počítačového hráča. Číslo blízke 0 značí úplný začiatok hry, keď je ešte len potrebné vybudovať základňu a naopak číslo okolo 0.5 by už malo ukazovať na moju úplnú prevahu a len dobíjanie nepriateľov. Pri tomto pohľade síce nie je úplne jasné, čo ktoré číslo znamená, ale na druhú stranu to viac vyhovuje neurónovej sieti, pretože situácie s podobnými vstupmi (napríklad len o jednu budovu viac na mojej strane) si budú i veľmi blízke hodnotou výstupu.

Keď je jasný globálny stav, vracia sa opäť vyhodnocovací cyklus na úroveň jednotlivých plôch. Pomocou piatich rôznych neurónových sietí sa spočíta päť rôznych hodnôt pre každú sieť. Najprv sa spočíta potreba aktívnych alebo útočných budova tejto ploche. Tá má nasledujúce vstupy:

- moja sila v podzemí na ploche,
- moja sila na zemi na ploche,
- moja sila vo vzduchu,
- súperova sila v podzemí,
- súperova sila na zemi,
- súperova sila vo vzduchu,
- počet mojich útočných jednotiek,
- počet mojich práve stavaných útočných jednotiek,
- počet mojich pracovných jednotiek,
- počet mojich práve stavaných pracovných jednotiek,
- počet mojich budov,
- počet mojich práve stavaných budov,
- počet cudzích útočných jednotiek,
- počet cudzích pracovných jednotiek,
- počet cudzích budov,
- moja aktivita na ploche,
- cudzia aktivita na ploche,
- globálny stav,
- lokálny stav na ploche.

Výstupom siete, ktorá je uložená v súbore **`network_for_phaze_a_b`**, je číslo medzi 0 a 0.5, kde 0 označuje nulovú potrebu postaviť útočnú budovu na tejto ploche a 0.5 bezpodmienečnú nutnosť postaviť tu takýto typ budovy.

Veľmi podobná je aj sieť pre ekonomické budovy. Odlišuje sa však vstupmi, ktoré sú v jej prípade:

- počet mojich budov,
- počet mojich práve stavaných budov,
- moja aktivita na ploche,
- cudzia aktivita na ploche,
- množstvo zdrojov 1. typu na ploche,
- množstvo zdrojov 2. typu na ploche,
- množstvo zdrojov 3. typu na ploche,
- množstvo zdrojov 4. typu na ploche,

- množstvo dodávanej energie,
- množstvo spotrebovávanej energie,
- množstvo dodávaného jedla,
- množstvo spotrebovávaného jedla,
- moje množstvo materiálu typu 1,
- moje množstvo materiálu typu 2,
- moje množstvo materiálu typu 3,
- moje množstvo materiálu typu 4,
- globálny stav,
- lokálny stav na ploche.

Táto sieť tvorí obsah súboru **`network_for_phaze_s_b`**, výstupom je číslo 0-0.5 s podobným významom ako v predchádzajúcom prípade.

Tretia a čo do počtu vstupov najväčšia sieť je sieť zo súboru **`network_for_phaze_a_u`**, ktorá slúži na vyhodnotenie potreby výroby útočných jednotiek. Tá svoj výsledok počíta z nasledujúcich štatistík.

- moja sila v podzemí,
- moja sila na zemi,
- moja sila vo vzduchu,
- cudzia sila v podzemí,
- cudzia sila na zemi,
- cudzia sila v podzemí,
- počet mojich útočných jednotiek,
- počet mojich práve stavaných jednotiek,
- počet mojich pracovných jednotiek,
- počet mojich budov,
- stav plochy, ktorá je SZ smerom (v rovnakej vrstve plôch),
- stav plochy, ktorá je S smerom,
- stav plochy, ktorá je SV smerom,
- stav plochy, ktorá je Z smerom,
- stav plochy, ktorá je V smerom,
- stav plochy, ktorá je JZ smerom,
- stav plochy, ktorá je J smerom,
- stav plochy, ktorá je JV smerom,
- moja aktivita,
- cudzia aktivita,

- moje množstvo materiálu typu 1,
- moje množstvo materiálu typu 2,
- moje množstvo materiálu typu 3,
- moje množstvo materiálu typu 4,
- globálny stav,
- lokálny stav na ploche.

Posledná zo sietí z tejto skupiny je uložená v súbore **`network_for_phaze_w`**, má rovnaký účel ako predchádzajúca, ale namiesto útočných jednotiek sa vzťahuje k pracovným jednotkám a na vstupe má tieto hodnoty:

- počet mojich pracovných jednotiek,
- počet mojich práve stavaných pracovných jednotiek,
- počet mojich práve stavaných budov,
- počet cudzích útočných jednotiek,
- moja aktivita,
- cudzia aktivita,
- množstvo zdrojov typu 1 na ploche,
- množstvo zdrojov typu 2 na ploche,
- množstvo zdrojov typu 3 na ploche,
- množstvo zdrojov typu 4 na ploche,
- množstvo dodávanej energie,
- množstvo spotrebovanej energie,
- množstvo dodávaného jedla,
- množstvo spotrebovávaného jedla,
- moje množstvo materiálu typu 1,
- moje množstvo materiálu typu 2,
- moje množstvo materiálu typu 3,
- moje množstvo materiálu typu 4,
- globálny stav,
- lokálny stav na ploche,

Posledné dve majú tiež výstup medzi 0 a 0.5, ale čísla menšie ako 0.25 značia, že tu mám viac príslušných jednotiek, než je potrebné. Hodnota 0.25 označuje „spokojnosť“ s počtom jednotiek a väčšie hodnoty potrebu privolania ďalších.

Trochu stranou od týchto sietí je sieť pre zistenie dôležitosti plochy, súbor **`network_for_importance`**. Jej cieľom je usporiadať plochy tak, ako je nutné vyhovieť ich požiadavkám. Pre ňu sú dôležitejšie skôr počty jednotiek než presne druhy a preto má nasledujúce vstupy:

- priemer mojich síl v jednotlivých segmentoch,
- priemer cudzích síl v jednotlivých segmentoch,
- celkový počet mojich jednotiek (vrátane tých, ktoré sú ešte len stavané),
- celkový počet cudzích jednotiek,
- moja aktivita,
- cudzia aktivita,
- množstvo zdrojov typu 1 na ploche,
- množstvo zdrojov typu 2 na ploche,
- množstvo zdrojov typu 3 na ploche,
- množstvo zdrojov typu 4 na ploche,
- moje množstvo materiálu typu 1,
- moje množstvo materiálu typu 2,
- moje množstvo materiálu typu 3,
- moje množstvo materiálu typu 4,
- globálny stav,
- lokálny stav na ploche.

Výsledné číslo je opäť medzi 0 a 0.5 a jeho interpretácia je jasná: čím vyššie číslo, tým dôležitejšia plocha.

7.5 Vlastné pohyby jednotiek

Tento ohodnocovací cyklus prebieha opakovane dokola pre všetkých počítačových hráčov v aktuálnej hre. Časový interval medzi dvomi ohodnoteniami jednotiek toho istého hráča je pevne daný a ohodnotenia ostatných hráčov sa rozplánujú rovnomerne do toho intervalu. Zároveň s tým prebieha počítanie aktivity jednotlivých políčok. Každé políčko mapy má vlastné počítadla aktivity pre každého hráča, do ktorých sa pravidelne pripočítava 0, pokiaľ na tomto políčku nie je žiadna jednotka príslušného hráča a nenulové číslo v opačnom prípade. Toto číslo ešte závisí na akcii, ktorú jednotka práve vykonáva, pretože napríklad útočenie je považované za dôležitejšiu činnosť ako státie.
Celý priebeh vyhodnocovania z pohľadu počítačového hráča vyzerá tak, že sa najprv prepočítajú všetky hodnoty plôch (najprv všetky štatistiky, potom všetky lokálne stavy plôch, globálny stav a nakoniec potreby jednotlivých druhov jednotiek a dôležitosť každej plochy), potom sa vyriešia všetky potreby (napr. budovy, ktoré nemôžu stavať jednotky, pretože im chýba jedlo, energia alebo suroviny), nasleduje uspokojenie niekoľkých plôch s najväčšou dôležitosťou a nakoniec sa prejdú zostávajúce jednotky, ktoré ešte nedostali žiaden príkaz (pracovníci sa pošlú ťažiť a útočné jednotky sa postavia do obranných pozícií).

8 Grafika

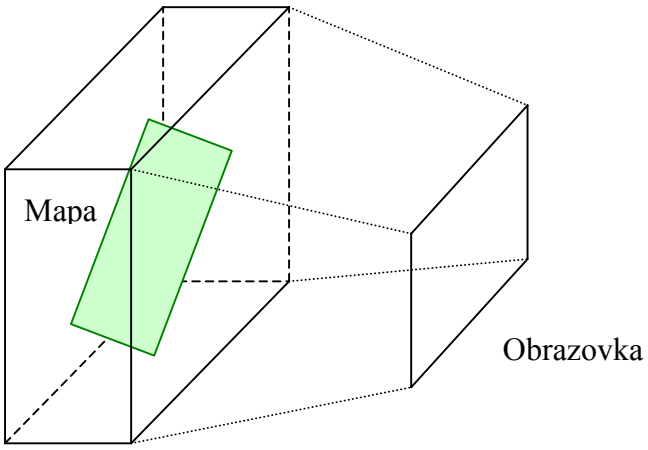
Peter Knut

8.1 Úvod

Hra Dark Oberon používa pre zobrazovanie grafiky technológiu OpenGL. Tá je síce primárne určená pre 3D aplikácie, no poskytuje široké možnosti aj pre programy založené na dvojrozmernej grafike. Výhodou je hardwarová podpora zobrazovania s priehľadnosťou (alfa-blending), projekcií a riešenia viditeľnosti.

8.2 Projekcie

Aplikácia používa v menu i v hre výhradne ortografické zobrazenie, pri ktorom je veľkosť vykreslených polygónov nezávislá na vzdialenosti od pomyslenej kamery. V menu je kváder zobrazenia nastavený podľa veľkosti okna. V hre je vždy použitá rovnaká veľkosť nezávisle na okne. Tým sa dosiahne efekt toho, že pri rôznych rozlíšeniach obrazovky a aplikácii pustenej na celú obrazovku je veľkosť jednotiek rovnaká. Týmto interným rozlíšením je 1024x768. V prípade priblíženia (oddialenia) mapy je kváder zobrazenia zmenšený (zväčšený).



Kváder zobrazenia

8.3 Zobrazenie mapy

Každá mapa obsahuje tri povrchové úrovne – segmenty. Každý segment je reprezentovaný ako dvojrozmerné pole a skladá sa zo štvorcových kusov povrchu – fragmentov. Segmenty sú zobrazované v izometrickom pohľade, takže fragmenty sa javia ako kosoštvorce so šírkou dvakrát väčšou ako je ich výška. Spolu s vhodnými textúrami tento pohľad vyvoláva dojem priestoru.

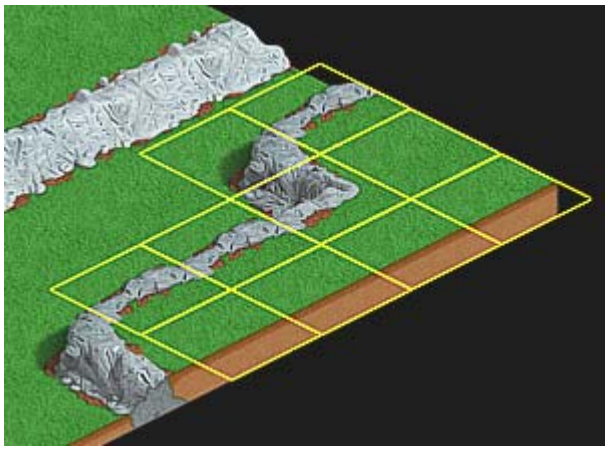
Je možné prepínať medzi zobrazením jedného segmentu a zobrazením všetkých segmentov naraz. Pri tomto kombinovanom móde sú segmenty vykresľované postupne od spodného k vrchnému. Môžu teda obsahovať i priehľadné textúry, cez ktoré bude vidno do nižšieho segmentu. To sa využíva predovšetkým v najvyššom segmente reprezentujúcom oblohu.

V hre sa používa tzv. warfog pre vyznačenie jednak neobjavených miest (čierna nepriehľadná farba) a potom miest, ktoré sú objavené ale žiadna jednotka na ne momentálne nevidí (farbu a intenzitu je možné zvoliť). Warfog nie je nič iné ako textúra generovaná priebežne podľa

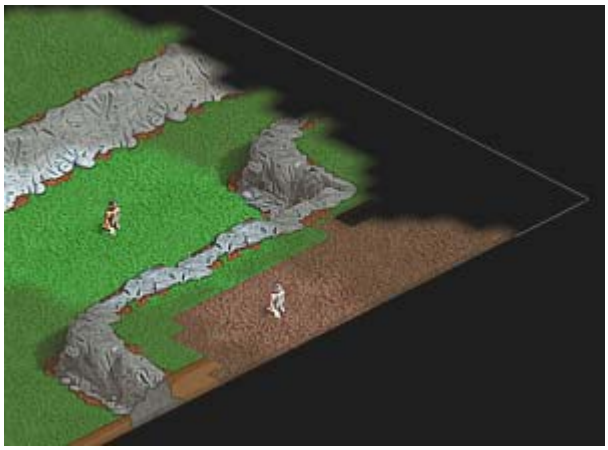
aktuálnej situácie na mape. Jeden bod textúry predstavuje jedno políčko na mape. Pri vykreslení warfogu sa jeho textúra roztiahne na celú mapu. Týmto vznikne efekt rozmazania na prechodoch medzi rôznymi typmi oblastí.

Priebežne sa udržiavajú štyri takéto textúry. Tri sú pre jednotlivé segmenty, jedna pre kombinované zobrazenie všetkých segmentov.

K zaujímavej situácii dochádza pri kombinovanom zobrazení ak je čiastočne objavený nižší segment na miestach kde nie je objavený vyšší segment. V tomto prípade vidno priamo do nižšieho segmentu. Vyšší segment je použitím Z-bufferu orezaný.



Obrázok 9: Rozloženie fragmentov na mape vytvára dojem priestoru.



Obrázok 8: Kombinované zobrazenie segmentov spolu s warfogom.

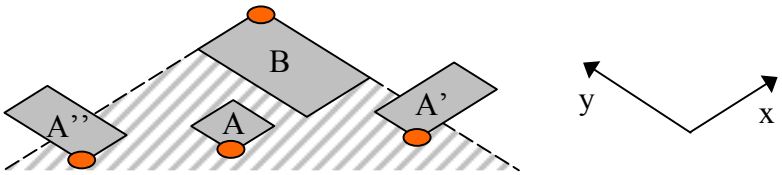
8.4 Jednotky

Jednoty sú vykresľované spolu so segmentom na ktorom stoja. Pre každý segment sa najprv vykreslia fragmenty a následne jednotky. Špeciálny prípad nastáva iba pri zobrazení všetkých segmentov naraz. Predpokladá, že textúry stredného segmentu sú nepriehľadné. Preto sa jednotky zo spodného segmentu vykreslia dvakrát. Jednak spolu so spodným segmentom a potom polopriehľadne spolu so stredným segmentom a to ešte pred vykreslením jednotiek zo stredného segmentu. Tým sa zabezpečí aby ich bolo vždy vidno.

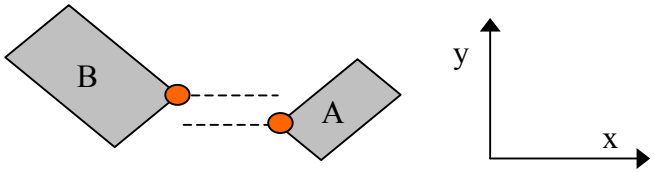
8.4.1 Zoraďovanie jednotiek

Jednotky je potrebné v každom segmente vykresľovať zozadu dopredu aby sa prirodzene prekrývali. Za týmto účelom sa zoznamy jednotiek pre každý segment udržiavajú zoradené. Keďže súradnice jednotiek neodpovedajú zobrazeniu mapy na obrazovke, je podmienka pre zoraďovanie pomerne zložitá. Predpokladom je, že sa jednotky svojimi podstavami neprekrývajú.

Základný prípad nastáva keď sa súradnice jednotky A nachádzajú vo vyznačenej oblasti jednotky B podľa obrázka. Vtedy sa jednotka A nachádza pred jednotkou B.



Takáto podmienka však ešte nestačí. Existujú vzájomné polohy jednotiek u ktorých by sme nevedeli rozhodnúť ktorá jednotka je bližšie. V takýchto prípadoch sa berie do úvahy poloha krajných rohov jednotiek vzhľadom na súradnice obrazovky. Na nasledujúcom obrázku je jednotka A pred jednotkou B.



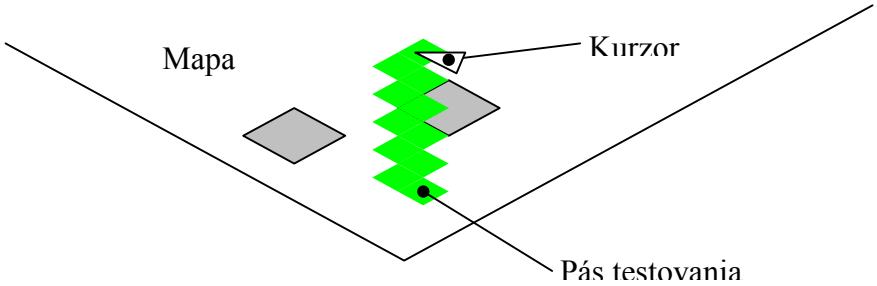
Jednotka môže mať naviac nastavený jeden z príznakov: **`lying_down`** alebo **`flying_up`**. Jednotky s príznakom **`lying_down`** ležia pod jednotkami bez tohto príznaku. Znamená to teda, že pri zoraďovaní sú prednostne radené dozadu. Príznak sa používa napríklad pre trosky zničenej budovy. Naopak **`flying_up`** znamená, že sa jednotky nachádzajú nad ostatnými a sú radené dopredu. Využívajú ho najmä letiace náboje. Je zrejmé, že sa jednotky s jedným z príznakov môžu prekrývať s jednotkami bez tohoto príznaku.

8.4.2 Označovanie jednotiek

Jednotky je možné označiť myšou prostredníctvom ich textúry. Detekcia textúr prebieha tak, že sa na pozadí vykresľujú jednotky ako čierne siluety. Postup je nasledovný:

1. pod kurzor myši sa vykreslí biely štvorec,
2. postupne sa prechádzajú jednotky stojace pod kurzorom zdola nahor (vzhľadom na obrazovku) a kreslia sa ich čierne siluety. Zakaždým sa otestuje, či je bod pod kurzorom stále biely. Ak nie je, jednotka sa označí a testovanie končí.

Na obrázku je znázornený zvislý pás mapy pod kurzorom myši. Jednotky zasahujúce svojou podstavou do tohto pásu prichádzajú do úvahy pre testovanie. Dĺžka pásu je ohraničená.



8.5 Radar

Radar zobrazuje náhľad na odkryté časti mapy spolu s viditeľnými jednotkami. Pre každý segment existuje textúra s celkovým pohľadom na mapu. Tieto textúry sa získavajú automaticky pred spustením hry. Každý segment sa na pozadí vykreslí tak aby presne pasoval do štvorca s veľkosťou 256x256 bodov. Táto oblasť sa uloží ako textúra.

Do radaru sa teda vykreslí vždy textúra celého segmentu, následne warfog rovnako ako na mape a nakoniec viditeľné jednotky v podobe farebných kosoštvorcov. Pri zobrazení všetkých segmentov naraz sa v radare zobrazuje iba stredný segment (jednotky samozrejme zo všetkých segmentov).

9 Grafické rozhranie

Peter Knut

9.1 Úvod

Grafické užívateľské rozhranie (GUI) vzniklo pôvodne ako súčasť hry Dark Oberon, neskôr sa z neho stala samostatná knižnica nezávislá na zvyšku projektu. Ide o nadstavbu knižnice GLWF (An OpenGL Framework) používanej v projekte. Jej základom je jednak práca s animovanými textúrami a potom systém panelov vytvárajúci variabilné grafické rozhranie. Tento systém je zameraný na jednoduchosť a potreby hry, takže neposkytuje úplné možnosti na ktoré sme zvyknutí z bežných aplikácií.

Zdrojový kód knižnice sa nachádza v súboroch glgui.*.

9.2 Textúry a animácie

O textúru načítanú z dátového súboru sa stará strieda **`TGUI_TEXTURE`**. Tu sú uložené všetky jej vlastnosti ako rozmery, počty snímkov, dĺžka animácie a podobne (viď dokumentáciu k Data Editoru). Naviac vie táto trieda metódou **`DrawFrame()`** vykresliť zvolený snímok. Inštancie týchto tried sú typicky usporiadané v poli.

Riadenie samotnej animácie má na starosti trieda **`TGUI_ANIMATION`**, ktorá sa odkazuje na textúru **`TGUI_TEXTURE`**. Jej hlavnými metódami sú:

- **`Update()`** – posúva animáciu v čase,
- **`Draw()`** – vykreslí aktuálny snímok,
- **`Play()`**, **`Pause()`** a **`Stop()`** – slúžia pre spúšťanie a zastavenie animácie,
- **`Hide()`**, **`Show()`** a **`SetVisible()`** – nastavujú viditeľnosť animácie.

9.3 Systém panelov

Samotné grafické rozhranie je tvorené systémom panelov na čele s triedou **`TGUI`**. Po vytvorení rozhrania je možné do neho pridávať rôzne aktívne či pasívne komponenty ako sú panely, tlačidlá, zoznamy, zaškrtávacie políčka a pod. Všetky komponenty sa vedia bez zásahu samé vykresliť s využitím štandartnej palety farieb. V prípade potreby je možné každému prvku nastaviť vlastné farby či priesvitnosť. Viaceré z nich podporujú užívateľské textúry.

Prepojením grafického rozhrania s aplikáciou sú okrem triedy **`TGUI`** hlavne tzv. callback funkcie obsiahnuté vo všetkých komponentoch. Tie umožňujú informovanie aplikácie o udalostiach vznikajúcich v danom komponente (napr. stlačenie alebo pustenie tlačidla myši, stlačenie klávesu, vykreslenie alebo získanie aktivity).

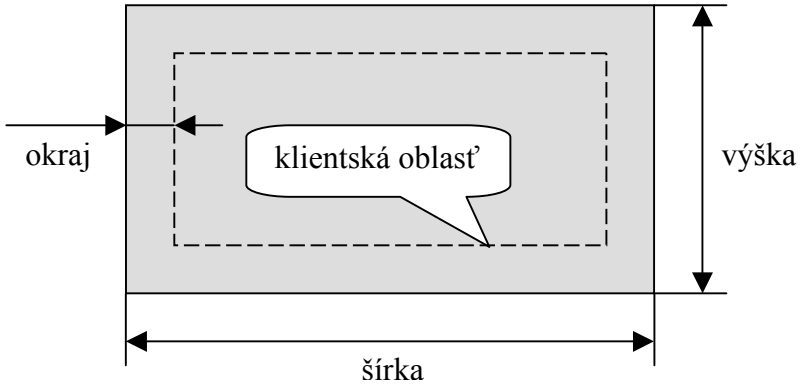
Hierarchia jednotlivých tried je zobrazená v diagrame na konci dokumentu.

9.3.1 Trieda TGUI_BOX

Základná trieda, ktorá tvorí spoločného predka pre všetky komponenty rozhrania. Definuje pravouhlú oblasť s vlastnosťami ako sú: pozícia, veľkosť, okraj a rôzne typy farieb. Pozícia vychádza z ľavého dolného rohu obrazovky. Okraj zmenšuje oblasť na tzv. klientskú oblasť.

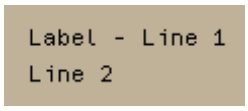
Ďalej je možné nastaviť, či je oblasť viditeľná, použiteľná prípadne aktívna. Naviac môže byť k oblasti priradená rýchla nápoveda (tzv. tooltip), ktorá sa automaticky zobrazí pri podržaní kurzoru myši nad oblasťou.

Je tu možné nastaviť väčšinu callback funkcií. **`TGUI_BOX`** obsahuje voliteľný celočíselný kľúč, pomocou ktorého sa dá v callback funkciách objekt identifikovať.



9.3.2 Trieda TGUI_LABEL

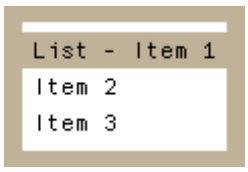
Slúži pre vykreslenie, zadaného textu či animácie. Text môže byť i viacriadkový. V takomto prípade je potrebné zadať reťazec, v ktorom sú jednotlivé riadky oddelené znakom `'\n'`. Užitočnou vlastnosťou je automatická zmena veľkosti komponenty podľa jej obsahu.



Obrázok 10: TGUI_LABEL

9.3.3 Trieda TGUI_LIST

Táto trieda je odvodená od **`TGUI_LABEL`**, ku ktorej pridáva možnosť označovať jednotlivé riadky textu myšou a pristupovať k označenému riadku. Obsahuje tiež callback funkciu pre zmenu označeného riadku.



Obrázok 11: TGUI_LIST

9.3.4 Trieda TGUI_BUTTON

Reprezentuje tlačidlo s popisom. Tlačidlo sa môže nachádzať v rôznych stavoch (stlačené, nestlačené, stlačené aktívne, nestlačené aktívne). Pre každý tento stav je možné definovať vlastnú textúru.

Obrázok 12: TGUI BUTTON

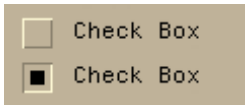
Existujú tri typy tlačidiel, ktoré sa od seba líšia správaním:

- obyčajné tlačidlo – po stlačení sa vráti do pôvodnej polohy (stavu),
- zaškrtávacie tlačidlo – po stlačení ostane v stlačenej polohe. Do pôvodnej polohy sa vráti po opätovnom stlačení.
- skupinové tlačidlo – funguje podobne ako zaškrtávacie tlačidlo. Naviac spolupracuje s ostatnými skupinovými tlačidlami, ktoré sú priradené do tej istej skupiny. V jednej skupine môže byť stlačené najviac jedno tlačidlo. Teda po stlačení skupinového tlačidla sa aktuálne stlačené tlačidlo v jeho skupine vráti automaticky do pôvodnej polohy.

Obsahuje callback funkciu pre zaškrtávanie.

9.3.5 Trieda TGUI_CHECKBOX

Toto je špecializovaná trieda, ktorá funguje presne tak isto ako zaškrtávacie tlačidlo. Jediný rozdiel je vo výzore komponenty.



Obrázok 13: TGUI_CHECK_BOX

9.3.6 Trieda TGUI_EDIT_BOX

Definuje pole do ktorého môže užívateľ napísať jednoriadkový text. Je možné zadať maximálnu dĺžku textu. V aktívnom poli sa vykreslí kurzor označujúci aktuálnu pozíciu. V tomto okamihu komponent prijíma vstup z klávesnice. Kurzor sa dá pomocou štandartných kláves presúvať.

Obrázok 14: TGUI_EDIT_BOX

Obsahuje callback funkciu pre zmenu textu.

9.3.7 Trieda TGUI_SLIDER

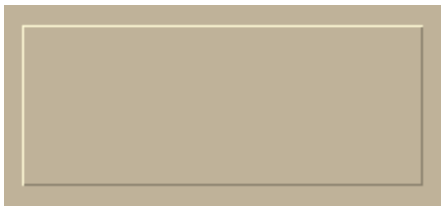
Zastupuje horizontálny, prípadne vertikálny posuvník. Krajné pozície posuvníka sú vždy 0 a 1.

Obrázok 15: TGUI_SLIDER

Obsahuje callback funkciu pre zmenu pozície.

9.3.8 Trieda TGUI_PANEL

Panel je vizualizovaná pravouhlá oblasť, do ktorej je možné pridávať ľubovoľné iné komponenty (okrem **`TGUI`**). Pozície všetkých komponentov v paneli sú relatívne vzhľadom na ľavý dolný roh klientskej oblasti panelu. Naviac sú komponenty orezané tak aby neprečnievali.



Obrázok 16: TGUI_PANEL

Panel podporuje užívateľské textúry.

9.3.9 Trieda TGUI_SCROLL_BOX

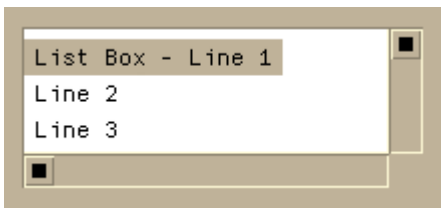
Scroll box je panel, ktorý automaticky obhospodaruje vstavané posuvníky. Ak niektorý z komponentov prečnieva mimo klientskú oblasť, je možné posunúť všetky komponenty a zobraziť prečnievajúci komponent.

Obrázok 17: TGUI_SCROLL_BOX

Posuvníky sa dajú podľa potreby zobraziť či skryť.

9.3.10 Trieda TGUI_LIST_BOX

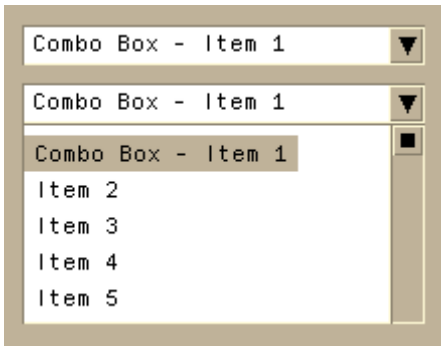
Táto trieda zobrazuje zoznam textových položiek, z ktorých je možné jednu označiť. Obsahuje vstavané posuvníky podobne ako **`TGUI_SCROLL_BOX`**. Pre svoju



Obrázok 18: TGUI_LIST_BOX

prácu využíva triedu **TGUI_LIST**.

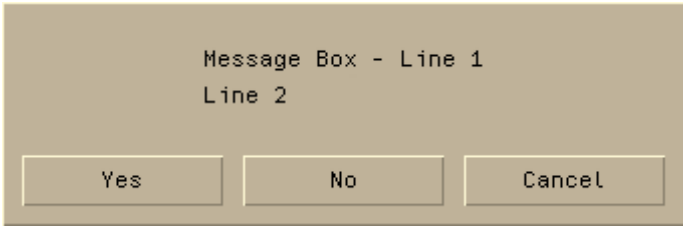
Obsahuje callback funkciu pre zmenu položky.



Obrázok 19: TGUI_COMBO_BOX

9.3.11 Trieda TGUI_COMBO_BOX

Combo box pracuje podobne ako trieda **TGUI_LIST_BOX**, ktorú využíva pre svoju prácu. Nezobrazuje však všetky položky ale iba aktuálne označenú. Na kraji sa nachádza tlačidlo, ktoré po stlačení zobrazí celý zoznam.



Obrázok 20: TGUI_MESSAGE_BOX

9.3.12 Trieda TGUI_MESSAGE_BOX

Ide o špecifickú triedu, ktorá vychádza z triedy **TGUI_PANEL**. Dokáže zobraziť zadaný text vo forme dialógu. Obsahuje viacero vstavaných tlačidiel (OK, Yes, No, Cancel), ktoré je možné navzájom kombinovať. Veľkosť dialógu sa vypočíta automaticky podľa veľkosti použitých tlačidiel a dĺžky textu. Pozícia dialógu je vždy uprostred obrazovky.

9.3.13 Trieda TGUI

TGUI je základnou triedou pre prácu s grafickým užívateľským rozhraním. Predpokladá sa existencia práve jednej inštancie tejto triedy. Ide vlastne o neviditeľný panel roztiahnutý cez celú obrazovku. Základnými funkciami sú:

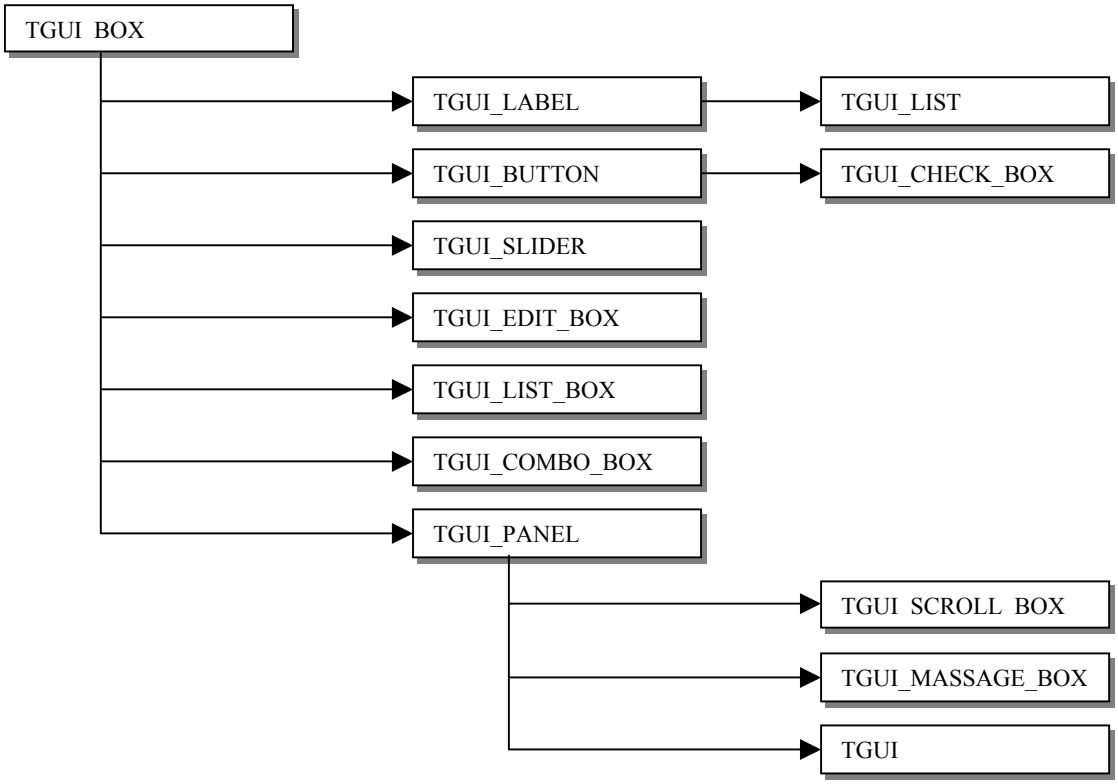
- **Update()** – túto funkciu je potrebné zavolať vždy pred vykreslením. Parametrom je časový posun od posledného volania tejto funkcie,
- **Draw()** – vykreslí všetky viditeľné komponenty,
- **MouseMove()** – potrebné zavolať vždy pri zmene polohy myši,
- **MouseDown()** – potrebné zavolať po každom stlačení tlačidla myši,
- **MouseUp()** – potrebné zavolať po každom pustení tlačidla myši,
- **KeyDown()** – potrebné zavolať po každom stlačení klávesu.

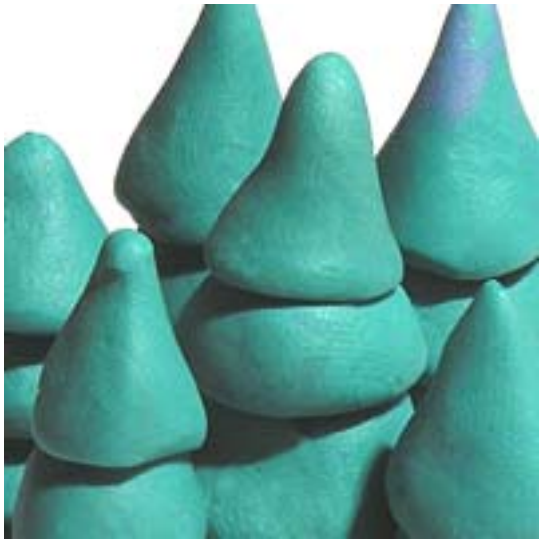
Obrázok 21: Ukážka grafického rozhrania s použitím užívateľských textúr a farieb.

Trieda tiež umožňuje nastaviť typ a farbu písma globálne pre všetky komponenty a zobrazovať dialóg s odkazom (s využitím **`TGUI_MESSAGE_BOX`**).

Podrobný popis všetkých tried a funkcií sa nachádza v zdrojovom kóde programu.

Hierarchia tried grafického rozhrania:





10 Sieť

Marián Černý

10.1 Úvod

Projekt Dark Oberon používa na komunikáciu po sieti protokol TCP/IP s architektúrou peer-to-peer, pričom každý počítač má vytvorené spojenia so všetkými počítačmi, ktoré sa zúčastňujú hry. Trochu odlišné je začatie hry, kedy sa používa architektúra klient-server. Počítač, ktorý vytvára hru, sa nazýva **leader**. Ostatné počítače, ktoré sa k nemu pripájajú, sa nazývajú **follower**.

Komunikácia prebieha vo forme správ, ktoré majú svoju hlavičku a vlastný obsah (data). Každá správa má príjemcu. Príjemca je číselný identifikátor hráča, prípadne špeciálna hodnota, ktorá znamená všetkých hráčov. Správa určená všetkým hráčom je na každý počítač doručená iba raz, aj keď na niektorom počítači beží naraz viacero hráčov (na leadrovi beží hyper hráč, ľudský hráč a prípadne ešte počítačoví hráči).

Všetky triedy a funkcie, ktoré majú na starosti sieťovú komunikáciu, sú implementované v súboroch **`donet.h`**, **`dohost.h`**, **`dofollower.h`**, **`doleader.h`** a príslušných **`.cpp`** súboroch.

10.2 Základné triedy pre prácu so sieťou

Základne triedy pre prácu so sieťou sú implementované v súboroch **`donet.h`** a **`donet.cpp`**. Medzi tieto triedy patria triedy:

- **`TNET_MESSAGE`** – reprezentujúca správu posielanú alebo prijatú zo siete,
- **`TNET_MESSAGE_QUEUE`** – fronta správ, do ktorej je možné vkladať a z ktorej je možné vyberať správy,
- **`TNET_LISTENER`** - trieda čakajúca na prichádzajúce spojenia a vkladajúca sieťové správy do fronty prichádzajúcich správ,
- **`TNET_TALKER`** - trieda, ktorá odosiela sieťové správy z fronty odchádzajúcich správ jednotlivým príjemcom,
- **`TNET_DISPATCHER`** - doručovateľ prichádzajúcich správ.

10.2.1 Trieda TNET_MESSAGE

Trieda **`TNET_MESSAGE`** reprezentuje sieťovú správu. Sieťová správa je postupnosť bajtov skladajúca sa z hlavičky a tela správy (dát). Hlavička správy obsahuje veľkosť správy, typ správy, podtyp správy a príjemcu správy. Potom nasledujú samotné dáta.

veľkosť správy	typ správy	podtyp správy	príjemca správy	telo správy (dáta)

10.2.2 Trieda TNET_MESSAGE_QUEUE

Sieťové správy sú pred odoslaním a pri prijatí zo siete ukladané do fronty. Táto fronta je reprezentovaná objektom triedy **TNET_MESSAGE_QUEUE**. Fronta má určitú veľkosť, ktorá sa zadáva ako parameter konštruktoru.

Trieda **TNET_MESSAGE_QUEUE** poskytuje dve základné funkcie:

- **PutMessage()** - vkladanie správy do fronty,
- **GetMessage()** - vyberanie správy z fronty.

Obidve funkcie môžu byť blokujúce. Funkcia **PutMessage()** sa blokuje pri vkladaní správy do plnej fronty a funkcie **GetMessage()** pri vyberaní správy z prázdnej fronty. Toto blokovanie zabezpečujú podmienené premenné a celá trieda je bezpečná na používanie viacerými vláknami.

Fronta je implementovaná ako cyklické pole s hlavou.

10.2.3 Triedy TNET_LISTENER a TNET_DISPATCHER

TNET_LISTENER je trieda, ktorá reprezentuje prijímač sieťových správ. Trieda obsahuje frontu prichádzajúcich správ. Objekt tejto triedy vytvára v konštruktore nové vlákno, ktoré čaká na prichádzajúce TCP spojenia. Pri každom novom prichádzajúcom spojení sa vytvorí nové vlákno, ktoré prijíma správy zo siete a vkladá ich do fronty.

TNET_DISPATCHER je trieda, ktorá má za úlohu doručovať prijaté správy. Obsahuje ukazateľ na frontu správ, z ktorej vyberá jednotlivé správy a podľa ich typu na nich vykoná príslušné zaregistrované funkcie. Túto úlohu vykonáva pomocná trieda **TNET_HANDLER**. Všetka činnosť triedy **TNET_DISPATCHER** beží autonómne v samostatnom vlákne, ktoré je vytvorené v konštruktore.

10.2.4 Trieda TNET_TALKER

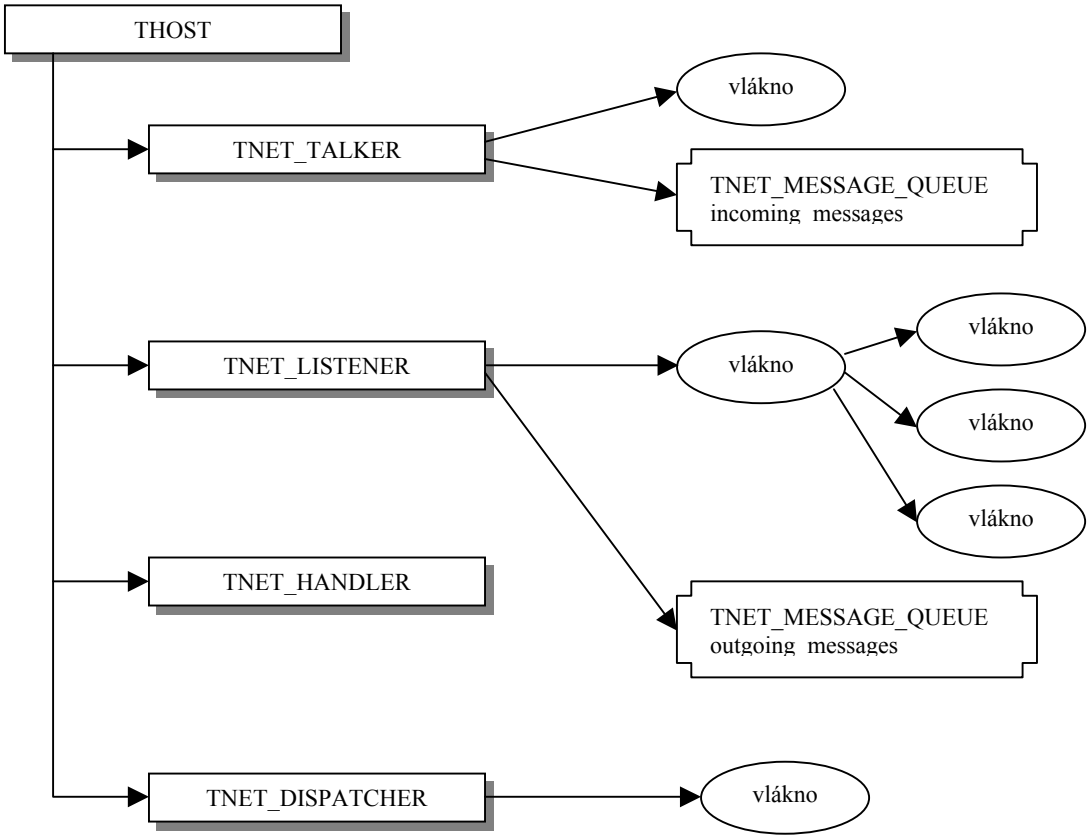
Odosielanie sieťových správ má na starosti trieda **TNET_TALKER**. Trieda obsahuje frontu odosielaných správ. Inštancia objektu si v konštruktore vytvorí nové vlákno, ktoré v nekonečnom cykle vyberá nové správy z fronty odosielaných správ a posiela ich po sieti udaným príjemcom. V prípade, že je príjemca špeciálna hodnota, správa je doručená všetkým príjemcom, na každú adresu však iba raz. Adresy príjemcov sa zadávajú pomocou funkcie **AddAddress()**. Pri pridaní adresy sa automaticky vytvorí TCP spojenie.

10.3 Sieťové rozhranie - trieda THOST

Trieda **THOST** reprezentuje jednoduché sieťové rozhranie obsahujúce všetko potrebné pre sieťovú komunikáciu. Integruje všetky základné sieťové triedy do jednej. Jej štruktúra je zobrazená na obrázku č.2:

Okrem základnej funkcie na posielanie správ **SendMessage()** a funkcie na vytvorenie sieťového spojenia **AddRemoteAddress()**, trieda obsahuje ešte funkcie **RegisterSimpleFunction()** a **RegisterExtendedFunction()**, pomocou ktorých je možné určiť, ktorá funkcia sa zavolá pri príjme jednotlivých typov správ.

Trieda host je implementovaná v súboroch **dohost.h** a **dohost.cpp**.



Obrázok č.2: Trieda THOST

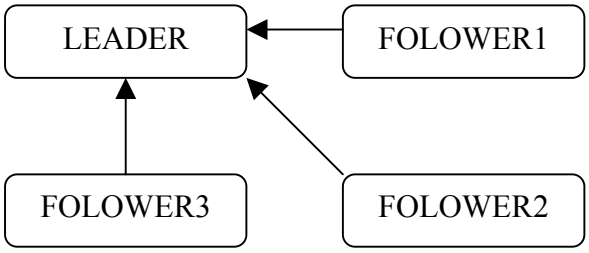
10.4 Začatie hry

Ako už bolo spomenuté vyššie, pri začínaní hry sa používa architektúra klient-server. Počítač, ktorý začína hru sa nazýva **leader**. Jeho úlohou je prijímať spojenia od ostatných počítačov a predávať im informácie o aktuálne pripojených počítačoch - ip adresy, mená hráčov a zvolené rasy. Leader ako jediný vyberá mapu, ktorá sa bude hrať a tiež na ňom beží hyper hráč - hráč, ktorý vlastní zdroje. Taktiež ako jediný môže pridávať počítačových hráčov. Leader používa ako sieťové rozhranie objekt triedy **`TLEADER`**, ktorá je potomkom triedy **`THOST`**. Oproti **`THOST`** má naviac funkcie, ktoré odosielajú sieťové správy, ktoré sú špecifické pre tento typ počítača (viď. Typy správ nižsie).

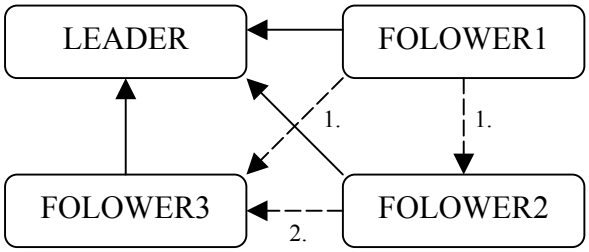
Pripojenie ďalšieho počítača sa vykoná vytvorením TCP spojenia na port, na ktorom počúva leader a poslaním sieťovej správy **`net_protocol_connect`**, ktorá obsahuje informácie o pripájanom počítači: port, na ktorom bude prijímať spojenia od ostatných followerov a meno pripájaného hráča. Leader si ho pridá do poľa pripojených hráčov a odpovedá správou **`net_protocol_hello`** obsahujúcu IP adresu pripájaného followera a aktuálny čas. IP adresa je potrebná na to, aby follower vedel určiť ktorý hráč v poli hráčov je jeho. K prijatému aktuálnemu času sa pripočíta polovička toho, za koľko prišla odpoveď od leadera. Takto sa vykoná prvá hrubá synchronizácia času. Ďalšie synchronizácie času sa vykonávajú opakovaným zasielaním sieťovej správy **`net_protocol_ping`**. Vždy, keď je odpoveď rýchlejšia než ľubovoľná z predchádzajúcich odpovedí, čas sa upraví podľa danej odpovede.

Pri každej zmene informácii o pripojených hráčoch leader pošle všetkým pripojeným počítačom sieťovú správu **`net_protocol_player_array`** obsahujúcu informácie o

všetkých hráčoch vrátane mena hráča, zvolenej rasy a ďalších doplnkových informácií, ako to, či ide o počítačového hráča alebo či ide o hráča, ktorý beží na leaderovi, a štartovacie pole (start_point), kde bude daný hráč začínať na mape. Okrem toho sa posiela informácia o aktuálne zvolenej mape. V prípade, že si hráč na followerovi zmení rasu, pošle leaderovi správu **`net_protocol_change_race`** s názvom novej rasy.



Hra sa začne klepnutím na tlačidlo **Play** na leaderovi. Vtedy sa pošle finálna podoba poľa pripojených hráčov sieťovou správou **`net_protocol_player_array`**, tentokrát však s podtypom 1, ktorý informuje všetky počítače, že sa má spustiť hra na zvolenej mape s danými hráčmi. Pre vytvorenie architektúry peer-to-peer je prípadne potrebné vytvoriť ďalšie spojenia medzi pripojenými followermi. Spojenie vytvára vždy follower, ktorý má hráča s nižším číslom, na followera, ktorý má hráča s vyšším číslom.



10.5 Typy správ

V súbore **`dohost.h`** sú deklarované jednotlivé typy sieťových správ, ktoré môžu byť posielané po sieti. Najdôležitejším typom správy je správa **`net_protocol_event`**, ktorá obsahuje štruktúru **`TEVENT`** popisujúcu nejakú akčnú správu, ktorú je potrebné naplánovať vo fronte vzdialeného počítača. Tieto správy popisujú všetky udalosti, ktoré môžu počas hry nastať - chodenie, útočenie, taženie, stavanie a opravovanie, vyrábanie jednotiek a regenerácia zdrojov. Preto ďalšie typy správ sú už len správy potrebné na synchronizáciu hráčov počas pripájania v menu a prípadné ďalšie udalosti:

- **Spoločné správy**:
  - **`net_protocol_chat_message`** – správa s textom, ktorá sa zobrazí ostatným hráčom,
  - **`net_protocol_synchronise`** – správa informujúca ostatných, že daný hráč je pripravený začať hru.
- **Správy, ktoré posiela leader**:
  - **`net_protocol_player_array`** – správa obsahujúca informácie o všetkých hráčoch a názov zvolenej mapy,

- `net_protocol_hello` – správa, ktorá sa posiela ako odpoveď na `net_protocol_connect`.

- **Správy, ktoré posiela follower**:
  - `net_protocol_connect` – správa posielaná pri pripojení,
  - `net_protocol_change_race` – zmena rasy daného hráča,
  - `net_protocol_ping` – zistenie aktuálneho času.

11 Konfiguračné súbory

Peter Knut

11.1 Úvod

Konfiguračné súbory vznikli už v počiatkoch projektu pre nastavovania základných vlastností aplikácie ako sú veľkosť okna, citlivosť myši a pod. Postupne ako sa menili požiadavky na tieto súbory, menila sa aj ich forma od jednoduchej riadkovej až po štruktúrovanú. Takto vznikol univerzálny systém pre prácu s konfiguračnými súbormi, ktorý sa požíva ako pre nastavenie vlastností aplikácie, tak pre definície jednotlivých máp, schém a rás. Poskytujú vysokú variabilitu pri zachovaní jednoduchosti zápisu. Vďaka textovej podobe môže užívateľ tieto súbory upravovať bez použitia špeciálneho editora.

11.2 Štruktúra súborov

Súbory rozpoznávajú štyri základné zložky: položky, sekcie, komentáre a prázdne riadky. Tieto zložky môžu byť pri zachovaní istých pravidiel navzájom kombinované.

11.2.1 Položky

Každá položka sa nachádza na samostatnom riadku súboru. Položky majú tvar:

"názov položky" ["hodnota" ["hodnota" …]]

Názov položky je ľubovoľný reťazec znakov. V prípade, že názov neobsahuje prázdne znaky (medzery alebo tabulátory), úvodzovky nie sú potrebné.

Za názvom môže byť uvedený rôzny počet hodnôt. Opäť platí, že ak hodnota neobsahuje prázdne znaky, úvodzovky nie sú potrebné. Jednotlivé hodnoty sú od seba a od názvu oddelené prázdnymi znakmi. Konfiguračné súbory podporujú niekoľko typov hodnôt. Naviac môže byť od užívateľa požadované aby niektoré hodnoty zadával v ohraničenom intervale.

Typy hodnôt:

- **string** – ľubovoľný reťazec znakov,
- **byte** – celé číslo v intervale <0, 255>,
- **integer** – celé číslo v intervale <–2 147 483 648, 2 147 483 647>,
- **bool** – povolené hodnoty sú: **true**, **false**, **yes**, **no**, 0, 1,
- **float** – reálne číslo v rozsahu 3.4E +/- 38,
- **double** – reálne číslo v rozsahu 1.7E +/- 308.

Jednotlivé položky musia mať v rámci sekcie navzájom odlišné názvy. Ak toto nie je dodržané, bude dostupná iba prvá položka.

Príklady položiek:

```
fullscreen false
show_fps yes
resolution "800x600"
"warfog color" 100 255 50
sensitivity 0.6
```

11.2.2 Sekcie

Sekcie slúžia na združovanie významovo príbuzných zložiek súboru. Majú tvar:

<"názov sekcie">

</[reťazec znakov]>

Názov sekcie je ľubovoľná postupnosť znakov. Úvodzovky nie sú povinné. Reťazec znakov sa pri spracovaní ignoruje. Je však vhodné využiť ho pre sprehľadnenie súboru. Začiatok i koniec sekcie musí byť uvedený na samostatnom riadku. Sekcia môže obsahovať ľubovoľné zložky súboru vrátane ďalších sekcií. Názvy sekcií na rovnakej úrovni sa nesmú zhodovať, inak bude prístupná iba prvá z nich.

Príklad použitia sekcií:

```
<players>
      max_count 2

      <player 0>
            race "humans"
            position 10 125
      </player>
      <player 1>
            race "orgs"
            position 200 53
      </player>
</players>
```

11.2.3 Komentáre a prázdne riadky

Pre zvýšenie prehľadnosti zápisu je dobré používať prázdne riadky a odsadenie riadkov.

Nasledujúci príklad ukazuje typické použitie všetkých zložiek:

```
# Definicia hracov
<players>
      # Maximalny pocet hracov
      max_count 2

      # Kazdy hrac obashuje rasu a startovnu poziciu.
      # Pozicie su v rozsahu od 0 do 255.
      <player 0>
            race "humans"
            position 10 125
      </player>


      <player 1>
            race "orgs"
            position 200 53
      </player>
</players>
```

11.3 Triedy a metódy

Pre prácu s konfiguračnými súbormi slúžia triedy a metódy z modulu dofile.*. So samotným súborom pracuje trieda **TCONF_FILE**. Pri vytváraní inštancie tejto triedy stačí zadať cestu k súboru a následne metódou **Reload()** načítať jeho obsah. Obsah sa načíta celý do stromovej štruktúry tvorenej inštanciami tried **TFE_SECTION**, **TFE_ITEM** a **TFE_LINE**.

Trieda **TCONF_FILE** obsahuje sadu funkcií pre pohyb v sekciách a samozrejme funkcie pre čítanie a zapisovanie položiek do súboru. Do sekcie sa vstúpi zavolaním funkcie **SelectSecion()** s názvom sekcie ako parametrom a vystúpi zavolaním **UnselectSection()**. To znamená, že k sekcii nachádzajúcej sa v hlbšej úrovni pristúpime viacnásobným volaním **SelectSection()** – postupne s názvami jednotlivých sekcií.

Z aktuálnej sekcie je potom možné čítať hodnoty jednotlivých položiek funkciami **Read<typ>()** a zapisovať funkciami **Write<typ>()**, kde typ je:

- **Str** – pre položky typu **string**,
- **Byte** – pre položky typu **byte**,
- **Int** – pre položky typu **integer**,
- **Bool** – pre položky typu **bool**,
- **Float** – pre položky typu **float**,
- **Double** – pre položky typu **double**,
- **Simple** – tato verzia je totožná s **Byte**. Používa sa vždy v súvislosti s rozmermi a pozíciami na mape.

Pre číselné typy existujú naviac funkcie: **Read<typ>GE()** – pre určenie spodnej hranice načítavanej hodnoty a **Read<typ>Range()** – pre určenie spodnej a hornej hranice načítavanej hodnoty.

Pri viacnásobnom volaní čítacích či zapisovacích funkcií na tú istú položku sa postupne čítajú / zapisujú jednotlivé hodnoty. Vo všetkých funkciách pre čítanie sa nachádza ako jeden z parametrov prednastavená hodnota, ktorá sa použije ako výsledok čítania v prípade, že dôjde k chybe (napríklad keď položka neexistuje).

Pre jednoduchšie otváranie a zatváranie konfiguračných súborov existujú globálne funkcie: **CreateConfFile()**, **OpenConfFile()** a **CloseConfFile()**.

Podrobný popis tried a funkcií sa nachádza v zdrojovom kóde.

12 Logovanie

Peter Knut

12.1 Úvod

Zápis do logov sa používa jednak pre výpis prípadných chýb pri načítavaní konfiguračných súborov a v neposlednom rade pre ladiace výpisy. Všetky záznamy sú uložené v súbore logs/full.log, chybové záznamy sú naviac vypísané v súbore logs/error.log. Záznamy sa vypisujú i na štandardný výstup. Tiež je možné registrovať vlastnú callback funkciu pre spracovanie záznamov. To sa využíva napr. pre výpis logov na obrazovku.

12.2 Typy záznamov

Logy môžu byť rôzneho typu podľa významu. Pre jednotlivé typy sú definované nasledovné makrá:

- **`Info()`** – informácia o prevádzanej akcii.
- **`Warning()`** – varovanie pri nepodstatnej chybe, ktorá nespôsobí zastavenie akcie.
- **`Error()`** – chyba vo vykonávaní akcie. Akcia je zrušená, no aplikácia beží ďalej.
- **`Critical()`** – kritická chyba, ktorá spôsobí ukončenie celej aplikácie.
- **`Debug()`** – ladiaci výpis. Pri skompilovaní Release verzie programu sa tieto výpisy ignorujú.

12.3 Formát záznamov

Formát záznamov sa líši podľa systému na ktorom je program skompilovaný a tiež podľa toho, či ide o Debug alebo Release verziu.

Záznamy sú všeobecne nasledujúceho tvaru:

```
[meno_súboru:číslo_riadku] <hlavička> <text záznamu>
```

Prvý stĺpec obsahuje v hranatých zátvorkách meno zdrojového súboru a riadok z ktorého bol záznam zapísaný. Tento stĺpec sa vypisuje iba v Debug konfigurácii. Hlavička sa líši podľa systému. Prehľad zobrazuje nasledujúca tabuľka:

Typ záznamu (makro)	Windows	Unix a iné
`Info()`	[Info]	Info:
`Warning()`	[Warning]	Warn:
`Error()`	[Error]	Err:
`Critical()`	[Critical]	Crit:
`Debug()`	[Debug]	DBG:

Podrobný popis jednotlivých makier a funkcií sa nachádza v zdrojovom kóde programu (súbory dologs.*).

13 Data Editor

Peter Knut

13.1 Úvod

Program Data Editor umožňuje vytvárať a upravovať dátové súbory používané v hre Dark Oberon. Dátový súbor obsahuje dva základné typy záznamov: textúry (obrazové dáta) a zvuky.

Aplikácia je naprogramovaná v prostredí Borland C++ Builder 5.0 pre operačný systém MS Windows.

13.2 Zdrojové súbory

Aplikácia obsahuje štyri základné moduly rozdelené do zdrojových súborov v adresári **src**:

- datedit.* – hlavný modul aplikácie s automaticky generovaným kódom,
- demainfrm.* – hlavné okno,
- deaboutfrm.* – okno O programe,
- dedatafile.* – obsahujú dátové štruktúry a metódy pre prácu s dátovým súborom a jeho záznamami.

13.3 Hlavné okno

Hlavné okno aplikácie (**`MainForm`**) obsahuje menu (**`MainMenu`**), panel nástrojov (**`ToolBar`**), stromový diagram zobrazujúci štruktúru dátového súboru (**`TreeView`**), oddeľovač pre dynamickú zmenu veľkosti diagramu (**`Splitter`**), a tri panely s ovládacími prvkami pre jednotlivé typy záznamov dátového súboru. Keď užívateľ označí v diagrame niektorú položku, zobrazí sa panel odpovedajúci tejto položke a jeho ovládacie prvky sa vyplnia aktuálnymi hodnotami. Ostatné panely ostanú skryté.

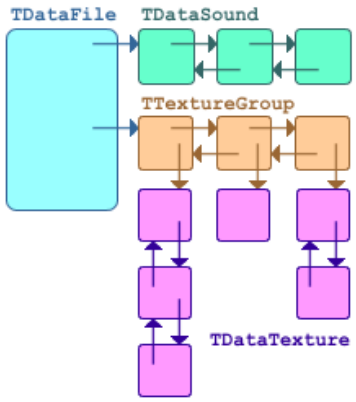
Okno obsahuje i niekoľko skrytých komponentov. Ikony pre položky menu a panel nástrojov sú uložené v komponente **`ImageList`**. Pre otváranie a ukladanie rôznych súborov slúžia štandartné dialógy **`FileOpenDialog`**, **`FileSaveDialog`**, **`TextureOpenDialog`**, **`TextureSaveDialog`**, **`SoundOpenDialog`**, **`SoundSaveDialog`**, **`ImportDialog`** a **`ExportDialog`**.

13.4 Okno O programe

Okno O programe (**`AboutForm`**) zobrazuje ikonu, názov, verziu programu a licenčné podmienky. Obsahuje tlačidlo pre zatvorenie okna (**`OKButton`**).

13.5 Editovanie dát

Prácu s dátovým súborom zabezpečuje trieda **`TDataFile`**. Obsahuje dva obojsmerné spojové zoznamy: zoznam skupín textúr a zoznam zvukových záznamov. Skupinu textúr reprezentuje štruktúra **`TTextureGroup`**. Tá v sebe obsahuje spojový zoznam textúrových záznamov



Obrázok 22: Štruktúra dát

(**TDataTexture**). Zvukový záznam je reprezentovaný štruktúrou **TDataSound**.

Detailný popis jednotlivých premenných a metód sa nachádza v zdrojovom kóde programu.

13.6 Formátovanie dátového súboru

Dátové súbory sú binárne a majú koncovku **dat**. Aktuálna verzia dátového súboru je 3. V zložení súboru je pri každej položke uvedený typ premennej použitej pre zapisovanie a načítavanie danej položky.

Zloženie súboru:

- hlavička – vždy reťazec „Dark Oberon data file“ – 21 x **char**,
- verzia súboru – **unsigned char**,
- počiatok bloku súboru so skupinami textúr – **long**. V prípade, že sa v súbore skupiny textúr nenachádzajú, tento počiatok je nastavený na nulu,
- počiatok bloku súboru so zvukmi – **long**, V prípade, že sa v súbore zvukové záznamy nenachádzajú, tento počiatok je nastavený na nulu,
- blok súboru obsahujúci textúry. Tento blok sa v súbore nachádza iba v prípade, že v ňom existuje aspoň jedna skupina textúr:
    - počet skupín textúr – **int**,
    - jednotlivé skupiny:
        - dĺžka názvu skupiny textúr – **unsigned char**,
        - reťazec s názvom skupiny – n x **char**,
        - počet textúr v skupine – **int**,
        - záznamy textúr:
            - dĺžka názvu textúry – **unsigned char**,
            - reťazec s názvom – n x **char**,
            - horizontálny počet snímkov v animovanej textúre – **unsigned char**. V intervale <0, 100>,
            - vertikálny počet snímkov v animovanej textúre – **unsigned char**. V intervale <0, 100>,
            - dĺžka animácie v milisekundách – **int**. V intervale <0, 10000>,
            - x-ová pozícia stredu súradnicovej sústavy – **int**. V intervale <-1024, 1024>,
            - y-ová pozícia stredu súradnicovej sústavy – **int**. V intervale <-1024, 1024>,
            - typ textúry – **unsigned char**. Povolené hodnoty sú uvedené v tabuľke nižšie,
            - veľkosť zdrojových dát – **unsigned int**. Môže byť i nula,
            - zdrojové dáta – n x **unsigned char**. Iba v prípade, že veľkosť dát je väčšia ako nula,
- blok súboru obsahujúci zvuky. Tento blok sa v súbore nachádza iba v prípade, že v ňom existuje aspoň jeden zvukový záznam:
    - počet zvukov – **int**,
    - jednotlivé záznamy zvukov:
        - dĺžka názvu zvuku – **unsigned char**,
        - reťazec s názvom zvuku – n x **char**,
        - formát zvuku – **char**. Povolené hodnoty sú uvedené v tabuľke nižšie,

- typ zvyku – **`char`**. Povolené hodnoty sú uvedené v tabuľke nižšie,
- veľkosť zdrojových dát – **`unsigned int`**. Môže byť i nula,
- zdrojové dáta – n x **`unsigned char`**. Iba v prípade, že veľkosť dát je väčšia ako nula.

Typy textúr:

0	Bežná štvorcová textúra
1	Textúra určená pre fragment terénu

Typy zvukov:

0	Sample
1	Stream

Formáty zvukov:

0	WAV
1	MP2
2	MP3
3	OGG
4	RAW
5	MOD
6	S3M
7	XM
8	IT
9	MID
10	RMI
11	SGT

14 Map Editor

Peter Knut

14.1 Úvod

Program Map Editor umožňuje vytvárať a upravovať konfiguračné súbory máp používaných v hre Dark Oberon. Tento editor nie je úplný, je zameraný iba na editovanie povrchu máp.

Aplikácia je naprogramovaná v prostredí Borland C++ Builder 5.0 pre operačný systém MS Windows.

14.2 Zdrojové súbory

Aplikácia obsahuje niekoľko základných modulov rozdelených do zdrojových súborov v adresári **src**:

- mapedit.* – základný, automaticky generovaný kód aplikácie,
- memain.* – kód hlavného okna,
- meabout.* – okno O programe,
- menewmap.* – dialóg s vlastnosťami mapy,
- mefragsize.* – dialóg s veľkosťou fragmentov,
- mefragments.* – dialóg pre definovanie farebnej schémy,
- memap.* – obsahujú triedu pre prácu so súborom mapy,
- mescheme.* – obsahujú triedu pre prácu so súborom farebnej schémy,
- mefile.* – obsahujú univerzálne dátové štruktúry a metódy pre prácu s konfiguračným súborom. Tieto súbory sú prevzaté zo zdrojových súborov hry Dark Oberon a upravené pre použitie v Map Editore.

14.3 Hlavné okno

Základné súčasti hlavného okno aplikácie (**frm_main**) sú: menu (**mnu_main**), tabuľka fragmentov a farebnej schémy (**grd_scheme**), pole pre výber aktuálneho segmentu (**rg_segment**), oddeľovač pre dynamickú zmenu veľkosti tabuľky (**splitter**), a obrázok pre vykresľovanie mapy (**img_map**). Veľkosť obrázka sa automaticky mení na základe veľkosti otvorenej mapy tak, aby políčko mapy predstavujúce jeden fragment bolo vždy rovnako veľké.

Okno obsahuje i niekoľko skrytých komponentov. Ikony pre položky menu sú uložené v komponente **il_menu**. Pre otváranie a ukladanie súborov slúžia štandartné dialógy **glg_open** a **dlg_save**.

14.4 Pomocné dialógy

Dialóg **frm_new_map** slúži jednak pre prvotné nastavenie vlastností mapy pri vytváraní novej mapy a potom pre zmenu jej vlastností počas editovania. Súčasťou dialógu sú polia pre nastavenie rozmerov mapy a veľkosti fragmentov; a tlačidlá pre potvrdenie prípadne zrušenie nastavenia.

Pri otváraní existujúcej mapy je potrebné zadať veľkosť fragmentov, ktorú mapa používa. Pre tento účel je vytvorený dialóg **frm_frag_size**.

Pomocou dialógu **frm_fragments** je možné upravovať farebnú schému. Dialóg obsahuje pole pre voľbu segmentu (**rg_segment**), editovacie polia pre číslo a meno fragmentu, obrázok pre zobrazovanie farby fragmentu (**pic_color**) a nakoniec tlačidlo pre zavretie dilógu (**btn_done**).

14.5 Okno O programe

Okno O programe (**frm_about**) zobrazuje ikonu, názov, verziu programu a licenčné podmienky. Obsahuje tlačidlo pre zatvorenie okna (**btn_ok**).

14.6 Dátové štruktúry

Prácu s konfiguračnými súbormi mapy a farebnej schémy zabezpečujú triedy **TMAP_FILE** (memap.*) a **TSCHEME_FILE** (mescheme.*) s využitím triedy **TCONF_FILE** (mefile.*).

Farebná schéma je udržiavaná v globálnej premennej **frg_info** typu **TFRAGMENTS_INFO**, mapa v poli **map_arr** (memain.*).

Detailný popis jednotlivých premenných a metód sa nachádza v zdrojovom kóde programu.

14.7 Formáty súborov

Formát konfiguračného súboru mapy je uvedený v dokumentácii k vytvoreniu vlastnej mapy v hre Dark Oberon.

Súbory s definíciami farebných schém sú textové a majú koncovku **col**. Používajú rovnaký formátovací systém ako ostatné konfiguračné súbory hry Dark Oberon. Obsahujú tri základné značky **<Segment #>** pre každý segment, kde # je číslo segmentu (0, 1 alebo 2). Značka **** obsahuje jednak položku **count** s počtom fragmentov v príslušnom segmente, a potom položky v tvare:

fragment_# r g b name

kde # je číslo fragmentu počnúc od nuly; r, g, b sú čísla z intervalu <0, 255> definujúce farbu fragmentu po zložkách (červená, zelená, modrá); name je meno fragmentu.

Príklad farebnej schémy:

```
<Segment 0>
  count 1
  fragment_0 108 221 114 "clay"
</Segment 0>
<Segment 1>
  count 3
  fragment_0 108 221 114 "grass"
  fragment_1 192 192 192 "rocks"
  fragment_2 72 196 255 "water"
</Segment 0>
<Segment 1>
  count 0
</Segment 0>
```