ANALÝZA A POPIS IMPLEMENTACE PROGRAMU

Hra ActionBomberman

Zápočtová práce – Programování II

Zhotovil: Lukáš Doležal
(skupina 41)

Zadání programu:

Naprogramovat hru ve stylu známého bomermana. Hra musí obsahovat hru proti počítači, možnost hry více hráčů na jedné klávesnici s možností výběru kláves. Nejsou kladeny požadavky na grafickou ani zvukovou stránku hry.

Analýza zadání:

Princip hry

Ačkoliv je bomberman známý, shrnu zde princip této hry. Všichni hráči se nacházejí na hracím poli čtvercové mřížky. Políčka na poli se rozdělují na prázdné, po nichž je možné s hráčem chodit, zničitelné, které lze výbuchem bomby zničit a nezničitelné. Úkolem hráčů je pomocí pokládání bomb na políčka mapy zneškodnit své protivníky. Hráč může během hry získat různé zvýhodnění jako větší počet bomb, jejich větší sílu, možnost bomby odkopávat a také může dostat na krátkou dobu znevýhodnění v podobě zpomalení či nemožnosti položit bombu.

Části programu

Program lze rozdělit na 2 části:

- Frontend – to co uživatel vidí
  - herní menu
  - načítání souborů s mapou
  - vykreslování hrací plochy
  - předávání vstupů ovládání hernímu modelu
- Herní model – to co se stará o „život“ hry a herní logiku
  - herní logika
  - umělá inteligence
  - obsluha vstupů ovládání

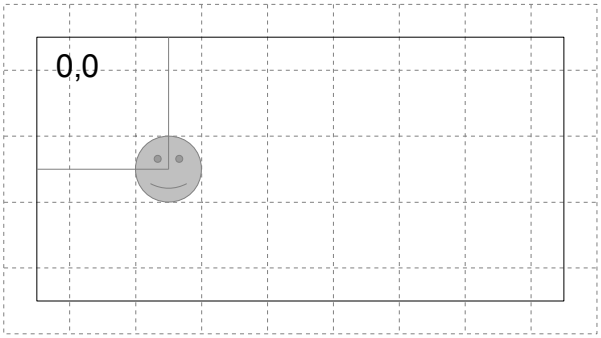
Díky tomuto rozdělení je možné do budoucna případná změna vzhledu hry, přičemž herní model se nebude muset měnit.

Implementace herního modelu

Pohyb po hrací ploše

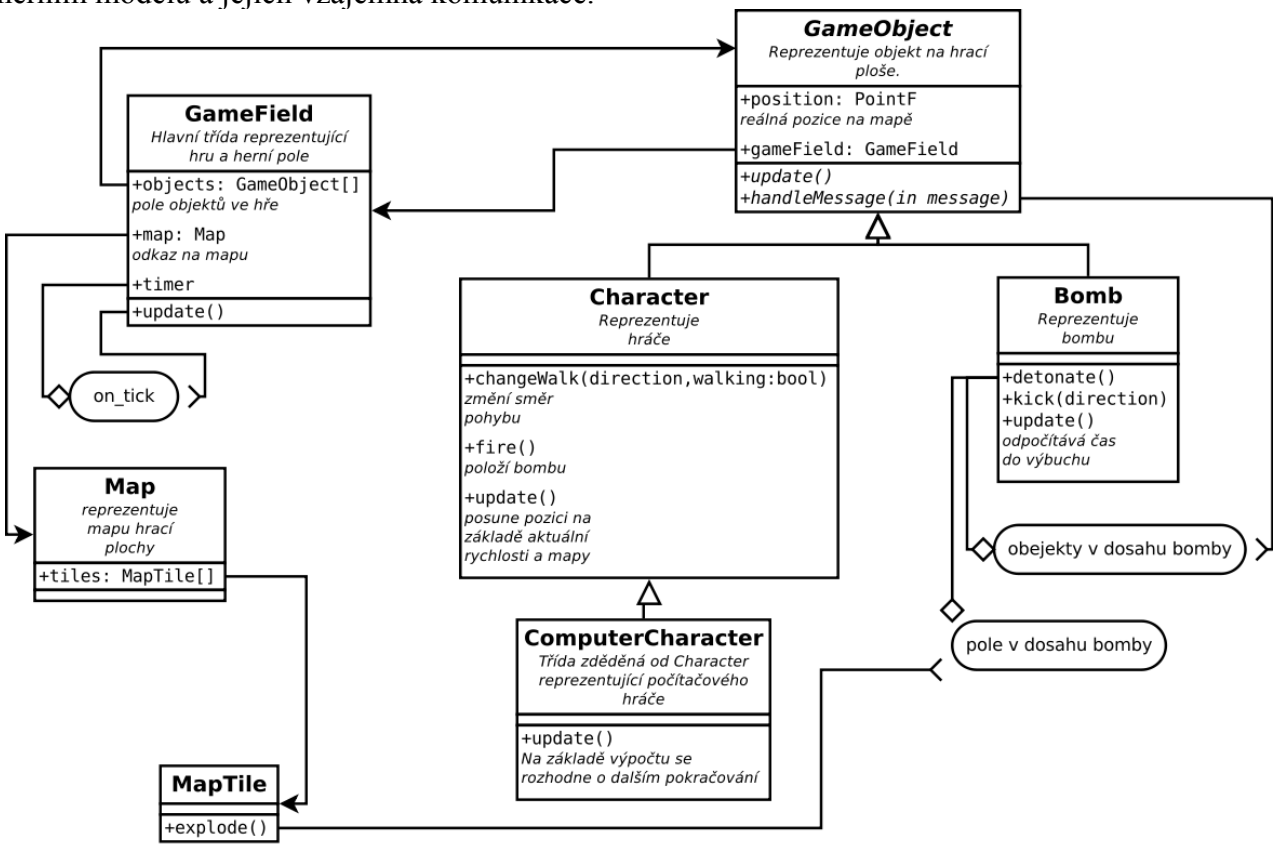
Hrací plocha je tvořena mřížkou čtvercových polí. Nechtěl jsem ale, aby se hráč mohl pohybovat pouze po této mřížce, protože by pohyb nebyl plynulý. Je proto určena délka strany polí v bodech. Hráči a ostatní objekty na hracím poli mají svou polohu zaznamenanou jako jejich střed v jednotkách odpovídajících dané délce strany polí. Stejně tak jejich rychlost je udávána v těchto bodech. Tím je navíc odstíněn herní model od zobrazení, kde může být hrací mapa zobrazena v různém měřítku.

Pozice objektů na mapě se ukládá jako jejich střed. Proto je souřadný systém vůči mapě posunut o polovinu šířky pole dolu a doprava. Znázorněno je to na následujícím obrázku:



Objektový návrh

Herní model je de facto samostatnou částí programu. Na jeho návrhu závisí přehlednost programu a obtížnost budoucích úprav či přidání funkcí. Proto jsem se snažil navrhnout ho tak, aby byl co nejvíce abstraktní a rozšiřitelný. Na následujícím schéma je znázorněna hierarchie objektů v herním modelu a jejich vzájemná komunikace.



GameField

Základem herního modelu je třída *GameField*. Třída obstarává kompletní běh hry. Její nejdůležitější datovou součástí jsou odkazy na instance objektů na hrací ploše a odkaz na instanci mapy. Hlavní činností třídy je kontrolovat stav hry, tj. uplynulí čas a zda již není pouze jeden živý hráč, a v intervalech aktualizovat herní objekty. O to se stará timer a metoda *update()*, která volá stejnojmenné metody na všech objektech ze seznamu *objects*.

GameObject

Abstraktní třída *GameObject* reprezentuje všechny objekty pohybující se po hrací ploše, ty jsou implementovány jako potomci této třídy. Objekty si uchovávají informaci o své poloze na mapě. Tato informace je uložena ve formě polohy středu objektu, tak jak bylo popsáno výše. Každý

potomek této třídy musí implementovat metodu *update*(), která je volána v intervalech instancí *GameField*, již je součástí. Další abstraktní metodou je *handleMessage*, která slouží k interakci mezi objekty (podrobněji u popisu odvozených tříd).

Character a ComputerCharacter

Třída *Character*, odvozená od *GameField,* reprezentuje hráče. Stará se o jeho pohyb, pokládání bomb, interakci s mapou a ostatními objekty. Uchovává aktuální stav hráče jako je počet bomb, jejich síla nebo „nemoci".

Z této třídy je odvozena třída *ComputerCharacter*, která reprezentuje hráče ovládaného počítačem. Vlastní má de facto jen metodu *update()*, ve které je počítán další postup. Podrobněji v další části.

Bomb

Pokládané bomby reprezentuje třída *Bomb*. Nese si informace o zbývajícím čase do výbuchu, který je odpočítáván v metodě *update()*. Při výbuchu informuje všechny objekty, které se nachází v místě výbuchu zprávou „inExplosion". *Character* se při této zprávě označí za mrtvého, ostatní *Bomb* se aktivují. Zároveň aktualizuje mapu o vybuchlé pole.

Počítačový hráč

Počítačový hráč (bot) je hráč, který se řídí samočinně na základě algoritmu.

Bot v této hře se rozhoduje o tom, jakým směrem se bude pohybovat a na jakém políčku položí bombu. Pro bota jsou podstatné 2 věci:

- ohodnocení políček, pomocí něhož si vybere nejvhodnější cíl
- nalezení cesty do vybraného cíle

Dále závisí na aktuální situaci bota:

- Je-li na políčku ohroženém bombou je potřeba se co nejrychleji dostat do bezpečí. Vybráno je tedy nejbližší neohrožené políčko.
- V ostatních případech je vybráno políčko na základě poměru skóre/vzdálenost. Navíc cesta nesmí vést přes políčko ohrožené výbuchem.

Poměr skóre/vzdálenost je zvolen proto, aby bot vybíral spíše bližší cíle, než vzdálené. Tím může lépe využít například většího počtu bomb a času, kdy by pouze při rozhodování na základě ohodnocení např. šel nejprve daleko na nejvíce ohodnocené pole a poté se vracel.

Výběr algoritmu hledání cesty

Po výběru nejlepšího políčka je potřeba zjistit, jakou cestou se k němu nejrychleji dostat.

V úvahu připadají

- Dijkstrův algoritmus hledání nejkratších cest v grafu
- A* algoritmus hledání nejkratší cesty

Vybrán byl Dijkstrův algoritmus. Protože výběr nejlepšího políčka závisí i na vzdálenosti je nutné tuto informaci při výběru pole znát. Dijkstrův algoritmus dokáže najednou zjistit vzdálenosti do všech polí a zároveň i cestu k nim. Proto stačí jedno jeho spuštění, díky němuž se zjistí nejlepší pole a hned poskytuje i informaci o cestě k němu. A* zjistí pouze cestu do konkrétního bodu a ostatní informace nejsou z jeho běhu zaručeny, není proto pro tento účel vhodný, i když by jako pouhé nalezení cesty byl rychlejší.

Ohodnocení polí

Skóre je rozděleno do 2 částí, které se sčítají. Kritéria pro přidělení skóre:

- Skóre pro položení bomby
  - (+) za každou cihlovou zeď, kterou lze zničit položením bomby na políčko (dle aktuální

síly bomb) ale není již v dosahu jiné bomby (tím se předejde zbytečnému plýtvání bomb)
    - (+) za protivníka v dosahu, ale tak, že pole které je přímo pod protivníkem má menší ohodnocení než pole které jsou dále v dosahu. Tímto se počítač snaží pokládat bomby před protivníka, tak aby nemohl bombou projít.
- Skóre pro navštívení:
    - (+) za políčka obsahující pozitivní bonus
    - (-) pole ohrožené výbuchem bomby. Čím je do výbuchu menší čas, tím je skóre sníženo více. Tím se bude preferovat cesta, která protíná co nejméně nebezpečných políček.
    - (-) za pole obsahující negativní bonus.

Nalezení cesty

K výpočtu vzdálenosti a cesty k políčkům se používá Dijkstrův algoritmus hledání nejkratší cesty v grafu. Graf je sestaven tak, že vrcholy tvoří políčka mapy a hrany mezi vrcholy reprezentují možné směry pohybu z každého políčka (tedy sever, jih, západ a východ).

Nalezená cesta při použití Dijkstrova algoritmu závisí nejvíce na váhové funkci hran. Protože pouze nejkratší cesta, kdy by každá hrana měla stejnou váhu, by nemusela být vždy nejlepší – mohla by procházet přes špatné bonusy apod., je váha hran závislá také na hodnocení pole do kterého hrany vedou:

w = *k*/*m*^*skore*

pokud by navíc pole do kterého hrana vede mělo menší čas do exploze než zdrojové je

w = w + 2*k*

kde *skore* je skóre pole, do kterého vede hrana a *k* a *m* jsou určité konstanty určená v implementaci. Hodnota *m* tak, aby pro běžný rozsah skóre nebyl dělitel příliš veliký. Hodnota *k* tak, aby existoval určitý rozsah pro snižování i zvyšování hodnoty vydělením. Tyto hodnoty nebyly vypočítány exaktními metodami, ale zvoleny empiricky.

Přesné hodnoty lze nalézt v souboru src/game_model/mappathfinder.cpp ve funkci MapPathFinderNode::relaxDistance.

Tímto se zajistí, že cesty do políček s vyšším skóre budou výhodnější přes políčka, které nemají nízké skóre průchodu. Obyčejná prázdná políčka mají skóre 0, tedy váha hrany do nich bude rovna *k*. Hrany vedoucí do políček s negativním skóre budou mít váhu rovnou *k* poděleném o číslo menší jak 1, tedy ve výsledku vetší jak *k*. Naopak políčka s kladným skóre budou mít „příchozí“ hrany ohodnoceny téměř nulovou vzdáleností.

Položení bomby

Jakmile bot dosáhne políčka vypočteného jako nejlepšího a skóre pro položení bomby není nulové, pokusí se položit na něm bombu. Aby však položením bomby nezpečetil svoji smrt, je nutné nejprve zjistit, zda bude při položení bomby existovat cesta do bezpečí. K tomu jsem použil princip algoritmu DFS.

Ideální by bylo, kdyby bot políčka kde bombu z tohoto důvodu položit nemůže ani nebral v „úvahu“ při vybírání cílů. Protože by ale vypočtení této informace předem byl náročné (spuštění DFS pro každý vrchol s upravenými hranami podle umístění bomb a jejich dosahu), vypočítává se to až když je bot na tomto poli a chce umístit bombu.

```
foreach soused in aktualni_policko.sousedi
   zasobnik.push(soused)
```

```
while (zasobnik.empty)
   policko = zasobnik.pop()
   if (policko.ohrozene = false)
        return true // nalezeno neohrožené políčko v DFS strome – existuje cesta do bezpečí

   if (policko.navstivene = false)
        policko.navstivene = true
        foreach soused in policko.sousedi
             zasobnik.push(soused)

   // pokracovat while

return false; // v DFS strome se nenalezlo neohrožené políčko
```

Pokud na políčko není z tohoto důvodu položit bombu, je označeno za nepovolené a vypočítá se cesta k jinému políčku. To zabrání zbytečnému setrvávání na políčku, kde není momentálně možné položit bombu.

Složitost a optimalizace

Odhad výpočetní složitosti nalezení políčka „kam jít“:

Vytvoření grafu a oskórování:	N
Dijkstra s Q jako pole:	$O(N^2 + 4N) = O(N^2)$
Vybrání nejlepšího políčka a cesty:	$<2N = O(N)$
Celkem:	**$O(N^2)$**

Bot vypočítává nový směr pouze jednou za 150 ms a/nebo pokud dosáhne políčka vybraného předchozím výpočtem.

Absolutní složitost jsem snížil ještě tak, že vytvoření a oskórování grafu je prováděno pouze jednou pro všechny boty během jednoho cyklu.

Implementace frontendu

Frontend hry zahrnuje následujcjednoduché menu pro nastavení parametrů hry, komponentu vykreslující hrací plochu během hry a loader souborů s mapami.

Tyto vizuální prvky byly implementovány pomocí frameworku Qt a jeho vizuálních komponent.

Propojení s herním modelem

Protože herní model se stará pouze o herní logiku, neobsahuje žádné dodatečné informace jako jména hráčů, jejich aktuální skóre nebo nekontroluje počet odehraných soubojů.

O to se stará frontend s jeho třídamy *PlayerHandler* a *GameSettings*.

PlayerHandler

Tato třída obsahuje nastavení kláves, jméno hráče, aktuální skóre, barvu pro vykreslení a další. Obsahuje také odkaz na instanci *Character* třídy obsažené v herním modelu.

GameSettings

Tato třída obsahuje seznam instancí *PlayerHandler* jako seznam hráčů a dále také nastavení hry jako počet vítězství ke konci turnaje, nastavení hráčů pro novou hru (počet bomb a síla).

GameWidget

Obě tyto třídy jsou obsluhovány komponentou starající se o vykreslování a zpracování vstupů hry *GameWidget*. Tato třída obsahuje instanci herního modelu a na základě vstupů z klávesnice ho ovládá a na základě dat z něho přečtených vykresluje hrací plochu.

K vykreslování se používají součástí Qt frameworku. Grafika je načítána z formátu SVG, je tedy škálovatelná beze ztráty kvality podle velikosti okna hry.

Načítání map

Při spuštění hry je potřeba načíst mapu. Mapy jsou uloženy v textovém formátu popsaném v souboru *Formát souborů s mapou*.

Mapy mohou deklarovat políčka s náhodným typem a bonusem. Tyto políčka jsou nastavena s použitím generátoru pseudonáhodných čísel právě při načítání do hry. Mapa se během jednoho turnaje nemění.