Programátorská dokumentace GrainSim

Marek Bečvář
31.7.2021

Obsah

I Rozbor specifikací

i. Popis

Projekt GrainSim měl za cíl vytvořit v jazyce C# a frameworku Monogame 2D herní prostředí, ve kterém si uživatel bude moci tvořit experimenty s látkami, jejichž vlastnosti budou založené na těch reálných. V prostředí pak mohou látky jak navzájem, tak v reakci na prostředí reagovat (výbušniny, hořlaviny, přeměny skupenství, atd.).

ii. Funkční požadavky

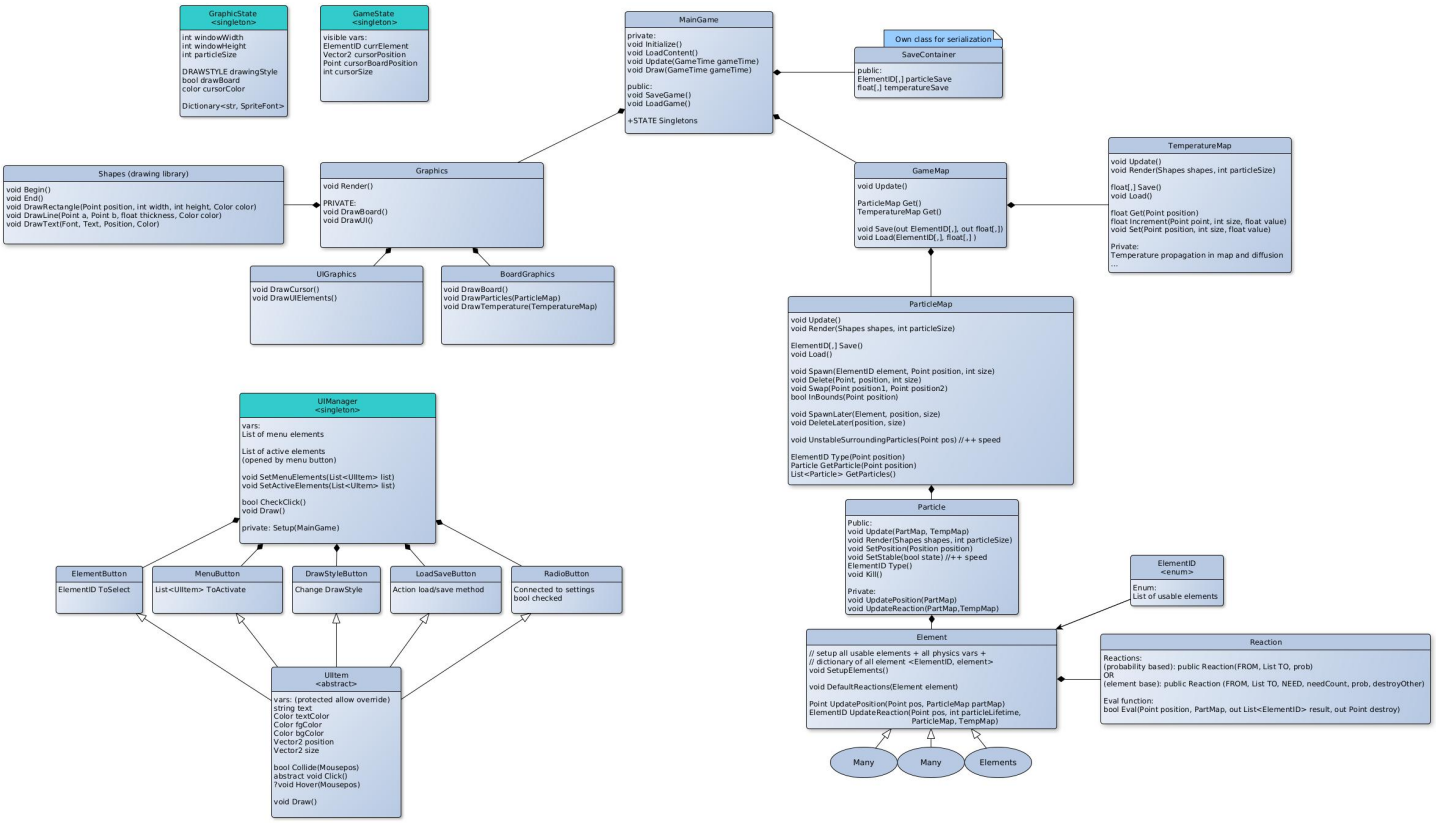
(z finální verze specifikace zápočtového projektu)

Na základě uživatelem vložených prvků aplikace:

- vykresluje aktuální mapu prostředí
- očekává další vstup od uživatele (výběr jiného prvku, změna vykreslovací mapy, aj.)
- v každém kroku simulace:
  - přesouvá podle daných pravidel prvky po prostředí
  - přepočítává teplotní mapu prostředí
  - kontroluje prvky na možné reakce (na základě okolních prvků a teplot)

II Architektura/Design

Architektura programu byla před zahájením projektu načtrtnuta do UML grafu (které se v průběhu práce na projektu někdy rozrostlo, ale velké změny nikdy nenastaly). To co si určitě z tohoto kurzu odnesu dál je síla takového náčrtu, kdy člověk dokáže vyřešit problémy v architektuře daleko dříve, než se k nim vlastně dostane.



Rozdělení Celý program se dělí na více částí. Centrální je ale třída `MainGame`. Zde se nachází veškerá logika spojená s Monogame. Zároveň se ale z tohoto místa posílají výzvy k aktualizacím a vykreslení prostředí a výsledky uživatelských vstupů.

i. High-level

Prvky/Teploty Centrální třídou kde se inicializují všechny potřebné části je `MainGame`. Práce s prvky a teplotami probíhá ve speciálních mapách, tudíž na `MainGame` navazuje třída `GameMap`, která v sobě drží odkazy na mapu částic (`ParticleMap`) a mapu teplot (`TemperatureMap`). Mapa částic pak v sobě má seznamy všech aktuálně simulovaných prvků. Každý takto simulovaný prvek je popsán pomocí vlastní instance třídy `Particle`. Jednou z hlavních vlastností této instance je typ elementu, který představuje. Všechny možnné typy částic jsou pak popsané v enum `ElementID` v tříde `Element`. Tato třída se stará o veškerou inicializaci jednotlivých prvků, logiku jejich pohybu po mapě a řeší jejich možné reakce, které jsou popsané vlastní třídou `Reaction`. Jednotlivé prvky jsou pak vytvářené jako třídy odvozené od třídy `Element` (každý prvek vlastní soubor) a jsou jim ručně nastaveny jejich vlastnosti.

Vykreslování Zpět v třídě `MainGame` se nachází odkaz na třídu `Graphics`, což je centrální třída pro vykreslování všeho. Pod ní se nachází třída `Shapes`, což je vlastní třída pro vykreslování tvarů a textu na obrazovku. Dále má tato třída odkaz na třídy `UIGraphics` a `BoardGraphics`. `UIGraphics` může inicializovat vykreslování jednotlivých UI prvků (menu, infotext) a zároveň řeší vykreslování kurzoru. `BoardGraphics` může vykreslovat čtverečkovanou sít a mapy částic a teplot.

UIManager O logiku a vlastní vykreslování UI elementů v menu sekci se stará vlastní singleton `UIManager`, který pod sebou má řadu různých typů tlačítek, odvozené od abstraktní třídy `UIItem`.

State singletons Pro určité speciální hodnoty programu jsem vytvořil dva singletony (*zejména protože se jedná o velmi specifické parametry potřebné skoro všude a jejich předávání by akorát přineslo větší zmatek do funkcí*). Jedná se o `GraphicState` (hodnoty velikosti herního okna, zvolená velikost částic, zvolený typ vykreslování, toggle vykreslování čtverečkované sítě, barvu kurzoru a list načtených fontů) a `GameState` (právě zvolený element, pozice myši na obrazovce, přepočet pozice myši do čtverečkované plochy a aktuální velikost kurzoru).

i.i Mapování na funkční požadavky

- vykresluje aktuální mapu prostředí → `Graphics` a připojené třídy
- očekává další vstup od uživatele (výběr jiného prvku, změna vykreslovací mapy, aj.) → `MainGame` s `UIManager`
- v každém kroku simulace: → skrz třídu `GameMap`
  - přesouvá podle daných pravidel prvky po prostředí → `ParticleMap` a vlastní třída `Element`
  - přepočítává teplotní mapu prostředí → celé v třídě `TemperatureMap`
  - kontroluje prvky na možné reakce (na základě okolních prvků a teplot) → třídy `Element` a

III Rozdělení do funkcí a procedur

Update Všechna volání procedur vychází opět z hlavní třídy `MainGame`. Zde se nachází dvě procedure základem v Monogame frameworku `Update` a `Draw`. V `Update` se řeší uživatelské inputy (v `UIManager.CheckClick`) a zároveň se odtud volají update procedury ve třídě `GameMap`, které se dále propagují do updatů tříd jednotlivých map a v mapě částic až do jednotlivých částic. V částicích se pak volají funkce třídy `Element` s připojením daného typu částice, které řeší pohyb částice a její možné reakce (*tato volání jsou velmi drahá s narůstajícím množstvím částic, a proto se částice, která neprochází změnami polohy ani stavu, po několika takových cyklech uspí a probouzí se, až když nastane nějaká aktivita v okolních částicích*).

Input handle Při hodnocení uživatelského vstupu se nejprve kontroluje, jestli nebylo stisknuté nějaké tlačítko klávesnice poté jestli nebylo stisknuté tlačítko v menu části - konrola v `UIManager.CheckClick` a pokud ne, tak kontrola, jestli neprobíhá stisk levého nebo pravého tlačítka na simulačním prostředí.

Pokud ano, pak se na základě právě zvoleného prvku rozhoduje předávání události do speciálních metod tříd `TemperatureMap` nebo `ParticleMap`. Ty si řeší uživatelský vstup samostatně na základě pozice kurzor v mapě a velikosti kurzoru.

Draw Na druhou stranu `MainGame` vysílá se zvé `Draw` procedury volání na `Graphics.Render`, kde se vyhodnocuje, jaké vše vykreslovací metody v `UIGraphics` a `BoardGraphics` zavolat. `UIGraphics` toto volání poté přesouvá na singleton `UIManager`, který drží všechna tlačítka potřebná k vykreslení a volá jejich vlastní `Draw` metody odvozené od abstraktní třídy `UIItem`. `BoardGraphics` vykreslování částic i teplot také předává do jednotlivých map, které mají vlastní logiku vykreslování ve svých `Render` funkcích.

IV Implementované datové struktury

Program nemá žádné speciální datové struktury. Užitečnými byly struktury **dictionary**, **list** a **víceúrovňové pole**.

Víceúrovňové pole Víceúrovňové pole se využívá pro indexování buněk mapy částis a pro udržení a práci s hodnotami teplot v mapě teplot.

List Datová struktura list je použita třeba pro obecné reakce jednotlivých elementů. Při inicializaci těchto elementů tak stačí do listu pouze naskládat kolik různých reakcí chceme a procedura třídy **Element** pak při kontrole reakcí všechny tyto načtené kontroluje.

Dále je list využíván v menu stavech. `UIManager` udržuje listy tzv. *menu elementů* (= velká tlačítka, kategorie) a *aktivních elementů* (= menší tlačítka, př. samotné elementy, možnosti nastavení). Dále pak existuje typ tlačítka `MenuButton`, který v sobě obsahuje list tlačítek a kategorii, do které mají být načtena. Takto je možno s předáváním dvou listů vytvářet v podstatě libovolně hluboké menu (*jen možná inicializace těchto tlačítek může být trochu nepřehledná - `MainGame.Setup`*).

Dictionary Dictionary je v tomto programu využito také na více místech. Důležitá je globální proměná `Element.elements`, která s klíčem z enum `ElementID` odkazuje na jednotlivé třídy vlastních prvků (*tak je možné se dostat k vlastnostem jednotlivých elementů bez potřeby všude předávat velké odkazy na celé třídy prvků*).

Dále je dictionary využita v mapě částic pro udržení a spravování aktivních částic. Zde je klíčem index pozice v mapě, na kterém se prvek nachází (*při pohybu se částice vyměňují na pozicích, index pozice mapy je po celou dobu stejný*). Původní implementace využívala list, ale udělal jsem přechod k dictionary z důvodu rychlostní převahy. V simulaci se obvykle nachází okolo 5000 částic a při potřebe odebrat z takového listu třeba 100 položek (při dostatečně velkém kurzoru možné) na úplně neznámých indexech, je rychlostně naprosto nezvladatelné. Proto bylo zvolené dictionary. Zjistit klíče, se kterým chci pracovat je jednoduché (z pozic v mapě - díky pozici kurzoru znám). Indexování pomocí klíče by poté bylo to nejrychlejší co může program nabídnout. Zároveň, dictionary je celé vytvořené a zaplněné při inicializaci, tudíž už nedochází k žádnému opakovanému vytváření. Stačí zvolit množstvím částic, které je zvladatelné a simulace nebude mít s daným počtem částic žádné problémy.

Posledním místem, kde je tato struktura využita je v `GraphicState` jako kontejner na použitelné fonty textu inicializované v `MainGame`. Klíčem k nim jsou v kódu zvolené proměnné, popisující využití těchto fontů.

V Závěr

Projekt byl vytvoření jako záverečná semestrální práce pro předmět *Programování 2* - Letní semestr 2021 - UK Matfyz.