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

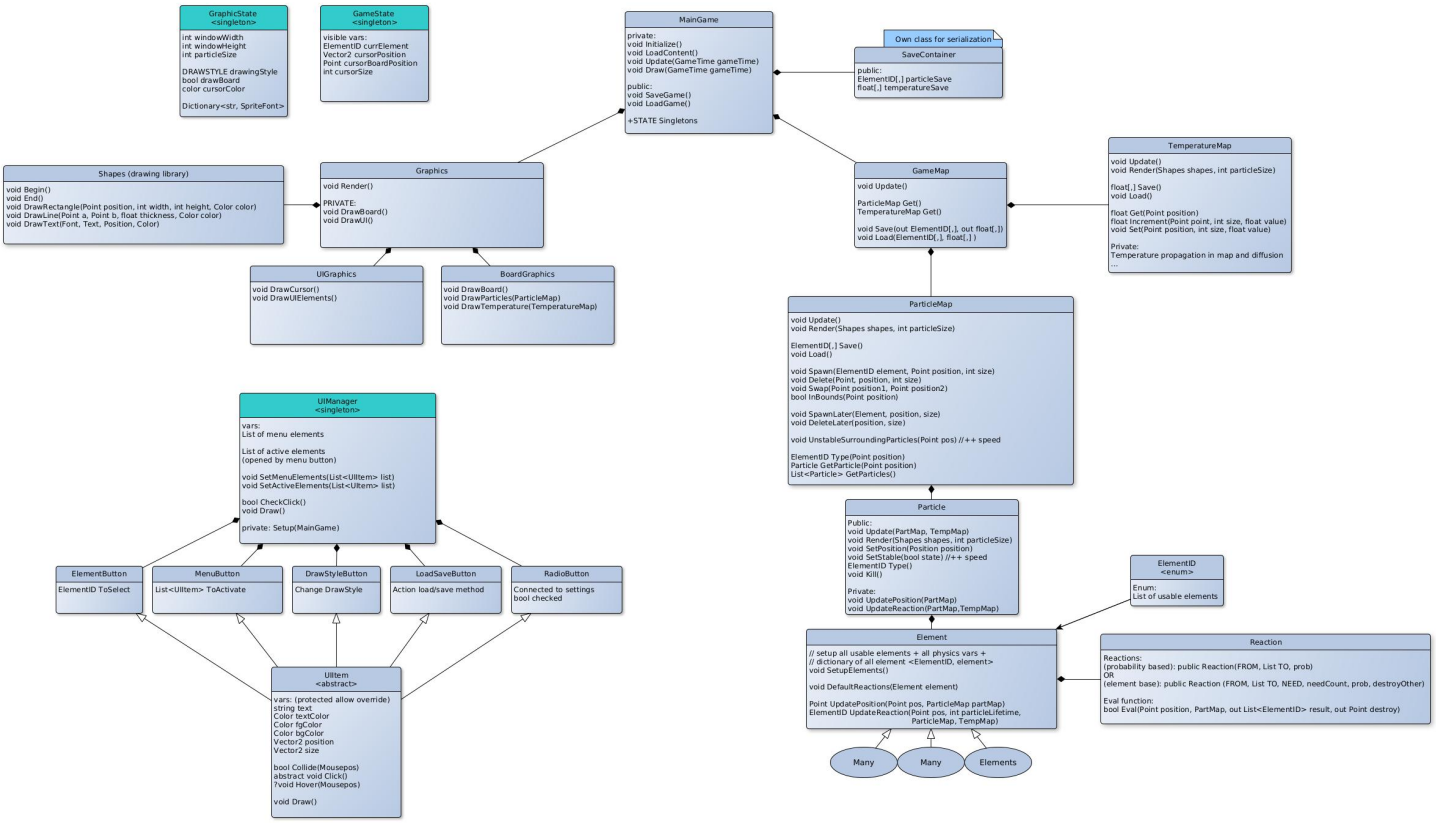
(z finální verze specifikace zápočtového projektu)

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).

Save Container Samostatná malá třída `SaveContainer` používaná jen pro uložení dvou 2D polí. Tato třída slouží jako kontejner pro serializaci. Je prakticky nemožné serializovat složité Monogame struktury jako Vector2 (*alespoň se to zdá nepodporované*). Tato dvě pole ale úplně popisují celý potřebný stav simulace v daném okamžiku tak, že po opětovném načtení se v simulačním prostředí nic nezmění.

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

ii. 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.

iii. 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 celočíselná hodnota (= id) částice uložená v 2D mapě na pozici, na kterém se prvek nachází (*při pohybu se vyměňují na pozicích pouze indexy a částicím se jen předá jejich nová poloha*). 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.

iv. Zpracování vstupu

O samotné zachycení vstupu se stará funkcionalita frameworku Monogame. S ní je možno zachytávat stisky klávesnice a sledovat stavy na myši (stisknutá tlačítka, pohyb, kolečko myši). V třídě `MainGame`, kam tyto vstupy přichází je filtruji podle typu. Některé se řeší přímo na místě, jako třeba kolečko myši. Jiné (stisknutí klávesy) se podle typu klávesy předávají daným třídám. Stavy stisknutí myši se nejprve posílají do `UIManager`, zda-li se nejednalo o interakci s menu objekty a pokud ne, tak se event posílá podle vybraného materiálu do mapy částic/teplot.

III Technická dokumentace

i. Výčet a popis funkcí a procedur

i.i MainGame

Initialize Metoda z frameworku Monogame, využita pro inicializaci grafického rozhraní a u tohoto projektu i všech elementů, map částic a teplot, vlastní třídy pro vykreslování grafiky a třídy starající se o UI.

LoadContent Opět metoda z Monogame, využívána pro načtení dat z externích zdrojů. Zde využita pro načtení předem připravených fontů.

Update Metoda Monogame starající se o aktualizaci logiky a zachytávání vstupů. Volají se odtud *update* metody map a testování na kliknutí do `UIManager`.

Draw Metoda Monogame, starající se o vykreslování. Volá *render* funkci grafické třídy.

SaveGame Vlastní public metoda volaná z tlačítek `UIManager` nebo z klávesové kombinace pro ukládání. Otevírá file explorer a stará se o případné volání speciálních metod třídy `GameMap` pro ukládání map prostředí.

LoadGame Opak `SaveGame`, opět otevírá file explorer, umožňuje vyhledat uložené simulace a pak se stará o volání *load* metod.

i.ii GameMap

Update Funkce sloužící pro předání volání *update* do jednotlivých map částic a teplot.

Save/Load Obě funkce předávající volání z `MainGame` pro uložení/načtení map prostředí. Funkci `Load` se pak ještě jako parametry předávají mapy k načtení.

GetParticleMap/GetTemperatureMap Třída `GameMap` je jediná, která drží vlastní odkazy na jednotlivé mapy. Jelikož jsou ale jejich data často potřebná i jinde (vykreslování, reakce, atd.), tak tyto dvě funkce umožňují získat vlastní odkaz na tyto třídy.

i.iii TemperatureMap

Save/Load Tyto dvě funkce jsou velmi jednoduché, jelikož už samotné teploty jsou ukládány jako dvourozměrné pole floatů, které jde lehce serializovat. S ukládáním a načítáním tedy zde není velký problém.

Get Všechny funkce pracující s nějakou pozicí v mapě si vždy přes privátní funkce `InBounds` kontrolují, jestli je pozice v mapě validní. Pokud je pak tato funkce vrací teplotu na pozici z mapy teplot.

Set/Increment Tyto funkce na základě pozice v mapě a velikosti kurzoru změní teplotu v celé oblasti pod kurzorem uživatele. Rozdíl mezi těmito funkcemi je, že `Set` nastaví teplotu v mapě na parametrem předanou hodnotu (parametr `value`). Funkce `Increment` tuto hodnotu k teplotě v mapě přičtě. V běhu aplikace je aktuálně využívána jen jedna možnost.

Render Funkce volaná z třídy `BoardGraphics`. Díky tomuto je i vykreslování zařízené z kódu starající se o mapu teplot. Stejně to tak má i mapa částic.

Update Metoda starající se o *update* teplot. To je rozdělené do dvou částí - propagace teploty prostředím a rozptýlení tepla v prostředí. Obě dvě činnosti řeší následující dvě `privátní` funkce.

Propagate Propagace tepla v simulaci je velmi zjednodušené, ale i tak dostatečně funkční. Propagace projde všechna místa v mapě, která nejsou zeď a u každého koukne na jeho čtyři sousedy (pouze sousedé přes zeď). Zjištění sousedů zajišťuje další privátní funkce `FindNeighbor`, která vrací pole pozic, kde se sousedé nacházejí. Mezi původní pozicí a sousedy se potom dle rozdílu teplot a vlastnosti rychlosti přenosu tepla materiálu na dané pozici spočte *flow* tepla, které je potřeba přenést. Toto množství je pak ještě upravováno pro rychlost simulace a dále externí hodnotou, která chování tepla upravuje do podoby, kdy se teplo směrem vzhůru šíří rychleji než jinam (*takto jde lehce obejít jinak velmi složité stoupání teplého vzduchu*).

Diffuse Jelikož se herní mapa chová jako uzavřený nepropustný box, který nemá moc velké tepelné kapacity, po chvíli hraní by se vždy stalo, že teplota vzroste na nepoužitelné hodnoty a pak by už nešlo v prosředí nic dělat. Proto je implementovan tento vnější zásah do chování tepla. Každý materiál má tedy umělou tendenci se dostat do své startovní teploty, která je popsána pro každý materiál. Rychlost rozptylování pak samozřejmě závisí na vzdálenosti aktuálního tepla materiálu od startovní teploty. Takto je ale docíleno i po velkém zahřátí celého prostředí postupného ochlazení.

InBounds Již zmíněná funkce kontrolující na základě předané pozice v mapě, zda-li je pozice validní a vrací výsledek.

FindNeighbors Taky již zmíněná funkce, opět na základě pozice v mapě vrací list validních pozic sousedů.

i.iv ParticleMap

Save/Load Funkce podobné dříve zmíněným. Pouze převádí částice uložené v `dictionary` částic, indexovaných pomocí indexů pozic v 2D mapě na 2D pole typu `ElementID`. Toto pole lze poté už lehce uložit. Pro načtení potom stačí na správné indexy vložit uložené typy elementů.

Update Update funkce, která prochází všechny uložené částice typu `particle` v `dictionary` a na každé částici, která není vzduchem, volá její vlastní `Update` metodu. Dále tato funkce vytváří a odebírá částice, které byly označeny k přidání/odebrání během tohoto updatu částic (dictionary se nesmí změnit v průběhu procházení, proto je nutné toto udělat až po dokončení procházení všech částic).

Render Vlastní vykreslovací funkce volaná opět z třídy `BoardGraphics`. I tato funkce prochází všehcny částice a předává odpovědnost za vykreslování do částic samotných.

Spawn/Delete Obě funkce fungují velmi podobně. Na základě pozice kurzoru v mapě, velikosti kurzoru a zvoleného elementu projdou všechny pozice pod kurzorem a (*tam, kde je vzduch - částice se nepřepisují*) vloží na ně zvolený element/odstraní z této pozice libovolný prvek (*existuje toggle v podobě speciálního elementu, který povoluje/zakazuje odstranit stěny - ve hře ERASE=erase all/ERASEP=erase particle*). Zároveň `Spawn` funkce při přidávání volá `TemperatureMap.Set` pro nastavení dané polohy v mapě na startovní teplotu částice.

SpawnLater/DeleteLater Veřejné funkce volané částicemi, když potřebují být odstraneni, nebo potřebují odstranit nějaké jiné. Tyto funkce přidají tyto požadavky do speciálních listů, které se vykonají jako obvyklé `Spawn/Delete` operace na konci `Update` cyklu.

Swap Funkce, volaná z třídy `Particle` při pohybu částic. Vyměňuje id uložené na daných polohách pro jednotlivé částice.

Type Funkce, která vrací typ elementu na dané pozici v mapě, je-li validní.

GetParticle/GetParticleID Téměř totožné funkce vracející buď odkaz na celou třídu částice na dané pozici v mapě, nebo jen ID uložené v mapě částic.

InBounds Stejná jako pro třídu `TemperatureMap`.

UnstableSurroundingParticles Funkce pomáhající optimalizovat běh aplikace. Částice se po chvíli neaktivity uspí. Jakmile se ale s nějakou částicí něco stane (pohyb, reakce, ...) zavolá tuto funkci pro probuzení i všech okolních částic (`nehledě na to jestli byly uspané`). Celá simulace má tendenci se dostat do nějaké rovnováhy, tudíž uspávání částic často šetří velmi drahá volání funkcí pro update polohy a reakcí částic.

i.v Particle

Update Funkce volaná třídou `ParticleMap`. Zde se *update* dělí na dvě části a to aktualizace reakcí a polohy (privátní funkce `UpdateReaction`, `UpdatePosition`).

Render Vlastní funkce pro vykreslování částice do mapy podle své pozice.

Type Funkce umožňující přístup k typu dané částice.

GetPositon/SetPosition Nastavení a získání pozice částice v mapě.

SetStable Nastavení *stability* částice (uspání částice). Používané jak vnitřními funkcemi, tak funkcí `ParticleMap.UnstableSurroundingParticles`.

UpdatePosition/UpdateReaction Obě privátní funkce aktualizující stav (= polohu/element) částice. `UpdatePosition` volá funkci `UpdatePosition` daného typu elementu a očekává návrat nějaké pozice v mapě. Pokud se jedná o tu samou jako na které právě je, neproběhl žádný přesun a prvku se odpočítává daný počet cyklů, dokud nebude uspán. Pokud není, použije funkci `Swap` z `ParticleMap` aby si vyměnil pozice s prvkem, na který se přesouvá a poté probudí všechny prvky v okolí.

`UpdateReaction` také volá funkci `UpdateReaction` pro daný element a podobně jako u pozice čeká jestli se ID jeho materiálu změní nebo Dále zde platí, že pokud je jako výsledek reakce zvolen `ElementID.VOID`, prvek zaniká a pokud `ElementID.EXPLOSION`, tak prvek prochází jinou sadu kroků vytvářející explozi v prostředí podle vlastností svého materiálu.

IV Závěr

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