Budapesti Műszaki és Gazdaságtudományi Egyetem
Villamosmérnöki és Informatikai Kar
Automatizálási és Alkalmazott Informatikai Tanszék

Rittgasszer Ákos

DUNGEON GENERÁLÁS

RAJACSICS TAMÁS

BUDAPEST, 2022

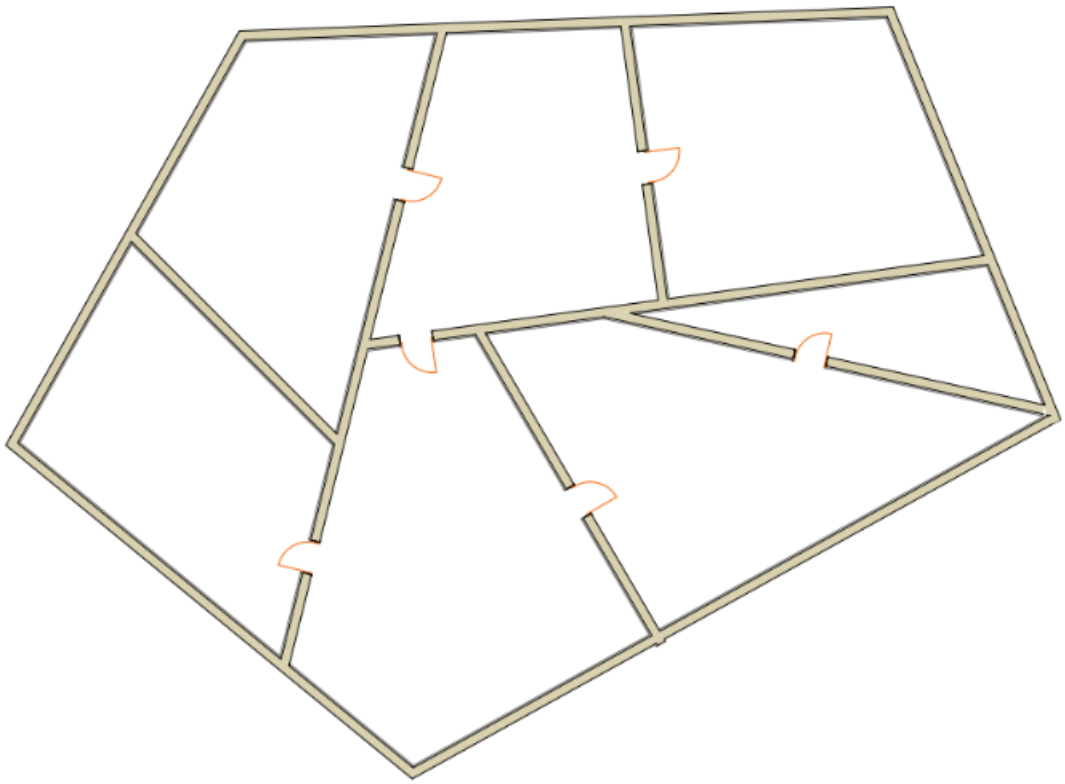
Tartalomjegyzék

1 Feladat

Az alapvető probléma, amire szeretnék megoldást találni, az a dungeon (kazamata) generálás. A megoldás elsősorban játékok készítése során lehet hasznos. A feladat lényege, hogy készítsünk egy szobákból álló rendszert, ahol némely szobák között találhatóak ajtók és az egész egy útvesztőt alkot. A probléma bonyolítása képpen olyan rendszer létrehozása a célom, ahol a szobák nem téglalap, hanem konvex sokszög alakúak.

A feladat két részre bontható. Az egyik a szobák alakjának és a közöttük lévő átjárók meghatározása. A másik pedig egy adott szoba feltöltése, berendezése. Első körben a szobákat és az őket összekötő útvesztőt szeretném létrehozni.

Egy, az előbbit kiegészítő feladatként szeretnék létrehozni egy olyan programot, ami ad egy grafikus felhasználó felületet a generáláshoz. Ezen a felületen fel lehetne paraméterezni a generáló algoritmust és megadni az alaprajzot, amit fel kellene osztani szobákra. Ezen kívül ez a grafikus felület meg is tudná jeleníteni a legenerált dungeon-t.



1. Látványterv

A fentebb látható képen (a kép kézzel rajzolt) ábrázolok egy elképzelést, arról, hogy hogyan is nézne ki egy generált eredmény. A fenti esetben a körvonal és a szobák száma a bemeneti paraméter.

Az eredménynek a bemeneti feltételek teljesítésén kívül még fontos tulajdonsága a szépség. Ez azt jelenti, hogy mivel az elsődleges cél, hogy az eredmény játékokhoz használható legyen, fontos, hogy ez alapján történjen a generálás. Itt olyan dolgokat kell figyelembe venni, hogy a szobák nagyjából ugyanolyan méretűek legyenek, az egyes szobák legyenek szépek és a hozzájuk adott ajtók egy értelmes labirintust határozzanak meg.

Az, hogy egy szoba szép, az abban nyilvánul meg, hogy nincsenek nagyon hegyes szögei és a befoglaló téglalapjának oldalainak aránya minél kisebb. Ahhoz, hogy a szobák szépek lehessenek, fontos követelmény, hogy a megadott körvonal, amiket szobákra kell osztani is egy szép poligont határozzon meg.

Az útvesztő generálásához több algoritmust szeretnék adni, ami közül lehet választani. Ezek közül a legegyszerűbb egy feszítőfát kereső Kruskal algoritmus. Ennek eredménye egy tökéletes labirintust ad, ami azt jelenti, hogy bármely két pontja között pontosan egy út vezet.

Miután sikeresen tudok megfelelő dungeon-t generálni, szeretném azt megjeleníteni. A megjelenítés két célt szolgál. Az egyik, hogy kapjunk egy előnézetet arról, hogy hogyan is néz ki az eredmény és ez alapján el tudjuk dönteni, hogy megfelelő-e nekünk. A másik cél, hogy egy játékhoz fel tudjuk használni.

Ez előbbi megjelenítéshez egy olyan grafikus programot szeretnék, ami egy grafikus felhasználói felületet biztosít az algoritmus használatához. Ezen a felületen szeretném, ha lehetőség lenne az algoritmus felparaméterezésére és futtatására, majd az eredmény megjelenítésére. Mivel az eredmények véletlen generáltak, így ugyanazon beállításokhoz több eredmény is tartozik. Ez a grafikus kezelőfelület felfogható egy kétdimenziós CAD (Computer-aided design) programként.

A másik megjelenítési módnak nem célja, hogy paraméterezhesse az algoritmust. Ez arra való, hogy meg tudjuk jeleníteni egy játékban. Ehhez a tervek szerint valamilyen játékmotort használnék. A megjelenítés történhet két- illetve három dimenzióban, valmint különböző nézetekben, attól függően mi a játék elvárása.

2 Tervezés

A tervezés során először felépítettem, hogy milyen sorrendben akarom elkészíteni a különálló részeket. Itt fontos volt megállapítani a függőségeket. A sorrend a tervek szerint így fog alakulni:

1. Algoritmus dungeon generálására és a szobák feltöltésére
2. 2D megjelenítő és CAD
3. Játékmotorral megjelenítés

A fejlesztés során bizonyosfokú párhuzamosság hasznos tud lenni. Például az algoritmus fejlesztése során segítséget tud nyújtani, ha grafikusan látom a generált eredményt. Ehhez a megjelenítéshez pedig célszerű a CAD programot használni.

2.1 Adatszerkezet

Minden rész szempontjából fontos az adatszerkezet kialakítása. Ez befolyásolja, hogy milyen formátumban érdemes elmenteni az elkészült terveket. Ahhoz, hogy a megfelelő adatszerkezetet ki tudjuk találni, szükséges tudni, hogy milyen struktúrákkal tudnak legegyszerűbb és leghatékonyabban dolgozni az egyes algoritmusok.

Ennek megállapításához célszerű összeszedni azokat az adatokat, amiket fontos tárolnunk. Első körben koncentráljunk a dungeon leírására, a szobák feltöltésével ne foglalkozzunk. Próbáljuk meg elkülöníteni a két problémát.

A dungeon leírásához az alábbiak elengedhetetlen információk:

- szobák száma
- szobák alakja
- szobák helye
- szobák szomszédsága
- átjárók a szobák között, útvesztő

Az előbbiekben felsorolt információk alapján már meg tudjuk határozni a szükséges adatszerkezeti elemeket, amiknek a perzisztenciáját majd meg kell oldani.

A szobák száma egy egyszerű számként tárolható és leginkább csak viszonyításként van rá szükség. A szobák, mivel konvex poligonok leírhatóak a csúcsaik koordinátáival. Itt a zéró koordináta meghatározása okozhat problémát. Ennek megállapítására a praktikus okokból a későbbiekben kerül sor (a megjelenítő fejlesztésekor véglegesedik). A szobák helye megadható külön koordinátával, vagy következtethető a csúcsok koordinátáiból. Az, hogy ezek közül melyik a célszerűbb ugyancsak a megjelenítő fejlesztésekor fog tisztázódni.

A szobák szomszédságának eldöntése megtehető azok csúcsainak ismeretében. Ugyanakkor az információ fontosságának függvényében célszerű lehet az információt nem származtatott adatként kezelni. Ebben az esetben figyelni kell, hogy ne kerüljön ellentmondásba a többi más forrásból származó azonos információkkal.

Az átjárókkal kapcsolatban két fontos információ is van. Az egyik, hogy melyik szobák között létezik út. A másik, hogy ez az ajtó a közös falszakaszon hol található. Az utóbbi tárolása elkerülhető, ha determinisztikusan számolhatóvá tesszük, például mindig a közös falszakasz felezőpontjában található. Ennek eldöntése is későbbi feladat.

Az alábbi problémák megoldásár szükséges algoritmust készíteni. Azok a problémák jelennek meg (teljesség igénye nélkül) amik a dungeon generálásához kellenek, a szobák feltöltésével nem foglalkoztunk. A problémák:

- véletlenszerű, szép konvex poligon generálása
- poligon osztása két szép poligonra, a megadott arányban
- labirintus készítése a szobák között

A szükséges algoritmusok és a tárolandó adatok figyelembevételével az alábbi fontosabb adattárolókra és modellekre biztosan szükség lesz (teljesség igénye nélkül):

- poligon
- szoba
- dungeon
- vektor (2D és 3D biztos)
- gráf

Az előbbiekben felsorolt első három elem az algoritmusok és logikák szempontjából fontos elemek. Mindegyike egy-egy logikai fogalomért felel, amikre a

későbbiekben hivatkozunk, mint valós és mint leképezett fogalom. Az utóbbi kettő adattárolás szempontjából érdekes. Itt különösen fontos a gráf. Ennek előnye, hogy több ismert gráfelméleti algoritmus használata nagyban megegyszerűsítheti a dolgunkat. Ilyen például a már említett Kruskal algoritmus, amivel így egyszerű lesz egy egyszerű útvesztő generálása.

2.2 Modulok

A következőkben az elkészítendő modulokról lesz felületesen szó. Elsősorban, hogy melyiknek milyen felelőssége van és hogy hogyan állnak kapcsolatban egymással, milyen szinten helyezkednek el a modul hierarchiában.

A moduloknál fontos, hogy minél függetlenebbek legyenek és így minél könnyebben lehessen őket változtatni. Ez úgy érhető el, ha mindegyik rendelkezik egy publikus API-val (application programming interface). Ez lehetővé teszi, hogy a belső változtatások minél kevésbé érintsék a használati helyeket.

Az előbb említettek nem csak a modulokra, hanem az egyes osztályokra is igazak. Alapvetően első körben szeretném én megírni minden osztály implementációját. Ennek oka, hogy szeretnék minél inkább képbe kerülni, hogy mire is van szükség és hogyan lehet a leghatékonyabb megoldást találni. Ez azonban lehet, hogy egy idő után nem lesz fenntartható. Erre példa lehet a gráf osztály, aminek rengeteg bonyolult algoritmusa létezek és ezek mindegyikének hatékony implementálása nem biztos, hogy része a feladatnak. Amennyiben eljutunk egy olyan állapotban, hogy külső megoldásra kell lecserélni a sajátomat előjöhet rengeteg probléma a használati helyeken. Ezt megoldandóan minden ilyen osztályt elhelyezek egy wrapper osztályba, ami biztosítja az egységes interface-t és lehetővé teszi a belső implementáció változtatását, a külvilág tudta nélkül.

Előreláthatólag az alábbi modulokra lesz szükség:

- Matematika
- Geometria
- DungeonGenerator
- Base

A matematikai modul felel az alapvető matematikai fogalmakért. Ilyen például a vektor, az egyenletrendszer vagy a gráf. Ez az egyik legalapvetőbb modul, a legtöbb másik használja.

A geometriai modulban kiegészítjük a matematikai fogalmakat. Itt kerül definiálásra például a poligon vagy a geometriai transzformációk. Ebben a modulban megjelennek olyan tudások, amik a generáló algoritmusra specifikusak. Például a képesség, hogy egy poligont felosszunk két megfelelő poligonra.

A Base modul olyan általános tudásokat tartalmaz, amik mindenhol szükségesek lehetnek. Ilyen például a logolás vagy a guard.

A DungeonGenerator a többi segítségével már konkrét feladatbeli fogalmakat definiál és ezekkel dolgozik. Ilyen például a labirintus vagy a szoba.

3 Fejlesztés

A fejlesztés C++ nyelven történik. A fejlesztés során szeretném minél jobban kihasználni a szabvány által biztosított lehetőségeket. Ezért c++20-as szabványt alkalmazva dolgozok.

A fejlesztés első lépéseként a matematikai és geometriai modul elkészítése a cél. Ezek mellett a Base modulba olyan funkciók kerülnek, amik segítik a fejlesztést és hibakeresést.

3.1 Base modul

A Base modulba eddig két funkcionalitás került. Ez a logolás és a guard. A logoláshoz az spdlog könyvtárat használtam. A guard funkció lényege, hogy a guard példányosítása és a blokk vége között lehet valamilyen védelmet adni. Erre példa a memóriaszivárgások ellenőrzése.

A modul funkcionalitás folyamatosan kiegészül a felmerülő szükséges tudásokkal. Olyan átfogó funkciók kerülhetnek ide, amik a kód minden részén hasznosak lehetnek. Ezek elsősorban a fejlesztést és a hibakeresést elősegítő funkciók.

3.2 Matematikai modul

A matematikai modulban a matematikai alapfogalmak találhatóak. Itt elsőkörben szeretném a saját implementációimat használni. Azonban fennáll az a lehetőség, hogy a saját implementációim tudása nem tudja követni az algoritmusok igényeit. Erre felkészülésként ezeket az implementációkat elrejtem egy-egy wrapper mögé. Ez lehetőséget ad arra, hogy a későbbiekben könnyebben le lehessen cserélni harmadik fél által készített megoldásra.

A modulon belül érdekesebb osztály a vektor. Ez felelős n dimenziós vektorok tárolásáért. Az implementálásnál megemlítendő érdekesség, hogy a Vec egy template paraméterekkel rendelkező osztály. Az első template paraméter a vektor dimenziója, ez egy pozitív egész szám. A másik paramétere egy típus, ami a vektorban tárolandó típust határozza meg. Ennek kapcsán az érdekesség, hogy a típusra tettem egy megkötést. Ezt a megkötést a c++20 szabványban bevezetett concept fogalom segítségével tettem.

Ezen kívül említésre méltó lehet az egyenletrendszerek megoldásáért felelős osztály. Ennek a neve LinearEquationSystem. Ez is egy template-es osztály, két paraméterrel. Ez a két paraméter a változók és az egyenletrendszerek száma. A megoldáshoz pedig Gauss eliminációt használ. A Gauss elimináció elvégzésében a mátrixokért felelős Mátrix osztály segít.

Az említetteken kívül még szögekért, kvaterniókért és természetes számokért felelős osztály készült.

3.3 Geometriai modul

A geometriai modul a matematikai modul segítségével definiál geometriai fogalmakat. Ilyen például a poligon vagy a transzformáció.

Az itt létrehozott adatszerkezetek már olyan tudásokat is tartalmaznak ami a generáláshoz is szüksége. Például itt lehet egy poligon kettéosztani a megadott arányban.

Még nem készült el a teljes funkcionalitása. Ennek oka, hogy a matematikai modul hiányosságai ezt nem teszik lehetővé.

3.4 DungeonGenerator modul

Ez felelős a dungeon generálásáért és a hozzá szükséges fogalmak biztosításáért. Ilyen fogalmak a szoba, a labirintus vagy a generálási beállítások. Egyelőre implementációval még nem nagyon rendelkeznek. Ennek oka, hogy még nem készült el ehhez minden alacsonyabb modulbeli adatszerkezet.

3.5 Egyéb

Fontos kérdés még, hogy a különböző modulok hogyan használják egymást. Erre az jelenleg az a megoldás, hogy statikus könyvtárként fordulnak le. Ezeket aztán a magasabb szintű modulok használhatják.

Cél a későbbiekben, hogy ezt a projektszerkezetet ne manuálisan kelljen összeállítani. Erre megoldást jelent valamilyen külső alkalmazás. Ezek közül eddig kettő merült fel. Ezek a Premake és CMake. Arról még nem hoztam döntést, hogy melyiket fogom beépíteni.

Verziókövetésként git-et használok. Ez könnyebbé teszi számomra a fejlesztést. Az egyetlen probléma vele, hogy amíg az előbbiekben említett projektgeneráló nincs beépítve nem tudom kényelmesen használni.

Fontos kérdés még a dokumentáció. Ezt igyekszem minél naprakészebben tartani. Ez nekem is segít, illetve esetleges más fejlesztők vagy felhasználók is tudnak belőle tájékozódni. Két féle dokumentáció készül. Az egyik a fejlesztők számára, a projektek felépítéséről, kapcsolatáról és a különböző osztályok felelősségeiről. A másik a felhasználókat hivatott segíteni.

4 Tesztelés

A tesztelés modulonként unit tesztekkel történik. Ezek a unit tesztek a különálló részek funkcionális helyességét hívatottak biztosítani.

Minél alacsonyabb szintű funkció tesztjéről van szó, annál fontosabb az átfogó tesztelés. Ennek oka, hogy az ezek által okozott hibák magasabb szinten szoktak előjönni és ekkor nehezebb lokalizálni a hiba okát.

Mivel a generált dungeon véletlenszerű, ezért ennek tesztelése nem egyszerű. Ehhez segédeszköz a CAD program, amin keresztül vizuálisan tudjuk ellenőrizni az eredményt. Ez elsősorban a helyes paraméterezésben tud segíteni egy már funkcionálisan működő algoritmusnál.

A kód minőség fenntartásához statikus analizáló programot használok. Erre a Visual Studio beépített megoldását használom.

5 Továbbfejlesztés

A program fejlesztése kezdeti stádiumban van, így rengeteg továbbfejlesztési lehetőségről tudunk beszélni. Itt elsősorban azokat szeretném megemlíteni, amik nem elsődleges céljai a feladatnak.

Ilyen lehetőség a háromdimenziós labirintus. Ez azt jelenti, hogy nem csak egy szinten helyezkednek el a szobák, hanem egymás fölött is. Az egymás fölött elhelyezkedő szobák között pedig a padlókon és plafonokon található ajtókon keresztül átjárni.

Az is egy lehetséges kiegészítés, hogy az algoritmus paraméterezéséhez és az eredmény megtekintéséhez egy webes felület készül. Ennek érdekében igyekszem a generálás interface-ét minél tisztábban és platformfüggetlenül megtartani. Ez teszi lehetővé, hogy később tetszőleges környezetben fel tudjam használni.

A generálásra többféle algoritmust fejlesztése is egy olyan lehetőség, amivel lehet továbbfejleszteni az alkalmazást.

További lehetőség, hogy a generálás kapjon egy nehézségi szintet, ami alapján tudja meghatározni az eredmény bonyolultságát. Ez jelentheti akár a szobák számának növelését, akár az útvesztő bonyolítását. Befolyásolhatja az egyes szobák berendezését is.

Az említett lehetőségeken kívül még számtalan további fejlesztési lehetőség és ötletet lehet kitalálni. Igyekszem az ilyen fajta kiegészítésekre felkészíteni a kódbázist.

6 Összefoglalás

Összefoglalva a projekt még nagyon kezdetleges állapotban van. Egyelőre körvonalazódott egy rövid és hosszútávú cél. Ezeket a célokat tudom figyelembe venni a fejlesztés során, ha valamilyen döntést kell hozni.

Már előfordultak olyan lépések, amiket az elején kihagytam és pótlásuk sokkal nagyobb munkának bizonyult, mint gondoltam. Ezeket szeretném a jövőben elkerülni és mindent időben elkészíteni.

Amire nagyon kell figyelnem, hogy megtaláljam az egyensúlyt az osztályok és modulok megfelelő szervezése között, úgy, hogy nem teszem feleslegesen érthetetlenné az implementációkat.

Egy nagy nehézség lesz a komplexebb algoritmusok implementálása. Itt a legfontosabb, hogy megfelelő adatszerkezeteket használjak. Az is nagyon fontos, hogy megfelelően tagoljam az algoritmusokat, ezzel segítve az átláthatóságot.

Fontos, hogy a végső algoritmus úgy legyen kialakítva, hogy könnyen lehessen paraméterezni. Ez azért kell, hogy mindenki elérhesse az általa várt eredményt. Emellett legyen meg az az interface-e, amin keresztül könnyen meg lehet hajtani.

7 Források

- spdlog: https://github.com/gabime/spdlog
- concept: https://en.cppreference.com/w/cpp/language/constraints
- Premake: https://premake.github.io/
- CMake: https://cmake.org/