# Nagyházi feladat

Programozás alapjai 2.

Ottó Lőrinc
L5N83Q 2023.05.28

# Dokumentáció

## Specifikáció

Repülős játék elkészítése, ahol a játékos megadja egy ország nevét, majd egy grafikus felületen egy repülő irányításával bejárhatja az adott ország pályáját. A pályán legyenek megfigyelhetők különböző, természetes kinézetű biomok! A pályát ne tárolja a számítógépen hanem a névből futásidőben kell generálnia a programnak. Az összes olyan országnévből képes kell legyen a program pályát generálni, amik az alábbi kritériumnak megfelelnek: angol ábécé karaktereiből illetve a '-' karakterből áll, és nem hosszabb 32 karakternél. A program a kis- és nagybetűk között ne tegyen különbséget. Az irányítás a WASD billentyűkkel történjen!

## Feladatspecifikáció

A program az ország nevéből, ami egy karaktersorozat, előállít egy seed-et, ez lesz a generálás alapja. A pálya 256*256 os csempéből fog állni. Perlin-zaj segítségével mindegyik csempéhez hozzárendeljük, hogy milyen biomba tartozik, majd attól a biomtól függően különböző mód színezzük a csempéket. Majd miután ez megtörtént, a csempékből exportálunk egy képet, amit betöltünk a grafikus felületbe, ahonnan indulhat a játék.

## Felhasználói specifikáció

### Belépés a játékba

A program futtatásakor egy konzol ablak várja a felhasználót. Ilyenkor a program egy 32 karakter hosszú, angol ábécé betűiből álló karaktersorozatot vár, ez a pálya kódja. Ekkor a tetszőleges kód alapján a program generál egy pályát, és kezdődhet a játék. (Ugyanazon kódra mindig ugyanazt a pályát kapjuk, a program kis és nagybetű között tesz különbséget)

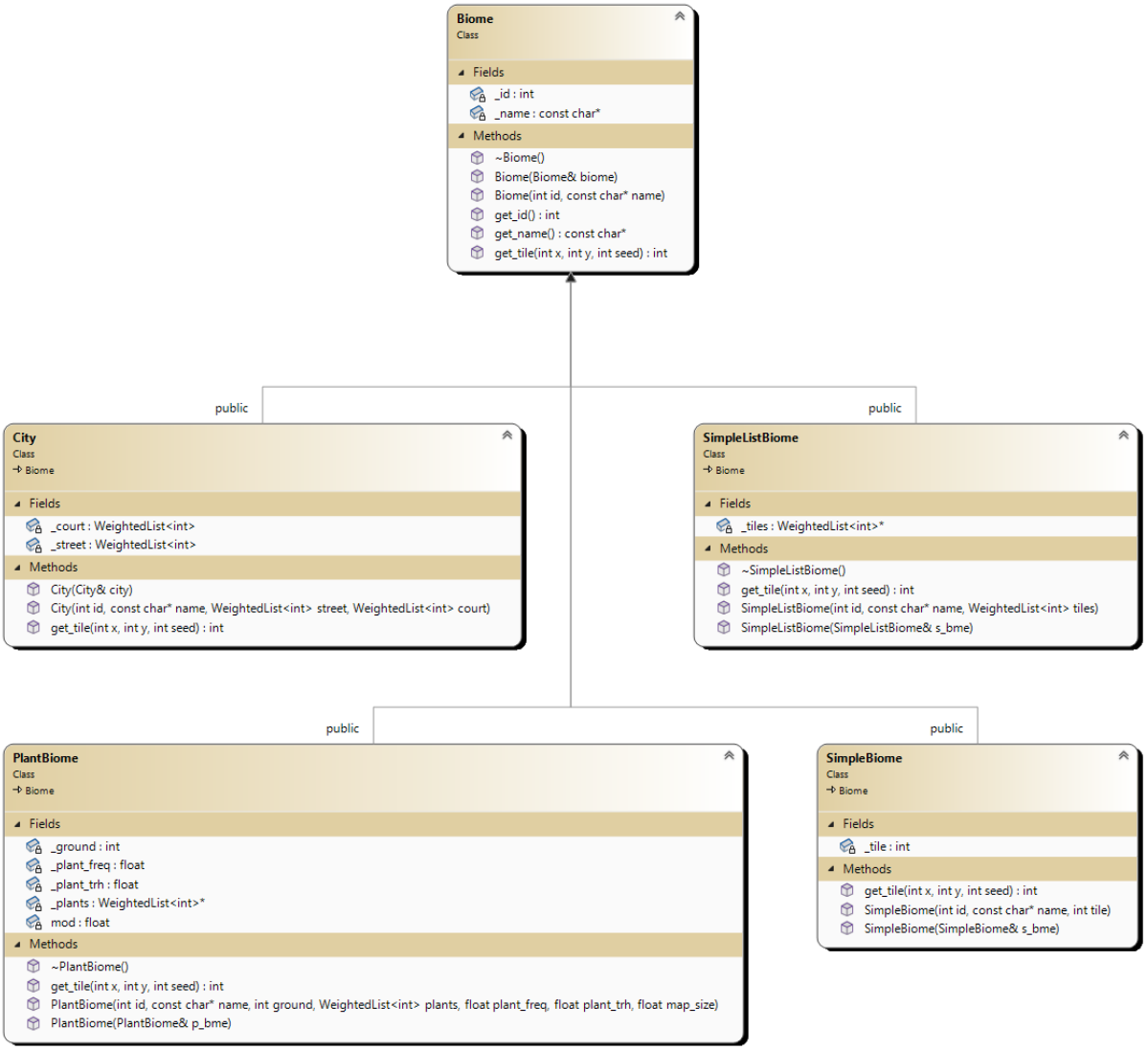
### Irányítás

A repülő irányát az A (balra) és D (jobbra) billentyűkkel tudjuk állítani, a sebességét pedig a W (gyorsul) és D (lassul) billentyűkkel.
Az ablak bezárásával illetve az Esc billentyűvel tudunk kilépni a játékból.

# Programozói Dokumentáció

## Biome osztályok

A biomokat osztályokként kell megvalósítani, amiknek közös őse lesz a Biome osztály. Rendelkezniük kell egy függvénnyel, ami megadja egy adott csempe az x y koordinátán milyen legyen.



**Biome**
Ez az ősosztály, az összes Biome tőle örököl, így egy heterogén kollekcióban tudjuk tárolni. Legfontosabb függvénye a

- `virtual int get_tile(int x, int y, int seed)`
  Ez a függvényt örökli az összes Biome, lényege, hogy az pozíció, és az aktuális seed alapján meghatározza, milyen csempét generáljon a játék.

Az ID, és a név egyedi azonosítók, azonban ezeket a program nem használja.

**SimpleBiome**
A pozíciótól és a seedtől függetlenül a _tile csempét adja vissza:

- `int get_tile(int x, int y, int seed) { return _tile; }`

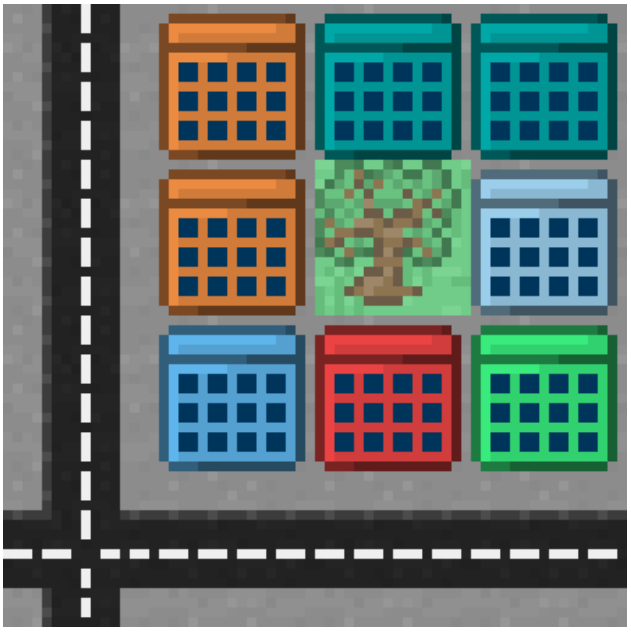
**SimpleListBiome**
A pozíciótól és a seedből generált hash alapján választ a _tile-ból:

- ```
  int get_tile(int x, int y, int seed) {
        return _tiles->get_rand(Hash::get_int(x, y, seed));
  }
  ```

**PlantBiome**
Amennyiben megugorja a küszöböt a perlin-zaj, a virágok közüli, más esetben a _ground csempét adja vissza;

- ```
  int get_tile(int x, int y, int seed) {
        if (Perlin::get_float(x / _plant_freq, y / _plant_freq, mod,
  seed) < _plant_trh)
              return _ground;
  }
  ```

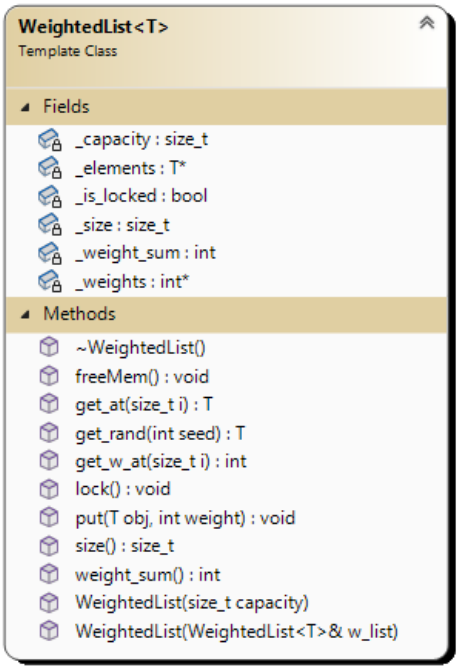
**City**
A 4-el osztható mezőkre utat, a többire házat és udvart rak:

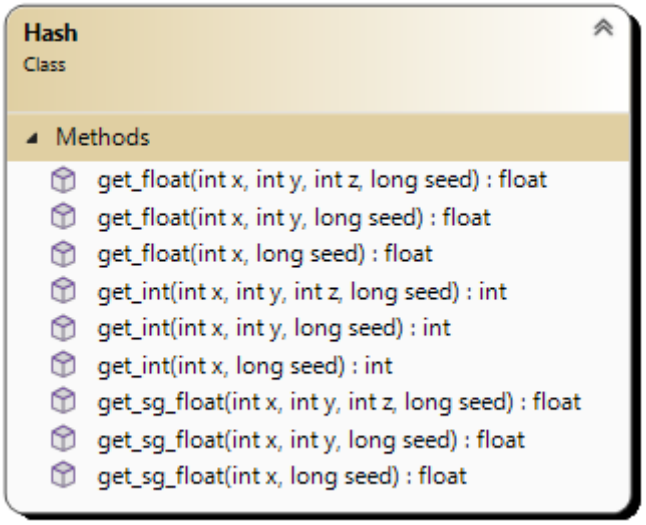


*Ábra a generáláshoz*

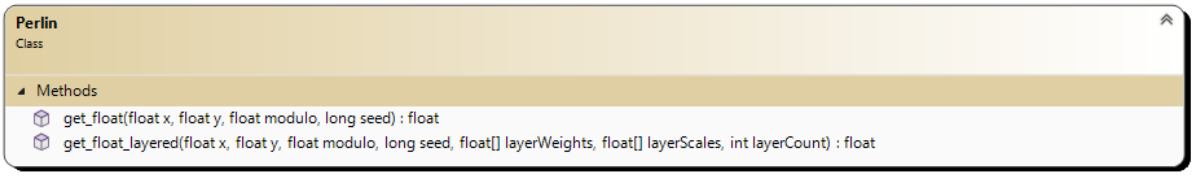
## WeightedList<T>

A súlyozott lista egy template-s osztály. Heterogén kollekció, mely minden elem mellé eltárolja a hozzátartozó súlyt is. Ennek lényege, hogy a listából véletlenszerűen tudjunk egy elemet kiválasztani úgy, hogy egy elem kiválasztásának esélye arányos legyen súlyának nagyságával. Ezt a funkcionalitást a aget_rand függvénnyel érjük el. Csak a program futtatásához szükséges funkciói vannak elkészítve, így például elemet törölni nem tudunk.



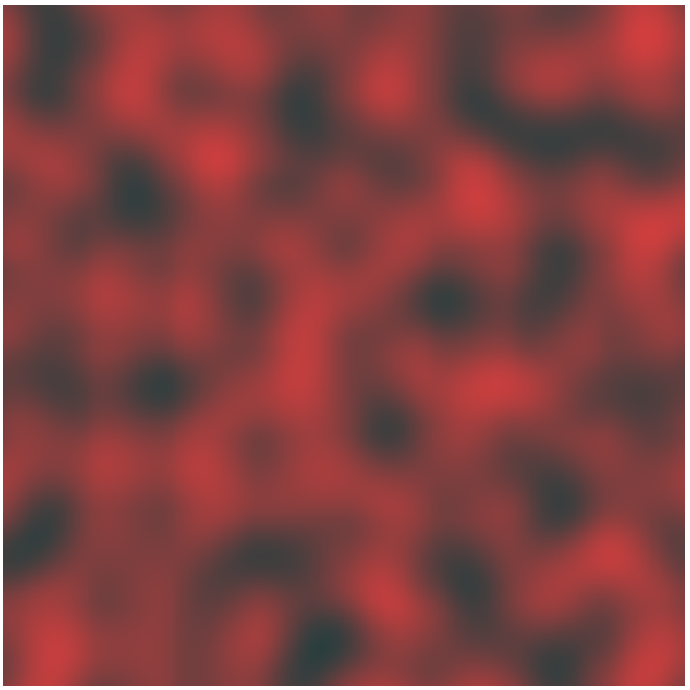

- ```
  T get_rand(int seed) {
        int rem = p_mod(seed, _weight_sum);
        int checked_w = 0;
        for (size_t i = 0; i < _size; i++) {
              checked_w += _weights[i];
              if (checked_w > rem) return _elements[i];
        }
        return _elements[_size - 1];
  }
  ```
- A függvény működési elve, hogy addig megy a listában, míg a véletlen számnál (seed) legnagyobb nála kisebb súlyt megtalálja, majd visszatér azzal az elemmel.
  ```

## Hash

A Hash osztály statikus függvényeket tartalmaz. A függvények lényege, hogy koordináták és egy seed alapján egyértelmű hozzárendeléssel (ugyanazon paraméterekre ugyanaz a kimenet) rendeljen véletlen számokat. Ezt a bitek összekeverésével érjük el.




Létezik függvény 1, 2 és 3 dimenziós pozícióra is, és ezekből is három fajta van:

- `get_int`, ami integer visszatérésű,
- `get_float`, ami ]0;1[ közötti float,
- `get_sg_float`, ami ]-1; 1[ közötti float visszatérésű.

Alább látható a 2 dimenziós integerrel visszatérő függvény:

- ```
  int Hash::get_int(int x, int y, long seed) {
        int _seed = (int)seed;
        _seed ^= hash_x_prime * x;
        _seed ^= hash_y_prime * y;

        return _seed * _seed * _seed;
  }
  ```
- A kívánt pszeudo-random számot XOR-al és szorzással érjük el, mivel ezek után túlcsordul az int, így véletlenszerű eredmények keletkeznek.
- A float-os verzió elve egyszerű: egy const float-tal megszorozzuk az integer eredményt, így az a megfelelő intervallumba kerül:
- float Hash::get_float(int x, int y, long seed) { return get_int(seed, x, y) * float_2_int + 0.5f; }

## Perlin

A Perlin osztály szintén statikus függvényt tartalmaz. Ennek a zajnak az a lényege, hogy egyértelmű hozzárendeléssel adjon a Perlin-zaj elvei alapján véletlen, azonban monoton számot. Ezzel érjük el, hogy a biomok határai a természethez hasonlóan ne egyenesek legyenek. Fontos tulajdonsága, hogy ismétlődik a határain, így feltünésmentes az átmenet.
  ```




Legfontosabb függvénye:

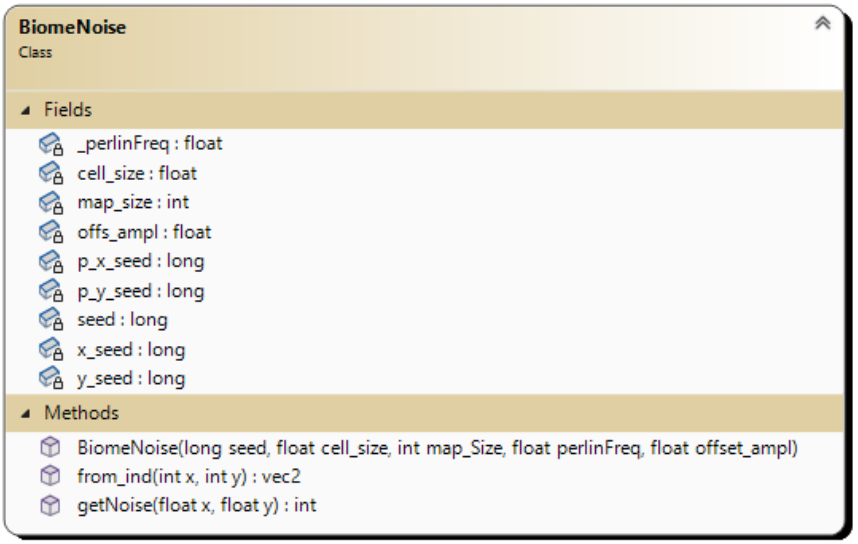
- `static float get_float(float x, float y, float modulo, long seed);` ahol a modulo a pálya nagyságát takarja, ez adja meg, hogy mikortól ismétlődjön.



*A perlin zaj vizualizálva, látható, hogy monoton*

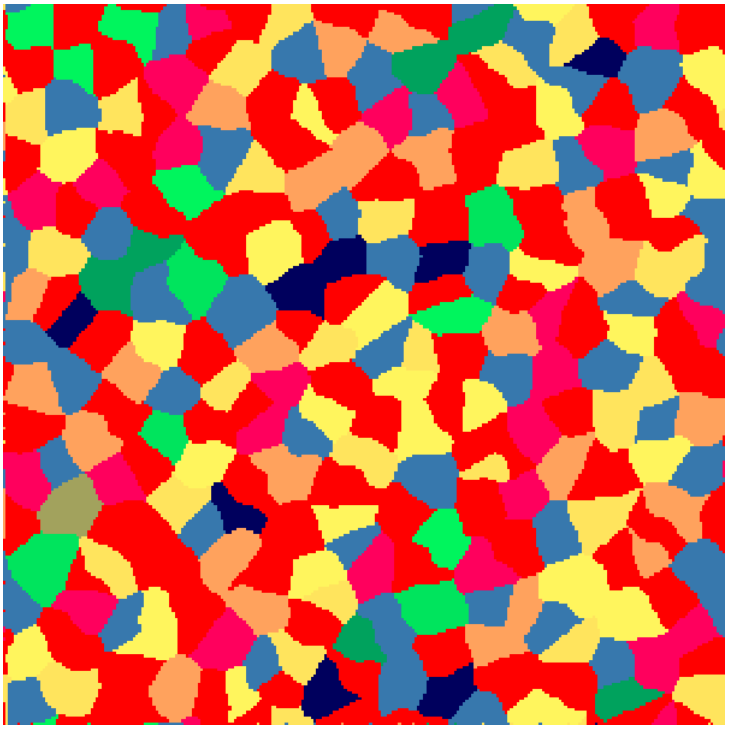
## BiomeNoise

A BiomeNoise a generálás lelke. Lényege, hogy a síkot felosztja régiókra, amik határai nem egyenesek. Ezt a Hash és a Perlin függvényeivel érjük el.



Legfontosabb függvénye:

- `int getNoise(float x, float y)`



*A BiomeNoise vizualizációja, minden szín egy Biome-ot jelképez*

## OpenGLObj

Az OpenGLObj osztály felelős a grafikus renderelésért (így a jPorta elől rejtve marad). Ez az osztály magába zárja a grafikus elemeket, hogy megfeleljen az OO elveknek.

## util.hpp

Az util.hpp egyéb segédfüggvényeket tartalmaz.