Rakenteiden valinnat

Koko ohjelma on pyritty luomaan niin, että kaikki tietoalkiot ovat tietorakenteessa townmap_, ja kaikki muut vectorit vain säilyttävät ja järjestelevät näihin viittaavia pointtereita.

Struct Towndata

```
struct TownData
{
    TownID id = NO_ID;
    std::string name = NO_NAME;
    int x = NO_VALUE;
    int y = NO_VALUE;
    int tax = NO_VALUE;
    TownData* host = nullptr;
    std::map<TownID, TownData*> vassalsDATA = {};
    std::vector<TownID> vassalsID = {};
};
```

Towndata structi mahdollistaa tiettyyn kaupunkiin liitettävän tiedon tallettamisen yhteen paikkaan. Vassal datan pitäminen mapissa antaa mahdollisuuden kutsua tiettyä vassalia pelkällä ID:llä ja käsitellä tätä helposti, sillä arvona on pointteri myöhemmin esiteltävään townmap_ tietorakenteeseen. VassalsID vectori taas antaa mahdollisuuden palauttaa kaikkien vassalien ID:t vektori muodossa helposti suoraan muistista.

townmap_

```
// map of all towns corresponding to their IDs
std::map<TownID, TownData*> townmap_ = {};
```

townmap_ on koko ohjelman päätietorakenne, muut vectorit säilyttävät vain pointtereita tämän tietorakenteen alkioihin. Taas tiedon säilöminen TownID:n taakse antaa mahdollisuuden kutsua alkioita suoraan ID:llä iteroimatta turhaan. Tämä on myös tehokkaampaa kuin ID:n etsiminen esim. vectorista.

townidvector_

```
// vector of town ids
std::vector<TownID> townidvector_ = {};
```

townidvector_ säilyttää townmap_in sisältämien alkioiden ID:t helposti saatavana vectorina, ja toimii paluu arvona muutamissa funktioissa, jotka eivät välitä järjestyksestä.

Alphalistat

```
// vector of town ids in alphapetical order
// and the corresponding sortflag
bool alphaflag_ = false;
std::vector<TownData*> alphasortedvectorDATA_ = {};
std::vector<TownID> alphasortedvectorID_ = {};
```

alphasortedvectorDATA säilyttää pointtereita townmap_ tietorakenteeseen ja alphasortedvectorID_ taas samojen alkioiden ID:t aakkosjärjestettynä. ID vectori toimii muistissa olevana helposti saatavana järjestetynä vectorina, ja DATA vectorilla on helppo pääsy townmap_ tietorakenteeseen, samalla ollessaan järjestetty versio townmap_-rakenteesta, ilman valtavia muistikuluja.

Distancelistat

```
// vector of town ids in distance order
bool distflag_ = false;
std::vector<TownData*> distsortedvectorDATA_ = {};
std::vector<TownID> distsortedvectorID_ = {};

// vector of town ids in distance order related to a given X and Y
std::vector<TownData*> distCOMPLEXsortedvectorDATA_ = {};
std::vector<TownID> distCOMPLEXsortedvectorID_ = {};
```

Jälleen samat muistiin talletus tavat. DATA vectorit viittaavat jälleen townmap_ tietorakenteeseen pelkillä pointtereilla, ja pointterit on tällä kertaa järjestetty matkan mukaan. ID vectorit ovat valmiina kun niitä funktioissa tarvitaan useasti.

min- ja maxdist

```
// min and max distance pointers
TownData* mindist_ = nullptr;
TownData* maxdist_ = nullptr;
```

Muistiin talletetut mindist_ ja maxdist_ saavat aikaan sen, ettei distance listaa tarvitse välttämättä sortata turhaan, sillä min- ja maxdist_ talletetaan add_towning yhteydessä.