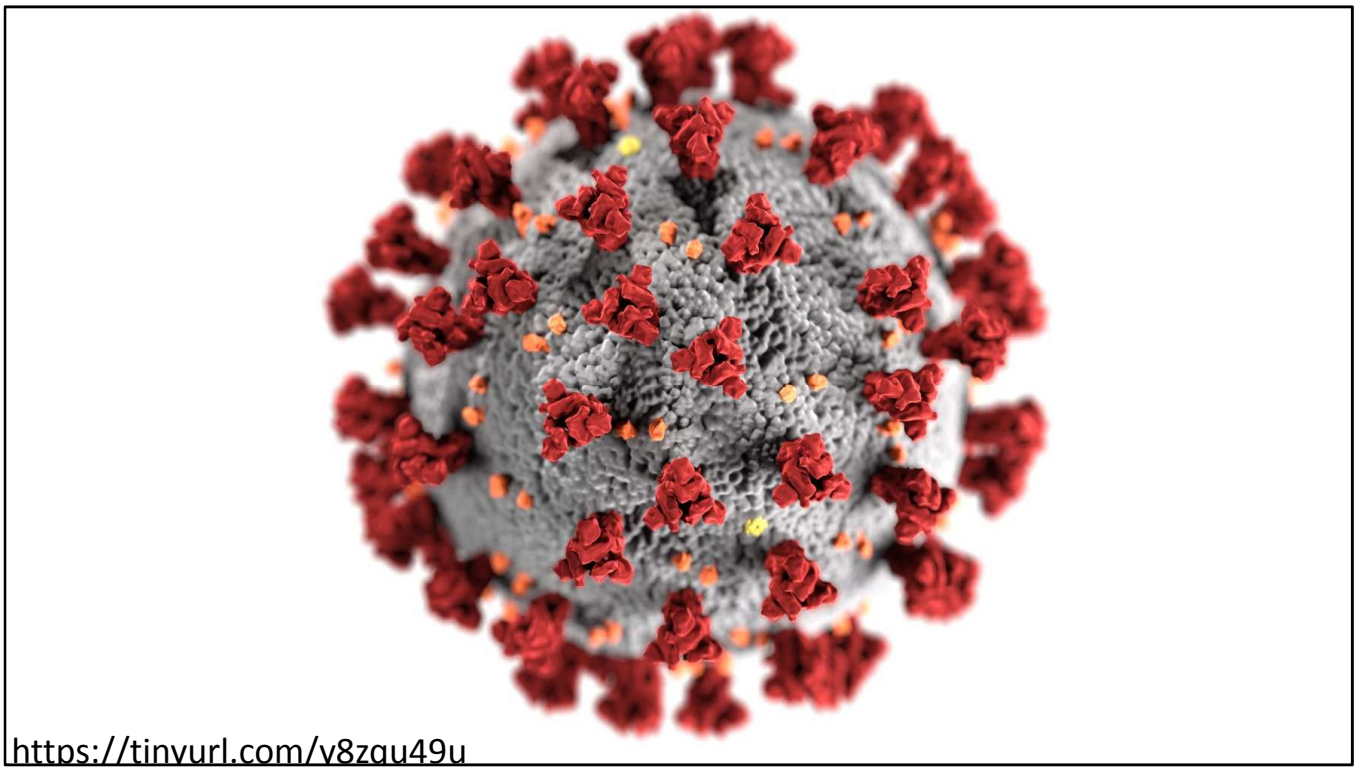
Covid-19 Spread Simulation

Aviv Yehoshua, Sander Mutsaerts, Joshua Moelans

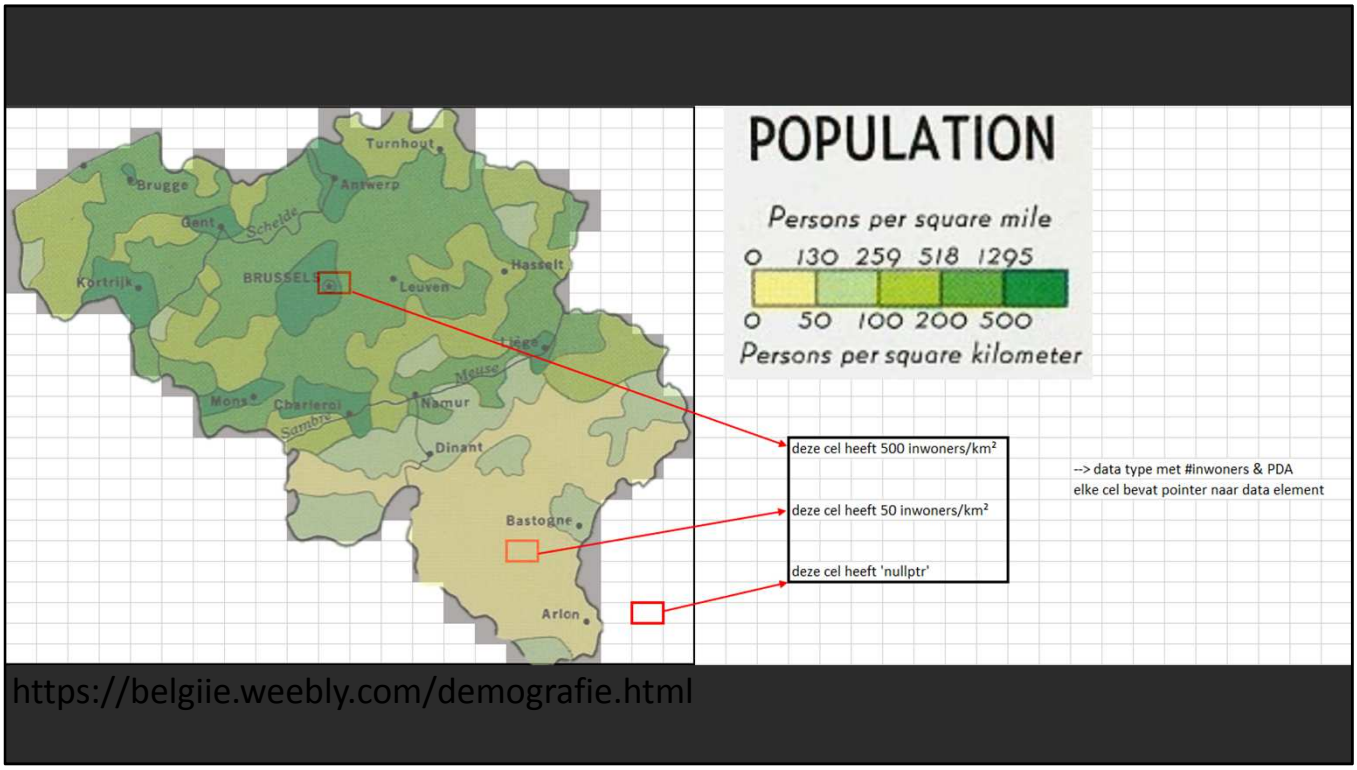
Geachte aanwezigen op deze conferentie. Joshua, Sander en ik gaan in dit filmpje ons onderwerp voor het groepswerk van talen en automaten voorstellen. Eerst gaan we bespreken wat ons project inhoudt en wat de motivatie erachter is. Dan gaan we het hebben over ons manier van werken en de implementatie van de features. Ten slotte gaat Joshua een demo van onze code tonen.



https://tinyurl.com/y8zgu49u

Het coronavirus heeft in een mum van tijd een enorme impact gehad op de levens van ons allemaal. Ook al zijn de maatregelen streng, worden ze niet even goed opgevolgd door iedereen. Daarom willen wij met onze Covid-19 simulator, mensen bewust maken van de ernst van de verspreding.

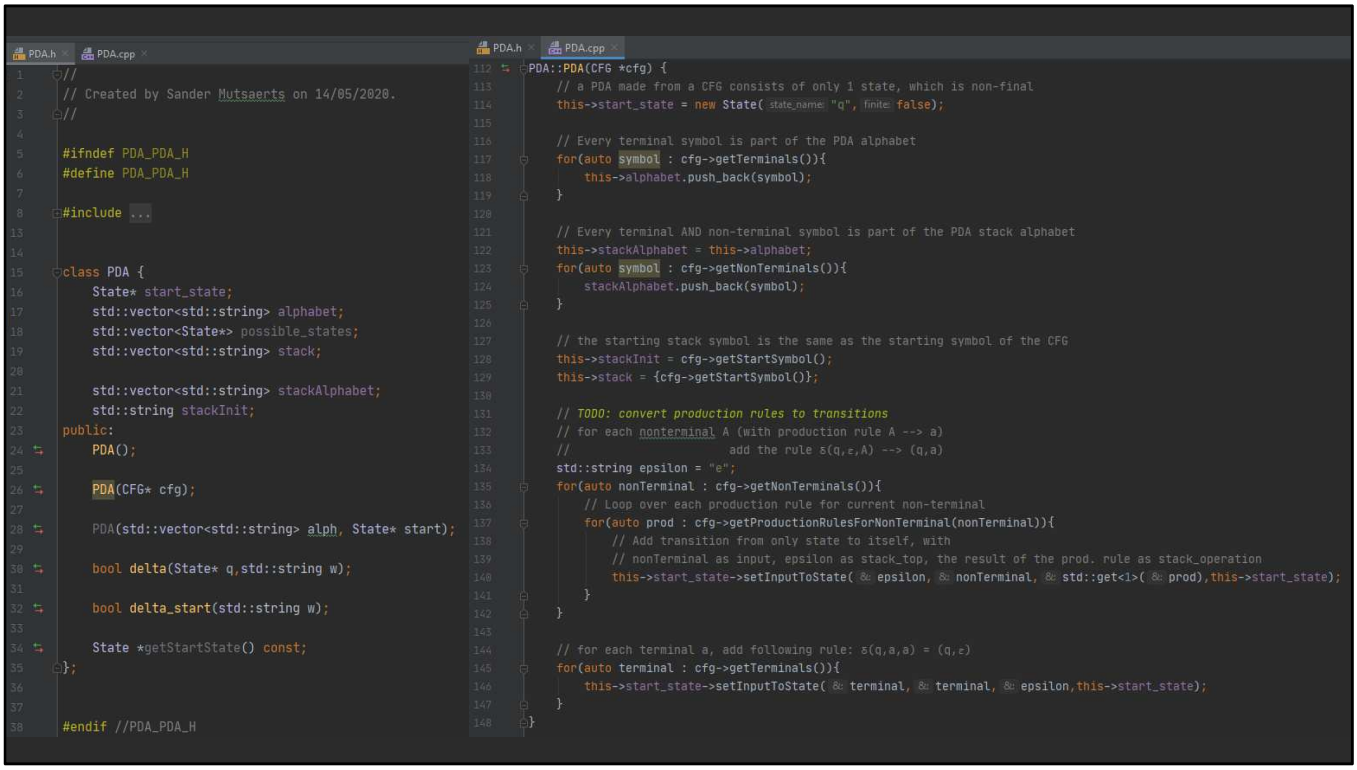
Als deze pandemie voorkomen kon worden, zou de gezondheid van de maatschappij beter verzekerd zijn. In onze simulatie gaan wij cellular automata implementeren om de verspreiding van het virus aan te tonen, aan de hand van meerdere features. Dit soort simulator is relevant omdat het duidelijker wordt in welke elementen er meer of minder moet worden geinvesteerd, om de verdere verspreiding van het virus te voorkomen.



https://belgiie.weebly.com/demografie.html

Wij hebben gekozen om Belgie voor te stellen in een 11 op 13 matrix van cells.
Elke cel in onze matrix stelt een regio voor in Belgie.
Een cel bestaat uit aantal individuen, hoe meer dichtbevolkt de regio, hoe meer individuen er in een cel zijn. Indien het aantal individuen gelijk is aan 0, is de regio onbewoond. Een cel kan geinfecteerdt zijn door het virus.
In onze simulation beginnen we telkens met een startfase waarbij 3 cellen geinfecteerd zijn. Bij elke iteratie over onze matrix bepalen wij of een cel geinfecteerd is of niet.
Een cel is geinfecteerd als 3 van zijn buurcellen

geinfecteerd zijn. Als de cel al geinfecteerd is, worden het aantal individuen die geinfecteerd zijn in de cel bijgewerkt aan de hand van de features.
Na enkele iteraties kan men al duidelijk een evolutie zien in de verspreiding van het virus. Joshua gaat deze op het einde tonen.



```
//
// Created by Sander Mutsaerts on 14/05/2020.
//

#ifndef PDA_PDA_H
#define PDA_PDA_H

#include ...


class PDA {
    State* start_state;
    std::vector<std::string> alphabet;
    std::vector<State*> possible_states;
    std::vector<std::string> stack;

    std::vector<std::string> stackAlphabet;
    std::string stackInit;
public:
    PDA();

    PDA(CFG* cfg);

    PDA(std::vector<std::string> alph, State* start);

    bool delta(State* q,std::string w);

    bool delta_start(std::string w);

    State *getStartState() const;
};


#endif //PDA_PDA_H
```

```
PDA::PDA(CFG *cfg) {
    // a PDA made from a CFG consists of only 1 state, which is non-final
    this->start_state = new State( state_name: "q", finite: false);

    // Every terminal symbol is part of the PDA alphabet
    for(auto symbol : cfg->getTerminals()){
        this->alphabet.push_back(symbol);
    }

    // Every terminal AND non-terminal symbol is part of the PDA stack alphabet
    this->stackAlphabet = this->alphabet;
    for(auto symbol : cfg->getNonTerminals()){
        stackAlphabet.push_back(symbol);
    }

    // the starting stack symbol is the same as the starting symbol of the CFG
    this->stackInit = cfg->getStartSymbol();
    this->stack = {cfg->getStartSymbol()};

    // TODO: convert production rules to transitions
    // for each nonterminal A (with production rule A --> a)
    //                      add the rule δ(q,ε,A) --> (q,a)
    std::string epsilon = "e";
    for(auto nonTerminal : cfg->getNonTerminals()){
        // Loop over each production rule for current non-terminal
        for(auto prod : cfg->getProductionRulesForNonTerminal(nonTerminal)){
            // Add transition from only state to itself, with
            // nonTerminal as input, epsilon as stack_top, the result of the prod. rule as stack_operation
            this->start_state->setInputToState( & epsilon, & nonTerminal, & std::get<1>( & prod),this->start_state);
        }
    }

    // for each terminal a, add following rule: δ(q,a,a) = (q,ε)
    for(auto terminal : cfg->getTerminals()){
        this->start_state->setInputToState( & terminal, & terminal, & epsilon,this->start_state);
    }
}
```

Om de regel te verwerken die in de cellulaire automaat de verspreiding van het virus tussen cellen bepaalt, maken we gebruiken van een Push-Down automaat. Deze PDA wordt dan opgebouwd aan de hand van een context-free-grammar, waarvan de symbolen en production rules bepaald worden door de regel die we instellen voor onze verspreiding. In de simulatie gebruikten we steeds de regel die zegt dat, om in een cel de infectie te laten starten, er minstens 3 buurcellen moeten bestaan die al geïnfecteerd zijn.

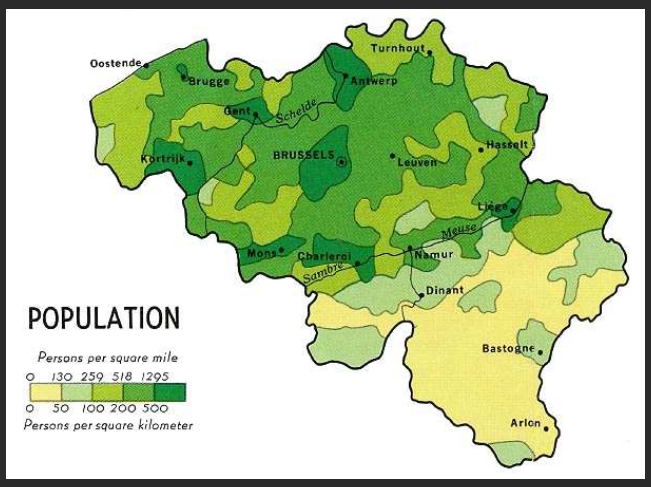
```cpp
int main(){
    std::vector<std::tuple<std::string,std::string>> prodRules;
    prodRules.emplace_back("S","HIIIO");
    prodRules.emplace_back("H","HI");
    prodRules.emplace_back("H","S");
    CFG* test = new CFG( nonTerminals: {"S","H"}, terminals: {"I","O"},prodRules, startSymbol: "S");
    PDA* pdaFromCfg = new PDA(test);
```

PDA.h | PDA.cpp | CFG.h | CFG.cpp | main.cpp | CellularAutomata.cpp

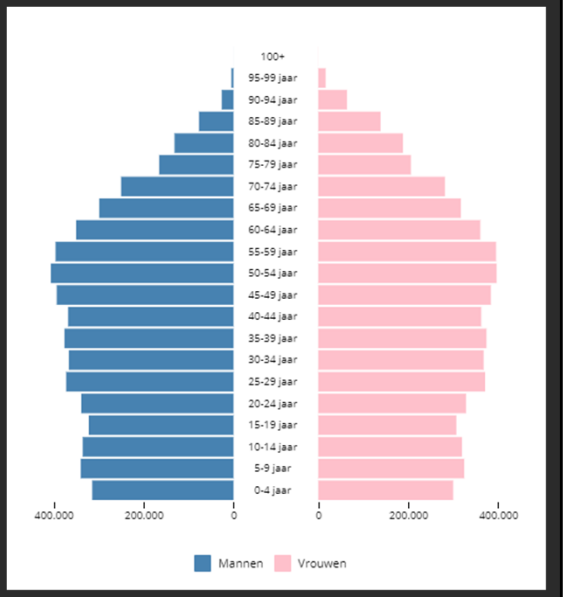
```cpp
            // Update cell if it's already infected
            if(state == 1){
                newCA->begin()[i][j].cellIteration(&Hospitals);
                continue;
            }
            // If not, count the infected neighbours
            int neighbour_dead = neighbourCount(j, i);
            // Make a string of I's for the PDA, each I representing 1 infected neighbour
            std::string PDAInput = "O";
            for(int k = 0; k < neighbour_dead; k++){
                PDAInput += "I";
            }
            bool infected = this->cellPDA->delta_start(PDAInput);
            if(infected){
                // Start each newly infected cell with 1 infectious person
                newCA->begin()[i][j].DeadOrAlive = 1;
                newCA->begin()[i][j].infectionCount = 1;
                newCA->begin()[i][j].population[0]->infected = true;
            }
        }
    }
```

Deze regel vertaalt zich dan naar de context-free grammar, door een production rule die zegt dat S zich afbeeldt op HIIIO, waarbij het aantal I'tjes bepaald hoeveel buren er geïnfecteerd moeten zijn.
Om ook cellen met meer dan 3 geïnfecteerde buren geïnfecteerd te laten worden, is er nog een tweede regel voor onze CFG: H beeldt zich af op HI. Om ook een string te accepteren die een H laat staan op de stack, is er voor H ook de regel H → S. We hebben vervolgens een algoritme geïmplementeerd om een PDA op te stellen vanuit een gegeven context-free-grammar. Deze PDA wordt één keer gebouwd voor de hele simulatie, maar er zou ook perfect een unieke CFG en PDA kunnen

bestaan voor elke cel in de cellulaire automaat, met telkens een andere verspreidingsregel.
Bij het uitvoeren van de simulatie wordt er dan voor elke niet-reeds geïnfecteerde cel het aantal geïnfecteerde buurcellen berekend. Dan wordt er voor elke getelde buur een “I” toegevoegd aan de string, welke begint met een “O” om 0 getelde buren niet geaccepteerd te laten worden.Deze string wordt door de PDA ingelezen, welke dan teruggeeft of de huidige cel nu ook geïnfecteerd is of niet.

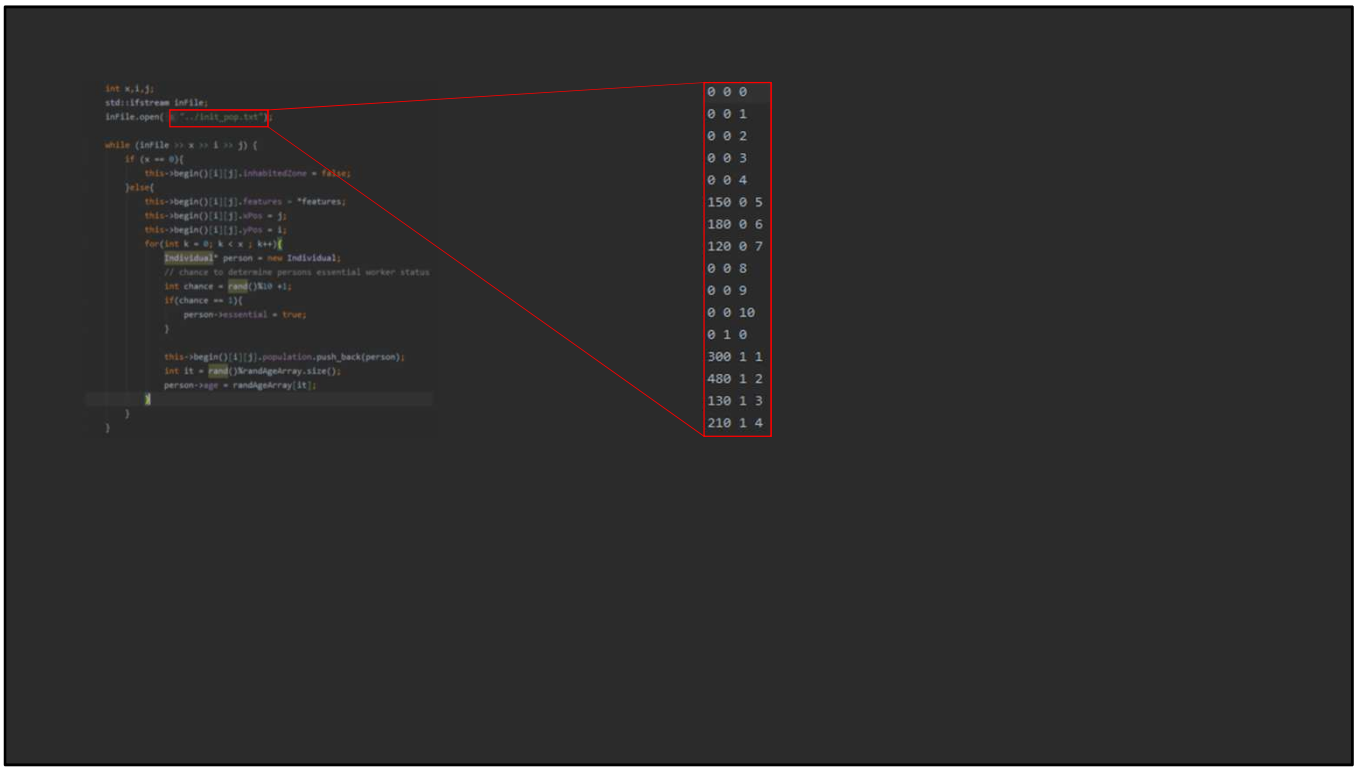


https://belgiie.weebly.com/demografie.html



https://tinyurl.com/yagljg2n

De initialisatie van ons systeem gebeurt aan de hand van een matrix die in ons geval België voorstelt. Deze matrix is gebaseerd op werkelijke cijfers van de populatie van België. Ook de leeftijdsverschillen in ons land hebben we opgezocht voor om deze zo accuraat mogelijk te kunnen beschrijven. Voor dit project hebben we ons gericht op alleen België, maar eender welke matrix die je invoert werkt voor deze simulator.

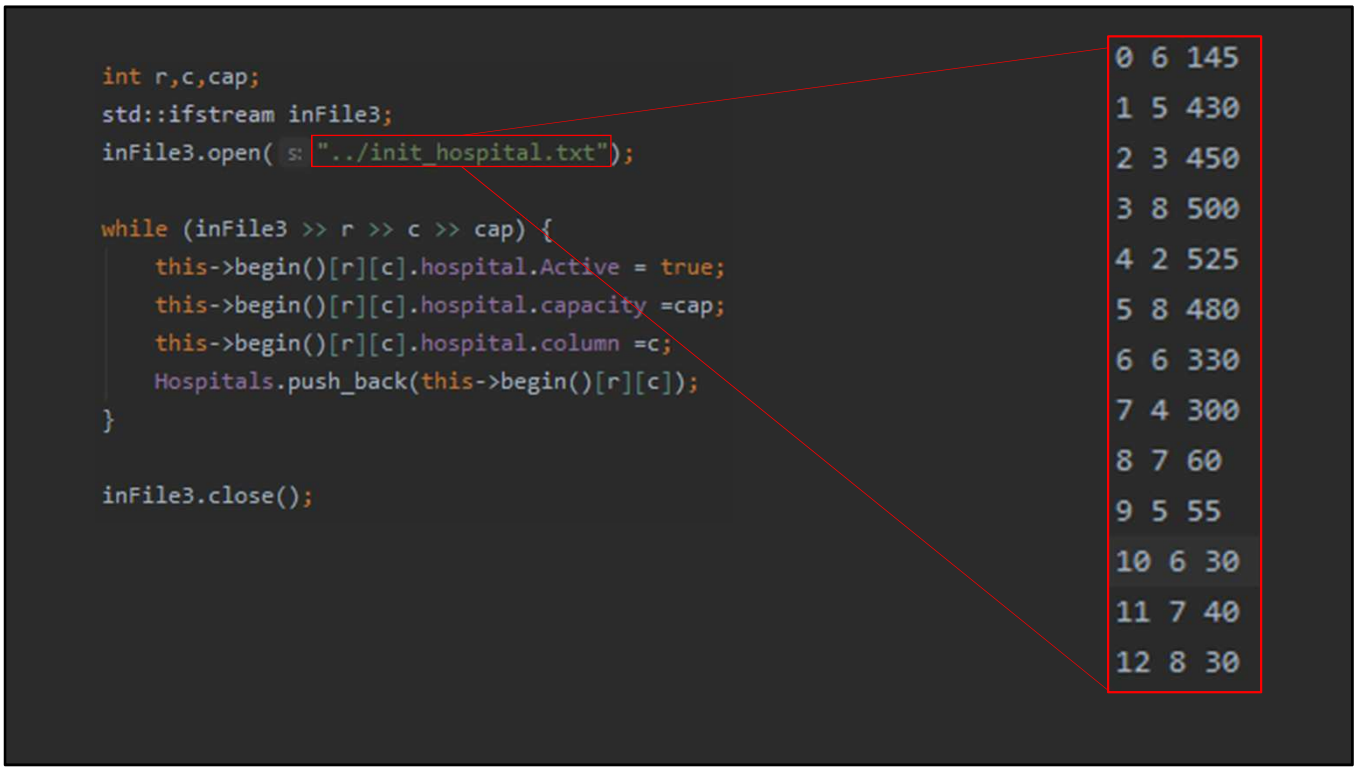


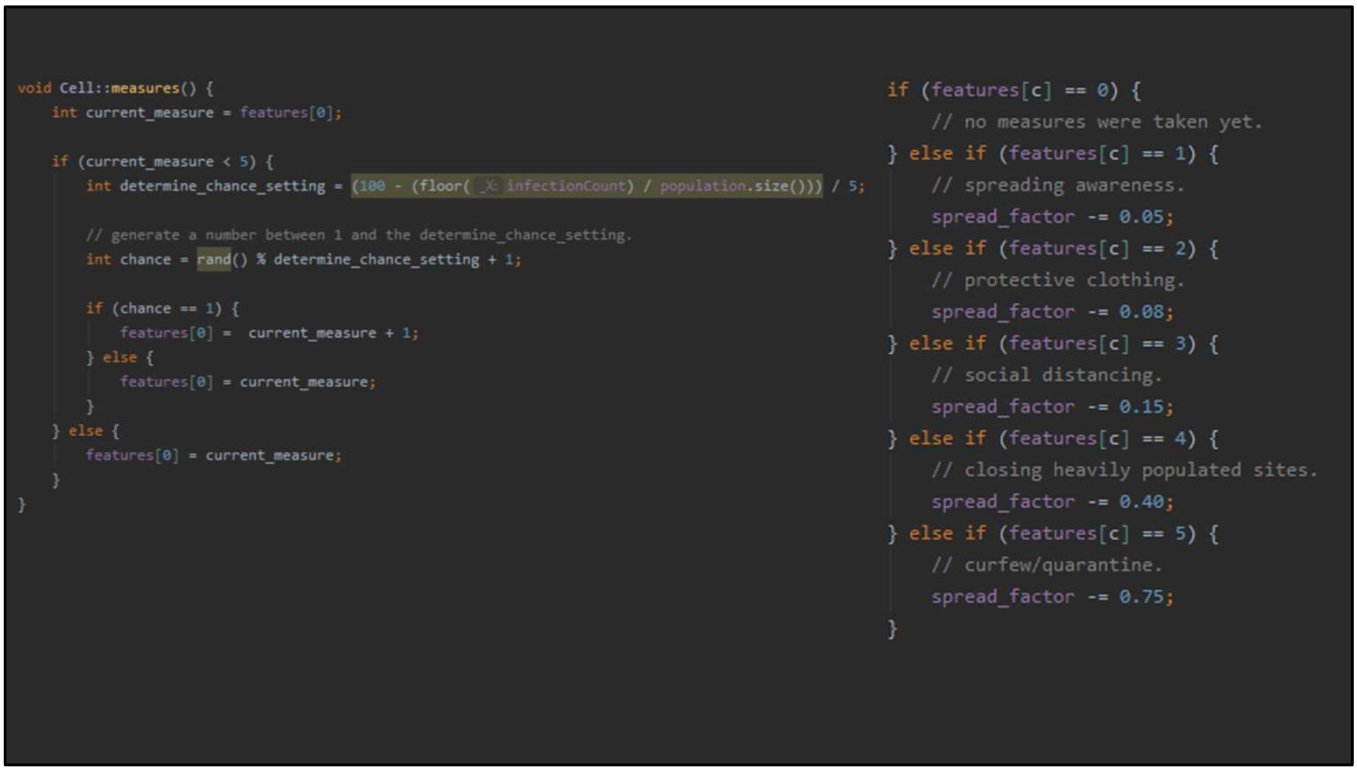
Bij het aanmaken van de cellulaire automaat wordt deze matrix van België dus verwerkt als volgt. De init_pop.txt file wordt gelezen, deze file bevat op elke lijn respectievelijk de populatie van de cel, de y coördinaat van de cel en ten slotte de x coördinaat van de cel. Op deze manier worden de cellen opgebouwd tot de hele matrix van cellen compleet is. Bij de initialisatie van de cellulaire automaat worden ook de leeftijden en de hospitalisatie bepaald. De leeftijden worden via een init bestand verwerkt dat de percentages van de bevolking per 10 jaar bepaald. De hospitalisatie wordt ook via een init bestand geïnitialiseerd dit init bestand heeft per lijn

respectievelijk de y coördinaat van de cel, de x coördinaat van de cel en ten slotte de capaciteit van het hospitaal.

```
// ************************************
// Making the age-distribution vector
std::string inputFileAges = "../init_age.txt";
std::ifstream inFile2;
inFile2.open(inputFileAges);

// We start by reading the given values, which are supposed to be percentages (ex: 0.010)  starting with the
// percentage for group 0-10, then 10-20, ...
double a;
std::vector<double> agePercentages;
while (inFile2 >> a) {
    agePercentages.push_back(a);
}
inFile2.close();
// For these percentages, we want to store the int value * 10 (so 0.010 --> 10)
for (int k = 0; k < agePercentages.size(); ++k) {
    agePercentages[k] = int(agePercentages[k] * 1000);
}
// Now, for each agePercentage we fill in the randAgeArray with the amount of ages given by that bracket,
// calculating the age value by (pseudo-) randomly selecting a value [0-9] to add to the current decimal value
// (starting with 0*10, then 1*10, ...
std::vector<int> randAgeArray;
for (int i = 0; i < agePercentages.size(); ++i) {
    for (int v = 0; v < agePercentages[i]; ++v) {
        int val = i * 10 + rand()%10;
        randAgeArray.push_back(val);
    }
}
//*****************************
```

```
0.112
0.112
0.123
0.130
0.132
0.139
0.116
0.079
0.047
0.010
```

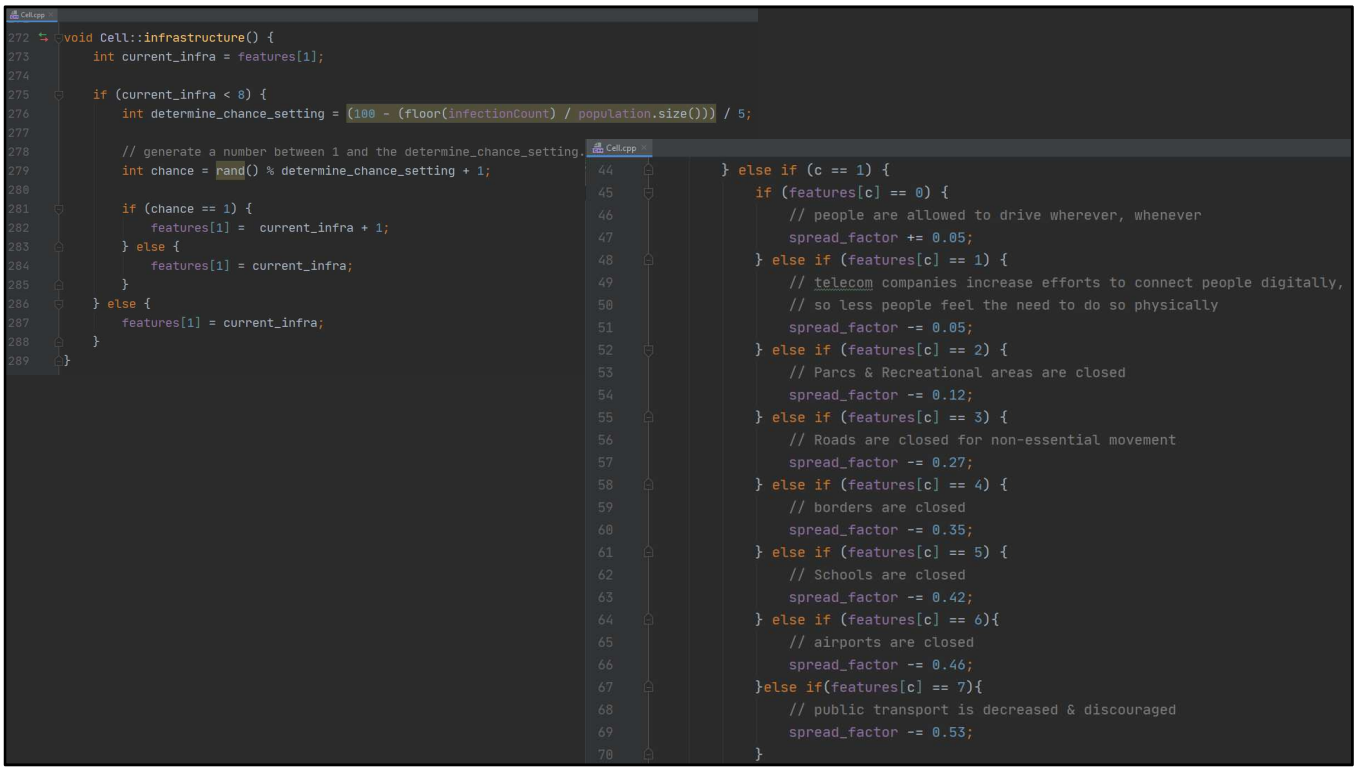

int r,c,cap;
std::ifstream inFile3;
inFile3.open(s: "../init_hospital.txt");
while (inFile3 >> r >> c >> cap) {
this->begin()[r][c].hospital.Active = true;
this->begin()[r][c].hospital.capacity =cap;
this->begin()[r][c].hospital.column =c;
Hospitals.push_back(this->begin()[r][c]);
}
inFile3.close();
0 6 145
1 5 430
2 3 450
3 8 500
4 2 525
5 8 480
6 6 330
7 4 300
8 7 60
9 5 55
10 6 30
11 7 40
12 8 30



```
void Cell::measures() {
    int current_measure = features[0];

    if (current_measure < 5) {
        int determine_chance_setting = (100 - (floor( X infectionCount) / population.size())) / 5;

        // generate a number between 1 and the determine_chance_setting.
        int chance = rand() % determine_chance_setting + 1;

        if (chance == 1) {
            features[0] =  current_measure + 1;
        } else {
            features[0] = current_measure;
        }
    } else {
        features[0] = current_measure;
    }
}
```

```
if (features[c] == 0) {
    // no measures were taken yet.
} else if (features[c] == 1) {
    // spreading awareness.
    spread_factor -= 0.05;
} else if (features[c] == 2) {
    // protective clothing.
    spread_factor -= 0.08;
} else if (features[c] == 3) {
    // social distancing.
    spread_factor -= 0.15;
} else if (features[c] == 4) {
    // closing heavily populated sites.
    spread_factor -= 0.40;
} else if (features[c] == 5) {
    // curfew/quarantine.
    spread_factor -= 0.75;
}
```

In onze simulator hebben we er ook voor gezorgd dat elke cel zijn eigen maatregelen zal nemen. De ernst en impact van deze maatregelen bouwen zich op tot een maximum. Wat dus ook wil zeggen dat voor je een quarantaine opstelt, dat eerst alle winkelcentra gesloten moeten worden en daarvoor nog social distancing moet zijn ingevoerd. De verschillende maatregelen die we een regio doen nemen is afhankelijk van hoe we een simulatie initialiseren en/of hoe de situatie in een regio verslechterd. Dat wil zeggen dat een cel die in een slechte situatie zit, makelijker maatregelen zal nemen om het virus te bestrijden, dan een cel die slechts enkele besmettingen heeft.



```cpp
void Cell::infrastructure() {
    int current_infra = features[1];

    if (current_infra < 8) {
        int determine_chance_setting = (100 - (floor(infectionCount) / population.size())) / 5;

        // generate a number between 1 and the determine_chance_setting.
        int chance = rand() % determine_chance_setting + 1;

        if (chance == 1) {
            features[1] =  current_infra + 1;
        } else {
            features[1] = current_infra;
        }
    } else {
        features[1] = current_infra;
    }
}
```

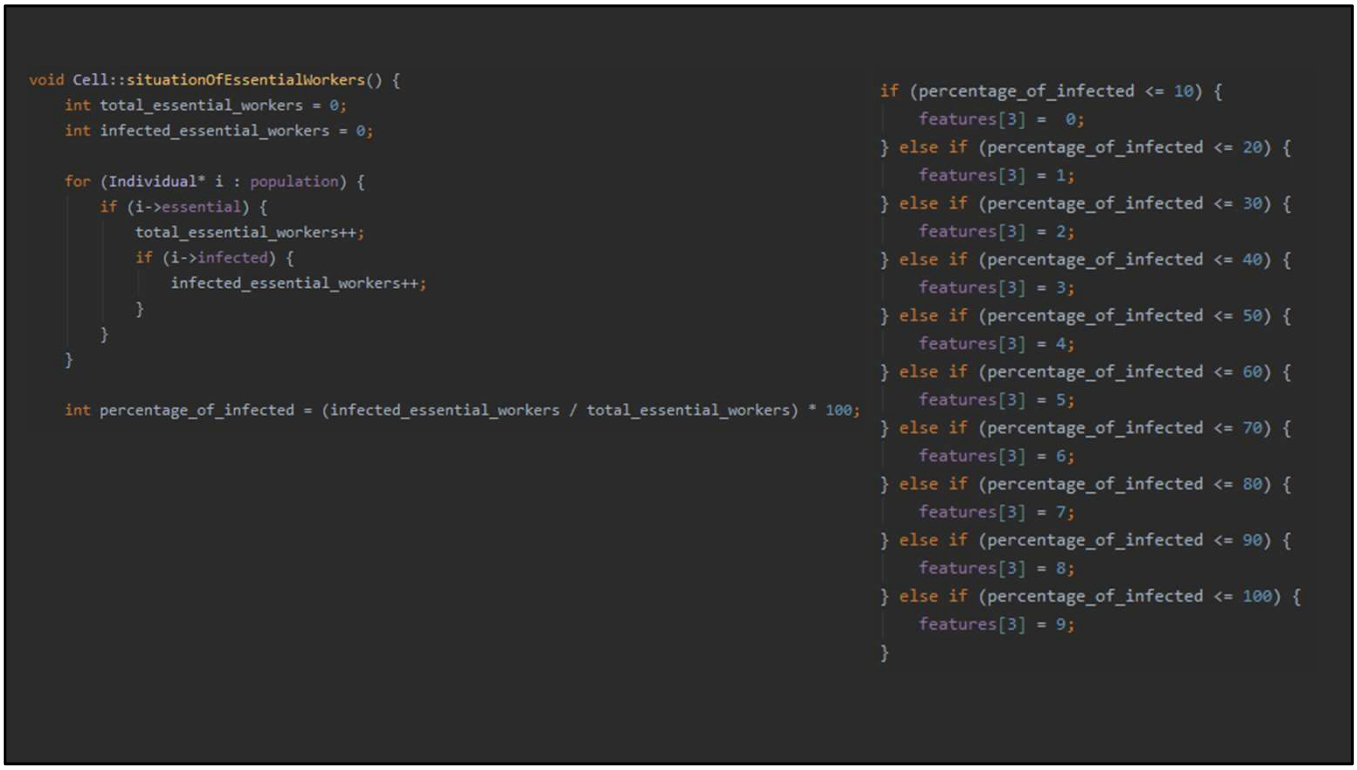
```cpp
    } else if (c == 1) {
        if (features[c] == 0) {
            // people are allowed to drive wherever, whenever
            spread_factor += 0.05;
        } else if (features[c] == 1) {
            // telecom companies increase efforts to connect people digitally,
            // so less people feel the need to do so physically
            spread_factor -= 0.05;
        } else if (features[c] == 2) {
            // Parcs & Recreational areas are closed
            spread_factor -= 0.12;
        } else if (features[c] == 3) {
            // Roads are closed for non-essential movement
            spread_factor -= 0.27;
        } else if (features[c] == 4) {
            // borders are closed
            spread_factor -= 0.35;
        } else if (features[c] == 5) {
            // Schools are closed
            spread_factor -= 0.42;
        } else if (features[c] == 6){
            // airports are closed
            spread_factor -= 0.46;
        }else if(features[c] == 7){
            // public transport is decreased & discouraged
            spread_factor -= 0.53;
        }
```

De feature infrastructuur werkt op een gelijkaardige manier als de measures, waarbij ook deze doorheen de simulatie een kans krijgt om strenger aangepakt te worden wanneer er meer geïnfecteerden in een cel zijn. Deze feature kan ook een startwaarde meekrijgen, waardoor de verspreiding van het virus binnen een cel snel geremd wordt. Hiermee simuleren we het sluiten van de openbare plaatsen, scholen, het aflassen van vluchten etc..

```
void Cell::HospitalCapabilities(std::vector<Cell>* hospitals, int r, int c) {
    int distance =0;

    if(this->DeadOrAlive){
        distance = abs( X (*hospitals)[r].hospital.column - c);
        if(this->population.size() > (*hospitals)[r].hospital.capacity){
            distance +=3;
        }
    }
    this->features[2] = distance;
}
```

```
} else if (c == 2) {
    if (features[c] <= 1) {
        // no patients with COVID-19 : sufficient capacity of staff and beds
    } else if (features[c] <= 3) {
        // first patients with COVID-19
        spread_factor += 0.15;
    } else if (features[c] <= 5) {
        // increase in COVID-19 cases : decreasing hospital staff and beds
        spread_factor += 0.25;
    } else if (features[c] <= 7) {
        // facilities at 50% of their capacity
        spread_factor += 0.35;
    } else if (features[c] == 8) {
        // facilities at = 70% of their capacity : limited beds left
        spread_factor += 0.43;
    }else if (features[c] == 9) {
        // facilities at >= 100% of their capacity : total chaos
        spread_factor += 0.49;
    }
```

De hospitalisatie van een cel wordt bepaald door een tweede init bestand. In dit bestand worden respectievelijk de y coördinaat, x coördinaat en ten slotte de capaciteit van het hospitaal meegegeven. Hierdoor kun je zelf aanduiden waar er zich een hospitaal bevindt en hoeveel mensen dit hospitaal kan behandelen. Dit is belangrijk omdat een ziekenhuis nog steeds een dichtbevolkte ruimte blijft, en als ernstig zieken niet meer opgevangen kunnen worden door een tekort aan bedden en zorgverleners, is de kans dat deze het virus verspreiden des te groter.



```
void Cell::situationOfEssentialWorkers() {
    int total_essential_workers = 0;
    int infected_essential_workers = 0;

    for (Individual* i : population) {
        if (i->essential) {
            total_essential_workers++;
            if (i->infected) {
                infected_essential_workers++;
            }
        }
    }

    int percentage_of_infected = (infected_essential_workers / total_essential_workers) * 100;
```

```
if (percentage_of_infected <= 10) {
    features[3] =  0;
} else if (percentage_of_infected <= 20) {
    features[3] = 1;
} else if (percentage_of_infected <= 30) {
    features[3] = 2;
} else if (percentage_of_infected <= 40) {
    features[3] = 3;
} else if (percentage_of_infected <= 50) {
    features[3] = 4;
} else if (percentage_of_infected <= 60) {
    features[3] = 5;
} else if (percentage_of_infected <= 70) {
    features[3] = 6;
} else if (percentage_of_infected <= 80) {
    features[3] = 7;
} else if (percentage_of_infected <= 90) {
    features[3] = 8;
} else if (percentage_of_infected <= 100) {
    features[3] = 9;
}
```

De situatie van de essentiële werkers zoals bij voorbeeld de doctoren en politici is belangrijk voor de verspreiding van het virus tegen te houden. De ernst van deze feature wordt bepaald door het percentage van de besmette essential workers binnen in een cel. Hoe ernstiger deze feature hoe groter de positieve inpact op de spread_factor dus. De individuen die een essential worker zijn worden bepaald per cel. Elke cel wordt opgevuld met individuen zoals eerder gezien in een afgelopen dia. Per cel wordt er een percent kans gegenereerd binnen bepaalde marges, deze kans is de kans dat een individu die in een cel zijn populatie geplaatst wordt een essential worker is of niet.

De eigenschappen van het virus zijn in onze simulatie beperkt, aangezien we enkel de verspreiding simuleren, en dus geen rekening houden met zaken zoals sterftecijfer of incubatieperiode. Toch is er een belangrijk cijfer wat we meegeven aan de simulatie, namelijk het ‘reproductiegetal’. Dit geeft een basiswaarde voor het aantal nieuwe geïnfecteerden die er zullen zijn voor elk reeds geïnfecteerd individu. Zo zal een drager van een virus met reproductiegetal 2 gemiddeld gezien 2 anderen besmetten. Op deze manier kunnen verschillende virussen met verschillende besmettingsgetallen gesimuleerd worden.

```
void Cell::AgeDifference() {

    int difference=0;
    double kid=0,teen=0,adult=0,elderly=0;

    for(auto &person: this->population){
        if(person->age <= 11)
            kid+=1;
        else if(person->age <= 25)
            teen +=1;
        else if(person->age <= 65)
            adult +=1;
        else if(person->age <=100)
            elderly +=1;
    }


    if((kid/this->population.size())*100 < 11)
        difference+=2;
    if((teen/ this->population.size())*100 <15)
        difference+=1;
    if((adult/ this->population.size())*100 > 48)
        difference+=3;
    if((elderly/ this->population.size())*100 >15)
        difference+=3;

    this->features[5] = difference;
}
```

```
}else if(c == 5){
    if (features[c] <= 1) {
        // age difference in population is stable
    } else if (features[c] == 2) {
        // more teens in population than elderly
        spread_factor -= 0.15;
    } else if (features[c] <= 5) {
        // slightly more adults or elderly in population
        spread_factor += 0.18;
    } else if (features[c] <= 7) {
        // age differences in population increases
        spread_factor += 0.32;
    } else if (features[c] <= 9) {
        // age difference in population is unstable
        spread_factor += 0.40;
    }
}
```

Afhankelijk van het leeftijdsverschil binnen een cel kan de spread-factor negatief of positief beïnvloed worden. Stel dat er meer bejaarden zijn dan gemiddeld dan is de kans dat het virus zich meer verspreid groter, mits bejaarden vaak kwetsbaarder zijn dan de jongeren.

Onze simulatie geeft dus een semi-realistisch beeld weer van de verspreiding van een virus. Wanneer we onze uitvoer bekijken, toont deze voor elke ‘populated’ cel het percentage van geïnfecteerden. Omdat het vrij lastig is om puur aan de hand van deze cijfers een vergelijking tussen verschillende startsituaties te maken, hebben we deze data steeds geëxporteerd naar Excel.
Dit geeft ons namelijk een zeer overzichtelijk beeld van de invloed van verschillende startscenario’s op onze simulatie. Zo zien we in de eerste kolom een simulatie waarin er geen enkele maatregel genomen werd vanaf het begin, en de features zoals measures en

infrastructure dus beide aan kans onderhevig zijn. Wanneer we dan naar het extreme geval kijken waarin België bij aanvang reeds aan zijn measures tegen het virus heeft gedacht, en bijvoorbeeld een verplichte quarantaïne opstelt, zien we dat de percentages binnen de cellen minder hoge waardes bereiken.
Ook kunnen we een verschil zien in effect tussen weinig beginmaatregelen qua infrastructuur of measures, en ook tussen een combinatie van de twee. Er is dus een resem aan mogelijkheden qua startsituaties, alsook de waardes die voor elke feature de simulatie beïnvloeden. We hopen op deze manier toch enigszins de boodschap over te kunnen brengen dat het belangrijk is dat iedereen zijn steentje bijdraagt en geduld heeft, want de opgelegde maatregelen hebben wel degelijk een effect.