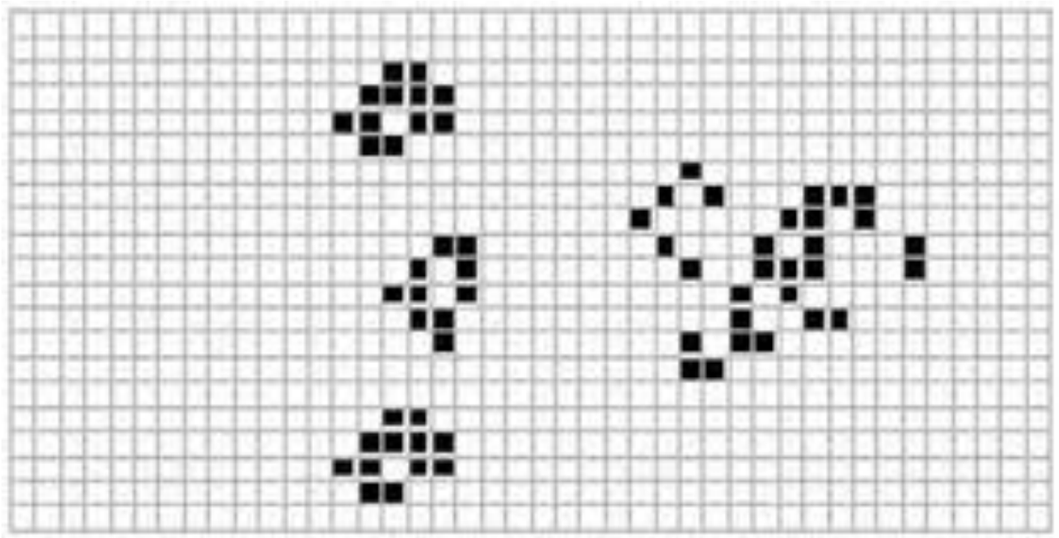
Simulating on an (ir)regular grid

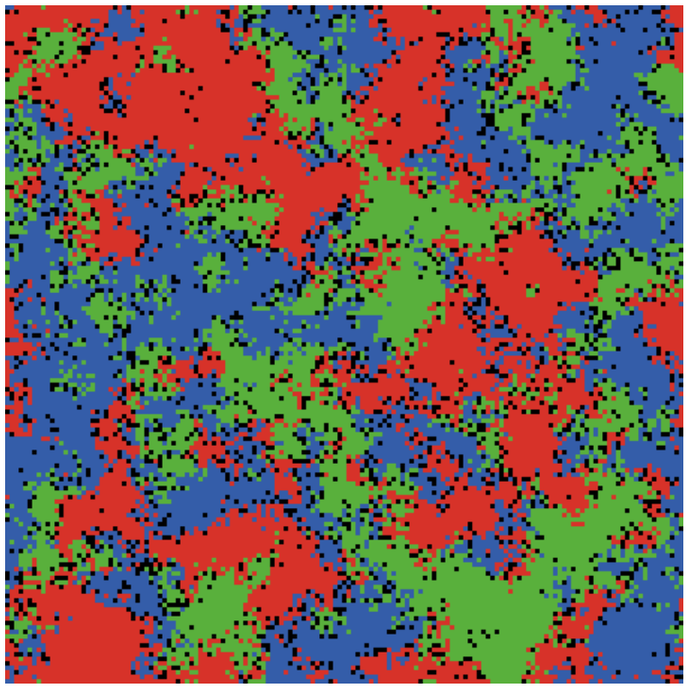
Thijs Janzen

Simulating on a grid

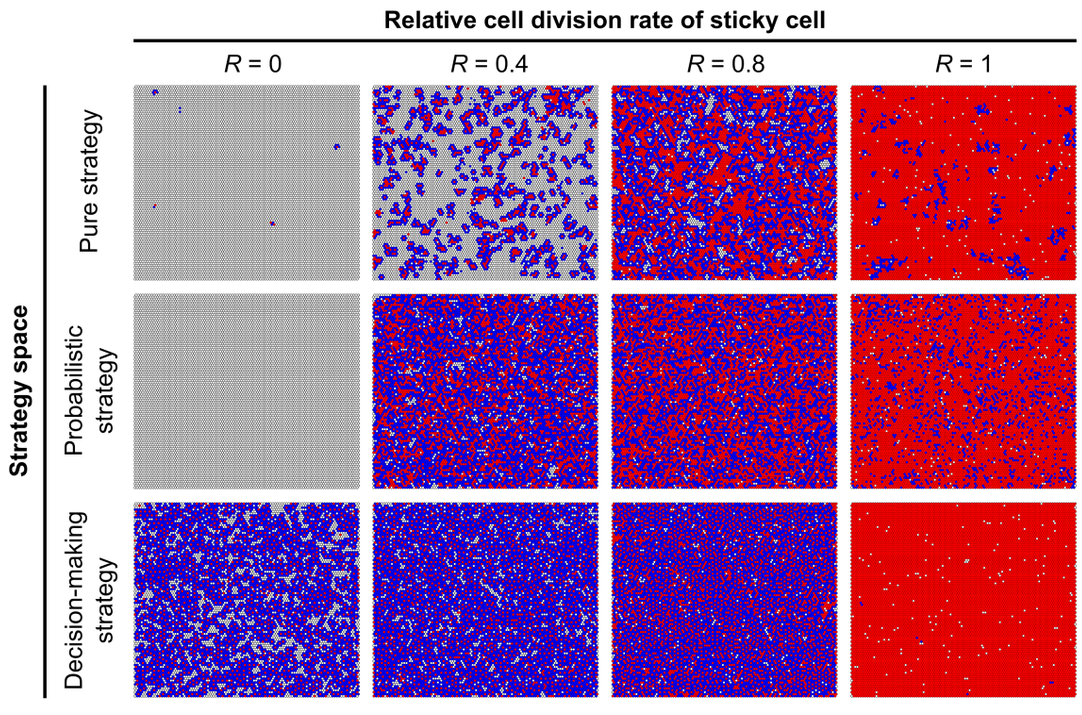
Conway Game of Life



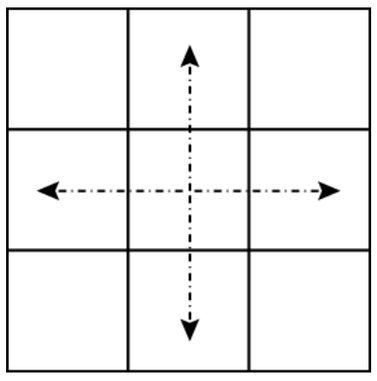
Spatial Rock-Paper-Scissors



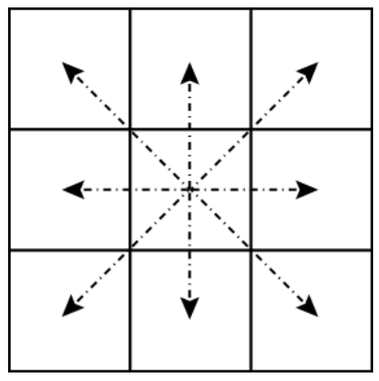
van Gestel & Nowak 2016



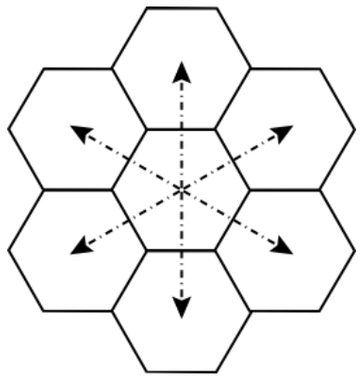
Different types of grids



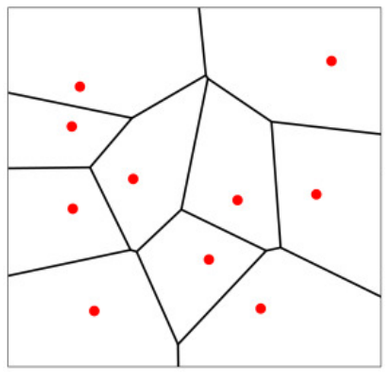
4 neighbours



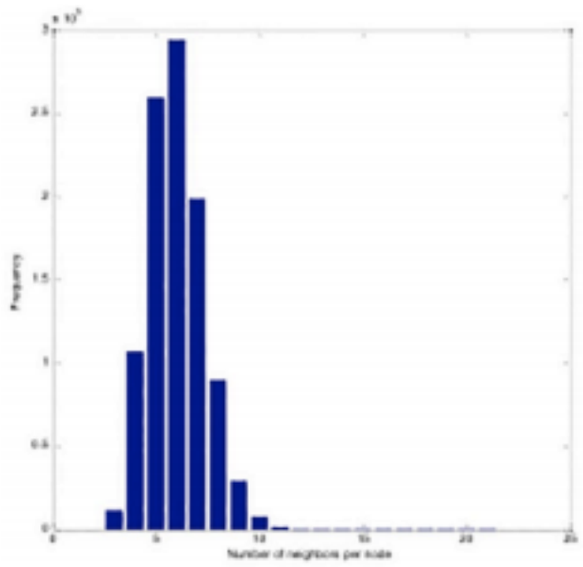
8 neighbours



6 neighbours



1-10 neighbours



Regular grid figures from Tang et al. 2020

Cancer model example

- Modeled after Berg et al. 2019
- Spread of tumour cells, which are treated with an oncolytic virus
- In collaboration with Darshak Bhatt

PLOS COMPUTATIONAL BIOLOGY

RESEARCH ARTICLE

In vitro and in *silico* multidimensional modeling of oncolytic tumor virotherapy dynamics

David R. Berg[1], Chetan P. Offord[2†], Iris Kemler[2], Matthew K. Ennis[2], Lawrence Chang[2,3], George Paulik[4], Zeljko Bajzer[5], Claudia Neuhauser[6]*, David Dingli[2]*

1 Department of Information Technology, Mayo Clinic, Rochester, Minnesota, **2** Molecular Medicine, Mayo Clinic, Rochester, Minnesota, **3** Boston Children's Hospital and Boston Medical Center, Boston, Massachusetts, **4** International Business Machines, Rochester, Minnesota, **5** Department of Biochemistry and Molecular Biology, Mayo Clinic, Rochester, Minnesota, **6** Department of Mathematics, University of Houston, Houston, Texas

† Deceased.
* cmneuhaus@central.uh.edu (CN); dingli.david@mayo.edu (DD)

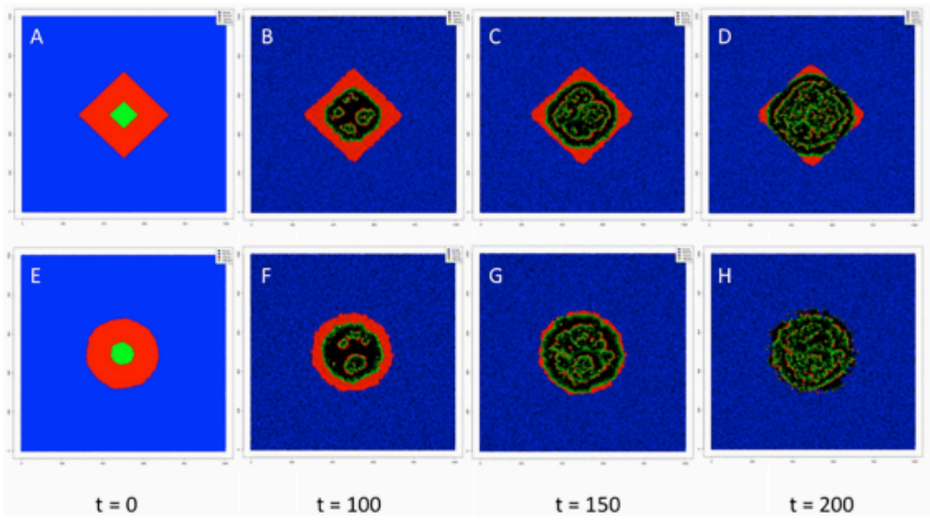


Fig 7. Two dimensional in silico simulations. Snapshots of the spread of an oncolytic virus in a 2D environment with a regular lattice architecture (A-D) having $n = 4$ nearest neighbors and with a Voronoi lattice (E–H) with $n = 6$ nearest neighbors. In both scenarios, there is one major focus of infection as in the equivalent in vitro scenario. Blue represents normal cells, red represents tumor cells, green represents infected tumor cells and black represents empty space.

https://doi.org/10.1371/journal.pcbi.1006773.g007

Oncolytic model

- 4 types of cells:
  - Empty
  - Normal / Healthy (blue)
  - Tumour (red)
  - Infected with Oncolytic virus (green)

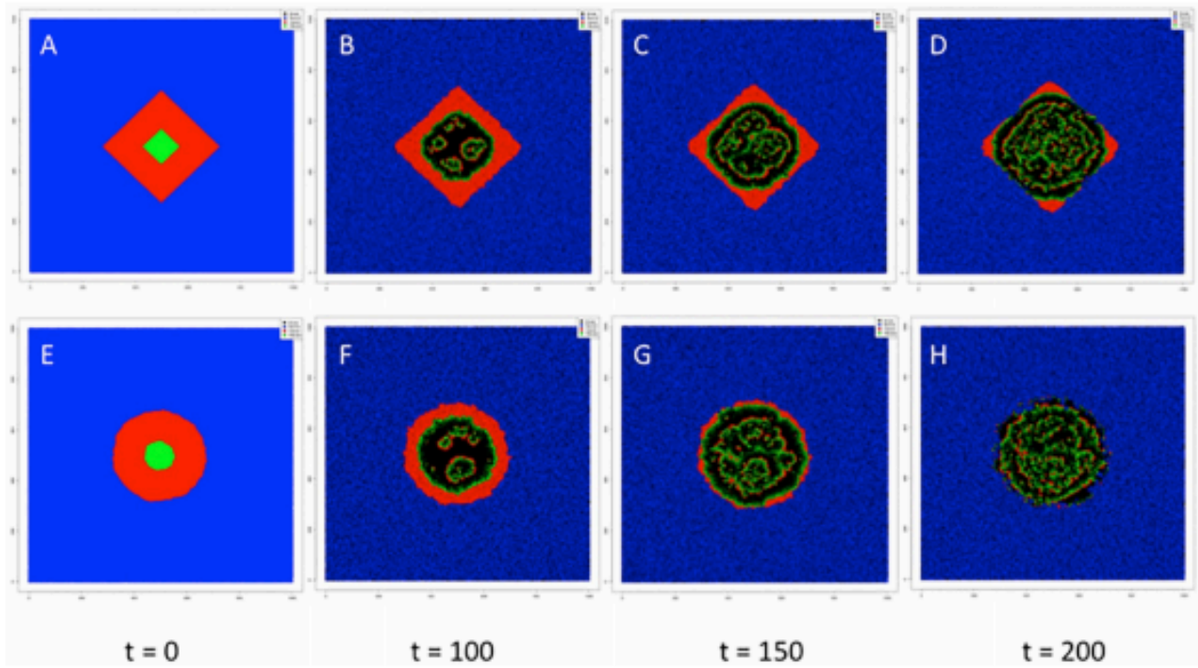


Fig 7. Two dimensional in silico simulations. Snapshots of the spread of an oncolytic virus in a 2D environment with a regular lattice architecture (A-D) having $n = 4$ nearest neighbors and with a Voronoi lattice (E–H) with $n = 6$ nearest neighbors. In both scenarios, there is one major focus of infection as in the equivalent in vitro scenario. Blue represents normal cells, red represents tumor cells, green represents infected tumor cells and black represents empty space.

https://doi.org/10.1371/journal.pcbi.1006773.g007

- Basic rules:
  - Normal cells only grow into empty cells
  - Tumour cells only grow into empty cells
  - Infected cells grow into Tumour cells (e.g. infect tumour cells)

Oncolytic Virus Simulator v0.5

Setup

Start

Stop

start setup: Full

grid type: regular

sqrt(#cells): 100

sqrt(#pixels): 1000

speed

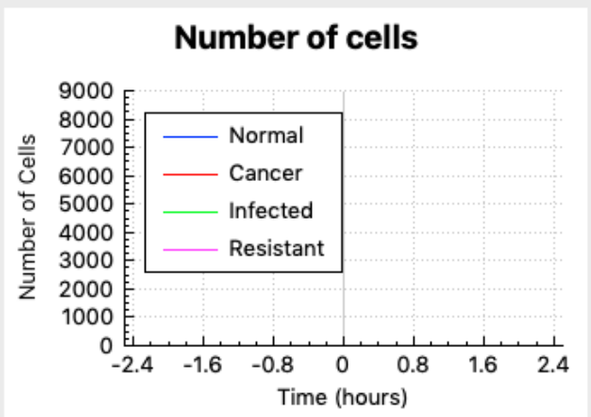
Maximum Time	10000
Cancer introductior	150
Virus introduction	300
initial normal cells	1000
initial cancer cells	400
birth_normal	0.50
death_normal	0.20
birth_cancer	1.00
death_cancer	0.10
birth_infected	1.20
death_infected	0.10

Infection_normal	0.01000
Frequency resistant cancer	0.00000
Distance infection upon death	1
Probability infection upon death	0.10
birth_cancer_resistant	0.90
death_cancer_resistant	0.10
color scheme	Cell types
diffusion	0.10
evaporation	0.10
inflammation amount	0.00
t-cell death rate	10
density scaler	1.000

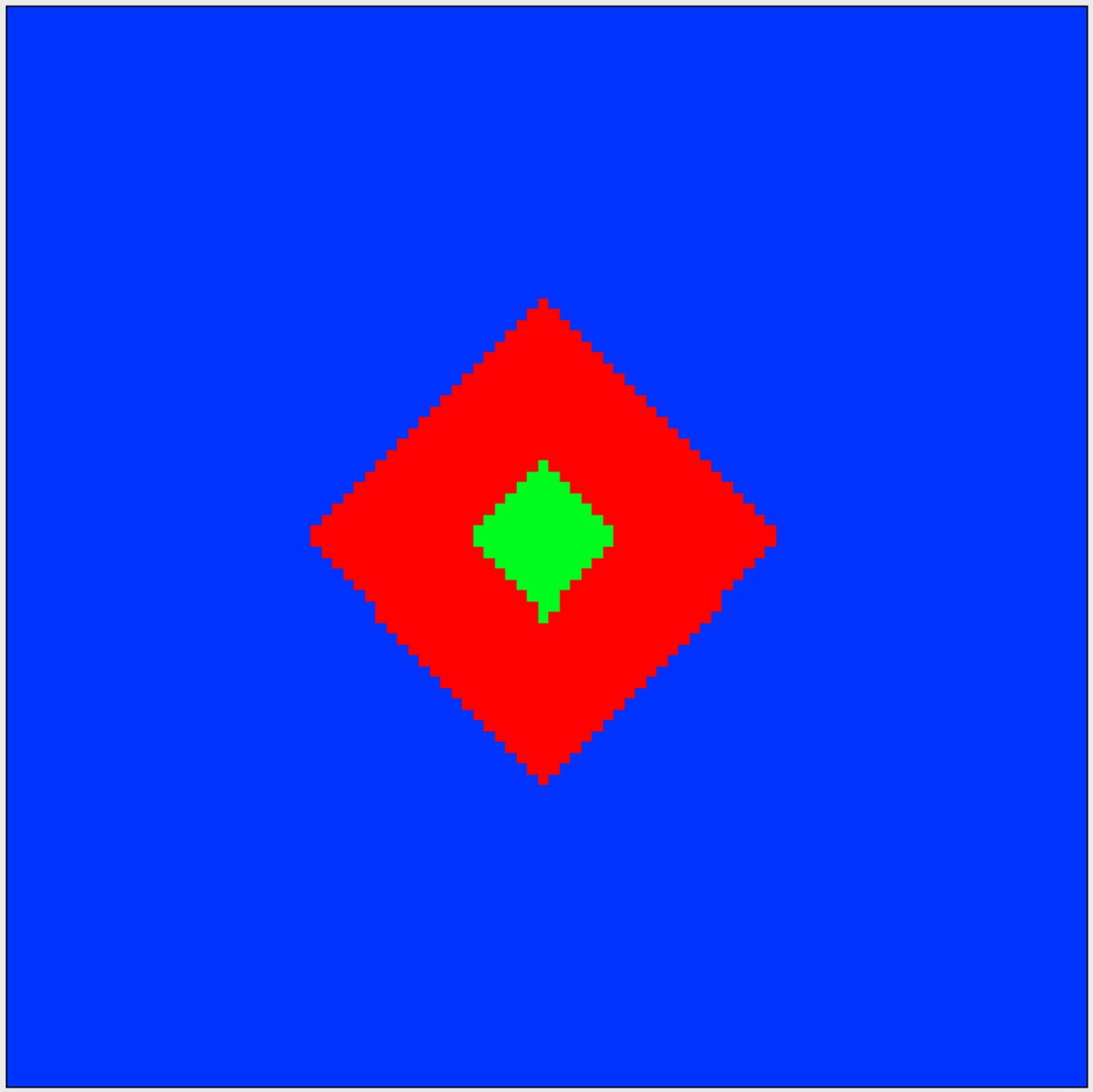
Add virus

Infection routine: Center

Fraction infected: 0.10



Prob normal infection 0.010000
Frequency resistant 0.000000
Distance infection upon death 1.000000
Probability infection upon death 0.100000

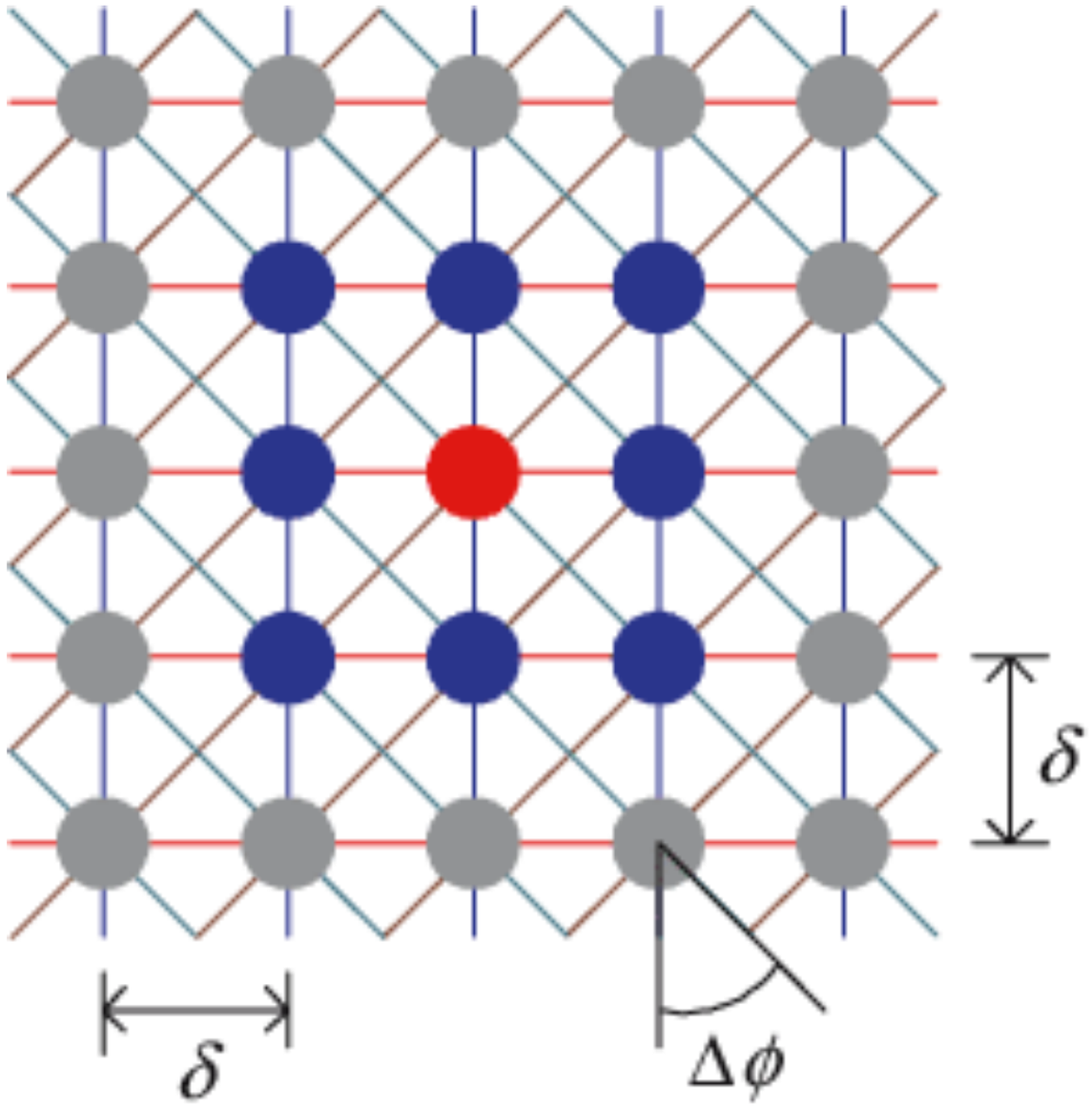


How to implement a grid

- Easiest way is to keep track of (relative) coordinates, and calculate who your neighbour is at runtime

	0,1	
-1,0	0,0	1,0
	0,-1	

Grid as a Graph



- Advantages:
  - only need to keep track of neighbours
  - irrespective of coordinates (!)

Code

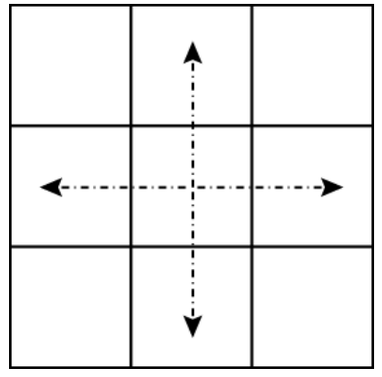
```
enum cell_type {normal, cancer, infected, empty, max_num};

struct node {

  size_t pos;
  float x_;
  float y_;
  cell_type node_type;

  std::vector< node* > neighbors;

  node(node&&) = delete;

  const node& operator=(node&&) = delete;
  node(const node&) = delete;
  const node& operator=(const node&) = delete;

  node();
};
```

Accessing neighbours

```
float node::freq_type_neighbors(const cell_type& ref_type) const {
  int count = 0;
  for(auto i : neighbors) {
    if(i->get_cell_type() == ref_type) count++;
  }
  return static_cast<float>(count * 1.0 / neighbors.size());
}
```

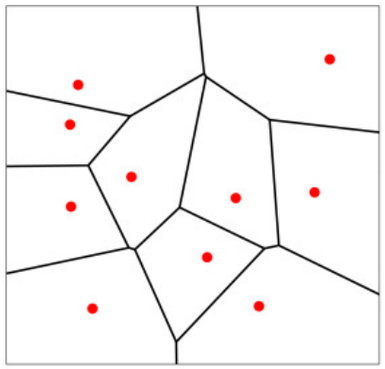
```
float node::freq_type_neighbors(const cell_type& ref_type) const {
  auto count =
       std::count_if(neighbors.begin(),
                     neighbors.end(),
                     [&ref_type](auto i) {
                         return ref_type == i->get_cell_type();
                     }
                    );

  return static_cast<float>(count * 1.0 / neighbors.size());
}
```

Adding nodes
regular grid with four neighbours



```
void node::update_neighbors(std::vector< node >& world,
                            size_t world_size) {

    static int relative_points[4][2] = { {-1,  0},
                                         { 1,  0},
                                         { 0,  1},
                                         { 0, -1} };

    for (int i = 0; i < 4; ++i) {
        int other_x = static_cast<int>(x_) + relative_points[i][0];
        int other_y = static_cast<int>(y_) + relative_points[i][1];
        if (other_x >= 0 &&
            other_y >= 0 &&
            other_x < static_cast<int>(world_size) &&
            other_y < static_cast<int>(world_size)) {

            int other_pos = other_x + other_y * static_cast<int>(world_size);

            node* neighbor = &world[static_cast<size_t>(other_pos)];
            neighbors.push_back(neighbor);
        }
    }
}
```

Voronoi case



- Used Voronoi code from: https://github.com/samkusin/gamelabs/tree/master/voron

- Draw *n* central points (for a 100x100 grid, *n* = 10,000), with x = U(0,1), y = U(0,1)

- Create Voronoi Tesselation based on points

- Now, go over each Voronoi Cell and:

    - Go over each edge and determine matching neighbour

Voronoi

- Draw *n* central points (for a 100x100 grid, *n* = 10,000), with x = U(0,1), y = U(0,1)

```
voronoi::Sites sites;
for(size_t i = 0; i < num_cells; ++i) {

    float x = rndgen.uniform() * sq_size;
    float y = rndgen.uniform() * sq_size;
    voronoi::Vertex temp_vertex(x, y);
    sites.push_back(temp_vertex);
}
```

- Create Voronoi Tesselation based on points

```
voronoi::Graph graph = voronoi::build(std::move(sites),
                                      sq_size, sq_size);
```

Voronoi

- Translate edges into neighbours

```
for (const auto& cell : graph.cells()) {
  for (const auto& edge : cell.halfEdges) {
   size_t left  = edge.left;
   size_t right = edge.right;
   world[left].add_neighbor(world, right);
   world[right].add_neighbor(world, left);
  }
}
```

Oncolytic Virus Simulator v0.5

Setup

Start

Stop

start setup: Full

grid type: voronoi

sqrt(#cells): 100

sqrt(#pixels): 1000

speed

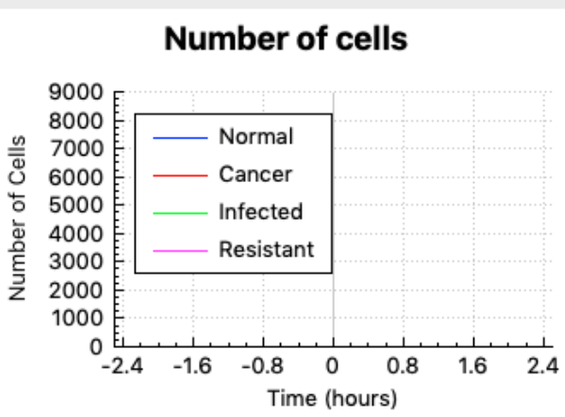
Maximum Time	10000
Cancer introductior	150
Virus introduction	300
initial normal cells	1000
initial cancer cells	400
birth_normal	0.50
death_normal	0.20
birth_cancer	1.00
death_cancer	0.10
birth_infected	1.20
death_infected	0.10

Infection_normal	0.01000
Frequency resistant cancer	0.00000
Distance infection upon death	1
Probability infection upon death	0.10
birth_cancer_resistant	0.90
death_cancer_resistant	0.10
color scheme	Cell types
diffusion	0.10
evaporation	0.10
inflammation amount	0.00
t-cell death rate	10
density scaler	1.000

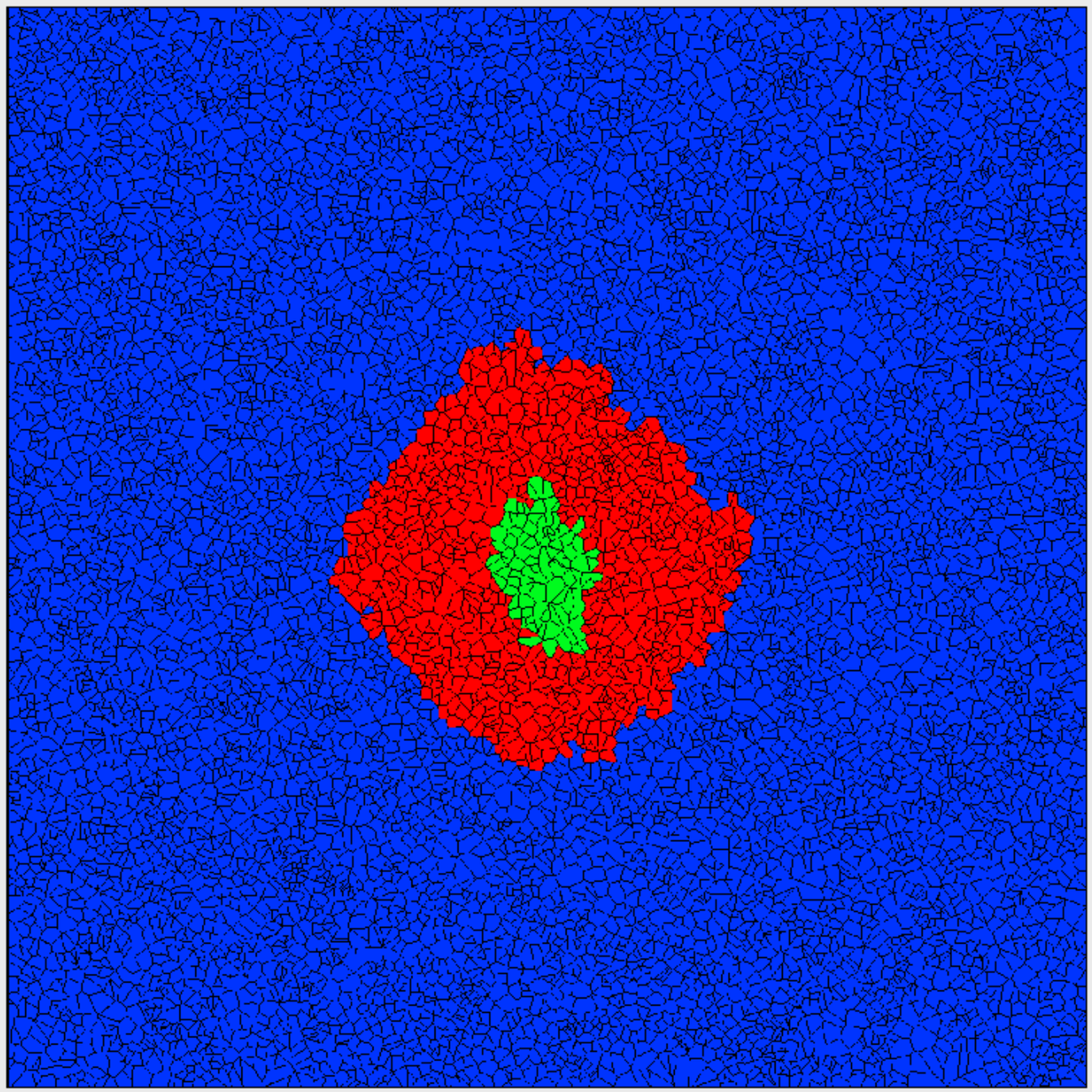
Add virus

Infection routine: Center

Fraction infected: 0.10

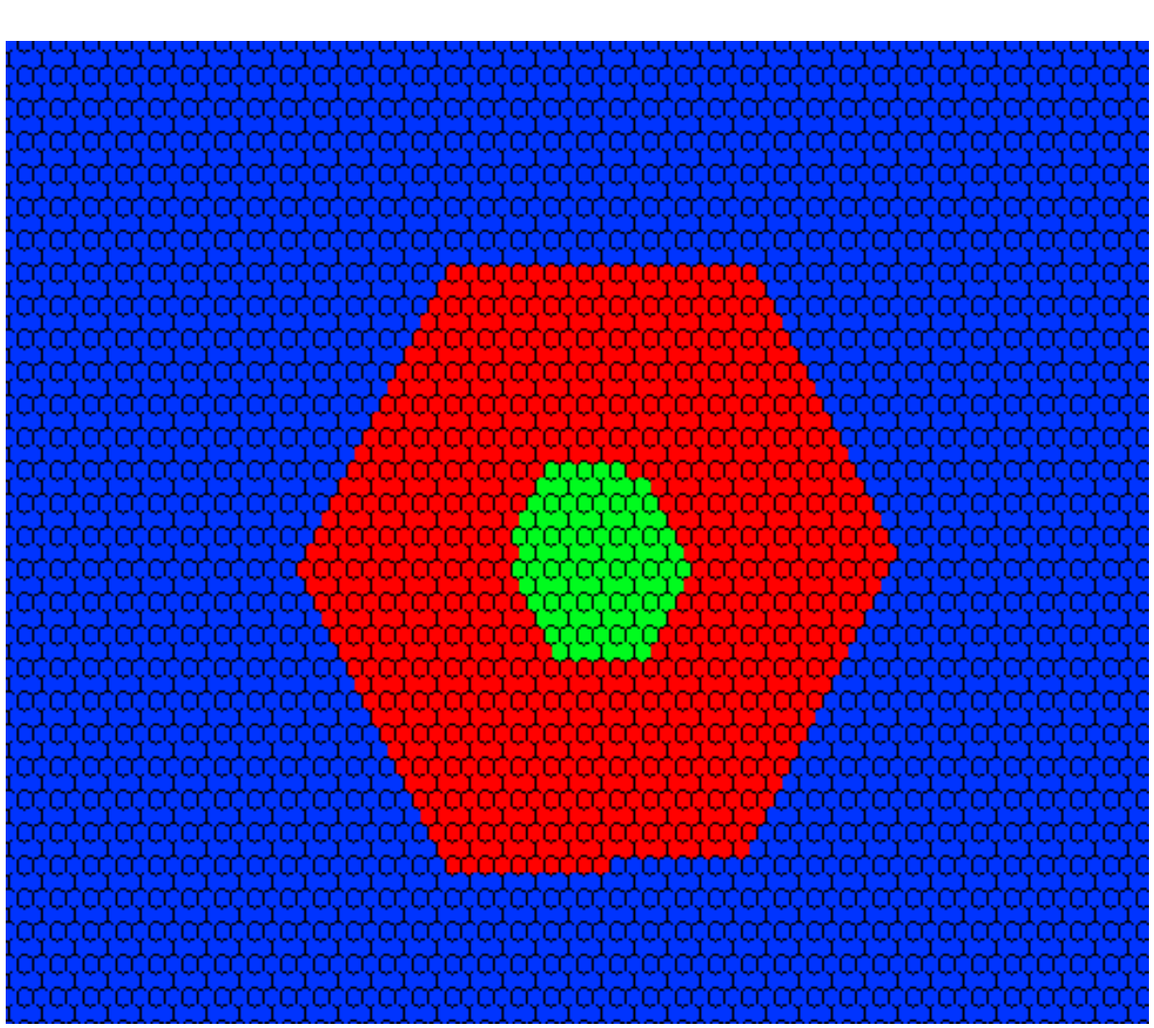


Prob normal infection 0.010000
Frequency resistant 0.000000
Distance infection upon death 1.000000
Probability infection upon death 0.100000



Hexagon via Voronoi

```
for(size_t i = 0; i < num_cells; ++i) {

  float x = i % sq_size;
  float y = i / sq_size;
  if ((i / sq_size) % 2 == 0)
     x += 0.5f;

  voronoi::Vertex temp_vertex(x, y);
  sites.push_back(temp_vertex);
}
```



Summary

- Represent the grid by a Graph
- Store neighbours as pointers

Thank you!