Emerging Collective behaviour in Protocell populations in Synthetic Biology Multi-Agent simulations

Project Report for AASMA course

João Miguel Areias Saraiva
joaomiguelsaraiva@tecnico.ulisboa.pt
Student number 86449 of
Departments of Computer Science and of Bioengineering
Instituto Superior Técnico, Lisboa, Portugal

ABSTRACT

Here it is proposed a multi-agent system for the simulation of genetic circuits in an *in vitro* environment. These simulations help biomedical researchers in the design of protocells (agents) that autonomously deliver drugs to tumorous cells. Simulation works correctly and multi-populations of agents can be designed to test their outcome in the environment.

KEYWORDS

multi-agent system, collective behaviour, synthetic biology, gene circuits

1 INTRODUCTION

Background. Synthetic biology (SB) is the field of bioengineering that provides the tools to prototype new cell-like structures called the protocells [1]. These protocells are said to be synthetic because they do not appear in nature, but they were designed in a laboratory. There is a branch in SB called genetic circuits (GC), where protocells are designed to behave like a computer circuit [2], and they can be coined as *nanorobots*. There is no electronic components in these protocells, but instead these circuits are made of single-stranded DNA (ssDNA) strategically re-designed to mimic the behaviour of a circuit component [1]. They can receive inputs (also ssDNA molecules), and the circuit produces an output (also in the form of ssDNA). Each protocell can then be designed to mimic a transducer, an amplifier or even a logic gate [2]. By having a population of protocells of multiple types, researchers can mimic a complete computer circuit, allowing them to add more complex behaviour to their gene therapy solutions [3]. Usually, this is achieved by a so-called rational design approach [4], which comprehends a cycle of two steps: i) design a protocell and predict its behaviour *in silico*, using computer models; ii) validate the results *in vitro*, in the laboratory.

Problem Definition and its Relevance. Nowadays, to simulate the behaviour of one protocell with a designed genetic circuit is quite a simple process. But in reality, many of these protocells will be produced and *injected* in the environment – a population. And the collective behaviour they will have is not trivial to predict anymore, and complex population simulations are required to test if the outcome is the desired [4]. Also, these therapies involve more than one type of protocells cooperating together [3]. Moreover, these protocells should later be inserted in living organisms, *in vivo*, and since these are very complex biological environments, the outcomes might be unpredictably complex as well. So, a good simulation should answer questions up to this research stage. How will these protocells behave in a foreign biological environment? Will their collective behaviour in this environment be the same as the one predicted before they were inserted there? Will the protocells damage the environment? Will the protocells engage in activities with the local entities that were not expected? All these questions need to be answered in order to redesign the product as many times as needed before proceeding to clinical trials [4].

Objectives. Over the past decade, multi-agent systems (MAS) have been proposed to simulate design cycles of protocells [5–9]. See [10] for an excellent review on the topic. Inspired by these works, in this project, I propose to follow some of them, but in a simpler and, if possible, more generalized way. A MAS suits the requirements of these simulations, where we can predict the collective behaviour of different types of agents – the protocells – in a simple environment – a Petri dish – under the known physical and biochemical laws.

2 SIMULATION APPLICATION

The simulation software was developed in Java 1.8 under the Spring Boot framework. A graphical user interface (GUI) was designed with Vaadin Designs to visually follow the development of the simulations and to customize them as needed. The code and a release version are available here.

3 THE ENVIRONMENT

There were designed three different environments that increase the level of complexity for collective behaviour of the agents to appear, but this project was only developed and tested on one – the Petri dish environment. The other two are intended to be further developed. The Petri dish is like a plate used by researchers at the laboratory, to test biological behaviour, where they can control nearly all variables. Figure 2 (left panel) shows how a Petri Dish should look like. One can also run the simulator to see the graphical representation of the three environments.

For the simulations purpose, the Petri dish environment is represented by a 2D grid of blocks. Blocks have two important properties: (i) each block can be only occupied by one agent at a time and (ii) each agent can occupy multiple blocks. The first only means that the environment does not allow agents to superimpose each other, as it is a physical impossibility in the Petri dish. The second means that some agents are bigger than others and can occupy multiple blocks. As it is depicted in Figure 1, in the software, each agent has a *Position* which is an object composed of a central block, given by a 2D point in the cartesian coordinates (x, y), and an area, given by a set of 2D points, measured by the given entity size in the cartesian coordinates ($w \times h$). For instance, an ssDNA agent can occupy only an area of one block (1×1), but the nanorobot agents can occupy an area of 9 blocks (3×3), since they are bigger than ssDNA agents. This will allow for ssDNA agents to be inside nanorobot agents as cargo, which is different from superimposition. The resolution of the environment GUI may be changed in the simulator, and these areas will change proportionally. From here onwards, the term *Position* and *Point* will be employed distinctly to refer to the center block and area of an agent, and to refer to a single arbitrary block, respectively.

Regarding its properties, the Petri dish environment is:

- **Partially accessible:** Agent sensors will only be able to perceive the nearby conditions. For instance, if an agent is at a given position, it can only sense the conditions of blocks immediately next to its area boundaries. And what it senses from these blocks is not a complete representation of their condition – actually it is only which agents are in those nearby blocks, if any.
- **Non-deterministic:** Since the model uses stochastic methods, one action will not lead to a single guaranteed effect, namely the agents movement through the environment.
- **Static:** While one agent is deciding its next action, the environment will not change. The simulation runs in only one thread, and so in each epoch the simulation *pauses* and computes each agent decision sequentially. This is the major unrealistic factor when compared to real events, but it was a choice of implementation to reduce complexity.
- **Discrete:** There are a finite number of possible actions and perceptions, including for the the movement actuators, since the possible positions of an agent are discrete in a 2-dimensional grid of blocks.
- **Non-episodic:** What happens in one simulation will not have influence on others.

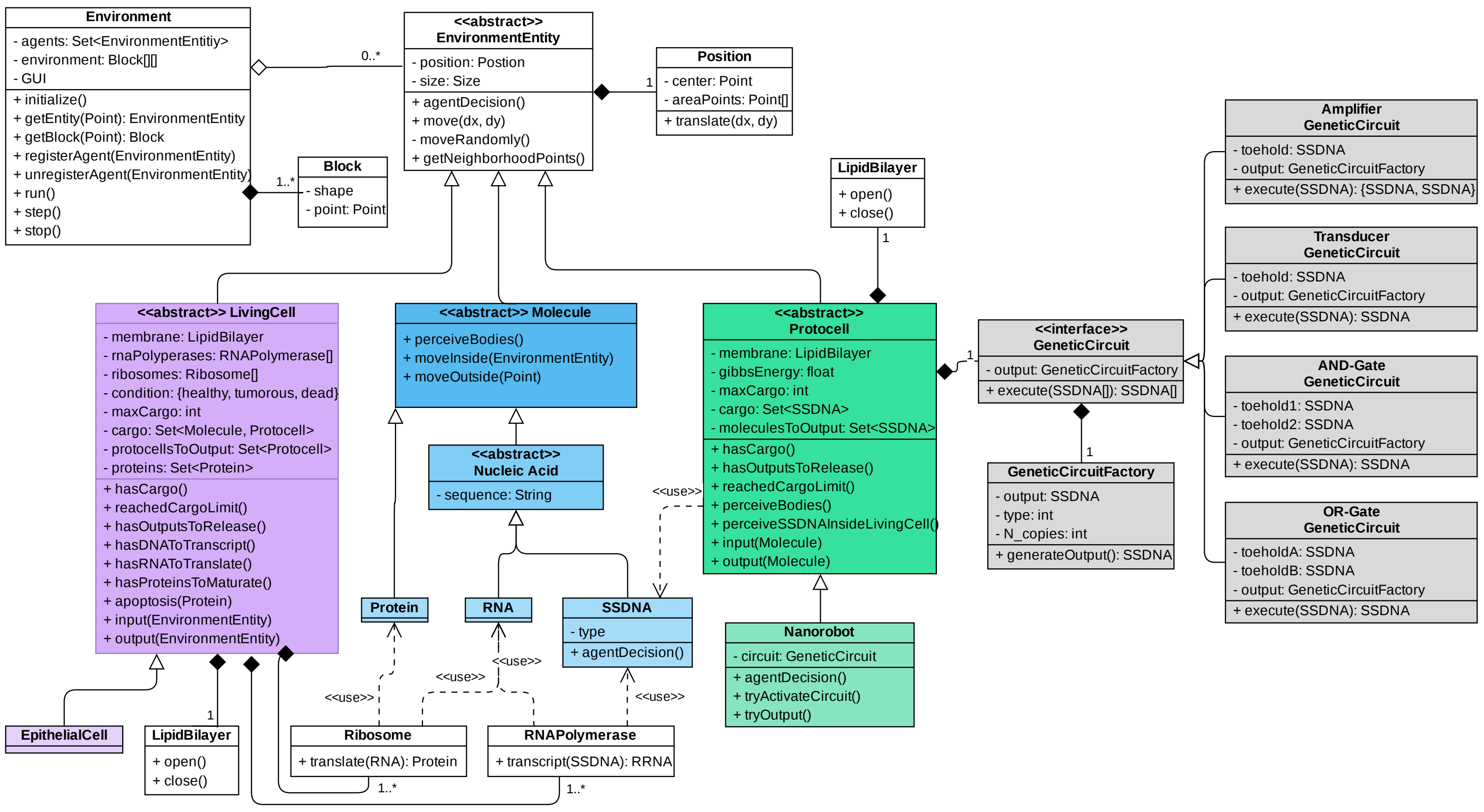


Figure 1: Simulation software "*ProtoCircuits*" organization in a UML class diagram. The tree major agent groups are colored in purple, blue and green. The genetic circuit related classes are colored in gray.

4 THE AGENTS

There are three main super-classes of agents – Protocells, Molecules, and Living Cells – respectively represented in green, blue and purple in Figure 1. Of course, although pretty naive, in terms of the simulations purpose the molecules also need to be implemented as agents as well, because they move around, enter and exit protocells. Although, in reality, they are not really the *actors of this play*.

All these three super-classes of agents derive from *EnvriornmentEntity*, so they can be modelled as agents with a position in the environment. Some environment entities can move in translational (different) ways in the environment grid, and they can have access to whatever it is in the neighborhood points of their area. All three kinds of agents are implemented with a reactive architecture, that maintains an ongoing interaction with the environment, and responds to changes that occur in it in useful time, although they have a minimal internal state for programming purposes. They are not purely reactive agents, since they have tasks of one or two steps, but all of them are triggered based on what they perceive from the environment at a given timepoint. Section 4.5 further discusses why reactive architectures are accurate to represent these agents.

4.1 Protocell Agents

Protocell agents are meant to represent real engineered protocells, as described in the Introduction. Being so, they need a membrane that opens and closes to allow selective cargo to come inside its body, and they have a maximum number of cargo molecules they can take in. They are somehow three times larger than molecules, so they can store some molecules inside as cargo. The properties of a protocell agent are autonomy and mobility (and cooperation – see Conclusion): they move around in the environment according to the laws of physics that rule the environment.

In this system, only nanorobots are implemented as protocells, which are basically protocells with genetic circuitry. In the literature, these concepts are not equivalent, but for the purpose of this report from here onwards *Protocell* and *Nanorobot* terms will be used interchangeably without loss of generality.

The sensors of a nanorobot agent are illustrated in Figure 1 and are described by the following procedures:

- hasOutputsToRelease
- hasCargo
- reachedCargoLimit
- perceiveBodies
- perceiveSSDNAInsideLivingCell

The **hasOutputsToRelease** sensor checks if their are any recently produced output molecules by its genetic circuit inside its body to release, returning True if that is the case.

The **hasCargo** sensor checks if there are any recent input molecules that entered the protocell's body, returning True if that is the case.

The **reachedCargoLimit** sensor checks if there is any space left inside the protocell's body to take in more input molecules as cargo, returning True if that is the case.

The **perceiveBodies** sensor looks for larger bodies in the neighborhood positions, such as living cells, and, if there is any, it attempts to enter inside its body. To do that, it calls the respective living cell' **input** actuator, working as kind of a request to be taken as cargo. If that is possible, the living cell will call the protocell **moveInside** actuator, taking it as cargo.

The **perceiveSSDNAInsideLivingCell** sensor tries to sense which ssDNA molecules are wandering around the body of the living cell the protocell is inside of. Of course, this is only valid when the protocell is inside a living cell. The protocell agent can know which molecules are those by the **continuousProductionOfMolecules** actuator of the living cell agents.

We will see how these sensors will be useful in a moment. The actuators of a nanorobot agent are described by the following procedures:

- move(*dx, dy*)
- openMembrane / closeMembrane
- input(*molecule*)
- output(*molecule*)
- tryActivateCircuit
- moveInside(*living cell*)
- moveOutside(*point*)

The **move**(*dx, dy*) is the actuator that allows for agent mobility. According to the values of *dx* and *dy*, the agent performs a movement of translation of the same number of blocks in the *xx* and *yy* axes, respectively, avoiding collisions. The Petri dish environment can effectively be a 2-dimensional space, so physically no agent can be at the same block at a given timepoint. Notice that there is no sensor for checking if the positions the agent is moving to are free or not, because the environment simulation does not allow for multiple agents at the same block, since that would break a physical law. Based on the Gibbs energy of the protocell agent, it moves to a certain direction accordingly to the thermodynamics second law, that is, to the environment block around it with less energy. But the process is essentially random, so the method is stochastic for simulation purposes.

The **openMembrane** and **closeMembrane** actuators simply open or close the protocell's membrane to allow for input molecules to cross it (enter and exit the protocell).

The **input**(*molecule*) actuator, checks if the there is any space left inside the protocell, by using the **reachedCargoLimit** sensor. It there is, it opens the membrane (**openMembrane**) and takes the molecule as cargo, calling the **moveInside** actuator of that molecule. In the end it closes the membrane (**closeMembrane**). This process is described as *endocytosis* in the biological literature [11]. Molecules can only cross the membrane if it is open, and only one molecule can enter at a time. So, this works like a traffic control programming scheme. Also, the membrane only opens for ssDNA molecules which are the tiniest in the environment.

The **output**(*molecule*) actuator, throws the molecule out of the protocell's body whenever there's space outside (free blocks), by calling the **moveOutside** actuator of that molecule. Notice again there is no sensor to detect free neighbor blocks, since moving the molecule outside to an occupied block would be a physical impossibility. This process if called *exocytosis* in the literature [11].

The **tryActivateCircuit** actuator tries to activate the genetic circuit of the protocell with the molecules it has in the cargo set, by calling the **execute** command pattern function of the circuit object. A genetic circuit may be of different types, as it is explained ahead. If the circuit is successfully activated, it returns one or more outputs, which are added to the *moleculesToOutput* set. Whenever possible, these molecules are outputted with the **output**(*molecule*) actuator.

The **moveInside**(*living cell*) actuator makes the nanorobot move from its current position to the *living cell*'s center position.

The **moveOutside**(*point*) actuator makes the nanorobot move from the living cell's center position it was in to the *point* specified, while giving it an impulsion force in the opposite direction of the larger body to reduce the probability of being recaptured by it in the next time step. This is a common phenomenon in *exocytosis*, although they might be recaptured and that is part of the stochastic nature of the environment.

Let us now discuss how the **agentDecision** procedure is implemented by nanorobots, which is called at each time step. Take a look at the pseudocode at the top of the next column. Essentially, the agent reacts differently depending on whether it is inside a living cell or not. When it is outside, in this order of priority, at each epoch, the nanorobot agent tries to output its recently produced molecules that are waiting in the *moleculesToOutput* set; or tries to activate its circuitry with the molecules it has in the cargo set; or ultimately it moves around in the environment. Only one of these instructions is executed per time step, since protocells are very limited in function and cannot multi-task. Their IF conditions are given by the sensors **hasOutputsToRelease** and **hasCargo** described above. If it is being released by a living cell, it tries to move (hopefully getting some distance from it). And if it is inside a living cell, it does not move (unless the living cell moves), so if it has cargo it tries to activate its genetic circuitry.

4.2 Genetic Circuits

In this Subsection, let us dive in the types of genetic circuits the system allows. All nanorobot agents are the same from the outside in terms of being protocells – they move, they take in cargo and they release output. But what really distinguishes them and gives them characteristic behaviour is the genetic circuit they have inside their nucleus.

Algorithm 1: Nanorobot Agent Decision

```
if is outside then
    if hasOutputsToRelease() then
        output();
    else
        if hasCargo() then
            tryActivateCircuit();
        else
            move();
        end
    end
end
if is being released then
    move();
end
if is inside and hasCargo() then
    tryActivateCircuit();
end
```

Two nanorobot agents are said to be different if they contain a different genetic circuit. A set of nanorobot agents is said to be a population if all agents have the same genetic circuit. For instance, we call a population of agents as a population of amplifiers if all nanorobot agents have amplifier genetic circuits inside. And two or more populations of nanorobots can coexist in the same environment and generate some collective behaviour.

In this work, four templates of genetic circuits were designed that implement the *GeneticCircuit* interface (Figure 1). This interface has a command pattern procedure – **execute** – that activates the circuit. To activate/execute the circuit, one or two ssDNA molecules should be provided and one or two ssDNA molecules can be returned. The cardinality depends on the type of circuit. The four types of circuits are:

- **Amplifier:** By receiving one ssDNA molecule (A), it produces 2 units of the same ssDNA (A).
- **Transducer:** By receiving one ssDNA molecule (A), it produces one unit of a different ssDNA (B).
- **AND-gate:** By receiving two different ssDNA molecules (A and B), it produces one unit of a different ssDNA (C).
- **OR-gate:** By receiving at least one unit of two different ssDNA molecules (A or B), it produces one unit of a different ssDNA (C).

In order to recognize specific types of ssDNA molecules, the circuits have one or two so-called *toeholds*. If the beginning of the nucleic acid sequence of the received ssDNA molecule matches a *toehold*, an output is said to be displaced and the nanorobot agent can later output it to the environment. This phenomenon is known in the literature as toehold-mediated strand displacement, and researchers have recently been taking advantage of it to engineer these circuits [1, 11]. A short video illustrating this phenomenon can be found here, and this is exactly what was implemented in the nanorobot agents in order to simulate what really happens.

An ssDNA molecule matches a *toehold* and displaces an output if the beginning of its sequence is complementary to the *toehold*. For example, an ssDNA of sequence `AACGTCGAT` matches the *toehold* `TTG`, because the ssDNA beginning `AAC` is base-complementary with `TTG`, in that order[1]. This matching is of extreme importance to the system, since all ssDNA molecules will be running around freely in the environment and they should only trigger circuits to which they match. This allows for a *filtering* ability without any complexity (since this knowledge is embedded in all DNA molecules), which will be the fundamental basis of the intelligent-*like* behaviour of the populations.

Beware though the AND-gate circuit is more complicated because it actually requires two strand displacements (two activations) to release an output. And they need to occur by a specific order (as can be seen in the recommended video). So, the circuit has two toeholds and they need to be matched in the correct order for an output strand to be released.

[1]In biology: A and T are complementary; C and G are complementary. [11]

Although, as Lavoisier would say, nothing appears from nowhere, so each nanorobot has a limited capability to release outputs from its circuitry. In fact, when they are being engineered at the laboratory, they are *injected* with multiple (hundreds of) small circuits, all equal, each of which is capable of being triggered only once. So, there is a maximum number of copies that can be outputted. In the software, in order to reduce time and space complexity, instead of creating each nanorobot with multiple circuits, only one template is created, which holds a factory that generates the ssDNA outputs, under the Factory design pattern. After the specified number of copies have been generated, the factory ceases to generate more outputs. In this scenario, the protocell will be no longer useful, but it will be wandering around the environment, difficulting other agents movement.

The reader may have been asking: are these generated ssDNA molecules becoming autonomous agents? Yes, they will become agents after being released to the outside of the protocell, to move freely in the environment, act independently, and to be taken as cargo by other protocells. The released outputs of the circuits are kept in the *moleculesToOutput* set of the protocell agent, and when outputted they are registered in the system as autonomous ssDNA agents.

4.3 Molecule Agents

In Figure 1, although one can notice many classes implementing *Molecule*, that is solely because they share biological common features. For the purpose of the multi-agent simulation, only ssDNA molecules are considered as agents, as they are the only class that implements the **agentDecision** method. The ssDNA molecules, which are the tiniest agents, and can enter inside the protocells – about three times smaller than them – and inside living cells – about nine times smaller than them. They are also modeled as reactive agents that maintain an ongoing interaction with the environment, and respond to changes that occur in it in useful time. To do that they only need one sensor: perceiveBodies.

The **perceiveBodies** sensor looks for larger bodies in the neighborhood positions, such as protocells and living cells, and, if there are any, it calls the respective **input** actuator of that agent, working as kind of a request to be taken as cargo. As aforementioned, if that is possible that entity will call the molecule' **moveInside** actuator, triggering *endocytosis.*

The actuators of a ssDNA agent are three:

- moveInside(*entity*)
- moveOutside(*point*)
- move(*dx, dy*)

The **moveInside**(*entity*) actuator makes the molecule move from its current position to the *entity*'s position center. The *entity* might be a protocell agent or a living cell agent.

The **moveOutside**(*point*) actuator makes the molecule move from the entity's center location it was in to the *point* specified, while giving it an impulsion force in the oposite direction of the larger body to reduce the probability of being recaptured by it in the next time step, just as explained for the protocells.

The **move**(*dx, dy*) is the actuator that allows for agent mobility and it is the same as the protocell's.

The **agentDecision** procedure of ssDNA agents, which is called at each time step, is described in the box at the top of the next column. Resuming, molecules *know* if they are outside or inside some larger body by a control variable, and if they are outside they have space to move randomly and to perceive larger bodies in their neighborhood as they move. Notice how in the time steps the agent is being outputted by a larger body, it will not use the **perceiveBodies** sensor in order to help with the release process, otherwise it would be recaptured. Also, while it is inside some larger body, the agent looses control over its behaviour until it is released to the outside, which will happen eventually.

Furthermore, ssDNA agents have a sequence of nucleic acids, represented as a String, where each character might be `A`, `T`, `C`, or `G`. And this will define what *type* of ssDNA the agent is. Two ssDNA agents are said to be of different *types* (or strands) if they present a different sequence. In the previous Subsection about Genetic Circuits, it was already explained the

Algorithm 2: ssDNA Agent Decision

```
if is outside then
    move();
    perceiveBodies();
end
if is being released then
    move();
end
if is inside then
    do nothing
end
```

filtering capability this sequence allows while triggering circuits, but we shall see later, why this sequence is even more important.

4.4 Living Cell Agents

Lastly, the living cell agents (or just cells) are meant to represent any cell that is present on living beings, such as humans. It can be a heart cell, a skin tissue cell, a neuron, whatever. The software allows for that customization, but in Figure 1 it is only present the epithelial cells for simplicity, which you can recall as being for example a skin cell.

These agents are goal-directed and have multiple components to perform their jobs, with stacks in between. Cells are very complex units, but we can abstract them to one single pipeline of jobs with one final goal, which is to produce proteins. The central dogma of molecular biology abstracts this pipeline quite simply: cells perceive DNA molecules (such as ssDNA), they transcript them into RNA, that RNA is translated to a protein, and that protein is maturated to start performing its task, whatever that is [11]. Resuming, the three steps to achieve the goal of producing a functional protein are: transcription, translation and maturation. And I have modeled the behaviour of the first two in the components of type *RNAPolymerase* and *Ribosome*, respectively, which every living cell has inside. I will not cover these processes in the report, but it is important to say that these processes needed to be coded in the system, since it is through them that the cell rules its behaviour. For instance, a resultant protein A might lead the cell to take nutrients inside, but protein B might cause a catastrophe such as the cell death. And these proteins are codified from the pipeline very beginning through a genetic code *known* by all cells. Hence, in order for this tool to be useful for researchers, they will need to program their ssDNA molecules with the sequences they wish, for them to be transcripted and translated into the correct protein they desire. Experiment 4 will illustrate this importance better.

Regarding the living cell agent sensors, they are basically the same as the protocell's and some more:

- hasProtocellsToOutput
- hasCargo
- reachedCargoLimit
- hasDNAToTranscript
- hasRNAToTranslate
- hasProteinsToMaturate

The sensors **hasProtocellsToOutput**, **hasCargo**, and **reachedCargoLimit** are basically equivalent to the ones described for the protocell agents, but here the cargo is protocells and they should be released after a while after being taken in.

The sensor **hasDNAToTranscript** checks if any recent inputted ssDNA as entered the cell's body, returning True if that is the case. All ssDNAs that are inputted are stacked in a stack to be transcripted by RNA polymerase components. This sensor checks that stack.

The sensor **hasRNAToTranslate** checks if there is any recently produced RNA waiting to be translated, returning True if that is the case. All ssDNAs that are transcripted generate an RNA that is stacked in a stack to be later translated. This sensor checks that stack.

The sensor **hasProteinsToMaturate** checks if there is any recently produced protein waiting to be maturated, returning True if that is the case. All RNAs that are translated generate a Protein that is stacked in a stack to be later maturated. This sensor checks that stack.

This sensors are all internal sensors, but if they were real they would characterize the cell ability to perceive ssDNA, RNA and proteins inside its body space. The actuators of living cell agents are the following:

- move(*dx, dy*)
- openMembrane / closeMembrane
- input(*environment agent*)
- output(*environment agent*)
- continuousProductionOfMolecules()
- apoptosis()

The **move**(*dx, dy*) is the actuator that allows for agent mobility and it is the same as the described before for the other agents. The cell agents also have a membrane and they use **openMembrane** and **closeMembrane** to open and close it, just as protocell agents, only allowing protocells and ssDNA agents to pass through.

The **input**(*environment agent*) is the equivalent actuator of **input** of the protocell agents. But here the living cells can in theory take in any environment agent, since all of them should be smaller than the cell. Of course it checks first if there is any space left inside, by using the **reachedCargoLimit** sensor. It there is, it opens the membrane (**openMembrane**) to take the protocells as cargo and ssDNAs to the transcription stack, calling their respective **moveInside** actuator to provoke *endocytosis*.

The **output**(*environment agent*) actuator, throws out the protocells that were taken as cargo after a couple of epochs. Of course, it waits to whenever there's space outside (free blocks), by calling the **moveOutside** actuator of the protocell agents to provoke *exocytosis*. Notice that the ssDNAs that were inputted are never outputted. Every ssDNA molecule that enters the cell agents goes directly to the transcript-translate machinery, which destroys the ssDNA and returns a protein.

The **continuousProductionOfMolecules** actuator is a simple simulator of all the molecules a cell agent produces continuously. Cells continuously produce ssDNAs to later give rise to proteins needed for their survival [11]. So this actuator is meant to abstract that, and to return the sequence of those ssDNA molecules. This is used by the protocells to *know* which ssDNA molecules are wandering around the cell's body.

The **apoptosis**(*protein*) actuator causes the cell death, known in the biological literature as *apoptosis* – cell programmed death [11]. Some proteins are fatal to the cells and, by irony of cells producing proteins to survive, some of those proteins are produced inside the own cell, leading to its death. Some protein sequences are known to trigger apoptosis, so if the given *protein* as such a sequence, it will trigger the apoptosis phenomenon in the cell, ceasing all machinery (RNA polymerases, ribosomes, membrane), stopping movement forever, and setting the cell condition to DEAD.

Transcription and translation of proteins are internal processes, that have no direct influence on the environment, so they are not considered actuators.

The living cell agents maintain a condition (a state), which might be HEALTHY, TUMOROUS or DEAD. Living cells can be healthy or tumorous and, depending on which, this will make them *say to produce* different molecules in the **continuousProductionOfMolecules** actuator. For instance, most types of tumorous cells present in breast cancer tumors produce some characteristic ssDNA molecules [12] that distinguish them from healthy cells. In experiment 4 we shall see how researchers are taking advantage of this phenomenon to better design protocell circuits.

The **agentDecision** procedure of living cell agents, which is called at each time step, is presented in the top of the next column. Notice how immediately after a protocell agent gets inputted (because it looks to the cell like a legit molecule from the outside), it gets outputted right on the next epoch of the simulation, because once it is inside, it is supposed to be recognized like a strange invading body and would eventually be outputted in a real environment [11]. Nevertheless, once inside, according to the Nanorobot Agent Decision presented above, one epoch is enough to trigger its genetic circuit if the conditions favor that, i.e., if it collects as cargo matching ssDNA molecules wandering around the cell body. This might be ssDNAs produced by the cell itself (by **continuousProductionOfMolecules**) or ssDNAs that entered the cell at the same instant as the protocell and were not transcripted yet.

Algorithm 3: Living Cell Agent Decision

```
if not DEAD then
    if hasProteinsToMaturate() then
        for all Protein p do
            try apoptosis(p);
        end
    end
    if hasProtocellsToOutput() then
        for all Protocell p do
            output(p);
        end
    end
    if hasRNAToTranslate() then
        for all RNA rna do
            ribosome.translate(rna);
        end
    end
    if hasDNAToTranscript() then
        for all ssDNA dna do
            ribosome.transcript(dna);
        end
    end
    if hasCargo() then
        for all Protocell p do
            protocellsToOutput.add(p);
        end
    end
end
```

4.5 Suitability of these architectures

The reactive behaviour is the most appropriate architecture for these type of agents, since the objective is to simulate what would happen in real life with these biological entities. Since these entities behaviour is only caused by (biochemical) reactions with what they encounter in the environment, in order to get the most accurate simulation of the events, with all its inefficiencies and failures, we must let this behaviour develop in a solely reactive way as it is. These biological entities have no planning nor learning capabilities, so the agents that represent them should neither.

Moreover, this agents have no explicit representation nor symbolic model of the world, and their actions are not decided based on logical deduction, so deliberative and deductive architectures are not suited as well. They also do not commit to specific goals nor desires, and they do not try to build a plan to reach those hypothetical goals, with the exception of the living cell agents that have always the same continuous goal of producing proteins. No specific utility function is trying to be maximized, so rational agents are not suited for this type of problem as well.

Nevertheless, it is this simple reactive behaviour that allows for the emergence of intelligent-*looking* collective behaviour, out of very complex environments in our body. And this is exactly what we intend to explore in this work, in the experiments next.

5 EMPIRICAL EVALUATION AND RESULTS

Based on the objectives introduced in the project proposal, here are the metrics to evaluate the multi-agent system:

- Protocell and molecule agents can move through the environment, while not superimposing with each other.
- Single-kind protocell populations can trigger their genetic circuit by solely reacting with the ssDNA agents in the environment.
- 2-wise and 3-wise combinations of different kinds of protocell populations can cooperate together in the environment.
- A tumorous Living Cell can die due to the action of protocell populations.

In the next Subsections, the results of four experiments are discussed thoroughly with pictures of some simulation epochs, but they are also voice-explained in the delivered video.

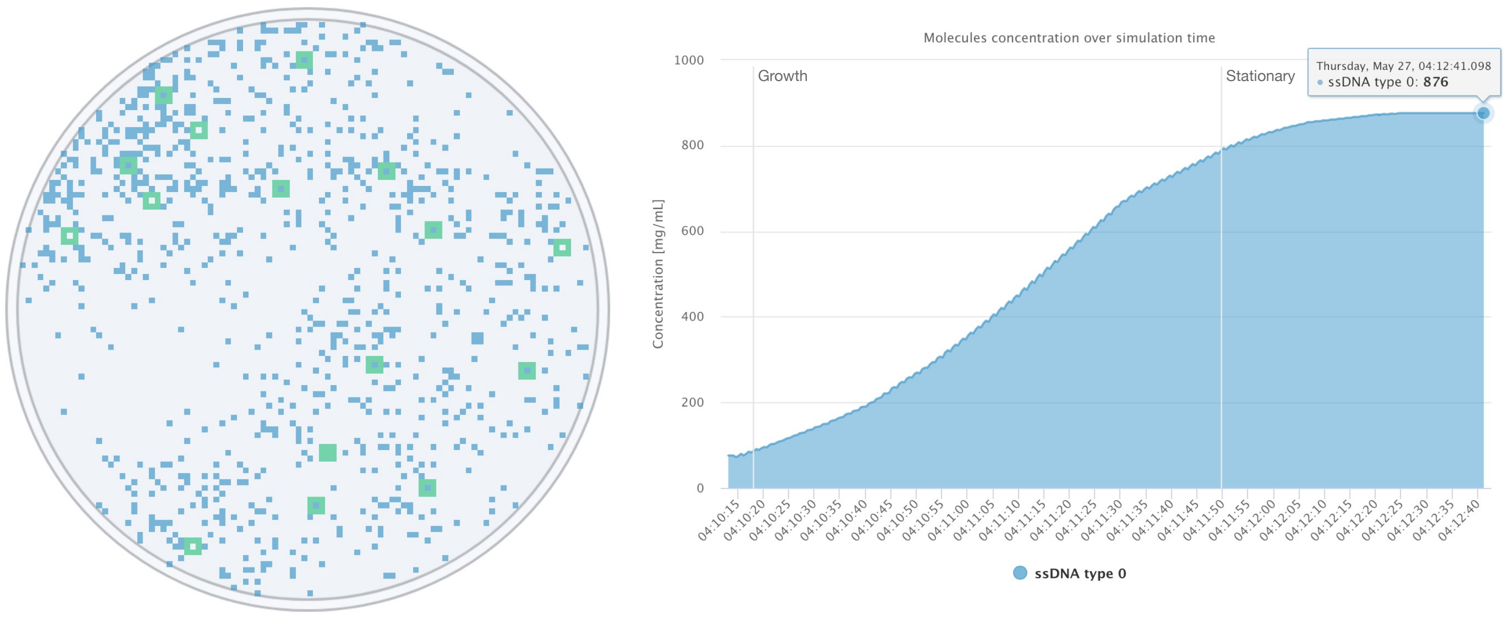


Figure 2: Result environment and concentration plot of experiment 1, retrieved from the *Running view GUI.* Amplifier agents are represented in green and ssDNA type 0 agents are represented in blue.

5.1 Experiment 1: Single protocell population of Amplifiers

Firstly, a simple experiment to test basic behaviour was performed that can be seen in the delivered video at 00:06 min. The Petri dish environment was loaded with 16 amplifier protocell agents. As their name suggests, these protocells were created with an Amplifier genetic circuit, with a toehold `AAC`, and they release two copies of the matched inputs. So, by loading the environment with 76 free ssDNAs molecules of sequence `TTGACTGTTAC`, randomly placed, these will be recognized by the protocells' circuit since its toehold (`AAC`) is complementary to these strands beginning (`TTG`). An illustration of the circuit is represented in Figure 3, to recall that by each of these ssDNAs recognition, two copies will be released to the environment. Important to say that the protocells were loaded with 100 copies of possible outputs, as they could be in real life.

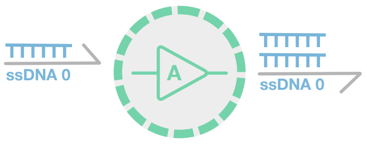


Figure 3: Circuit of one protocell population of Amplifier agents (in green) that duplicate each ssDNA type 0 agent (in blue).

Figure 2 shows the result plot, drawn in the *Running view* of the GUI for this simulation. The plot represents the time series of the ssDNA concentration over a simulation of 3 minutes and 25 seconds (epoch = 0.5 s). Let us examine it in detail. Initially, the environment presents a constant concentration of 76 ssDNA type 0 molecules and in a couple of seconds their concentration starts increasing exponentially for about 1 minute and 33 seconds. More precisely, we notice a increment rate of about 7.58 ssDNA molecules per second (mol/s). This is known in the biological literature as the *growth phase* [11], marking this as a first sign that this might be an accurate simulation of a real-word experiment. From the timepoint 04:11:50 onwards, we can observe a lower increment rate of the ssDNA population (about 2.86 mol/s), suggesting we enter the *stationary phase*, described in the literature [11]. The stationary phase is inevitable since most of the protocells have released all their possible output strands, that were 100. At 04:12:23, the concentration curve reaches a *plateau* at 876 ssDNA molecules. This indicates that all protocells have released their 100 copies of output ssDNA. Equation 5.1 gives us the theoretical maximum amplification of ssDNA molecules, given the initial amount in the environment $N_i > 0$, the number of amplifier agents, P, and the number of copies each agent has, C.

$$N_f^{amp} = N_i + (P \cdot C) - (P \cdot \frac{C}{2}) = N_i + P \cdot \frac{C}{2} \tag{1}$$

If we introduce the configuration of this experiment in Equation 5.1, we can confirm that indeed 876 copies was the maximum number of ssDNA molecules there could be in the environment, after amplification, and being this an absolute maximum we can stop the simulation being sure that no more ssDNA will be released.

$$N_f^{amp} = 76 + 16 \cdot \frac{100}{2} = 876$$

Notice in the delivered video how ssDNA clusters are formed around the protocells half-way the *growth phase*. This is described in the biological literature as *aggregation* [11] and, although this is a fairly simple environment and *aggregation* contributed to rapidly achieve the absolute maximum, this phenomenon might become a problem in more rich environments as we shall see next.

5.2 Experiment 2: Two-wise protocell populations of Amplifiers and Transducers

In the second experiment, we can add another population of protocell agents to the previous configuration, namely a population of transducer protocells. As illustrated in Figure 5, these transducers should be designed to be triggered when they take ssDNA type 0 as cargo. For each ssDNA type 0 (`TTGACTGTTAC`), the triggering of these circuits should output one ssDNA type 1 molecule (`CCGCTACGAGCG`). So in order to do this, we should design the transducer circuits with a toehold of `AAC`, just like the toehold of the amplifiers, in order to recognize the ssDNA type 0 sequence as an eligible strand to displace the output.

Before running the simulation, the environment was loaded with an initial amount of 40 ssDNA type 0 molecules and none ssDNA type 1. Also, 7 amplifier agents and 7 transducer agents were randomly placed on the Petri dish. The amplifier agents dispose of 50 copies to amplify, while the transducer agents will dispose of 100 copies to transduce. The simulation run for 15 minutes (epoch = 0.5 s) and can also be seen fast-forwarded in the delivered video beginning at 00:35 min.

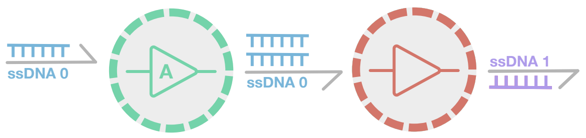
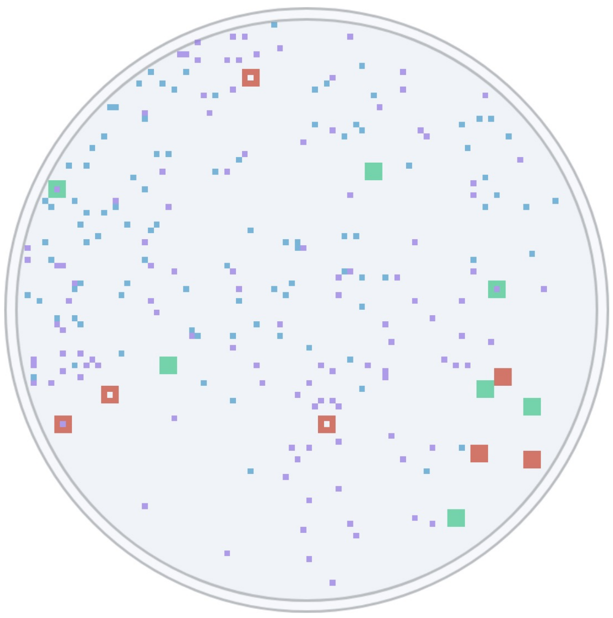


Figure 5: Circuit of two protocell populations of Amplifier agents (in green), that duplicate each ssDNA type 0 agent (in blue), and Transducer agents (in red) that convert each ssDNA type 0 into one ssDNA type 1 agent (in purple).



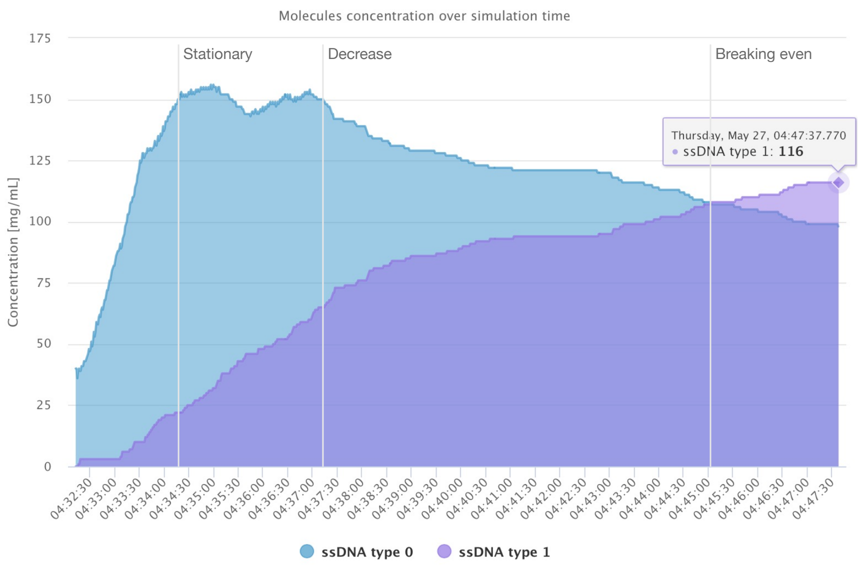


Figure 4: Result environment and concentration plot of experiment 2, retrieved from the *Running view GUI*. Amplifier agents are represented in green, transducer agents in red, ssDNA type 0 agents in blue, and ssDNA type 1 agents in purple.

The ssDNA type 0 agents start being amplified exponentially and just as before we notice a *stationary phase* – Figure 4. The first molecules of ssDNA type 1 start appearing in the environment also at the beginning, but in a much lower rate than type 0. Notice that, transducer agents input the ssDNA type 0 molecules in the environment and in the first 3 minutes although they are being consumed by them, their concentration does not decrease. This is because the rate of production of type 0 molecules by the amplifier agents is eager than the rate of consumption by the transducer agents (+2 : -1). So far it appears that every agents is full-filling their coded behaviour.

Only at the timepoint 04:37:15 we can start observing an assumed decrement on the concentration of ssDNA type 0. What happened? If the environment were to be populated homogeneously, here we could assume that half of the amplifier circuit factories would have released at least 25 copies of ssDNA type 0, leaving the population as a whole producing type 0 molecules at a rate lower than the rate of consumption by the transducers.

As expected, at the timepoint 04:45:00, the amount of ssDNA type 1 in the environment surpasses the amount of ssDNA type 0, which is what we are going to call the *breakeven* point. And when stopping the simulation, we can observe 116 ssDNA type 1 molecules and 98 ssDNA type 0 molecules. One could only expect that if we would let the simulation run for more time, all ssDNA type 0 would be consumed and the concentration of ssDNA type 1 would reach a plateau of 215 molecules. This result comes from Equation 5.1 which states that only with the amplifiers population it would be achieved an environment with 215 ssDNA type 0 molecules. So, since each of these is transduced into one ssDNA type 1, that same amount (215) would be the type 1 expected amount. However, bear in mind that the theoretical maximum production of ssDNA type 1 by the transducer agents would be 700, since the population has 7 agents each of which designed to output 100 copies.

5.3 Experiment 3: Three-wise protocell populations of Amplifiers, Transducers, and AND-gates

In the third experiment, we can increment a little bit, by adding one more population of protocell agents. As Figure 6 illustrates, we can build a macro genetic circuit with any protocell agent populations. Here an AND-gate population is added that should be triggered when ssDNA type 0 is inputted and then ssDNA type 1 is inputted. Beware that this correct order should not be disregarded, as it is how they work in reality, and the system accounts for that. What happens is that the first input binds to the first toehold and displaces one part of the circuit, which reveals the second toehold, and only then the second input can displace the rest, releasing the output [11].

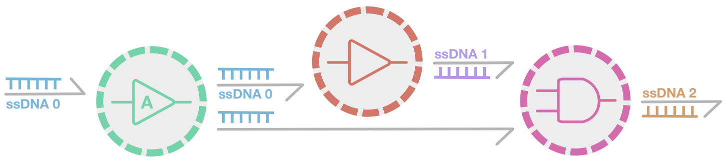


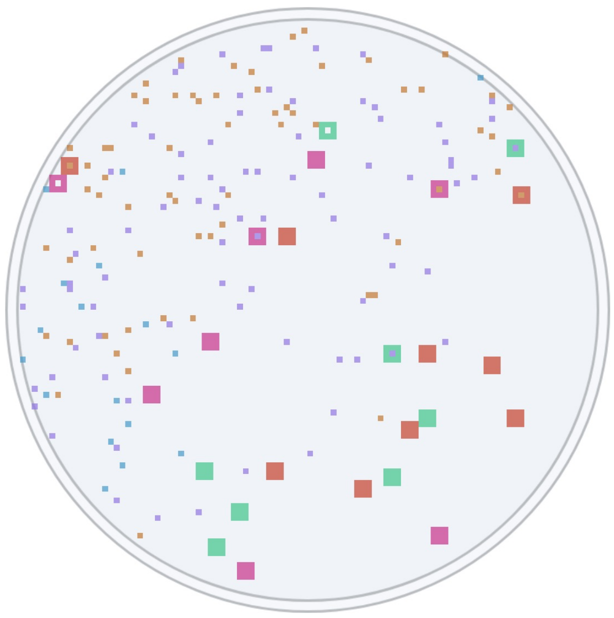
Figure 6: Circuit of three protocell populations of Amplifier agents (in green), that duplicate each ssDNA type 0 agent (in blue), and Transducer agents (in red) that convert each ssDNA type 0 into one ssDNA type 1 agent (in purple), and also AND-gate agents (in pink) that consume one ssDNA type 0 and one ssDNA type 1 to produce one ssDNA type 2 agent (in pale orange).

In order to achieve this collective behaviour we need to design the AND-gate toeholds correctly. The first toehold should be `AAC` which is complementary to the beginning of ssDNA type 0 sequence, and the second toehold should be `GGC` which is complementary to the beginning of ssDNA type 1 sequence. The output for these AND-gate agents can be an arbitrary ssDNA, let's say, ssDNA type 2 with the sequence `ACTCGATGTC`. It might take a while to achieve the first type 2 molecules in the environment, since this agent requires two asynchronized inputs to release the output.

The simulation started with 41 initial ssDNA type 0 molecules randomly spread in the environment and none of the other types. There were also added 8 amplifier agents (50 copies each), 7 transducer agents (100 copies each) and 8 AND-gate agents (100 copies each) to the environment. Note this numbers are *quasi*-arbitrary, since I am trying to balance the production rates to achieve a pedagogical experiment. The simulation run for 1 hour and 44 minutes (epoch = 1 s), which is equivalent to 52 minutes (epoch = 0.5 s), and the reader can follow it fast-forwarded in the delivered video at 01:28 min.

As Figure 7 suggests, ssDNA types 0 and 1 follow the same behaviour as in the previous experiment. The first type 2 molecule, result of some AND-gate agent, is only observed after 7 minutes from the beginning of the simulation, and considerable growth of its concentration only after 13 minutes from the beginning.

The ssDNA type 2 molecules continue to increase and if one would let the simulation run for another hour, we could expect the ssDNA type 2 production to cease when all the type 0 molecules (only 17 left) were to be consumed. As we can see, the overall orange curve is already reaching a *plateau*.



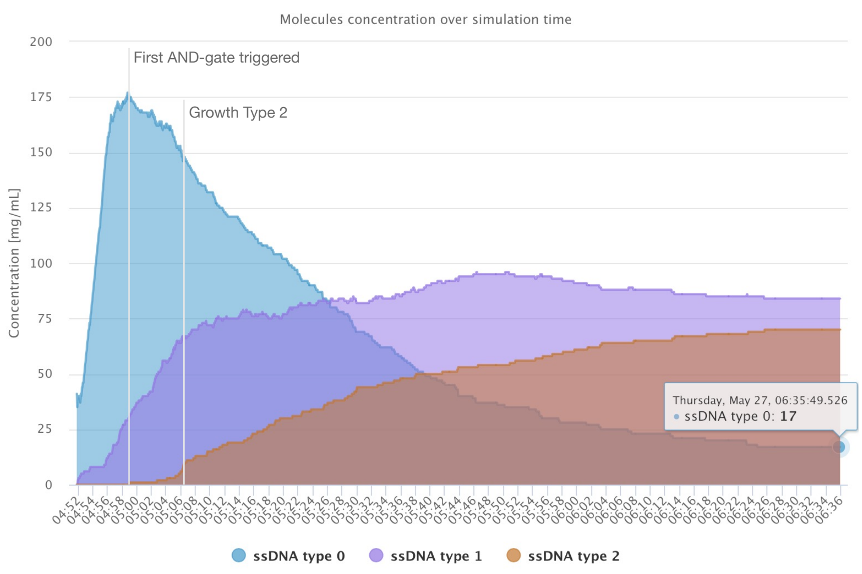


Figure 7: Result environment and concentration plot of experiment 3, retrieved from the *Running view GUI*. Amplifier agents are represented in green, transducer agents in red, AND-gate agents in pink, ssDNA type 0 agents in blue, ssDNA type 1 agents in purple, and ssDNA type 2 agents in pale orange.

5.4 Experiment 4: AND-gate protocells population selectively kill tumorous cells

Now that we understand that AND-gate agents work, we are ready to test whether we can achieve intelligent-*like* behaviour by killing tumorous cells and leaving healthy cells intact. To do this I present in Figure 8 a simple population of AND-gate protocells, but notice that in reality researchers can build much more complex macro circuits out of many populations, with the building blocks I have presented here.

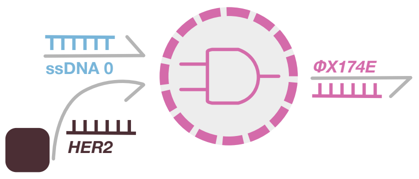


Figure 8: One population of AND-gate agents (in pink), that consume ssDNA type 0 and tumorous-specific ssDNA (e.g. *HER2* [12]) to produce ssDNA that leads to cell apoptosis (e.g. Φ*X174E* [12]).

We know, by fact, that some tumorous cells continuously produce characteristic DNA strands (e.g. *HER2*), so here let's give as an example they produce the sequence `AACTGCGCATG` and ssDNA strands with this sequence can be found inside their body. Now, let's suppose a physician would inject locally some ssDNA type 0 in a specific site of a tumor, of sequence `CCGCTACGAGCG` (different from the previous one). So, the AND-gate we want to design is one where the first toehold is `GGC`, to recognize the ssDNA type 0, and the second toehold would be `TTG` to recognize the interior ssDNA strands that can be found inside those tumorous cells. And the output that the AND-gates will release upon sensing these two strands will be an ssDNA of sequence `TACGGCGCTATC`. This sequence codifies the protein `MPR-`, and let's assume it causes apoptosis of any cell (healthy or tumorous).

The simulation started with 80 units of ssDNA type 0 agents and 18 AND-gate agents, randomly placed, and it can be followed in the delivered video at 02:20 min. Also, as depicted in Figure 9 two healthy cells (light blocks) and two tumorous cells (dark blocks) were placed in the Petri dish. We can notice in the beginning that many AND-gates take ssDNA type 0 as cargo and displace the first toehold of its circuit, but only when they enter a tumorous cell they can take the second tumorous-specific ssDNA as cargo and can complete the circuit displacement, releasing the output. This ssDNA output (`TACGGCGCTATC`) is not an agent, and is perceived by the cell agent as any other ssDNA molecule to transcript and translate, as it would do with one of its own. After translation it produces the `MPR-` protein, causing its death. This occurs first to the top-left-most cell and later to the bottom-left-most cell. We can confirm the cell transitions to the `DEAD` condition, because the GUI turns it into a necrotic-look color (more dark) and it stops tanking any other agents to the inside – see Figure 9.

In the delivered video, the reader may notice that AND-gate agents are also taken as input into the healthy cell agents, and their circuit was already partially-displaced from some ssDNA type 0 outside, but no output was released since inside tumorous cells the tumorous-specific ssDNA is not present. So, they end up moving outside the healthy cells without outputting the therapy. Furthermore, notice how important it is the protocells only release the output therapy inside the tumorous cells. If they would release it outside, those ssDNA agents would be wandering around the environment and, eventually, could be inputted by an healthy cell agent, which also has that capability, and would then be transcribed and translated into the `MPR-` protein, ending up killing the healthy cell.

6 CONCLUSIONS

This work correctly simulates protocell populations as a multi-agent system and the presented experiments allow us to guarantee the correctness of the collective behaviour that emerges when we join multiple protocell populations together. The experiments are also adequate to their purpose, which is to simulate what normally happens at the laboratory, which is to incrementally increase the level of complexity of our design.

Can we say these agents implement a cooperative social architecture? Well, that is a tricky question. By definition, a cooperative behaviour can be present when agents cannot achieve a common goal alone by themselves, or the outcome will be better if they cooperate. If the goal is to kill tumorous cells, one can argue that protocells were not implemented to have that goal. Indeed, they were not explicitly told to kill a tumorous cell, but the whole macro-circuit was engineered to follow pathways that will eventually lead to tumorous cells apoptosis. Other authors coined the term "bi-directional communication" between protocell agents, where the "messages" exchanged are the ssDNA molecules. Although it is a good metaphor, I would have to discard that argument to state that this is a cooperative behaviour by messages, since the protocell agents are not implemented to send a message, whatever that is, to other agent nor broadcast it. Therefore, I guess we can agree there is a meta-cooperation between protocell and ssDNA agents to kill tumorous cells, but it is not rational for them, since the goal is implicit by who engineered them, and not explicit in the agents architecture.

I believe the proposed solution has quality to be used by other users. As aforementioned, there are some alternative solutions in the market, but none of them allows for a visual representation of the environments. Later this year, in November, me and a team from Técnico will represent Portugal in the iGEM synthetic biology competition in Paris, and I intend to improve and bring the system there to better help with our pre-design and conceptualization.

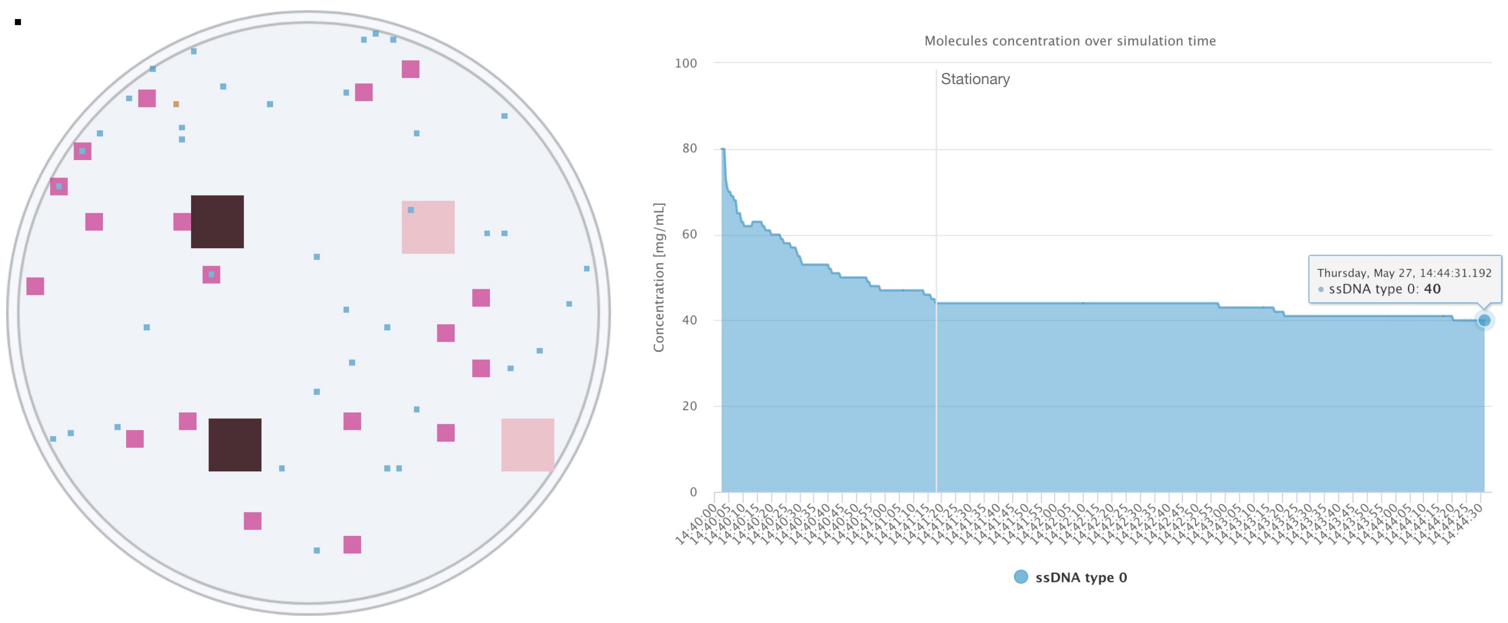


Figure 9: Result environment and concentration plot of experiment 4, retrieved from the *Running view GUI*. AND-gate agents are represented in pink, ssDNA type 0 agents in blue, and tumorous cells (already dead) are represented in a necrotic colour, and healthy cells are pale pink.

REFERENCES

[1] Alex Joesaar, Shuo Yang, Bas Bögels, Ardjan van der Linden, Pascal Pieters, B. V. V. S. Pavan Kumar, Neil Dalchau, Andrew Phillips, Stephen Mann, and Tom F. A. de Greef. DNA-based communication in populations of synthetic protocells. *Nature Nanotechnology*, 14(4):369–378, April 2019.

[2] Jennifer A N Brophy and Christopher A Voigt. Principles of genetic circuit design. *Nature Methods*, 11(5):508–520, May 2014.

[3] Ferdinand Sedlmayer, Dominique Aubel, and Martin Fussenegger. Synthetic gene circuits for the detection, elimination and prevention of disease. *Nature Biomedical Engineering*, 2(6):399–415, June 2018.

[4] Jing Wui Yeoh, Salvador Gomez-Carretero, Wai Kit David Chee, Ai Ying Teh, and Chueh Loo Poh. Genetic Circuit Design Principles. In Gerald Thouand, editor, *Handbook of Cell Biosensors*, pages 1–44. Springer International Publishing, Cham, 2020.

[5] T. Emonet, C. M. Macal, M. J. North, C. E. Wickersham, and P. Cluzel. AgentCell: a digital single-cell assay for bacterial chemotaxis. *Bioinformatics*, 21(11):2714–2721, June 2005.

[6] Guopeng Wei, Paul Bogdan, and Radu Marculescu. Efficient Modeling and Simulation of Bacteria-Based Nanonetworks with BNSim. *IEEE Journal on Selected Areas in Communications*, 31(12):868–878, December 2013.

[7] Thomas E. Gorochowski, Antoni Matyjaszkiewicz, Thomas Todd, Neeraj Oak, Kira Kowalska, Stephen Reid, Krasimira T. Tsaneva-Atanasova, Nigel J. Savery, Claire S. Grierson, and Mario di Bernardo. BSim: An Agent-Based Tool for Modeling Bacterial Populations in Systems and Synthetic Biology. *PLoS ONE*, 7(8):e42790, August 2012.

[8] Timothy J. Rudge, Paul J. Steiner, Andrew Phillips, and Jim Haseloff. Computational Modeling of Synthetic Microbial Biofilms. *ACS Synthetic Biology*, 1(8):345–352, August 2012.

[9] Laurent A. Lardon, Brian V. Merkey, Sónia Martins, Andreas Dötsch, Cristian Picioreanu, Jan-Ulrich Kreft, and Barth F. Smets. iDynoMiCS: next-generation individual-based modelling of biofilms: iDynoMiCS for Biofilm Modelling. *Environmental Microbiology*, 13(9):2416–2434, September 2011.

[10] Thomas E. Gorochowski. Agent-based modelling in synthetic biology. *Essays in Biochemistry*, 60(4):325–336, November 2016.

[11] Akif Uzman. *Student companion to [accompany] fundamentals of biochemistry life at the molecular level.* Wiley, Place of publication not identified, 2012. OCLC: 1118010640.

[12] Nida Iqbal and Naveed Iqbal. Human Epidermal Growth Factor Receptor 2 (HER2) in Cancers: Overexpression and Therapeutic Implications. *Molecular Biology International*, 2014:852748, 2014.