



# Simulation, Visualization and Experimental analysis for Population Protocols and Network Constructor in 2-Dimensional Case

Author:  
Supervisor:

Haoxuan Wang  
Dr. Othon Michail  
Dr. Vitaliy Kurlin

May 4, 2018

## **Declaration of academic integrity**

I confirm that I have read and understood the University's Academic Integrity Policy.

I confirm that I have acted honestly, ethically and professionally in conduct leading to assessment for the programme of study.

I confirm that I have not copied material from another source nor committed plagiarism nor fabricated, falsified or embellished data when completing the attached piece of work.

I confirm that I have not copied material from another source, nor colluded with any other student in the preparation and production of this work.

Date: May 4, 2018

## **Abstract**

The abstract[1] [2] [3]

# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Aims and Objectives . . . . .	1
1.2	The challenges in the project . . . . .	1
1.2.1	Heterogeneous for different types of Models . . . . .	1
1.2.2	Heterogeneous for different types of Protocols . . . . .	1
1.2.3	Human factor: Lacking Experience for model visualization . . . . .	1
1.2.4	Brief introduction to the programme . . . . .	1
1.2.5	Evaluation of the project . . . . .	2
<b>2</b>	<b>Background</b>	<b>2</b>
2.1	The problem to be solved . . . . .	2
2.2	Existing solution . . . . .	2
2.3	Brief Introduction to Population Protocols [1, 4] . . . . .	2
2.4	From Population Protocols to Network Constructor [2] . . . . .	3
2.5	Grid Terminating Network Constructor [3] . . . . .	4
2.6	Vector transformation: handle coordinate rotations in 2-dimension . . . . .	5
2.6.1	2D centred coordinate rotation, with origin point as the centre . . . . .	6
2.6.2	2D centred coordinate rotation, with any points as the centre . . . . .	6
2.6.3	Affine transformation form . . . . .	7
<b>3</b>	<b>Data Required</b>	<b>7</b>
<b>4</b>	<b>Design</b>	<b>7</b>
4.1	Anticipated components of the system . . . . .	7
4.2	Data structure, algorithms and Pseudo-code for key method . . . . .	8
4.2.1	Adjacency List . . . . .	8
4.2.2	Graph . . . . .	8
4.2.3	Terminating grid network constructor: Check acceptable configuration & Handle rotation . . . . .	8
4.3	Interaction (Sequence) Design . . . . .	8
4.4	UI Interface Design . . . . .	10
	<b>Bibliography</b>	<b>11</b>
<b>A</b>	<b>Brief Proof for equation (1)</b>	<b>12</b>
<b>B</b>	<b>Brief Proof for equation (3)</b>	<b>13</b>

# 1 Introduction

## 1.1 Aims and Objectives

The project aimed to study general population protocols [1] and its two derived model, network constructor [2] and terminating grid network constructor [3] (specifically for grid network construction). It also attempted to experimentally simulate, visualise and compare these protocols via building the simulator and visualizer.

## 1.2 The challenges in the project

### 1.2.1 Heterogeneous for different types of Models

The theoretical models involved in three main different models initially originated in population protocols. These three models share inherently common points but there are also some conceptual differences in between them. For instance, the network constructor [2] and terminating grid network constructor [3] involve state of connections in between two nodes while the original population protocol does not. The node of terminating grid network constructor has its complexity structurally compared with the other two types of model.

### 1.2.2 Heterogeneous for different types of Protocols

The protocols discussed in the related papers [1, 2, 3] involve many different protocols. The protocols are totally different on many characteristics, such as their different computational ability, different ending in either or termination, computation target. These differences between protocol to protocol may lead the simulator and visualizer hard to develop and test.

### 1.2.3 Human factor: Lacking Experience for model visualization

Prior to this project, the author has totally no experiences on model simulation and also no knowledge on what kind of related library will be involved. Learning may take more time than its expected.

### 1.2.4 Brief introduction to the programme

The final programme contains an UI with an fix-sized area to illustrate the interaction process of a particular protocol and shows the states of elements\* in the population. In addition, it contains a information panel containing some related information with regard to the population itself, including:

- Number of nodes
- Number of nodes distinguished in different status
- Number of selections for pairs of nodes<sup>†</sup> that scheduler had took
- Number of effective interactions the population executed

---

\*"Elements" refers nodes in general population protocol, but also includes edge if the protocol involves edge states.

<sup>†</sup>may also include pair of ports for terminating grid network constructor

Additionally, it provides a set of parameters' settings regarding the initial configuration of a protocol and a population to be simulated, which includes:

- The number of nodes included in the simulation
- The initial state for each node<sup>‡</sup>
- The protocol type (and also different sets of transition rules for the protocol)
- Option on whether to use fast-forward simulation method for initially  $n$ -times selection, and the value of  $n$  if the option is enabled. A fast-forward simulation executed in the model but does not present that process in the viewer so it normally faster than the case that does not enable this option.

### 1.2.5 Evolution of the project

In general, the simulator successfully implemented a series of different protocols for the three model mentioned above.

**UI** The UI functions of simulator is verified through a large number of different population simulations. This ensures the UI functions work as they expected in design stage. These experiments on theoretical model may also be asserted the correctness of model through the output configuration of these simulations.

**Model** The model partition of the simulator developed through Testing driven development method. According to the specification, the testing suits had been written before the written of any model code . The model code has to pass all testing suits after their implementation.

## 2 Background

### 2.1 The problem to be solved

### 2.2 Existing solution

### 2.3 Brief Introduction to Population Protocols [1, 4]

Population protocols are theoretical models for distributed computation. The model contains a collection of indistinguishable agents. They (i.e. the agents) carry out computation tasks through directly pair-wised interactions. The interaction pattern of agents is unpredictable from perspective of agents themselves and is controlled through an adversary scheduler with fairness constraints.

**Formal Definition** A protocol can be formally defined as:

- $Q$ , a finite set of passible states for an agent,
- $\Sigma$ , a finite set of input alphabet,

---

<sup>‡</sup>The state of edge for network constructors (i.e. network constructor and terminating grid network constructor) should be always "0" (i.e. inactivated) at initial, so it is omitted here.

- $Y$ , a finite set of output range,
- $\iota = \Sigma \rightarrow Q$ , is an input map from  $\Sigma$  to  $Q$ , hence  $\iota(\sigma)$  represents the initial state whose input is  $\sigma \in \Sigma$
- $\omega = Q \rightarrow Y$ , is an output map from  $Q$  to  $Y$ , and
- $\delta \subseteq Q^4$ , a transition relation describes how pairs of agents can interact.

A *configuration* for a population is a vector of all the agents' states. Because agents with a same state are indistinguishable with each other, each configuration could also be an unordered multiset of states. It can be represented as  $C$ .

At any point of the discrete time, the interaction is unpredictable from the perspective of the agents in the population and the population itself. The interactions at any time are decided by a adversary scheduler necessarily enforced with *fairness* condition. The fairness condition means that the scheduler cannot avoid a possible step forever. Formally, it means if a infinitely often configuration  $C$  and  $C \rightarrow C'$ , the  $C'$  must also appear infinitely of in the execution.

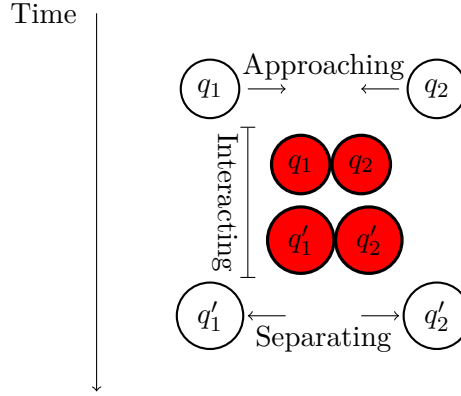


Figure 1: A typical (simple) interaction in population protocol

The current implementation of this work assumes a more strict scheduler called random scheduler, which resulting uniform random interactions (i.e. at each step, it presents equally possibility for every pair of agents to interact.) Essentially, a random scheduler is a "fair" scheduler, but a "fair" scheduler is not necessarily a random scheduler. The assumption actually increases the power of the model because it allows a leader agent detects the absence of agents in a particular state after a long enough waiting. Note this is only for simplifying the current implementation and possibly extends to other fair schedulers that is not a random scheduler in the future.

## 2.4 From Population Protocols to Network Constructor [2]

A variance of population protocol is called "network constructor" [2]. It involves the extra elements do not existing in general population protocol called "edge", which is the connection in between any two agents (or "processes"). In general, the connection is similar to the agents, on the characteristic that they all only have a finite number of states. For instance, if a connection have  $k + 1$  states, state 0 may represent the connection does not exist and for state  $i \in \{1, 2, 3, \dots, k\}$  shows the strength for a particular connection.

In the following paragraph and the currently implemented simulator, we solely consider the simplest case containing only *on* and *off* two states for edges. A connection is said to be in *on* or *active* state, if at any discrete time, the connection exists in between two particular nodes; otherwise if the connection is stated as in *off* or *inactive* state. At initial, all edge are in *off* state, which means that there is no connection in a given population at discrete time 0. The state for an edge may or may not change through an interaction in between the two nodes it connected or disconnected to.

The network constructor also follows an adversary scheduler with "fairness" constraint. This is exactly same as the paragraphs mentioned in the previous section 2.3.

**Formal Definition** Formally, a network constructor can be defined as:

- $Q$ , a finite set of passible states for an agent,
- $Q_{out}$ , a finite set of output range,
- $q_0 \in Q$ , is initial state of node,
- $\delta : Q \times Q \times \{0, 1\} \rightarrow Q \times Q \times \{0, 1\}$ , a transition function, where the  $\{0, 1\}$  is the states for edges with initial value 0.

The main target of this model concentrates on network construction rather than specific function computation, which is the task for general population protocol. Notice that a transition can be either *effective* or *ineffective*. Define  $\delta(a, b, c) = (a', b', c')$  as a transition function (Here,  $a, b$  are "node-state" whereas  $c$  is state for edge.) as it is in the formal definition,  $\delta_1(a, b, c) = a'$ ,  $\delta_2(a, b, c) = b'$  (representing two nodes states' change) and  $\delta_3(a, b, c) = c'$  (representing edge state change), the transition  $(a, b, c) \rightarrow (a', b', c')$  is called effective if at least one  $x \in \{a, b, c\} \neq x'$ ; otherwise it is called ineffective.

Name the set of nodes (or "distributed processes" under this context)  $V_I$  and the set of pairs of nodes as  $E_I$ . A configuration  $C$  is a mapping  $V_I \cup E_I \rightarrow Q \cup \{0, 1\}$  determines the states of nodes and edges in the population at any discrete time. The output of configuration  $C$  is defined as the graph  $G(C) = (V, E)$  where  $V = \{u \in V_I : C(u) \in Q_{out}\}$  and  $E = \{uv : u, v \in V, u \neq v, \text{ and } C(uv) = 1\}$ .

## 2.5 Grid Terminating Network Constructor [3]

Another paper [3] presents a different automata but very similar to network constructor. Each node in this model has a fixed number of ports with it. In the 2-dimensional case, it will have 4 distinguished ports  $p_y, p_x, p_{-y}$  and  $p_{-x}$ , may simply donated as  $u, r, d$  and  $l$ , respectively. The port, which neighbour with each other, are also perpendicular to each other, forming local axes. Hence,  $u \perp r, r \perp d, d \perp l$ , and  $l \perp u$ . The coordinates are only for local purposes and do not necessarily represent the actual orientation of a node. A connection (or edge) can only be built through pairs of ports, which is different from the previous model (network constructor).

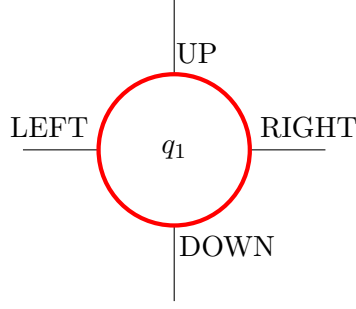


Figure 2: Structure of a node in Grid Terminating Network Constructor model

**Formal Definition** Formally, a terminating grid network constructor in 2-D can be defined as:

- $Q$ , a finite set of passible states for an agent,
- $Q_{out}$ , a finite set of output range,
- $q_0 \in Q$ , is initial state of node
- $\delta : (Q \times P) \times (Q \times P) \times \{0, 1\} \rightarrow Q \times Q \times \{0, 1\}$ , a transition function, where the  $\{0, 1\}$  is the states for edges with initial value 0.
- When required, there will be also a special initial leader-state  $L_0 \in Q$  defined.

A transition can be either *effective* or *ineffective*. Define  $\delta((a, p_1), (b, p_2), c) = (a', b', c')$  as a transition function as it is in the formal definition, it is called effective if at least one  $x \in \{a, b, c\} \neq x'$ ; otherwise it is called ineffective.

Every configuration  $C$  forms a set of shapes  $G[A(C)]$ , where a shape means that the network induced by active edges of  $C$ . Given the geometrical restrictions (i.e. the connection are made at unit distance and are perpendicular whenever they correspond to consecutive ports of a node), not all possible  $A(C)$  are valid.  $A(C)$  is valid if any connected component defined by it is a subnetwork of 2D grid network with unit distances. A valid  $A(C)$  at time  $t - 1$  also restricts the possible valid  $A(C)$  at time  $t + k$ , where  $k \in Integer$  and  $k \geq 0$ .

The paper [3] also defines a set of shapes  $G_{out}(C) = \{V_s, E_s\}$  as output of a configuration where  $V_s = \{u \in V : C_v(u) \in Q_{out}\}$  and  $E_s = A(C) \cap \{(v_1, p_1) : v_1 \neq v_2 \in V_s; p_1, p_2 \in P\}$ . Less formally, the output shapes of a configuration contains all nodes in output states and the active edges in between them. This model focuses on terminating protocols so it assumes  $Q_{out} \subseteq Q_{halt} \subseteq Q$ . All rules containing  $q_{halt} \in Q_{halt}$  is ineffective.

## 2.6 Vector transformation: handle coordinate rotations in 2-dimension

As mentioned in the previous section 2.5, Grid Terminating Network Constructor enforces perpendicular ports and geometrical restrictions. This means that handling rotation in some cases becomes necessary. This section will cover some basic but related vector mathematics. The detailed algorithms will be discussed later on, in the "Realisation" section (??).

### 2.6.1 2D centred coordinate rotation, with origin point as the centre

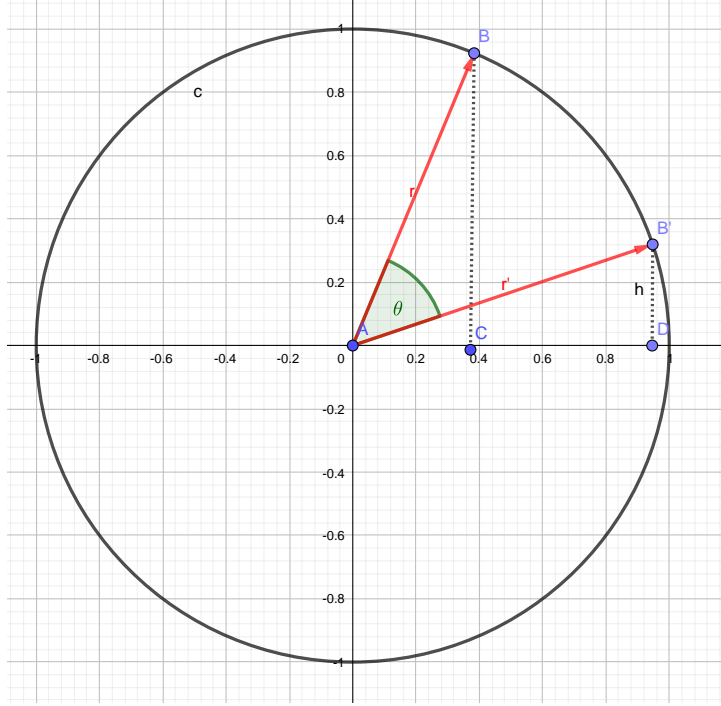


Figure 3: Vector  $\vec{r}$  and  $\vec{r}'$  in unit circle

The figure 3 has two same-length vectors,  $\vec{r}$  (or  $\vec{AB}$ ) and  $\vec{r}'$  (or  $\vec{AB'}$ ) in its Cartesian coordinate and their destination point located at the same unit circle. The point A is the origin point with coordinate (0,0). Suppose it has already known for the coordinate of B is  $(x, y)$ , and the  $\theta$  angle. The question is that what the coordinate of  $B'$  (representing in  $(x', y')$ ) is, after the first vector  $\vec{r}$  rotates the given angle  $\theta$  and becomes  $\vec{r}'$ . Note that, the "positive" rotation direction in current context and the implementation of the simulator is defined as "clock-wised".

To solve this, the following equation can be concluded:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos \theta & \sin \theta \\ -\sin \theta & \cos \theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} \quad (1)$$

A brief proof can be found in appendix partition below and it also can be found in [5] with a more precise and general proof.

### 2.6.2 2D centred coordinate rotation, with any points as the centre

Given the equation (1), it can be easily deduced that the rotation equation for any centred points  $c$  with coordinate  $(c_x, c_y)$  can be achieved through panning  $c$  to origin point with coordinate (0,0), carry out the rotation and panning back to its original coordinate  $(c_x, c_y)$ . Suppose the target point that is required to be rotated is called  $p$  (with coordinate  $(x, y)$ ), it then has :

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos \theta & \sin \theta \\ -\sin \theta & \cos \theta \end{bmatrix} \left( \begin{bmatrix} x \\ y \end{bmatrix} - \begin{bmatrix} c_x \\ c_y \end{bmatrix} \right) + \begin{bmatrix} c_x \\ c_y \end{bmatrix} \quad (2)$$



### 2.6.3 Affine transformation form

After a series of transformation, the equation 2 becomes the following form:

$$W \begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos \theta & \sin \theta \\ -\sin \theta & \cos \theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} c_x(1 - \cos \theta) - c_y \sin \theta \\ c_y(1 - \cos \theta) + c_x \sin \theta \end{bmatrix} \quad (3)$$

Let  $\mathbf{M} = \begin{bmatrix} \cos \theta & \sin \theta \\ -\sin \theta & \cos \theta \end{bmatrix}$  and  $\mathbf{B} = \begin{bmatrix} c_x(1 - \cos \theta) - c_y \sin \theta \\ c_y(1 - \cos \theta) + c_x \sin \theta \end{bmatrix}$ , then  $\begin{bmatrix} x' \\ y' \end{bmatrix} = \mathbf{M} \begin{bmatrix} x \\ y \end{bmatrix} + \mathbf{B}$ , where it forms an *affine transformation* [6].  $\mathbf{M}$  and  $\mathbf{B}$  can be calculated directly after knowing the rotation centre  $(c_x, c_y)$  and the rotation angle  $\theta$ , and therefore the calculated  $\mathbf{M}$  and  $\mathbf{B}$  can be applied for multiple target coordinates (those coordinates required to be rotated). This simplifies the calculation for a large batch of centred rotation with multiple target coordinates.

## 3 Data Required

## 4 Design

### 4.1 Anticipated components of the system

The main partition of the system was a software that enables to simulate and visualise each interaction before convergence (or terminate). The project would also attempt to explore one or more protocol to check their internal status after finishing the software.

The simulator would allow users to choose from a set of well-defined protocol and specify the parameters for these protocol (e.g. number of agents participating). The simulator would allow users to easily extend their own protocol though writing structured few lines of code, and the extension should be as easy as possible.

The system expected to have a ability to simulate some variations (which at least includes the network constructor [2] and terminating grid network constructor in 2D [3]) of the original protocol according the protocol and dynamically visualise how process (nodes) transfer for each interaction.

The interface provided would be a desktop application running on Java Virtual Machine Platform. A web interface was desirable but would not be guaranteed to deliver.

### Modification from original design

- The original design mentioned it could include "a configuration parser to enable user define a protocol from configuration files" finally replaced by a set of well-defined set of protocol and a programmable interface. This is because the original design requires a design and implementation of Domain Specific Language (DSL), and a parser of this kind of "new language", which is a large work load and might be infeasible to finish all of them during the project period. This might be done in the following development plan after finishing the project itself.

## 4.2 Data structure, algorithms and Pseudo-code for key method

### 4.2.1 Adjacency List

For some specific kinds of models, specifically ,the network constructor, a data structure to maintain a undirected graph  $G$  formed from current configuration  $C$  is required. In other words, it requires to maintain a group of status of "active" edges. It aims to construct "shapes" and there are some status for the edge  $uv$  in between any nodes (processors)  $u$  and  $v$  in the population.

To make it, an adjacency list would be used to record the connections associated with each

Modification from original design

### 4.2.2 Graph

Modification from original design

### 4.2.3 Terminating grid network constructor: Check acceptable configuration & Handle rotation

Modification from original design

## 4.3 Interaction (Sequence) Design

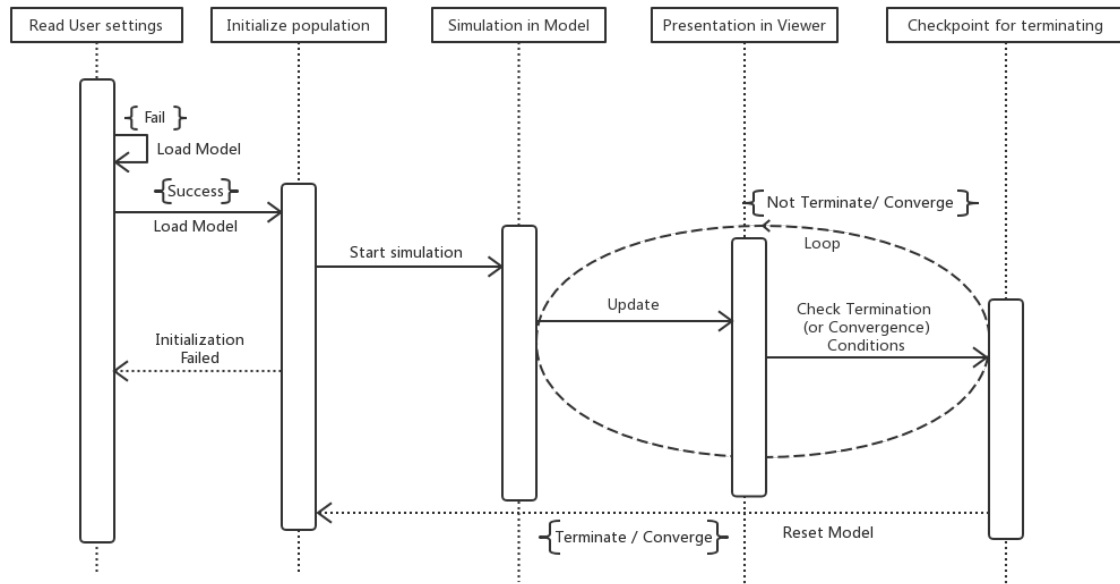


Figure 4: Interaction Design Diagram

The figure 4 illustrates the entire interaction process of the simulator. Initially, the user select one protocol from a set of well-defined protocols and then select a set of parameters required such as how many nodes to be included in the simulation, the initial status distribution for nodes and so forth. Then the user click the "Apply settings button" to apply the model settings. before applying the model, the program itself would check whether all parameters are well-defined, i.e. locating in the value range they should be. If they are, the software would load the model otherwise it will reject this particular request from user and allow user to repeat its (possibly) another setting.

After the program successfully loaded a model, the user can press the "Start simulation" button to start a simulation. A simulation contains multiple rounds. For each round, the model executes interaction simulation under the control of scheduler specified in one of parameters mentioned before and then the model would update the states of elements in viewer to show the what happened in the model itself. Following that, the model would check whether it reaches a configuration that should stop the simulation. A stop can be caused by user's temporarily interruption (pause) or the fact that the number of consistent accumulation of inefficient interactions overwhelms a pre-configured parameters defined in the model (which indicates a terminating or convergence with a high possibility). If the model does not detect a stop, it will continue the "interact - update - checking stop" process as a loop. Once it detect a stop, the software will return to state of waiting for starting of a simulation process (while the model would keep its internal states and user can build an another new model under this situation).

**Modification from original design** The current design removed a step of sequence in original sequence design called "Initialize scheduler", which located after "Initialize population" step. the "Initialize scheduler" process had been integrated into "Initialize population" because the scheduler partition does not necessarily separated from the population and can be treated as a partition of a "population" in concept.

## 4.4 UI Interface Design

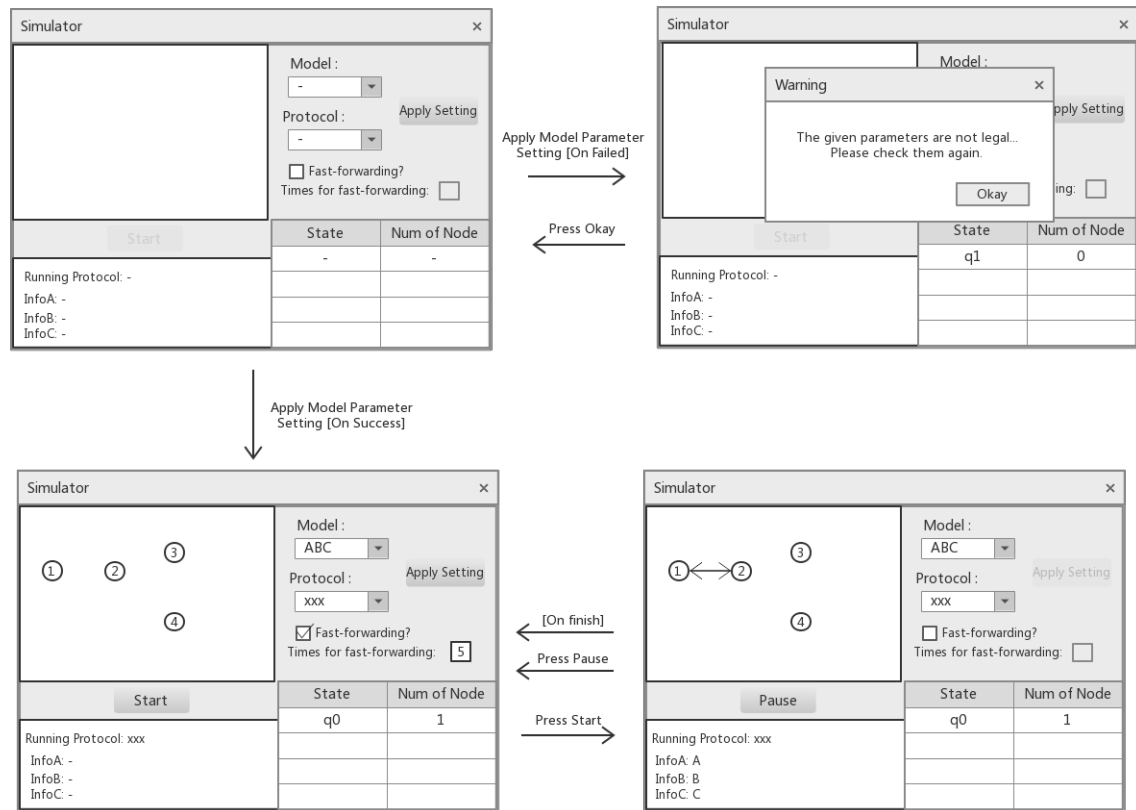


Figure 5: Interaction Design Diagram

**Modification from original design** The UI was redesigned entirely. This is because the original design built on the assumption that user loads a setting for some population from domain specification language codes in some files, which is found infeasible to be finished due to the limited time of implementation. Hence, the design would allow user to set their population through the UI containing a set of well-defined protocols and allow the users to write their own extensional protocol through few lines of codes. As the functionality changes, the UI removed the functionality entrance for loading files from disk and added some options to allow user finish such a functionality.

## Bibliography

- [1] J. Aspnes and E. Ruppert, “An introduction to population protocols,” *Bulletin of the European Association for Theoretical Computer Science*, vol. 93, pp. 98–117, Oct. 2007.
- [2] O. Michail and P. G. Spirakis, “Simple and efficient local codes for distributed stable network construction,” *Distributed Computing*, vol. 29, no. 3, pp. 207–237, 2016. [Online]. Available: <https://link.springer.com/article/10.1007/s00446-015-0257-4>
- [3] O. Michail, “Terminating distributed construction of shapes and patterns in a fair solution of automata,” *Distributed Computing*, pp. 1–23, 2017. [Online]. Available: <http://link.springer.com/article/10.1007/s00446-017-0309-z>
- [4] O. Michail, I. Chatzigiannakis, and P. G. Spirakis, *New Models for Population Protocols*, ser. N. A. Lynch (Ed), Synthesis Lectures on Distributed Computing Theory, N. A. Lynch, Ed. Morgan & Claypool, 2011.
- [5] S. Widnall, “Lecture 13 - vectors, matrices and coordinate transformations,” Fall 2009. [Online]. Available: [https://ocw.mit.edu/courses/aeronautics-and-astronautics/16-07-dynamics-fall-2009/lecture-notes/MIT16\\_07F09\\_Lec03.pdf](https://ocw.mit.edu/courses/aeronautics-and-astronautics/16-07-dynamics-fall-2009/lecture-notes/MIT16_07F09_Lec03.pdf) Accessed: 2018-04-30.
- [6] E. Weisstein, “Affine transformation.” [Online]. Available: <http://mathworld.wolfram.com/AffineTransformation.html> Accessed: 2018-05-01.

## A Brief Proof for equation (1)

For convenience, here name  $\angle CAB$  as  $\alpha$  in figure 3. Then it has

$$|AD| = |\vec{r}'| \cos(\alpha - \theta)$$

$$|B'D| = |\vec{r}'| \sin(\alpha - \theta)$$

By applying angle sum identities,

$$|AD| = |\vec{r}'|(\cos(\alpha) \cos(-\theta) - \sin(\alpha) \sin(-\theta))$$

$$|B'D| = |\vec{r}'|(\sin(\alpha) \cos(-\theta) + \sin(-\theta) \cos(\alpha))$$

Because  $|\vec{r}'| = |\vec{r}|$ ,  $\sin(\theta) = -\sin(-\theta)$  for  $\theta \in [0, \pi/2)$  and  $\cos(\theta) = \cos(-\theta)$  for  $\theta \in [0, \pi/2)$ ,

$$|AD| = |\vec{r}|(\cos(\alpha) \cos(\theta) + \sin(\alpha) \sin(\theta))$$

$$|B'D| = |\vec{r}|(\sin(\alpha) \cos(\theta) - \sin(\theta) \cos(\alpha))$$

Then,

$$|AD| = (|\vec{r}| \cos(\alpha)) \cos(\theta) + (|\vec{r}| \sin(\alpha)) \sin(\theta)$$

$$|B'D| = (|\vec{r}| \sin(\alpha)) \cos(\theta) - (|\vec{r}| \cos(\alpha)) \sin(\theta)$$

Hence,

$$|AD| = |AC| \cos(\theta) + |BC| \sin(\theta)$$

$$|B'D| = |BC| \cos(\theta) - |AC| \sin(\theta)$$

Finally,

$$|x'| = |x| \cos(\theta) + |y| \sin(\theta)$$

$$|y'| = |x|(-\sin(\theta)) + |y| \cos(\theta)$$

For  $\theta \in [0, \pi/2)$ , the equations can be written simply as

$$x' = x \cos \theta + y \sin \theta$$

$$y' = x(-\sin \theta) + y \cos \theta$$

which can be easily extended to  $\theta \in [0, 2\pi + 2k\pi)$  via discussion under different cases, given that  $k \in \text{Integer}$ .

The two equations can be written in matrix form :

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos \theta & \sin \theta \\ -\sin \theta & \cos \theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

## B Brief Proof for equation (3)

The equation 2 can be transformed to:

$$\begin{aligned}\begin{bmatrix} x' \\ y' \end{bmatrix} &= \begin{bmatrix} \cos \theta & \sin \theta \\ -\sin \theta & \cos \theta \end{bmatrix} \left( \begin{bmatrix} x \\ y \end{bmatrix} - \begin{bmatrix} c_x \\ c_y \end{bmatrix} \right) + \begin{bmatrix} c_x \\ c_y \end{bmatrix} \\ &= \begin{bmatrix} \cos \theta & \sin \theta \\ -\sin \theta & \cos \theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} - \begin{bmatrix} \cos \theta & \sin \theta \\ -\sin \theta & \cos \theta \end{bmatrix} \begin{bmatrix} c_x \\ c_y \end{bmatrix} + \begin{bmatrix} c_x \\ c_y \end{bmatrix} \\ &= \begin{bmatrix} \cos \theta & \sin \theta \\ -\sin \theta & \cos \theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} - \begin{bmatrix} c_x \cos \theta + c_y \sin \theta \\ -c_x \sin \theta + c_y \cos \theta \end{bmatrix} + \begin{bmatrix} c_x \\ c_y \end{bmatrix} \\ &= \begin{bmatrix} \cos \theta & \sin \theta \\ -\sin \theta & \cos \theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} c_x(1 - \cos \theta) - c_y \sin \theta \\ c_y(1 - \cos \theta) + c_x \sin \theta \end{bmatrix}\end{aligned}$$