Playing With Population Protocols

Olivier Bournez[1] Jérémie Chalopin[2,5] Johanne Cohen[3,5]
Xavier Koegler[4]

[1]Ecole Polytechnique, France

[2]University of Marseille, France

[3]University of Versailles, France

[4]Paris VII University, France

[5]Centre National de la Recherche Scientifique

Réunion SHAMAN.
Tuesday January 27th 2009.

Plan

Population Protocols

Variants

Population Protocols and Games

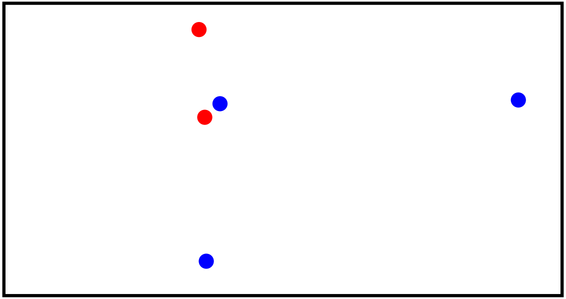
Population protocols

- Introduced by [Angluin,Aspnes,Diamadi,Fischer,Peralta 2004] in the context of distributed systems.

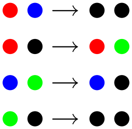
- A model of sensor networks, with absolutely minimal assumptions about

  - sophistication of mobile units :
    - finite state machines.
  - infrastructure : none
    - no topology,
    - not even unique ids.
  - synchrony :
    - totally asynchronous.
  - communications :
    - communications are occasionally possible between pairs of agents.

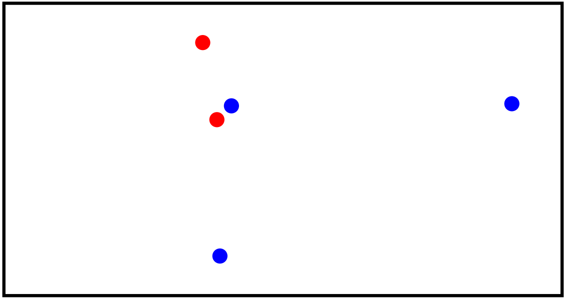
Example 1 : starting with 3 blues, 2red



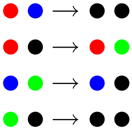
Program :



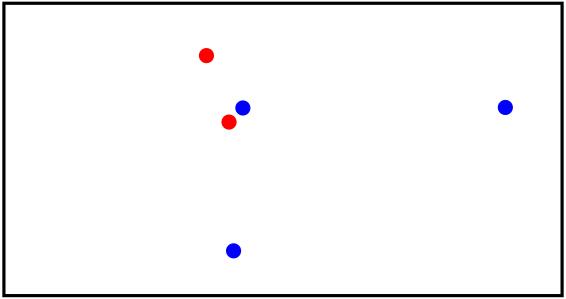
Example 1 : starting with 3 blues, 2red



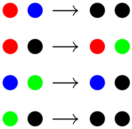
Program :



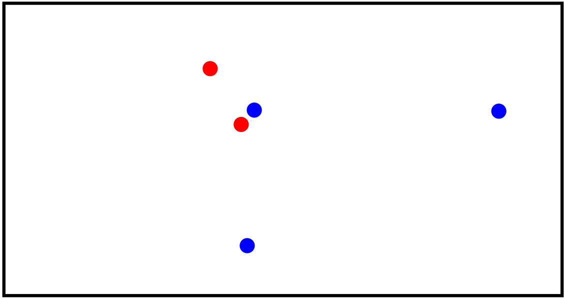
Example 1 : starting with 3 blues, 2red



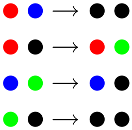
Program :



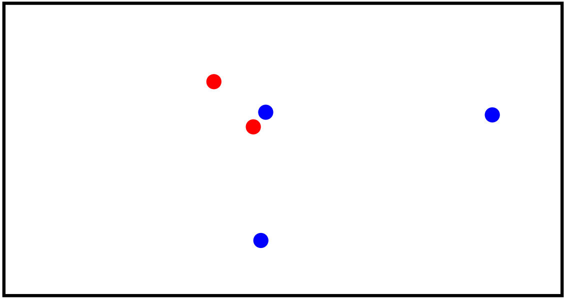
Example 1 : starting with 3 blues, 2red



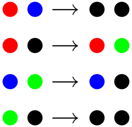
Program :



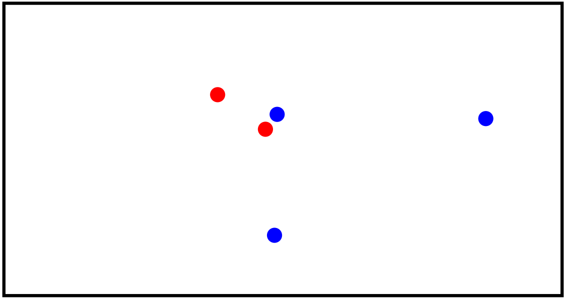
Example 1 : starting with 3 blues, 2red



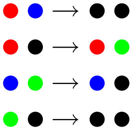
Program :



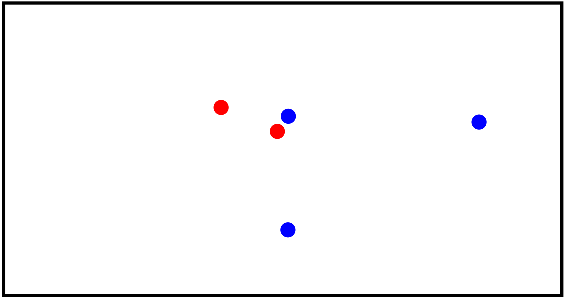
Example 1 : starting with 3 blues, 2red



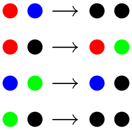
Program :



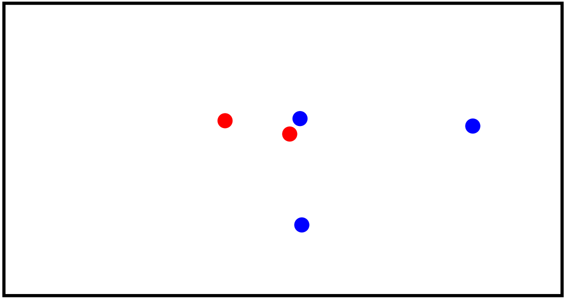
Example 1 : starting with 3 blues, 2red



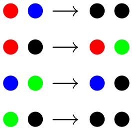
Program :



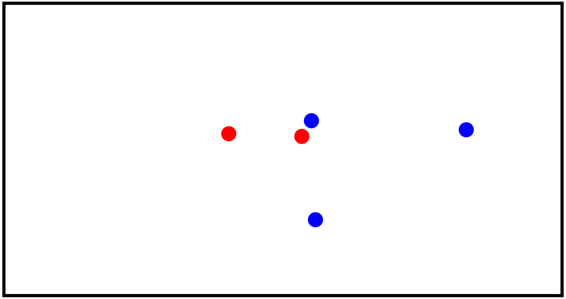
Example 1 : starting with 3 blues, 2red



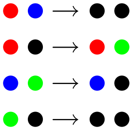
Program :



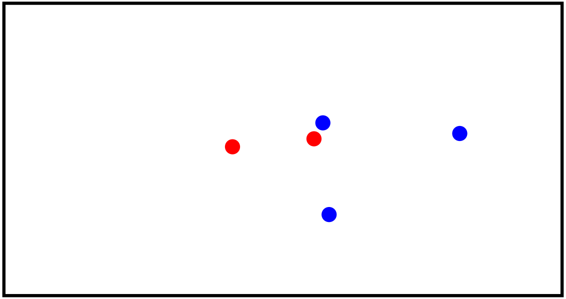
Example 1 : starting with 3 blues, 2red



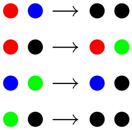
Program :



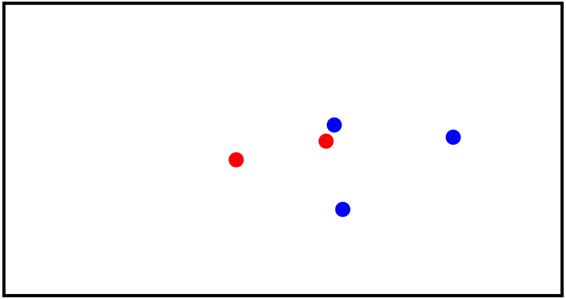
Example 1 : starting with 3 blues, 2red



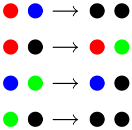
Program :



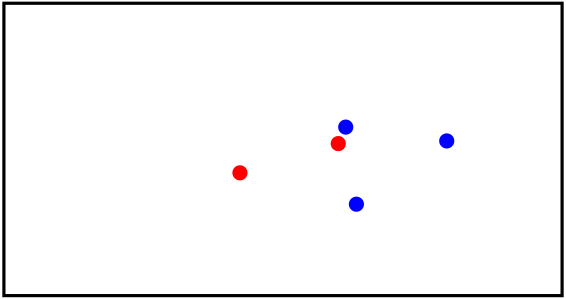
Example 1 : starting with 3 blues, 2red



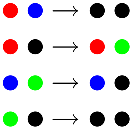
Program :



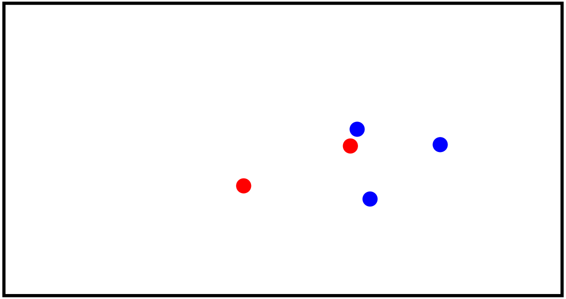
Example 1 : starting with 3 blues, 2red



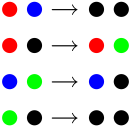
Program :



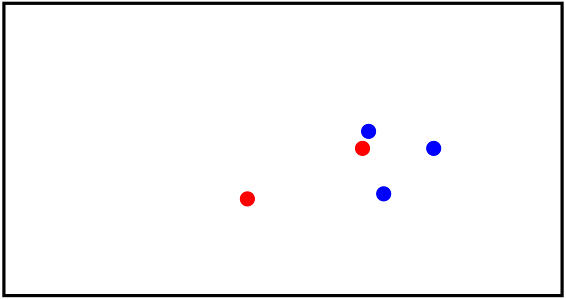
Example 1 : starting with 3 blues, 2red



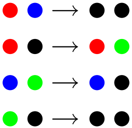
Program :



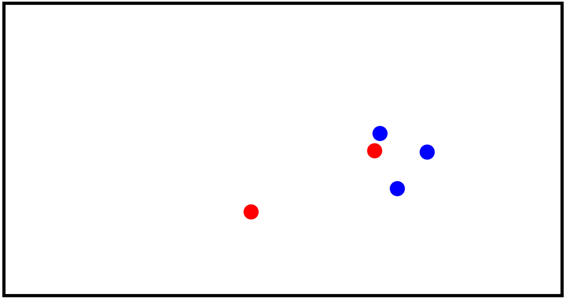
Example 1 : starting with 3 blues, 2red



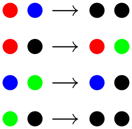
Program :



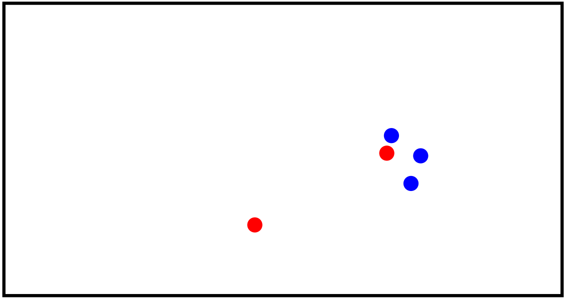
Example 1 : starting with 3 blues, 2red



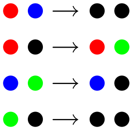
Program :



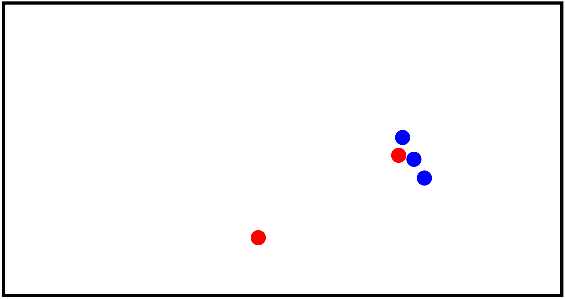
Example 1 : starting with 3 blues, 2red



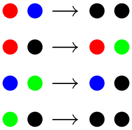
Program :



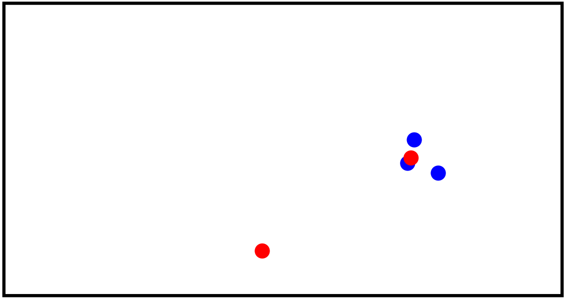
Example 1 : starting with 3 blues, 2red



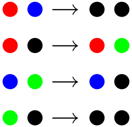
Program :



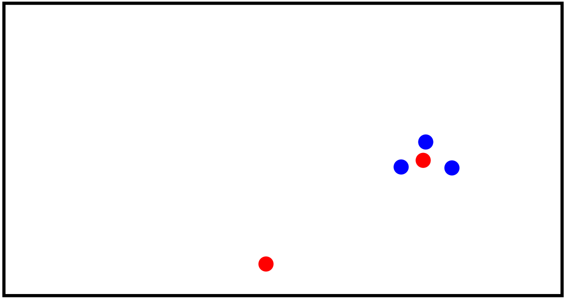
Example 1 : starting with 3 blues, 2red



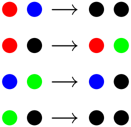
Program :



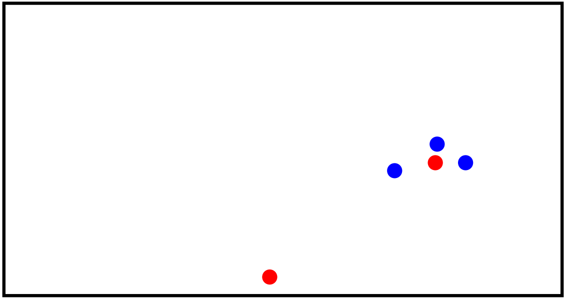
Example 1 : starting with 3 blues, 2red



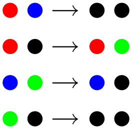
Program :



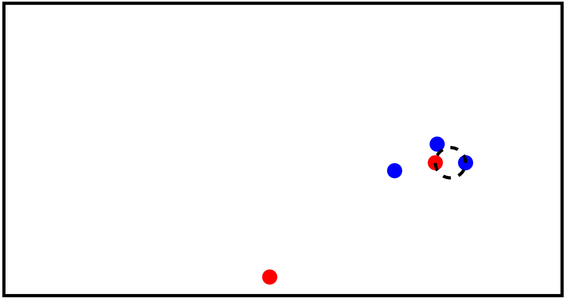
Example 1 : starting with 3 blues, 2red



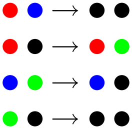
Program :



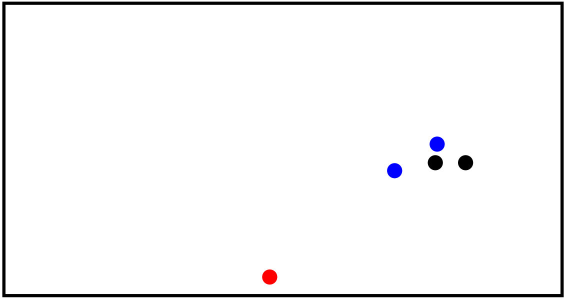
Example 1 : starting with 3 blues, 2red



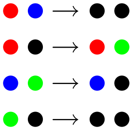
Program :



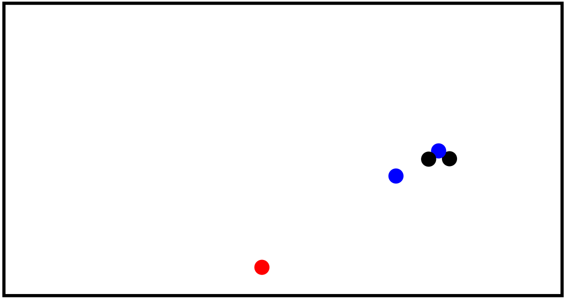
Example 1 : starting with 3 blues, 2red



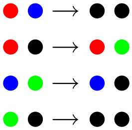
Program :



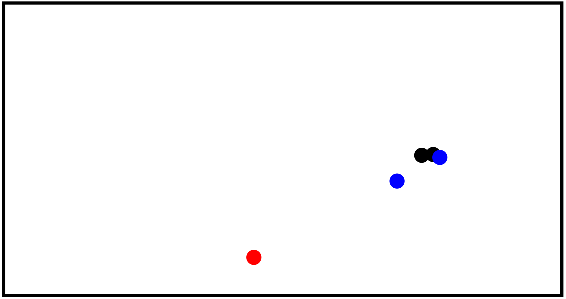
Example 1 : starting with 3 blues, 2red



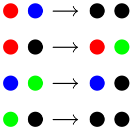
Program :



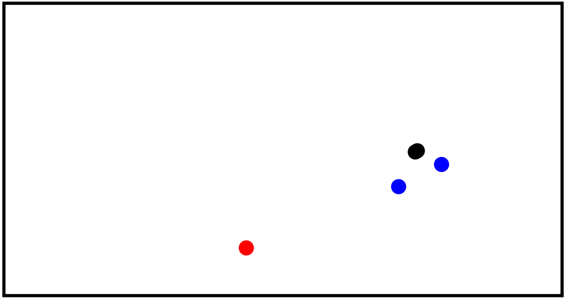
Example 1 : starting with 3 blues, 2red



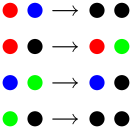
Program :



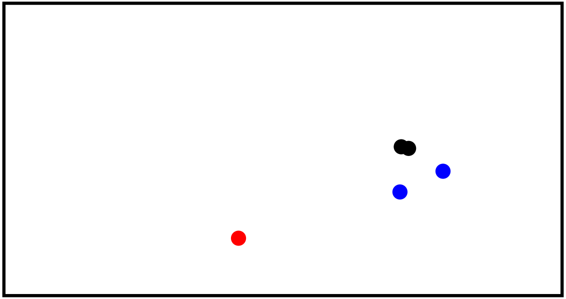
Example 1 : starting with 3 blues, 2red



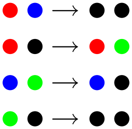
Program :



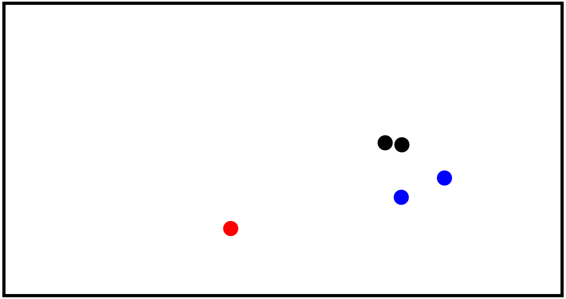
Example 1 : starting with 3 blues, 2red



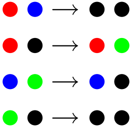
Program :



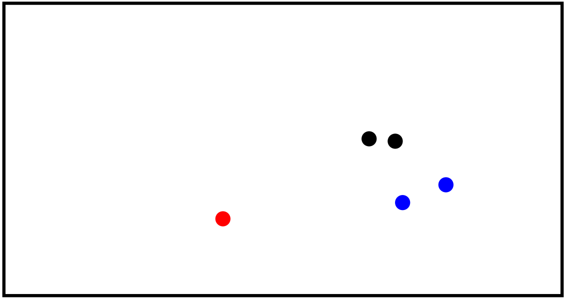
Example 1 : starting with 3 blues, 2red



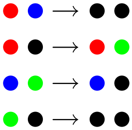
Program :



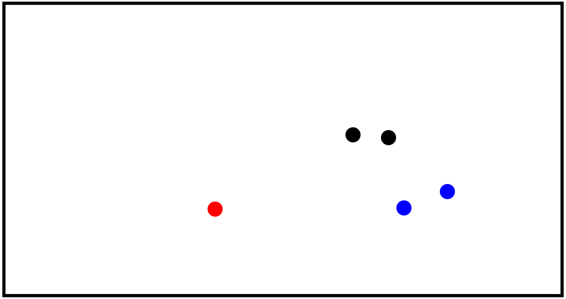
Example 1 : starting with 3 blues, 2red



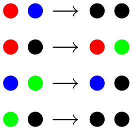
Program :



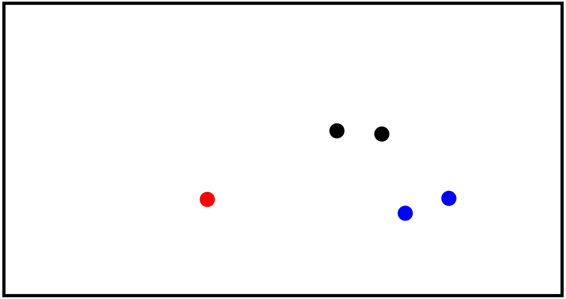
Example 1 : starting with 3 blues, 2red



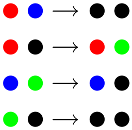
Program :



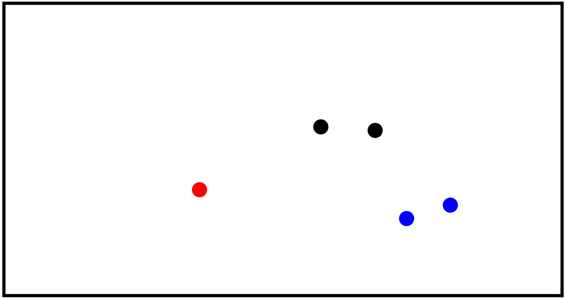
Example 1 : starting with 3 blues, 2red



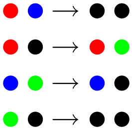
Program :



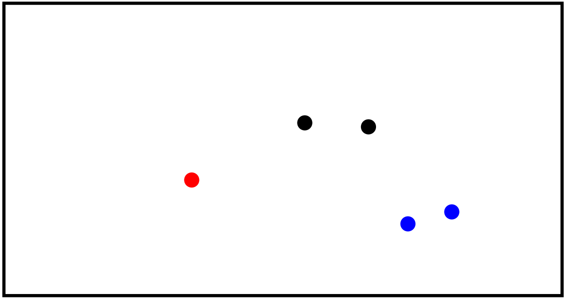
Example 1 : starting with 3 blues, 2red



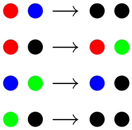
Program :



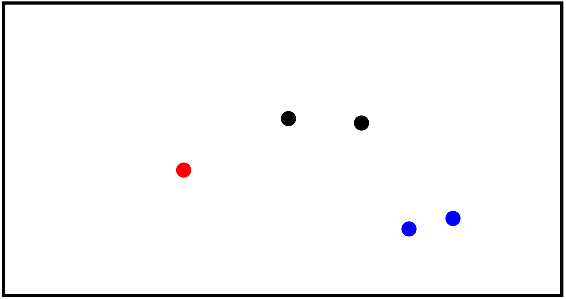
Example 1 : starting with 3 blues, 2red



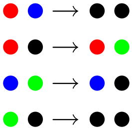
Program :



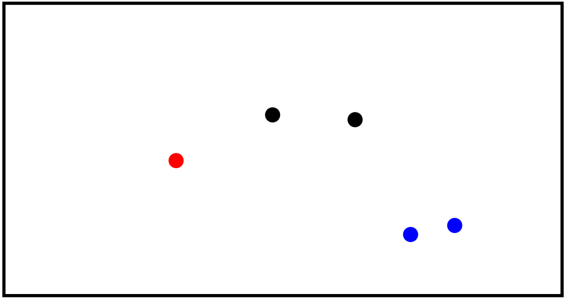
Example 1 : starting with 3 blues, 2red



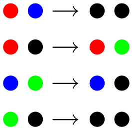
Program :



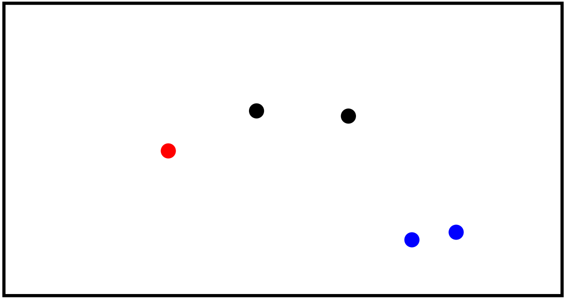
Example 1 : starting with 3 blues, 2red



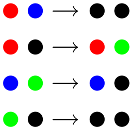
Program :



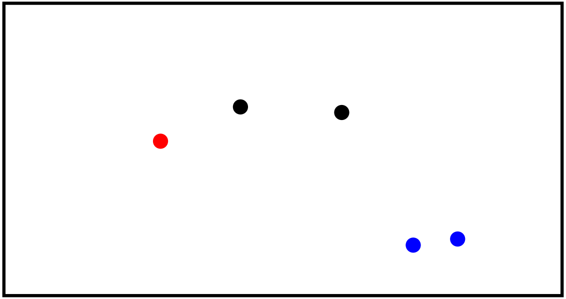
Example 1 : starting with 3 blues, 2red



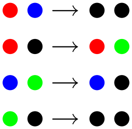
Program :



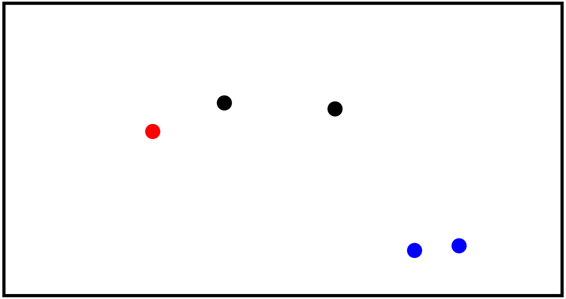
Example 1 : starting with 3 blues, 2red



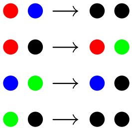
Program :



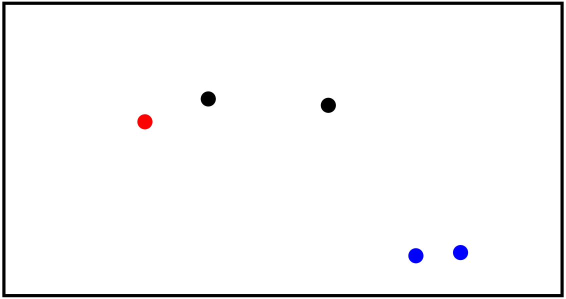
Example 1 : starting with 3 blues, 2red



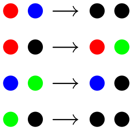
Program :



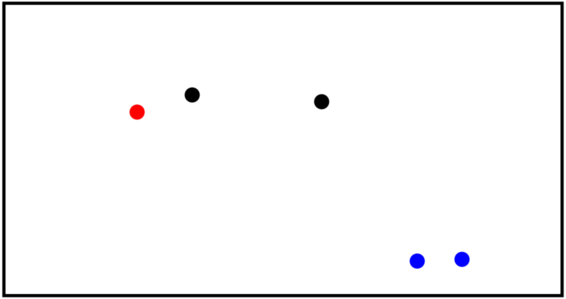
Example 1 : starting with 3 blues, 2red



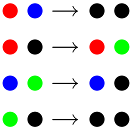
Program :



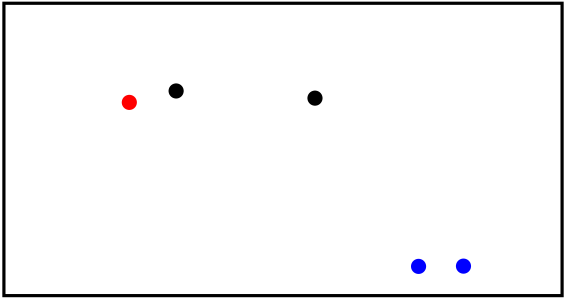
Example 1 : starting with 3 blues, 2red



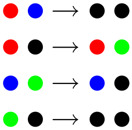
Program :



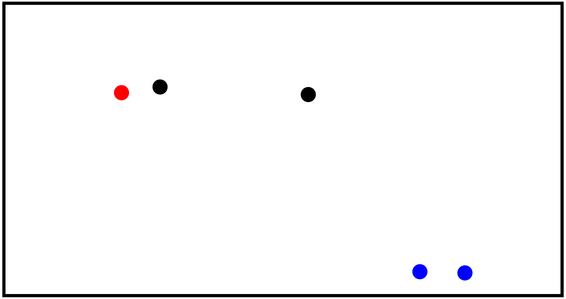
Example 1 : starting with 3 blues, 2red



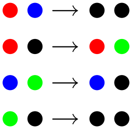
Program :



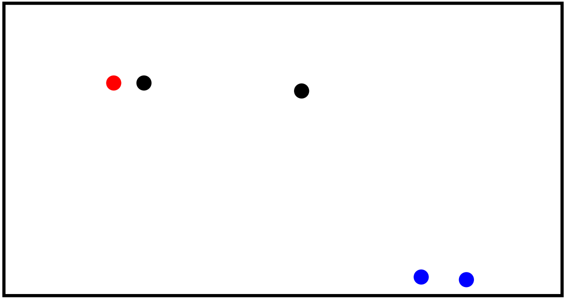
Example 1 : starting with 3 blues, 2red



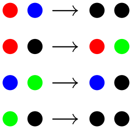
Program :



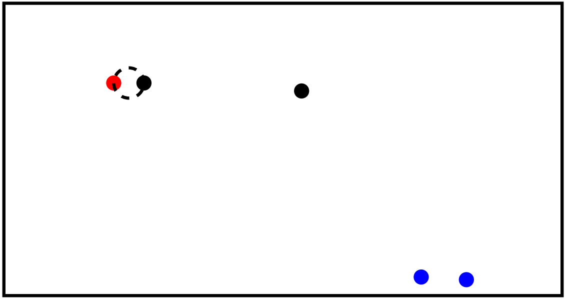
Example 1 : starting with 3 blues, 2red



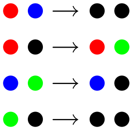
Program :



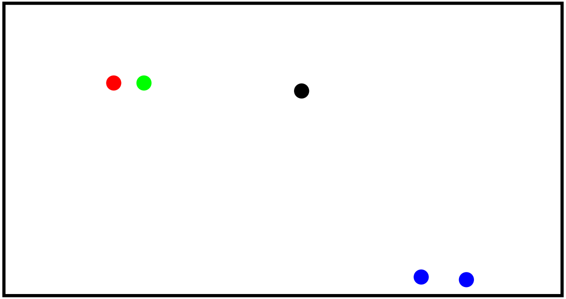
Example 1 : starting with 3 blues, 2red



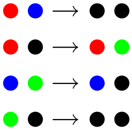
Program :



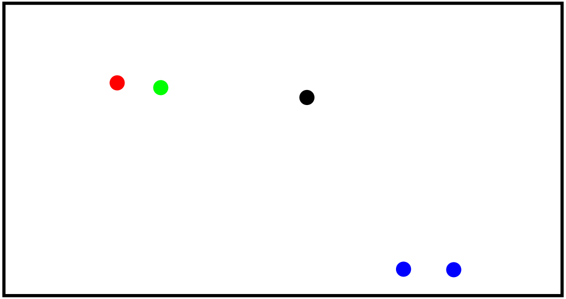
Example 1 : starting with 3 blues, 2red



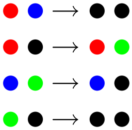
Program :



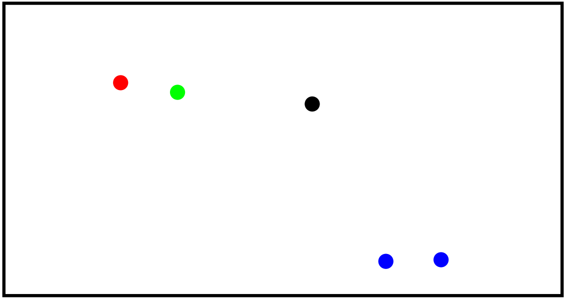
Example 1 : starting with 3 blues, 2red



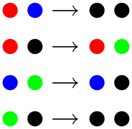
Program :



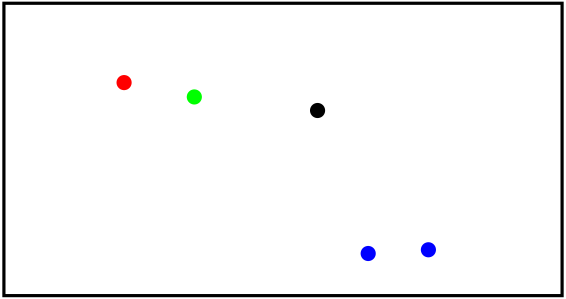
Example 1 : starting with 3 blues, 2red



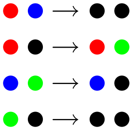
Program :



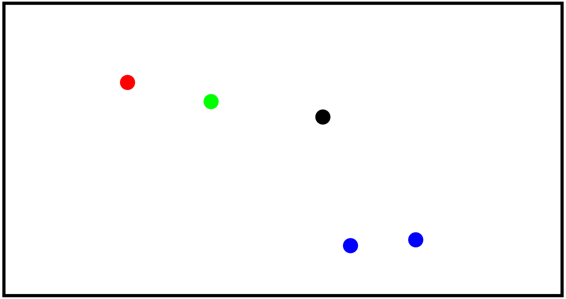
Example 1 : starting with 3 blues, 2red



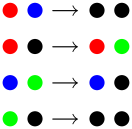
Program :



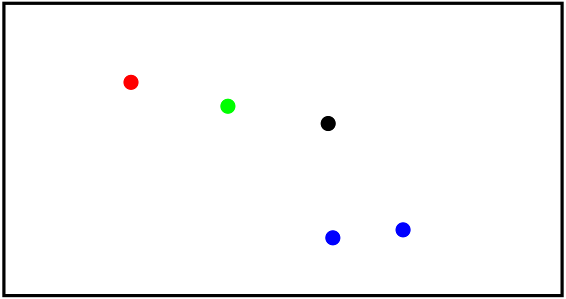
Example 1 : starting with 3 blues, 2red



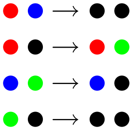
Program :



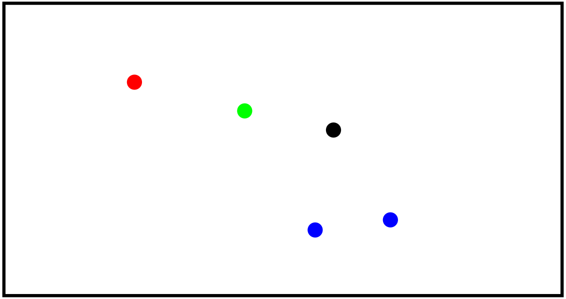
Example 1 : starting with 3 blues, 2red



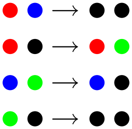
Program :



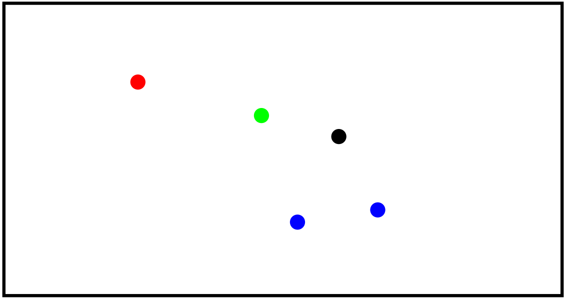
Example 1 : starting with 3 blues, 2red



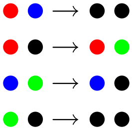
Program :



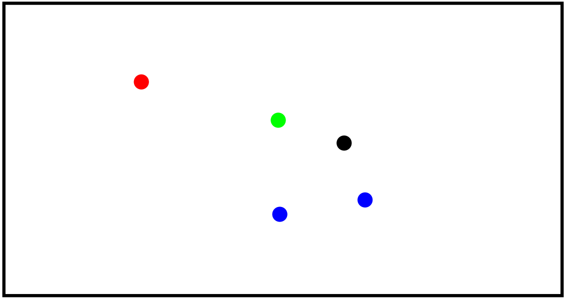
Example 1 : starting with 3 blues, 2red



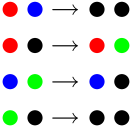
Program :



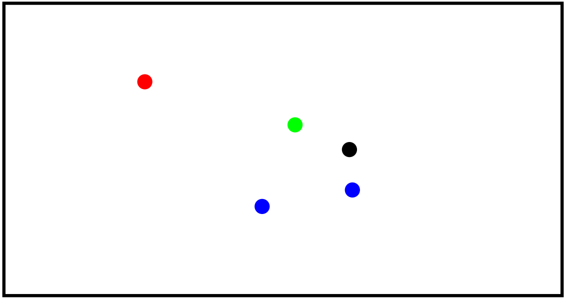
Example 1 : starting with 3 blues, 2red



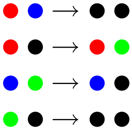
Program :



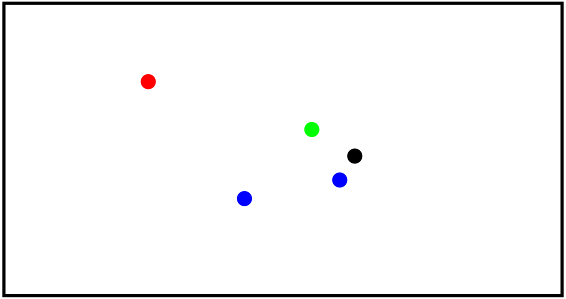
Example 1 : starting with 3 blues, 2red



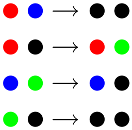
Program :



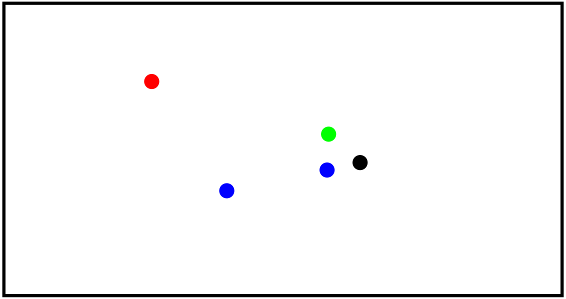
Example 1 : starting with 3 blues, 2red



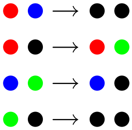
Program :



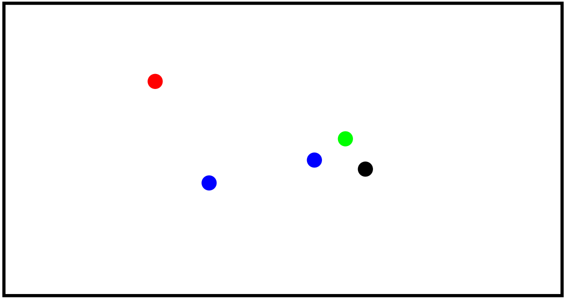
Example 1 : starting with 3 blues, 2red



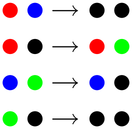
Program :



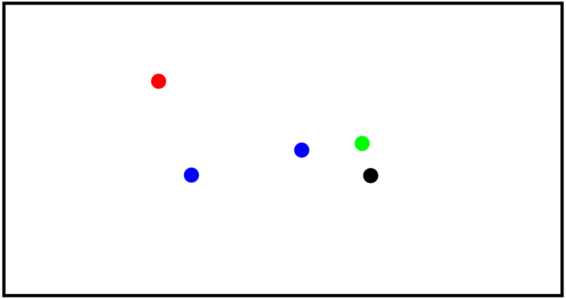
Example 1 : starting with 3 blues, 2red



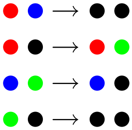
Program :



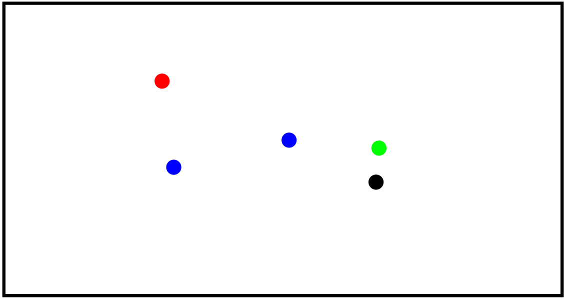
Example 1 : starting with 3 blues, 2red



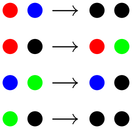
Program :



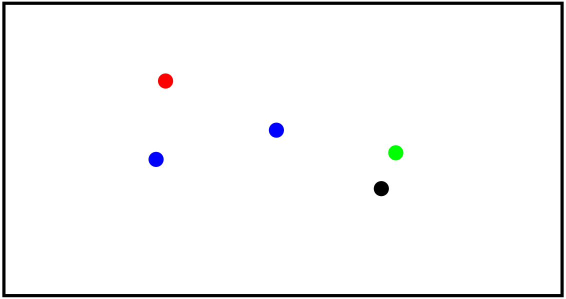
Example 1 : starting with 3 blues, 2red



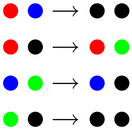
Program :



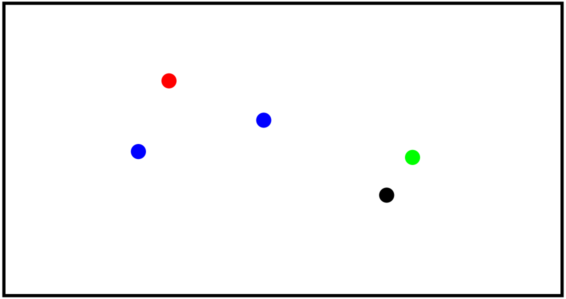
Example 1 : starting with 3 blues, 2red



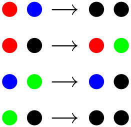
Program :



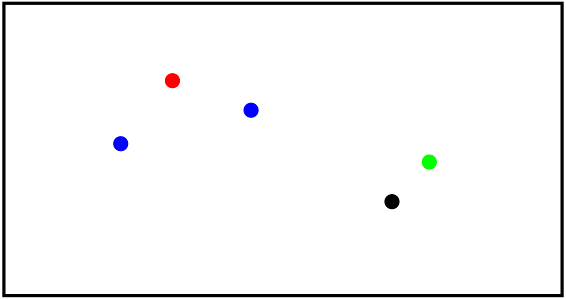
Example 1 : starting with 3 blues, 2red



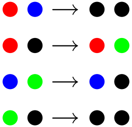
Program :



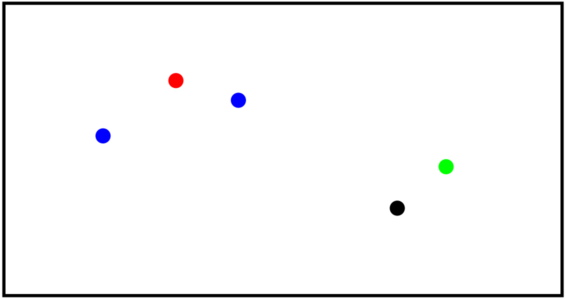
Example 1 : starting with 3 blues, 2red



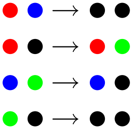
Program :



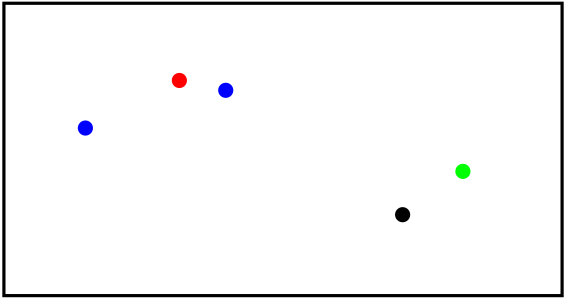
Example 1 : starting with 3 blues, 2red



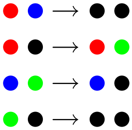
Program :



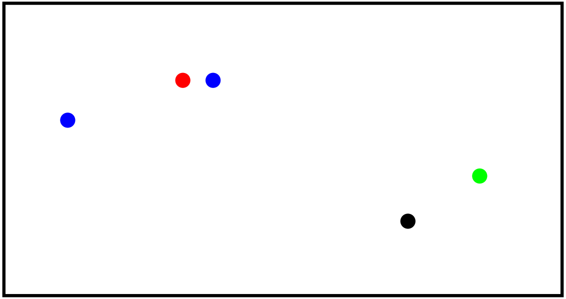
Example 1 : starting with 3 blues, 2red



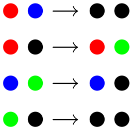
Program :



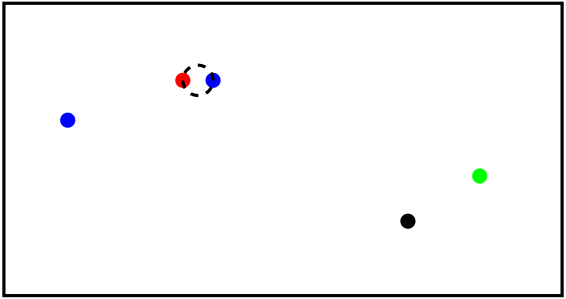
Example 1 : starting with 3 blues, 2red



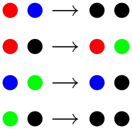
Program :



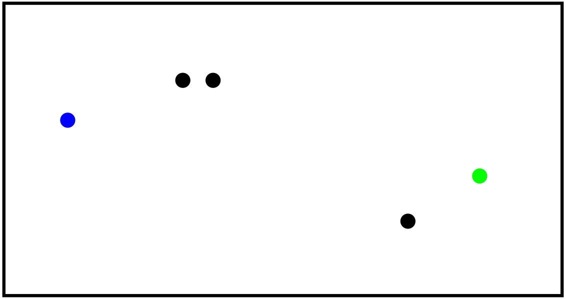
Example 1 : starting with 3 blues, 2red



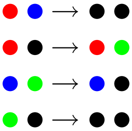
Program :



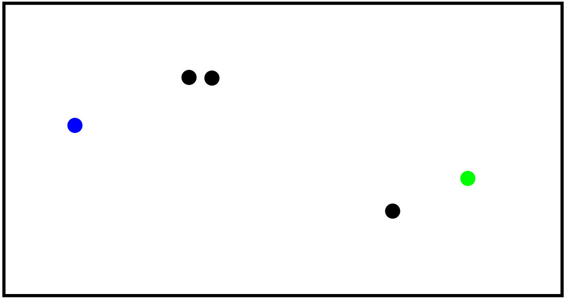
Example 1 : starting with 3 blues, 2red



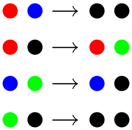
Program :



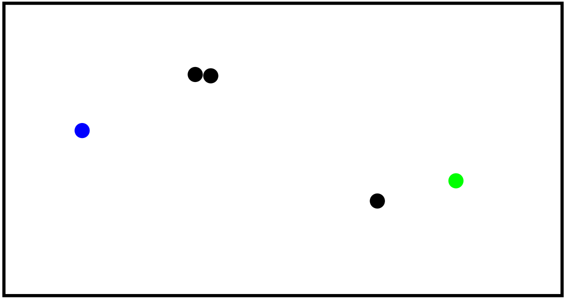
Example 1 : starting with 3 blues, 2red



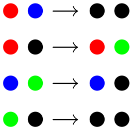
Program :



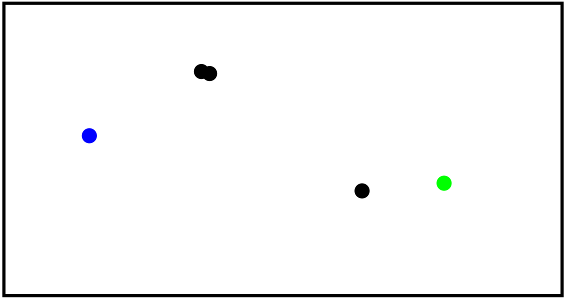
Example 1 : starting with 3 blues, 2red



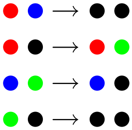
Program :



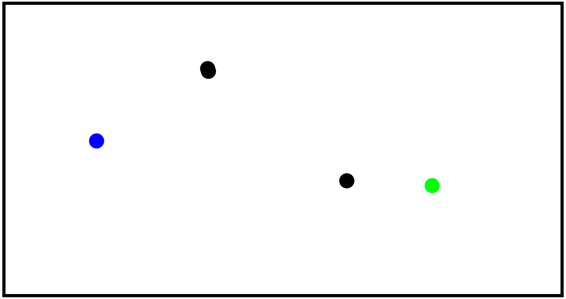
Example 1 : starting with 3 blues, 2red



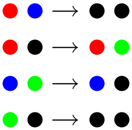
Program :



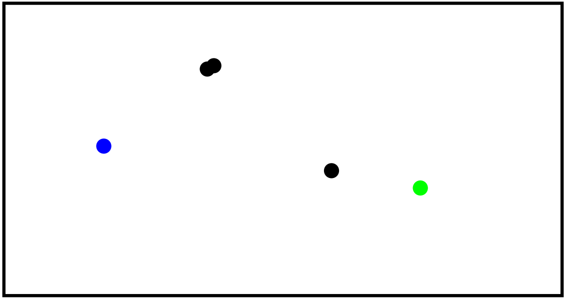
Example 1 : starting with 3 blues, 2red



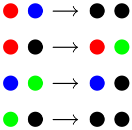
Program :



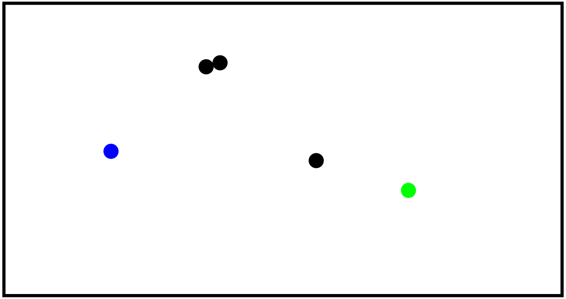
Example 1 : starting with 3 blues, 2red



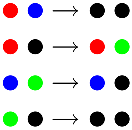
Program :



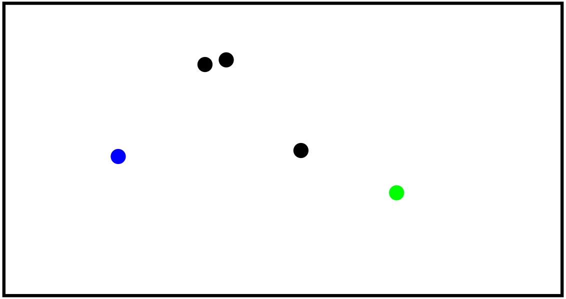
Example 1 : starting with 3 blues, 2red



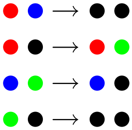
Program :



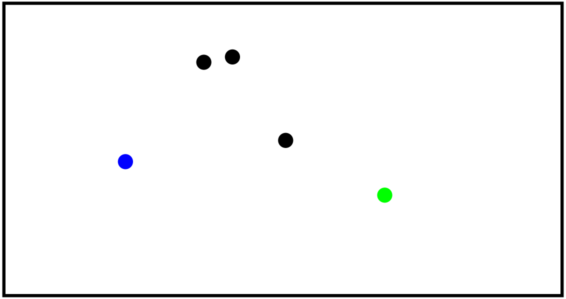
Example 1 : starting with 3 blues, 2red



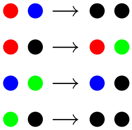
Program :



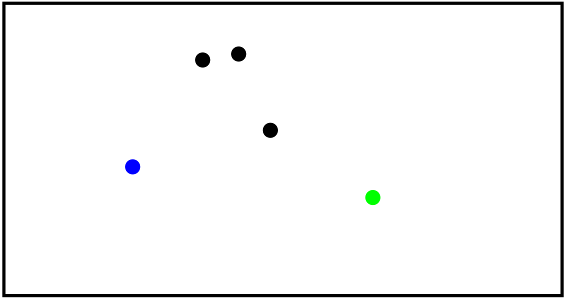
Example 1 : starting with 3 blues, 2red



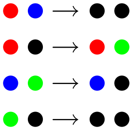
Program :



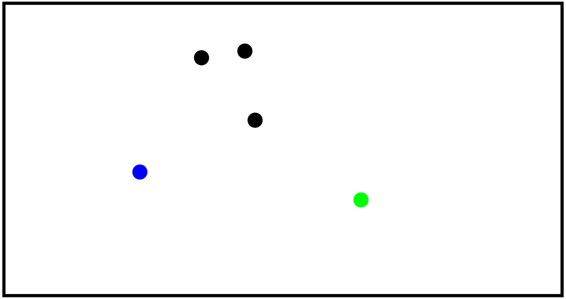
Example 1 : starting with 3 blues, 2red



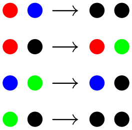
Program :



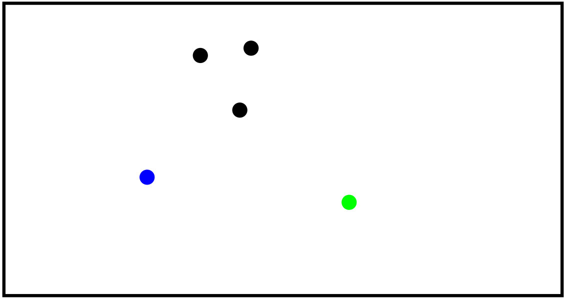
Example 1 : starting with 3 blues, 2red



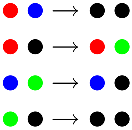
Program :



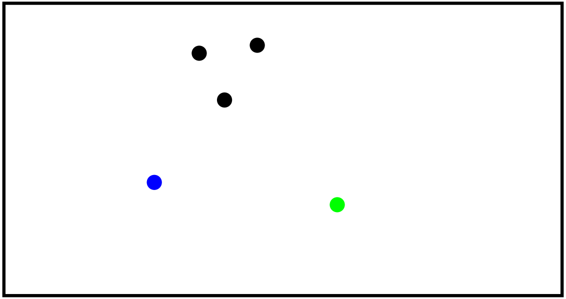
Example 1 : starting with 3 blues, 2red



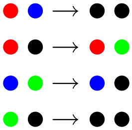
Program :



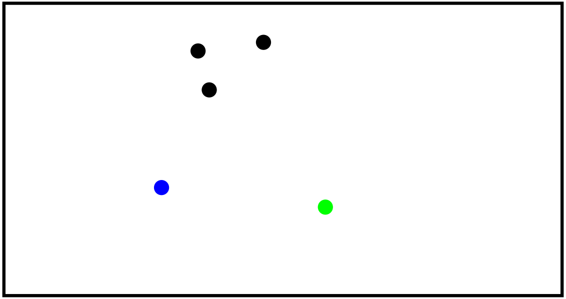
Example 1 : starting with 3 blues, 2red



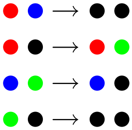
Program :



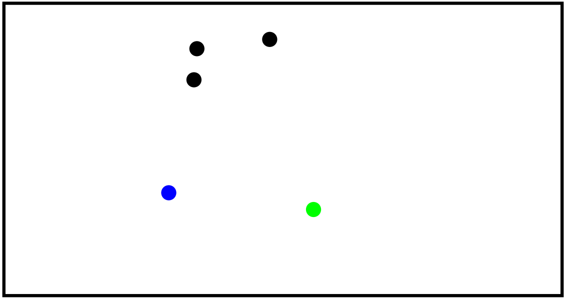
Example 1 : starting with 3 blues, 2red



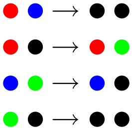
Program :



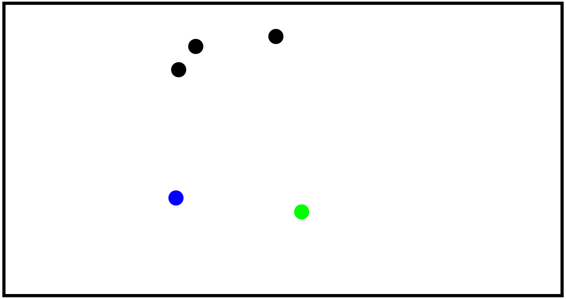
Example 1 : starting with 3 blues, 2red



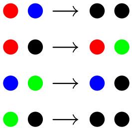
Program :



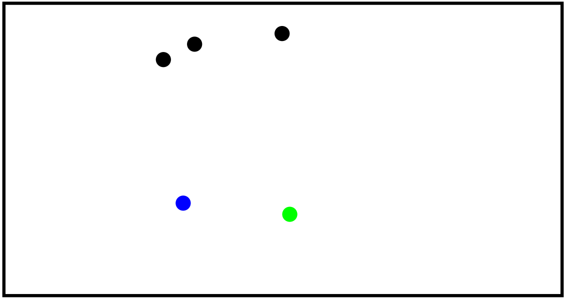
Example 1 : starting with 3 blues, 2red



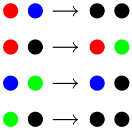
Program :



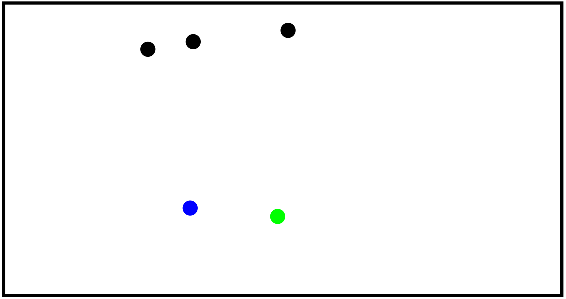
Example 1 : starting with 3 blues, 2red



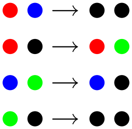
Program :



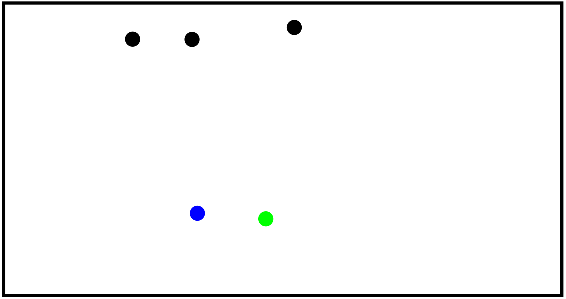
Example 1 : starting with 3 blues, 2red



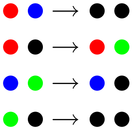
Program :



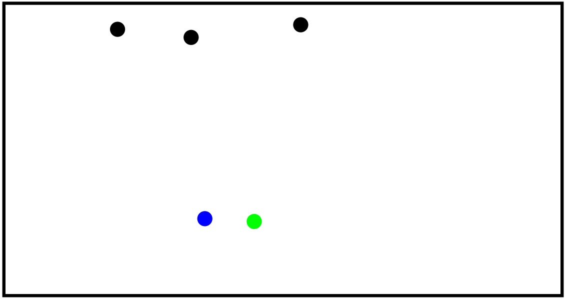
Example 1 : starting with 3 blues, 2red



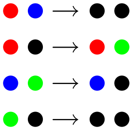
Program :



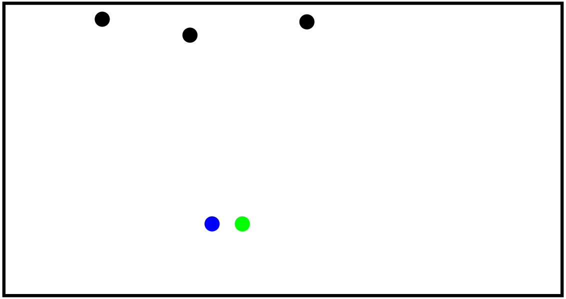
Example 1 : starting with 3 blues, 2red



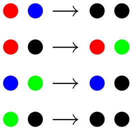
Program :



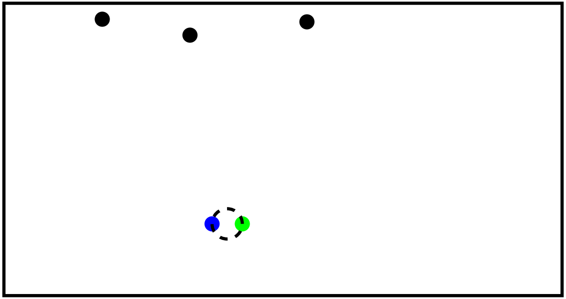
Example 1 : starting with 3 blues, 2red



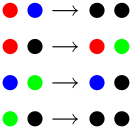
Program :



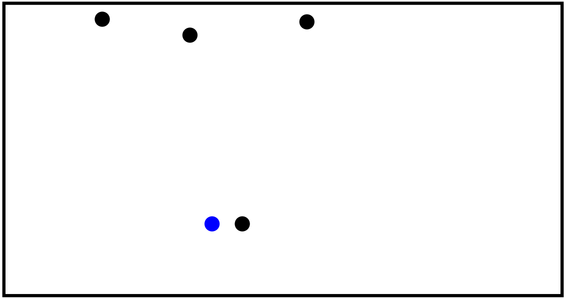
Example 1 : starting with 3 blues, 2red



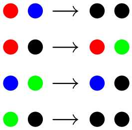
Program :



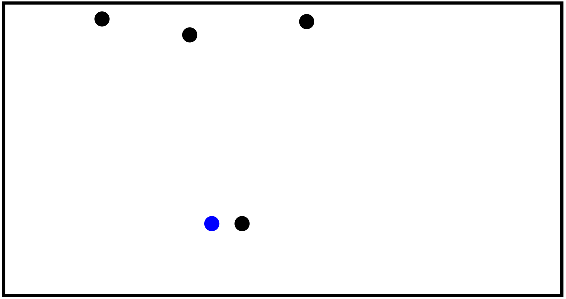
Example 1 : starting with 3 blues, 2red



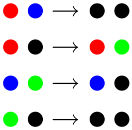
Program :



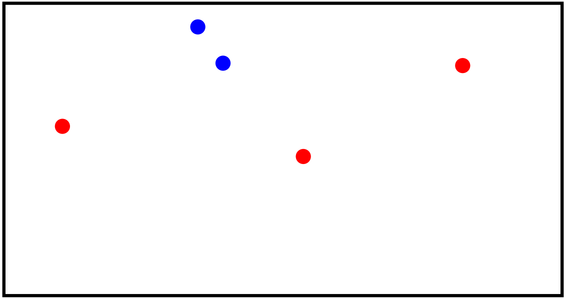
Example 1 : Final result



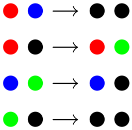
Program :



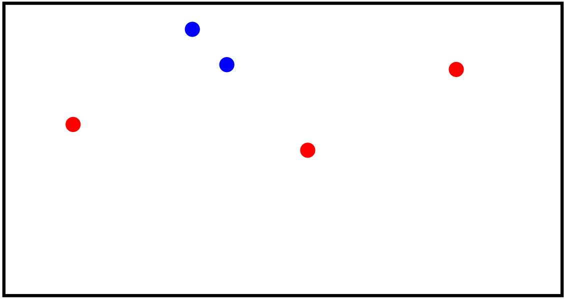
Example 2 : starting with 2 blues, 3red



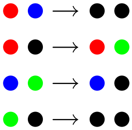
Program :



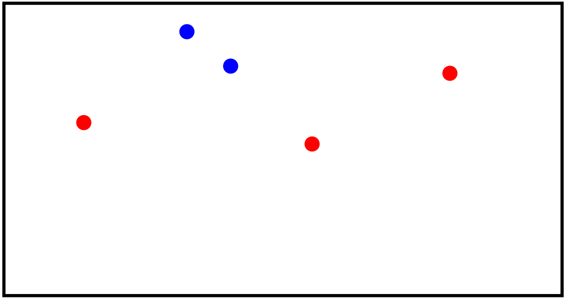
Example 2 : starting with 2 blues, 3red



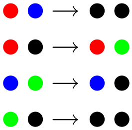
Program :



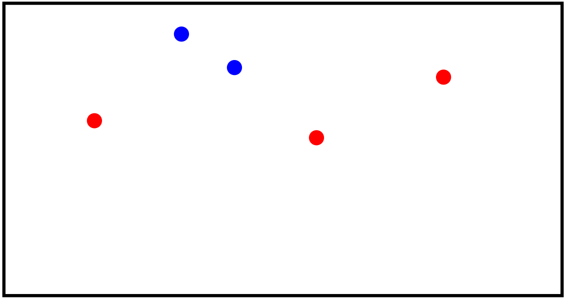
Example 2 : starting with 2 blues, 3red



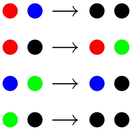
Program :



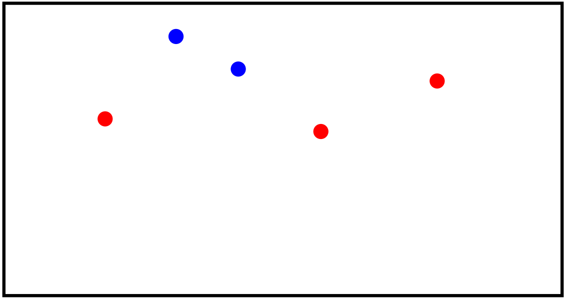
Example 2 : starting with 2 blues, 3red



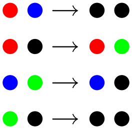
Program :



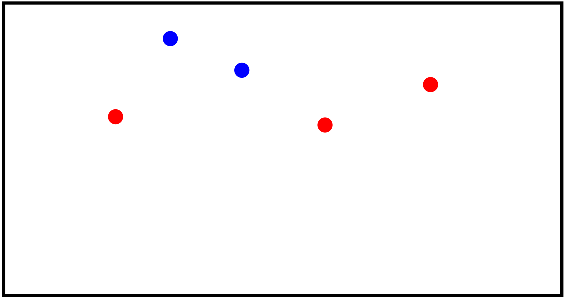
Example 2 : starting with 2 blues, 3red



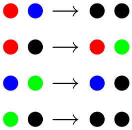
Program :



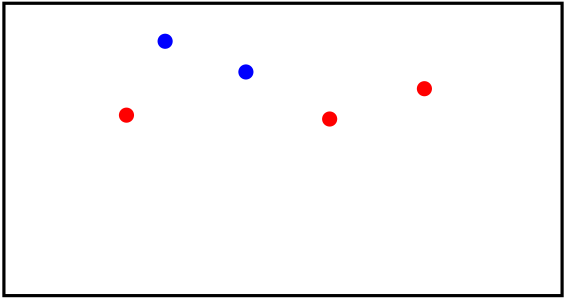
Example 2 : starting with 2 blues, 3red



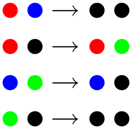
Program :



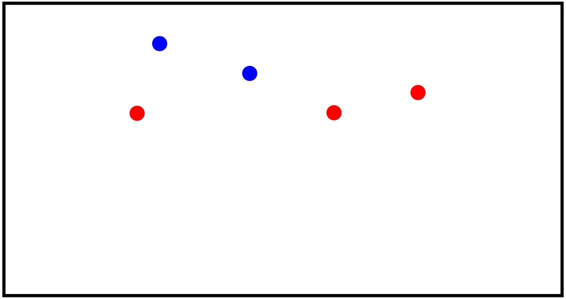
Example 2 : starting with 2 blues, 3red



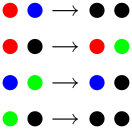
Program :



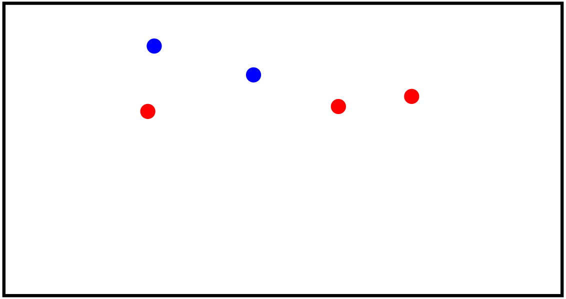
Example 2 : starting with 2 blues, 3red



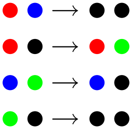
Program :



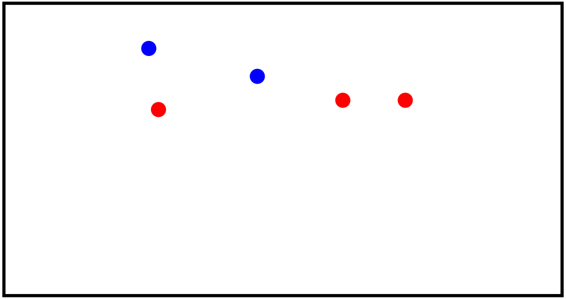
Example 2 : starting with 2 blues, 3red



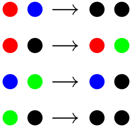
Program :



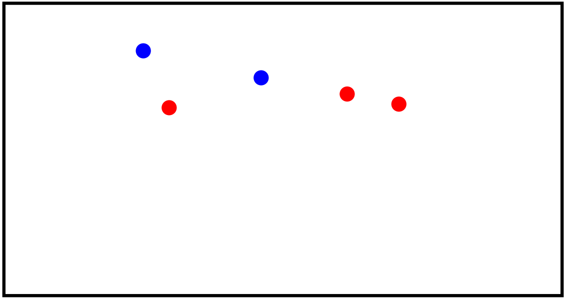
Example 2 : starting with 2 blues, 3red



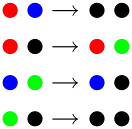
Program :



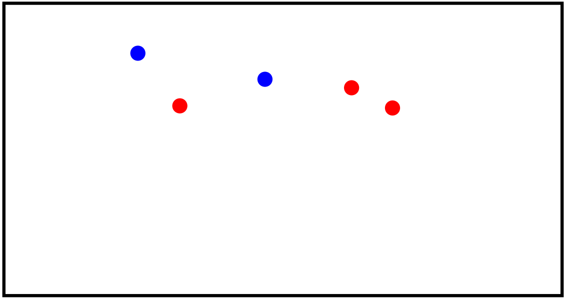
Example 2 : starting with 2 blues, 3red



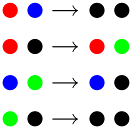
Program :



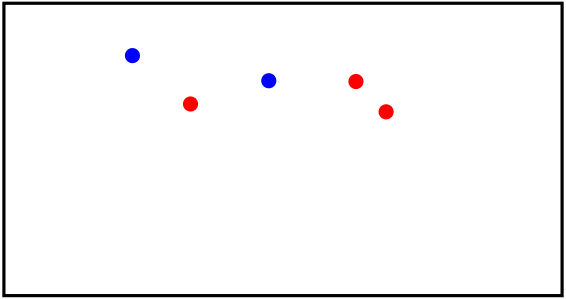
Example 2 : starting with 2 blues, 3red



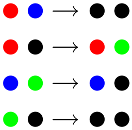
Program :



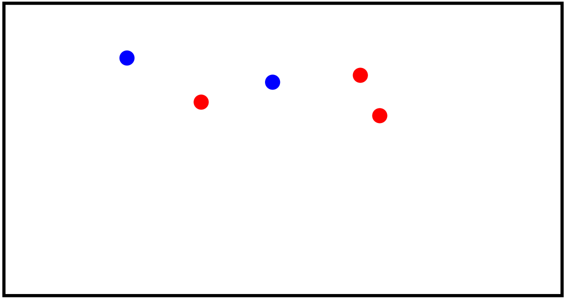
Example 2 : starting with 2 blues, 3red



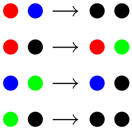
Program :



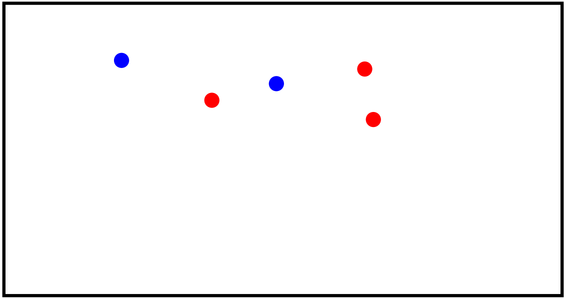
Example 2 : starting with 2 blues, 3red



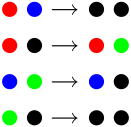
Program :



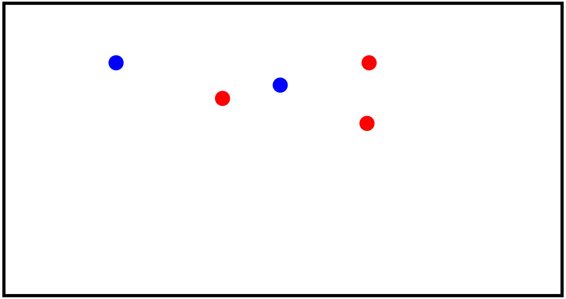
Example 2 : starting with 2 blues, 3red



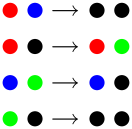
Program :



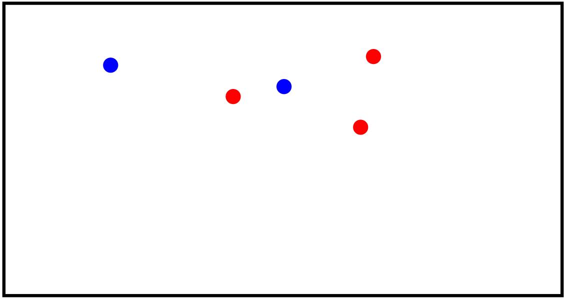
Example 2 : starting with 2 blues, 3red



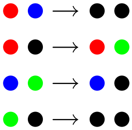
Program :



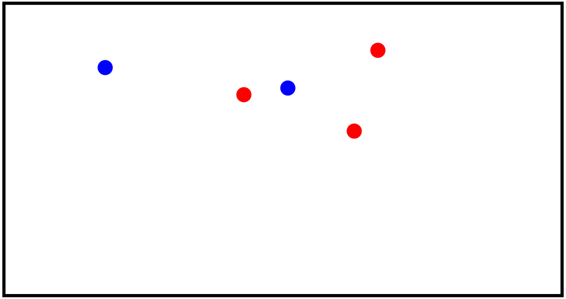
Example 2 : starting with 2 blues, 3red



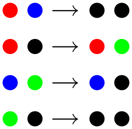
Program :



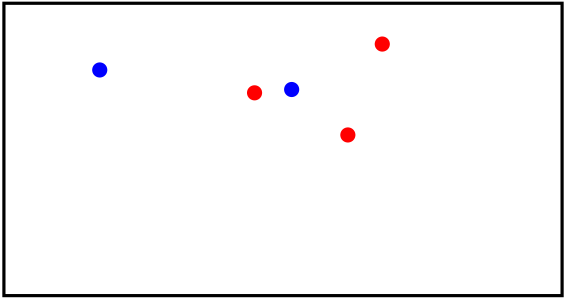
Example 2 : starting with 2 blues, 3red



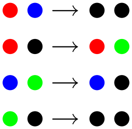
Program :



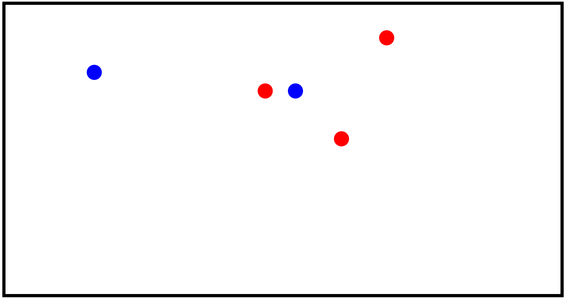
Example 2 : starting with 2 blues, 3red



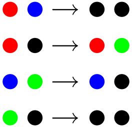
Program :



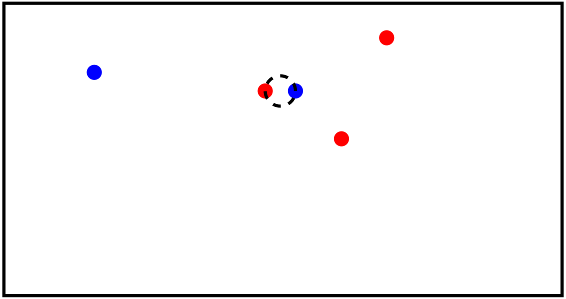
Example 2 : starting with 2 blues, 3red



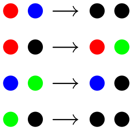
Program :



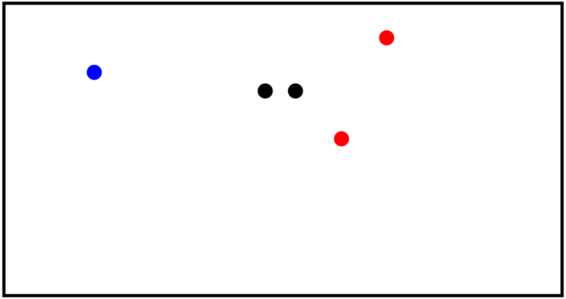
Example 2 : starting with 2 blues, 3red



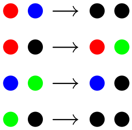
Program :



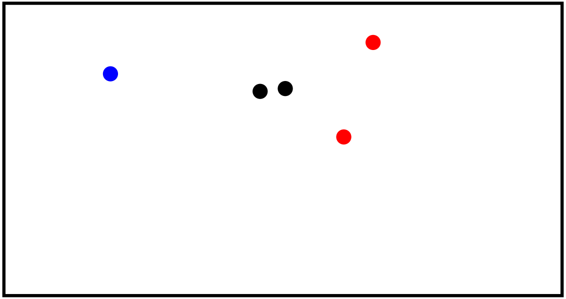
Example 2 : starting with 2 blues, 3red



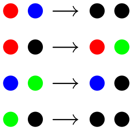
Program :



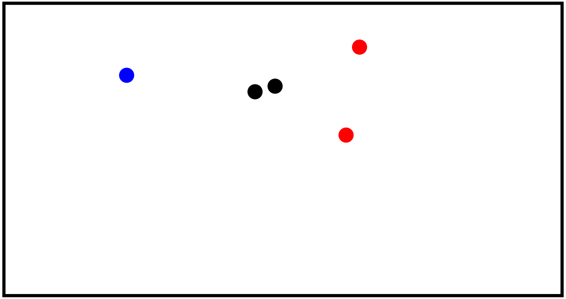
Example 2 : starting with 2 blues, 3red



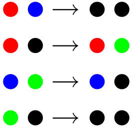
Program :



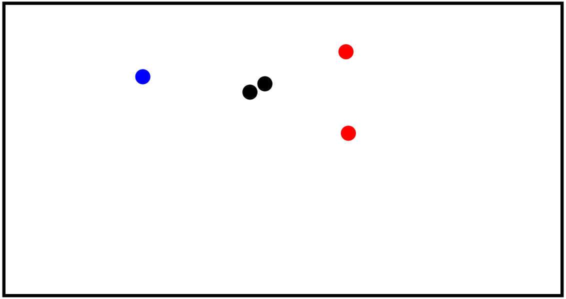
Example 2 : starting with 2 blues, 3red



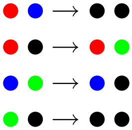
Program :



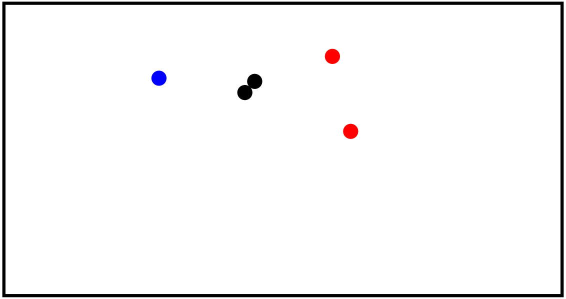
Example 2 : starting with 2 blues, 3red



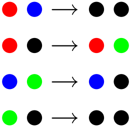
Program :



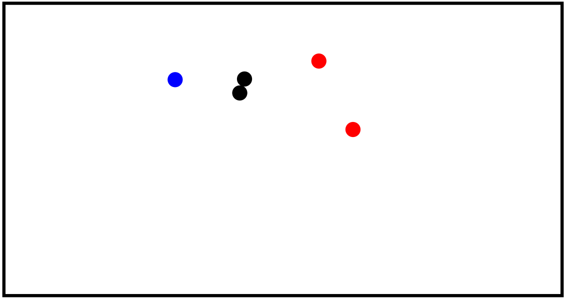
Example 2 : starting with 2 blues, 3red



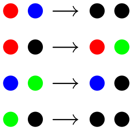
Program :



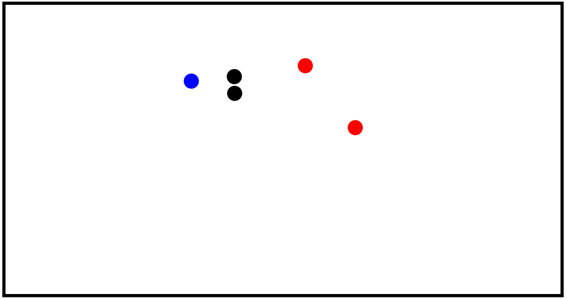
Example 2 : starting with 2 blues, 3red



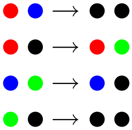
Program :



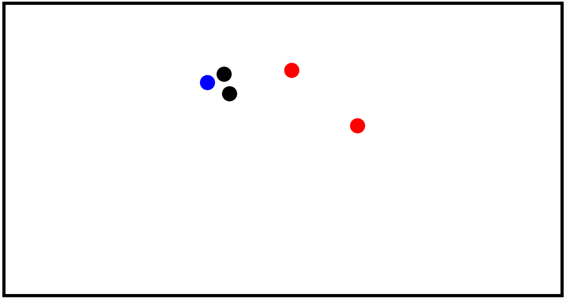
Example 2 : starting with 2 blues, 3red



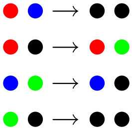
Program :



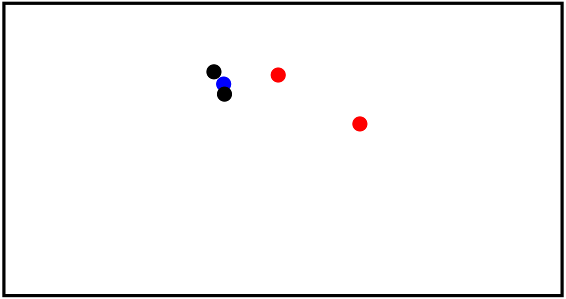
Example 2 : starting with 2 blues, 3red



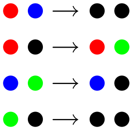
Program :



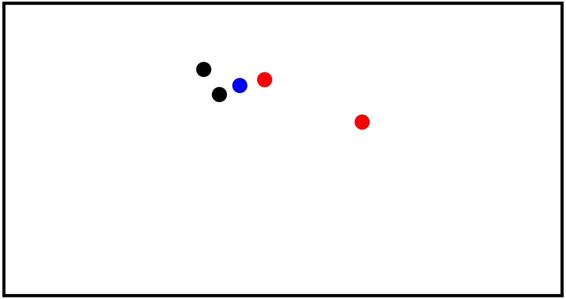
Example 2 : starting with 2 blues, 3red



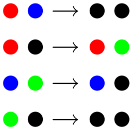
Program :



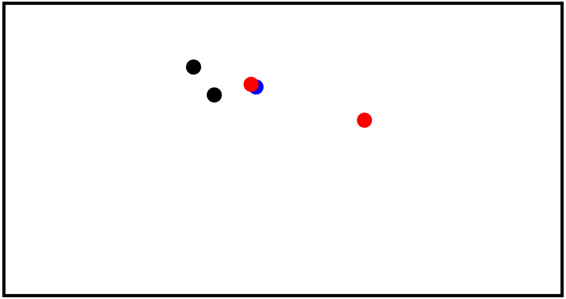
Example 2 : starting with 2 blues, 3red



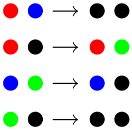
Program :



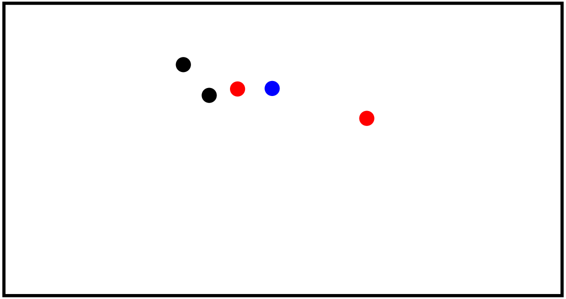
Example 2 : starting with 2 blues, 3red



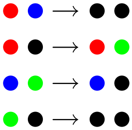
Program :



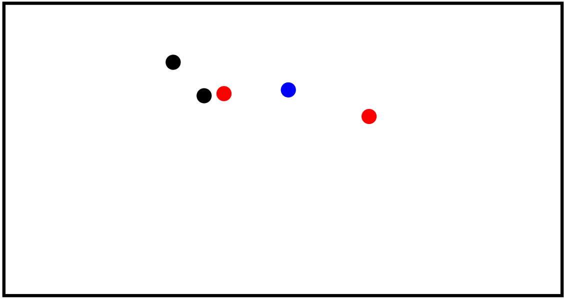
Example 2 : starting with 2 blues, 3red



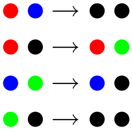
Program :



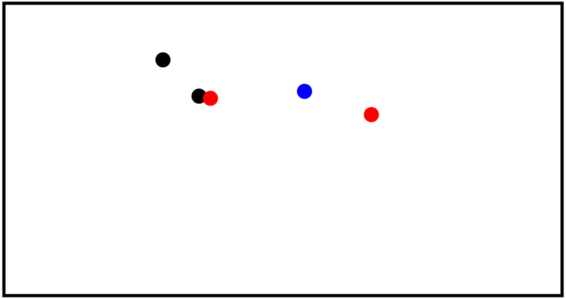
Example 2 : starting with 2 blues, 3red



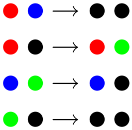
Program :



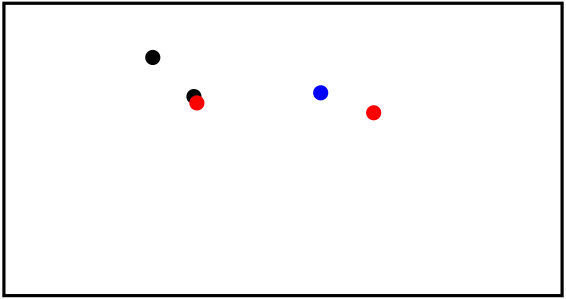
Example 2 : starting with 2 blues, 3red



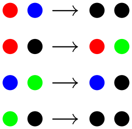
Program :



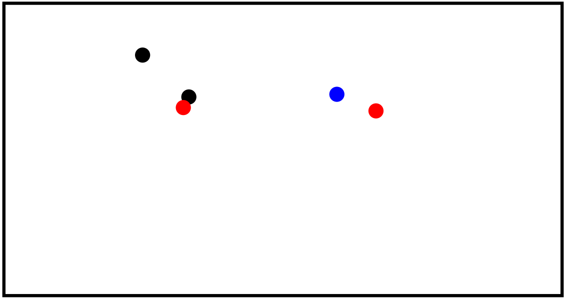
Example 2 : starting with 2 blues, 3red



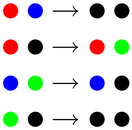
Program :



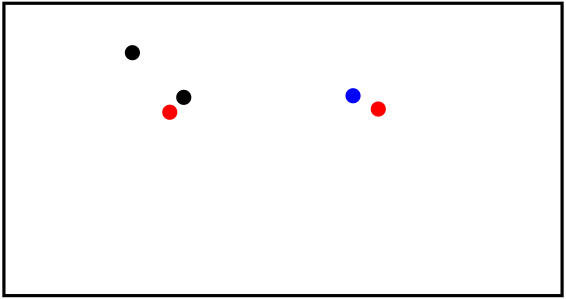
Example 2 : starting with 2 blues, 3red



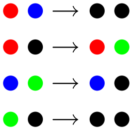
Program :



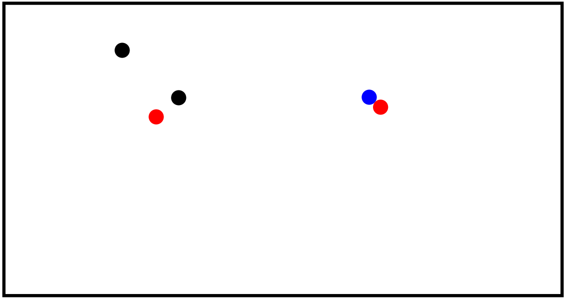
Example 2 : starting with 2 blues, 3red



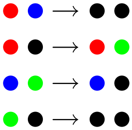
Program :



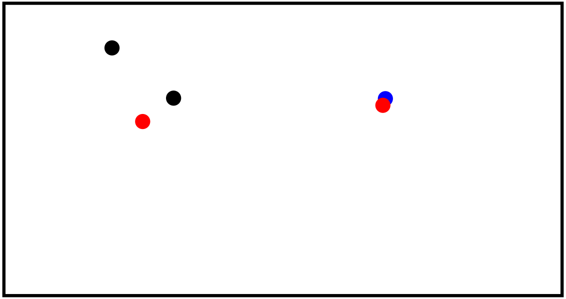
Example 2 : starting with 2 blues, 3red



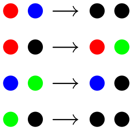
Program :



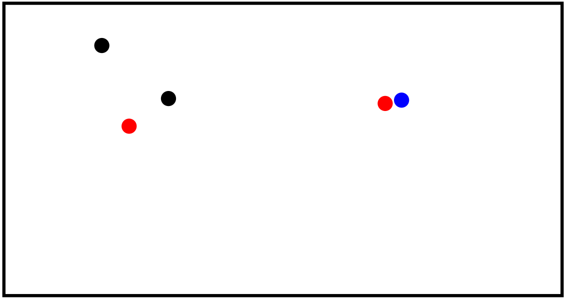
Example 2 : starting with 2 blues, 3red



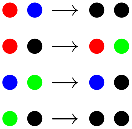
Program :



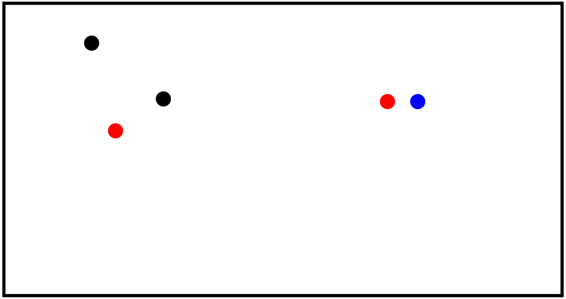
Example 2 : starting with 2 blues, 3red



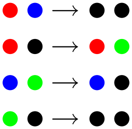
Program :



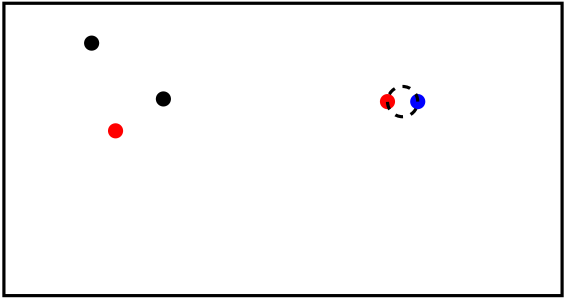
Example 2 : starting with 2 blues, 3red



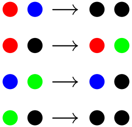
Program :



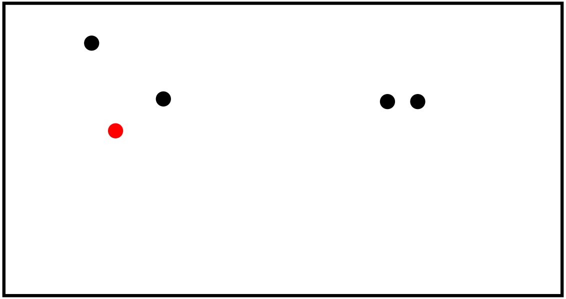
Example 2 : starting with 2 blues, 3red



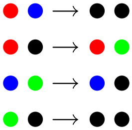
Program :



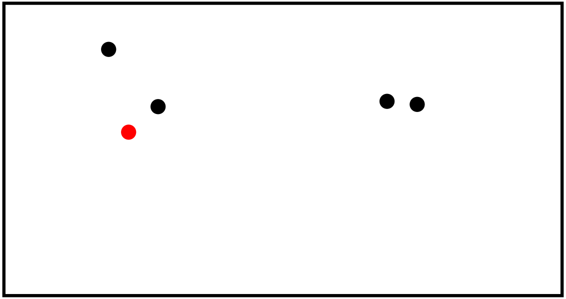
Example 2 : starting with 2 blues, 3red



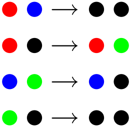
Program :



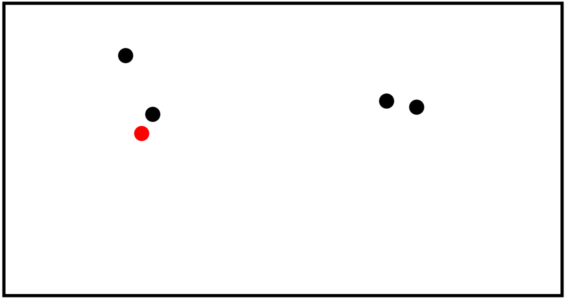
Example 2 : starting with 2 blues, 3red



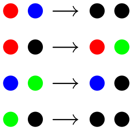
Program :



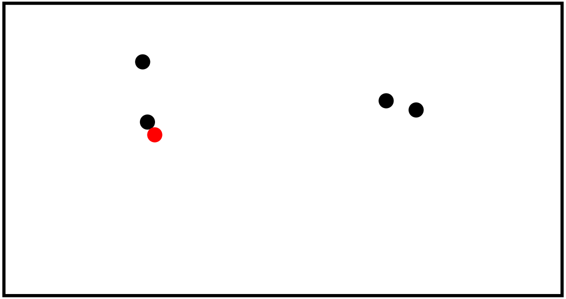
Example 2 : starting with 2 blues, 3red



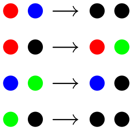
Program :



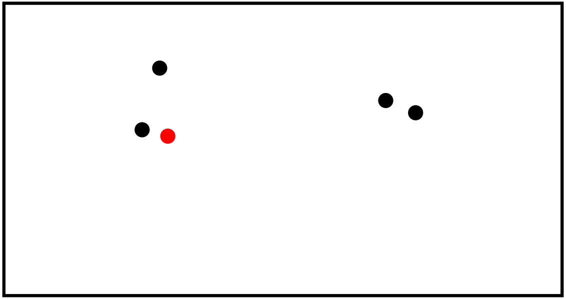
Example 2 : starting with 2 blues, 3red



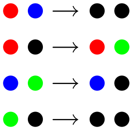
Program :



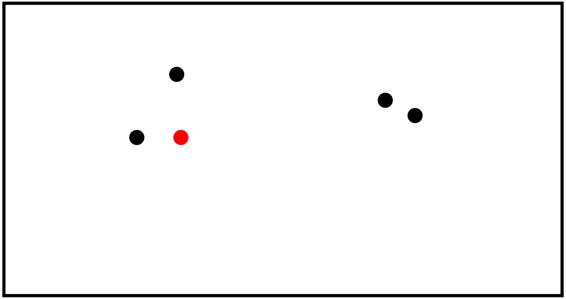
Example 2 : starting with 2 blues, 3red



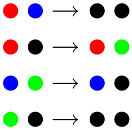
Program :



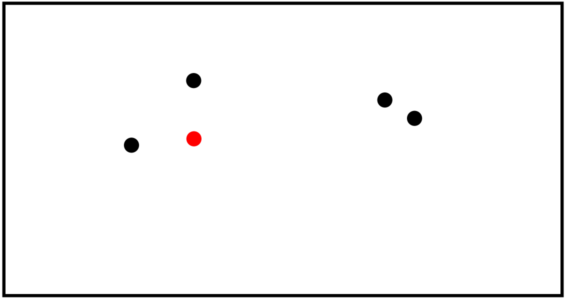
Example 2 : starting with 2 blues, 3red



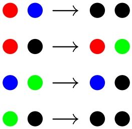
Program :



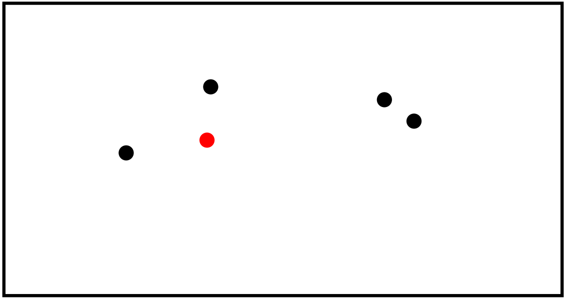
Example 2 : starting with 2 blues, 3red



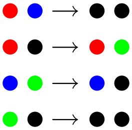
Program :



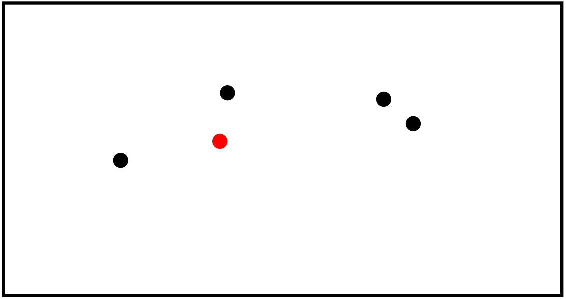
Example 2 : starting with 2 blues, 3red



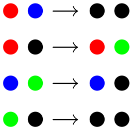
Program :



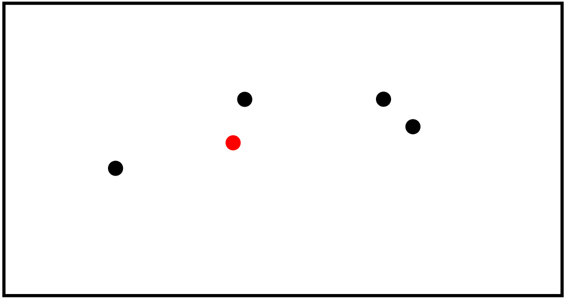
Example 2 : starting with 2 blues, 3red



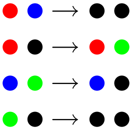
Program :



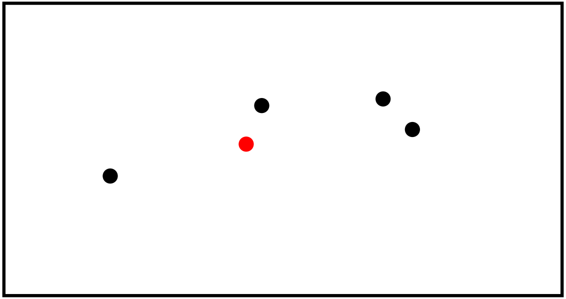
Example 2 : starting with 2 blues, 3red



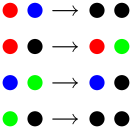
Program :



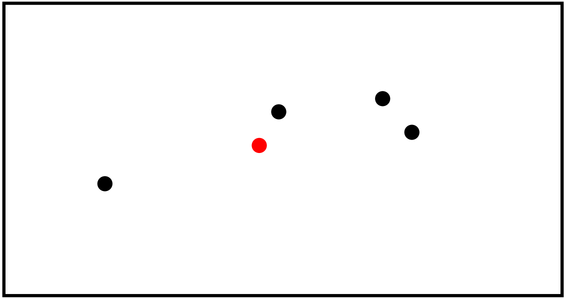
Example 2 : starting with 2 blues, 3red



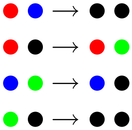
Program :



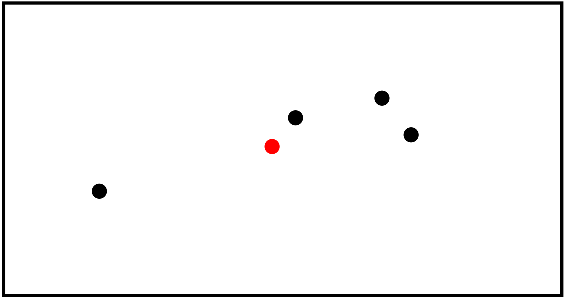
Example 2 : starting with 2 blues, 3red



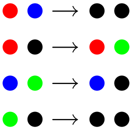
Program :



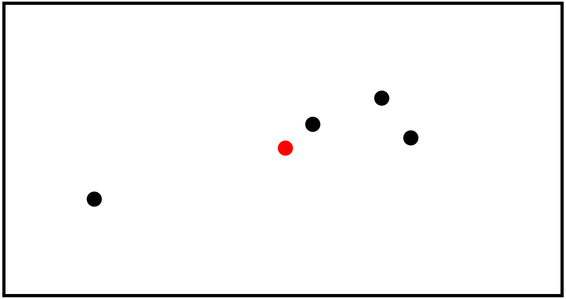
Example 2 : starting with 2 blues, 3red



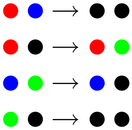
Program :



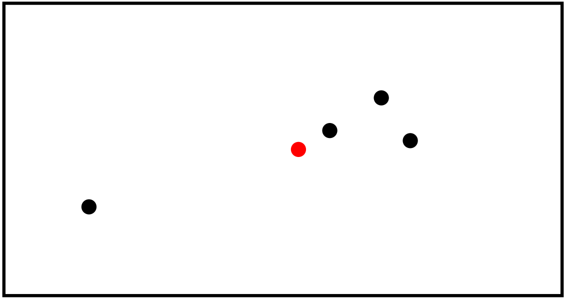
Example 2 : starting with 2 blues, 3red



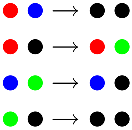
Program :



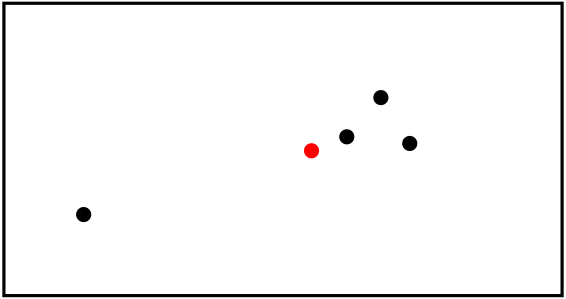
Example 2 : starting with 2 blues, 3red



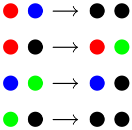
Program :



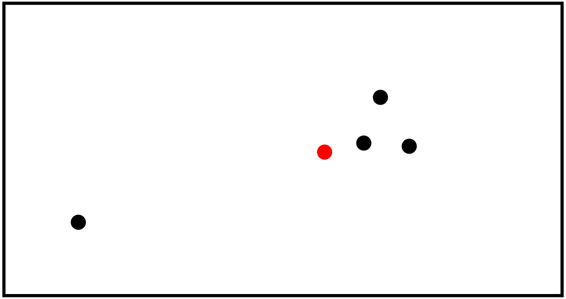
Example 2 : starting with 2 blues, 3red



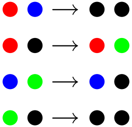
Program :



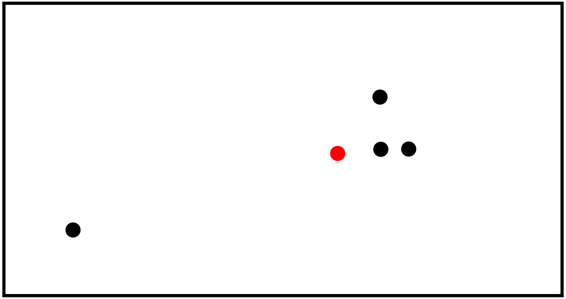
Example 2 : starting with 2 blues, 3red



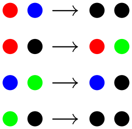
Program :



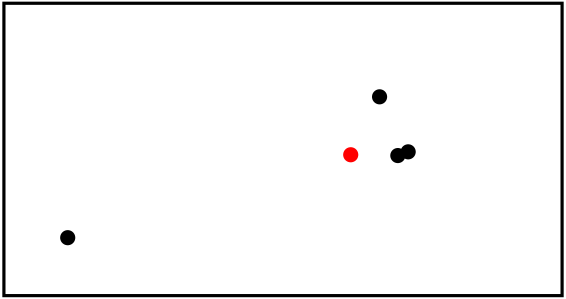
Example 2 : starting with 2 blues, 3red



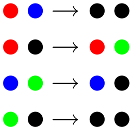
Program :



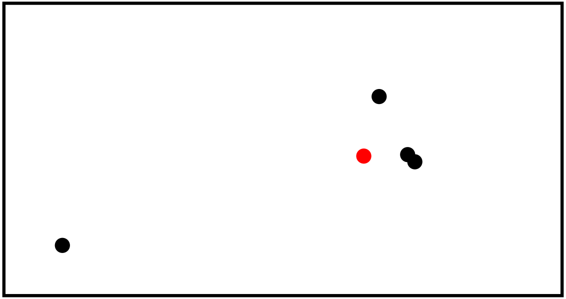
Example 2 : starting with 2 blues, 3red



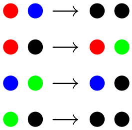
Program :



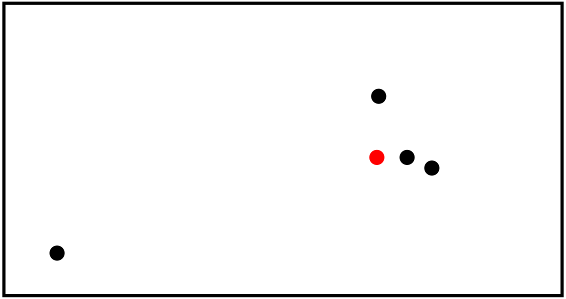
Example 2 : starting with 2 blues, 3red



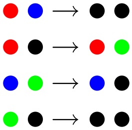
Program :



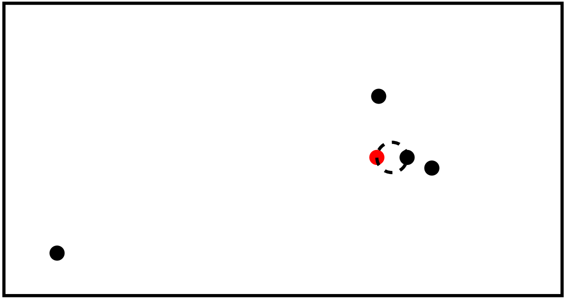
Example 2 : starting with 2 blues, 3red



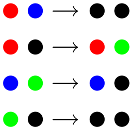
Program :



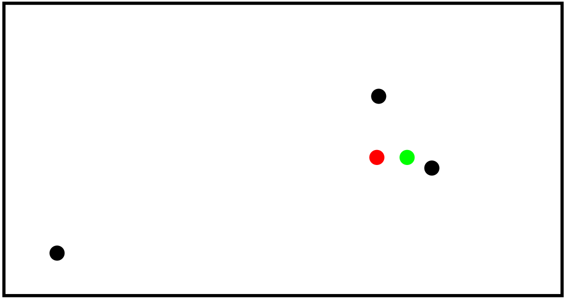
Example 2 : starting with 2 blues, 3red



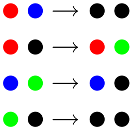
Program :



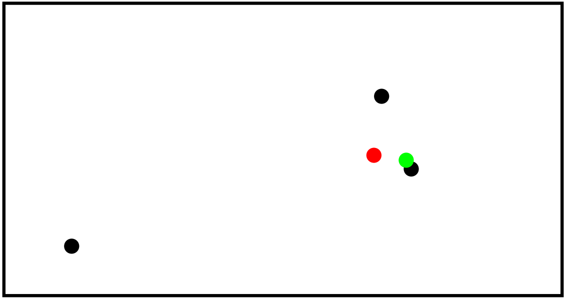
Example 2 : starting with 2 blues, 3red



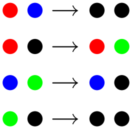
Program :



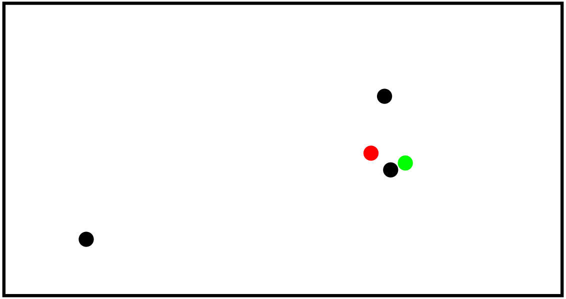
Example 2 : starting with 2 blues, 3red



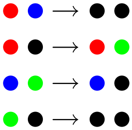
Program :



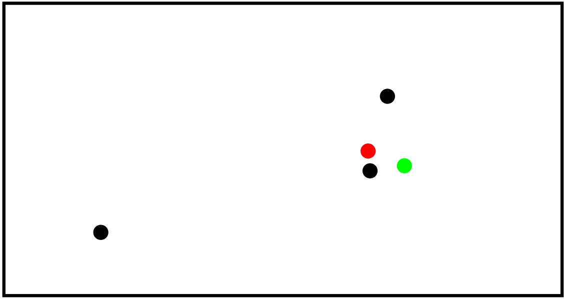
Example 2 : starting with 2 blues, 3red



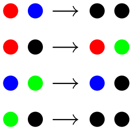
Program :



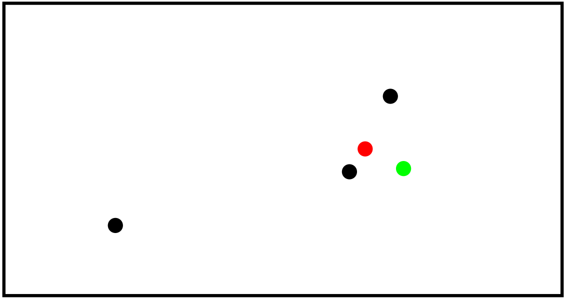
Example 2 : starting with 2 blues, 3red



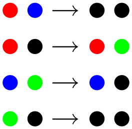
Program :



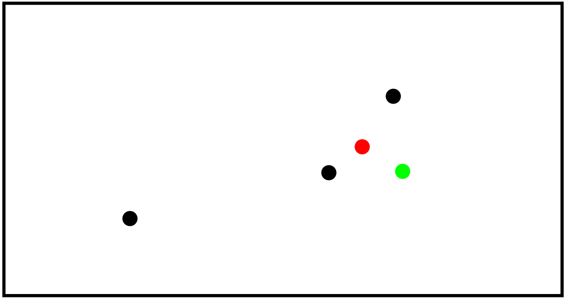
Example 2 : starting with 2 blues, 3red



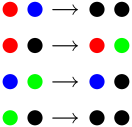
Program :



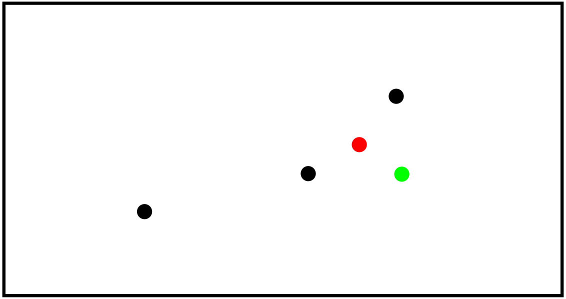
Example 2 : starting with 2 blues, 3red



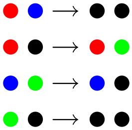
Program :



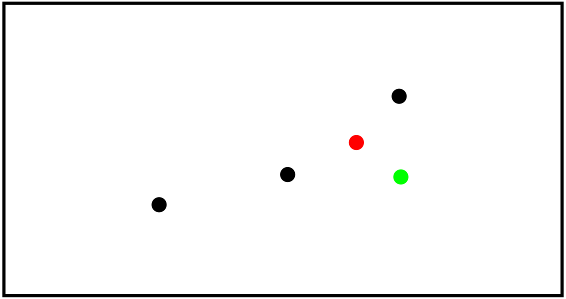
Example 2 : starting with 2 blues, 3red



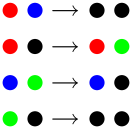
Program :



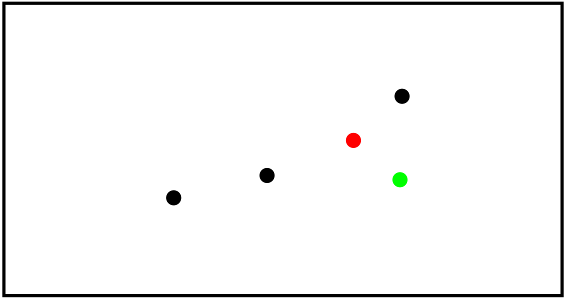
Example 2 : starting with 2 blues, 3red



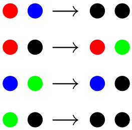
Program :



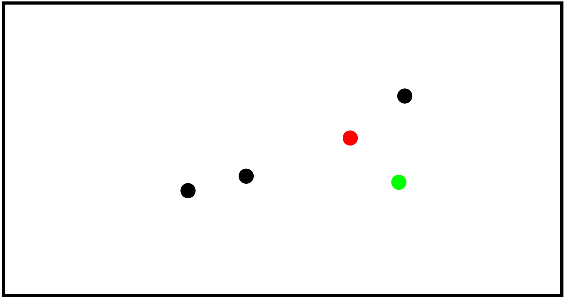
Example 2 : starting with 2 blues, 3red



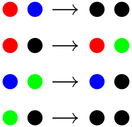
Program :



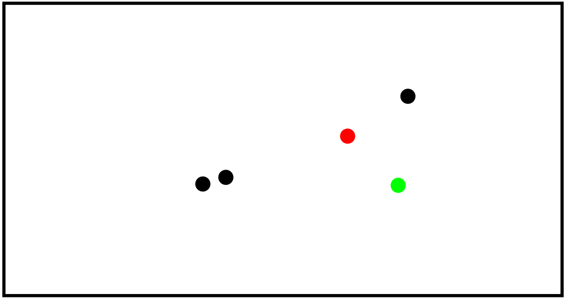
Example 2 : starting with 2 blues, 3red



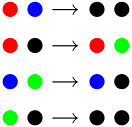
Program :



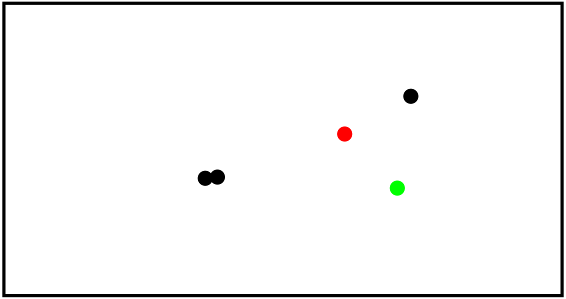
Example 2 : starting with 2 blues, 3red



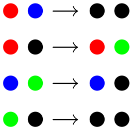
Program :



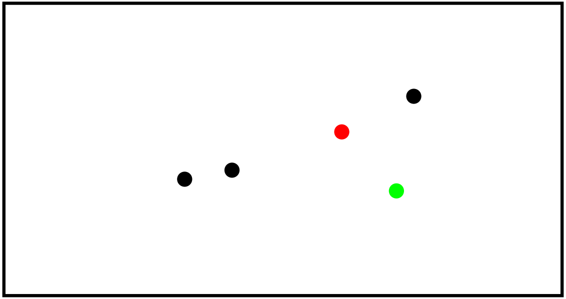
Example 2 : starting with 2 blues, 3red



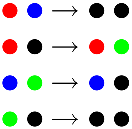
Program :



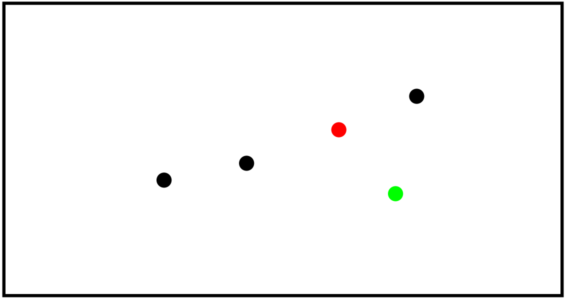
Example 2 : starting with 2 blues, 3red



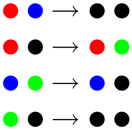
Program :



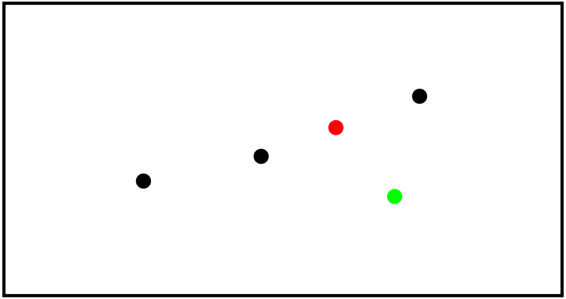
Example 2 : starting with 2 blues, 3red



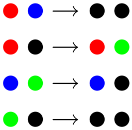
Program :



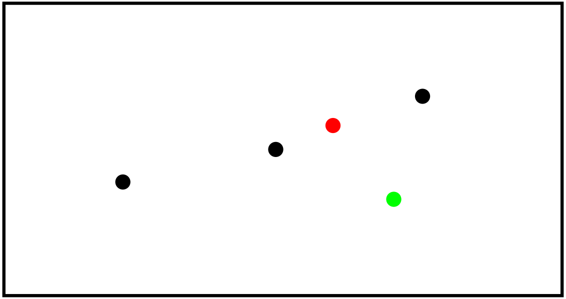
Example 2 : starting with 2 blues, 3red



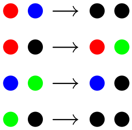
Program :



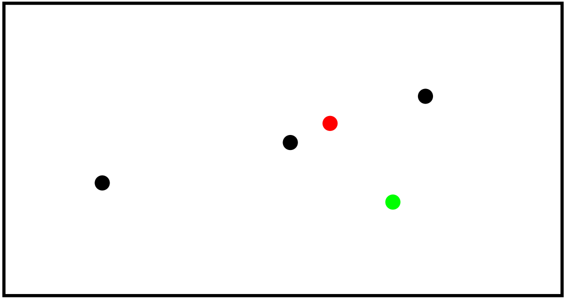
Example 2 : starting with 2 blues, 3red



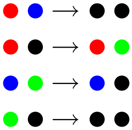
Program :



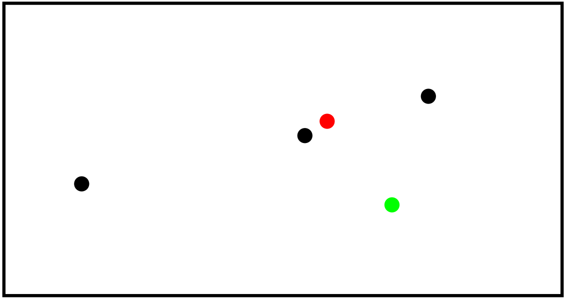
Example 2 : starting with 2 blues, 3red



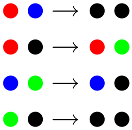
Program :



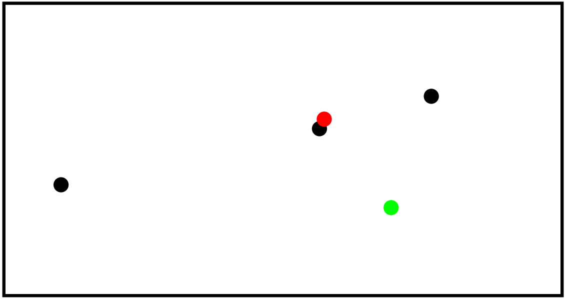
Example 2 : starting with 2 blues, 3red



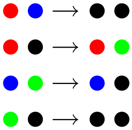
Program :



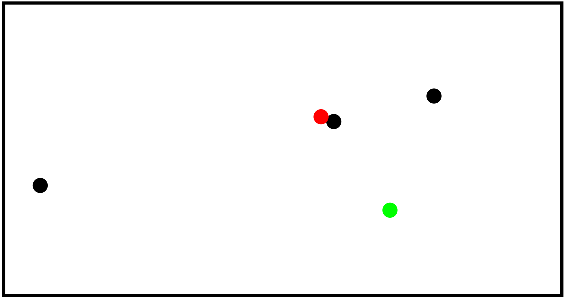
Example 2 : starting with 2 blues, 3red



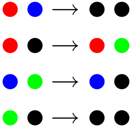
Program :



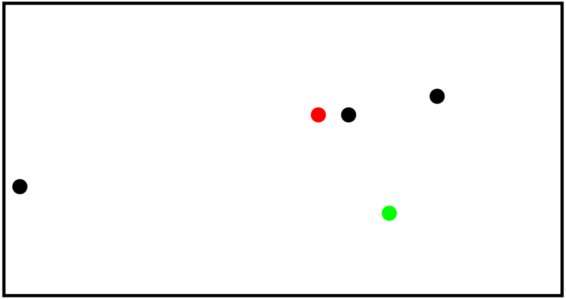
Example 2 : starting with 2 blues, 3red



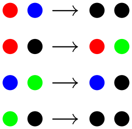
Program :



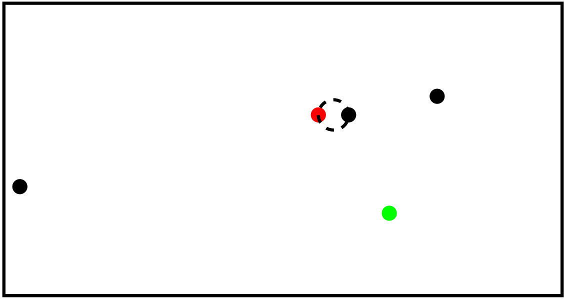
Example 2 : starting with 2 blues, 3red



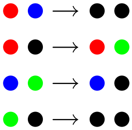
Program :



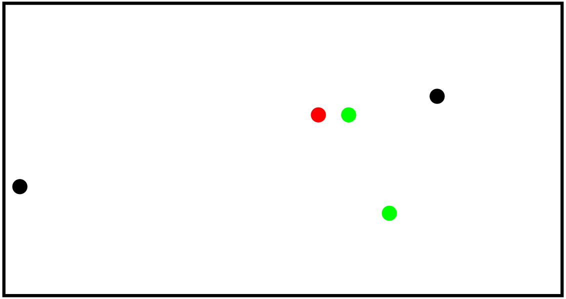
Example 2 : starting with 2 blues, 3red



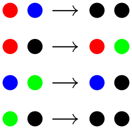
Program :



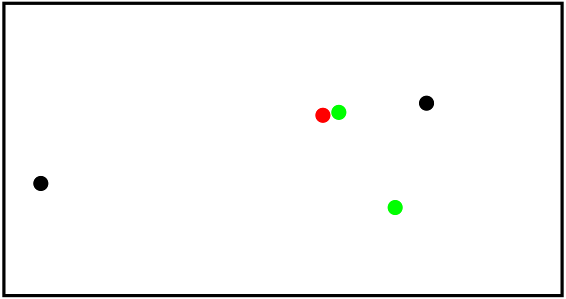
Example 2 : starting with 2 blues, 3red



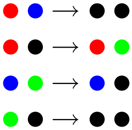
Program :



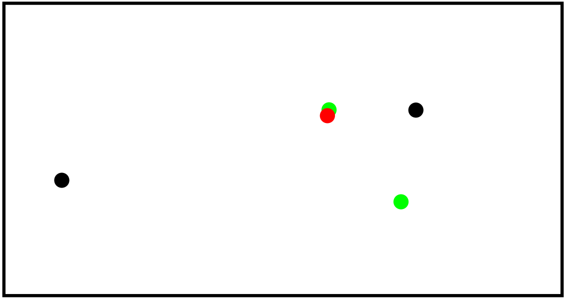
Example 2 : starting with 2 blues, 3red



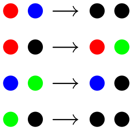
Program :



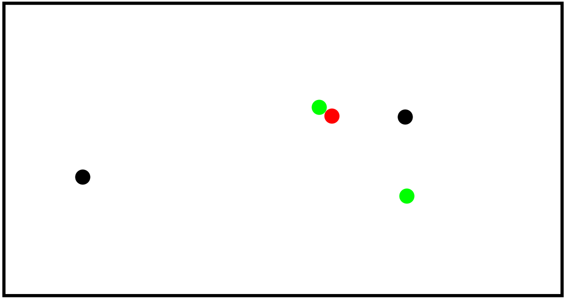
Example 2 : starting with 2 blues, 3red



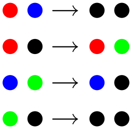
Program :



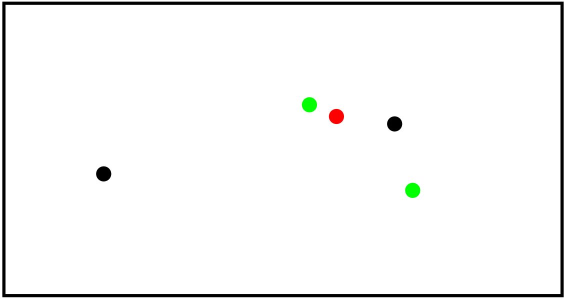
Example 2 : starting with 2 blues, 3red



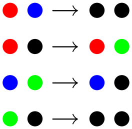
Program :



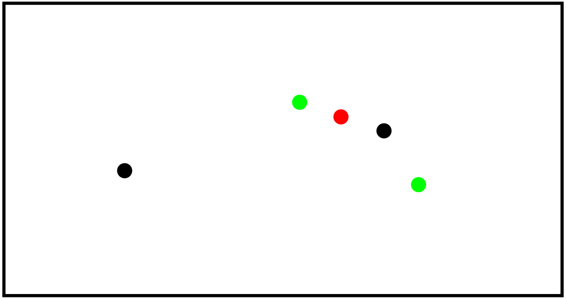
Example 2 : starting with 2 blues, 3red



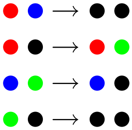
Program :



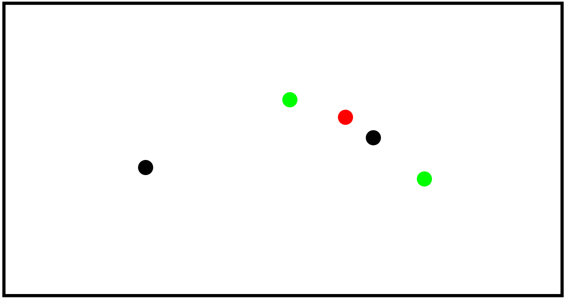
Example 2 : starting with 2 blues, 3red



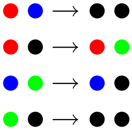
Program :



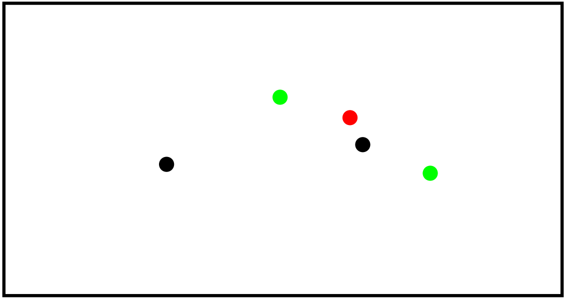
Example 2 : starting with 2 blues, 3red



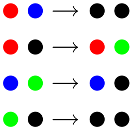
Program :



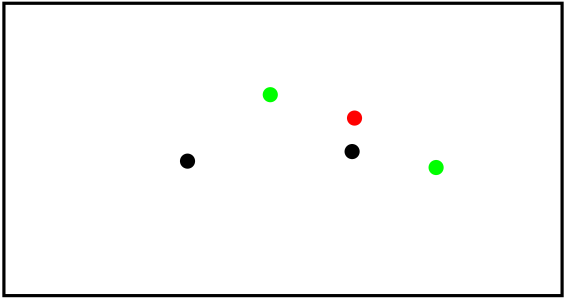
Example 2 : starting with 2 blues, 3red



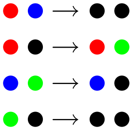
Program :



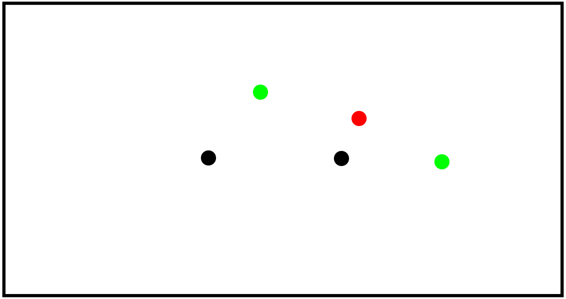
Example 2 : starting with 2 blues, 3red



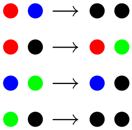
Program :



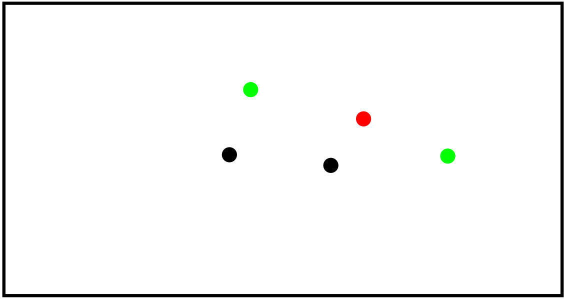
Example 2 : starting with 2 blues, 3red



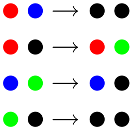
Program :



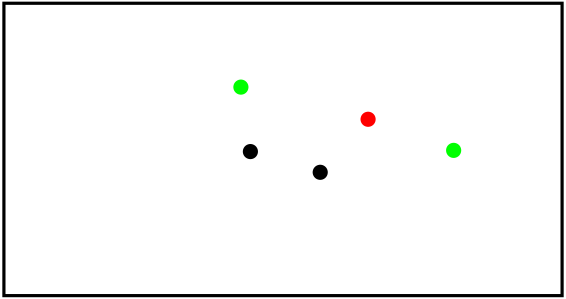
Example 2 : starting with 2 blues, 3red



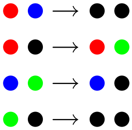
Program :



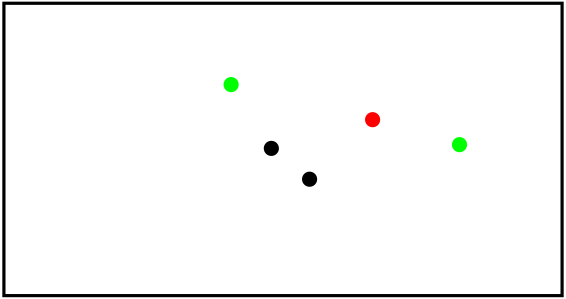
Example 2 : starting with 2 blues, 3red



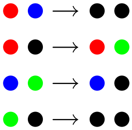
Program :



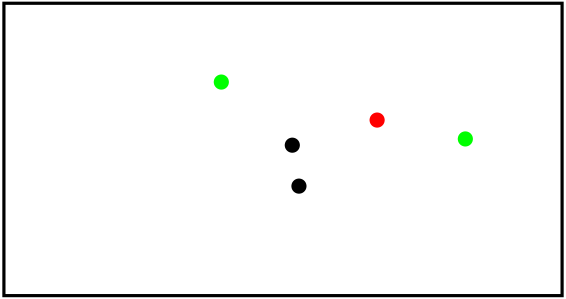
Example 2 : starting with 2 blues, 3red



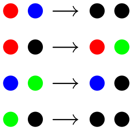
Program :



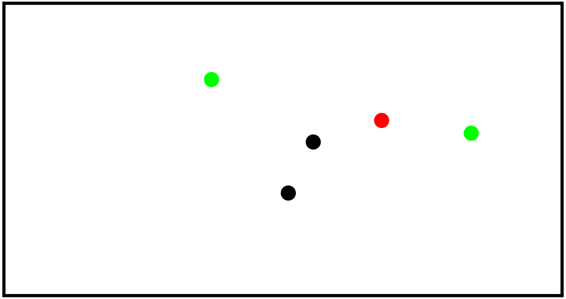
Example 2 : starting with 2 blues, 3red



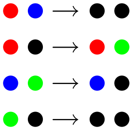
Program :



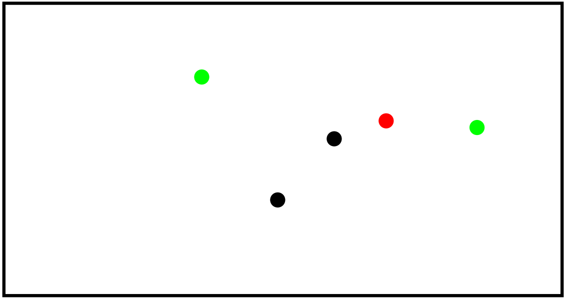
Example 2 : starting with 2 blues, 3red



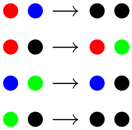
Program :



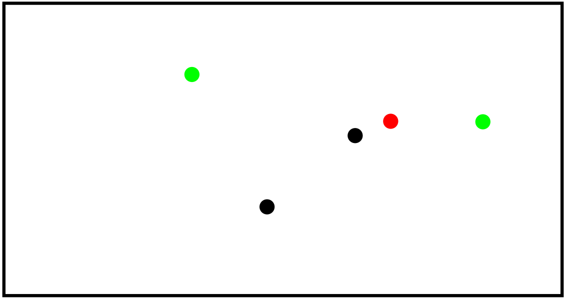
Example 2 : starting with 2 blues, 3red



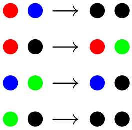
Program :



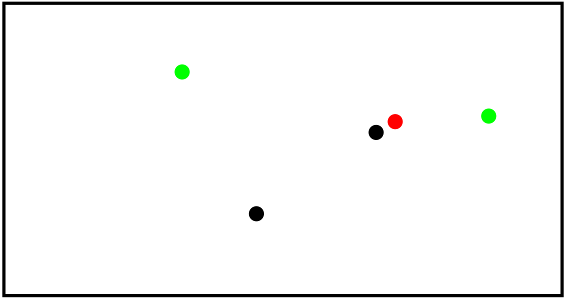
Example 2 : starting with 2 blues, 3red



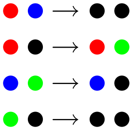
Program :



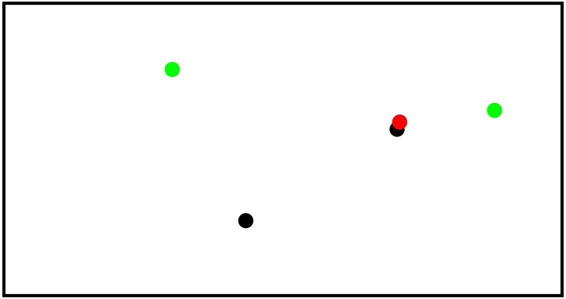
Example 2 : starting with 2 blues, 3red



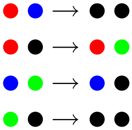
Program :



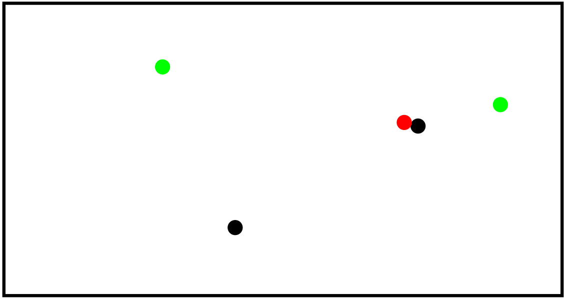
Example 2 : starting with 2 blues, 3red



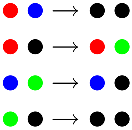
Program :



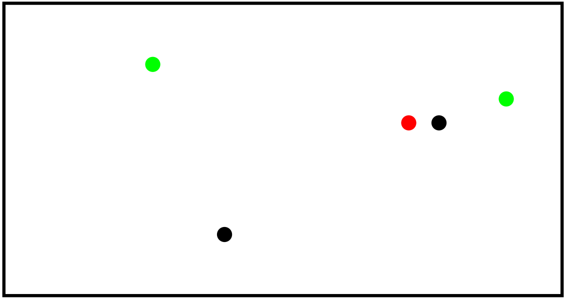
Example 2 : starting with 2 blues, 3red



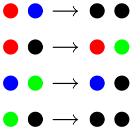
Program :



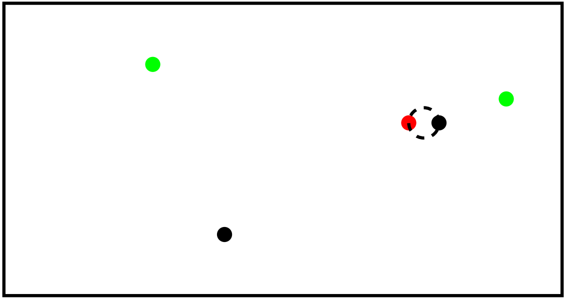
Example 2 : starting with 2 blues, 3red



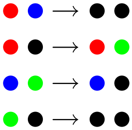
Program :



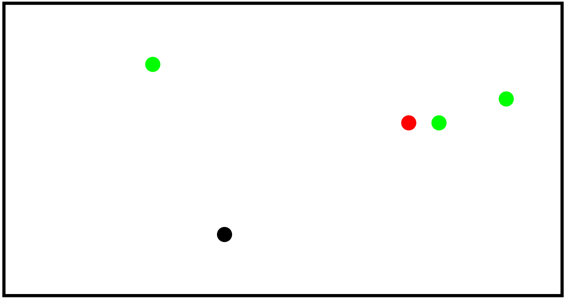
Example 2 : starting with 2 blues, 3red



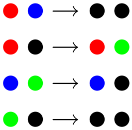
Program :



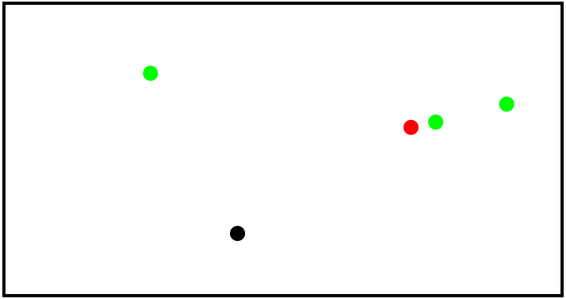
Example 2 : starting with 2 blues, 3red



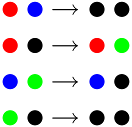
Program :



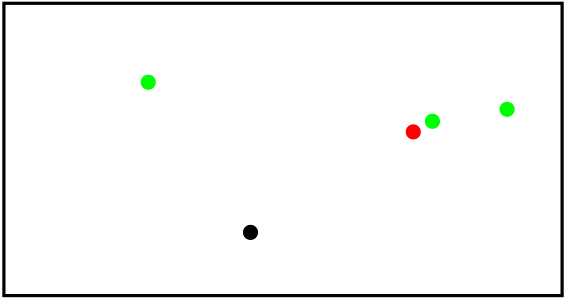
Example 2 : starting with 2 blues, 3red



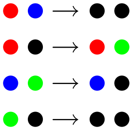
Program :



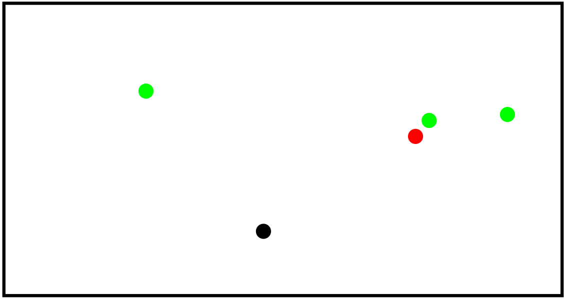
Example 2 : starting with 2 blues, 3red



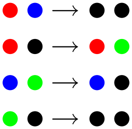
Program :



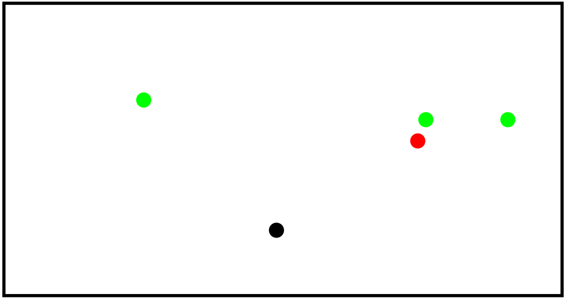
Example 2 : starting with 2 blues, 3red



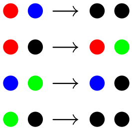
Program :



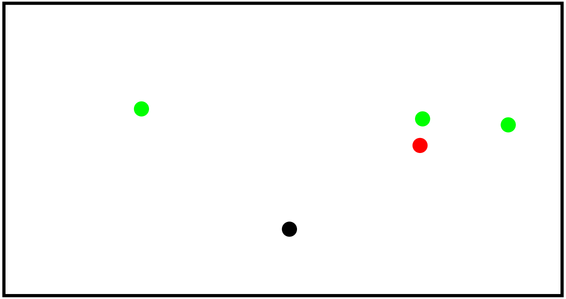
Example 2 : starting with 2 blues, 3red



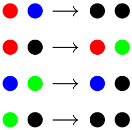
Program :



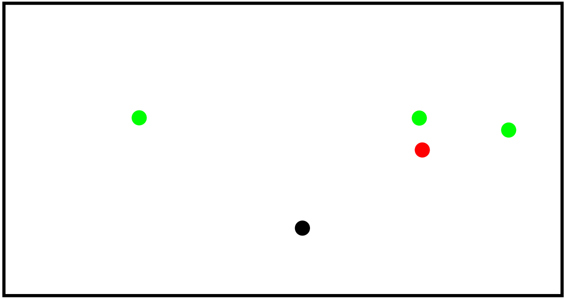
Example 2 : starting with 2 blues, 3red



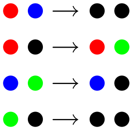
Program :



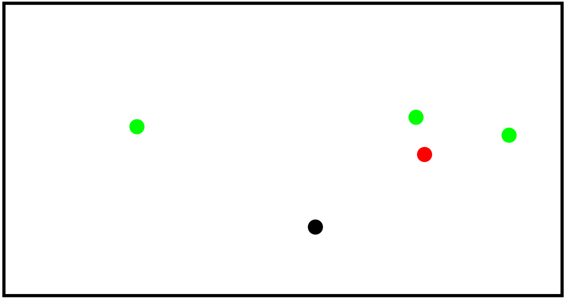
Example 2 : starting with 2 blues, 3red



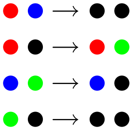
Program :



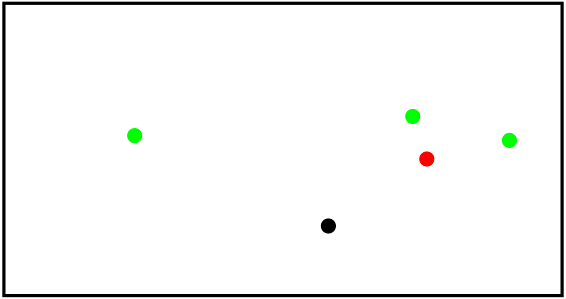
Example 2 : starting with 2 blues, 3red



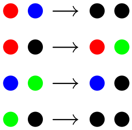
Program :



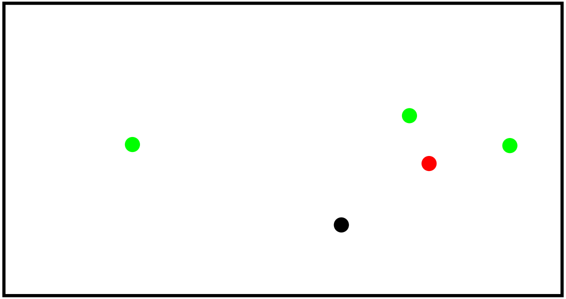
Example 2 : starting with 2 blues, 3red



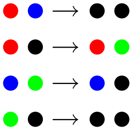
Program :



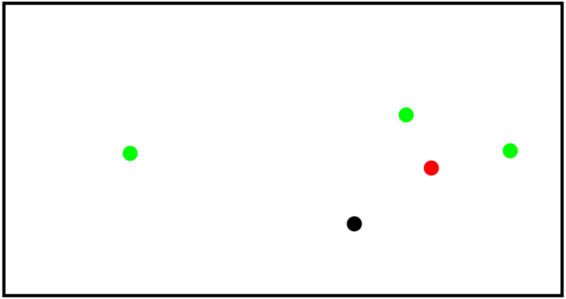
Example 2 : starting with 2 blues, 3red



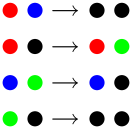
Program :



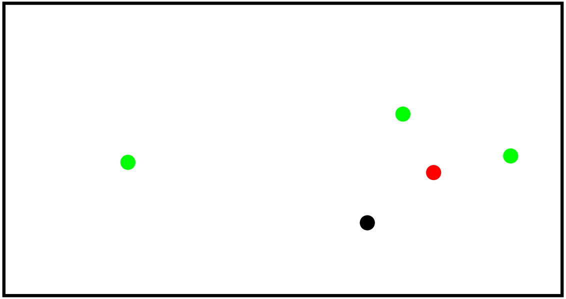
Example 2 : starting with 2 blues, 3red



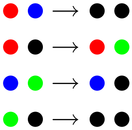
Program :



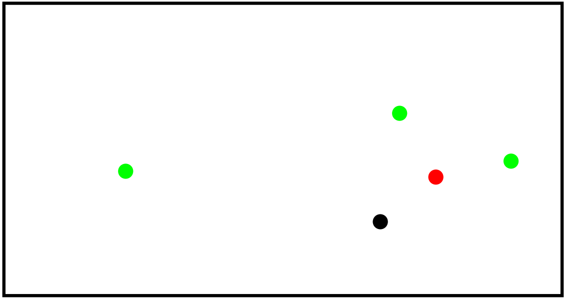
Example 2 : starting with 2 blues, 3red



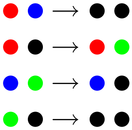
Program :



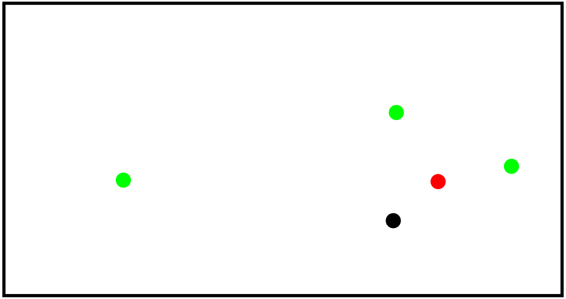
Example 2 : starting with 2 blues, 3red



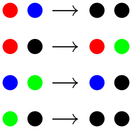
Program :



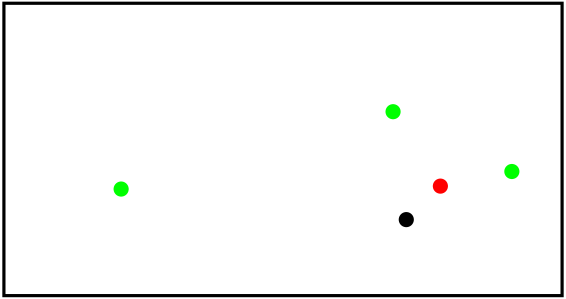
Example 2 : starting with 2 blues, 3red



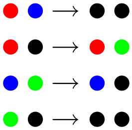
Program :



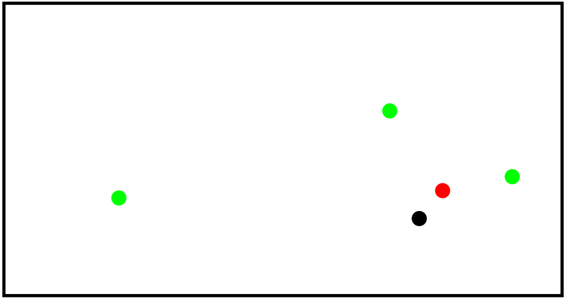
Example 2 : starting with 2 blues, 3red



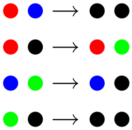
Program :



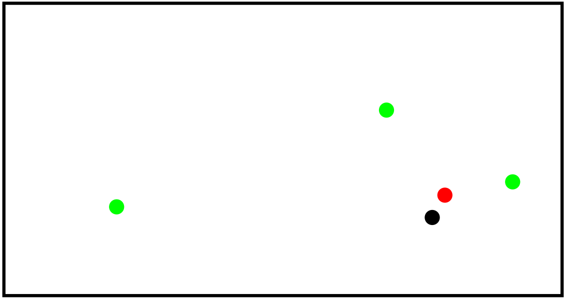
Example 2 : starting with 2 blues, 3red



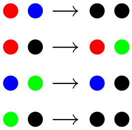
Program :



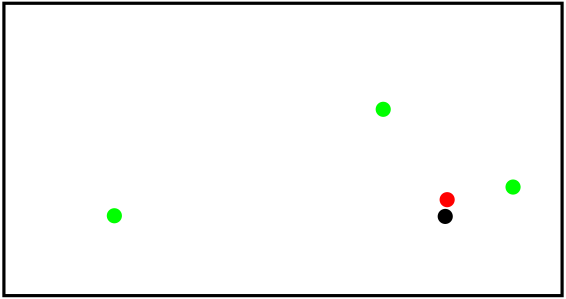
Example 2 : starting with 2 blues, 3red



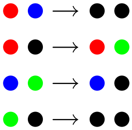
Program :



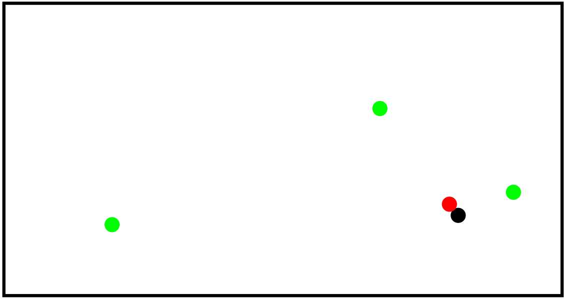
Example 2 : starting with 2 blues, 3red



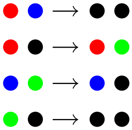
Program :



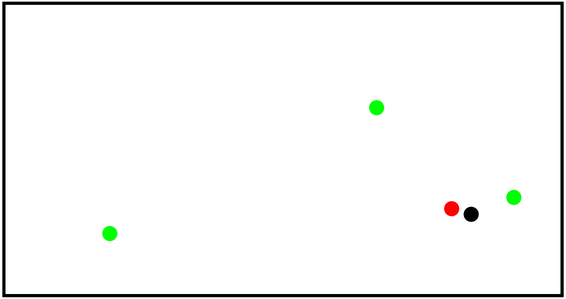
Example 2 : starting with 2 blues, 3red



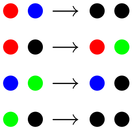
Program :



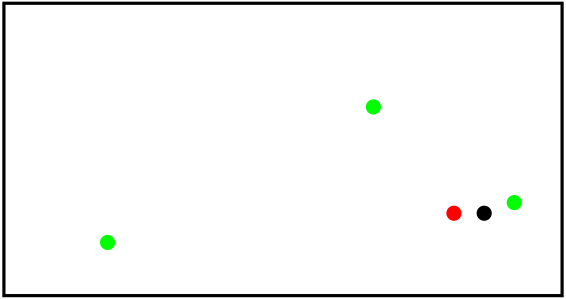
Example 2 : starting with 2 blues, 3red



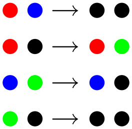
Program :



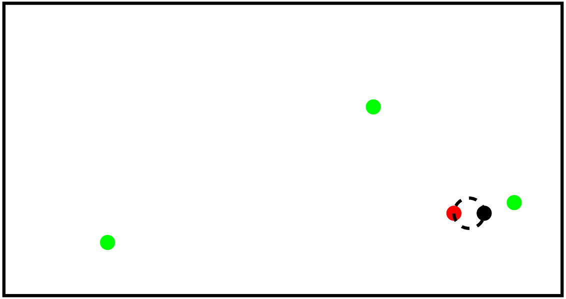
Example 2 : starting with 2 blues, 3red



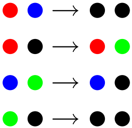
Program :



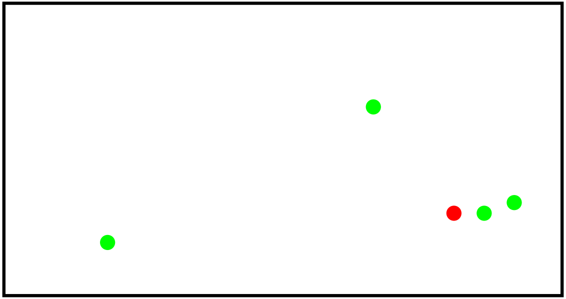
Example 2 : starting with 2 blues, 3red



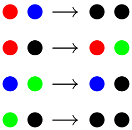
Program :



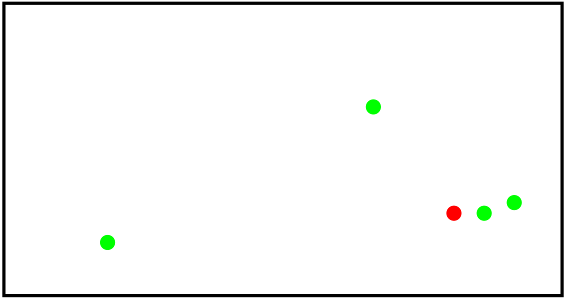
Example 2 : starting with 2 blues, 3red



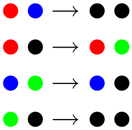
Program :



Example 2 : Final result



Program :



What is computed ?

Let's interpret

- ● and ● by *yes*.
- ● and ● by *no*.

- Whatever the initial state is, ultimately, all agents will agree.

- They agree on *yes* iff the initial population of ● is strictly greater than the initial population of ●.

Alternative statement

- A configuration can be seen as an element of $\mathbb{N}^4$.
  - (n_b, n_r, n_g, n_{bl}) if there is n_b ●, n_r ●, n_g ●and n_{bl} ●
  - $n_b + n_r + n_g + n_{bl}$ is preserved.
- An initial configuration is of type $(n_b, n_r, 0, 0)$.
- This protocol computes predicates $n_r > n_b$, i.e.
  MAJORITY .

General Case : Algorithm

An algorithm consists of

- a finite set of states $Q = \{q_1, q_2, \cdots, q_k\}$.
- transition rules, mapping pairs of states to pairs of states

Executions are given by :

- instantaneous configuration : a multiset of states = an element of $\mathbb{N}^k$.
- transitions between configuration : two agents are picked, and updated according to the rules of the algorithm.
- output interpretation : mapping states to $\{0, 1\}$.
- a computation is over when all agents agree on 0 or 1.

Remark : Algorithm are assumed independent of size of population !

Simplest example : Computing OR of input bits

States : $\textcolor{blue}{\bullet}$,$\textcolor{red}{\bullet}$

One transition rule : $\textcolor{red}{\bullet}\ \textcolor{blue}{\bullet} \longrightarrow \textcolor{red}{\bullet}\ \textcolor{red}{\bullet}$

Output of an agent is its state.

If all inputs are $\textcolor{blue}{\bullet}$, all agents will remain in state $\textcolor{blue}{\bullet}$
If some agent is $\textcolor{red}{\bullet}$, eventually all will have state $\textcolor{red}{\bullet}$

A remark : Fairness

- One need to guarantee that all possible interactions happen eventually
  - an execution is *fair* if for all configurations C that appear infinitely often in the execution, if $C \rightarrow C'$ for some configuration C', then C' appears infinitely often in the execution.
  - can be seen as capturing the idea of probabilistic adversary : there is some (unknown) underlying probability distribution on interactions such that events are independent.

- True notion of computation : For any input I , for any fair sequence of executions starting from I agents ultimately agree on 0 or 1.

Leader Election

Initially, all agents in same state $\color{red}{\bullet}$

Eventually, exactly one agent is in a special leader state $\color{red}{\bullet}$

Program :

$$\color{red}{\bullet}\,\color{red}{\bullet} \color{black}{\longrightarrow} \color{blue}{\bullet}\,\color{red}{\bullet}$$

Threshold Predicate

- Suppose each agent starts with input ● or ●
- Determine whether at least five agents have input ●.

5%

- Each agent is initially ● or ●
- Determine whether at least 5% of the inputs are ●

5%

- Each agent is initially ● or ●
- Determine whether at least 5% of the inputs are ●

- Similar to majority, except each ● can cancel 19 ●'s.

40%

- Each agent is initially ● or ●
- Determine whether at least 5% of the inputs are ●

40%

- Each agent is initially ● or ●
- Determine whether at least 5% of the inputs are ●

- A bit trickier (exercice).

How to Compute $\sum_{i=1}^{k} c_i x_i \geq a$

Input convention : each agent with ith input symbol starts in state c_i.

Each agent has a **leader bit** governed by $\textcolor{red}{\bullet}\ \textcolor{red}{\bullet} \rightarrow \textcolor{blue}{\bullet}\ \textcolor{red}{\bullet}$.

Let $m = \max(|a| + 1, |c_1|, ..., |c_k|)$.

Each agent also stores a value from $-m, -m+1, ..., m-1, m$.

If a leader meets a non-leader, their values change as follows :

$$\begin{array}{ll}
x, y \rightarrow x + y, 0 & \textit{if } 0 \leq x + y \leq m \\
x, y \rightarrow m, x + y - m & \textit{if } x + y > m \\
x, y \rightarrow -m, x + y + m & \textit{if } x + y < -m
\end{array}$$

(In each case first agent on right hand side is the leader.)

Each agent also remembers **output** of last leader it met.

Correctness

Sum of agents' values is invariant.

Sum is eventually gathered into the unique leader
(up to maximum absolute value of m) :

If $sum > m$, leader has value m $\Rightarrow$ Output Yes.
If $sum < -m$, leader has value -m $\Rightarrow$ Output No.
If $-m \leq sum \leq m$, leader's value is the actual sum $\Rightarrow$ Output depends on sum.

In each case, leader knows output and tells everyone else.

Computable Predicates

Theorem (Angluin et al. 2006)

A predicate is computable iff it is on the following list.

- $\sum_{i=1}^{k} c_i x_i \geq a$, *where* a, c_i*'s are integer constants*
- $\sum_{i=1}^{k} c_i x_i \equiv a \pmod{b}$ *where* a, b *and* c_i*'s are constants*
- *Boolean combinations of the above predicates*

Alternate Characterization : Presburger Arithmetic

A predicate is computable iff it can be expressed in first-order logic using the symbols $+, 0, 1, \vee, \wedge, \neg, \forall, \exists, =, <, (,)$ and variables.

(This system is known as Presburger Arithmetic [1929].)

Note : no multiplication.

Examples :
majority : $x_0 < x_1$
divisible by 3 : $\exists y : y + y + y = x_1$
at least 40% :$x0 + x0 < x1 + x1 + x1$

Alternate Characterization : Semilinear Sets

A predicate is computable iff it the set of inputs with output yes is semilinear.

A set of vectors $\vec{x} = (x_1, x_2, ..., x_k) \in \mathbb{N}^k$ is **linear** if it is of the form $\{\vec{v_0} + c_1\vec{v_1} + c_2\vec{v_2} + \ + c_m\vec{v_m} : c_1, ..., c_m \in \mathbb{N}\}$

A set of vectors is **semilinear** if it is a finite union of linear sets.

Variants of the model

The basic classical model is now already understood pretty well.

Variants considered in literature :

- Limited interaction graph
- One-way communication
- Failures

Dana Angluin, James Aspnes, Melody Chan, Carole Delporte-Gallet, Zoë Diamadi, David Eisenstat, Michael J. Fischer, Hugues Fauconnier, Rachid Guerraoui, Hong Jiang, René Peralta, Eric Ruppert

Plan

Population Protocols

Variants

Population Protocols and Games

One-way communication

- Classical model assumes an interaction can update the state of both agents simultaneously.
- One-way interactions :
  - Information can flow from sender to receiver, but not vice-versa.

Variants

- Is sender aware that it has sent a message ?
- Does the information flow instantaneously ?
  - Immediate transmission : message delivered instantly
  - Immediate observation : receiver sees sender's current state
  - Delayed transmission : unpredictable delay in delivery
  - Delayed observation : receiver sees an old state of sender
- Can incoming message be queued ?

Overview

Exact characerizations exist :

- Delayed observation : can count up to 2.
- Immediate observation : can count up to any constant.
- Immediate and delayed transmission : characterization exists
  - strictly stronger than observation (can compute mods)
  - strictly weaker than two-way (cannot compute majority)
- Queued transmission is equivalent to two-way interactions.

Plan

Population Protocols

Variants

Population Protocols and Games

Our question

- Can one say that all protocols are games ?
- What is the power of protocols that correspond to games ?

Basic of Game theory

- Two players games : I and II, with a finite set of *pure strategies*, $Strat(I)$ and $Strat(II)$.
- Denote by $A_{i,j}$ (respectively : $B_{i,j}$) the score for player I (resp. II) when I uses strategy $i \in Strat(I)$ and II uses strategy $j \in Strat(II)$.
- The game is termed *symmetric* if A is the transpose of B.
- Famous prisonner's dilemma :

$$\begin{array}{ccc|cc|} & & & \multicolumn{2}{c}{\text{Opponent}} \\ & & & \color{green}\bullet & \color{red}\bullet \\ \cline{4-5} \text{Player} & & \color{green}\bullet & R & S \\ & & \color{red}\bullet & T & P \\ \cline{4-5} \end{array}$$

with $T > R > P > S$ and $2R > T + S$, where $Strat(I) = Strat(II) = \{\color{green}\bullet, \color{red}\bullet\}$.

Best response

- A strategy $x \in Strat(I)$ is said to be a best response to strategy $y \in Strat(II)$, denoted by $x \in BR(y)$ if

$$A_{z,y} \leq A_{x,y} \tag{1}$$

  for all strategies $z \in Strat(I)$.

- In the prisonner's dilemma,

$$BR(\textcolor{green}{\bullet}) = BR(\textcolor{red}{\bullet}) = \textcolor{red}{\bullet},$$

  hend $\textcolor{red}{\bullet}$ is the best rational choice, but $\textcolor{green}{\bullet}$ would be the better social choice.

- We write $\mathbf{x} \in BR_{\neq \mathbf{x}'}(\mathbf{y})$ for

$$A_{z,y} \leq A_{x,y} \tag{2}$$

  for all strategy $\mathbf{z} \in Strat(I), \mathbf{z} \neq \mathbf{x}'$.

Turing a Game into a Dynamic : Pavlovian's behavior

Assume a symmetric two-player game is given. Let Δ be some threshold.

- The protocol associated to the game is a population protocol whose set of states is $Q = Strat(I) = Strat(II)$ and whose transition rules δ are given as follows :

  $$q_1, q_2 \rightarrow q_1', q_2'$$

  where

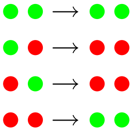
  - $q_1' = q_1$ when $A_{q_1,q_2} \geq \Delta$
  - $q_1' \in BR_{\neq q_1}(q_2)$ when $A_{q_1,q_2} < \Delta$

  and symmetrically.

- A population protocol is *Pavlovian* if it can be obtained from a game as above.

Example : The Prisonner's Dilemma

$T > R > \Delta > P > S$



- Several studies of this dynamic over various graphs : see e.g. [Dyer et al. 02], [Fribourg et al. 04].
- Somehow, our question is : can any dynamic be termed "a game", or "pavlovian".

First observation : Pavlovian => Symmetric

- We say that a population protocol is *symmetric* if, whenever $q_1, q_2 \to q_1', q_2'$ in the program, one has also $q_2, q_1 \to q_2', q_1'$.
- Pavlovian implies symmetric.

Theorem

Any symmetric deterministic 2-states population protocol is Pavlovian.

Symmetric $\neq$ Pavlovian

- Write any rule of the protocol

$$q_1 q_2 \to \delta_1(q_1, q_2)\delta_1(q_2, q_1)$$

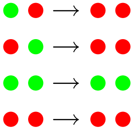
- Consider a 3-states population protocol with set of states $Q = \{\textcolor{green}{\bullet}, \textcolor{red}{\bullet}, \textcolor{blue}{\bullet}\}$ and a joint transition function δ such that $\delta_1(\textcolor{green}{\bullet}, \textcolor{green}{\bullet}) = \textcolor{red}{\bullet}$, $\delta_1(\textcolor{red}{\bullet}, \textcolor{green}{\bullet}) = \textcolor{blue}{\bullet}$, $\delta_1(\textcolor{blue}{\bullet}, \textcolor{green}{\bullet}) = \textcolor{green}{\bullet}$.

- One can not find a matrix of a game that would lead to this dynamic.

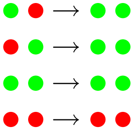
- Corollary : Not all protocols are Pavlovian.

Basic Pavlovian Protocols

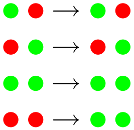
- *OR* is computed by 2-state protocol :



- *AND* is computed by 2-state protocol :



- Remark : *XOR* is not computed by 2-state protocol :

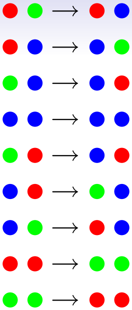


Electing a Leader

- Classical solution : $\textcolor{red}{\bullet}\ \textcolor{red}{\bullet} \rightarrow \textcolor{blue}{\bullet}\ \textcolor{red}{\bullet}$ is not symmetric.
- Proposition : The following Pavlovian protocol solves the leader election problem, as soon as the population is of size ≥ 3.

$$
\begin{array}{c}
\textcolor{red}{\bullet}\ \textcolor{green}{\bullet} \longrightarrow \textcolor{red}{\bullet}\ \textcolor{blue}{\bullet} \\
\textcolor{red}{\bullet}\ \textcolor{blue}{\bullet} \longrightarrow \textcolor{blue}{\bullet}\ \textcolor{green}{\bullet} \\
\textcolor{green}{\bullet}\ \textcolor{blue}{\bullet} \longrightarrow \textcolor{blue}{\bullet}\ \textcolor{red}{\bullet} \\
\textcolor{blue}{\bullet}\ \textcolor{blue}{\bullet} \longrightarrow \textcolor{blue}{\bullet}\ \textcolor{blue}{\bullet} \\
\textcolor{green}{\bullet}\ \textcolor{red}{\bullet} \longrightarrow \textcolor{blue}{\bullet}\ \textcolor{red}{\bullet} \\
\textcolor{blue}{\bullet}\ \textcolor{red}{\bullet} \longrightarrow \textcolor{green}{\bullet}\ \textcolor{blue}{\bullet} \\
\textcolor{blue}{\bullet}\ \textcolor{green}{\bullet} \longrightarrow \textcolor{red}{\bullet}\ \textcolor{blue}{\bullet} \\
\textcolor{red}{\bullet}\ \textcolor{red}{\bullet} \longrightarrow \textcolor{green}{\bullet}\ \textcolor{green}{\bullet} \\
\textcolor{green}{\bullet}\ \textcolor{green}{\bullet} \longrightarrow \textcolor{red}{\bullet}\ \textcolor{red}{\bullet}
\end{array}
$$

- Indeed, ultimately there will be exactly one leader, that is one agent in state $\textcolor{red}{\bullet}$ or $\textcolor{green}{\bullet}$



- Taking $\Delta = 4$, this corresponds to matrix

Opponent

Player

$$\begin{bmatrix} 1 & 4 & 1 \\ 3 & 1 & 1 \\ 2 & 1 & 4 \end{bmatrix}$$

Majority

- The majority problem (given some population of ${\color{red}\bullet}$ and ${\color{blue}\bullet}$, determine whether there are more ${\color{red}\bullet}$ than ${\color{blue}\bullet}$) can be solved by a Pavlovian population protocol.

- Proof :

$$
\begin{aligned}
{\color{black}\bullet}\ {\color{green}\bullet} &\longrightarrow {\color{green}\bullet}\ {\color{green}\bullet}\\
{\color{green}\bullet}\ {\color{black}\bullet} &\longrightarrow {\color{green}\bullet}\ {\color{green}\bullet}\\
{\color{black}\bullet}\ {\color{red}\bullet} &\longrightarrow {\color{green}\bullet}\ {\color{red}\bullet}\\
{\color{red}\bullet}\ {\color{black}\bullet} &\longrightarrow {\color{red}\bullet}\ {\color{green}\bullet}\\
{\color{green}\bullet}\ {\color{blue}\bullet} &\longrightarrow {\color{black}\bullet}\ {\color{blue}\bullet}\\
{\color{blue}\bullet}\ {\color{green}\bullet} &\longrightarrow {\color{blue}\bullet}\ {\color{black}\bullet}\\
{\color{red}\bullet}\ {\color{blue}\bullet} &\longrightarrow {\color{black}\bullet}\ {\color{green}\bullet}\\
{\color{blue}\bullet}\ {\color{red}\bullet} &\longrightarrow {\color{green}\bullet}\ {\color{black}\bullet}
\end{aligned}
$$

This corresponds to the following matrix.

$$
\begin{array}{cc|cccc|}
 & & \multicolumn{4}{c}{\text{Opponent}} \\
 & & \color{black}\bullet & \color{green}\bullet & \color{red}\bullet & \color{blue}\bullet \\
\hline
 & \color{black}\bullet & 3 & 1 & 1 & 3 \\
\text{Player} & \color{green}\bullet & 2 & 3 & 3 & 1 \\
 & \color{red}\bullet & 2 & 2 & 2 & 1 \\
 & \color{blue}\bullet & 2 & 2 & 1 & 2 \\
\hline
\end{array}
$$

Dicussions and conclusions

- We are still far from understanding the power of Pavlovian population protocols.
- Simple protocols become rather complicated (and hard to explain).
- We don't even know how to compute *mod* 2.
- We don't even know how to compute $\geq k$, for a fixed k.
- Nor have a proof that this is not possible.

Some Side Effects of this Study

- Conclusion 1 : Not all distributed algorithms are games ! ! !
- A contribution ? :

Proposition

Any population protocol can be simulated by a symmetric population protocol, as soon as the population is of size ≥ 3.

Corollary

A predicate is computable by a symmetric population protocol if and only if it is semilinear.