PhysarumSoft - a Software Tool for Programming Physarum Machines and Simulating Physarum Games

Andrew Schumann* and Krzysztof Pancerz†

*University of Information Technology and Management in Rzeszów, Poland
Email: andrew.schumann@gmail.com
†University of Management and Administration in Zamosć, Poland
Email: kpancerz@wszia.edu.pl

Abstract—In the paper, we describe selected functionality of the current version of a new software tool, called PhysarumSoft, developed for programming Physarum machines and simulating Physarum games. The tool was designed for the Java platform. A Physarum machine is a biological computing device implemented in the plasmodium of Physarum polycephalum or Badhamia utricularis that are one-cell organisms able to build programmable complex networks. The plasmodial stage of such organisms is a natural transition system that can be used as a medium for solving different computational tasks as well as creating bio-inspired strategy games.

I. INTRODUCTION

A Physarum machine is a programmable amorphous biological computing device, experimentally implemented in the plasmodium of Physarum polycephalum, also called true slime mould [1]. Physarum polycephalum is a single cell organism belonging to the species of order Physarales. In the considered case, the term of Physarum machine covers, in general, a hybrid device implemented in two plasmodia (cf. [2]), namely the plasmodium of Physarum polycephalum as well as the plasmodium of Badhamia utricularis. Badhamia utricularis is also the species of order Physarales. The plasmodium of Physarum polycephalum or Badhamia utricularis, spread by networks, can be programmable. In propagating and foraging behavior of the plasmodium, we can perform useful computational tasks. This ability was firstly discerned by T. Nakagaki et al. [3]. The Physarum machine comprises an amorphous yellowish mass with networks of protoplasmic veins, programmed by spatial configurations of attracting and repelling stimuli. When several attractants are scattered in the plasmodium range, the plasmodium looks for attractants, propagates protoplasmic veins towards them, feeds on them and goes on. Repellents play the role of elements blocking propagation of protoplasmic veins.

Solving computational tasks by means of Physarum machines is one of the main goals of the Physarum Chip Project: Growing Computers from Slime Mould [4], funded by the Seventh Framework Programme (FP7). In this project, we are going to construct an unconventional computer on programmable behaviour of Physarum polycephalum.

To program computational tasks for Physarum machines, we are developing a new object-oriented programming language [5], [6], [7], called a Physarum language. The Physarum language is a prototype-based language [8] consisting of inbuilt sets of prototypes corresponding to both the high-level models used for describing behaviour of Physarum polycephalum (e.g., ladder diagrams, transition systems, timed transition systems, Petri nets) and the low-level model (distribution of stimuli). More information is given in Section II.

Another task that can be performed in Physarum machine environments concerns bio-inspired strategy games. Fundamental topics of the research area related to bio-inspired games on Physarum machines were earlier considered, for example in [2] and [9]. Simulating Physarum games is considered in Section III.

To support research on programming Physarum machines and simulating Physarum games, we are developing a specialized software tool, called the Physarum software system, shortly PhysarumSoft. The tool was designed for the Java platform. In the paper, we describe selected functionality of the current version of PhysarumSoft. A general structure of this system is shown in Figure 1. We can distinguish three main parts of PhysarumSoft:

- **Physarum language compiler.** The Physarum language is an object-oriented high-level programming language.
For generating the compiler of the language, the Java Compiler Compiler (JavaCC) tool [10] was used. JavaCC is the most popular parser generator for use with Java applications. For the programming purpose, a compiler embodied in our tool translates the high-level code describing a model of the Physarum machine into the spatial distribution (configuration) of stimuli (attractants, repellents) controlling propagation of protoplasmic veins of the plasmodium. A grammar of our language was described in [6].

- Module of programming Physarum machines described in Section II.
- Module of simulating Physarum games described in Section III.

The main features of PhysarumSoft are the following:

- Portability. Thanks to the Java technology, the created tool can be run on various software and hardware platforms. In the future, the tool will be adapted for platforms available in mobile devices and as a service in the cloud.
- User-friendly interface (see some screenshots shown in Sections II and III).
- Modularity. The project of PhysarumSoft and its implementation covers modularity. It makes the tool extend in the future easily.

II. PROGRAMMING PHYSARUM MACHINES

To program Physarum machines (i.e., to set the spatial distribution (configuration) of stimuli (attractants, repellents) controlling propagation of protoplasmic veins of the plasmodium), we are developing a new object-oriented programming language [5], [6], [7], called the Physarum language. Our language is based on the prototype-based approach (cf. [8]) that is less common than the class-based one, although, it has a great deal to offer. This approach is also called classless or instance-based programming because prototype-based languages are based upon the idea that objects, representing individuals, can be created without reference to class-defining. In this approach, the objects, that are manipulated at runtime, are prototypes. In our language, there are inbuilt sets of prototypes corresponding to both the high-level models used for describing behaviour of Physarum polycephalum (e.g., ladder diagrams, transition systems, timed transition systems, Petri nets) and the low-level model (distribution of stimuli). According to the prototype-based approach, objects are created by means of a copy operation, called cloning, which is applied to a given prototype. Objects can be instantiated (cloned) via the keyword new using defined constructors. Different methods are used to manipulate features of the objects and create relationships between objects.

In our approach, the starting point, in programming the behaviour of the Physarum machine, is a high-level model describing propagation of protoplasmic veins of Physarum. We have proposed several high-level models used in programming Physarum machines, i.e.:

- ladder diagrams (see [11]),
- transition systems ([5]) and timed transition systems (see [12]),
- Petri nets (see [13]).

In the remaining part of this section, we recall basic definitions concerning transition systems, timed transition systems and Petri nets. They are general purpose tools that can be used to model dynamic systems with distinguished states and transitions between states. Application of ladder diagrams is restricted to modelling digital circuits. The recalled definitions are illustrated with some examples.

Transition systems are a simple and powerful tool for explaining the operational behaviour of models of concurrency. Formally, a transition system is a quadruple \( TS = (S, E, T, I) \), cf. [14], where:

- \( S \) is the non-empty set of states,
- \( E \) is the set of events,
- \( T \subseteq S \times E \times S \) is the transition relation,
- \( I \) is the set of initial states.

Usually transition systems are based on actions which may be viewed as labelled events. If \( (s, e, s') \in T \), then the idea is that \( TS \) can go from \( s \) to \( s' \) as a result of the event \( e \) occurring at \( s \). Any transition system \( TS = (S, E, T, I) \) can be presented in the form of a labelled graph with nodes corresponding to states from \( S \), edges representing the transition relation \( T \), and labels of edges corresponding to events from \( E \).

The behaviour of Physarum machines is often dynamically changed in time. It is assumed, in the transition systems mentioned earlier, that all events happen instantaneously. Therefore, in [12], we proposed to use another high-level model, based on timed transition systems [15]. In the timed transition systems, timing constraints restrict the times at which events may occur. The timing constraints are classified into two categories: lower-bound and upper-bound requirements.

Let \( N \) be a set of nonnegative integers. Formally, a timed transition system \( TTS = (S, E, T, I, I, u) \) consists of:

- an underlying transition system \( TS = (S, E, T, I) \),
- a minimal delay function (a lower bound) \( l : E \to N \) assigning a nonnegative integer to each event,
- a maximal delay function (an upper bound) \( u : E \to N \cup \infty \) assigning a nonnegative integer or infinity to each event.

In Physarum machines, timing constraints can be implemented through activation and deactivation of stimuli (attractants and/or repellents). Each state corresponds to either an original point of the plasmodium or an attractant. Especially, initial states of transition systems can be presented by original points, where protoplasmic veins originate from. Edges represent plasmodium transitions between attractants as well as the original points of the plasmodium.

In case of transition system and timed transition system models, the main prototypes defined in the Physarum language and their selected methods are collected in Table I.

Let us consider an exemplary timed transition system shown in Figure 2. Formally, we have \( TTS = (S, E, T, I, I, u) \), where:

- \( S = \{s_1, s_2, s_3, s_4\} \).
TABLE I
Main prototypes, corresponding to transition system and timed transition system models, defined in the Physarum language, and their selected methods

<table>
<thead>
<tr>
<th>Prototype</th>
<th>Selected methods</th>
</tr>
</thead>
<tbody>
<tr>
<td>TS State</td>
<td>setDescription, setAsInitial</td>
</tr>
<tr>
<td>TS Event</td>
<td>setDescription, setTimingConstraints</td>
</tr>
<tr>
<td>TS Transition</td>
<td></td>
</tr>
</tbody>
</table>

- \( E = \{e_1, e_2, e_3\} \)
- \( T = \{(s_1, e_1, s_2), (s_2, e_2, s_3), (s_3, e_3, s_4)\} \)
- \( I = \{s_1, s_2\} \)
- \( l(e_1) = l(e_2) = l(e_3) = 0 \)
- \( u(e_1) = u(e_2) = \infty, u(e_3) = 3 \)

• Repellent \( R_{-2} \) is placed next to attractant \( A_{-2} \) because timing constraints are set for event \( e_3 \).
• For \( t > 3 \), \( R_{-2} \) must be activated to annihilate the vein of the plasmodium between \( A_{-1} \) and \( A_{-2} \).

![Fig. 2. An exemplary timed transition system TTS](image)

![Fig. 3. The code for the model in the form of TTS](image)

The code, for the model in the form of TTS, in the Physarum language written in the editor of the module of programming Physarum machines, is shown in Figure 3. The result of compilation, i.e., the spatial distribution of stimuli (attractants, repellents) controlling propagation of protoplasmic veins of the plasmodium, is shown in Figure 4.

One can see that:
- **Physarum** \( Ph_{-1} \) represents initial state \( s_1 \), attractants \( A_{-1}, A_{-2}, A_{-3} \) represent states \( s_2, s_4, s_3 \), respectively.
- Splitting the plasmodium at \( A_{-1} \) is supported by repellent \( R_{-1} \).

![Fig. 4. The result of compilation](image)

Petri nets introduced by C.A. Petri [16] are a formal tool used to model discrete event systems. In [13], we proposed to use Petri nets with inhibitor arcs (cf. [17]) to model behaviour of *Physarum polycephalum*. The inhibitor arcs test the absence of tokens in a place and they can be used to disable transitions. This fact can model repellents in *Physarum* machines. A transition can only fire if all its places connected through inhibitor arcs are empty (cf. [18]).

Formally, a marked Petri net with inhibitor arcs is a five-tuple

\[
MPN = (Pl, Tr, Ar, w, m),
\]

where:

- \( Pl \) is the finite set of places (marked graphically with circles),
- \( Tr \) is the finite set of transitions (marked graphically with rectangles),
- \( Ar = Ar_O \cup Ar_I \) such that \( Ar_O \subseteq (Pl \times Tr) \cup (Tr \times Pl) \) is the set of ordinary arcs (marked graphically with arrows) from places to transitions and from transitions to places whereas \( Ar_I \subseteq Pl \times Tr \) is the set of inhibitor arcs (marked graphically with lines ended with small circles) from places to transitions,
- \( w : Ar \rightarrow \{1, 2, 3, \ldots \} \) is the weight function on the arcs,
- \( m : Pl \rightarrow \{0, 1, 2, \ldots \} \) is the initial marking function on the places.

In describing the Petri net behaviour, it is convenient to use for any \( t \in Tr \):

- \( I_O(t) = \{p \in Pl : (p, t) \in Ar_O\} \) - a set of input places connected through ordinary arcs to the transition \( t \),
- \( I_I(t) = \{p \in Pl : (p, t) \in Ar_I\} \) - a set of input places connected through inhibitor arcs to the transition \( t \),
- \( O(t) = \{p \in Pl : (t, p) \in Ar_O\} \) - a set of output places connected through ordinary arcs from the transition \( t \).

In the proposed approach, we have additionally assumed the following limits for the Petri net:
TABLE II
THE MEANING OF TOKENS IN PLACES REPRESENTING CONTROL STIMULI

<table>
<thead>
<tr>
<th>Token</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Present</td>
<td>Stimulus activated</td>
</tr>
<tr>
<td>Absent</td>
<td>Stimulus deactivated</td>
</tr>
</tbody>
</table>

TABLE III
THE MEANING OF TOKENS IN PLACES REPRESENTING OUTPUT STIMULI

<table>
<thead>
<tr>
<th>Token</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Present</td>
<td>Stimulus occupied by plasmodium of Physarum polycephalum</td>
</tr>
<tr>
<td>Absent</td>
<td>Stimulus not occupied by plasmodium of Physarum polycephalum</td>
</tr>
</tbody>
</table>

TABLE IV
MAIN PROTOTYPES, CORRESPONDING TO PETRI NET MODELS, DEFINED IN THE PHYSARUM LANGUAGE, AND THEIR SELECTED METHODS

<table>
<thead>
<tr>
<th>Prototype</th>
<th>Selected methods</th>
</tr>
</thead>
<tbody>
<tr>
<td>PN.Place</td>
<td>setDescription, setRole</td>
</tr>
<tr>
<td>PN.Transition</td>
<td>setDescription</td>
</tr>
<tr>
<td>PN.Arc</td>
<td>setAsInhibitor, setAsBidirectional</td>
</tr>
</tbody>
</table>

- \( w(a) = 1 \) for each \( a \in Ar \),
- \( m(p) \leq 1 \) for each \( p \in Pl \) (the capacity limit).

If \( m(p) = 1 \), then a token (i.e., a black dot) is drawn in the graphical representation of the place \( p \). Assuming limits as the ones above, a transition \( t \in Tr \) is said to be enabled if and only if \( m(p) = 1 \) for all \( p \in I_O(t) \), i.e., the token is present in all input places \( p \) connected with the transition \( t \) through the ordinary arcs, and \( m(p) = 0 \) for all \( p \in I_I(t) \), i.e., the token is absent in all input places \( p \) connected with the transition \( t \) through the inhibitor arcs, and \( m(p) = 0 \) for all \( p \in O(t) \), i.e., the token is absent in all output places \( p \) of the transition \( t \). If the transition \( t \) is enabled, we say that it can fire. A new marking function \( m' : Pl \to \{0, 1, 2, \ldots \} \) defines the next state of the Petri net after firing the transition \( t \):

\[
m'(p) = \begin{cases} 
m(p) - 1 & \text{if } p \in I_O(t) \text{ and } p \notin O(t), \\
m(p) + 1 & \text{if } p \in O(t) \text{ and } p \notin I_O(t), \\
m(p) & \text{otherwise.}
\end{cases}
\]

It is worth noting that in all figures including Petri net models, to simplify them, we have used bidirectional arcs between input places and transitions instead of arcs from input places to transitions and from transitions to input places. A bidirectional arc causes that the token is not consumed (removed) from the input place after firing a transition. This fact has a natural justification, i.e., firing a transition does not cause deactivation of the attractants and disappearance of the plasmodium from the original point. The plasmodium grows to build a dendritic network of veins.

In the proposed Petri net models of Physarum machines, we can distinguish three kinds of places:

- Places representing Physarum polycephalum.
- Places representing control stimuli (repellents).
- Places representing output stimuli (attractants).

In the Physarum language, the kind of a place is determined by the role played by it.

For each kind of places, we adopt different meaning (interpretation) of tokens. The meaning of tokens in places representing Physarum polycephalum is natural, i.e., the token in a given place corresponds to the presence of the plasmodium of Physarum polycephalum in an original point, where it starts to grow. The meaning of tokens in places representing control stimuli is shown in Table II, whereas the meaning of tokens in places representing output stimuli is shown in Table III.

In case of control stimuli, we are interested in whether a given stimulus is activated or not. In case of output stimuli (attractants), we are interested in whether a given attractant is occupied by the plasmodium of Physarum polycephalum. Transitions in Petri net models represent the flow (propagation) of the plasmodium from the original points to attractants as well as between attractants.

In case of Petri net models, the main prototypes defined in the Physarum language and their selected methods are collected in Table IV.

Let us consider an exemplary Petri net shown in Figure 5. Formally, we have \( MPN = (Pl, Tr, Ar, w, m) \), where:

- \( Pl = \{P_1, P_2, P_3, P_4\} \),
- \( Tr = \{T_1, T_2\} \),
- \( Ar = Ar_O \cup Ar_I \), such that
  - \( Ar_O = \{(P_1, T_1), (T_1, P_2), (P_2, T_2), (T_2, P_3), (T_2, P_4)\} \),
  - and \( Ar_I = \emptyset \),
- \( w(a) = 1 \) for all \( a \in Ar \),
- and \( m(p) = 0 \) for each \( p \in Pl \),

![Fig. 5. An exemplary Petri net MPN](image)

The code, for the model in the form of \( MPN \), in the Physarum language written in the editor of the module of programming Physarum machines, is shown in Figure 6.

The result of compilation is shown in Figure 7. One can see that Physarum Ph1 represents place \( P_1 \), attractants \( A_1, A_2, A_3 \) represent places \( P_2, P_3, P_4 \), respectively. Splitting the plasmodium at \( A_2 \) is supported by repellent \( R_1 \).

In [6], we showed how to use high-level models (transition systems and Petri nets) to describe four basic forms of Physarum motions:
• direct - direction, i.e., a movement from one point, where the plasmodium is located, towards another point, where there is a neighbouring attractant,
• fuse - fusion of two plasmodia at the point, where they meet the same attractant,
• split - splitting the plasmodium from one active point into two active points, where two neighbouring attractants with a similar power of intensity are located,
• repel - repelling of the plasmodium or inaction.

It is worth noting that four basic forms are fundamental components used to build or describe more complex systems.

III. SIMULATING PHYSARUM GAMES

In [2], we showed that the slime mould (Physarum polycephalum) is a natural transition system which can be considered a biological model for strategic games. General assumptions for such games were presented both in [2] and [9]. They are based on the experiments performed by A. Adamatsky and M. Grube. If there are only two agents of the plasmodium game, where the first agent is presented by a usual Physarum polycephalum plasmodium and the second agent by its modification, a Badhamia utricularis plasmodium, then both start to compete with each other.

To simulate games on Physarum machines, we are developing a special module of PhysarumSoft called the Physarum game simulator. This module works under the client-server paradigm. A general structure of the Physarum game simulator is shown in Figure 8.

The server-side application of the Physarum game simulator is called PGServer. The main window of PGServer is shown in Figure 9. In this window, the user can:
• select the port number on which the server listens for connections,
• start and stop the server,
• set the game strategy:
  – strategy by stimulus placement,
  – strategy by stimulus activation,
• shadow information about actions undertaken.

The client-side application of the Physarum game simulator is called PGClient. The main window of PGClient is shown in Figures 10 and 12. In this window, the user can:
• set the server IP address and its port number,
• start the participation in the game,
• manipulate stimuli (place or activate them) during the game,
• monitor the current result.

In the Physarum game simulator, we have two players:
• the first one plays for the Physarum polycephalum plasmodia,
the second one plays for the *Badhamia utricularis* plasmodia.

Locations of the original points of both plasmodia are randomly generated. The players can control motions of plasmodia via attracting or repelling stimuli. There are two strategies which can be defined for the game:

1) Locations of attractants and repellents are *a priori* generated in a random way. During the game, each player can activate one stimulus (attractant or repellent) at each step.

2) Locations of attractants and repellents are determined by the players during the game. At each step, each player can put one stimulus (attractant or repellent) at any location and this stimulus becomes automatically activated.

The client-side main window for the first strategy (locations of attractants and repellents are *a priori* generated in a random way) is shown in Figure 10. At the beginning, the original points of *Physarum polycephalum* and *Badhamia utricularis*, as well as stimuli, are scattered randomly on the plane. The window after several player’s movements is shown in Figure 11. A box labelled by $P$ represents an original point of *Physarum polycephalum*. A box labelled by $B$ represents an original point of *Badhamia utricularis*. A single circle denotes an attractant whereas a double circle - repellent. Different background colors of stimuli differentiate between players.

The client-side main window for the second strategy (locations of attractants and repellents are determined by the players during the game) is shown in Figure 12. At the beginning, the original points of *Physarum polycephalum* and *Badhamia utricularis* are scattered randomly on the plane. During the game, players can place stimuli. New veins of plasmodia are created. The window after several player’s movements is shown in Figure 13.

Communication between clients and the server is realized through text messages containing statements of the *Physarum* language. The exemplary code responsible for creation of stimuli has the form:

```java
p1_a1=new Attractant(195,224,1);
p1_a2=new Attractant(541,310,1);
p1_a1=new Attractant(580,92,2);
p2_r1=new Repellent(452,130,2);
p2_r1=new Repellent(659,327,1);
```

The first two parameters of stimulus constructors determine the location whereas the last parameter is the player’s ID.

The server sends to clients information about the current configuration of the *Physarum* machine (localization of the original points of *Physarum polycephalum* and *Badhamia utricularis*, localization of stimuli, as well as a list of edges, corresponding to veins of plasmodia, between active points) through the XML file. The exemplary XML file has the form:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<network>
<elements>
```
The attribute “player” equal to 0 means that elements are created by the system, in this case, the original points of plasmodia (Physarum polycephalum or Badhamia utricularis).

As payoffs for the created bio-inspired games on Physarum machines, we may define a variety of tasks, including simple ones like achieving as many attractants as possible, occupied by the plasmodium of Physarum polycephalum or Badhamia utricularis. It means that a proper neighborhood of ROI of plasmodium cannot be simultaneously occupied by the plasmodia of Badhamia utricularis. During the game, the players can switch between two possible tactics according to the current game configuration. At the end of the game, we determine who wins.

IV. CONCLUSIONS AND FURTHER WORK

In the paper, we have described selected functionality of the current version of a new software tool called PhysarumSoft. This tool is intended for programming Physarum machines and simulating Physarum games. For over the last two years, we have designed a new object-oriented programming language, called the Physarum language, that constitutes the basis for modelling behaviour of Physarum machines. This language is used in PhysarumSoft. In the nearest future, we plan to extend PhysarumSoft to other high-level models, e.g., \( \pi \)-calculus and cellular automata. Moreover, we are developing a new model, based on rough sets [20], to approximate payoffs of strategy games created on Physarum machines.

ACKNOWLEDGMENT

This research is being fulfilled by the support of FP7-ICT-2011-8.

REFERENCES


