DOI: 10.15439/2015F199
ACSIS, Vol. 5

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 Zamość, 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 reserach 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

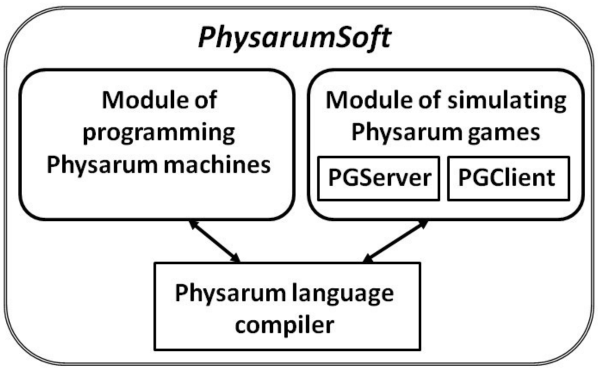

PhysarumSoft
Module of programming Physarum machines
Module of simulating Physarum games
PGServer
PGClient
Physarum language compiler

Fig. 1. A general structure of *PhysarumSoft*

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 $TTS = (S, E, T, I, l, u)$ 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 $TTS = (S, E, T, I, l, u)$, 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

Prototype	Selected methods
TS.State	*setDescription*, *setAsInitial*
TS.Event	*setDescription*, *setTimingConstraints*
TS.Transition	

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

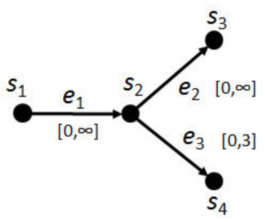
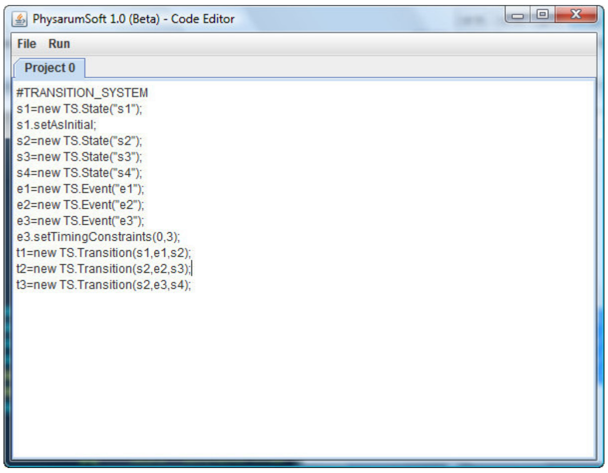


Fig. 2. An exemplary timed transition system TTS



```
#TRANSITION_SYSTEM
s1=new TS.State("s1");
s1.setAsInitial;
s2=new TS.State("s2");
s3=new TS.State("s3");
s4=new TS.State("s4");
e1=new TS.Event("e1");
e2=new TS.Event("e2");
e3=new TS.Event("e3");
e3.setTimingConstraints(0,3);
t1=new TS.Transition(s1,e1,s2);
t2=new TS.Transition(s2,e2,s3);
t3=new TS.Transition(s2,e3,s4);
```

Fig. 3. The code for the model in the form of TTS

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.
- 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.

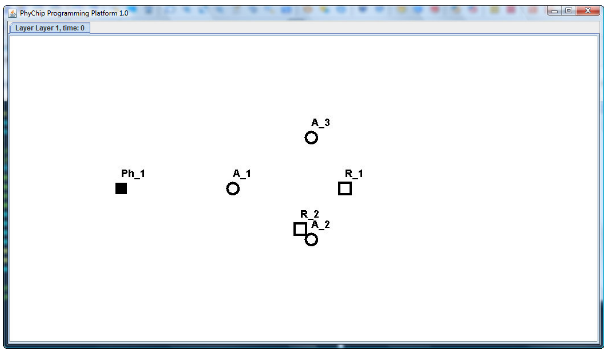


Fig. 4. The result of compilation

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,
- $w : Ar \to \{1, 2, 3, \dots\}$ is the weight function on the arcs,
- $m : Pl \to \{0, 1, 2, \dots\}$ 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

Token	Meaning
Present	Stimulus activated
Absent	Stimulus deactivated

TABLE III
THE MEANING OF TOKENS IN PLACES REPRESENTING OUTPUT STIMULI

Token	Meaning
Present	Stimulus occupied by plasmodium of *Physarum polycephalum*
Absent	Stimulus not occupied by plasmodium of *Physarum polycephalum*

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

Prototype	Selected methods
PN.Place	*setDescription*, *setRole*
PN.Transition	*setDescription*
PN.Arc	*setAsInhibitor*, *setAsBidirectional*

- $w(a) = 1$ 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 $m' : Pl \to \{0, 1, 2, \dots\}$ 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 $p \in Pl$,

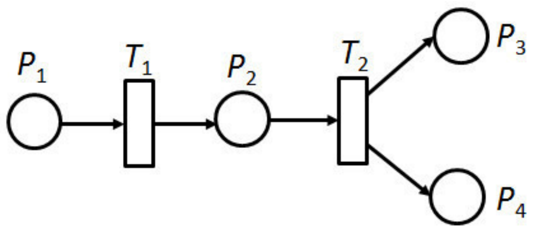
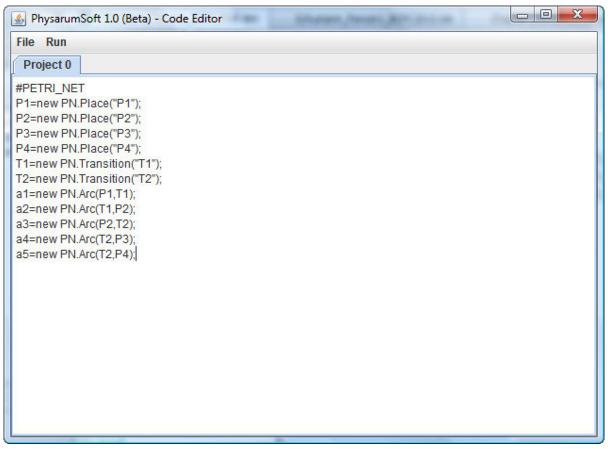


Fig. 5. An exemplary Petri net MPN

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* Ph_1 represents place P_1, attractants A_1, A_2, A_3 represent places P_2, P_4, P_3, 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:



```
#PETRI_NET
P1=new PN.Place("P1");
P2=new PN.Place("P2");
P3=new PN.Place("P3");
P4=new PN.Place("P4");
T1=new PN.Transition("T1");
T2=new PN.Transition("T2");
a1=new PN.Arc(P1,T1);
a2=new PN.Arc(T1,P2);
a3=new PN.Arc(P2,T2);
a4=new PN.Arc(T2,P3);
a5=new PN.Arc(T2,P4);
```

Fig. 6. The code for the model in the form of MPN

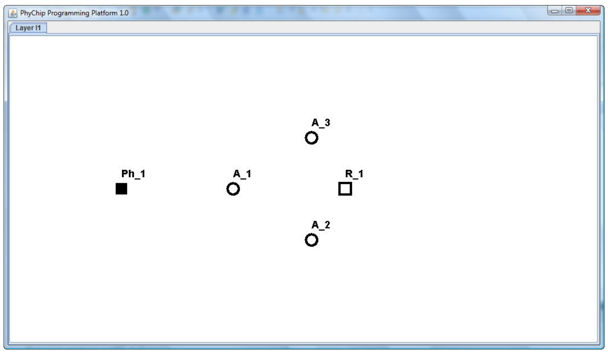


Fig. 7. The code for the model in the form of MPN

- *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.

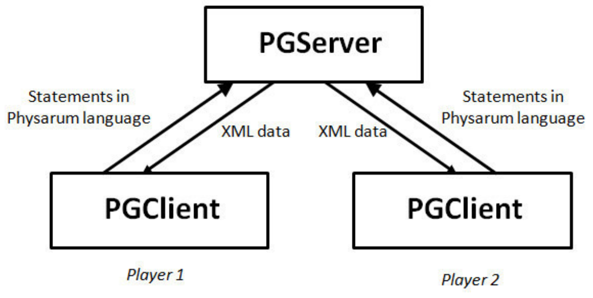
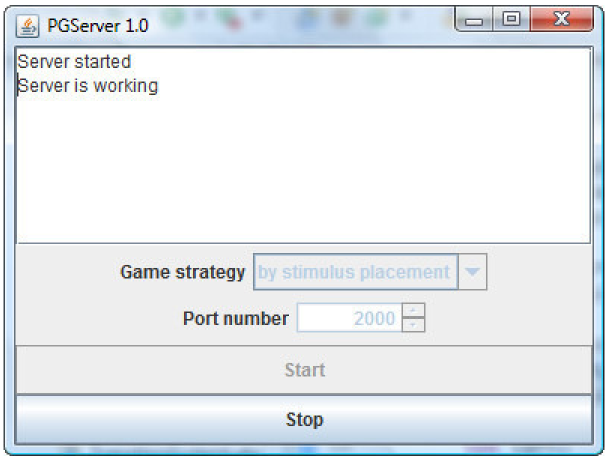

PGServer
Statements in Physarum language
XML data
XML data
Statements in Physarum language
PGClient
PGClient
Player 1
Player 2

Fig. 8. A general structure of the *Physarum* game simulator

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.



```
Server started
Server is working
```

Game strategy: by stimulus placement
Port number: 2000

Fig. 9. The main window of *PGServer*

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

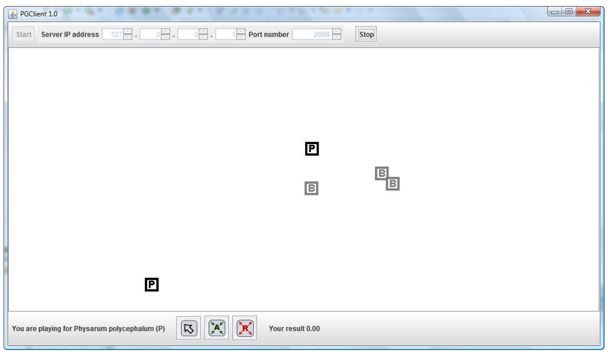


Fig. 10. The main window of *PGClient* for the first strategy

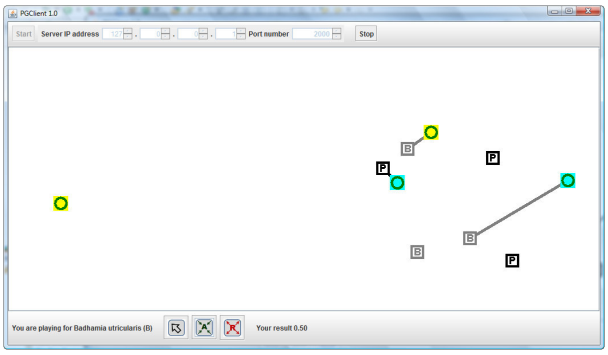


Fig. 11. The main window of *PGClient* for the first strategy after several player's movements

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.

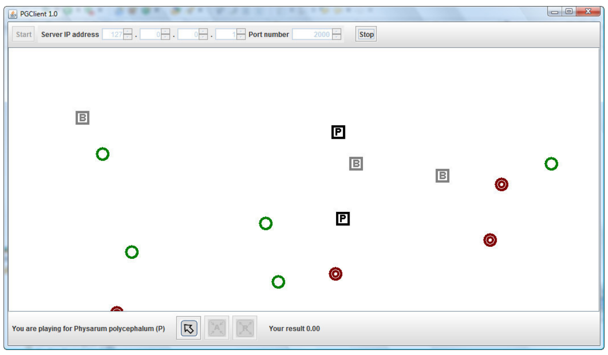


Fig. 12. The main window of *PGClient* for the second strategy

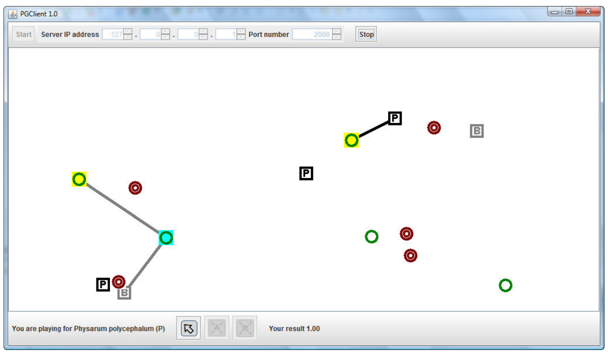


Fig. 13. The main window of *PGClient* for the second strategy after several player's movements

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:

```
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>
```

```
<element id="0" player="0" type="0" x="96" y="310"/>
<element id="1" player="0" type="0" x="766" y="178"/>
<element id="2" player="0" type="0" x="566" y="248"/>
<element id="3" player="0" type="3" x="550" y="53"/>
<element id="4" player="0" type="3" x="374" y="534"/>
<element id="5" player="0" type="3" x="746" y="217"/>
<element id="6" player="1" type="1" x="195" y="224"/>
<element id="7" player="1" type="1" x="541" y="310"/>
<element id="8" player="2" type="1" x="580" y="92"/>
<element id="9" player="2" type="2" x="452" y="130"/>
<element id="10" player="1" type="2" x="659" y="327"/>
</elements>
<veins>
<vein createdBy="0" firstNodeID="2" secondNodeID="7"/>
<vein createdBy="3" firstNodeID="3" secondNodeID="8"/>
</veins>
</network>
```

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 plasmodia of organisms for which we play or constructing the longest path consisting of attractants occupied by plasmodia. Determining different payoffs for *Physarum* games appears to be an interesting field of research due to a huge number of different methodologies and paradigms which can be applied.

The activated attractant A^* causes that the plasmodia propagate protoplasmic veins towards it and feed on it. It means that new transitions are created between the current active points of plasmodia and a new one on the attractant A^*. Propagating protoplasmic veins is possible if the current active points are located in the region of influence (ROI) of A^*. It means that a proper neighborhood of A^* is taken into consideration. From that moment, the activated attractant A^* is occupied by plasmodia. It is worth noting that, as the experiments showed, the attractant occupied by the plasmodium of *Physarum polycephalum* cannot be simultaneously occupied by the plasmodium of *Badhamia utricularis* and vice versa. Moreover, the *Physarum polycephalum* plasmodium grows faster and could grow into branches of *Badhamia utricularis*, while the *Badhamia utricularis* plasmodium could grow over *Physarum polycephalum* veins.

The activated repellent R^* can change the direction of plasmodium motions or can avoid propagating plasmodium protoplasmic veins towards activated attractants. Such influences are possible if plasmodia are in the region of influence (ROI) of R^*.

The control capabilities presented above enable the players to choose, at each step, one of the possible tactics:

1) The attractant or repellent activated by the player can help propagation of his/her plasmodia (of either *Physarum polycephalum* or *Badhamia utricularis*).
2) The attractant or repellent activated by the player can disturb propagation of the second player's plasmodia.

The second possibility is worth considering if we adopt the payoff approximations based on the rough set model described in [19]. It will be the main direction of further investigations. 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* maachines. This language is used in *PhysarumSoft*. In the nearest future, we plan to extend *PhysarumSoft* to other high-level models, e.g., π-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

[1] A. Adamatzky, *Physarum Machines: Computers from Slime Mould.* World Scientific, 2010.

[2] A. Schumann, K. Pancerz, A. Adamatzky, and M. Grube, "Bio-inspired game theory: The case of Physarum polycephalum," in *Proceedings of the 8th International Conference on Bio-inspired Information and Communications Technologies (BICT'2014)*, Boston, Massachusetts, USA, 2014. doi: 10.4108/icst.bict.2014.257869

[3] T. Nakagaki, H. Yamada, and A. Toth, "Maze-solving by an amoeboid organism," *Nature*, vol. 407, pp. 470–470, 2000. doi: 10.1038/35035159

[4] A. Adamatzky, V. Erokhin, M. Grube, T. Schubert, and A. Schumann, "Physarum Chip Project: Growing computers from slime mould," *International Journal of Unconventional Computing*, vol. 8, no. 4, pp. 319–323, 2012.

[5] K. Pancerz and A. Schumann, "Principles of an object-oriented programming language for Physarum polycephalum computing," in *Proceedings of the 10th International Conference on Digital Technologies (DT'2014)*, Zilina, Slovak Republic, 2014. doi: 10.1109/DT.2014.6868725 pp. 273–280.

[6] ——, "Some issues on an object-oriented programming language for Physarum machines," in *Applications of Computational Intelligence in Biomedical Technology*, ser. Studies in Computational Intelligence, R. Bris, J. Majernik, K. Pancerz, and E. Zaitseva, Eds. Springer International Publishing, Switzerland, 2016, vol. 606, pp. 185–199.

[7] A. Schumann and K. Pancerz, "Towards an object-oriented programming language for Physarum polycephalum computing," in *Proceedings of the Workshop on Concurrency, Specification and Programming (CS&P'2013)*, M. Szczuka, L. Czaja, and M. Kacprzak, Eds., Warsaw, Poland, 2013, pp. 389–397.

[8] I. Craig, *Object-Oriented Programming Languages: Interpretation.* London: Springer-Verlag, 2007.

[9] A. Schumann and K. Pancerz, "Interfaces in a game-theoretic setting for controlling the plasmodium motions," in *Proceedings of the 8th International Conference on Bio-inspired Systems and Signal Processing (BIOSIGNALS'2015)*, Lisbon, Portugal, 2015. doi: 10.5220/0005285203380343 pp. 338–343.

[10] JavaCC, http://java.net/projects/javacc/.

[11] A. Schumann, K. Pancerz, and J. Jones, "Towards logic circuits based on physarum polycephalum machines: The ladder diagram approach," in *Proceedings of the International Conference on Biomedical Electronics and Devices (BIODEVICES'2014)*, A. Cliquet Jr., G. Plantier, T. Schultz, A. Fred, and H. Gamboa, Eds., Angers, France, 2014. doi: 10.5220/0004839301650170 pp. 165–170.

[12] A. Schumann and K. Pancerz, "Timed transition system models for programming Physarum machines: Extended abstract," in *Proceedings of the Workshop on Concurrency, Specification and Programming (CS&P'2014)*, L. Popova-Zeugmann, Ed., Chemnitz, Germany, 2014, pp. 180–183.

[13] ——, "Towards an object-oriented programming language for Physarum polycephalum computing: A Petri net model approach," *Fundamenta Informaticae*, vol. 133, no. 2-3, pp. 271–285, 2014. doi: 10.3233/FI-2014-1076

[14] M. Nielsen, G. Rozenberg, and P. Thiagarajan, "Elementary transition systems," *Theoretical Computer Science*, vol. 96, no. 1, pp. 3–33, 1992. doi: 10.1016/0304-3975(92)90180-N

[15] T. A. Henzinger, Z. Manna, and A. Pnueli, "Timed transition systems," in *Real-Time: Theory in Practice*, ser. Lecture Notes in Computer Science, J. de Bakker, C. Huizing, W. de Roever, and G. Rozenberg, Eds. Berlin Heidelberg: Springer, 1992, vol. 600, pp. 226–251.

[16] C. A. Petri, "Kommunikation mit automaten," Institut für Instrumentelle Mathematik, Bonn, Schriften des IIM Nr. 2, 1962.

[17] T. Agerwala and M. Flynn, "Comments on capabilities, limitations and 'correctness' of Petri nets," in *Proceedings of the 1st Annual Symposium on Computer Architecture (ISCA'1973)*, Atlanta, USA, 1973, pp. 81–86.

[18] H. Verbeek, M. Wynn, W. van der Aalst, and A. ter Hofstede, "Reduction rules for reset/inhibitor nets," *Journal of Computer and System Sciences*, vol. 76, no. 2, pp. 125–143, 2010. doi: 10.1016/j.jcss.2009.06.003

[19] K. Pancerz and A. Schumann, "Rough set models of Physarum machines," *International Journal of General Systems*, vol. 44, no. 3, pp. 314–325, 2015. doi: 10.1080/03081079.2014.997529

[20] Z. Pawlak, *Rough Sets. Theoretical Aspects of Reasoning about Data*. Dordrecht: Kluwer Academic Publishers, 1991.