In The Name Of God

Competition Documentation (final phase - 2015)

JAVA CHALLENGE 2015

Previously on JAVA challenge:

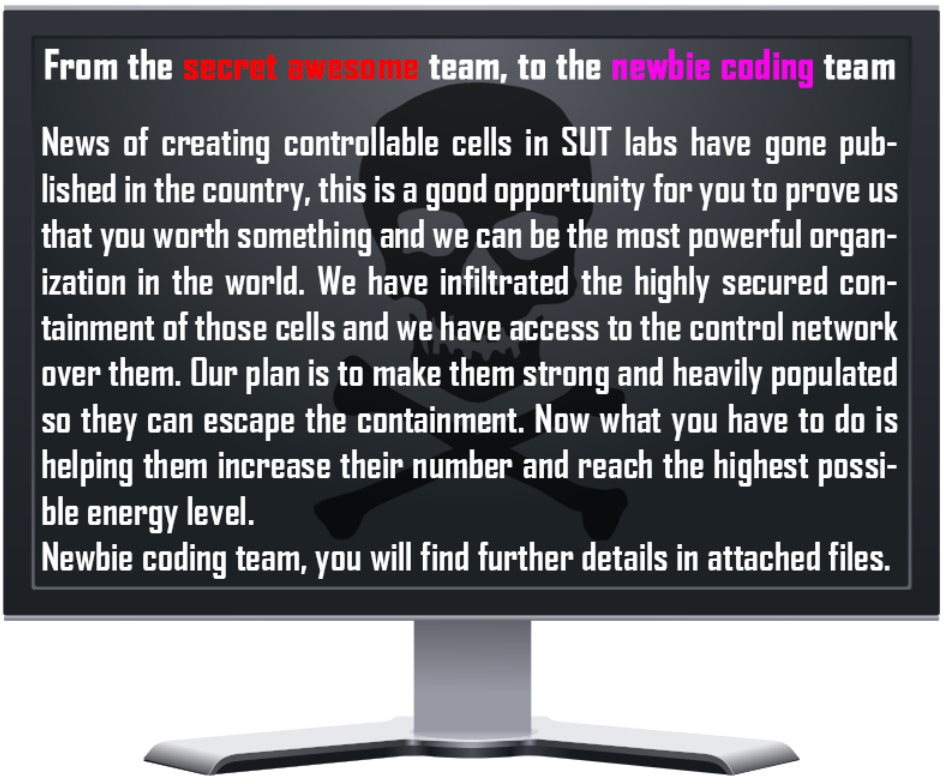
A.N.I NEWS

Feb 3015 SUT

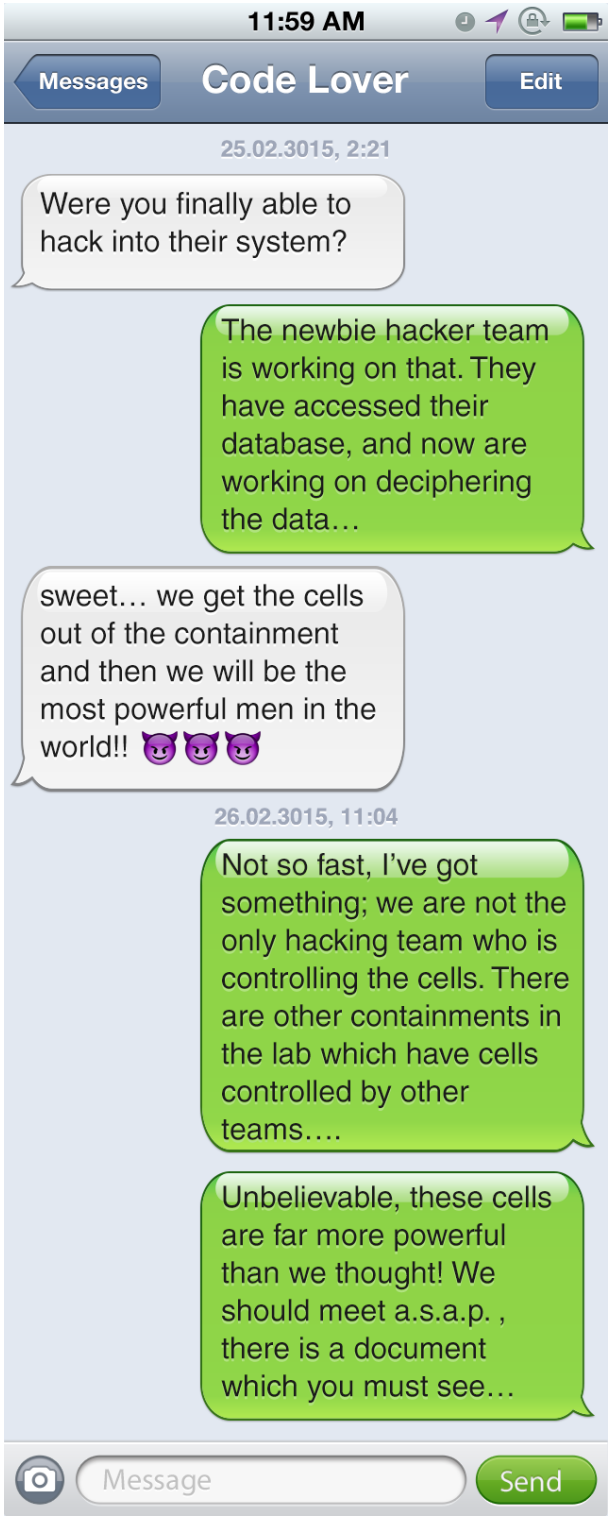
Approximately Natural Intelligence

Today in one of SUT labs, new cells were created using new methods of mutation. These cells were created with the help of a bunch of great minds in A.N.I and some experts in biology. The goal of these experiments was to make a cell whose behaviors were controlled via an A.N.I program.

After a close observation and study on DNA of these cells, scientists realized that they have powerful potentials and they also act so aggressively. So they decided to keep them in highly secured containment. According to the executive manager of the project, the



And now the rest:


11:59 AM
Messages
Code Lover
Edit
25.02.3015, 2:21
Were you finally able to hack into their system?
The newbie hacker team is working on that. They have accessed their database, and now are working on deciphering the data…
sweet… we get the cells out of the containment and then we will be the most powerful men in the world!! 😈😈😈
26.02.3015, 11:04
Not so fast, I've got something; we are not the only hacking team who is controlling the cells. There are other containments in the lab which have cells controlled by other teams….
Unbelievable, these cells are far more powerful than we thought! We should meet a.s.a.p. , there is a document which you must see…
Message
Send

LAB A901 SUT	CONFIDENTIAL DOCUMENT NUMBER 18

Studying the controllable cells has shown their new abilities. In recent experiments we have realized that they can imitate other creature's behaviors. During initial experiments a number of these cells were put in front of a human-droid and they quickly started to mimic its shape and behaviors. Their adaptability to the environment has increased through time, and due to observations, they are evolving too fast in a way that we may not be able to control them anymore.

We will continue to study these cells behavior.

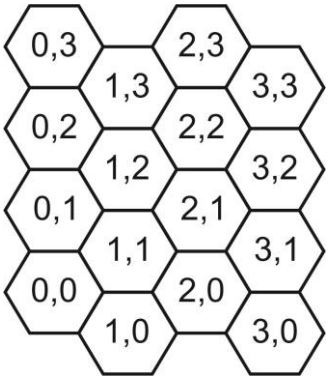
Contents

Introduction

In this phase, the game is going to be held in a competitive manner and in each match there will be 2-4 teams. In each match, each team must try to increase the total energy of its cells. At the end, the ranking will be based upon the total energy of all cells of each team. The game will take place in a map covered with hexagon blocks. At the very beginning, each team has some cells which are located in the blocks on the map, and as the game goes on the cells will act based on the code of the participating teams. This game is turn-based and each cell can perform one legal action on each turn. These legal actions include: moving to an abutting block, gaining resource, undergoing cell Mitosis and attacking other cells. Notice that 'gaining resources' and 'undergoing cell Mitosis' are only allowed in some special blocks on the map. Each cell can only see blocks which are in its field of view and it includes the blocks which are near to its location. At the end, the total energy of all of your cells is calculated as your score. Cells in this phase have the ability to mutate and evolve in mitosis blocks. Further explanations about the details will follow.

Map

The coordinates of the map is as illustrated in the figure. The coordinates of the southwestern point is (0, 0). The first element is for x and the second one is assumed to show y. At the end of this document you can find a map covered with these hexagons, which you can print and use it to understand the map better.



Blocks

Each of the blocks in the map has a height. This height affects the movements of cells, which means that a cell cannot move to a block which is much higher than the block in which it is. There are different types of blocks:

- **None Block**: a block which has not been in field of view of the cell yet, thus its type is not clear.
- **Resource Block**: Resource is stacked in this type of blocks. While a cell is on this type and it is its turn when it decides to gain resource, it gains specific amount of resource. Each resource block

contains particular amount of resource and as the cells gain those, its resource lessens until it finally runs out of resource.
The resource blocks have an additional height beside the regular height which is caused as a result of the stacked resource in them. The final height of a resource block is calculated as shown below:

$$h = min\{H + R/50, 9\}$$

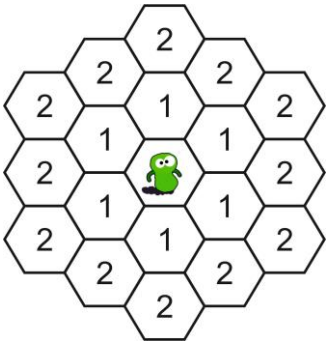
H is the normal height of the block, R is the amount of stacked resource at the block and h is the height of the resource block.

- **Mitosis Block**: the blocks in which cells can undergo cell mitosis (divide itself into two parts). While a cell is in one of mitosis blocks and it decides to undergo mitosis -under certain circumstances- the cell will be divided to two cells and the new cell will move to an abutting block. There are four properties for the cells which have the mutation ability and can undergo mitosis (these properties will be explained in "cells" part). In each of the mitosis blocks, some of these properties may increase which leads to the cell's evolution. Notice that all of these evolvements are applied to the new cell (child) and the old cell (parent) will have its former properties.
- **Impassable Blocks**: it is not possible for the cells to move to these blocks.
- **Normal Blocks**: all the other blocks which has none of the mentioned trait, are assumed normal blocks.

Cells

Each team has some cells, and during the game the team tries to maximize the total energy of them. In this document by using the term 'cell' we always mean cellular creatures, and not the elements that cover the map (we refer to them as "block"). Each cell has an energy level which shows the amount of stored glucose in it. This value is no larger 100 units. The cells have the ability to evolve in mitosis blocks. We will explain the evolvable properties of the cell in the following paragraphs:

- **Cell Gain Rate** determines how fast the cell can gain resources from a resource block. If the amount of available resource at a resource block is less than the gain rate, the cell will gain the whole resource. The gain rate is no smaller than 15 and no larger than 45.
- **Depth of view** specifies how far the cell can see. The minimum of it is 2, which means that the cell can see the blocks which have the distance at most 2 blocks from its residence. The cell can identify type and height of the blocks it sees. It also determines if a block is occupied by another cell or not. In this figure you can see which blocks a cell can see (in this case the depth of view is equal to 2) and also you can see the distance of each block to the cell. The maximum value of the depth of view is 5.



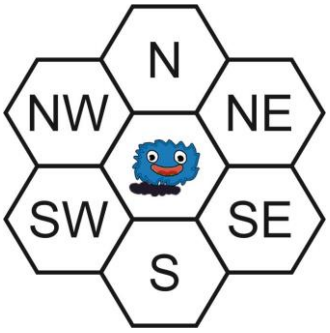
- **Jump height** specifies the maximum of the height difference between the present location of the cell and the destination block which the cell intends to go. The minimum of this property is 2;

meaning that the cell can go to blocks which are at most 2 units higher than its present location. Jump height is 5 units at its maximum.

- **Damage** is the amount of the energy that the enemy cell will lose in an attack, which can be at most 20; meaning that in an attack to an enemy cell, the attacked cell will lose 20 units of its energy. Maximum value for damage is 35.

When it is your turn to play, you can command each cell to perform an act, and the cell would obey only if the situation is appropriate to do so. If you give more than one command to a cell, the final command would be considered for execution. The possible commands are explained in the following paragraphs.

- **Move**: This command needs a direction in order to determine which of the abutting blocks the destination is. There are 6 directions: north, north-east, south-east, south, south-west, north-west. In the figure the directions with the corresponding blocks are shown.
  Here are some explanations on how this works:



  - A cell can only move to a block if the block is not impassable, the block is not occupied by another cell, the height difference between the residence block and the destination block is not greater than the jump height (in case the destination block has a greater height than the residence block), height of the destination block is equal or less than the residence block (note that the cell can only move within the blocks of the map and cannot exceed the map or exit it).
  - If 2 cells simultaneously want to enter one block, in the next turn of the game, one of them (randomly) is placed in the destination block and the other cell will stay still.
  - if a cell tries to move to a block which is going to be emptied at this turn of the game (which means that the cell in the destination block is going to leave the block in that very turn), then the action would take place, contrary to the preliminary phase. There's even a chance that two cells in two abutting block swap their locations.
- **Mitosis**: if the cell commands to undergo mitosis, under certain circumstances, which are explained below, the cell would be divided to two cells and the new cell (child) would absorb the properties which the mitosis block would add.

  - Calling mitosis command is only allowed in mitosis blocks and if a cell in a non-mitosis block sends the mitosis command, undergoing mitosis will not happen.
  - Undergoing mitosis is only possible if in time of sending the mitosis command, the child can move to an abutting block (the conditions of the moving act is the same as before). The child will be located randomly on an abutting passable block.
  - The energy level of the cell must be above 80 in order to undergo mitosis, otherwise the mitosis command would be ignored.
  - After the mitosis, both child and parent have the energy level equal to 40.
  - Notice that mitosis has higher priority than the move act. If a cell has sent a mitosis command and the randomly chosen block for the location of the child, is also the destination of another

cell which had sent a move command, the move command will be ignored and the mitosis command will be executed.

- **Gaining Resource**: If a cell is in a resource block and it sends the gather resource command, a specific amount of energy is gained by the cell and the block will lose precisely the same amount of energy. In case that the energy stacked in the block is greater or equal to the cell gain rate, the cell will gain an amount of energy equal to its gain rate, otherwise the whole energy of the block would be absorbed by the cell.
- **Attack**: like the 'move' command, the 'attack' command needs a direction. When a cell attacks a block which has a cell in it, the attacked cell will have an energy loss equal to the damage of the attacker cell, and if the attacked cell's energy is less than the damage, then the attacked cell will die. Notice that "friendly fire" is on, meaning that you could possibly damage your own cells.

'Mitosis' has the highest priority among the commands and 'Move' has the lowest one. The order of priorities are Mitosis, Attack, Gaining Resource and Move.

Hints about the game and judging the final phase

Here are some hints and points about the game and the competition

Game:

- Each game has 500 turns.
- The map size is at maximum 60×40.
- Score for each team is the total sum of its cells' energy.
- The procedure of competition and qualifications and will be announced.
- In order to run the game, two clients must be executed. For the second client, there is a 'jar' file available for you. Notice that this client is just the default random algorithm.
- The maps used in the previous phase (preliminary phase) are not usable for this phase. You have to use the new ones.
- The arrow keys on the keyboard are used for rotating and magnifying graphics. The {W, A, S, D} keys are for adjusting the camera and using the {R} key you can reset the camera to the default mode.
- The maps which the competition would be held on will definitely be different with the ones which have been released as samples.
- Most of the matches in this phase would be held with four teams.
- There is no specific rule for the location of the different types of blocks and each type may be in any place on the map.

The maps which will be used for judging would be the same for all teams.

Execution restrictions:

- You have 1GHz processing resource for you code.
- There will be 500MB of storage available for your code.

Installing the Prerequisites

In order to begin coding and executing the game, you have to install the prerequisites:

1. Java Development Kit 1.8.0
2. Python 3
3. C++ 11
4. C++ Boost 1.55

case '1' is necessary to run the game, case '2' is needed in case you want to use 'python' to code, and the cases '3' and '4' are only essential if you want to use C++. Notice that C++ is only supported on the Linux OS. The installation procedure for each of the Prerequisites will be explained in the following parts.

1. Java Development Kit 1.8.0
   In order to install this, you can use the link below and download the compatible version with your OS.
   http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html

   You can find the proper help based on your OS in the links below:

Windows	http://www.wikihow.com/Install-the-Java-Software-Development-Kit
Linux	http://www.wikihow.com/Install-Oracle-Java-JDK-on-Ubuntu-Linux
Mac-OS	http://www.wikihow.com/Install-the-JDK-(Java-Development-Kit)-on-Mac-OS-X

2. Python 3
   Installing 'Python 3' is only needed for the teams who want to use python client. Use one of the links below, based on you OS, to download the proper version.

Windows	https://www.python.org/downloads/windows/
Linux	https://www.python.org/downloads/source/
Mac-OS	https://www.python.org/downloads/mac-osx/

   You can find an installation manual in the link below:
   www.wikihow.com/Install-Python

3. C++11 Compiler
   The teams who want to use C++ client must have a compiler able to compile C++11 codes. This client will be compiled on the Linux OS.

4. C++ Boost 1.55
   The teams willing to use C++ client should also have Boost library version 1.55 installed on their systems. This library is available at the link below:
   http://www.boost.org/users/history/version_1_55_0.html

Manual on Coding Interface

You have a class called ‘AI’ at your possession. In this class there is a function called ‘doTurn’. In each turn this function is called once. You have to write the commands which you want to send to your cells, in this function. So you have to write this function in a way that your cells act intelligently. Each turn of the game will last one second, which you have only the first 400 milliseconds to send commands to your cells. The ‘doTurn’ function receives an object from the ‘World’ class as an input. This object describes the present situation of the world, and contains all the information which is accessible to you. You can see the general form of this class, in different coding languages, below. Teams who use ‘python’ must notice that all the needed data are in ‘Model.py’ file. For more information you can refer to the parts below.

Java

```
class AI {
  doTurn(World world) {
    // AI code comes here
  }
}
```

C++

```
class AI {
  void doTurn(World*
world) {
    // AI code comes here
  }
};
```

Python

```
class AI():
  def doTurn(world):
    // AI code comes here
```

World Class:

This class contains all the needed information to describe the present situation of the game. Notice that in each turn of the game this data changes and if you want to access the previous data, you should store it yourself. The proper access to the data will be explained in the following parts.

Java	C++	Python
`String[] getTeams()`	`std::vector<std::string> getTeams()`	`World.teams`

Teams:
Accessing the name of the teams participating in the game as an array of strings:

Java	C++	Python
`String[] getTeams()`	`std::vector<std::string> getTeams()`	`World.teams`

My Id:
Accessing my team number:

Java	C++	Python
`int getMyId()`	`int getMyId()`	`World.my_id`

My Name:
Accessing my team name:

Java	C++	Python
`String getMyName()`	`std::string getMyName()`	`World.my_name`

Map Size:
Accessing the size of the map in a form of and ‘MapSize’ object:

Java	C++	Python
`MapSize getMapSize()`	`MapSize getMapSize()`	`World.map_size`

Map:

Accessing the present map of the game and its information:

Java	C++	Python
`Map getMap()`	`Map* getMap()`	`World.map`

Turn:

Accessing the current turn number in the game:

Java	C++	Python
`int getTurn()`	`int getTurn()`	`World.turn`

All Cells:

Accessing all the available cells in the field of view of your own cells in a list of cells form:

Java	C++	Python
`ArrayList<Cell> getAllCells()`	`std::vector<Cell*> getAllCells()`	`World.all_cells`

My Cells:

Accessing all the cells in your team in a list of cells form:

Java	C++	Python
`ArrayList<Cell> getMyCells()`	`std::vector<Cell*> getMyCells()`	`World.my_cells`

Enemy Cells:

Accessing the enemy's cells in a list of cells format:

Java	C++	Python
`ArrayList<Cell> getEnemyCells()`	`std::vector<Cell*> getEnemyCells()`	`World.enemy_cells`

Map Size:

'MapSize' object contains length and width of the map. It can be easily accessed using the patterns below:

Java	C++	Python
`int getHeight()` `int getWidth()`	`int getHeight()` `int getWidth()`	`MapSize["height"]` `MapSize["width"]`

Map Class:

This class holds the information and functions related to the blocks of the map:

At:
This function returns a block which is located at the input 'Position' or input (x, y), as described below:

Java	C++	Python
`Block at(Position)` `Block at(int x, int y)`	`Block* at(Position)` `Block* at(int x, int y)`	`at(pos)`

Cell Class:

Each object of this class is one of the available cells and has the following functions and information on it.

Id:
Accessing the cell number:

Java	C++	Python
`int getId()`	`int getId()`	`Cell.id`

Position:
Accessing the cell location in the form of an object of 'Position' class:

Java	C++	Python
`Position getPos()`	`Position getPos()`	`Cell.pos`

Team Id:
Accessing the cell's team number:

Java	C++	Python
`int getTeamId()`	`int getTeamId()`	`Cell.team_id`

Energy:
Accessing the current cell's energy level:

Java	C++	Python
`int getEnergy()`	`int getEnergy()`	`Cell.energy`

Depth of Field:
Accessing to the depth of field value for the cell (if you attempt to get this for an enemy cell, the returning value would be something irrelevant from the real value):

Java	C++	Python
`int getDepthOfField()`	`int getDepthOfField()`	`Cell.depth_of_field`

Jump Height:
Accessing to the jump height value for the cell (if you attempt to get this for an enemy cell, the returning value would be something irrelevant from the real value):

Java	C++	Python
`int getJump()`	`int getJump()`	`Cell.jump`

Gain Rate:
Accessing to the gain rate value for the cell (if you attempt to get this for an enemy cell, the returning value would be something irrelevant from the real value):

Java	C++	Python
`int getGainRate()`	`int getGainRate()`	`Cell.gain_rate`

Damage:
Accessing to the damage value for the cell (if you attempt to get this for an enemy cell, the returning value would be something irrelevant from the real value):

Java	C++	Python
`int getAttack()`	`int getAttackValue()`	`Cell.attack_value`

Move:
In order to command a cell to move you can use one of the six directions available in 'Direction' class and send this function to a cell.

Java	C++	Python
`void move(Direction)`	`void move(Direction)`	`move(direction)`

Gain Resource:
Using this function you can command a cell to gain resource.

Java	C++	Python
`void gainResource()`	`void gainResource()`	`gain_resource()`

Mitosis:
Using this function you can command a cell to undergo mitosis.

Java	C++	Python
`void mitosis()`	`void mitosis()`	`mitosis()`

Attack:
In order to command a cell to attack, you have to use one of the directions in available in 'Direction' class, and send the function to the cell.

Java	C++	Python
`void attack(Direction)`	`void attack(Direction)`	`attack(direction)`

Direction:

It is an enumerator (in Java and C++) or it is a class (as in 'Python') which has the six available directions in it. These directions are accessible as it is described in the codes below.

Java	C++	Python
Direction.NORTH Direction.NORTH_EAST Direction.SOUTH_EAST Direction.SOUTH Direction.SOUTH_WEST Direction.NORTH_WEST	Direction::NORTH Direction::NORTH_EAST Direction::SOUTH_EAST Direction::SOUTH Direction::SOUTH_WEST Direction::NORTH_WEST	Constants.Direction.NORTH Constants.Direction.NORTH_EAST Constants.Direction.SOUTH_EAST Constants.Direction.SOUTH Constants.Direction.SOUTH_WEST Constants.Direction.NORTH_WEST

Position:

The coordinates of the related position and all the possible positions which are passable from a block, can be accessed through this function.

Next Position:
It provides the position of an abutting block in an specific given direction.

Java	C++	Python
Position getNextPos(Direction)	Position getNextPos(Direction)	get_next_pos(direction, position)

X:
accessing to the X property of the position:

Java	C++	Python
int getX()	int getX()	position["x"]

Y:
accessing to the Y property of the position:

Java	C++	Python
int getY()	int getY()	position["y"]

Block Class:

This class contains all the information about a block.

Position:
Use this function in order to find the position of a particular block.

Java	C++	Python
`Position getPos()`	`Position getPos()`	`Block.pos`

Minimum Height:
To find the initial height of a block, you can call this function. Notice that the initial height and the height of a resource block may be different.

Java	C++	Python
`int getMinHeight()`	`int getMinHeight()`	`Block.min_height`

Height:
To find the height of a block you can call this function, but notice that the height of a block may change if it is a resource block.

Java	C++	Python
`int getHeight()`	`int getHeight()`	`Block.height()`

Resource:
To find the amount of resource stacked in a resource block, you can use this function. This function returns zero value for the non-resource blocks.

Java	C++	Python
`int getResource()`	`int getResource()`	`Block.resource`

Type:
This function determines the type of a block.

Java	C++	Python
`String getType()`	`std::string getType()`	`Block.type`

Gain Rate Improvement Amount:
Returns the gain rate improvement amount of a mitosis block. Notice that this is only for mitosis blocks.

Java	C++	Python
`int getGainImprovementAmount()`	`int getGainImprovementAmount()`	`Block.gain_improvement_amount`

Depth of Field Improvement Amount:
Returns the amount of improvement for the depth of field in a mitosis block. Only works for mitosis blocks.

Java	C++	Python
`int getDepthOfFieldImprovementAmount()`	`int getDepthOfFieldImprovementAmount()`	`Block.depth_of_field_improvement_amount`

Jump Improvement Amount:
Returns the amount of jump improvement in a mitosis block. Only works for mitosis blocks.

Java	C++	Python
`int getJumpImprovementAmount()`	`int getJumpImprovementAmount()`	`Block.jump_improvement_amount`

Improvement Amount:
Return the amount of improvement for the damage in a mitosis block. Only works for mitosis blocks.

Java	C++	Python
`int getAttackImprovementAmount()`	`int getAttackImprovementAmount()`	`Block.attack_improvement_amount`

Constants Class:

In this class the constant values and strings which contain the type of all blocks are stored.
None Block:
A string with contents of 'none'

Java	C++	Python
`String BLOCK_TYPE_NONE`	`std::string BLOCK_TYPE_NONE`	`Constants.BLOCK_TYPE_NONE`

Normal Block:
A string with contents of 'normal'

Java	C++	Python
`String BLOCK_TYPE_NORMAL`	`std::string BLOCK_TYPE_NORMAL`	`Constants.BLOCK_TYPE_NORMAL`

Mitosis Block:
A string with contents of 'mitosis'

Java	C++	Python
`String BLOCK_TYPE_MITOSIS`	`std::string BLOCK_TYPE_MITOSIS`	`Constants.BLOCK_TYPE_MITOSIS`

Resource Block:
A string with contents of 'resource'

Java	C++	Python
`String BLOCK_TYPE_RESOURCE`	`std::string BLOCK_TYPE_ RESOURCE`	`Constants.BLOCK_TYPE_ RESOURCE`

Impassable Block:
A string with contents of 'impassable'

Java	C++	Python
`String BLOCK_TYPE_IMPASSABLE`	`std::string BLOCK_TYPE_ IMPASSABLE`	`Constants.BLOCK_TYPE_ IMPASSABLE`

Min Energy for Mitosis:
Minimum value for undergoing mitosis (80)

Java	C++	Python
`int CELL_MIN_ENERGY_FOR _MITOSIS`	`int CELL_MIN_ENERGY_FOR _MITOSIS`	`Constants.CELL_MIN_ENERGY_F OR_MITOSIS`

Max Energy:
Maximum value of a cell's energy (100)

Java	C++	Python
`int CELL_MAX_ENERGY`	`int CELL_MAX_ENERGY`	`Constants.CELL_MAX_ENERGY`

Gain Rate:
Amount of resource which a cell can gain (15 at its minimum)

Java	C++	Python
`int CELL_GAIN_RATE`	`int CELL_GAIN_RATE`	`Constants.CELL_GAIN_RATE`

Depth of Field:
The minimum value of depth of field of a cell (2)

Java	C++	Python
`int CELL_DEPTH_OF_FIELD`	`int CELL_DEPTH_OF_FIELD`	`Constants.CELL_DEPTH_OF_FIELD`

Initialization

In order to begin coding and execute your codes, you have to run ‘JavaChallenge2015’ which is in the compressed file ‘NoobFriendly’ which will be given to you. This program actually runs the server program. Any information you may need can be found in appendix1. The other appendixes are about how to start coding for the client program. If you had any problems, use them to find the solution.

Hints and Suggestions

- You can only use one programming language (C++, JAVA or Python) and you cannot submit code for several languages.
- In case of having any questions or problems, you can use the Q&A page available at http://javachallenge.ir/qa/ and ask your questions.
- Use tags in the Q&A page on the programming language (C++, JAVA, Python) and your OS (Mac OS X, Linux, Windows).
- Make sure that your question has not been asked yet and avoid submitting repeated questions.
- The procedure of submitting codes will be announced.
- To access the internet, a LAN connection is given to each team and they can use Sharif ID VPN to connect to the internet. To do so, you can use the manual available at https://id.sharif.ir . Use the given username and password.

Appendix1: NoobFreindly

The game which you will encounter consists of two parts: Server and Client. The game will be executed over the server. In the files provided for you there is file named 'Server' which you can use to test your codes before submitting them.

In order to run the game you must do the procedure below:

Extract the 'NoobFriendly' file. Inside it depending on your OS there is a file named 'JavaChallenge2015.bat' for Win OS, 'JavaChallenge2015.sh' for Linux and Mac OS. Run this file. If you ran to 'Permission denied' error in Linux , use the command below in order to get rid of it.

chmod -R 777 *

On the OSs mentioned below, if you encountered 'libudev.so.0 file not found', go to NOOBFRIENDLY_PATH\dest\Terminal_UI\node-webkit-v0.11.6-linux-x64 folder and run 'buf-fix.sh' file. (this is only need to be done once)

- Ubuntu 13.04+
- Fedora 18+
- Arch
- Gentoo
- Any deviation of each of the mentioned above

Whilst running 'JavaChallenge2015' if you encountered 'connection failed' error, and the problem was not solved by clicking 'try again', then the problem must be one of these:

- JDK 1.8 is not properly installed or is not correctly added to the path. In order to install it properly, use the manual available at page 7 of the doc.
- One of the ports which server uses is open. The simplest way to investigate this problem is to restart the computer. If the problem was solved, then make sure that you exit server and the client completely after running a game.

If your problem was still unsolved, ask it in the Q&A page.

After running 'JavaChallenge2015' you will see a graphical interface. There click on 'choose map' icon and choose a map (there are some maps in the folder NOOBFRIENDLY_PATH\dest\map which you can use in order to test your codes).

Now click on 'New game'. Server will wait for you to run your code. The helping manual to begin coding in each programing language are in other appendixes (run your code and don't be worried about its connection to the server, we will handle this.

After running your code, click on 'Start Game' and enjoy the game.

Do these modifications in NoobFriendly:

Mitosis.jar in directory NoobFriendly [OS_NAME]/dest/ must be replaced.

File 'JGTerminal.nw' at directory NoobFriendly_[OS_NAME]/dest/Terminal_UI/ must be replaced.

'resources' folder must be replaced at the directory NoobFriendly_[OS_NAME]/dest/ .

Appendix2: How to begin in Java

Coding style:

- You must place your AI code in ‘do_Turn’ function in ‘AI.java’ file.
- You can add files and modify all files in ‘client’ package except ‘Main.java’.

How to run:

1. In case you have installed JDK 1.8 but you don't have access to it via eclipse, take a look at appendix5.
2. Import ‘MitosisClient’ into your IDE. To do so, in ‘File’ menu choose ‘import’ option. Then in the popped up window, in General folder choose ‘Existing Projects into Workspaces’ option and click ‘Next’. In the new window, choose ‘Select archive file’ and enter the directory for ‘MitosisClient.zip’. Click on ‘Finish’. Now we have to import the used libraries to the project. On the left-side of eclipse at ‘Package Explorer’ in ‘libs’ folder right-click on ‘gson-2.3.1.jar’ and in menu ‘Build Path’, choose ‘Add to Build Path’.
3. Make sure you have imported that file properly by building it once. In case of proper build you yet may see some Exceptions with ‘Connection Refused’ errors or ‘error while connecting server’ errors, which are normal as the server is not up.

Submitting files:

For submitting your codes, compress your ‘client’ package with ‘zip’ format and change its name to ‘client.zip’. ‘client.zip’ is the file which you upload.

Appendix3: How to begin in C++

Coding style:

- You must place your AI code in ‘do_Turn’ function in ‘AI.cpp’ file.
- You can add files and modify all files for ‘client’ except ‘main.cpp‘.

How to run:

To install C++ client in Linux, you have to install these packages by entering these commands in terminal.

Compiler C++11:	sudo apt-get install build-essentials
Eclipse C++:	sudo apt-get install eclipse-cdt
Boost library:	sudo apt-get install libboost-all-dev

To run the Client C++ project you have to unzip the attached file with name ‘Cpp_Client.zip’ in eclipse’s workspace. Then in eclipse choose ‘Import Project’ and in ‘General’ part choose ‘Existing projects into Workspace’. Then at ‘Select Root Directory’ select the folder which you have unzipped at ‘workspace’ (JC) and then click ‘Finish’.

Now you can compile the project you have created. You may see some Exceptions with ‘Connection Refused’ errors or ‘error while connecting server’, which are normal as the server is not up.

For teams who use C++, there exists two ways to submit their codes:

1.

Creating ‘Makefile’ for the project and submitting it along with the codes. The name of the output file which your Makefile will generate should be ‘out’.

2.

Export your project via eclipse using ‘General->Archive File’. In this case you have to choose codes, ‘.project’ type file and file with ‘.cproject’ postfix to be exported and there is no need for the rest.

Submitting files:

For submitting your codes, compress all files related to ‘client’ alongside with ‘Makefile’ with ‘zip’ format and change its name to ‘client.zip’. ‘client.zip’ is the file which you upload.

Appendix4: How to begin in Python

Coding style:

- You must place your AI code in ‘do_Turn’ function in ‘AI.py file.
- You can add files and modify all files except ‘Controller.py’.

How to run:

First use appendix1 to run the server, and then according to your os, run ‘client’ with one of the methods explained below:

- Windows: run ‘Controller.py’ file. In case you have any problem executing it, and have installed Python 2, you can right-click on the file and in ‘open with’ section, choose ‘python.exe’ from the directory where you have installed ‘Python 3’.
- Linux: open terminal and enter ‘python-client’ folder. Then use these command to run the code.

Python3 Controller.py

Wait until you see the ‘connected to server’ message. If you got error, make sure that the server is already running. Now press start button in JavaChallenge2015.

Submitting files:

For submitting your codes, compress all files related to ‘client’ with ‘zip’ format and change its name to ‘client.zip’. ‘client.zip’ is the file which you upload.

Appendix5: Importing JDK1.8 to Eclipse

If you are using ‘Luna’ version of eclipse then you don't need this manual, but if you use ‘Kepler’ version then it might be useful. Other versions of eclipse don't support JDK 1.8, and if you are working with them, you must download one of the aforementioned versions.

First choose help, then select submenu ‘Eclipse Marketplace’. In the field ‘Find’ enter ‘java 8’ and then install ‘java 8 support for Eclipse’. Restart your Eclipse then.

Right-click on the project and select ‘properties’. In the menu on the left choose ‘java compiler’ and change ‘compliance level’ to 1.8.

Now click on ‘java build path’ on the menu on the left side of ‘properties’ and if ‘jre 1.8’ had any problems, click on it and choose ‘edit’, select ’installed JREs’. in case you couldn't find ‘jre 1.8’ click on ‘add’, after choosing ‘Standard VM’ click next, and afterward enter the address of ‘jre 1.8’ on the directory part, and then click ‘Finish’.