Rules and tasks description

Gliwice, April 9 – 10, 2016

Contents

1. Editorial

Welcome to the finals of the eighth edition of the programming marathon Deadline24. As in the previous years, the contest is organized by Future Processing and supported by four universities and the Ministry of Digital Affairs. For the first time, however, we are in Muzeum Śląskie in Katowice. Its collection refers to the engineering traditions and postindustrial heritage of the Silesia region.

As usual, we hand to you three tasks, the plot of which has been constructed around the world of extraordinary creatures – beetlejumpers. This year, you will be helping them acquire, maintain and breed a local type of cattle (*Wild wild space*), and discover the secrets of beetlejumping intergalactic trade (*Interstellar overdrive*). Also, you will influence the training of the future officers (*DESK*). What will be the result of your interference in the world of the beetlejumpers? We will find out soon.

We hope that also this time the fight for the podium will be tremendous and thrilling, and that all of the contestants will stay excited for 24 contest hours. Good luck!

Deadline24 Team

2. Server communication

Receiving the current information about the virtual world and issuing commands is performed with TCP/IP protocol. Each team connects as a client to a proper competition server. The IP address and port which you use to connect are specified in *Servers* section of the tasks specification. Multiple connections can be established simultaneously, however, the total transfer for each computer is limited. The maximum number of connections and bandwidth limitation are provided in *Technical Arrangements*. Communication is conducted in text mode. A login and a password are required immediately after establishing a connection. Then, the session switches to command mode.

2.1. Logging in

Right after establishing a connection, the server sends a login request terminated with the end of line character: `LOGIN`. You should send your login first, and then the end of the line character. Once it is done, the server will ask for a password (`PASS`). You should send your password in a similar manner as the login. If the authorization succeeds, the server will respond with the `OK` message and will be then ready to receive commands. If the authorization fails, then you will receive the `FAILED 1 bad login or password` message and after that the connection will be closed.

Below you can find an exemplary record of the communication when logging in.

client → server	server → client
	`LOGIN`
`login1`	
	`PASS`
`secret`	
	`OK`

2.2. Commands

Each command consists of a command name, arguments (the number of which depends on the command type) and the end of line character. Parameters should be separated with at least one whitespace.

Server responds to each command with one of the following character strings:

- '`OK`' — if a command is accepted,
- '`FAILED` e msg' — in case of error; where e is the error code, and msg — error message.

Depending on the command, the server may optionally send or receive additional data. If the additional data are sent from a client to the server, then the latter will reply after receiving them in the above manner. Examples of communication records and the list of possible errors for each task are available in their descriptions.

Limitation on the number of commands There is a limit of the maximum number of commands issued during one turn for each server running each task. Reaching the limit will be signaled by the following error: `FAILED 6 commands limit reached, forced waiting activated`. When the error occurs, the server will send the additional message: `WAITING x` — where x ($x \in \mathbb{R}$) is the number of seconds left to wait, i.e., till the end of a current turn.

2.3. Conventions

If not stated otherwise, we assume that:

- Every line ends with a single character of ASCII code 10 ('\n'). Carriage return character ('\r'; ASCII code 13) which may appear in certain operating systems will be treated as a whitespace character.

- In case of data sent by the server, words and numbers will be separated with a single space character.
- In case of data sent by the client to the server (i.e., command parameters), any number of white characters (greater than zero) is allowed between the data and also at the beginning or at the end of the line.
- Whitespace characters are: space, carriage return ('\r') and tabulation sign ('\t').

3. Overall score

3.1. Contest speed up

K coefficient For each server, the score of a given team is additionally multiplied by the K coefficient – the coefficient of the contest speed. During the contest, the coefficient value is being increased exponentially, from 1 to 8. It means that a team may gain much more points in the last hours of the contest rather than at the beginning of the game. Therefore, it is important to keep improving your solutions, so as not to be outscored by other teams. The current value of K is available in the proper command of each task.

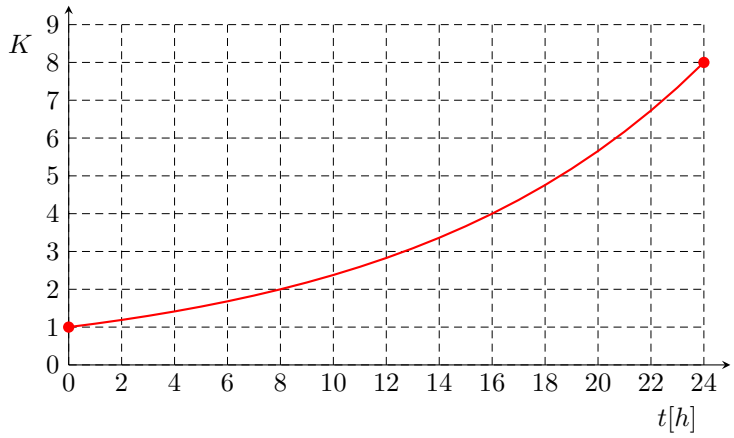


Figure 1: The change of the K coefficient in time.

3.2. Ranking points

Points Teams get points on each server as defined in a task specification. Points gained on each server will be converted into **ranking points** which are defined as a product of the value 100 and the ratio of the team's result (on a given server) and the arithmetic mean of the three best results (on a given server). Points given to a team for a specific task are the arithmetic mean of ranking points from all servers of a given task.

General ranking Competition raking is based on the sum of ranking points of all tasks.

4. Ranking system in specific tasks

Tasks that employ ranking in their internal scoring system calculate the results based on predetermined place-value relation. The team which occupies the i-th place in the ranking according to employed score is given amount of *internal ranking points* according to the Table 5. It is possible for a team to be outside of the ranking entirely due to task-specific conditions (a common reason is a complete lack of participation). A team that is not ranked receives a ranking score of 0.

If two or more teams would occupy the same position in ranking, they receive equal amount of ranking points based on the highest rank they qualify for, and adequate number of places (places they would otherwise occupy) is skipped. For example, three teams with scores of 10, 10, and 5 would respectively be ranked 1. (100 internal ranking points), 1. (100 internal ranking points), and 3. (60 internal ranking points).

Table 1: Team places in the ranking and the corresponding internal ranking point values

Rank	1	2	3	4	5	6	7	8	9	10
Internal ranking points	100	80	60	50	45	40	36	32	29	26
Rank	11	12	13	14	15	16	17	18	19	20
Internal ranking points	24	22	20	18	16	15	14	13	12	11
Rank	21	22	23	24	25	26	27	28	29	30
Internal ranking points	10	9	8	7	6	5	4	3	2	1

5. Emergency situations

In case of situation where not every team is able to participate in the contest on the stated terms (e.g., because of the electricity outage, total or partial failure of local network, problems with contest hardware, etc.), and blame lies neither on those teams nor their equipment, the organisers will suspend the activity of the contest system. Additionally, the participants will be informed of a restart of the system by a local WWW service of the contest. Points will not be calculated during the breakdown. In that case all connections with the contest server might be lost.

6. Wild wild space

6.1. Introduction

Beetlejumpers, despite their apparent toughness and resilience, are sometimes subject to diseases which is a consequence of being exposed to numerous pathogens on the conquered planets. Fortunately, the Beetlejumper Universal Health Care solves most of the health issues in record time. However, sometimes a unique disease manages to stop a previously successful campaign in its tracks – as it was the case this time.

A new disease – *Pacis morbus* – contracted by 9451^{st} Expeditionary Force, was a reason for many sleepless nights for beetlejumper scientists. But in the end, they managed to find a promising cure – with a catch. It required a rare protein, found solely in milk of a certain species of megacows. The main problem is that megacows live in a rather "dimensional unstable" area of space, and due to limited time breeding them is not an option. Therefore, the task of acquiring milk was given to the best of the best teams – and that means you.

6.2. Problem

You are tasked with providing as much milk as possible. The only way to do it, according to the Rules of Engagement Regarding Megacows in Multidimensional Space (a document issued by General Command), is to milk as many megacows as possible.

In order to improve yields, General Command has given the same task to more than one team in the same location. The team that will provide most milk will receive appropriate rewards, eternal glory, personal thanks from General Command, and more dangerous new assignment. Competition, regulated by aforementioned Rules of Engagement, is expected to be fierce, but fair. Combat is expected, but should not take priority over megacows acquisition.

6.3. Game model

The game is divided into turns of equal duration. During every turn, the teams communicate with the server and give orders to groups of beetlejumpers that operate from a rancho. The groups acquire megacows, fight with each other over megacow herds, and bring megacows to a rancho belonging to a given team, where they are milked.

6.4. World

As megacows exist in various, rarely stable conditions, the acquisition operation is performed in D-dimensional space, where $D \in \mathbb{N}$, $2 \leqslant D \leqslant 4$. Every map, depending on D, is either a square of $m \times m$ areas, a cube of $m \times m \times m$ cubes, or a hypercube of $m \times m \times m \times m$ hypercubes, where $m \in \mathbb{N}$. For the sake of simplicity, components of the world (areas, cubes, or hypercubes) are called **fields**.

Some of the world parameters described in further sections have common values if specific number of dimensions is used. The most important of these parameters are shown in Table 2 on page 19.

All distances are measured via Chebyshev (chessboard) distance, given as:

$$dist(A, B) = \max_i(|A_i - B_i|) \tag{1}$$

where A and B denote two points and subscript i indicates iteration over the points' coordinates.

The fields come in two types: accessible, which are a majority, and inaccessible. Due to varying nature of the areas, inaccessible fields may vary from mountains or shrubbery to four-dimensional disturbances in space. Regardless of the reason, entering an inaccessible field is not possible. Nonetheless, they do not hinder sight – beetlejumper teams have been given sufficient equipment to see through obstacles.

Operation area is limited to the section of the map where megacows occur naturally. Leaving the assigned operation area is not allowed.

6.4.1. Important locations

Every team has their own rancho, where megacows are secured and milked. The rancho always has the size of one field and is the base of operations for the team. Locations of ranchos belonging to other teams can be listed – rancho is too big object to be effectively concealed. Additionally, in order to boost competitiveness and ensure acknowledgement of successful teams, locations of ranchos of the teams that are best in acquiring milk are publicly listed by General Command – this will happen to any team that gets ahead of the competition, to give them the opportunity to revel in their fame.

6.4.2. Maps

Groups receive maps of operation territories before the operations commence. Unfortunately, as this is a remote part of the Universum, the maps were created a long time ago and since then, they have not been updated to contemporary beetlejumper standards. Moreover, the maps have been created by lowest-bid contractor, who did not provide any documentation for them. Therefore, you have to discover the data format of the map by yourselves.

6.5. Units

Due to the size of megacows, inherent dangers of the world, and collectivist nature beetlejumpers operate in **groups**. Groups can be composed and disbanded on the rancho only – other fields are far too dangerous to perform such operations. As General Command dislikes chaos, number of the groups controlled by each team must not exceed 15.

6.5.1. Unit types

The groups are composed of two types of units – cowboys and gunmen. The cowboys are poor fighters, but are capable of handling megacows. The gunmen are great fighters, but they are unable to tell a difference between a megacow and a megagoat. In other words, cowboys increase probability of cow capture and make herd management possible, while gunmen make – obviously rare and absolutely unnecessary – skirmishes possible and provide safety to the groups.

Cowboy Meet the veteran of a thousand cow captures! The cowboys are very impressive in capturing megacows; not so much in combat. The more of them the group has, the more probable is the megacow capture within one turn. If pressed, they can defend the megacows – and themselves – but that would be waste of their talents. Not to mention a very ineffective process.

Cowboy has *power* of P_c and can capture megacows and take care of them. A group without any cowboys is unable to handle megacows at all.

Gunman Meet the veteran of a thousand duels! The gunmen are prepared to fight – and they do it well; most of them are retired soldiers. Unfortunately, this means they are not able to do much else. They do not contribute to megacows captures, but if an enemy group makes a mistake and attacks – they will defend and retaliate. Simply put, they are tasked with defending both megacows and cowboys. With the emphasis on the megacows.

Gunman has *power* of P_g and is unable to capture or take care of megacows.

6.5.2. Group attributes

Composition The group composition is given by two numbers that describe the count of cowboys and gunmen. This parameter does not change – it may only be modified by disbanding the group on the rancho and forming it anew.

Sight range Groups have two ranges of sight – one, of V_L fields, allows for detection of megacows and other teams, but does not grant information about their composition. To acquire more details – like the enemy group condition – the group has to approach to be in range of V_S (where $V_S < V_L$) fields. These parameters do not change during the group lifetime.

Power Each group has a ***power*** attribute that depends on their composition. The more specialists are in the group, the stronger the group is. The initial (maximum) power value (P_{max}) can be expressed as:

$$P_{max} = N_c \cdot P_c + N_g \cdot P_g \tag{2}$$

where N_c and N_g states for the number of cowboys and gunmen, respectively; P_c is the power of a single cowboy, P_g is the power of a gunmen. This parameter decreases with movement, megacow capture, and combat. It may increase with resting and visiting their own rancho.

Group whose *power* dropped to 0 is incapable of anything but movement. Capturing megacows and attacking becomes impossible.

Group that remains in the field, but does not perform any activity regenerates *power* slowly. Group that visits their own rancho immediately regains all their *power* – coffee brewed in the Ultimate Huge Fusion Coffee Maker really does wonders. Unfortunately, the coffee decays far too fast to be used in the field.

Experience Contrary to popular belief, beetlejumpers are capable of learning, although mainly as a group. When it comes to megacows, learning comes from experience with large herds. Handling a one-megacow herd is not going to teach the group anything useful, while every doubling in size of the controlled herd gives the group a 5% bonus to future megacow captures (see the Capturing megacows section). The bonus is permanent and does not decrease once the group passes the herd – after all, they did work on a large project. Only the largest herd the group ever handled counts – experience with smaller herds ceases to be relevant once a larger one is handled. Due to the collectivist nature of beetlejumpers, this bonus is lost once the group is disbanded.

6.5.3. Purchase of additional units

It is possible to call upon reinforcements by requesting the Command to provide more specialists to bolster ranks of brave beetlejumpers. Unfortunately, such reinforcements are not completely legal, therefore require a bribe – delivered in precious milk. After a bribe is delivered, a beetlejumper of a desired the class will arrive at the rancho at the beginning of the next turn.

As corruption is expected and even encouraged in this remote part of the Universum, there is no risk of arrest or fine due to bribery. Unfortunately, the bribes required for acquisition of units increase by 50% with every request player makes for the given type of beetlejumper (i.e. purchasing a cowboy does not make gunmen more expensive). As different operators take requests from different teams, bribe costs grow per team.

6.5.4. Group actions

Movement Groups move at the speed of one field per turn. Movement may happen in any direction – in an D-dimensional space, there are $(3^D - 1)$ fields to choose from – as long as the field is accessible. It is possible to enter a field already occupied by a group, even if it is not of the same affiliation.

Movement reduces *power* for every field crossed – traversing dangerous multidimensional space is exhausting. The reduction is proportional to the group size, no matter the profession of the units. The group loses from its power L_m multiplied by the number of all beetlejumpers in the group.

Search for megacows A group may look for megacows in two ways: using sophisticated equipment for a long-range bioscan that reveals megacows concentration areas, or via visual scan which reveals precise locations of megacows – albeit on far shorter range. Search does not tire anyone, thus uses no *power*.

6.5.5. Combat

Combat is a sad necessity in the highly competitive hunt for megacows. It serves several purposes, primarily – acquisition of the megacows that have previously been captured by the competitors. An attack must be intentional and clearly indicated. Results are based on *power* attribute of the groups. In general, all other parameters being equal, group with a higher value of its *power* will be victorious. However, the combat will lower its *power*.

It is important to note that Beetlejumper Conventions clearly prescribe one-on-one combat. If a group from the team A is engaged in combat with a group from the team B, then the team C may not attack either group with their own group until the battle is finished. Both teams A and B are free to introduce more of their own forces in the battle via the mechanism of reinforcements.

Fighting groups, as well as their reinforcements, may not move during the engagement.

It is not possible to fight on the rancho – as it contains the valuable megacows, no team may attack or be attacked on any rancho.

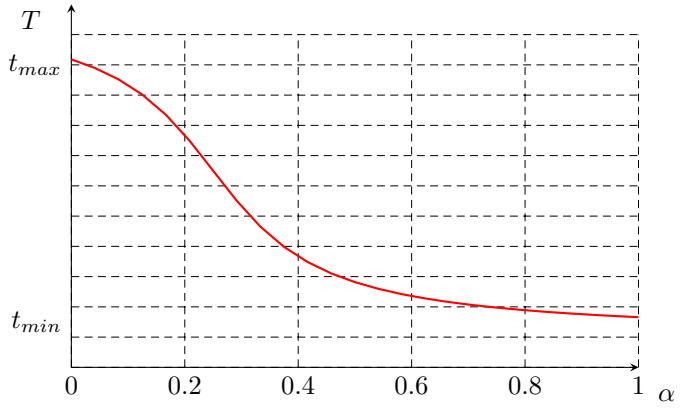
Combat is initiated when one of the groups chooses to confront a group on the same field. A challenge cannot be refused, even if the group tries to run away, the challenge forces them to stop and fight. Combat begins at the beginning of the next turn.

Combat takes time. It will never take less than t_{cmin} turns; this is the minimum time to engage and fire. It can take no more than t_{cmax} – the maximum time the convention allows. In most cases, time of the fight will vary depending on groups *power* – the closer the values are, the longer the combat. The time T depends on α – a ratio between both groups *power*, calculated as:

$$\alpha = \frac{|P_1 - P_2|}{\max(P_1, P_2)} \tag{3}$$

P_1 and P_2 represent power of each team. The approximate relation between T and α is depicted in Figure 2.

Figure 2: Dependency between fight time and difference between forces



Reinforcing As the combat takes some time, it is possible to reinforce the combatants. It requires the reinforcing team to be in the same field as the combatants. As this changes the situation of previously engaged forces, time of the encounter is extended. It is increased by the value calculated with P_1 being the *power* of newly arrived group(s), and P_2 is the power of the most recently arrived opponent group(s). Note that if more than one group from the same side join the fray in the same turn, the total sum of their *power* is considered for encounter time recalculation. The same rule (use the sum of the power) is applied in case of starting a fight with several groups already present in the field.

The arriving teams join the battle automatically once they enter the field it takes place on. As the Convention prohibits engagements of more than two factions and the intra-team fights are strictly forbidden, a group can only reinforce its own troops.

Reinforcements share the fight result of the battle – suffering exhaustion depending on the result and loss of megacows – in the same degree as the initial fighters. The only exception concerns the spoils – all megacows lost by the loser go to the first challenger or the first group faced with a challenge.

Resolution Team whose groups have higher total *power* at the end of the encounter emerges victorious, but never unscarred. Each member of the losing team loses L_l of its *power* and – per Convention rules – cannot be attacked by anyone for the next t_g turns. This does not forbid **them** from initiating a combat, even if this is rarely a wise choice for the battered group.

Each member of the winning group loses L_w of its *power* and takes control over loser's herd. Reinforcements are treated the same way, depending which side has won – reinforcements of the losing side lose their megacows as well. The herd is attached to the team that was the initial combatant, as long as it has at least one cowboy. Should the group that was supposed to receive the megacows have no cowboy, the megacows will become wild again, subject to capture by anyone.

As beetlejumpers are a naturally collectivist society, the *power* loss is shared across the entire group, therefore the collective *power* of the group is reduced by number of members multiplied by the appropriate coefficient.

Tie It is possible, however unlikely, that both sides end with the same *power*. In such a case, a tie occurs. All groups involved suffer loss of *power* as they lost the battle, but they do not lose control over their herds – no megacows change hands. Neither do they enjoy the grace period, thus being susceptible to another attack.

6.6. Megacows

Megacows are non-player units that wander the multidimensional countryside. They are the prize to be acquired and fought over. They are rather docile and do not pose a direct threat to veteran beetlejumpers, but are rarely an easy target to capture.

6.6.1. Capturing megacows

Capturing a cow is a difficult task, fitting a professional. It is performed solely by cowboys – the more of them in the search group, the greater is the chance of capture within one turn.

Finding a megacow First step to capture a megacow is to locate it. There are two ways: using a long-distance bioscanner that provides information regarding megacow concentration in the area, or by visual observation that provides precise information regarding megacows' locations. One of the flaws of the scanner is its inability to detect megacows inside a rancho due to shielding used – it detects them everywhere else in range.

Catching a megacow Second step is to approach the chosen megacow group and attempt capture. The capture action is far faster than movement, therefore it is considered to happen in one turn. It consists of a sequence of independent attempts done by each cowboy within a group does. Statistical analysis performed by General Command has shown that every cowboy – due to their unified training – has the same chance to capture a megacow. Due to specific code of honor, they attempt the capture sequentially within a group: when first one fails, next one makes the attempt. Clearly, this means the more cowboys in the group, the greater chance of capture is during a single turn.

If there is more than one megacow in the field, it is possible to capture more of them. The cowboys in a group attempt to capture as many megacows as possible in one turn. Obviously, it is not possible to capture more megacows in a single turn than the group has cowboys, nor is it possible to capture more megacows than there are present in the field.

There are two limitations regarding capturing megacows. First, as this is a dangerous and precision-requiring task, it is not possible to attempt it while being exhausted. Thus, a group with the *power* of 0 is unable to initiate a capture. Second, a completely failed capture – one that yields no megacows – causes

a state of deep dissatisfaction and lowers morale of the group, making any capture attempt by given group impossible for 7 turns. Because of the sequential nature of capture action, probability of not capturing any megacow by a group in a single turn (P_{fail}) can be expressed as follows:

$$P_{fail} = 1 - (1 - P_{cowboy})^N \tag{4}$$

where P_{cowboy} is the actual probability value for a cowboy to successfully complete a single attempt to capture a megacow, N is the number of cowboys in the group performing capture action.

Capture is a tiring operation, but only to the cowboys, as the gunmen are not participating. Therefore, every time a capture is attempted, the group loses L_c multiplied by number of cowboys from its *power*.

6.6.2. Herds

All megacows held by a given group of beetlejumpers are considered to belong to one herd.

Due to the fact that capture and traveling with a band of beetlejumpers is a stressful experience to megacows, they form bonds to cope. A herd that formed such bonds is considered **stable**. Any other herd is considered **unstable**. Every change in herd's composition – no matter whether it is an addition or reduction of megacows – destabilizes the herd, as the megacows suffer the loss of friends or become suspicious of newcomers. Passing a herd between groups does not destabilize it, providing the receiving group does not hold any megacows – if they do, the herds merge and become unstable.

The only way to stabilize a herd is to take them out in the wild, away from a rancho, for at least 5 turns. This, obviously, presents a danger due to possible attacks from the hostile groups. In order to stabilize, megacow herd must stay away from any rancho and may not move. Walking in the range of the rancho before the herd stabilizes makes megacows nervous again, wasting time spent on stabilizing them. The minimum distance from rancho required to stabilize the herd (d_{stable}) depends on the space in which the operation is performed: 12 fields in two-dimensional space, 5 fields in three-dimensional space, and 3 fields in four-dimensional space.

Moving while operating in safe distance from rancho does not cancel the process, but puts it on hold – turns spent moving are not counted towards time required for stabilization. As combat does not involve group movement nor causes cows to be engaged, it does not stop the stabilization process.

It is important to note that a herd cannot exist without at least one cowboy guarding it. Disbanding a group that holds a herd will cause the megacows to disperse, wasting the effort of many and delaying the victory. And, of course, irritate the hard-working beetlejumpers.

6.6.3. Milking cows

The megacows are milked automatically once they are on the rancho, as the command provided all ranchos with a milking machine. Each megacow gives Y_m bottles of milk per turn (containers are provided and are not a subject of the beetlejumper concern; it is possible for the container to be filled only partially, thus the bottle count may end with a fraction). Yet, it is not sufficient to just deliver the megacows and reap the benefits. The ranchers have to take care of megacows' morale and mood. Thus, only a stable herd can be milked.

The Command provided every rancho with a milking machine – each rancho has only one. And herds are too nervous in each others' presence to share the machine. Therefore, even if there are numerous herds on the site, only the largest stable herd is milked. It is, of course possible to milk only your own megacows on your own rancho – competitors do not share their milking machine.

Megacows are social creatures. The bigger the herd, the better they feel, becoming more relaxed, in turn increasing efficiency of milking. For every doubling of the herd size, amount of milk given by each megacow increases by 5%.

In order to reward the best teams, the Beetlejumper Operation Command lists top teams with the largest herds – along with their ranchos' locations – as great examples to follow.

6.6.4. Megacow behavior

Megacows, if not captured, tend to wander around without any plan or order. They neither run away from beetlejumpers nor are drawn towards them. It is important to note that their reproductive cycle is not well known. It is nonetheless a verified fact that when population in the wild decreases, new megacows

begin to appear on the map. Operating groups are advised to keep an eye open for new megacows in the field.

6.7. Start and the end

Initial state of the game In the beginning, all megacows are randomly distributed around the map, and initial set of beetlejumper specialists – several gunmen and several cowboys – are in their ranchos. No groups are formed.

Each team receives the same starting number of specialists.

End of the game The competition ends when the Beetlejumper Operation Command realizes there is sufficient amount of milk to produce required amount of medicine, no matter who gathered it. There is also a second option: if the operation takes too long, milk will cease to be useful, as the disease will cause permanent damage to the infected, making their treatment impossible. The operation will then be closed. In other words, the time to gather the milk is limited. Time in which the milk has to be collected is well known, therefore the maximum number of turns the operation may take is provided.

6.8. The aim of the game and competition

Your main goal is to gather the most milk in comparison to other teams.

Scoring Teams are ranked according to amount of milk they have on their rancho. The score given to each team (S) is a value expressed with the formula below:

$$S = 10^5 \cdot \frac{M_T}{M_L} \tag{5}$$

where M_T is the number of milk bottles the team acquired, and M_L is the total limit of milk bottles to gain in the game.

6.9. Commands

General guidelines on the communication protocol (connecting, logging in, commands, response format) are described in section *Server communication*. Below you can find commands for the problem Wild wild space.

Whenever *Location* is used as a type for a specific value it contains D values separated with a single whitespace and each value has the same limits as N (see `DESCRIBE_WORLD` command).

Whenever *Relative* is used as a type for a specific value it must contain D values separated with a single whitespace and each value must comply the range $\{-1, 0, 1\}$.

DESCRIBE_WORLD Returns the game parameters and the value of the score scaling coefficient.

Parameters: none

Data (from server):

The server returns one line containing the following values separated with a single whitespace:

- D ($D \in \mathbb{N}$, $2 \leqslant D \leqslant 4$) — number of dimensions,
- N ($N \in \mathbb{N}$, $1 \leqslant N \leqslant 300$) — length of the side of the board,
- T ($T \in \mathbb{N}$, $1 \leqslant T \leqslant 3$) — the duration of a single turn in seconds,
- V_L ($V_L \in \mathbb{N}$, $1 \leqslant V_L \leqslant 8$) — the long sight range of a group – cows and enemies in this range are visible to the group,
- V_S ($V_S \in \mathbb{N}$, $1 \leqslant V_S \leqslant 5$) — the short sight range of a group – details of other groups (their *power*),
- Map (Map is a string) — name of the map file which describes the type of each field,
- L ($L \in \mathbb{N}$, $1 \leqslant L \leqslant 100$) — the maximum number of commands which can be issued in one turn,
- K ($K \in \mathbb{R}$, $1 \leqslant K \leqslant 8$) — value of the score scaling coefficient,
- P_c ($P_c \in \mathbb{N}$, $0 \leqslant P_c \leqslant 2000$) — *power* of a single cowboy,
- P_g ($P_g \in \mathbb{N}$, $0 \leqslant P_g \leqslant 2000$) — *power* of a single gunman,
- t_g ($t_g \in \mathbb{N}$, $0 \leqslant t_g \leqslant 30$) — grace period – time in turns in which defeated group cannot be attacked,
- L_l ($L_l \in \mathbb{N}$, $0 \leqslant L_l \leqslant 300$) — loser's loss – coefficient of *power* lost when defeated in combat,
- L_w ($L_w \in \mathbb{N}$, $0 \leqslant L_w \leqslant 300$) — winner's loss – coefficient of *power* lost when victorious in combat,
- L_c ($L_c \in \mathbb{N}$, $0 \leqslant L_c \leqslant 100$) — capture loss – coefficient of *power* lost with successful cow capture,
- L_m ($L_m \in \mathbb{N}$, $0 \leqslant L_m \leqslant 100$) — movement loss – coefficient of *power* lost with every field the group moves,
- t_{cmin} ($t_{cmin} \in \mathbb{N}$, $1 \leqslant t_{cmin} \leqslant 30$) — minimum combat time in turns,
- t_{cmax} ($t_{cmax} \in \mathbb{N}$, $t_{cmin} \leqslant t_{cmax} \leqslant 30$) — maximum combat time in turns,
- p_{cap} ($p_{cap} \in \mathbb{R}$, $0 < p_{cap} \leqslant 1$) — probability of megacow capture by single cowboy,
- Y_m ($Y_m \in \mathbb{R}$, $0 < Y_m \leqslant 100$) — amount of milk bottles per turn received from a single megacow.

PROGRESS Returns the information about the game progress and the current team achievements.

Parameters: none

Data (from server):

The first line returned by server includes:

- G_{ID} ($G_{ID} \in \mathbb{N}$) — number which identifies current game (constant value during a single game),
- E ($E \in \mathbb{N}$) — number of turns until the end of the game (including the current turn),
- M ($M \in \mathbb{R}$) — number of milk bottles still left to acquire until the end of the game.

- M_L ($M_L \in \mathbb{R}$) — initial limit of milk bottles in the game (this is equal to M at the beginning of the game).

In the second line the server returns:

- M_T ($M_T \in \mathbb{R}$) — number of milk bottles the team acquired,
- R ($R \in \mathbb{N} \cup \{\texttt{NONE}\}$) — current team rank based on M_T or `NONE` if M_T equals 0.

The third line returned by the server contains:

- N_B ($N_B \in \mathbb{N}$) — number of ranchos with the greatest number of milked megacows. Three greatest different numbers of milked megacows are considered. Ranchos with no milked megacows are not displayed.

The following N_B lines describe best teams during the current game with values:

- H_C ($H_C \in \mathbb{N}$) — head count of the largest stable herd on the rancho,
- L ($L \in \mathit{Location}$) — location of team rancho.

LIST_GROUPS Returns the list of groups being under command of the team.

Parameters: none

Data (from server):

The first line returned by the server contains:

- G ($G \in \mathbb{N}$) — number of groups.

The following lines describe the units with the values separated with a single whitespace:

- *ID* (*ID* $\in \mathbb{N}$) — group's *ID*,
- L ($L \in \mathit{Location}$) — the coordinates of the field with the corresponding group,
- C ($C \in \mathbb{N}$) — number of cowboys in the group,
- G ($G \in \mathbb{N}$) — number of gunmen in the group,
- S ($S \in \mathbb{N}$) — the total value of *power* of all units in the group,
- B ($B \in \mathbb{N}$) — number of cows the group is tending,
- H ($H \in \mathbb{N} \cup \{\texttt{NA}\}$) — number of turns left for the megacows to form a stable herd (0 for a stable herd, `NA` if no cows are conducted by the group),
- U ($U \in \mathbb{N} \cup \{\texttt{INF}\}$) — number of consecutive turns, in which the group is unable to catch megacows (`INF` when C is 0),
- *Status* (*Status* $\in \{\texttt{NORMAL}, \texttt{PROTECTED}, \texttt{FIGHTING}\}$) — status of the group,
- Z ($Z \in \mathbb{N}$) — number of consecutive turns, in which the group is being protected.

MAKE_GROUP Creates a new group from the units on the rancho not being currently engaged in any group.

Parameters:

- C ($C \in \mathbb{N}$) — number of cowboys in the group,
- G ($G \in \mathbb{N}$) — number of gunmen in the group.

Data (from server):

- *ID* (*ID* $\in \mathbb{N}$) — group's *ID*.

UNGROUP Disbands a group that is located in the team's rancho.

Parameters:

- *ID* (*ID* $\in \mathbb{N}$) — group's *ID*.

MOVE Moves to the adjoining field, indicated by relative coordinates. At least one of the relative coordinates must be non-zero.

Parameters:

- ID ($ID \in \mathbb{N}$) — group's ID,
- L_D ($L_D \in Relative$) — movement along all dimension axes.

CAPTURE Tries to catch a cow (or cows) placed in the same location as the group.

Parameters:

- ID ($ID \in \mathbb{N}$) — ID of the group that is trying to catch megacows.

Data (from server):
The only line returned by the server contains the single value:

- C ($C \in \mathbb{N}$) — number of cows captured.

HAND_COWS Given number of cows are passed to be conducted with another group. Both groups should be located in the same field. This influences the herds stable state in every case except handing all cows to the group not conducting any megacows.

Parameters:

- ID_A ($ID_A \in \mathbb{N}$) — ID of the group that is trying to give megacows away,
- ID_B ($ID_B \in \mathbb{N}$) — ID of the group that the megacows is given to,
- C ($C \in \mathbb{N}$) — number of megacows to be passed,

BIOSCAN Looks for cows in a given region. The space to search for animals is defined by two opposite fields of the region. None of the fields can be located further than V_L units from the scanning group. Rancho cows are protected from scanning – cows in a rancho are not counted.

Parameters:

- ID ($ID \in \mathbb{N}$) — group's ID,
- L_1 ($L_1 \in Location$) — the coordinates of the first field defining a region,
- L_2 ($L_2 \in Location$) — the coordinates of the second field defining a region.

Data (from server):
The only line returned by the server contains the following values separated with a single whitespace:

- F_N ($F_N \in \mathbb{N}$) — number of fields scanned,
- F_C ($F_C \in \mathbb{N}$) — number of fields occupied with cows,
- C_T ($C_T \in \mathbb{N}$) — total number of cows in the scanned region.

ATTACK Attacks a selected enemy group, indicated by its ID. The attacked group must be in the same location as the aggressor and cannot be already under attack of another team. It is impossible to attack any group on a rancho site.

Parameters:

- ID ($ID \in \mathbb{N}$) — ID of the attacking group,
- ID_E ($ID_E \in \mathbb{N}$) — ID of the enemy group to be attacked.

LIST_ENEMIES Returns the list of the enemies visible for all the groups of the team. This command may be called only once a turn.

Parameters: none

Data (from server):

The first line returned by the server contains:

- G ($G \in \mathbb{N}$) — number of visible enemy groups.

The following lines describe the enemy groups with the values separated with a single whitespace:

- ID ($ID \in \mathbb{N}$) — group's ID,
- L ($L \in Location$) — the coordinates of the group,
- C ($C \in \mathbb{N}$) — number of cowboys in the group,
- G ($G \in \mathbb{N}$) — number of gunmen in the group,
- B ($B \in \mathbb{N}$) — number of cows the group is conducting,
- S ($S \in \mathbb{N} \cup \{\texttt{UNKNOWN}\}$) — the total value of *power* of all units in the group or `UNKNOWN` if the group is too far away from the team's groups or rancho,
- $Status$ ($Status \in \{\texttt{NORMAL}, \texttt{FIGHTING}, \texttt{PROTECTED}\}$) — status of the group,
- $Team$ ($Team \in \mathbb{N}$) — team's ID (assigned randomly at the beginning of each game).

LIST_FIGHTS Returns the list of the fights the team is currently part of.

Parameters: none

Data (from server):

The first line returned by the server contains:

- F ($F \in \mathbb{N}$) — number of fights the team is taking part in.

The following lines describe the team current fights with the values separated with a single whitespace:

- L ($L \in Location$) — the coordinates of the fight,
- B ($B \in \mathbb{N}$) — number of cows the fight is for (both yours and enemy's),
- T ($T \in \mathbb{N}$) — number of turns till the end of the fight (including the current turn),
- G ($G \in \mathbb{N}$) — number of groups of your team in the fight,
- S ($S \in \mathbb{N}$) — total value of your *power*,
- $Team$ ($Team \in \mathbb{N}$) — enemy team's ID (assigned randomly at the beginning of each game),
- G_E ($G_E \in \mathbb{N}$) — number of enemy groups in the fight,
- S_E ($S_E \in \mathbb{N}$) — total value of enemy's *power*.

LIST_COWS Returns the list of cows visible for all groups of the team. This command may be called only once a turn.

Parameters: none

Data (from server):

The first line returned by the server contains:

- F ($F \in \mathbb{N}$) — number of fields with visible cows.

The following lines describe fields with the values separated with a single whitespace:

- L ($L \in Location$) — the coordinates of the field,
- B ($B \in \mathbb{N}$) — total number of cows in the field,
- B_G ($B_G \in \mathbb{N}$) — total number of conducted cows in the field (cows that belong to any group, including your groups).

LIST_RANCHOS Returns the list of all ranchos on the map.

Parameters: none

Data (from server):

The first line returned by the server contains:

- R ($R \in \mathbb{N}$) — number of ranchos on the map.

The following lines describe a single rancho each:

- L ($L \in Location$) — the coordinates of the rancho,

RECRUIT Trains a new beetlejumper to be a cowboy or a gunman.

Parameters:

- $Type$ ($Type \in \{\texttt{COWBOY}, \texttt{GUNMAN}\}$) — type of training (cowboy or gunman, respectively).

RANCHO_REPORT Returns the information about your rancho location and dairy details.

Parameters: none

Data (from server):

The only line returned by the server contains the following values separated with a single whitespace:

- L ($L \in Location$) — the coordinates of the rancho,
- C ($C \in \mathbb{N}$) — number of cowboys on the rancho that are not a part of any group,
- G ($G \in \mathbb{N}$) — number of gunmen on the rancho that are not a part of any group,
- P_C ($P_C \in \mathbb{R}$) — number of milk bottles needed to recruit a new cowboy,
- P_G ($P_G \in \mathbb{R}$) — number of milk bottles needed to recruit a new gunman,
- ID ($ID \in \mathbb{N} \cup \{\texttt{NONE}\}$) — ID of group which cows are being milked or `NONE` if no group has a herd which can be milked,
- M ($M \in \mathbb{N}$) — number of cows currently milked,
- P ($P \in \mathbb{R}$) — number of milk bottles each cow is producing each turn.

WAIT Waits till the next turn begins.

Parameters: none

Data (from server):

The server returns a single line of characters:

- WAITING S

where S ($S \in \mathbb{R}, S \geqslant 0$) stands for the number of seconds left to wait. Once the period is over, the server sends an additional line:

- OK

6.10. Errors

In case the command is incorrect, the server responds according to the description in section *Server communication* with the following message:

- '`FAILED` *e msg*',

where *e* is an error code, and *msg* — an error message. The table below consists of errors that may occur during communication process in problem Wild wild space.

error code	error message
1	bad login or password
2	unknown command
3	bad format
4	too many arguments
5	internal error, sorry...
6	commands limit reached, forced waiting activated
100	not enough power to do that
101	there are no cowboys in this group
102	unable to capture cows yet
103	group is busy
104	no cows to pass
105	target field is not accessible
106	not enough recruits to make team
107	requested group not found
108	group is not on a rancho
109	groups are not on the same field
110	unable to attack own group
111	unable to fight on rancho
112	not enough milk for recruitment
113	invalid cow count
114	group is protected
115	no cows to capture
116	group limit reached
117	invalid movement direction
118	command already used
119	group is fighting

6.11. Servers

The games will be held on servers with different multidimensional space each. Some of the world parameters have common values if specific number of dimensions is used. These parameters are shown in Table 2.

Table 2: Server addresses and number of space dimensions.

Name	**Address:Port**	**Number of space dimensions** $- D$
Cows1	`universum.dl24:20000`	2
Cows2	`universum.dl24:20001`	3
Cows3	`universum.dl24:20002`	4

Table 3: World parameters connected to the number of space dimensions.

Description	**Symbol**	**Value when** $D = 2$	**Value when** $D = 3$	**Value when** $D = 4$
Power lost with each move	L_m	1	2	3
Short sight range	V_S	4	2	1
Long sight range	V_L	8	4	2
Stabilization distance from rancho	d_{stable}	12	5	3
Grace period of defeated group	t_g	12	5	3

6.12. Example

Below you will find an exemplary record of the communication with the server.

client → server	server → client
	`LOGIN`
`login1`	
	`PASS`
`secret`	
	`OK`
`DESCRIBE_WORLD`	
	`OK`
	`2 10 8 5 8 world1.map 500 1.000188 50 80 10 10 5 1 1 2 22 0.550000 1.000000`
`PROGRESS`	
	`OK`
	`812 67.000000`
	`0.000000 0`
	`0`
`RANCHO_REPORT`	
	`OK`
	`2 5 16 6 100 100 NONE 0 0.000000`
`LIST_COWS`	
	`OK`
	`0`
`LIST_RANCHOS`	
	`OK`
	`5`
	`10 3`
	`5 3`
	`2 5`
	`3 6`
	`6 6`
`MAKE_GROUP 16 6`	
	`OK`
	`3`
`PROGRESS`	
	`OK`
	`2 811 67.000000`
	`0.000000 NONE`
	`0`
`BIOSCAN 3 1 4 3 6`	
	`OK`
	`9 1 1`

```
LIST_COWS
                OK
                6
                2 6 1 0
                3 2 5 0
                3 8 1 0
                3 10 1 0
                4 2 1 0
                6 2 1 0
MOVE 3 0 1
                OK
WAIT
                OK
                WAITING 0.052000
                OK
CAPTURE 3
                OK
                1
LIST_COWS
                OK
                6
                2 6 0 1
                3 2 5 0
                3 8 1 0
                3 10 1 0
                4 2 1 0
                6 2 1 0
LIST_ENEMIES
                OK
                2
                1 9 9 18 8 7 UNKNOWN NORMAL 1
                2 2 6 18 8 11 1150 NORMAL 4
ATTACK 3 2
                OK
LIST_FIGHTS
                OK
                1
                2 6 8 5 1 1240 4 1 1150
```

7. DESK

7.1. Introduction

One of the interesting aspects of beetlejumpers is their unique culture, best reflected in the entertainment industry. An exemplary product of this industry is the Didactic Engagement Simulation Kit (DESK) – a tool designed to simultaneously entertain and educate younger beetlejumpers. It allows them to absorb the correct mindset and learn basic skills they may find useful in adult life – like tactics and logistics – in a relatively violent-free environment.

DESK is a multiplayer game, designed to be enjoyed by many people at the same time. Games consist of individual matches, in the one-versus-one formula. Every player participates in several matches at the same time. This certainly helps improve the multitasking skills of beetlejumpers. Being victorious allows the player to gain in fame and glory – and, possibly, improve the chances of getting his dream job.

7.2. Problem

Your team represents young, competitive, and ambitious – not to mention, a bit bored with other forms of entertainment – beetlejumpers. You have decided to try your chances with the intellectual and challenging DESK – for the aforementioned fame, glory, and opportunities. You will face numerous opponents in one-versus-one battles and – hopefully – triumph over them thanks to your outstanding skills. Are you ready?

7.3. Game model

The crux of the game is the skirmish between two teams. Each player controls a small cadre of units and has a goal of defeating opponent's units. The cadres are identical for both players – one side is attacking, while the other is defending. These roles are assigned randomly before the battle begins. Fasten your seatbelts – four battles (against different opponents) are played simultaneously by each team.

The game is divided into turns of equal duration. During every turn, the teams communicate with the server and give orders to their units.

7.3.1. Glossary

The following terms and concepts come directly from the DESK User Manual:

- **Attribute** – a numerical value quantifying the abilities of a unit in a certain domain. Attributes are described in detail in section 6.4.1.
- **Board** – the battlefield: a grid of 12×10 fields, numbered from 1 to 12 (the X coordinate) and from 1 to 10 (the Y coordinate).
- **Cycle** – a time unit made of turns. It is defined as range of turns in which all stacks of both sides have an opportunity to perform one (non-delayed) action.

- **Field** – a part of the board, it can be accessible or inaccessible. Each field can be either empty or occupied by exactly one stack (see below).
- **Unit** – a basic representation of forces at the player's disposal, characterized by type, attributes and traits.
- **Unit type** – a numeric value that identifies units of the same type. This is an immanent feature of a unit that cannot be changed.
- **Unit size** – the number of fields the stack occupies; two options are possible: 1×1 and 2×2.
- **Stack** – a group of units of the same type, occupying the same fields. Only one stack at a time may occupy a given field, stacks may not split nor merge in the field – their composition is determined in the preparation phase (see Section 6.5.1.). The stack takes damage as one, with the top unit absorbing all damage until the unit is eliminated.
- **Starting area** – a rectangle of 3×10 fields. There are two starting areas on the opposite sides of the board, encompassing the units of each player at the beginning of the battle.
- **Trait** – a specific, named characteristic, attributed only to some units. Traits are described in detail in Section 6.4.2.

7.4. Units and stacks

Each unit has a type. A unit type determines its basic attributes and traits.

7.4.1. Attributes

Each unit is characterized by several attributes which quantify its abilities on the battlefield, each of them expressed as a number. In general, higher values are better, but this rule may have exceptions for some strategies. The attributes are summarized in Table 3.

Table 4: Unit attributes

Attribute	Description
Hit Points	The number of damage points the unit can take before being eliminated.
Initiative	Determines the order in which the stacks take orders and perform actions.
Movement Points	Determines how far a stack of a given type can move.
Attack	Strength of the melee attack.
Defense	Ability to resist attacks.
Damage	Base damage dealt by the unit.
Ranged Attack	Strength of the ranged attack. Equals 0 for melee fighters.

7.4.2. Traits

Units are characterized by traits. These traits can be either generated automatically or assigned by the players. In the latter case, the traits are selected from a set of all available ones given as an array of possible traits. Note that while any unit can be characterized by any trait, not all of the units will benefit from certain traits. Traits are gathered in Table 4.

7.5. The game

Each game consists of four phases. In the **preparation** phase, players assign traits to their units, group them into stacks, and place the stacks in their respective starting areas. In the **tactics** phase, players plan their actions. In the **skirmish** phase, players command the fighting units. In the **results** phase, events from the last skirmish are presented.

Table 5: Unit traits

ID	Trait	Description
1	**Big**	The unit occupies a 2×2 area on the board
2	**Flight**	Can ignore inaccessible terrain and other stacks that would otherwise be in the way. The target field (or fields) must be empty
3	**No counter**	The opponent attacked in a melee will be unable to counterattack
4	**First strike**	Counterattack of the unit which is being attacked will come before the attack happens, unless the attacking unit is also annotated with this trait or the `No counter` trait
5	**Agile**	Can counterattack any number of times in a cycle
6	**Fast**	Initiative receives +100% bonus
7	**No melee penalty**	No penalty for the melee attack for the unit with the ranged attack
8	**Shielded**	Unit takes only half of the damage from ranged attacks
9	**Berserker**	+100% to Attack, −100% to Defense
10	**Poor counter**	Unit deals 50% damage when counterattacking
11	**Impatient**	Unit cannot execute the `DELAY` command
12	**Charge**	+50% to Attack, +20 to Movement Points, +25% to Initiative

Note that every player plays exactly 4 games simultaneously, on 4 separate boards, with 4 different opponents. This requires some tactical skill and multitasking abilities. These four simultaneous games are called a **set of games**.

7.5.1. Phase 1: Preparations

Both players receive their units – a few of each type. Units of a given type are identical in regard to attributes and traits. Additionally, every player receives several free traits they can assign to a chosen unit type. Assigning the same trait to a unit type for several times is useless – the bonuses (or penalties) will be introduced only once. Trait assignment is a separate process for each of the four simultaneously played games.

Once the player has attached the traits – or has chosen not to – the units are grouped into stacks. As described earlier, a stack may consist of one or more units of the same type. Thus, given 4 available units of the same type, it is possible to group them into: (a) a stack of four units, (b) two stacks of two units each, (c) two stacks of one and three units, respectively. It is also possible to, for example, make a stack of two and a stack of one. It is worth noting that some units can be left unused, should that suit the player's strategy. Stacks deployment is a separate process for each of the four simultaneously played games. A player may deploy no more than 8 stacks in any single battle.

Having stacks composed, the player places them in their starting area – the attacker is placing the units in the $(1, 1) - (3, 10)$ area, the defender in the $(10, 1) - (12, 10)$ area. Each stack may be positioned in any empty field in the starting area. Once both players are done with this step, this phase ends.

If no units are deployed, then the DESK's system scatters them randomly to make sure the losers will bite the dust. The opponents will be happy to score some easy points.

7.5.2. Phase 2: Tactics

Once this phase begins, the entire board and all stacks (and, therefore, all units' details) become visible to both players. Both players are set up now, and are able to acquire information about their opponents. This phase is intended to plan the player's actions.

7.5.3. Phase 3: Skirmish

With the beginning of this phase, the players may start issuing orders to their units.

This phase is divided into **cycles**, each containing a number of turns. In every cycle, a stack can have no more than one action with these two exceptions:

- the unit may perform the melee attack after it is moved,
- the unit may use delay action and later perform move, attack or defend.

A stack may also counterattack no more than once in a cycle, unless its trait allows different behavior. Each turn is a period in which one stack can be given orders and can perform a single action. Once a queue of stacks waiting for orders is empty, the cycle ends and another one begins, as long as the phase continues.

The order of unit actions is determined by the initiative – the larger initiative means acting sooner in a cycle. In the case of the initiative tie, the attacking side acts first.

Counterattack is usually performed right after a unit was attacked in melee and may occur for a given attacked unit only once during a single cycle. This behavior may change with some traits (`No counter`, `First strike`, `Agile`).

The possible actions include:

- **Defend** – Do not act and stand your ground – the stack receives a bonus to the Defense attribute until the next possible action of this stack in the next cycle.
- **Delay** – Do not act yet. The initiative of the unit given this order turns negative for this cycle (i.e. if it was 10, then it becomes -10), and the unit will be able to act in the same cycle according to the new initiative attribute. For each unit this action can be activated only once within a cycle.
- **Move** – Move along a described path; the fields on the way must be empty for this action to succeed. Movement in straight line costs 10 Movement Points, the diagonal movement costs 14 Movement Points. If the unit occupies 4 fields, then each time 4 fields need to be empty for the move to be possible.
- **Attack** – Attack an enemy unit. If the enemy is on the field next to the stack, the attack becomes a melee combat. This may trigger a melee counterattack if the attacked unit survives and can still counterattack in this cycle. The melee attack may occur after a movement (in the same turn).
  If the enemy is further away, the attack becomes ranged. It may occur only if the unit is capable of ranged attacks, has not moved this cycle, and no field in the closest neighborhood is occupied by an enemy unit. The ranged attack will not trigger a counterattack. A ranged unit fighting in melee suffers additional 50% penalty to damage dealt. Whenever the whole stack is eliminated from the game, the field(s) occupied by it become(s) available again.

No action will be performed, if the unit which is allowed to make action in the current turn would not be given any command.

The skirmish phase ends when the battle time elapses (the time limit is known to both players), no matter which units are still standing. Even if one of the armies is eliminated, the timer counts down to the end of the battle.

Damage calculation If one stack of units attacks another, the damage dealt are calculated by the following formula:

$$D_{received} = N \cdot D_{unit} \cdot f, \tag{6}$$

where $D_{received}$ is the number of damage points the defender receives, N is the attacking stack unit count, D_{unit} is `Damage` parameter of the attacking unit. The f factor is calculated as follows:

$$f = 1 + \frac{1}{50}(ATK - DEF), \tag{7}$$

where ATK is the `Attack` or `Ranged Attack` value of the attacker (depends on the attack type) and DEF is the `Defense` value of the defender. The value f must fit in the range of $[\frac{1}{2}, 2]$. If it does not, it is changed to the closer boundary of the range.

Whenever needed, the server uses floor function to convert a real value to an integer. That includes: calculation of the number of damage points received, all bonuses and penalties connected with traits.

7.5.4. Phase 4: Results

This phase serves one purpose – providing the players with information regarding the last skirmish turn.

7.6. The aim of the game and scoring

Your main goal is to maximize your score for all the games you play.

Scoring Several kinds of activities are given points during each battle:

- Dealing damage in an attack (W_A).
- Dealing damage in a counterattack (W_C).
- Eliminating an enemy unit (W_U).
- Eliminating an entire stack of enemy units (W_K).
- Eliminating an entire enemy army (W_V).

The points given for a certain type of action may vary among different sets of games. The values W_A and W_C are given for every damage point dealt. The value W_V might be given only once at the end of the battle.

The sum of all points received within a single game produces a **battle result**. If the battle result value is negative, then it becomes 0. If a player does not issue any commands, then the battle result also becomes 0 in the corresponding game. Laziness is a sin in the Universum.

The four battle results of a team (each from one of the simultaneously played games) sum up to the **cumulative result**. Whenever all phases of a set of games end, a separate ranking for all competing teams is created. The ranking is based on the cumulative results of all teams. The team which occupies the i-th place in the ranking is given the final **score** according to Table 5. If the team cumulative result value is equal 0, then the team does not take part in the ranking and their score is also 0.

Table 6: Team places in the ranking and the corresponding score values

Rank	1	2	3	4	5	6	7	8	9	10
Score	100	80	60	50	45	40	36	32	29	26
Rank	11	12	13	14	15	16	17	18	19	20
Score	24	22	20	18	16	15	14	13	12	11
Rank	21	22	23	24	25	26	27	28	29	30
Score	10	9	8	7	6	5	4	3	2	1

7.7. Commands

General guidelines on the communication protocol (connecting, logging in, commands, response format) are described in section *Server communication*. Below you can find commands for the problem DESK.

In all cases where the "upper left corner" is mentioned, it concerns the field with the minimum coordinates; for example, for a unit occupying fields (1,1), (1,2), (2,1), (2,2), the upper left corner is (1,1).

As far as action commands are concerned (`DEFEND`, `DELAY`, `ATTACK`, `MOVE`), the parameter ID_S is required only to avoid accidental usage of the command with another stack.

DESCRIBE_GAME Returns the game parameters and the value of the score scaling coefficient.

Parameters: none

Data (from server):

The server returns two lines containing the following values separated with a single whitespace:

First line of the server response:

- T ($T \in \mathbb{R}, 1 \leqslant T \leqslant 2$) — length of turn in seconds,
- K ($K \in \mathbb{R}, 1 \leqslant K \leqslant 8$) — score scaling coefficient,
- L_1 ($L_1 \in \mathbb{N}, 1 \leqslant L_1 \leqslant 10$) — length of phase 1 (Preparation) in turns,
- L_2 ($L_2 \in \mathbb{N}, 1 \leqslant L_2 \leqslant 10$) — length of phase 2 (Tactics) in turns,
- L_3 ($L_3 \in \mathbb{N}, 1 \leqslant L_3 \leqslant 300$) — length of phase 3 (Skirmish) in turns,
- L_4 ($L_4 \in \mathbb{N}, 1 \leqslant L_4 \leqslant 3$) — length of phase 4 (Results) in turns.

Second line of the server response:

- W_A ($W_A \in \mathbb{R}, -1000 \leqslant W_A \leqslant 1000$) — points for attack damage; this is multiplied by the number of damage points dealt,
- W_C ($W_C \in \mathbb{R}, -1000 \leqslant W_C \leqslant 1000$) — points for counterattack damage; this is multiplied by the number of damage points dealt,
- W_U ($W_U \in \mathbb{R}, -1000 \leqslant W_U \leqslant 1000$) — points for unit kill; this is multiplied by the number of units killed,
- W_K ($W_K \in \mathbb{R}, -1000 \leqslant W_K \leqslant 1000$) — points for stack kill; this is multiplied by the number of stacks killed,
- W_V ($W_V \in \mathbb{R}, -1000 \leqslant W_V \leqslant 1000$) — points for victory; this is given once at the end of the set of games.

MY_ID Returns team ID in the game.

Parameters: none

Data (from server):

- ID ($ID \in \mathbb{N}, 1 \leqslant ID \leqslant 100$)

RIVALS Returns IDs of the opponents the team is currently playing against, and IDs of the battles the set of games is based on.

Parameters: none

Data (from server):

- N ($N \in \mathbb{N}, 0 \leqslant N \leqslant 4$) — number of opponents the team is currently fighting,
- N sets of three values, separated by single whitespace: ID_e ($ID_e \in \mathbb{N}, 1 \leqslant ID_e \leqslant 100$) denoting enemy ID, ID_B ($ID_B \in \mathbb{N}, 1 \leqslant ID_B \leqslant 1000$) denoting battle ID, and $Side$ ($Side \in \{\texttt{ATK}, \texttt{DEF}\}$) denoting whether the team is attacking or defending.

CURRENT_STAGE Returns the name of the current phase and the number of turns before it ends.

Parameters: none

Data (from server):

- *Phase* ($Phase \in \{$`PREPARATION`, `TACTICS`, `SKIRMISH`, `RESULTS`$\}$) — name of the phase,
- L ($L \in \mathbb{N}, L \geqslant 0$) — number of turns before the phase ends (excluding the current turn).

SHOW_BOARD Returns the current terrain map – the board. The map is the same for all current battles. This map does not contain unit placement.

Parameters: none

Data (from server): The server returns 10 lines with 12 characters in each line. Each character indicates one field and may take one of two values:

- `#` — inaccessible field,
- `.` — empty field.

ALL_UNITS Returns all available units and their numbers. This command may be called only during `PREPARATION`.

Parameters: none

Data (from server):

- N_U ($N_U \in \mathbb{N}, 1 \leqslant N_U \leqslant 10$) — The number of unit types.

Next $N_U \cdot 5$ lines contain units descriptions, one type of unit is described by 5 lines:

First line of the unit description:

- ID_U ($ID_U \in \mathbb{N}, 1 \leqslant ID_U \leqslant N_U$) — unit type ID.

Second line of the unit description:

- U_{HP} ($U_{HP} \in \mathbb{N}, 1 \leqslant U_{HP} \leqslant 1000$) — Hit Points attribute,
- U_{INIT} ($U_{INIT} \in \mathbb{N}, 1 \leqslant U_{INIT} \leqslant 1000$) — Initiative attribute,
- U_{MOVE} ($U_{MOVE} \in \mathbb{N}, 1 \leqslant U_{MOVE} \leqslant 1000$) — Movement Points attribute,
- U_{ATK} ($U_{ATK} \in \mathbb{N}, 1 \leqslant U_{ATK} \leqslant 1000$) — melee Attack attribute,
- U_{DEF} ($U_{DEF} \in \mathbb{N}, 1 \leqslant U_{DEF} \leqslant 1000$) — Defense attribute,
- U_{DMG} ($U_{DMG} \in \mathbb{N}, 1 \leqslant U_{DMG} \leqslant 1000$) — Damage attribute,
- U_{RNG_ATK} ($U_{RNG\_ATK} \in \mathbb{N}, 0 \leqslant U_{RNG\_ATK} \leqslant 1000$) — Ranged Attack attribute.

Third line of the unit description:

- T_C ($T_C \in \mathbb{N}, 0 \leqslant T_C \leqslant 12$) — number of traits the unit has,
- T_C numbers denoting unit traits, each in form of T_i ($T_i \in \mathbb{N}, 1 \leqslant T_i \leqslant 12$, $i \in \mathbb{N}, 1 \leqslant i \leqslant T_C$).

Fourth line of the unit description:

- T_A ($T_A \in \mathbb{N}, 0 \leqslant T_A \leqslant 12$) — how many more traits can be given to this unit.

Fifth line of the unit description:

- K ($K \in \mathbb{N}, 1 \leqslant K \leqslant 10000$)— the number of available units of a given type.

ALL_ABILITIES All traits that can be given to units. Can be called only during `PREPARATION`.

Parameters: none

Data (from server):

- T_N ($T_N \in \mathbb{N}, T_N \geqslant 0$) — the number of available traits,

- T_N numbers denoting available traits, each in form of T_i ($T_i \in \mathbb{N}, 1 \leqslant T_i \leqslant 12$, $i \in \mathbb{N}, 1 \leqslant i \leqslant T_N$); numbers are not necessarily unique or in a particular order.

ASSIGN_ABILITIES Assigns trait to the unit type. This command may be called only during `PREPARATION`. Any proper call will overwrite traits assignment done with this command previously for a given battle.

Parameters:

- ID_B ($ID_B \in \mathbb{N}, 1 \leqslant ID_B \leqslant 1000$) — ID of the battle,
- A_P — The number of assignment pairs of unit and trait,
- A_P pairs of numbers, separated by whitespaces, in a form of ID_U ID_T ($ID_U \in \mathbb{N}, 1 \leqslant ID_U \leqslant 10$, $ID_T \in \mathbb{N}, 1 \leqslant ID_T \leqslant 12$), where ID_U is the identifier of the unit type and ID_T is the trait assigned to the unit type.

PLACE_UNITS_ON_BOARD Sets stacks on the board. This command may be called only during `PREPARATION`. Any proper call will overwrite units placement done with this command previously for a given battle.

Parameters:

- ID_B ($ID_B \in \mathbb{N}, 1 \leqslant ID_B \leqslant 1000$) — ID of the battle,
- S_C ($S_C \in \mathbb{N}, 1 \leqslant S_C \leqslant 8$) — the number of stacks to place
- S_C of 4-number sets separated by single whitespace, all values separated by single whitespace: ID_U ($ID_U \in \mathbb{N}, 1 \leqslant ID_U \leqslant 10$) – unit type identifier, S_U ($S_U \in \mathbb{N}$, $1 \leqslant S_U \leqslant 10000$) – how many units should be placed in the stack, and two coordinates U_X, U_Y ($U_X \in \mathbb{N}$, $1 \leqslant U_X \leqslant 12$; $U_Y \in \mathbb{N}$, $1 \leqslant U_Y \leqslant 10$), denoting position of the stack. In case of units of 2×2 size, the coordinates of the upper left corner of the unit should be given.

UNIT_TYPES Shows all unit types in all battles currently fought. This command may be called only during `TACTICS` or `SKIRMISH`, and only once every 10 turns.

Parameters: none

Data (from server): First line contains:

- C_C ($C_C \in \mathbb{N}, C_C \geqslant 0$) — the number of all units types within all battles.

Next $C_C \cdot 3$ lines contain unit type descriptions; each unit type is described by 3 lines:

First line of the unit description:

- ID_B ($ID_B \in \mathbb{N}, 1 \leqslant ID_B \leqslant 1000$) — ID of the battle the unit takes part in,
- ID_N ($ID_N \in \mathbb{N}, 1 \leqslant ID_N \leqslant 20$) — ID of the unit type within the given battle (a new one),
- ID_U ($ID_U \in \mathbb{N}, 1 \leqslant ID_U \leqslant 10$) — ID of the unit type this unit came from (this is value ID_U from ALL_UNITS).

Second line of the unit description (unit attributes with all bonuses calculated in):

- U_{HP} ($U_{HP} \in \mathbb{N}, 1 \leqslant U_{HP} \leqslant 1000$) — Hit Points attribute,
- U_{INIT} ($U_{INIT} \in \mathbb{N}, 1 \leqslant U_{INIT} \leqslant 1000$) — Initiative attribute,
- U_{MOVE} ($U_{MOVE} \in \mathbb{N}, 1 \leqslant U_{MOVE} \leqslant 1000$) — Movement attribute,
- U_{ATK} ($U_{ATK} \in \mathbb{N}, 1 \leqslant U_{ATK} \leqslant 1000$) — melee Attack attribute,
- U_{DEF} ($U_{DEF} \in \mathbb{N}, 1 \leqslant U_{DEF} \leqslant 1000$) — Defense attribute,
- U_{DMG} ($U_{DMG} \in \mathbb{N}, 1 \leqslant U_{DMG} \leqslant 1000$) — Damage attribute,
- U_{RNG_ATK} ($U_{RNG\_ATK} \in \mathbb{N}, 0 \leqslant U_{RNG\_ATK} \leqslant 1000$) — Ranged Attack attribute.

Third line of the unit description (unit traits):

- T_C ($T_C \in \mathbb{N}, 0 \leqslant T_C \leqslant 12$) — number of traits the unit has,

- T_C numbers denoting unit traits, each in a form of T_i ($T_i \in \mathbb{N}, 1 \leqslant T_i \leqslant 12$, $i \in \mathbb{N}, 1 \leqslant i \leqslant T_C$).

UNITS_ON_BOARD Returns current positions of stacks on the board. This command may be called only during `TACTICS` and `SKIRMISH`.

Parameters:

- ID_B ($ID_B \in \mathbb{N}, 1 \leqslant ID_B \leqslant 1000$) — ID of the battle.

Data (from server):

First line of the response:

- S_C ($S_C \in \mathbb{N}, 0 \leqslant S_C$) — the number of stacks still in the battle.

Next S_C lines describe one stack each:

- ID_S ($ID_S \in \mathbb{N}, 1 \leqslant ID_S \leqslant 30$) — stack ID, unique within the battle,
- ID_O ($ID_O \in \mathbb{N}, 1 \leqslant ID_O \leqslant 100$) — stack's owner ID,
- ID_N ($ID_N \in \mathbb{N}, 1 \leqslant ID_N \leqslant 100$) — ID of the unit type that builds the stack (this ID binds the stack with unit type from UNIT_TYPES),
- C ($C \in \mathbb{N}, C > 0$) — the number of units in the stack,
- H ($H \in \mathbb{N}, H > 0$) — the number of Hit Points left to the unit on the top of the stack,
- S_X ($S_X \in \mathbb{N}, 1 \leqslant S_X \leqslant 12$) — X coordinate (of the upper left corner) of the unit,
- S_Y ($S_Y \in \mathbb{N}, 1 \leqslant S_Y \leqslant 10$) — Y coordinate (of the upper left corner) of the unit.

UNIT_QUEUE Queue to orders in a given cycle; every stack is listed here not more than once. The queue changes every time someone issues DELAY order or a stack dies. This command may be called only during `SKIRMISH`.

Parameters:

- ID_B ($ID_B \in \mathbb{N}, 1 \leqslant ID_B \leqslant 1000$) — ID of the battle.

Data (from server):

- S_C ($S_C \in \mathbb{N}, 0 \leqslant S_C$) — the number of stacks awaiting orders in this cycle,
- S_C of ID_S ($ID_S \in \mathbb{N}, 1 \leqslant ID_S \leqslant 16$) – IDs of the stacks that still wait for orders, separated by single whitespace; the first one on the list is now to act.

DEFEND Orders the unit to defend, increasing its defense by 5 until its next cycle. This command may be called only during `SKIRMISH`.

Parameters:

- ID_B ($ID_B \in \mathbb{N}, 1 \leqslant ID_B \leqslant 1000$) — ID of the battle,
- ID_S ($ID_S \in \mathbb{N}, 1 \leqslant ID_S \leqslant 30$) — ID of the stack the order is given to.

DELAY Orders the unit to wait. This command may be called only during `SKIRMISH`.

Parameters:

- ID_B ($ID_B \in \mathbb{N}, 1 \leqslant ID_B \leqslant 1000$) — ID of the battle,
- ID_S ($ID_S \in \mathbb{N}, 1 \leqslant ID_S \leqslant 30$) — ID of the stack the order is given to.

ATTACK Orders the unit to attack. This command may be called only during `SKIRMISH`. Note: if coordinates indicate a field that is not a neighbor of the unit, the attack becomes ranged.

Parameters:

- ID_B ($ID_B \in \mathbb{N}, 1 \leqslant ID_B \leqslant 1000$) — ID of the battle,

- ID_S ($ID_S \in \mathbb{N}, 1 \leqslant ID_S \leqslant 30$) — ID of the stack the order is given to,
- S_X ($S_X \in \mathbb{N}, 1 \leqslant S_X \leqslant 12$) — X coordinate of the target field,
- S_Y ($S_Y \in \mathbb{N}, 1 \leqslant S_Y \leqslant 10$) — Y coordinate of the target field.

MOVE Orders the unit to move. This command may be called only during `SKIRMISH`.

Parameters:

- ID_B ($ID_B \in \mathbb{N}, 1 \leqslant ID_B \leqslant 1000$) — ID of the battle,
- ID_S ($ID_S \in \mathbb{N}, 1 \leqslant ID_S \leqslant 30$) — ID of the stack the order is given to,
- F_C ($F_C \in \mathbb{N}, 1 \leqslant F_C \leqslant 30$) — number of sequential fields the unit must cross to its destination,
- F_C pairs of S_X S_Y ($S_X \in \mathbb{N}, 1 \leqslant S_X \leqslant 12$, $S_Y \in \mathbb{N}, 1 \leqslant S_Y \leqslant 10$) — X and Y coordinates of the field crossed; for units of 2×2 size, those are coordinates of the upper left corner. Those fields must be neighboring each other sequentially.

LAST_TURN Returns report about last turn events. This command may be called only during `SKIRMISH` and `RESULTS`.

Parameters:

- ID_B ($ID_B \in \mathbb{N}, 1 \leqslant ID_B \leqslant 1000$) — ID of the battle.

Data (from server):

- E_C ($E_C \in \mathbb{N}, 0 \leqslant E_C$) — number of the previous turn events.

Next E_C lines describe events that occurred, one event a line. Each line can take one and only one of the following forms:

- `MOVED` ID X Y ($ID \in \mathbb{N}, 1 \leqslant ID \leqslant 30$; $X \in \mathbb{N}, 1 \leqslant X \leqslant 12$; $Y \in \mathbb{N}, 1 \leqslant Y \leqslant 10$) — stack with ID moved to coordinates (X, Y),
- `ATTACKED` ID ID_V Dmg ($ID \in \mathbb{N}, 1 \leqslant ID \leqslant 30$; $ID_V \in \mathbb{N}, 1 \leqslant ID \leqslant 30$; $Dmg \in \mathbb{N}, Dmg > 0$) — stack ID attacked stack ID_V, dealing Dmg damage to it,
- `COUNTERED` ID ID_V Dmg ($ID \in \mathbb{N}, 1 \leqslant ID \leqslant 30$; $ID_V \in \mathbb{N}, 1 \leqslant ID \leqslant 30$; $Dmg \in \mathbb{N}, Dmg > 0$) — stack ID counterattacked stack ID_V, dealing Dmg damage to it,
- `DESTROYED` ID ($ID \in \mathbb{N}, 1 \leqslant ID \leqslant 30$) — stack ID has been destroyed,
- `DEFENDED` ID ($ID \in \mathbb{N}, 1 \leqslant ID \leqslant 30$) — stack ID is defending,
- `DELAYED` ID ($ID \in \mathbb{N}, 1 \leqslant ID \leqslant 30$) — stack ID is waiting.

MY_SCORE Returns your team score for the current set of games. This command may be called only during `RESULTS`.

Parameters: none

Data (from server):

First line of the response contains one value:

- S ($S \in \mathbb{N}$) — your score for the current set of games (based on your rank of four simultaneous games together).

Each of the next 4 lines describe one battle with values separated with single whitespaces:

- ID_B ($ID_B \in \mathbb{N}, 1 \leqslant ID_B \leqslant 1000$) — ID of the battle,
- S_{my} ($S_{my} \in \mathbb{R}$) — the battle result of your team,
- S_{opp} ($S_{opp} \in \mathbb{R}$) — the battle result of your opponent.

WAIT Waits till the next turn begins.

Parameters: none

Data (from server):

The server returns a single line of characters:

- WAITING S

where S ($S \in \mathbb{R}, S \geqslant 0$) stands for the number of seconds left to wait. Once the period is over, the server sends an additional line:

- OK

7.8. Errors

In case the command is incorrect, the server responds according to the description in section *Server communication* with the following message:

- '`FAILED` *e msg*',

where *e* is an error code, and *msg* — an error message. The table below consists of errors that may occur during communication process in problem DESK.

error code	error message
1	bad login or password
2	unknown command
3	bad format
4	too many arguments
5	internal error, sorry...
6	commands limit reached, forced waiting activated
101	improper current turn stage
102	too many abilities assigned
103	you do not take part in the skirmish with given id
104	it is not this stack turn
105	you are not this stack owner
106	the stack has already done something
107	the stack has already delayed the turn
108	too long distance to move
109	too many stacks placed on the board
110	a unit is placed outside the starting rectangle
111	a unit is placed on an unavailable field
112	some units occupy the same field
113	not all features are available
114	too many features assigned to a unit
115	no unit with the given id exists
116	too many units of a given type were trying to be used
117	no unit to attack on a given field
118	cannot attack own unit
119	range attack is not possible now
120	not all subsequent fields are neighbors
121	impassable field on the route to move
122	stack cannot be empty
123	too many calls within a specific turns period
124	there is no stack with given id on the board

7.9. Servers

The games will be held on servers with different parameters.

Table 7: Server addresses and parameters.

Name	**Address:Port**	**Team influence on the unit traits assignment**
Desk1	`universum.dl24:20003`	None
Desk2	`universum.dl24:20004`	Major

7.10. Example

Below you will find an exemplary record of the communication with the server.

client → server	server → client
	`LOGIN`
`login1`	
	`PASS`
`secret`	
	`OK`
`CURRENT_STAGE`	
	`OK`
	`PREPARATION 0`
`DESCRIBE_GAME`	
	`OK`
	`1 4.064634 1 1 15 1`
	`1.000 1.000 1.000 1.000 1.000`
`MY_ID`	
	`OK`
	`1`
`RIVALS`	
	`OK`
	`4`
	`2 1 ATK`
	`3 2 ATK`
	`2 4 DEF`
	`3 5 DEF`
`SHOW_BOARD`	
	`OK`
	`...#....#...`
	`...#....#...`
	`...#....#...`
	`............`
	`............`
	`............`
	`............`
	`...#....#...`
	`...#....#...`
	`...#....#...`

client → server	server → client
ALL_UNITS	
	OK 6 1 5 4 45 1 1 5 20 0 1 200 2 6 4 48 1 1 5 23 0 1 166 3 29 27 94 14 16 25 0 4 5 1 2 4 1 34 4 172 28 67 37 21 79 45 2 7 1 1 5 5 25 27 89 13 15 25 0 4 5 1 2 4 1 40 6 106 12 47 11 45 26 0 1 8 1 9
ALL_ABILITIES	
	OK 6 2 2 3 4 5 8
ASSIGN_ABILITIES 1 2 1 2 2 2	
	OK
PLACE_UNITS_ON_BOARD 1 6 1 200 1 1 ↪ 2 166 1 2 6 9 1 3 5 40 1 4 ↪ 3 34 1 6 4 5 1 8	
	OK
WAIT	
	OK WAITING 0.833000 OK

client → server	server → client
CURRENT_STAGE	
	OK
	TACTICS 0
WAIT	
	OK
	WAITING 0.988000
	OK
CURRENT_STAGE	
	OK
	SKIRMISH 14
UNIT_TYPES	
	OK
	48
	1 1 1
	5 4 45 1 1 5 20
	1 2
	1 2 2
	6 4 48 1 1 5 23
	1 2
	1 6 6
	106 12 47 11 45 26 0
	1 8
	1 5 5
	25 27 89 13 15 25 0
	4 5 1 2 4
	1 3 3
	29 27 94 14 16 25 0
	4 5 1 2 4
	1 4 4
	172 28 67 37 21 79 45
	2 7 1
	1 11 1
	5 4 45 1 1 5 20
	0
	1 12 2
	6 4 48 1 1 5 23
	0
	1 13 3
	29 27 94 14 16 25 0
	4 5 1 2 4
	1 14 4
	172 28 67 37 21 79 45
	2 7 1
	1 15 5
	25 27 89 13 15 25 0
	4 5 1 2 4
	1 16 6
	106 12 47 11 45 26 0
	1 8
	Please note that the rest of this command's output has been omitted.

client → server	server → client
`UNITS_ON_BOARD 1`	
	`OK`
	`12`
	`1 1 1 200 5 1 1`
	`2 1 2 166 6 1 2`
	`3 1 6 9 106 1 3`
	`4 1 5 40 25 1 4`
	`5 1 3 34 29 1 6`
	`6 1 4 5 172 1 8`
	`7 2 11 200 5 10 1`
	`8 2 12 166 6 10 2`
	`9 2 13 34 29 11 1`
	`10 2 14 5 172 11 3`
	`11 2 15 40 25 11 5`
	`12 2 16 9 106 10 3`
`UNIT_QUEUE 1`	
	`OK`
	`12 6 10 4 5 9 11 3 12 1 2 7 8`
`ATTACK 1 6 10 1`	
	`OK`
`ATTACK 1 6 10 2`	
	`FAILED 106 the stack has already done something`
`WAIT`	
	`OK`
	`WAITING 0.829000`
	`OK`
`CURRENT_STAGE`	
	`OK`
	`SKIRMISH 13`
`LAST_TURN 1`	
	`OK`
	`1`
	`ATTACKED 6 7 742`

8. Interstellar overdrive

8.1. Introduction

A common image of a beetlejumper in the mind of an average citizen of the Universum is one of a relentless conqueror and a great warrior. While it is often true, it is an incomplete image. Beetlejumpers are also good organizers and great bureaucrats. Sometimes they even make good traders and economists.

Recent conquest of *Sector D2L4* left a lot of chaos and destruction in its wake. This sector's resources are gravely needed for the future conquests. Thus Beetlejumper General Command, after making sure they control the area, renamed its administration to Beetlejumper Civil Command with orders to restore the economy as soon as possible.

Beetlejumper Civil Command went about it in a simple, military manner (after all, it was an institution comprised of retired officers). They took a contingent of Beetlejumper Military Academy, and turned them into *Trade Officers* (traders in short) and *Trade Commanders*, who were ordered to restore commerce to the area. This is all to achieve some nice revenue on the side to finance future campaigns.

8.2. Problem

Your team encompasses well-experienced Trade Commanders. Under your control, traders (Trade Officers) should operate on recovering the glory of planets, invest in them and collect revenue.

Your main goal is to maximize your profit from reestablished trade routes and to become the best Trade Commander – the one with most money collected at the end of the operation.

8.3. Game model

The game is divided into turns of equal duration. During each turn, teams communicate with the server and give orders to their units. Once the orders have been executed, world parameters are recalculated and a new turn begins.

8.4. World

The game takes place in *Sector D2L4* that is undergoing the revitalization process. The sector is represented by the $N \times M$ area, which is a two-dimensional orthographic projection of this part of the Universum.

The game area contains a number of stars and **planets**. Each planet orbits a star and it is a separate trading post, which can be a source of revenue for beetlejumpers. Stars cannot make profit. Thus, planets are the only points of interest for traders.

Trade routes Planets are connected with each other by predetermined, one-directional paths that are called *trade routes*. These routes might be used to redirect a trade income from one planet to another. Most of the planets have at least one route connected to it. Cases of isolated set of planets may happen, but they are rare. It is possible for a planet to have only incoming or only outgoing routes, or to have

a huge disproportion between the numbers of these routes. The routes have effectively unlimited capacity – it is not possible to clog a route with trade.

World in motion All objects are in motion – stars change their locations, and planets orbit the stars. This part of the Universum is characterized by very simple laws of physics to move celestial bodies. The laws are consistent during all the games. However, they will not be announced by beetlejumper engineers at any time.

Likewise, no maps of the game area are provided. The discovery of star location each planet is moving around, current (or future) locations of all planets, and their connections with each other is Trade Commander responsibility.

8.5. Traders

So as to rejuvenate the sector's economy, a **group of traders** is given to each team. These beetlejumpers are trained officers that can manage teams' affairs on the planet. They can either transfer the revenue to another planet or collect it as cash. The collected cash can be used for further investments (buying shares or facilities – see below) or left to contribute to the final score.

A trader can be placed on any planet which does not already have a trader commanded by the same team. In other words, every planet can host any number of traders, as long as each of them belongs to a different team. A planet does not have to host any traders and this is the initial state as every game begins. A trader that is not placed on any planet does not provide any profit to anyone.

There is no way to increase the number of traders a team owns.

Movement of traders Each trader can be relocated to another planet as long as they have not been given any orders recently. There is a short period of accommodation ($DelayNo$ number of turns) in which a trader is too busy with executing recent movement action to respond to any new commands.

8.6. Trade

Teams invest in specific places, support traders to manage revenue, and try to boost their global production. The trade value is generated on the planets and can either be transferred along the trade routes to the next planet or cashed in to provide money directly. Both of these operations require a trader to be present on the planet.

There are numerous bonuses and penalties that influence the trade, usually expressed in %. All bonuses are summed up linearly. For example, getting +40%, +30% and −35%, will give the final bonus of +35%. A bonus is applied to the base trade value of the planet and contributes to the final value available to the traders on the planet – to either collect (and increase the score) or to transfer further, as each of them decides.

8.6.1. Individual planet trade

Every planet is a trading post. Therefore, the terms "planet" and "trading post" can be used interchangeably. Every trading post has a basic, randomly assigned, **local income** value. This is the amount of trade generated on a given planet every turn. The local income, together with trade that may be redirected from other trading posts, form the **trade output** of the planet.

The local income of a planet may change a bit randomly by a limited amount, as the local economy boosts or collapses. The trend of these changes remains the same during the whole game for a given planet. Teams may also have impact on the local income value – it may be increased by placing a production facility on the planet (see below).

Investing in shares Each planet can host a number of traders. Therefore, **shares** are used to distribute the total trade output of the planet. Each trader controls only a part of the planet trade output. The amount of revenue the trader is able to use is determined by the team's share in the planet (together with bonuses and penalties).

Every team can invest their cash in shares in the planet. There is no limit for the amount of shares a team can purchase on any planet. The trade value offered to a team is a result of the percentage

calculated in relation to other teams. Therefore, if a team invested 1 share in the planet and no one else owns any, they control 100% of the planet trade output (assuming no efficiency drops – the details are given below). If two teams invested in 1 share each, and no one else does, each of them has control over 50% of the trade output. This percentage indicates how much of the planet's trade output can the team's trader control once it takes its place in the trading post.

Teams may receive free vouchers for planet investments – the Command is eager to get the traders straight out into the wilderness, after all. Each voucher can be exchanged for a single planet share without any additional cost. Buying a share will automatically decrease the number of vouchers a team owns or (if none of them left) reduces the amount of cash that traders can spend.

8.6.2. Planet penalties

Fatigue Traders are not exactly welcome on the planets – for some reasons, they are seen as agents of the enemy power. If they remain in one location for too long, then their network of contacts starts to fray and break down, and they become less and less effective. Each turn after placing a beetlejumper on a planet, the effectiveness of the trader operations drops by a certain fraction – *Drop* percentage points – reducing the amount of trade a trader can effectively collect or send to another planet.

The value which represents the beetlejumper fatigue is called **efficiency**. Its base value for each pair team-planet is 100%. The current efficiency influences the actual (effective) trade distribution of a planet (how much of the trade output a single trader can send or cash in).

This value is team-based: should the trader be replaced by another one from the same team, the efficiency will not change. Only retreating for a certain amount of time will allow the planet's denizens to forget about issues they had. If that happens, for each team-planet pair efficiency is restored to the previous level with the speed of *Rise* percentage points per turn. The efficiency value cannot be greater than 100% nor smaller than zero.

Crowded planets Traders, being beetlejumpers, are extremely competitive. This rarely looks good in the eyes of planets denizens – conflicts are not welcome on war-ravaged planets. Every trader present on the planet reduces the local income generated on the planet by $CF\%$. Therefore, if there are $\lceil \frac{100}{CF} \rceil$ traders on the planet and there are no facilities (see below), then the value generated by this planet drops to zero.

Only the local income can be reduced – it has no impact on trade outputs redirected from other planets.

8.6.3. Facilities

A production facility is a building that can be purchased and placed on a specific planet. It has a positive impact on the planet and its surroundings, increasing their local income value.

Facilities cannot be relocated (moved to another planet) and shall remain on the planet until the end of the operation. Nonetheless, the orbital movement of the planet does cause them to change their location in relation to other planets.

Impact zone The closer a location is to a planet with a facility, the stronger the positive effect. The planet a facility is on receives the local income percentage bonus of $FacBonusMax\%$. The nearest planet receives bonus of $(FacBonusMax - FacBonusDrop)\%$, the next one – $(FacBonusMax - 2 \cdot FacBonusDrop)\%$, and so forth. The bonus value stops decreasing once it reaches zero. There is no option for a draw in calculating distance to a facility – in an unlikely case of any two planets being equidistant from a given facility, the choice of influence is decided arbitrarily by Civil Command.

Common ownership The facilities are purchased by teams, but their influences cover the whole planet, no matter what the shares distribution is and no matter if the original buyer is still present on the planet. It is possible to give bonuses to the planets that the team does not have any traders on. Every planet may receive bonuses from any number of facilities – they sum up in a linear fashion.

Prices Facilities are purchased for the hard-earned cash and immediately placed in a chosen location. Due to laws of supply and demand, the facility prices increase globally in relation to the number of facilities purchased. Recalculation of price takes place with all other recalculations – at the end of the turn.

Therefore, two (or more) facilities purchased in the same turn will have the same price. The price of a facility in the current turn (F_{price}) is:

$$F_{price} = FacCost + F_N \cdot FacCostInc, \tag{8}$$

where $FacCost$ is a base facility price, $FacCostInc$ is a parameter indicating the price increment factor, and F_N is the number of facilities created till the current turn.

Limitations The materials used to build facilities are very rare. Moreover, beetlejumpers are running out of these resources, so there is a limit for a total number of facilities that can be placed in *Sector D2L4*. All teams cannot build more than F^G_{lim} facilities altogether, and there cannot be more than F^P_{lim} facilities on each planet.

Please note that due to facilities being ordered with delivery to a specific place, it is not possible to stock them for future deployment.

8.6.4. Interplanetary trade

Each trader placed on a planet may choose to send the collected profit to another planet instead of the instant cash in. The transfer needs to be done along the existing trade routes. This option may have a great positive impact on the final team score as there are several options which can increase the value sent, making it possible to cash in more money on the target planet.

However, planets are not stationary. They move as the operation progresses, changing their position relative to production facilities on other planets and changing trade routes profits. These changes are gradual, but do influence trade in a noticeable manner.

Distance bonus Each route gives a bonus value that improves the trade value of goods sent via it – the longer the distance, the higher the improvement. The distance bonus (DB) is calculated with the formula below:

$$DB = DB_A \cdot d^{DB_B}, \tag{9}$$

where d is the actual distance between the planets, DB_A and DB_B are the improvement coefficients.

The bonus is applied as the value added to the trade value sent using this route. This value changes as the planets move every turn.

Synergy Two heads are better than one – in most cases – even if they have different masters. A trader redirecting the value via a given route receives a $SB\%$ bonus to the value sent. For example, if two traders – from different teams, obviously – redirect the value along the same route, each of them receives a bonus of $2 \cdot SB\%$ each to the value sent. The synergy of more than two traders increases in a linear manner. This bonus is cumulative with the distance bonus in a linear manner.

Chain of traders A well-maintained chain of trade posts provides excellent business opportunities – not to mention, a certainty that the goods are going to end up in right hands. If the value is redirected once or more on subsequent planets by the same team (which requires several traders), the planets participating in the chain receive a bonus that grows with the chain length. This bonus **increases the effective number of shares** a team controls on a planet, not the trade value itself.

Each planet which trade output is transferred to the next planet (and on both planets there are traders of a given team) receives an additional CB chain bonus comparing with the next planet in chain. The last planet in chain (the one where trader is collecting cash) does not get any chain bonus.

For example, if there is a chain three planets long, then the first planet receives $2 \cdot CB\%$ increase, the second one receives $1 \cdot CB\%$, and the third (where the trade output is collected) receives no bonus. If the chain is five planets long, then the first planet receives $4 \cdot CB\%$ increase, second one – $3 \cdot CB\%$, and so on. The last planet in chain does not receive a bonus at all.

Keep in mind that the bonus does not change the actual number of shares, only the effective share value. For example, if the teams A and B have 1 share each – therefore control 50% of the planet's trade output – and the team A builds a chain that gives them a bonus of 5%, then is the team A considered to have the effective share of 1.05, and controls about 51.2% of the planet's value, while the team B control

drops to about 48.8%. If the chain breaks, then they will both revert to controlling 50% of the planet's power.

The bonus is adjusted during the share percentage calculation at the end of the turn.

8.6.5. Formulas

The formulas below complement the description of all bonuses and penalties.

Trade output collected from a single planet As stated previously, the local income of a planet might be modified with facilities and crowd on the planet. The modified (with bonuses and penalties), final value of local income becomes:

$$I_{bonus} = I_{local} \cdot \frac{100 + FB - n \cdot CF}{100}, \tag{10}$$

where I_{local} is the base local income of the planet, FB is a facility bonus (in %), n is the number of traders on the planet, and CF is a crowd factor (in %).

Cashing in a transferred trade If the trade is transferred to another planet, then more bonuses may be applied to the value sent. The final income that a single trader may cash in (I_{trader}) is a product of the effective number of shares in the planet, team effectiveness on the planet and a sum of planet local income and those transferred. Then we have:

$$I_{trader} = (I_{trans} + I_{bonus}) \cdot eff \cdot \frac{share_{trader}}{\sum_{i=1}^{n} share_i}, \tag{11}$$

where

$$\begin{aligned} I_{trans} &= \sum_{i=1}^{k} I_i \cdot \frac{100 + DB_i + SB_i}{100}, \\ share_i &= TeamShares_i \cdot \frac{100 + CB}{100} \end{aligned} \tag{12}$$

where DB, SB, and CB are the bonuses for distance, synergy, and chain, respectively; $TeamShares_i$ represent the number of shares a team owns for the planet, $share_i$ is the effective share value of the i-th trader/team (with possible chain bonus), I_i is the trade output sent from another planet via a single trade route, k is the number of transferred incomes from other planets, I_{trader} is the total income value (with all bonuses and penalties) a single trader can cash in or transfer, I_{trans} means the income value transferred from another planets, I_{bonus} is a modified local income of the planet (see above), eff is the team efficiency for the planet, n is the number of traders on the planet.

8.7. The beginning and the end

Initial state of the game At the beginning of the game, no one owns any shares on any planets and there are no facilities. Each team begins with the same number of traders, the same number of vouchers for shares and the same amount of cash for their future investments.

End of the game The traders have limited time to achieve advantage over the opponents. The game ends after a specified amount of time. There is no early ending condition.

8.8. The aim of the game and scoring

Your main goal is to maximize your profit in cash and outrun the opponents.

Scoring Only your cash profit matters for Beetlejumper Civil Command. Therefore, your current cash earned is the only criterion to evaluate your achievements. The score is based on the value rounded off to three decimal places:

$$S_{base} = \frac{C - C_{begin}}{T_{current}}, \tag{13}$$

where C is the amount of cash your team owns at the moment, C_{begin} is your initial amount of cash, and $T_{current}$ is a total cash owned by all teams at the very moment of a game. If the calculated value is negative, then the final score is 0.

$$S = max(S_{base}, 0). \tag{14}$$

8.9. Commands

General guidelines on the communication protocol (connecting, logging in, commands, response format) are described in section *Server communication*. Below you can find commands for the problem Interstellar overdrive.

DESCRIBE_WORLD Returns the game parameters and the value of the score scaling coefficient.

Parameters: none

Data (from server):

The server returns one line containing the following values separated with a single whitespace:

- N ($N \in \mathbb{R}$, $0 < N \leqslant 1000$) — the length of the side of the board (dimension X),
- M ($M \in \mathbb{R}$, $0 < M \leqslant 1000$) — the length of the side of the board (dimension Y),
- SPP ($SPP \in \mathbb{N}$, $0 < SPP \leqslant 10^7$) — the price of purchasing a share,
- SB ($SB \in \mathbb{N}$, $0 \leqslant SB \leqslant 1000$) — synergy bonus value, in %, for sending profit to the another planet,
- CB ($CB \in \mathbb{N}$, $0 \leqslant CB \leqslant 1000$) — chain bonus value, in %, for sending profit through chain of planets,
- DB_A ($DB_A \in \mathbb{R}$, $0 \leqslant DB_A \leqslant 100$) — distance bonus value for sending profit to another planet calculated using distance between planets – linear component expressed in %,
- DB_B ($DB_B \in \mathbb{R}$, $0 < DB_B \leqslant 2$) — distance bonus value for sending profit to another planet calculated using the distance between the planets – expotential component,
- CF ($CF \in \mathbb{N}$, $0 \leqslant CF \leqslant 100$) — crowding factor value, in %, which determines decreasing trade output for each trader located in the planet,
- $DelayNo$ ($DelayNo \in \mathbb{N}$, $0 \leqslant DelayNo \leqslant 100$) — the number of turns in which trader cannot perform or change any action,
- $Drop$ ($Drop \in \mathbb{R}$, $0 \leqslant Drop \leqslant 100$) — efficiency drop value, in % points, which determines decreasing team efficiency in specific planet when team's trader located in this planet,
- $Rise$ ($Rise \in \mathbb{R}$, $0 \leqslant Rise \leqslant 100$) — efficiency rise value, in % points, which determines increasing team efficiency in specific planet when team's trader is not located in this planet,
- $FacCost$ ($FacCost \in \mathbb{N}$, $0 < FacCost \leqslant 10^7$) — the cost of creating a facility,
- $FacCostInc$ ($FacCostInc \in \mathbb{N}$, $0 \leqslant FacCostInc \leqslant 10^7$) — value which determines how the cost of creating facility is increased for the next facility,
- $FacBonusMax$ ($FacBonusMax \in \mathbb{N}$, $0 \leqslant FacBonusMax \leqslant 10000$) — the bonus, in %, for a planet with a facility,
- $FacBonusDrop$ ($FacBonusDrop \in \mathbb{N}$, $0 < FacBonusDrop \leqslant FacBonusMax$) — value in % points which determines how the Facility Bonus is decreased for a planet which is further from the facility than other planets,
- F^P_{lim} ($F^P_{lim} \in \mathbb{N}$, $0 \leqslant F^P_{lim} \leqslant 100$) — facility limit per each planet,
- F^G_{lim} ($F^G_{lim} \in \mathbb{N}$, $0 \leqslant F^G_{lim} \leqslant 1000$) — global facility limit,
- T ($T \in \mathbb{N}$, $1 \leqslant T \leqslant 3$) — the duration of a single turn in seconds,
- L ($L \in \mathbb{N}$, $1 \leqslant L \leqslant 100$) — the maximum number of commands which can be issued in one turn,
- K ($K \in \mathbb{R}$, $1 \leqslant K \leqslant 8$) — the value of the score scaling coefficient.

TIME_TO_END Returns the number of turns till the end of the game (including the current turn).

Parameters: none

Data (from server):

The server returns a single value:

- T ($T \in \mathbb{N}$) — the number of turns till the end of the game.

GET_TOPOLOGY Returns the topology of all planets. It could be called once every 100 turns.

Parameters: none

Data (from server):

The first line returned by the server contains:

- N ($N \in \mathbb{N}$) — the number of planets.

The following lines describe the planet with the values separated with a single whitespace:

- $PlanetID$ ($PlanetID \in \mathbb{N}$) — planet's *ID*,
- M ($M \in \mathbb{N}$) — the number of outgoing branches,
- $PlanetIDs$ ($PlanetIDs \in \mathbb{N}$) — list of M planet identifiers which are connected with the $PlanetID$ planet, separated with a single whitespace.

GET_PLANETS_BASIC_INFO Returns the list of all planets. It could be called once every 10 turns.

Parameters: none

Data (from server):

The first line returned by the server contains:

- N ($N \in \mathbb{N}$) — the number of planets.

The following lines describe the planet with the values separated with a single whitespace:

- *ID* (*ID* $\in \mathbb{N}$) — planet's *ID*,
- X ($X \in \mathbb{R}$) — the X coordinate of the planet,
- Y ($Y \in \mathbb{R}$) — the Y coordinate of the planet,
- LC ($LC \in \mathbb{R}$) — local income of the planet.

GET_PLANET_DETAILS Returns the details of the planet. The result reflects the state from the beginning of the current turn.

Parameters:

- *ID* (*ID* $\in \mathbb{N}$) — the *ID* of the planet.

Data (from server):

The following lines describe the planet with the values separated with a single whitespace:

- YS ($YS \in \mathbb{N}$) — your shares in the planet (base value – no bonuses included),
- OS ($OS \in \mathbb{N}$) — shares of other teams invested in the planet (base value – no bonuses included),
- YE ($YE \in \mathbb{R}$) — your efficiency,
- N ($N \in \mathbb{N}$) — the number of traders operating on the planet.

The following lines describe all traders in the planet with the values separated with a single whitespace:

- *ID* (*ID* $\in \mathbb{N} \cup \{\texttt{Me}\}$) — team ID, `Me` in the case the trader belongs to the asking team,
- *DestID* (*DestID* $\in \mathbb{N} \cup \{\texttt{Collects}\}$) — destination planet ID if trader redirects the trade output or `Collects` if the trader collects it.

GET_FACILITIES Returns the list of all facilities. The result reflects the state from the beginning of the current turn.

Parameters: none

Data (from server):

The first line returned by the server contains:

- F_N ($F_N \in \mathbb{N}$) — the number of facilities.

The second line is returned when $F_N > 0$. It contains F_N values separated with single whitespace characters:

- $PlanetID$ ($PlanetID \in \mathbb{N}$) — *ID* of the planet the facility is on.

GET_TRADERS Returns the list of traders being under command of the specified team. The result reflects the state from the beginning of the current turn.

Parameters:

- *ID* ($ID \in \mathbb{N} \cup \{\texttt{MYSELF}\}$) — the *ID* of the team, `MYSELF` if asking about own traders.

Data (from server):
The first line returned by the server contains:

- G ($G \in \mathbb{N}$) — the number of traders.

The following lines describe the trader with the values separated with a single whitespace:

- *ID* ($ID \in \mathbb{N}$) — trader's *ID*,
- *PlanetID* ($PlanetID \in \mathbb{N} \cup \{\texttt{None}\}$) — planet's *ID* (`None` if the trader is idle),
- *TargetPlanetID* ($TargetPlanetID \in \mathbb{N} \cup \{\texttt{None}\}$) — *ID* of the planet trader is redirecting the trade to (`None` if the trader is collecting cash),
- $Next$ ($Next \in \mathbb{N}$) — the number of turns after which the next command is possible (how long the trader will remain busy).

GET_SHARES Returns the list of shares in planets for which shares are greater than zero. The result reflects the state from the beginning of the current turn.

Parameters:

- *ID* ($ID \in \mathbb{N} \cup \{\texttt{MYSELF}\}$) — the *ID* of the team, `MYSELF` if asking about own shares.

Data (from server):
The first line returned by the server contains:

- N ($N \in \mathbb{N}$) — the number of planets where the team has more shares than zero.

The following lines describe the team share in the planet with the values separated with a single whitespace:

- *ID* ($ID \in \mathbb{N}$) — planet's *ID*,
- P ($P \in \mathbb{N}$) — team's share in the planet. No bonuses from any source are included – this is a raw, base value.

GET_EFFICIENCIES Returns the list of efficiencies in planets for which efficiency is less than 100%.

Parameters:

- *ID* ($ID \in \mathbb{N} \cup \{\texttt{MYSELF}\}$) — the *ID* of the team, `MYSELF` if asking about own efficiencies.

Data (from server):
The first line returned by the server contains:

- N ($N \in \mathbb{N}$) — the number of planets where team has efficiency less than 100%.

The following lines describe the team efficiency in the planet with the values separated with a single whitespace:

- *ID* ($ID \in \mathbb{N}$) — planet's *ID*,

- E ($E \in \mathbb{R}$) — the efficiency of the team in the planet.

GET_SCORE Returns your cash and score. The result reflects the state from the beginning of the current turn.

Parameters: none

Data (from server):

The server returns one line containing the following values separated with a single whitespace:

- C ($C \in \mathbb{R}$) — the amount of cash your team owns,
- S ($S \in \mathbb{R}$) — score calculated as (your cash profit / amount of all cash in the game),
- I ($I \in \mathbb{N}$) — the number of vouchers to be exchanged for shares.

GET_TEAMS Returns some statistics about all teams.

Parameters: none

Data (from server):

The first line returned by the server contains:

- N ($N \in \mathbb{N}$) — number of teams.

The following lines describe each team with the values separated with a single whitespace:

- *ID* ($ID \in \mathbb{N} \cup \{\texttt{Me}\}$) — team *ID*, `Me` for your team,
- T_{cash} ($T_{cash} \in \mathbb{N}$) — the number of traders used to collect the cash,
- T_{send} ($T_{send} \in \mathbb{N}$) — the number of traders used to transfer trade output,
- $Shares$ ($Shares \in \mathbb{N}$) — total number of shares a team owns (regardless the planets; no bonuses).

PUT_TRADER Places an indicated trader on an indicated planet. After invoking this command, cash earned in current planet are collected automatically. If a trader is already on the chosen planet, then he will not be moved, but his mode of operation will change to collection.

Parameters:

- *TraderID* (*TraderID* $\in \mathbb{N}$) — the *ID* of the trader,
- *PlanetID* (*PlanetID* $\in \mathbb{N}$) — the *ID* of the planet where the trader will be placed.

Data (from server): none

SEND_PROFIT Sends profit between planets which are connected. This command will move the trader if he is not on source planet. The command will not work if the trader is busy.

Parameters:

- *TraderID* (*TraderID* $\in \mathbb{N}$) — the *ID* of the trader,
- *FromPlanetID* (*FromPlanetID* $\in \mathbb{N}$) — the *ID* of the planet where trader will be placed and profit will be send from (the ID of the planet where the trader currently resides is also a valid one),
- *DestPlanetID* (*DestPlanetID* $\in \mathbb{N}$) — the *ID* of the planet where profit will be send to.

Data (from server): none

BUILD_FACILITY Builds a facility on a planet.

Parameters:

- *ID* ($ID \in \mathbb{N}$) — the *ID* of the planet to build the facility on.

Data (from server): none

BUY_SHARES Increases share in the planet. It costs SPP for each share. If the team has any vouchers for shares, they are spent before incurring any monetary cost.

Parameters:

- ID ($ID \in \mathbb{N}$) — the ID of the planet,
- N ($N \in \mathbb{N}$) — the number of shares bought.

Data (from server): none

WAIT Waits till the next turn begins.

Parameters: none

Data (from server):

The server returns a single line of characters:

- WAITING S

where S ($S \in \mathbb{R}, S \geqslant 0$) stands for the number of seconds left to wait. Once the period is over, the server sends an additional line:

- OK

8.10. Errors

In case the command is incorrect, the server responds according to the description in section *Server communication* with the following message:

- '`FAILED` *e msg*',

where *e* is an error code, and *msg* — an error message. The table below consists of errors that may occur during communication process in problem Interstellar overdrive.

error code	error message
1	bad login or password
2	unknown command
3	bad format
4	too many arguments
5	internal error, sorry...
6	commands limit reached, forced waiting activated
101	invalid planet id
102	invalid team id
103	invalid trader id
104	team does not own trader
105	not enough cash
106	cannot transfer trade there
107	trader cannot operate
108	this command limit reached
109	your trader is already there
110	invalid quantity
131	facilities planet limit reached
132	facilities total limit reached

8.11. Servers

The games will be held on two servers. Each server represents a subspace of *Sector D2L4* with a common characteristics about games parameters – some world attributes will use only a narrowed scope of possible values.

Table 8: Server addresses and parameters.

Name	**Address:Port**	**Possible subspaces of *Sector D2L4***
Traders1	`universum.dl24:20005`	A, B, C
Traders2	`universum.dl24:20006`	D, E, F

8.12. Example

Below you will find an exemplary record of the communication with the server.

client → server	server → client
	`LOGIN`
`login1`	
	`PASS`
`secret`	
	`OK`
`DESCRIBE_WORLD`	
	`OK`
	`11.000000 11.000000 40 5 5 1.000000 0.800000 60 10`
	`↪ 2.000000 0.500000 2000 20 20 5 2 30 2 45 1.000186`
`GET_PLANETS_BASIC_INFO`	
	`OK`
	`3`
	`0 0.952737 1.045127 10.000000`
	`1 1.997316 0.103571 14.000000`
	`2 0.145938 2.996448 9.000000`
`GET_TOPOLOGY`	
	`OK`
	`3`
	`0 2 1 2`
	`1 1 2`
	`2 0`
`GET_TRADERS MYSELF`	
	`OK`
	`2`
	`41000000 None None 0`
	`41000001 None None 0`
`PUT_TRADER 41000000 2`	
	`OK`
`SEND_PROFIT 41000001 0 2`	
	`OK`
`BUY_SHARES 0 1`	
	`OK`
`BUY_SHARES 2 2`	
	`OK`
`WAIT`	
	`OK`
	`WAITING 00:00:01.998314`
	`OK`

```
GET_PLANET_DETAILS 0
                        OK
                        1 0 98 1
                        Me 2
GET_EFFICIENCIES MYSELF
                        OK
                        2
                        0 98.000000
                        2 98.000000
GET_SCORE
                        OK
                        887.876146 0.470304 0
GET_TRADERS MYSELF
                        OK
                        2
                        41000000 2 None 9
                        41000001 0 2 9
PUT_TRADER 41000000 1
                        FAILED 107 trader cannot operate
```