CodeWars 2017

Rules

Revision 1.2.0

November — December, 2017

Оглавление

Глава 1

The Announcement of the Competition

The limited liability company “Mail.Ru”, established and existing in accordance with the legislation of the Russian Federation and located at the address: 125167, Moscow, Leningradsky prospect, 39, building 79, hereinafter “The Organizer of the Competition”, invites individuals reached by the time of publication of this Announcement to 18 years, hereinafter “Participant”, to participate in the competition for the following conditions:

1.1 The Name Of The Competition

“Russian AI Cup”.

The purposes of the Competition:

- increasing public interest to creation of software;
- providing the Participants an opportunity to reveal their creative abilities;
- the development of professional skills of Participants.

The Competition consists of 3 (three) stages, each of which ends with the determination of the Winners. The last stage of the Competition is decisive.

1.2 Information about the Organizer of the Competition

Name: The LLC “Mail.Ru”

The address of the location: 125167, Moscow, Leningradsky prospect, 39, building 79,

Postal address: 125167, Moscow, Leningradsky prospect, 39, building 79, Business Center “SkyLight”

Phone number: (495) 725-63-57

Website: http://www.russianaicup.ru

E-mail: russianaicup@corp.mail.ru

1.3 The period of the Competition

The Competition period: from 00.00 hours on 7 November 2017 to 24.00 hours 24 December 2017 Moscow time.

First week (from 00.00 hours on 7 November 2017 to 24.00 hours on 12 November 2017) and fourth week (from 00.00 hours on November 27, 2017 to 24.00 hours 3 December 2017) of the Competition is testing. During this period, the functionality of the website and judging system of the Competition may be incomplete, and rules are subject to significant changes.

The timetable of the Competition:

- the first stage – from 00 hours 00 minutes on 25 November 2017 to 24 hours 00 minutes 26 November 2017;
- the second stage – from 00 hours 00 minutes on 9 December 2017 to 24 hours 00 minutes 10 December 2017;
- the third stage (final) – from 00 hours 00 minutes 16 December 2017 to 24 hours 00 minutes 17 December 2017.

1.4 The conditions for obtaining the status of the Participant

For participation in the Competition it is necessary to register in the System of the Organizer of the Competition. This System are available on the website of the Organizer in the Internet at the following address: http://www.russianaicup.ru.

1.5 The period of registration of Participants in the System of the Organizer

Registration of Participants will be held from 00.00 hours on 7 November 2017 to 24.00 hours on 24 December 2017 inclusively.

1.6 The territory of the Competition

The Competition is held on the territory of the Russian Federation. Conducting all stages of the Competition is carried out by remote access to the System of the Organizer via the Internet.

1.7 The conditions of the Competition (the essence of the tasks, criteria and evaluation procedure

The order of conducting of the Competition, the essence of the task, criteria and evaluation procedure specified in Chapter 2 of this document.

Documentation includes:

- The Announcement of the Competition;

- The Agreement on organization and conducting of the Competition;
- The Rules of the Competition;
- Information data, which are contained in the System of the Organizer of the Competition.

The Participant can view the documents on the website of the Organizer in the Internet at the following address: http://www.russianaicup.ru. Also the Participant can view the documents during the procedure of registration in the System of the Organizer of the Competition.

The Organizer of the Competition has the right to change the documentation and conditions and to refuse to conduct the Competition in accordance with the documentation and the provisions of the legislation of the Russian Federation. In this case, the Organizer should notify the Participants about all changes by sending a notice, in order and in the terms specified in the documentation.

1.8 The procedure of determining the Winners and award Prizes. The prize Fund of the Competition

Evaluation criteria of the Competition, the number and order of determining the Winners can be found in Chapter 2 of this document.

The prize Fund is formed at the expense of the Organizer of the Competition.

The prize Fund:

- 1st place — Apple Macbook Pro;
- 2nd place — Apple Macbook Air;
- 3rd place — Apple iPad;
- 4th place — Samsung Gear S3;
- 5th place — WD My Cloud 6 TB;
- 6th place — WD My Passport Ultra 4TB;
- 1-6 places in the Sandbox — WD My Passport Ultra 2TB.

All Participants who took part in the second or third stages, will be awarded a t-shirt. All Participants who took participation in the third stage, will also receive a hoodie with the logo of the competition.

All Participants, who will become winners, will be notified by sending a message to the email address, indicated during the registration in the System of the Organizer.

Prizes will be sent out to Participants as packages by the Russian Post or by other postal service during two months after the end of the final stage. Terms of delivery of the prize to the postal address specified by the Participant depends on the terms of delivery of the corresponding postal service. Postal addresses of the winners the Organizer receives from the credentials of Participant in the System of the Organizer. The address must be specified by the Participant prize-winner during three days after receipt of the notification about the prize.

In case of absence of a response in the designated period or failure to provide accurate data required for the delivery of prizes, the Organizer has the right to refuse a Participant in the prize of the Competition. The cash equivalent of the prize is not provided.

The winners of the Competition must give the Organizer copies of all necessary documents for accounting and tax reporting. The list of documents which the Winner should give the Organizer, may include:

- a copy of the Winner's passport;
- a copy of the Winner's certificate on statement on the tax account;
- a copy of the Winner's pension certificate;
- information about the Bank account of the Winner;
- Other documents that the Organizer will require of the Participant for the purposes of reporting on the conducted Competition.

Along with copies the Organizer of the Competition has the right to request the originals of the documents.

In accordance with subparagraph 4 of paragraph 1 of article 228 of the Tax Code of the Russian Federation, the Winner of the Competition who became the owner of the Prize, bear all costs payment of all applicable taxes, stipulated by the legislation of the Russian Federation.

1.9 The procedure and method of informing Participants

Informing of Participants is carried out by placing the information on the Internet on the Website of the Organizer at the following address: http://www.russianaicup.ru and also via the System of the Organizer of the Competition, during the period of Competition.

Глава 2

About CodeWars 2017 world

2.1 General concept of the game and the rules of the tournament

This competition gives you the opportunity to test your programming skills by creating an artificial intelligence (strategy), managing a large number of military units (vehicles) in a special game world (more detailed information about the features of the world CodeWars 2017 can be found in the next points of this chapter). A specific feature of this year's task is that the set of activities available to your strategy is similar to the control capabilities in conventional computer games of RTS genre. Also there is a limit on the number of actions per unit of playing time. You will be opposed by another player's strategy in each game. In order to win you need to score more points than your opponent. Points are awarded for various game actions. Of course, significant number of points is given for the complete opponent destruction which almost completely levels out other achievements in the game. There is a theoretical possibility to destroy the opponent, but in the same time to lose by points, but this is almost impossible in practice. The number of points received during the game becomes more important, if none of the participants managed to achieve a complete victory for the time allotted for the game.

The battle occurs on different types of terrain and under different weather conditions that affect some parameters of the vehicles. Neutral facilities can be present on the map in some game modes, capturing which the strategy gets the opportunity to produce new vehicles or to gain other gaming advantages. Games of the last stage of the tournament are held in conditions of partial visibility.

The tournament is held in several stages preceded by a qualification in the Sandbox. Sandbox is a competition that takes place throughout the championship. The player has a certain rating value — an indicator of how successful his strategy is involved in games within each stage.

The initial value of the rating in the Sandbox is 1200. At the end of the game this value can both be increased and decreased. At the same time victory over a weak (with a low rating) opponent gives a small increase, also the defeat from a strong opponent slightly decreases your rating. Over time the rating in the Sandbox becomes more and more inert, which makes it possible to decrease the impact of random long series of victories or defeats on the participant's place, but at the same time makes it difficult to change his position with a significant improvement in strategy. To cancel such effect the participant can reset the variability of the rating to the initial state when sending a new strategy, including the corresponding option. If the new strategy is adopted, the rating system of the participant will fall dramatically after the next game in the Sandbox, however, further participation in games will quickly recover and even become higher if your strategy has really become more effective. It is not recommended to use this option with minor, incremental improvements to your strategy, as well as in cases where a new strategy insufficiently tested and the effect of changes in it is not known reliably.

The initial value of the rating at each main stage of the tournament is 0. For each game the participant receives a certain number of rating points depending on the occupied place (a system similar to that used in the championship "Formula-1"). If two or more participants share some place, then the total number of rating points for this place and for the following `number_of_such_members` − 1 of places is shared equally among these

participants. For example, if two participants share the first place, then each of them will receive half of the rating points number for the first and second places. When sharing rounding always takes place in a smaller direction. More detailed information about the stages of the tournament will be provided in the announcements on the project website.

First all participants can participate only in the games that take place in the Sandbox. Players can send their strategies to the Sandbox, and the last one taken from them is taken by the system for participation in qualifying games. Each player participates in approximately one qualifying game for an hour. The jury reserves the right to change this interval based on the throughput of the testing system, but for the majority of participants it remains constant. There are a number of criteria by which the interval of participation in qualifying games can be increased for a specific player. For every N-th full week that has elapsed since the player sent the last strategy, the interval of participation for this player is increased by N basic test intervals. Only the strategies adopted by the system are taken into account. An additional penalty which is equal to 20% from the basic testing interval is charged in the Sandbox for each strategy "crash" in 10 last games. More details about the causes of the strategy "crashing" can be found in the following sections. The player's participation interval in the Sandbox can not become bigger than a day.

Games in the Sandbox are held according to a set of rules corresponding to the rules of an accidental past stage of the tournament or to the rules of the next (current) stage. At the same time, the closer the rating value of the two players rating within the Sandbox, the more likely that they will be in the one game. The Sandbox starts before the start of the first stage of the tournament and ends after some time after the final stage (see the schedule of stages to clarify the details). In addition, the Sandbox is frozen during the stages of the tournament. Following the results of the games in the Sandbox there is a selection for participation in Round 1, which will involve 1080 of participants with the highest rating at the beginning of this stage of the tournament (if the rating is equal, priority is given to the player who previously sent the latest version of his strategy), as well as an additional selection to the next stages of the tournament, including the Finals.

Tournament stages:

- In **Round 1**, you will learn the rules of the game and master the control of a large number of units. At the beginning of the game you are given 500 units of vehicles. Your opponent is given the same number of vehicles. The task is — to destroy! It's simple. Round 1, as all further stages, consists of two parts, between which there will be a short break (with the renewal of the Sandbox work), which allows to improve its strategy. The last strategy sent by the player before the beginning of this part is selected for the games in each part. Games are conducted in waves. In each wave, each player participates exactly in one game. The number of waves in each part is determined by the capabilities of the testing system, but it is guaranteed that it will not be less than ten. 300 highest rated participants will be held in Round 2. Also in Round 2 there will be an additional selection of 60 participants with the highest rating in the Sandbox (at the moment of Round 2 beginning) among those who did not passed according to the results of Round 1.
- In **Round 2** you have to improve your management skills for a big number of units. Also facilities are appearing on the map, which your strategy can capture, thus gaining a game advantage over your opponent. The task is further complicated that after summarizing the Round 1, the part of the weak strategies will be eliminated and you will have to confront stronger opponents. According to the results of Round 2 of the best 50 strategies will reach the Finals. Also in the Finals there will be an additional selection of 10 participants with the highest rating in the Sandbox (at the beginning of the Finals) from those who did not go through the main tournament.
- **Finals** is the most important stage. After the selection, held following the results of the first two stages, the strongest participants will be remained. Also a fog of war is introduced in the Finals limiting the visibility of the opponent's vehicles. Strategies are always available for complete terrain maps and weather, as well as information about all facilities on the map. The range of the units observation is quite large. Thus, changing the rules should not greatly affect the local tactical control. However, to obtain information about the remote parts of the map strategy will need to send some of the vehicles into the exploration. The system of holding the Finals has its own peculiarities. The stage is still divided into two parts, but they will no longer consist of waves. In each part of the stage, games will be played between all pairs of Finals participants. If the time and capabilities of the testing system permit, the operation will be repeated.

All finalists are ranked according to the non-increase in the rating after the end of the Finals. If the ratings are equal, a higher place is taken by that finalist, whose strategy, which was part of the Final, was sent out earlier. Prizes for the Final are distributed based on the occupied place after this ordering.

After the completion of the Sandbox, all its participants, except for the Finals winners, are ranked according to the non-increase in the rating. If the ratings are equal a higher place is taken by the participant who sent the latest version of his strategy earlier. Prizes for the Sandbox are distributed on the basis of occupied place after this ordering.

2.2 Description of the game world

The game world is two-dimensional, and all units in it have the form of a circle. The axis of abscissas in this world is directed from left to right, the axis of ordinates is — from top to bottom, the angle 0.0 coincides with the direction of the abscissa axis, and the positive rotation angle means clockwise rotation. Gaming area is bounded by a square which top-left corner has coordinates (0.0, 0.0), and the side length is 1024.0. None of the units can be completely or partially located outside the playing area.

At the beginning of each game, the vehicles of the first player is in the upper left corner of the game area, and the vehicles of the second player is in the lower right corner. In this case, the coordinates for the strategy of the second player are transferred in a transformed form. Thus, the strategy always "thinks" it starts the game in the upper left corner of the map, and the opponent is in the lower right corner. The amount of each player vehicles at the beginning of the game is a multiple of 100. The vehicles are divided into groups of 100 units in each. All equipment in one such group has the same type. The formation of the group is a square 10×10. If at the beginning of the game to turn the game area to 180° relative to its center, then the position of each unit of the second player coincides with the position of the first player's unit before the turn. In this case, units with a matching position will have the same type.

Time in the game is discrete and is measured in "ticks". At the beginning of each tick, the game simulator transmits the world state data to the participants' strategies, receives control alarms from them and updates the state of the world in accordance with these alarms and the limitations of the world. Then makes calculation of the change of the world and objects in it for this tick, and the process is repeated again with the updated data. The maximum duration of any game is equal to 20000 ticks, but the game can be terminated prematurely if all units of at least one strategy have been destroyed or all strategies "have crashed". It is extremely unlikely, but still it is possible that all units of both players will be destroyed in the same tick. Then additional points will be given to all participants of the game.

The "crashed" strategy can no longer control the vehicles. The strategy is considered as "crashed" in the following cases:

- The process in which the strategy is started has unexpectedly terminated, or an error has occurred in the protocol of interaction between the strategy And game server.
- The strategy exceeded one (any) of the time constraints assigned to it. Strategy for one tick is allocated not more than 20 seconds of real time. But in sum for the whole game the strategy process is given

$$20 \times < duration\_of\_game\_in\_ticks > + 20000 \tag{2.1}$$

milliseconds of real time and

$$10 \times < duration\_of\_game\_in\_ticks > + 20000 \tag{2.2}$$

milliseconds of processor time.[1] The formula takes into account the maximum duration of the game. The time limit remains the same, even if the actual duration of the game is different from this value. All time

[1] Despite the fact that the restriction of real time is much higher than the limitation of processor time, it is forbidden artificially "slow down" testing of strategy by commands like "`sleep`" (as well as try to slow down/destabilize the testing system in other ways). In case of revealing such irregularities, the jury reserves the right to apply measures to this user at its discretion, up to disqualification from the competition and Account Lock-out.

limits apply not only to the participant code, but on the interaction of the client-shell strategy with the game simulator.

- The strategy exceeded the memory limit. At any point in time the strategy process should not consume more than 256 MB of RAM.

2.3 Vehicles types

In the world of CodeWars 2017 all units are vehicles. There are 5 types of vehicles:

- **tank**: ground unit, effective against other ground units;
- **IFV** — infantry fighting vehicle: ground unit, effective against air units;
- **attack helicopter**: air unit, effective against ground units;
- **fighter**: air unit, effective against other air units;
- **ARRV** — armored recovery and repair vehicle: repairs damaged evehicles.

The main characteristics of the vehicles are the current and maximum durability. With a drop in durability to zero, a unit is considered destroyed and removed from the game world. The initial and maximum durability of each unit is 100. All units are circles of radius of 2.0. The interval between two successive attacks of vehicle of any type other than ARRV is 60 ticks. ARRV can not attack.

Comparative characteristics of vehicles types are given in the following table:

Characteristic \Vehicle type	Tank	IFV	Helicopter	Fighter	ARRV
Speed	0.3	0.4	0.9	1.2	0.4
Vision range	80	80	100	120	60
Attack range against ground targets	20	18	20	—	—
Attack range against aerial targets	18	20	18	20	—
Attack damage against ground targets	100	90	100	—	—
Attack damage against aerial targets	60	80	80	100	—
Defence against attacks of ground targets	80	60	40	70	50
Defence against attacks of air targets	60	80	40	70	20
Production cost, ticks	60	60	75	90	60

In the absence of a fog of war, the range of vision has almost no effect on the game and is taken into account only when certain actions. With the fog of war on, the participant's strategy will only receive data on those enemy units that are within range[2] of vision of at least one unit of this participant.

All attacking units automatically inflict damage if within the range of its attack there is at least one opponent unit, and also passed enough time since the last attack. An random one is chosen for the attack if there are multiple targets. In this case, the higher the potential damage of one attack ($< damage > - < defence >$) for a specific target, the more likely this goal will be chosen. Durability value of the target is not taken into account. If the potential damage to the target is not a positive number, it is considered that the unit can not attack the target. Damage is applied instantly. It is also considered that all units making an attack in one tick do it simultaneously. Thus, a duel of two identical units of different players will end with the destruction of both of these units.

ARRV automatically repair on each tick to 0.1 one of the friendly units the durability of which is less than the maximum, and the distance to ARRV does not exceed 10. If there are several targets, the random one is selected for repair. At the same time, the lower the durability of this target, it is more likely to be chosen. The value of repair speed is less than one, therefore for several ticks it may seem that the durability of the vehicles is not

[2]Here and below the distance between units means the distance between its centers, unless explicitly indicated another.

restored, but it is not true. Total recovery of durability for past ticks is accumulated in a special pool. The vehicle is considered destroyed if the integer part of its durability drops to zero, regardless of the value in the pool.

2.4 Types of terrain and weather

The terrain and weather maps are divided into cells with the size 32.0×32.0. Each of these cells can have one of three types of terrain and one of three types of weather. The type of terrain affects various parameters of ground vehicles; type of weather, respectively, — aerial. Terrain and weather maps turn into themselves if they are rotated around the center of the game area at 180°. Both maps will not be changed in the game process and strategies are always available, regardless of the war fog presence.

Characteristic \Type of terrain	Plain	Swamp	Forest
Speed factor	1.0	0.6	0.8
Vision range factor	1.0	1.0	0.8
Stealth factor	1.0	1.0	0.6

Characteristic \Weather type	Clear	Solid clouds	Heavy rain
Speed factor	1.0	0.8	0.6
Vision range factor	1.0	0.8	0.6
Stealth factor	1.0	0.8	0.6

If everything is obvious with speed and vision range factors, then the vehicles stealth factor affects the vision range of any enemy unit when checking the visibility of these vehicles. Thus, the unit sees the target if and only if the distance to the target is less than or equal to

$$<vision\_range\_of\_unit> \times <vision\_range\_factor\_of\_unit> \times <stealth\_factor\_of\_target> \tag{2.3}$$

If fog of war is disabled, the stealth factor do not affect the game in any way; also the vision range factor only used to restrict some strategy actions.

2.5 Facilities

Facilities appear in Round 2 of the tournament and represent square areas on the map. The length of the side of each such square is 64.0, and its upper left corner coincides with the upper left corner of one of the terrain/weather map cells. The location of the facilities is symmetrical for both players. Totally there can be up to 8 pairs of facilities on the map (which are reflections of each other). The strategy receives information about all facilities regardless of the presence of fog of war.

At the beginning of the game all facilities are neutral. Strategies can capture them by moving ground vehicles into the facilities area. Each unit in the facilities area generates 0.005 capture points of this facility each game tick. With the accumulation of 100.0 capture units the process is terminated, and the facility goes under control of the strategy. If your opponent completely or partially captured the facility, it is necessary to reset its capture level at first. Zeroing the capture of an opponent occurs in the same way and with the same speed as the actual capture. If the opponent already controls the facility, then he will retain control over it until its capture level falls to zero.

Types of facilities:

- control center: increases the limit of the number of strategy actions by 3 for 60 of ticks, and reduces the interval between tactical nuclear strikes by 60 ticks;
- factory: automatically produces vehicles, the type of vehicle is determined by the strategy, the production speed depends on the type of vehicle.

A new strategy vehicles appear at the plant in rows from left to right, from top to bottom. The opponent's vehicles — are also in rows, but from right to left, from the bottom to the top. The distance between the centers of two neighboring units produced at the factory is 6.0. If the next position of the unit is occupied, then it will be skipped and so on until a free position is found. If all the positions for the production of units are occupied, the factory will suspend production of new vehicles.

2.6 Control

At the beginning of each tick, the game simulator sends out strategies for information about the current state of the visible part of the world. In response, the strategy sends a set of instructions (encapsulated in a class object `Move`) to control the vehicles or simply skip the move without setting the field `move.action` or initializing it with the value `NONE`. Initially, the number of possible actions of the strategy is limited to 12 moves for 60 ticks. This value can be increased by capturing one or more control centers. Strategy action will be ignored by the game simulator, if for the last $60 - 1$ ticks it has already committed the maximum amount of actions available to it.

Strategy instructions are processed in the following order:

- First, there is a change in the world in accordance with the wishes of the strategy: all actions are carried out to select units, assignments of groups, the production of vehicles at the factory is adjusted, etc. Also, orders of units are updated.
- Then all the units are randomly ordered, and they move according to the received or already existing orders, as well as limitation of the maximum speed of these units, taking into account the type of terrain or weather. In the world of CodeWars 2017 there is no inertia, and the movement occurs instantaneously or does not occur at all. Moving technique is carried out sequentially, according to selected order. At the same time, partial movement of vehicles is not applied. If the position of technology can not be changed to full the value calculated by the game simulator moving[3], then its movement is postponed. After the end of the iteration the game simulator again iterates over all units and tries to move those whose position in this tick has not yet been changed. It is repeated until all units are moved. If none unit has been moved during an iteration, then the operation is also interrupted.
- Then, all units that are not on recharge, simultaneously perform attacking actions, and ARRV goes to repairs.
- Lastly, the level of facilities capturing is changed and a new vehicles are produced.

As it was already mentioned, the control in CodeWars is similar with control in conventional computer games of the RTS genre, although it does not pretend to be complete conformity. The following actions are available to the strategy:

- `CLEAR_AND_SELECT`. Standard selection of friendly units by frame or selection of a previously created group of units. In the first case you should additionally specify the frame borders (`move.left`, `move.top`, `move.right` and `move.bottom`), also type of vehicles can be selected additionally (`move.vehicleType`), in the second — group number (from 1 to 100). If the group number is given, then all other parameters of the object `move` will be ignored.
- `ADD_TO_SELECTION`. Similar to the action `CLEAR_AND_SELECT` with the exception that the existing selection of units will not be reset.
- `DESELECT`. Removes selection from units corresponding to the specified parameters. The setting of the action is no different from given above two types of actions.

[3] The vehicles after moving is partially or completely outside the map or intersects with some other vehicles, such that the clash of these two units is prohibited by the rules of the game.

- `ASSIGN`. Sets the group membership for all selected units. You should additionally set the number of group. In this case, the units added to the group earlier remain in it. A unit can be in more than one group at the same time.
- `DISMISS`. Removes all selected units from the group. You should additionally set the group number.
- `DISBAND`. Removes all units from the specified group.
- `MOVE`. Orders selected units to move in the specified direction. Parameters `move.x` and `move.y` set the vector. Thus, units move, keeping the formation. In addition, you can limit the maximum speed of movement so that slow units do not lag behind faster ones.
- `ROTATE`. Instruct the selected units to rotate relative to the specified point. Parameters `move.x` and `move.y` set this point, and `move.angle` — angle of rotation. Units move around the circle. In this way, the distance from the unit to the specified point does not change during the movement. In addition, you can limit the maximum linear or the maximum angular velocity of movement, so that the formation rotates synchronously, and slow units do not lag from faster units.
- `SCALE`. Scales the formation of the selected units relative to the specified point with the specified factor. Parameters `move.x` and `move.y` set this point, and `move.factor` — factor from 0.1 to 10.0. With the factor value more 1.0 there is an expansion of the formation, if values are lower than 1.0 — compression. To determine the position of the unit at end of the movement, you need to build a vector from the specified point to the position of the unit before the movement, multiply both coordinates of this vector on the factor and add the resulting vector to the point specified in the order. All units will move linearly each to its target. In addition, you can limit the maximum speed of movement.
- `SETUP_VEHICLE_PRODUCTION`. Configures the production of equipment in the captured factory. You should specify the type of vehicle and factory identifier. At the same time, the progress of production will be reset, even if this type of vehicle is equal to the type of equipment, produced at the factory at the moment.
- `TACTICAL_NUCLEAR_STRIKE`. Requests a tactical nuclear strike at the specified coordinates. It is required additionally specify the identifier of friendly vehicle, which will mark the target. You can not request a strike more often than once each 1200 ticks. This interval is slightly reduced for each captured control center. Striking occurs not instantly, but in 30 ticks after the request. For a successful strike, the target should be in the vision area of the specified unit, both at the time of the request and all subsequent ticks before the strike. If either the unit that marks the target dies or the target leaves the vision area of the unit for at least one tick, the strike is canceled. The strike hits all targets at a distance not exceeding 50.0, both enemy and allied. Also, the damage in the center of the explosion is 99 units and evenly drops to zero on the edge. A unit can mark a target if and only if the distance to this target is less than or equal to

$$<vision\_range\_of\_unit> \times <vision\_range\_factor\_of\_unit> \tag{2.4}$$

2.7 Collision of units

The collision of ground units among themselves, as well as with the maps borders is not allowed by the game simulator. The collision of air units with each other, and also with the map borders is not allowed by the game simulator. The exception is air units belonging to different players.

2.8 Scoring

Score points are awarded for the following actions:

- 1 score point is given for the destruction of the enemy unit.
- The capturing of the facilities brings 100 score points to the strategy.

- When all opponents' units are destroyed, the strategy receives 1000 score points. The game is completed.
- Every 1000 ticks, starting at 10000 and not including the last tick, the strategy that have less vehicles gains 1 score point for each 10 units of difference.

Глава 3

Strategy creation

3.1 Technical part

First, to create a strategy, you need to choose one of a number of supported programming languages[4]: Java (Oracle JDK 8), C # (Roslyn 1.3+), C ++ 14 (GNU MinGW C ++ 6.2+), Python 2 (Python 2.7+), Python 3 (Python 3.5+), Pascal (Free Pascal 3.0+), Ruby (JRuby 9.1+, Oracle JDK 8). Perhaps this set will be extended. On the project site you can download a custom package for each of the languages. Only one file intended for the content of your strategy is allowed to be modified in the package, that is, for example, MyStrategy.java (for Java) or MyStrategy.py (for Python)[5]. All other package files will be replaced with standard versions when the strategy is built. However, you can add your code files to the strategy. These files should be in the same catalogue as the the main strategy file. When submitting a solution, all of them should be placed in one ZIP-archive (the files should be in the root of the archive). If you do not add new files to the package, just send the file of the strategy (using the file selection dialog) or insert its code into the text field.

After you send your strategy, it falls into the testing queue. Firstly the system will try to compile the package with your files, and then, if the operation was successful, create several short (for 200 ticks) games of different formats[6]: 1 vs 1, 1 vs 1 with addition of facilities and 1 vs 1 with adding facilities and fog of war. To control the vehicles of each of the participants in these games, a separate client process will be launched with your strategy, and in order for the strategy to be considered as accepted (correct), none of the strategy instances should "crash". The names will be given to the players in these test games in the format `<player_name>`, `<player_name> (2)`, `<player_name> (3)` and etc.

After the successful completion of the described process, your package receives the status "Accepted". The first successful package simultaneously means your registration in the Sandbox. You get a starting rating (1200) and your strategy starts to participate in the periodic qualifying games (see the description of the Sandbox for more details). Also you can access the function to create your own games, in which as an opponent you can choose any strategy of any player (including your own), Created before your last successful sending. The games you create do not affect the rating.

---

[4] For all programing languages 32-bit versions of compilers/interpreters are used.

[5] The exception is C++, for which you can modify two files: MyStrategy.cpp and MyStrategy.h. Moreover, the presence of MyStrategy.cpp file is mandatory in the archive (otherwise the strategy will not be compiled), and the presence of MyStrategy.h file is optional. If not, the standard file from the package will be used.

[6] The main parameters of the format of the game are the number of players participating in it, and the number of units that are under the control of each player. Briefly, the format is written in the form `<number_players>` × `<number_units>`, for example 4×3 means The format of the game, in which 4 is involved in a player who controls three units each. In the championships, all games are always held in the format of duels, an alternative form of record can be used, for example, a 1 player vs 1 player ($2 \times$ `N` in the canonical form), 1 unit vs 1 unit (2×1) or 3 units vs 3 units (2×3). The words "player" and "unit" in the format record can be replaced with corresponding icons or even get rid of, if the context is clear about what is at stake. Explanation can be added to the format of the game if the short form for the different stages of the championship matches.

There are restrictions on the number of packages and user games in the system, namely:

- You can not send the strategy more than three times within twenty minutes. The total size (without compression) of the strategies sent for twenty minutes can not be more than 3 MB. The limit on the size of one submission is 2 MB.
- Within twenty minutes you can not create more than three user games. After the completion of each main stage of the competition, this the number is automatically increased by one.

To simplify the debugging of small changes in the strategy in the system, there is an opportunity to make a test submission (checkbox "Testing submit" on the form of submitting the strategy). The test package is not displayed to other users, does not participate in the qualifying games in the Sandbox and games in the stages of the tournament, it is also impossible to create games with its participation. However, after the acceptance of this submission, the system automatically adds a test game with two participants (1 format for 1): directly by the test submission and the strategy from the section "Quick start". The test game is visible only to the participant who made this test submission. The base duration of such a test game is 2000 ticks. The frequency of the testing submissions is affected by the same restriction as the frequency of the regular submissions. Test games don't influence frequency of games creation by the user.

Players have the opportunity to view past games in a special visualizer. To do this, click the "View" button in the list of games or click "View game" on the game page.

If you watch a game involving your strategy and notice some strangeness in its behavior, or your strategy does not do what you want from it then you can use the special utility Repeater to play the local repeat of this game. Local Repeat of the game is the ability to run a strategy on your computer so that it sees the game world around itself as it was when testing on the server. This will help you to perform debugging, log and monitor the reaction of your strategy at any time. To do this, download Repeater from CodeWars 2017 (section "Documentation" → "Utility Repeater") and unzip. To run Repeater, you need the Java 8+ Runtime Environment software installled. Please note that any interaction between your strategy with the game world during local repeat is completely ignored. This means that at every moment of time the world around us for strategy exactly coincides with the world, as it was in the game when testing on the server and does not depend on what actions your strategy undertakes. The Repeater utility has only the data that was sent to your strategy, but not the complete recording of the game. Therefore the game is not rendered. More information about the Repeater utility can be found in the corresponding section on the website.

In addition to all of the above, players have the opportunity to run simple test games locally on their computers. For this you need to download the archive with the Local runner utility from the section "Documentation" → "Local runner". Usage of this utility will allow you to test your strategy under conditions similar to the test game on the site, but without any limitations on the the number of games being created. Renderer for local games is noticeably different from the renderer on the site. All game objects in it are displayed schematically (without the use of colorful models). Creating a local test game is easy: run Local runner with the corresponding startup script (*.bat for Windows or *.sh for *n*x systems), then run your strategy from the development environment (or using any another way that is convenient for you) and watch the game. During local games, you can debug your strategy, set breakpoints. However, it should be remembered that Local runner expects response from the strategy not more than 30 minutes. After this time it will mark strategy as "crashed" and will continue to work without it.

3.2 Vehicles control

For your player at the beginning of the game, an object of class `MyStrategy` is created, in the fields of which the strategy can store information about the progress games. The vehicles are controlled using the `move` method of the strategy, which is called once per tick. The method is called with the following parameters:

- your player `me`;
- current state of the world `world`;
- set of game constants `game`;

- object `move`, setting the properties of which, the strategy controls the vehicles.

Implementation of the client-shell strategy in different languages may differ, but in general, **not** is guaranteed that for different calls to the method `move` as parameters to it will be passed references to the same objects. Thus, it is wrong, for example, to save references to objects `world` or `vehicle` and retrieve updated information about these objects in the following ticks, reading its fields.

3.3 Examples of implementation

Next for all programming languages are the simplest examples of strategies that first select all of your vehicles, and then are sent to approach the opponent. Full documentation on classes and methods for the Java language can be found in the following sections.

3.3.1 Example for Java

```
import model.*;

public final class MyStrategy implements Strategy {
    @Override
    public void move(Player me, World world, Game game, Move move) {
        if (world.getTickIndex() == 0) {
            move.setAction(ActionType.CLEAR_AND_SELECT);
            move.setRight(world.getWidth());
            move.setBottom(world.getHeight());
            return;
        }

        if (world.getTickIndex() == 1) {
            move.setAction(ActionType.MOVE);
            move.setX(world.getWidth() / 2.0D);
            move.setY(world.getHeight() / 2.0D);
        }
    }
}
```

3.3.2 Example for C#

```
using Com.CodeGame.CodeWars2017.DevKit.CSharpCgdk.Model;

namespace Com.CodeGame.CodeWars2017.DevKit.CSharpCgdk {
    public sealed class MyStrategy : IStrategy {
        public void Move(Player me, World world, Game game, Move move) {
            if (world.TickIndex == 0) {
                move.Action = ActionType.ClearAndSelect;
                move.Right = world.Width;
                move.Bottom = world.Height;
                return;
            }

            if (world.TickIndex == 1) {
                move.Action = ActionType.Move;
                move.X = world.Width / 2.0D;
                move.Y = world.Height / 2.0D;
            }
        }
    }
}
```

3.3.3 Example for C++

```
#include "MyStrategy.h"

#define PI 3.14159265358979323846
#define _USE_MATH_DEFINES

#include <cmath>
#include <cstdlib>

using namespace model;
using namespace std;

void MyStrategy::move(const Player& me, const World& world, const Game& game, Move& move) {
    if (world.getTickIndex() == 0) {
        move.setAction(ActionType::CLEAR_AND_SELECT);
        move.setRight(world.getWidth());
        move.setBottom(world.getHeight());
        return;
    }

    if (world.getTickIndex() == 1) {
        move.setAction(ActionType::MOVE);
        move.setX(world.getWidth() / 2.0);
        move.setY(world.getHeight() / 2.0);
    }
}

MyStrategy::MyStrategy() { }
```

3.3.4 Example for Python 2

```
from model.ActionType import ActionType
from model.Game import Game
from model.Move import Move
from model.Player import Player
from model.World import World


class MyStrategy:
    def move(self, me, world, game, move):
        """
        @type me: Player
        @type world: World
        @type game: Game
        @type move: Move
        """
        if world.tick_index == 0:
            move.action = ActionType.CLEAR_AND_SELECT
            move.right = world.width
            move.bottom = world.height

        if world.tick_index == 1:
            move.action = ActionType.MOVE
            move.x = world.width / 2.0
            move.y = world.height / 2.0
```

3.3.5 Example for Python 3

```
from model.ActionType import ActionType
from model.Game import Game
from model.Move import Move
from model.Player import Player
from model.World import World


class MyStrategy:
    def move(self, me: Player, world: World, game: Game, move: Move):
        if world.tick_index == 0:
            move.action = ActionType.CLEAR_AND_SELECT
            move.right = world.width
            move.bottom = world.height

        if world.tick_index == 1:
            move.action = ActionType.MOVE
            move.x = world.width / 2.0
            move.y = world.height / 2.0
```

3.3.6 Example for Pascal

```
unit MyStrategy;

interface

uses
  StrategyControl, TypeControl, ActionTypeControl, CircularUnitControl, FacilityControl,
  FacilityTypeControl, GameControl, MoveControl, PlayerContextControl, PlayerControl,
  TerrainTypeControl, UnitControl, VehicleControl, VehicleTypeControl, VehicleUpdateControl,
  WeatherTypeControl, WorldControl;

type
  TMyStrategy = class (TStrategy)
  public
    procedure Move(me: TPlayer; world: TWorld; game: TGame; move: TMove); override;

  end;

implementation

uses
  Math;

procedure TMyStrategy.Move(me: TPlayer; world: TWorld; game: TGame; move: TMove);
begin
  if world.TickIndex = 0 then begin
    move.Action := ACTION_CLEAR_AND_SELECT;
    move.Right := world.Width;
    move.Bottom := world.Height;
    exit;
  end;

  if world.TickIndex = 1 then begin
    move.Action := ACTION_MOVE;
    move.X := world.Width / 2.0;
    move.Y := world.Height / 2.0;
  end;
end;

end.
```

3.3.7 Example for Ruby

```
require './model/game'
require './model/move'
require './model/player'
require './model/world'

class MyStrategy
  # @param [Player] me
  # @param [World] world
  # @param [Game] game
  # @param [Move] move
  def move(me, world, game, move)
    if world.tick_index == 0
      move.action = ActionType::CLEAR_AND_SELECT
      move.right = world.width
      move.bottom = world.height
    end

    if world.tick_index == 1
      move.action = ActionType::MOVE
      move.x = world.width / 2.0
      move.y = world.height / 2.0
    end
  end
end
```

Глава 4

Package model

Package Contents *Page*

Classes

4.1 Classes

4.1.1 CLASS **ActionType**

Available player actions.

A player can not perform any new action, if during last `game.actionDetectionInterval - 1` game ticks he already performed maximum possible actions. At the start of the game this limit is `game.baseActionCount` for each player. The limit increases for each captured control center.

The most actions require additional parameters to perform. The strategy can set up an action with parameters updating fields of a `move` object. If the specified parameters are incorrect or not sufficient, the action will be ignored by game simulator. However, each action that is specified and is not `NONE` counts in the list of last actions.

DECLARATION

public final class ActionType

extends Enum

FIELDS

- public static final ActionType NONE
  - Do nothing.
- public static final ActionType CLEAR_AND_SELECT
  - Select units that match specified parameters. Deselect units that do not match these parameters. The units of other players are automatically excluded.
- public static final ActionType ADD_TO_SELECTION
  - Select units that match specified parameters. Already selected units stay selected. The units of other players are automatically excluded.
- public static final ActionType DESELECT
  - Deselect units that match specified parameters.
- public static final ActionType ASSIGN
  - Assign selected units to the specified group.
- public static final ActionType DISMISS
  - Dismiss selected units from the specified group.
- public static final ActionType DISBAND
  - Disband the group.
- public static final ActionType MOVE

- Order selected units to move in the specified direction direction.

- public static final ActionType ROTATE
  - Order selected units to rotate around the specified point.
- public static final ActionType SCALE
  - Scale the formation of selected units relative to the specified point.
- public static final ActionType SETUP_VEHICLE_PRODUCTION
  - Set up production of the specified vehicle type on factory.
- public static final ActionType TACTICAL_NUCLEAR_STRIKE
  - Request tactical nuclear strike.

4.1.2 CLASS **CircularUnit**

This base class describes any circular unit in the game world.

DECLARATION

public abstract class CircularUnit **extends** Unit

METHODS

- *getRadius*
  `public double` **`getRadius`**`( )`
  - **Returns** - the radius of this unit.

4.1.3 CLASS **Facility**

This class describes a facility. A facility is a rectangular area on the map.

DECLARATION

public class Facility **extends** Object

- *getCapturePoints*
  `public double` **getCapturePoints**`( )`
  - **Returns** - the capture level of the facility in the range of `-game.maxFacilityCapturePoints` to `game.maxFacilityCapturePoints`. The positive level means, that you are capturing this facility. The negative level means, that some other player is capturing this facility.

- *getId*
  `public long` **getId**`( )`
  - **Returns** - the unique facility ID

- *getLeft*
  `public double` **getLeft**`( )`
  - **Returns** - the leftmost X of the facility.

- *getOwnerPlayerId*
  `public long` **getOwnerPlayerId**`( )`
  - **Returns** - the owner player ID or `-1`.

- *getProductionProgress*
  `public int` **getProductionProgress**`( )`
  - **Returns** - the nonnegative number that is vehicle production progress. For control center the value is always `0`.

- *getTop*
  `public double` **getTop**`( )`
  - **Returns** - the topmost Y of the facility.

- *getType*
  `public FacilityType` **getType**`( )`
  - **Returns** - the facility type.

- *getVehicleType*
  `public VehicleType` **getVehicleType**`( )`
  - **Returns** - the type of the vehicle that this factory produces or `null`. For control center the value is always `null`.

4.1.4 CLASS **FacilityType**

Facility type.

DECLARATION

public final class FacilityType **extends** Enum

FIELDS

- public static final FacilityType CONTROL_CENTER
  - Increases the limit of player actions for `game.additionalActionCountPerControlCenter` per `game.actionDetectionInterval` game ticks. Also slightly reduced cooldown of tactical nuclear strikes.
- public static final FacilityType VEHICLE_FACTORY
  - The factory can produce vehicles of any type.

4.1.5 CLASS **Game**

An instance of this class contains all game constants.

DECLARATION

public class Game **extends** Object

METHODS

- *getActionDetectionInterval*
  `public int` **getActionDetectionInterval**`( )`
  - **Returns** - the interval that is used to limit player actions.
- *getAdditionalActionCountPerControlCenter*
  `public int` **getAdditionalActionCountPerControlCenter**`( )`
  - **Returns** - the additional action count per each captured control center.
- *getArrvAerialDefence*
  `public int` **getArrvAerialDefence**`( )`
  - **Returns** - the defence of an ARRV against aerial attacks.
- *getArrvDurability*
  `public int` **getArrvDurability**`( )`
  - **Returns** - the maximal durability of an ARRV.
- *getArrvGroundDefence*
  `public int` **getArrvGroundDefence**`( )`
  - **Returns** - the defence of an ARRV against ground attacks.

- *getArrvProductionCost*
  `public int` **getArrvProductionCost**`( )`
  - **Returns** - the amount of ticks to produce an ARRV on a factory.

- *getArrvRepairRange*
  `public double` **getArrvRepairRange**`( )`
  - **Returns** - the repair range of an ARRV.

- *getArrvRepairSpeed*
  `public double` **getArrvRepairSpeed**`( )`
  - **Returns** - the repair amount of an ARRV per tick.

- *getArrvSpeed*
  `public double` **getArrvSpeed**`( )`
  - **Returns** - the maximal speed of an ARRV.

- *getArrvVisionRange*
  `public double` **getArrvVisionRange**`( )`
  - **Returns** - the base vision range of an ARRV.

- *getBaseActionCount*
  `public int` **getBaseActionCount**`( )`
  - **Returns** - the base action count that player can perform in `actionDetectionInterval` ticks.

- *getBaseTacticalNuclearStrikeCooldown*
  `public int` **getBaseTacticalNuclearStrikeCooldown**`( )`
  - **Returns** - the base cooldown of tactical nuclear strike request.

- *getClearWeatherSpeedFactor*
  `public double` **getClearWeatherSpeedFactor**`( )`
  - **Returns** - the speed factor of aerial vehicles in clear weather.

- *getClearWeatherStealthFactor*
  `public double` **getClearWeatherStealthFactor**`( )`
  - **Returns** - the vision range factor of any vehicle that is trying to detect an aerial vehicle in clear weather.

- *getClearWeatherVisionFactor*
  `public double` **getClearWeatherVisionFactor**`( )`
  - **Returns** - the vision range factor of an aerial vehicle in clear weather.

- *getCloudWeatherSpeedFactor*
  `public double` **getCloudWeatherSpeedFactor**`( )`
  - **Returns** - the speed factor of aerial vehicles in cloud weather.

- *getCloudWeatherStealthFactor*
  `public double` **getCloudWeatherStealthFactor**`( )`
  - **Returns** - the vision range factor of any vehicle that is trying to detect an aerial vehicle in cloud weather.

- *getCloudWeatherVisionFactor*
  `public double` **getCloudWeatherVisionFactor**`( )`

- **Returns** - the vision range factor of an aerial vehicle in cloud weather.

- *getFacilityCapturePointsPerVehiclePerTick*
`public double` **getFacilityCapturePointsPerVehiclePerTick**`( )`

  - **Returns** - the speed of facility capturing per each vehicle inside the facility area.

- *getFacilityCaptureScore*
`public int` **getFacilityCaptureScore**`( )`

  - **Returns** - the amount of score points for capturing a facility.

- *getFacilityHeight*
`public double` **getFacilityHeight**`( )`

  - **Returns** - the height of facility area.

- *getFacilityWidth*
`public double` **getFacilityWidth**`( )`

  - **Returns** - the width of facility area.

- *getFighterAerialAttackRange*
`public double` **getFighterAerialAttackRange**`( )`

  - **Returns** - the attack range of a fighter against aerial targets.

- *getFighterAerialDamage*
`public int` **getFighterAerialDamage**`( )`

  - **Returns** - the attack damage of a fighter against aerial targets.

- *getFighterAerialDefence*
`public int` **getFighterAerialDefence**`( )`

  - **Returns** - the defence of a fighter against aerial attacks.

- *getFighterAttackCooldownTicks*
`public int` **getFighterAttackCooldownTicks**`( )`

  - **Returns** - the attack cooldown of a fighter.

- *getFighterDurability*
`public int` **getFighterDurability**`( )`

  - **Returns** - the maximal durability of a fighter.

- *getFighterGroundAttackRange*
`public double` **getFighterGroundAttackRange**`( )`

  - **Returns** - the attack range of a fighter against ground targets.

- *getFighterGroundDamage*
`public int` **getFighterGroundDamage**`( )`

  - **Returns** - the attack damage of a fighter against ground targets.

- *getFighterGroundDefence*
`public int` **getFighterGroundDefence**`( )`

  - **Returns** - the defence of a fighter against ground attacks.

- *getFighterProductionCost*
`public int` **getFighterProductionCost**`( )`

- **Returns** - the amount of ticks to produce a fighter on a factory.

- *getFighterSpeed*
`public double` **getFighterSpeed**`( )`

  - **Returns** - the maximal speed of a fighter.

- *getFighterVisionRange*
`public double` **getFighterVisionRange**`( )`

  - **Returns** - the base vision range of a fighter.

- *getForestTerrainSpeedFactor*
`public double` **getForestTerrainSpeedFactor**`( )`

  - **Returns** - the speed factor of ground vehicles in forest terrain.

- *getForestTerrainStealthFactor*
`public double` **getForestTerrainStealthFactor**`( )`

  - **Returns** - the vision range factor of any vehicle that is trying to detect a ground vehicle in forest terrain.

- *getForestTerrainVisionFactor*
`public double` **getForestTerrainVisionFactor**`( )`

  - **Returns** - the vision range factor of a ground vehicle in forest terrain.

- *getHelicopterAerialAttackRange*
`public double` **getHelicopterAerialAttackRange**`( )`

  - **Returns** - the attack range of a helicopter against aerial targets.

- *getHelicopterAerialDamage*
`public int` **getHelicopterAerialDamage**`( )`

  - **Returns** - the attack damage of a helicopter against aerial targets.

- *getHelicopterAerialDefence*
`public int` **getHelicopterAerialDefence**`( )`

  - **Returns** - the defence of a helicopter against aerial attacks.

- *getHelicopterAttackCooldownTicks*
`public int` **getHelicopterAttackCooldownTicks**`( )`

  - **Returns** - the attack cooldown of a helicopter.

- *getHelicopterDurability*
`public int` **getHelicopterDurability**`( )`

  - **Returns** - the maximal durability of a helicopter.

- *getHelicopterGroundAttackRange*
`public double` **getHelicopterGroundAttackRange**`( )`

  - **Returns** - the attack range of a helicopter against ground targets.

- *getHelicopterGroundDamage*
`public int` **getHelicopterGroundDamage**`( )`

  - **Returns** - the attack damage of a helicopter against ground targets.

- *getHelicopterGroundDefence*
  `public int` **getHelicopterGroundDefence**`( )`
  - **Returns** - the defence of a helicopter against ground attacks.

- *getHelicopterProductionCost*
  `public int` **getHelicopterProductionCost**`( )`
  - **Returns** - the amount of ticks to produce a helicopter on a factory.

- *getHelicopterSpeed*
  `public double` **getHelicopterSpeed**`( )`
  - **Returns** - the maximal speed of a helicopter.

- *getHelicopterVisionRange*
  `public double` **getHelicopterVisionRange**`( )`
  - **Returns** - the base vision range of a helicopter.

- *getIfvAerialAttackRange*
  `public double` **getIfvAerialAttackRange**`( )`
  - **Returns** - the attack range of an IFV against aerial targets.

- *getIfvAerialDamage*
  `public int` **getIfvAerialDamage**`( )`
  - **Returns** - the attack damage of an IFV against aerial targets.

- *getIfvAerialDefence*
  `public int` **getIfvAerialDefence**`( )`
  - **Returns** - the defence of an IFV against aerial attacks.

- *getIfvAttackCooldownTicks*
  `public int` **getIfvAttackCooldownTicks**`( )`
  - **Returns** - the attack cooldown of an IFV.

- *getIfvDurability*
  `public int` **getIfvDurability**`( )`
  - **Returns** - the maximal durability of an IFV.

- *getIfvGroundAttackRange*
  `public double` **getIfvGroundAttackRange**`( )`
  - **Returns** - the attack range of an IFV against ground targets.

- *getIfvGroundDamage*
  `public int` **getIfvGroundDamage**`( )`
  - **Returns** - the attack damage of an IFV against ground targets.

- *getIfvGroundDefence*
  `public int` **getIfvGroundDefence**`( )`
  - **Returns** - the defence of an IFV against ground attacks.

- *getIfvProductionCost*
  `public int` **getIfvProductionCost**`( )`
  - **Returns** - the amount of ticks to produce an IFV on a factory.

- *getIfvSpeed*
  `public double` **getIfvSpeed**`( )`
  - **Returns** - the maximal speed of an IFV.

- *getIfvVisionRange*
  `public double` **getIfvVisionRange**`( )`
  - **Returns** - the base vision range of an IFV.

- *getMaxFacilityCapturePoints*
  `public double` **getMaxFacilityCapturePoints**`( )`
  - **Returns** - the maximal possible amount of facility capture points.

- *getMaxTacticalNuclearStrikeDamage*
  `public double` **getMaxTacticalNuclearStrikeDamage**`( )`
  - **Returns** - the damage in the center of a nuclear explosion.

- *getMaxUnitGroup*
  `public int` **getMaxUnitGroup**`( )`
  - **Returns** - the max index of a unit group.

- *getPlainTerrainSpeedFactor*
  `public double` **getPlainTerrainSpeedFactor**`( )`
  - **Returns** - the speed factor of ground vehicles in plain terrain.

- *getPlainTerrainStealthFactor*
  `public double` **getPlainTerrainStealthFactor**`( )`
  - **Returns** - the vision range factor of any vehicle that is trying to detect a ground vehicle in plain terrain.

- *getPlainTerrainVisionFactor*
  `public double` **getPlainTerrainVisionFactor**`( )`
  - **Returns** - the vision range factor of a ground vehicle in plain terrain.

- *getRainWeatherSpeedFactor*
  `public double` **getRainWeatherSpeedFactor**`( )`
  - **Returns** - the speed factor of aerial vehicles in rain weather.

- *getRainWeatherStealthFactor*
  `public double` **getRainWeatherStealthFactor**`( )`
  - **Returns** - the vision range factor of any vehicle that is trying to detect an aerial vehicle in rain weather.

- *getRainWeatherVisionFactor*
  `public double` **getRainWeatherVisionFactor**`( )`
  - **Returns** - the vision range factor of an aerial vehicle in rain weather.

- *getRandomSeed*
  `public long` **getRandomSeed**`( )`
  - **Returns** - the number that is highly recommended to use as a seed for RNG (generator of pseudo-random numbers).

- *getSwampTerrainSpeedFactor*
`public double` **getSwampTerrainSpeedFactor**`( )`

  - **Returns** - the speed factor of ground vehicles in swamp terrain.

- *getSwampTerrainStealthFactor*
`public double` **getSwampTerrainStealthFactor**`( )`

  - **Returns** - the vision range factor of any vehicle that is trying to detect a ground vehicle in swamp terrain.

- *getSwampTerrainVisionFactor*
`public double` **getSwampTerrainVisionFactor**`( )`

  - **Returns** - the vision range factor of a ground vehicle in swamp terrain.

- *getTacticalNuclearStrikeCooldownDecreasePerControlCenter*
`public int` **getTacticalNuclearStrikeCooldownDecreasePerControlCenter**`( )`

  - **Returns** - the cooldown decrease of tactical nuclear strike request per each captured control center.

- *getTacticalNuclearStrikeDelay*
`public int` **getTacticalNuclearStrikeDelay**`( )`

  - **Returns** - the delay between the request of tactical nuclear strike and the nuclear explosion.

- *getTacticalNuclearStrikeRadius*
`public double` **getTacticalNuclearStrikeRadius**`( )`

  - **Returns** - the radius of a nuclear explosion.

- *getTankAerialAttackRange*
`public double` **getTankAerialAttackRange**`( )`

  - **Returns** - the attack range of a tank against aerial targets.

- *getTankAerialDamage*
`public int` **getTankAerialDamage**`( )`

  - **Returns** - the attack damage of a tank against aerial targets.

- *getTankAerialDefence*
`public int` **getTankAerialDefence**`( )`

  - **Returns** - the defence of a tank against aerial attacks.

- *getTankAttackCooldownTicks*
`public int` **getTankAttackCooldownTicks**`( )`

  - **Returns** - the attack cooldown of a tank.

- *getTankDurability*
`public int` **getTankDurability**`( )`

  - **Returns** - the maximal durability of a tank.

- *getTankGroundAttackRange*
`public double` **getTankGroundAttackRange**`( )`

  - **Returns** - the attack range of a tank against ground targets.

- *getTankGroundDamage*
`public int` **getTankGroundDamage**`( )`

- **Returns** - the attack damage of a tank against ground targets.

- *getTankGroundDefence*
`public int` **getTankGroundDefence**`( )`

  - **Returns** - the defence of a tank against ground attacks.

- *getTankProductionCost*
`public int` **getTankProductionCost**`( )`

  - **Returns** - the amount of ticks to produce a tank on a factory.

- *getTankSpeed*
`public double` **getTankSpeed**`( )`

  - **Returns** - the maximal speed of a tank.

- *getTankVisionRange*
`public double` **getTankVisionRange**`( )`

  - **Returns** - the base vision range of a tank.

- *getTerrainWeatherMapColumnCount*
`public int` **getTerrainWeatherMapColumnCount**`( )`

  - **Returns** - the count of columns in terrain/weather maps.

- *getTerrainWeatherMapRowCount*
`public int` **getTerrainWeatherMapRowCount**`( )`

  - **Returns** - the count of rows in terrain/weather maps.

- *getTickCount*
`public int` **getTickCount**`( )`

  - **Returns** - the base game duration in ticks. A real game duration may be lower. Equals to `world.tickCount`.

- *getVehicleEliminationScore*
`public int` **getVehicleEliminationScore**`( )`

  - **Returns** - the amount of score points for destroying a single enemy unit.

- *getVehicleRadius*
`public double` **getVehicleRadius**`( )`

  - **Returns** - the radius of a vehicle.

- *getVictoryScore*
`public int` **getVictoryScore**`( )`

  - **Returns** - the amount of score points received for destroying all enemy units.

- *getWorldHeight*
`public double` **getWorldHeight**`( )`

  - **Returns** - the height of the map.

- *getWorldWidth*
`public double` **getWorldWidth**`( )`

  - **Returns** - the width of the map.

- *isFogOfWarEnabled*
`public boolean` **isFogOfWarEnabled**`( )`

  - **Returns** - `true` if and only if the fog of war is enabled in the current game.

4.1.6 CLASS **Move**

An encapsulated result of each move of your strategy.

DECLARATION

public class Move

extends Object

METHODS

- *getAction*
  `public ActionType` **`getAction`**`( )`
  - **Returns** - the current action.
- *getAngle*
  `public double` **`getAngle`**`( )`
  - **Returns** - the current rotation angle.
- *getBottom*
  `public double` **`getBottom`**`( )`
  - **Returns** - the current bottommost Y of selection rectangle.
- *getFacilityId*
  `public long` **`getFacilityId`**`( )`
  - **Returns** - the current facility ID.
- *getFactor*
  `public double` **`getFactor`**`( )`
  - **Returns** - the current scale factor.
- *getGroup*
  `public int` **`getGroup`**`( )`
  - **Returns** - the current group of units.
- *getLeft*
  `public double` **`getLeft`**`( )`
  - **Returns** - the current leftmost X of selection rectangle.
- *getMaxAngularSpeed*
  `public double` **`getMaxAngularSpeed`**`( )`
  - **Returns** - the current angular speed limit.

- *getMaxSpeed*
`public double` **getMaxSpeed**`( )`
  - **Returns** - the current speed limit.

- *getRight*
`public double` **getRight**`( )`
  - **Returns** - the current rightmost X of selection rectangle.

- *getTop*
`public double` **getTop**`( )`
  - **Returns** - the current topmost Y of selection rectangle.

- *getVehicleId*
`public long` **getVehicleId**`( )`
  - **Returns** - the current vehicle ID.

- *getVehicleType*
`public VehicleType` **getVehicleType**`( )`
  - **Returns** - the current vehicle type.

- *getX*
`public double` **getX**`( )`
  - **Returns** - the current X of a point or vector.

- *getY*
`public double` **getY**`( )`
  - **Returns** - the current Y of a point or vector.

- *setAction*
`public void` **setAction**`( ActionType` **action** `)`
  - **Usage**
    - Sets the desired action.

- *setAngle*
`public void` **setAngle**`( double` **angle** `)`
  - **Usage**
    - Sets the rotation angle.

      This is required parameter for `ActionType.ROTATE`. The positive values mean clockwise rotation.

      The correct values are real numbers in range of `-PI` to `PI` both inclusive.

- *setBottom*
`public void` **setBottom**`( double` **bottom** `)`
  - **Usage**
    - Sets the bottommost Y of selection rectangle.

      This is required parameter for `ActionType.CLEAR_AND_SELECT`, `ActionType.ADD_TO_SELECTION` and `ActionType.DESELECT`, if the group is not set. Otherwise this value will be ignored.

      The correct values are real numbers in range of `top` to `game.worldHeight` both inclusive.

- *setFacilityId*
`public void` **setFacilityId**`( long` **facilityId** `)`
  - **Usage**
    - Sets the facility ID.

      This is required parameter for `ActionType.SETUP_VEHICLE_PRODUCTION`. If there is no factory with this ID or it is not owned by your strategy, then the action will be ignored.

- *setFactor*
`public void` **setFactor**`( double` **factor** `)`
  - **Usage**
    - Sets the scale factor.

      This is required parameter for `ActionType.SCALE`. The values greater than `1.0` increase formation size, the values less than `1.0` decrease formation size.

      The correct values are real numbers in range of `0.1` to `10.0` both inclusive.

- *setGroup*
`public void` **setGroup**`( int` **group** `)`
  - **Usage**
    - Sets the group of units for various actions.

      This parameter is optional for `ActionType.CLEAR_AND_SELECT`, `ActionType.ADD_TO_SELECTION` and `ActionType.DESELECT`. If the group is set for these actions, then all other parameters (`vehicleType`, `left`, `top`, `right`, `bottom`) will be ignored.

      This parameter is required for `ActionType.ASSIGN`, `ActionType.DISMISS` and `ActionType.DISBAND`. This is the only parameter for `ActionType.DISBAND`.

      The correct values are integers in range of `1` to `game.maxUnitGroup` both inclusive.

- *setLeft*
`public void` **setLeft**`( double` **left** `)`
  - **Usage**
    - Sets the leftmost X of selection rectangle.

      This is required parameter for `ActionType.CLEAR_AND_SELECT`, `ActionType.ADD_TO_SELECTION` and `ActionType.DESELECT`, if the group is not set. Otherwise this value will be ignored.

      The correct values are real numbers in range of `0.0` to `right` both inclusive.

- *setMaxAngularSpeed*
`public void` **setMaxAngularSpeed**`( double` **maxAngularSpeed** `)`
  - **Usage**
    - Sets the angular speed limit.

      This is optional parameter for `ActionType.ROTATE`. If this parameter is set, then `maxSpeed` will be ignored.

      The correct values are real numbers in range of `0.0` to `PI` both inclusive. The special `0.0` value means that there is no limit.

- *setMaxSpeed*
`public void` **setMaxSpeed**`( double` **maxSpeed** `)`
  - **Usage**
    - Sets the speed limit.

      This is optional parameter for `ActionType.MOVE`, `ActionType.ROTATE` and `ActionType.SCALE`. If for `ActionType.ROTATE` the max angular speed is set, then this parameter will be ignored.

      The correct values are real nonnegative numbers. The special `0.0` value means that there is no limit.

- *setRight*
`public void` **setRight**`( double` **right** `)`
  - **Usage**
    - Sets the rightmost X of selection rectangle.

      This is required parameter for `ActionType.CLEAR_AND_SELECT`, `ActionType.ADD_TO_SELECTION` and `ActionType.DESELECT`, if the group is not set. Otherwise this value will be ignored.

      The correct values are real numbers in range of `left` to `game.worldWidth` both inclusive.

- *setTop*
`public void` **setTop**`( double` **top** `)`
  - **Usage**
    - Sets the topmost Y of selection rectangle.

      This is required parameter for `ActionType.CLEAR_AND_SELECT`, `ActionType.ADD_TO_SELECTION` and `ActionType.DESELECT`, if the group is not set. Otherwise this value will be ignored.

      The correct values are real numbers in range of `0.0` to `bottom` both inclusive.

- *setVehicleId*
`public void` **setVehicleId**`( long` **vehicleId** `)`
  - **Usage**
    - Sets the vehicle ID.

      This is required parameter for `ActionType.TACTICAL_NUCLEAR_STRIKE`. The action will be ignored, if there is no vehicle with this ID, if unit with this ID is owned by other player or if the nuclear strike target is beyond the vision range of the specified unit.

- *setVehicleType*
`public void` **setVehicleType**`( VehicleType` **vehicleType** `)`
  - **Usage**
    - Sets the vehicle type.

      This is optional filter parameter for `ActionType.CLEAR_AND_SELECT`, `ActionType.ADD_TO_SELECTION` and `ActionType.DESELECT`. This parameter will be ignored, if the group is set.

      This is optional filter parameter for `ActionType.SETUP_VEHICLE_PRODUCTION`. The production progress will be zeroed in any case.

- *setX*
`public void` **setX**`( double  x )`
  - **Usage**
    * Sets the X of a point or vector.

      This is required parameter for `ActionType.MOVE` and specifies the X offset.

      This is required parameter for `ActionType.ROTATE` and specifies the X of the rotation center.

      This is required parameter for `ActionType.SCALE` and specifies the X of the scale pivot.

      This is required parameter for `ActionType.TACTICAL_NUCLEAR_STRIKE` and specifies the X of the explosion center.

      The correct values for `ActionType.MOVE` are real numbers in range of `-game.worldWidth` to `game.worldWidth` both inclusive. The correct values for `ActionType.ROTATE` and `ActionType.SCALE` are real numbers in range of `-game.worldWidth` to `2.0 * game.worldWidth` both inclusive. The correct values for `ActionType.TACTICAL_NUCLEAR_STRIKE` are real numbers in range of `0.0` to `game.worldWidth` both inclusive.

- *setY*
`public void` **setY**`( double  y )`
  - **Usage**
    * Sets the Y of a point or vector.

      This is required parameter for `ActionType.MOVE` and specifies the Y offset.

      This is required parameter for `ActionType.ROTATE` and specifies the Y of the rotation center.

      This is required parameter for `ActionType.SCALE` and specifies the Y of the scale pivot.

      This is required parameter for `ActionType.TACTICAL_NUCLEAR_STRIKE` and specifies the Y of the explosion center.

      The correct values for `ActionType.MOVE` are real numbers in range of `-game.worldHeight` to `game.worldHeight` both inclusive. The correct values for `ActionType.ROTATE` and `ActionType.SCALE` are real numbers in range of `-game.worldHeight` to `2.0 * game.worldHeight` both inclusive. The correct values for `ActionType.TACTICAL_NUCLEAR_STRIKE` are real numbers in range of `0.0` to `game.worldHeight` both inclusive.

4.1.7 CLASS **Player**

The instance of this class contains all the data about player's state.

DECLARATION

public class Player

extends Object

METHODS

- *getId*
  `public long` **getId**`( )`
  - **Returns** - the unique player ID.

- *getNextNuclearStrikeTickIndex*
  `public int` **getNextNuclearStrikeTickIndex**`( )`
  - **Returns** - the tick index of this player's next nuclear strike or `-1`.

- *getNextNuclearStrikeVehicleId*
  `public long` **getNextNuclearStrikeVehicleId**`( )`
  - **Returns** - the nuclear strike spotter vehicle ID or `-1`.

- *getNextNuclearStrikeX*
  `public double` **getNextNuclearStrikeX**`( )`
  - **Returns** - the X of this player's next nuclear strike or `-1.0`.

- *getNextNuclearStrikeY*
  `public double` **getNextNuclearStrikeY**`( )`
  - **Returns** - the Y of this player's next nuclear strike or `-1.0`.

- *getRemainingActionCooldownTicks*
  `public int` **getRemainingActionCooldownTicks**`( )`
  - **Returns** - the amount of ticks remaining before any next action. If this value is `0`, then the player may perform an action in the current tick.

- *getRemainingNuclearStrikeCooldownTicks*
  `public int` **getRemainingNuclearStrikeCooldownTicks**`( )`
  - **Returns** - the amount of ticks remaining before next tactical nuclear strike request. If this value is `0`, then the player may request a nuclear strike in the current tick.

- *getScore*
  `public int` **getScore**`( )`
  - **Returns** - the amount of score points.

- *isMe*
  `public boolean` **isMe**`( )`
  - **Returns** - `true` if and only if this is your player.

- *isStrategyCrashed*
  `public boolean` **isStrategyCrashed**`( )`
  - **Returns** - `true` if and only if the strategy of this player is crashed.

4.1.8 CLASS **TerrainType**

Terrain type.

DECLARATION

public final class TerrainType

extends Enum

FIELDS

- public static final TerrainType PLAIN
  - Plain.
- public static final TerrainType SWAMP
  - Swamp.
- public static final TerrainType FOREST
  - Forest.

4.1.9 CLASS **Unit**

Base class that describes any object ("unit") in the game world.

DECLARATION

public abstract class Unit

extends Object

METHODS

- *getDistanceTo*
  `public double` **getDistanceTo**`( double  x, double  y )`
  - **Parameters**
    - `x` - X-coordinate of the point to get the distance to.

* `y` - Y-coordinate of the point to get the distance to.

– **Returns** - the range between the specified point and the center of this unit.

- *getDistanceTo*
`public double` **getDistanceTo**`( Unit` **unit** `)`

  – **Parameters**
    * `unit` - the unit to get the distance to.
  – **Returns** - the range between the center of the specified unit and the center of this unit.

- *getId*
`public long` **getId**`( )`

  – **Returns** - the unique unit ID.

- *getSquaredDistanceTo*
`public double` **getSquaredDistanceTo**`( double` **x**`, double` **y** `)`

  – **Parameters**
    * `x` - X-coordinate of the point to get the distance to.
    * `y` - Y-coordinate of the point to get the distance to.
  – **Returns** - the squared range between the specified point and the center of this unit.

- *getSquaredDistanceTo*
`public double` **getSquaredDistanceTo**`( Unit` **unit** `)`

  – **Parameters**
    * `unit` - the unit to get the distance to.
  – **Returns** - the squared range between the center of the specified unit and the center of this unit.

- *getX*
`public final double` **getX**`( )`

  – **Returns** - the X of the unit's center. The X-axis is directed from left to right.

- *getY*
`public final double` **getY**`( )`

  – **Returns** - the Y of the unit's center. The Y-axis is directed downward.

4.1.10 CLASS **Vehicle**

Class describing a vehicle. Also contains all properties of a circular unit.

DECLARATION

---

public class Vehicle **extends** CircularUnit

- *getAerialAttackRange*
`public double` **getAerialAttackRange**`( )`
  - **Returns** - maximal range (from center to center) at which this vehicle can attack aerial units.

- *getAerialDamage*
`public int` **getAerialDamage**`( )`
  - **Returns** - damage of one attack dealt to an aerial unit.

- *getAerialDefence*
`public int` **getAerialDefence**`( )`
  - **Returns** - value of defence from aerial units' attacks.

- *getAttackCooldownTicks*
`public int` **getAttackCooldownTicks**`( )`
  - **Returns** - minimal possible interval between attacks.

- *getDurability*
`public int` **getDurability**`( )`
  - **Returns** - current durability.

- *getGroundAttackRange*
`public double` **getGroundAttackRange**`( )`
  - **Returns** - maximal range (from center to center) at which this vehicle can attack ground units.

- *getGroundDamage*
`public int` **getGroundDamage**`( )`
  - **Returns** - damage of one attack dealt to a ground unit.

- *getGroundDefence*
`public int` **getGroundDefence**`( )`
  - **Returns** - value of defence from ground units' attacks.

- *getGroups*
`public int[]` **getGroups**`( )`
  - **Returns** - list of groups that the vehicle belongs to.

- *getMaxDurability*
`public int` **getMaxDurability**`( )`
  - **Returns** - maximal durability.

- *getMaxSpeed*
`public double` **getMaxSpeed**`( )`
  - **Returns** - maximal distance that can be travelled in one tick when not affected by terrain or weather. While rotating arc length is taken into account, not the distance between start and finish positions.

- *getPlayerId*
`public long` **getPlayerId**`( )`
  - **Returns** - owner player's ID.

- *getRemainingAttackCooldownTicks*
`public int` **getRemainingAttackCooldownTicks**`( )`

  – **Returns** - amount of ticks before the vehicle can attack again. In order to attack this value has to be equal zero.

- *getSquaredAerialAttackRange*
`public double` **getSquaredAerialAttackRange**`( )`

  – **Returns** - squared maximal range (from center to center) at which this vehicle can attack aerial units.

- *getSquaredGroundAttackRange*
`public double` **getSquaredGroundAttackRange**`( )`

  – **Returns** - squared maximal range (from center to center) at which this vehicle can attack ground units.

- *getSquaredVisionRange*
`public double` **getSquaredVisionRange**`( )`

  – **Returns** - squared maximal range (from center to center) at which this vehicle can detect other objects, when not affected by terrain or weather.

- *getType*
`public VehicleType` **getType**`( )`

  – **Returns** - vehicle's type.

- *getVisionRange*
`public double` **getVisionRange**`( )`

  – **Returns** - maximal range (from center to center) at which this vehicle can detect other objects, when not affected by terrain or weather.

- *isAerial*
`public boolean` **isAerial**`( )`

  – **Returns** - `true` iff the vehicle is aerial.

- *isSelected*
`public boolean` **isSelected**`( )`

  – **Returns** - `true` iff the vehicle is selected.

4.1.11 CLASS **VehicleType**

Vehicle type.

DECLARATION

public final class VehicleType **extends** Enum

FIELDS

- public static final VehicleType ARRV
  - Armored repair and recovery vehicle. Ground unit. Gradually restores durability of nearby vehicles.
- public static final VehicleType FIGHTER
  - Fighter. Aerial unit. Effective against other aerial units. Can not attack ground units.
- public static final VehicleType HELICOPTER
  - Attack helicopter. Aerial unit. Can attack both aerial and ground units.
- public static final VehicleType IFV
  - Infantry fighting vehicle. Ground unit. Can attack both aerial and ground units.
- public static final VehicleType TANK
  - Tank. Ground unit. Effective against other ground units. Can also attack aerial units.

4.1.12 CLASS **VehicleUpdate**

Class that partly describes a vehicle. Contains a unique vehicle's identifier as well as all vehicle's fields that may be changed during the game.

DECLARATION

public class VehicleUpdate **extends** Object

METHODS

- *getDurability*
  `public int` **getDurability**`( )`
  - **Returns** - current durability, or `0` if the vehicle has either been destroyed or became invisible to you.
- *getGroups*
  `public int[]` **getGroups**`( )`
  - **Returns** - list of groups that the vehicle belongs to.
- *getId*
  `public long` **getId**`( )`
  - **Returns** - unique object's identifier.

- *getRemainingAttackCooldownTicks*
  `public int` **getRemainingAttackCooldownTicks**`( )`
  - **Returns** - amount of ticks before the vehicle can attack again. In order to attack this value has to be equal zero.

- *getX*
  `public double` **getX**`( )`
  - **Returns** - the X of the unit's center. The X-axis is directed from left to right.

- *getY*
  `public double` **getY**`( )`
  - **Returns** - the Y of the unit's center. The Y-axis is directed downward.

- *isSelected*
  `public boolean` **isSelected**`( )`
  - **Returns** - `true` iff the vehicle is selected.

4.1.13 CLASS **WeatherType**

Weather type.

DECLARATION

---

public final class WeatherType

extends Enum

FIELDS

- public static final WeatherType CLEAR
  - Clear.
- public static final WeatherType CLOUD
  - Cloud.
- public static final WeatherType RAIN
  - Rain.

4.1.14 CLASS **World**

This class describes a game world. A world contains all players and game objects ("units").

DECLARATION

public class World

extends Object

METHODS

- *getFacilities*
  `public Facility[] getFacilities( )`
  - **Returns** - list of facilities (in random order). Depending on implementation, corresponding objects may be or not recreated each tick.

- *getHeight*
  `public double getHeight( )`
  - **Returns** - the world height.

- *getMyPlayer*
  `public Player getMyPlayer( )`
  - **Returns** - your player.

- *getNewVehicles*
  `public Vehicle[] getNewVehicles( )`
  - **Returns** - list of vehicles that had no information availavle in the precious tick. It contains both newly created vehicles and those that already existed but were not visible before.

- *getOpponentPlayer*
  `public Player getOpponentPlayer( )`
  - **Returns** - opponent's player.

- *getPlayers*
  `public Player[] getPlayers( )`
  - **Returns** - all players (in random order). After each tick the player objects are recreated.

- *getTerrainByCellXY*
  `public TerrainType[][] getTerrainByCellXY( )`
  - **Returns** - terrain map.

- *getTickCount*
  `public int getTickCount( )`
  - **Returns** - the base game duration in ticks. A real game duration may be lower. Equals to `game.tickCount`.

- *getTickIndex*
  `public int getTickIndex( )`
  - **Returns** - the current game tick.

- *getVehicleUpdates*
  `public VehicleUpdate[] getVehicleUpdates( )`
  - **Returns** - changing field values for each visible vehicle if at least one of these fields has been changed. Durability value zero means that the vehicle is either destroyed or became invisible for you.

- *getWeatherByCellXY*
  `public WeatherType[][] getWeatherByCellXY( )`
  - **Returns** - weather map.

- *getWidth*
  `public double getWidth( )`
  - **Returns** - the world width.

Глава 5

Package <none>

Package Contents *Page*

Interfaces

Strategy 49

This interface contains the methods that each strategy should implement.

5.1 Interfaces

5.1.1 INTERFACE Strategy

This interface contains the methods that each strategy should implement.

DECLARATION

public interface Strategy

METHODS

- *move*
  `public void` **move**`( Player` **me**`, World` **world**`, Game` **game**`, Move` **move** `)`
  - **Usage**
    - Main strategy method, controlling the vehicles. The game engine calls this method once each time tick.
  - **Parameters**
    - `me` - the owner player of this strategy.
    - `world` - the current world snapshot.
    - `game` - many game constants.
    - `move` - the object that encapsulates all strategy instructions.